Slicing in Heterogeneous Software-Defined Radio Access Networks

Dissertation

submitted to

Sorbonne Université

in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Author:

Robert Schmidt

Scheduled for defense on the 4th of November 2021, before a committee composed of:

Reviewers		
Prof.	**Wolfgang Kellerer**	Technical University of Munich, Germany
Prof.	**Oriol Sallent**	Polytechnic University of Catalonia, Spain
Examiners		
Prof.	**Alexandros Kaloxylos**	University of Peloponnese, Greece
Prof.	**Raymond Knopp**	EURECOM, France
Prof.	**Adlen Ksentini**	EURECOM, France
Prof.	**Lucile Sassatelli**	Côte d'Azur University, France
Invited Guest		
MS	**Abhimanyu Gosain**	Northeastern University, USA
Thesis Advisor		
Prof.	**Navid Nikaein**	EURECOM, France

SORBONNE UNIVERSITÉ

EURECOM
Sophia Antipolis

Découpage des réseaux d’accès radio hétérogènes définis par logiciel

Thèse

soumise à

Sorbonne Université

pour l’obtention du grade de docteur

presentée par:

Robert Schmidt

Soutenance de thèse prévue le 4 novembre 2021 devant un jury composé de:

Rapporteurs		
Prof.	**Wolfgang Kellerer**	Université technique de Munich, Allemagne
Prof.	**Oriol Sallent**	Université polytechnique de Catalogne, Espagne
Examinateurs		
Prof.	**Alexandros Kaloxylos**	Université du Péloponnèse, Grèce
Prof.	**Raymond Knopp**	EURECOM, France
Prof.	**Adlen Ksentini**	EURECOM, France
Prof.	**Lucile Sassatelli**	Université Côte-d’Azur, France
Invité		
MS	**Abhimanyu Gosain**	Université Northeastern, Etats-Unis
Directeur de thèse		
Prof.	**Navid Nikaein**	EURECOM, France

Abstract

5G networks are envisioned to be a paradigm shift towards *service-oriented* networks. Unlike previous generations, they are supposed to carry multiple services with possibly conflicting requirements over a shared infrastructure. Slicing emerged as a technology to create virtualized networks over such infrastructure tailored towards each service. In this context, the radio access network (RAN) has been designed with key principles including flexibility, ultra-lean design and forward-compatibility in order to adapt the RAN for novel or even unforeseen services and use cases. Additionally, this adaptability is further enhanced by means of software-defined radio access networking (SD-RAN), which provides runtime programmability of the desired RAN functions and services. Leveraging the complementary concepts of slicing and SD-RAN makes it possible to flexibly manage services over a common infrastructure, granting each tenant (or service owner) control over their slice, providing isolation to other slices, and satisfying service requirements.

In this thesis, we investigate how to efficiently combine slicing and SD-RAN to provide the required level of flexibility and programmability in the RAN infrastructure to realize service-oriented multi-tenant networks. First, we devise an abstraction of a base station that, on a first level, describes and represents *logical* base stations that unify RAN functionality towards the needs of services, and, on a second level, describes a *virtualized* network service on top of a subset of logical base stations that is isolated and customized for a particular service. Second, we propose a novel standard-compliant SD-RAN platform, named FlexRIC, in the form of a software development kit (SDK). The SDK allows to flexibly build use-case-specific, specialized SD-RAN controllers and compose a controller infrastructure with reusable libraries and service models. Based on concrete implementations leveraging FlexRIC, detailed benchmarking is carried out to assess the applicability and performance of the FlexRIC SDK for different types of SD-RAN controllers, each specialized to meet the requirements of a particular service. Third, we provide a modular design for a slice-aware MAC scheduling framework to efficiently manage and control the radio resources in a multi-service environment with quality-of-service (QoS) support. The framework enables forward-compatibility towards novel 5G use cases while being backward-compatible to 4G. It is integrated into the OpenAirInterface 4G and 5G platforms and FlexRIC SDK, and validated for a number of MAC and slice scheduling algorithms, indicating its efficiency, performance, and scalability. Finally, we present a dynamic SD-RAN virtualization layer based on the FlexRIC SDK and MAC scheduling framework to flexibly compose a multi-service SD-RAN infrastructure. It provides programmability for multiple SD-RAN controllers by recursively exposing a virtualized RAN control interface. Experimental results demonstrate that the proposed system provides isolation among multiple virtual network operators (i.e., multiple tenants) while being able to efficiently share the RAN infrastructure in terms resources and state, and guaranteeing conflict-free operation.

Abrégé

Les réseaux 5G sont envisagés comme un changement de paradigme vers des réseaux *orientés services*. Contrairement aux générations précédentes, ils sont censés transporter de multiples services aux exigences potentiellement contradictoires sur une infrastructure partagée. Le découpage en tranches (slicing) est apparu comme une technologie permettant de créer des réseaux virtualisés sur cette infrastructure, adaptés à chaque service. Dans ce contexte, le réseau d'accès radio (RAN) a été conçu selon des principes clés tels que la flexibilité, la conception ultra-légère et la compatibilité avec l'avenir, afin d'adapter le RAN à des services et des cas d'utilisation nouveaux, voire imprévus. En outre, cette adaptabilité est renforcée par le réseau d'accès radio défini par logiciel (SD-RAN), qui permet de programmer les fonctions et services RAN spécifiques durant l'exécution. L'utilisation des concepts complémentaires de découpage en tranches et de SD-RAN permet de gérer avec souplesse les services sur une infrastructure commune, en accordant à chaque propriétaire de service le contrôle de sa tranche, en assurant l'isolation par rapport aux autres tranches et en satisfaisant aux exigences de service.

Dans cette thèse, nous étudions comment combiner efficacement le découpage en tranches et le SD-RAN afin de fournir le niveau requis de flexibilité et de programmabilité dans l'infrastructure RAN pour réaliser des réseaux orientés services. Tout d'abord, nous concevons une abstraction d'une station de base qui, à un premier niveau, décrit et représente des stations de base *logiques* qui unifient la fonctionnalité RAN en fonction des besoins des services, et, à un second niveau, décrit un service de réseau *virtualisé* au-dessus d'un sous-ensemble de stations de base logiques qui est isolé et personnalisé pour un service particulier. Deuxièmement, nous proposons une nouvelle plateforme SD-RAN conforme aux normes, appelée FlexRIC, sous la forme d'un kit de développement logiciel (SDK). Le SDK permet de construire de manière flexible des contrôleurs SD-RAN spécialisés et spécifiques à un cas d'utilisation et de composer une infrastructure de contrôleurs avec des bibliothèques et des modèles de services réutilisables. Sur la base d'implémentations concrètes exploitant FlexRIC, un test de performance détaillé est effectué pour évaluer l'applicabilité et la performance du SDK FlexRIC pour différents types de contrôleurs SD-RAN, chacun étant spécialisé pour répondre aux exigences d'un service particulier. Troisièmement, nous fournissons une conception modulaire pour un cadre d'ordonnancement pour la couche MAC tenant compte du découpage en tranches pour gérer et contrôler efficacement les ressources radio dans un environnement multiservice avec un support de qualité de service (QoS). Le cadre permet la compatibilité avec les nouveaux cas d'utilisation de la 5G tout en étant rétrocompatible avec la 4G. Il est intégré aux plateformes OpenAirInterface 4G et 5G et au SDK FlexRIC, et validé pour un certain nombre d'algorithmes d'ordonnancement d'utilisateurs et de tranches, montrant son efficacité, ses performances et son extensibilité. Enfin, nous présentons une couche de virtualisation SD-RAN dynamique basée sur le SDK FlexRIC et le cadre d'ordonnancement MAC pour composer de manière flexible une infrastructure SD-RAN multiservice. Elle offre une programmabilité pour de multiples contrôleurs SD-RAN en exposant de manière récursive une interface de contrôle RAN virtualisée. Les résultats expérimentaux montrent que le système proposé assure l'isolation entre plusieurs opérateurs de réseaux virtuels (c'est-à-dire plusieurs locataires) tout en étant capable de partager efficacement l'infrastructure RAN en termes de ressources et d'état, et en garantissant un fonctionnement sans conflit.

Acknowledgements

This thesis would not exist with the support of so many people. First and foremost, I would like to thank my supervisor, Navid Nikaein, to give me the opportunity to pursue a PhD in this challenging topic. His motivational character, as well as his understanding for the research challenges and technical knowledge of cellular networks guided me throughout my thesis. Furthermore, I thank Chia-Yu Chang for his help in the early stages of this endeavour when I was writing my first papers. I am also thankful to Konstantinos Alexandris and Mikel Irazabal for the fruitful discussions and collaboration that often made me see clearer when I was in doubt about how to proceed. I thank Chieh-Chun Chen for the solid work regarding the implementation of the 5G MAC scheduler during her internship at EURECOM.

Working towards this thesis would not have been as enjoyable without the friendly environment at EURECOM. I would like to thank former and present members of the Mosaic5G team, former and present office mates, colleagues and friends that I met in and around the campus. In particular, I would like to thank Ali, Alireza, Alison, Andreas, Antonio, Dimitris, Ehsan, Guilherme, Harald, Ismail, Jin, Jonas, Judy, Lorenzo, Marius, Matteo, Mounia, Naser, Nasim, Omid, Placido, Roya, Sagar, Sarra, Sebastian, Shahab, Sihem, Sofia, Thomas, Xenofon, and everyone else who I met during my PhD.

I would like to specially thank Tansu. Even though you were physically far away most of the time, you were still an important emotional support on which I could count, and you are inseparably connected to the experiences during the time of my PhD. I am also particularly thankful to my family. Your support and advice always makes me stay on my path.

Thank you all.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

Acronyms

3GPP	3rd Generation Partnership Project
5GC	5G Core
5GS	5G System
AMF	Access and Mobility Management Function
API	Application Programming Interface
BBU	Base Band Unit
BET	Blind Equal Throughput
BS	Base Station
BSR	Buffer Status Report
BWP	Bandwidth Part
C-RAN	Cloud-RAN
CAPEX	Capital Expenditure
CCE	Control Channel Element
CDF	Cumulative Distribution Function
CN	Core Network
CP	Control Plane
CSI	Channel State Information
CU	Centralized Unit
DCI	Downlink Control Information
DL	Downlink
DRX	Discontinuous Reception
DU	Distributed Unit
E2SM	E2 Service Model
eMBB	enhanced Mobile Broadband
eNB	evolved Node B
EPC	Evolved Packet Core
ETSI	European Telecommunications Standards Institute

FDD	Frequency Division Duplexing
GBR	Guaranteed Bitrate
gNB	next-generation Node B
HARQ	Hybrid Automatic Repeat Request
HSS	Home Subscriber Server
iApp	internal Application
IMSI	International Mobile Subscriber Identity
ITU	International Telecommunications Union
lBS	logical Base Station
LTE	Long-Term Evolution
M2M	machine-to-machine
MAC	Medium Access Control
MCS	Modulation and Coding Scheme
micro-SDK	micro-Service Development Kit
MLB	Mobility Load Balancing
MME	Mobility Management Entity
mMTC	massive Machine-Type Communications
mmWave	millimeter-Wave
MT	Maximum Throughput
NFV	Network Function Virtualization
NR	New Radio
NSSAI	Network Slice Selection Assistance Information
NSSF	Network Slice Selection Function
OAI	OpenAirInterface
OPEX	Operational Expenditure
PDCCH	Physical Downlink Control Channel
PDCP	Packet Data Convergence Protocol
PDU	Protocol Data Unit
PF	Proportional Fair
PHY	Physical Layer
PLMN	Public Land Mobile Network
PNF	Physical Network Function
PUCCH	Physical Uplink Control Channel

QoS	Quality of Service
RAN	Radio Access Network
RAT	Radio Access Technology
RB	Resource Block
RIC	RAN Intelligent Controller
RLC	Radio Link Control
RR	Round Robin
RRC	Radio Resource Control
RRH	Remote Radio Head
RRM	Radio Resource Management
RTT	Round-Trip Time
RU	Remote Unit
SD	Slice Differentiator
SD-RAN	Software-Defined Radio Access Networking
SDAP	Service Data Adaptation Protocol
SDK	Software Development Kit
SDN	Software-Defined Networking
SDR	Software-Defined Radio
SDU	Service Data Unit
SGW	Serving Gateway
SLA	Service-Level Agreement
SR	Scheduling Request
SST	Slice/Service Type
TC	Traffic Control
TDD	Time Division Duplexing
TN	Transport Network
UE	User Equipment
UL	Uplink
UP	User Plane
URLLC	Ultra Reliable, Low-Latency Communications
V2X	Vehicle-to-Everything
vBS	virtual Base Station
VNF	Virtual Network Function
xApp	external Application

Glossary

CP With respect to SD-RAN, we define the Control Plane (CP) as the *decision part* in the RAN, such as a handover, a scheduling decision, etc.

E2SM An E2 Service Model (E2SM) is a specification that defines the message content for information exchange between an RIC and a RAN function that is transported over the E2 interface. For instance, the E2SM-KPM (key performance metrics) transports statistics measurements.

iApp An internal application (iApp) is an application internal to the controller that utilizes the FlexRIC server library. A composition of iApps results in a specialization of a controller.

lBS A logical Base Station (lBS) is a “synthesized” representation of a base station spanning the whole protocol stack, i.e., that is formed from (possibly disaggregated) RAN network functions in a first-level abstraction.

micro-SDK A micro-Service Development Kit (micro-SDK) encapsulates a RAN control endpoint, i.e., specific customizable CP functionality, and makes it available to multiple services/service-specific controllers.

Service-Oriented RAN A service-oriented RAN transports multiple services while being tailored towards them. It turns the physical network into multiple virtual networks (slices) that are customized and extended towards the service’s requirements, and that can be programmed via service-specific controllers, such that each service owner can program their RAN slice independently.

UP With respect to SD-RAN, we define the User Plane (UP) as the *action part* in the RAN, such as processing a handover, applying a scheduling decision, etc.

vBS A virtual Base Station (vBS) is a representation of a service in a base station that is split from an lBS in a second-level abstraction. It defines a service's SLA and gives it control over a part of the RAN.

xApp In the FlexRIC sense, an external application (xApp) is an application that lies outside of the controller, i.e., running in a separate process. In the O-RAN sense, an xApp is an application that runs on top of a RIC to provide radio resource management functionality. In FlexRIC, this functionality might be achieved through an iApp as well.

Chapter 1

Introduction

1.1 Evolution towards Service-Oriented Networks

Mobile wireless networks have become a part of everybody's daily life. Within a couple of decades, generations of mobile networks reshaped the way how we interact and communicate. Starting in the 1990s, 2G (2^{nd} generation) networks allowed to communicate from everywhere using small portable mobile and provided limited data services. In the 2000s, 3G networks allowed greater network data rates, especially through the 3G evolution HSPA (high-speed packet access), allowing people to use multimedia services on the go. In the 2010s, 4G networks based on Long-Term Evolution (LTE) further boosted network data rates through spectrally more efficient transmissions while also being able to multiplex more users on the same spectrum. The emergence of these networks roughly coincided with the advent of the smartphone, which brought new use cases, such as social media, and made mobile video in high definition a reality.

Now, in the 2020s, 5G networks are arriving. Every generation increased user-experienced throughput, and 5G is not different. As surveyed in Cisco's annual internet report [1], the average global mobile data rates are rising at an annual 20 % rate, reaching 43.9 Mbps by 2023 from 13.2 Mbps in 2019, which is in particular attributed to 5G. However, increasing data rates, used for multimedia services destined to people, are only one part of the development that is happening. In fact, an increasing number of subscribers in mobile networks are *machines*. As also reviewed by Cisco, there is a trend towards machine-to-machine (M2M) communications that can be seen by the number of M2M devices, increasing from 33 % in 2018 to 50 % in 2023 among all devices in mobile networks. (During the same time frame, while still being the prevalent device, the share of smartphones is supposed to *shrink*, from 46 % to 41 % percent.) The number of M2M connections is set to grow almost 2.5-fold, from 6.1 billion in 2018 to 14.7 billion connections in 2023, and consist of a mixture of services, such as connected home, connected cars, or energy. These services are associated to industry verticals that provide such service to a specific customer group, and many of them have a 20 % annual growth rate. In other words, M2M communications are becoming more and more widespread. However, the requirements that such use cases exhibit vary from traditional, data-rate-oriented use cases.

In 2015, the International Telecommunications Union (ITU) defined three broad usage scenarios of services that mobile networks should support [2] towards the international mobile telecommunications (IMT) standard in and beyond the year 2020 (IMT-2020):

- Enhanced Mobile Broadband (eMBB) is a continuation of the mobile broadband vision of 3G and 4G, with use cases exhibiting high to extreme data rate requirements for many users in small areas, i.e., hotspots, while additionally emphasizing a need for high mobility with a good user experience. This category also includes new, mostly human-centric use cases, such as 3D and ultra-high definition video.
- Ultra Reliable, Low-Latency Communications (URLLC) refers to use cases that have high requirements with respect to reliability, latency, and availability that mostly include M2M communications, e.g., vehicle-to-vehicle with the famous self-driving car, or industrial or otherwise mission-critical applications, such as factory automation. Additionally, other human-centric use cases like gaming require low latency and high reliability for a good experience as well.
- Massive Machine-Type Communications (mMTC) use cases are purely machine-centric such as the smart city, where numerous devices are deployed to collect data which is sent sporadically in small chunks, e.g., sensor measurements. In total, this usage scenario is characterized by a high connection density and requires high energy efficiency, as these devices are deployed for an extended period of time.

In this context, previously mentioned verticals, such as media, automotive, or eHealth, provide a service that is associated to a usage scenario. For instance, the connected car is a pure URLLC service that relies on very reliable, low-latency communication services.

To support these services with different requirements, the ITU also put forward a number of requirements that IMT-2020 should fulfill in the radio interface [3]. Among these, we highlight the following three requirements in comparison to the former IMT-Advanced for 4G networks, as each highlights a particular requirement of one the usage scenarios:

- 10x increase of user experienced rate to 100 Mbps, particularly for eMBB services,
- 10x lower latency, down to 1 ms, particularly for URLLC services, and
- 10x higher connection density, up to 10^6 devices/km^2, particularly for mMTC services.

However, not all usage scenarios require the same performance; worse, fulfilling some might even be detrimental to others. For instance, low latency requirements are partially incompatible with high user-experienced rates, as the former requires more frequent control signaling, limiting achievable data rates. Therefore, ITU identified an “importance” of key performance indicators, shown in Figure 1.1, which need to be fulfilled by the network depending on the services multiplexed onto the infrastructure, and their heterogeneous requirements. For instance, machine-centric usage scenarios have relaxed throughput requirements, and require lower latency and higher connection density in comparison to more human-centric usage scenarios.

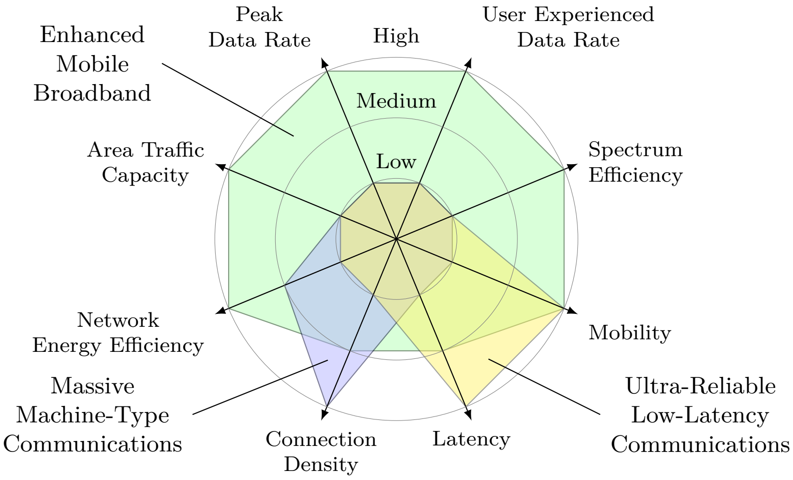


Figure 1.1 – The “importance” of performance indicators for the three usage scenarios. [2]

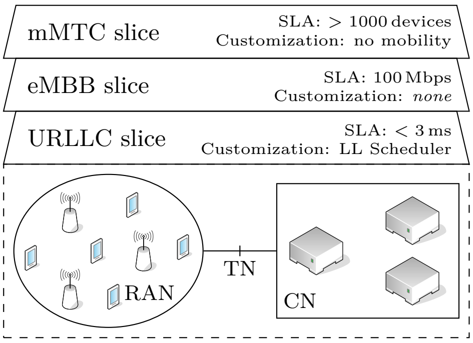


Figure 1.2 – The concept of slicing: multiple slices are overlaid over a common infrastructure.

The concept of network slicing [4] has emerged to support multiple services with heterogeneous requirements over a shared infrastructure. Using network slicing, virtual networks are created for each service over a common underlying infrastructure, as shown in Figure 1.2. The virtual networks, called slices, create service-specific end-to-end networks that tailor the infrastructure towards the service's requirements across Radio Access Network (RAN), Core Network (CN), and Transport Network (TN). In this setting, an infrastructure provider owns the network and multiplexes services through slices onto its multi-tenant infrastructure. Service owners, such as verticals, wish to deploy their service(s) with a guaranteed Service-Level Agreement (SLA) over the infrastructure.

Correspondingly, the support of network slicing in 5G networks has been identified early as a key requirement to provide the necessary adaptability [5]. The 3rd Generation Partnership Project (3GPP) standardized a set of technologies, the 5G System (5GS), in fulfillment of IMT-2020 [6]. In the corresponding CN, a service-based architecture[1]

[1] Note that in the context of the CN, service-based refers to services provided by the individual functional blocks of the CN, and *not* to a service of a vertical, such as a multi-media service.

allows to slice the CN by combining network functions to create service-specific CNs [7]. For service owners, it is thus straight-forward to control and program their slices by combining the relevant network functions. On the other hand, the RAN is logically a single network function [8], and slicing remains implementation-specific [9]. While a customization towards the different usage scenarios is foreseen, there are no means for service owners such as verticals to control their virtual network.

In summary, unlike previous generations of mobile networks, which rather followed a “one-size-fits-all” approach, the proliferation of services with heterogeneous requirements requires 5G networks to adapt of the transported services in order to ultimately fulfill the corresponding use cases. While the CN can be straight-forwardly customized towards vertical needs and be controlled by them, the RAN is still a relatively rigid function that might be customized but not controlled by service owners such as verticals towards their specific use cases.

This calls for a *service-oriented* RAN design that is flexibly sliced, customized and extended towards the services’ requirements, and in which service owners gain the required level of control over their virtual networks through dedicated, service-specific controllers while being isolated from each other. Such RAN design is also an enabler for the future Network 2030 [10] envisioned by ITU, where a controller layer hosts controller capability for an access network, including service-specific controllers.

1.2 Challenges for Service-Oriented RAN

Introducing the necessary flexibility towards a service-oriented RAN is challenging, as it is the most complex part of the overall network, and plays a significant role in the achievement of the requirements put forward by ITU. To allow a maximum of flexibility, a number of principles guided the design of the RAN, termed New Radio (NR) [11]. First, it should be *flexible* in spectrum usage to support the high requirements on peak throughput and user-experienced data rate. In a broader scope, this flexibility might also refer to the adaptability of NR towards the different uses cases. Second, NR should be *forward-compatible* to ensure sustainability and evolution to support future, not-yet-foreseen use cases. For instance, time-frequency resources should be as configurable as possible towards a flexible use, which also avoids backward compatibility issues, as future releases do not need to handle legacy devices that rely on specific functionality. Third, NR should follow an *ultra-lean* design, minimizing always-on signals and in general the footprint of the network, in order to ensure higher energy efficiency and further enable forward-compatibility.

These design principles are accompanied by technological enablers that are widely considered, such as millimeter-Wave (mmWave) operation, network densification, and massive multiple-input multiple-output [12], to achieve performance requirements like a 10-fold increase of peak throughput. For instance, network densification promises to further improve traffic capacity and connection density, and mmWave spectrum allows using hitherto unused spectrum. However, they incur new challenges. Both network densification and mmWave spectrum require coordination between base stations to efficiently use the available resources with little interference. The coordination in turn

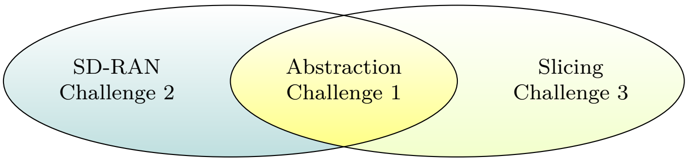


Figure 1.3 – The service-oriented RAN incurs challenges addressed in this thesis.

can be achieved by considering the Cloud-RAN (C-RAN) approach [13], which centralizes the RAN processing in a central cloud and allows to jointly execute baseband processing. To increase the flexibility with respect to centralization, 3GPP investigated and defined multiple splits in the RAN [14], resulting in Centralized Unit (CU), Distributed Unit (DU), and Remote Unit (RU). This leads to a *disaggregated* RAN, which allows to flexibly split the network to allow a varying degree of coordination and multiplexing gains weighed against TN requirements, at the price of an increased complexity of the infrastructure (disaggregation is covered in more detail in Section 2.1.3).

This RAN evolution is accompanied by an IT evolution. In particular the two principles of virtualization and softwarization are considered as key enablers for 5G networks [15], [16] to enable control flexibility in the network. Towards a service-oriented RAN, we highlight softwarization, which allows to decouple Control Plane (CP) and User Plane (UP) processing and enables programmability and control of the network, and is enabled through Software-Defined Networking (SDN) and, in the RAN, Software-Defined Radio Access Networking (SD-RAN). The RAN-IT co-evolution also gave birth to an industry trend applying these principles to the RAN, known as the "Open" RAN [17], focusing on open interfaces in order to open up the traditionally closed and rigid architecture of the RAN and allowing increased flexibility and interoperability.

In summary, the forthcoming 5G networks go beyond a new radio and wider spectrum; they are about the evolution of computing for wireless networks in support of a variety of services and capabilities in diverse usage scenarios. However, we observe that while much work went into attaining the heterogeneous performance requirements, there is still a lack of providing the necessary control to service owners within the RAN to allow them to independently control their slices. In this respect, we identify three key challenges, also highlighted in Figure 1.3, that need to be addressed to attain the vision of a *service-oriented* RAN:

Base Station Abstraction. The trend towards disaggregated RANs allows a flexible centralization of the RAN processing, but also leads to increasingly diverse deployment options. Further, the network densification that is required to achieve ITU usage scenario requirements, such as a higher connection density, necessitates a coordination of base station and the services that are multiplexed. Finally, 5G does not only provide a specific Radio Access Technology (RAT), but uses diverse spectrum bands (e.g., mmWave) and different access technologies (4G, 5G, Wi-Fi), which leads to a highly heterogeneous landscape with respect to the RAN. In total, all these developments augment the already high complexity of the RAN.

Further, given that the vertical market is a driving force of 5G [18], the RAN should be adapted towards the vertical's need, *without* necessarily having to run a dedicated network[2]. Thus, the network shall be open to allow services to control and customize the network towards the vertical's needs via lightweight abstractions, corresponding to the service-oriented RAN.

Therefore, it is of paramount importance to represent the heterogeneous infrastructure in a simplified manner towards the services. Services should be able to customize the infrastructure towards their requirements. Additionally, service owners should have programmable control over their slice, while services are isolated from each other. To this end, a lightweight and customizable service definition on top of a common RAN should adapt the RAN towards the requirements of the services, and enable programmability for multiple services, while hiding unnecessary complexity of the RAN, ensuring full use of 5G's flexibility to actually achieve the required performance, and sharing as much infrastructure as possible.

SD-RAN Infrastructure According to 5G Principles. As has been depicted, 5G-NR provides unprecedented flexibility in the RAN to support diverse use cases in a multi-tenant environment. In this context, the need for programmability and control through SD-RAN is of high importance [20]. However, there is still a lack of proper SD-RAN design and prototype implementation that adheres to the fundamental design principles of 5G, as existing platforms are either not flexible enough to support the modularity for protocol and service extension of 5G, or they follow a "one-size-fits-all" architecture which implies overhead that might not be needed for all or even be detrimental to some uses cases and deployments (we refer the reader to Section 3.1 regarding the related work).

In general, existing SD-RAN approaches either do not offer the modularity that is found in the underlying RAN infrastructure, or come as large container clusters that impose overhead even on use cases that do not need it. Furthermore, they do not provide flexibility to implement novel controller designs, e.g., to implement the abstractions necessary to unlock the potential of a service-oriented RAN, and limit research opportunities to develop use-case-specific controllers.

Slice-Aware 5G MAC Scheduling. 5G networks are supposed to operate in a number of different deployment scenarios, e.g., sub-6 GHz and mmWave, need to remain forward-compatible to new use cases, and handle multiple services. The way radio resources are handled must be adapted towards the considered deployments and adopt the corresponding technological enablers of NR [21], such as the use of specific numerologies, mini-slot scheduling, Time Division Duplexing (TDD) patterns or even dynamic TDD, or beamforming. Accordingly, the flexibility of NR has to be used to effectively use the radio resources. Furthermore, in order to multiplex multiple services onto a common infrastructure, the corresponding slice SLAs need to be enforced within the RAN, where the slicing system shall guarantee (1) isolation so that changes in a virtual network of one

[2] Non-private networks, where a vertical leases resources from an operator, are complemented by private networks, operated by the vertical itself [19].

service do not affect another service, e.g., violating their SLA, and (ii) sharing of resources to maximize resource utilization. While this needs to be enforced in the complete RAN, which manages radio resources, user connections, and their flows, the radio resources are the most scarce resources in the RAN. Thus, it is of paramount importance to ensure that each service gets the allowed radio resources to fulfill their SLA, while using the appropriate technological enablers (e.g., mini-slots if granted).

In a nutshell, the Medium Access Control (MAC) scheduler that allocates radio resources to users needs to adapt to fulfill all these requirements. This demands a *forward-compatible* design of the scheduler, such that the scheduling algorithm can be adapted to the deployments and use cases at hand. Furthermore, the design should be *flexible* enough to allocate resources to slices considering the heterogeneous service requirements. Finally, the support of *programmability* is critical to provide control of the RAN, e.g., to deploy and release slices, control slices individually, and tailor the slice scheduling according to the SLA requirements of the slices.

1.3 Thesis Statement

Following these challenges, a number of questions arise:

- How to achieve the flexibility and programmability across the SD-RAN-based infrastructure in order realize a service-oriented RAN?
- How can the RAN be customized and extended towards service requirements, and to make full use of the flexibility provided by the underlying RAN?
- How can third parties, such as service owners, control their virtual network (slice)?

In this thesis, we investigate both the SD-RAN and slicing concepts to provide flexibility and programmability in the RAN infrastructure to realize service-oriented RANs while being able to adapt the network to service requirements. To this end, we will

1. Investigate and introduce abstractions to simplify the RAN and embed services,
2. Design a novel SD-RAN platform that follows 5G design principles,
3. Realize a practical MAC scheduling framework to provide deployment flexibility, extensibility and programmability in a multi-service infrastructure, and
4. Propose an SD-RAN virtualization layer to implement a service-oriented RAN that allows multiple service-specific controllers belonging to different service owners or operators to control their virtual network.

1.4 Contribution and Outline of the Thesis

The thesis starts with a discussion of the relevant concepts and technologies with respect to SD-RAN and slicing in Chapter 2. Specifically, it provides (1) a background on 5G/NR

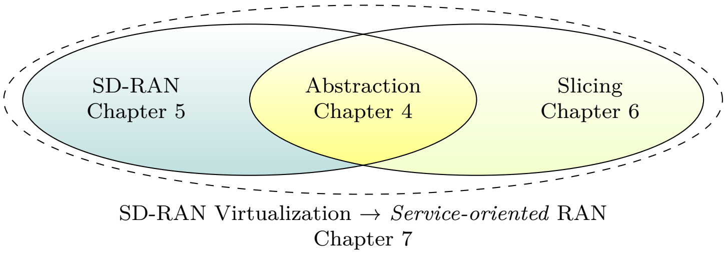


Figure 1.4 – Mapping of thesis chapters to challenges.

including the protocol stack, Physical Layer (PHY) basics, and the scheduler operation for user and slice scheduling, explains the concepts of (2) SDN/SD-RAN and the newly standardized E2 interface that is used in Chapter 5, and (3) RAN virtualization. Then, in Chapter 3, we examine the current literature with respect to SD-RAN and slicing, including an analysis of per-slice RAN customization and extension aspects.

After this point, the thesis is structured around the challenges discussed in Section 1.2. Each chapter addresses one topic, and we provide a mapping of chapters to the challenges in Figure 1.4. This thesis makes the following contributions:

Chapter 4: Base Station Abstraction. This chapter proposes a generic abstraction of the RAN to facilitate the embedding of services and their respective customization requirements in a multi-service environment and while considering the increasing complexity of cellular networks. First, we describe a generic two-level abstraction of a base station which results in descriptors comprising resources, processing (capabilities), and state such as statistics and configuration. Further, using the FlexVRAN framework, we explain the concept of a two-level abstraction: First, it "re-aggregates" the physical RAN into *logical* base stations, and provides a service-oriented description of the physical infrastructure. Second, this logical description of base stations is then split into *virtual* base stations to describe individual services in the service-oriented RAN. Further, we design the RAN Engine that furthermore explains how services can customize and extend processing towards their needs using containerized micro-services including per-service programmability, corresponding to the service-oriented vision of the RAN. We complement this discussion with a mathematical optimization problem considering the "re-aggregation" of a logical base station spanning the full protocol stack from disaggregated RAN network functions while considering constraints of the TN in the form of a single shared midhaul. We show the NP-hardness of the problem, and solve a simplified form analytically.

This chapter is based on the publications as identified in Table 1.1.

Chapter 5: FlexRIC SDK. In this chapter, we design FlexRIC, a flexible, forward-compatible and ultra-lean Software Development Kit (SDK) to build specialized, service-specific controllers according to 5G principles. Using FlexRIC, SD-RAN controllers can be specialized for (i) use cases through corresponding service models (e.g., slicing), (ii) network entities, including CU and DU, (iii) RATs like LTE and NR, or also (iv) software/equipment vendors such as OpenAirInterface (OAI) or SRSRAN. At the same time, FlexRIC is suitable for real-time control and multi-RAT-capable. It is interoperable with industry efforts by means of the E2 interface, and has an internal representation to adapt the E2 protocol to diverse encoding/decoding schemes and transport protocols in function of the use case. We show how FlexRIC can be leveraged to re-implement existing SD-RAN controllers and slicing frameworks. We extensively evaluate FlexRIC's performance and compare it to the existing FlexRAN controller and O-RAN RAN Intelligent Controller (RIC). For instance, FlexRIC scales better as the number of base stations increase: we measured a x10 lower CPU usage and x3 lower memory utilization for FlexRIC than for FlexRAN in considered scenarios. FlexRIC unleashes the potential of SD-RAN in the envisioned 5G use cases [2], where (i) the radio spectrum, (ii) the traffic flows, and (iii) user associations and handovers can be controlled, coordinated, and optimized. Finally, we present an integration of a network store into FlexRIC to customize and extend the RAN.

This chapter is based on the publications as identified in Table 1.1.

Chapter 6: Flexible MAC Scheduling. In this chapter, we introduce a highly flexible, RAT-agnostic MAC scheduling framework to adapt the scheduler operation towards deployment scenarios and the requirements of overlaid services. First, we present the framework which foresees a separation of “decision” (CP) from “action” (UP) part to ensure forward-compatible scheduler adaptation to deployments, e.g., sub-6 GHz or mmWave operation. A multi-service extension describes how to support multiple slices with possibly heterogeneous requirements. Further, we devise an accompanying Slicing service model that allows an SD-RAN controller, such as a FlexRIC-based one from the previous chapter, to configure the scheduler functionality, providing the necessary programmability to dynamically deploy and control services. It is RAT-independent, and abstracts the slice configuration from the physical infrastructure.

Furthermore, we design a slice algorithm with Quality of Service (QoS)-awareness, allowing to multiplex slices with heterogeneous service requirements, i.e., fixed spectrum allocation, efficient resource utilization, and specific latency requirements. We formulate the corresponding utility functions, and propose a weight-based scheduling algorithm to schedule slices of these three categories. This algorithm neatly integrates into the MAC framework.

We perform extensive simulations using a 3GPP-aligned simulator to validate the MAC scheduling framework with slice capabilities, and evaluate the QoS-aware slice algorithm with respect to the stated service requirements and customization properties. Further, we implemented the framework in OAI [22], [23] for both 4G/LTE and 5G/NR, including multiple slice and user scheduling algorithms, and integrated the Slicing service model for use with FlexRIC. Using an emulator over 4G, we validated the prototype with regard to customizability and scalability in a dynamic emulation scenario for up to 50 users and 8 slices. Finally, the framework is evaluated using 5G radio deployments. Since we implemented the OAI 5G MAC scheduler from scratch in the context of this work, we first assess the 5G/NR radio performance, showing similar performance compared to schedulers of commercial products in terms of latency, and second validate multi-service capabilities using commercial-off-the-shelf phones.

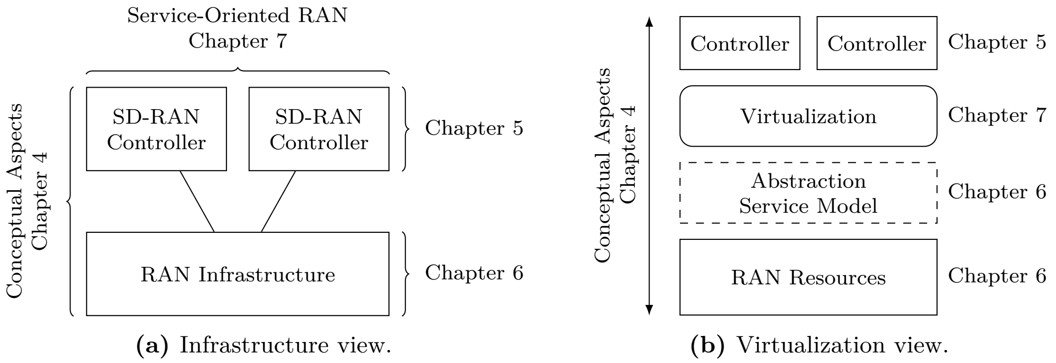


Figure 1.5 – The contributions in this thesis allow to implement a service-oriented RAN.

This chapter is based on the publications as identified in Table 1.1.

Chapter 7: SD-RAN Virtualization. This chapter combines the work of previous chapters and proposes an SD-RAN virtualization layer to realize a service-oriented RAN. It allows multiple SD-RAN controllers, e.g., of different operators, to concurrently control their (virtual) networks by slicing and virtualizing the SD-RAN control. Using the FlexRIC SDK of Chapter 5, we design a specialized controller that recursively exposes an E2 interface to multiple, tenant-specific controllers. It uses the Slicing service model of the MAC scheduling framework provided in Chapter 6 to virtualize the SD-RAN control. To this end, we discuss the per-operator resource virtualization on the example of the NVS slice algorithm [24] and how it provides isolation between the tenants. We implement a proof-of-concept to validate our design, and an assessment proves its feasibility and infrastructure advantages, such as multiplexing gain of up to 100 %. Finally, it natively supports disaggregated, multi-RAT deployments through the employed Slicing service model and FlexRIC SDK.

This chapter is based on the publication as identified in Table 1.1.

Table 1.1 lists selected publications in chronological order, and identifies their contribution to the thesis chapters.

The contributions in this thesis can be seen from an infrastructure point of view, as shown in Figure 1.5a. Here, multiple service-specific controllers operate over a common infrastructure and control their virtual networks. Chapter 6 enables programmability of the MAC layer in the RAN infrastructure, especially for multi-service operation through the configuration of slice algorithms and the slices. Further, the FlexRIC SDK from Chapter 5 allows building specialized SD-RAN controllers towards the need of the service owners. The SD-RAN virtualization layer in Chapter 7 explains how multiple controllers can control their virtual networks in the shared infrastructure while being isolated from each other.

Corresponding to the independent control of multiple service-specific controllers, the contributions in these chapters can be seen from a virtualization point of view, as shown in Figure 1.5b. Under this angle, the RAN has associated resources that can be managed. This is conceptualized in Chapter 4. The (radio) resources of the RAN correspond to the MAC framework in Chapter 6, which also provides the necessary control through an abstraction. By leveraging the FlexRIC SDK, we build the virtualization layer in the form of a proxy in Chapter 7, which then allows to expose control towards multiple SD-RAN controllers that are also based on FlexRIC from Chapter 5.

Table 1.1 – Publications and their contributions to the chapters.

Type	Reference	Ch. 4	Ch. 5	Ch. 6	Ch. 7
C	R. Schmidt, C. Chang, and N. Nikaein, "FlexVRAN: A flexible controller for virtualized RAN over heterogeneous deployments", in *ICC 2019 – 2019 IEEE International Conference on Communications (ICC)*, Shanghai, China, May 2019, pp. 1–7	✘			
C	R. Schmidt, C.-Y. Chang, and N. Nikaein, "Slice scheduling with QoS-guarantee towards 5G", in *IEEE Global Communications Conference (Globecom) 2019*, Waikoloa, Hawaii, USA, 2019, pp. 1–7			✘	
D	R. Schmidt and N. Nikaein, "Demo: Efficient multi-service ran slice management and orchestration", in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–2			✘	
D	R. Schmidt and N. Nikaein, "Service-oriented intelligent and extensible RAN", in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '20, London, United Kingdom: Association for Computing Machinery, 2020		✘		
J	R. Schmidt and N. Nikaein, "Ran engine: Service-oriented ran through containerized micro-services", *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, Mar. 2021, pp. 469–481	✘		✘	
C	R. Schmidt, M. Irazabal, and N. Nikaein, "Flexric: An SDK for next-generation SD-RANs", in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21)*, Munich, Germany, virtual, Dec. 2021, pp. 411–425		✘	✘	✘

C = Conference, D = Demo, J = Journal

Chapter 2

Background

This chapter introduces the reader to the technologies and concepts used in this thesis. We start by outlining the main concepts of 5G/NR and highlight differences to 4G/LTE in Section 2.1. As the focus of this thesis is on the RAN, we elaborate on the RAN protocol stack, the physical layer design, and the function of the scheduler. Afterwards, Section 2.2 introduces the general concept of software-defined networking and software-defined radio access networking, and surveys the main features of the E2 interface. Finally, in Section 2.3, we briefly summarize how the RAN can be virtualized, which is an important enabler for sliced software-defined radio access networks.

2.1 5G/NR

2.1.1 Overview

The high-level architecture of the 5G System (5GS) is shown in Figure 2.1. It consists of the CN, called the 5G Core (5GC), and the RAN, called New Radio (NR).

The RAN consists of base stations and User Equipments (UEs). The base stations are called next-generation Node B (gNB). They are responsible for any radio functionality, such as providing wireless connectivity to the UEs. The gNBs are connected through the

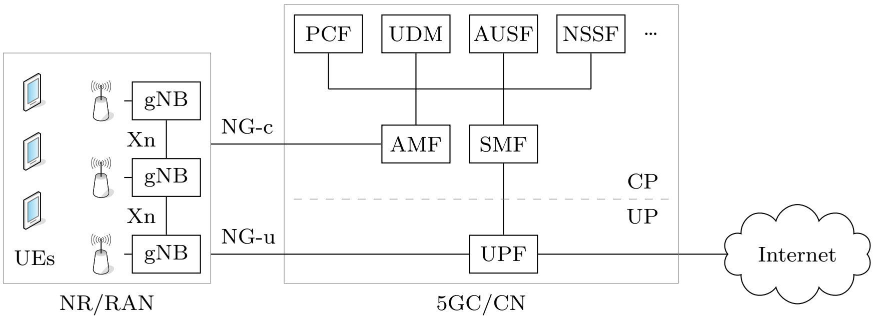


Figure 2.1 – Architecture of the 5G System with core and radio access network.

Xn interface for information exchange. As an example, the gNBs use this interface for preparation and execution of handover of a UE from one cell to another.

The CN follows a UP/CP split. Figure 2.1 shows the CN in the so-called service-based architecture, which puts the emphasis on the services provided by network functions instead of logical nodes and their interfaces. The network functions can be combined to create slice-specific CNs, either being dedicated or shared among slices. Note that service here refers to a specific functionality provided by a network function, and not a 5G service from a service owner, e.g., multimedia service.

The CP anchor from the point of view of the RAN is the Access and Mobility Management Function (AMF). It manages the control signaling between the CN and the UEs, such as registration or mobility. The Session Management Function (SMF) assigns IP addresses to UEs and manages their sessions. Further, functions include the Policy Control Function (PCF), managing policy rules, the Unified Data Management (UDM), providing a data store for user data, the Authentication Server Function (AUSF), handling authentication, or the Network Slice Selection Function (NSSF), handling the slice that has been selected for a device. More network functions exist, and we omit them for brevity and readability, as this thesis is not concerned with the CN.

In the UP, a single network function, called the User Plane Function (UPF), is responsible for data forwarding. It acts as a gateway between the RAN and external data networks, such as the internet, and performs routing, packet inspection, or QoS handling. QoS handling is achieved through one or more PDU sessions per user which in turn consists of QoS flows, allowing the overall network to perform differentiated packet treatment. In the CN, each IP flow is assigned to a QoS flow and the packets are marked accordingly, allowing the core network to be aware of service requirements in the context of a network slice. In the RAN, the QoS flows are mapped to data radio bearers, which are used in RAN signaling between the gNB and UE.

The 3GPP-based 4G network, called the Evolved Packet System (EPS), is similar to the 5GS. It consists of LTE, the RAN part, and the Evolved Packet Core (EPC), the CN. The base stations in the RAN are called evolved Node B (eNB) instead of gNB and are interconnected through the X2 interface. The Mobility Management Entity (MME) performs connection and mobility management and is the CP anchor for eNBs via the S1-c interface. It is supported by the Home Subscriber Server (HSS) for subscription management. The UP anchor for eNBs is the Serving Gateway (SGW), connected through the S1-u interface. Due to space constraints, we omit a description of the EPS, and refer the reader to Dahlman *et al.* [25] for an in-depth discussion of the architecture.

2.1.2 Radio Access Network

Protocol Stack

The protocol stack of a single gNB is shown in Figure 2.2. As can be seen, the functionality of the different layers can be divided into a CP and a UP part, where the CP can be seen as standardized control functionality that needs to be ensured by a particular layer, e.g., retransmission, whereas UP refers to data processing. Both the gNB and the UE have the same protocol stack, but not all layers on both sides have the same responsibility,

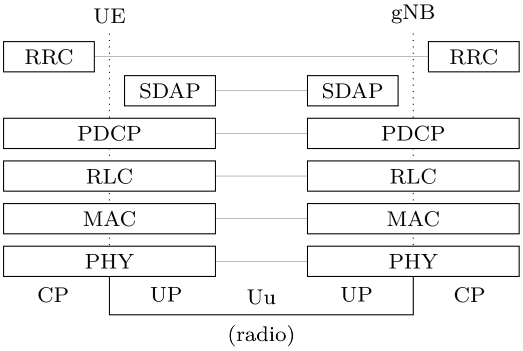


Figure 2.2 – Protocol stack of the 5G/NR RAN.

owing to the fact that the gNB controls the UEs in the network. In the following, we describe the layers from the gNB perspective.

The Radio Resource Control (RRC) is a pure CP layer. It is responsible for broadcasting of system information, management of UE connections, security, mobility, or measurement reporting in support of mobility functions. It also forwards CP messages to the AMF.

A packet that travels in Downlink (DL) from the CN to the UE will first be processed in the Service Data Adaptation Protocol (SDAP) layer. The SDAP is responsible for the mapping from QoS flows to radio bearers, and marking of the corresponding packets in Uplink (UL) and DL. In the UE, the mapping might be configured explicitly, or reflective QoS is applied, where the UE applies the same mapping in UL as it appeared in the DL.

Next, the packet is processed by the Packet Data Convergence Protocol (PDCP) layer. In the UP, the PDCP transfers user data, which relies on sequence numbering for retransmission and reordering purposes, header compression, and ciphering. In the CP, the PDCP notably performs integrity protection and transports control data.

The Radio Link Control (RLC) layer buffers data received from the PDCP until it can be sent on the wireless link. It furthermore segments data packets that cannot be sent at once, and performs retransmissions if configured in the Acknowledged Mode (AM) as opposed to the Unacknowledged Mode (UM).

The Medium Access Control (MAC) layer manages the wireless link by assigning resources to UEs by means of scheduling. This involves priority handling between multiple UEs through dynamic scheduling, and priority handling between a UE's radio bearers by prioritization. Through the corresponding scheduling decisions, it multiplexes the data of a UE into transport blocks that are sent over the air. Errors in transmissions to/from a UE are handled by the Hybrid Automatic Repeat Request (HARQ) protocol, which triggers retransmissions of data.

Finally, the Physical Layer (PHY) encodes data to be sent over the antenna, following the commands of the MAC. This involves (de-)coding and (de-)modulation of data, mapping to antennas, etc.

The layers exchange data vertically. In general, data that is passed "down" to a lower layer is called a Protocol Data Unit (PDU), and data passed "up" to a higher layer is

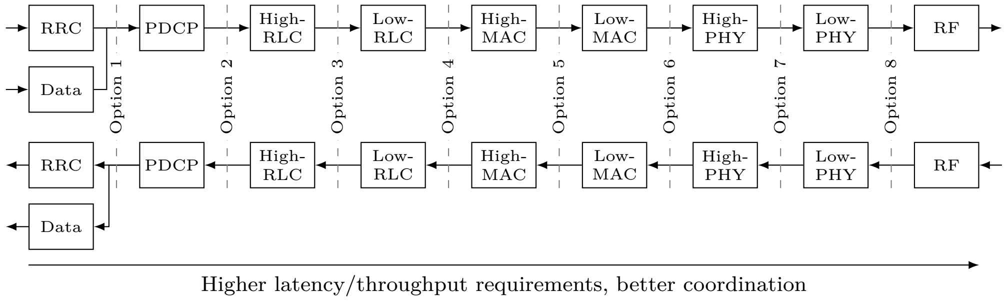


Figure 2.3 – RAN functional split options.

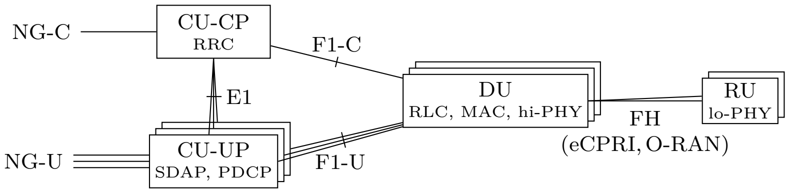


Figure 2.4 – A disaggregated base station, consisting of CU-UP(s), CU-CP, DU(s), and RU(s).

called an Service Data Unit (SDU). For instance, the RLC PDU is the same as the MAC SDU. Processing in the MAC prepends/appends data, resulting in the MAC PDU.

2.1.3 Functional Splits and Disaggregation

The term distributed RAN is used to refer to a network of base stations that are geographically distributed. This is a “natural” deployment, as a base station is located at the cell towers, and performs all base station processing locally at the cell site.

In 2011, China Mobile proposed Cloud-RAN (C-RAN), in which most of the processing functionality is aggregated in a central cloud [13]. This concept exploits so-called functional splits of the base stations that transfer radio symbols to the antenna. For C-RAN, this means that most of the processing functionality of the base station, from RRC down to the upper part of the PHY, reside in the cloud, and an interface called the fronthaul transports signals to geographically dispersed cell towers. The cloud hosts a pool of Base Band Unit (BBU), and only Remote Radio Head (RRH) functionality needs to be present at the cell site. This reduces Capital Expenditure (CAPEX) and Operational Expenditure (OPEX), as less infrastructure needs to be deployed at the cell sites, allows to more easily scale the network to the user demand, and enables increased coordination, as most of the processing happens in a central place. On the other hand, a powerful cloud is needed, and the split between BBUs and RRHs requires a high throughput connection, such as an optical interconnect.

3GPP surveyed the possible split options in TR 38.801 [26], shown in Figure 2.3. In general, a “lower” split, i.e., towards the radio, increases requirements on the transport in terms of latency and/or throughput, but also allows better coordination through joint signal packet processing. For instance, option 8 implies sending I/Q samples in the time-domain to an antenna with high latency and throughput requirements, whereas option 1 implies sending IP packets (for data) with correspondingly relaxed requirements. On the other hand, coordination of multiple cells with option 8 is higher, as the joint signal processing of the antenna signals allows to, for instance, increase the signal-to-noise ratio for individual UEs (coordinated multipoint, CoMP). Finally, 3GPP standardized a number of splits, resulting in four RAN network functions: the Centralized Unit (CU), which is further split into the CU-CP and CU-UP, the Distributed Unit (DU), and the Remote Unit (RU), shown in Figure 2.4 [8] The CU-CP contains the RRC for radio resource management, and the CU-UP handles user plane-related functionality through SDAP and PDCP. They are connected by the standardized E1 interface (option 1 for CP data) to configure the CU-UP. Both CU-CP and CU-UP are connected to the DU through the F1 midhaul (option 2), handling RLC and MAC functionality for radio resource scheduling. Finally, the DU connects to one or more RUs via a fronthaul split such as nFAPI [27], eCPRI [28] or the O-RAN fronthaul [29], which also defines the exact split point of the PHY. For instance, nFAPI results in a split between the MAC and PHY (option 6), whereas eCPRI and O-RAN result in a split where the upper part of the PHY is also located in the DU (options 7 and 8).

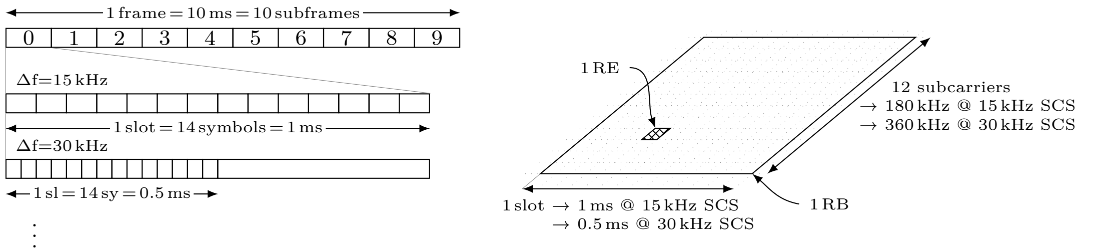
This leads to a disaggregation of a gNB, where the base station processing functionality can be spread among multiple entities. For instance, radio resource management in the RRC can be centralized for many DUs in a central cloud, relaxing the requirements on the connection compared to C-RAN. Also, multiple DUs could be connected to one CU, enabling multiplexing gains and a tighter coordination of cells.

Note that these splits are possible deployment scenarios and nothing prevents a monolithic implementation or reaggregation of the RAN functions into one network function, comprising the full protocol stack. With RAN disaggregation, a base station may be partially shared and partially dedicated, e.g. multiple CU-UPs can be created to (a) balance load and (b) implement service-specific functionalities on top of the shared RAN. On the other hand, this flexibility of deployment options also increases the complexity: for instance, there are multiple handover scenarios for intra- or inter-CU, and a RAN controller needs to connect to multiple RAN network functions.

2.1.4 Physical Layer Basics

The NR specifications mandate Orthogonal Frequency-Division Multiplexing (OFDM) as the waveform for both DL and UL. OFDM uses multiple subcarriers that can be individually modulated, using a basic subcarrier spacing is 15 kHz. However, NR is designed to operate in sub-6 GHz (frequency range 1) as well as mmWave ranges (24 GHz and above, frequency range 2), and the same subcarrier spacing is not suitable due to the different propagation behavior in the frequency ranges. Therefore, NR uses a numerology $\mu = \{0, 1, 2, 3, 4\}$ to spread the subcarrier spacing according to $15\,\text{kHz} \cdot 2^{\mu}$, allowing to adapt the subcarrier spacing to the deployment.

The basic time-domain unit is the frame of 10 ms length. As shown in Figure 2.5a, a frame is divided into subframes of 1 ms length. Depending on the numerology, the subframe is again split into slots. For instance, for numerology 1/subcarrier spacing



(a) Time-domain structure.

(b) Frequency-domain structure.

Figure 2.5 – The 5G/NR physical layer structure.

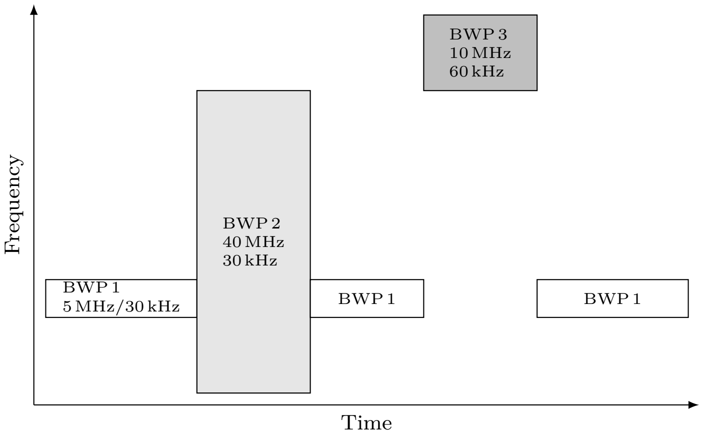


Figure 2.6 – Bandwidth parts and switching, UE perspective.

30 kHz, one subframe contains two slots. Each slot contains 14 OFDM symbols, the basic OFDM time unit.

As has already been mentioned, in the frequency-domain, the minimum unit is the subcarrier. As highlighted in Figure 2.5b, the smallest physical resource is the resource element, made from one subcarrier in one symbol. Further, 12 subcarriers are combined into one Resource Block (RB). One RB during one slot always contains 144 resource elements. Due to the flexible subcarrier-spacing, with increasing numerology, the RB spans a wider bandwidth on a shorter time duration.

The RB is the minimum scheduling resource. The scheduler dynamically schedules UEs on the grid of RBs. It uses channel state information to perform rate adaptation to dynamically vary the amount of data sent to a UE and respond to the channel quality. The maximum number of RBs is 275, and the minimum is 11 (20 for synchronization signals). The maximum carrier bandwidth supported by NR is 400 MHz, which can be increased by combining multiple carriers (carrier aggregation, CA).

Further, NR introduces the concept of the Bandwidth Part (BWP). The BWP can be used to adapt the bandwidth on which a UE can be scheduled. Therefore, each configured BWP can have an individual bandwidth, and the numerology can vary as well, as shown in Figure 2.6. This has the advantage that (1) different device capabilities can be handled, as not all devices will be able to handle the full bandwidth of up to 400 MHz specified in NR, (2) the energy consumption can be lowered, as only a small carrier, e.g., 5 MHz, needs to be active most of the time, and (3) the BWPs with different numerologies can be used to multiplex services with different requirements onto the same carrier. A UE is active on only one BWP, while a gNB handles UEs on multiple BWPs.

The above might seem complicated, but it is an expression of the flexibility of NR to adapt to different deployments. Further, following the ultra-lean principle, there are almost no "always-on" signals in NR. Unlike e.g., LTE, which periodically sends reference signals for channel measurement, this is not the case in NR. Rather, such signals are explicitly configured through RRC signaling, making NR forward-compatible. For instance, NR can be operated with flexible (arbitrary) or even dynamic time-division duplex (TDD) schemes, which would not be possible with fixed "always-on" signals.

OFDM is also used in the DL of LTE. While numerologies are not defined in LTE, the carrier-spacing is the one of the 15 kHz subcarrier spacing in NR. In UL, LTE uses DFT-spread OFDM. More information can be found in Dahlman *et al.* [25].

2.1.5 MAC Scheduler

The MAC scheduler is the entity that makes scheduling decisions every slot in order to allocate resources to UEs. More precisely, the scheduler decides every slot about the allocation of RBs. This basic principle is valid for both LTE and NR. The scheduler pulls data from the RLC, and triggers sending of this data through the PHY, determining how the data is sent. UEs are informed about assigned resources through Downlink Control Informations (DCIs) which are sent in the control region of every slot. Note that only a few UEs might be scheduled in every slot due to the limited resources in the control region.

The scheduler is crucial for the performance of the base station, and allows for vendor differentiation. Therefore, it is not specified by 3GPP. In the following, we briefly survey a number of slice and user scheduling algorithms that are used later in the thesis.

Slice Scheduler

The slice scheduling algorithm decides on which group of UEs receives which resources. We consider the resources $R = \{r_1, r_2, \ldots, r_N\}$ with N the number of resource blocks[1].

Probably the simplest slice scheduling algorithm is "static" slicing, where each slice is allocated a contiguous amount of resources that is fixed in every slot. More formally, all RBs from the l-th to the h-th RB inclusive, $l \leq h$, are attributed to slice i, $R_i = \{r_l, \ldots, r_h\}$, and resources among slices are orthogonal, i.e., $R_i \cap R_j = \emptyset$, $i \neq j$. See Figure 6.4a on page 96 for an example. Static slicing is simple to implement and can guarantee low latency, but slices have to share Control Channel Elements (CCEs) in the control region, limiting the number of DCIs they can allocate. The minimum resource granularity is the RB.

[1] Depending on the used resource allocation type, the addressable resources might be limited to resource block groups, which, as the name says, are multiple RBs addressed as one.

The NVS slice algorithm [24] defines two types of slices: (1) capacity slices $s \in C$ with a share of resources t_s, and (2) rate slices $s \in R$ with a reserved rate r_s^{rsv} over a reference rate r_s^{ref}. The reference rate is an indicator for the minimum rate such slice needs to achieve in scheduled slots to prevent excess utilization of resources under bad channel conditions. If a rate slice does not achieve the reference rate, it will be given less resources to prevent excess resource utilization. NVS defines an optimization problem, shows that the slice types are equivalent for the considered utility functions, and that each slice can be guaranteed its reserved resources under the assumption that the total resource share does not exceed the base station resources:

$$\sum_{s \in C} t_s + \sum_{s \in R} \frac{r_s^{\text{rsv}}}{r_s^{\text{ref}}} \leq 1 \ . \tag{2.1}$$

An exponential moving average $t_{s,i}^{\text{exp}}$, $s \in C$, and $r_{s,i}^{\text{exp}}$, $s \in R$, is used to keep track of the amount of resources scheduled for capacity and rate slices, respectively. It is updated at time instant i as follows:

$$t_{s,i}^{\text{exp}} = (1-\beta) t_{s,i-1}^{\text{exp}} + \beta I(s_i == s) \tag{2.2}$$

$$r_{s,i}^{\text{exp}} = (1-\beta) r_{s,i-1}^{\text{exp}} + \beta r_{s,i}^{\text{inst}} I(s_i == s) \tag{2.3}$$

where β is a small, positive constant, s_i the slice scheduled at slot i, I the indicator function, and $r_{s,i}^{\text{inst}}$ the rate achieved by the rate slice in the current slot i. Then, the algorithm schedules the slice with the highest weight

$$w_{s,i} = \begin{cases} \frac{t_s}{t_{s,i}^{\text{exp}}}, & \text{if } s \in C \\ \frac{r_s^{\text{rsv}}}{r_{s,i}}, & \text{if } s \in R \end{cases} \tag{2.4}$$

on all resources in slot i. It is shown that this scheme guarantees the resource isolation in (2.1). NVS allows any resource granularity, but can incur increased scheduling latencies, especially for slices with a small resource share.

User Scheduler

The user scheduler decides on RB allocation among the set of UEs. To fulfill this task, it uses information about the channel quality, i.e., the radio conditions, and QoS requirements of the individual UE. In the following, we briefly outline the scheduling algorithms used in this thesis. For more information on scheduling algorithm, we refer the reader to Capozzi *et al.* [30] for an in-depth discussion.

One of the easiest schedulers is the Round Robin (RR) scheduler. It allocates RBs in a round-robin fashion, i.e., first RB to UE 1, second RB to UE 2, and so forth. If there are fewer users than available CCEs, the first UE is scheduled again. If there are more users than CCEs, the remaining UEs are scheduled in the next slot. The RR scheduler does not consider channel state information.

The Proportional Fair (PF) scheduler considers the average channel quality of UEs, and opportunistically schedules UEs with good channel quality. Here, the assumption is that on average, all UEs experience a similar channel (assuming only fast fading); favoring UEs with currently better channel quality will then improve the cell throughput, while UEs with worse channel will either be scheduled later (when their channel is good), or when they have not been scheduled for a long time, guaranteeing minimum service.

The PF scheduler allocates data to UE u' among all UEs U in slot i and on RB m

$$u'_m = \arg\max_{u \in U} \frac{R_u(m,i)}{\bar{R}_u(i)}. \tag{2.5}$$

$R_u(m,i)$ is the rate of UE u if it was scheduled, and $\bar{R}_u(i)$ the long-term average throughput of user u following an exponential moving average

$$\bar{R}_u(i) = (1-\beta)\,\bar{R}_u(i-1) + \beta R_u(i)\ , \tag{2.6}$$

for small β and assuming an achieved rate $R_u(i)$ in slot i.

It is possible to enforce the same throughput for all UEs maximizing

$$u'_m = \arg\max_{u \in U} \frac{1}{\bar{R}_u(i)}\ , \tag{2.7}$$

resulting in the Blind Equal Throughput (BET) scheduler. On the other hand, the Maximum Throughput (MT) scheduler maximizes cell throughput. It follows

$$u'_m = \arg\max_{u \in U} R_u(m,i). \tag{2.8}$$

No fairness is enforced, and UEs at the cell edge starve as they are not served.

2.2 Software-Defined Radio Access Networking

2.2.1 SDN

Software-Defined Networking (SDN) is a paradigm to centralize the control of a wired network [31]. Instead of having networking devices make decisions locally of how to route packets or flows, a central controller with a network-wide view makes routing decisions and announces them to the devices. This change is introduced through an abstraction of the network devices, effectively separating the Control Plane (CP) from the User Plane (UP)[2] and introducing *programmability* into the network.

The principle is shown in Figure 2.7. At the bottom is the UP. It consists of various networking devices, in SDN parlance called switches, that are interconnected and that route packets between each other. These switches only implement forwarding functionality, but no routing functionality. If a packet of a flow arrives that the networking device can not handle, it forwards the packet to the central controller via an out-of-band interface (dashed line) in order to receive a decision on how to proceed.

On the next level is the controller. The controller centralizes the CP and makes any routing and management decisions for the whole network. After a packet is received from

[2] The *user* plane is often also called the *data* plane in this context.

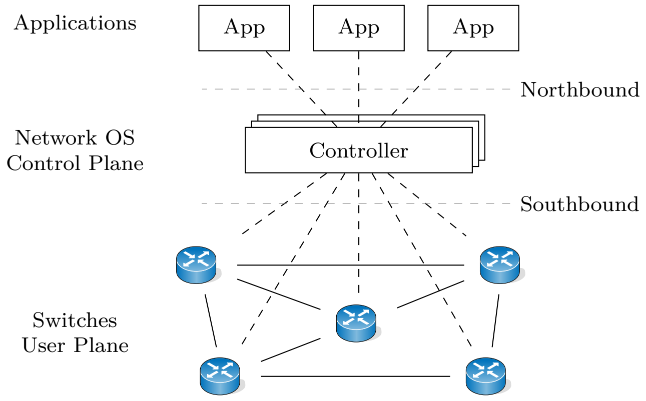


Figure 2.7 – Architecture of software-defined networking (SDN).

a switch, the controller makes a decision using global network state, e.g., calculating the best route for the flow considering the whole network, and instructs the switch accordingly. The interface between the controller and the switch is called the southbound interface (south of the controller).

In order to provide further network programmability, the controller exposes control to applications. Applications use the control to make network decisions, e.g., load balancing, which is enforced by the controller. The interface, called the northbound interface (north of the controller), thereby provides an abstraction of the whole network, which facilitates the implementation of network policies, e.g., new routing algorithms. Due to this abstraction, the controller is also called the network operating system (OS), providing an environment to run networking applications, similar to how a computer operating system provides an environment for computer applications.

In the context of SDN, the open southbound interface OpenFlow [32] is a de facto standard. It provides an abstraction of a switch in the form of a flow table that can be programmed. The flow table consists of flow entries that match certain fields of a packet, e.g., Ethernet or IP addresses, and associates an action such as “forward to port” or “drop this packet”.

2.2.2 SD-RAN and E2

Similarly to switches in the wired domain, SDN can be applied in the wireless domain as well: the application of SDN principles to the RAN is called Software-Defined Radio Access Networking (SD-RAN). SD-RAN introduces decoupling of CP and UP in order to provide programmability of the whole cellular network. However, there are a number of specificities that are inherent to SD-RAN due to the way RANs work:

1. Base stations emit radio signals, and therefore cause interference to each other. Further, there is dedicated connection management of UEs, including admission control, connection re-establishment, handover, etc., which does not exist in “classical” SDN.

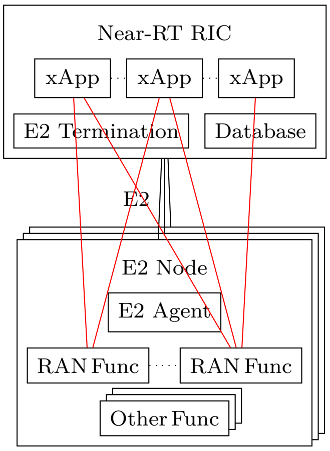


Figure 2.8 – E2 enables applications ("xApps") to control the RAN through RAN functions.

2. There is a trend towards base station disaggregation. Multiple RAN network functions belonging to the same base station need to be managed, including coordination among them.

3. The RAN resource allocation on the MAC layer happens at high frequency (every millisecond or more frequent in 5G). These time constraints are a particular challenge, and do not exist in SDN.

Due to the above reasons, the CP of a base station is much more complex than that of a switch, and it is not possible or useful to completely separate the CP from the UP. For instance, the CP protocol decisions in the RAN about HARQ retransmissions might quickly overload a controller. Therefore, in this thesis, we adopt the definition of CP-UP separation as already found in FlexRAN [33]. The RAN layer functionality can be divided into a decision part (for handover, scheduling) from an action part (processing the handover via RRC, applying the scheduling decision via MAC). In this sense, SD-RAN separates the decision part (CP) from the action part (UP) and centralizes the decision part in the controller, unlike the CP protocol such as HARQ that remains in the base station.

As in SDN, a southbound protocol connects an SD-RAN controller to the base stations. However, to date, there does not seem to be a consensus on a southbound protocol. Given that a large industry consortium named O-RAN[3] recently standardized an SD-RAN protocol, which is also the basis of FlexRIC (presented in Chapter 5), we provide a primer for this protocol in the following.

E2 Overview

The southbound interface for interconnecting a RAN controller and RAN network functions is the E2 interface [34] as shown in Figure 2.8. The RAN elements are called

[3] See www.o-ran.org.

Table 2.1 – E2 Terminology.

Abbrev.	**Meaning**
RIC	RAN “Intelligent” Controller
E2	Interface RAN/RIC
E2 Node	RAN under RIC control, e.g., eNB, CU, DU
RAN function	Controllable functionality within E2 Node
E2AP	Protocol for E2 Management and service encapsulation
E2SM	Specification for information exchange (e.g., statistics: E2SM-KPM)
Service Message	Base E2SM exchange messages, “RIC functional procedure”
RIC Service	Message flows to implement information exchange (cf. Figure 2.9)

“E2 nodes” and can refer to an eNB, gNB, or just a part thereof, e.g., CU or DU, while the RAN controller is called the near-realtime RAN Intelligent Controller (RIC). Table 2.1 lists the E2 terminology that is used in E2, and which we adopt in this thesis.

The E2 interface has been defined by O-RAN in an attempt to provide a standardized interface for multi-vendor interoperability [34, Section 5.1] in order to:

- Send pre-defined information on pre-defined trigger events from the RAN to the RIC (*reports*),
- Send *control* messages and *policies* from RIC to RAN, and
- Enable the RIC to process procedures at the RAN’s place (*insert*).

In short, the objectives of E2 are “exposure of selected E2 Node data” (configuration, statistics, measurements, ...) and “enabl[ing] the Near-RT RIC to control selected functions” [34, Section 5.2] of the RAN, such as triggering procedures, installing policies, RAN configuration, etc.

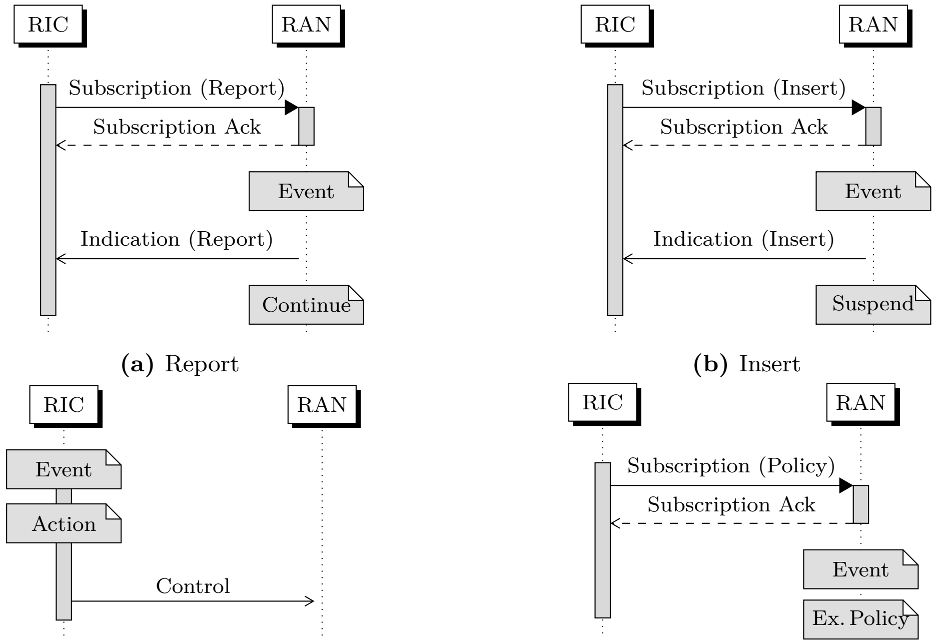
One RIC controls multiple E2 nodes. Each E2 node has an agent, managing the E2 connection, and a number of RAN functions. RAN functions refer to controllable functionality within an E2 Node (e.g., triggering a handover). The RIC is a controller that provides a terminating point for all E2 Nodes, hosts external applications (i.e., xApps), and additional services (e.g., database functionality).

The purpose of the E2 interface is to connect xApps to RAN functions[4], as highlighted in Figure 2.8 in red. As can be seen, there exists a many-to-many mapping between xApps and RAN functions. This link allows xApps to control the RAN functions. The interaction between the RIC and xApps on one side and the E2 node and RAN functions on the other is enabled through (i) the E2 application protocol (E2AP) [35] and (ii) E2 Service Models (E2SMs).

E2AP

E2AP provides signaling procedures to provide basic connectivity between the RIC and E2 nodes. The protocol messages are divided into two classes:

[4] Note that the E2 standard considers xApps to be part of the controller, whereas in SDN applications are separated from the controller through the northbound interface.



(a) Report

(b) Insert

(c) Control

(d) Policy

Figure 2.9 – The four basic "services" that are specified in E2SMs to define the interaction between an xApp and a RAN function.

E2 Global Procedures are messages for connection management between a controller and the agent, such as connection setup, or reset.

E2 Functional Procedures are messages specifically destined to RAN functions within the E2 agent for functional message exchange. They are further divided into (1) *subscriptions* to allow xApps to subscribe to even triggers in RAN functions, such as RAN statistics, (2) *indications* which allow RAN functions to forward information to the RIC, such as a specific statistics report, and (3) *control* messages through which xApps execute a procedure within a RAN function.

E2SM

Until now, the actual message content for interactions between an xApp and a RAN function has not been described. This is the role of E2 Service Models (E2SMs), which are specifications in their own right. Each E2SM provides the "service" that a RAN function can fulfill. There are four basic actions/services named *report*, *insert*, *control*, and *policy*, each of which is transported through E2AP procedures.

Reports contain information that are sent from the E2 nodes to the RIC/xApps. As it can be seen in Figure 2.9a, an xApp sends an E2AP subscription for requesting a report. The E2SM specifies the types of reports ("action definition") that can be sent and the trigger events that lead to them. Once the trigger is detected, the RAN function sends an E2SM report that is transported through an E2AP indication message, and continues processing.

Inserts are messages sent from E2 nodes to inform an xApp of an event. As it is shown in Figure 2.9b, an xApp first subscribes for an insert message via an E2AP subscription. Once the trigger has been detected, the RAN function sends an E2SM insert message that is transported via an E2AP indication, and waits for a response from the controller.

Control messages are sent from the RIC/xApp to the RAN function to execute an operation, as highlighted in Figure 2.9c.

Policies are predefined operations that the RAN function should execute upon a trigger as shown in Figure 2.9d.

Standardized E2SMs

At the time of writing this thesis, two E2SMs have been standardized:

1. Performance metrics (E2SM-KPM) [36] defines various report types on periodic timer expires.

2. Network interface (E2SM-NI) [37] is a service model for interface manipulation, supporting interfaces such as X2, S1, etc. It defines (i) E2 report messages for message exchange on X2, S1, etc. interfaces, (ii) insert messages, copying the occurred interface message to the xApp, (iii) control messages to inject interface messages into interfaces, or (iv) policies to help the RAN function to decide how to proceed with a message on a certain interface.

For instance, an xApp might have subscribed for an E2SM-NI insert message when SgNB Addition Request messages arrive over the X2 interface. This is the means for adding a new UE to an NR cell in a 5G-NSA scenario. Hence, the RAN function sends the insert message, to which the xApp might respond with a control message to decide on the addition request, either rejecting or acknowledging such UE.

In another scenario, an xApp subscribed to E2SM-KPM reports for PDCP in the same NR cell to evaluate cell performance. Through these report messages, it learns that the cell is close to overload, and sends a E2SM-NI subscription with a policy to reject all future SgNB Addition Requests. This shows that not only services from one E2SM might be combined, but also across E2SMs.

To summarize, E2AP provides basic exchange mechanism for the E2 Node–RIC interface, and message primitives to implement the services of type report, insert, control, and policy. The meaning of these services is defined by E2SMs (service models), which define the message content that is transported via E2AP.

2.3 RAN Virtualization

Early base station deployments, especially in the pre-4G era, consisted of dedicated hardware in order to meet the stringent timing requirements of the RAN. During the 4G era, software-implementations of RANs appeared thanks to more efficient general-purpose processors. OAI [22], [23], which is used as the primary RAN platform in this thesis, is

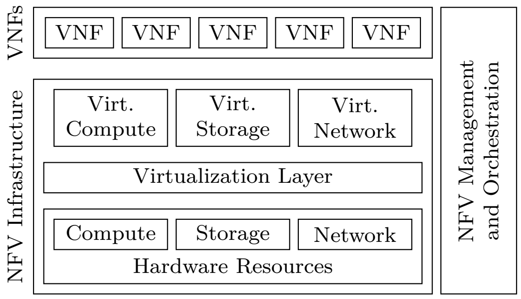


Figure 2.10 – Overview over network function virtualization.

one example. RAN virtualization is also relevant in the context of disaggregation (see Section 2.1.3), as the standardized interfaces allow a part of the RAN, e.g., CU and DU, to be fully virtualized, while the rest, i.e., the RU including all radio hardware, is a physical appliance.

Related to RAN virtualization is the concept of Network Function Virtualization (NFV). It is a paradigm of virtualizing network functions, such as networking equipment like routers, through virtualization techniques such as virtual machines. The approach promises to reduce cost by reusing standard, general-purpose instead of proprietary, specialized hardware, improve flexibility by dynamically deploying network functions through automation processes, and to cut down power usage by scaling appliances according to the current workload. NFV allows multi-tenancy in the network, whereby each tenant deploys the required virtual network functions over a shared infrastructure.

The NFV architecture is standardized by the European Telecommunications Standards Institute (ETSI) in GS NFV 001 [38] and shown in Figure 2.10. Generic compute, storage and network hardware build the foundation, forming the hardware resources. They are virtualized by a virtualization layer into virtualized compute, storage and network resources. Virtual Network Functions (VNFs) are then deployed over the NFV infrastructure. A separate NFV management and orchestration entity manages the life-cycle of services through VNFs and their relationships, and orchestrates hardware and software resources in the system.

Example VNFs are 5GS network functions, such as the AMF in the CN. A non-virtualized network function, such as a (hardware) RU, is a Physical Network Function (PNF).

Cloud computing builds on the notion of NFV, where a central or (especially in the context of the RANs) edge cloud runs VNFs. Cloud-native principles[5] have emerged in order to efficiently operate software in the cloud environment, such as lightweight virtualization and isolation through containerization using operating system virtualization, or a micro-service architecture towards which the 5GC lends with its service-based architecture. The aspects of agility and scalability, i.e., fast deployment of network slices and scaling to network load, are highlighted as outcomes of cloud-native architectures.

[5] Cf. the Cloud Native Computing Foundation CNCF: `https://www.cncf.io/`

Chapter 3

Related Work

As discussed in Chapter 1, this thesis considers SD-RAN and slicing to realize service-oriented RANs. In this chapter, we review the literature on these topics and assess their relevance and shortcomings that are addressed in this thesis.

3.1 Software-Defined Radio Access Networking

SDN. SDN introduces programmability in computer networks by decoupling the data and the control plane, while allowing to centralize decision-making in a controller with a global view of the network (see also Section 2.2). Although the concept of programmable networks is not recent [31], SDN came to wider success in the form of OpenFlow [32], an *abstraction* of a network switch ("OpenFlow switch") to be managed by a controller. Soon after the inception of OpenFlow, the first SDN controller for wireless networks, OpenRoads [39], [40], extended OpenFlow to Wi-Fi and WiMAX networks. However, OpenRoads focuses on mobility management and has no support for managing radio resources.

Going one step further towards a slicing of SDN networks, FlowVisor [41] introduces a proxy that slices control of a shared network of OpenFlow switches and exposes each slice towards a slice-specific controller. It uses OpenFlow as an abstraction layer, slices and virtualizes the network resources to allow multiple SDN controllers to concurrently control their (virtual) network. To northbound controllers, FlowVisor appears like a switch by exposing the southbound OpenFlow protocol at its northbound again.

The OpenFlow protocol exposes a programmable interface of a switch, and defines a number of protocol headers and fields on which to match and execute corresponding actions. P4 [42] introduces an abstraction of the switch forwarding model, allowing to reprogram how switches process headers to match arbitrary fields, and which is configured through OpenFlow. This allows the controller to redefine packet processing in a packet protocol-independent way, and is (switch) target independent through the definition of the P4 language.

SD-RAN Platforms. The SD-RAN concept is the application of SDN principles in the RAN. It is studied in several conceptual and theoretical works, arguing a centralization of CP functionalities for better coordination of the base stations in the RAN. The work in [43] argued that the application of SDN in RANs would allow a better management of radio resource allocation, admission control, handover due to centralization at the controller, but pointed out that such implementation was missing at that time. Fully centralized architectures are proposed in OpenRAN [44] and SoftAir [45]. However, these architectures could be overwhelmed by the required real-time control, given the networking delay between the controller and the underlying RAN. To overcome these problems, the SoftRAN architecture [46] refactors the control functions into centralized and distributed ones based on the time criticality and whether such function is necessary in a central view. SoftMobile [47] abstracts CP processing across network layers based on their functionalities and can perform the control functionalities through an Application Programming Interface (API).

A number of SD-RAN controller implementations were proposed, bringing a separation of CP and UP with the associated programmability. FlexRAN [33] pioneered the first practical open-source implementation of an SD-RAN controller on top of OAI [22], [23]. It does not completely separate the CP from the UP as in the SDN sense, i.e., some CP protocol operations such as HARQ remain at the base station, but rather separates the decision part (control logic) in the CP from the action part in the UP in order to perform centralized decisions. FlexRAN provides the separation through a custom southbound protocol with a focus on real-time RAN control applications. Unfortunately, the southbound protocol is tailored towards specific control operations, limiting its extensibility, and it lacks modularity, limiting the contributions it received since its inception. Another implementation of an SD-RAN controller, the EmPOWER platform [48], focuses on management of the RAN without support of (real-time) RAN control, limiting the applicable use cases. Moreover, similarly to FlexRAN, it uses a non-standardized custom southbound. It targets multi-RAT environments for Wi-Fi (OpenWrt) and LTE (SRSRAN), but the southbound protocol differs between these RATs, raising the question of interoperability between these implementations.

More recently, the industry initiative O-RAN was formed to promote open interfaces in the RAN to enhance interoperability, and “network intelligence”, aiming to manage the increasingly complex cellular network [49]. As part of this effort, an interface between base stations and controllers (E2) [34] was standardized. Furthermore, O-RAN developed a reference SD-RAN platform, the O-RAN RIC [50][1], to validate their architecture. O-RAN proposes that the resources of the RAN should be controlled through E2SMs that are connected to xApps. In this manner, they aim to virtualize and control RAN resources efficiently. O-RAN’s RIC is containerized and deployed through Kubernetes, consuming a non-negligible amount of resources (e.g., 160 GB of storage for a default installation[2]). Such architectural design and implementation decisions clearly violate the ultra-lean design and flexibility principles of 5G to minimize the platform footprints.

[1] Throughout this thesis, we refer to its “Cherry” release.

[2] Source: Getting started guide of Cherry release, `https://wiki.o-ran-sc.org/display/GS/Near+Realtime+RIC+Installation`

Table 3.1 – Taxonomy of paradigms realizable through the FlexRIC SDK.

Paradigm	**Platform**	**Function**
SD-RAN Control	[33], [48], [50]	Monitoring, Handover
Recursive	FlowVisor [41]	Proxy
Hypervisor	Orion [53]	Multi-service remote scheduling

Table 3.2 – Comparison of existing SD-RAN and slicing platforms.

Criterion	**RIC** [50]	**FlexRAN** [33]	**EmPOWER** [48]	**Orion** [53]	**FlexRIC**
Specialization	✘	✘	✘	✘	✔
Multi-service	✘	✘	✘	✔	✔
Multi-RAT	✔	✘	(✔)	✘	✔
E2	✔	✘	✘	✘	✔
Fast/RT	✘	✔	✘	✔	✔

In essence, there exists a considerable amount of use cases, e.g., beam coordination or coordinated multipoint transmission/reception (CoMP), for which O-RAN design will not be able to provide the required low latency, making it inflexible. Similarly, the OpenNetworkingFoundation announced an O-RAN compliant E2 agent and microservices-based RIC [51], [52]. Unfortunately, the implementation is closed, and thus, we could not investigate further.

To the best of our knowledge, there is no platform that provides SD-RAN according to the 5G principles of flexibility, forward compatibility and ultra-lean design. We propose the FlexRIC SDK in Chapter 5 to enable SD-RAN controllers that can be specialized towards diverse requirements. FlexRIC is designed to consume the resources demanded by the service, without imposing extra features when they are not required. By adopting the E2 protocol structure, it remains fully-compliant to industry efforts. Its modular SDK structure allows a smooth composition of specialized controllers, and, if isolation is a concern, its different components can be easily ported to a micro-services architecture.

Unlike FlowVisor for SDN, there exists no proxy-like virtualization that slices the RAN, virtualizes its resources, and recursively exposes the SD-RAN control towards multiple northbound controllers. The closest of such controller is Orion [53], a hypervisor to slice the radio resources of a base station (cf. also Section 3.2.3). Thanks to its flexibility, FlexRIC allows implementing different controller paradigms through controller specializations, as shown in Table 3.1, including virtualization proxies or hypervisors like FlowVisor or Orion. Leveraging the flexibility of FlexRIC, we develop a virtualization layer, similar to FlowVisor, that allows multiple controllers of different operators to concurrently control their (virtual) network over a shared infrastructure.

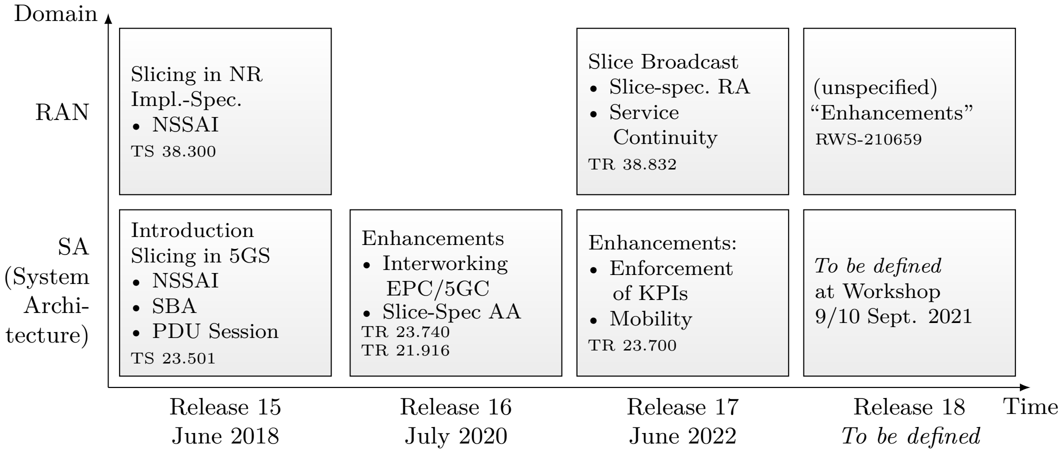
Table 3.2 lists publicly available SD-RAN controllers, and additionally the Orion platform to compare FlexRIC's capabilities. FlexRIC can implement different controller specializations (e.g., as shown in Table 3.1), and allows exposing control to multiple northbound controllers (multi-tenant). It is multi-RAT capable, based on E2 for guaranteeing interoperability, while being lightweight and fast.

Programmability in the infrastructure. The RAN has to be modified to provide the necessary programmability in the infrastructure. A number of works deal with slicing of the radio resources in the MAC layer and its abstraction, similar to how OpenFlow abstracts a network switch. The two-level scheduler concept [54] separates slice and user scheduling and integrates it with FlexRAN. The work does not detail a protocol, nor does it use the offered programmability for dynamic slice deployment or differentiated slice scheduling algorithms. Similarly, Costanzo *et al.* [55] propose a slicing framework with SD-RAN programmability through FlexRAN, but the resource slicing is static, providing limited flexibility for the programmability aspect of the network. In the context of EmPOWER, a flow abstraction of slices has been proposed [56], but the applicability of the flow abstraction to radio resources is unclear, as the MAC layer does not deal with packet flows. Also, it seems the slice algorithm cannot be changed either. Finally, none of the works address how the flexibility of the NR protocol stack might be used in such a context, e.g., towards mmWave scheduling. In summary, the above works provide a certain level of programmability, but neither do they address how to adapt the scheduler towards novel 5G/NR deployment scenarios, nor how to customize slice scheduling towards service requirements, e.g., QoS, and couple it with a flexible abstraction mechanism for programmability. To address these issues, in this thesis we propose a MAC scheduling framework in Chapter 6 that can be flexibly adapted both to 5G deployment scenarios and service requirements. It allows to decouple the slice and user scheduling algorithms, introduces programmability to describe and configure slices depending on the selected slice algorithm, enables dynamical slice management, and is open to be extended with custom slice and user scheduling algorithms.

For completeness, we report on a number of earlier works that restructure the RAN architecture itself to increase the level of programmability. OpenRadio [57] proposes a decoupling of wireless protocols into processing and decision planes for digital signal processor. The decision plane can be expressed through a declarative language to assemble a protocol. Similarly, PRAN [58] introduces flexibility in the LTE PHY by allowing to chain the data path blocks with performance guarantees on commodity services. CloudMac [59] provides an OpenFlow abstraction for MAC to process Wi-Fi MAC frames at virtualized access points. This effectively centralizes MAC processing in a central cloud through the OpenFlow abstraction.

3.2 Network Slicing

Network slicing considers a collection of virtual overlay networks over a physical network [60], where each overlay network constitutes a slice. It allows replacing multiple dedicated networks with virtual networks over a shared infrastructure with associated benefits, such as cost savings due to a single network instead of multiple dedicated ones. The concept of slicing involves a number of principles, such as automation for on-demand deployment, isolation between slices, customization of the slices to meet the slice's requirements, or programmability to allow to control the allocated resources [61].



Domain	Release 15 June 2018	Release 16 July 2020	Release 17 June 2022	Release 18 *To be defined*
RAN	Slicing in NR Impl.-Spec. • NSSAI TS 38.300		Slice Broadcast • Slice-spec. RA • Service Continuity TR 38.832	(unspecified) “Enhancements” RWS-210659
SA (System Architecture)	Introduction Slicing in 5GS • NSSAI • SBA • PDU Session TS 23.501	Enhancements • Interworking EPC/5GC • Slice-Spec AA TR 23.740 TR 21.916	Enhancements: • Enforcement of KPIs • Mobility TR 23.700	*To be defined* at Workshop 9/10 Sept. 2021

Legend: RA = Random Access AA = Authentication & Authorization

Figure 3.1 – 3GPP slicing and changes in releases.

Standardization. Apart from the research perspective, a number of industry consortia and standardization organizations have highlighted the architectural need for slicing such as ITU [2], NGMN alliance [4], and GSMA [62].

The 5GS and NR support slicing since their inception in Release 15 (see Figure 3.1). A slice is signalled within a Public Land Mobile Network (PLMN) through the Network Slice Selection Assistance Information (NSSAI) [7], which is sent by a UE after random access in order to assist the network with slice selection. The NSSAI is further divided into the Slice/Service Type (SST), which can be either the eMBB, URLLC, or “massive Internet of Things” (mIoT), and the Slice Differentiator (SD) to differentiate slices with the same SST. The service-based architecture (SBA) allows to customize the CN for specific slices. The RAN should respect principles of slicing such as the support of QoS and resource isolation, but slicing is otherwise implementation-specific [9]. A PDU session is created for each slice per UE, allowing the handling of slices in the core network and appropriate mapping to the radio bearers in the RAN through the SDAP layer. In Release-16, some changes for 4G/5G CN interworking and slice-specific authentication and authorization were added. In the future Release-17, the RAN can broadcast slices to enable slice-specific random-access and service continuity, i.e., cell reselection, and further enhancements are planned for the overall 5GS. The future Release-18 slicing enhancements will bring (yet unspecified) enhancements, and a workshop for the 5GS slicing enhancements proposals is to happen at the beginning of September 2021.

Theoretical Approaches. Numerous works consider slicing from a mathematical point of view. Su *et al.* [63] suggest that resource allocation for network slicing can be broadly grouped into general optimization problems, game-theoretic economic models, models based on prediction or machine-learning, and failure recovery methods for unpredictable network events. For instance, Fossati *et al.* [64] formalize the slicing problem for multiple heterogeneous resources, e.g., networking bandwidth and CPU, assuming that resource scarcity in the network. They show that such multi-resource allocation leads to more efficient operation, and increases the satisfaction unless all resources are equally congested. OKpi [65] similarly allocates network and computing resources, optimizing for heterogeneous performance indicators such as availability and reliability, through a combination of graph theory and optimization. Guan *et al.* [66] use complex network theory to deploy different services such as eMBB, URLLC, and mMTC. Halabian [67] uses an auction-based approach to solve the problem of distributed resource allocation. Similarly, Qin *et al.* [68] propose an architecture for resource allocation in a heterogeneous SD-RAN and solve it using an auction mechanism. AZTEC [69] anticipates resource allocation for slices using a deep learning approach. Gutterman *et al.* [70] predict RAN resource usage using Long Short-Term Memory (LSTM) neural networks for slice provisioning.

3.2.1 Core Network Slicing

CN slicing is relatively advanced, and has already been added to the (4G) EPC. Dedicated core networks (DECOR) have been standardized, where a dedicated CN might be selected based on subscriber information in the HSS [71]. An enhanced version (eDECOR) further uses assistance information from the UE to select a dedicated CN [72].

In the 5GC, network slicing is enabled through the Network Slice Selection Function (NSSF), which supports the selection of network slice instances for a UE and the set of candidate AMFs to be used [7].

In general, network slices in the CN can be orchestrated and customized straightforwardly through VNFs running on a common infrastructure, utilizing NFV [73]. As such, standard virtualization and containerization techniques can be used to provide dedicated core network slices [74], [75]. The application of cloud-native principles [76], such as containerized micro-services, improves the life cycle management of slices based on NFV. A generic architecture for core slicing and customization is presented in [77]. It has been pointed out that slicing can be modeled as a special problem of virtual network embedding [78], and Prados-Garzon *et al.* [79] propose a mathematical model for network slice planning in the core network.

3.2.2 RAN Resource Slicing

RAN resource slicing can be traced back to the RAN sharing concept. In RAN sharing, multiple operators share the RAN resources and either still operate dedicated core networks (multi-operator core network, MOCN), or even share (parts) of the core network (gateway core network, GWCN) [80]. Another form of RAN sharing is sharing the infrastructure, but not the actual radio resources (sometimes denoted as multi-operator ran access network, MORAN). RAN sharing can be seen as part of the greater framework of spectrum sharing [81], which addresses the general issue of spectrum scarcity, at least considering the sub-6 GHz frequency ranges.

RAN slicing can be enforced on different levels, i.e., as early as during spectrum planning, using interference coordination mechanisms, at the packet scheduling, or during admission control [82]. A large body of work studies RAN slicing in terms of resource allocation at the packet level, as it allows isolating resources while efficiently sharing resources (see also Richart *et al.* [83] for an overview of RAN resource slicing). NVS [24] slices the radio resources for two types of slice, data rate and resource capacity. NetShare [84] extends NVS to implement RAN sharing across many base stations using a central gateway, ensuring resource isolation and optimal resource distribution. The resource abstraction approach in [85] can support three types of resource requirements: physical RBs, virtual RBs, and virtual transport block. The work in [54] generalizes the concept to a two-level scheduler (slice vs. user scheduler). It abstracts radio resources from physical RBs to virtual RBs to accommodate slices, but no further customization of the RAN such as different algorithms is considered. The two-level scheduler approach is further formalized in [86], showing that a parameterization can strike a balance between isolation and multiplexing gains. The theoretic work of RadioVisor [87] slices the network in the form of resource sub-grids to isolate operators from each other in time, frequency, and space, which can then sub-slice their network according to subscriber attributes.

A number of optimization problems have been formulated to extend the problem towards latency-critical applications. García-Morales *et al.* [88] consider three classes of slices (non-deterministic, deterministic aperiodic, and deterministic periodic traffic), formulate a non-linear optimization problem, and solve a linear approximation. Similarly, Korrai *et al.* [89] consider an optimization problem for scheduling eMBB and URLLC. However, both works perform such optimization with queue updates happening only every frame, raising the question of realism.

Papa *et al.* [90] consider slice resource allocation using a Lyapunov optimization approach for rate and delay requirements. However, they only consider the average of these quantities, and thus, the average targeted delay might need to be set considerably low to keep Quality of Service (QoS) requirements for excess packet delays. The Earliest Deadline First (EDF) slice algorithm [91] explicitly takes into account the deadline requirements of slices for all services. Further, overriding policies can be formulated to ensure hard deadlines.

Nevertheless, none of above works consider the impacts of QoS on the RAN domain resources for three key types of resource requirements: physical (static) resources, throughput, and latency. Therefore, we propose a QoS-aware resource slicing framework including formulated utility functions in Chapter 6, which follows a weight-based scheduling approach. It is furthermore integrated into the MAC framework, highlighting the modularity and extensibility of the framework. (We also implement the NVS [24] and EDF [91] slice scheduling algorithms in the framework.)

Joint slice/user scheduling. The above works all consider a separation of scheduling phase into slice scheduling and user scheduling in a single base station. This simplifies the scheduling phase by reducing the number of scheduling dimensions, but could lead to suboptimal slicing decisions. A joint scheduling of the slice and user scheduling phases is optimal [92]. On the other hand, CellSlice [93] avoids a slice scheduling phase by setting the Guaranteed Bitrate (GBR) of bearers of the respective slice UEs such that slice resource isolation is maintained, but does not provide any customization. Note that this requires a scheduler that schedules according to GBR[3]. Nojima *et al.* [94] design a slice resource allocation method by limiting the resources the PF scheduler can schedule per slice with an advantage for simplicity, but do not allow any customization possibilities; additionally, there is no proof of optimality. Note that like the above two-level based approaches, it is possible to implement the joint slice/user scheduling algorithms in the MAC framework proposed in this thesis.

Machine-learning approaches. A number of papers consider machine learning approaches to slice resources. Aijaz [95] proposes a radio resource slicing strategy across multiple base stations, which is based on reinforcement learning and particularly adapted to haptic communications (e.g., the tactile internet). Albonda and Pérez-Romero [96] propose a RAN slicing algorithm for eMBB and Vehicle-to-Everything (V2X) services and formulate an optimization problem to maximize overall resource utilization, solving it using an offline reinforcement learning approach due to the high complexity of the problem. Raza *et al.* [97] use reinforcement learning to decide about admission of slices with different resource requirements while maximizing the profit of an infrastructure provider. Finally, deep-reinforcement learning is used in [98] to allocate resources, also considering that the number of slices varies. These approaches might be used to implement slicing in the MAC framework; due to the additional complexity of the machine-learning part, we do not further consider those in this thesis.

NR Bandwidth Parts. Bandwidth allocation, based on BWPs in NR, is used to adjust the observed radio spectrum depending on UE and service requirements. BWPs allow saving energy for idle UEs through a reduction of the used spectrum, also reducing interference, while providing a larger bandwidth on-demand [99]. Baccelli and Kalamkar [100] provide a statistical model for BWP scheduling in device-to-device scenarios, and show that such adaptation can provide better success probability for users with small bandwidth requirements, while users with larger requirements achieve better Shannon throughput, confirming the advantages of BWPs.

Random Access and Admission Control. The above works do not or only partially model the random access and admission control stages before a slice is admitted for radio scheduling. Mancuso *et al.* [101] develop a stochastic model for random access and RRC connection procedure for eMBB and URLLC services. The admission control is modeled as a Markov Chain in [102]. Additionally, the authors consider radio resource allocation, but it is fixed and based on the RRC Allocation and Retention Priority (ARP) for GBR bearers. No isolation for non-GBR is foreseen, which might be the case, e.g., for eMBB.

3.2.3 RAN Slice Customization

In recent years, RAN slicing with a focus on customization and extensibility is investigated to enable a multi-tenant RAN. In correspondence, openness of the RAN for customization

[3] OAI as the cellular network implementation used in this thesis does not implement such a scheduler at the time of writing of this thesis.

is of increasing importance to enable access for vertical players [18].

Initially, the concept of RAN-as-a-service [103] emerged as a way of running multiple services concurrently and efficiently in the context of a cloudification through C-RAN to enable customization of the RAN towards these services. Rost *et al.* [104] advocate for a flexible RAN through a parameterization of upper layers to realize RAN network slices over a hypervisor. However, the abstractions are not detailed, and no proof-of-concept is given. The parameterization concept is further investigated without a hypervisor by using slice descriptors [105] to configure the RAN towards different slices, but the authors do not consider dynamic service-specific functional customizations of the slices. The customization capabilities and trade-offs of the 5G RAN regarding slicing are investigated by Sexton *et al.* [21] with a particular focus on physical layer customizations for different slices. However, the customization of higher layers is not considered. The work in [106] analyzes gaps in the current 5G architecture and proposes a slice-aware programmable architecture including a cloud-enabled protocol stack for slice differentiation in the RAN. While the authors identify key slice customizations, e.g., the scheduler, those are not further explored.

While the aforementioned works consider customization, a functional isolation of slices, e.g., through RAN virtualization, is crucial to completely isolate slices from each other and enable customization through the service tenants [107].

With the disaggregation of the RAN into several network functions (CU-CP, CU-UP, DU), the RAN enables finer per-network function customization through virtualization of (parts of) the RAN following NFV principles (see also Section 2.3), and RAN disaggregation can offer the potentials of flexibility, scalability, upgradability, and sustainability via the RAN service chaining notion [108]. To this end, the 5G RAN network functions have been analyzed with respect to the different sharing options and the feasible level of customization and isolation for each RAN function [109]. Adamuz-Hinojosa *et al.* [110] harmonize 3GPP 5G standard and NFV architecture viewpoints to enable the translation of slice requirements into customized RAN functionality through VNFs. A slice-based "network store" architecture [75] proposes a collection of 5G network functions to chain service-specific functionality, similar to a mobile app store, on top of a common infrastructure. However, no prototype implementation is provided.

Zaki *et al.* [111] advocate functional isolation of the complete RAN of multiple operators through a hypervisor over physical resources, where each tenant (network operator) has its own virtualized base station. Nakao *et al.* use FLARE [74] to implement an end-to-end network slicing with virtualized base stations, without any multiplexing gain between the slices. Two works further investigate the idea of a hypervisor to share the physical resources of the Software-Defined Radio (SDR) between multiple RANs. The Extensible Virtualisation Layer (XVL) [112] shares the SDR to create virtual radios on demand. PV-RAN [113] follows a similar approach while using the Xen hypervisor to guarantee isolation between SDRs. However, SDR hypervisors yet have to show the feasibility of dynamic, possibly high-frequency sharing to achieve multiplexing gain.

A hypervisor with dynamic resource multiplexing has been implemented by Orion [53]. It isolates slices by abstracting and multiplexing the radio resources of each slice onto the common resources. However, Orion's virtualization is limited to radio resource scheduling, and does not provide support for connection, mobility management, or flow control. The approach has the drawback of a “fat virtualization” approach where a stateful network function implements the control plane of a complete service on top of the hypervisor. Additionally, since it relies on a single hypervisor at the base station, its design limits its deployment in disaggregated base stations, and no coordination between hypervisors is foreseen to avoid interference. The implemented prototype is not publicly available.

The RAN runtime [114] allows to dynamically slice and customize a RAN to meet slice requirements, such as isolation, sharing, and customization options. The system fulfills RAN customization requirements by chaining RAN layers in a customized manner, but does not address micro-customizations for individual control plane logic and possible state conflicts between the slice-specific RAN layers. Further, in a follow-up work [115], the authors describe how the RAN runtime can be leveraged to enable single- and cross-domain applications to monitor and control the underlying RAN. In total, no implementation of a RAN runtime for custom RAN chaining is provided.

Table 3.3 summarizes a selection of the existing work on RAN slicing customization and extension and compares it with respect to the used approach, the level of isolation, what processing elements can be customized and the corresponding overhead, and whether these processing elements are stateful or not. We observe that the existing work only partially combine and apply SDN/SD-RAN, NFV, hypervisor approaches and cloud-native principles in the design of a multi-service RAN architecture. Simultaneously enabling programmability, virtualization, customizability and extensibility, as well as agility and scalability of the fundamentally shared multi-tenant RAN infrastructure, is a main, yet unresolved, challenge addressed in this thesis. To this end, Chapter 4 describes an abstraction of services, using a combination of SDN/SD-RAN, NFV, and cloud-native principles to allow controlling, customizing, and extending specific RAN control endpoints, e.g., scheduling. Applying the ideas of this approach, we further implement an SD-RAN virtualization layer in Chapter 7 that provides concurrent programmability for multiple SD-RAN controllers over a shared infrastructure.

Table 3.3 – Comparison of selected RAN customization approaches in the literature.

Name	**Approach**	**Isolation Lvl.**	**Cust. Granularity**	**Cust. OH**	**Processing State**
FlowVisor [41]	OpenFlow Proxy	Service	OpenFlow	Light	N/A
FlexRAN [33]	SD-RAN Controller	Base station	Push control logic	Light	N/A
Ferrús *et al.* [105]	Configuration/SDN	Service	No	N/A	N/A
Sexton *et al.* [21]	Configuration/SDN	Service	No	N/A	N/A
Network Store [75]	NFV	Service	Network Function	Fat	Stateful
Zaki *et al.* [111]	NFV	Base station	Network Function	Fat	Stateful
Nakao *et al.* [74]	NFV	Base station	Network Function	Fat	Stateful
PV-RAN [113]	Hypervisor/SDR	Base station	Network Function	Fat	Stateful
Orion [53]	Hypervisor/MAC	Service	Complete CP	Fat	Stateful
RAN Runtime [114]	Multi-level SDN Controller	Service	RAN Layer	Medium	Stateful RAN layer
BS Abstraction/Ch. 4	*Micro-SDK* Composition	Service	Per control endpoint	Light	Stateless Micro-services
SD-RAN Virt./Ch. 7	Proxy for E2SMs	Service	Per E2SM	Light	N/A

Lvl. = Level, Cust. = Customization, OH = Overhead

Chapter 4

Base Station Abstraction

4.1 Introduction

As has been pointed out in the introduction, a number of trends lead to increasingly complex RANs, and we observe that it has to serve multiple dimensions as shown in Figure 4.1. First, disaggregated Base Station (BS) deployments allow centralizing RAN processing at centralized or edge clouds and flexibly composing BS functionalities, but complicate the management of individual BSs. Second, network densification allows increasing area traffic capacity and connection density, but requires better coordination in the RAN across BSs and among services. Third, 5G does not only provide a specific RAT but also utilizes the available spectrum bands (e.g., mmWave) and access technologies (e.g., 4G, 5G) in a heterogeneous manner to fulfill diverse service requirements. Therefore, a proliferation of RATs entails an increased complexity for the management of the network, as UEs need to be managed across these RATs. Additionally, complexity is exacerbated considering that 5G networks need to multiplex a variety of services with heterogeneous performance and customization requirements. Regarding verticals, they wish to provide their network without having to run a dedicated network, while maintaining control over their virtual network.

We envision an abstraction of the RAN to efficiently manage the increasingly complex RAN. In order for such an abstraction to be useful, it is first of paramount importance to reduce the complexity from a representational view to simplify the management of

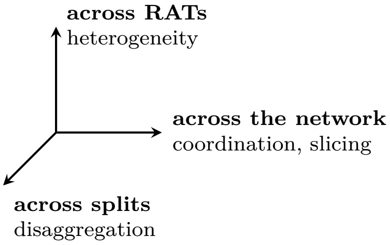


Figure 4.1 – The 5G network can be deconstructed into a split-network-RAT view.

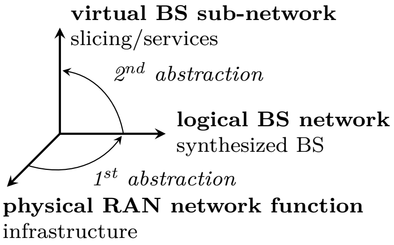


Figure 4.2 – The two-level abstraction creates the physical, the logical, and the virtual representations of the RAN.

and coordination within the RAN. Furthermore, it is necessary to describe services that are embedded into the RAN, i.e., services that are transported through a slice, including their required RAN customizations. Also, in order to go one step further towards the service-oriented RAN, abstractions need to be found to enable multiple service-specific controllers to manage their virtual networks. Finally, there is a dependency between the RAN and TN resources, since the RAN network functions in a disaggregated RAN need to be interconnected.

In this chapter, we conceptualize an abstraction of the RAN to manage the increasing complexity and describe the mapping and adaptation of services in the RAN. We start by considering a generic BS abstraction through a descriptor that disaggregates a BS into resources, processing, and state, in Section 4.2. Using this descriptor, Section 4.3 further develops the concept of a two-level abstraction from disaggregated RAN network functions to (1) a "logical" BS representation for complexity reduction and eased management, and to (2) a "virtual" BS for service-to-RAN mapping, using the FlexVRAN framework. Leveraging the descriptor and associated two-level abstraction, we provide a design in the form of the RAN Engine in Section 4.4 that multiplexes services onto a common RAN infrastructure while allowing the service owners to program the RAN towards their requirements using RAN control endpoints. In Section 4.5, we link the RAN to TN resources. To this end, we formulate an optimization problem for the composition of logical Base Stations (lBSs) following the first-level abstraction into a network of M DUs and N CUs with a single bottleneck midhaul between them, show its NP-hardness, and solve a simplified version. Finally, Section 4.6 discusses how to map the presented concepts to E2 as well as future work possibilities.

4.2 Generic Base Station Abstraction

We conceptualize the RAN through descriptors that simplify the description of available resources and (processing) capabilities of a (physical) base station, and define the required resources and processing customizations of embedded services. To link the base station and service description to the RAN infrastructure, we use a two-level abstraction.

As shown in Figure 4.2, the physical infrastructure using functional splits and various RATs can be abstracted to compose an lBS spanning the full protocol stack, e.g., 4G eNB or 5G gNB. Such an lBS denotes a "synthesized" representation of the physical network, and can be seen as an enabler for effective management and control of the network by reducing network complexity of heterogeneous deployments (e.g., distributed, disaggregrated, centralized). This lBS is further abstracted to provide a virtualized slice-specific virtual Base Station (vBS) for network slices, while still maintaining user plane programmability. A virtual sub-network is formed from a set of vBSs to reveal slice-specific resources, processing and state, corresponding to service requirements, in the whole RAN.

We resort to a representation comprising resources, state, and processing (RSP) to describe both lBSs and vBSs. This representation abstracts the underlying infrastructure independent of the RAT and deployment. For the first-level abstraction, which results in an lBS, we associate a *pipeline descriptor*. Further, the second-level abstraction has an associated *service descriptor* that describes a vBS of a service, including service requirements and slice customizations that should be mapped into the RAN.

Both pipeline and service descriptors consist in the following:

Resources describe the amount of radio resources available to a base station. This includes the operating bands, the numerology, and the available operating bandwidth (including the number of carriers). A number of performance indicators can be inferred, such as sustainable data rate, maximum number of devices, or minimum latencies. Given that the radio resource allocation among services is governed through a slicing algorithm according to SLAs (minimum rate, maximum delay, etc.), the service resources shall include such objectives.

State is the status of the base station CP/UP processing and the associated configuration that are built up during runtime. This typically includes the deployment configuration (e.g., separate CU-UP), RAN layer-related configuration, and (potentially ephemeral) user plane statistics for monitoring purposes.

Processing denotes the functional RAN control endpoints of the pipeline in the form of capabilities, i.e., functionality that might be customized or extended, such as scheduling. Processing is tightly bound to the resources and the state, since any processing decision (control plane behavior steering) operates on top of the current state, and shall respect the constraints imposed by the resources (see also Section 4.5). With respect to a particular service, the processing part lists the customization(s) of the pipeline when embedding the service into the RAN.

Above, the resources of a base station have been defined solely in terms of radio resources. In general, and corresponding to the split options of a base station into DU/CU-CP/CU-UP as defined by 3GPP [8], resources of a base station correspond to three dimensions:

- Radio resources at the MAC level (radio resource level, DU): physical resource blocks with a specific numerology which form a bandwidth part, component carriers, multiple directional beams/antenna ports, transmission powers, multiple transmission layers, or supplementary uplink. Note that while it might be argued that the physical layer possesses these resources, the MAC layer decides about their allocation, and we therefore consider all resources to belong to the MAC.

- Resources for flows at the SDAP/PDCP/RLC layers (flow level, CU-UP and DU): guaranteed and maximum bitrates, priority levels, packet delay budget, packet error rates, averaging window, and maximum data burst volume.

- Resources for connection management at the RRC layers (radio control level, CU-CP): maximum number of admitted UEs, maximum number of bearers, guaranteed and maximum bitrates, allocation and retention policy (ARP), and reflective QoS attribute[1].

Therefore, the resource part of a descriptor might furthermore integrate resources of additional RAN layers. In particular, a service descriptor might use these resources to describe specific service requirements to express SLAs, and which are used to map such service into the infrastructure. Note that some resources might be considered a configuration (such as the ARP value), and we include them for completeness within the resources.

Similarly, the state and processing parts of the descriptor affects all three dimensions. For example, processing on the radio resource level includes the scheduler and employed algorithms (user scheduling, slice scheduling, …), whereas processing on the flow level includes flow-level management entities (e.g., the traffic control scheduler) and algorithms, or RRC policies at the radio control level to execute such as handover algorithms.

Through these descriptors, it is possible to describe service requirements which need to be enforced when embedding services into the RAN. We identified the following properties that allow a description and efficient embedding of services into the RAN infrastructure:

- SLA/performance enforcement: SLAs and associated QoS requirements, such as maximum latencies, must be enforced when multiple services are present.

- Forward-/backward-compatible: the descriptor should be RAT- and deployment-independent to maintain compatibility to future technologies, and that services might be mapped to different RATs and deployments as long as all requirements can be fulfilled.

- Performance isolation: deviations in resource requirements, e.g., traffic spike of one service, should not affect other services, whose resources should be guaranteed at all times.

- Sharing: under-utilization of resources should be avoided by multiplexing resources between services such that resources not needed by one service are attributed to another service, resulting in a multiplexing gain.

- Functional isolation: services should be allowed to make individual control decisions, which only has impact on their slice, while other parts of the RAN are isolated from such decisions.

[1] Note that while the resources of flow and radio control levels are similar, the flow level resources applies to individual packets, while radio control resources refer to entire UE connections.

- Customization: the descriptor should allow to tailor the RAN function towards service requirements, e.g., specific handover algorithms.
- Extension: the descriptor should allow specifying processing customizations or extensions that should be replaced or added in the RAN, e.g., application traffic analysis.
- Flexible information granularity: the descriptor for services should allow a variable degree of information, ranging from a simple monitoring interface for specific key performance indicators to full RAN control, e.g., remote scheduling.

Finally, the descriptor does not only allow to describe the RAN in a more service-oriented way, but possibly also allows constructing service-specific "views", such as hiding deployment details (e.g., base stations splits) or specific topology details.

A lightweight and programmable RAN control endpoint abstraction allows to customize the processing behavior, where control endpoint refers to atomic CP functionality that might be customized, such as scheduling or a handover algorithm. This allows to have services incorporate slice-specific customizations and extensions into the infrastructure through their descriptor, and per-service programmability can be realized by providing access to such control endpoint.

The first- and second-level abstraction can also be seen in Figure 4.7 on page 51 in the context of the RAN Engine presented in Section 4.4. There, a first-level abstraction creates the pipeline description for an lBS with associated resources, processing, and state. Further, multiple vBSs are mapped to the infrastructure. They are described through service descriptors with associated resources, processing, and state, where some services have processing customizations or gain programmability through RAN control endpoints.

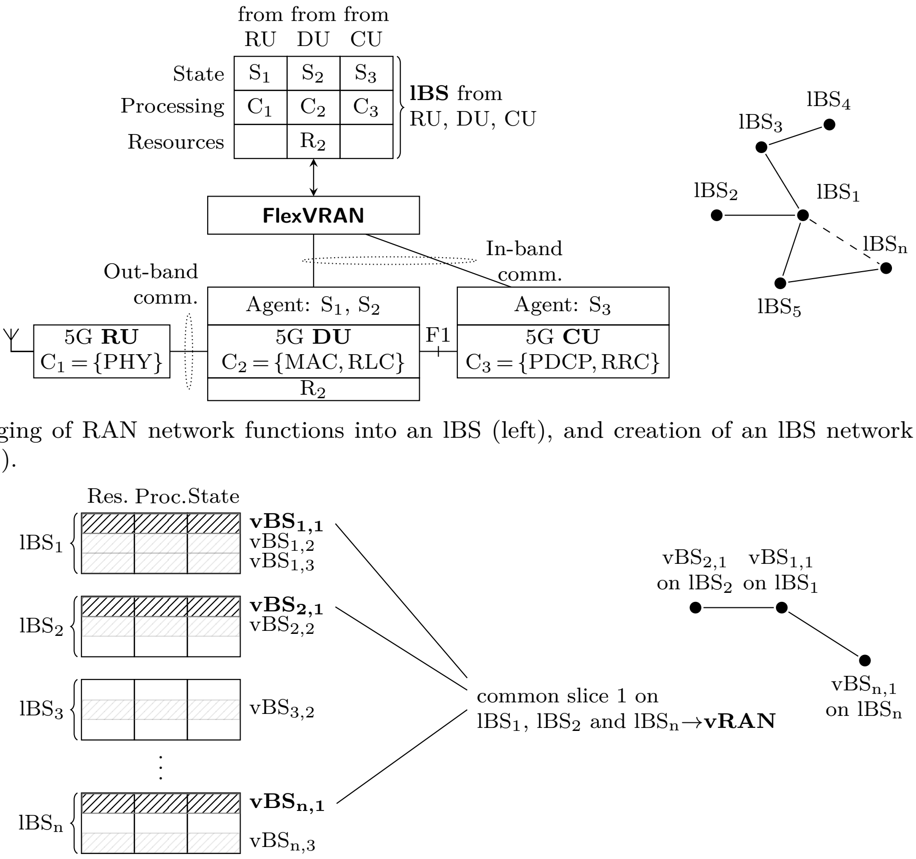
We introduced descriptors for describing the RAN infrastructure and service in order to address the challenges of embedding services into a heterogeneous sliced RAN. We will further detail the concept of the two-level abstraction and the resulting descriptors using the FlexVRAN framework in Section 4.3 to form a virtual sub-network that is revealed to a service owner. After that, we provide a design in the form of the RAN Engine to multiplex services onto the common infrastructure in Section 4.4.

4.3 Concept using the FlexVRAN Framework

4.3.1 FlexVRAN

The FlexVRAN framework, shown in Figure 4.3, consists of a controller that interacts with the RAN infrastructure. The heterogeneous physical RAN network functions host RAN PNFs/VNFs for CP/UP processing[2]. They are annotated with (a part of) the descriptor

[2] Some passive RAN network functions do not possess the RAN agent due to their limited processing capabilities and therefore rely on in-band control through other functions. For instance, as can be seen in Figure 4.3a, the RU relies on control through the DU. To this end, the operating functionalities and the relation toward other RAN network functions for the RUs are maintained explicitly by the connecting DU. Also, since the RU is not activated without the DU, we assume the DU to possess the radio resources.



(a) Merging of RAN network functions into an lBS (left), and creation of an lBS network topology (right).

(b) vBS are split from lBS (left), and a vRAN is embedded into the network topology (right).

Figure 4.3 – The operations of FlexVRAN to provide the (a) first-level abstraction and (b) second-level abstractions. A missing topological relation between lBS (dashed line) is present in the virtual sub-network.

and expose it to FlexVRAN. Due to RAN disaggregation coupled with heterogeneous deployments and technologies, it might however not be possible or feasible to provide a complete descriptor of one BS. Hence, for the first-level abstraction from physical to logical BS, FlexVRAN performs the operation of (1) merging descriptors of multiple RAN network functions into one lBS and (2) creating a view of the network topology and annotating the lBS with the complete descriptor, as shown in Figure 4.3a (the CU might be considered to have resources, but here we only consider radio resources). By matching capabilities of different RAN network functions reflecting the actual chaining, i.e. if $\forall \mathrm{C_i}, \bigcup_\mathrm{i} \mathrm{C_i} = \mathrm{C_{full}}$ with $\mathrm{C_{full}}$ being the capabilities needed for an operational BS for UE connectivity, the lBS is created as a unified representation of different deployments with a common data model. After this, the overall network topology is shown to represent the network cell structure in its spatial distribution, i.e., with the location of transmission/reception points/antennas.

To account for the heterogeneity of the underlying system, stemming from multi-vendor use-case-driven deployments, the descriptor could be represented as a collection of service models[3]. The descriptor therefore reflects the capabilities of the RAN functions in the infrastructure, along with their resources and state (as will be detailed exemplary for the MAC resources in Chapter 6). The merging operation does not reduce the informational content, but consolidates it on a per-cell basis, and possibly annotates additional metadata, such as the functional splits. Due to this direct relation of the physical and logical representations, the infrastructure owner is able to perform a direct mapping between those two.

FlexVRAN performs a second-level abstraction from lBS to vBS, creating a virtual sub-network specific to each service owner. As shown in Figure 4.3b, it performs the operation of (1) splitting (slice-wise customizable) lBSs into vBSs and (2) embedding them into the logical network topology in order to reveal a customized view of the virtual network (vRAN) to the service owner. Splitting refers to exposing only a part of the resources, state and processing from the underlying lBS to each vBS in virtualized form. The specific virtualization mechanism depends on the protocol stack layer, e.g., a flow abstraction (flow level) vs. a resource abstraction (radio resource level). Radio resource abstraction and virtualization are described in Chapters 6 and 7, respectively. Embedding on the other hand maps the vBSs within the topology of lBSs while decoupling it from the actual geographical position. This withholds network topology information from service owners and allows easier management of resource conflicts among vBSs. For instance, a service owner might be an untrusted third party, or physical user location needs to be protected while verticals need to identify who uses their service. Similarly, a slice owner might see a network of vBSs which do not expose the actual geographical position, where vBSs might be moved from one lBS to the next following the user mobility pattern.

The above two-level abstraction enables (i) a coexistence of multiple services that share resources within the same RAN, (ii) enforces service SLA and resource isolation through suitable multiplexing of services on the radio resource, flow, and radio control levels, and (iii) restricts the information exposed to a particular owner, e.g., for security

[3] E2SMs will be used as service models in the later chapters of the thesis; see Section 2.2.2 for a description of what an E2SM is.

reasons. Through the processing capabilities, a service owner is able to detect which processing functions exist and might customize slice-specific functionality, e.g., handover control logics, as we will explain in Section 4.4.

Finally, one important task is to resolve conflicts occurring across different levels of abstraction. Ideally, abstractions should ensure conflict-free operation, e.g., as it is done in Chapter 7. If conflicts arise, they are resolved on the corresponding level of abstraction:

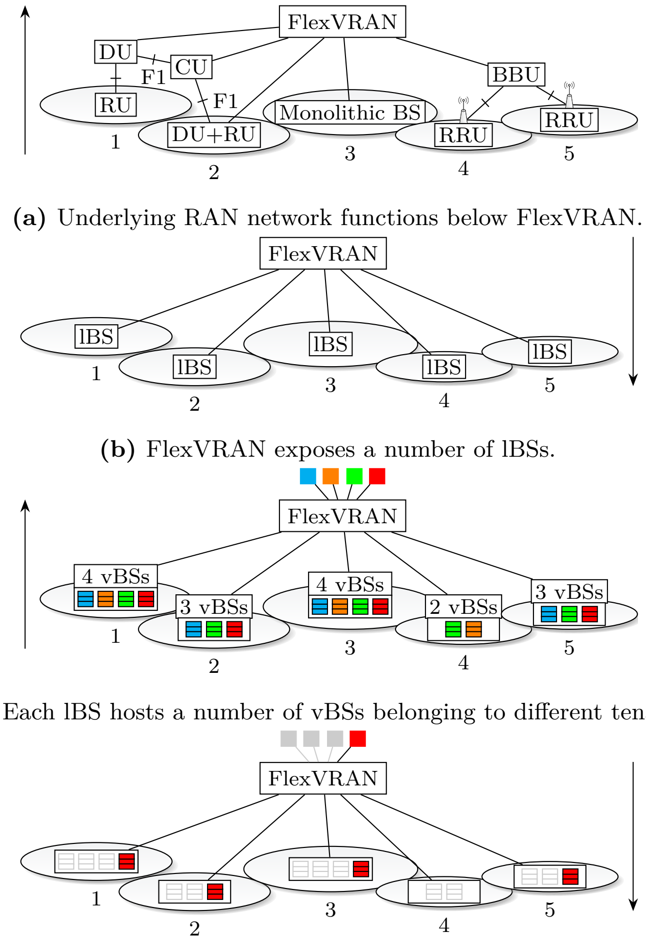
- The service owner can handle slice-specific conflicts in its vBSs, e.g., harmonizing the quality-of-experience witnessed by its subscribers across its virtual network.
- FlexVRAN is responsible for resolving the conflicts within lBSs, e.g., the proper chaining of RAN network functions to compose available lBSs (see Section 4.5), and between slices when customization attempts are made or new slices are to be instantiated.
- The RAN infrastructure resolves conflicts in the underlying CP/UP processing, states and resources, e.g., RAN customizations such as a user scheduling algorithm that cannot be loaded.

Nonetheless, there might be some conflicts that are not detectable by a given level; thus, the underlying level may reject or amend the control decisions. For instance, when the service owner aims to reconfigure a scheduling algorithm that is not available, the controller might temporarily delay such request while fetching the algorithm from a trusted network store.

4.3.2 Example

To illustrate FlexVRAN, we consider an example network as shown in Figure 4.4. It shows five different cells that are formed from multiple BSs with different degrees of centralization.

1. *Physical Infrastructure*: as shown in Figure 4.4a, the example network is heterogeneous. Cells 1 and 2 are deployed through a disaggregated BS deployment, consisting of a CU, a DU, and an RU. Cell 3 uses a monolithic BS, while cells 4 and 5 are deployed through a C-RAN setup.
2. *Logical Base Station*: As for the first-level abstraction, it merges the information of RAN network functions into lBS with associated pipeline descriptor and provides the topological information. Figure 4.4b shows the abstracted RAN. Note that information granularity is retained, e.g., split information is available. On the other hand, this allows to configure a BS on MAC and RRC layers, located on the DU and CU, respectively, without necessarily considering the used split. We formalize the composition of lBSs in Section 4.5.
3. *Virtual Base Station from a network perspective*: The second-level abstraction further includes the RAN slicing notion by splitting lBS into vBS and embedding



(a) Underlying RAN network functions below FlexVRAN.

(b) FlexVRAN exposes a number of lBSs.

(c) Each lBS hosts a number of vBSs belonging to different tenants.

(d) One service owner applies its control logic towards its vBSs.

Figure 4.4 – Examples for first- (a, b) and second-level (c, d) abstractions. The arrows represent the "view direction", colors represent service owners.

a virtualized and customized sub-network into the logical network topology for each slice owner. Figure 4.4c first shows the vBSs are embedded into the RAN infrastructure/lBSs, controlled through FlexVRAN. For instance, cell 5 embeds 3 vBSs.

4. *Virtual Base Station from a slice-owner perspective*: Finally, towards the service owners, FlexVRAN only exposes the corresponding vBSs (Figure 4.4d). Note that embedded slices might be moved from one lBS to another and the exact location might be hidden from a service owner, subject to SLA.

Figure 4.5 shows the process of abstracting physical RAN network functions into lBSs and from there into a vBS from a descriptor point of view as it might happen for cells 1 or 2 in Figure 4.4. Both agents of DU and CU provide information about the underlying RAN network functions, which is merged into a common lBS. This information includes

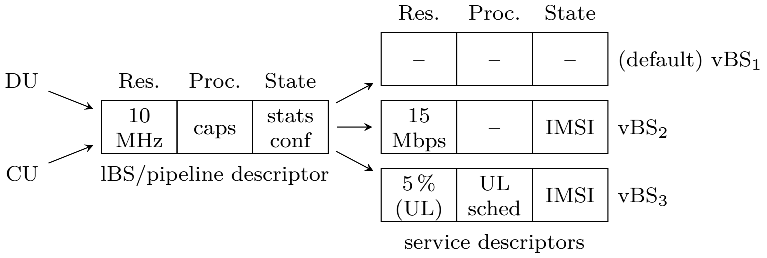
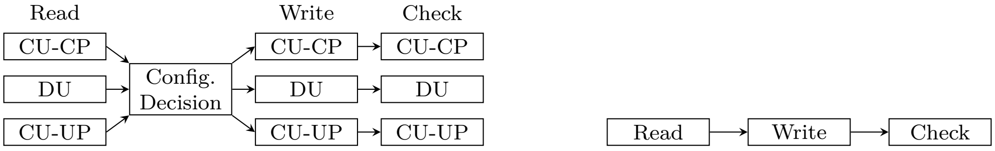


Figure 4.5 – A pipeline descriptor is merged from DU and CU for one lBS, then split into three service descriptors for vBSs (one default, two with reserved resources, specified user-list, and customizations).



(a) Without FlexVRAN.

(b) With FlexVRAN.

Figure 4.6 – Reduced control complexity for slice creation and modification.

the radio resources (10 MHz), capabilities of the underlying RAN network functions, and its state in the form of configuration and statistics. The lBS also reduces the complexity of management, shown in Figure 4.6: in case of the creation of a slice, not all involved network functions need to be programmed independently but only the single interface of FlexVRAN is sufficient. Coming back to Figure 4.5, three slices are embedded: a default slice, working as a “placeholder” slice, one slice for video services and a slice targeting the mMTC use case. The video slice requests a throughput of 15 Mbps (resources), and a list of associated users in the form of their International Mobile Subscriber Identitys (IMSIs) (state configuration), without any further customization (processing). The mMTC slice requests a fixed amount of resources in the uplink and customizes the uplink scheduler to more efficiently handle the data chunks that might be known for such service. The resource isolation and efficient sharing are enforced through suitable algorithms in the RAN as explained in Chapter 6, and respective slice-specific controllers might operate through the RAN Engine as explained in the following section.

4.4 Design using the RAN Engine Framework

While FlexVRAN elaborates on the two-level abstraction to simplify the handling of disaggregated, complex RAN deployments through a unified representation, it does not explain how services can customize or extend processing. In the following, we present the RAN Engine, which enables services to configure, customize and extend the desired processing behavior on top of a shared RAN infrastructure and on an as-a-service basis.

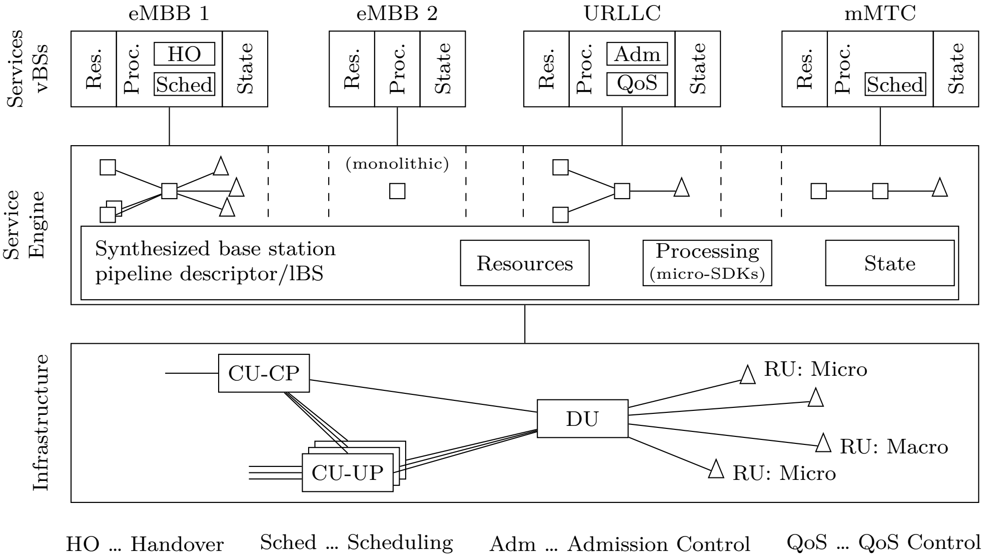


Figure 4.7 – Overview over the RAN Engine architecture, consisting of an infrastructure, the service engine, and services that are multiplexed onto the infrastructure. The service engine exposes deployment and topology information to the services as depicted in the little RAN representations within the service engine.

4.4.1 Overview

The RAN Engine framework is shown in Figure 4.7. It consists in a service engine that maps and multiplexes services onto a RAN infrastructure (with possibly multiple RAN network functions), and enables service owners to monitor, configure, customize, and extend slices, subject to access control.

The service engine synthesizes a pipeline descriptor from the RAN corresponding to an lBS, which references the amount of resources, customizable processing functionality, and state built up during runtime. This corresponds to the first-level abstraction of FlexVRAN. In the second-level abstraction, the engine maps services into the RAN following their service descriptor, corresponding to a vBS. In detail, the service engine (i) reserves the appropriate amount of (radio) resources, (ii) keeps and exposes the state of each service, and (iii) integrates the processing customization in or apply control commands to the pipeline. For the latter, the service engine allows pipeline customization through micro-Service Development Kits (micro-SDKs), which encapsulate individual RAN control endpoints, such as handover control. Furthermore, the service engine can expose the topology of the RAN infrastructure if requested by service, e.g., for deployment of additional CU-UPs.

A service is characterized by the service descriptor consisting of resources, processing, and state which delineates the SLA and any processing customizations or extensions that are requested for this service. Services can customize the behavior of the pipeline by providing isolated, stateless micro-services in the processing part of the descriptor that implement the interface towards micro-SDKs, e.g., for implementing a specific handover

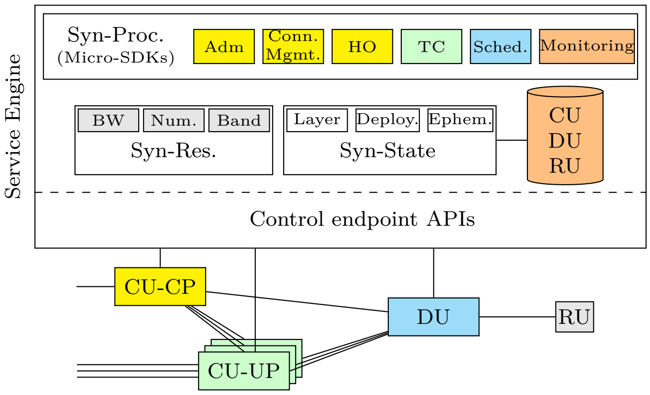


Figure 4.8 – The service engine synthesizes the RAN infrastructure. Processing provides micro-SDKs for control endpoint abstraction.

policy for mobility load balancing.

As an example, consider the two eMBB services in Figure 4.7. Service “eMBB 1” uses all macro- and micro-cell sites, and requests deployment of additional CU-UPs to efficiently handle the traffic; due to this reason, topology information is also exposed to the service itself. It also customizes handover logic (for mobility load balancing) and scheduling (efficient, customized scheduling algorithm). Service “eMBB 2” merely reserves some resources; therefore, no topology information is exposed. No processing is customized either.

The service engine allows (i) to ensure functional isolation by providing control-endpoint-specific protocols, mapping multiple services through micro-SDKs into the pipeline, (ii) provides customization possibilities through the services models, allowing services to control their slice, and (iii) can extend the RAN by deploying additional micro-SDKs at the infrastructure. The latter can be achieved through a network store, where additional micro-SDKs are deployed in a service engine, and thus the control possibilities for services are extended, as described later in Section 5.6.

4.4.2 Synthesized Infrastructure Representation

To capture the control plane of the RAN infrastructure, the engine interfaces with the infrastructure through a control endpoint API as shown in Figure 4.8. The synthesized resources capture RAN resources of the infrastructure, such as bandwidth and numerologies, allowing to infer a sustainable data rate during runtime. Note that this corresponds to a description of the resources of the infrastructure and not an abstraction which is ensured by processing customizations.

The synthesized state captures RAN configuration as well as user context information and forwards it to a shared storage that gathers the information, consolidating the state information. The state is then exposed to the services as generic base station information or in relation to a micro-SDK, i.e., a particular RAN control endpoint, to simplify control handling within the services.

The synthesized processing of the pipeline descriptor abstracts the access to specific

RAN control endpoints in the form of micro-SDKs. The micro-SDKs provide atomic, independent, extensible CP abstractions, and implement an abstract interface towards the services for RAN customization and extension, allowing a lightweight virtualization of the base station's CP *when needed.* They open up the RAN for per-service customizations and extensions at the corresponding RAN control endpoint through service models, i.e., control endpoint-specific protocols that expose granted RAN customization and extension facilities towards services. They provide operations to map, deploy and release services for a RAN control endpoint, apply service actions, and resolve (or preclude) conflicts. A runtime manages the lifecycle of the micro-SDK, which can be deployed and released dynamically within the service engine like applications. Additionally, micro-SDKs might compose existing micro-SDKs to reflect complex operations, and additional micro-SDKs might be fetched from a network store to extend processing customization capabilities.

The set of micro-SDKs of a RAN pipeline groups the actual control plane APIs for services in a deployment-independent and vendor-independent fashion, and can be classified into the radio resource, flow handling, and radio control dimensions. The micro-SDKs reflect the pipeline's modification possibilities in the form of a *capability* that might be customized by a service. For radio resource allocation, for instance, a micro-SDK could give access to a part of the spectrum for custom slice creation. The implementation of the micro-SDKs further depends on the actual endpoint.

The synthesis of the RAN infrastructure allows to logically consolidate the control plane of the RAN infrastructure. The descriptor's processing groups control endpoints for customization and configuration, and the resources in the descriptor are used to reserve RAN resources within the RAN. The state of the RAN is logically centralized, allowing to execute control plane behavior logic independently of the infrastructure and rendering parts of the infrastructure "stateless", i.e., control decisions may not need to be taken within the infrastructure. By definition, the RAN cannot be made completely stateless, since (instantaneous) state is necessary for consistency in control plane protocol operations (such as HARQ) and interaction among the layers (e.g., amount of buffered data). However, information such as user context and identifiers, runtime configuration such as bearer information, or the scheduling policies, is stored to perform failover or replication in the CU-UP and DU. Also, this means that the pipeline description is fundamentally independent of the deployment scenario (C-RAN, monolithic, or disaggregated), and a redeployment of control logic on top of different infrastructure pipelines within a multi-vendor context becomes feasible, since the micro-SDKs expose a consistent interface towards the services. The consolidation also allows a tighter control coordination among different RAN network functions, while retaining the benefit of multiplexing gain through the centralization of part of the base station processing. As an example, the consolidation enables a joint SDAP (CU-UP) and RLC (DU) packet scheduling opportunity to reduce bufferbloat (see Appendix A.1), which otherwise can not be supported in a disaggregated RAN, since such information is not available via the F1 interface.

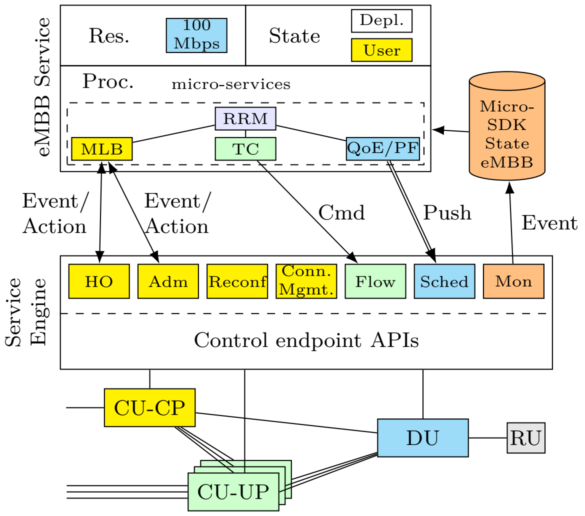


Figure 4.9 – Services multiplex their micro-service customizations through the service engine onto the underlying RAN infrastructure by matching micro-SDKs. Based on the micro-SDK, a message-based exchange might be employed (e.g., for radio control) or functionality will be pushed into the engine for timing constraint reasons (radio resource scheduling). Rate information is used to reserve resources in the RAN, and state is partitioned per service.

4.4.3 Service Description

With the help of the synthesized pipeline descriptor, services can now be multiplexed into the RAN infrastructure based on their SLA, as shown in Figure 4.9.

The resources of the service specify the desired performance indicators that need to be supported in the RAN infrastructure (e.g., a resource capacity). Upon admission control, it is mapped by the service engine in order to reserve resources in RAN layers and select appropriate network functions. The service state describes the non-ephemeral configuration of the service, such as usage of specific features, e.g., enabling split bearers for increased reliability, or specific RAN network function configuration such as of a separate CU-UP. Further, the state includes ephemeral data such as statistics for service optimization. State is partitioned by a monitoring micro-SDK on a per-service basis in order to provide isolation.

The service processing allows customizing and extending processing functionality by combining micro-SDKs. The service descriptor marks customization intentions for this service, and either supplies the execution logic in binary form, or an endpoint for message-based interaction. Here, customization refers not only to a reconfiguration or replacement of specific RAN control endpoints for a service, but also possible extensions of control plane behavior logic, e.g., for using machine learning to predict user traffic and adjust the scheduling algorithms. This allows the service owner to quickly deploy and modify service-specific functionality, and extend basic RAN functionality by composing multiple micro-services or through specialized micro-SDKs in the engine. For instance, a specialized micro-SDK reuses monitoring and MAC scheduling micro-SDKs: through traffic analysis of a group of users and reconfiguration of resource allocation, scheduling latencies are reduced and QoS is improved (see the "burst analysis" example in Section 5.6).

Based on the specific RAN control endpoint and its timing requirements, the interaction between a service and its associated micro-SDKs inside the engine shall support both hard and soft real-time operations. The micro-SDKs link the infrastructure to the services, and thus control triggers might come from the RAN (e.g., event for handover) or the services (command for policy update). This is supported by general event-action messages, direct service-originated commands, or the embedding of custom logic within or close to the engine ("push"), and might be enabled via service models. As shown in Figure 4.9, a service can supply micro-services for Mobility Load Balancing (MLB), Traffic Control (TC), and scheduling (Sched) to extend RAN processing. For the MLB, the service execution logic might be triggered through an event which is handled remotely. Similarly, the TC micro-service continuously monitors the RLC queue state to update the SDAP processing. For scheduling, a co-location of execution logic is required to keep the tight deadlines imposed by 5G, e.g., through shared memory on the same execution environment.

The admission control process enforced through the service engine is a three-step process.

1. The resource needs of the service need to be fulfilled. This is dependent on a separate admission control in at least a subset of micro-SDKs; for instance, a scheduling micro-SDK needs to verify that radio resources are available, potentially translating latency-based or other requirements into rate requirements.

2. The specific configuration of the services need to be checked for consistency with the current state in the pipeline descriptor, and configuration conflicts need to be resolved. Consider an eMBB service with high rate requirements, requiring the set-up of a new CU-UP.

3. The service engine's micro-SDKs need to accommodate the micro-services of the new services, e.g., start MAC scheduling micro-services, such that the service owners can chain their micro-services.

If admission control passes, the service engine will finally map the service onto the existing RAN infrastructure.

If a service has been admitted, it implements a service-specific controller through a composition of micro-services implementing the interface towards micro-SDKs as shown in Figure 4.9. Each controller is only aware of state pertaining to the corresponding service, and controls, reconfigures and/or reprograms its virtual subnetwork.

4.4.4 Micro-SDKs

The service-specific execution logic behaves like containerized, stateless micro-services following cloud-native principles with a limited, service-specific view on parts of the control plane, which is enabled through micro-SDKs. The micro-SDKs encapsulate an API to a control endpoint in the RAN, effectively allowing to *program* a control endpoint the pipeline. In this thesis, we will focus on an abstraction of MAC scheduling (radio

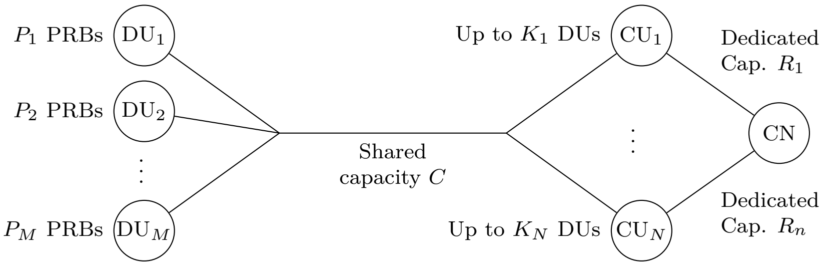


Figure 4.10 – Considered network topology for the optimization problem.

resource level, Chapter 6) while ensuring properties such as QoS enforcement, resource isolation and sharing, and customization and extension. Then, this abstraction is used to create a micro-SDK to virtualize control of the MAC for multiple controllers (Chapter 7).

4.5 Logical Base Station Composition

By splitting an lBS, multiple vBSs are embedded, according to an algorithm depending on the nature of the resources in the RAN control endpoint. For instance, MAC layer resources for RAN slicing might be split according to a slice algorithm; see Chapter 6 for more details.

However, the number of lBSs that can be composed of RAN functions (CU, DU) not only depends on RAN but also on TN resources. For instance, the TN needs to support the data rates that the RAN radio resources offer. In the following, we consider an optimization problem regarding the composition of lBSs from DUs and CUs considering the available spectrum bandwidth as well as midhaul and backhaul capacity constraints.

4.5.1 Problem Formulation

Consider a network topology comprising disaggregated RAN functions of M DUs and N CUs as shown in Figure 4.10. Between DUs and CUs is a single bottleneck link in the midhaul network with capacity C (in bps).

The i-th DU ($1 \leq i \leq M$) is connected to a single CU. It provides service to UEs using a variable amount of spectrum (through dedicated RUs). The available spectrum options are modeled through a set of bandwidths in terms of RBs as $\mathcal{P}_i = \{P_{i,1}, P_{i,2}, \ldots, P_{i,N}\}$. The exact number of RBs depends on the RAT, e.g., for LTE, $\mathcal{P}_i = \{6, 15, 25, 50, 75, 100\}$. For simplicity, we assume that the individual cells do not cause interference to each other. Due to the limitations from midhaul and/or backhaul capacity, not all RBs can be used to compose an lBS, and the notation of $p_i \in \mathcal{P}_i, \forall i$ is used to model the amount of orchestrated RBs in the first-level abstraction. Moreover, based on the amount of orchestrated RBs, we translate a peak rate requirement from radio into the transport network domain in terms of maximum bps required for both midhaul $m_i = m_i(p_i)$ and backhaul network $b_i = b_i(p_i)$.

The j-th CU ($1 \leq j \leq N$) can support up to K_j DUs, and we model its dedicated backhaul link to the core network with R_j bps.

Over this network topology, the objective is to compose lBSs while maximizing resource utilization for all DUs. The reason behind this objective is to utilize as many radio resources as possible due to huge expenses on national/regional spectrum auctions. In addition, to facilitate the problem formulation and express the mapping of N CUs and M DUs to compose lBSs, we define a composite binary control variable $x_{i,j}$, $1 \leq i \leq M$, $1 \leq j \leq N$, which is 1 if the i-th DU is associated to the j-th CU, and 0 otherwise. Finally, we pose the problem formulation as follows:

$$\begin{aligned}
\underset{p_i}{\text{maximize}} \quad & \sum_{i=1}^{M} p_i \left(\sum_{j=1}^{N} x_{i,j} \right) && \text{(4.1a)} \\
\text{subject to} \quad & p_i \in \mathcal{P}_i, \quad 1 \leq i \leq M, && \text{(4.1b)} \\
& \sum_{j=1}^{N} x_{i,j} \leq 1, \quad 1 \leq i \leq M, && \text{(4.1c)} \\
& \sum_{i=1}^{M} x_{i,j} \leq K_j, \quad 1 \leq j \leq N, && \text{(4.1d)} \\
& \sum_{i=1}^{M} m_i(p_i) \left(\sum_{j=1}^{N} x_{i,j} \right) \leq C, && \text{(4.1e)} \\
& \sum_{i=1}^{M} b_i(p_i) x_{i,j} \leq R_i, \quad 1 \leq j \leq N && \text{(4.1f)}
\end{aligned}$$

In objective function (4.1a), as mentioned before, we aim to maximize the number of orchestrated RBs for all M DUs subject to the following constraints. Constraint (4.1b) requires that the number of orchestrated RBs of each DU is a bandwidth supported by the DU. Also, a DU shall be connected to no more than 1 CU as expressed through (4.1c). Similarly, the j-th CU can accommodate at most K_j DUs, constrained in (4.1d). Note that these two constraints are based on software/hardware implementation limitations of CU and DU, and might be lifted or relaxed when using different equipment from other vendors. Constraints (4.1e) and (4.1f) model the midhaul and backhaul limitations. More specifically, (4.1e) expresses that the maximum generated midhaul traffic of all DUs shall not exceed C bps, whereas (4.1f) specifies that the generated backhaul traffic of DUs connected to the j-th CU shall not exceed the backhaul link capacity R_i bps.

We categorize the mapping of these constraints to the descriptor of resources (either the RAN or TN), processing, and state in Table 4.1.

4.5.2 Problem Analysis

The formulated optimization problem (4.1) maximizes the RB allocation, but also implicitly incorporates the sub-problem of the association of DUs to CUs. For instance, the problem might be restated to maximize the number of DUs at a CU, e.g., in order to

Table 4.1 – Constraints to descriptor mapping.

Constraint	Number	Mapping	Notes
(4.1b)	M	Resources/RAN	Each DU is orchestrated with limited RBs for wireless transmission.
(4.1c)	M	Processing	Each DU needs to be associated to CU for forming an lBS to complete radio/protocol stack processing.
(4.1d)	N	State	Each CU has a limited number of DU for which it can be configured.
(4.1e)	1	Resources/TN	Limited shared midhaul capacity.
(4.1f)	N	Resources/TN	Limited dedicated backhaul capacity.

Table 4.2 – RB to midhaul/backhaul capacity mapping from 4G measurements.

RBs (p_i)	Midhaul m_i (Mbps)	Backhaul b_i (Mbps)	UDP traffic (Mbps)
25	19.52	19.37	17.5
50	39.18	38.97	35.2
100	76.28	75.41	69.2

save energy and scale down the pool of CUs. In the following, we concentrate on the problem of bandwidth maximization.

To simplify the problem, we assume the same number of DUs and CUs, i.e., $M = N$, to form M lBSs, and statically associate DUs and CUs following a 1:1 mapping, i.e., $K_j = K = 1, \forall j$, and

$$x_{i,j} = \begin{cases} 1, & i = j \\ 0, & \text{otherwise} . \end{cases} \tag{4.2}$$

Furthermore, we measured the LTE midhaul (F1 interface) and backhaul (S1 interface) throughput for a number of RB allocations as shown in Table 4.2. From these measurements, we approximate both midhaul and backhaul throughput with

$$b_i \approx m_i \approx 0.75 p_i \; . \tag{4.3}$$

This leads to the simplified optimization problem

$$\begin{aligned} \underset{p_i}{\text{maximize}} \quad & \sum_{i=1}^{M} p_i && \text{(4.4a)} \\ \text{subject to} \quad & p_i \in \mathcal{P}_i, \quad 1 \leq i \leq M, && \text{(4.4b)} \\ & 0.75 \sum_{i=1}^{M} p_i \leq C, && \text{(4.4c)} \\ & 0.75 p_i \leq R_i, \quad 1 \leq i \leq M && \text{(4.4d)} \end{aligned}$$

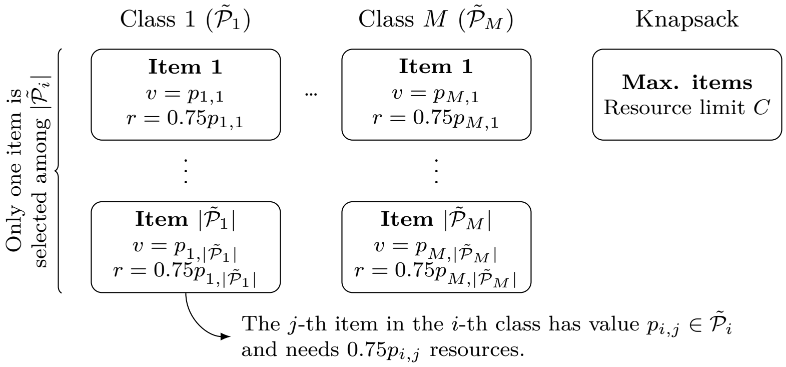

Class 1 ($\tilde{\mathcal{P}}_1$)
Class M ($\tilde{\mathcal{P}}_M$)
Knapsack
Only one item is selected among $|\tilde{\mathcal{P}}_i|$
Item 1
$v = p_{1,1}$
$r = 0.75p_{1,1}$
...
Item 1
$v = p_{M,1}$
$r = 0.75p_{M,1}$
Max. items
Resource limit C
Item $|\tilde{\mathcal{P}}_1|$
$v = p_{1,|\tilde{\mathcal{P}}_1|}$
$r = 0.75p_{1,|\tilde{\mathcal{P}}_1|}$
Item $|\tilde{\mathcal{P}}_M|$
$v = p_{M,|\tilde{\mathcal{P}}_M|}$
$r = 0.75p_{M,|\tilde{\mathcal{P}}_M|}$
The j-th item in the i-th class has value $p_{i,j} \in \tilde{\mathcal{P}}_i$ and needs $0.75p_{i,j}$ resources.

Figure 4.11 – Graphical representation of the multi-dimensional multiple-choice knapsack problem in (4.4).

Theorem 1. *The optimization problem* (4.4) *is NP-hard.*

Proof. The problem is a multi-dimensional multiple-choice knapsack problem (MMKP) [116]. We rewrite constraints (4.4b) and (4.4d) into a single constraint $p_i \in \tilde{\mathcal{P}}_i$, where $\tilde{\mathcal{P}}_i = \{x | x \in \mathcal{P}_i \cap x \leq \frac{4}{3}R_i\}$. Then, the problem follows the standard form of an MMKP with M classes of items, i.e., the sets of available RBs per DU $\tilde{\mathcal{P}}_i$. Within each class, only one item can be selected from all $|\tilde{\mathcal{P}}_i|$ items. We map the problem visually in Figure 4.11, in which the j-th item in set $\tilde{\mathcal{P}}_i$ has value $p_{i,j}$ and associated resource $0.75p_{i,j}$. The resources of all classes sum up to midhaul constraint C. □

It follows that since problem (4.4) is NP-hard, problem (4.1) is NP-hard, too.

4.5.3 Analytical Solution

The RB set for NR base stations can be assumed to be any integer value between 11 and a numerology-dependent number of RBs P_i (N_{RB} in TS 38.101-1) due to the flexibility provided by BWPs[4]. Hence, $\mathcal{P}_i = \{11, \ldots, P_i\}$, $P_i \geq 11$.

To make the problem analytically tractable, and motivated by the fact that any integer RB can be used, we relax p_i in Constraint (4.4b) to a continuous variable. For a 1:1 association for DU/CUs and the same midhaul/backhaul approximation as before, and combining Constraints (4.4b) and (4.4d) into a new constraint $p_i \leq \min\left(P_i, \frac{4}{3}R_i\right)$, we obtain the following linear program:

[4] The RRC `locationAndBandwidth` value allows setting arbitrary RB sizes, see TS 38.331.

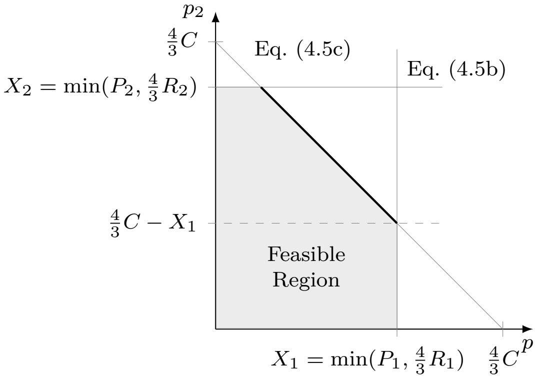


Figure 4.12 – Graphical representation of problem (4.5) for the special case $M = N = 2$ DUs/CUs.

$$\begin{aligned} \underset{p_i}{\text{maximize}} \quad & \sum_{i=1}^{M} p_i && \text{(4.5a)} \\ \text{subject to} \quad & p_i \leq \min\left(P_i, \frac{4}{3}R_i\right), \quad 1 \leq i \leq M, && \text{(4.5b)} \\ & \sum_{i=1}^{M} p_i \leq \frac{4}{3}C && \text{(4.5c)} \end{aligned}$$

We introduce the shorthand $X_i = \min\left(P_i, \frac{4}{3}R_i\right)$, which captures the maximum bandwidth P_i and the backhaul capacity R_i (through the 1:1 mapping with the CU) of DU i to express the maximum data rate the DU can support. Figure 4.12 plots the feasible region of the optimization problem for the special case $M = N = 2$ DUs/CUs. From the figure, we read

$$f^*(p_1, p_2) = \begin{cases} X_1 + X_2, & \frac{4}{3}C > X_1 + X_2 \\ \frac{4}{3}C, & \frac{4}{3}C \leq X_1 + X_2 \; . \end{cases} \tag{4.6}$$

In the first case, the midhaul capacity C is larger than the joint maximum data rates of both DUs, and the optimal allocation is to use all resources for both DUs, $p_i^* = \min\left(P_i, \frac{4}{3}R_i\right)$. This means that the midhaul is overprovisioned compared to the achievable data rates over the combined bandwidth and/or backhaul. For instance, this would be the case for an identical backhaul $R_i = 0.2\,\text{Gbps}$, a midhaul of $C = 1\,\text{Gbps}$, and a bandwidth of $P_i = 100\,\text{RB}$, resulting in $X_i = 0.1\,(\text{Gbps})$.

In the second case, the network is underprovisioned with midhaul capacity C. Four cases can be distinguished:

1. $X_i \geq \frac{4}{3}C$, $i = 1, 2$. The minimum of bandwidth and backhaul of both DUs is larger than what the midhaul can support. The optimal solutions for maxing out the

midhaul (highlighted in Figure 4.12) are $p_1^* = c$ and $p_2^* = \frac{4}{3}C - c$, $0 \leq c \leq \frac{4}{3}C$, i.e., the midhaul can be arbitrarily shared between both DUs.

2. $X_1 < \frac{4}{3}C$, and $X_2 \geq \frac{4}{3}C$. DU$_2$ can max out the midhaul, but DU$_1$ can not. The optimal solutions are $p_1^* = c$ and $p_2^* = \frac{4}{3}C - c$, $0 \leq c \leq X_1$. In the special case $X_1 < \frac{2}{3}C$, the midhaul can not be shared equally by both DUs.

3. Like 2., but with X_1 and X_2 switched.

4. $X_1 < \frac{4}{3}C$, and $X_2 < \frac{4}{3}C$. Both DUs alone can not max out the midhaul. The optimal solutions are $p_1^* = c$ and $p_2^* = \frac{4}{3}C - c$, with $\frac{4}{3}C - X_2 \leq c \leq X_1$.

Combining these cases, the shared midhaul is maxed out for the optimal solutions $p_1^* = c$ and $p_2^* = \frac{4}{3}C - c$, with $\max(0, \frac{4}{3}C - X_2) \leq c \leq \min(\frac{4}{3}C, X_1)$. Recall that $X_i = \min\left(P_i, \frac{4}{3}R_i\right)$ captures both bandwidth and backhaul limitations, and that in the special case of 1:1 mapping, the backhaul should easily be able to carry the RAN traffic (given that the maximum might be around 1 Gbps for NR, Ethernet technology can easily support the traffic in the backhaul). The midhaul might be shared equally in a max-min fashion[5] to equally use bandwidth in the whole network, or unequally, e.g., to adapt to user load. Thus, by assuming a certain service/user load within the lBSs (by means of embedded vBSs), the optimal p_i^* might be adapted to fulfill the service requirements, and give excess resources to other lBSs; this might be re-evaluated whenever new vBSs are deployed/released.

Similarly, we can apply the simplex method with M slack variables for M DUs and M CUs, and obtain

$$f^*(p_1, \dots, p_M) = \begin{cases} \dfrac{4}{3}C, & \dfrac{4}{3}C \leq \displaystyle\sum_{i=1}^{M} X_i \\ \displaystyle\sum_{i=1}^{M} X_i, & \dfrac{4}{3}C > \displaystyle\sum_{i=1}^{M} X_i \end{cases} \tag{4.7}$$

$$= \min\left(\frac{4}{3}C, \sum_{i=1}^{M} \min\left(P_i, \frac{4}{3}R_i\right)\right) . \tag{4.8}$$

A similar analysis of the optimal values for p_i^*, $i = 1, \dots, M$, might be done as for the special case before.

The results link RAN and TN resources through a closed-form analytical solution, allowing the controller to dimension lBSs depending on both service requirements (vBSs) and (expected) network load. As an example, it is possible to orchestrate lBSs with additional radio resources near hotspots without providing more resources than what the TN (here, the shared midhaul) supports, or ensure that the TN supports the combined load of multiple vBSs, expressed through radio resources, at a given location.

[5] Max-min fairness is achieved when an additional allocation (in terms of c) to one DU results in the decrease to the other, which is achieved for $c = \frac{2}{3}C$.

Table 4.3 – Abstraction terminology and its mapping to E2.

Term	**Meaning**	**Mapped E2 term**
lBS/pipeline descriptor	logical BS representation and description	E2SMs
Merging	Reaggregation of DU, CU	Merge of E2SMs
vBS/service descriptor	virtual BS/service representation/SLA	–
Splitting	Multiplexing of services onto BS	– *(Virtualization)*
Embedding	Revealing service-specific RAN view	–
Synthesization	Abstracted RAN functionality per RAN control endpoint	E2SMs
Micro-SDK	Specific RAN control endpoint	RAN function
Micro-service	Set of micro-services control RAN	xApp
Service Engine	Allows service-specific control	– *(Virtualization)*

Note that the optimal results presented in the context of the relaxation of p_i to a continuous variable do not hold in the general case for $p_i \in \mathcal{P}_i$, which can only be calculated by solving the integer linear program in (4.4) in non-polynomial time complexity using a combinatorial optimization approach. Another common approach is to solve a relaxed version of the problem as mentioned above and then do local search around the optimal values to save time complexity on average.

4.6 Discussion and Future Work

This chapter introduced descriptors that are of conceptual nature. Through an abstraction of the infrastructure, multiple controllers can be isolated from each other while maintaining control over their respective virtual network. Abstractions are thus key to implement a service-oriented RAN.

An implementation of these descriptors, as presented later in this thesis, will use E2 Service Models (E2SMs) (see also Section 2.2.2) in the context of FlexRIC in Chapter 5. In order to facilitate the understanding of the presented concepts and their mapping to E2, we summarize the terminology and its meaning together with a mapping to E2 terms [34] in Table 4.3. In this context, the first-level abstraction will be implemented by a controller that re-aggregates the E2SMs. Further, the second-level abstraction is implemented by a *virtualization* of the corresponding E2SMs. Chapter 6 devises an E2SM for the MAC radio resources, which is then virtualized and split towards multiple service owners, presented in Chapter 7.

Regarding future work beyond what is presented in this thesis, the following directions are possible.

The first direction is to investigate concrete abstractions for the first- and second-level abstractions apart from the radio resource abstraction considered in this thesis. Thus, E2SMs for flow management and UE radio control should be developed, and means to virtualize the control towards different tenants need to be investigated. Especially for the flow level, this should happen across the CU-UP and DU RAN network functions. The corresponding virtualization should fulfill the properties defined in the beginning of this chapter.

The second direction concerns the considered optimization problem. The problem might be restated to maximize the number of DUs at individual CUs while minimizing the number of CUs in order to save energy. Apart from maximizing the bandwidth usage, service requirements and a corresponding minimization of lBS bandwidth usage might be considered, as well. Apart from throughput limitations, the optimization problem might be extended with latency constraints to model limitations of service embeddings (vBS) in the network. This could include a modeling of the numerologies to map minimum achievable latencies.

4.7 Conclusion

In this chapter, we introduced descriptors to formalize the functionality within and capabilities of a base station towards a service-oriented representation, as well as the base station's resources and state. This notion has been extended to services to describe their requirement and custom processing. Alongside these descriptors, we introduced properties that such abstractions should fulfill to efficiently multiplex services into the RAN.

Using these descriptors, we further discussed the concept using FlexVRAN and proposed a design using the RAN Engine. Through FlexVRAN, we conceptualized the two-level abstraction from disaggregated RAN into a logical representation, and how to embed multiple services into the form of virtual base stations in such logical representation. Using the RAN Engine, we further formalized the processing capabilities of the RAN in the form of micro-SDKs, which abstract control endpoints in the RAN. They are used by services to customize and extend RAN processing towards the service's requirements using containerized micro-services.

Finally, we formulated an optimization problem of embedding lBSs over a network of M DUs and N CUs as a concrete example of forming base stations along the first-level abstraction, linking the domains of RAN and TN. We analytically solved a relaxed version of this optimization problem in the form of a linear program.

The work has been discussed in the context of the E2 interface used later in this thesis, and future work directions were shown.

Chapter 5

FlexRIC SDK

5.1 Introduction

Unlike previous mobile networks, 5G-NR provides unprecedented flexibility in the RAN to support diverse use cases in a multi-tenant environment. In this context, and in general as outlined in Section 2.2, the need for programmability and control through SD-RAN is of high importance. However, while the underlying RAN is designed to be ultra-flexible and lean, existing SD-RAN controllers are either not flexible to address all use cases, or they use a “one-size-fits-all” approach.

FlexRAN [33] was the first attempt to realize a real-time SD-RAN platform for research purposes. It applies the principles of SDN by defining a custom southbound control protocol, but it is coupled with the underlying RAT and tailored towards specific control operations. The operator-lead O-RAN consortium, on the other hand, specifies an extensible control protocol (E2) and provides a reference implementation, the O-RAN RIC [50]. The RIC entails a micro-service architecture with associated overhead (e.g., fat resource footprint, increased latencies) even on use cases that do not need it. In other words, it enforces a “one-size-fits-all” design, violating central 5G principles such as flexibility and ultra-lean design, and as a cluster deployment, is not easy to use.

These two controllers are representative for the current SD-RAN controller landscape. We observe that there is a lack of SD-RAN design that combines their advantages, such as a lean design for real-time operation (like FlexRAN) and forward-compatibility (like O-RAN RIC). In correspondence to the overall 5G vision, such a design should be able to flexibly adapt to use cases, without imposing unnecessary overhead. This is even more important as future use cases can not be foreseen, and thus extensibility is of high significance to remain forward-compatible. Further, multiple services with independent programmability, as described in the preceding chapter, should be supported, all while being ultra-lean to enable real-time operation, multi-RAT-capable, and compatible with industry efforts employing the E2 protocol.

In this chapter, we propose FlexRIC, which is a Software Development Kit (SDK) in line with 5G principles. The SDK is flexible to allow conceiving specialized SD-RAN controllers adapted to particular use cases, is ultra-lean, and forward-compatible by means of the E2 protocol. We provide an overview over the FlexRIC SDK in Section 5.2.

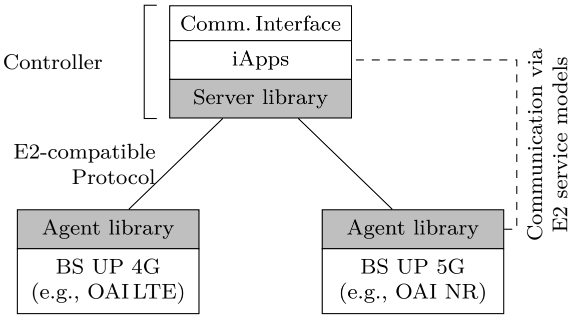


Figure 5.1 – The FlexRIC SDK consists of an agent and a server library. The agent library provides integration of a base station to a controller, which is specialized to a particular use case using iApps.

We detail the components of the SDK in Section 5.3: in particular, we address how a FlexRIC-based agent is multi-service-capable, how a FlexRIC-based controller can be specialized, and how the message protocol can be adapted to use cases. We describe many controller specializations in Section 5.4, including an implementation of FlexRAN with corresponding, multi-RAT-capable monitoring capabilities. An in-depth performance evaluation using the FlexRAN specialization and comparisons to the original FlexRAN and the O-RAN RIC controllers in Section 5.5 proves FlexRIC's versatility and real-time capabilities. Extensibility is provided through an integration of a "Network Store" on which we elaborate in Section 5.6. Further extensions and future work are discussed in Section 5.7.

5.2 Overview

FlexRIC is an SDK, consisting of a server library and an agent library with two optional extensions: controller-internal Applications (iApps) and communication interfaces. The objective of the SDK is to facilitate the realization of specialized SD-RAN controllers to target specific use cases, while being simple to use. FlexRIC does not support extra features endogenously following the *zero-overhead principle*. In its simplest form, the SDK can be used to implement an SD-RAN controller using an E2-compatible protocol, as it is shown in Figure 5.1.

The agent library is the basis to extend a base station with the agent functionalities. It provides an API to implement custom RAN functions, i.e., RAN functionality that can be monitored and/or controlled by applications, and comes with a bundle of pre-defined RAN functions that implement a set of service models (E2SM) that can be included.

A controller is built through the server library, iApps, and optionally a communication interface. The server library manages agent connections, and routes messages between iApps and the agents. Through the iApps, it is possible to modularly build specialized controllers: based on the considered use case, iApps can implement service models themselves, or expose information to external Applications (xApps) via a northbound communication interface. In the latter case, xApps can control the RAN while being

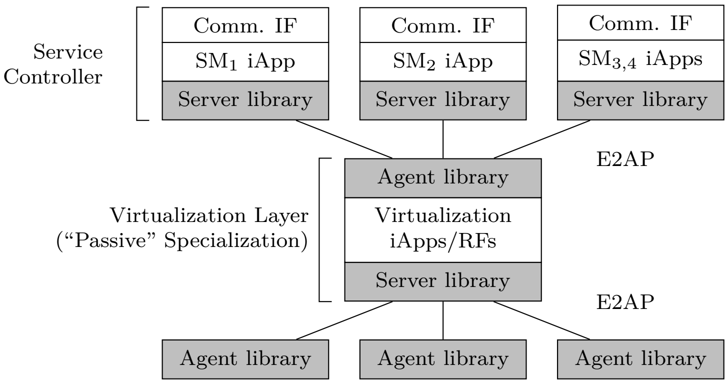


Figure 5.2 – Using the FlexRIC SDK, different specialized controllers can be built, like a "recursive" virtualization controller.

functionally isolated from the controller, which is the favored method of the reference O-RAN RIC, but bears a certain overhead. Finally, it is even possible to recursively expose an agent interface at the northbound by reusing the agent library, as shown in Figure 5.2. Such a virtualization layer delegates control to multiple (per-slice) controllers, each possibly operating on a different set of E2SMs, without controlling the network itself. This recursive property allows not only to compose various specialized controllers into a multi-service SD-RAN infrastructure but also abstract out the heterogeneity of the 4G/5G deployment topology. We will explore such a design in Chapter 7.

The FlexRIC SDK provides an abstraction of the E2 interface via an internal representation of E2 messages, and the SDK handles the actual encoding and transport of the messages. It is therefore straight-forward to integrate FlexRIC with a pre-existing E2-compliant SD-RAN infrastructure. However, the standard mandates an encapsulation of ASN.1-encoded data inside of ASN.1 and its transport over the SCTP protocol, which may be inefficient in certain cases, for instance when the message size becomes large (case for monitoring information). Therefore, the SDK also supports to change both the encoding scheme and transport protocol allowing the integration of a vendor-specific, possibly more efficient E2 protocol for low-overhead, real-time control.

5.3 FlexRIC SDK Architecture

In the following, we elaborate on the FlexRIC architecture, with a particular focus on the following design challenges:

1. A RAT-agnostic and vendor-independent SD-RAN design and its integration of any base station implementation (Section 5.3.1),

2. An SD-RAN design according to 5G principles, i.e., (a) ultra-lean for low-latency or resource-restricted use cases, avoiding unnecessary overhead, and with (b) flexibility and forward-compatibility towards novel use cases (Sections 5.3.1-5.3.3),

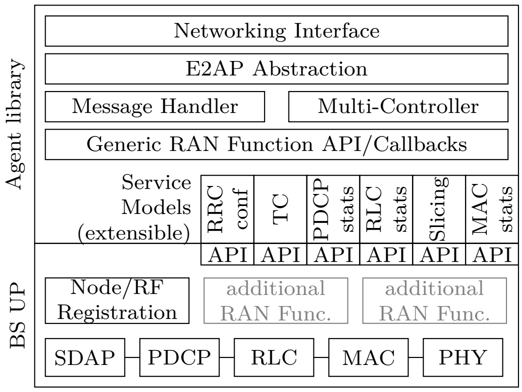


Figure 5.3 – The FlexRIC agent architecture and integration with a user plane implementation.

3. A low-coupled design that allows the specialization of SD-RAN controllers (Section 5.3.2) towards particular use case requirements, such as traffic control, slicing, and recursive slicing through network virtualization (as presented in Section 5.4).

5.3.1 FlexRIC Agent

The main design principle of the agent library relies on its easy integration into various base station implementations and deployment scenarios interfacing with an E2 controller, while at the same time providing additional functionality for handling multiple controllers in a multi-service environment, which are detailed in the following.

Agent Architecture

The architecture of an agent using the FlexRIC SDK is shown in Figure 5.3. It consists of the agent library, and a user-plane implementation, such as a specific base station or a testing agent.

The agent library provides the necessary support to connect to a controller. It consists of a networking interface, the E2AP abstraction, a message handler, and a generic RAN function API. The E2AP abstraction provides an intermediate representation for the E2 protocol, relieving RAN functions from E2 protocol specificities (encoding, decoding, see Section 5.3.3). Further, the agent provides the generic RAN function API to implement RAN functions with custom service model-specific logic. This API defines callbacks for E2AP messages, i.e., (i) subscription requests (for new information subscription), (ii) subscription delete request (removal of subscriptions), and (iii) control messages (to trigger service-model-specific actions), which need to be implemented by RAN functions. Finally, we pre-defined service models for the considered use cases, namely slicing control (see Chapter 6) and traffic control (c.f. Appendix A). The service models are tailored towards specific RAN layers (for statistics) or RAN control endpoints (for control) to easily integrate the agent library in disaggregated base stations, and expose a simplified API to the generic RAN function API, facilitating the integration with various base station implementations. It is furthermore possible to dynamically deploy additional

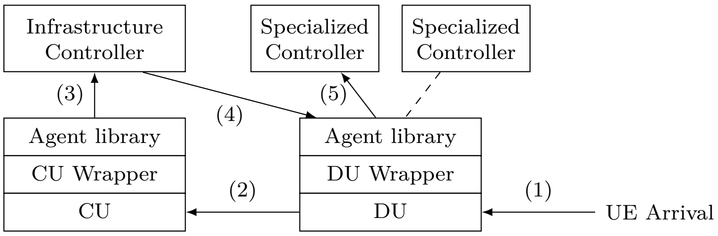


Figure 5.4 – The agent library handles multiple controllers. For disaggregated base stations, a controller needs to configure the UE-to-controller association in all agents (step 4) to ensure that a connecting UE (step 1) is exposed to the correct specialized controller (step 5).

RAN functions with custom service models (see Section 5.6).

The base station provides basic node information, such as the configured PLMNs and slices, and registers RAN functions according to the underlying node's capability: unlike what is shown in Figure 5.3, not all RAN layers are present in a node for disaggregated base stations with CU and DU, and FlexRIC natively supports such disaggregation through the selection of the appropriate RAN functions. The base station uses the interface of pre-defined RAN functions to expose data and handle control messages, or it defines additional RAN functions using the generic RAN function API.

The agent library is not tied to any existing cellular user plane implementation or RAT and is therefore inherently vendor-neutral and RAT-neutral, making it a reusable component for multi-vendor and multi-RAT scenarios.

Multi-controller support at the agent library

The agent library provides means to connect to multiple controllers. This becomes useful in a multi-service context, e.g., for "recursive" controllers that virtualize RAN control (Chapter 7) or base station hypervisors like Orion [53], which permits shared control and exposes information to multiple controllers while also providing isolation between them.

The FlexRIC agent library enables multiple service controllers, as shown in Figure 5.4, and assists their handling through (1) management of additional controllers (setup, teardown, providing controller origin to RAN functions for message handling), and (2) a UE-to-controller association.

The UE-to-controller association is used to indicate the UEs that are to be exposed to each controller. An active E2 subscription addresses all (or an indicated subset) of UEs. However, a given RAN function does not know a priori which UEs are to be exposed to a particular controller. Therefore, when handling messages, the agent is able to look up and reveal the UEs that belong to the corresponding controllers.

The agent library associates every UE to the first controller, and provides no means for an automatic association of UEs to additional controllers, e.g., through 5G RRC slice identifiers (NSSAI). Rather, this has to be triggered through a controller based on information from an E2SM, as the agent might not always be capable to determine an association. Consider the example of a split base station with CU and DU, and a separate controller that is concerned with the DU and exposes a remote scheduling E2SM, as shown in Figure 5.4. When a new UE arrives and should be connected to the separate controller, its association depends on the selected PLMN of the UE, which is decoded in the CU. The agent of the DU therefore cannot infer such an association and needs to be assisted from the infrastructure controller about the arrival of this new UE, configuring the agent to expose the corresponding UE at the specialized controller.

Note that SLAs are *not* part of multi-controller management. In fact, SLAs are tied to specific E2SMs, which are not handled by the agent library as the underlying resources vary, e.g., resource blocks in MAC, queues in RLC/PDCP/SDAP, UE connections in RRC. Thus, a unified SLA that could be handled by the agent library is not possible. Therefore, the RAN function has to perform sufficient admission control upon subscriptions of the controllers, and handle any conflicts that might arise from their control operations.

Micro-SDKs

Micro-SDKs, described in Section 4.4, encapsulate and package concurrent multi-service control on top of a single base station. They abstract and virtualize RAN control endpoints, and allow customizing and extending RAN functionality on a per-service basis.

The FlexRIC SDK enables to implement micro-SDKs. Through the use of the multi-controller functionality described above, a RAN function can multiplex the control of multiple controllers. By defining suitable service models, virtualized control for a specific RAN function can be exposed, which is then used by the service controllers to steer the RAN. The actual, controllable functionality depends on the service model, and SLAs have to be enforced within the RAN function.

5.3.2 FlexRIC Controller

FlexRIC permits to smoothly create specialized (service-specific) controllers. As shown in Figure 5.5, it consists of (1) the FlexRIC server library, providing means to communicate with agents using an E2AP abstraction, and (2) a controller specialization.

Controller Specializations

A controller specialization implements SD-RAN-related functionality, such as a generic SD-RAN controller, or a “recursive” virtualization layer. The specialization is realized through internal applications (*iApps*), a communication interface, and external applications (*xApps*).

The iApps implement specific controller behavior, either directly through E2SMs within the iApps themselves, or by providing platform services that can be leveraged by external applications (*xApps*). For this purpose, a controller specialization typically exposes a northbound communication interface using a custom protocol, such as a simple REST interface (e.g., FlexRAN [33]), the RMR library (e.g., O-RAN RIC), a message broker (e.g. Redis), or E2AP itself.

An iApp uses the server library by subscribing for new agent connections. If it finds suitable RAN functions, it may use the included information to send a subscription. After this, it uses the event-based interface of the server library to interact with the agents.

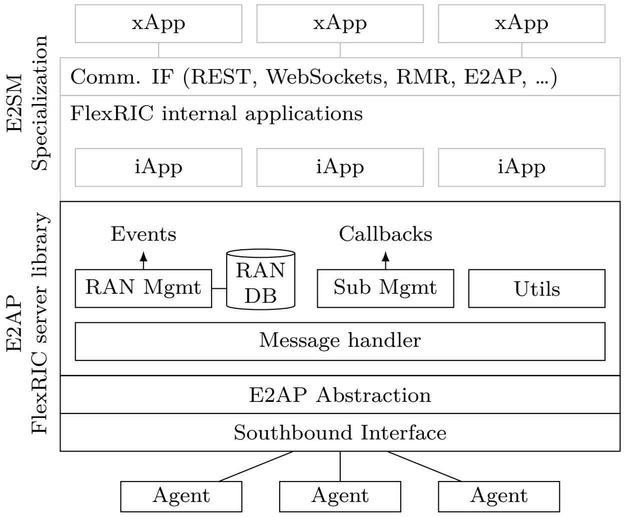


Figure 5.5 – A FlexRIC-based controller uses the server library for basic E2-related function handling, and implements functionality through iApps. They might provide services to xApps through custom, controller-specific communication interfaces.

By leveraging the FlexRIC SDK as a basis, implementing new iApps, or extending and combining existing iApps, various controller designs can be realized. Additionally, it is also possible to implement a fully O-RAN compatible controller, where iApps implement *platform functions* as a basis to have xApps perform radio resource management through service models. In this regard, we describe different controller designs in Section 5.4.

FlexRIC server library

The FlexRIC server library's objective is to multiplex agent connections and dispatch E2AP messages. As the agent library, the server library abstracts the E2AP communication. The server library is designed as an event-driven/callback-driven system with minimal overhead that invokes applications only when there are new messages, unlike systems like FlexRAN that use polling.

The RAN management functionality handles connection-related events such as an agent connection. Further, the RAN management functionality stores information in the RAN database (RAN DB), allowing to query information about the composition of the RAN network. For this purpose, the RAN management also merges agents that belong to the same base station (e.g., CU agent and DU agent) into the same RAN network function, facilitating base station control across agents, and provides events to signal when a complete RAN is formed from disaggregated entities.

The subscription management's task is to (i) keep track of existing subscriptions and (ii) route arriving subscription-related messages to the corresponding iApps. When an iApp requests a new subscription directly or on-behalf of an xApp, it provides a set of callbacks that are called to inform the iApp about the subscription outcome, and to dispatch the corresponding indication messages. When a message arrives, the subscription management simply selects the iApp for which the message is sent and forwards it through the provided callback.

It has to be pointed out that the server library itself does *not* implement any E2SM, and does not request any information from the agent by itself. Instead, iApps have to trigger service-model-related communication (generally on behalf of xApps), and the server library provides the platform to multiplex messages between agents and iApps.

5.3.3 E2 Protocol Abstraction

We identified four orthogonal abstractions in O-RAN's E2 specification.

1. The transport protocol, defined by O-RAN to be SCTP.
2. The encoding/decoding algorithms at the procedures (E2AP), defined by O-RAN to be ASN.1 PER.
3. The encoding/decoding algorithms at the E2SM, defined by O-RAN to be ASN.1 PER.
4. The semantics of E2AP, defined by O-RAN around subscriptions and control messages.

For handling the transport protocol, we created a wrapper around a Linux networking socket to abstract the communication interface allowing to easily switch between different transport protocols. O-RAN uses SCTP, which is already used at various other layers in the RAN.

For the encoding/decoding algorithms of the E2 procedures, such as Setup, Indication, or Control messages, we modeled an intermediate representation for all messages (see Section 2.2.2). In this manner, we are able to represent E2AP procedures without loss of information and independently of any particular encoding/decoding algorithms. To this end, we implemented the E2AP procedures using the O-RAN default ASN.1 PER (Packet Encoding Rules). However, we observed that ASN.1 PER optimizes for encoding size, making it computationally expensive, and it has been shown that replacing it in the core network can considerably increase the number of concurrent connections [117]. Hence, we also implemented E2AP using Google Flatbuffers (FB), which is a serialization format designed for performance-critical applications that avoids superfluous allocation or copying steps [118]. This increases the flexibility of the SDK: in scenarios where the wired bandwidth is scarce, ASN.1 presents better compression rates, while in scenarios where the CPU represents the bottleneck, Flatbuffers is preferred as it uses less resources (see Section 5.5.2 for details). Due to its intermediate representation, FlexRIC is open for adding a new encoding/decoding algorithm, without having to modify its source code, which enables future implementations of encoding/decoding algorithms.

Similarly, for the service models defined by O-RAN, ASN.1 PER is foreseen. FlexRIC allows custom encoding/decoding algorithms aiming to support future changes and with the same motivation as the abstraction for E2AP.

Finally, we did not create an abstraction around the semantic of E2AP, as this would imply to completely redesign the protocol (e.g., making it stateless instead of the

Table 5.1 – Possible combinations of FlexRIC abstractions.

Transp.	AP Enc.	E2SMs	Semantics	Example
SCTP	ASN.1	O-RAN Spec. (ASN.1)	Sub./Control	O-RAN/E2
SCTP	FB	Stats, TC, Slicing (FB)	Sub./Control	FlexRAN specialization

current stateful implementation that is achieved, for example through the setup request procedure) [35].

Table 5.1 lists the abstraction levels and possible combinations for some example controllers.

5.3.4 Implementation of the FlexRIC SDK

The implementation of the FlexRIC SDK provides minimum, yet sufficient mechanisms to implement a service-specific controller. The agent and server libraries (SDK), including the E2 abstraction, are roughly 10K lines of C11 code (we use generics to achieve compile time polymorphism), not using any external dependencies except for Flatbuffers and/or ASN.1, and dedicating more than 4K lines to the E2AP message encoding/decoding, where we implemented the most common 20 (out of 26) E2AP message handling in ASN.1, and 12 in Flatbuffers. We chose C as it is the *de facto lingua franca* in computer science, and therefore, an interface for another language can be implemented (e.g., through SWIG). Moreover, it can be smoothly integrated into nearly any programming language, contrary to other *low-level* programming languages (e.g., C++ or Rust).

A controller might need to handle indication messages from many agents. While the current implementation is single-threaded, a multi-thread extension is conceivable. The interface between the FlexRIC server library and iApps uses an event-based/callback-based system to pass E2 messages. For scenarios where message processing time is an issue, and given that the handling of indication messages in the server library is stateless, it is possible to pass messages to different threads, facilitated by the event-based system, and handle messages in multiple threads for better scalability. We remark that POSIX sockets are thread-safe, and sending messages from multiple threads is also feasible.

5.4 Controller Specializations

FlexRIC's modular design permits to easily compose specialized controllers on top of the server library through the iApps, xApps and selection of E2SMs. Moreover, its design opens up new opportunities for the development of state-of-the-art research controllers (e.g., recursive or multi-RAT controllers).

Some tradeoffs that need to be evaluated when designing a new controller can be summarized in the following:

1. Transport. FlexRIC's internal representation of the E2AP messages permits different encodings for the messages. We implemented ASN.1 PER for scenarios with scarce bandwidth and Flatbuffers for scenarios where the CPU is the bottleneck.

However, FlexRIC's intermediate representation transport abstraction is open for extension. Thus, new encoding schemes could be easily integrated in the future if needed, following the forward compatibility principle of 5G.

2. Service Models (E2SM). The controller exposes or modifies the RAN through the E2SMs. Therefore, suitable service models that forward/receive data to/from the controller need to be present at the RAN. However, adding E2SMs that are not needed by the controller just consumes resources, and unnecessarily complicates the design, which violates the ultra-lean design principle.

3. iApps/communication interface/xApps. Service model handling is either handled by iApps directly or over the communication interface through xApps. Use-case requirements, e.g., low latency or security, might require or preclude xApps. In the case of xApps, iApps submit the RAN information to the xApps and may receive commands from them. Thus, flexibly selecting a correct mechanism to communicate with xApps may be crucial in some scenarios.

4. Isolation. FlexRIC, iApps and xApps are software programs. Therefore, if security is a concern, techniques applied to software programs can be used (e.g., virtual machines, containers) with their associated tradeoffs.

5.4.1 Service-Specific SD-RAN Controllers

On the one hand, the heterogeneous scenarios for which 5G is designed, hinders to identify the features that a controller should implement, while on the other hand, 5G claims to provide forward compatibility. Using the FlexRIC SDK, we customize controllers tailored specifically to applications needs, achieving the flexibility which 5G necessitates.

We prototyped different service-specific SD-RAN controllers on top of FlexRIC to validate its design and that represent state-of-the-art 5G scenarios. For each, we implemented separate service models tailored towards specific use cases:

- We present a RAT-independent slicing E2SM in Chapter 6. This service model brings increased flexibility for the control of radio resources for both slicing and user scheduling algorithms. It is RAT-independent and forward-compatible, since new algorithms can be added through the use of shared objects, e.g., through a network store. Extensive evaluations for both 4G and 5G are presented as well.

- We prototyped a traffic control E2SM as presented in Appendix A.1. This service model increases the flexibility for the control of flow level resources to reduce the problem of bufferbloat [119], where many packets fill up the queues in the RAN, leading to excessive packet delays. Note that such service model jointly operates on the RLC (within the DU) and SDAP/PDCP (within the CU-UP) layers, and as such, over two RAN function in a split base station, which is natively supported by FlexRIC.

- Finally, we prototyped a load-balancing controller for eHealth services, shown in Appendix A.2. By observing cell load and considering RRC measurements, the controller can balance the load within the network, especially for moving UEs, and thereby improve service offering in the network.

5.4.2 Simple RAN Monitoring

For database integration purposes, FlexRAN [33] employs the “FlexRAN producer”, a plugin to export statistics at configurable intervals. During our analysis of FlexRAN, we observed that the implementation of the plugin is inefficient, as FlexRAN collects statistics in the RIB first, before sending them to the database in a second step[1]. Furthermore, there are separate configuration options for the periodicity of statistics messages received at the controller, and the periodicity with which they are sent to the database.

A controller specialization for RAN data monitoring can eliminate this overhead and simplify the configuration of the system. An iApp for statistics configures subscriptions for selected statistics E2SMs. Upon reception of the corresponding indication messages, the controller converts them to a message format suitable for the considered database, before exporting it directly. A northbound REST interface allows configuring requested statistics, message periodicity, and information granularity similar to the “FlexRAN producer”, allowing an easy replacement of existing deployments for generating data sets of RAN information.

Such a controller is highly relevant for monitoring and data-analysis use cases, e.g., to collect data sets, or the integration with machine-learning algorithms in order to optimize the RAN performance. In a collaboration, we explored such a monitoring use case in the context of the ElasticSDK [120], which is a Monitoring Software Development Kit that enables apps to collect, incrementally process and further expose information flows in a flexible Pub/Sub fashion via appropriate SDK API calls. Due to space reasons, we omit details of the design and prototype of the ElasticSDK, and refer the interested reader to the corresponding publication[2].

5.4.3 Generic SD-RAN Controller

FlexRAN was the first openly available SD-RAN controller platform that allowed a separation of control and user plane on top of a RAN with support for real-time control of radio resources. It demonstrated programmability at the RAN level, such as cross-base station interference coordination and RAN sharing. In the following, we detail how FlexRAN can be prototyped within the FlexRIC SDK. We focus on the following three main features of FlexRAN and how to implement them within FlexRIC: (1) separation of the control and user plane through the FlexRAN protocol, (2) the management of network information through the RAN Information Base (RIB), and (3) virtual control functions.

FlexRAN consists of a master controller and an agent which communicate through the FlexRAN protocol. FlexRAN’s main functionality can be summarized through its protocol functionality, which is broadly categorized into the groups of (1) configuration

[1] Disclaimer: the author of this thesis implemented the FlexRAN producer.

[2] ElasticSDK was presented by the author of this thesis online and is available at `https://youtu.be/pBrXkWmtnX4`.

Table 5.2 – FlexRAN protocol message groups and mapping to E2.

FlexRAN Message	Example	E2AP Type	E2SM (example)
Configuration (Get, Set)	Cell ID	Indication (Get), Control (Set)	RRC conf.
Statistics	MAC stats	Indication	MAC, PDCP, …
Commands	Slice creation	Control	Slicing
Event-trigger	UE attach	Indication	UE Events
Control Delegation	Scheduler policy	Policy	Remote Scheduling

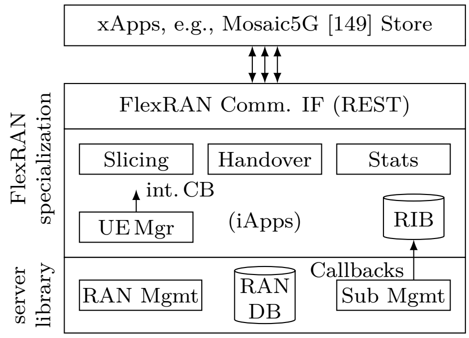


Figure 5.6 – FlexRAN controller specialization based on the FlexRIC SDK.

messages, (2) (user plane) statistics, (3) control commands, (4) event triggers, and (5) control delegation. Table 5.2 lists the FlexRAN message groups including an example, and how these messages correspond to E2AP messages. To re-implement the messages, we developed service models, as also indicated in the table. In particular, the statistics service models have been designed to offer the same information content as FlexRAN's statistics while covering both LTE and NR in a RAT-independent way where possible (e.g., scheduled resource blocks), while resorting to RAT-specific extensions where necessary (e.g., the numerology for NR). Similarly, for the other message groups, service models might be defined as well. In summary, the FlexRAN protocol can be re-implemented using suitable E2SMs.

Central to FlexRAN is the RAN Information Base (RIB), a data structure that maintains statistics and configuration of all agents, base stations, and UEs. An application called the RIB updater updates this structure whenever new configuration or statistics from the agent arrive. This behavior can be emulated by FlexRIC: a specific stats iApp requests statistics for all RAN layers, and resulting E2 indication messages are directly passed from the subscription management into a data structure comparable to the RIB, as shown in Figure 5.6. Note that the data structure emulating the RIB is separate from the RAN DB of FlexRIC, which is maintained internally by the service library. Furthermore, this specialization uses a REST northbound interface that mimics the interface of FlexRAN, e.g., to configure the slice configuration, handover commands, or statistics configuration.

Finally, one feature of the FlexRAN agent design is the support of control modules

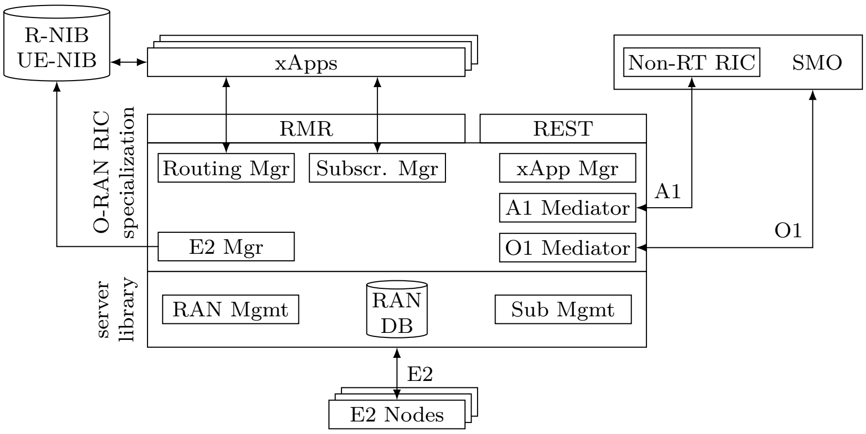


Figure 5.7 – O-RAN controller specialization based on the FlexRIC SDK.

per RAN layer. Each control module aggregates a number of virtual control functions, and each such control function might be updated ("VSF updation") and reconfigured with a specific policy. Due to the E2AP policy messages coupled with the network store capability of the SDK, this functionality can be replaced with RAN functions and suitable service models within the FlexRIC SDK. As such, it is possible to dynamically deploy applications and push scheduler implementations into the infrastructure, as will be demonstrated in Chapter 6.

Such controller might be used as a backward-compatible replacement for FlexRAN deployments, which is also less complex than an O-RAN-compliant controller. The FlexRAN implementation over the FlexRIC SDK is used for the evaluation of the FlexRIC SDK in Section 5.5, showing the feasibility of such implementation.

5.4.4 O-RAN-Compatible Controller

While FlexRIC employs an E2-compatible southbound (allowing the agent to interface with any E2-compatible controller), the FlexRIC SDK does *not* provide a fully O-RAN-compatible "Near-realtime RAN Intelligent Controller" (Near-RT RIC) [121]. Instead, it focuses on the core functionality of handling E2 nodes, multiplexing the messages between iApps and RAN functions, and allowing interoperability on the E2AP level between a FlexRIC controller and an E2 node, following the 5G principles of flexibility, forward compatibility, and ultra-lean design.

The overall requirement for a near-realtime controller stipulates that it consists of multiple xApps and a set of platform functions in support of the xApps. A FlexRIC controller specialization targeting O-RAN xApps (see Figure 5.7) implements such platform functions as (E2SM-independent) iApps, and uses custom northbound interfaces to communicate using a REST interface and the RMR (RIC message router) library. xApps implement the Radio Resource Management (RRM) functionality through E2SMs, and interact with the controller for E2 node discovery, message passing, etc. A list of the most important platform functions that need to be present to support xApps, and platform functions to implement interfaces to other controller entities (e.g., towards the non-realtime RIC), is given in the following:

1. A messaging infrastructure that allows xApps to send messages between xApps and the controller. The O-RAN RIC uses the RIC message router (RMR) [50], which routes messages based on message type, distributed to xApps through a routing manager.

2. A database for xApps to read and write information about E2 nodes (R-NIB, radio network information base). Additionally, they can use the database to store, update, and delete additional UE-related information (UE-NIB, UE network information base).

3. An E2 Manager that allows to manage connections with E2 nodes (connect, disconnect, reset), and exports information about present E2 nodes to the R-NIB.

4. A more elaborate subscription management that is not implemented by FlexRIC due to the ultra-lean design goal. For instance, identical subscriptions from xApps should be merged, such that a given E2 node only needs to send one indication, which is then duplicated by the RIC. The subscription manager receives new subscriptions, and forwards them to the subscription management entity of FlexRAN to route the messages to the correct E2 node. Since there is no separate "E2 termination" service as in the reference RIC, the subscription manager also has the role of forwarding any indication to xApps.

5. xApp management for the life-cycle management (deploy, undeploy, configuration) of applications implementing RRM functionality. In the simplest case, unlike the O-RAN design, such xApp manager would not deploy new applications within a Kubernetes cluster, but simply employ a dedicated REST northbound to receive commands about new xApps.

6. An A1 interface [122] to allow policy configuration from a Non-RT RIC, e.g., for RAN guidance and optimization through AI/ML algorithms.

7. An O1 interface [123] for communication with a service management and orchestration (SMO) entity. The RIC exposes such interface to allow management of xApps via NETCONF[3], expose file management services, etc.

8. Additional platform functions for security, logging, or fault management.

Having a simple-to-use E2 controller, as opposed to cluster-based implementations such as O-RAN RIC or ONF SD-RAN, is of high importance. First, such a controller would facilitate research using standard O-RAN xApps from commercial deployments. Second, it significantly reduced xApp development and debugging lifecycle, which does not need to communicate with multiple services. The corresponding FlexRIC controller specialization could be used as a simple-to-use O-RAN RIC replacement, hosting xApps that implement standard O-RAN use cases [124].

[3] NETCONF is protocol to manage and configure network devices. See IETF RFC 4741.

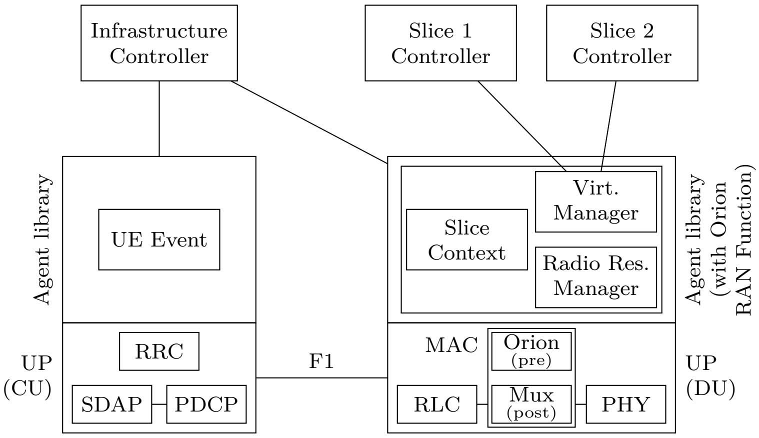


Figure 5.8 – Orion implementation over a disaggregated base station.

5.4.5 Orion

Orion [53] introduced RAN control for multiple tenants while *functionally* isolating slices, i.e., tenants have independent control over their slice. Orion employs a hypervisor on top of the base station to enable per-tenant MAC scheduling (see the taxonomy in Section 3.1). It uses a custom resource abstraction to present a virtualized view of radio resources to tenant's slice controllers. These controllers independently schedule the corresponding UEs, and the hypervisor multiplexes scheduling decisions back into the common resource grid. Through the separate slice controllers, tenants are functionally isolated, and the resource abstraction ensures resource isolation.

Although Orion is not a *controller* specialization, we include it into the present list of controller specializations, as it shows the capabilities of the FlexRIC SDK to implement various SD-RAN designs, of which Orion certainly is one. Through a custom RAN function and accompanying service model, the Orion hypervisor is re-implemented in the FlexRIC SDK, more specifically the agent library, as shown in Figure 5.8. Since Orion manages radio resources, this RAN function is within the DU agent; the CU agent is only required to identify the selected slices, e.g. NR NSSAI. A corresponding slice-specific controller would implement the scheduling functionality.

Orion's functionality can be decomposed into (1) slice and UE management e.g., controller connections and UE associations, and (2) resource abstraction and multiplexing. First, for slice and UE management, the agent library provides means to connect to multiple controllers, and maintains a list of UEs associated to each controller, corresponding to the slice context and UE association managers of Orion. Through E2 subscriptions over a custom, Orion-inspired service model, the slice lifecycle is modeled in the slice context in the Orion RF, such as the slice SLA, associated UEs, etc. Note that Orion's design does not foresee the support of disaggregated base stations. Through a supporting infrastructure controller that also triggers the deployment of new slices and the corresponding controllers, the Orion RAN function is informed about the UE-to-slice association.

For resource abstraction and multiplexing, Orion uses a radio resource manager to attribute radio resources to tenants (enforcing resource isolation), and a virtualization manager to abstract those resources in order to allow a tenant to schedule (virtualized) resources. This functionality is ensured through corresponding Orion RAN function modules: (1) resources are attributed to each slice tenant for the upcoming allocation window, and (2) are virtualized into the "vRRB pools" employed by Orion. The RAN function sends E2 indication (insert) messages to the corresponding slice controllers. These schedule the UEs on the resources, and inform the RAN function about the scheduling outcome through E2 control messages. To apply the scheduling decisions in the user plane, a specific Orion preprocessor might be used within the MAC scheduling framework presented in Chapter 6.

Such specialization could be used to further research the RAN-as-a-service paradigm and in the context of the SD-RAN virtualization layer presented in Chapter 7.

5.4.6 SD-RAN Virtualization Layer

By reusing the agent library on top of the server library, a controller can recursively expose an E2 interface. This can be used to implement a virtualization layer for SD-RAN control that works like a proxy, isolating multiple northbound controllers from each other (see also the simple taxonomy in Section 3.1). A design for concurrent slicing control is described in Chapter 7.

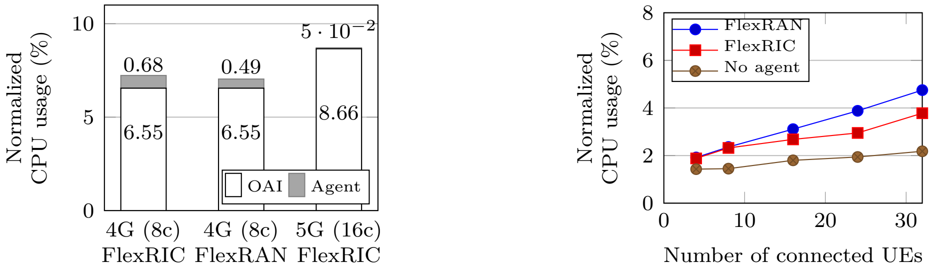
5.5 Performance Evaluation

In this section, we rigorously evaluate the performance of the FlexRIC SDK, starting from the agent library via the E2 abstraction to the server library, and finally compare to the O-RAN RIC. Depending on the experiment, we use either a test agent, or a FlexRIC agent on top of OAI [22], [23] (tag 2021.w20). We deactivate the CPU sleep states on all machines to ensure consistent measurements.

5.5.1 Overhead in the User Plane

We measure the CPU usage introduced by the FlexRIC agent library and compare it to FlexRAN [33] (LTE only). To this end, we export statistics measurement using the built-in statistics of FlexRAN and the integrated statistics E2SMs of FlexRIC at 1 ms frequency; in both cases, we enable all statistics for MAC, RLC, and PDCP (excluding HARQ), covering approximately the same data (PDCP/RLC packet and byte counters, MAC statistics such as CQI and used resource blocks, etc.). Note that both use different encoding schemes (i.e., Protobuf for FlexRAN, here Flatbuffers for FlexRIC); our purpose is to verify that FlexRIC incurs comparable overhead as FlexRAN, which was designed for real-time control.

We measure the CPU usage over both LTE and NR (non-standalone mode, NSA) base stations. We used an RF setup with Ettus B210 radios for both cells. The LTE cell runs on an Intel Core i7 with 8 cores @ 3.2 GHz, uses a bandwidth of 5 MHz (25 RBs)



(a) Radio deployment, LTE/NR.

(b) L2 simulator for LTE.

Figure 5.9 – Normalized CPU usage of FlexRIC and FlexRAN. Note that the CPU of the computing hosting the LTE cell has 8 cores, the NR cell's CPU 16.

and serves 3 UEs at MCS 28. The NR cell runs on an Intel Xeon Gold 6208U with 16 cores @ 2.9 GHz, has a bandwidth of 20 MHz (106 RBs) and serves 3 UEs at MCS 20. Figure 5.9a shows that both FlexRIC and FlexRAN incur a small overhead. The relative overhead decreases when deploying FlexRIC over NR, due to a more demanding PHY.

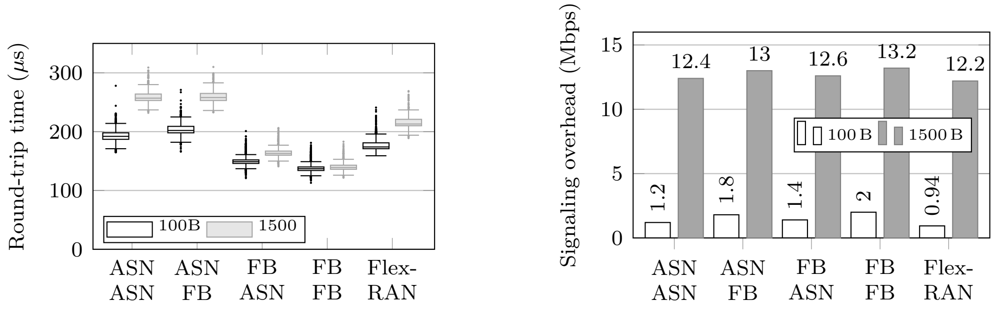
To analyze the overhead for more UEs, we used the "L2 simulator" of OAI, an emulation mode without the physical layer. Figure 5.9b shows that FlexRIC performs slightly better than FlexRAN especially for more UEs, and thus, FlexRIC is valid for the same scenarios where FlexRAN has been validated (e.g., real-time MAC scheduling).

The results confirm the scalability of the FlexRIC agent library, achieving slightly better performance for many UEs (up to 1 % less CPU load for 32 UEs) due to more efficient encoding through Flatbuffers.

5.5.2 Impact of E2AP/E2SM Encoding

In the following, we evaluate the impact of two encoding schemes, Flatbuffers (FB) and ASN.1, for E2AP and the E2SM. E2 enforces a double encoding of messages: a first encoding pass is done for the "inner" E2SM, and then again for the "outer" E2AP. We modified the "Hello World" service model (HW-E2SM) provided by O-RAN to perform a *ping* by sending a control message to the RAN function, to which the agent responds with an indication message. To also study the impact of the E2SM encoding, we translated the E2SM 1:1 from ASN.1 to FB. Further, we modified FlexRAN to measure the Round-Trip Time (RTT) and signaling rate. FlexRAN does not oblige a double encoding as described above.

To measure the RTT, the iApp pings the agent every second for small (100 B) and medium (1500 B) message payloads. The results in Figure 5.10a indicate that switching from all-ASN.1 to all-Flatbuffers encoding reduces the average RTT by roughly 25 % and 66 % for small and medium payloads, due to the encoding overhead of ASN.1. Using an ASN.1/FB (E2AP/E2SM) encoding even increases the round-trip time, since the larger FB E2SM message (for each FB message, we observe 30-40B overhead) needs to be encoded again by ASN.1 for E2AP. Thus, where ultra-low latency is required, FB might be preferable, especially for larger payloads, but networking delay (our Ethernet-based



(a) Round-trip time.

(b) Signaling rate.

Figure 5.10 – Comparison of E2AP/E2SM encoding schemes using E2SM-HW *ping*.

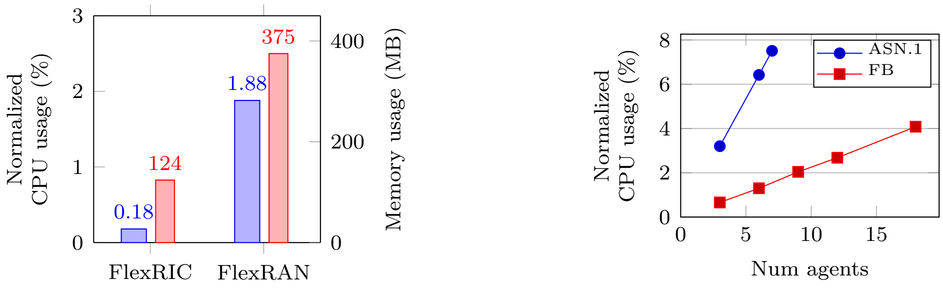
campus network has RTTs below 1 ms) would make this difference much less pronounced. The results of FlexRAN indicate the arrival of ping replies *at its networking queues*. We observe that its absolute RTT and relative RTT increase is between that of FB and ASN.1 cases, most likely due to the Protobuf encoding, but always slower than the FB E2AP encoding of FlexRIC. Furthermore, due to FlexRAN's design, an application has to poll for the results every ms; thus, in FlexRAN, the RTT from an application's point of view *is always one ms*, which we omit for readability.

To put the RTT improvements into perspective, we measured the generated message signaling rate for a high *ping* rate (one packet every 1 ms, which is 4G's transmission time interval). The results in Figure 5.10b indicate that switching from an ASN.1/ASN.1 to an FB/FB encoding increases the signaling rate by 67 % for small payloads due to the overhead of FB; for large payloads, however, the signaling rate overhead is almost negligible. Comparing the "mixed" encodings, we see that the FB/ASN.1 encoding signaling overhead only slightly increases compared to ASN.1/ASN.1, whereas the ASN.1/FB combination does not decrease signaling load, rendering this combination useless. In comparison, FlexRAN has the smallest signaling rate, since it does not enforce a double encoding. This advantage almost vanishes for large payloads.

In conclusion, Flatbuffers encoding is useful for larger payloads and/or frequent message exchange under the assumption that the wired capacity is not the bottleneck at least for E2AP. On the other hand, if the wired capacity is the bottleneck or if infrequent, small messages are forwarded, ASN.1's processing overhead in terms of CPU is balanced by a lower signaling overhead. In general, it might be preferable to use Flatbuffers in E2AP, as it only slightly increases signaling overhead while keeping RTTs low.

5.5.3 Scalability of the Controller

Next, we evaluate the scalability of the FlexRIC server library in terms of CPU utilization and memory usage and compare it to FlexRAN [33]. We use the FlexRAN controller specialization presented in Section 5.4.3 (in the following called simply "FlexRIC" to avoid confusion with the original FlexRAN controller), which employs a statistics iApp that saves incoming messages to an in-memory data structure like FlexRAN. We only



(a) Comparison to FlexRAN.

(b) Overhead for many UEs.

Figure 5.11 – CPU usage at the controller.

consider the agent-to-controller direction, which typically carries more traffic than in the opposite direction, even for high-traffic scenarios such as remote scheduling [33]. Both controllers are on an Intel Core-i7 machine with 12 cores at 3.2 GHz. As can be seen in Figure 5.11a, FlexRIC incurs only one tenth of the CPU usage of FlexRAN, due to using Flatbuffers instead of Protobuf. Also, FlexRIC organizes its internal data structure more efficiently, leading to reduced memory consumption.

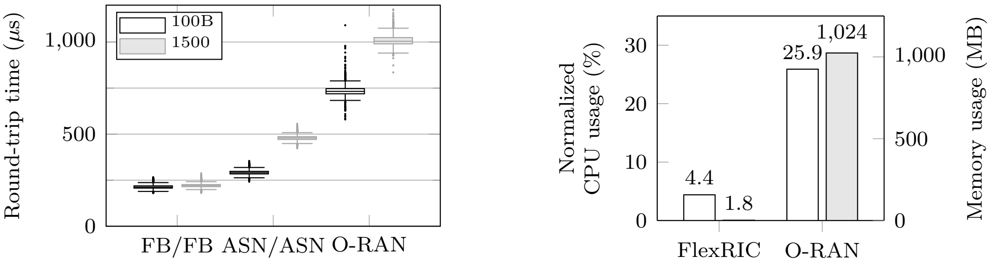
To further understand the limitations of FlexRIC, we test using dummy test agents (not connected to any base station) that export the same statistics (in FB) as from a real base station (statistics for MAC excluding HARQ, and RLC and PDCP), each agent emulating a connection of 32 UEs with a unique default bearer. Figure 5.11b shows the CPU utilization of FlexRIC over varying number of agents when using a FB or ASN.1 encoding for E2AP. It is apparent that FB has around 4 times lower CPU usage than ASN.1. Since FB's design avoids an explicit decoding step, reading directly from "raw" bytes, the subscription management in the server library can look up the corresponding subscription much faster, resulting in less CPU usage, directly translating to serving many more agents. This suggests that ASN.1 encoding on E2AP can become a limiting factor in terms of CPU, whereas FB is rather limited by the network, as for 18 agents, the signaling reaches almost 700 Mbps. When reducing the message frequency to 10 ms, the FlexRIC SDK was furthermore able to handle around 100 agents (not depicted in the figures), confirming FlexRIC's scalability.

5.5.4 Comparison to O-RAN RIC

The reference O-RAN RIC architecture relies on micro-services, containerized through Docker and orchestrated by Kubernetes. Correspondingly, each micro-service's image uses disk space. Table 5.3 lists the image sizes of a dockerized FlexRIC for a RTT test (as in Section 5.5.2) and a statistics use case (as in Section 5.5.3), and the O-RAN RIC platform (15 platform components) with corresponding O-RAN xApps. It is apparent that due to containerization of all platform components individually, the O-RAN RIC requires much more storage, going contrary the 5G design principle of ultra-leanness, forcing to need resources even for scenarios where the advantages offered by Docker and Kubernetes are not needed.

Table 5.3 – Docker image sizes.

Component	Size (MB)
FlexRIC + HW-E2SM	76
FlexRIC + Stats E2SMs (FB)	94
O-RAN RIC (platform)	2469
HW xApp	170
Stats xApp	166



(a) Round-trip times for two hops.

(b) CPU and memory usage.

Figure 5.12 – Comparison of O-RAN RIC and (dockerized) FlexRIC.

The O-RAN architecture impacts the latency, since it imposes two hops for messages (from xApp via "E2 termination" to agent). In the following experiment, we measured the round trip time with two distinct payloads (i.e., 100 B and 1500 B) as in Section 5.5.2, between (i) a FlexRIC agent and a FlexRIC controller utilizing the E2SM-HW, and (ii) a FlexRIC agent and O-RAN RIC. In FlexRIC, we use a relaying controller to emulate two hops, which, unlike O-RAN RIC, is not imposed by FlexRIC but added to carry out a fair comparison. As it can be observed from Fig. 5.12a, O-RAN RIC increases the RTT by at least three times for small and two times for medium load compared to FlexRIC. Note that FlexRIC incurs less jitter for the RTT.

In a second experiment, we consider a monitoring use case similar to Section 5.5.3, in which 10 dummy agents export MAC statistics (excluding HARQ) for 32 UEs using E2AP indication messages every ms. We measure CPU usage and memory footprint as reported by *docker stats*, shown in Fig. 5.12b. (For the O-RAN case, we added the CPU and memory usage of the platform components and xApp.) The design of O-RAN RIC imposes that indication messages are decoded twice, once in the "E2 termination", and the xApp. The CPU usage of FlexRIC is 83 % lower than O-RAN RIC, the O-RAN xApp alone using as much CPU as FlexRIC, and the rest being used by the "E2 termination". We attribute this to an inefficient implementation (we used the default "E2 termination" image, version 5.4.8). Also, the O-RAN RIC uses much more RAM, which is due to the high number of platform functions running in separate containers, which are also partially written in higher-level languages, such as Go, as opposed to FlexRIC's C. These findings confirm that O-RAN RIC imposes additional overhead by design for communication between xApps and agents, even if not required, which might become a bottleneck.

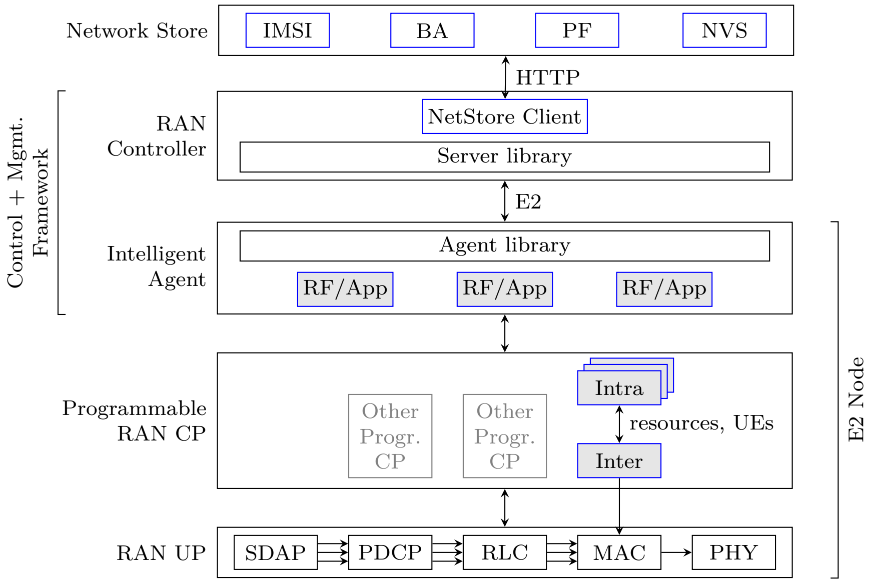


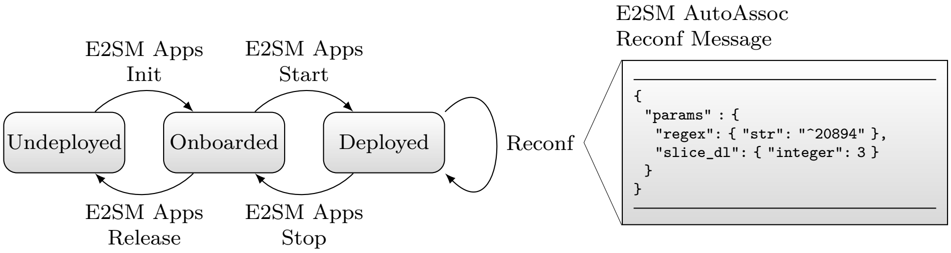
Figure 5.13 – Integration of a (trusted) network store in the FlexRIC SDK (blue boxes). Gray boxes mark extensible CP functionality in the agent or RAN CP.

5.6 FlexRIC Extension through a Network Store

As has been discussed in Section 4.2, properties such as resource isolation, sharing, and forward- and backwards compatibility need to be enforced to enable a multi-service SD-RAN control. Through a selection of suitable RAN functions, the FlexRIC SDK can enable multi-service programmability in the RAN and fulfill mentioned properties.

A network store can fulfill the service requirements of network slices by providing customized network functions [75]. Such a store can enable service customizations and RAN extensions, two additional properties for efficient multi-service RAN control. We now discuss how such a network store might be integrated with the FlexRIC SDK. It is then possible to (1) enhance and extend the behavior of the control plane to cater for the requirements of services, and (2) bring intelligence to the RAN through data-driven control beyond standardized functionality.

Figure 5.13 shows how to integrate a trusted distribution platform that serves as the network store and provides a number of network functions that can be deployed in the RAN. They are fetched through a "NetStore" client in the RAN controller. The RAN controller is implemented through FlexRIC and uses E2 primitives for deployment, i.e., subscriptions with a new policy. The agent library provides an environment to load new RAN functions to extend the functionality of the RAN, similar to how mobile operating systems extend functionality through new applications. Such RAN functions might extend the agent with service models in order to provide additional information to the controller. Alternatively, through existing RAN functions, it is also possible to customize the (programmable) RAN CP with an object as shown for the intra- and inter-schedulers in Figure 5.13 (see Chapter 6 for details).



```
{
  "params" : {
    "regex": { "str": "^20894" },
    "slice_dl": { "integer": 3 }
  }
}
```

Figure 5.14 – RAN function lifecycle and E2SM reconfiguration message.

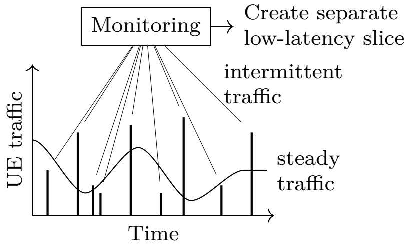


Figure 5.15 – Principle of recognizing traffic patterns of low-latency users.

Rule-based configuration and Intelligence. New UEs connecting to the base station are to be associated to a certain slice. Upon connection of a UE, it delivers hints on preferred slices, such as the NSSAI or simply the IMSI or selected PLMN, which is forwarded to the controller to reconfigure the slice association. However, since all required information is already available at the base station, no controller intervention is necessary and an association of UEs to slices can be pre-configured (e.g., prefix-match for IMSIs, i.e., starting on a pattern). To accomplish this, the controller deploys an application/RAN function within the agent to auto-associate UEs to slices. The lifecycle of the application `AutoAssoc` is shown in Figure 5.14, which is onboarded and deployed through an E2SM between the controller and agent. Further, a separate E2SM allows configuring `AutoAssoc` with the association rules. For instance, in the shown example, all UEs with IMSI starting on `20894` are associated to slice `3`.

Further, apart from simple rule-based slicing, a data-driven approach might be used to intelligently slice the network based on traffic patterns. Assume that a RAN uses the NVS slice algorithm to slice the radio resources for two slices. Due to the specificities of this algorithm, scheduling latencies might be incurred, negatively impacting the RTT. A data-driven slicing approach allows to reduce the RTTs of low-latency users. The recognition of the low-latency users necessitates a real-time analysis of the traffic of such users, which can be resource intensive when done frequently. Therefore, the `burst_analysis` application at the agent relieves the controller from the overhead analyzing the traffic of all UEs, and instead analyzes the traffic patterns, as shown in Figure 5.15. If a certain UE has only intermittent, bursty traffic of a specific pattern, it will be considered "latency-critical" traffic. Such traffic is handled in a separate low-latency slice: the application loads the QoS-aware slice scheduling algorithm (described in Section 6.3),

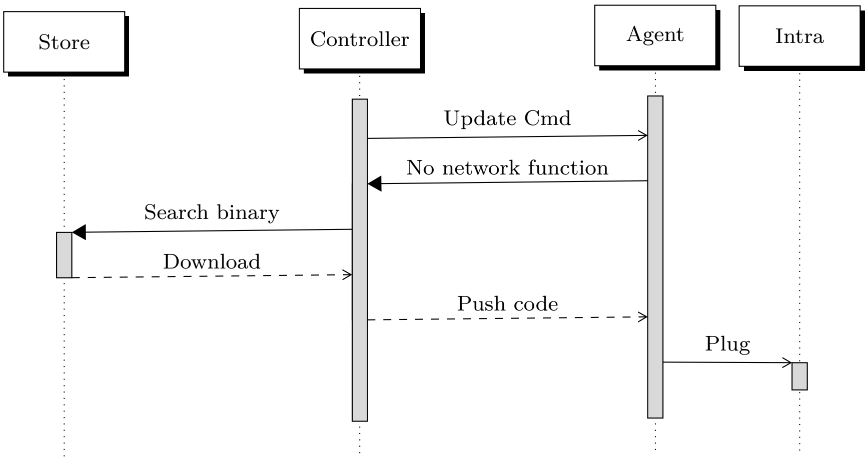


Figure 5.16 – Sequence diagram for extending CP functionality.

creates a separate slice and associates the respective UE to this individual slice. This reduces the RTT significantly by on average 50% as reported in Section 6.4.2. As soon as the traffic pattern does not match anymore, the created low-latency slice is removed and the UE associated to the original slice. In the demonstration[4], we used *ping* with a data size of 200 B and 0.25 s interval time to simulate such traffic. A service model might be employed to make the analysis more configurable.

Behavior Extension through Network Store. Further, the control plane of the RAN can be enhanced and extended with new behavior by plugging RAN CP network functions on-demand from the trusted network store. If after a slice update initiated by the controller, new requested functionality such as a user scheduling implementation is not available at the RAN, it can be downloaded from the network store as shown in Figure 5.16. In this case, the agent requests a specific network function that is needed to fulfill the request. The controller checks on the store for a network function and sends it to the agent, which is then plugged by the agent into the RAN CP. Once the slice is decommissioned, the network function will be disposed.

5.7 Discussion and Future Work

FlexRIC is a modular SDK to build a number of specialized controllers while guaranteeing high performance with minimal overhead, which is in line with the 5G design principles. It also allows adapting controllers to novel, yet unforeseen use cases. The following suggestions represent discussion and indications for future research in the context of FlexRIC.

First, there is a range of specialized controllers that might be built on top of the SDK. For instance, it has been shown that SD-RAN controllers optimized for real-time operation

[4] See `https://youtu.be/KxTEpFe5dJU`

are not scalable enough for the large number of devices expected for 5G [125], [126]. Distributed controllers have shown to increase scalability for (wired) SDN networks [127], [128]. A distributed SD-RAN platform could be implemented on top of the FlexRIC SDK, promising to handle more base stations and devices in the network.

Second, a number of use cases with associated E2SMs have been considered in the context of FlexRIC. However, there are many use cases, e.g., massive MIMO optimization [124], for which novel E2SMs have to be developed in order to enable the corresponding use case.

Third, artificial intelligence (AI) and machine learning are promising approaches to cope with the increasing complexity of modern RANs [129], [130]. The A1 interface is foreseen by O-RAN to guide a near-realtime RIC, which could be built using the FlexRIC SDK, through policies based on machine-learning decisions. Developing such interface would be a starting point for AI guidance of applications. In the long term, such a controller can support research of machine learning approaches for mobile network control.

To foster the adoption of FlexRIC, an interface of the FlexRIC SDK for other programming languages than C can be developed, e.g., using SWIG[5]. For instance, a Python interface would allow an easier integration with other Python frameworks, simplifying the development of new, specialized controllers.

5.8 Conclusion

In this chapter, we proposed FlexRIC, an SDK that serves as the pillar to build specialized, multi-service SD-RAN controllers. Using the abstractions provided by the SDK, it is possible to compose customized controllers that smoothly adapt to the envisioned state-of-the-art 5G scenarios. We presented the components of the SDK, a detailed performance evaluation including comparisons to other SD-RAN controllers, and described various controller specializations. In the context of this work, we designed multiple service models for monitoring, slicing (an in-depth evaluation is in Chapter 6), and traffic control (see Appendix A.1). Finally, FlexRIC enables the implementation of “recursive” virtualization layer, which will be the topic of Chapter 7.

[5] Simplified Wrapper and Interface Generator: `http://swig.org/`

Chapter 6

Flexible MAC Framework

6.1 Introduction

5G is envisioned as an ultra-flexible, ultra-lean network that can be adapted to different deployment scenarios, and that is supposed to multiplex a variety of services with heterogeneous requirements onto a shared, common infrastructure through corresponding slices. To do so, 5G/NR provides great flexibility with different protocol stack enablers; especially the PHY supplies a number of new features, such as bandwidth parts, extended beamforming capabilities (especially for mmWave deployments), or dynamic TDD. A crucial point is to exploit this flexibility, and adapt the way radio resources are scheduled and allocated depending on factors such as the considered deployment scenarios and services that are currently transported.

The MAC scheduler has particular importance here, as it is the "brain" for short-term radio resource allocation. However, existing works do not address deployment scenarios or adaptability to service requirements. In short, they are too simplified, as they do not consider the necessary scheduler adaptability for different deployments and NR features, and typically consider only a single slice algorithm for multi-service resource allocation on the MAC layer in the RAN. For an overview of programmability approaches in the RAN, we refer the reader back to Section 3.1.

In order to support different deployment scenarios while multiplexing services, it is important to have an extensible and forward-compatible design for the MAC scheduler that (1) makes use of NR's flexibility and employs scheduling schemes that are adapted to deployment scenarios (e.g., sub-6 GHz and above), and (2) multiplexes services with heterogeneous requirements (eMBB, URLLC) over the radio resources according to their respective SLAs. The multiplexing of slices therefore depends on the SLA, which needs to be mapped to radio resources through suitable slice scheduling algorithms, guaranteeing performance isolation and sharing of resources for multiplexing gains. Further, the scheduler design should foresee a customization of the user scheduling operation to adapt to service requirements. Finally, it must be programmable to allow the management and control of slices transporting the services, which should happen dynamically without any impact on the performance of other slices.

This chapter addresses these challenges. First, in Section 6.2, we design a MAC

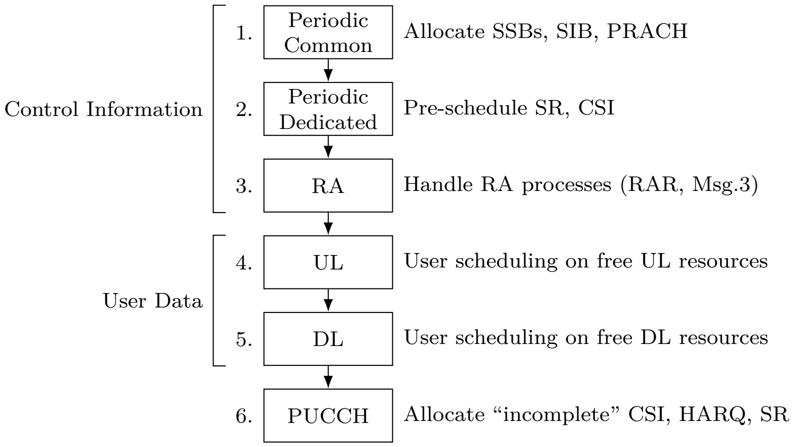

Control Information
User Data
1. Periodic Common
Allocate SSBs, SIB, PRACH
2. Periodic Dedicated
Pre-schedule SR, CSI
3. RA
Handle RA processes (RAR, Msg.3)
4. UL
User scheduling on free UL resources
5. DL
User scheduling on free DL resources
6. PUCCH
Allocate "incomplete" CSI, HARQ, SR

Figure 6.1 – The steps of the 5G/NR OAI scheduling pipeline, executed in every slot. Not all steps are applicable to every slot.

scheduling framework to flexibly adapt scheduling behavior to deployment scenarios and service requirements. It puts particular emphasis on a separation of concern between slice and user scheduling algorithms to enable flexibility and extensibility for multi-service operation, and to encapsulate the user scheduling algorithms of slices, governed by a slice scheduling algorithm. To provide configurability through FlexRIC, we furthermore design a multi-RAT abstraction that allows to dynamically control this framework. Second, in Section 6.3, we present a QoS-aware slice algorithm with priority-based resource allocation to simultaneously schedule services corresponding to three different usage scenarios, which integrates neatly as a slice algorithm into the proposed MAC framework design. Third, Section 6.4 provides extensive simulation, emulation, and measurements over real hardware of the proposed scheduling framework and QoS-aware slice algorithm for 4G and 5G systems. To this end, we implemented the framework in OAI [22], [23] for 4G/LTE and 5G/NR, including multiple slice and user scheduling algorithms. Finally, future work directions are discussed in Section 6.5.

6.2 MAC Scheduling Framework

In the following, we describe the MAC scheduling framework to adapt the scheduling behavior towards deployment and service requirements. As this framework has been implemented within OpenAirInterface (OAI), and to provide some context, we first describe OAI's NR scheduling pipeline. Then, we proceed with the MAC scheduling framework and its multi-service extension, and describe the E2SM.

6.2.1 Overview of OAI NR Scheduling Pipeline

Figure 6.1 shows the scheduling pipeline as implemented in OAI. It consists of a number of "high-level" scheduling tasks that are triggered depending on the current slot. Not all tasks are executed in every slot; for instance, broadcast information should be sent periodically in a fixed slot number, and downlink user scheduling is executed in every *downlink* slot, which can vary depending on the TDD configuration. The pipeline implicitly prioritizes control information over user data: periodic common control information (i.e., information to all UEs, like broadcast) is prioritized over periodic dedicated control information (i.e., per-UE information, like random access), which in turn has precedence over the actual data scheduling.

Note that the pipeline steps are almost the same for both 4G/LTE and 5G/NR. Therefore, we focus on 5G/NR and highlight important differences to 4G/LTE where necessary. OAI does not implement multiple BWPs as specified in 5G/NR, which result in different slot lengths for different numerologies. However, for the design of the MAC scheduling framework, we assume that a pipeline would be run independently for each BWP.

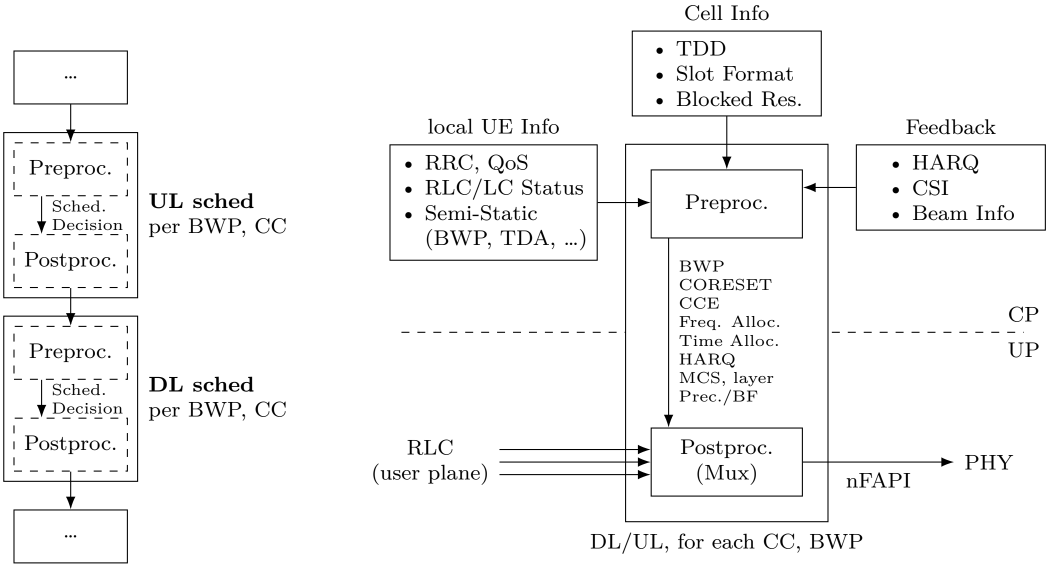
The described pipeline is specific to OAI. However, given that this ordering is well motivated, as shown above, it is reasonable that another implementation applies similar steps.

Periodic Common. First, the scheduler allocates common periodic control information. This includes synchronization signals that a device uses to find and synchronize to a cell, e.g., the synchronization signal block (SSB) in 5G/NR or the master information block (MIB) in 4G. Further, basic radio resource configuration is provided through system information, e.g., the system information block 1 (SIB1), necessary for random access (RA). Also, the scheduler allocates resources for the physical random access channel (PRACH). Finally, resources are marked in an internal structure to prevent reuse of these resources in later steps.

Periodic Dedicated. Second, in the case of 5G, periodic, dedicated control information for the Physical Uplink Control Channel (PUCCH) is scheduled internally. Such information includes Scheduling Requests (SRs) or Channel State Information (CSI) that has been allocated through RRC signaling upon connection of a UE. Note that this step is not present for 4G, as PUCCH is allocated statically.

Random Access. In this step, possible PRACH attempts are detected, and the scheduler transitions through the RA state machine for a particular UE. This includes scheduling of Msg. 2 (RA response, RAR), and Msgs. 3 and 4 are used to perform contention resolution. Afterwards, the scheduler creates a MAC Context to keep track of the connected devices at the MAC layer.

Uplink and Downlink traffic. At this stage, UL and DL resources can be scheduled. Depending on the deployment and use case, different MAC scheduling schemes might be employed, implemented through a separation of the scheduler into preprocessor and postprocessor stages, as described in Section 6.2.2. Both UL and DL scheduling decisions result in DCIs to be transmitted in the Physical Downlink Control Channel (PDCCH). To ensure that any UE can be scheduled at any time to guarantee its synchronization, UL transmissions are scheduled first, such that no DL-specific DCI might block the



(a) Pipeline callflow.

(b) Separation of scheduling logic into CP/UP (here for DL).

Figure 6.2 – Decomposition of UL/DL scheduling steps into preprocessor and postprocessor.

scheduling of a UL grant. The DL phase is also responsible for multiplexing HARQ feedbacks with SR and CSI that have been allocated previously. Therefore, the DL scheduler also partially allocates UL control information, which needs to be left free in the UL scheduling stage.

PUCCH. Finally, the PUCCH phase (5G only) allocates remaining "open" HARQ, SR, and CSI information in case they have not yet been allocated, e.g., PUCCH resources with only 1 bit HARQ information.

6.2.2 Preprocessor/Postprocessor Separation

To allow for adaptation in the scheduling pipeline towards different deployment scenarios and diverse services, we decouple both the UL and DL user scheduling phases into a *preprocessor* and a *postprocessor*, as it is shown in Figure 6.2a. The preprocessor is an exchangeable module that takes the scheduling decisions, whereas the postprocessor applies the preprocessor decisions to the user plane.

The postprocessor is tightly bound to the decisions of the preprocessor: it follows the scheduling decisions of the preprocessor to construct the MAC transport blocks, consisting of MAC control elements and the actual data that is forwarded from the RLC, writes UE statistics, and sends physical layer instructions that are created from the information passed from the preprocessor. From a conceptual point of view, the split between the preprocessor and the postprocessor implements a separation of control plane and user plane, as shown in Figure 6.2b, where the scheduling decisions of the preprocessor are enforced in the user plane by the postprocessor.

The preprocessor implements deployment-specific and use-case-specific scheduling logic, such as the implementation of slice scheduling, BWP handling, or dynamic TDD schemes. It encapsulates the algorithmic decisions that need to be taken for a deployment and use case type, and thus is aware of the limitations of the use case at hand. This includes knowledge on

- Frequency range (sub-6 GHz/FR1, mmWave/FR2) and beamforming,
- Operation mode (Frequency Division Duplexing (FDD)/TDD, TDD schemes),
- Cell configuration (secondary uplink, carrier aggregation, beamforming),
- Joint scheduling of multiple cells including knowledge of the fronthaul,
- Service multiplexing (slice algorithms),
- Knowledge about the state of other cells (data-driven scheduling and prediction using machine learning).

For instance, a preprocessor for scheduling FR2 (operating at mmWave spectrum) would schedule multiple UEs exclusively in a TDD fashion within a slot, since analog beamforming is typically employed to beamform the signal into a particular direction, focusing it only on one UE at a time. Therefore, the separation follows 5G principles, as the preprocessor is adapted to the deployment and use case (flexibility), can be extended to new use cases and deployments (forward compatibility), and only has to implement what is needed for an efficient implementation (ultra-lean design).

Since UL and DL scheduling are executed in separated steps, the preprocessor consists of two exchangeable modules for the UL and DL preprocessors. However, depending on the deployment and use case, there might be constraints on scheduling decisions, necessitating coordination between the DL and UL preprocessors. For instance, dynamic TDD requires coordination where one preprocessor should decide on the transmission direction in future slots, which has to be followed by the other preprocessor.

In order to operate, the preprocessor is dependent on information about the configuration of individual users, their state, and dynamic scheduling constraints. As shown in Figure 6.2b for DL, the preprocessor uses three groups of information:

Cell information includes static and semi-static information that is common to all UEs, such as the TDD and slot format configuration or resource blocks that cannot be used for user scheduling.

Local UE-specific information is (a) RRC configuration, such as including QoS information for multiple bearers or configuration for Discontinuous Reception (DRX), (b) semi-static information, such as the currently active BWP for the given UE, and (c) the current user plane state, such as RLC buffer status (number of bytes, head-of-line delays, etc).

Feedback information from the UEs to the eNB/gNB is critical for proper operation, such as HARQ Ack/Nacks, information for adaptive modulation and coding (e.g., channel state information) or uplink scheduling for the UL preprocessor (SR, Buffer Status Report (BSR)), and beam information for selecting beams in case of analog beamforming.

The preprocessor uses this information to make scheduling decisions; this might include long-term decisions such as slot format selection in the case of dynamic TDD. For every UE that needs to be scheduled, it returns per-UE information to the postprocessor that is used to multiplex data of UEs and trigger the physical layer while allowing to use advanced 5G features:

- **BWP**: the postprocessor might generate DCIs to inform the UE about a BWP switch. (5G only)
- **CORESET**: the control region that should be used to schedule the DCIs, if multiple CORESETs are configured within the BWP. (5G only)
- **CCEs** to use for the DCI: this avoids CCE conflicts when allocating the UEs, as otherwise the UE allocation would need to be dropped.
- **Frequency-domain allocation**: the RBs to use for this UE.
- **Time-domain allocation**: the index of the RRC-stored time-domain allocation table, which allows scheduling multiple UEs in the time-domain, e.g., when considering analog beamforming. (5G only)
- **HARQ process** to use for this UE: e.g., to select a retransmission among potentially multiple transmission with differing criticality.
- **Modulation and Coding Scheme (MCS)** and **layer information**: information about the MCS and MCS table for link adaptation, and how many layers to use for a given UE.
- **Precoding information** to multiplex UEs in space (digital beamforming) or **beam parameters** for mmWave transmissions (analog beamforming, 5G only).

Therefore, the preprocessor is completely free in making any scheduling decisions, and it is possible to adapt it to any use case and using any (possibly future) 5G feature as long as the interface supports it.

6.2.3 Multi-Service Preprocessor

Whereas the presented MAC scheduling framework enables high flexibility to customize the scheduler towards specific deployment scenarios and use cases, it does not allow multiplexing multiple services onto the radio resources. However, multi-service/multi-tenancy within the different use cases is one of the cornerstones of 5G. This calls for a multi-service extension of the MAC framework. Through a particular slice algorithm, the requirements of the services can be mapped to radio resources, such as data rate

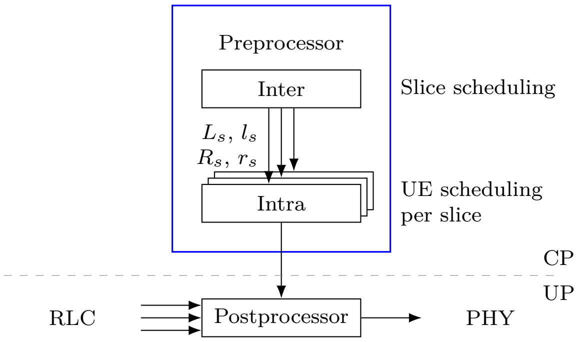


Figure 6.3 – Decomposition of the preprocessor into an inter-scheduling and an intra-scheduling phase. This allows to implement various slice algorithms, and helps to manage bandwidth parts, DRX schemes, etc.

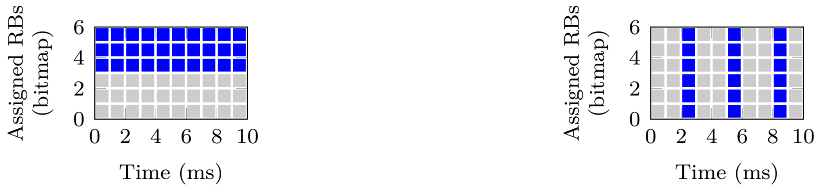
requirements for eMBB, taking into account the efficiency of the transmissions, or latency requirements for URLLC in order to prioritize services.

Depending on the services, as well as the considered deployments, slicing might need to be carried out differently, necessitating different algorithms. For instance, consider the case of mmWave radio spectrum, in which two slices cannot be scheduled at the same time, as their UEs cannot be scheduled concurrently due to analog beamforming, and a separation in time with a specific slice algorithm has to be achieved, while guaranteeing resource isolation. Therefore, a flexible slicing system is necessary that allows to use different slice algorithms depending on the usage requirements of the services in the considered scenarios.

For complexity reasons, typical approaches for slice scheduling separate the slice and user scheduling phases into two disjoint functionalities [54]. We follow this approach and divide the preprocessor in an *inter*-scheduling phase, handling the distribution of resources to groups of users (*slices*), and an *intra*-scheduling phase, that distributes the resources to individual UEs, as shown in Figure 6.3. In short, the inter-scheduler works at the level of slices (groups of UEs), whereas the intra-scheduler carries out per-user scheduling (radio bearer level). Note that there exist slice algorithms that do not follow such a separation for optimality reasons (see Section 3.2.2); it is still possible to implement such algorithms with a dedicated preprocessor, without taking the inter-intra separation into consideration.

The task of the inter-scheduling phase is to decide which resources are attributed to a slice in each slot. The way resources are attributed varies depending on the slice algorithm. Therefore, the inter-scheduler calls, in each slot, the intra-scheduler for each slice s with the following information:

1. a list L_s of UEs that might be scheduled,
2. the maximum number of UEs l_s to schedule,
3. a bitmap R_s of resources allowed for that slice, and
4. the maximum number of resource blocks r_s to schedule.



(a) Static slicing; each slice has 3 assigned RBs each slot.

(b) Slicing using NVS; slices have 33 % and 66 % resources over 10 ms.

Figure 6.4 – Example resource bitmap(s) over time for two slice algorithms.

This interface captures the "essence of a slice", which at the RAN level is not more than a list of UEs and associated resources, governed by a particular slice algorithm. Note that a slice algorithm effectively *abstracts* the resource information, as further explained in Section 6.2.5.

With this information, the inter-scheduler is able to implement any slice algorithm by identifying the resources for each slice in a given slot, and calling the corresponding intra-schedulers of each slice together with the allowed resources and the slice user list. The task of the intra-scheduling phase is to perform both channel-aware and QoS-aware scheduling for a slice. It encapsulates a user scheduling scheme, and can be chosen independently for every slice. This also simplifies the implementation of the user scheduling algorithms, as the inter-scheduler omits resources from the resource bitmap that should not be scheduled, such as RBs carrying broadcast information. The scheduling output of the intra-scheduler algorithm(s) is finally used as the output of the preprocessor.

As an example, consider two slices that are active in the RAN:

1. In the case of static slicing, each slice has a fixed number of resources that it uses in every slot. Hence, the inter-scheduler calls each of the intra-schedulers with a fixed, slice-specific bitmap in every slot, as shown in Figure 6.4a, and the intra-scheduler schedules users accordingly.

2. NVS [24] schedules all resources for one slice in each slot. Hence, the inter-scheduler selects a slice to schedule, and the bitmap indicates all resources as shown in Figure 6.4b.

The purpose of limiting the number of UEs is to schedule multiple slices in one slot: since only a finite amount of DCIs can be allocated, limiting the number of UEs avoids that one slice blocks all DCIs, preventing another slice from making UE allocations. Similarly, limiting the number of resource blocks allows giving freedom to a slice to select its resources within a superset of its allowed resources, e.g., for slices requiring high reliability.

Further, not only the resource bitmap and the scheduled slices might vary on a slot level: depending on the use case, the list of UEs might change, too. For this reason, the inter-scheduler does not signal a *slice* (via its ID), but a *list of UEs*. For instance, to implement DRX efficiently, the inter-scheduler could select the UEs that are active in a given slot, and the intra-scheduler would schedule only those UEs. Thereby, the inter-intra preprocessor separation is not restricted to slice scheduling, and can be used more generally to signal UEs that are *eligible and capable* of being scheduled.

To enable slicing while staying compatible to any deployment or use case, the resources are signaled with respect to the current numerology. This means that the inter-scheduler has the responsibility to ensure that the intra-scheduler is only called for RBs that do not overlap with any allocated RBs in other numerologies. This simplifies the implementation of intra-schedulers, since they only need to handle single numerologies, and allows a straight-forward use of BWPs as "PHY-layer slices" by partitioning in the inter-scheduler.

6.2.4 Implementation

The presented MAC framework is RAT-agnostic. Hence, we implemented it in OAI for LTE and NR.

In the LTE version, we modified the existing "default" MAC scheduler (as opposed to the alternative "fairRR" scheduler[1]) to support the presented MAC scheduling framework. We implemented the user scheduling algorithms RR, PF, and MT, as well as the slice scheduling algorithms for "no" slicing[2], static slicing, NVS, "SCN19" (the algorithm presented in Section 6.3), and the EDF slice algorithm [91]. The "no" slicing inter-scheduler for both UL and DL, as well as the intra-schedulers, have been implemented in roughly 1100 lines of code (LOC); the slice algorithms for static, NVS, and SCN19 slicing (DL only) are implemented in roughly 1100 LOC.

The NR version of OAI had a simplistic, single-UE scheduler which allocated all resources in every slot in both DL and UL to a single UE. We implemented a new multi-UE MAC scheduler following the design of the presented MAC scheduling framework in roughly 3000 LOC, including "no" slicing and a PF scheduler, in DL and UL. For slicing, we added the NVS algorithm in roughly 400 LOC.

From an implementation point of view, the inter-scheduler is equal to the preprocessor, i.e., changing the slice algorithm is equivalent to changing the preprocessor, which then calls the intra-schedulers depending on the configured slices. This is shown in the sequence diagram in Figure 6.5. First, the generic pipeline calls into the postprocessor, which immediately calls a preprocessor. Here, the preprocessor implements the static slice algorithm (function `static_dl()`), which is equal to the inter-scheduler. According to the static slicing algorithm, it calculates the resource bitmaps and number of UEs per slice, and then calls the intra-scheduler with the RR algorithm for two slices with the corresponding parameters.

6.2.5 Slicing E2SM

From an E2 point of view (see Section 2.2.2), the MAC scheduling framework is a RAN function that can be controlled via an E2SM. This E2SM provides means to change the components of the RAN function (preprocessor) flexibly at runtime. More precisely, the inter-scheduler and intra-scheduler are configurable in order to define the slice scheduling

[1] See also in the OAI wiki: `https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/fairRR-scheduler-and-multiple-UEs-extensions`

[2] "No" slicing behaves like a slice algorithm that gives all resources to a single slice.

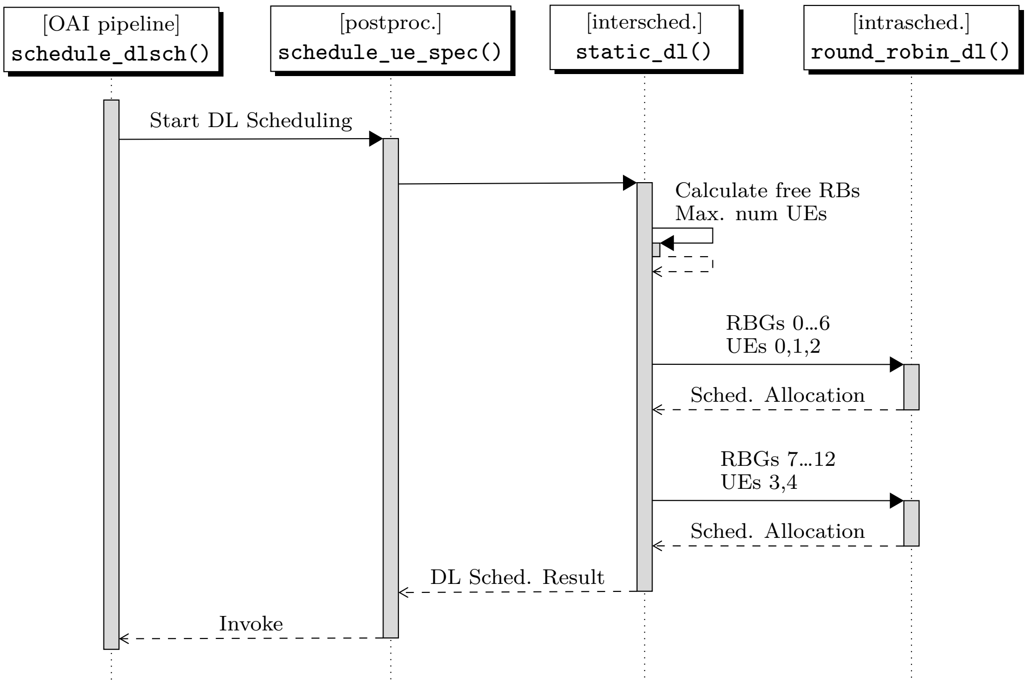


Figure 6.5 – Sequence diagram for the 4G DL scheduling flow with static slicing and two configured slices.

algorithm, deploy slices, define their resources, set user scheduling policies, and associate UEs to slices. More generally, the E2SM provides an *abstraction* to program the RAN through a RAN controller to customize and even extend the scheduler operation. In the following, we present the corresponding Slicing E2SM.

The Slicing E2SM allows to configure a slice algorithm and the slices, and is independent of the employed RAT. For each slice, the following configuration can be set: (1) generic configuration (state) like the slice ID and slice label, (2) the slice's user scheduling algorithm (inter-scheduler, processing), and (3) slice algorithm-specific resources (per intra-scheduler), e.g., RBs for static slicing, rate and capacity (in percent) for NVS, abstracting the radio resources according to the slice algorithm-specific slice

Resources	Processing	State
• Slice Resources depending on algorithm - Resource Blocks (Static) - Rates/Percentage (NVS) - Max. %, Avg. Window	• Slice Algorithm • User Sched. Algorithms	• Slice Conf./IDs • UE Association • Labels

Figure 6.6 – Mapping of the E2SM components to resources, state, and processing. The choice of the slice algorithm (inter-scheduling) has an impact on the resources that can be expressed (intra-scheduling).

Listing 6.1 – JSON representation of Slicing E2SM for “no” slice configuration, 5G/NR.

```
{
  "sliceConfig": {
    "dl": {
      "algorithm": "None",   Inter-scheduler
      "scheduler": "pf_dl"   Intra-scheduler
    },
    "ul": {
      "algorithm": "None",
      "scheduler": "pf_ul"
    }
  },
  "ueSliceConfig": {
  }
}
```

description. Finally, each UE is associated to a slice (state). In Figure 6.6, we map E2SM components to resources, processing, and state, as introduced for the base station and service descriptors in Section 4.2. The properties associated to the descriptors, such as forward-/backwards-compatibility (the E2SM is RAT-independent), resource isolation and sharing (through the use of slice algorithms), and customization and RAN extension (scheduling algorithms might be changed and/or deployed dynamically on the RAN), are ensured.

In the simplest case, as shown in Listing 6.1, no slice algorithm is configured. The “no slicing” inter-scheduler (“`None`”) calls a single intra-scheduler for all UEs on all free RBs; the intra-scheduler uses a PF scheduling algorithm (5G “`pf_dl`” in DL, UL correspondingly). (For the mapping of the inter-/intra-schedulers, compare to Figure 6.3.)

In Listings 6.2 and 6.3, the static and NVS algorithms are loaded for the inter-scheduler, respectively. In both cases, two slices (corresponding to two intra-schedulers) are created with slice algorithm-specific parameters (“`params`”). For static slicing, a number of RBs are statically configured, whereas for NVS, rate-based (slice ID 0) and capacity-based slices (slice ID 1) are configured. The static slicing example in Listing 6.2 further shows how UEs are associated to the individual slices, i.e., intra-schedulers.

6.3 QoS-Aware Slice Scheduling

The MAC scheduling framework’s multi-service extension explicitly distinguishes slice and user scheduling algorithms, and the corresponding interface enables a clear separation of concern. This allows to adjust slice and user scheduling algorithms separately.

The slice scheduling algorithm has to accommodate multiple slices while striking a balance among their heterogeneous objectives. We observed in Section 3.2.2 that previous slice algorithms do not consider resource slicing for three key types of resource requirements: physical (static) resources, throughput, and latency. In the following, we develop a multi-objective slice algorithm for these resource requirements. We integrate

Listing 6.2 – JSON for E2SM, static slicing.

```
{
  "sliceConfig": {
    "dl": {
      "algorithm": "Static",
      "slices": [{
          "id": 0,
          "label": "default",
          "scheduler": "pf_dl",
          "params_type": "static",
          "params": {
            "posLow": 0,
            "posHigh": 5
          }
        },{
          "id": 1,
          "label": "second",
          "scheduler": "pf_dl",
          "params_type": "static",
          "params": {
            "posLow": 6,
            "posHigh": 12
          }
        }
      ],
    },
    "ul": {
      "algorithm": "None"
    }
  },
  "ueSliceConfig": {
    "ues": [
      {
        "rnti": 1234,
        "dlId": 0
      },{
        "rnti": 5678,
        "dlId": 1
      },
    ]
  }
}
```

Listing 6.3 – JSON for E2SM, NVS slicing.

```
{
  "sliceConfig": {
    "dl": {
      "algorithm": "NVS",
      "slices": [{
          "id": 0,
          "label": "default",
          "scheduler": "pf_dl",
          "params_type": "nvs",
          "params": {
            "config_type": "capacity",
            "config": {
              "pctReserved": 0.5
            }
          }
        },{
          "id": 1,
          "label": "rate slice",
          "scheduler": "pf_dl",
          "params_type": "nvs",
          "params": {
            "config_type": "rate",
            "config": {
              "mbpsRequired": 9
              "mbpsReference": 18
            }
          }
        }
      ]
    },
    "ul": {
      "algorithm": "None"
    }
  },
  "ueSliceConfig": {
    "ues": [
      {
        "rnti": 1234,
        "dlId": 0
      },{
        "rnti": 5678,
        "dlId": 1
      },
    ]
  }
}
```

Inter-scheduler

Intra-scheduler 1

Intra-scheduler 2

the slice algorithm as an inter-scheduler, which demonstrates the extensibility of the MAC scheduling framework.

6.3.1 Addressed Challenges

The slice scheduling algorithm needs to share unused resources between slices while providing performance isolation and SLA enforcement. A number of challenges emerge during the procedure of scheduling multiple slices into a single base station. Specifically, we highlight these three major challenges:

1. **Requested resource isolation**: For instance, emergency services will request dedicated radio resources to ensure complete isolation from others.
2. **Efficient resource utilization**: Taking the eMBB service as an example, it aims to pursue efficient resource utilization with loose requirements on delay and isolation.
3. **Performance guarantee**: The URLLC service will request hard delay guarantees that need to be maintained with sub-optimal resource utilization efficiency.

6.3.2 QoS-Aware Slice Scheduling Algorithm

To better depict these challenges for the service embedding, we identify three corresponding slice resource requirement types, which will be elaborated in this section. We first identify three types of resource requirements, then formulate the corresponding utility functions, and finally propose a weight-based QoS-aware resource slicing framework for the slice embedding operation.

Three types of slice resource requirements

Three types of slice resource requirements are of interest in this context:

- A *fixed* slice requests dedicated radio resources along time and frequency domains. Therefore, it is isolated from other slices and does not multiplex its resources with others. One example is the BWP that operates on disjoint parts of the spectrum with a given numerology.
- A *dynamic* slice requests a share of resources in terms of aggregate throughput. It can be mapped to the eMBB usage scenario, and thus a precise radio resource allocation is less important than the efficient use of available resources for a guaranteed throughput.
- An *on-demand* slice exhibits more stringent requirements in terms of latency, and hence it can be mapped to the URLLC scenario. Therefore, such slice should be assigned radio resources with short delay, in comparison with the dynamic slice.

Based on these slice resource requirements, we propose respective utility functions from which we will introduce a novel slice scheduling algorithm with QoS awareness.

Utility functions for different types of slice

Corresponding to the three types of slice resource requirements, we form three slice sets grouping the slices with respective types of resource requirements: $\mathcal{K}_{\text{fix}}$, $\mathcal{K}_{\text{dyn}}$, and $\mathcal{K}_{\text{ond}}$. As for the k-th slice, its utility function is represented either as $\mathcal{U}_k$, $\mathcal{V}_k$, or $\mathcal{W}_k$, for $k \in \mathcal{K}_{\text{fix}}$, $k \in \mathcal{K}_{\text{dyn}}$ or $k \in \mathcal{K}_{\text{ond}}$, respectively. From the perspective of the operator, it aims to maximize the summation of utility functions from all instantiated slices, while still guaranteeing QoS requirements:

$$\mathcal{U} = \sum_{k \in \mathcal{K}_{\text{fix}}} \mathcal{U}_k + \sum_{k \in \mathcal{K}_{\text{dyn}}} \mathcal{V}_k + \sum_{k \in \mathcal{K}_{\text{ond}}} \mathcal{W}_k. \tag{6.1}$$

Fixed slices. For a fixed slice $k \in \mathcal{K}_{\text{fix}}$, its SLA stipulates that such slice requests P_k RBs in every scheduling interval, starting at RB P_k^{start}. To ensure that no overlapping between fixed slices occurs, extra admission control is required and will be elaborated later on. Then, the slice's utility function can be written as

$$\mathcal{U}_k = \begin{cases} 1 & \text{if RB range } \left[P_k^{\text{start}}, P_k^{\text{start}} + P_k\right) \text{ allocated} \\ 0 & \text{otherwise} \end{cases}, \tag{6.2}$$

i.e., such slice can only be satisfied when the exact requested resources are allocated, as required in Section 6.3.1.

Dynamic slices. We modify the formulation of bandwidth-based slices in [24] to enable concurrent slice scheduling (multiple slices scheduled within the same slot). The SLA of a dynamic slice $k \in \mathcal{K}_{\text{dyn}}$ requires a requested rate R_k as well as a prioritization rate R_k^{ref}, both in bps. The former describes the rate that the slice shall achieve in the long run. It is measured through the slice's achieved cumulative rate r_k in bps, and should fulfill $r_k \geq R_k$. The prioritization rate serves as a reference which this slice should achieve within the scheduled RBs to maintain its resource efficiency. It is checked through the average per-RB rate $\bar{r}_k$.

For a BS with N RBs, we control the attempted allocation rate R'_k which considers the resource efficiency via dividing the estimated allocation rate $\bar{R}_k = N \cdot \bar{r}_k$ by the prioritization rate R_k^{ref}:

$$R'_k = R_k \cdot \min\left(1, \frac{\bar{R}_k}{R_k^{\text{ref}}}\right) = R_k \cdot \min\left(1, \frac{N \cdot \bar{r}_k}{R_k^{\text{ref}}}\right) . \tag{6.3}$$

Note that if $\bar{R}_k$ is above the reference rate R_k^{ref}, the requested rate R_k is targeted. Otherwise, the requested rate is proportionally scaled down to get the attempted allocation rate R'_k. The rationale behind R'_k is to increase the resource efficiency, as tailored in the second challenge of Section 6.3.1.

Finally, to achieve proportional fairness among different dynamic slices, the utility function can be expressed as

$$\mathcal{V}_k(r_k) = \frac{R'_k}{N \cdot \bar{r}_k} \log(r_k) . \tag{6.4}$$

On-demand slices. The SLA of an on-demand slice $k \in \mathcal{K}_{\text{ond}}$ includes a delay-threshold Δ_k in milliseconds, a packet-loss probability δ_k, and a long-run maximum resource share $t_{k,\max}$ in terms of the ratio of RBs (i.e., $0 \leq t_{k,\max} \leq 1$). Beyond the aforementioned SLA, for each slice scheduling interval, a slice signals the number of bits B_k to be scheduled to satisfy its delay-threshold Δ_k. Based on the above parameters (i.e., B_k and $t_{k,max}$) as well as measured statistics (e.g., channel quality, user number), an on-demand slice can estimate its delay bound of the scheduling process.

To maintain the above SLA, the operator measures the achieved resource share t_k as a fraction of the N RBs, and the average per-RB transportation bit rate $\bar{b}_k$.

Finally, we write the utility function using a similar approach as for dynamic slices as:

$$\mathcal{W}_k(t_k) = \frac{B_k}{N \cdot \bar{b}_k} \log(t_k) = T_k \log(t_k), \tag{6.5}$$

in which $T_k = \frac{B_k}{N \cdot \bar{b}_k}$ represents the requested resource[3]. It is subject to $t_{k,\max}$ as elaborated in Section 6.3.2.

Since we follow the same formulation approach as for the dynamic slice, (6.4) and (6.5) show several similarities. Such observation is also found in [24], and the maximization over these two utility functions can be shown to be equivalent[4].

Proposed multi-slice scheduler with QoS-awareness

To utilize above utility functions, we propose a multi-slice scheduler comprising three steps:

1. Based on the SLA of fixed slices, we first allocate the dedicated resources.

2. Then, following the QoS requirements of on-demand slices, we allocate necessary resources.

3. Finally, we schedule resources for dynamic slices.

Nevertheless, as mentioned in Section 6.3.2, extra *admission control* is required prior to the creation of slices in order to ensure that the resource usage of all slices can be reasonably accommodated. Specifically, two constraints are applied. In the first constraint, we restrict that available N RBs can sustain the resource requirements from all slice instances. Thus, we write the ratio of requested resources from fixed slices as $\frac{P_k}{N}, \forall k \in \mathcal{K}_{\text{fix}}$, the allocated resource ratio for on-demand slice as $t_k, \forall k \in \mathcal{K}_{\text{ond}}$, and the allocated resource ratio for dynamic slices as $\frac{r_k}{R_k}, \forall k \in \mathcal{K}_{\text{dyn}}$. Assuming that on-demand slices use their agreed maximum resource share $t_k = t_{k,\max}$ and dynamic slices the

[3] Since the resource efficiency is not targeted by on-demand slices, the down-scaling operation of (6.3) is not applied here.

[4] Due to space reasons, we encourage the readers to refer to Lemma 1 of [24].

Algorithm 1 − Allocation of resource blocks for fixed slices.

```
Input  : SLA$_k = \{P_k^{\text{start}}, P_k\}, \forall k \in \mathcal{K}_{\text{fix}}$
Output : Allocation results for all fixed slices
forall $k \in \mathcal{K}_{fix}$ do
    Allocate $P_k$ RBs starting from $P_k^{\text{start}}$ to slice $k$
end
```

attempted rate $r_k = R'_k$, we have

$$\begin{aligned}&\sum_{k\in\mathcal{K}_{\text{fix}}} \frac{P_k}{N} + \sum_{k\in\mathcal{K}_{\text{ond}}} t_{k,\max} + \sum_{k\in\mathcal{K}_{\text{dyn}}} \frac{R'_k}{\bar{R}_k} \\ \leq &\sum_{k\in\mathcal{K}_{\text{fix}}} \frac{P_k}{N} + \sum_{k\in\mathcal{K}_{\text{ond}}} t_{k,\max} + \sum_{k\in\mathcal{K}_{\text{dyn}}} \frac{R_k}{R_k^{\text{ref}}} \leq 1 \end{aligned} \tag{6.6}$$

where the first inequality follows from (6.3). This gives a sufficient condition on the slices' SLAs. In the second constraint, admission control needs to ensure that no fixed slice is overlapping with other fixed slices. By denoting $[P_k^{\text{start}}, P_k^{\text{start}} + P_k)$ as the range of RBs allocated to the fixed slice $k \in \mathcal{K}_{\text{fix}}$, we have the following restriction:

$$[P_i^{\text{start}}, P_i^{\text{start}} + P_i) \cap [P_j^{\text{start}}, P_j^{\text{start}} + P_j) = \varnothing, \forall i \neq j\ . \tag{6.7}$$

After admission control, we follow the aforementioned three steps for the scheduling operation. Note that we adapt a weight-based approach in each step, i.e., the weight for the k-th slice is w_k, to maximize the marginal utility of different types of service requirements.

Fixed slices. As for the fixed slice, we directly follow the resource requirements within the SLA, as shown in Algorithm 1. Note that the weights for all fixed slices are set as $w_k \leftarrow \infty, \forall k \in \mathcal{K}_{\text{fix}}$, since they are guaranteed to be satisfied after applying (6.7) in admission control.

On-demand slices. Before scheduling resources for on-demand slices, we first derive the weights w'_k based on the marginal utility via a derivation approach:

$$w'_k = \frac{d}{dt_k}\mathcal{W}_k(t_k) = \frac{T_k}{t_k} = \frac{B_k}{b_k} \tag{6.8}$$

in which $b_k = N \cdot \bar{b}_k \cdot t_k$ is the on average transported bits during each scheduling interval for slice k. However, this weight ignores the QoS requirements between slices. Therefore, we apply a similar approach as the M-LWDF scheduler [131] and multiply this weight with the QoS-related coefficients, i.e., the delay threshold Δ_k, the packet loss probability δ_k, and the delay reported by the slice D_k. Finally, the weight for slice k can be represented as:

$$w_k = -\frac{\log \delta_k}{\Delta_k} \cdot D_k \cdot w'_k = -\frac{\log \delta_k}{\Delta_k} \cdot D_k \cdot \frac{B_k}{b_k}\ , \tag{6.9}$$

which will be used by our proposed algorithm to determine the scheduling ordering for on-demand slices. The average rate b_k is approximated through an exponential moving average b_k^{exp} at the scheduling interval j:

$$b_{k,j}^{\text{exp}} = (1-\beta) \cdot b_{k,j-1}^{\text{exp}} + \beta \cdot b_{k,j-1}^{\text{inst}} \;, \tag{6.10}$$

where $b_{k,j-1}^{\text{inst}}$ is the achieved rate at scheduling interval $j-1$, and β is a small positive constant.

Afterwards, we need to determine the amount of RBs to be allocated for each on-demand slice. A naive approach would be to limit each slice to its maximum share of resources of $t_{k,\max}$. However, this would penalize slices that did not request resources in the past, and we therefore resort to fulfill the maximum resource share over a moving time window. In this sense, more than $t_{k,\max}$ resources can be scheduled in a single scheduling interval, if few or zero resources were requested before. To better represent such condition, we collect the historic resource usage ratio for the k-th slice at the j-th scheduling interval as $t_{k,j}$, and denote the average resource usage ratio also at the j-th scheduling interval as $\bar{t}_{k,j}$ with the following relations:

$$\bar{t}_{k,j} = \frac{1}{\tau_k} \sum_{i=0}^{\tau_k - 1} t_{k,j} \approx \left(1 - \frac{1}{\tau_k}\right) \bar{t}_{k,j-1} + \frac{1}{\tau_k} t_{k,j} \;. \tag{6.11}$$

The average resource usage ratio is approximated through an exponential moving average, considering the historic resource share within previous τ_k scheduling intervals. We derive the maximally allowed resource share within the j-th scheduling interval $t_{k,j,\max}$ as:

$$\begin{aligned} \bar{t}_{k,j} &\leq \left(1 - \frac{1}{\tau_k}\right) \bar{t}_{k,j-1} + \frac{1}{\tau_k} t_{k,j,\max} = t_{k,\max} \\ \Rightarrow \quad t_{k,j,\max} &= \tau_k \cdot t_{k,\max} + (1 - \tau_k) \cdot \bar{t}_{k,j-1} \;. \end{aligned} \tag{6.12}$$

Finally, the assigned resources are limited by the maximally allowed resource share over time or the requested resources:

$$T'_{k,j} = \min(t_{k,j,\max}, T_k) \;. \tag{6.13}$$

In summary, as shown in Algorithm 2, on-demand slices are allocated in the order of their weight w_k from (6.9) with a maximum resource share $T'_{k,j}$ from (6.13) at the j-th scheduling interval. A slice is not scheduled if $B_k = 0$.

Dynamic slices. To efficiently schedule dynamic slices, we allocate the remaining RBs to the slice with the highest marginal utility among those that have data to send. Based on the utility function of the dynamic slice, we deduce the scheduling weights w_k using the derivative of the utility function. Note that the achieved rate r_k is the product of average RB rate $\bar{r}_k$, the number of RB N, and the resource share t_k of this slice:

$$\begin{aligned} w_k = \frac{d}{dt_k} \mathcal{V}_k(r_k) &= \frac{R'_k}{N \cdot \bar{r}_k} \cdot \frac{d}{dt_k} \log(N \cdot \bar{r}_k \cdot t_k) \\ &= \frac{R'_k}{N \cdot \bar{r}_k} \cdot \frac{N \cdot \bar{r}_k}{N \cdot \bar{r}_k \cdot t_k} = \frac{R'_k}{r_k} \;. \end{aligned} \tag{6.14}$$

Algorithm 2 − Allocation of resource blocks for on-demand slices.

```
Input  : $B_k, D_k, \mathrm{SLA}_k = \{t_{k,\max}, \Delta_k, \delta_k\}, b_k^{\exp}, \forall k \in \mathcal{K}_{\mathrm{fix}}$
Output : Allocation results for on-demand slices $\in \mathcal{K}_{\mathrm{fix}}$
forall $k \in \mathcal{K}_{ond}$ do
    Calculate $w_k$ as in (6.9)
    Calculate $T'_k$ as in (6.13)
end
$\mathcal{K}_{\mathrm{ond}}^{\mathrm{sorted}} \leftarrow$ sort $\mathcal{K}_{\mathrm{ond}}$ desc. by $w_k$, and skip if $B_k = 0$
forall $k \in \mathcal{K}_{ond}^{sorted}$ and while RBs available do
    Allocate round$(T'_k \cdot N)$ RBs to slice $k$
end
```

Algorithm 3 − Allocation of resource blocks for dynamic slices.

```
Input  : $\mathrm{SLA}_k = \{R_k, R_k^{\mathrm{ref}}\}, r_k^{\exp}, \forall k \in \mathcal{K}_{\mathrm{dyn}}$
Output : Allocation results for dynamic slices $\in \mathcal{K}_{\mathrm{dyn}}$
forall $k \in \mathcal{K}_{dyn}$ do
    Update $r_k^{\exp}$ as in (6.15)
    Calculate $w_k$ as in (6.14)
end
$k_{\max} \leftarrow \arg\max_k(w_k)$
Allocate all remaining RBs to slice $k_{\max}$
```

Finally, as shown in Algorithm 3, the dynamic slice with the highest weight w_k from (6.14) is allocated, where the achieved rate r_k in (6.6) can be approximated through an exponential moving average $r_{k,j}^{\exp}$ for slice k in (6.15):

$$r_{k,j}^{\exp} = (1-\beta) \cdot r_{k,j-1}^{\exp} + \beta \cdot r_{k,j-1}^{\mathrm{inst}} \, . \tag{6.15}$$

6.4 Evaluation

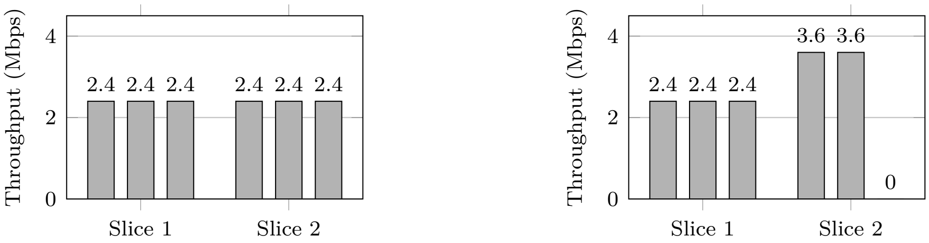
This section evaluates both the MAC scheduling framework and the QoS-aware slice scheduling algorithm. First, extensive simulations are performed in Section 6.4.1 to show the feasibility of the framework and algorithm. Second, an emulation campaign using OAI is performed in Section 6.4.2 to assess the performance and scalability using a fully compliant LTE protocol stack. Finally, we assess the NR scheduler implementation and the considered MAC scheduling framework using a radio setup with commercial-off-the-shelf (COTS) UEs in Section 6.4.3.

6.4.1 Simulation

We implemented the proposed scheduling algorithm ("SCN19") as well as NVS in a 3GPP LTE-aligned simulator, written in Matlab, with the system parameters as described in

Table 6.1 – Experimental parameters for simulation.

Parameter	Value
Access Technology	LTE
Bandwidth	5 MHz/25 RBs
Channel Model	Static (CQI 15) unless noted otherwise
No. of Users	2–6
Mobility	Static
Traffic	Full buffer, CBR, VBR
Slice schedulers	NVS, SCN19
User schedulers	RR, PF, BET



(a) Both slices have three users.

(b) One user in slice 2 disconnects, the slice increases per-user throughput while maintaining its aggregate throughput.

Figure 6.7 – Resource isolation between two dynamic slices.

3GPP TS 36.212.[5] The simulations are done for a bandwidth of 5MHz (25 RBs) in a single antenna FDD mode. We consider a set of slices with a varying number of users. The scheduling is performed according to the MAC scheduling framework, and the focus is on the DL direction with variable wide-band channel qualities allowing dynamic MCS per user and subframe. Also, traffic profiles are defined on a per-user basis. Table 6.1 summarizes the experimental parameters.

We first analyze the main slice requirements reusing the same methodology as in NVS [24], and assess the results for a 3GPP LTE system. Then, we evaluate scenarios in which more stringent QoS constraints and strict isolation are necessary.

Main Slice Requirements

Resource isolation. We first show the resource isolation capabilities. Consider two dynamic slices with the same SLA $R_k = 7.2\,\text{Mbps}$ and $R_k^{\text{ref}} = 14.4\,\text{Mbps}$. Each slice has three users with constant backlogged traffic. All users have equal throughput as shown in Figure 6.7a. After one of the users of slice 2 disconnects, we observe in Figure 6.7b that slice 2 maintains the aggregate throughput by redistributing its resources, without any impact on slice 1.

[5] The code is available at https://gitlab.eurecom.fr/schmidtr/mac-slice-sim upon request.

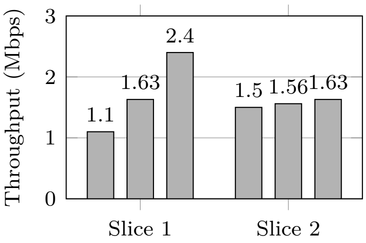


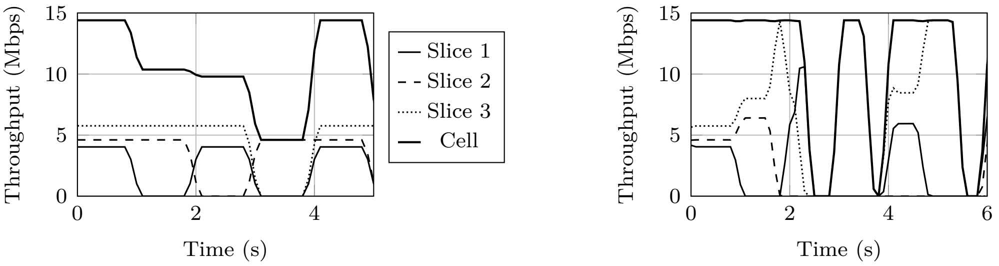
Figure 6.8 – Slice user scheduling customization: slice 1 operates with a PF scheduler, slice 2 applies a BET scheduler.

Customization. It should be possible to customize slices to achieve a differentiated treatment of user data flows. We perform the same experiment of two dynamic slices with different schedulers: Slice 1 uses a PF scheduler allocating resources proportionally to a user's current achievable throughput and its past average throughput. On the other hand, slice 2 employs a BET scheduler that strives for achieving the same average throughput, regardless of per-user channel quality. Both slices have three users with different channel qualities (MCSs 18, 23, and 28) and backlogged traffic. Figure 6.8 shows that slice 2 achieves a lower aggregate throughput compared to slice 1 (4.6 Mbps instead of 5.1 Mbps) since the PF algorithm weighs the instantaneous rate against the average throughput, thereby increasing cell resource utilization, while BET blindly forces equal throughput. This confirms the customization property of the proposed framework and its impact on per-user and per-slice performance.

Efficient resource utilization. One of the goals of resource slicing is an efficient resource utilization, and unused radio resources should be shared if allowed by the SLA. In this experiment, we consider two cases with (i) static slicing with three perfectly isolated slices and a reservation P_k of 28 %, 32 % and 40 % of bandwidth, against (ii) three dynamic slices that dynamically share their resources and a reservation R_k of 4.032 Mbps (corresponds to 20 %), 4.608 Mbps (32 %) and 5.76 Mbps (40 %). Each slice has one user that periodically downloads a large file. As can be seen in Figure 6.9a, resources are wasted in static slicing when not all users are active, as it does not share additional resources. In the dynamic case in Figure 6.9b, resources can be shared between slices. This leads to increased throughput for active slices, and improves cell usage. Another effect is that the cell becomes completely unloaded, implying energy saving potential.

Study of QoS-aware scheduling algorithm

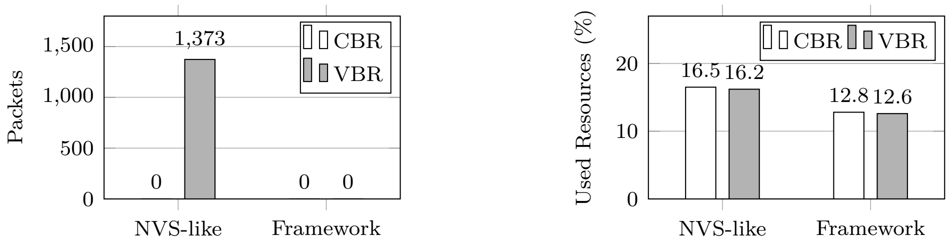
Faster Delay Response. Within the prior work of NVS, there exist two critical problems. First, NVS "penalizes" lightly loaded slices, typical for URLLC, by scaling them down (cf. (6.3)) since it can not distinguish between bad channel quality and lightly loaded slices. Second, a slice has no control over *when* it is assigned resources. Thus, no delay bound can be formed. To overcome these problems, we introduced the on-demand slice in the QoS-aware slice algorithm, for which resources are only allocated when needed.



(a) Static resource reservation.

(b) Dynamic resource reservation.

Figure 6.9 – Resource utilization for three slices.



(a) Number of lost packets.

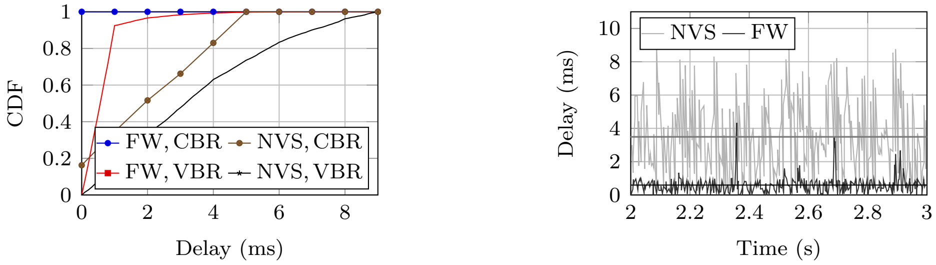
(b) Percentage of used resources over total system.

Figure 6.10 – Comparison of for dynamic/NVS-like and on-demand slices for 10 s of simulation.

We deploy three dynamic slices with each 4.0 Mbps requested throughput R_k (14.4 Mbps reference rate R_k^{ref}) and serving two users each with backlogged traffic. We compare the impact of the resource assignments with respect to QoS for (i) the original NVS slices (equivalent to our dynamic slice) to that of (ii) the on-demand slice. In case (i), we deploy an additional dynamic slice for QoS-sensitive traffic with a requested throughput R_k of 1.6 Mbps over a reference rate R_k^{ref} of 10 Mbps (corresponding to 16 %). In case (ii), we deploy an additional on-demand slice with a maximum resource share $t_{k,\max}$ of 16 %. The averaging window τ_k is set to 25 ms such that the slice can reuse previously unused resources.

Both additional slices serve four users with the same MCS 20. Each user requests packets of 255 B size and with 9 ms of allowed delay as for the 5G flow class 82 for delay-critical GBR traffic [7]. Packets arrive either in constant intervals of 10 ms and with 2 ms difference between the four users (denoted as CBR), or according to a Poisson Process with mean inter-arrival time 10 ms (corresponding to roughly 100 packets per user per second, denoted as VBR). We simulate 30 s.

Figure 6.10a shows that the NVS-like slicing loses many packets for VBR traffic, since packets arriving grouped cannot be completely scheduled in the next scheduling window. The on-demand slice of the proposed QoS-aware algorithm, on the contrary, does not drop any packets at all. From Figure 6.10b, we also observe that the on-demand slice uses less resources as it requests resources only when needed allowing efficient resource utilization. On the contrary, NVS slices are assigned a complete subframe for transmission which



(a) CDF for CBR and VBR traffic.

(b) Packet delays over 1 s simulated time for VBR traffic.

Figure 6.11 – Comparison of delay of non-discarded packets between the proposed Framework (FW) and an NVS-like scheduling scheme (NVS). Note the respective average delays, marked with horizontal lines.

might not always be possible to fill. This could also result in down-scaling of the NVS slice.

We report the packet scheduling delay as experienced by the users within a QoS-sensitive slice for non-dropped packets. Figure 6.11a shows the Cumulative Distribution Function (CDF) for the delay of packet transmissions. For CBR, our framework allows packet scheduling immediately after arrival, whereas NVS increases the delay to up to 5 ms. For VBR, we observe that our framework schedules packets reliably with 90 % of packets being scheduled before 1 ms and 99.9999 % before 8.0 ms[6] for the considered traffic scenario. NVS achieves the same percentile after 9 ms for VBR traffic by dropping packets. This is due to non-responsive behavior of NVS in which slices are scheduled purely with respect to their past throughput and not the actual queue state. Figure 6.11b reports the delay for scheduled packets over a time duration of 1 s. Note how spikes in delay for on-demand slices (due to high traffic intensity) seldom reach the average delay of the NVS approach.

Finally, we note that the above traffic patterns favor NVS, since lower traffic would automatically result in packet loss due to down-scaling or resource limitation, subject to SLA. On the contrary, for on-demand slices, low traffic and after inactivity, a higher averaging duration τ_k in (6.12) can enable a more aggressive resource allocation.

In summary, we remark that the on-demand slice provides deterministic behavior subject to SLA when scheduling packets and thus greatly improves the latency and reliability of user data flows. Since the frame structure of 5G is highly similar, we expect analogous results if on-demand slices are scheduled on a slot basis (i.e., no mini-slot scheduling is employed).

Reaction to channel quality changes. As shown in Section 6.4.1, individual dynamic slices are isolated from each other. Further, isolation of slices under changing channel qualities is ensured, as demonstrated in NVS [24].

[6] This duration includes all packets for the considered packet error rate, and the maximum data burst volume would exclude packet bursts that cannot be handled. See TS 23.501.

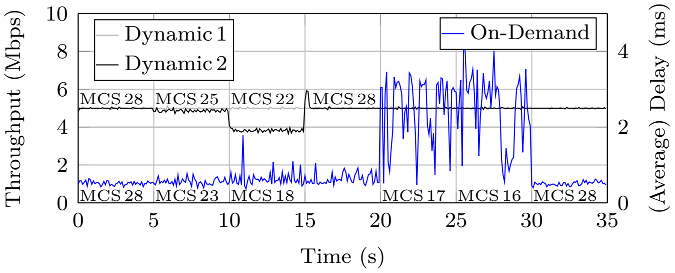


Figure 6.12 – Behavior of dynamic (rate) and on-demand slices (delay) if user channel quality changes. The delay is an average over 100 ms.

In the following, we investigate the impact of changing channel qualities on a mix of dynamic and on-demand slices. We consider two dynamic slices with each 5.0 Mbps requested throughput R_k over a 12.5 Mbps reference rate R_k^{ref}. They each have two users with a constant 2.5 Mbps downlink bit rate and an initial MCS 28. At 5 s, the channel quality of users of one slice drops to MCS 25; at 10 s, to MCS 22; and at 15 s, MCS returns to 28 (cf. Fig 6.12). Furthermore, there is an on-demand slice with a maximum resource reservation $t_{k,\max}$ of 20 %. To increase the effect of increased delay due to decreased channel quality, we set the slice's resource averaging window τ_k in (6.12) to 10 ms. This slice contains four users and a packet arrival identical to the Poisson Process in Section 6.4.1. The user channel quality, initially at MCS 28, drops to 23, 18, 17, and 16, at 5 s, 10 s, 15 s, and 25 s, respectively. It returns to MCS 28 at 30 s. The strong degradation of quality in the on-demand slice is motivated by the assumption that the slice still seeks to serve even bad users with high reliability requirements. The dynamic slice might be able to average out user quality drops over all its users, maintaining a high efficiency.

As shown in Figure 6.12, the dynamic slice with diminishing user channel quality initially experiences instability in the throughput before dropping due to low channel quality. Note how in both cases, an adaptation is almost immediate. When the user channel quality reestablishes, the throughput for this slice initially overshoots for two reasons. First, since the rate adaptation needs less radio resources for the same amount of data, free resources can be used to catch up the previously low throughput. Second, the moving average $r_{k,j}^{\exp}$ in (6.15) needs some time to adapt to the new, real throughput r_k, thus artificially boosting the weight of the dynamic slice in (6.14).

For the on-demand slice, Figure 6.12 plots average delays over 100 ms windows. Down to MCS 18, a drop in channel quality results in a stronger variation of channel delays; the lower the quality, the higher the variation. Since the lower channel quality results in a lower rate for transmission, packet transmissions for some packets tend to take longer themselves and queuing up subsequent transmissions. Note that the quality of the dynamic slices has no impact on the on-demand slice since the on-demand slice is scheduled preferentially. This can also be seen in Figure 6.13, which shows almost no difference in the CDF for the time windows 10-15 and 15-20 s. Starting at 20 s (Figure 6.12) for qualities lower than MCS 18, a strong increase in latency (and a loss

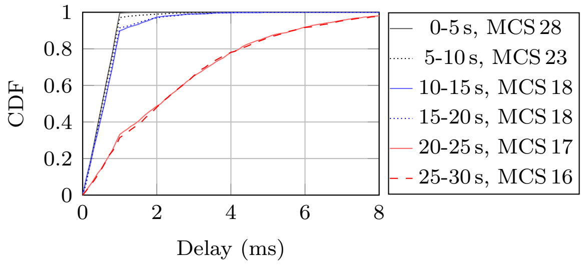


Figure 6.13 – Comparison of scheduling delay for changing user channel qualities.

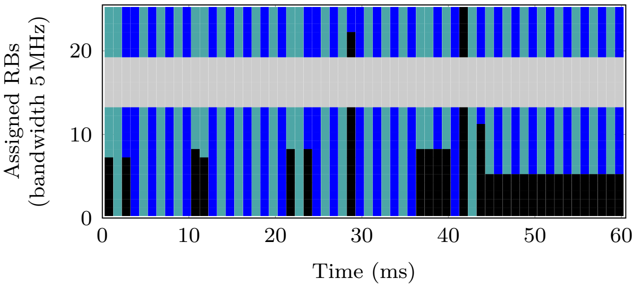
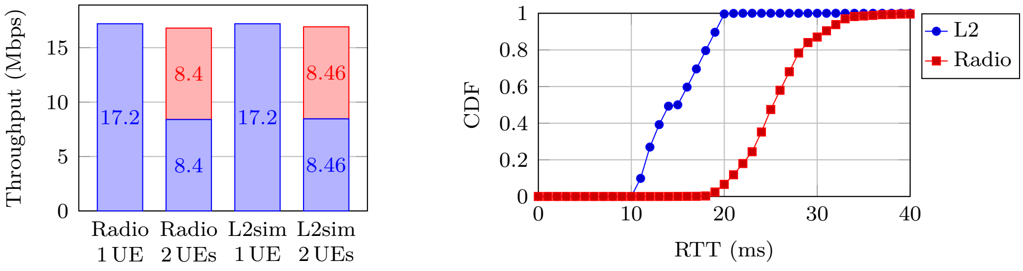


Figure 6.14 – RB allocation for multiple slices: (light gray) fixed slice, (blue and green) two dynamic slices, (black) on-demand slice.

of packets) is noticeable for a slight decrease in MCS. It can be supposed that a sweet spot is reached for the parameter settings (arrival rate, resource share $t_{k,\max}$, windowing duration τ_k) such that the slice can frequently not request more than its maximum resource share to guarantee a short delivery time. For a fixed arrival rate, the latency could be reduced by setting a higher resource share $t_{k,\max}$ for this slice. However, this would constantly block resources in admission control and might create instability if overbooking is employed. Hence, it would be advisable to increase the windowing duration τ_k such that the scheduler might be temporarily granted more resources.

In summary, we note that on-demand slices have comparative, low scheduling delays down to a minimum channel quality for a given traffic pattern and parameter setting. Below this minimum quality, they show behavior similar to NVS-like or fixed slices (cf. Figure 6.11a), since they are constantly restricted to their minimum resource share $t_{k,\max}$.

Behavior of on-demand slices. The on-demand slice is limited in the amount of RBs it can allocate to prevent that such a slice occupies all radio resources of the BS. In fact, a service owner could fabricate the parameter B_k to be very high which the inter-scheduler uses without further checks. Hence, we introduced the averaging parameter τ_k in (6.12) to limit an on-demand slice to its maximum allowed resource share $t_{k,\max}$, on average. This is shown in Figure 6.14 where the on-demand slice uses almost all resources (at 30 ms), followed by no activity. However, as soon as it constantly requests resources, it is limited to its resource share $t_{k,\max} = 0.2$ (at 45 ms).



(a) Maximum Throughput.

(b) Round-trip-time.

Figure 6.15 – Comparison of OAI radio deployment and "L2 simulator".

6.4.2 LTE Emulation

To verify the MAC scheduling framework implementation in OAI within a controlled environment without interference and for a possibly larger number of UEs, we use an emulation mode in the form of the "L2 simulator": in this mode, the eNB directly exchanges MAC-layer nFAPI [27] messages with an UE emulator, also based on OAI. Thus, no PHY layer is present, and no PHY channels (with associated noise, interference, or other models) are emulated. This mode is functionally equivalent to a radio deployment, as it runs in real-time, i.e., subframes are handled exactly every millisecond.

The used "L2 simulator" emulates an eNB with 5 MHz (25 RB) that has a maximum application throughput of 17,2 Mbps and a minimum application plane (*ping*) RTT of 20 ms. Figure 6.15a shows that both radio deployment and L2 simulator achieve the same application throughput for 1 or 2 UEs. Due to the missing PHY, the RTT is lowered by approximately 10 ms, as shown in Figure 6.15b, but otherwise follows a similar distribution (radio deployment RTTs have a heavier tail due to retransmissions, which do not happen without a physical radio interface). We claim that the L2 simulator (1) suffices to demonstrate the MAC framework, as there are no modifications on the MAC (or higher) layers with regard to a radio deployment, and (2) has the advantage of scalability, since many UEs can be emulated, which would be infeasible with full PHY channel emulation.

In the following, we present results for the MAC scheduling framework for dynamic slicing, scalability with respect to many slices and UEs, the execution time of the MAC framework, the customization of the intra-scheduler algorithm, and QoS-aware/low-latency slice scheduling. Note that while these results have been obtained using an emulator setup, the implementation can also be used in a radio deployment with custom-of-the-shelf UEs. Table 6.2 summarizes the experimental parameters used in this section.

Dynamic Slicing

Through the Slicing E2SM, a RAN controller can change the active slice algorithm in the MAC framework dynamically to deploy new services with dedicated slices, or in general to adapt the RAN's behavior. Such automation might happen at any time during runtime of the infrastructure.

Table 6.2 – Experimental parameters for 4G emulation.

Parameter	Value
Access Technology	LTE
Bandwidth	5 MHz/25 RBs
Channel Model	Static (CQI 15) unless noted otherwise
No. of Users	2–128
Mobility	Static
Traffic	Full buffer unless noted otherwise
Slice schedulers	Static, NVS [24], EDF [91], SCN19
User schedulers	RR, PF, MT

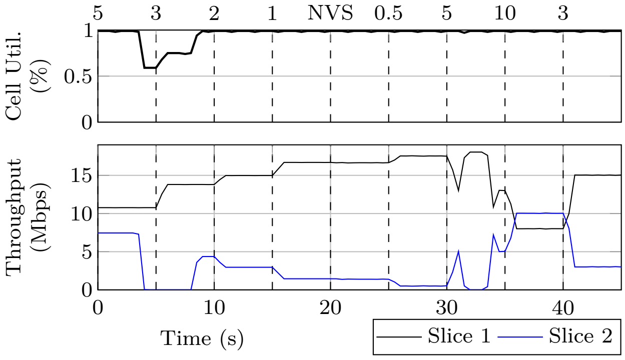


Figure 6.16 – Cell utilization and slice throughput during dynamic slice algorithm changes. Both share the same x-axis (time). At first, a static slice algorithm is set with decreasing amount of RBGs for slice 2, as indicated at the top. After a switch to NVS ($t = 20\,\mathrm{s}$), resources are shared automatically ($t = 33\,\mathrm{s}$), and precise slice throughputs can be set.

We compare static slicing and NVS. Static slicing incurs low scheduling delays and a strict isolation in frequency, making this scheme more adequate for public safety networks, interference scenarios, or constant traffic load. However, for fluctuating traffic demands, this means less sharing opportunities (unless a frequent re-configuration takes place), and only a couple of slices can be active due to the limited number of CCEs in LTE. Also, the minimum sliceable resource granularity is the resource block group (RBG). NVS performs slicing by tracking the amount of resources that each slice received through a weight, and schedules slices in a complete slot. Therefore, more slices that for static slicing can be present, and both rate-based and capacity-based slices with arbitrary precision are available. Also, the algorithm adapts to traffic fluctuations.

Consider the cell utilization and slice throughput over time in Figure 6.16. Starting at $t = 0$, two slices are present, and slice 2 has 5 RBGs as noted at the top of the graph. Note how slice 2 has no traffic at $t = 4\,\mathrm{s}$. Since static slicing has no notion of sharing resources, the cell remains underloaded. The resources of slice 2 are increasingly reduced by setting the number of RBGs down to 3, 2, and 1 RBGs; the resources of slice 1 are increased correspondingly. Note that the resource granularity is coarse.

By dynamically switching the slice algorithm, the MAC framework is able to address

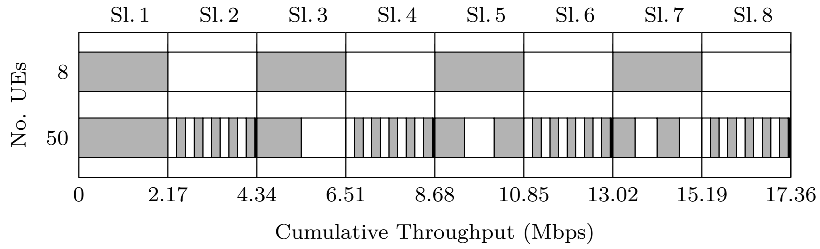


Figure 6.17 – Resource isolation and scalability for slicing. Above: Eight slices with eight users have the same throughput. Below: With additional users, the per-slice throughputs remain the same.

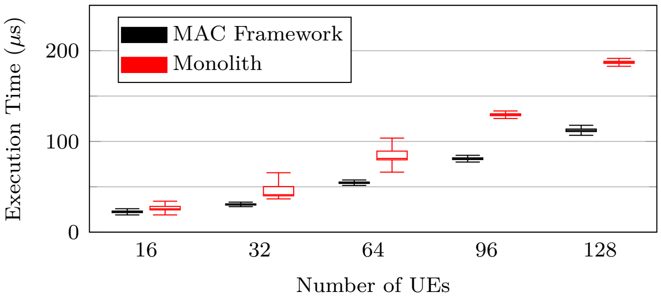


Figure 6.18 – Boxplots of the execution time of the MAC scheduling framework vs. the OAI “fairRR” scheduler for varying number of UEs.

the problems regarding sharing and resource granularity. At $t = 20\,\mathrm{s}$, the slice algorithm is changed to NVS without any impact on cell performance. At $t = 30\,\mathrm{s}$, the throughput of slice 2 is reconfigured to 5 Mbps, but uses the assigned resources only briefly before stopping transmission. The resources are shared with slice 1 automatically by NVS, leading to a multiplexing gain of 27,7 %. It is furthermore possible to set exact rates as indicated, e.g., 10 Mbps at $t = 35\,\mathrm{s}$, or 3 Mbps at $t = 40\,\mathrm{s}$.

Scalability of the MAC Framework

Handling multiple slices over a common infrastructure is increasingly important, while guaranteeing resource isolation. We configured eight slices with equal throughput (through NVS) and each one user and measured application throughput with *iperf* over 10 s, as shown in Figure 6.17. To assess both the isolation between the slices and the scalability for many slices, we connected additional UEs: slices 1, 3, 5, and 7 have 1, 2, 3, and 4 UEs, respectively, and all other slices have 10 UEs (50 UEs in total). Note how the throughput per slice does not change. This shows, that (a) isolation between slices is guaranteed, and (b) the MAC scheduling framework is scalable enough to handle eight concurrent slices with 50 UEs.

To better understand the limits of scalability, we compare the MAC scheduling framework execution time against the alternative, monolithic OAI 4G scheduler, known as the “fairRR” scheduler, as a baseline. For the MAC framework, we measure all phases,

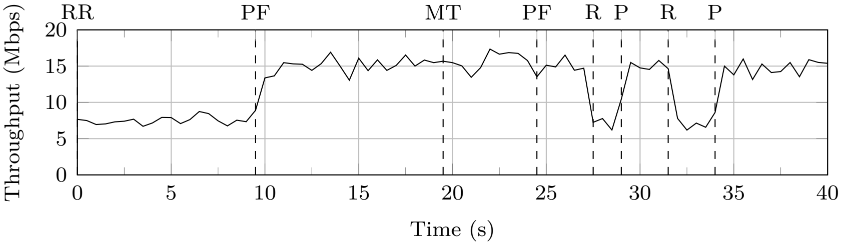


Figure 6.19 – Cell throughput over time and switching schedulers dynamically (RR, R: round-robin; PF, P: proportional fair; MT: maximum throughput).

i.e., preprocessor with intra- and inter-scheduling, and postprocessor to have a direct comparison to the baseline. Since the inter scheduler typically partitions resources in a fixed manner (Static) or according to a weight (NVS) with little associated computational complexity, we assume a limited overhead for the inter scheduler. However, the intra slice schedulers scale with the number of users, frequency resources, and possibly space such as beams.

Figure 6.18 shows the scheduling execution time as a boxplot, measured for every subframe over 30 s. For the MAC framework, we use one slice and the default RR scheduler; the monolithic scheduler uses RR, too. The MAC framework shows an approximately linear overhead with the number of users. This indicates that running multiple intra-schedulers sequentially will incur an additional execution time that is linear to the number of users (roughly 1 μs/UE), plus a small fixed overhead. On the other hand, the monolithic scheduler's execution complexity not only seems to be sup-linear, but the execution time also shows a higher variance, making it impractical for sequential execution. This is surprising, as the monolithic scheduler is implemented using multiple threads. Since the MAC framework has no limitation on the number of UEs (apart from OAI limitations) and is not multi-threaded yet, we see potential to increase the number of UEs while lowering the execution time, scheduling up to 1000 UEs.

Intra-scheduler Processing Customization

The scheduling algorithm of a slice can be changed arbitrarily through the E2SM. Since the scheduling state such as UE information (CQI, etc.) is kept outside the intra scheduler, the intra scheduler can be changed on-the-fly by service owners to adapt to user behavior. We consider one slice with eight users with fast fading, and a full buffer traffic model. Note that in this evaluation, we do not attempt an exact assessment of the scheduling schemes already present in the literature, but rather highlight the extension capabilities of the MAC framework. For simplicity, we use one intra scheduler, but multiple ones might have been used in conjunction with a slice algorithm in the inter scheduler.

Figure 6.19 shows the MAC throughput over time when changing scheduling schemes dynamically. In the beginning, due to the channel-unawareness of RR scheduling, the cell throughput is low, but all UEs get a fair, equal amount of resources. When changing to PF scheduling, the cell throughput doubles, since PF scheduling opportunistically

Table 6.3 – Slice parameters and UE association.

Algorithm	UE 1	UE 2	UE 3
NVS [24]	rate slice $r^{\text{rsv}} = 16.7\,\text{Mbps}$ $r^{\text{rsv}} = 17.5\,\text{Mbps}$	rate slice $r^{\text{rsv}} = 0.8\,\text{Mbps}$ $r^{\text{rsv}} = 17.5\,\text{Mbps}$	*in Slice of UE 2*
SCN19 Section 6.3	dynamic slice $r^{\text{rsv}} = 16.7\,\text{Mbps}$ $r^{\text{rsv}} = 17.5\,\text{Mbps}$	dynamic slice $r^{\text{rsv}} = 0.6599\,\text{Mbps}$ $r^{\text{rsv}} = 17.5\,\text{Mbps}$	on-demand slice $t_{\text{max}} = 0.008$ $\tau = 100\,\text{ms}$ (other params: 1)
EDF [91]	$D = 1000\,\text{ms}$ $N = 23857$	$D = 200\,\text{ms}$ $N = 188$	$D = 20\,\text{ms}$ $N = 4$

schedules high-CQI UEs. At the time of change, no performance degradation is visible, since the proposed design and implementation is able to switch the scheduler atomically between subframes. Switching to maximum throughput yields only marginal gains, with no fairness being ensured. Note the dynamicity of the scheduling scheme switches, starting around 25 s in Figure 6.19. The service owner can arbitrarily change the schedulers without any impact on service continuity.

QoS-aware/Low-Latency Slice Scheduling

NVS allows to translate the spectrum into a notion of rates and can effectively share the resources, but its design can lead to larger scheduling delays and resource usage, since a low-rate slice might be scheduled only rarely and uses the whole subframe. Static slicing guarantees low latency by scheduling every slice in every subframe, at the expense of sharing possibilities and a greatly reduced number of slices (due to a limited number of CCEs).

In Section 6.3, we designed a slice algorithm ("SCN19") that extends NVS with delay-bound slices while enforcing a maximum aggregate resource share over a time window. This allows delay-bound slices to temporarily use more resources when needed, effectively sharing resources in favor of low-latency slices, but resources are lost if not used. The slice scheduling algorithm EDF [91] goes one step further by defining a deadline for *all* slices during which they need to be guaranteed a certain number of RBs.

We compare the NVS, SCN19 and EDF algorithms with respect to the incurred scheduling delays. We consider two slices: one slice with a high resource share for an eMBB service, and a "small" slice for mMTC service, each having one UE. A third UE in the second, "small" slice is considered a low-latency user. As has been mentioned, the NVS algorithm will incur some scheduling delays to this user. For SCN19 and EDF algorithms, we create a third slice for UE 3 to ensure low scheduling delays. All slice parameters and UE associations are shown in Table 6.3. UEs 1 and 2 receive a TCP stream in DL. For UE 3, measure the RTT using *ping* packets with size 64 B sent every 250 ms.

Figure 6.20 shows the CDFs of RTT for UE 3 when using the three scheduling

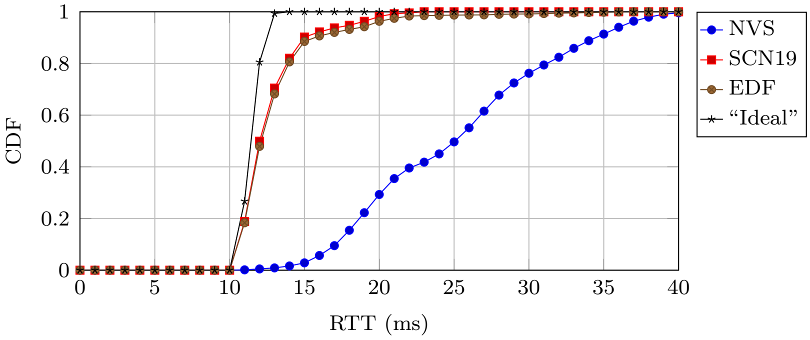


Figure 6.20 – Comparison of the slice algorithms NVS, SCN19, and EDF w.r.t. measured *ping* RTT for the UE 3. SCN19 and EDF keep scheduling delays lower than NVS.

algorithms. NVS incurs scheduling delays that stem from the delay between the time of arrival of a packet and the allocation of resources to the corresponding slice, negatively impacting the RTT. In comparison, both SCN19 and EDF guarantee short scheduling delays, and a correspondingly low RTT. Due to immediate scheduling of the slice after the packet arrives at the base station, their CDFs are almost identical. Furthermore, Figure 6.20 includes the "ideal" RTT in a completely unloaded system. The improvements with respect to SCN19/EDF stem from the fact that no other packets are processed in the system, such that other possible delay sources, e.g., UL scheduling and IP packet processing, are excluded. In summary, on the application level, using a QoS-aware/low-latency algorithm means a decrease of the average packet RTT from 25 ms (NVS) to 12 ms (SCN19, EDF), a decrease by 50 %.

Note that SCN19 uses less resources than NVS, since only a part of a subframe is used when scheduling a slice, and we refer the reader back to Section 6.4.1 for a discussion. Thus, SCN19 strikes a balance between the multiplexing needs of the infrastructure provider and the latency-sensitive service requirements of service owners. Note that EDF only schedules users when they have data and only the necessary RBs, similar to SCN19. Therefore, similar results would be obtained for EDF.

In the context of the "Network Store" described in Section 5.6, we designed a `burst_analysis` application that analyzes the traffic dynamically and slices the RAN to ensure low scheduling latencies for UEs matching specific traffic pattern. More specifically, the application creates on-demand slices for UEs matching intermittent traffic (created by *ping*), resulting in a strong decrease of RTT shown above. A video of the demo is available online[7]

6.4.3 NR Radio Deployment

Prior to this work, there was no multi-user scheduler in OAI 5G, as already mentioned in Section 6.2.4. Therefore, we implemented a new multi-user scheduler corresponding to the proposed framework, including slicing functionality. In the following, we present

[7] See `https://youtu.be/KxTEpFe5dJU`.

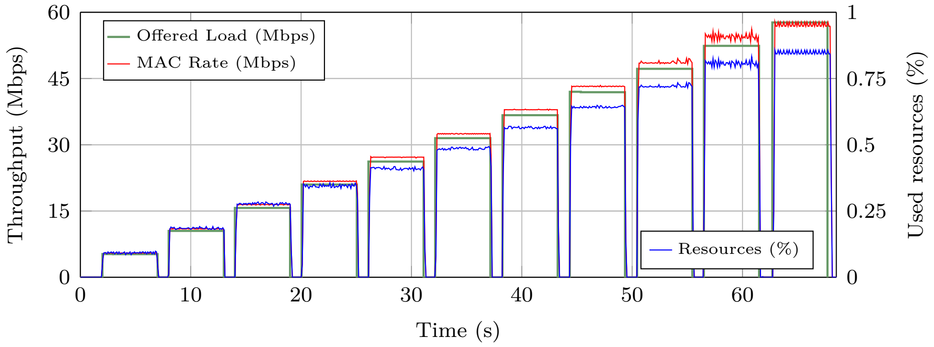


Figure 6.21 – 5G DL MAC throughput and used resources depend on offered load.

the results for scheduling without slicing over a radio connection, and complement them with additional slicing results. Note that since the underlying framework is the same, the results for slicing, obtained via simulation and the 4G emulation mode, are valid for 5G (e.g., configurability, scalability, etc.), while the radio measurements prove the slicing properties over an NR radio connection.

For the measurements, we use OAI tag 2021.w20 in non-standalone mode (NSA, i.e., the radio CP is via an eNB, whereas the UP is over the gNB). Both the eNB and gNB use an Ettus B210 SDR. The eNB uses a 25 RB (5 MHz) carrier in band 7 (which is of minor interest, given that we only evaluate UP performance indicators from the gNB). The gNB uses a 106 RB (40 MHz at subcarrier spacing of 30 kHZ) NR carrier in band 78 (TDD with 5 ms period length, 7 DL slots, 2 UL slots, 1 mixed). The MCS is fixed to 20 for all UEs in all experiments, such that the maximum throughput of the gNB achieves 60 Mbps. Depending on the experiment, we use two Google Pixel 5, one Oppo Reno 5 5G, and one Quectel RM500Q-GL module.

Multi-User Scheduler

We assess the behavior of the scheduler with respect to throughput. First, in Figure 6.21, we plot the MAC layer throughput and percentage of used resources (in DL) depending on an increasing offered load using *iperf*. As expected, the scheduler uses only the resources necessary for delivering the application data of *iperf* with a slightly higher MAC throughput due to overhead such as MAC, RLC, and PDCP headers[8]. The gNB's radio resources are saturated when the offered application throughput is 57.7 Mbps, since the MAC throughput is capped at the application data rate (but should be higher if all packets were delivered, leading to dropped packets). The gNB does not use all resources: there are 6 slots[9] and an additional mixed slot, but the current MAC implementation only uses PUCCH Format 0 in 3 UL slots, while the PHY layer also supports PUCCH Format 2 with more acknowledgment bits. In total, only 6 PUCCH occasions for 7 slots are available in every TDD period, limiting the resource usage to $\frac{6}{7} = 85.7\,\%$.

[8] There is no SDAP layer, as the gNB is connected to an EPC.

[9] The TDD pattern foresees 7 DL slots, but due to PHY layer problems, we only use 6.

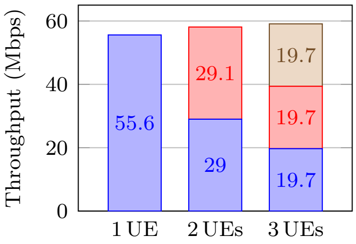


Figure 6.22 – 5G DL throughput fairness for multiple UEs.

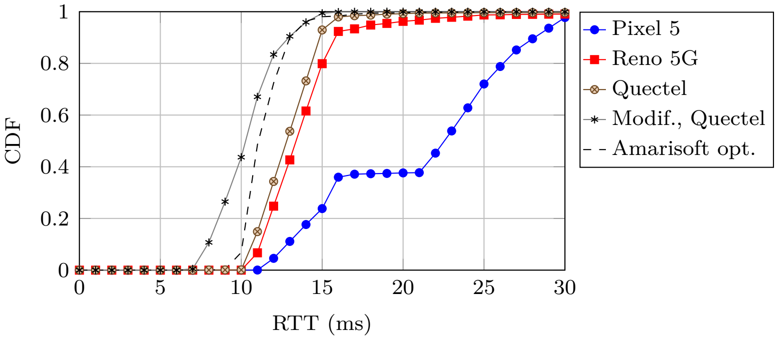
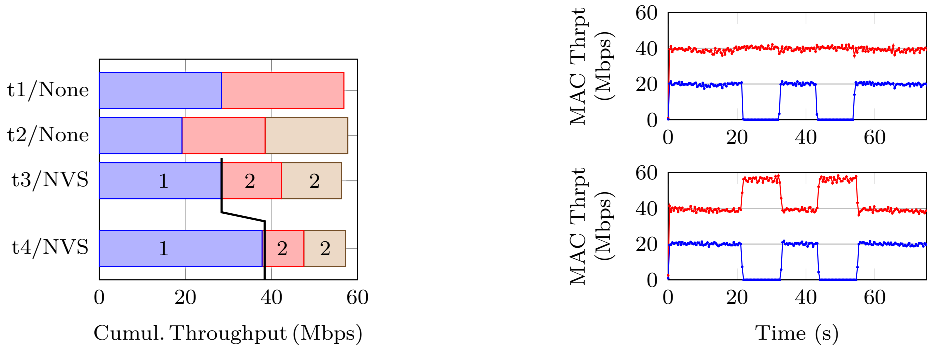


Figure 6.23 – RTT of *ping* packets for different UEs.

Figure 6.22 plots the throughput for an increasing number of UEs. We observe that the application throughput is equally shared for multiple UEs, which results from the equal MCS and PF scheduler, equalizing UE throughput under the same channel qualities. Further, with increasing number of UEs, the cell throughput slightly increases. Again, this is due to only using PUCCH Format 0 with 6 HARQ feedback occasions per UE for 7 slots. Thus, when the PUCCH resources of one UE are completely used, another one can still be scheduled. Note that currently, adaptive modulation and coding has not been implemented within the 5G scheduler, and therefore we can not further compare the fairness of the scheduler at different MCSs.

Since one of the main goals of NR is to achieve low latency, we evaluate the RTT from the CN to the UE via the NR RAN and back using *ping* packets with size 64 B sent every 200 ms. Figure 6.23 plots the CDF of the RTT for different UE models. We observe that the RTT can be as low as 10 ms. However, we measure higher RTTs of up to 30 ms for the Google Pixel 5 phone. More precisely, 40 % of packets travel for 15 ms or less, and 60 % of packets for more than 20 ms. This gap of 5 ms corresponds to roughly one TDD period. However, comparing to the Oppo phone, the difference is surprising, as both use the same Snapdragon 765G 5G chipset, which should exhibit the same behavior. For the Quectel and Oppo UEs, on the other hand, almost all packets arrive after a maximum of 15 ms, i.e., below the minimum of an LTE RAN.

The NR cell is configured with SR opportunities every 5 ms. Thus, if a packet arrives too late, it has to wait up to 5 ms before it can request UL resources again. To avoid this effect, we configured the gNB to schedule the UE in *every* UL slot, and measured the



(a) Isolation (UEs are marked corresponding slice). **(b)** Static resource attribution vs. resource sharing.

Figure 6.24 – Resource slicing isolates services, but resources might be shared for efficiency.

RTT for the Quectel model again ("Modif., Quectel"). In this case, the RTT is further lowered, with almost half of the packets arriving back at the CN after 10 ms.

We compare the measured RTTs to a commercial RAN software, the Amarisoft LTE50 in version 2021-03-15. We deploy a single gNB in standalone mode with a 106 RB (40 MHz at subcarrier spacing of 30 kHZ) NR carrier in band 78 (TDD), and we apply the manufacturer-proposed configuration for low RTT, i.e., a TDD 2.5 ms period length with 2 DL slots, 2 UL slots and 1 mixed, and an SR period of 1 ms. We connect the same Quectel as before. The results indicate that the OAI 5G scheduler allows comparable RTT performance to the commercial base station. Further, the optimized OAI version enables even a lower RTT.

Nevertheless, the RTT can be lowered. The used Ettus B210 SDR has a negative impact on the RTT. It is scheduled 3 ms in advance to give enough time to process the samples before sending/after receiving, adding inherent latency to RAN processing. This could be reduced by lowering the advance, e.g., by using another SDR. Further, the chosen TDD period of 5 ms with two UL slots increases the RTT, as the UE can only send data every 5 ms, increasing waiting times, and the 2,5 ms period length as for Amarisoft might be used to further decrease the RTT.

Slicing

We demonstrate basic slicing properties, such as isolation and sharing, for NR. We omit additional results for dynamic configurability, customization and scalability due to space reasons and/or since they are hard to reproduce in a physical system (e.g., large number of UEs).

In a first experiment, we demonstrate the isolation property of slicing (Figure 6.24a). The objective is to ensure that the UE in slice 1 (blue) gets 50 % of the resources, or around 30 Mbps. At time instance 1, no slicing is active; however, since only two UEs are present, they equally share the resources. Upon a new connection of a UE, at time instance 2, the resources are equally shared between all three UEs, violating the requirement that the blue UE receives 50% of resources. Therefore, at time instance 3, we deploy two slices with equal resource share, and associate the blue UE to slice 1 and the others to slice 2, which ensures half of the resources for the blue UE. Similarly, at time 4, the user plane is reconfigured to give 66 % resources to slice 1; correspondingly, the blue UE receives 66 % of throughput.

A fixed allocation of resources can lead to underutilization of the already scarce radio resources in the RAN. Therefore, it is advantageous to flexibly distribute or share resources between slices if UEs do not consume the resources reserved for their slice. In the second experiment, we connect two UEs, associated with two slices with 66 % (red) and 34 % (blue) of resources. In the upper graph of Figure 6.24a, we configured slices to not share any resources. This leads to wasted resource as the red slice does not use the additional resources when the blue slice is inactive. On the other hand, if resource sharing is permitted, the red slice increases throughput by 50 %, which shows how the NVS algorithm exploits the idea of offering isolation while efficiently using the radio spectrum, applied to a 5G RAN.

6.5 Discussion and Future Work

In this chapter, we presented the MAC scheduling framework that allows to adapt the behavior towards deployment options, e.g., for frequency ranges (sub-6 GHz, mmWave). We further devised an extension for multi-service operation, including a slicing E2SM, and designed a QoS-aware slicing algorithm. Using the slicing capabilities of the MAC framework and the S1-flex feature for connecting multiple CNs to the same RAN, it is possible to build an end-to-end slicing solution, as we have shown in a demo at the virtual 2020 NOMS conference[10]. In the following, we discuss the MAC scheduling framework and how it may be extended and enable future work:

First, we implemented OAI's 5G/NR scheduler with slicing capabilities from scratch, and a number of improvements are necessary and features missing for a fully functional multi-UE scheduler: (1) There is no adaptive modulation and coding in the scheduler, and the DL/UL MCS is fixed. Dynamic rate adaptation, e.g., depending on HARQ feedback and reference signals in DL, and measurements and UE power headroom reports in UL, are important to dynamically adapt the MCS. The fairness of the PF DL/UL implementation could be further evaluated. (2) As has been discussed, currently only PUCCH Format 0 is used in the MAC, and a multiplexing using Format 2 is necessary to allocate all slots to a single UE. Also, the first DL slot 0 is not used yet, limiting throughput, and which would also require more acknowledgment bits. (3) In a mid- to long-term perspective, support for multiple BWPs is necessary to support high bandwidths while saving power when the phone is idle [132], and to adapt the numerology according to the service type.

Second, slicing is an ongoing trend, and new slice algorithms are frequently presented in the literature. The presented MAC slicing framework allows the research community to implement and evaluate new algorithms for both slicing or user scheduling over OAI for 4G/LTE and 5G/NR, which was more difficult to add previously. Thus, more slicing

[10] A presentation is available online, see `https://youtu.be/zBxQPqGN_po`.

algorithms apart from the implemented four could be added (together with extending the E2SM). Similarly, additional QoS-aware user scheduling algorithms could be implemented as well. Further, it is feasible to reimplement the Orion [53] architecture (of which the code has not been published) to functionally isolate slices from each other, which might be combined with the work in Chapter 7.

Third, the scheduler opens a number of research challenges that can be explored due to its CP-UP decoupling. A separate preprocessor for mmWave operation can be developed: Assuming that transmissions are directed to UEs using analog beamforming, UEs have to multiplexed, within a slot, in *time* rather than *frequency* (the default mode in LTE and NR for sub-6 GHz operation), and the scheduler has to be adapted to use the time-domain allocation tables defined in RRC. Further, it needs to take beamforming feedback information into consideration. A more experimental research direction, even at sub-6 GHz operation, would be the support of dynamic TDD. To this end, HARQ feedback allocations are fully flexible in OAI's 5G/NR implementation[11], such that an experimental preprocessor for dynamic TDD can dynamically allocate the feedback according to the slot directions for UL and DL transmissions.

6.6 Conclusion

In this chapter, we presented a modular MAC scheduling framework that can be adapted to different deployment scenarios and service requirements. This framework allows to modularly change a preprocessor for deployment adaptation, and is further divided into an inter-scheduler and an intra-scheduler, separating slice and user scheduling. An abstraction in the form of an E2SM allows an E2-compatible controller, e.g., based on FlexRIC, to configure the slice algorithm and slices in the RAN. We further proposed and integrated a QoS-aware slice algorithm as an inter-scheduler into the MAC scheduling framework. Finally, we provided extensive simulation, emulation over 4G, and real radio measurements over 5G systems. To this end, the 5G/NR scheduler was implemented from scratch, providing a multi-UE, multi-service scheduler for the OAI project. We showed that properties such as resource isolation, multiplexing of resources, or customization and extension are fulfilled by the framework, and that the QoS-aware slice scheduling algorithm allows to concurrently schedule services with different resource objectives. The framework allows to implement scalable schedulers with many slices. Finally, this work paves the road to perform research on advanced 5G scheduling schemes, and we outlined the corresponding future work directions.

[11] The HARQ feedback in OAI 4G/LTE follows a static pattern.

Chapter 7

SD-RAN Virtualization

7.1 Introduction

The increasing RAN densification leads to growing CAPEX and OPEX for cellular networks, as more BSs need to be deployed. RAN *sharing* allows reducing costs, since multiple operators share a common infrastructure instead of maintaining dedicated ones. For instance, O-RAN assumes that a sharing operator might host the VNFs for DU and CU of one or more visiting operator(s), and enable remote access ("remote" E2) to control those VNFs [124]. While this allows to share the computing infrastructure, there is no potential for sharing the radio resources through a common DU. On the other hand, since the operators are competitors, the sharing operator may not grant full access to prevent malicious access or erroneous configurations.

Sharing the DU between operators enables full potential for multiplexing gains in the radio spectrum. If both operators are furthermore enabled to additionally and individually control their (virtual) network via their respective SD-RAN controllers, then this corresponds to the service-oriented RAN. A related concept is Orion [53], where a hypervisor directly on top of the base station multiplexes the scheduling decisions of multiple service-specific controllers onto a shared RAN. However, the Orion concept does not work in a disaggregated infrastructure, as multiple hypervisors would be necessary, and the hypervisor has no global network view, which could lead to conflicts between base stations. In "wired" SDN, FlowVisor [41] implements a proxy-like virtualization platform with global network view that slices the control of the infrastructure and exposes each slice to individual SDN controllers.

While the concept of a "proxy" is applicable to SD-RAN, it has not been realized, and developing the corresponding virtualization reveals a number of challenges. It needs to isolate the different controllers from each other, such that a control decision from one controller does not impact the virtual network of another controller. To this end, it is of paramount importance to specify the protocol such that conflicts are *avoided* by design. Furthermore, where an SDN controller such as FlowVisor only considers packet flows, an SD-RAN controller has to additionally address radio resource and radio control levels (mobility, interference). The entity that enforces the virtualization (we will call it the "virtualization layer" from now) should be capable of operating over a disaggregated

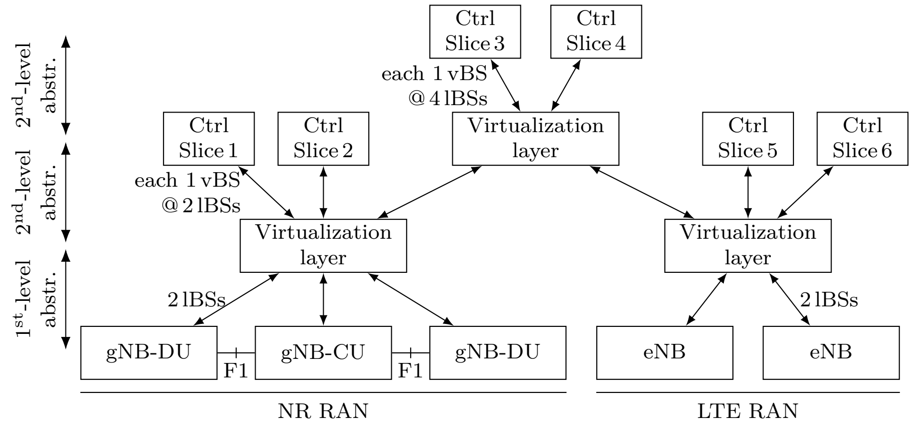


Figure 7.1 – Control of the RAN can be shared between multiple controllers through a virtualization layer, which can also abstract topologies and build hierarchies of controllers.

RAN infrastructure to be able avoiding conflicts between the RAN network functions and across base stations. Finally, the virtualization should work for multiple RATs for forward and backward-compatibility, following the design principles of 5G.

In this chapter, we propose an SD-RAN virtualization layer to enable multiple controllers of different service owners, e.g., operators, to concurrently and conflict-free program the shared infrastructure. The chapter is organized as follows. In Section 7.2, we introduce the concept of an SD-RAN virtualization layer to enable multiple controllers of different service owners to concurrently program their slice in the shared infrastructure. In Section 7.3, we provide the design of the virtualization layer and detail the virtualization of the control on the example of the NVS [24] slice algorithm. In Section 7.4, we present results from a prototype implementation that leverages the FlexRIC SDK. Finally, Section 7.5 discusses future research possibilities.

7.2 Concept

The proposed SD-RAN virtualization layer is shown in Figure 7.1. It is placed between the RAN infrastructure and (multiple) SD-RAN controllers. It partitions the infrastructure and *virtualizes* the SD-RAN control, such that each controller can only control the associated virtual network/slice. To (northbound) SD-RAN controllers, it appears like the RAN infrastructure; to the (southbound) infrastructure, it appears like an SD-RAN controller.

The virtualization layer adopts the ideas of the first- and second-level abstraction from Chapter 4 to construct an lBS and embed the vBSs of the respective operators. First, it re-aggregates a RAN infrastructure into lBSs. For instance, Figure 7.1 shows two gNB-DUs and one gNB-CU (and two eNBs). The virtualization layer re-aggregates the DUs and CU into two lBSs, *merging* their resources, processing, and state. This simplifies the infrastructure towards the northbound controllers, which do not have to handle separate DUs and CUs, and the virtualization layer abstracts the network topology as

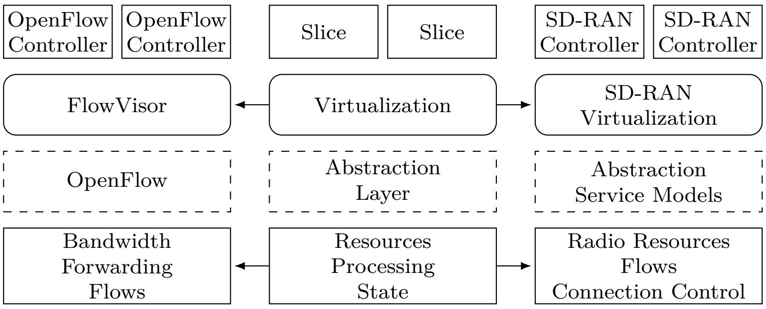


Figure 7.2 – The SD-RAN virtualization uses RAN level abstractions to enable multiple SD-RAN controllers to concurrently control the same infrastructure, similar to how FlowVisor [41] implements a virtualization for OpenFlow-based switch networks. [41]

foreseen by the first-level abstraction.

The virtualization layer further implements the second-level abstraction when virtualizing the control and exposing it to pre-configured northbound controllers. Thus, it *splits* the resources, processing, and state of each lBS into vBSs (one per controller). Since the sliceable resources within a base station are highly heterogeneous, comprising flow control for multiple bearers, UE radio connection control, and radio resources, the way this splitting happens depends on the considered level: on the radio resource level, a certain resource share of the total base station resources would be exposed; for the radio control level, the virtualization layer would partition the UEs into groups, and each group is exposed to the respective controller. Thus, the northbound controllers are not aware that they only operate on a slice in the underlying infrastructure.

Figure 7.2 shows the concept of such virtualization, and compares it to FlowVisor [41]. FlowVisor virtualizes the control of a network of switches to share it between multiple controllers. It uses OpenFlow as an abstraction layer of the actual resources, partitions it (by means of “FlowSpaces”, see the paper), and allows multiple controllers to control the infrastructure while isolating them. Similarly, the proposed SD-RAN virtualization uses service models that abstract the resources, state, and processing of the RAN infrastructure for the three RAN levels, and exposes a virtualized view towards the SD-RAN controllers. In the terminology of the RAN Engine, the virtualization layer implements a micro-SDK for concurrent control of the RAN control endpoints in the infrastructure. Therefore, each SD-RAN controller individually controls its virtual network, and can customize and control it towards the requirements of the services.

Since the SD-RAN virtualization layer, by definition, exposes the same interface at the northbound interface as at the southbound, it can be *recursively* stacked on top of each other, as also shown in Figure 7.1. It is thus possible to layer SD-RAN virtualization layers. This might for example be used to realize “resellers” of (RAN) infrastructure who not only guarantee a certain amount of resources in the network, but also provide (indirect) access to the infrastructure for monitoring and control. Similarly, a higher-level virtualization layer can virtualize the control of multiple networks (e.g., the NR and LTE RANs as shown in Figure 7.1). It is then feasible to have controllers on a lower level handle performance-critical tasks, e.g., remote scheduling; going higher, controllers gain

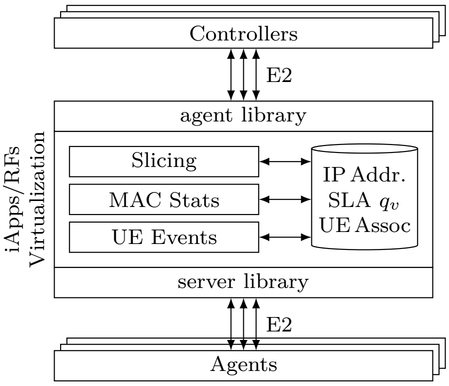


Figure 7.3 – Architecture of the SD-RAN virtualization layer, which is a FlexRIC controller specialization.

a larger view of the network, e.g., for cross-RAT coordination. Every level above the first-level abstraction is again a second-level abstraction, where a vBS is split recursively into multiple vBSs. In that regard, the proposed SD-RAN virtualization layer also allows to implement a recursive slicing scheme.

7.3 Design

In the following, we provide a design for such SD-RAN virtualization layer. For concreteness, we use the FlexRIC SDK introduced in Chapter 5 and the MAC framework abstraction (E2SM) provided in Chapter 6 to propose a realizable design. Due to the MAC framework abstraction, it is implicitly multi-RAT-capable. By using FlexRIC, it supports disaggregated base stations out-of-the-box and uses the E2 interface, making it interoperable with other E2-compliant controllers as long as they use the same E2SMs.

7.3.1 Virtualization Layer

The architecture of the SD-RAN virtualization layer is shown in Figure 7.3. It is a controller specialization leveraging the FlexRIC SDK to *recursively* expose the E2 interface (cf. Figure 7.1). It is built on top of the server library to provide a controller interface towards the RAN infrastructure, and an agent library to provide an agent interface towards the northbound controllers. In particular, the SD-RAN virtualization layer uses the multi-controller support of the agent library to handle multiple controllers (see Section 5.3.1).

The SD-RAN virtualization layer enables the northbound controllers to slice their virtual networks independently. We focus on the NVS algorithm [24] for the virtualization of the RAN resources. For an overview over controllable resources and state, we refer back to Table 6.6 on page 98. The virtualization layer implicitly abstracts the RAT-specific heterogeneity of the network, because the E2SM enables control for both 4G and 5G based infrastructure.

The virtualization layers uses an internal data structure to (1) keep track of the controllers (e.g., IP address), (2) their corresponding SLA, indicating the resource share of physical resources q_v that are assigned to an operator v, and (3) the association of UEs. It slices resources and state of the underlying RAN in order to expose them to the northbound controllers. Each controller should be aware of the UEs that are connected to the RAN by means of the chosen PLMN during RRC connection setup, be able to sub-slice its guaranteed resources q_v using the NVS algorithm, and associate the UEs in its virtual network to the slices. To achieve this, the following E2SMs are in use:

UE Events E2SM associates UEs to operators by using the 5G NSSAI and/or 4G/5G selected PLMN ID provided during RRC connection.

MAC Stats E2SM provides MAC statistics of the underlying RAN, i.e., state information of each UE.

Slicing E2SM enables the configuration of slices in the RAN. Thus, this E2SM provides information about the NVS slices in the MAC of the RAN. Furthermore, state of the underlying RAN can be modified by associating UEs.

These E2SMs are used by three iApps over the server library, and manage any incoming messages. In particular, the UE Events iApp updates the UE-to-controller association of connected UEs. This association is used by the MAC Stats iApp to rewrite incoming MAC statistics indication messages and forward to each northbound controller, revealing only the associated UEs. Towards the agent library, the MAC Stats iApp implements a RAN function that sends indication messages to the northbound controllers.

The virtualization layer abstracts both the assigned radio resources and slice IDs from their physical representation (southbound) to a virtual representation (northbound). The corresponding Slicing iApp virtualizes the resources of NVS by scaling the physical resource share as described in the following section. Furthermore, slice IDs can be translated by, for instance, using a look-up table that maps the slice IDs chosen by the northbound controllers to free slice IDs in the infrastructure.

7.3.2 Virtualizing NVS

Recall that NVS defines (1) capacity slices $s \in C$ with a share of resources t_s, and (2) rate slices $s \in R$ with a reserved rate r_s^{rsv} over a reference rate r_s^{ref}. Each slice can be guaranteed its reserved resources under the assumption that the total resource share does not exceed the total lBS resources:

$$\sum_{s \in C} t_s + \sum_{s \in R} \frac{r_s^{\mathrm{rsv}}}{r_s^{\mathrm{ref}}} \leq 1 \,. \tag{7.1}$$

We reuse the NVS notation to show how the resources are split and virtualized among the set of vBSs $\mathcal{V}$. Each vBS $v \in \mathcal{V}$ (slice controller) has an associated SLA q_v that denotes its maximum resource share, and $\sum_{v \in \mathcal{V}} q_v \leq 1$. Each vBS configures (sub-)slices,

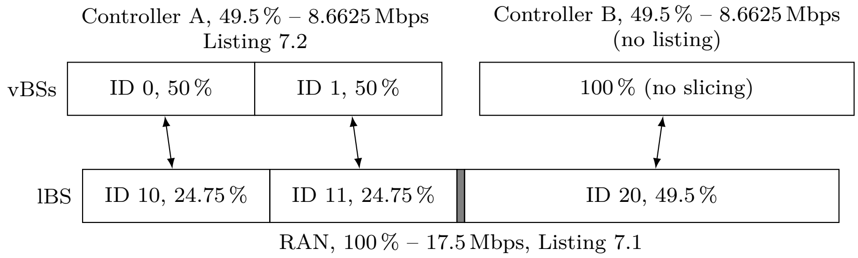


Figure 7.4 – A graphical representation of the virtualization of radio resources for two operators.

denoted C_v and R_v for capacity and rate slices, respectively. Thus, the total resource share of configured (sub-)slices of a vBS v in terms of "physical resources" is:

$$\sum_{s \in C_v} t_s^{\text{phys}} + \sum_{s \in R_v} \frac{r_s^{\text{rsv,phys}}}{r_s^{\text{ref,phys}}} \leq q_v \ . \tag{7.2}$$

Dividing both sides of (7.2) by q_v results in a virtual representation of the resources

$$\sum_{s \in C_v} t_s^{\text{virt}} + \sum_{s \in R_v} \frac{r_s^{\text{rsv,phys}}}{r_s^{\text{ref,virt}}} \leq 1 \ , \tag{7.3}$$

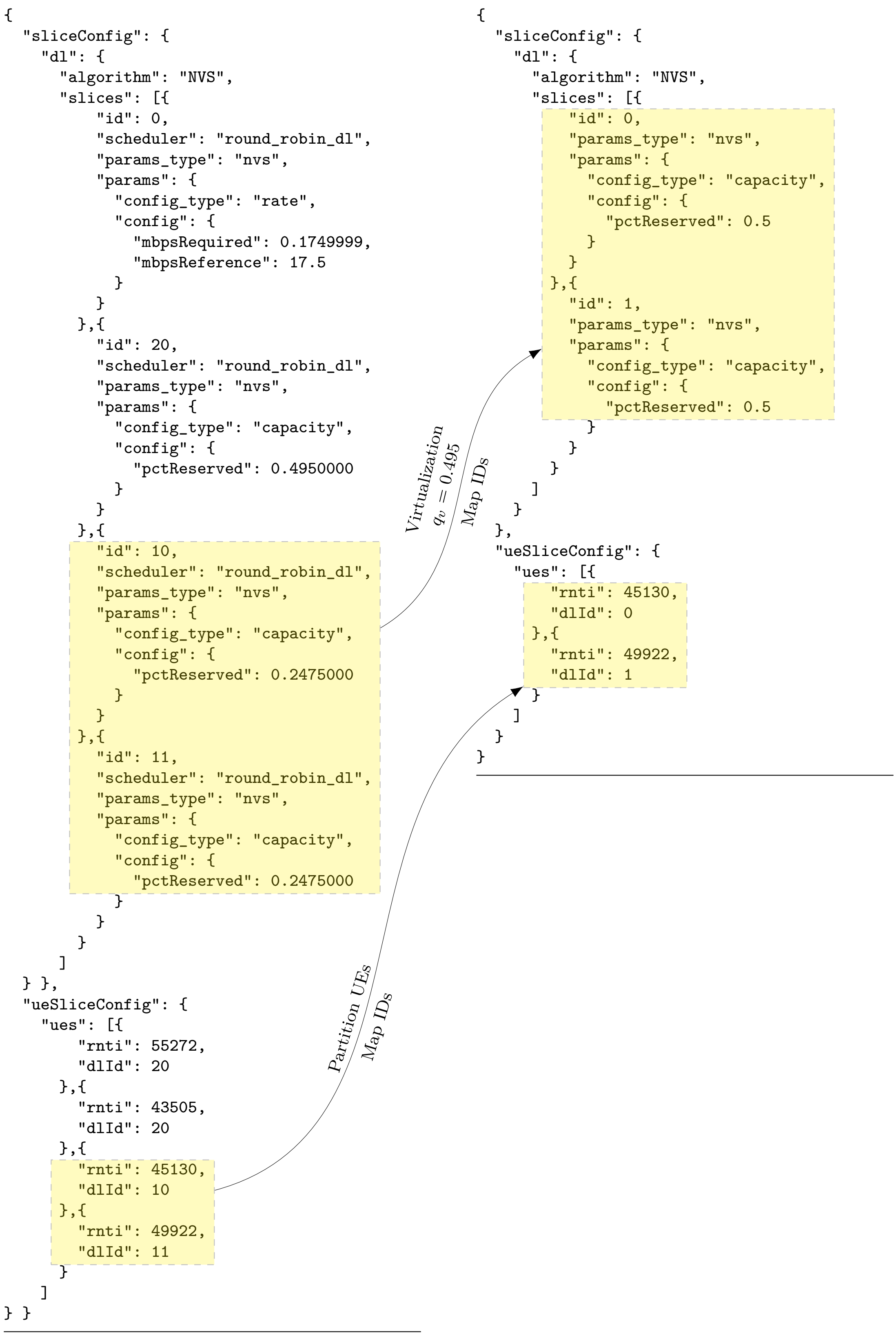
i.e., the virtual resources of all slices sum up to 100 %. Each capacity (sub-)slice has a physical capacity scaled down by factor q_v, i.e., $t_s^{\text{phys}} = q_v t_s^{\text{virt}}$. Each rate (sub-)slice has a virtual reserved rate that is equal to its physical rate, but the reference is scaled to respect the SLA, i.e., $r_s^{\text{ref,virt}} = q_v r_s^{\text{ref,phys}}$.

This scheme guarantees that no controller can exceed its assigned resources, effectively avoiding any conflicts. Since admission control of the virtual slices ensures that the total resource share does not exceed 100%, the physical resource share cannot be higher than the guaranteed SLA, and NVS ensures that every slice receives its configured amount of resources. If a controller did not configure any slices, it is mapped to a physical capacity slice with capacity q_v.

7.3.3 Example

We illustrate the virtualization operation. We consider two operators A and B over a shared base station, as shown in Figure 7.4. They are both assigned $q_v = 49.5\,\%$ of the total resources (1 % is for a "default" slice that schedules UEs after random access, before they send the operator identification, marked in gray). In this example, operator A configured two slices with each 50 % of the (virtualized) resources, which are mapped to each 50 % of the assigned 49.5 % resource share (24.75 %). Operator B did not configure any slices.

An example of an E2 indication message sent from the agent to the virtualization layer, and then to the controller of operator A (first controller), is shown in Listings 7.1 (before virtualization) and 7.2 (after virtualization). The virtualization of slice resources



Listing 7.1 – E2SM before virtualization.

```
{
  "sliceConfig": {
    "dl": {
      "algorithm": "NVS",
      "slices": [{
          "id": 0,
          "scheduler": "round_robin_dl",
          "params_type": "nvs",
          "params": {
            "config_type": "rate",
            "config": {
              "mbpsRequired": 0.1749999,
              "mbpsReference": 17.5
            }
          }
        },{
          "id": 20,
          "scheduler": "round_robin_dl",
          "params_type": "nvs",
          "params": {
            "config_type": "capacity",
            "config": {
              "pctReserved": 0.4950000
            }
          }
        },{
          "id": 10,
          "scheduler": "round_robin_dl",
          "params_type": "nvs",
          "params": {
            "config_type": "capacity",
            "config": {
              "pctReserved": 0.2475000
            }
          }
        },{
          "id": 11,
          "scheduler": "round_robin_dl",
          "params_type": "nvs",
          "params": {
            "config_type": "capacity",
            "config": {
              "pctReserved": 0.2475000
            }
          }
        }
      ]
  } },
  "ueSliceConfig": {
    "ues": [{
        "rnti": 55272,
        "dlId": 20
      },{
        "rnti": 43505,
        "dlId": 20
      },{
        "rnti": 45130,
        "dlId": 10
      },{
        "rnti": 49922,
        "dlId": 11
      }
    ]
} }
```

Listing 7.2 – E2SM after virtualization.

```
{
  "sliceConfig": {
    "dl": {
      "algorithm": "NVS",
      "slices": [{
          "id": 0,
          "params_type": "nvs",
          "params": {
            "config_type": "capacity",
            "config": {
              "pctReserved": 0.5
            }
          }
        },{
          "id": 1,
          "params_type": "nvs",
          "params": {
            "config_type": "capacity",
            "config": {
              "pctReserved": 0.5
            }
          }
        }
      ]
    }
  },
  "ueSliceConfig": {
    "ues": [{
        "rnti": 45130,
        "dlId": 0
      },{
        "rnti": 49922,
        "dlId": 1
      }
    ]
  }
}
```

and their IDs is highlighted between the sections `sliceConfig`; the corresponding UE association is highlighted between sections `ueSliceConfig`.

In this concrete example, slice IDs are mapped from a virtual to a physical representation as follows: each controller is restricted to (virtual) slice IDs in the range $0-9$. The virtual IDs are mapped to physical IDs in disjoint intervals, i.e., range $10-19$ for controller 1, range $20-29$ for controller 2, etc. Correspondingly, as seen in the figure and listings, operator A's slices with IDs 0 and 1 in the virtual representation is mapped to the physical IDs 10 and 11. The same applies for the UE associations.

Regarding the resources virtualization, the example considers a single eNB with a maximum throughput of 17.5 Mbps. If a virtual slice of one vBS has a (virtual) capacity of $t_s^{\mathrm{virt}} = 0.5$, its physical capacity is $t_s^{\mathrm{phys}} = 0.2475$. If one operator created a rate slice with a $r_s^{\mathrm{rsv,phys}} = 5\,\mathrm{Mbps}$ slice over reference $r_s^{\mathrm{ref,virt}} = 8.662\,\mathrm{Mbps}$ (0.495 out of 17.5 Mbps), it is mapped back into the real resources as a 5 Mbps slice and reference rate of $r_s^{\mathrm{ref,virt}} = 17.5\,\mathrm{Mbps}$ (not shown in the example).

7.3.4 Implementation

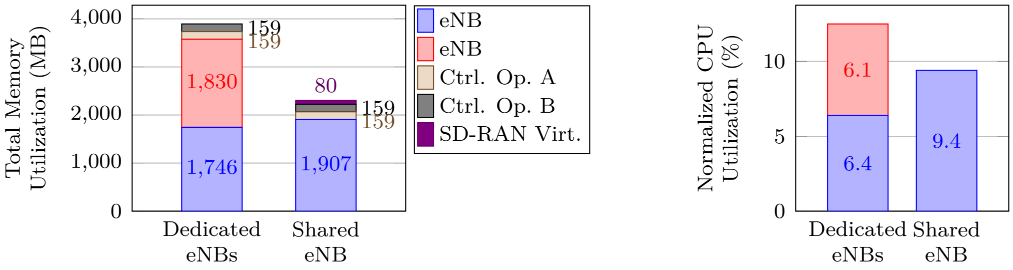
The SD-RAN virtualization has been implemented in roughly 1500 lines of C++ code as a FlexRIC controller specialization. Upon start, it connects to two pre-configured slice controllers. Currently, only the NVS slice algorithm is supported, but might easily be extended to, e.g., use the algorithm presented in Section 6.3 to allow also fixed configurations and low-latency slices.

7.4 Results

We verify the virtualization layer over a 4G/LTE OAI (tag 2021.w20) radio deployment using Ettus B210 SDRs. We consider the scenario of two operators A and B that each use the FlexRIC-based FlexRAN controller specialization (Section 5.4.3). Each operator has two UEs. We compare (1) the use of two dedicated eNBs per operator with each 25 RBs (5 MHz), both in band 7, and (2) a single shared eNB using 50 RBs (10 MHz) in band 7. The operator share is 49.5 %. In both cases, the agent(s) export data using the Slicing and MAC stats E2SM. In case (2), the eNB connects to the virtualization layer, which in turn virtualizes the E2 messages and sends them to the RAN controllers of operators A and B.

We compare the infrastructure overhead in terms of memory utilization for both cases in Figure 7.5a. We observe almost double the memory utilization when comparing the dedicated to the shared case, since two eNBs are deployed. The virtualization layer, on the other hand, only uses 80 MB of memory, and has a negligible memory usage compared to the infrastructure. The SD-RAN controllers incur the same overhead in both cases. Sharing the infrastructure thus greatly reduces the memory usage.

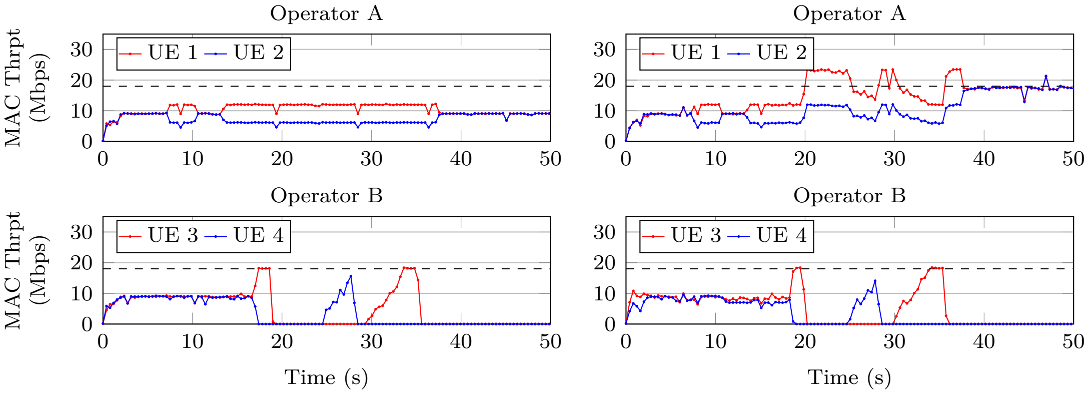
Further, we compare the CPU utilization for dedicated vs. shared cases in Figure 7.5b under full load, i.e., when all UEs of both operators have an active TCP DL stream. We observe that the combined CPU usage of the two dedicated eNBs is roughly 30 % higher than in the shared case, although the total served spectrum is 10 MHZ in both cases. We



(a) Memory utilization.

(b) CPU utilization.

Figure 7.5 – Infrastructure overhead for dedicated vs. shared eNB(s).



(a) Dedicated infrastructure.

(b) Shared infrastructure.

Figure 7.6 – UE performance observed by operator controllers, where each operator has two UEs. The horizontal dashed line marks the maximum throughput of 18 Mbps (MAC throughput) at 25 RBs.

conclude that sharing the infrastructure also substantially decreases CPU usage. Note that the employed Slicing and MAC stats E2SMs only exchange messages infrequently, i.e., every 100 ms for MAC stats and only on reconfiguration for MAC stats. The CPU usage of the virtualization layer and SD-RAN controllers is negligeable, and we therefore exclude if from the above comparison.

Next, we investigate the isolation and sharing capabilities of the virtualization layer for a single shared eNB, against those of two dedicated eNBs. Figure 7.6 compares the two cases, and plots the throughput as observed by the respective controllers. Although the scenario comprises four UEs, only the UEs belonging to the respective operators are exposed to each controller, thanks to the virtualization layer in the shared case. In all graphs, the dashed line indicates the maximally achievable MAC throughput for 25 RBs/5 MHz, i.e., of a single dedicated eNB.

Initially, both controllers have no slice configured, and since the resource share is the same, all UEs have the same throughput. At around 8 s, operator A configures two sub-slices (66 %, 33 %). In the shared case, operator B's performance is not impacted by the reconfiguration (it is the same as in the dedicated case), proving the isolation capabilities of the virtualization layer.

Once UE 4 of operator B has no traffic, UE 3 benefits from more resources. However, when the UEs of operator B do not cause any traffic at all, the resources are redistributed equally to the two sub-slices of operator A, proving the sharing capabilities. Further, when the UEs of operator B receive traffic again, the throughput is the same in the shared and dedicated dedicated cases. Thus, the SLAs of the operators are guaranteed.

At 37 s, operator A reconfigures its slices to both 50 %. Observe how both UEs achieve the throughput as if they had separate dedicated eNBs, indicated by the dashed line. This shows the potential of sharing the infrastructure, and in this case of the unloaded operator B, the multiplexing gain can achieve 100 %.

7.5 Discussion and Future Work

The SD-RAN virtualization layer proposed in this chapter enables multiple operators to control their virtual networks over a shared infrastructure. We point out that the slicing concept defined by 3GPP imposes the use of a new PDU session per slice [7], which compromises new PDCP, RLC and possibly SDAP entities per slice. Therefore, creating a slice within a slice is a logical concept rather than physical one, as, for example, no RLC entity can be recursively generated inside an RLC entity. However, we argue that the logical concept of recursively assigning resources remains a valid and useful abstraction that opens a wide range of research opportunities.

First, the (virtual) slice IDs are required to be in the range $0 - 9$, and the presented controller prototype can therefore not be used for recursive virtualization, as a higher layer virtualization would always map to physical IDs > 9. More intelligent abstraction techniques, e.g., similar to virtual memory using look-up tables, might be applied to make such controller truly recursive.

Second, the proposed virtualization layer virtualizes an E2SM for the control of radio resources, and E2SMs of additional RAN levels could be considered. For instance, one could conceive an abstraction for flow management and virtualize such control, similar to FlowVisor [41], to expose monitoring and control application-specific data flows. Similarly, radio control, such as admission control, bearer configuration, or handover management, could be exposed towards multiple operator-specific controllers. Finally, to allow controllers to not only configure sub-slices within their virtual network but control the exact allocation of UEs, the abstraction introduced by Orion [53] could be added. In this context, the latency introduced by the SD-RAN virtualization layer (at least one additional hop with respect to Orion) is critical. Given that Orion reports a throughput drop for hypervisor-controller latencies above 2 ms [53, Figure 13.b] and FlexRIC can be operated at $250\,\mu$s for two hops (Figure 5.12a on page 84), we are confident that this challenge can be overcome.

Third, the presented proof-of-concept provides control for multiple controllers over the same infrastructure and the *same* E2SM. The control for multiple controllers using *different* E2SM could be evaluated to enable service-specific controllers for heterogeneous services over the same infrastructure.

7.6 Conclusion

In this chapter, we introduced the concept of a virtualization layer. The virtualization layer abstracts the RAN infrastructure, and shares the SD-RAN control among multiple controllers, belonging to different service owners or operators. We provide a design for such a virtualization layer that leverages the FlexRIC SDK to recursively expose an interface northbound to allow multiple operators to control their virtual subnetwork in the common infrastructure. We detailed the virtualization for the MAC framework abstraction, which virtualizes the resources in the abstraction (Slicing E2SM) of the MAC scheduling framework, and partitions the MAC statistics such that each operator controller only sees the correspondin UEs. We prototyped the virtualization layer. Measurements show a memory overhead reduction by almost 50 % (due to using only one RAN), a CPU reduction by 30 %, and a multiplexing gain of 100 % (due to sharing the resources between two operators with equal resource share). This work is a first step towards a range of additional research opportunities which have been discussed as well.

Chapter 8

Conclusion

8.1 Summary

5G shifts the current paradigm beyond new radio and spectrum in three major areas: (i) from monolithic to disaggregated architecture, (ii) from proprietary protocols and closed systems to commodity hardware, standardized interfaces and an open environment, and (iii) from a communication-oriented to service-oriented networking.

First, there is an ongoing trend towards RAN disaggregation that allows centralizing the processing of RAN functionality for coordination and multiplexing gains, but increases the complexity of the RAN infrastructure. Second, through the application of virtualization and softwarization principles, the RAN becomes a virtualized, programmable network over commodity hardware that can be customized towards considered use cases, and that is dynamically controlled by an SD-RAN controller. Third, services with heterogeneous requirements are multiplexed over the same infrastructure. However, the service requirements are partially incompatible with each other, and the concept of network slicing has been introduced to accommodate these services in a common infrastructure while fulfilling their requirements. Additionally, while service owners can freely customize and control their slice in the CN thanks to the service-based architecture, the RAN is still a logical single network function that is not open towards service owner control and programmability.

In this context, the concepts of SD-RAN and slicing are of high importance to enable a service-oriented RAN. In this thesis, we made the following contributions towards this goal:

Base Station Abstraction. Chapter 4 conceptualizes the service-oriented RAN. We introduced descriptors to describe the RAN infrastructure following a generalized approach, and use them to further delineate service requirements. Using the FlexVRAN framework, we link these descriptors to logical and virtual base stations, formed through a two-level abstraction, to simplify the RAN by "re-aggregating" the infrastructure, and describe the service embedding. The complementary RAN Engine framework further described how to customize the RAN infrastructure through micro-SDKs, which encapsulate RAN control endpoints and allow the creation of service-specific controllers.

Finally, we analyze the composition of logical base stations, relevant in the context of such “re-aggregation”, through the formulation of an optimization problem.

SD-RAN according to 5G principles. In Chapter 5, we proposed FlexRIC, an SDK that serves as the pillar to build specialized, multi-service SD-RAN controllers. Using the abstractions provided by the SDK, it is possible to compose customized controllers that smoothly adapt to the envisioned state-of-the-art 5G scenarios. It introduces greater flexibility for novel use cases by following 5G design principles such as flexibility and ultra-leanness. We furthermore describe a number of designs that might be implemented using the FlexRIC SDK. Extensive assessments prove its performance and scalability, making it better suited for 5G requirements than comparable SD-RAN controllers, e.g., for an increased number of devices.

Flexible MAC Scheduling Framework. In Chapter 6, we presented a flexible MAC scheduling framework that adapts MAC operation to deployment scenarios and multi-service operation. Through a separation of the scheduler into a preprocessor and postprocessor, the scheduler is easily adapted to different deployments. Additionally, a multi-service extension allows a separation of concern between slice and user scheduling, where the respective functionality can be changed dynamically to respond to service or network requirements. A corresponding E2SM provides an abstraction for fine-grained programmability of the MAC for slices and slice algorithm-specific slice description. We furthermore propose a QoS-aware slice algorithm, which is neatly integrated into the framework. Extensive measurements using simulation, emulation over a 4G emulation mode, and 5G radio deployments show the feasibility, scalability, and multi-service capabilities of this framework. In the context of this work, we implemented the OAI 5G/NR MAC scheduler from scratch.

SD-RAN Virtualization. In Chapter 7, we built an SD-RAN virtualization layer as a proof-of-concept of a service-oriented RAN. It multiplexes the SD-RAN control of multiple tenants, such as operators, onto a shared infrastructure. Such virtualization control exposes a virtualized view of the RAN and enables operator-specific controller to independently control their slice, even over disaggregated base stations. To this end, we used the FlexRIC SDK to implement the corresponding SD-RAN system, and virtualized the slicing control leveraging the MAC scheduling framework E2SM. This allows independent programmability of the respective slices, isolating their networks, while making full use of sharing potentials. A prototype confirmed the feasibility, showing infrastructure savings and muliplexing gains.

Technical Contribution. Apart from the contributions above, a number of technical contributions have been made during this thesis, which we document in Appendix B. The maintenance of the FlexRAN platform and the MAC slicing system in OAI, part of the presented MAC scheduling framework, enabled numerous researchers to contribute original work in the context of 5G [68], [133]–[136].

8.2 Future Work

The future work related to the contributions have been discussed individually within the corresponding chapters; see Sections 4.6, 5.7, 6.5, and 7.5. In the following, we restrict the discussion to high-level future work directions.

The service-oriented RAN has a lot of potential for future research. It is enabled through a virtualization of control over an existing abstraction, and allows providing concurrent control to multiple controllers over a shared infrastructure. In this context, virtualization is important, as it also increases security by isolating the different controllers from each other, even in a public 5G network, and we realized a virtualization of radio resource configuration in Chapter 7. Follow-up work could study the virtualization along the RAN flow (SDAP, PDCP, RLC) and radio control levels (RRC) to abstract the respective control. For instance, in the flow level, such a virtualization might orient on the FlowVisor [41] in SDN networks; in the radio control level, multiple controllers could implement different handover algorithms, adapted to the considered services (e.g., depending on the expected mobility, etc). In general, the programmability in the RAN has to be enabled, E2SMs need to be designed, and the corresponding virtualization mechanisms have to be put in place (compare also with Figure 7.2 on page 127).

Programmability of the RAN is a foundational building block to realize the service-oriented RAN, and we believe it will be increasingly important in the years to come. The MAC framework in Chapter 6 enables an adaption of the scheduler operation, and an integration with an SD-RAN controller, e.g., based on the FlexRIC platform presented in Chapter 5, is essential to further improve the quality of experience for UEs. The programmability in Chapter 6 is restricted to scheduling of data in the shared channels (DLSCH, ULSCH). It might be extended to program beam directions, e.g., to coordinate multiple beams for coordinated multipoint transmission/reception (CoMP). Given that the future 3GPP Release-17 considers slice-specific random access [137], programmability scheduler adaptation for random access could be studied as well. Apart from programmability, the MAC framework provides the ideal foundation to experiment with new approaches to scheduling of radio resources while making use of the advanced enablers of the PHY of NR.

The SDK approach taken by FlexRIC enables service-specific, customized SD-RAN controllers, allowing service owners, such as verticals, to customize and optimize their virtual network through their own controller without the intervention of the infrastructure provider. In this context, a service-specific controller could employ data-driven control decision making [138] to optimize control of its virtual network, without imposing any overhead on other network services, or to implement control at different timescales depending on the considered use cases. Furthermore, the use of a Network Store, as discussed in the context of FlexRIC, opens up a research path to extend the RAN with new functionality.

In a similar direction, artificial intelligence (AI) is a cornerstone in the future Network-2030 [10] and is believed to shape 6G towards a vision of "connected intelligence" vision [139] and "intelligence everywhere" [140]. Through big data analytics and AI-enabled closed-loop optimization, the vast amounts of data that networks will generate can be analyzed, and “intelligent” decision-making allows better steering of the network. As an example, to optimize transmission, reconfigurable intelligent surfaces can improve propagation environments, but their configuration is complex, requiring machine learning approaches such as deep learning to handle the associated complexity [141]. While reconfigurable surfaces seem still to be at an experimental phase, they are part of candidate topics for the future Release-18 of 5G [142]. Through the combination of the MAC framework and custom controllers through FlexRIC, novel scheduling schemes incorporating AI could be researched.

Appendices

Appendix A

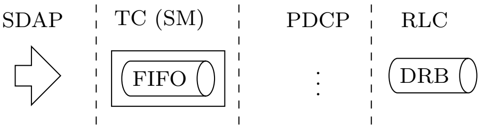
Other Service-Specific RAN Controllers

A.1 Flow-Based Traffic Controller

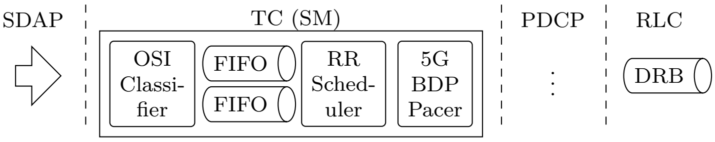
Since latency requirements have increased in 5G, controlling the traffic is of utmost importance in a packet-switched network, where time-sensitive networking support through Packet Delay Budget is standardized [7]. 5G inherits the benefits, as well as the drawbacks, of previous packet-switched cellular networks generations. One of its most important drawbacks is the phenomenon known as bufferbloat [119]. The bufferbloat specifically occurs at the cellular network since i) the bottleneck is commonly located at the radio link; ii) the RLC sublayer is provided with large buffers to absorb the brusque changes that the radio channel may suffer and avoid starving it; and iii) most of contemporary traffic is transported through a lossy based congestion control algorithm (i.e., TCP Cubic), where the algorithm cannot differentiate between the propagation time and the large sojourn time that packets experience in a bloated buffer.

To avoid the bufferbloat, standardization efforts propose to segregate into different radio bearers the traffic of flows prone to generate bufferbloat and such that do not cause it [143]. In a similar spirit, we designed a Traffic Control (TC) E2SM (TC SM) in the context of the FlexRIC work (see Chapter 5) to abstract the configuration of multiple flows within the RAN, similarly to how OpenFlow [32] abstracts flows in a switch. As shown in Figure A.1, the TC SM uses a classifier to segregate packets into flows, schedules the queues, and limits the rates through a pacer. In this manner, we enhanced current 3GPP specifications, similarly to the architecture explored at [144], [145].

To illustrate the simple, yet complete and realistic example from [146], and show how a traffic control xApp can improve the latency, we generated two flows in downlink. One emulating a one minute G.711 VoIP conversation through UDP data frames of 172 bytes with an interval of 20 ms using *irtt* [147], resulting in a bandwidth consumption of 64 Kbps, and a second flow emulating a bufferbloat-prone flow using *iperf3*. We start the *irtt* flow 5 seconds before the *iperf3* flow. However, instead of segregating the flows in data radio bearer (DRB) queues as suggested in [146], we created a second queue at the

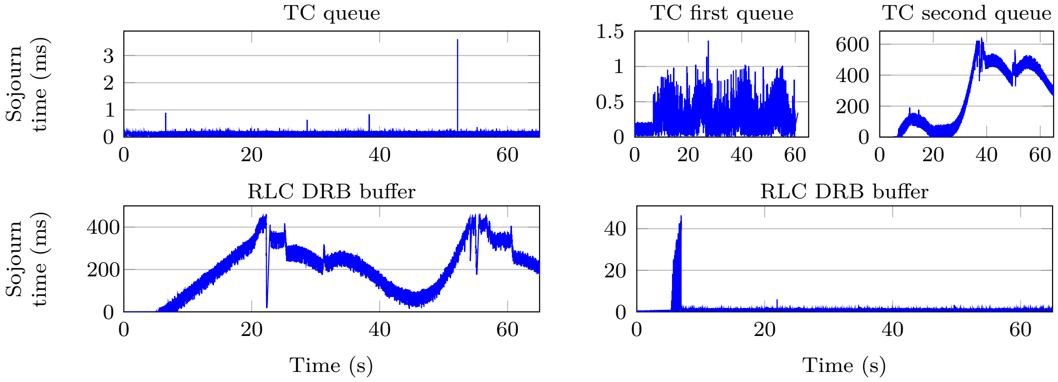


(a) TC SM in transparent mode.



(b) TC SM with an OSI classifier, two FIFO queues, a RR scheduler and a 5G-BDP pacer.

Figure A.1 – Sublayers with buffers in 5G downlink path.



(a) Sojourn time in transparent mode.

(b) Sojourn time in xApp case.

Figure A.2 – Sojourn times for TC transparent mode and when steered through xApp.

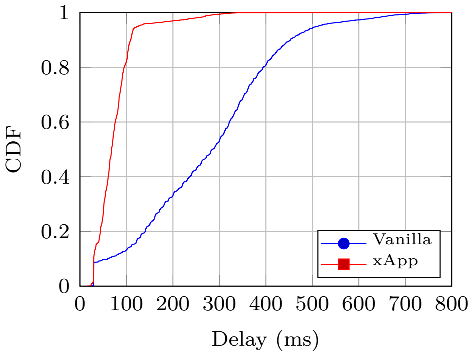


Figure A.3 – CDF of the RTT of VoIP flows.

TC SM, as shown in Fig A.1b. We used the 5G Quectel 500Q-GL COTS as the UE and OAI (tag 2021.w20) with an agent and validated the proposed TC SM. We used Redis as a message broker that forwards the messages received from the RIC to the xApp. Similarly, we used a REST interface between the xApp and the RIC for submitting control messages.

The example can be summarized as follows: The xApp first subscribes to the RLC and TC statistics through their monitoring E2SMs. Note that this information might come from two RAN functions, i.e., DU and CU-UP, which is supported out-of-the-box by FlexRIC. Once the xApp notices that the sojourn time of the packets belonging to the low-latency flow increase beyond a limit, it decides to perform three actions. As its first action, it generates a second FIFO queue. Next, it creates a 5-tuple filter (i.e., source and destination addresses and ports, as well as protocol) to segregate the low-latency flow packets from the rest. Following, it loads a 5G-BDP pacer [119], to backlog the packets into the queues located at the TC, and thus, avoid bloating the RLC bearer buffer. Lastly, the scheduler implements a RR algorithm, that pulls packets from active queues.

On one hand, Figure A.2a shows the bufferbloat effect in vanilla cellular network systems. TCP Cubic bloats the last buffer before the bottleneck link, which in this case is the RLC bearer buffer. Therefore, if flows with low latency requirement share the data path with a greedy flow, they will suffer large sojourn times. On the other hand, Figure A.2b shows how segregating the traffic and using a pacer, limits the bufferbloat problem to flows that share an RLC queue with the greedy flow. The 5G-BDP pacer maintains the DRB buffer uncongested and backlogs the packets into the TC SM. It tries to submit just enough packets to the DRB to not starve it, without bloating it. In this manner, almost full RB utilization can be achieved, while maintaining uncongested the DRB buffer even when the radio channel link varies under realistic traffic conditions [144]. Moreover, generating a second queue and segregating the traffic at the TC avoids experiencing large sojourn times to the VoIP flow's packets. Lastly, Figure A.3 shows the clear advantage of intelligently using TC through an xApp. The RTT of the VoIP flow when is previously segregated is in the order of four times faster. However, Figure A.3 also shows that even though the largest part of the delay is associated with the downlink path, there exists other buffers that may need to be investigated further, as the RTT when no *iperf3* traffic is added varies between 20 and 40 ms.

We implemented the queues, the classifier, the scheduler and the pacer as shared objects to enable loading them online. In this way, the behavior of the traffic flows within our SM can be dynamically changed according to the traffic and network conditions through an xApp. The xApp could even download them from a trusted server or network store [75].

Some examples of 5G scenarios that will need traffic control capabilities to fulfill their latency requirements are URLLC (e.g., V2X) or eMBB (cloud AR) services.

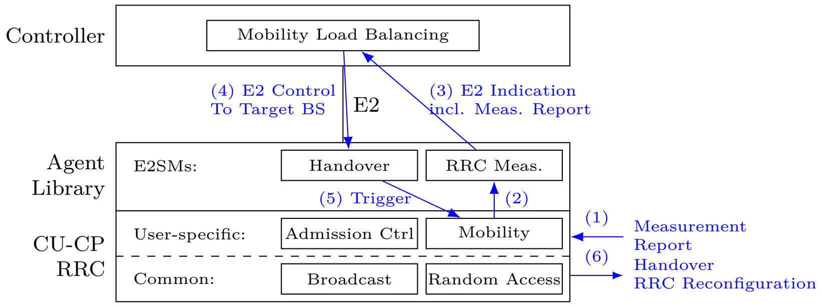


Figure A.4 – A controller can control user-specific RRC functionality, e.g., handover as shown in this information flow.

A.2 Load-Aware Mobility Controller

The RRC in the CU-CP hosts functionality for connection management, mobility, radio resource management, QoS, etc. The functionality can be divided into cell-common and user-specific functions [114]. Common functionality includes information not destined to individual users, such as cell information broadcast or emergency services. User-specific functionality includes UE connection reconfiguration (bearers, measurements, etc.), admission control, mobility management, or secondary cell control (i.e., control for 5G non-standalone mode), and can be handled by a controller in an isolated fashion, exposed through an E2SM. As processing in the CU-CP has soft real-time requirements, the corresponding control decisions can be taken in a controller after forwarding the RRC messages, allowing to customize CP decision or extending the information base for making CP decisions. Through a virtualization as shown in Chapter 7, the RRC context information might also be partitioned for presenting controllers a virtualized view.

An exemplary information flow is shown in Figure A.4. The RRC receives measurement reports for the UE-perceived signal strength of the active and neighboring cells. If a controller subscribed to be notified about the measurement, this information would be forwarded through a dedicated E2SM via an E2 indication message. The controller then evaluates the signal strength (possibly taking into account other quantities, e.g., current cell-load as described below), and might trigger a handover through an E2 control message utilizing a handover E2SM. The RRC creates a reconfiguration message to the UE, and the UE subsequently performs the handover to another cell.

Consider for example an eHealth scenario as depicted in Figure A.5. In this scenario, we consider a paramedic user, e.g., an ambulance, that travels to a hospital. It uses network connectivity to support the mission. A controller is connected to two eNBs that, for the purpose of handover, are connected over the X2 interface [148]. The controller uses network indicators and UE-delivered measurements and can proactively trigger handovers to improve network performance as the ambulance travels through the network, considering load at the base stations, travel path, etc.

We developed a proof of concept controller regarding such scenario[1]. The controller

[1] A video of the described scenario, shown at EuCNC 2019 in Valencia, Spain, in the context of the SliceNET EU project, is available at `https://www.youtube.com/watch?v=2D1Gs192exc`.

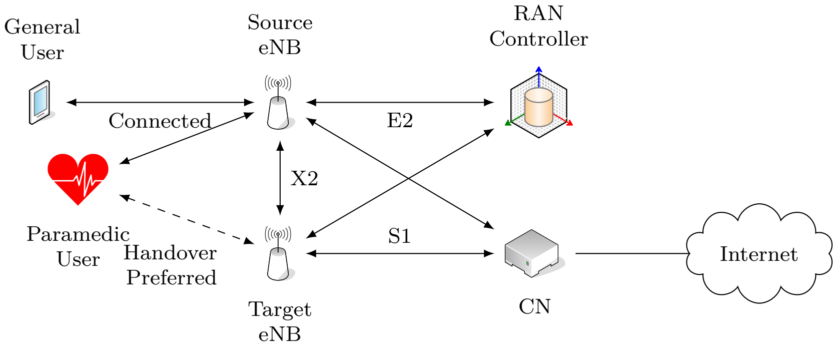
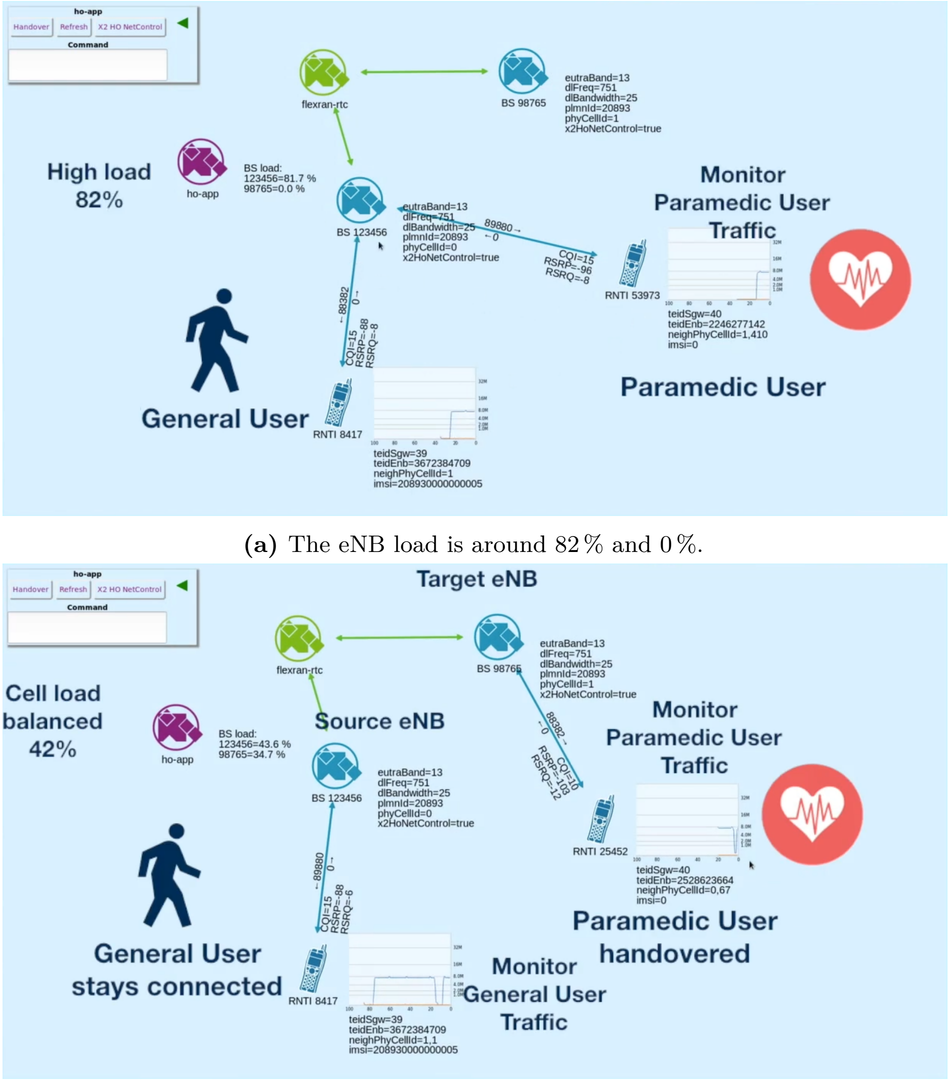


Figure A.5 – Demo setup for the load-aware mobility controller scenario.

measures the load in the base stations, and can trigger handovers to even the load in the network. As shown in Figure A.6a, the controller observes a high eNB load, whereas a neighboring eNB is unloaded. Since the UE measures the eNB's cell strength to be sufficient for good connectivity, the controller triggers a handover, balancing the load in both eNBs as shown in Figure A.6b.



(a) The eNB load is around 82 % and 0 %.

(b) Balanced load (around 40 %).

Figure A.6 – Screenshots of the demo dashboard: (a) before handover, (b) after handover.

Appendix B

Technical Contributions

During this thesis, the author maintained the FlexRAN SD-RAN controller[1] [33], and contributed to the Mosaic5G[2] [149] and OpenAirInterface[3] [22], [23] platforms. In the following, we report on a number of activities that have been carried out during this thesis, in no particular order:

- Numerous improvements in FlexRAN:
  - Merge of the FlexRAN agent into OAI
    `https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/276`
  - Support of multiple agents (in the context of the FlexVRAN work, presented in Chapter 4)
    `https://gitlab.eurecom.fr/flexran/flexran-rtc/-/tags/v2.1`
  - Event subsystem for app notification
    `https://gitlab.eurecom.fr/flexran/flexran-rtc/-/tags/v2.1`
  - Integration into the ElasticSDK Monitoring framework [120]
    `https://gitlab.eurecom.fr/flexran/flexran-rtc/-/tags/v2.1`
  - Report of base station splits, S1AP statistics
    `https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/726`
    `https://gitlab.eurecom.fr/flexran/flexran-rtc/-/tags/v2.2`
  - Network-controlled handover, and RRC and GTP statistics
    `https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/613`
    `https://gitlab.eurecom.fr/flexran/flexran-rtc/-/tags/v2.2`
  - NetStore support (Section 5.6)
    `https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/913`
    `https://gitlab.eurecom.fr/flexran/flexran-rtc/-/tags/v2.4.0`
  - *apidoc* documentation
    `https://mosaic5g.io/apidocs/flexran/`

[1] Available at `https://gitlab.eurecom.fr/flexran/flexran-rtc`
[2] Available at `https://gitlab.eurecom.fr/mosaic5g/mosaic5g/`
[3] Available at `https://gitlab.eurecom.fr/oai/openairinterface5g/`

  - FlexRAN applications, e.g., `recorder` for statistics tracing

- "L2 simulator" improvements to make many UEs (> 4) work (used in Chapter 6)
  https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/922
- Implementation of the MAC framework in 4G (Chapter 6)
  https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/798
  https://gitlab.eurecom.fr/flexran/flexran-rtc/-/tags/v2.3.0
- Implementation of the Multi-user scheduler in 5G, including MAC framework (Chapter 6)
  https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/908
  https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/980
  https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/987
  https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/1015
  https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/1100
- Contribution to the F1 split in OAI (in the context of the FlexVRAN work, presented in Chapter 4)
  https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/524
  https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/f1-interface
- "Soft-restart" functionality of OAI, notably used for a "Spectrum management application" [150] for dynamic spectrum management
  https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/384
- S1-flex functionality in OAI and corresponding programmability through FlexRAN (add/remove MMEs, rewrite PLMNs)
  https://gitlab.eurecom.fr/oai/openairinterface5g/-/merge_requests/404
  https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/s1-flex-conf-nnsf
  https://gitlab.eurecom.fr/flexran/flexran-rtc/-/tags/v2.3.0
  https://mosaic5g.io/apidocs/flexran/#api-PlmnManagement

  Together with the MAC framework, this can be used to create end-to-end slices with dedicated CNs.
- Mosaic5G store apps, such as the `drone`, `rrm_kpi_app`, `ho_app` applications, and maintenance
  https://gitlab.eurecom.fr/mosaic5g/mosaic5g/-/wikis/tutorials/store
- Mosaic5G community work: heavy contribution to the wiki, especially with tutorials, intensive support of the mailing lists
  https://gitlab.eurecom.fr/mosaic5g/mosaic5g/-/wikis/home
  https://gitlab.eurecom.fr/mosaic5g/mosaic5g/-/wikis/mailing-lists

Appendix C

Summary

C.1 Introduction

C.1.1 Evolution towards Service-Oriented Networks

Mobile wireless networks have become a part of everybody's daily life. Within a couple of decades, generations of mobile networks reshaped the way how we interact and communicate. From 2G (2$^{\text{nd}}$ generation) via 3G to 4G networks, every generation boosted network data rates to enable new multimedia use cases. Now, 5G networks are arriving, and 5G is not different. As surveyed in Cisco's annual internet report [1], the average global mobile data rates are rising at an annual 20 % rate, reaching 43.9 Mbps by 2023 from 13.2 Mbps in 2019, which is in particular attributed to 5G. However, increasing data rates, used for multimedia services destined to people, are only one part of the development that is happening. In fact, an increasing number of subscribers in mobile networks are *machines*. As also reviewed by Cisco, there is a trend towards machine-to-machine (M2M) communications. The number of M2M connections is set to grow almost 2.5-fold, from 6.1 billion in 2018 to 14.7 billion connections in 2023, and consist of a mixture of services, such as connected home, connected cars, or energy. These services are associated to industry verticals that provide such service to a specific customer group. M2M communications are becoming more and more widespread, but their requirements differ from traditional, data-rate-oriented use cases.

In 2015, the International Telecommunications Union (ITU) defined three broad usage scenarios of services [2] towards the international mobile telecommunications (IMT) standard in and beyond the year 2020 (IMT-2020). Those are (i) enhanced Mobile Broadband (eMBB) for human-centric, multi-media-focused use cases, (ii) Ultra Reliable, Low-Latency Communications (URLLC) for services with requirements on reliability, latency, and availability, such as industry automation or gaming, and (iii) massive Machine-Type Communications (mMTC) for purely machine-centric use cases, especially for those including many devices, such as the smart city. The vertical's use cases can be associated to these usage scenarios, and ITU put forward a number of requirements, such as 10x increase of user experienced rate (mostly eMBB), 10x lower latency (mostly URLLC), or 10x higher connection density (mostly mMTC).

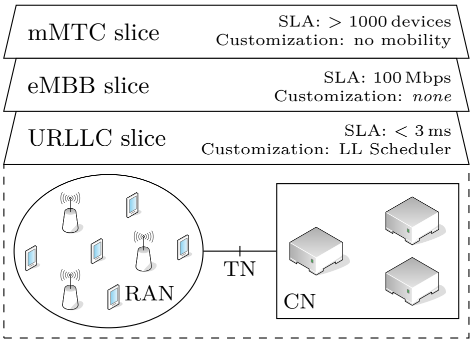


Figure C.1 – The concept of slicing: multiple slices are overlaid over a common infrastructure.

However, the requirements of use cases within these usage scenarios are partially mutually incompatible. Correspondingly, the concept of network slicing [4] has emerged to support multiple services with heterogeneous requirements over a shared infrastructure. Using network slicing, virtual networks are created for each service over a common underlying infrastructure, as shown in Figure C.1. The virtual networks, called slices, create service-specific end-to-end networks that tailor the infrastructure towards the service's requirements across Radio Access Network (RAN), Core Network (CN), and Transport Network (TN). The support of network slicing has been identified early as a key requirement to provide the necessary adaptability in 5G networks [5]. Within the 3$^{\text{rd}}$ Generation Partnership Project (3GPP)-defined 5G System, while customization and control is relatively straight-forward in the CN, the RAN is logically a single network function [8], and slicing remains implementation-specific [9]. While a customization towards the different usage scenarios is foreseen, there are no means for service owners such as verticals to control their virtual network. To enable such control by service owners, a *service-oriented* RAN design is required that is not only flexibly sliced, but in which service owners also gain the required level of control over their virtual networks while being isolated from each other.

C.1.2 Motivation and Contribution of the Thesis

Introducing the necessary flexibility towards a service-oriented RAN is challenging, as it is the most complex part of the overall network, and plays a significant role in the achievement of the requirements put forward by ITU. The Radio Access Technology (RAT) of 5G, named New Radio (NR), is designed towards key principles such as *flexibility*, *forward-compatibility*, and *ultra-leanness*, to be adaptable to novel and future use cases. This is accompanied by technological enablers such as millimeter-Wave (mmWave) or network densification which promise to achieve the requirements of the use cases, but require higher control and coordination. The concept of RAN disaggregation, where the RAN is split into network functions to enable centralization, allows increasing coordination and achieves multiplexing gains, but in turn increases complexity. Softwarization and virtualization are further considered key enablers for 5G networks [16]. The separation of control plane and user plane through softwarization enables programmability, is

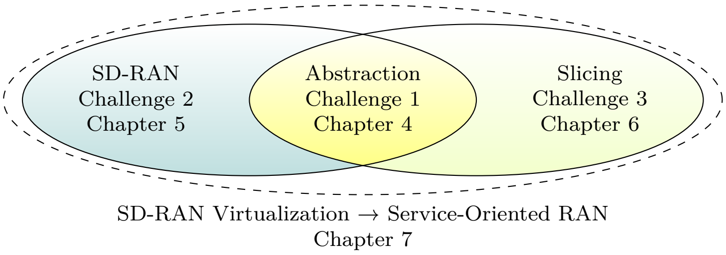


Figure C.2 – Mapping of thesis chapters to challenges.

well established [20], and achieved through Software-Defined Radio Access Networking (SD-RAN). Virtualization allows running RANs on commodity hardware.

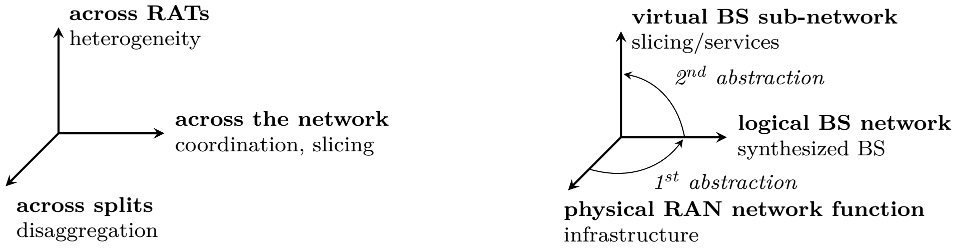
In summary, 5G networks go beyond a new radio and wider spectrum; they are about the evolution of computing for wireless networks in support of a variety of services and capabilities in diverse usage scenarios. However, we observe that while much work went into attaining the heterogeneous performance requirements, there is still a lack of flexible programmability within the RAN to enable independent monitoring, control, and coordination on a per-service basis. In this respect, we identify three key challenges that need to be addressed to attain the vision of a *service-oriented* RAN.

Challenge 1: **Base Station Abstraction.** It is of paramount importance to enable a lightweight and customizable RAN service definition on top of a common infrastructure to adapt the RAN towards the requirements of the services, and enable programmability for multiple services, while hiding unnecessary complexity of the RAN. To this end, the disaggregated infrastructure has to be abstracted and represented towards the services in a simplified manner, and multiple service-specific controllers should be enabled.

Challenge 2: **SD-RAN Infrastructure According to 5G Principles.** As has been depicted, 5G-NR provides unprecedented flexibility in the RAN to support diverse use cases in a multi-tenant environment. In general, existing SD-RAN approaches do not adhere to 5G principles, since they either do not offer the flexibility and composability that is found in the underlying RAN infrastructure, or come as large "one-size-fits-all" solutions, violating the ultra-lean principle. Furthermore, they do not enable the realization of mentioned abstractions.

Challenge 3: **Slice-Aware 5G MAC Scheduling.** 5G networks are designed (a) to operate in a number of different deployment scenarios, e.g., sub-6 GHz and above, (b) to remain forward-compatible to new use cases, and (c) to handle multiple services. Therefore, the scheduler shall be designed to support the corresponding requirements.

In this thesis, we investigate the SD-RAN and slicing concepts to provide flexibility and programmability in the RAN infrastructure to realize service-oriented RANs all while being able to adapt the network to service requirements, and cope with the accompanying complexity. The thesis is structured around the challenges as shown in Figure C.2. To this end, we (1) investigate abstractions to simplify the RAN and embed services (challenge 1, Chapter 4, summary in Section C.2), (2) design a novel SD-RAN platform that follows 5G design principles (challenge 2, Chapter 5, summary in Section C.3),



(a)

(b)

Figure C.3 – The 5G network can be deconstructed into various dimensions: (a) a split-network-RAT view, generalizing into (b) the physical, the logical, and the virtual dimensions.

(3) present a Medium Access Control (MAC) scheduling framework to provide deployment flexibility, programmability and extensibility in a multi-service infrastructure (challenge 3, Chapter 6, summary in Section C.4), and leverage this work to finally (4) propose an SD-RAN virtualization layer to implement a service-oriented RAN that allows multiple service-specific controllers belonging to different service owners or operators to control their virtual network (Chapter 7, summary in Section C.5, incorporates work from all previous chapters).

C.2 Chapter 4: Base Station Abstraction

A number of trends lead to increasingly complex RANs. First, disaggregated Base Station (BS) deployments allow centralizing RAN processing at centralized or edge clouds and flexibly composing BS functionalities, but complicate the management of individual BSs. Second, network densification allows increasing area traffic capacity and connection density, but requires better coordination in the RAN across BSs and among services. Third, 5G does not only provide a specific RAT but also utilizes the available spectrum bands (e.g., mmWave) and access technologies (e.g., 4G, 5G) in a heterogeneous manner to fulfill diverse service requirements. Therefore, a proliferation of RATs entails an increased complexity for the management of the network, as User Equipments (UEs) need to be managed across these RATs. In total, we observe that the 5G RAN has to serve multiple dimensions as shown in Figure C.3a, which increases the complexity for control and management. Additionally, complexity is exacerbated considering that 5G networks need to multiplex a variety of services with heterogeneous performance and customization requirements. Regarding verticals, they wish to provide their network without having to run a full-fledged network, while maintaining control over their virtual network.

To tackle this complexity, we envision a framework that simplifies and abstracts the RAN to efficiently manage this complexity. As shown in Figure C.3b, the physical infrastructure using functional splits and various RATs can be abstracted to compose a logical Base Station (lBS) spanning the full protocol stack, e.g., 4G evolved Node B (eNB) or 5G next-generation Node B (gNB). Such an lBS denotes a "synthesized" representation of the physical network, and can be seen as an enabler for effective management and control of the network by reducing network complexity of heterogeneous deployments (e.g., distributed, disaggregrated, centralized). This lBS is further abstracted to provide a virtualized slice-specific sub-network for network slices, while still maintaining user plane programmability. Such a virtual sub-network is formed from a set of virtual Base Stations (vBSs) to reveal slice-specific resources and processing customizations, corresponding to service requirements. Through a lightweight and customizable RAN control endpoint abstraction, slice-specific customizations and extensions can then be incorporated into the infrastructure, enabling ervice owners to control their virtual sub-networks while being isolated from each other.

We conceptualize the RAN through descriptors that simplify the description of the RAN, and define the required resources and processing customizations of embedded services. They are linked to the infrastructure using the two-level abstraction. The descriptors define the lBSs and vBSs using a representation comprising resources (amount of radio resources), state (configuration, statistics) and processing (RAN functionality that can be customized/extended). After the first-level abstraction, an lBS is represented as a *pipeline descriptor*. Further, a *service descriptor* allows describing service requirements and slice customizations corresponding to the vBS after the second-level abstraction. Properties such as performance isolation, sharing, customization, etc., should be ensured when mapping services to the RAN. Additionally, the descriptor should be forward-/backward-compatible such that services might be mapped to different RATs and deployments as long as all requirements can be fulfilled.

The two-level abstraction and the resulting descriptors are detailed using the FlexVRAN framework. FlexVRAN interacts with the RAN infrastructure that is annotated with (a part of) the descriptor. FlexVRAN then performs the first-level abstraction by (1) merging the descriptors into one lBS, and (2) creating a view of the network topology. This lBS is annotated with the pipeline descriptor, and to account for the heterogeneity of the underlying system, it could be represented as a collection of service models, "synthesizing" the capabilities of the RAN functions in the infrastructure along their resources and state. Further, FlexVRAN performs the second-level abstraction by (1) splitting (slice-wise customizable) lBSs into vBSs and (2) embedding them into the logical network topology in order to reveal a customized view of the virtual network (vRAN) to the service owner. Splitting refers to exposing only a part of the resources, state and processing from the underlying lBS to each vBS in a virtualized form, where the specific virtualization mechanism varies and depends on the protocol stack layer.

While FlexVRAN elaborates on the two-level abstraction to simplify the handling of disaggregated, complex RAN deployments through a unified representation, it does not explain how services can customize or extend processing. The RAN Engine framework (cf. Figure C.4) allows the mapping and multiplexing of services onto the RAN infrastructure, while enabling service owners to monitor, configure, customize, and extend slices, subject to access control. To this end, the service engine enables pipeline customization through micro-Service Development Kits (micro-SDKs), which provide atomic, independent, extensible Control Plane (CP) abstractions, and implement an abstract interface towards the services for RAN customization and extension, allowing a lightweight virtualization

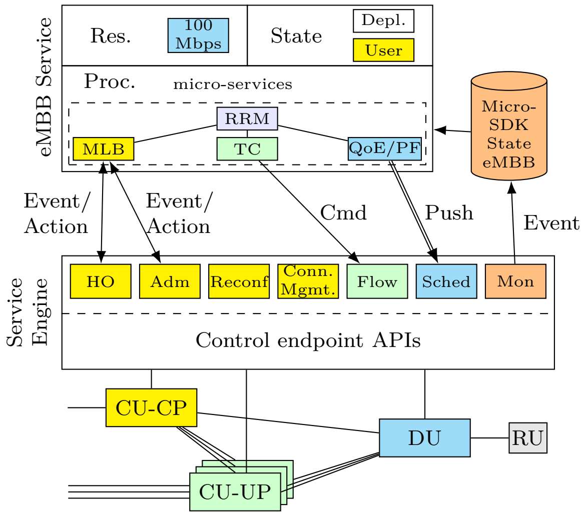


Figure C.4 – Services multiplex their micro-service customizations through the service engine onto the underlying RAN infrastructure by matching micro-SDKs. Based on the micro-SDK, a message-based exchange might be employed (e.g., for radio control) or functionality will be pushed into the engine for timing constraint reasons (radio resource scheduling). Rate information is used to reserve resources in the RAN, and state is partitioned per service.

of the BS's CP *when needed*. The set of micro-SDKs of a RAN pipeline groups the actual control plane Application Programming Interfaces (APIs) for services in a deployment-independent and vendor-independent fashion, and can be classified into the radio resource, flow handling, and radio control dimensions. By combining micro-SDKs, the service descriptor marks customization intentions for this service, and either supplies the execution logic in binary form, or an endpoint for message-based interaction. This is supported by general event-action messages, direct service-originated commands, or the embedding of custom logic within or close to the engine ("push"). As shown in Figure C.4, a service can supply micro-services for Mobility Load Balancing (MLB), Traffic Control (TC), and scheduling (Sched) to extend RAN processing and steer its service.

The number of lBSs that can be composed of RAN functions (Centralized Unit (CU), Distributed Unit (DU)) not only depends on RAN but also on TN resources. For instance, the TN needs to support the data rates that the RAN radio resources offer. Therefore, the chapter is complemented by a problem formulation regarding the composition of lBSs from DUs and CUs considering the available spectrum bandwidth as well as midhaul and backhaul capacity constraints. We show that such problem, even under relaxed constraints, is NP-hard, and solve a simplified version analytically.

C.3 Chapter 5: FlexRIC SDK

Unlike previous mobile networks, 5G-NR provides unprecedented flexibility in the RAN to support diverse use cases in a multi-service environment. In this context, the need for programmability and control through SD-RAN is well established. However, while the

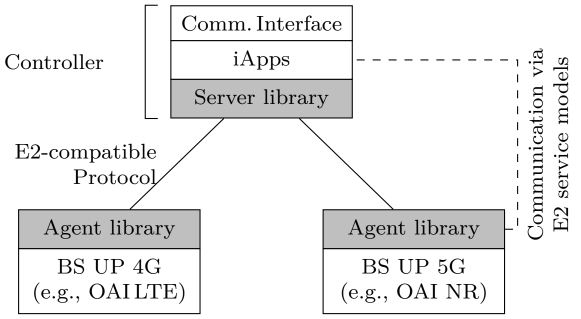


Figure C.5 – The FlexRIC SDK consists of an agent and a server library. The agent library provides integration of a base station to a controller, which is specialized to a particular use case using iApps.

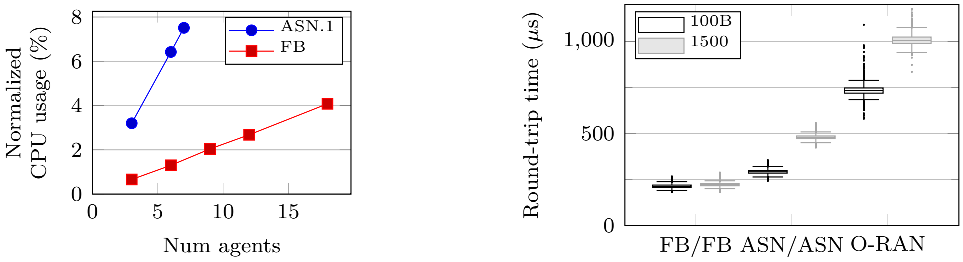
underlying RAN is designed to be ultra-flexible and lean, existing SD-RAN controllers are either not flexible to address all use cases, or they use a "one-size-fits-all" approach.

FlexRAN [33], the first attempt of a real-time SD-RAN platform for research purposes, defines a custom southbound control protocol which is coupled with the underlying RAT and tailored towards specific control operations, limiting its extensibility. The O-RAN RAN Intelligent Controller (RIC) [50] entails a micro-service architecture with associated overhead (e.g., fat resource footprint, increased latencies) even on use cases that do not need it, following a "one-size-fits-all" design, violating central 5G principles such as flexibility and ultra-lean design. We observe that there is still a lack of proper SD-RAN design that adheres to the fundamental design principles of 5G, and that is flexible enough to help implement the abstractions described in Chapter C.2.

FlexRIC is a flexible and efficient Software Development Kit (SDK) that allows to build specialized SD-RAN controllers, and that is in line with the 5G principles. The objective of the SDK is to facilitate the realization of specialized SD-RAN controllers to target specific use cases, while being simple to use. FlexRIC does not support extra features endogenously following the *zero-overhead principle*. It consists of a server library and an agent library. In its simplest form, the SDK can be used to implement an SD-RAN controller using an E2-compatible protocol, as it is shown in Figure C.5.

The agent library is the basis to extend a base station with the agent functionalities in order to integrate it with an E2 controller. It provides an API to implement custom RAN functions, i.e., RAN functionality that can be monitored and/or controlled by applications, and comes with a bundle of pre-defined RAN functions that implement a set of E2 Service Models (E2SMs) that can be included. Additionally, it allows handling multiple controllers.

A controller is built through the server library, internal Applications (iApps), and optionally a communication interface. The server library manages agent connections, and routes messages between iApps and the agents. Through the iApps, it is possible to modularly build specialized controllers: based on the considered use case, iApps can implement E2SMs themselves, or expose information to external Applications (xApps) via a northbound communication interface. In the latter case, xApps can control the



(a) CPU utilization for FlexRIC.

(b) RTT for two hops vs. O-RAN RIC.

Figure C.6 – The E2AP encoding can be changed to adapt to the use case. FlexRIC is faster than O-RAN RIC.

RAN while being functionally isolated from the controller, which is the favored method of the O-RAN RIC, but bears a certain overhead. It has to be pointed out that the server library itself does *not* implement any E2SM, and does not request any information from the agent by itself. Instead, iApps have to trigger service-model related communication, and the server library provides the platform to multiplex messages between agents and iApps.

The FlexRIC SDK provides an abstraction of the E2 interface via an internal representation of E2 messages. It is therefore straight-forward to integrate FlexRIC with pre-existing E2-compliant SD-RAN infrastructure. However, the standard mandates an encapsulation of ASN.1-encoded data (E2SM) inside of ASN.1 messages (E2AP), which may be inefficient in certain cases (see below). Therefore, the SDK also supports to change both the encoding scheme and transport protocol allowing the integration of other E2 protocols for low-overhead, real-time control.

In particular, we implemented the E2AP encoding in both ASN.1 and Flatbuffers (FB) [118]. ASN.1 leads to smaller messages sizes at the expense of higher encoding overhead, whereas Flatbuffers leads to comparatively larger messages with faster encoding. The encoding overhead is measurable, as shown in Figure C.6a. Here, we measure the CPU utilization for a statistics information from multiple base station agents. ASN.1 leads to higher CPU utilization, and therefore limits how many base stations can be supported. Thus, Flatbuffers might be preferred for use cases involving many base stations and a frequent message exchange. On the other hand, the message Round-Trip Time (RTT), as shown in Figure C.6b, is always lower for FlexRIC than the O-RAN RIC, making it particularly useful for low-latency use cases.

FlexRIC's modular design permits to easily compose specialized controllers on top of the server library through the iApps, xApps and selection of E2SMs. Moreover, its design opens up new opportunities for the development of state-of-the-art research controllers (e.g., recursive or multi-RAT controllers). In this context, we develop specialized, service-specific SD-RAN controllers for slicing (see Section C.4), traffic control, and load-balancing. Furthermore, leveraging FlexRIC, we detail the designs for a generic, FlexRAN-like [33] controller, a simple RAN monitoring tool, an O-RAN-compatible controller, and Orion [53]. It is also possible to recursively expose an agent interface at the northbound for an

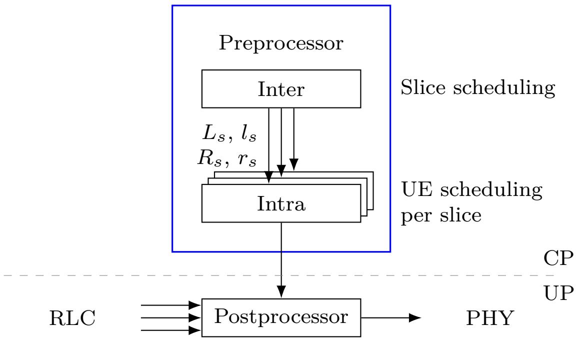


Figure C.7 – Decomposition of scheduling into preprocessor and postprocessor, and multi-service extension.

SD-RAN virtualization layer, as is detailed in Section C.5. Finally, we also detail how to incorporate a Network Store to extend the RAN with customized network functions.

C.4 Chapter 6: Flexible MAC Framework

5G is envisioned as an ultra-flexible, ultra-lean network that can be adapted to different deployment scenarios, and that is supposed to multiplex a variety of services with heterogeneous requirements onto a shared, common infrastructure. A crucial point is to manage this flexibility, and adapt the way radio resources are scheduled depending on the considered deployment scenarios and services that are currently transported. The MAC scheduler has particular importance, as it is the “brain” of short-term radio resource allocation.

To allow for adaptation in the scheduling pipeline towards different deployment scenarios and supporting diverse services, we decouple both the uplink and downlink user scheduling phases into a preprocessor and a postprocessor, as shown in Figure C.7. The preprocessor is an exchangeable module that takes the scheduling decisions, whereas the postprocessor applies the preprocessor decisions to the user plane. The preprocessor is adapted to the use case, including knowledge on frequency ranges, operation mode, service multiplexing, etc. Using cell information (e.g., Time Division Duplexing (TDD) configuration and slot formats), local UE-specific information (e.g., buffer status), and feedback information (e.g., HARQ Ack/Nacks), it makes scheduling decisions and feeds the corresponding information, e.g., resource block allocation and modulation and coding scheme (MCS), to the postprocessor.

Further, a multi-service extension of the preprocessor decouples it into an inter-scheduler and an intra-scheduler. The former works at the level of groups of users, i.e., slices, and decides about the resource allocation following a two-level approach [54]. Depending on the decision outcome governed by a slice algorithm, the intra-scheduler is called with the eligible resources R_s and maximum number r_s, and a list of UEs L_s and maximum number l_s to schedule. Then, the intra-scheduler schedules individual UEs according to a user scheduling algorithm.

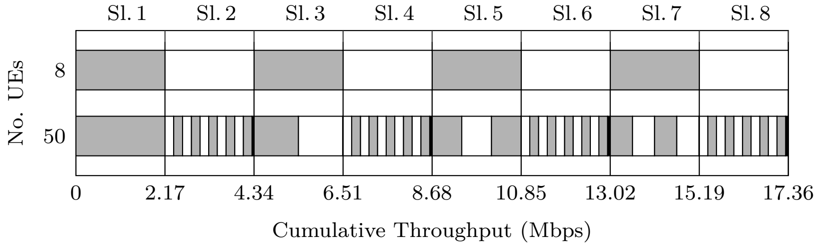


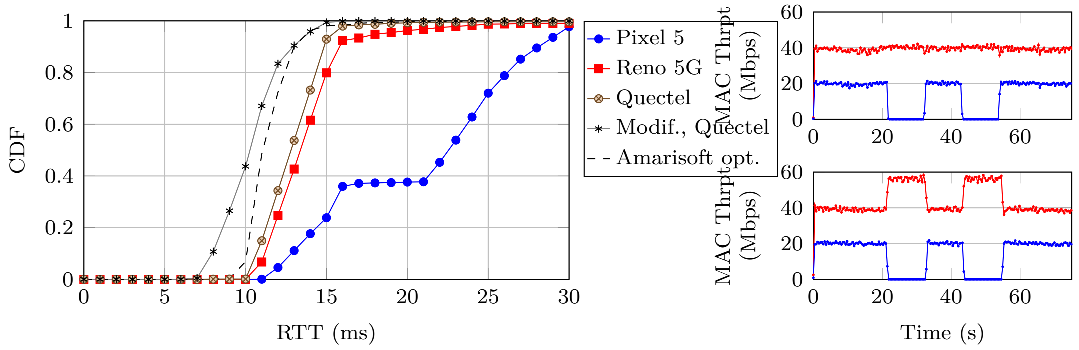
Figure C.8 – Resource isolation and scalability for slicing. Above: Eight slices with eight users have the same throughput. Below: With additional users, the per-slice throughputs remain the same.

The MAC framework has been implemented in OpenAirInterface (OAI) [22], [23] for both 4G/Long-Term Evolution (LTE) and 5G/NR. In both cases, a number of slice and user scheduling algorithms have been implemented, namely Static slicing, NVS [24], and "SCN19" (see below) slice algorithms, and round-robin, proportional-fair, and maximum-throughput user schedulers. In the case of 5G/NR, the scheduler has been implemented from scratch, as OAI did not dispose of a scheduler. Through the Slicing E2SM, the preprocessor, more precisely the slice algorithms and present slices, including algorithm-specific slice parameters such as resources or rates, can be configured to for use with an E2-compatible controller, such as FlexRIC.

A slice scheduling algorithm has to accommodate multiple slices with heterogeneous requirements. We observed that previous algorithms do not consider three key types of resource requirements: physical (static) resources, e.g., for emergency services, throughput, e.g., for eMBB services, and latency, e.g., for URLLC services. We develop Quality-of-Service-aware slice algorithm (named "SCN19") by formulating utility functions, and propose a weight-based scheduling algorithm to schedule slices with any of the three resource requirements. This algorithm is furthermore integrated into the MAC framework, showing its extensibility.

The MAC scheduling framework, including the QoS-aware slice scheduling algorithm, is evaluated using simulation, emulation over the OAI 4G/LTE "L2 simulator", and radio setup over the OAI 5G/NR platform and commercially available phones. Using the simulator, we study main slice requirements, such as isolation and customization, and extensively evaluate the QoS-aware slice scheduling algorithm. The emulator allows to evaluate the MAC scheduling framework implementation over OAI in a controlled environment. Also, it enables to emulate many UEs. As shown in Figure C.8, the implementation is scalable. First, 8 slices are deployed for 8 UEs, and resources are shared equally. When additional 42 UEs connect, the slice algorithm enforces the radio resource isolation between slices, such that the application throughput of the individual slices remains the same.

For 5G/NR, we implemented the corresponding multi-user/slice-aware scheduler from scratch according to the MAC scheduling framework, and we highlight the following measurements. Since one of the main goals of NR is to achieve low latency, we measure the RTT from the CN to the UE via the NR RAN and back using *ping* packets with



(a) RTT of *ping* packets over the RAN for different UE models. **(b)** Resource isolation and sharing.

Figure C.9 – 5G/NR radio deployment measurements.

size 64 B sent every 200 ms. We show the Cumulative Distribution Function (CDF) in Figure C.9a for different commercially-available phone models. We observe that most of the models exhibit RTT below that of LTE networks (minimum 15 ms). The behavior of the Pixel 5 is surprising, as it uses the same chipset as the Oppo phone. Also, regarding the "Quectel modif." measurement, the UE is scheduled in every uplink slot, such that it does not have to request a scheduling grant, which further reduces RTT times. Finally, Figure C.9b shows slice isolation and sharing properties applied in 5G/NR. First, each slice follows its attributed resource share of 33 % and 66 %. Second, resource sharing allows to boost a slice's performance if another slice does not use it, maximizing resource usage and leading to a multiplexing gain.

C.5 Chapter 7: SD-RAN Virtualization

The increasing RAN densification leads to growing costs for cellular networks, as more BSs need to be deployed. RAN *sharing* allows reducing costs, since multiple operators share a common infrastructure instead of maintaining dedicated ones. In this chapter, we present how a base station can be shared between multiple operators, while giving both operators the possibility to *slice* their individual networks using their respective SD-RAN controllers. To this end, we develop an SD-RAN virtualization layer that functions as a proxy to virtualize the slicing control, ensuring isolation between operators while exposing control access to their controllers, as shown in Figure C.10.

This SD-RAN virtualization layer *recursively* exposes the E2 interface at its northbound. To (southbound) E2 agents, it appears as an E2 controller; to (northbound) E2 controllers, it appears as an E2 agent. Since E2 is exposed recursively, it is possible to build hierarchies of controllers, where controllers at higher layers perform higher-layer tasks. The virtualization layer natively supports disaggregation through the server library capabilities of FlexRIC, and simplifies the infrastructure towards northbound controllers as also shown in Figure C.10.

The virtualization layer leverages the FlexRIC SDK presented in Section C.3. It uses

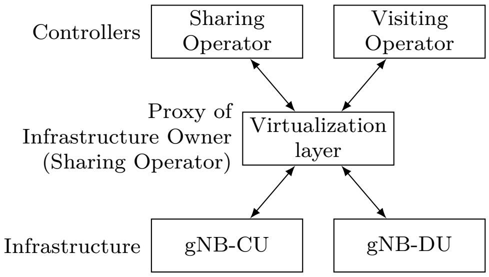


Figure C.10 – Two operators concurrently control slicing in the same infrastructure. A virtualization layer provides the necessary isolation between the operators.

the server library to handle multiple agents southbound, and uses the agent library as its communication interface to expose virtualized E2 control to northbound SD-RAN controllers. In this context, iApp used to specialize a controller take the role as E2 RAN functions towards the agent library.

Since the sliceable resources within a base station are highly heterogeneous, comprising flow control for multiple bearers, radio connection control, and radio resources, the exact virtualization is E2SM-specific. For concreteness, we employ the Slicing E2SM used to control the MAC scheduling framework in Section C.4. The virtualization layer abstracts the assigned radio resources and slice ID from a physical representation (southbound) to a virtual representation (northbound). The IDs are mapped to avoid conflicts. For radio resource configuration, we use the NVS slice algorithm [24]. The resources of slices are scaled by an operator-specific resource share to provide the virtualized resource share for a virtual base station, which sums up to 1.

We implemented a proof-of-concept of the virtualization layer using the FlexRIC SDK over OAI. Memory utilization comparisons indicate an almost 50 % decrease of resources for the considered setup using the "L2 simulator"[1], which comes by using a only one instead of two infrastructures. Additionally, we ran an experiment with two operators A and B and equal resource share of the infrastructure. The multiplexing gain for data rates amounts to up to 100 % if one operator is inactive.

[1] This emulation mode does not use the physical layer. For radio deployments, the memory saving would be around 30 %.

Annexe D

Résumé

D.1 Introduction

D.1.1 Evolution vers les réseaux orientés services

Les réseaux mobiles sans fil font désormais partie de la vie quotidienne de chacun. En l'espace de quelques décennies, les générations de réseaux mobiles ont remodelé notre façon d'interagir et de communiquer. De la 2G (2^{e} génération) aux réseaux 4G en passant par la 3G, chaque génération a augmenté les débits de données du réseau pour permettre de nouveaux cas d'utilisation multimédia. Aujourd'hui, les réseaux 5G arrivent, et la 5G n'est pas différente. Comme l'indique le rapport annuel de Cisco sur l'Internet [1], les débits de données mobiles mondiaux moyens augmentent de 20 % par an, pour atteindre 43,9 Mbps en 2023, contre 13,2 Mbps en 2019, ce qui est notamment attribué à la 5G. Cependant, l'augmentation des débits de données, utilisés pour les services multimédias destinés aux personnes, n'est qu'une partie de l'évolution en cours. En effet, un nombre croissant d'abonnés aux réseaux mobiles sont des machines. Comme le souligne également Cisco, la tendance est aux communications de machine à machine (M2M). Le nombre de connexions M2M devrait être multiplié par près de 2,5, passant de 6,1 milliards en 2018 à 14,7 milliards de connexions en 2023, et se compose d'un mélange de services, tels que la maison connectée, les voitures connectées ou l'énergie. Ces services sont associés à des verticaux industriels qui fournissent un tel service à un groupe de clients spécifique. Les communications M2M sont de plus en plus répandues, mais leurs exigences diffèrent des cas d'utilisation traditionnels, axés sur le débit de données.

En 2015, l'Union internationale des télécommunications (UIT) a défini trois grands scénarios d'utilisation des services [2] en vue de la norme internationale des télécommunications mobiles (IMT) en 2020 et au-delà (IMT-2020). Il s'agit de (i) l'amélioration du haut débit mobile (eMBB) pour les cas d'utilisation centrés sur l'homme et les multimédias, (ii) les communications ultra-fiables et à faible temps de latence (URLLC) pour les services ayant des exigences en matière de fiabilité, de temps de latence et de disponibilité, comme l'automatisation industrielle ou les jeux, et (iii) les communications massives de type machine (mMTC) pour les cas d'utilisation purement centrés sur la machine, en particulier pour ceux comprenant de nombreux dispositifs, comme la ville intelligente.

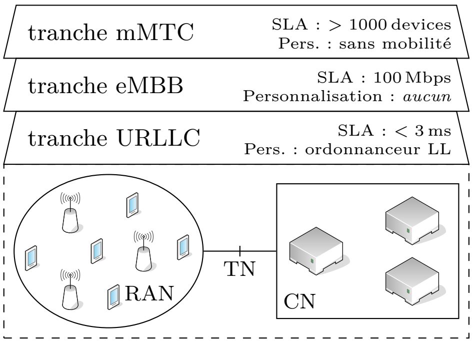


Figure D.1 – Le concept de découpage en tranches (*slicing*) : plusieurs tranches sont superposées à une infrastructure commune.

Les cas d'utilisation des verticaux peuvent être associés à ces scénarios d'utilisation, et l'UIT a proposé un certain nombre d'exigences, telles que la multiplication par 10 du débit expérimenté par l'utilisateur (principalement eMBB), une latence 10 fois plus faible (principalement URLLC), ou une densité de connexion 10 fois supérieure (principalement mMTC), pour l'IMT-2020.

Cependant, les exigences des cas d'utilisation dans ces scénarios d'utilisation sont partiellement incompatibles entre elles. En conséquence, le concept de découpage du réseau (*slicing*) [4] est apparu pour prendre en charge plusieurs services ayant des exigences hétérogènes sur une infrastructure partagée. Grâce au découpage du réseau, des réseaux virtuels sont créés pour chaque service sur une infrastructure sous-jacente commune, comme le montre la Figure D.1. Les réseaux virtuels, appelés tranches, créent des réseaux de bout en bout spécifiques au service qui adaptent l'infrastructure aux exigences du service dans le réseau d'accès radio (RAN), le réseau central (CN) et le réseau de transport (TN). La prise en charge du découpage du réseau a été identifiée dès le départ comme une exigence essentielle pour assurer l'adaptabilité nécessaire des réseaux 5G [5]. Dans le système 5G défini par le *Third Generation Partnership Project* (3GPP), si la personnalisation et le contrôle sont relativement simples dans le CN, le RAN est logiquement une fonction de réseau unique [8] et le découpage en tranches reste spécifique à l'implémentation [9]. Bien qu'une personnalisation en fonction des différents scénarios d'utilisation soit prévue, il n'existe aucun moyen pour les propriétaires de services, tels que les verticaux, de contrôler leur réseau virtuel. Pour permettre aux propriétaires de services d'exercer un tel contrôle, il est nécessaire de concevoir un RAN *orienté services* qui soit non seulement découpé de manière flexible, mais dans lequel les propriétaires de services obtiennent également le niveau de contrôle requis sur leurs réseaux virtuels tout en étant isolés les uns des autres.

D.1.2 Motivation et contribution de la thèse

L'introduction de la flexibilité nécessaire à un RAN orienté services est un défi, car il s'agit de la partie la plus complexe de l'ensemble du réseau et elle joue un rôle important dans la réalisation des exigences formulées par l'UIT. La technologie d'accès radio (RAT) de la 5G, appelée *New Radio* (NR), est conçue selon des principes clés tels que la *flexibilité*, la *compatibilité avec l'avenir* et l'*ultra-légèreté*, afin de pouvoir s'adapter aux cas d'utilisation nouveaux et futurs. Elle s'accompagne d'outils technologiques tels que les ondes millimétriques (*millimeter wave*) ou la densification du réseau, qui promettent de répondre aux exigences des cas d'utilisation, mais nécessitent un contrôle et une coordination accrus. Le concept de désagrégation du RAN, dans lequel le RAN est divisé en fonctions de réseau pour permettre la centralisation, permet d'accroître la coordination et de réaliser des gains de multiplexage, mais augmente à son tour la complexité. La softwarisation et la virtualisation sont également considérées comme des facteurs clés pour les réseaux 5G [16]. La séparation du plan de contrôle et du plan utilisateur par la softwarisation permet la programmabilité, est bien établie [20] et est réalisée grâce au réseau d'accès radio défini par logiciel (SD-RAN). La virtualisation permet de faire fonctionner les RAN sur du matériel standard.

En résumé, les réseaux 5G vont au-delà d'une nouvelle radio et d'un spectre plus large ; ils concernent l'évolution de l'informatique pour les réseaux sans fil afin de prendre en charge une variété de services et de capacités dans divers scénarios d'utilisation. Cependant, nous observons que, bien que beaucoup de travail ait été fait pour atteindre les exigences de performance hétérogène, il y a encore un manque de programmabilité flexible dans le RAN pour permettre une surveillance, un contrôle et une coordination indépendants sur une base par service. À cet égard, nous identifions trois défis clés qui doivent être relevés pour atteindre la vision d'un RAN *orienté services.*

Défi 1 : **Abstraction de la station de base.** Il est primordial de permettre une définition légère et personnalisable des services RAN sur une infrastructure commune afin d'adapter le RAN aux exigences des services et de permettre la programmabilité de plusieurs services, tout en masquant la complexité inutile du RAN. À cette fin, l'infrastructure désagrégée doit être abstraite et représentée vers les services de manière simplifiée, et de multiples contrôleurs spécifiques aux services doivent être autorisés.

Défi 2 : **Infrastructure SD-RAN selon les principes de la 5G.** Comme cela a été décrit, la 5G-NR offre une flexibilité sans précédent dans le RAN pour prendre en charge divers cas d'utilisation dans un environnement multi-locataires. En général, les approches SD-RAN existantes n'adhèrent pas aux principes de la 5G, car elles n'offrent pas la flexibilité et la composabilité que l'on trouve dans l'infrastructure RAN sous-jacente, ou bien elles se présentent comme de grandes solutions « taille unique », violant ainsi le principe d'ultra-légèreté. En outre, ils ne permettent pas la réalisation des abstractions mentionnées.

Défi 3 : **Ordonnancement MAC 5G tenant compte des tranches.** Les réseaux 5G sont conçus (a) pour fonctionner dans un certain nombre de scénarios de déploiement différents, par exemple, en dessous de 6 GHz et au-dessus, (b) pour rester compatibles avec les nouveaux cas d'utilisation, et (c) pour gérer de multiples services. Par conséquent, l'ordonnanceur doit être conçu pour prendre en charge les exigences correspondantes.

Dans cette thèse, nous étudions les concepts de SD-RAN et de découpage en tranches afin de fournir de la flexibilité et de la programmabilité dans l'infrastructure RAN pour réaliser des RAN orientés services tout en étant capable d'adapter le réseau aux

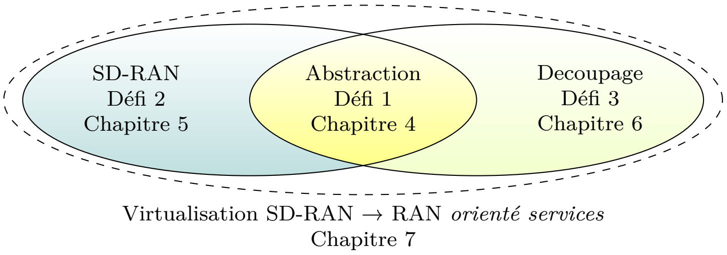


Figure D.2 – Mise en correspondance des chapitres de la thèse avec les défis.

exigences des services et de faire face à la complexité qui l'accompagne. La thèse est structurée autour des défis présentés dans la figure D.2. À cette fin, nous (1) étudions les abstractions pour simplifier le RAN et intégrer les services (défi 1, Chapitre 4, résumé dans la Section D.2), (2) concevons une nouvelle plate-forme SD-RAN qui suit les principes de conception de la 5G (défi 2, Chapitre 5, résumé dans la Section D.3), (3) présentons un cadre d'ordonnancement de la couche *Medium Access Control* (MAC) pour fournir une flexibilité de déploiement, une programmabilité et une extensibilité dans une infrastructure multiservice (défi 3, Chapitre 6, résumé dans la Section D.4), et nous nous appuyons sur ces travaux pour enfin (4) proposer une couche de virtualisation SD-RAN pour mettre en œuvre un RAN orienté services qui permet à plusieurs contrôleurs spécifiques aux services appartenant à différents propriétaires ou opérateurs de services de contrôler leur réseau virtuel (chapitre 7, résumé dans la Section D.5, qui intègre les travaux de tous les chapitres précédents).

D.2 Chapitre 4 : Abstraction de la station de base

Un certain nombre de tendances conduisent à des RANs de plus en plus complexes. Premièrement, les déploiements de stations de base (BS) désagrégées permettent de centraliser le traitement du RAN dans des nuages (*cloud*) centralisés ou en périphérie et de composer de manière flexible les fonctionnalités des BS, mais compliquent la gestion des BS individuelles. Deuxièmement, la densification du réseau permet d'augmenter la capacité de trafic de la zone et la densité de connexion, mais nécessite une meilleure coordination dans le RAN entre les stations de base et entre les services. Troisièmement, la 5G ne fournit pas seulement une RAT spécifique mais utilise également les bandes de fréquences disponibles (par exemple, les ondes millimétriques) et les technologies d'accès (par exemple, 4G, 5G) de manière hétérogène pour répondre à diverses exigences de service. Par conséquent, la prolifération des RAT entraîne une complexité accrue de la gestion du réseau, car les équipements des utilisateurs (UE) doivent être gérés à travers ces RAT. Au total, nous observons que le RAN 5G doit servir de multiples dimensions, comme le montre la Figure D.3a, ce qui augmente la complexité du contrôle et de la gestion. De plus, la complexité est exacerbée si l'on considère que les réseaux 5G doivent multiplexer une variété de services avec des exigences de performance et de personnalisation hétérogènes. En ce qui concerne les verticaux, ils souhaitent fournir leur réseau sans avoir à exploiter un réseau à part entière, tout en conservant le contrôle de

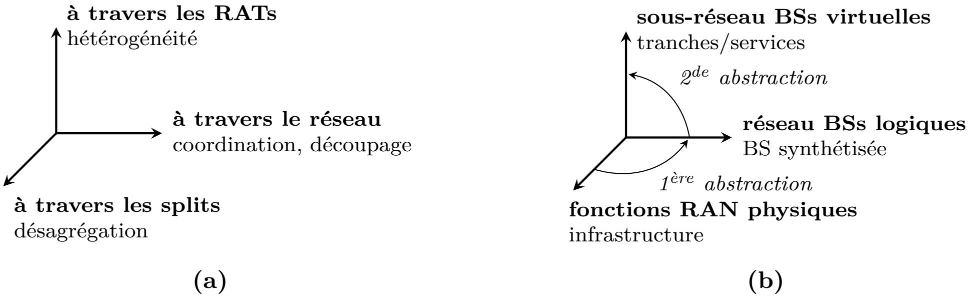


Figure D.3 – Le réseau 5G peut être déconstruit en plusieurs dimensions : (a) une vue split-network-RAT, se généralisant en (b) dimensions physique, logique et virtuelle.

leur réseau virtuel.

Pour faire face à cette complexité, nous envisageons un cadre qui simplifie et abstrait le RAN pour gérer efficacement cette complexité. Comme le montre la figure D.3b, l'infrastructure physique utilisant des divisions fonctionnelles et diverses RATs peut être abstraite pour composer une station de base logique (lBS) couvrant toute la pile de protocoles, par exemple un *evolved Node B* 4G (eNB) ou un *next-generation Node B* 5G (gNB). Un tel lBS désigne une représentation « synthétisée » du réseau physique et peut être considéré comme un outil permettant une gestion et un contrôle efficaces du réseau en réduisant la complexité des déploiements hétérogènes (par exemple, distribués, désagrégés, centralisés). Cette structure de réseau local est encore plus abstraite pour fournir un sous-réseau virtualisé spécifique aux tranches de réseau, tout en maintenant la programmabilité du plan utilisateur. Un tel sous-réseau virtuel est formé à partir d'un ensemble de stations de base virtuelles (vBS) pour révéler des ressources et des personnalisations de traitement spécifiques à la tranche, correspondant aux exigences du service. Grâce à une abstraction légère et personnalisable du point de contrôle du RAN, les personnalisations et des extensions spécifiques aux tranches peuvent ensuite être incorporées dans l'infrastructure, ce qui permet aux propriétaires de services de contrôler leurs sous-réseaux virtuels tout en étant isolés les uns des autres.

Nous conceptualisons le RAN par le biais de descripteurs qui simplifient la description du RAN, et définissent les ressources requises et les personnalisations de traitement des services intégrés. Ils sont liés à l'infrastructure en utilisant l'abstraction à deux niveaux. Les descripteurs définissent les lBS et les vBS à l'aide d'une représentation comprenant les ressources (quantité de ressources radio), l'état (configuration, statistiques) et le traitement (fonctionnalité du RAN qui peut être personnalisée/étendue). Après l'abstraction de premier niveau, un lBS est représenté comme un *descripteur de pipeline*. En outre, un *descripteur de service* permet de décrire les exigences de service et les personnalisations de tranche correspondant à la vBS après l'abstraction de deuxième niveau. Des propriétés telles que l'isolation des performances, le partage, la personnalisation, etc. doivent être garanties lors de la mise en correspondance des services avec le RAN. En outre, le descripteur doit être compatible en amont et en aval, de sorte que les services puissent être mappés sur différents RAT et déploiements, pour autant que toutes les exigences soient satisfaites.

L'abstraction à deux niveaux et les descripteurs qui en résultent sont détaillés à l'aide le cadre FlexVRAN. FlexVRAN interagit avec l'infrastructure RAN qui est annotée avec (une partie d') un descripteur. FlexVRAN effectue ensuite l'abstraction de premier niveau en (1) fusionnant les descripteurs en un lBS, et (2) en créant une vue de la topologie du réseau. Cette lBS est annotée avec le descripteur de pipeline, et pour tenir compte de l'hétérogénéité du système sous-jacent, elle pourrait être représentée comme une collection de modèles de service, « synthétisant » les capacités des fonctions RAN dans l'infrastructure avec leurs ressources et leur état. En outre, FlexVRAN effectue l'abstraction de deuxième niveau en (1) divisant (personnalisable par tranche) les lBS en vBS et (2) en les intégrant dans la topologie logique du réseau afin de révéler une vue personnalisée du réseau virtuel (vRAN) au propriétaire du service. Le fractionnement consiste à n'exposer qu'une partie des ressources, de l'état et du traitement de la lBS sous-jacente à chaque vBS sous une forme virtualisée, où le mécanisme de virtualisation spécifique varie et dépend de la couche de la pile de protocoles.

Si FlexVRAN développe l'abstraction à deux niveaux pour simplifier le traitement des déploiements RAN complexes et désagrégés grâce à une représentation unifiée, il n'explique pas comment les services peuvent personnaliser ou étendre le traitement. Le cadre du RAN Engine (cf. figure D.4) conceptualise le mappage et le multiplexage des services sur l'infrastructure RAN, tout en permettant aux propriétaires de services de surveiller, configurer, personnaliser et étendre les tranches, sous réserve du contrôle d'accès. A cette fin, le *service engine* (moteur de services) permet la personnalisation du pipeline par le biais de kits de développement de services micro (micro-SDK), qui fournissent des abstractions de plan de contrôle (CP) atomiques, indépendantes et extensibles, et mettent en œuvre une interface abstraite vers les services de personnalisation et d'extension du RAN, permettant une virtualisation légère du CP de la BS si nécessaire. L'ensemble des micro-SDK d'un pipeline RAN regroupe les interfaces de programmation d'application (API) du plan de contrôle pour les services de manière indépendante du déploiement et du fournisseur, et peut être classé dans les dimensions ressources radio, traitement des flux et contrôle radio. En combinant les micro-SDK, le descripteur de service marque les intentions de personnalisation pour ce service, et fournit soit la logique d'exécution sous forme binaire, soit un point de terminaison pour l'interaction par message. Ceci est pris en charge par des messages d'action-événement, des commandes directes provenant du service, ou l'intégration d'une logique personnalisée à l'intérieur ou à proximité du moteur (« pousser »). Comme le montre la Figure D.4, un service peut fournir des microservices pour l'équilibrage de la charge de mobilité (MLB), le contrôle du trafic (TC) et l'ordonnancement (Sched) afin d'étendre le traitement RAN et de piloter son service.

Le nombre de lBS qui peuvent être composées de fonctions RAN (unité centralisée, CU, unité distribuée, DU) ne dépend pas seulement du RAN mais aussi des ressources TN. Par exemple, le TN doit supporter les débits de données que les ressources radio du RAN offrent. Par conséquent, le chapitre est complété par la formulation d'un problème concernant la composition des lBS à partir des DU et des CU en tenant compte de la largeur de bande spectrale disponible ainsi que des contraintes de capacité de midhaul et de backhaul. Nous montrons que ce problème, même sous des contraintes relaxées, est

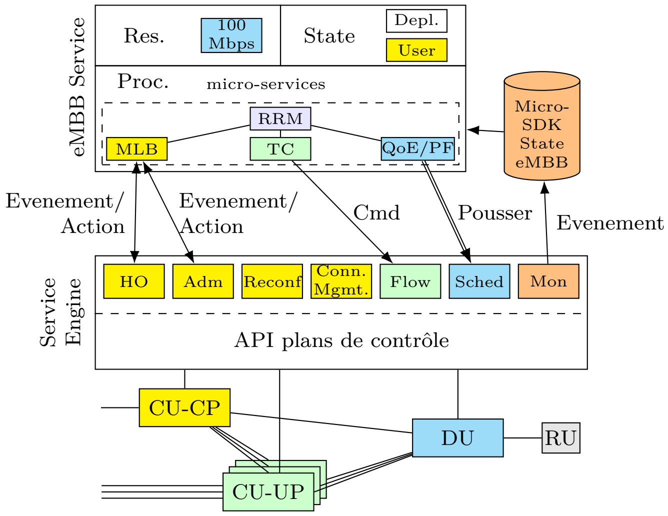


Figure D.4 – Les services multiplexent leurs personnalisations de microservices par le biais du moteur de service sur l'infrastructure RAN sous-jacente en faisant correspondre les micro-SDK. En fonction du micro-SDK, un échange basé sur des messages peut être utilisé (par exemple, pour le contrôle radio) ou la fonctionnalité sera poussée dans le moteur pour des raisons de contraintes de temps (programmation des ressources radio). Les informations sur le débit sont utilisées pour réserver les ressources dans le RAN, et l'état est partitionné par service.

NP-difficile, et résolvons une version simplifiée analytiquement.

D.3 Chapitre 5 : SDK FlexRIC

Contrairement aux réseaux mobiles précédents, la 5G-NR offre une flexibilité sans précédent dans le RAN pour prendre en charge divers cas d'utilisation dans un environnement multi-service. Dans ce contexte, le besoin de programmabilité et de contrôle par le biais du SD-RAN est bien établi. Cependant, alors que le RAN sous-jacent est conçu pour être ultra-flexible et léger, les contrôleurs SD-RAN existants ne sont pas assez flexibles pour répondre à tous les cas d'utilisation, ou bien ils utilisent une approche « *one-size-fits-all* ».

FlexRAN [33], la première tentative de plateforme SD-RAN en temps réel à des fins de recherche, définit un protocole de contrôle personnalisé de la limite sud qui est couplé au RAT sous-jacent et adapté à des opérations de contrôle spécifiques, ce qui limite son extensibilité. Le contrôleur intelligent RAN (RIC) O-RAN [50] implique une architecture de microservices avec les frais généraux associés (par exemple, une empreinte de ressources importante, des latences accrues) même pour les cas d'utilisation qui n'en ont pas besoin, suivant une conception « *one-size-fits-all* », violant les principes centraux de la 5G tels que la flexibilité et la conception ultra-légère. Nous constatons qu'il n'existe toujours pas de conception SD-RAN appropriée qui respecte les principes de conception fondamentaux de la 5G et qui soit suffisamment souple pour aider à mettre en œuvre les abstractions décrites à la Section D.2.

FlexRIC est un kit de développement logiciel (SDK) flexible et efficace qui permet de

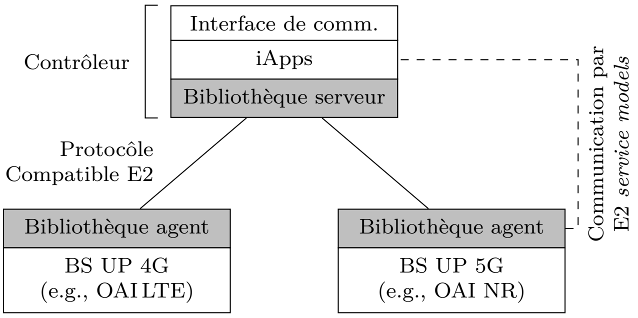


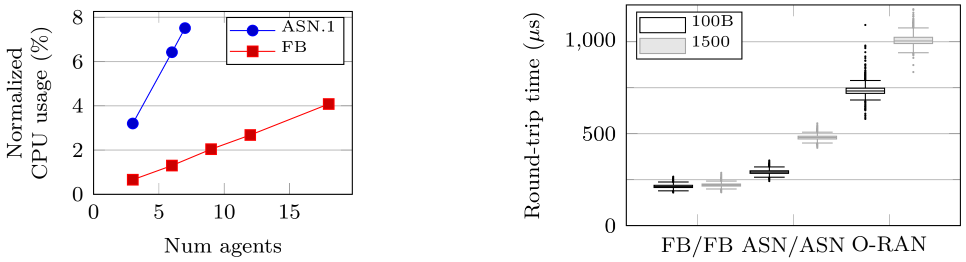
Figure D.5 – Le SDK FlexRIC se compose d'une bibliothèque d'agent et d'une bibliothèque de serveur. La bibliothèque d'agents permet l'intégration d'une station de base à un contrôleur, qui est spécialisé dans un cas d'utilisation particulier en utilisant des iApps. particulier en utilisant des iApps.

construire des contrôleurs SD-RAN spécialisés, et qui est conforme aux principes de la 5G. L'objectif du SDK est de faciliter la réalisation de contrôleurs SD-RAN spécialisés pour cibler des cas d'utilisation spécifiques, tout en étant simple à utiliser. FlexRIC ne prend pas en charge de fonctionnalités supplémentaires de manière endogène, conformément au « *zero-overhead principle* ». Il se compose d'une bibliothèque de serveur et d'une bibliothèque d'agent. Dans sa forme la plus simple, le SDK peut être utilisé pour mettre en œuvre un contrôleur SD-RAN utilisant un protocole compatible E2, comme le montre la Figure D.5.

La bibliothèque d'agents est la base pour étendre une station de base avec les fonctionnalités d'agent afin de l'intégrer à un contrôleur E2. Elle fournit une API pour mettre en œuvre des fonctions RAN personnalisées, c'est-à-dire des fonctions RAN qui peuvent être surveillées et/ou contrôlées par des applications, et est fournie avec un ensemble de fonctions RAN prédéfinies qui mettent en œuvre un ensemble d'E2SM qui peuvent être inclus. En outre, il permet de gérer plusieurs contrôleurs.

Un contrôleur est construit à partir de la bibliothèque serveur, des applications internes (iApps), et éventuellement d'une interface de communication. La bibliothèque serveur gère les connexions des agents et achemine les messages entre les iApps et les agents. Grâce aux iApps, il est possible de construire de manière modulaire des contrôleurs spécialisés : en fonction du cas d'utilisation considéré, les iApps peuvent mettre en œuvre des E2SM elles-mêmes, ou exposer des informations à des applications externes (xApps) via une interface de communication nord. Dans ce dernier cas, les xApps peuvent contrôler le RAN tout en étant fonctionnellement isolées du contrôleur, ce qui est la méthode privilégiée par le RIC O-RAN, mais entraîne une certaine surcharge. Il convient de souligner que la bibliothèque serveur elle-même ne met en œuvre aucun E2SM et ne demande aucune information à l'agent par elle-même. Au lieu de cela, les iApps doivent déclencher la communication liée au modèle de service, et la bibliothèque serveur fournit la plate-forme pour multiplexer les messages entre les agents et les iApps.

Le SDK FlexRIC fournit une abstraction de l'interface E2 via une représentation interne des messages E2. Il est donc facile d'intégrer FlexRIC à une infrastructure SD-RAN préexistante conforme à la norme E2. Toutefois, la norme impose l'encapsulation



(a) Utilisation du CPU de FlexRIC.

(b) RTT pour deux sauts vs. O-RAN RIC.

Figure D.6 – L'encodage E2AP peut être modifié pour s'adapter au cas d'utilisation. FlexRIC est plus rapide que le RIC O-RAN.

des données codées ASN.1 (E2SM) au sein de messages codés ASN.1 (E2AP), ce qui peut s'avérer inefficace dans certains cas (voir ci-dessous). Par conséquent, le SDK permet également de modifier à la fois le schéma d'encodage et le protocole de transport, ce qui permet d'intégrer d'autres protocoles E2 pour un contrôle en temps réel à faible coût.

En particulier, nous avons mis en œuvre le codage E2AP à la fois en ASN.1 et en Flatbuffers (FB) [118]. L'ASN.1 permet de réduire la taille des messages au prix d'une surcharge de codage plus élevée, tandis que Flatbuffers permet d'obtenir des messages comparativement plus grands avec un codage plus rapide. La surcharge de codage est mesurable, comme le montre la Figure D.6a. Ici, nous mesurons l'utilisation du CPU pour une information statistique provenant de plusieurs agents de station de base. L'ASN.1 entraîne une utilisation plus importante du CPU et limite donc le nombre de stations de base pouvant être prises en charge. Ainsi, les Flatbuffers peuvent être préférés pour les cas d'utilisation impliquant de nombreuses stations de base et un échange fréquent de messages. D'autre part, le temps de propagation aller-retour (RTT) des messages, comme le montre la Figure D.6b, est toujours plus faible pour FlexRIC que pour le RIC O-RAN, ce qui le rend particulièrement utile pour les cas d'utilisation à faible latence.

La conception modulaire de FlexRIC permet de composer facilement des contrôleurs spécialisés au-dessus de la bibliothèque de serveurs grâce aux iApps, aux xApps et à la sélection d'E2SM. En outre, sa conception ouvre de nouvelles possibilités pour le développement de contrôleurs de recherche de pointe (par exemple, des contrôleurs récursifs ou multi-RAT). Dans ce contexte, nous développons des contrôleurs SD-RAN spécialisés et spécifiques aux services pour le découpage en tranches (voir Section D.4), le contrôle du trafic et l'équilibrage des charges dans le réseau. En outre, en tirant parti de FlexRIC, nous détaillons les conceptions d'un contrôleur générique de type FlexRAN [33], d'un outil simple de surveillance du RAN, d'un contrôleur compatible O-RAN et d'Orion [53]. Il est également possible d'exposer de manière récursive une interface d'agent au niveau de la liaison nord pour une couche de virtualisation SD-RAN, comme le détaille la Section D.5. Enfin, nous détaillons également la manière d'incorporer un « magasin de réseau » (*network store*) pour étendre le RAN avec des fonctions de réseau personnalisées.

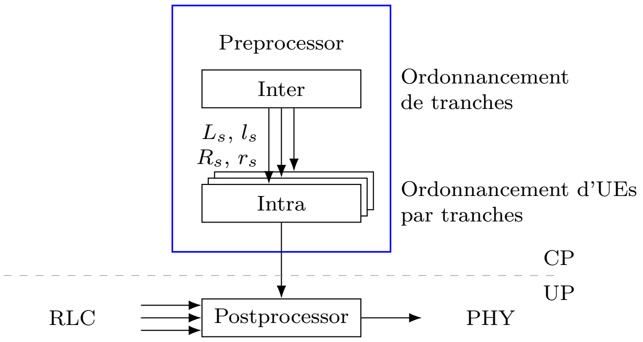


Figure D.7 – Décomposition de l'ordonnancement en *preprocessor* et *postprocessor* et extension multi-services avec les ordonnanceurs inter-tranche et intra-tranche.

D.4 Chapitre 6 : Cadre flexible de la couche MAC

La 5G est envisagée comme un réseau ultra-flexible et ultra-léger qui peut être adapté à différents scénarios de déploiement et qui est censé multiplexer une variété de services aux exigences hétérogènes sur une infrastructure commune et partagée. Un point crucial est de gérer cette flexibilité et d'adapter la façon dont les ressources radio sont programmées en fonction des scénarios de déploiement envisagés et des services actuellement transportés. L'ordonnanceur MAC revêt une importance particulière, car il est le « cerveau » de l'allocation des ressources radio à court terme.

Pour permettre l'adaptation du pipeline d'ordonnancement à différents scénarios de déploiement et la prise en charge de divers services, nous découplons les phases d'ordonnancement des utilisateurs de la liaison montante et de la liaison descendante en un *preprocessor* et un *postprocessor*, comme le montre la Figure D.7. Le preprocessor est un module échangeable qui prend les décisions d'ordonnancement, tandis que le postprocessor applique les décisions du preprocessor au plan utilisateur. Le preprocessor est adapté au cas d'utilisation, y compris les connaissances sur les plages de fréquences, le mode de fonctionnement, le multiplexage des services, etc. À l'aide d'informations sur les cellules (par exemple, le multiplexage temporel (TDD) et les *slot formats*), d'informations locales propres à l'utilisateur (par exemple, l'état de la mémoire tampon) et d'informations de retour (par exemple, HARQ Ack/Nacks), il prend des décisions d'ordonnancement et transmet les informations correspondantes, par exemple, l'allocation de blocs de ressources et le schéma de modulation et de codage (MCS), au postprocessor.

En outre, une extension multiservice du preprocessor le décompose en un ordonnanceur inter-tranche (*inter-scheduler*) et un ordonnanceur intra-tranche (*intra-scheduler*). Le premier travaille au niveau des groupes d'utilisateurs, c'est-à-dire des tranches, et décide de l'allocation des ressources selon une approche à deux niveaux [54]. En fonction du résultat de la décision régie par un algorithme de tranche, l'intra-ordonnanceur est appelé avec les ressources éligibles R_s et le nombre maximum r_s, et une liste d'utilisateurs (UE) L_s et le nombre maximum l_s à ordonnancer. Ensuite, l'intra-ordonnanceur planifie les UE individuels selon un algorithme d'ordonnancement des utilisateurs.

Ce cadre MAC a été mis en œuvre dans OpenAirInterface (OAI) [22], [23] pour la

4G/Long-Term Evolution (LTE) et la 5G/NR. Dans les deux cas, un certain nombre d'algorithmes d'ordonnancement de tranches et d'utilisateurs ont été mis en œuvre, à savoir les algorithmes de tranches *static slicing*, NVS [24] et « SCN19 » (voir ci-dessous), et les ordonnanceurs d'utilisateurs *round-robin*, *proportional-fair* et *maximum-throughput*. Dans le cas de 5G/NR, l'ordonnanceur a été mis en œuvre à partir de zéro, car l'OAI ne disposait pas d'un ordonnanceur. Grâce au E2SM Slicing, le preprocessor, et plus précisément les algorithmes de découpage et les tranches présentss, y compris les paramètres de découpage spécifiques à l'algorithme tels que les ressources ou les taux, peuvent être configurés pour être utilisés avec un contrôleur compatible E2, tel que FlexRIC.

Un algorithme d'ordonnancement de tranches doit prendre en compte plusieurs tranches ayant des exigences hétérogènes. Nous avons observé que les algorithmes précédents ne tiennent pas compte de trois types d'exigences clés en matière de ressources : les ressources physiques (statiques), par exemple pour les services d'urgence, le débit, par exemple pour les services eMBB, et la latence, par exemple pour les services URLLC. Nous développons un algorithme de tranches tenant compte de la qualité de service (appelé « SCN19 ») en formulant des fonctions d'utilité, et proposons un algorithme d'ordonnancement basé sur le poids pour ordonnancer les tranches avec l'une des trois exigences de ressources. Cet algorithme est en outre intégré dans le cadre MAC, ce qui montre son extensibilité.

Le cadre d'ordonnancement MAC, y compris l'algorithme d'ordonnancement des tranches en fonction de la qualité de service, est évalué à l'aide d'une simulation, d'une émulation sur le « L2 simulator » de l'OAI 4G/LTE et d'une configuration radio sur la plateforme OAI 5G/NR avec des téléphones disponibles dans le commerce. Avec le simulateur, nous étudions les principales exigences des tranches, telles que l'isolation et la personnalisation, et nous évaluons en détail l'algorithme d'ordonnancement des tranches en fonction de la qualité de service. L'émulateur permet d'évaluer l'implémentation du cadre d'ordonnancement MAC sur OAI dans un environnement contrôlé. Il permet également d'émuler de nombreux UE. Comme le montre la Figure D.8, l'implémentation est scalable. Tout d'abord, 8 tranches sont déployées pour 8 UEs, et les ressources sont partagées de manière égale. Lorsque 42 UE supplémentaires se connectent, l'algorithme de tranche applique l'isolation des ressources radio entre les tranches, de sorte que le débit de l'application des tranches individuelles reste le même.

Pour la 5G/NR, nous avons mis en œuvre l'ordonnanceur multi-utilisateurs et sensible aux tranches à partir de zéro selon le cadre d'ordonnancement MAC, et nous soulignons les mesures suivantes. Comme l'un des principaux objectifs de la NR est d'atteindre une faible latence, nous mesurons le RTT du CN à l'UE via le RAN et retour en utilisant des paquets *ping* de taille de 64 octets envoyés toutes les 200 ms. Nous montrons la fonction de distribution cumulative (CDF) dans la Figure D.9a pour différents modèles de téléphones disponibles dans le commerce. Nous observons que la plupart des modèles présentent un RTT inférieur à celui des réseaux LTE (minimum 15 ms). Le comportement du Pixel 5 est surprenant, car il utilise le même chipset que le téléphone Oppo. De plus, en ce qui concerne la mesure "Quectel modif.", l'UE est programmé dans chaque slot de liaison montante, de sorte qu'il n'a pas à demander un octroi de transmission, ce qui réduit

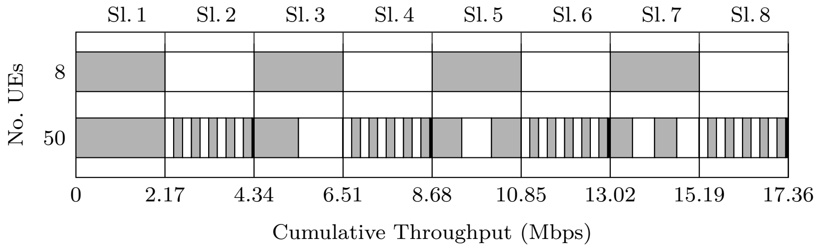
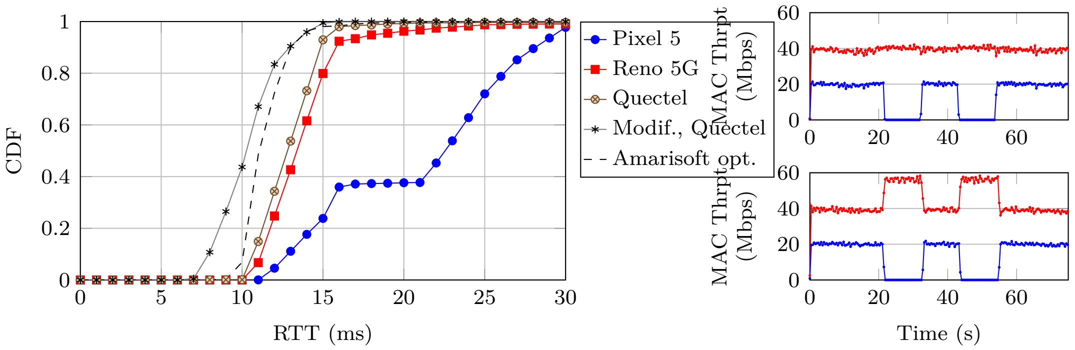


Figure D.8 – Isolation des ressources et scalabilité pour le découpage en tranches. Ci-dessus : Huit tranches avec huit utilisateurs ont le même débit. En bas : Avec des utilisateurs supplémentaires, les débits par tranche restent les mêmes.



(a) RTT des paquets *ping* sur le RAN pour différents modèles d'UE.

(b) Isolation et partage des ressources.

Figure D.9 – Mesures sur le déploiement radio 5G/NR.

encore les RTTs. Enfin, la Figure D.9b montre les propriétés d'isolation et de partage des tranches appliquées dans la 5G/NR. Premièrement, chaque tranche suit la part de ressources qui lui est attribuée, soit 33 % et 66 %. Ensuite, le partage des ressources permet d'augmenter les performances d'une tranche si une autre tranche ne l'utilise pas, ce qui maximise l'utilisation des ressources et entraîne un gain de multiplexage.

D.5 Chapitre 7 : Virtualisation du SD-RAN

La densification croissante du RAN entraîne une augmentation des coûts des réseaux cellulaires, car il faut déployer davantage de stations de base. Le *partage* du RAN permet de réduire les coûts, puisque plusieurs opérateurs partagent une infrastructure commune au lieu de maintenir des infrastructures dédiées. Dans ce chapitre, nous présentons comment une station de base peut être partagée entre plusieurs opérateurs, tout en donnant aux deux opérateurs la possibilité de *découper en tranches* leurs réseaux individuels en utilisant leurs contrôleurs SD-RAN respectifs. À cette fin, nous développons une couche de virtualisation SD-RAN qui fonctionne comme un intermédiaire pour virtualiser le contrôle de découpage, assurant l'isolation entre les opérateurs tout en exposant l'accès

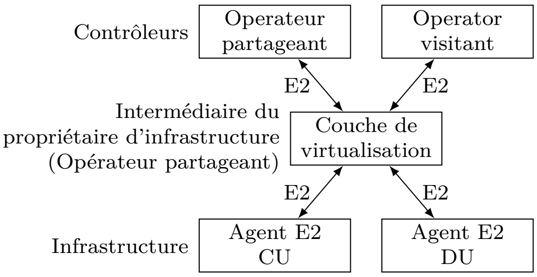


Figure D.10 – Deux opérateurs contrôlent simultanément le découpage en tranches dans la même infrastructure. Une couche de virtualisation assure l'isolation nécessaire entre les opérateurs.

au contrôle à leurs contrôleurs, comme le montre la Figure D.10.

Cette couche de virtualisation SD-RAN expose *récursivement* l'interface E2 à sa limite nord. Pour les agents E2 (situés au sud), elle apparaît comme un contrôleur E2 ; pour les contrôleurs E2 (situés au nord), elle apparaît comme un agent E2. Comme E2 est exposé de manière récursive, il est possible de construire des hiérarchies de contrôleurs, où les contrôleurs des couches supérieures exécutent des tâches supérieures. La couche de virtualisation prend nativement en charge la désagrégation grâce aux capacités de la bibliothèque de serveur, et simplifie l'infrastructure vers les contrôleurs liés au nord, comme le montre également la Figure D.10.

La couche de virtualisation exploite le SDK FlexRIC présenté à la section D.3. Elle utilise la bibliothèque de serveur pour gérer plusieurs agents au sud, et utilise la bibliothèque d'agent comme interface de communication pour exposer le contrôle E2 virtualisé aux contrôleurs SD-RAN au nord. Dans ce contexte, les iApps utilisées pour spécialiser un contrôleur prennent le rôle de fonctions RAN E2 vers la bibliothèque d'agent.

Comme les ressources pouvant être découpées en tranches dans une station de base sont très hétérogènes, comprenant le contrôle de flux pour plusieurs supports radio, le contrôle de connexion radio et les ressources radio, la virtualisation exacte est spécifique aux E2SMs utilisés. Pour être plus concret, nous utilisons l'E2SM Slicing utilisé pour contrôler le cadre d'ordonnancement MAC à la Section D.4. La couche de virtualisation abstrait les ressources radio attribuées et l'identifiant de tranche d'une représentation physique (vers le sud) à une représentation virtuelle (vers le nord). Les identifiants sont mis en correspondance pour éviter les conflits. Pour la configuration des ressources radio, nous utilisons l'algorithme de tranche NVS [24]. Les ressources des tranches sont mises à l'échelle par une part de ressources spécifique à l'opérateur pour fournir la part de ressources virtualisée pour une station de base virtuelle, dont la somme est égale à 1.

Nous avons mis en œuvre un prototype de la couche de virtualisation en utilisant le SDK FlexRIC sur la plateform OAI. Les comparaisons de l'utilisation de la mémoire indiquent une diminution de près de 50 % des ressources pour la configuration considérée en utilisant le « L2 simulator »[1], qui provient de l'utilisation d'une seule infrastructure

[1] Ce mode d'émulation n'utilise pas la couche physique. Pour les déploiements radio, l'économie de mémoire serait d'environ 30 %.

au lieu de deux. En outre, nous avons réalisé une expérience avec deux opérateurs A et B et un partage égal des ressources de l'infrastructure. Le gain de multiplexage des débits s'élève jusqu'à 100 % si un opérateur est inactif.

References

[1] "Cisco annual internet report (2018–2023)", Cisco, White paper, Mar. 2020.

[2] "Imt vision – framework and overall objectives of the future development of imt for 2020 and beyond, Recommendation itu-r m.2083-0", ITU-R, Tech. Rep., Sep. 2015.

[3] "Minimum requirements related to technical performance for imt-2020 radio interface(s), Report itu-r m.2410-0", ITU-R, Tech. Rep., Nov. 2017.

[4] P. Hedman, "Description of network slicing concept", NGMN Alliance, white paper, Jan. 13, 2016.

[5] P. Marsch, I. D. Silva, O. Bulakci, *et al.*, "5g radio access network architecture: Design guidelines and key considerations", *IEEE Communications Magazine*, vol. 54, no. 11, pp. 24–32, Nov. 2016.

[6] "Detailed specifications of the terrestrial radio interfaces of international mobile telecommunications-2020 (imt-2020), Recommendation itu-r m.2150-0", ITU-R, Tech. Rep., Feb. 2021.

[7] 3GPP, "System architecture for the 5g system (5gs)", 3GPP, TS 23.501 V15.12.0, Jan. 2021.

[8] ——, "Ng-ran; architecture description", 3GPP, TS 38.401 V15.9.0, Nov. 2020.

[9] ——, "Nr and ng-ran overall description", 3GPP, TS 38.300 V15.12.0, Apr. 2021.

[10] "Network 2030 architecture framework, Fg net-2030 technical specification", ITU-T, Tech. Rep., Jun. 2020.

[11] E. Dahlman, S. Parkvall, and J. Sköld, *5G NR: the Next Generation Wireless Access Technology.* Oxford: Academic Press, 2021.

[12] M. Shafi, A. F. Molisch, P. J. Smith, *et al.*, "5g: A tutorial overview of standards, trials, challenges, deployment, and practice", *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 6, pp. 1201–1221, Jun. 2017.

[13] "C-ran: The road towards green ran", China Mobile, Tech. Rep., version version 2.5, Oct. 1, 2011.

[14] L. M. P. Larsen, A. Checko, and H. L. Christiansen, "A survey of the functional splits proposed for 5g mobile crosshaul networks", *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 146–172, 2019.

[15] F. Granelli, A. A. Gebremariam, M. Usman, *et al.*, "Software defined and virtualized wireless access in future wireless networks: Scenarios and standards", *IEEE Communications Magazine*, vol. 53, no. 6, pp. 26–34, 2015.

[16] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, *et al.*, "Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges", *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80–87, 2017.

[17] "Transition toward open & interoperable networks", 5G Americas, Whitepaper, Nov. 2020.

[18] S. E. Elayoubi, S. B. Jemaa, Z. Altman, and A. Galindo-Serrano, "5g ran slicing for verticals: Enablers and challenges", *IEEE Communications Magazine*, vol. 57, no. 1, pp. 28–34, Jan. 2019.

[19] C. Guimarães, X. Li, C. Papagianni, *et al.*, "Public and non-public network integration for 5growth industry 4.0 use cases", *IEEE Communications Magazine*, vol. 59, no. 7, pp. 108–114, 2021.

[20] Z. Zaidi, V. Friderikos, Z. Yousaf, *et al.*, "Will sdn be part of 5g?", *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 3220–3258, Oct. 2018.

[21] C. Sexton, N. Marchetti, and L. A. DaSilva, "Customization and trade-offs in 5g ran slicing", *IEEE Communications Magazine*, vol. 57, no. 4, pp. 116–122, 2019.

[22] N. Nikaein, M. K. Marina, S. Manickam, *et al.*, "Openairinterface: A flexible platform for 5g research", *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 33–38, Oct. 2014.

[23] F. Kaltenberger, A. P. Silva, A. Gosain, L. Wang, and T.-T. Nguyen, "Openairinterface: Democratizing innovation in the 5g era", *Computer Networks*, vol. 176, p. 107 284, 2020.

[24] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan, "Nvs: A substrate for virtualizing wireless resources in cellular networks", *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1333–1346, Oct. 2012.

[25] E. Dahlman, S. Parkvall, and J. Sköld, *4G LTE/LTE-Advanced for Mobile Broadband*. Academic Press, 2011.

[26] 3GPP, "Study on new radio access technology: Radio access architecture and interfaces", 3GPP, TR 38.801 V14.0.0, Mar. 2017.

[27] "Fapi and nfapi specifications", Small Cell Forum, SCF 082, version 9.0, May 2017.

[28] "Common public radio interface: Ecpri interface specification", CPRI, eCPRI Specification V1.2, Jun. 2018.

[29] "O-ran fronthaul cooperative transport interface transport control plane specification", O-RAN Alliance, O-RAN.WG4.CTI-TCP.0-v01.00, Apr. 2020.

[30] F. Capozzi, G. Piro, L. A. Grieco, G. Boggia, and P. Camarda, "Downlink packet scheduling in lte cellular networks: Key design issues and a survey", *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 678–700, Apr. 2013.

[31] X. Foukas, M. K. Marina, and K. Kontovasilis, "Software defined networking concepts", in *Software Defined Mobile Networks (SDMN): Beyond LTE Network Architecture*. 2015, pp. 21–44.

[32] N. McKeown, T. Anderson, H. Balakrishnan, *et al.*, "Openflow: Enabling innovation in campus networks", *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[33] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "Flexran: A flexible and programmable platform for software-defined radio access networks", in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16, Irvine, California, USA: Association for Computing Machinery, 2016, pp. 427–441.

[34] "Architecture & e2 general aspects and principles", O-RAN Working Group 3, Technical Specification O-RAN.WG3.E2GAP-v01.01, 2020.

[35] “E2 application protocol (e2ap)”, O-RAN Working Group 3, Technical Specification O-RAN.WG3.E2AP-v01.01, 2020.

[36] “E2 service model (e2sm), kpm”, O-RAN Working Group 3, Technical Specification ORAN-WG3.E2SM-KPM-v01.00.00, 2020.

[37] “E2 service model (e2sm), ran function network interface (ni)”, O-RAN Working Group 3, Technical Specification ORAN-WG3.E2SM-NI-v01.00.00, 2020.

[38] “Network functions virtualisation (nfv); architectural framework”, ETSI, Group Specification NFV 002, v1.2.1, Dec. 2014.

[39] K.-K. Yap, M. Kobayashi, R. Sherwood, *et al.*, “Openroads: Empowering research in mobile networks”, *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 125–126, Jan. 2010.

[40] K.-K. Yap, R. Sherwood, M. Kobayashi, *et al.*, “Blueprint for introducing innovation into wireless mobile networks”, in *Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, ser. VISA ’10, New Delhi, India: Association for Computing Machinery, 2010, pp. 25–32.

[41] R. Sherwood, G. Gibb, K.-K. Yap, *et al.*, “Flowvisor: A network virtualization layer”, Tech. Rep., 2009.

[42] P. Bosshart, D. Daly, G. Gibb, *et al.*, “P4: Programming protocol-independent packet processors”, *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[43] L. E. Li, Z. M. Mao, and J. Rexford, “Toward software-defined cellular networks”, in *2012 European Workshop on Software Defined Networking*, Oct. 2012, pp. 7–12.

[44] M. Yang, Y. Li, D. Jin, *et al.*, “OpenRAN: A software-defined RAN architecture via virtualization”, *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 549–550, Aug. 2013.

[45] I. F. Akyildiz, P. Wang, and S.-C. Lin, “Softair: A software defined networking architecture for 5g wireless systems”, *Computer Networks*, vol. 85, pp. 1–18, 2015.

[46] A. Gudipati, D. Perry, L. E. Li, and S. Katti, “Softran: Software defined radio access network”, in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN ’13, Hong Kong, China: ACM, 2013, pp. 25–30.

[47] T. Chen, H. Zhang, X. Chen, and O. Tirkkonen, “Softmobile: Control evolution for future heterogeneous mobile networks”, *IEEE Wireless Communications*, vol. 21, no. 6, pp. 70–78, Dec. 2014.

[48] E. Coronado, S. N. Khan, and R. Riggio, “5g-empower: A software-defined networking platform for 5g radio access networks”, *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 715–728, 2019.

[49] C.-L. I and S. K. (eds.), “O-ran: Towards an open and smart ran”, O-RAN Alliance, Whitepaper, Oct. 2018.

[50] B. Balasubramanian, E. S. Daniels, M. Hiltunen, *et al.*, “Ric: A ran intelligent controller platform for ai-enabled cellular networks”, *IEEE Internet Computing*, vol. 25, no. 2, pp. 7–17, 2021.

[51] O. Sunay, S. Ansari, S. Condon, *et al.*, “Onf’s software-defined ran platform consistent with the o-ran architecture”, OpenNetworkingFoundation, Whitepaper, Aug. 2020.

[52] OpenNetworkFoundation. (Jun. 25, 2021). Open air interface virtual summer workshop, [Online]. Available: `https://www.openairinterface.org/docs/workshop/2021-06-SUMMER-VIRTUAL-WORKSHOP/PDF/09_SHAD_ANSARI_MTS-OPEN_NETWORKING_FOUNDATION_O-RAN_E2AP_IMPLEMENTATION_IN_OAI.pdf`.

[53] X. Foukas, M. K. Marina, and K. Kontovasilis, "Orion: RAN slicing for a flexible and cost-effective multi-service mobile network architecture", in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '17, Snowbird, Utah, USA: ACM, 2017, pp. 127–140.

[54] A. Ksentini and N. Nikaein, "Toward enforcing network slicing on ran: Flexibility and resources abstraction", *IEEE Communications Magazine*, vol. 55, no. 6, pp. 102–108, 2017.

[55] S. Costanzo, I. Fajjari, N. Aitsaadi, and R. Langar, "A network slicing prototype for a flexible cloud radio access network", in *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2018, pp. 1–4.

[56] E. Coronado and R. Riggio, "Flow-based network slicing: Mapping the future mobile radio access networks", in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, 2019, pp. 1–9.

[57] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "Openradio: A programmable wireless dataplane", in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12, Helsinki, Finland: ACM, 2012, pp. 109–114.

[58] W. Wu, L. E. Li, A. Panda, and S. Shenker, "Pran: Programmable radio access networks", in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIII, Los Angeles, CA, USA: ACM, 2014, 6:1–6:7.

[59] P. Dely, J. Vestin, A. Kassler, *et al.*, "Cloudmac — an openflow based architecture for 802.11 mac layer processing in the cloud", in *2012 IEEE Globecom Workshops*, 2012, pp. 186–191.

[60] L. U. Khan, I. Yaqoob, N. H. Tran, Z. Han, and C. S. Hong, "Network slicing: Recent advances, taxonomy, requirements, and open research challenges", *IEEE Access*, vol. 8, pp. 36 009–36 028, 2020.

[61] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions", *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2429–2453, Sep. 2018.

[62] "Network slicing use case requirements", GSMA, Tech. Rep., Apr. 2018.

[63] R. Su, D. Zhang, R. Venkatesan, *et al.*, "Resource allocation for network slicing in 5g telecommunication networks: A survey of principles and models", *IEEE Network*, vol. 33, no. 6, pp. 172–179, 2019.

[64] F. Fossati, S. Moretti, P. Perny, and S. Secci, "Multi-resource allocation for network slicing", *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1311–1324, 2020.

[65] J. Martín Pérez, F. Malandrino, C.-F. Chiasserini, and C. Bernardos, "Okpi: All-kpi network slicing through efficient resource allocation", Dec. 2019.

[66] W. Guan, X. Wen, L. Wang, Z. Lu, and Y. Shen, "A service-oriented deployment policy of end-to-end network slicing based on complex network theory", *IEEE Access*, vol. 6, pp. 19 691–19 701, 2018.

[67] H. Halabian, "Distributed resource allocation optimization in 5g virtualized networks", *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 627–642, Mar. 2019.

[68] Q. Qin, N. Choi, M. R. Rahman, M. Thottan, and L. Tassiulas, "Network slicing in heterogeneous software-defined rans", in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 2371–2380.

[69] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "AZTEC: Anticipatory Capacity Allocation for Zero-Touch Network Slicing", in *IEEE INFOCOM*, Toronto, Canada, Jul. 2020.

[70] C. Gutterman, E. Grinshpun, S. Sharma, and G. Zussman, "Ran resource usage prediction for a 5g slice broker", in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. Mobihoc '19, Catania, Italy: ACM, 2019, pp. 231–240.

[71] 3GPP, "Architecture enhancements for dedicated core networks", 3GPP, TR 23.707 V13.0.0, Dec. 2014.

[72] ——, "Enhancements of dedicated core networks selection mechanism", 3GPP, TR 23.711 V14.0.0, Sep. 2016.

[73] K. Katsalis, N. Nikaein, E. Schiller, A. Ksentini, and T. Braun, "Network slices towards 5G communications: Slicing the LTE network", *IEEE Communications Magazine, Vol.55, NÂ°8, August 2017*, Aug. 2017.

[74] A. Nakao, P. Du, Y. Kiriha, *et al.*, "End-to-end network slicing for 5g mobile networks", *Journal of Information Processing*, vol. 25, pp. 153–163, 2017.

[75] N. Nikaein, E. Schiller, R. Favraud, *et al.*, "Network store: Exploring slicing in future 5g networks", in *Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture*, ser. MobiArch '15, Paris, France: ACM, 2015, pp. 8–13.

[76] S. Sharma, R. Miller, and A. Francini, "A cloud-native approach to 5g network slicing", *IEEE Communications Magazine*, vol. 55, no. 8, pp. 120–127, 2017.

[77] S. Kukliński, L. Tomaszewski, T. Osiński, *et al.*, "A reference architecture for network slicing", in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, Jun. 2018, pp. 217–221.

[78] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, *et al.*, "The algorithmic aspects of network slicing", *IEEE Communications Magazine*, vol. 55, no. 8, pp. 112–119, Aug. 2017.

[79] J. Prados-Garzon, A. Laghrissi, M. Bagaa, T. Taleb, and J. M. Lopez-Soler, "A complete lte mathematical framework for the network slice planning of the epc", *IEEE Transactions on Mobile Computing*, vol. 19, no. 1, pp. 1–14, Jan. 2020.

[80] X. Costa-Perez, J. Swetina, T. Guo, R. Mahindra, and S. Rangarajan, "Radio access network virtualization for future mobile carrier networks", *IEEE Communications Magazine*, vol. 51, no. 7, pp. 27–35, Jul. 2013.

[81] R. H. Tehrani, S. Vahid, D. Triantafyllopoulou, H. Lee, and K. Moessner, "Licensed spectrum sharing schemes for mobile operators: A survey and outlook", *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2591–2623, Oct. 2016.

[82] O. Sallent, J. Perez-Romero, R. Ferrus, and R. Agusti, "On radio access network slicing from a radio resource management perspective", *IEEE Wireless Communications*, vol. 24, no. 5, pp. 166–174, Oct. 2017.

[83] M. Richart, J. Baliosian, J. Serrat, and J. Gorricho, “Resource slicing in virtual wireless networks: A survey”, *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 462–476, 2016.

[84] R. Mahindra, M. A. Khojastepour, Honghai Zhang, and S. Rangarajan, “Radio access network sharing in cellular networks”, in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, Oct. 2013, pp. 1–10.

[85] C. Chang, N. Nikaein, and T. Spyropoulos, “Radio access network resource slicing for flexible service execution”, in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2018, pp. 668–673.

[86] D. Marabissi and R. Fantacci, “Highly flexible ran slicing approach to manage isolation, priority, efficiency”, *IEEE Access*, vol. 7, pp. 97 130–97 142, 2019.

[87] A. Gudipati, L. E. Li, and S. Katti, “Radiovisor: A slicing plane for radio access networks”, in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN ’14, Chicago, Illinois, USA: ACM, 2014, pp. 237–238.

[88] J. García-Morales, M. C. Lucas-Estañ, and J. Gozalvez, “Latency-sensitive 5g ran slicing for industry 4.0”, *IEEE Access*, vol. 7, pp. 143 139–143 159, 2019.

[89] P. Korrai, E. Lagunas, S. K. Sharma, *et al.*, “A ran resource slicing mechanism for multiplexing of embb and urllc services in ofdma based 5g wireless networks”, *IEEE Access*, vol. 8, pp. 45 674–45 688, 2020.

[90] A. Papa, M. Klugel, L. Goratti, T. Rasheed, and W. Kellerer, “Optimizing dynamic ran slicing in programmable 5g networks”, in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, May 2019, pp. 1–7.

[91] T. Guo and A. Suárez, “Enabling 5g ran slicing with edf slice scheduling”, *IEEE Transactions on Vehicular Technology*, vol. 68, no. 3, pp. 2865–2877, Mar. 2019.

[92] S. Mandelli, M. Andrews, S. Borst, and S. Klein, “Satisfying network slicing constraints via 5g mac scheduling”, in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 2332–2340.

[93] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan, “Cellslice: Cellular wireless resource slicing for active ran sharing”, in *2013 Fifth International Conference on Communication Systems and Networks (COMSNETS)*, Jan. 2013, pp. 1–10.

[94] D. Nojima, Y. Katsumata, T. Shimojo, *et al.*, “Resource isolation in ran part while utilizing ordinary scheduling algorithm for network slicing”, in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, 2018, pp. 1–5.

[95] A. Aijaz, “Hap-slicer: A radio resource slicing framework for 5g networks with haptic communications”, *IEEE Systems Journal*, vol. 12, no. 3, pp. 2285–2296, Sep. 2018.

[96] H. D. R. Albonda and J. Pérez-Romero, “An efficient ran slicing strategy for a heterogeneous network with embb and v2x services”, *IEEE Access*, vol. 7, pp. 44 771–44 782, 2019.

[97] M. R. Raza, C. Natalino, P. Öhlen, L. Wosinska, and P. Monti, “Reinforcement learning for slicing in a 5g flexible ran”, *Journal of Lightwave Technology*, vol. 37, no. 20, pp. 5161–5169, 2019.

[98] Y. Abiko, T. Saito, D. Ikeda, *et al.*, “Flexible resource block allocation to multiple slices for radio access network slicing using deep reinforcement learning”, *IEEE Access*, vol. 8, pp. 68 183–68 198, 2020.

[99] F. Abinader, A. Marcano, K. Schober, *et al.*, “Impact of bandwidth part (bwp) switching on 5g nr system performance”, in *2019 IEEE 2nd 5G World Forum (5GWF)*, 2019, pp. 161–166.

[100] F. Baccelli and S. S. Kalamkar, “Bandwidth allocation and service differentiation in d2d wireless networks”, in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 2116–2125.

[101] V. Mancuso, P. Castagno, M. Sereno, and M. A. Marsan, “Slicing cell resources: The case of htc and mtc coexistence”, in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 667–675.

[102] I. Vilà, J. Pérez-Romero, O. Sallent, and A. Umbert, “Characterization of radio access network slicing scenarios with 5g qos provisioning”, *IEEE Access*, vol. 8, pp. 51 414–51 430, 2020.

[103] D. Sabella, P. Rost, Y. Sheng, *et al.*, “Ran as a service: Challenges of designing a flexible ran architecture in a cloud-based heterogeneous mobile network”, in *2013 Future Network Mobile Summit*, 2013, pp. 1–8.

[104] P. Rost, C. Mannweiler, D. S. Michalopoulos, *et al.*, “Network slicing to enable scalability and flexibility in 5g mobile networks”, *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, May 2017.

[105] R. Ferrus, O. Sallent, J. Perez-Romero, and R. Agusti, “On 5g radio access network slicing: Radio interface protocol features and configuration”, *IEEE Communications Magazine*, vol. PP, no. 99, pp. 2–10, 2018.

[106] M. Shariat, Ö. Bulakci, A. D. Domenico, *et al.*, “A flexible network architecture for 5g systems”, *Wireless Communications and Mobile Computing*, 2019.

[107] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, “Network slicing in 5G: survey and challenges”, *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, May 2017.

[108] A. Tzanakaki, M. P. Anastasopoulos, and D. Simeonidou, “Optical networking interconnecting disaggregated compute resources: An enabler of the 5g vision”, in *2017 International Conference on Optical Network Design and Modeling (ONDM)*, May 2017, pp. 1–6.

[109] O. Adamuz-Hinojosa, P. Muñoz, P. Ameigeiras, and J. M. Lopez-Soler, “Sharing gnb components in ran slicing: A perspective from 3gpp/nfv standards”, in *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*, e, 2019, pp. 1–7.

[110] O. Adamuz-Hinojosa, P. Munoz, J. Ordonez-Lucena, J. J. Ramos-Munoz, and J. M. Lopez-Soler, “Harmonizing 3gpp and nfv description models: Providing customized ran slices in 5g networks”, *IEEE Vehicular Technology Magazine*, vol. 14, no. 4, pp. 64–75, 2019.

[111] Y. Zaki, L. Zhao, C. Goerg, and A. Timm-Giel, “Lte mobile network virtualization”, *Mobile Networks and Applications*, vol. 16, no. 4, pp. 424–432, Aug. 2011.

[112] J. F. Santos, M. Kist, J. van de Belt, J. Rochol, and L. A. DaSilva, “Towards enabling ran as a service - the extensible virtualisation layer”, in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[113] M. Sander-Frigau, T. Zhang, H. Zhang, A. E. Kamal, and A. K. Somani, “Physical wireless resource virtualization for software-defined whole-stack slicing”, in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, 2021, pp. 106–114.

[114] C. Chang and N. Nikaein, “Ran runtime slicing system for flexible and dynamic service execution environment”, *IEEE Access*, vol. 6, pp. 34 018–34 042, 2018.

[115] ——, “Closing in on 5g control apps: Enabling multiservice programmability in a disaggregated radio access network”, *IEEE Vehicular Technology Magazine*, vol. 13, no. 4, pp. 80–93, 2018.

[116] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Berlin Heidelberg: Springer-Verlag, 2004.

[117] M. Ahmad, S. U. Jafri, A. Ikram, *et al.*, “A low latency and consistent cellular control plane”, in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM ’20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 648–661.

[118] Google. (May 14, 2021). Flatbuffers white paper, [Online]. Available: `https://google.github.io/flatbuffers/flatbuffers_white_paper.html`.

[119] (Jun. 21, 2021). Bufferbloat, [Online]. Available: `https://www.bufferbloat.net/projects/`.

[120] X. Vasilakos, B. Köksal, D. H. Izaldi, *et al.*, “Elasticsdk: A monitoring software development kit for enabling data-driven management and control in 5g”, in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–7.

[121] “Near-real-time ran intelligent controller; near-rt ric architecture”, O-RAN Working Group 3, Technical Specification O-RAN.WG3.RICARCH-v01.01, 2020.

[122] “A1 interface: Application protocol”, O-RAN Working Group 2, Technical Specification O-RAN.WG2.A1AP-v03.01, Mar. 2021.

[123] “O-ran operations and maintenance interface specification”, O-RAN Working Group 1, Technical Specification O-RAN.WG1.O1-Interface.0-v04.00, Aug. 2020.

[124] C. Li and A. A. (eds.), “O-ran use cases and deployment scenarios – towards open and smart ran”, O-RAN Alliance, Whitepaper, Feb. 2020.

[125] A. Papa, R. Durner, F. Edinger, and W. Kellerer, “Sdrbench: A software-defined radio access network controller benchmark”, in *2019 IEEE Conference on Network Softwarization (NetSoft)*, May 2019, pp. 36–41.

[126] A. Papa, R. Durner, E. Goshi, *et al.*, “Marc: On modeling and analysis of software-defined radio access network controllers”, *IEEE Transactions on Network and Service Management*, pp. 1–1, 2021.

[127] A. Tootoonchian and Y. Ganjali, “Hyperflow: A distributed control plane for openflow”, in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, ser. INM/WREN’10, San Jose, CA: USENIX Association, 2010, p. 3.

[128] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, “Elasticon: An elastic distributed sdn controller”, in *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS ’14, Los Angeles, California, USA: Association for Computing Machinery, 2014, pp. 17–28.

[129] C. Jiang, H. Zhang, Y. Ren, *et al.*, “Machine learning paradigms for next-generation wireless networks”, *IEEE Wireless Communications*, vol. 24, no. 2, pp. 98–105, Apr. 2017.

[130] S. Niknam, A. Roy, H. S. Dhillon, *et al.*, *Intelligent o-ran for beyond 5g and 6g wireless networks*, 2020. arXiv: `2005.08374 [eess.SP]`.

[131] M. Andrews, K. Kumaran, K. Ramanan, *et al.*, “Providing quality of service over a shared wireless link”, *IEEE Communications Magazine*, vol. 39, no. 2, pp. 150–154, Feb. 2001.

[132] J. Jeon, “Nr wide bandwidth operations”, *IEEE Communications Magazine*, vol. 56, no. 3, pp. 42–46, Mar. 2018.

[133] J. Thota and A. Aijaz, “Slicing-enabled private 4g/5g network for industrial wireless applications”, in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom ’20, London, United Kingdom: Association for Computing Machinery, 2020.

[134] S. Matoussi, I. Fajjari, S. Costanzo, N. Aitsaadi, and R. Langar, “5g ran: Functional split orchestration optimization”, 7, vol. 38, 2020, pp. 1448–1463.

[135] C. V. Nahum, L. De Nóvoa Martins Pinto, V. B. Tavares, *et al.*, “Testbed for 5g connected artificial intelligence on virtualized networks”, *IEEE Access*, vol. 8, pp. 223 202–223 213, 2020.

[136] M. Maule, P.-V. Mekikis, K. Ramantas, J. Vardakas, and C. Verikoukis, “Real-time dynamic network slicing for the 5g radio access network”, in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[137] 3GPP, “Study on enhancement of radio access network (ran) slicing”, 3GPP, TR 38.832 V17.0.0, Jun. 2021.

[138] W. Kellerer, P. Kalmbach, A. Blenk, *et al.*, “Adaptable and data-driven softwarized networks: Review, opportunities, and challenges”, *Proceedings of the IEEE*, vol. 107, no. 4, pp. 711–731, 2019.

[139] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y.-J. A. Zhang, “The roadmap to 6g: Ai empowered wireless networks”, *IEEE Communications Magazine*, vol. 57, no. 8, pp. 84–90, 2019.

[140] M. Hoffmann, M. Uusitalo, M.-H. Hamon, *et al.*, “6g vision, use cases and key societal values”, Hexa-X, Deliverable D1.1, Feb. 2021.

[141] X. Cao, B. Yang, C. Huang, *et al.*, “Ai-assisted mac for reconfigurable intelligent-surface-aided wireless networks: Challenges and opportunities”, *IEEE Communications Magazine*, vol. 59, no. 6, pp. 21–27, 2021.

[142] B. Bertényi. (Jul. 2, 2021). Summary of ran rel-18 workshop. 3GPP, Ed., [Online]. Available: `https://www.3gpp.org/ftp/TSG_RAN/TSG_RAN/TSGR_AHs/2021_06_RAN_Rel18_WS/Docs/RWS-210659.zip`.

[143] “Use of l4s in 5gs”, 3GPP, Change Request S2-2103904, May 18, 2021.

[144] M. Irazabal, E. Lopez-Aguilera, I. Demirkol, R. Schmidt, and N. Nikaein, “Preventing rlc buffer sojourn delays in 5g”, *IEEE Access*, vol. 9, pp. 39 466–39 488, 2021.

[145] M. Irazabal, E. Lopez-Aguilera, I. Demirkol, and N. Nikaein, “Dynamic buffer sizing and pacing as enablers of 5g low-latency services”, *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[146] “Enabling time-critical applications over 5g with rate adaptation”, Ericsson, Tech. Rep., May 2021.

[147] P. Heist. (Jun. 21, 2021). Irtt (isochronous round-trip tester), [Online]. Available: `https://github.com/heistp/irtt`.

[148] 3GPP, “X2 general aspects and principles”, 3GPP, TS 36.420 V10.2.0, Oct. 2011.

[149] N. Nikaein, C.-Y. Chang, and K. Alexandris, “Mosaic5g: Agile and flexible service platforms for 5g research”, *SIGCOMM Comput. Commun. Rev.*, vol. 48, no. 3, pp. 29–34, Sep. 2018.

[150] C.-Y. Chang, L. Kulacz, R. Schmidt, A. Kliks, and N. Nikaein, “Spectrum management application - A tool for flexible and efficient resource utilization”, in *GLOBECOM 2018, IEEE Global Communications Conference*, Abu Dhabi, United Arab Emirates, Dec. 2018.