
THE UNIVERSITY
*of* EDINBURGH

# Towards a Programmable and Virtualized Mobile Radio Access Network Architecture

*Xenofon Foukas*


Doctor of Philosophy

Institute of Computing Systems Architecture

School of Informatics

University of Edinburgh

2018

# Abstract

Emerging 5G mobile networks are envisioned to become multi-service environments, enabling the dynamic deployment of services with a diverse set of performance requirements, accommodating the needs of mobile network operators, verticals and over-the-top service providers. The Radio Access Network (RAN) part of mobile networks is expected to play a very significant role towards this evolution. Unfortunately, such a vision cannot be efficiently supported by the conventional RAN architecture, which adopts a fixed and rigid design. For the network to evolve, flexibility in the creation, management and control of the RAN components is of paramount importance. The key elements that can allow us to attain this flexibility are the programmability and the virtualization of the network functions. While in the case of the mobile core, these issues have been extensively studied due to the advent of technologies like Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) and the similarities that the core shares with other wired networks like data centers, research in the domain of the RAN is still in its infancy.

The contributions made in this thesis significantly advance the state of the art in the domain of RAN programmability and virtualization in three dimensions. First, we design and implement a software-defined RAN (SD-RAN) platform called FlexRAN, that provides a flexible control plane designed with support for real-time RAN control applications, flexibility to realize various degrees of coordination among RAN infrastructure entities, and programmability to adapt control over time and easier evolution to the future following SDN/NFV principles. Second, we leverage the capabilities of the FlexRAN platform to design and implement Orion, which is a novel RAN slicing system that enables the dynamic on-the-fly virtualization of base stations, the flexible customization of slices to meet their respective service needs and which can be used in an end-to-end network slicing setting. Third, we focus on the use case of multi-tenancy in a neutral-host indoors small-cell environment, where we design Iris, a system that builds on the capabilities of FlexRAN and Orion and introduces a dynamic pricing mechanism for the efficient and flexible allocation of shared spectrum to the tenants. A number of additional use cases that highlight the benefits of the developed systems are also presented. The lessons learned through this research are summarized and a discussion is made on interesting topics for future work in this domain. The prototype systems presented in this thesis have been made publicly available and are being used by various research groups worldwide in the context of 5G research.

# Lay Summary

The rapid increase in the network traffic demands of mobile devices in recent years has created the need for a transition towards a next generation of mobile networks, frequently referred to as 5G networks. In contrast to previous mobile network generations like 4G, where the network infrastructure was composed of sophisticated hardware boxes specifically designed to provide high quality broadband services to users, 5G networks will be composed of reprogrammable hardware like commodity servers. This change will allow 5G networks to simultaneously support a multitude of diverse services including critical communications, massive broadband and machine-type communications through the automated and dynamic creation of multiple virtual networks over the common underlying infrastructure.

Perhaps the most critical part of the mobile network architecture, which is also expected to undergo this change is the radio access network or RAN, that is responsible to provide the wireless access to users. In contrast to the other parts of the mobile network infrastructure, the RAN presents certain unique challenges in realizing the aforementioned 5G vision, due to its role on managing the wireless spectrum that is made available to the users in real-time.

The goal of this thesis is to provide novel solutions in order to enable the multi-service capabilities envisioned for 5G in the context of the RAN. As a result, in the first part of the thesis, we present a platform called FlexRAN that introduces programmability in the RAN, taking into consideration the unique characteristics of the RAN and allowing the modification of its behavior on-the-fly. Building on the FlexRAN platform, the second part of the thesis focuses on the problem of how to efficiently share the RAN infrastructure among multiple tenants like mobile network operators and vertical industries. To this end, a system called Orion is developed that provides such multi-service capabilities for the RAN, while ensuring that the co-located services of tenants meet their requirements. Finally, in the third part of the thesis, we consider the specific use case of mobile access in indoor spaces and the problem of efficiently sharing the available spectrum among multiple competitive network tenants. We design a system called Iris , that builds on the capabilities of FlexRAN and Orion and introduces a dynamic pricing mechanism for the distribution of the spectrum to the tenants. All the designs presented in the thesis are accompanied by concrete prototype system implementations in order to highlight their benefits.

# Acknowledgements

I would first like to thank my supervisor Mahesh K. Marina for all his support and guidance throughout my PhD. Not only did he teach me how to do quality research and always strive for the best result, but also taught me how to enjoy the process and find balance in my personal and professional life. He is a true mentor for me in every sense of the word.

I am also very thankful to my second supervisor, Kimon Kontovasilis for his guidance and for all the helpful discussions that greatly influenced the outcome of my research. A special thanks goes to Navid Nikaein for his invaluable help during the first steps of my PhD.

I am grateful to my thesis examiners Paul Patras and Jacobus Van der Merwe for their time and effort as well as for their valuable feedback and for making my thesis defense a very pleasant and intellectually stimulating experience. Also, many thanks to Myungjin Lee for all his instructive comments during my annual reviews, which helped me improve the quality of my work.

Throughout my PhD, I have been very fortunate to work next to and collaborate with some amazing colleagues, Mohamed Kassem, Galini Tsoukaneri, Saravana Rathinakumar, Valentin Radu, Rajkarn Singh and Mah-Rukh Fida.

I owe so much to Dimitris, Kostas, Spiros, Agis, my sister Vasiliki and to the rest of my friends and family for helping me take my mind off my work when I needed it the most. My deepest gratitude goes to my parents Nikos and Fotini. Both starting and finishing my PhD would have been impossible without the unconditional love and support they have shown me throughout all of these years.

Last, but definitely not least, a huge thanks goes to Maria for being there for me both in the good and the bad moments and for always helping me stay positive. She truly made this PhD a unique experience and she is one of the main reasons that I will remember this period of my life so fondly.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified. Part of the material used for the contributions made by my thesis has been published in the papers listed below. For the FlexRAN work, the contribution of my co-authors, other than my supervisors, is limited to helping with the implementation of the respective software.

- **Xenofon Foukas, Navid Nikaein, Mohamed M. Kassem, Mahesh K. Marina, and Kimon 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 (CoNEXT)*, ACM, 2016.

- **Xenofon Foukas, Mahesh K. Marina, and Kimon 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 (MobiCom)*, ACM, 2017.

(*Xenofon Foukas*)

Στη μνήμη του πατέρα μου

In memory of my father

# Contents

# List of Figures

# List of Tables

# Acronyms

**5G NR** 5G New Radio.

**ABS** Almost-Blank Subframe.

**ARQ** Automatic Repeat Request.

**BBU** Baseband Processing Unit.

**C-RAN** Cloud RAN.

**CBRS** Citizen Broadband Radio Service.

**CCE** Control Channel Element.

**CMI** Control Module Interface.

**CoMP** Coordinated Multipoint.

**CQI** Channel Quality Indicator.

**D2D** Device-to-Device.

**DAS** Distributed Antenna System.

**DECOR** Dedicated Core Network.

**DRX** Discontinuous Reception.

**E-UTRAN** evolved UMTS Terrestrial Radio Access Network.

**eICIC** Enhanced Inter-Cell Interference Coordination.

**eNB** eNodeB.

**EPC** Evolved Packet Core.

**FDD** Frequency Division Duplexing.

**GTP** GPRS Tunneling Protocols.

**HARQ** Hybrid Automatic Repeat Request.

**HetNet** Heterogeneous Network.

**HSS** Home Subscriber Service.

**IDS** Intrusion Detection System.

**KPI** Key Performance Indicator.

**LSA** Licensed Shared Access.

**LTE** Long-Term Evolution.

**M2M** Machine-to-Machine.

**MAC** Medium Access Control.

**MANO** Management and Orchestration.

**MCS** Modulation and Coding Scheme.

**MDP** Markov Decision Process.

**MEC** Mobile Edge Computing.

**MIMO** Multiple-Input Multiple-Output.

**MME** Mobility Management Entity.

**mmWave** millimeter Wave.

**MNO** Mobile Network Operator.

**MOCN** Multi-Operator Core Network.

**MORAN** Multi-Operator RAN.

**MVNO** Mobile Virtual Network Operator.

**NAS** Non-Access Stratum.

**NFV** Network Functions Virtualization.

**NFVI** NFV Infrastructure.

**OAI** OpenAirInterface.

**OFDM** Orthogonal Frequency Division Multiplexing.

**OTT** Over-The-Top.

**P-GW** Packet Data Network Gateway.

**PDCP** Packet Data Convergence Protocol.

**PDU** Protocol Data Unit.

**PHY** Physical Layer.

**PRB** Physical Resource Block.

**QoS** Quality of Service.

**RaaS** RAN-as-a-Service.

**RAN** Radio Access Network.

**RAT** Radio Access Technology.

**RB** Resource Block.

**RE** Resource Element.

**RIB** RAN Information Base.

**RLC** Radio Link Control.

**RRC** Radio Resource Control.

**RRU** Remote Radio Unit.

**RTT** Round-Trip Time.

**S-GW** Serving Gateway.

**SAS** Spectrum Access System.

**SC-FDMA** Single-Carrier Frequency Division Multiple Access.

**SD-RAN** Software-Defined RAN.

**SDN** Software-Defined Networking.

**SLA** Service Level Agreement.

**SON** Self Organizing Network.

**TDD** Time Division Duplexing.

**TTI** Transmission Time Interval.

**UE** User Equipment.

**VNF** Virtual Network Function.

**vRAN** virtual RAN.

**vRIB** virtual RAN Information Base.

**vRRB** virtual Radio Resource Block.

**VSF** Virtual Subsystem Function.

# Chapter 1

# Introduction

## 1.1 History of Mobile Networks

Mobile wireless networks have been a disruptive technology in the context of the telecommunications industry, since they have revolutionized the way to communicate, alleviating the need for a fixed wired connection and allowing communication on-the-go. Mobile networks have come a very long way in a very short time, forming a major part of the telecommunications market with reports forecasting a further increase of their significance in the near future [110, 34]. This is due to the constant introduction of new capabilities, which go beyond the basic voice and messaging communication and aim to provide support for many novel use cases in various domains of our lives, including health, entertainment, industrial and home automation, vehicular communication etc. To better understand the direction in which mobile networks are moving, it would be interesting to provide a brief overview of their evolution so far.

Mobile networks are distinguished into generations (conventionally denoted by a number preceding a capital *G*), with each generation characterized by a set of features that either introduces new capabilities to the network or enhances and extends those offered by previous generations. As the baseline, we can consider the first generation of mobile networks or 1G, which was initially launched in Japan by NTT in 1979 and only provided voice services based on analog radio transmission techniques.

While 1G was the first true cellular mobile network architecture, it presented major limitations in terms of the number of users that it could support. As a result, the second generation of cellular technologies (or 2G) was released in the beginning of the 90s. This generation was characterized by the digitization and compression of speech, supporting a larger number of mobile users connected to the network. More-

over, 2G networks introduced for the first time data services for mobiles, initially with the hugely popular feature of SMS text messages and later with the implementation of GPRS (the so-called 2.5G), which introduced a new packet-switched domain.

The major rise of personal computers and the Internet during the 90s created a need for a further evolution of mobile networks in order to support high-speed data transfers and inter-communication of mobile devices with the Internet. As a result, a third generation of mobile networks (or 3G) appeared in the early 2000s, providing higher data rates that could reach up to 21.6Mbps. This evolution enabled the appearance of a number of applications over 3G networks, including mobile Internet access, video calls, mobile TV, GPS etc.

The applications enabled by 3G and the emergence of smartphones in the 2000s had a major impact in the telecommunications market, leading to a very high adoption rate of mobile devices by users. This increase in the scale of mobile networks, along with the need for significant improvements in the Quality of Service (QoS) of users led to Long-Term Evolution (LTE), which formed the fourth generation (or 4G) of mobile networks, was released commercially just before 2010 and is still the most widespread mobile network architecture. Among others, 4G was the first generation to introduce all-IP packet switched networks that supported peak downlink data rates of more than 100Mbps in mobility conditions and greatly improved the spectral efficiency of the radio interface to support more simultaneous users per cell.

## 1.2 Towards 5G Mobile Networks

The brief overview presented in the previous section only highlights the most important milestones in the evolution of the mobile networking industry. However, it clearly demonstrates that one of the key driving factors behind the transition from one generation to the next is the increasing demand for mobile services, which requires an improvement in the network capabilities and performance, in order to support more users and to provide them with an improved QoS.

Most interestingly, the aforementioned observation is still very relevant when considering the current 4th generation of mobile networks, with the demands of the mobile networking ecosystem changing once again. Mobile network traffic is experiencing an unprecedented growth in recent years and this trend is expected to continue in the foreseeable future. As Figure 1.1 illustrates, and just as an indication of the magnitude of this demand, Cisco forecasts that mobile traffic is expected to increase from 7 Ex-

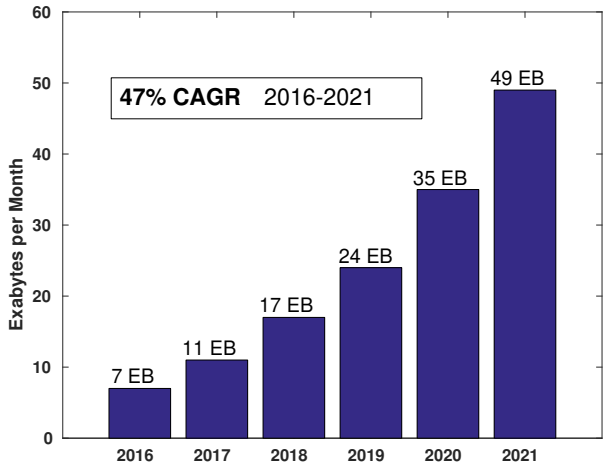



Figure 1.1: Forecast of global volume of mobile data traffic per month (Source: Cisco VNI Mobile, 2017).

abytes per month in 2016 to 49 Exabytes per month in just 5 years time, accounting for a growth rate of 47%.

Table 1.1: Global mobile devices and connections growth (Source: Cisco VNI Mobile, 2017)

| | **8 Billion Mobile Devices (2016)** | **11.6 billion Mobile Devices (2021)** | **CAGR 2016-2021** |
|---|---|---|---|
| **Smartphones** | 3.04 | 4.988 | 10.4% |
| **Non-smartphones** | 3.28 | 1.508 | -14.4% |
| **M2M** | 0.8 | 3.364 | 33.2% |
| **Phablets** | 0.56 | 1.16 | 15.7% |
| **Tablets** | 0.16 | 0.348 | 16.8 |
| **PCs** | 0.16 | 0.232 | 7.7% |
| **Other portable Devices** | 0.008 | 0.0116 | 7.7% |

This growth in the mobile network traffic is related both to the expected increase in the number of connected mobile devices as well as to the type of traffic that these devices are expected to generate. More specifically, as shown in Table 1.1, the number of globally connected mobile devices is expected to increase from 8 billion in 2016 to 11.6 billion in 2021. At the same time, the share of non-smartphone devices is expected to significantly drop in favor of smartphones, phablets and tablets, while Machine-to-Machine (M2M) communications are expected to obtain a very large portion of the

market share.

These new trends in the mobile ecosystem also lead to significant changes regarding the performance requirements that mobile networks need to fulfill. For example, the increased adoption of smartphone over non-smartphone devices is followed by a demand for an improvement of the user data rate and the spectral efficiency of the network. This is because the use of the mobile phone is no longer just for voice calls and short messages, but also for social networking, viewing high quality multimedia content (e.g. HD video streaming and high-resolution images), gaming, etc. Similarly the significant rise of M2M communications introduces new requirements that must be met in terms of energy efficiency, latency, etc.

Due to the aforementioned changes, it has already become apparent that the capabilities offered by 4G networks are no longer adequate to cover the newly emerging requirements of the mobile ecosystem. This drives the need for an evolution towards the next generation of 5G mobile networks and has ultimately led to a new wave of research with the main goal of improving the network performance. In this context, there have been a number of indicative new goals to be achieved at an operational level [64], including:

- 1000 times higher mobile data volume per geographical area
- 10-100 times more connected devices
- 10-100 times higher data rates
- 10 times lower energy consumption
- End-to-end latency that is below 1ms
- Ubiquitus 5G access

Similarly to previous generations, in order to move from 4G to 5G and to achieve these Key Performance Indicator (KPI) oriented goals there is a need to introduce technological innovations in terms of the involved Radio Access Technologies (RATs), network protocols etc. This has given rise to a number of new hot topics of research in the domain of mobile networks, including millimeter Wave (mmWave) communications, massive Multiple-Input Multiple-Output (MIMO), Device-to-Device (D2D) communications, Heterogeneous Networks (HetNets) etc.

Complementary to this performance-oriented view of 5G networks and following the observations drawn from Table 1.1 there also exists a *service-oriented* view, based on which, the 5G network is expected to cater to a wide range of services differing in their requirements and types of devices. For example, the ITU and 5G-PPP have identified three broad use case families; enhanced mobile broadband, massive machine-type

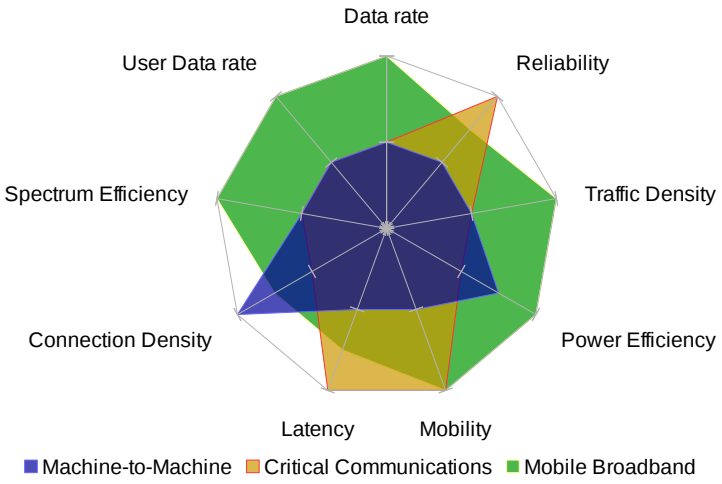



Figure 1.2: Key 5G use cases requirements. In this illustration, the further the distance of a requirement is from the center, the more important it is to the use case.

communications and critical communications. Within those, it is possible to define several specific use cases [104] ranging from general broadband access with global coverage to specialized networks for sensors or extreme mobility. The stark differences between these use cases translates to a set of heterogeneous requirements that can be seen in Figure 1.2. Meeting these requirements calls for the network to take different forms depending on the service in question, leading naturally to the notion of slicing the network on a per-service basis.

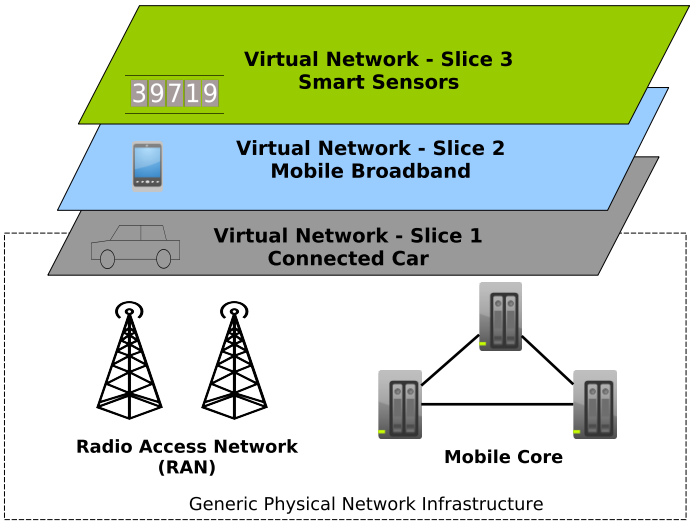



Figure 1.3: High-level idea of network slicing for various types of services.

Realizing this service-oriented view of 5G networks requires a radical rethink of the mobile network architecture to turn it into a more flexible and programmable fabric that

can be used to simultaneously provide a multitude of diverse services over a common generic underlying infrastructure. To achieve this, a concept widely considered as a key feature of the 5G architecture is network slicing, which is the capability to create end-to-end logical networks spanning both the Radio Access Network (RAN) and the core and tailored for a specific service's needs as illustrated in Figure 1.3. In contrast to this, the architecture of previous mobile network generations followed a one-size-fits-all approach, with the main goal being the optimization of mobile broadband services. This makes slicing a characteristic unique to the 5G environment and arguably a legacy that 5G networks will leave behind for future generations of mobile networks.

## 1.3 The Role of Radio Access Networks in 5G

In light of the imminent changes expected towards 5G, the RAN part of the mobile network architecture, is expected to be one of the main focuses of attention. This is natural, considering the fact that the RAN is the most complex part of the network infrastructure that significantly differs from conventional IP networks. Among others, it has to deal with a number of very significant problems in terms of coverage, interference and mobility management, energy efficiency of the connected devices etc. To add to this, the RAN has always been one of the major bottlenecks of mobile networks in terms of capacity [18], which constitutes a significant problem when scaling the network to more connected users and devices with increased throughput requirements. Due to the aforementioned reasons, it comes as no surprise that some of the biggest technological innovations when moving from one mobile network generation to the next take place in the RAN, with 5G being no exception to this.

In terms of the physical layer and more generally the radio interface, a new RAT called 5G New Radio (5G NR) is being considered to complement the existing radio interfaces [118]. 5G NR is expected to help in reducing the network latency by bringing a new and more flexible radio resource grid. It is also specifically designed with the goal of introducing support for technologies like mmWaves and massive MIMO, which are expected to greatly boost the network capacity. These new additions will form a part of a wider multi-RAT environment, which is expected to enable ubiquitous access to mobile devices as well as to cater the diverse requirements of the heterogeneous services that must be supported in the context of 5G.

The 5G RAN is also expected to evolve in terms of its architecture in order to be able to accommodate the increasing demands that arise from the proliferation of mobile

devices. The densification of the RAN using small-cell deployments for the increase of the network's capacity, as well as the centralization of its processing in a Cloud RAN (C-RAN) deployment for improved coordination among the cells are some of the most prominent changes that are expected to take place in 5G. Furthermore, the emerging paradigm of Mobile Edge Computing (MEC) is expected to bring services closer to the edge, allowing them to tap into the data as well as the processing and storage capabilities offered by the RAN [62]. This will relieve some of the stress posed to the mobile core by serving part of the generated traffic locally in the RAN, while at the same time it can lead to significant latency reductions for delay-sensitive services.

Apart from the performance improvements that the aforementioned changes are expected to bring in various aspects of the RAN (throughput, latency, capacity etc) it is equally important for future RANs to provide inherent support for network slicing to accommodate the multi-service capabilities envisioned in the context of 5G. The conventional RAN architecture found in current 4G networks is characterized by a rigid design and cannot efficiently accommodate a diverse set of services with different requirements and characteristics. It is therefore critical to re-shape the RAN into a more flexible and adaptable component of the mobile network architecture.

## 1.4 Thesis Contributions

Motivated by the need for creating a multi-service environment, we try to address the question of what changes are required in the RAN to introduce the desired flexibility. We argue that softwarization and virtualization are key technologies towards this direction, as is evident by the major success of Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) in solving similar problems in the wired networking domain. Towards this direction, in this thesis we make the following high-level contributions:

- Create a platform for introducing programmability in the RAN.
- Design and implement a virtualized and sliceable RAN system.
- Build on the above two contributions and focus on the emerging paradigm of neutral-hosts in small cell deployments, considering a novel dynamic pricing mechanism for the allocation of virtualized radio resources.

In the remainder of this section, we elaborate on these three contributions and their respective challenges. A complete list of publications and activities related to the contributions of this thesis can be found in Appendix A.

### 1.4.1 Programmability and Softwarization in the RAN

SDN is among the key technologies considered in the context of evolving mobile networks. SDN has gained significant traction over the past decade, mainly in the context of data centers and wired networks. This is brought about by paradigm shifting ideas underlying SDN, which are the separation of the control from the data plane through a well-defined API (e.g., OpenFlow), the consolidation of the control plane and the flexibility introduced to the network through its programmability. These fundamental SDN ideas can contribute towards addressing various challenges faced by current and future mobile networks. Not surprisingly, there has been considerable research interest in software-defined mobile networks in recent years, with much of the early work focusing on mobile core, given its similarity to wired networks (e.g., [66],[100]).

The RAN part of mobile networks arguably offers even greater opportunities to benefit from SDN ideas. One reason is that strategies and technologies being adopted to improve spectrum efficiency and scale system capacity — cell densification, use of multiple radio access technologies (e.g., LTE and WiFi), use of advanced PHY techniques like Coordinated Multipoint (CoMP), etc. — require a high level of coordination among base stations, which SDN can naturally enable. As another reason, softwarization of control in mobile networks, especially in the RAN, not only allows easier evolution to the future through programmability but also enables a wide range of use cases and novel services. At the same time, a Software-Defined RAN (SD-RAN) design is challenging given the unique nature of wireless resources to be managed and the stringent timing constraints associated with some key RAN control operations (e.g., radio resource scheduling). Existing SD-RAN work, although abundant (e.g., [56], [11], [29], [142]), is largely conceptual with no implemented solution that researchers can use as a reference to evaluate their SD-RAN designs and to assess the benefit of new SD-RAN enabled services.

Based on these observations, we develop FlexRAN that to the best of our knowledge is the first open-source SD-RAN platform, thereby filling the above mentioned void. FlexRAN incorporates an API to separate control and data planes that is tailored for the mobile RAN. The FlexRAN controller design and implementation factors in the

need to make real-time RAN control applications feasible. Moreover, FlexRAN is designed with flexibility, programmability and ease of deployment in mind. FlexRAN offers a great degree of flexibility to easily and dynamically realize different degrees of coordination among base stations reflecting centralized to fully distributed modes of operation. It offers programmability at two levels, one in the form of RAN control/-management applications that can be built over the FlexRAN controller and the other within the controller to be able to update the implementation of any control function on the fly. FlexRAN is also transparent to the end-devices, aiding easier deployment and evolution.

Specifically, for this thread of work in the context of the thesis, we make the following contributions:

- Realizable SD-RAN design in the form of FlexRAN. FlexRAN incorporates an API (FlexRAN Agent API) for clean separation of control and data planes in the RAN. Its hierarchical master-agent controller architecture is well suited for real-time RAN control operations while allowing reprogrammability and reconfigurability via its features of virtualized control functions and control delegation following NFV principles.

- Implementation of FlexRAN over the OpenAirInterface (OAI) LTE platform [106] is shown to be efficient, to the extent that the use of FlexRAN is imperceptible to an end user device compared to using vanilla OAI, even when considering the time critical radio resource scheduling operations. We also thoroughly characterize the FlexRAN performance behavior under different network conditions, varying number of mobile devices, and in the presence of control delegation.

- We show results from using FlexRAN in a diverse set of use cases relevant to current and future mobile networks, namely interference management, mobile edge computing and RAN sharing. This demonstrates the ease with which new applications and services can be realized with FlexRAN, thereby its effectiveness as an SD-RAN platform. We also discuss additional use cases that FlexRAN can enable.

### 1.4.2 Isolated and Efficient Virtualization of the RAN

Since a one-size fits all architecture is unlikely to be suitable for the diverse use cases that future mobile networks are expected to support, realizing the service-oriented 5G

vision in a cost-effective manner necessitates a flexible mobile network architecture that can turn the physical infrastructure into multiple slices, one per service instance. Each slice in such an architecture is an end-to-end virtualized network instance, spanning both the core and RAN, and is tailored in terms of resources to meet the requirements of the service in question. Here resources comprise of different types including computing, network, storage, radio, access hardware and Virtual Network Functions (VNFs). Unsurprisingly, most prominent 5G architectural visions embrace slicing [104, 107, 120, 12, 41, 121], building on its earlier success in multi-experiment testbed infrastructures such as PlanetLab [33] and multi-tenant data centers. As virtualization and softwarization (via SDN and NFV) are key slicing enablers, research prototypes and operational systems using virtualization technologies and SDN/NFV principles have started to appear, especially in the mobile core [115, 66, 100, 90, 16, 117, 46]. In fact, an early form of core slicing called Dedicated Core Network (DECOR) is already specified by the 3GPP standards [6].

In this part of the thesis, the focus is on RAN slicing which refers to the ability to dynamically create and manage virtual RANs, each customized to meet the requirements of an end-to-end service. RAN slicing is a challenging problem that is only starting to receive attention. The key difficulty is that the RAN virtualization and apportionment into different slices should satisfy two key objectives: (1) to ensure slice independence (functional isolation) so that tenants maintain full control of their slices to be able to tailor them to meet the respective service requirements; (2) to flexibly and adaptively share RAN resources (radio, processing, memory, networking), among different slice owners (or tenants), so that the RAN infrastructure is used as efficiently as possible, *without* violating objective (1).

Previous work on RAN slicing represents extreme points in the design space and therefore has managed to only partially address the aforementioned objectives. One approach originating in RAN sharing focuses mainly on efficient sharing of radio resources with no support for functional isolation, giving the infrastructure provider full visibility and control over slices [46, 105, 43]. The other approach puts the isolation at the center stage without considering the efficient use of resources [103, 102].

With respect to this problem, the key contribution of this thesis is Orion, which to our knowledge is the first RAN slicing system that provides functional isolation among slices while facilitating efficient sharing of the RAN resources. The design of Orion makes an explicit distinction between the infrastructure provider and service providers (slice owners), providing mechanisms that allow flexible and adaptive provisioning of

resources to slices based on their requirements. Each slice can manipulate its allocated resources in a virtualized form completely independently, and similarly customize its control plane as per the service needs. The `Base Station Hypervisor` in Orion plays a key role in achieving this by virtualizing the radio resources via a novel set of abstractions introduced in this work and by having the control plane of each slice operating in an isolated container over it, exploiting the control and data plane separation provided by the FlexRAN platform. Moreover, Orion's design allows the control planes of slices to be composed flexibly and independently, employing different levels of centralization, effectively leading to a more efficient utilization of the RAN resources and simplifying the coordination of base stations where and when required.

In summary, in this part of the thesis regarding the virtualization of the RAN, we make the following contributions:

- Present a realizable RAN slicing design in the form of Orion, that enables both the functional isolation among slices, as well as the efficient utilization of the underlying RAN resources via the novel `Hypervisor` component and by facilitating the flexible composition of the control planes of individual slices.
- Introduce a novel set of abstractions for the virtualization of the radio resources that is applicable to both current (LTE) and future (5G NR) RANs.
- Provide a concrete implementation of Orion, building over the OAI LTE platform [106] and exploiting the capabilities of the FlexRAN platform. This is accompanied by a detailed experimental evaluation of the various aspects of the system that highlight its performance and scalability as well as its various capabilities compared to the state-of-the-art solutions.
- Present an extended form of Orion that supports Over-The-Top (OTT) service providers and demonstrating its benefits.

### 1.4.3 Shared Spectrum Mobile Access For Neutral-Host Indoor Small-Cell Environments

Much of the traffic demand in 5G networks is expected to be from indoors, amounting to 80% as of 2014 according to a Gartner study and expected to rise to over 95% by the time 5G gets deployed [99]. However, indoor mobile coverage has traditionally been an issue. Distributed Antenna Systems (DAS) aimed at addressing this issue had little success, due to high deployment costs. Deployment of small cells on

the other hand is a relatively more promising means to address the coverage issue and scale the infrastructure with user density/demand. Indeed, making cells smaller and denser has historically been the biggest contributor to capacity scaling of cellular networks [144]. Despite this potential, indoor small cell deployments have been hampered due to operator concerns over deployment costs (initial capex for infrastructure deployment and return on that investment) and issues such as site access and backhaul. As a result, seamless mobile access is hardly a reality with users forced to hop between areas with WiFi coverage for high-speed wireless data access in indoor environments with high data demand (e.g., enterprises, campuses, public spaces such as airports and malls). Recently there is an emerging consensus around the notion of a *"neutral-host"* [9, 53, 94] to simplify the deployment process of indoor small-cell networks. The essential idea is that a third party entity (the neutral-host) takes on the responsibility of deploying and managing the small cell infrastructure, which is shared by multiple operators for a fee, offering *small-cells as a service*. The neutral-host becomes the only entity that needs to liase with the site owner and address issues such as power and backhaul, relieving the operators of deploying their own infrastructure and dealing with associated issues. As virtualization is a natural means for sharing the small-cell infrastructure across several operators, the neutral-host concept also aligns well with the 5G vision of supporting a diverse array of services across different Mobile Virtual Network Operators (MVNOs) and verticals via network slicing. From this perspective, the neutral-host provides each operator a virtual RAN (vRAN) spanning the area of the indoor environment it covers and this vRAN in turn becomes part of the operator's end-to-end network solution, including its existing core network or a cloud realization of the core. However, a vanilla realization of the neutral-host concept that serves just traditional Mobile Network Operators (MNOs) bringing their own licensed spectrum offers limited incentives for the neutral-host and operators alike.

We envision that the potential of the neutral-host's infrastructure sharing capability would be significantly amplified through access to a pool of spectrum that is dynamically shared among operators. Firstly, traditional MNOs would be able to gain access to additional spectrum for offloading purposes. Secondly, by removing the requirement to possess licensed spectrum (which typically only a handful of operators have), it allows new non-traditional operators into the fray, who may come with innovative revenue models differing from the traditional subscription-based model (e.g., providing free access that is monetized by advertising and analytics *a la* Internet services and free mobile apps). Lastly, the neutral-host can increase its revenue generation oppor-

tunities, by serving a wider array of operators and offering dynamic spectrum sharing capabilities. In fact, there is more support around the neutral-host model following the 3GPP defined Multi-Operator Core Network (MOCN) form of network sharing that requires use of common spectrum shared between operators [9].

The neutral-host's common (and dynamic) spectrum pool could in principle be made up of licensed spectrum pooled from different traditional MNOs, unlicensed spectrum or shared access spectrum. Regarding the latter, recent regulatory developments below 6 GHz allow sharing of lightly used spectrum held by legacy or public-sector incumbents (e.g., radars) via tiered spectrum access models [45, 109], offering substantial amounts of spectrum at a lower acquisition cost compared to licensed spectrum and without the complex coexistence issues of unlicensed spectrum. The Citizen Broadband Radio Service (CBRS) in the US [98] is a case in point, allowing the shared use of the 3.5 GHz band via a three-tier access model. A cloud-based management entity called Spectrum Access System (SAS) orchestrated by the regulator ensures that when higher tier users need to use the spectrum, they get interference protection from lower tier ones. Licensed Shared Access (LSA) model for spectrum sharing [93] that is promoted for some bands in Europe, especially in its dynamic form, is another such relevant development. *In view of the above, we consider the scenario where the neutral host is powered by shared access spectrum in the style of CBRS or LSA.*

The focus of this chapter is on addressing the challenges that arise with respect to managing access to shared spectrum in an indoor neutral-host small-cell environment. Firstly, as the neutral-host needs to support multiple (traditional and non-traditional) operators all competing in offering the same type of service to users, it should facilitate service differentiation over rival tenants and control over their share of resources without requiring direct/explicit interaction or complex coordination protocols among tenants. These are key real world concerns from the operators perspective to incentivize their participation in neutral-host small cells [9]. Secondly, the neutral-host needs to support the dynamic and efficient sharing of the spectrum among the tenants as per the relative value they attach to the spectrum. Given the competitive nature of the environment, tenants should be able to adopt their own spectrum allocation policies, without having to reveal any private information regarding their business model to the neutral-host or the other tenants. Thirdly, spectrum acquisition incurs a time-varying cost for the neutral-host depending on the amount of shared spectrum acquired to meet the overall demand. This cost needs to be recouped from the tenants in a dynamic manner as any pre-agreed static fee would either overcharge the tenants or put the neutral-host

in losses. Fourthly, using shared spectrum implies that strict Service Level Agreements (SLAs) may be infeasible for the tenants depending on the spectrum availability dynamics, so tenants should accordingly be served respecting this constraint. Last but not least, the spectrum management mechanism employed by the neutral host should be realizable in the context of a practical neutral-host small-cell system architecture.

To this end, we propose Iris, a practical shared spectrum access architecture for indoor neutral-host small cells enabled by a novel dynamic pricing mechanism that mediates access to time-varying shared spectrum by diverse tenants. The design rationale of Iris reflects the underlying intention to incentivize both the neutral-host and operators to participate.

In summary, in this part of the thesis we make the following contributions:

- We propose a dynamic pricing mechanism to regulate the demand from multiple tenants in real-time, allowing tenants to retain full control over the amount of resources to obtain and allocate amongst their users, while allowing the neutral-host to recover its dynamic spectrum acquisition costs. The main difficulty for our proposed approach lies in deciding on the price at any time instant, given that tenant behaviors are hidden from the neutral host. To resolve this issue, we leverage recent advances in deep reinforcement learning and continually observe the behavior of tenants through their actual radio resource allocation data to train a neutral-host agent on the price decision process.

- To demonstrate the practicality of the proposed approach, we leverage the virtualization capabilities of Orion and embed the dynamic pricing mechanism in a prototype neutral-host system that follows C-RAN and RAN slicing design principles. To our knowledge, this is the first concrete implementation considering the idiosyncrasies of indoor small-cell deployments and incorporating market mechanisms for shared spectrum management.

- We conduct extensive experimental evaluations using the prototype implementation of Iris, showing a number of aspects including: its learning behavior; comparable performance to an optimal but impractical alternative; superiority of Iris with respect to static pricing and representative state of the art spectrum sharing approach; and its feasibility in practice.

Overall, Iris makes a significant advance with respect to the state of the art in two key respects: (1) the proposed dynamic pricing based shared spectrum management

mechanism for an indoor neutral-host small-cell setting addresses the concerns of all entities (tenants and the neutral host) such as service differentiation among tenants and spectrum acquisition cost recovery for the neutral host, compared to existing approaches [71, 50, 146]; (2) Relative to existing neutral-host designs [44, 32, 47], it proposes a design that accounts for the peculiarities of spectrum sharing and the indoor small-cell environment, along with a concrete implementation.

## 1.5 Thesis Organization

The remainder of this thesis is organized as follows:

**Chapter 2** provides a background on the basic concepts behind mobile networks, software-defined networking and (network) virtualization. In this context, the most prevalent and emerging technologies that are relevant to the focus of this thesis are outlined and their main capabilities and limitations are briefly discussed.

**Chapter 3** makes a thorough examination of the literature related to softwarization and virtualization in the RAN. It also discusses the shortcomings of the most relevant works, explaining why solutions like the ones presented in the following chapters are required.

**Chapter 4** investigates the challenges in introducing flexible and real-time programmability capabilities in the RAN part of mobile networks. In this context, the detailed design and concrete implementation of the FlexRAN SD-RAN platform is presented. This is coupled with evaluation results and a discussion of use cases that can be enabled by FlexRAN . This content of this chapter is focused on the work presented in [46].

**Chapter 5** considers the problem of introducing virtualization capabilities in the RAN in order to enable a flexible and efficient sharing of the underlying infrastructure. Building on the softwarization concepts and capabilites established by FlexRAN in the previous chapter, the Orion RAN slicing system is presented, along with a concrete prototype implementation and evaluation results. Finally, an extension of Orion is presented for accommodating the needs of OTT service providers. The content of this chapter if focused on the work presented in [47].

**Chapter 6** investigates the benefits of virtualization for the use case of multi-tenancy in a neutral-host indoor small-cell environment with shared spectrum. The dynamic pricing mechanism of Iris for the allocation of shared spectrum to tenants is presented, along with a system architecture that embeds this mechanism alongside

cloud-RAN and RAN slicing design principles. Leveraging the virtualization capabilities enabled by Orion, a prototype implementation of Iris is presented, accompanied by evaluation results.

**Chapter 7** concludes this thesis, summarizing the work presented and discussing the limitations of the contributions as well as possible directions for future research.

# Chapter 2

# Background

This chapter provides a detailed background of the mobile network technologies that are studied in the context of this thesis. While the issues discussed in the following chapters are oriented towards problems that are very important in the context of the 5G RAN, the LTE network architecture is used throughout the thesis as a reference for the design of the proposed solutions and as the basis for concrete implementations of the corresponding systems. Towards this end, Section 2.1 provides a high-level overview of the LTE mobile network architecture, followed by a more detailed discussion on the RAN-side protocols and mechanisms that are significant in the context of the thesis.

Following this discussion, Section 2.2 focuses on presenting technologies and emerging mobile network trends that are expected to play a major role in the evolution of mobile networks towards 5G. It should be noted that the list of items presented here is not exhaustive, but rather focuses on aspects of 5G, which are relevant to the problems addressed in the thesis.

## 2.1 Overview of LTE

### 2.1.1 High-level Architecture

A high-level overview of the core and the RAN architecture of LTE is illustrated in Figure 2.1. The RAN part of the network is called the evolved UMTS Terrestrial Radio Access Network (E-UTRAN), while the network core is known as the Evolved Packet Core (EPC). The E-UTRAN, which will be so forth called the LTE RAN, is composed of a number of base stations called eNodeBs (eNBs), which are responsible for providing the wireless link to the User Equipment (UE). The eNBs are interconnected using

the so-called X2 interface, which is used for their coordination, for performing data transfers during handover operations etc. Each eNB is connected to the core network both through a control plane (S1-MME) and a user plane (S1-u) interface.

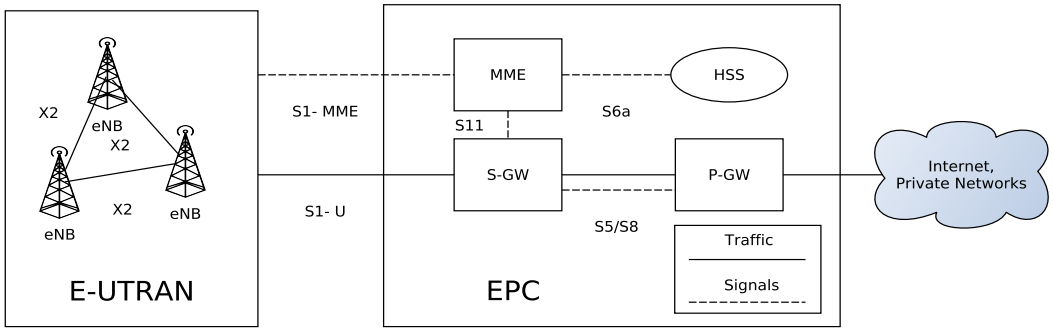



Figure 2.1: High-level overview of LTE architecture

The EPC is a (fixed) backhaul network that is composed of a number of entities responsible for managing and routing the mobile traffic. The Serving Gateway (S-GW) is an entity that acts as a switch and is responsible for forwarding the user traffic within the EPC. The Packet Data Network Gateway (P-GW) is an entity that is similar to the S-GW and acts as a gateway responsible for routing traffic in and out of the EPC, e.g. to the Internet or in some other external private network. The Mobility Management Entity (MME) is responsible for handling operations related to the mobility and the security of the UEs (authentication, UE attachment, handovers etc.), while the Home Subscriber Service (HSS) is a database containing subscriber-related information. Figure 2.1 also depicts the names of all the interfaces used in the EPC for signaling and traffic exchanges among the various entities. The dashed lines represent interfaces that are used for signaling (control) operations, while the solid lines represent interfaces that are used for user data traffic.

UE data are transmitted to and from the eNBs and within the EPC through virtual "pipes", called bearers. Bearers are essentially virtual end-to-end connections with pre-defined QoS levels, with each bearer encapsulating one or more user flows. A UE can have a number of active bearers, depending on the number of connections that it has established and its QoS requirements (e.g. guaranteed or non-guaranteed bitrates). Finally, all the traffic traversing the LTE network is encapsulated in a group of IP-based communication protocols called GPRS Tunneling Protocols (GTP). More specifically, the GTP-c protocol is used within the EPC to carry signaling traffic, while the GTP-u protocol is used for carrying user data within the EPC and between the LTE RAN

and the EPC. Any incoming traffic to the mobile network (e.g. IP flows) is always encapsulated into the GTP-u protocol.

### 2.1.2 Radio Access Network Architecture

Since the main focus of this work is on the RAN, it is important to present a more detailed view of the LTE RAN protocol stack, as well as a brief description of the physical layer design for the air interface.

#### 2.1.2.1 LTE RAN Protocol Stack

A simplified version of an eNB protocol stack is presented in Figure 2.2. The illustrated protocols can be distinguished into two categories; protocols of the air interface and protocols that are used for the communication with the EPC.

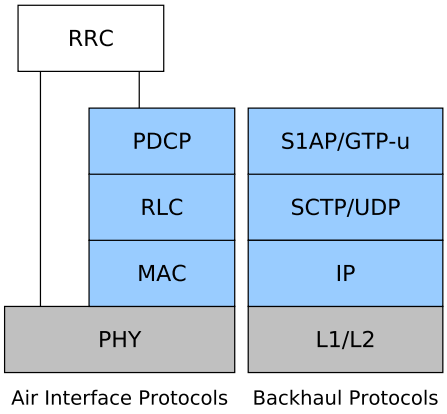



Figure 2.2: Air interface and backhaul protocol stack of an eNB

Following a top-down approach, the air interface protocol stack consists of:

- Radio Resource Control (RRC) – This is a control protocol that among others is responsible to decide on the cell-related information to broadcast, the establishment of secure connections with the UEs, handovers, paging as well as the configuration of UEs for measurements related to mobility, QoS etc.

- Packet Data Convergence Protocol (PDCP) – This is the higher L2 sub-layer of the air interface protocol stack and is responsible to provide functionalities like ciphering and integrity protection (both RRC messages and data traffic), as well as in-sequence delivery, duplicate detection and retransmission of data traffic.

- Radio Link Control (RLC) – This is the middle L2 sub-layer that is responsible to transport PDCP Protocol Data Units (PDUs) to the lower layer. It offers different modes of operation for the transmission of PDUs, where among others it can provide Automatic Repeat Request (ARQ) error correction, concatenation and segmentation of PDUs, in-sequence delivery etc.

- Medium Access Control (MAC) – This is the lowest L2 sub-layer, which is responsible for making the scheduling decisions for the transmission of the data over the air interface. Among others it is responsible to handle the prioritization of the scheduling of data originating from different logical channels of the same UE, the dynamic scheduling between UEs, the retransmission of lost or corrupt data through Hybrid Automatic Repeat Request (HARQ) error correction etc.

- Physical Layer (PHY) – This layer is responsible to carry control and data information from the MAC to the air interface. It is also responsible to perform operations like link adaptation, power control, cell search and measurements of UE signal quality.

On the backhaul of the mobile network, we can distinguish the following protocols:

- S1AP/GTP-u – These protocols are used by the eNB for the transmission of control messages to/from the MME (S1AP) and for the transmission of encapsulated user data to/from the S-GW (GTP-u).

- SCTP/UDP – SCTP is the transport layer protocol used for the transmission of S1AP related data, while UDP is normally used as the transport layer protocol for GTP-u traffic.

- IP – The EPC is an all-IP network, meaning that IP is used as the network protocol for the communication among the EPC entities and the eNBs.

- L1/L2 – The specification of LTE does not provide strict guidelines regarding the use of L1/L2 protocols on the backhaul side. In the simplest case, the communication could be performed using Ethernet, however other approaches, like microwave communications could also be employed.

#### 2.1.2.2 Physical Layer Design

LTE uses Orthogonal Frequency Division Multiplexing (OFDM) on the downlink and a precoded version of OFDM called Single-Carrier Frequency Division Multiple Access (SC-FDMA) on the uplink.

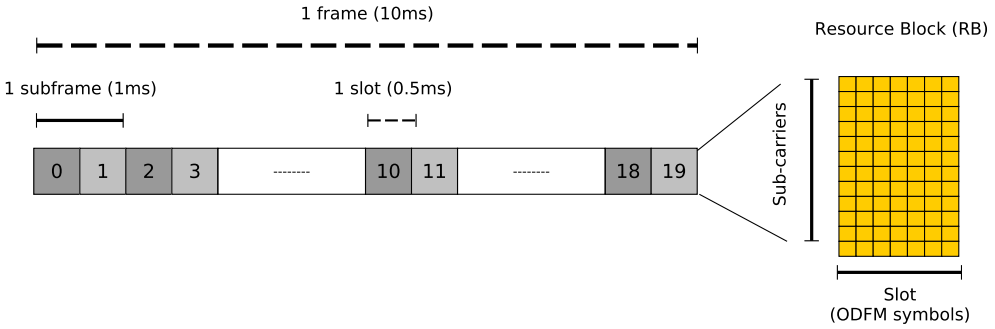



Figure 2.3: LTE frame structure

A simplified version of the structure of an LTE frame is illustrated in Figure 2.3. Each frame is composed of 10 subframes, which in turn are composed of 2 slots. A Resource Block (RB) is the smallest unit of resources that can be allocated to a user and is 180kHz wide in frequency and 1 slot long in time. In the frequency dimension, each RB is divided into 12 sub-carriers (15kHz each) and in the time dimension, each slot is composed of 6 or 7 OFDM symbols, depending on whether a normal or extended cyclic-prefix is used for mitigating inter-symbol interference. The minimum data carrier of the LTE frame is the Resource Element (RE), which is 1 subcarrier x 1 OFDM symbol. The amount of RBs that are available in the resource grid can vary depending on the available bandwidth. For example, in the case of 5MHz of bandwidth, 25 RBs are available in each subframe, while in the case of 20MHz, there are 100 RBs available.

Each slot in LTE is 0.5ms long, resulting in subframes that are 1ms long and frames with a duration of 10ms. The MAC scheduler of the air interface is responsible to make its scheduling decisions and perform its scheduling operations within a period of a Transmission Time Interval (TTI) that is equal to the duration of a subframe, i.e. 1ms long. Obviously, this makes scheduling in LTE a real-time operation with very strict time constraints. Missing a scheduling deadline can lead to the degradation of the system's performance in terms of the throughput experienced by UEs.

Given the aforementioned frame structure, two types of frames are supported in LTE for the transmission of data; (i) Type 1 uses Frequency Division Duplexing (FDD),

where the uplink and downlink are separated by frequency and (ii) Type 2 uses Time Division Duplexing (TDD), where uplink and downlink are separated in time, i.e. different subframes are used for the uplink/downlink operations over the same frequencies.

## 2.2 Emerging Mobile Network Trends and Technologies

### 2.2.1 Software-Defined Networking

The SDN approach allows the management of network services through the abstraction of lower level functionality. Instead of dealing with low level details of network devices regarding the way that packets and flows are managed, network administrators now only need to use the abstractions available in the SDN architecture. The way that this is achieved is by decoupling the control plane from the data plane following the layered architecture illustrated in Figure 2.4.

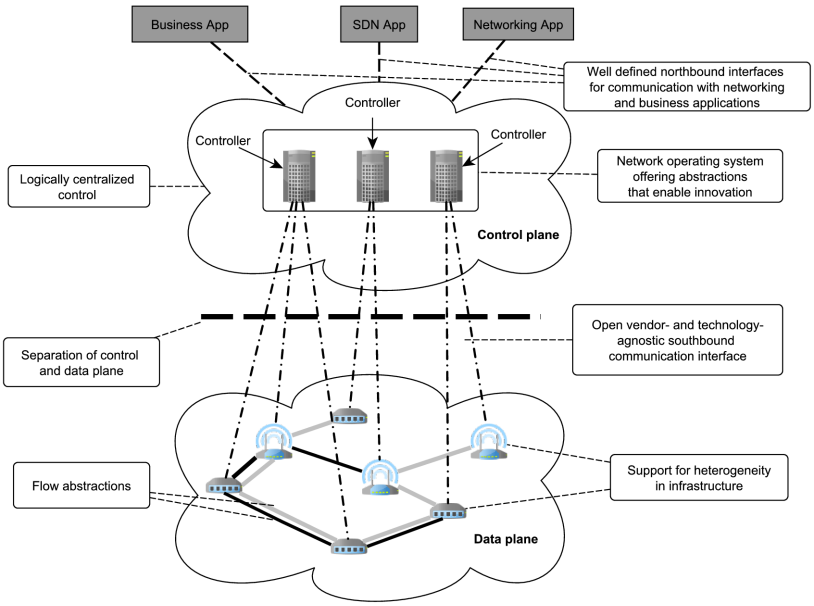



Figure 2.4: Key ideas underlying the SDN paradigm

At the bottom layer we can observe the data plane, where the network infrastructure lies. In the context of SDN these devices have been stripped of all control logic (e.g., routing algorithms) simply implementing a set of forwarding operations for manipulating network data packets and flows, providing an abstract open interface for the communication with the upper layers. In the SDN terminology these devices are

commonly referred to as network switches.

Moving to the next layer we can observe the control plane, where an entity referred as the controller lies. This entity encapsulates the networking logic and is responsible for providing a programmatic interface to the network, which is used to implement new functionality and perform various management tasks. The control plane of SDN is ripped entirely from the network device and is considered to be logically centralized, while physically it can be either centralized or decentralized residing in one or more servers, which control the network infrastructure as a whole.

At the top of the SDN stack lies the application layer, which includes all the applications that exploit the services provided by the controller in order to perform network-related tasks like load balancing, network virtualization etc. One of the most important features of SDN is the openness it provides to third-party developers through the abstractions it defines for the easy development and deployment of new applications in various networked environments from data centers to wireless and cellular networks. Moreover, the SDN architecture eliminates the need for dedicated middleboxes like firewalls and Intrusion Detection Systems (IDSs) in the network topology, as it is now possible for their functionality to be implemented in the form of software applications that monitor and modify the network state through an API provided by the controller. Obviously, the existence of this layer adds great value to SDN, since it gives rise to a wide range of opportunities for innovation, making SDN a compelling solution both for researchers and the industry.

Finally, the communication of the controller to the data plane and the application layer can be achieved through well-defined interfaces (APIs). We can distinguish two main APIs in the SDN architecture: i) a southbound API, like OpenFlow ([95]), for the communication between the controller and the network infrastructure; and ii) a northbound API defining an interface between the network applications and the controller. This is similar to the way communication is achieved among the hardware, the operating system and the user space in most computer systems.

### 2.2.2 Network Functions Virtualization

NFV is a carrier-driven initiative with a goal to transform the way that operators architect networks by employing virtualization related technologies like virtual machines and containers, to virtualize network functions such as switches, routers, IDSs and NATs so that they can run in software. Through the introduction of virtualization it

is possible to run these functions over generic industry-standard high volume servers, switches and storage devices instead of using proprietary purpose-built network devices. This approach reduces operational and deployment costs, since operators no longer need to rely on expensive proprietary hardware solutions. Finally, flexibility in network management increases as it is possible to quickly modify or introduce new services to address changing demands.

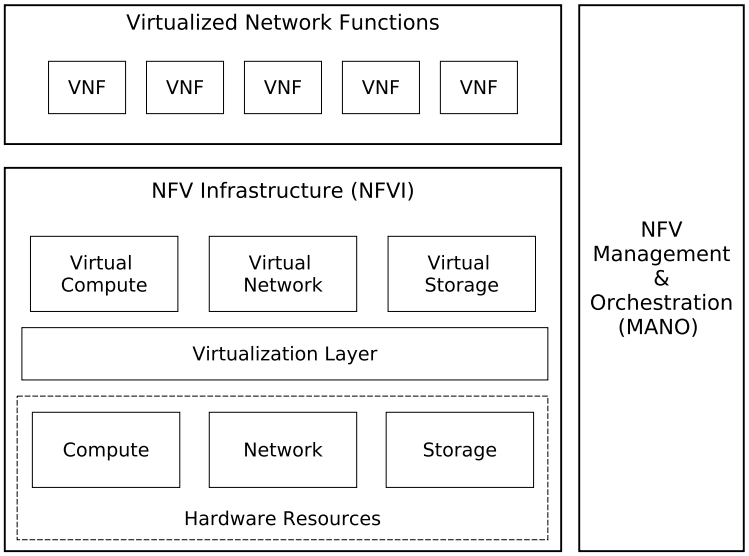



Figure 2.5: NFV high-level reference architecture.

A generic high-level NFV architecture can be seen in Figure 2.5, based on the NFV framework of ETSI ([54]). At the bottom layer, we can observe the NFV Infrastructure (NFVI), which is composed of the hardware resources. These resources are generally distinguished into Compute, Network and Storage resources, which are abstracted and virtualized through a virtualization layer. The virtualized resources can then be allocated for the use of different VNFs that reside at the top of the architecture (e.g. firewalls, routers etc.). Finally, an NFV Management and Orchestration (MANO) entity is responsible for the on-boarding of network services, their life-cycle management as well as for the management of the underlying NFVI resources.

### 2.2.3 Cloud RAN

The C-RAN ([31]) is a concept that has attracted a great deal of attention because of the way it manages to centralize RAN computational resources. The centralized processing enabled by this architecture allows the C-RAN to deal with complex coor-

dination issues that are becoming increasingly important in the context of 5G, like that of interference and mobility management in dense network settings.

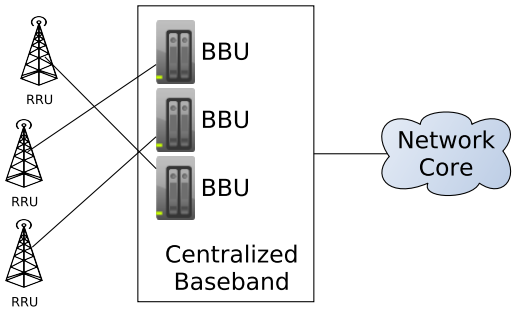



Figure 2.6: High-level overview of C-RAN

In order to achieve this, multiple sites are connected to a datacenter where the baseband processing is performed using a pool of Baseband Processing Units (BBUs), as illustrated in Figure 2.6. The signals are transmitted from Remote Radio Units (RRUs) to the datacenter over a fronthaul network composed of high-speed transmission lines (e.g. fibers or microwaves in the case of line of sight).

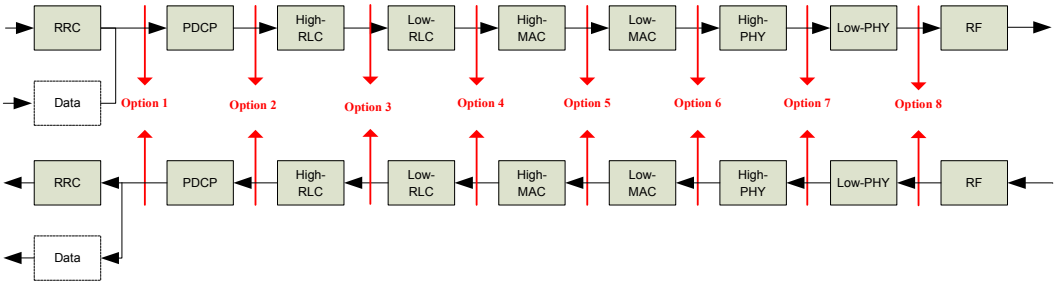



Figure 2.7: General description of functional split options (Source: 3GPP TR 38.801 )

Depending on the functionality that needs to be centralized, the functions of the RAN protocol stack can be split into different levels, with the lower layer functions (e.g. PHY and MAC) residing at the RRUs and the higher layer ones (e.g. RLC/PDCP and RRC) residing at the BBUs. Depending on the functional split configuration, the centralization of the processing comes with a trade-off in terms of the network's efficiency and the cost for the implementation of the fronthaul link. Since no functional split is ideal for all deployment scenarios due to the cost-efficiency trade-off, 3GPP specifies a number of splits that could be adopted, as illustrated in Figure 2.7.

### 2.2.4 Heterogeneous Networks and Small Cell Deployments

With the term HetNet we refer to radio access networks that present heterogeneity in the size and the placement of the cells within the same radio access technology.

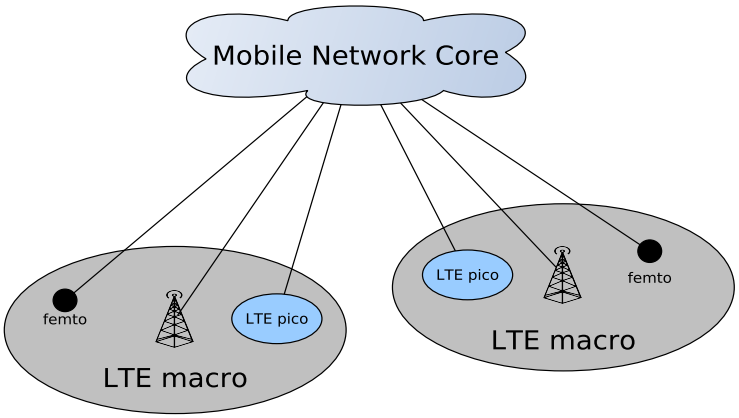



Figure 2.8: Example of heterogeneous network

Generally, we can distinguish the cells in macro and small cells, as illustrated in Figure 2.8. A macro cell is a high-powered cell that has traditionally been used to cover large areas with a range of up to 20$km$. Small cells are low-powered radio access nodes operating in licensed and/or unlicensed spectrum and have a much smaller range of up to 2$km$. In the context of LTE, these are usually deployed in an unplanned manner in densely populated areas with the goal of increasing the spectral capacity by offloading traffic or by patching areas with bad signal quality. Small cells can be further distinguished in micro (2$km$ range), pico (200$m$ range) and femto (10$m$ range) cells. The latter are usually found indoors in homes and offices and are closed-type cells allowing access only to a predefined group of subscribers. The co-existence of various small cells within the area of a macro cell is one of the most common causes for interference in HetNets. This leads to the need for interference management among the cells, through various coordinated techniques commonly known as Enhanced Inter-Cell Interference Coordination (eICIC). In the context of this thesis and for the remaining chapters, the term small cell will be used to refer to pico and femto cells with very small ranges.

# Chapter 3

# Related Work

This chapter gives an overview of the literature related with the problems discussed in the context of this thesis, building on the concepts and the background material presented in Chapter 2. The works that are most relevant to the contributions of this thesis are listed, explaining the challenges that they try to address as well as their shortcomings, which drive the need for the novel solutions presented in the following chapters.

## 3.1 Software-Defined Radio Access Networks

### SDN in the Mobile Core

Software-defined control of mobile networks has received substantial attention from the research community in recent years with both academia and industry recognizing its benefits and proposing ways to integrate SDN principles to operational mobile networks [79]. Owing to the similarity between mobile core and wired networks much of the cellular SDN research to date focuses on the core part of mobile networks. The focus of this body of work [66],[100], [16], [117], [115], [90], [36], [88] is on novel control designs based on SDN and NFV to address key core network issues of traffic and mobility management, and enable mobile networks to scale in the presence of high volumes of traffic.

### SDN in the RAN

While the scope of some of the above mentioned works includes the RAN, none of them address radio resource management, a vital aspect of the RAN and the focus of

our work. RAN radio resource management is unique in the type of resources managed and the stringent timing constraints associated with some of the key functions (e.g., scheduling). In the last few years, there have been *several high-level conceptual works on SD-RANs* that do consider the radio resource management aspect [56], [11], [29], [142], [141], [10], [14]. SoftRAN [56] is among the earliest of these works. It introduced the idea of a big base station abstraction aimed at turning dense network deployments into sparse ones through the separation of the control and the data plane. In SoftRAN, control functions are *statically* separated into central and distributed ones based on their time criticality and their requirement for a centralized network view (e.g., centralized handovers and distributed downlink scheduling). Later works outline several potential SD-RAN designs, targeting different applications and settings. Similarly to SoftRAN, these works consider the tradeoffs between cost and efficiency for the control of the RAN and accordingly propose designs that follow either a hierarchical approach as in SoftRAN (e.g., [11], [29], [142]), where different layers of controllers are responsible for different operations based on their time criticality, or a fully centralized approach (e.g., [141], [10], [14]) where all the processing (L1/2/3) is performed centrally at a cloud data center.

A common and key limitation of the aforementioned SD-RAN works, which serves as the main motivation of our work, is that none of them have been implemented (and thus do not consider the associated challenges such as real-time control in their designs) nor do they tackle the issue of separating the control and data planes in the RAN in a practical and concrete manner. Moreover, none of these works offer mechanisms to make control in the RAN adaptive and flexible by allowing a dynamic functional split (e.g., centralized to distributed scheduling and vice versa) depending on the deployment scenario and the constraints posed by the underlying network conditions at any given point in time. Finally, not all of the proposed SD-RAN architectures are transparent to the UEs, with some designs proposing the introduction of programmability even at the UE-level (e.g., [142], [20]), raising backward compatibility concerns for legacy devices.

The only concrete implementation of a programmable SD-RAN platform similar in spirit to our work is 5G-EmPOWER [119]. However, this work mainly focuses on enabling programmability and providing mobility and power management abstractions for WiFi. In terms of mobile network technologies like LTE, it does not consider the challenges of critical real-time control operations like scheduling, only providing high-level APIs focusing on the monitoring and coarse-grained control of base stations.

### Other Works on RAN Control and Programmability

There are also works in the literature that are relevant from a RAN programmability perspective but can be viewed as complementary to our focus on a SD-RAN platform capable of performing radio resource management. For example, RadioVisor [57] deals with the challenge of radio resource virtualization in the RAN, allowing individual SD-RAN controllers to manage their own isolated slices. OpenRadio [17] and PRAN [139] deal with data plane programmability and how new wireless protocols can be implemented on-the-fly programmatically. Finally, the work of Tsagkaris et al. [134] proposes a software-defined framework for Self Organizing Networks (SON) that simplifies the management of SON functions via a controller based on SDN principles.

## 3.2 Network Slicing

### RAN Slicing

State of the art on RAN slicing can be traced back to the earlier works on active RAN sharing [49, 145, 35, 108, 124]. Broadly speaking, two approaches under the names of MOCN and Multi-Operator RAN (MORAN) have been considered. While both approaches imply the use of separate core networks for each participating operator, MOCN allows for spectrum sharing among operators while MORAN requires dedicated spectrum for each. Relatively, more attention has been given in the literature to the MOCN approach (e.g., [76, 89, 59]), which has been standardized for LTE in Release 8. NVS [76] is a representative example. Note that the use of the term virtualization in some of these works is somewhat misleading as it refers only to the UE perceived performance isolation (i.e. throughput) among operators sharing the RAN radio resources and not on the functional isolation and corresponding performance isolation of the slices' virtual network functions in terms of the required computing resources (processing, memory, networking). However, functional isolation is additionally essential in the RAN slicing context. The major focus of these works is on the design of efficient radio resource scheduling algorithms while considering certain guarantees for operators. The fact that radio resource sharing is also relevant for efficient RAN slicing is reflected in the more recent algorithmic work in this thread [68, 92, 51, 26]. This body of work is complementary to our focus on systems support for RAN slicing. In fact, we employ the NVS scheduling algorithm in our prototype to highlight the

efficient radio resource use feature of Orion.

From a systems perspective, RAN sharing oriented slicing (with no functional isolation among slices) has been explored through the use of the FlexRAN SD-RAN platform [46, 78, 105, 43], which is presented in Chapter 4 and we include in our comparative evaluations. Network slicing is enabled with FlexRAN by programmatically defining the way in which the radio resources need to be allocated among the connected UEs based on the requirements of the slice they belong to. A unified control plane, which is controlled by a single entity (usually the infrastructure provider), is responsible to perform the corresponding control operations. This approach can be limiting, since the capabilities of the slices are fully dependent on the types of control functions that are bundled in the control plane of FlexRAN . In contrast, Orion allows independent and fully customizable control planes for each slice so that slice owners can flexibly introduce their own functionality in the RAN and tailor their slice as per the needs of their service.

To accommodate this need for slice customizability, the other RAN slicing approach taken in the literature seeks full isolation (by running the virtual base station instance of a slice within a Docker container for example) but assumes dedicated radio hardware and spectrum per slice [103, 102], bearing some similarity to the MORAN form of RAN sharing in that resource sharing among slices is limited at best to computing, memory and storage resources. This has the downside of inefficient use of radio resources and foregoing potential statistical multiplexing gains. The work presented in [107] also takes the same approach, although the focus there is on the idea of a network store for VNFs to aid in dynamic network slicing. On a more general note, wireless virtualization overview and position papers like [143, 57, 82, 10] advocate functional isolation, which strengthen the motivation for the approach we take in Orion to have an isolated and customizable control plane for each slice.

In terms of radio resource virtualization, recent works in the domain of RAN slicing [57, 82, 78, 52] have advocated the need for abstractions that decouple the control plane decisions from the physical radio resource grid. However, the abstractions presented in these works are high-level and do not consider the constraints that can be imposed by the physical layer (e.g., frequency-dependencies in scheduling). The abstractions introduced in Orion for radio resource virtualization not only overcome these limitations but are also generic in that they are applicable to both current LTE and future 5G NR air interface technologies.

### Mobile Core Slicing

There has been significant progress on mobile core slicing to the point that it is fairly mature and also made its way into 3GPP standards in a basic form under the name of DECOR [6]. Virtualization of core network functions combined with the use of mature virtualization technologies (e.g., KVM [75], LXC [60], Docker [96], VLANs [15, 127]) have led to systems that realize core network slicing [103, 107, 74]. Even the possibility of EPC as a service over the cloud has been explored [133]. Several research proposals leveraging NFV in the core appeared in the recent past, aimed at its optimization for better scalability (e.g., [16]), customization for particular use cases (e.g., [132]) and in general making it more flexible (e.g., [117]).

### Management and Network Orchestration

The end-to-end nature of services create the need for a MANO entity responsible for realizing end-to-end slices spanning the core and the RAN as well as their life-cycle management to ensure compliance with their service requirements. This has led to MANO oriented research work [131] and several open-source MANO framework implementations (e.g., OSM [42], ONAP [84]). More pertinent to our work, the authors of Proteus [131] acknowledge the lack of a flexible RAN slicing solution and therefore consider use cases that focus on innovations around the mobile core. Orion fills this void. We present how Orion can be used in an end-to-end slicing context by interfacing with any of the MANO solutions mentioned above.

## 3.3 Shared Spectrum Indoor Small-Cell Neutral-Host Environments

### Neutral-host system designs and specifications

A number of recent designs consider multi-tenancy support in mobile RANs that are applicable to the indoor neutral-host small cell setting. Perhaps the ones most relevant are: Orion [47], SESAME [44] and ESSENCE [32]. However, these works do not consider the use of shared spectrum and its implications.

In terms of specifications targeting the neutral-host setting, nFAPI [129] is the most relevant one. It specifies a functional split at the MAC layer. Each virtual operator gets a VNF implementing the protocols from the MAC layer and above and a physical

network function for the PHY. In contrast to our work, nFAPI tenants do not share a common pool of spectrum that is dynamically distributed in real-time. Instead, a coarse grained approach is taken, where each physical network function is assigned its own chunk of spectrum (shared or licensed). Another closely related specification is MulteFire [101], which is a form of LTE deployment in unlicensed bands. In contrast to our work, the focus of MulteFire is on the air interface and the ways to enable co-existence with other technologies operating over unlicensed spectrum (e.g. WiFi).

### Spectrum allocation mechanisms

Radio resource sharing in the neutral-host context can be seen as a specific scenario of the more general problem of RAN slicing. Many algorithmic works in the domain of RAN slicing [126, 25, 76, 77, 65] focus on solving optimization problems that consider the available resources and the slices' SLAs, to decide the optimal allocations either at the base station level (e.g., [76, 65]) or at the RAN level (e.g., [77, 25, 126]). However, these works do not consider the shared spectrum acquisition cost incurred to the neutral-host and rely on the centralized allocation of resources, assuming the prior knowledge of the tenants' valuation for the spectrum by the infrastructure provider.

Shared spectrum allocation has been considered in settings where participants can deploy infrastructure independently [27, 70, 87], as well as in the neutral-host context [72]. However, as in RAN slicing, these works assume centralized allocation schemes that rely on SLAs or require pre-defined agreements among the operators and the spectrum provider regarding the operators' access to the spectrum (e.g., through static priorities or proportionally to the load of the operators).

### Market Mechanisms

The work in [19] proposes a spectrum allocation mechanism in which revenue maximization for the infrastructure provider is also factored in, but again in the context of SLA-based service provisioning, which is limiting for our shared spectrum setting. More similar in spirit to this work, dynamic pricing has been considered in auction-based shared spectrum allocation mechanisms [50, 146]. However, these mechanisms are impractical from an implementation point of view, by making simplified assumptions regarding the way the spectrum can be shared among the tenants [50] or by requiring the involvement of the users' UEs to the auction [146]. Moreover, they do not consider the shared spectrum acquisition cost of the neutral-host; a problem that would

require to dynamically decide on the appropriate level of the reserve price for the tenant bids to ensure that the neutral-host does not experience losses. From this perspective, the mechanism we present for dynamic pricing will be an essential component of any auctioning based mechanism for the target setting.

### Dynamic Pricing in Other Contexts

Dynamic pricing in mobile networks also appears in the literature for the interaction among operators and end-users (e.g. in TUBE [58]). In these schemes, prices can vary throughout the day and are usually announced to the end-users on the previous day. This allows users to decide on their network access pattern for the coming day. The goal there is to incentivize the users to shift their demand into periods of the day when the network is less congested. However, such schemes cannot be readily applied in this work, since the end-user demand cannot be directly controlled by the neutral-host, but rather from the tenants.

Finally, this work shares some similarities with the problem of demand-response in smart-grids, where electricity providers need to control the network load and to match the users' demand with their supply, maximizing their utility. Dynamic pricing has been shown as an effective mechanism in this context (e.g. [80, 85]). It is also relevant to mention that reinforcement learning has been successfully used in this context for the adjustment of prices and allocation of power to consumers (e.g. [138, 73, 111]), although the time granularity of the considered solutions in that domain is significantly coarser compared to ours (hours or even days).

# Chapter 4

# FlexRAN: A Software-Defined Radio Access Network Platform

## 4.1 Introduction

In this chapter, we focus on the first problem considered in the context of this thesis, regarding the softwarization and programmability of the RAN. As already explained in Section 1.4.1, introducing software-defined capabilities in the RAN to create a so-called SD-RAN architecture, can greatly improve its flexibility and adaptability, allowing its behavior to be modified on-the-fly.

However, the differences in the nature of the RAN compared to wired networks, introduce new challenges in terms of the software-defined control. This is due to the fundamentally different nature of the controlled resources (radio resource blocks vs flows) and the stringent time constraints posed by critical control operations like scheduling. These challenges, along with the fact that no concrete SD-RAN implementation existed to date, led us in the creation of the FlexRAN platform, which, to our knowledge, is the first concrete prototype platform of its kind.

In this chapter, we provide an in-depth discussion regarding the design of FlexRAN and how the mechanisms that it introduces help us overcome the aforementioned challenges. The concrete implementation of the FlexRAN platform provides a tool that can prove very useful for research and experimentation not just in the domain of the SD-RAN, but also of 5G more generally, due to the various applications and use cases that it can enable. This is something that will be demonstrated not just in this chapter, but also in Chapter 5, since FlexRAN acted as an enabler for the implementation of the Orion RAN slicing system.

This chapter begins by first providing an overview of FlexRAN in Section 4.2. It then proceeds in presenting the details of its design and implementation in Section 4.3, followed by an evaluation of the system's performance in Section 4.4. A number of concrete and diverse use cases that demonstrate the benefits of FlexRAN are presented in Section 4.5, accompanied by evaluation results. Finally, a further discussion on additional use cases of FlexRAN and its applicability to other radio access technologies beyond LTE is made in Section 4.6.

## 4.2 FlexRAN Overview

This section gives a high-level overview of the FlexRAN SD-RAN platform. We present FlexRAN in the context of LTE for concreteness and to match with its current implementation, using the LTE terminology of eNBs and UEs for base stations and mobile devices. It is however important to note that there is nothing LTE-specific that FlexRAN assumes, thus its design is general and equally suitable for future mobile RAN architectures.

Figure 4.1 provides a high-level schematic of the FlexRAN platform, which is made up of two main components: the **FlexRAN Control Plane** and the **FlexRAN Agent API**. The control plane follows a hierarchical design and is in turn composed of a **Master Controller** that is connected to a number of FlexRAN **Agents**, one for each eNB. The agents can either act as local controllers with a limited network view and handling control delegated by the master, or in concert with other agents and the master controller. The control and data plane separation is provided by the FlexRAN Agent API which acts as the southbound API (*a la* OpenFlow) with the FlexRAN control plane on one side and the eNB data plane on the other side.

The FlexRAN **Protocol** facilitates communication between the master controller and the agents by allowing a two-way interaction between them. In one direction, the agent sends relevant messages to the master with eNB statistics, configurations and events, while in the other direction the master issues control commands that define the operation of the agents. In contrast to typical SDN controllers found in the wired domain, the FlexRAN controller has been designed with support for time critical RAN operations (e.g., MAC scheduling) in mind. Due to this real-time aspect and in order to fully utilize the power of FlexRAN, in the ideal case the communication channel between the agents and the master would be a high-bandwidth and low-latency channel (e.g., optical fiber path). However it should be noted that this is not a hard constraint,

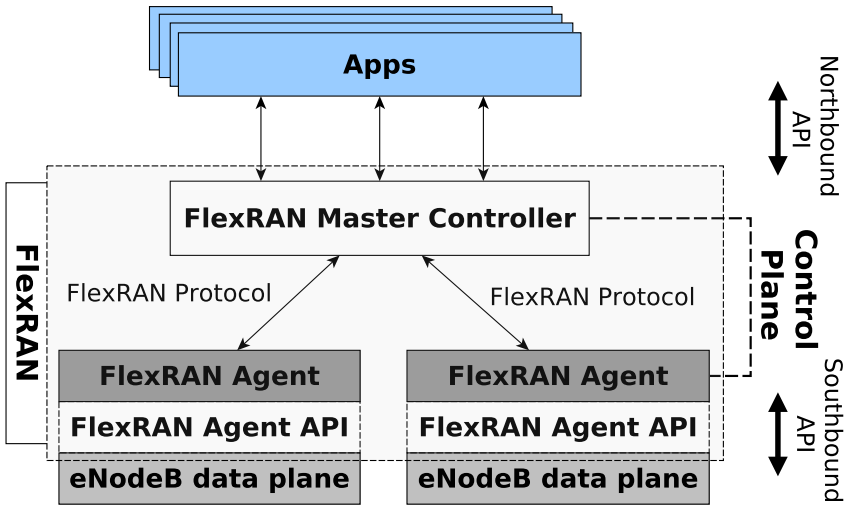



Figure 4.1: High-level schematic of the FlexRAN platform.

as the system provides flexibility to operate in non-ideal networking conditions with a small impact on its performance and capabilities (see Section 4.3.3.1).

The centralization in the control of the radio resources offered by FlexRAN might appear to share some similarities with the control offered by older mobile architectures like for instance the Radio Network Controller controlling the NodeBs at the RAN of UMTS. However, in contrast to such architectures, FlexRAN proposes a unique design that introduces great flexibility in the way in which the network is managed and controlled. More specifically, as it will be discussed in-depth in the following sections, the FlexRAN agents allow the control of the RAN operations to be delegated to the agents by the controller when the underlying network conditions demand it, while it also allows network operators to easily modify the operation of the RAN in order to support new functionality based on their requirements, making the RAN evolvable.

Note that Figure 4.1 does not include a UE, reflecting the fact that FlexRAN is transparent to UEs. The FlexRAN agent ensures that any command issued by the master that would affect the operation of a UE will be passed to the eNB, which will in turn apply the modification using the RAN technology standard in use (LTE protocol in our implementation). This transparency ensures the evolvability of the system, since new LTE and FlexRAN protocol extensions can be implemented at the RAN without causing backwards compatibility issues for users and without disrupting the use of their devices with constant updates to enable support of new network features. As long as a UE adheres to the LTE standard, it will be fully compatible with the FlexRAN platform.

On top of the master controller lies a northbound API, which allows RAN control

and management applications to modify the state of the network infrastructure (eNBs and UEs) based on the statistics and events gathered from the eNBs in the FlexRAN control plane. Such applications could vary from simple monitoring applications that obtain statistics reporting which can be used by other apps (e.g., mobile edge computing app) to more sophisticated applications that modify the state of the RAN (e.g., MAC scheduler).

## 4.3 FlexRAN Design & Implementation

### 4.3.1 Design Challenges

As already discussed, in order for FlexRAN to be an effective SD-RAN platform, it should resolve a number of challenges:

- Separation of the control from the data plane in a clean and programmable way.
- Adaptive and flexible RAN control with support for a dynamic control function placement depending on the deployment scenario and the constraints posed by the underlying network conditions.
- Support for the deployment of network applications over the controller, considering critical real-time applications (e.g. a MAC scheduler).

In the following subsections, we describe the way in which we overcame these challenges through the detailed description of the components that make up the FlexRAN platform (Figure 4.1) as well as their implementation details over the OAI LTE software platform. We present our design in a bottom-up manner, starting with the FlexRAN Agent API.

### 4.3.2 FlexRAN Agent API

All the access stratum protocols of LTE (RLC/MAC, PDCP, RRC) can be decomposed into two parts; the control part that makes the decisions for the radio link and the action part that is responsible for applying those decisions. For example, the control part of the MAC makes scheduling decisions (resource block allocation, Modulation and Coding Scheme (MCS) etc.), while the action part applies them. Similarly, part of the logic of the RRC protocol decides on UE handovers, while the actual handover operation requires RRC to perform the corresponding action. Based on this taxonomy, FlexRAN

separates the RAN control plane from the data plane by detaching the control logic from the action and consolidating all the control operations in a logically centralized controller, which in FlexRAN comprises of Master Controller and Agents interacting via the FlexRAN Protocol (see Figure 4.1). As a result, eNBs only handle the data plane to perform all the action-related functions (e.g., applying scheduling decisions, performing handovers, applying power control commands, (de)activating component carriers in carrier aggregation).

To control and manage the eNB data plane, we introduce the FlexRAN Agent API, which defines a set of functions that constitute the southbound API of FlexRAN and are the primary enabler for software-defined control of the RAN. These functions allow the control plane to interact with the data plane in five ways: (1) to obtain and set configurations like the UL/DL bandwidth of a cell; (2) to request and obtain statistics like transmission queue sizes of UEs and SINR measurements of cells; (3) to issue commands to apply control decisions (e.g., calls for applying MAC scheduling decisions, performing handovers, activating secondary component carriers); (4) to obtain event notifications like UE attachments and random access attempts; and (5) to perform a dynamic placement of control functions to the master controller or the agent (e.g. centralized scheduling at the master controller or local scheduling at the agent-side). These API calls can be invoked either by the master controller through the FlexRAN protocol using the message handler and dispatcher entity residing at the agent side (Figure 4.2) or directly from the agent if control for some operation has been delegated to it. The API calls are currently defined using the C language. A detailed list of the function call types is shown in Table 4.1.

Through the FlexRAN Agent API it becomes possible to develop two types of applications: (1) applications related to the control and management of the RAN resources like schedulers, interference and mobility managers etc. (e.g. by controlling resource block allocations, MCS, handovers and Discontinuous Reception (DRX) cycles) and (2) applications relying on the monitoring of the RAN resources to make more sophisticated decisions (e.g. adaptive video streaming based on Channel Quality Indicator (CQI), resource-block allocations for RAN sharing, etc. ) (see Sections 4.5 and 4.6.1). It should be noted that FlexRAN does not deal with the control of flows in the wired domain and therefore does not directly support related applications like routing. To enable this, FlexRAN should be coupled with corresponding flow-control solutions, like an OpenFlow-based SDN controller.

| Call Type | Description | Target | Example outcomes of calls |
|---|---|---|---|
| Configuration (Synchronous) | Get/Set configurations of target | eNB/Cell/UE/ Logical Channel | eNB ID, Number of cells, Cell id, UL/DL bandwidth, Number of antenna ports, RNTIs, UE transmission mode |
| Statistics (Asynchronous) | Request/reply statistics reporting. | List of cells/UEs | Transmission queue size, CQI measurements, SINR measurements |
| Commands (Synchronous) | Apply control decisions | Agent Control Modules (See Section 4.3.3.1) | Scheduling decisions, DRX commands, Handover initiation |
| Event-triggers (Asynchronous) | Notify control plane about changes in the data plane | Master Controller | Initiation of TTI, UE attachment, Random access attempt, Scheduling requests |
| Control Delegation (Synchronous) | Push control functions and modify their behavior at the agent side | Agent Control Modules (See Section 4.3.3.1) | Swap DL scheduler or mobility manager, Modify threshold of signal quality for handover initiation |

Table 4.1: Type of function calls in FlexRAN Agent API.

### 4.3.3 FlexRAN Controller Architecture

#### 4.3.3.1 FlexRAN Agent

In this subsection, we elaborate on the different subcomponents of the FlexRAN Agent architecture shown in Figure 4.2.

**Virtualized Control Functions.** To allow flexible and programmable control of the RAN infrastructure, the FlexRAN Agent provides a number of eNB *Control Modules* as illustrated in Figure 4.2. These modules reflect a logical separation of the control operations that an eNB in the standard LTE architecture has to perform on the radio side and can be seen as a set of individual control subsystems, each targeting a specific area of control. Since the LTE standard (the technology supported by our current implementation) already provides a precise definition and scope for such control operations through the Access Stratum protocols (RRC, MAC/RLC, PDCP), FlexRAN adopts the same structure for the agent's control modules, with each module providing functionality according to the scope of its corresponding LTE protocol (e.g., MAC/RLC control module for scheduling, RRC control module for the radio resource control).

Each control module is in turn composed of a well-defined set of functions called *Virtual Subsystem Functions (VSFs)*. The VSFs implement the action that needs to be taken by the agent during a corresponding operation. As an example, consider the

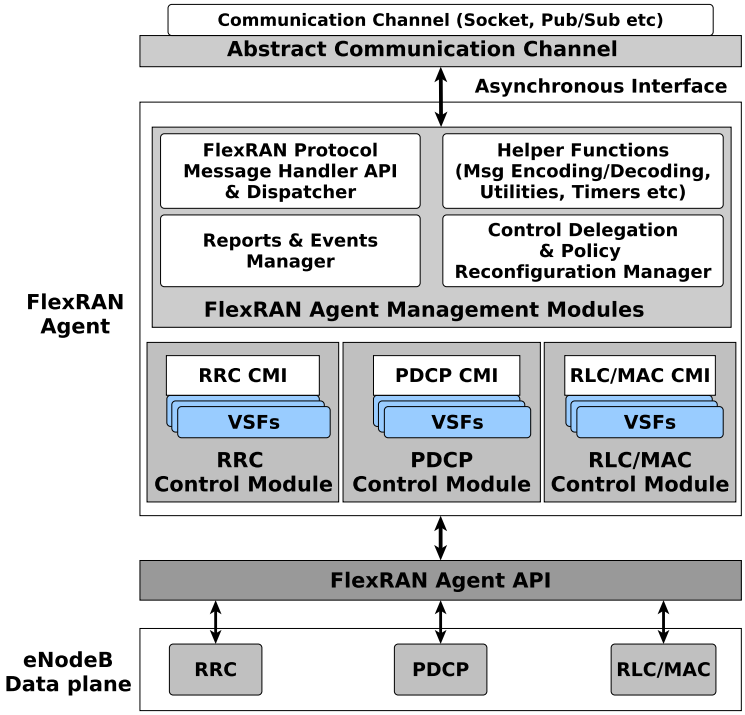



Figure 4.2: The architecture of a FlexRAN Agent.

MAC control module which is broadly responsible for various scheduling operations of the eNB. For each of the scheduling operations (e.g., UE specific downlink and uplink scheduling, broadcast scheduling), FlexRAN defines a VSF that designates how the agent should behave for the corresponding operation. For instance, the UE specific downlink scheduling VSF could designate that the agent should forward the scheduling decision sent by the master controller to the data plane or that the agent should make its own decision based on a higher-level policy defined by the master.

The number and type of VSFs that each control module supports is defined through a *Control Module Interface (CMI)*, which essentially allows the agent to abstract the set of operations of each control module from their corresponding implementations. In this way, the agent reacts to a specific event (e.g., time for downlink scheduling) without having to worry about the underlying implementation of the operation. Through this design decision, the FlexRAN agent becomes very flexible, programmable and extensible since new operations can be introduced simply by extending the CMI, while at the same time the functionality of these operations can easily be redefined in a technology-agnostic manner via the abstract FlexRAN Agent API. As already discussed, a message handler and dispatcher entity residing at the agent side (see Figure 4.2) is responsible to receive FlexRAN protocol messages from the FlexRAN master controller and forward

them to be handled by the appropriate VSF of the corresponding control module using the FlexRAN Agent API.

**Control Delegation.** FlexRAN allows the master controller to assume control of the underlying infrastructure and orchestrate its operation by making and pushing control decisions at a very fine time granularity (per subframe). While this is a feature that is essential when considering centralized time-critical applications (e.g., coordinated remote scheduling of eNBs), such fine grained control is not always desirable nor even possible. For example, FlexRAN enabled small cells may not all be connected to the master via a high-speed backhaul (e.g., optical fiber link), making the exchange of all the required FlexRAN protocol messages (MAC statistics, scheduling decisions, event notifications, etc.) at a subframe granularity a very challenging task. In such cases, it is preferable to let the individual agents make time-critical decisions as per the policy specified by the master.

A naive way to achieve this would be to identify a set of hardcoded policies at the agent for each delegated operation. For example, the agent might offer a local scheduler with two policies (round-robin and proportional fair) to choose from. If the master chooses to delegate scheduling to an agent due to an unsuitable master-agent communication channel (e.g., high latency), it would have to choose among the available hardcoded policies. However, such a mechanism can be very limiting since it does not allow the modification of the agent's behavior at runtime nor its extension with new functionality in the future. FlexRAN avoids this via two complementary mechanisms: **VSF updation** and **policy reconfiguration**.

**VSF updation** This mechanism exploits the control function virtualization feature described above and allows the master to modify the behavior of the VSFs of a control module on-the-fly by "pushing" new code to the agent over the FlexRAN protocol. This code is actually a callback assigned to one of the CMI function calls that corresponds to a specific control module - VSF pair. The callback function is able to access and modify the underlying eNB data plane using the FlexRAN agent API discussed in Section 4.3.2. A VSF update FlexRAN protocol message designates the name of the control module and the VSF that the new code is intended for and contains the actual code in the form of a shared library that has been compiled against the agent architecture. The pushed code is initially stored in a cache memory at the agent-side until the master decides to modify the behavior of the corresponding VSF. The agent cache can store many different implementations for a specific VSF, which the master can swap at run-

time. As an example, considering the case of the downlink UE scheduling VSF, the master could push two schedulers to the agent, a local proportional fair scheduler and a stub for a centralized remote scheduler, which it could switch at runtime (e.g., based on the network conditions).

**Policy reconfiguration** This mechanism is complementary to VSF updation since it allows the master to swap the agents' VSFs and reconfigure their parameters at runtime. A policy reconfiguration FlexRAN protocol message indicates the VSFs to be modified using YAML syntax (Figure 4.3). At the top level, the control module name is indicated, followed by a sequence of VSFs to be modified, each composed of two (optional) sections; *behavior* and *parameters*. The *behavior* section is used for swapping VSFs, i.e., it is an instruction to the agent to link a specific CMI function call to one of the callbacks stored at its cache through the VSF updation mechanism. The *parameters* section indicates a list of parameters that can be modified for the specific VSF. These parameters act as a public API that the controller can modify and can either refer to a single value or a sequence of values (e.g., an array). The available parameters depend on the VSF implementation (e.g., two scheduler implementations could have different sets of parameters based on their functionality).

```
<Control Module Name>:
  - <VSF Name1>:
      behavior: <callback name>
      parameters:
        parameter<i>: val
  - <VSF Name2>:
      behavior: <callback name>
      parameters:
        parameter<j>: [val1, val2, val3]
```

Figure 4.3: Structure of a policy reconfiguration message.

The control delegation capabilities through the virtualization of control functions offered by FlexRAN follow the NFV principles and indeed bring runtime service function chaining capabilities to the RAN by adding a virtualization layer over the RAN infrastructure and allowing the flexible placement of RAN control functions closer or further away from the base station based on the networking conditions, the available computing resources and the requirements of the operator in terms of performance. However, it should be noted that such capabilities relevant to the RAN are yet to be considered by NFV standards specifications, like that of ETSI NFV.

An important issue when considering control delegation via the above mechanisms is concerned with the security implications that might arise from pushing new VSFs to the agent which modify the behavior of the eNB. One way to deal with this is to force the agent to accept only code that is signed from a trusted authority, similarly to how third-party device drivers for an operating system need to be verified before installation. Since FlexRAN targets a critical part of the mobile infrastructure, getting certification from trusted authorities is expected to become a common practice for developers of VSFs; one could also envision an online VSF store similar to mobile app stores. Another measure to limit the effects of unwanted behavior from the VSFs is to allow the control modules of the agent to run in a sandboxed mode, where each VSF will need to ask for permissions to use the various parts of the FlexRAN agent API, similarly to how Android apps request the permission of the user to access different services of their device. In this way, the network operator could quickly identify VSFs that present an unexpected behavior.

**eNB Report and Event Management.** One of the responsibilities of the FlexRAN agent is to provide statistics reports and event notifications to the master for the various control modules (e.g., transmission queue sizes and random access attempts for the MAC module, radio bearer statistics and reference signal received power measurements for the RRC module). The master can use the FlexRAN protocol to make asynchronous requests for such reports and notifications and the FlexRAN agent is responsible to register these requests and to notify the master through the Reports & Events Manager (illustrated in Figure 4.2) once the results are available.

FlexRAN supports three types of reports: *one-off*, *periodic* and *triggered*. A *one-off* report is sent by the agent to the master only once as a reply to its initial request, while a *periodic* report is sent at fixed intervals that are designated in the initial request sent by the master, using the TTI as a time reference for the length of the interval. A *triggered* report is sent by the agent aperiodically and only when there is a change in the contents of the requested report. Similarly, the master can choose whether or not to be notified for a specific event occurring at the eNB by registering for it at the agent using the FlexRAN protocol. These statistics reports offer high level of flexibility in tuning the level of interaction between the agents and the master, thereby the degree of coordination. For example, triggered reports of MAC statistics is required for a remote scheduler deployed at the master. However, if the scheduling application of the controller is intended to be used as a non-real time application at a more coarse-grained timescale as a hypervisor of a local agent-side scheduler, the master can only request

periodic or even one-off reports.

**Extending OAI with FlexRAN.** We implement the FlexRAN platform over OAI [106]. OAI is (to our knowledge) the most complete open-source LTE software implementation. As such, it provides the right base to realize FlexRAN. Recall that the focus of FlexRAN is to achieve software based control of a mobile RAN that is totally decoupled from the data plane. Since we use LTE as the concrete setting for our implementation, it essentially involves using the implementation of FlexRAN as the LTE RAN control plane over the data plane implementation that is retained from the base OAI. To achieve this, we had to bypass the control plane of OAI, which does not clearly separate control and data planes, and then interface it with FlexRAN via the newly defined FlexRAN Agent API. This clean separation offered by FlexRAN simplifies the development of control applications, which now appear as modules completely separated from the data plane (based on OAI in the current implementation); this is not possible with the standard LTE RAN and therefore even with vanilla OAI. Although in our implementation the FlexRAN Agent resides over the OAI eNB data plane, it is important to note that FlexRAN as a whole (including the FlexRAN Agent) is a separate entity allowing the agent to be realized even on a physically different machine.

All the FlexRAN Agent Management Modules (Figure 4.2) and the FlexRAN Agent API were implemented from scratch, creating a basic agent stub that could be enriched with functionality through the implementation of the FlexRAN Control Modules identified earlier in this section. For our prototype the focus was on the RLC/MAC control module, due to the significant challenges that it presents in terms of its stringent time constraints. For this module, the appropriate CMI was identified and the corresponding scheduling VSFs were implemented. Even though the functionality related to these VSFs was already present in OAI, the involved control and data plane operations were tightly coupled, which was counter to the intention behind FlexRAN. To overcome this, the OAI eNB source code was refactored to allow the separation of the eNB data plane from the control logic as per the design of FlexRAN and all the required function calls were added to the FlexRAN Agent API to support this separation. For example, function calls were added to the API to obtain data plane MAC/RLC related information like transmission queue sizes of UEs, along with calls for applying scheduling decisions. To better highlight the effort required for this prototype implementation, more than 10000 lines of code were written for the agent management modules and Agent API and more than 6000 lines of original OAI code were refactored to support the control and data plane separation.

The FlexRAN agent API, as well as the CMIs are written in C, while the whole eNB/agent implementation currently supports x64 and ARM Linux systems. As a consequence, any VSF written for an OAI FlexRAN agent currently needs to be implemented in C and compiled against this architecture. The agent is multi-threaded with one thread responsible for the management of reports and events, one for dispatching the incoming protocol messages to the control modules and calling the corresponding VSFs, and two for the asynchronous network communication with the master controller.

#### 4.3.3.2 FlexRAN Protocol

The abstract communication channel is another feature of the FlexRAN agent ( Figure 4.2) for the interactions of the agent with the master. The main FlexRAN agent components communicate and exchange protocol messages with the master through an asynchronous interface that abstracts the communication operations. The communication channel implementation can vary (socket-based, pub/sub, etc.) depending on the master that the agent is interfacing with. It should be noted that the interface between the master and the agent needs to be asynchronous as the agent may need to perform certain operations in specific time intervals (e.g., a scheduling decision per TTI) while protocol messages from the master can arrive in asynchronously (e.g., receiving a request for a measurement report). In the current implementation, TCP is used for the communication of the agents with the master and the exchange of protocol messages. The protocol messages are implemented using Google Protocol Buffers [2], which provides an optimized platform-neutral serialization mechanism and allows the expression of protocol messages in a language-agnostic manner. A detailed specification of the protocol messages can be found in Appendix B.

#### 4.3.3.3 FlexRAN Master Controller

The master controller (Figure 4.4) constitutes the brain of the FlexRAN control plane as it manages the operation of the FlexRAN agents. In FlexRAN, we employ a custom design for the master instead of using a conventional OpenFlow-based one and this is for two reasons: (1) the nature of control in the RAN is tied to a significant extent to the control of radio resources which cannot be effectively captured by the flow abstraction; (2) the RAN presents a requirement for the support of real-time applications with very quick reaction times, a feature not essential for SDN control in the wired domain.

**Management of network information.** The RAN Information Base (RIB) is a key component that maintains all the statistics and configuration related information about the underlying network entities, i.e. UEs, eNBs and FlexRAN agents. The RIB is always loaded in memory for improved performance and is structured as a forest graph. The root node of each tree in this forest is an agent associated with the master, while the nodes of the second level are the cells associated with a specific agent/eNB. Finally, the leaves of the trees are the UEs associated to a specific (primary) cell. It should be noted that the current implementation of the RIB does not provide any high-level abstraction for the stored information, revealing raw data to the northbound API.

**Support for real-time applications.** The applications as well as the Events Notification Service of the master consult the RIB to perform any operation, but they do not modify the RIB directly. Instead they issue control commands through the northbound interface, which indirectly affect the RIB state through the modifications performed to the eNB data plane and the agent state. These modifications are reflected back to the master through the statistics reports and event notifications sent by the agents. Only the RIB Updater component of the master can update the RIB with the information received from the agents (Figure 4.5). This design decision improves the support of real-time applications (e.g., MAC scheduler) that must be non-blocking in order to meet their time constraints (e.g., a TTI for the scheduler). Allowing all components to modify the RIB could give rise to write conflicts, negatively affecting the system performance. Having just a single writer (the RIB Updater) and multiple readers helps avoid this problem. The update frequency of the RIB depends on the way the FlexRAN agent reporting and event notification mechanism is configured by the master.

Since the FlexRAN controller is designed to support real-time RAN applications, a Task Manager is responsible for handling all the tasks (both applications and core management modules) running on the master. More specifically, it is responsible to start, stop and pause applications; to assign priorities to running services and to allow them to execute based on their time constraints. For example, the Task Manager would assign a very high priority to a centralized MAC scheduler running on the master, whereas a non time-critical monitoring application would get a lower priority. To support real-time control, the Task Manager is implemented as a non-preemptive thread running in an infinite loop and operating in cycles of length equal to a TTI, where each cycle is composed of two slots — one for the execution of the RIB Updater (e.g., 20% of the TTI) and the other for the execution of the applications as well as the Event Notification Service threads (e.g., 80% of the TTI). This guarantees the mutual ex-

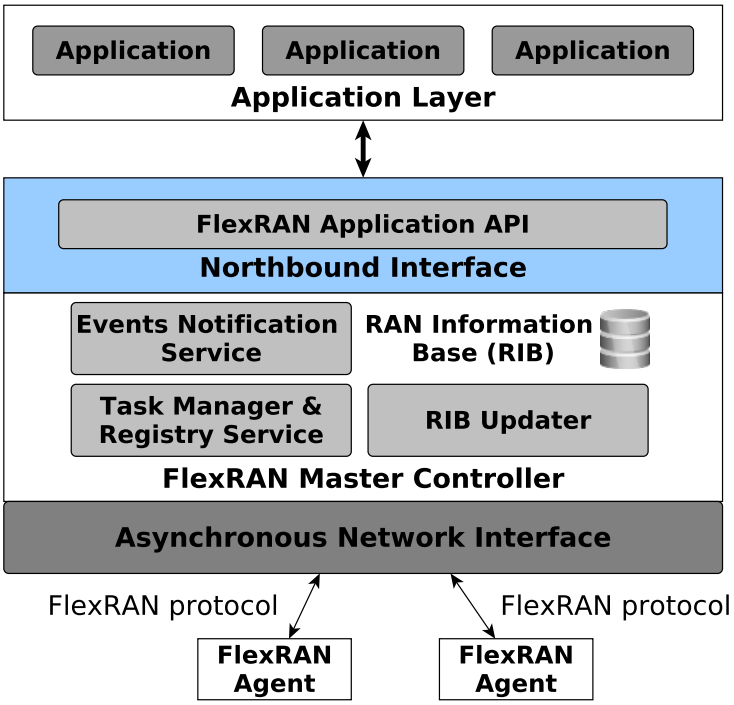



Figure 4.4: Components of the FlexRAN master controller and its interface to the application layer.

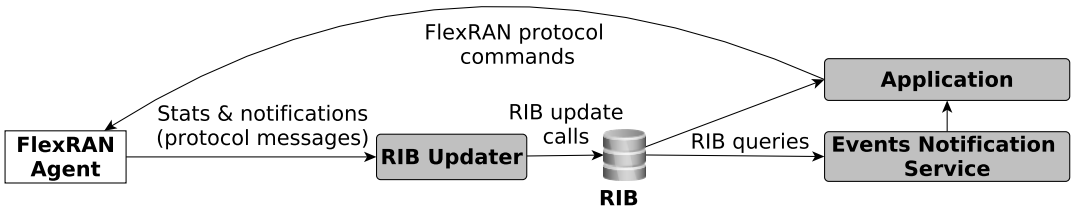



Figure 4.5: Flow of information for updating the RIB.

clusion of the reads and writes in the RIB and allows the applications to operate in a non-blocking mode while accessing the RIB, ensuring the real-time operation of the controller.

**Master controller implementation.** The master controller is implemented from scratch using C++ and currently supports x64 Linux systems (kernel $\geq$ 3.14 for support of real-time applications). The master can operate in a non real-time mode, supporting only applications that are not time-critical, with the advantage of being more lightweight. The Task Manager in the non-real time mode does not enforce a strict duration of the cycle as tasks are not scheduled with a real-time priority and thus could take longer than a TTI.

### 4.3.4 Northbound API and Applications

RAN control and management applications in the application layer communicate with the master through the northbound interface (Figure 4.4), which allows the applications to monitor the infrastructure through the information obtained from the RIB and apply their control decisions through the agent control modules. The applications run as threads and use the FlexRAN Application API (currently in C++) to register with the Registry Service of the master, access the RIB and send control messages to the agents. Applications can be divided into two categories: (*periodic* or *event-based*). Periodic applications employ a periodic execution pattern (e.g,, a periodic scheduler) whereas the execution of event-based applications is triggered by specific events (e.g., a mobility manager that expects changes in the received signal strength of a UE to react). The Events Notifications Service of the master controller notifies the applications (mainly of the event-based type) about any changes that might have occurred on the agent side. Some applications could fall into both categories and it is ultimately the application developer who would choose the appropriate execution pattern.

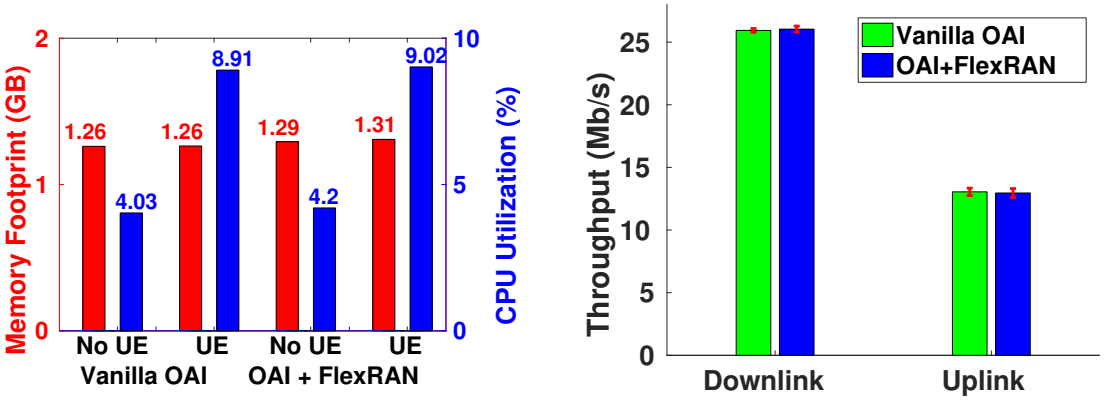
## 4.4 System Evaluation

In this section we evaluate the engineering decisions and the design choices behind FlexRAN. For the experiments, a FlexRAN master controller was deployed, in which one or more agent-enabled eNBs (depending on the experiment) were connected through dedicated Gigabit Ethernet connections. Each eNB was also connected to a machine acting as the EPC, running the corresponding EPC software implementation (openair-cn [112]). All the test machines were equipped with quad-core Xeon CPUs at 3.4GHz and 16GB of RAM. Depending on the experiment, the testbed was used either in emulation mode (emulated PHY layer and emulated UEs) or with a real RF front-end (Ettus B210 USRP) and a physical UE (Nexus 5 smartphone). All the experiments were conducted with the same eNB configuration, namely FDD with transmission mode 1 and 10MHz bandwidth in band 5. For all the experiments in this section, and unless stated otherwise, TCP traffic was generated using iperf to saturate the network.

### 4.4.1 Comparison to Vanilla OAI

We begin by investigating the overhead introduced to an eNB by the addition of the FlexRAN agent in terms of memory and CPU utilization (Figure 4.6a) comparing a

vanilla OAI eNB and a FlexRAN-capable eNB in two cases; one with the system in idle state and one with a UE connected and generating traffic to saturate the network. The memory and CPU utilization are measured using standard information provided by Linux distributions, i.e. memory information provided by the *top* tool and CPU utilization obtained by measuring the jiffies allocated to the RAN-related processes and dividing this value with the total jiffies of all the CPUs for all system processes during the course of the experiment. As we can observe, there is a very slight increase in the memory footprint and the CPU utilization in the FlexRAN case, due to the threads used for the operation of the agent and the protocol messages exchanged with the FlexRAN master controller. Despite the aforementioned overhead incurred from the FlexRAN agent, the communication of the eNB with the UE is fully transparent, with the UE experiencing the same service quality in its connection as with the vanilla OAI (Figure 4.6b).




(a) Normalized CPU utilization (8-cores) and memory footprint with(out) UE

(b) Downlink and Uplink throughput comparison

Figure 4.6: Comparison of vanilla OAI to FlexRAN .

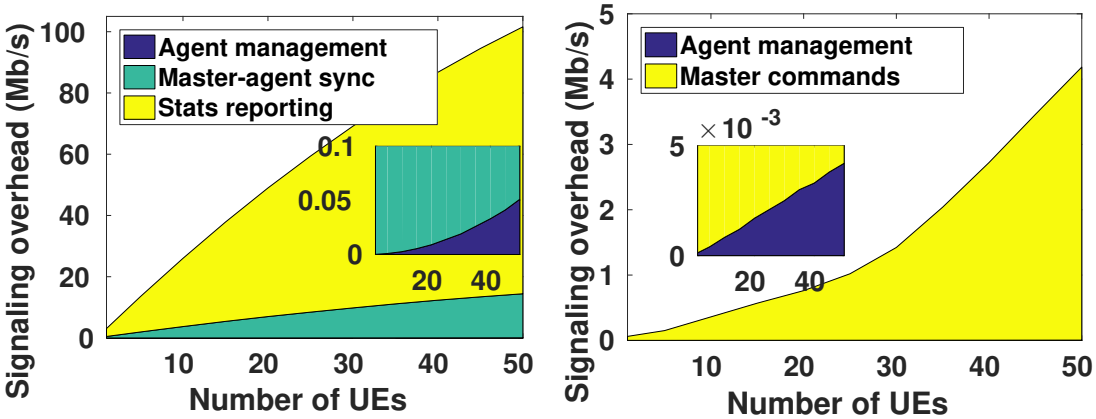
### 4.4.2 Scalability

Next, we evaluate how FlexRAN scales as the number of eNBs and UEs increases. For these experiments we used OAI in emulation mode with the PHY layer abstracted in order to perform tests with a large number of UEs. It is noted that this choice has a minimum effect on the obtained results, since the focus of our evaluation was on operations occurring above the PHY which were unaffected by the emulation.

#### 4.4.2.1 Controller-agent signaling overhead

One very important element regarding the scalability of FlexRAN is the network overhead incurred by the FlexRAN protocol, especially when support for real-time applications is required. For this, we measured the signaling overhead between the agent and the master in the demanding case of deploying a centralized scheduling application and for a varying number of UEs using the *iftop* network monitoring tool. To study the system's scalability, we tested a scenario with the worst case configuration signaling-wise, where statistics reports were sent from the agent to the master every TTI, the centralized scheduler undertook all scheduling decisions at a TTI granularity and the master was fully synchronized with the agent at a TTI level using the appropriate FlexRAN protocol synchronization messages. During the experiment uniform downlink UDP traffic was generated for all the UEs, in order to force the centralized scheduler to frequently send scheduling decisions to the agent.

The agent-to-master network overhead for 50 UEs can reach 100 Mb/s (Figure 4.7a). The main source of this overhead are the periodic statistics reports (buffer status reports, CQI measurements etc.) followed by the master-agent synchronization messages, with the overhead of the agent management related messages being negligible. One important thing to notice is that the agent-to-master signaling overhead increases sublinearly with the number of connected UEs due to the aggregation of relevant information in the FlexRAN protocol messages (e.g. list of UE status reports) and their optimized serialization by the Google Protocol Buffers library.




(a) Agent-to-master

(b) Master-to-agent

Figure 4.7: Signaling overhead for the communication between the master and the agent using the FlexRAN protocol.

In the case of the master-to-agent signaling (Figure 4.7b), the overhead is much lower compared to the previous case (less than 4Mbps) and comes almost entirely from the scheduling decisions sent by the centralized scheduler. In contrast to the previous case, this overhead is increasing with a higher rate as the number of UEs goes up. The reason is that the larger the number of UEs, the less resources are available for scheduling each UE and therefore more TTIs will be required for scheduling, leading to an increase in the overall number of scheduling decisions sent by the controller to the agent.

The aforementioned results show that FlexRAN is suitable even in demanding scenarios like multi-cell scheduling on a per-TTI basis, where depending on the deployment (macro or small-cell) the agent can be connected to the master either through a dedicated high-bandwidth channel (e.g. optical fibers) or through a lower bandwidth channel (e.g. a VDSL connection). In practice, the controller will apply policies or delegate the scheduling to the agent, which will significantly reduce the network overhead. This overhead could be further reduced through agent configuration changes. For example, by setting the periodicity of the MAC reports to 2 TTIs, this overhead could be reduced to almost half without any significant impact on the system's performance. Other methods that could be considered to reduce this overhead could be compression algorithms for the protocol messages or event-triggered instead of periodic message transmissions.

#### 4.4.2.2 Master controller resources

Using the previous setup, we measured (Figure 4.8) the requirements of the master in computing resources and memory for a varying number of connected agents (16 UEs/eNB). As already discussed, the master operates in TTI cycles, where part of each cycle is allocated to the execution of applications (80% in this experiment) and the rest to the execution of the core components of the master. As we can observe, the operation of the master is lightweight, with only a small fraction of the total cycle being actually utilized. The execution time of the core components increases as we add more agents, mainly due to the increase in the updates that need to be performed by the RIB updater. The memory footprint of the master is also very small and its increase is mainly related to the increase of the RIB size.

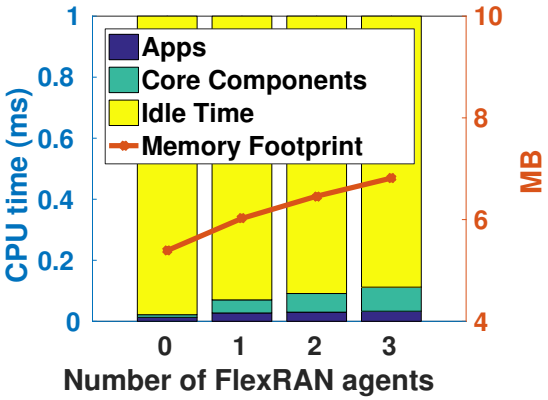



Figure 4.8: Utilization of master TTI cycle and memory footprint (16 UEs/eNB).

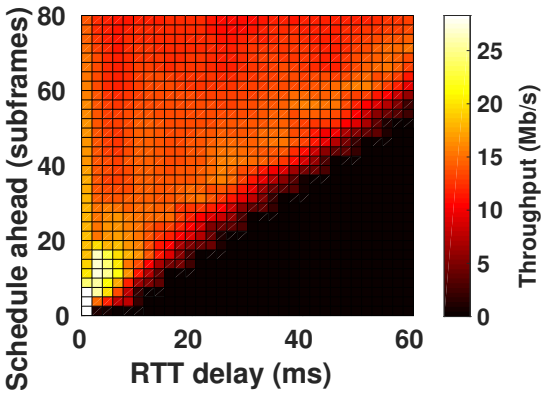



Figure 4.9: Effect of latency and scheduling ahead time on downlink throughput.

### 4.4.3 Control channel latency impact

One very important aspect for real-time control in FlexRAN is the impact of the control channel latency between the master and the agent. To test its effect we used a physical UE scheduled in the downlink by a centralized scheduling application running at the master. The scheduler was implemented so that it could be parametrized to make scheduling decisions $n$ subframes ahead of time, meaning that the scheduler would observe the MAC state (transmission queue sizes of UEs, signal quality etc) at subframe $x$ and would issue a scheduling decision that should be applied by the data plane in subframe $x+n$. The scheduling decision will be valid and can be applied by the agent only if its designated time is greater than the latency in the master-agent control channel.

Based on this setup we modified the schedule ahead parameter of the application and the latency in the control channel using the *netem* tool [48] and measured the UE downlink throughput for various configurations (Figure 4.9). As we can observe, the lower triangular region of the figure depicts a throughput of 0, indicating that for these configurations the UE was unable to complete network attachment. This is due to the one-way delay of the control channel being greater than the schedule ahead time, meaning that scheduling decisions always miss their deadline. Moreover, the control channel delay affects the synchronization between the master and the agent, since the agent subframe reported to the master is always outdated by an offset equal to half the Round-Trip Time (RTT). Since the scheduler relies on this outdated value to make a scheduling decision, the scheduling ahead time should take it into account. Assuming

a symmetrical RTT, the schedule ahead time should be set to a value that is at least equal to it; half to make up for the outdated subframe value reported by the agent and half to ensure that the scheduling decision will be valid at the time it arrives to the agent.

In the upper triangular region we have all the cases in which the controller application successfully manages to schedule the UE. In this case the application is able to schedule the UE even for a control channel with a very high latency, as long as the schedule ahead parameter is configured properly. As the RTT delay and the schedule ahead time increases, the throughput gradually drops. One reason for this is that higher RTT delays make the information stored in the RIB (e.g. CQI measurements) more outdated, leading to wrong scheduling decisions (e.g. due to a bad MCS choice) that could affect the throughput. Moreover, for increased values of the schedule ahead parameter, the scheduler needs to make predictions further into the future, while making assumptions about the outcome of previous transmissions for which it has not yet received any feedback. Depending on the use case and on the scheduling performance requirements, one could choose to either use approximation methods like scheduling ahead of time to mitigate the effect of latency or to delegate control to the agents for the time critical functions that are affected by this latency as long as coordinated operation among the eNBs is not a hard constraint.

### 4.4.4 Control delegation performance

Since control delegation is one of the most important features of FlexRAN we evaluated the mechanisms of VSF updation and policy reconfiguration in terms of efficiency and service continuity. For this experiment we used the same setup as in Section 4.4.3, starting with a centralized scheduler running as an application at the master. At the same time, we built an equivalent (in terms of functionality) local scheduler, as a VSF for the MAC control module of the agent, using the FlexRAN agent API. The experimental scenario involved the pushing of this code to the agent using the FlexRAN protocol and the dynamic switching between the local and the remote scheduler through the policy reconfiguration mechanism.

Using this setup, we tested the downlink throughput of the attached UE, while swapping the local and the remote schedulers with various frequencies down to the TTI level (1ms), and observed the same application performance of 25 Mbps, as illustrated in Figure 4.10. The code is pushed to the agent-side only once and is stored in its

local cache meaning that no additional overhead is incurred for the control delegation. Moreover, the absolute VSF load time required to swap between the local and remote scheduler is very small (˜103ns) and therefore does not disrupt service continuity as it only forms an insignificant fraction of the overall TTI.

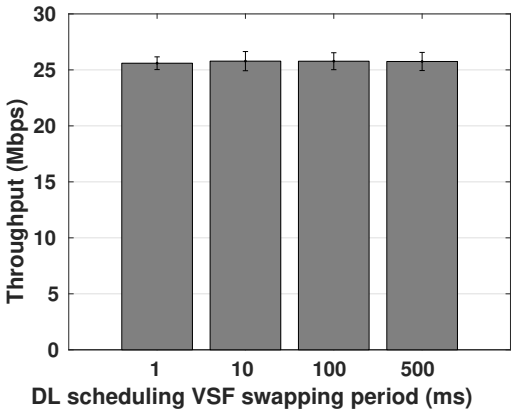



Figure 4.10: Downlink throughput of UE during frequent downlink scheduler policy reconfigurations

## 4.5 FlexRAN Use Cases

To date, the lack of an implemented SD-RAN platform meant that there was no way to actually study SD-RAN benefits and use cases. To demonstrate the usefulness of FlexRAN towards this end, we now present diverse use cases that SD-RANs in general and FlexRAN in particular can enable.

### 4.5.1 Interference Management

One of the ways to cope with the requirement for higher throughput in the RAN is the creation of HetNets composed of multiple small cells within the area of a macro cell. However, such dense network deployments are more susceptible to interference. One proposed solution is eICIC [38] that introduces the concept of Almost-Blank Subframes (ABSs) during which the macro cells are muted to allow small-cells to transmit user traffic. Even though eICIC is one effective way to manage interference, the semi-static configuration of ABSs can lead to an underutilization of the radio resources when small cells are idle. To remedy this, we consider an optimized eICIC mechanism that allows the macro-cell to exploit periods of inactivity of the small-cells to transmit to

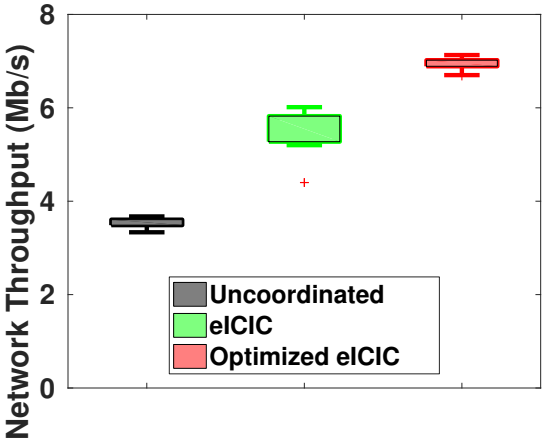
UEs even during ABSs as long as no small-cell is transmitting at the same time. Such a mechanism requires a high-level of coordination among cells which cannot be easily achieved using the traditional X2 interface [30] [13]. To deal with this, we build a downlink UE scheduling application over FlexRAN that exploits its consolidated control plane to implement this optimized eICIC mechanism.

Specifically, we implemented a centralized scheduler application on top of the master and two different types of local agent-side downlink schedulers, one for the macro-cell and one for the small-cells of a region. During a non-ABS, the macro-cell eNB performs the scheduling using its agent-side scheduler, while the agent-side schedulers of the small-cells remain inactive exactly as in a normal eICIC case. However, during an ABS the centralized scheduler at the master performs a coordinated UE scheduling of all cells and decides whether the macro or the small cells should be scheduled, always giving priority to the small-cells. The scheduling decisions are then pushed to the agents using the FlexRAN protocol and are applied by the agent-side schedulers that during an ABS act as stubs of the centralized scheduler.

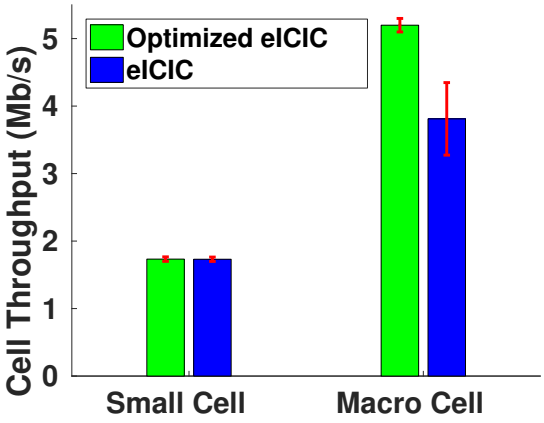
Two agent-enabled OAI eNBs were used, one acting as a macro-cell and one as the small-cell, both running in emulation mode over the same physical machine. The reason we resorted to emulation over the same machine is that eICIC requires a microsecond-level synchronization of eNBs, which was not supported by our hardware. Moreover, due to a limitation of the current OAI implementation, the association of UEs over different eNBs is not supported in emulation mode when the PHY is abstracted. This forced us to use the more computationally intensive full-PHY emulation mode of OAI (that involves convolution of the real PHY signal with an emulated-channel in real time) and to limit the number of emulated UEs used for this experiment to 4.

One UE was associated with the small-cell and three were associated with the macro-cell. Downlink UDP traffic was generated uniformly for all the UEs and the overall network throughput was measured (Figure 4.11a) in three cases: (i) an uncoordinated case, where each eNB performed scheduling independently, (ii) a simple eICIC use case with 4 ABSs and (iii) the optimized eICIC use case with the same ABS configuration as case (ii). The optimized eICIC use case almost doubled the overall network throughput over the uncoordinated scenario and had an improvement of about 22% over the simple eICIC scenario. The reason for the difference between the two eICIC use cases is that while the small-cell throughput remains the same (Figure 4.11b), the throughput of the macro-cell increases under optimized eICIC because the ABS not

used by the small-cell are assigned by the centralized scheduling application to the macro-cell.




(a) Downlink network throughput in uncoordinated, simple and optimized eICIC.




(b) Overall macro and small cell downlink throughput for simple and optimized eICIC.

Figure 4.11: Throughput benefits of optimized eICIC.

### 4.5.2 Mobile Edge Computing

For our second use case, we consider FlexRAN as a deployment platform for MEC applications. MEC allows developers and content providers to deploy their services over the network edge. This presents many benefits including ultra-low latency, high bandwidth and real-time access to radio network information which can be used by applications for optimization purposes [62]. Such applications are expected to be deployed in a centralized manner and therefore doing this using the conventional LTE architecture becomes a challenging task. Moreover, their deployment assumes the existence of a programmable network, which is naturally enabled by FlexRAN.

In this context, we show how the consolidated control plane and the real-time network information provided by FlexRAN can be beneficial for services such as video streaming. To show this, we used the DASH [130] streaming service and studied the effects of fluctuating a mobile's signal strength to the video streaming bitrate adaptation performed by the DASH reference client [37]. The idea of the experiment is that the channel quality reported by the UEs to the eNB through a CQI value (in the range [0,15]) indicates the MCS that the scheduler should use for the data transmissions of a UE and therefore has a direct impact in its highest achievable throughput. Knowing

this can allow the streaming service to make smarter decisions regarding the optimal bitrate compared to a case when only transport layer information is available.

In order to obtain reproducible results, we emulated the fluctuations of the channel quality between the eNB and the UE and measured the maximum achievable TCP throughput of a physical UE for various fixed CQI values. Moreover, for the same CQI values, we used the reference DASH client and the available test videos that it offered, to measure the maximum sustainable bitrate of a video stream (i.e. a bitrate that would never lead to buffer freezes) for the offered bitrate levels. The measurement results (Table 4.2) indicate that the TCP throughput needs to be greater (even double) than the video bitrate in order to always maintain a high quality; this is consistent with related observations in the literature [135].

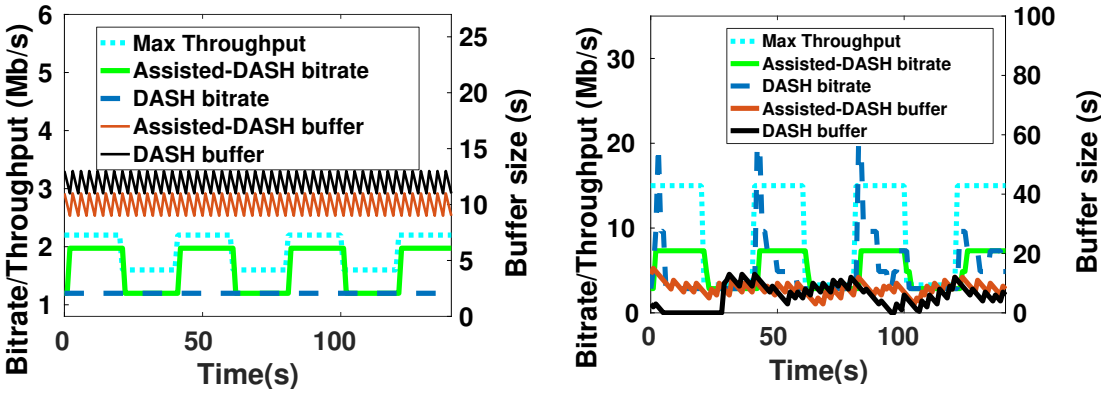
| CQI | TCP Throughput (Mbps) | Max sustainable bitrate (Mbps) |
|---|---|---|
| 2 | 1.63 | 1.4 |
| 3 | 2.2 | 2 |
| 4 | 3.3 | 2.9 |
| 10 | 15 | 7.3 |

Table 4.2: Measurements of max TCP throughput and max sustainable bitrate of video stream for various CQI levels.

We used FlexRAN to implement a simple MEC application that uses the RIB to obtain real-time information about the CQI values of the attached UEs. The application computes an exponential moving average of the UE CQI and maps it to the optimal video bitrate based on the measurements of Table 4.2. The bitrate is then forwarded through an out-of-band channel to a modified version of the DASH reference client where it is used to adapt the video stream's quality. To simplify things, we performed the experiment in an ideal setting, where the channel quality fluctuation was the only cause for a change in the bitrate (a single UE attached to the network with the DASH video being the only source of traffic).

For our experiment, we considered two cases. In the first, we used a video [3] with three bitrates (1.2, 2 and 4 Mbps) and we introduced a small variation in the CQI value (from 3 to 2 and vice versa). In the second, we used a 4K video [4] with six available bitrates (2.9, 4.9, 7.3, 9.6, 14.6 and 19.6 Mbps) in which the CQI value was changed drastically (from 10 to 4 and vice versa). In both cases, we measured the bitrate selected by the default and the FlexRAN-assisted players in comparison to the maximum achievable TCP throughput, as well as their buffer sizes (Figure 4.12). In the

first case (Figure 4.12a), the default player always kept the bitrate at the lowest value (1.2 Mbps), even in times when the available throughput increased by 40%, meaning that the change in channel quality did not become apparent to the transport layer. On the other hand, the FlexRAN-assisted player exploited the information obtained by the RAN and managed to better adapt to the changing network conditions. Therefore, even though neither player experienced buffer freezes, the default player underutilized the available resources. In the second case (Figure 4.12b), the default player aggressively attempts to increase the bitrate when the CQI increases, setting it to 19.6 Mbps, even though the maximum achievable throughput is 15 Mbps. This quickly results in TCP congestion, leading the player to significantly lower the bitrate (lower than the max sustainable bitrate) in order to adapt, while the video buffer is frequently empty (e.g. sec 5-30). This has a significant impact to the Quality of Experience (QoE), since an empty buffer means that users experience video freezes. On the other hand, the MEC application running over FlexRAN can quickly identify the maximum sustainable bitrate (7.3 Mbps) given the CQI measurements observed at the RIB and does not follow the same aggressive behavior as the default player, leading to a more stable video stream that avoids buffer freezes.




(a) Low throughput variability

(b) High throughput variability

Figure 4.12: Rate adaptation of DASH vs FlexRAN assisted DASH and corresponding buffer sizes.

The MEC application and the metrics considered in the experiments of this use case are intended to highlight the benefits of an SD-RAN design by focusing on QoE aspects that are linked to the smooth play of video streams (avoiding video freezes), in line with the philosophy and the experimental methodology of works like [140]. However, an SD-RAN design could also be exploited to improve the QoE of users in

other ways, like for example through the content-aware schedulers in [81] and [113], that correlate the video quality with the content of the video. It should be noted that in such cases, the perceived benefits of the users in terms of QoE could be evaluated through different quality metrics like PSNR or SSIM, rather than the use of bitrates and buffer sizes.

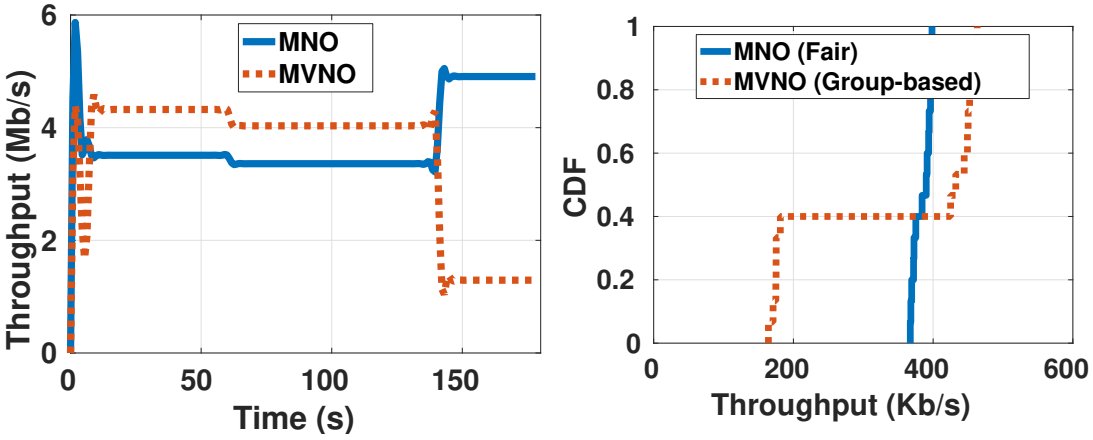
### 4.5.3 RAN Sharing & Virtualization

A side effect from the densification of cells is the increase of the infrastructural CAPEX and OPEX. This leads to the creation of new business models, where multiple MNOs share the same *passive* infrastructure such as masts and backhaul links in order to save costs. On top of that, a second level of *active* sharing can happen, where MNOs share the network equipment as well as provide wholesale access to MVNOs, allowing them to provide voice and data services using part of the available resources [35]. However, this can pose a significant challenge for the management of the RAN, since the requirements of operators in terms of the radio resources and the applied policies of scheduling and mobility management can constantly change based on the needs of their subscribers and the underlying setting. The control and management of such operations can be greatly simplified through the introduction of programmability in the RAN.

Based on this, we used FlexRAN as an enabler of active RAN sharing and on-demand resource allocation over an LTE network. More specifically, exploiting the capabilities of FlexRAN , we implemented a downlink UE scheduler for the agent-side that supports the dynamic introduction of new MVNOs to the RAN and the on-demand modification of the scheduling policy per operator. An application running at the master exploits the policy reconfiguration mechanism of FlexRAN to modify the parameters of the agent-side scheduler (scheduling policy and number of resource blocks per MVNO). To test this and in order to be able to support a large number of UEs, we used a single agent-enabled OAI eNB in emulation mode with the PHY abstraction enabled. We configured the agent-based scheduler to support one MVNO that shared the available resources with the RAN's MNO.

For our first experiment, each operator was assigned 5 UEs for which uniform UDP downlink traffic was generated, while the percentage of radio resource blocks allocated to each operator was dynamically adjusted based on their requirements. The results in terms of the total throughput per operator are illustrated in Figure 4.13a. Initially,

the MNO was allocated 70% of the radio resources and the MVNO was allocated the remaining 30%. At 10s, the master controller application sent a policy reconfiguration message, setting the available resources of the MNO to 40% and of the MVNO to 60%, simulating a brief requirement for additional resources for the MVNO. Then, at 140s, the application sent a second policy reconfiguration message that re-adjusted the radio resources so that 80% were allocated to the MNO.




(a) Dynamic allocation of resources

(b) CDF of UE throughput based on scheduler

Figure 4.13: Policy reconfiguration for MVNO management

As a second experiment, we implemented an agent-side scheduler supporting a fair scheduling policy and a group-based policy of premium and secondary users, with 70% of the resources allocated to the premium users and the rest to the secondary. One MNO and one MVNO was employed, where the MNO was assigned the fair policy and the MVNO the group-based one. Each operator was allocated half of the available radio resources and was assigned 15 UEs. In the group-based MVNO case, nine UEs belonged to the premium group and the remaining six acted as secondary users. We generated uniform UDP downlink traffic for all the UEs and measured the throughput of each UE per operator. The results are illustrated in the CDFs of Figure 4.13b. In the case of the MNO, all the UEs had a throughput of about 380Kb/s due to their fair scheduling policy. On the other hand, in the case of the MVNO, UEs assigned to the premium group had a throughout of about 450Kb/s, while the UEs of the secondary group achieved a throughput of less than 200Kb/s.

It should be noted that the active RAN sharing use case considered here is one of the extreme but not ideal design points for RAN slicing. In Chapter 5, a more ideal form of RAN slicing that can provide both functional and performance isolation among slices will be presented in the form of Orion.

## 4.6 Discussion

### 4.6.1 Other Example Use Cases

Here we make a brief discussion on a non-exhaustive list of further use cases that were currently not implementable due to the limitations posed either by the OAI platform underlying FlexRAN or by the current LTE specifications.

**Mobility Management.** In current practice, handover decisions mainly rely on the signal strength of mobile devices. However, the centralized network view offered by FlexRAN could enable more sophisticated mobility management mechanisms that consider additional factors, e.g., the load of cells or use-driven resource requirements of mobile devices, leading to an optimization of the device-network associations.

**Device Centric Networking.** Another interesting application of FlexRAN could be in the domain of device-centric networking [21]. In this, the association of mobile devices is decoupled from the cell, e.g. by using different cells for control and data traffic, simplifying the adoption of new communication paradigms like D2D. While realizing such a paradigm would also require changes in the physical layer, its deployment could greatly be simplified through the centralized control of FlexRAN, as it inherently involves the coordinated control of multiple network nodes, e.g. control traffic through one cell and data traffic through another.

**Spectrum Sharing.** Shared spectrum access is seen as a promising solution that allows operators to cope with the high increase of mobile data traffic. One spectrum sharing mechanism that is of particular interest to MNOs is LSA [69]. It enables incumbents not concerned with civilian wireless and mobile data communications to authorize other users (e.g. the MNOs) to access all or part of the spectrum allocated to them for designated periods of time and in designated places based on some agreement. An LSA controller dynamically manages the access to the shared spectrum based on these agreements. Such an operation could easily be implemented as an application on top of FlexRAN.

### 4.6.2 Adaptability Beyond LTE

As already discussed in Section 4.2, FlexRAN was presented in the context of LTE for concreteness and to match its current implementation. However, the design and the mechanisms supported by FlexRAN are not LTE-specific and are therefore equally

suitable for future mobile RAN architectures. More specifically, the mechanisms of virtualized control functions and control delegation are technology-agnostic and use API calls and protocol messages that are completely decoupled from the underlying technology. The main difference that these mechanisms would present in the context of another technology is that the number and type of the control modules and VSFs on the agent side would change to reflect the capabilities and needs of the new technology (e.g. no PDCP module for WiFi). However, the mechanisms of policy reconfiguration and VSF updation make no assumption about the exact type and structure of the control modules at the agent side, allowing the controller to designate it on the fly based on the underlying implementation (see Figure 4.3).

Apart from the technology agnostic part of the agent API, FlexRAN also requires a number of technology specific API calls. For example, LTE requires scheduling commands which are not applicable and therefore are not required in the WiFi domain. This means that for FlexRAN to be fully compliant with other technologies, the FlexRAN Agent API needs to be extended with additional function calls specifically tailored for the domain of interest, closely resembling the idea of device drivers found in operating systems. The more extended the API is for a specific technology, the more capabilities are offered for the control of the underlying RAT. It should also be noted that the FlexRAN protocol has been structured in such a way that it could be easily extended to support new messages that are technology specific without affecting the functionality of the existing ones.

Another important issue is the extensibility of FlexRAN in future 5G fronthaul architectures where base stations are expected to adopt a C-RAN design, with RRUs decoupled from the BBUs and connected through a fronthaul link (e.g. optical fibers) [30]. In such an architecture, the functional split between the RRU and the BBU is expected to play a significant role, depending on the capacity and the latency of the fronthaul link. The design of FlexRAN could be adopted in such an architecture, where the data plane and the agents could be placed on the RRU side and the control plane could be consolidated at the master residing at the BBU. Based on the quality of the fronthaul link, the control delegation features of FlexRAN could be exploited to perform a dynamic and adaptive functional split of the control operations while retaing the real-time deadlines. For example, in case of a high latency fronthaul link, the master could delegate control of the time critical PHY and MAC operations to the RRU, while in case of a low latency link all operations could be residing at the BBU side.

## 4.7 Conclusions

In this chapter we have presented FlexRAN, a flexible and programmable SD-RAN platform. FlexRAN enables the separation of the control from the data plane through a custom-tailored southbound API, while providing inherent support for real-time RAN applications. FlexRAN offers significant benefits to the RAN, including the flexibility to dynamically modify the degree of coordination among base stations to realize both distributed and centralized modes of operation and the programmability to adapt control over time turning the RAN into an evolvable network. All these while being transparent to end-devices, simplifying its deployment and promoting innovation both for the industrial and the academic research community. The implementation of FlexRAN over the OAI LTE platform and our evaluation results confirm the feasibility of its deployment even when considering time critical operations like MAC scheduling. The effectiveness of FlexRAN as an SD-RAN platform was highlighted through a diverse set of use cases in the context of current 4G and future 5G networks, focusing on interference management, mobile edge computing and RAN sharing.

# Chapter 5

# Orion: A RAN Slicing System

## 5.1 Introduction

The softwarization capabilities introduced in the RAN through FlexRAN can help in improving the flexibility and the adaptability of the network in terms of its control and coordination, while allowing the use of generic hardware to achieve this. However, these capabilities alone are not enough to enable the multi-service environment envisioned in the context of 5G, where multiple slices, with very diverse performance requirements must co-exist over a common underlying infrastructure, sharing the available resources.

In this chapter, we focus on the problem of introducing virtualization capabilities in the RAN in order to enable the slicing of the network based on the diverse needs of the deployed services. As already stated in Section 1.4.2, RAN slicing is at a premature stage, with the main challenge being that apart from computing, storage and network resources, the limited radio resources need to also be virtualized and assigned to the slices. This must be achieved while guaranteeing the functional and performance isolation between tenants and the infrastructure provider, and among tenants themselves, so that these tenants can maintain full control and independence of their slices to tailor them to their respective service requirements.

Focusing on this problem, this chapter presents Orion, which in our knowledge is the first RAN slicing system that provides full functional and performance isolation, while also facilitating the efficient sharing of the radio and spectrum resources. To achieve this, among others, Orion leverages the software-defined capabilities of FlexRAN and introduces a novel set of abstractions for the virtualization of the radio resources among tenants. The concrete implementation of Orion can be a useful tool

for further research and experimentation in 5G, since, combined with existing slicing solutions for the mobile core, it can act as an enabler of end-to-end mobile network slicing.

This chapter begins by providing a high-level overview of Orion in Section 5.2. This is followed by a detailed presentation of the design and implementation aspects of the system in Section 5.3 and a thorough evaluation of its performance compared to the state-of-the-art in Section 5.4. The benefits and the capabilities of Orion are further investigated through a set of case studies in Section 5.5. Finally, the extensions in the design of Orion in order to support OTT service providers are discussed in Section 5.6, accompanied by a set of evaluation results that demonstrate the added benefits of the system's extended form.

## 5.2 Orion Overview

The core contribution of this chapter, Orion, is a novel RAN slicing system design that is in line with the spirit of network slicing and the needs of a flexible and cost-effective multi-service mobile network architecture – isolation among multiple virtual networks provides the necessary flexibility to customize and control a slice, whereas efficient sharing of the underlying physical infrastructure allows supporting diverse services in a cost-effective manner. Simultaneously being able to satisfy both these concerns of functional isolation between slices and efficient resource use in the context of the RAN is the main, as yet unresolved, challenge addressed by Orion.

Towards this end, Orion's design (Figure 5.1) explicitly distinguishes the infrastructure provider from the service providers (the slice owners). The infrastructure provider is the owner of physical base stations, comprising of hardware resources (i.e., radio equipment, processing, memory, and network) and a chunk of spectrum. While it can be realized either via dedicated specialized hardware or in a cloud environment using re-programmable hardware (e.g., C-RAN BBUs and RRUs), each physical base station in our model supports a single RAT, meaning that all radio and spectrum resources available at the base station can be exploited through a *shared* physical layer. Note that, for concreteness in the description of Orion's design and implementation, we consider LTE as the underlying RAT and downlink scheduling as a running example throughout.

The `Base Station Hypervisor` that sits over the physical layer is the heart of Orion's design. It is the component used for managing RAN slices, for ensuring their

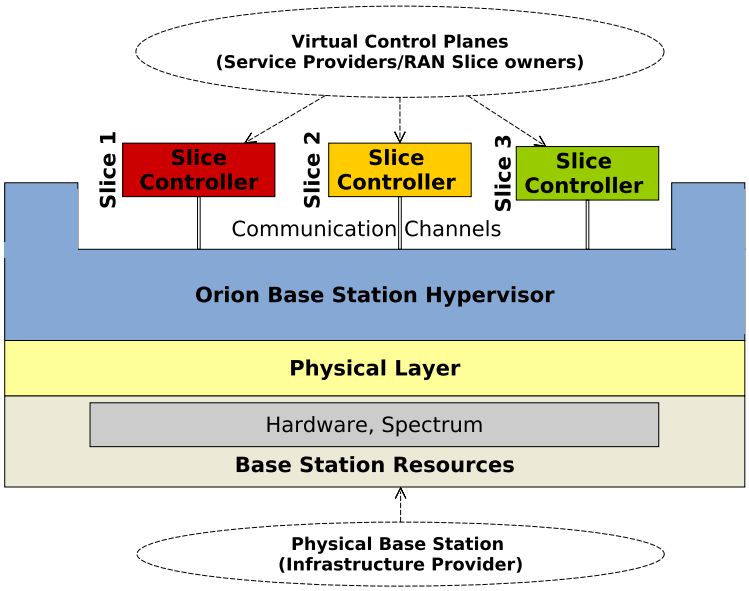



Figure 5.1: High-level architecture of Orion.

full isolation – control logic for functional isolation and resources for performance isolation, as well as facilitating efficient sharing of underlying physical resources. Essentially, the `Hypervisor` binds the individual and isolated slices to the physical infrastructure, providing them with a virtual view of the underlying radio resources via a novel set of abstractions and the data plane state as well as applying their state changes over the physical data plane by mapping virtual to physical resources. The `Hypervisor` is part of the infrastructure provider's software infrastructure to support RAN slicing. The infrastructure provider is also responsible for admission control.

Service providers (e.g., MVNOs and verticals) in Orion realize their RAN slices through the creation of virtual base stations over the `Hypervisor`. Each virtual base station is a composition of a virtual control plane, responsible for managing data plane state that is revealed to it by the `Hypervisor`. The virtual control plane of a slice is effectively a local RAN-level slice controller running as a *separate process*, responsible to tailor the functionality and manage the allocation of resources to UEs associated with the slice as if it was operating using its own dedicated infrastructure. The virtual control plane is also responsible for implementing the control protocols required for the communication and coordination of the virtual base station with the rest of the mobile infrastructure (e.g., S1 and X2 interfaces in LTE). This means that all operations defined for a given mobile network architecture can be supported by slices (including roaming) so long as the appropriate interfaces and messages are implemented as part

of the respective virtual control planes. Note that although we discuss the data plane aspect both in our design and implementation, our primary focus expectedly is on the control plane. Following SDN principles, our design assumes control-data plane separation that is now widely accepted in the mobile networking domain [79, 115, 142, 46].

The communication of the slices' virtual control planes with the `Hypervisor` is message-based and happens through independent physical/virtual communication channels. This approach allows the deployment of slices either over the same physical machine as their `Hypervisor` or over separate physical machines. It also gives the flexibility to slice owners to compose their control planes using different levels of centralization to enable coordination based on their services' needs, independently of other slices.

The design of Orion provides strong isolation guarantees while allowing efficient resource sharing. First, since each of the slice controllers is running as a separate process, isolation among controllers in terms of memory and CPU can be achieved by employing well known OS and process virtualization techniques, like virtual machines (e.g., KVM) or containers (e.g., LXD and Docker). Second, the `Hypervisor` is the sole entity responsible for handling actual radio resources which it distributes among slices after virtualizing them, ensuring isolation from a radio resource perspective. Third, it can internally facilitate efficient resource use via a suitable allocation algorithm that also considers slices SLAs and underlying physical conditions. Additionally, from a UE perspective, the whole slicing operation is transparent, with each slice appearing as a different MVNO as in RAN sharing.

## 5.3 System Design & Implementation

This section details the key components of Orion: `Base Station Hypervisor` and `Virtual Control Plane` of a slice.

### 5.3.1 Base Station Hypervisor

The `Base Station Hypervisor` (Figure 5.2) acts as the intermediary between the data plane of the physical base station and the virtual control planes of slices. The `Hypervisor` communicates with the data plane via an API to access and modify the data plane state (e.g., obtain signal quality measurements and transmission queue sizes of UEs, apply scheduling decisions, etc.). The `Hypervisor` is responsible for allocat-

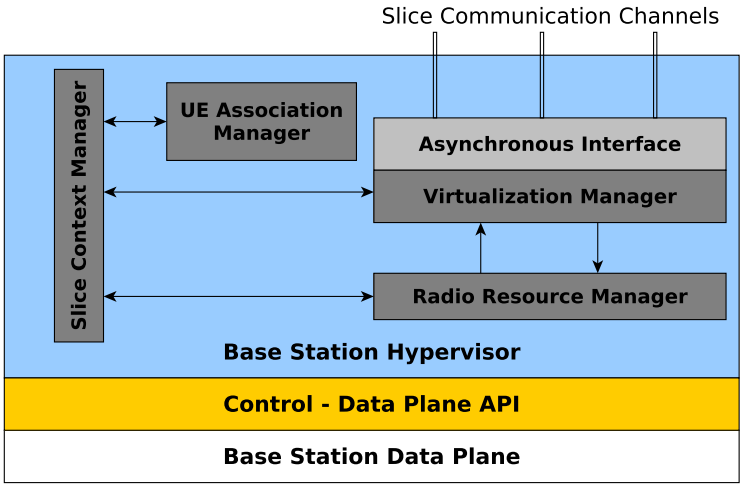



Figure 5.2: Architecture of the Orion Base Station Hypervisor.

ing physical radio resources to slices with respect to their SLAs, transforms them into virtualized resources that are revealed to virtual control planes of slices in an isolated manner. On the other direction, slices use the provided virtual resources for their control plane operations and send their commands (e.g., scheduling decisions) back to the `Hypervisor`, which then translates them to a physical representation and applies them to the data plane. We now detail the internals of the `Hypervisor`.

#### 5.3.1.1 Slice Context Manager

Prior to a slice's creation, the slice owner comes to an agreement on the required service type with the infrastructure provider, subject to admission control. This is formally translated into a *slice service description* that includes two elements:

**An SLA** that identifies the service requirements of the slice owner (e.g., average throughput, resource blocks). Orion does not restrict the SLA parameters and instead provides a flexible framework to implement suitably tailored mechanisms for admission control and fine-grained resource allocation.

**A list of UE identifiers** required for the mapping of UEs to slices. Any unique identifier provided by the UE during the attachment process could be used.

The `Slice Context Manager` (Figure 5.2) is responsible for the life-cycle management of slices at the base station. Each slice is associated with a context (Table 5.1) capturing the slice's capabilities and current state. Apart from the slice's SLA, this

| Slice Context |
|---|
| Slice id |
| UE identifier lookup function |
| Current slice SLA/policy settings |
| UEs currently connected to the slice |
| Communication channel configuration to slice local controller |
| Bookkeeping on usage of physical resources |

Table 5.1: Per slice context maintained by the Hypervisor.

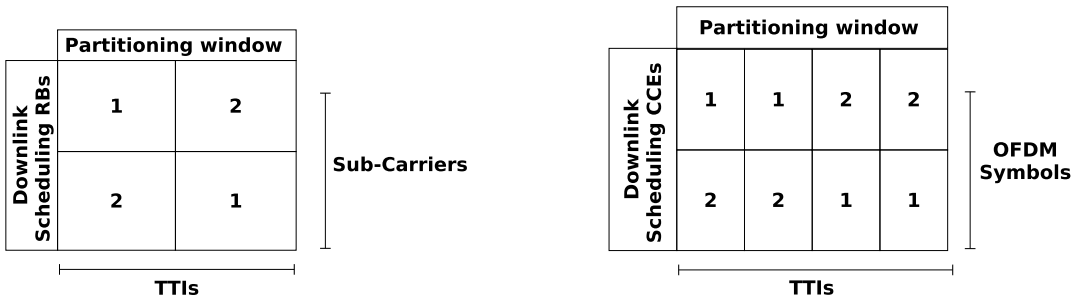
context keeps track of the active UEs on the slice, stores the configuration of the communication channel between the `Hypervisor` and the slice's virtual control plane, and all data structures for bookkeeping operations related to the use of physical resources by the slice in the past and present. The `Slice Context Manager` can obtain the identifiers of the UEs belonging to the slice from the management plane through a lookup function, further explained in Section 5.3.3.

Upon request for creation of a new slice, the `Slice Context Manager` does admission control. By checking the number of active slices, their SLAs and activity (through their context), and using an admission control policy adopted by the infrastructure provider (e.g., along the lines of [76]), either the slice is admitted with a corresponding context created, or rejected due to insufficient resources.

#### 5.3.1.2 Radio Resource Manager

The `Radio Resource Manager` is responsible for the allocation of physical radio resources among co-located slices in a flexible and efficient manner, while ensuring slice isolation. While this is dependent on the nature of each resource, Orion adopts the principle that any physical radio resource meant for individual UEs can also be allocated among slices in an isolated fashion, because, by definition, they can be quantized and assigned to specific UEs. Such resources include radio resources allocated for downlink/uplink user and control traffic as well as for paging.

As an example, consider downlink LTE scheduling, involving two types of radio resources: (i) RBs for transmission of user traffic and (ii) Control Channel Elements (CCEs), essentially a set of OFDM symbols, for transmission of the corresponding scheduling decisions. Due to their nature, these resources can be dynamically allocated to individual slices based on various criteria, like their SLAs, the channel conditions experienced by UEs in the slice etc. On the other hand, physical resources used either for the transmission of cell-related information (e.g., broadcast) or in a




(a) Resource Blocks

(b) Control Channel Elements

Figure 5.3: Resource partitioning between 2 slices for the LTE downlink scheduling example.

contention-based manner (e.g., random access) should not be allocated to slices because concurrent modifications from multiple slice control planes could lead to conflicts. Such resources are exclusively managed by the `Hypervisor` as discussed later in this section.

Orion provides a generic framework for the dynamic allocation of radio resources, which allows the implementation of different radio resource allocation mechanisms to fulfill different types of SLAs. Any allocation algorithm implemented in Orion would have inputs in the form of the physical radio resource grid and the context information of the active slices. The radio resource allocation process occurs periodically once every *allocation window* of duration $t$. At the beginning of each window, the `Radio Resource Manager` obtains the bookkeeping information and the SLAs of all the active slices from the `Slice Context Manager`. Through this information, and with the current cell configuration and allocation mechanism implemented, the `Radio Resource Manager` decides a splitting of the available radio resources among slices for the upcoming window. For each type of partition-able radio resource, it fills a two dimensional array that expresses the slice assignments of resources through a generic representation of the resource in the time and frequency domain. As an example, Figure 5.3 illustrates LTE downlink scheduling for 2 slices. Each column represents a 1ms LTE subframe in TTI units and each row the corresponding resource in the frequency domain (RBs in Figure 5.3a and OFDM symbols in Figure 5.3b). The tabulated numbers correspond to the ids of the slices to which the resources were allocated.

The decision of the `Radio Resource Manager` is forwarded to the `Slice Context Manager` to update the context of slices with allocated resources. Note that the `Slice Context Manager` and `Radio Resource Manager` can be seen to provide the func-

tionality of the 5G network slice broker introduced in the context of the 3GPP network sharing management architecture in [124]. When the slices' control planes make the scheduling decisions based on a virtualized and isolated form of their allocated resources via the `Virtualization Manager`, the `Radio Resource Manager` translates and updates the base station data plane state through the control-data plane API, and keeps track of the physical resources used by slices (via context updates) to inform the allocation in upcoming windows.

#### 5.3.1.3 Virtualization Manager

The `Virtualization Manager` (Figure 5.2) directly interacts with the virtual control plane of slices, maintaining slice isolation in terms of physical network resources. The interaction occurs through dedicated slice communication channels managed by an asynchronous interface. Besides, the `Virtualization Manager` undertakes two main tasks: (i) presenting an abstract view of radio resources to slices by mapping physical to/from virtual resources at runtime; (ii) presenting a virtual/abstract view of the data plane state to slices. The challenge for this component is to realize these tasks in a manner that does not compromise slice isolation.

**Abstracting radio resources** To ensure radio resource isolation, the `Virtualization Manager`, creates a virtualized view of the radio resources tailored to each slice, by obtaining the map of resources assigned by the `Radio Resource Manager` from the slice contexts. This virtualized view omits all resources not dedicated to a slice, including resources for random access, broadcasting, resources used only by the physical layer (e.g., for reference signals) and those allocated to other slices. It also omits the exact placement of the resources in the resource grid along the frequency dimension and instead reveals an abstract representation to the slices' control planes. On one hand, this allows the control planes of slices to directly and independently view and control the radio resources allocated to them, which can be dynamically re-assigned to different slices over time. On the other hand, by withholding the frequency dimension, potential inference and manipulation of resources allocated to a slice by other slices is prevented, keeping in mind that different tenants could in fact be direct competitors. It should be noted that this inference issue is not present in RAN sharing or in a static radio resources allocation setup. In the RAN sharing case, a single entity (the infrastructure provider) performs all the control operations and thus the exact usage of the resources is 'hidden' from the slice owners, while in the other case, the radio

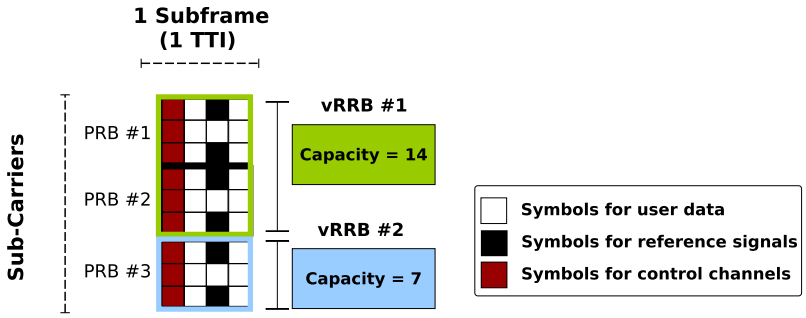


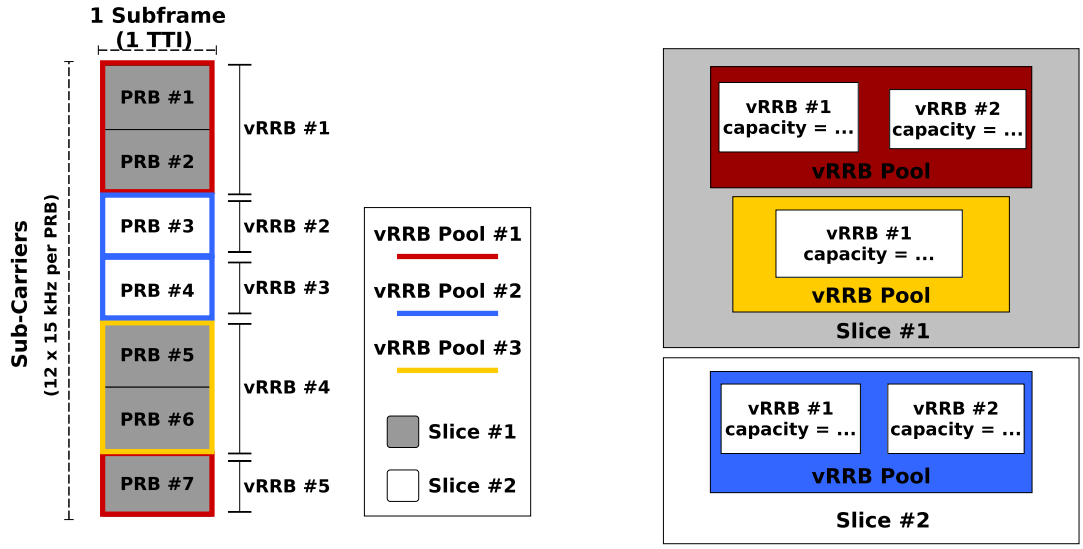

Figure 5.4: Mapping of PRBs to vRRBs in the context of LTE.

resources are statically assigned to slices, providing no visibility to the resources of other co-located slices.

For the case of downlink/uplink UE data transmissions, one of the main challenges is how to reveal the virtualized radio resources to slices so that they can be allocated to the UEs in the most flexible way and keeping in mind that the slices must be completely unaware of the physical resource grid layout, while at the same time respecting the allocation constraints imposed by the physical layer. Such constraints include group-based allocations, where more than one Physical Resource Block (PRB) must be used as the minimum allocation unit for a UE (e.g., as specified in some types of LTE allocations [8]). Other constraints include frequency dependencies that might arise in the radio resource allocation process, where the use of a certain PRB limits the scheduling of a UE to only a subset of all the available resources. Such frequency dependencies are very common in many radio resource allocation schemes, like for example in the type 1 downlink resource allocation of LTE [8] or in scheduling schemes where frequency hopping is employed for frequency diversity gains (e.g., in some types of LTE uplink allocation [8]).

In order to deal with these issues, Orion introduces two abstractions for the virtualization of the radio resources, i.e., the *virtual Radio Resource Block (vRRB)* and the *vRRB pool*.

The vRRB is the fundamental radio resource abstraction employed by Orion. Each vRRB is characterized by a *capacity*, used to indicate the amount of data that it can hold. Capacity is expressed as the number of OFDM symbols contained in the vRRB and can be directly translated into a number of bits by the virtual control planes of slices based on the MCS used. A vRRB can aggregate the OFDM symbols of one or more PRBs, after omitting symbols used for control channels and reference signals.




(a) vRRB and vRRB pool abstractions for LTE resource grid

(b) Abstractions of Figure 5.5a from the slices' point of view

Figure 5.5: Illustration of Orion uplink/downlink radio resource virtualization abstractions in the context of LTE.

This aggregation allows Orion to group together PRBs that must be used as a single unit, meaning that the capacity among different vRRBs can vary, as illustrated in Figure 5.4 (vRRBs of one or two PRBs).

The `Hypervisor` aggregates and reveals the vRRBs to slices in the form of *vRRB pools*, where different sets of pools exist for the uplink and downlink. In each subframe, each slice can be assigned zero, one or more pools for downlink and correspondingly for uplink, each containing at least one vRRB. For example, in Figure 5.5a, slice 1 has two vRRB pools, while slice 2 has only one. A slice can only view and allocate the vRRBs that are available in its pools (Figure 5.5b), with the only constraint being that a UE can only be allocated vRRBs from the same pool. For example, in Figure 5.5, a UE belonging to slice 1 can only be allocated resources from the red or yellow pool, but not both. The aggregation of vRRBs to pools captures the aforementioned frequency dependencies, since RBs that are mutually exclusive for a UE are assigned to different pools by the `Hypervisor`.

It should be noted that the abstractions of Orion are suitable both for current LTE as well as for future 5G NR networks, since both are based on OFDM and have the same fundamental frame structure [118]. The main difference is that NR is expected to support a more flexible resource grid, in which the number of TTIs and the sub-carrier

spacing of the RBs in a subframe can vary (Figure 5.6) to support the diverse requirements of services and to provide support for mmWave communications [114]. In this context, the physical grid layout of the radio resources is one additional dimension that the `Hypervisor` should consider for mapping the physical resources to vRRBs. For example, for the FDD downlink grid presented in Figure 5.6, a low data rate service without latency constraints should be allocated vRRBs 2 to 5 by the `Hypervisor` to maximize efficiency, while a low-latency service should be allocated vRRBs 6 and 7.

In contrast to the resources used for user traffic, control resources are provided in a simpler form, where the `Hypervisor` indicates to the virtual control planes of slices the role and amount of the assigned control resources through a message-based API. On the downlink, a virtual control plane can find out through this API how many CCEs are available for transmitting the scheduling decisions of its UEs, the resources available for sending uplink power control instructions as well as the resources available for uplink resource grants. On the uplink, this involves the TTIs in which a UE of the slice could report its signal measurements, so that it can be configured correspondingly by the higher layers of its virtual control plane (e.g., RRC).

The `Virtualization Manager`, besides presenting virtualized resources to slices, also performs the reverse mapping from slices' decisions over virtual resources to their physical counterparts. This involves consulting the context of the slices and performing all transformations required so that a command issued by the virtual control plane of a slice can be realized by the physical layer. Referring again to the example of LTE downlink scheduling, this mapping would involve a modification of the virtual control plane's scheduling decision to convert the vRRB allocation into a physical one, and the merging of the scheduling decisions from different slices into a single scheduling decision for the physical layer. Similarly, it would require the conversion of CCEs from the virtual to the physical form.

**Virtualizing the data plane state** To apply its control logic, a slice must be aware of its data plane state besides the resources allocated to it. Another task for the `Virtualization Manager` is to reveal the relevant data plane state to slices in an isolated manner, guaranteeing that any type of information related to specific UEs or their flows will be reported only to the corresponding slice. This virtualization in the data plane state ensures the isolation of data plane operations among slices, since the control plane of each slice is unaware of the data plane aspects that are relevant to the UEs of other co-located slices. Examples of relevant UE information include signal

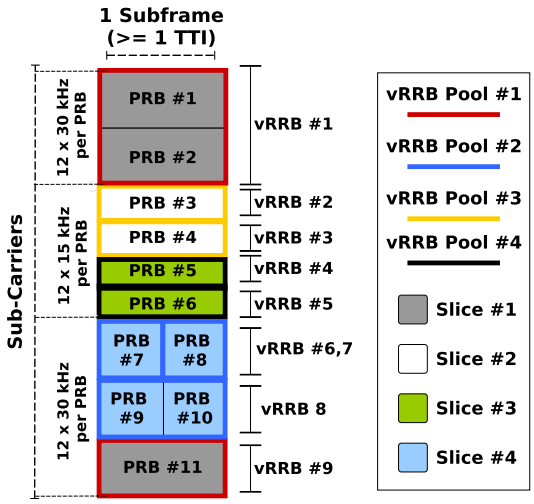



Figure 5.6: Orion uplink/downlink radio resource virtualization abstractions in the context of 5G NR – Flexible TTI and sub-carrier spacing numerology.

quality measurements, transmission queue sizes and number of flows (bearers in LTE) as well as event notifications like the (de)activation of a UE, the establishment of new flows, scheduling requests, etc. Given that the frequency domain is abstracted from the resource grid of slices, any UE-specific information related to the frequency domain is reported to the slice in an abstract form. As an example, relating to signal quality measurements and to the more fine-grained sub-band CQI reports that LTE can provide, the `Virtualization Manager` furnishes the measurements in the context of the vRRBs rather than the actual frequencies, through suitable mapping.

Apart from UE specific information, the `Virtualization Manager` also informs slices about cell-related configuration information like resource block sizes, supported transmission modes, nominal transmission power of base station, etc. Since all slices need to be aware of this information and in order to ensure a conflict-free operation of the physical base station, all such cell-related information is provided to slices in a read-only form, while the infrastructure provider retains the exclusive privilege for making changes, when needed.

#### 5.3.1.4 UE Association Manager

The `UE Association Manager` (Figure 5.2) associates UEs with slices in two steps: (i) the discovery of slices by UEs via the physical base station; (ii) the mapping of UEs to slices. To achieve this, the `UE Association Manager` interacts with the broadcast

and random access processes of the base station, which are internal to the Hypervisor and not revealed to slices.

To aid in discovery, the UE Association Manager obtains the active slices from the Slice Context Manager. This information is then broadcasted by the base station. In the context of LTE, this can be done as in RAN sharing, where the base station broadcasts the list of PLMN ids, indicating all the MVNOs that are present. When a UE discovers a cell from its slice, it initiates the random access process. If successful, the attachment process begins and the UE is expected to send a unique identification (e.g., IMSI in LTE). Using this unique identifier, the UE Association Manager maps the UE to the appropriate slice by consulting the UE lookup function (Table 5.1) available for each slice through the Slice Context Manager. Once the correct slice is identified, the UE Association Manager updates the list of active UEs in the context of that slice. The corresponding virtual control plane is notified about the event through the Virtualization Manager and from that point on takes up control of the UE (authentication, establishment of flows, scheduling, etc.). The attachment process is fully transparent to UEs in the sense that no changes are required to the protocols and signaling messages used for the interaction between the base station and the UEs.

### 5.3.2 Virtual Control Plane

As already alluded to in Section 5.2, Orion isolates the memory and processing resources of slices by separating their virtual control planes from each other and the Hypervisor, thereby achieves functional isolation. Virtual control planes of slices in Orion are separate processes that exchange messages with the Hypervisor through dedicated communication channels. This separation allows slice deployment in a fully isolated manner using OS virtualization technologies, like containers and virtual machines, to enforce limits in terms of the allocated amount of memory and CPU.

A virtual control plane of a slice can be associated with multiple physical base stations (through their Hypervisors). Thus, the virtual control planes of individual slices can be composed independently and flexibly, like in the example of Figure 5.7 for the case of two slices. As it can be seen, control plane centralization can be introduced into different regions of the RAN for each of the slices, based on the corresponding service's needs. This approach gives the flexibility to slice owners to enable RAN coordination (for load balancing, improved mobility management, etc.) when and where required. At the same time, it also enables an efficient utilization of the available com-

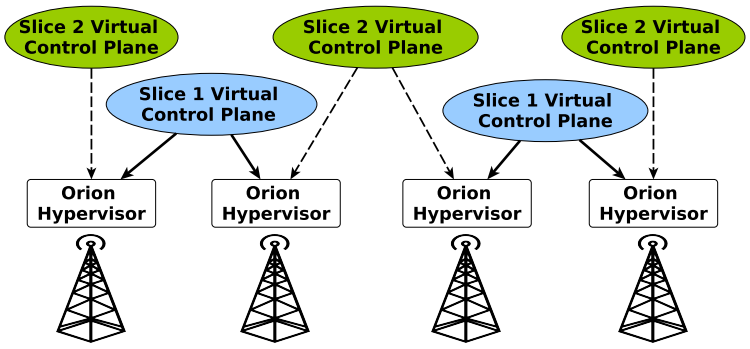



Figure 5.7: Flexible control plane composition enabled by Orion.

puting resources for the placement of the slices' virtual control planes as demonstrated later through experiments in Section 5.4.2.

The communication between the virtual control plane of a slice and the `Hypervisor` is asynchronous (Figure 5.8). To perform its operations, the control plane obtains the state of the virtual data plane and resources from the `Hypervisor` and stores it locally in a virtual RAN Information Base (vRIB). In its simplest form, the vRIB contains data structures with a raw representation of the virtual data plane state.

The vRIB can be both read and written by the slice's control plane, so changes due to control operations are first reflected in the vRIB. To maintain its consistency, the vRIB state is periodically synchronized with the `Hypervisor` via its `Virtualization Manager` and an asynchronous interface. Since the virtual control plane of slices and the `Hypervisor` can be deployed either on the same or on separate physical machines, depending on the setup and the available resources, the synchronization frequency of the vRIB can be tuned to the characteristics of the communication channel.

### 5.3.3 End-to-End Network Slicing

We now discuss how Orion can be integrated in an end-to-end network slicing setting. The virtualization of various components making up the slice (including network functions) is key to enabling flexible component placement over the network infrastructure through a MANO entity, based on the needs of the service corresponding to the end-to-end slice. This is certainly not a hurdle for Orion as its components can be turned into VNFs. For a complete solution, it is also very important to investigate the interactions of the Orion components with the management plane and with the other components making up the slice. These interactions and the entities involved are depicted in Figure 5.9, where each number-annotated arrow represents an interaction required for the

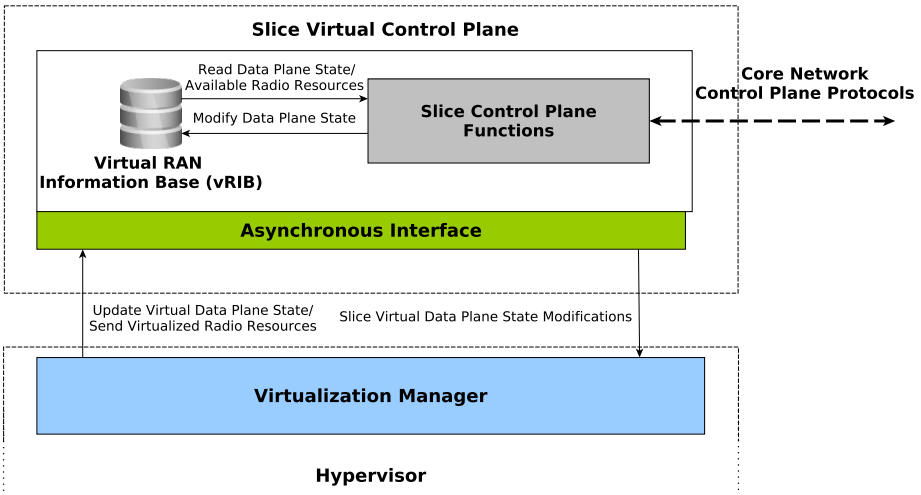



Figure 5.8: Virtual control plane of a slice in Orion.

right instantiation, operation and management of a slice throughout its life-cycle.

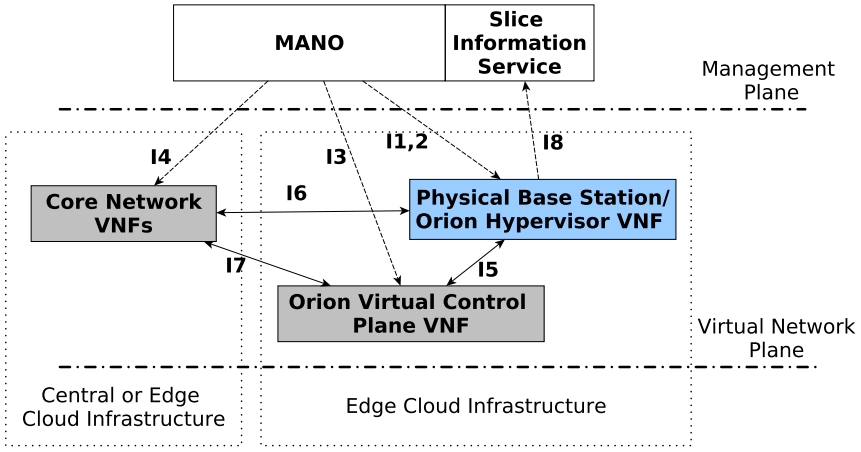



Figure 5.9: Interactions among components involved in an end-to-end network slicing setting.

Starting from the RAN and before the deployment of any slice, the MANO must deploy the VNFs of the physical base stations and the Orion `Hypervisor` for the RAT of choice (`I1`). This VNF is expected to outlive the slices' VNFs. Management of slices should occur through an appropriate interface between the MANO and the `Hypervisor` (`I2`). The orchestration of a slice by the MANO involves the creation, migration and destruction of the Orion virtual control plane VNF (`I3`), as well as of the core network VNFs (`I4`) (e.g., LTE EPC components). While a slice is active, its core network VNFs interact with the `Hypervisor` for the user plane (`I6`) and with the virtual control plane for the control plane operations (`I7`), while the `Hypervisor` VNF

also communicates with the slice's virtual control plane (I5). A *Slice Information Service* is required on the management plane, so that the Slice Context Manager in the Hypervisor can locate the information required for the mapping of UEs to slices (I8). This resembles the role of the HSS in LTE, but storing slice-related registration information instead.

Figure 5.9 also illustrates that the Orion VNFs are deployed over an edge cloud near the RF front-end, while the core network VNFs can be deployed either at the edge or in a central cloud, depending on the slice's requirements. For example, control plane VNFs (e.g., MME) could be centralized to simplify control/management, while data plane VNFs (e.g., S-GW) could be placed closer to the user [40]. Similarly, different slices could employ different core network VNF setups, optimized for their particular use case (e.g., as in [132] for M2M traffic). The flexible placement of VNFs, coupled with the capabilities offered by Orion for the customization of the RAN, allow the creation of flexible slices adapted to the service needs.

### 5.3.4 Implementation

We developed a prototype implementation of Orion, following the design described so far in this section and considering LTE as the RAT. The Orion Hypervisor was implemented from scratch in C. For the interaction of the Hypervisor with the base station data plane, we leveraged the control–data plane API of FlexRAN that was presented in Chapter 4.

Our implementation provides full support for the virtualization of downlink radio resources, with the Hypervisor creating a single vRRB pool per slice in each subframe (equal to a TTI of 1ms in LTE), with each of its vRRBs corresponding to a group of PRBs (2, 3 or 4 based on the bandwidth of the cell). This is because OAI currently supports resource allocation type 0 on the downlink, in which, based on the LTE specification [8], resource blocks can be assigned to UEs only in groups of 2, 3 or 4 blocks depending on the available spectrum.

The identifier used for mapping UEs to slices is the IMSI stored in the SIM card of UEs. This is sent to the MME of the network by UEs during their attachment using the Non-Access Stratum (NAS) protocols of LTE and is normally not visible to the eNB. Therefore, the RRC layer of OAI was modified to capture the relevant signaling messages, obtain these ids and pass them to the UE Association Manager of the Hypervisor. OAI was further modified to allow core networks (EPCs) of different

slices to be connected over the same eNB. This required enabling the association of the MMEs, HSSs and S/P-GWs of different slices with the same eNB as well as the redirection of newly attached UEs to the correct core network by the `UE Association Manager` both for signaling and user traffic. Once a UE completes the random access process, its signaling messages are directed to the correct MME so that it can be authenticated by the corresponding slice's HSS and a bearer can be established with the slice's S/P-GW. From that point on, all the UE traffic goes through the core network of the hosting slice.

The current implementation allows expressing the SLA parameters stored in the context of active slices (Table 5.1) in three ways: (i) a fixed allocation of resource blocks; (ii) an average of the aggregate amount of resource blocks allocated to the slice over a window of 100ms; and (iii) an average target throughput for the slice over a window of 100ms. For the second and the third case, the current implementation uses the NVS radio resource allocation algorithm [76]. It should also be noted that admission control is not the main focus of this work so for the purpose of this study a first come, first served policy is employed for the admission of slices.

To realize the virtual control plane of slices, we implemented a modified version of the FlexRAN controller that operates over the abstract virtual resources presented by the `Hypervisor` and also with an enhanced RIB structure that supports write primitives. Given that the virtual control planes of slices can be deployed either locally or over a networked setting, we implemented two types of channels for message exchanges with the `Hypervisor`: one TCP-based channel optimized for a minimum message exchange delay in a network setting and one optimized for inter-process communication using ZeroMQ [61]. The appropriate type is specified in the slice's service description during the creation of the slice and the proper communication channel is deployed accordingly. Moreover, depending on the type of deployment, the synchronization of the `Hypervisor` with the vRIB must also be optimized. Towards this end, in the current implementation with inter-process communication the vRIB is fully synchronized with the Hypervisor every 1ms, while in a networked setting, only the time critical parts of the vRIB (virtual resource allocations, time synchronization in subframes) are synchronized at this rate, with the rest synchronized infrequently every 4ms.

Finally, we created VNFs for the `Hypervisor` and the virtual control plane of Orion using Juju charms, enabling their deployment over an OpenStack-based environment and allowing usage of Orion with any OpenStack-compatible MANO framework (e.g., OSM).

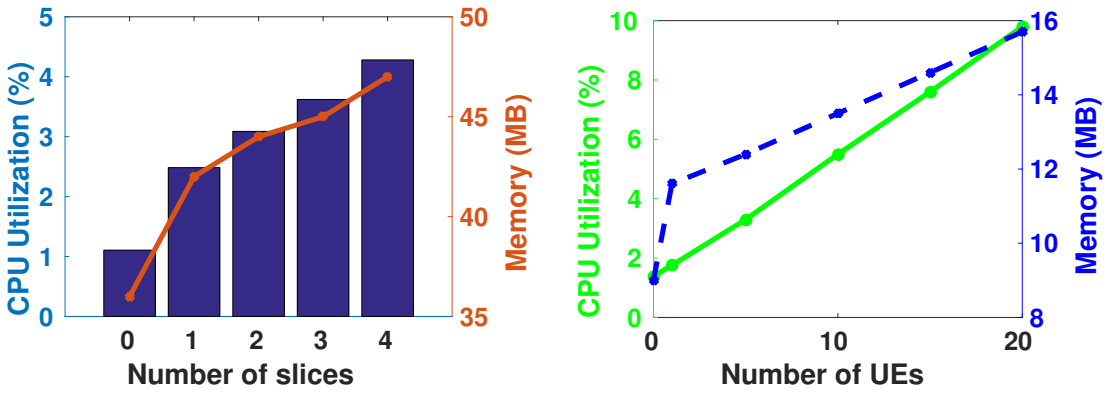
## 5.4 Evaluation

In this section, we quantitatively study Orion's resource consumption in different scenarios and benchmark it against other proposed RAN slicing solutions. We also examine the impact of the communication channel between the `Hypervisor` and the slice control plane. For the experiments, we used 2 Intel Xeon machines (E3-1245 @ 3.4GHz, 16GB RAM each) and 1 Intel i7 machine (5557U @ 3.10GHz, 8GB of RAM) for deploying physical base stations, the Orion Hypervisor and the virtual control planes of slices, depending on the specific experiment. All machines have Ubuntu 16.04 with a low-latency kernel and had support for Docker v1.13.1 [96], allowing the Orion components (Hypervisor, slice control planes) to be deployed in isolated containers. The core part of the network was deployed on an additional Intel-based machine (i7-4770R @ 3.2GHz, 8GB RAM), running the openair-cn open source EPC implementation [112]. For the RF front-end of physical base station, we used Ettus USRP B210 SDR boards and for UEs, up to 4 physical units (LG Nexus 5 and Samsung Galaxy Note 4 and 2 Huawei E3372 LTE dongles).

### 5.4.1 Scalability

Here we quantify how Orion scales in terms of the processing and memory requirements. To assess the overhead incurred for the virtualization operations performed by the Orion `Hypervisor`, we used a setup where each slice had 20 assigned (emulated) UEs and the available spectrum was saturated with TCP traffic. The results in Figure 5.10a show that, apart from the initial overhead associated with the first slice, the addition of extra slices incurs a small and almost constant incremental overhead, both in terms of CPU and memory. This increase is mainly due to the `Hypervisor`'s `Virtualization Manager` operations (construction of the virtual resource grids of slices, synchronization with slices' virtual control planes). The initial bigger increase observed after the creation of the first slice is due to the activation of the asynchronous interface as well as the periodic execution of the `Radio Resource Manager` for the allocation of resources to slices, which is inactive when there are no slices. The actual number of slices that can be supported by a physical base station depends both on the `Hypervisor` overhead presented here and on the physical layer requirements, which can greatly vary depending on where the most demanding physical layer operations occur (e.g., at the baseband processing unit or at a remote radio head).

Next we turn our attention to the virtual control plane of a slice and quantify its




(a) Hypervisor overhead (4-core CPU and 20 UEs/slice)

(b) Virtual control plane overhead (2-core CPU)

Figure 5.10: Orion CPU and memory consumption scaling.

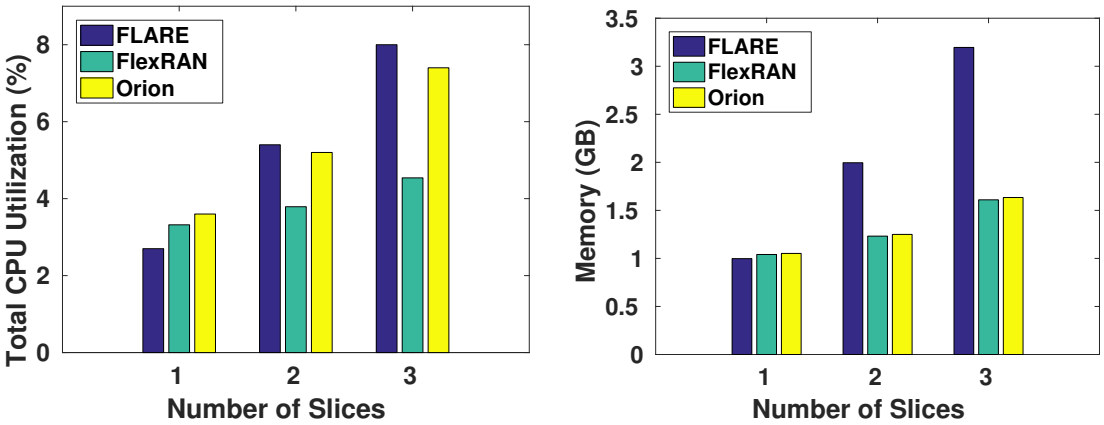
overhead with a varying number of UEs (0 to 20) which increases linearly with the number attached UEs (Figure 5.10b). The increases in the memory consumption are mainly due to the additional information stored in the vRIB as more UEs get attached and the memory required for the message exchanges with the `Hypervisor`. The initial sharp rise in the memory consumption is due to the creation of the UE-related part of the vRIB, once the first UE gets attached. The rise in CPU utilization is a result of the communication with the `Hypervisor` for the synchronization of the vRIB and from the scheduling of the attached UEs. However, the overhead is still fairly modest, meaning that the virtual control planes of multiple slices could be deployed side-by-side over the same physical machine (e.g., over an edge cloud data-center) without scaling issues. For example, in the commodity machine used for these experiments, at least eight slice control planes could be deployed without any issues.

### 5.4.2 Comparison with the State-of-the-Art

Here we compare the overall memory and processing requirements of Orion versus the state-of-the-art virtualization and RAN sharing solutions from the literature. One representative approach is *FLARE*, the RAN slicing solution proposed in [103, 102], which ensures isolation by deploying different instances of OAI over Docker containers and each slice is *statically* assigned a dedicated chunk of spectrum. The other alternative approach is represented by FlexRAN [46, 105, 43], which as was shown in Section 4.5.3 can provide RAN slicing capabilities following a RAN sharing approach but without functional isolation. It should be noted that, like Orion, both FLARE and

| System / # Slices | FLARE | FlexRAN | Orion |
|---|---|---|---|
| 1 | 5MHz | 5MHz | 5MHz |
| 2 | 5MHz/slice | 10MHz | 10MHz |
| 3 | 5MHz for slices 1,2<br>10MHz for slice 3 | 20MHz | 20MHz |

Table 5.2: Allocation of channel bandwidth among slices. Both Orion and FlexRAN use shared spectrum for all slices.




(a) CPU Utilization (10 cores)

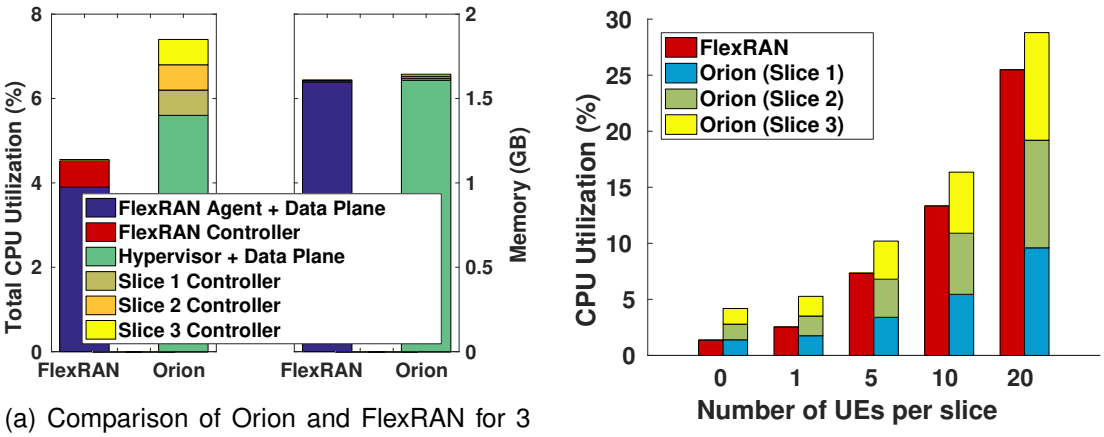
(b) Memory footprint

Figure 5.11: Comparison of Orion with FLARE and FlexRAN with varying number of slices.

FlexRAN are built using OAI as their basis. Therefore, any performance differences among them come mainly from the design choices of each system regarding how to perform RAN slicing; a fact that makes this comparison fair.

We design an experiment to compare Orion with these two alternatives for a varying number of slices, where each slice has TCP traffic (mimicking an aggregate demand from a set of UEs) to a single physical UE. Each slice was allocated spectrum according to the configuration of Table 5.2 for fairness. Three physical machines were used for this experiment as compute nodes for the various components required for each architecture. To capture the overall system performance, we measured the aggregate resources required for all the RAN components of each architecture across all physical machines. For the CPU utilization, this meant measuring the jiffies allocated to the RAN-related processes of each architecture over a constant number of 10 CPU cores (the total number of cores of all physical machines) and dividing this value with the total jiffies of all the CPUs for all system processes during the course of the experiment.

We see from Figure 5.11a that Orion has relatively higher CPU requirements com-




(a) Comparison of Orion and FlexRAN for 3 slices (1 UE/slice)

(b) Control plane CPU Utilization (2 cores)

Figure 5.12: Breakdown of resource requirements for Orion vs FlexRAN .

pared to the FlexRAN case, which is expected due to the `Hypervisor`'s operation and the unavoidable cost to pay for the additional control plane processes that must be deployed as new slices are added. On the other hand, Orion starts with higher CPU requirements than FLARE, but soon becomes more lightweight, due to the fact that each new FLARE slice requires the deployment of a complete protocol stack, including the very demanding (CPU-wise) physical layer. Since in Orion all slices share the same physical layer, its overhead comes mainly from the `Hypervisor` and the slices' virtual control planes, which, as shown earlier, are fairly lightweight. The same observations and for the same reasons can be made for the memory consumption (Figure 5.11b), with Orion performing significantly better compared to FLARE and marginally worse compared to FlexRAN .

Since both Orion and FlexRAN are composed of a number of different network functions (`Hypervisor`/physical layer and virtual control planes in Orion; Agent/data plane and controller in FlexRAN ), it is insightful to break down the results of Figure5.11 in order to understand their differences. This is illustrated in Figure5.12a for three slices. As it can be observed, the Orion virtual control planes and the FlexRAN controller consume a negligible amount of memory. On the other hand, in terms of CPU utilization, Orion's biggest impact comes from the physical layer and the Hypervisor, but, in contrast to memory, a significant contribution is also made from the virtual control planes of slices. In the FlexRAN case, the base station network function (FlexRAN Agent and data plane) has the biggest CPU utilization, also with a significant contribution from the controller.

One important thing to notice is that while some components of both systems,

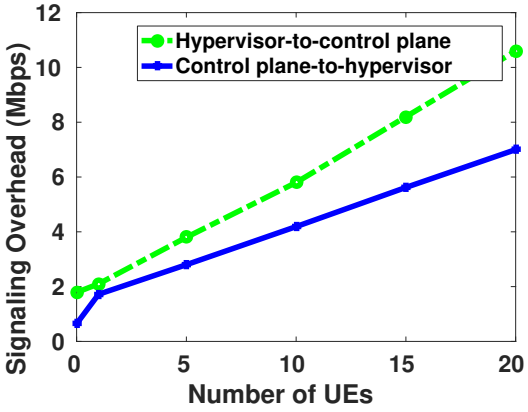
namely the Orion `Hypervisor` and the FlexRAN Agent, are placed near the base station and are effectively monolithic components regardless of the number of deployed slices, this analogy is not carried over to the other network functions. While in the case of FlexRAN the controller is also a monolithic component, Orion provides individual control planes to slices. Apart from the functional isolation that this ensures, it also enables a better utilization of the underlying resources through the flexible placement of the control plane functions over the available compute nodes. We illustrate this point through an experiment in which three slices are deployed using Orion and FlexRAN and the overall control plane CPU utilization is measured for a varying number of UEs per slice. Results shown in Figure 5.12b indicate that Orion always performs slightly worse than FlexRAN , however this overhead is compensated by the fact that the control plane can be broken down into three functions that can be flexibly placed over different CPU cores or even completely different compute nodes; something that FlexRAN is unable to do.

### 5.4.3 Impact of Communication Channel

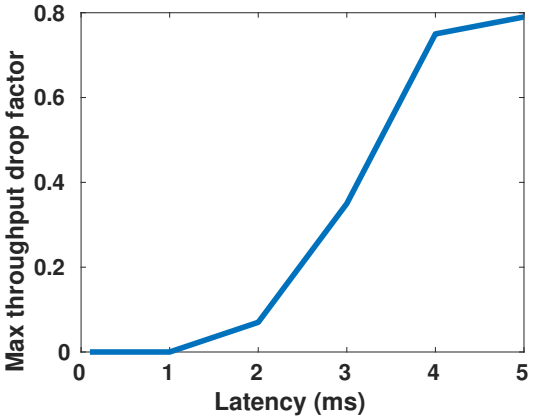
We now assess Orion's requirements for the communication between a slice control plane and the `Hypervisor` (in terms of bandwidth and latency) for their deployment in a networked setting. Understanding these requirements is very important from an end-to-end perspective for deploying the components of Orion as VNFs, since the orchestration entity needs to be aware of the physical constraints regarding their placement as part of the overall network service description.

To measure the bandwidth requirements of the control channel, we varied the number of UEs for a slice over the physical base station with 5MHz spectrum. In this experiment, UEs continuously had traffic to keep the slice at full load. The highest overhead was incurred for messages sent by the `Hypervisor` towards the control plane of the slice (Figure 5.13a), for the synchronization of the vRIBs and for the provision of the slice with virtual radio resources. The overall bandwidth requirements for this setup did not exceed 20Mbps, implying that such a scenario is feasible in practice.

To study latency constraints, we used a similar setup but with a single COTS UE attached to the slice. We used *netem* [48] to set the latency between the `Hypervisor` and the slice's virtual control plane and measured the drop in the maximum achievable throughput for the UE compared to a co-located deployment. Figure 5.13b shows that networked deployment of Orion is efficient only when the latency remains below 2ms.




(a) Signaling bandwidth requirements




(b) Effect of signaling latency on maximum throughput

Figure 5.13: Network requirements for the interaction of the `Hypervisor` with the virtual control plane when not co-located.

This constraint could be relaxed to an extent with some optimizations (e.g., HARQ reporting optimizations) but a co-located deployment is preferable if possible. The slice controller could also be made distributed and hierarchical, so that time-critical scheduling operations can be handled at or near the physical base station, while other operations can be placed flexibly at other locations.

## 5.5 Case Studies

### 5.5.1 Isolation Capabilities

We demonstrate the ability of Orion to provide radio resource and functional isolation, which is not possible with RAN sharing oriented slicing approaches [46, 43]. In this experiment, 2 slices with proportional fair downlink schedulers were created and statically allocated 5MHz of spectrum in equal shares by the `Hypervisor`. Initially, we connected 2 physical UEs (one per slice) and performed a throughput measurement to both using iperf, ensuring ideal channel conditions. As shown in instance $t1$ of Figure 5.14, the `Hypervisor` guarantees the radio resource isolation so that each slice can obtain only half of the available radio resources, leading to an equal throughput for the UEs. Further aspects of this isolation of radio resources are demonstrated in instances $t2$ and $t3$ of Figure 5.14 where more UEs are added to the slices and where each UE is constrained by the `Hypervisor` to the radio resources allocated within its slice. For

example, in instance $t2$ the performance of UE3 remains unaffected by the addition of UE2 since UE2 belongs to a different slice.

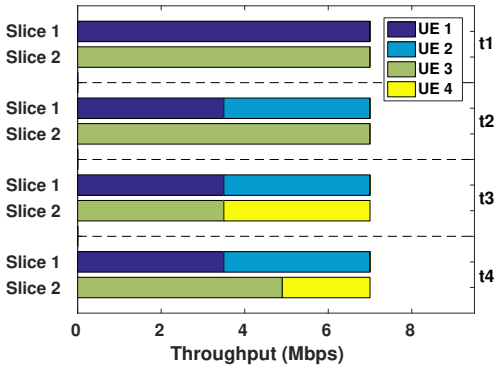


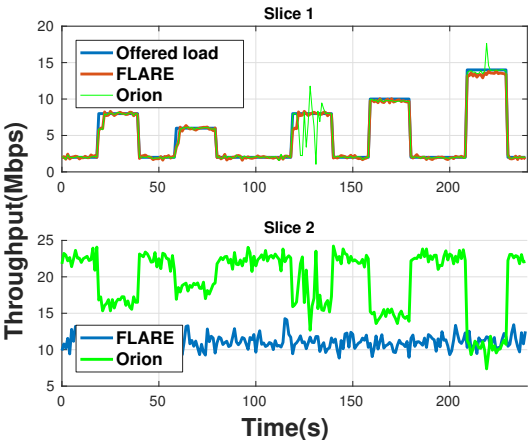

Figure 5.14: Isolation of RAN slices in terms of radio resources and control functions (scheduling).

Finally, the inter-slice functional isolation is illustrated in instance $t4$ of Figure 5.14. The scheduler in the virtual control plane of slice 2 was replaced with a class-based scheduler allocating 70% of the resources to UEs in class 1 and the rest to class 2. In the experiment, UE3 and UE4 were assigned to class 1 and class 2, respectively. As shown in Figure 5.14, the change in the control plane of slice 2 only affects the throughput of the UEs belonging to that slice, leaving the control plane and the UEs of slice 1 completely oblivious.

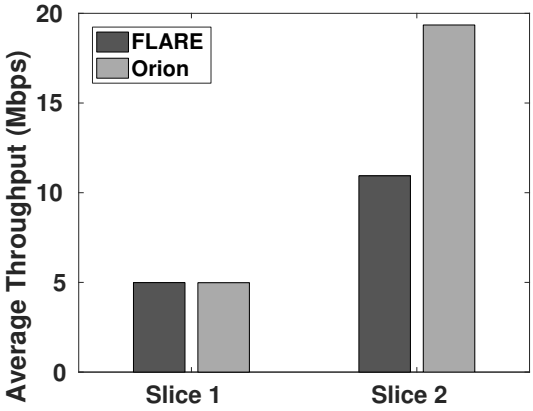
### 5.5.2 Flexible Radio Resource Allocation

We now demonstrate the flexibility offered by Orion for dynamic radio resource allocation and how it can be used for efficiently utilizing the spectrum among co-existing slices. We created 2 slices, each with an average aggregate throughput requirement of 14Mbps. As a baseline, we considered FLARE, with each slice statically allocated 5MHz of spectrum. For the same scenario, we employed Orion with a pool of 10MHz for both slices and implemented the radio resource allocation using the algorithm of [76]. In the beginning of each allocation window, the resources were allocated to slices based on their effective traffic rate and on their average throughput requirement.

We connected 1 UE on slice 1 and 3 UEs on slice 2 and we used the D-ITG traffic generator [22] to generate TCP flows in slice 1. Specifically, we created a TCP flow with a constant rate of 2Mbps. On top of it, we sporadically created some short-lived flows (20s each), capable of attaining various rates (4-12Mbps). In slice 2, all the




(a) Instantaneous slice throughput in static (FLARE) vs dynamic resource allocation (Orion).




(b) Average throughput achieved through static and dynamic spectrum use.

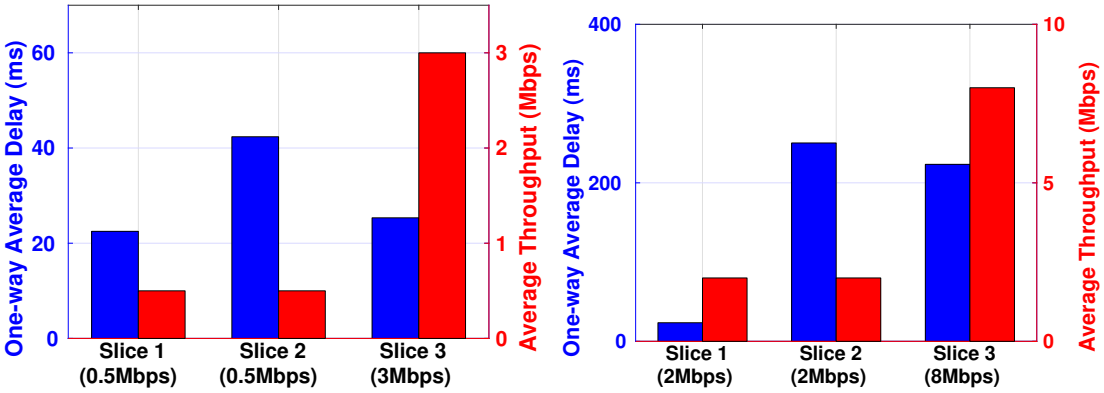
Figure 5.15: Benefit of Orion's flexible radio resource allocation.

connected UEs accessed a DASH-based video streaming service to stream a video offering a wide range of bitrates. Using this setup, we measured the instantaneous (Figure 5.15a) and the average (Figure 5.15b) aggregate throughput achieved by each slice in FLARE and Orion for a period of 240s. As observed, for slice 1, both FLARE and Orion give similar results, fully covering the offered load and respecting the SLAs of slice 1 for up to 14Mbps of *average* aggregate throughput.

For slice 2, however, Orion is seen to perform much better than FLARE, dynamically reallocating the idle resources from slice 1 and allowing the UEs in slice 2 to stream videos with a higher bitrate. FLARE, unlike Orion, is spectrum-limited and cannot go beyond 14Mbps at any point in time, irrespectively of what happens in slice 1. The flexibility of Orion comes without compromising slice SLAs. When slice 1 requires all of its expected resources (e.g., 210-230s in Figure 5.15a), the `Radio Resource Manager` allocates fewer radio resources to slice 2, so the aggregate throughput of this slice drops.

### 5.5.3 Deployment in an End-to-End Setting

Here we highlight the capability of Orion to be deployed in an end-to-end network slicing setting and the effect of the slice configuration to overall performance in terms of throughput and delay. For this experiment we created 3 slices, each composed of VNFs for an EPC (HSS, MME, S/P-GW) and a virtual base station deployed over




(a) Low offered load (b) High offered load

Figure 5.16: Performance impact of Core-eNB latency and radio resource allocation policy/guarantees for low and high offered loads (load values in parentheses on x-axis).

Orion. For slices 1 and 3, the S/P-GW VNFs of the EPC were placed on hosts 1 hop away from the eNB, to reduce the delay of the EPC–eNB communication as much as possible (less than 0.3ms). For slice 2, we deployed the S/P-GW VNF on a host connected to the eNB through a 20ms delay link (emulated with netem [48]) to reflect the placement of the mobile core functionality far from the eNB (e.g., in a central cloud). The eNB in this experiment was allocated 5MHz of spectrum, supporting a total rate of 14Mbps in ideal channel conditions. Finally, the Orion `Hypervisor` was configured so that slice 1 was guaranteed a static allocation of 4 RBs per subframe (maximum achievable throughput of 2Mbps), while slices 2 and 3 were guaranteed an average throughput of 2 and 8Mbps respectively, performing the allocation using the algorithm in [76].

In this setup, we used the D-ITG traffic generator [22] to generate downlink UDP traffic with exponentially distributed packet inter-arrival times and measured slice performance in terms of average throughput and delay. Initially, the load offered to slices was kept low, compared to the slices' SLA guarantees, and all slices manage to support the traffic demand (Figure 5.16a). Moreover, we can observe that the average packet delay for slice 2 was higher than that of slices 1 and 3 by about 20ms, something expected considering the (emulated) 'far from eNB' placement of the core functions for slice 2.

Next, we re-run the experiment, increasing the load offered to slices to reach their limit (Figure 5.16b). As we can observe, all slices still achieve an average throughput close to their offered load, which was expected since their load remains within the

guaranteed limits. However, while the average delay of slice 1 stays at the same level, the average delay of slices 2 and 3 is over 200ms. The reason is that the static allocation used for slice 1, although inflexible, ensures stricter delay guarantees, as arriving packets could always be served without queuing for long. On the other hand, the flexible dynamic allocation of radio resources to the other slices guarantees the average throughput, but this was done over 100 ms windows (as described in Section 5.3.4), rather than per subframe. In such almost saturating traffic conditions the packets are more likely to experience a longer waiting time in the slice queues, increasing the average delay.

## 5.6 Multi-Service Slices Extension

The design of Orion enables RAN slicing in cases where the UE has a 1:1 relationship with the slice (e.g., MVNOs and verticals). However, deployment of OTT services as distinct slices breaks this assumption and control decisions among slices cannot be guaranteed to be conflict-free. To overcome this, we propose an extension to Orion in the form of *service containers*, which targets slices offering multi-service capabilities (e.g., an MVNO with OTT services).

The idea behind service containers is that an OTT service provider agrees with an MVNO to assume control of the flows corresponding to its service through its own isolated functions deployed over the virtual control plane of the slice. This concept is similar in spirit to the 3GPP Service Capability Exposure Function [7] and with research works like [123], where an API is defined for the interaction of applications with the mobile network to modify the service level or to obtain information about the network state. The deployment of service containers over the base stations of MVNOs can enhance this model with *real-time* control and monitoring capabilities, enabling the further optimization of OTT applications.

The design of this extension (Figure 5.17) inherits the design principles discussed in Section 5.3. Service containers are deployed as processes over the virtual control plane of a slice, each with its own isolated memory and processing resources. The virtual control plane of the slice is extended with a Northbound API, enabling the exchange of information between the MVNO and the service provider. Using the API, a service can obtain information about the flows of UEs assigned to it (available UE flows, transmission queue sizes, signal quality etc.), as well as about the (abstracted) radio resources available for it. On the other direction, the service can issue scheduling

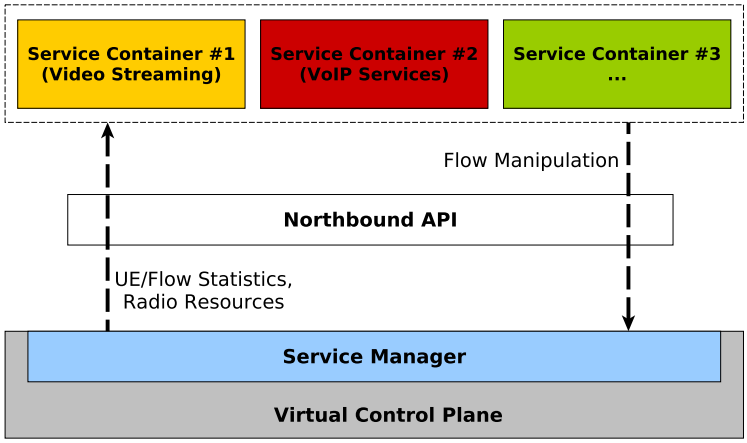
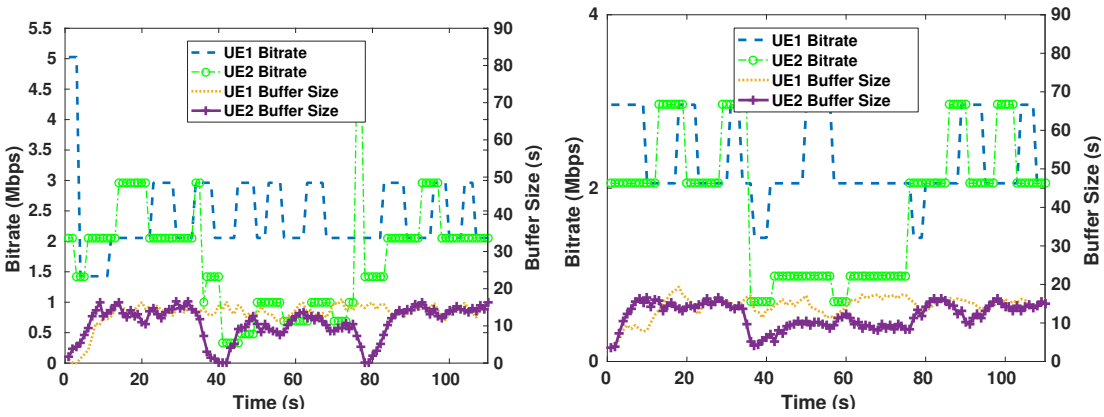



Figure 5.17: Extension of Orion to support multi-service slices.

decisions, indicating how its allocated resources should be distributed among its flows. A `Service Manager` sitting between the virtual control plane of the MVNO and the Northbound API is responsible to map flows to services, distribute the abstract radio resources to the containers and perform access control, ensuring that service containers can only access information and radio resources related to their serving flows.

In order to demonstrate the benefits of service containers, we consider the example of a DASH-based adaptive bitrate video streaming service provider for which a service container was developed and deployed over Orion for monitoring and scheduling the video flows of streaming UEs. The control function of this container is composed of a monitoring and a scheduling component. The monitoring component observes the CQI of the UEs that are streaming videos and provides real-time feedback to the video streaming client about the maximum sustainable bitrate that a UE can achieve given its CQI. Based on that feedback, the client readjusts the bitrate of the video to avoid freezes. The scheduling component makes the scheduling decisions (priority and MCS) for the video flows of the UEs by communicating with the video streaming clients every second and observing the buffer size of each video stream. Flows of video streams with lower buffer sizes are given a higher priority, on the premise that such flows need faster treatment to avoid buffer freezes.

For our evaluation, we considered a scenario with 2 UEs streaming a video [97] using the DASH reference client [37]. UE 1 had a fixed optimal signal quality and for UE 2 we introduced a big drop of the signal quality at $t = 35$s and then a big increase at $t = 75$s. Results obtained when the service container was active and managing the video flows (Figure 5.18b) are compared against the regular non-assisted




(a) MVNOs managed

(b) OTT cross-layer optimized

Figure 5.18: Video streaming performance in MVNO-managed vs. OTT-optimized control of flows.

DASH service with the MVNO being responsible for scheduling all flows through a proportional-fair scheduler (Figure 5.18a). When the signal quality of UE 2 changes suddenly, the non-assisted mechanism fails to adapt fast, leading to buffer freezes. In comparison, the cross-layer optimized control of flows enabled by the service containers allowed the bitrate to adjust more appropriately and to avoid stalls in the video stream, thus improving the overall quality of the service.

This example is merely intended to demonstrate the capabilities enabled by service containers for OTTs, by allowing them to make control decisions using information unavailable to a non application-aware MVNO. More sophisticated control mechanisms could also be employed, like the cross-layer optimization mechanism for video delivery in [67] or the content-aware schedulers in [81] and [113].

## 5.7 Conclusions

We have presented Orion, a flexible RAN slicing architecture. Orion slices can be deployed dynamically over the network infrastructure, co-existing in a fully isolated manner in terms of radio resources and control functions. At the same time, Orion facilitates efficient radio resource sharing among slices. A prototype implementation of Orion was developed for LTE, with the performance evaluation indicating that it is a lightweight and flexible RAN virtualization solution. Orion's capabilities were highlighted through a number of use cases, revealing its suitability for future multi-service mobile networks.

# Chapter 6

# Iris: Shared Spectrum Access for Neutral-Host Small-Cell Deployments

## 6.1 Introduction

The design of Orion that was presented in Chapter 5 can enable the creation of isolated slices over a generic underlying network infrastructure that can be shared among multiple tenants. However, as explained in Section 1.4.3, while the radio resource allocation mechanism provided by Orion is designed so that it can accommodate the needs of tenants with strict performance guarantees (e.g. in a setting of verticals), it can be limiting in scenarios where providing SLAs is not possible or desirable.

In this chapter, we focus into one such emerging use case in the context of indoor small-cell deployments with shared spectrum. We propose a system called Iris, that acts as a framework for enabling multi-tenancy by leveraging the virtualization capabilities of Orion. The main challenge in this scenario is that, given the absence of SLAs and the similarity of the provided services, tenants need to have a way to differentiate and compete in the market. In this context, we propose extension to the capabilities of the Orion `Hypervisor` through an alternative radio resource allocation mechanism based on dynamic pricing. Such a mechanism can provide more control to the tenants of the network, allowing them to dynamically request and allocate resources based on their valuations.

This chapter begins by presenting a modeling framework for the dynamic pricing mechanism of Iris in Section 6.2. This mechanism is embedded into a holistic system design specifically tailored for multi-tenant indoor small-cell deployments that is presented in Section 6.3, along with the system's concrete implementation. Finally,

a thorough evaluation of Iris is presented in Section 6.4, characterizing the behavior and the performance of Iris's dynamic pricing mechanism and comparing it against alternative pricing mechanisms.

## 6.2 Iris Dynamic Pricing Mechanism

This section describes the core component of Iris – its dynamic pricing mechanism – starting with the requirements the mechanism should meet.

### 6.2.1 Requirements and Overview

1. The price set by the neutral-host per unit of spectrum should seek to match the spectrum supply with the tenants' demand. A high price can lead to underutilization of resources. A low price, on the other hand, comes in the way of service differentiation among tenants, since all tenants will look to buy resources aggressively to fully satisfy their demand. Consequently, in periods of congestion, when the aggregate demand can only be partially satisfied, tenants who value the spectrum more will end up getting less resources relative to their requested amount, leading to their dissatisfaction.

2. The shared spectrum acquisition cost of the neutral-host must be factored in setting the price to ensure that the host does not incur losses in the long run.

3. Given the tenants are mutual competitors, they do not reveal any private information regarding their business model to the neutral-host. Instead they operate in isolation with respect to each other and the neutral-host and can dynamically change their private spectrum valuations. The pricing policy of the neutral-host should be able to adapt to such changes.

4. The pricing mechanism should be practical enough to admit a practical implementation in the context of a shared spectrum based neutral host small cell system architecture.

The obvious approach for the pricing is a centralized scheme for the allocation of resources and charging of tenants. However, this approach does not meet requirement (3) above as private valuation information of the tenants need to be revealed to the centralized entity. The other alternative is to employ a static pricing approach; the issue with this approach is that there may not be a single optimal fixed price as tenant

behavior, traffic load and spectrum availability varies. So instead, we adopt a dynamic pricing approach that allows the neutral-host to dynamically adjust the price depending on the network conditions and the tenants' behavior. In particular, considering the stochastic nature of the shared spectrum based neutral host small-cell environment (varying traffic loads, fluctuation in spectrum availability, changing tenant behaviors), *we formulate the dynamic pricing decision problem of the neutral host as a Markov Decision Process (MDP) and solve it using a deep reinforcement learning algorithm.* The following subsection first describes our system model (summary of model parameters in Table 6.1) before proceeding to the problem formulation and the solution.

Table 6.1: Summary of Iris model parameters.

| Parameter | Description |
|---|---|
| $t$ | Dynamic pricing epoch |
| $p^t$ | Price announced in epoch $t$ |
| $e$ | Number of epochs in each period |
| $n^t$ | Number of RBs available in epoch $t$ |
| $C(\cdot)$ | Shared spectrum acquisition cost |
| $I$ | Set of tenants participating in Iris |
| $l_i^t$ | Load of tenant $i$ in epoch $t$ |
| $\nu_i^t$ | Resource blocks requested by tenant $i$ in epoch $t$ |
| $u_i^t$ | Radio RBs allocated to tenant $i$ in epoch $t$ |
| $w^+$, $w^-$ | Bias parameters of reward function for demand–supply mismatch |

### 6.2.2 System Model

**Tenant resource requests.** In our model, tenants express their resource requests in terms of radio RBs. We assume that the tenants have a way to map their aggregate throughput demands into the number of radio RBs required (e.g., by assuming an average spectral efficiency for every RB). Such a mapping is reasonable, considering that the small-cell deployment throughout the indoor space is typically planned and it can provide near-optimal performance to user UEs within 20-30m [122].

**Neutral-host and tenant interaction.** The neutral-host follows a time slotted operation for the allocation of shared spectrum, where each slot will henceforth be referred to as an *epoch*. The duration of an epoch is expected to be short (e.g., 20-100ms), allowing the neutral-host to allocate radio resources in real-time. In each epoch $t$, the neutral-host determines a RB price $p^t \in [p_{min}, p_{max}]$ with which the tenants can buy

the available resources. The range of possible prices is assumed to be known to the tenants before they join the system (e.g., specified in their contract). Without restricting generality, in the following, we consider dynamic pricing for the allocation of downlink radio resources; the uplink can be treated similarly.

In each epoch $t$, all tenants see the price $p^t$ announced by the neutral-host and decide how many resources to buy. The neutral-host is oblivious to the behavior of the tenants, not knowing the internal mechanism (possibly changing over time) that governs their decisions. Consequently, the high level goal of the neutral host would be to "predict" the demand of tenants at any point in time and dynamically decide on a price that would utilize the resources as efficiently as possible while maximizing the tenant satisfaction, by allocating the (virtual) radio resources to those that need it the most. As discussed later, the model presented here is compatible with very general tenant behavior patterns, deterministic (e.g., driven by the optimization of utility functions) or not.

To model the temporal evolution of tenants' demand, we divide a day into $H$ periods, each $e$ epochs long, so that $He$ epochs make up 24h in the day. This construction makes a period correspond to an appropriate time interval within a day (e.g., an hour in a day) so that tenants' behavior is not expected to vary within a period but could across periods. Clearly, the shorter the period, the finer the granularity at which tenant behavioral changes can be captured. Without restricting generality, one may index periods within a day in the range $0 \leq h < H$ and may take the evolution of the system to start at epoch $t = 0$ coinciding with the beginning of a day. With this convention, the index of the current epoch $t$ maps to the index of the current period of the day as: $h(t) = (t \operatorname{div} e) \bmod H$. In the rest of this section, we will use superscripts of the form $\cdot^t$ to denote the time dependency of any quantity including cases when it occurs indirectly through $h(t)$.

**Shared spectrum acquisition and cost for the neutral-host.** Let $n^t \in \mathbb{Z}$ be the number of radio RBs obtained from the external/public spectrum repository for the tenants in epoch $t$. To maintain flexibility, the pricing mechanism regards the interaction between the public repository and the neutral-host in abstract terms. Consequently, $\{n^t, t \geq 0\}$ is a stochastic process and the neutral-host, although informed about the current value $n^t$, is unaware of the process' future dynamics so dynamics of a very general form can be accommodated. The only assumption (to enable the MDP framework discussed later) is that $n^{t+1}$, conditioned on the value of $n^t$, follows an (unknown to the neutral-host) probability distribution that may depend on $t$ and/or the current load of

the tenants. This is a very mild assumption compatible with virtually all scenarios of practical interest.

To model the shared spectrum acquisition cost, we introduce a convex, increasing function $C(\cdot)$. The value $C(n^t)$ represents the cost of the neutral-host to obtain $n^t$ RBs and pertains to the particular small-cell in question.

**System dynamics and neutral-host's small-cell resource allocation.** Let $I$ be the set of tenants served by the system. For each tenant $i \in I$, the expected load of a cell in epoch $t$ is denoted by $l_i^t$, representing the total traffic that tenant $i$ is expected to serve during epoch $t$. For example, this could be the bytes stored in the transmission buffers of all the UEs attached in the slice of the tenant, including a forecast of any new traffic expected during epoch $t$. Again, the model can accommodate very general dynamics for the evolution of $l_i^t$, for all $i \in I$. The only assumption made (to enable the MDP formulation) is that $l_i^{t+1}$, conditioned on the value of $l_i^t$, follows a probability distribution that may depend on one or more of: the time $t$, the amount of radio resources $n^t$, and the price $p^t$.

As already explained, the expected loads $l_i^t$, $i \in I$ and the amount of shared spectrum resources $n^t$ may mutually affect each other's values at the next epoch. The load $l_i^t$ may also affect the tenant's current RB request $\nu_i^t \in \mathbb{Z}$. For maximum flexibility, such a potential dependence is not made explicit and tenant $i$'s behavior at epoch $t$ is directly captured through the resource request $\nu_i^t$. The dynamics of $\nu_i^t$ can be general, the only restriction being that $\nu_i^{t+1}$, conditioned on the value of $\nu_i^t$, follows an (unknown to the neutral-host) probability distribution, whose form may depend on one or more of: the time $t$, the current tenant load $l_i^t$ and the price $p_t$.

The collective request across all tenants may exceed the amount of available resources, i.e., it is possible for $\sum_{i\in I} \nu_i^t > n^t$. In such a case, the neutral-host would allocate the available resources proportionally to the tenants' requests. By using $u_i^t$ to denote the resources actually allocated to tenant $i$ in epoch $t$, this allocation rule translates into

$$u_i^t(\vec{\nu}^t) = \frac{\nu_i^t}{\sum_{j\in I} \nu_j^t} \min\{ n^t, \sum_{j\in I} \nu_j^t \}, \qquad \forall i \in I, \tag{6.1}$$

where $\vec{\nu}^t$ stands for a vector containing the requests from all tenants. Always, $\sum_{i\in I} u_i^t \leq n^t$. Moreover, $u_i^t = \nu_i^t$ for all $i \in I$, when there are sufficient radio resources to meet all resource requests.

### 6.2.3 Problem Formulation

Given the system model just described and due to the stochastic nature of the shared spectrum based neutral host small-cell environment (varying traffic loads, fluctuation in spectrum availability, changing tenant behaviors), we formulate the neutral-host's dynamic pricing problem in Iris as a discrete-time, continuous state and action space MDP. Essentially, the neutral-host observes the state of its environment and makes a decision for an action based on this observation, bringing the environment into a new state and obtaining a reward for the corresponding transition. In the dynamic pricing context, the action of the host is to decide on the price $p^t$. We denote this action as $a^t \in A$ where $a^t = p^t$.

The state of the neutral-host's environment in epoch $t$ is denoted by $x^t$ and includes the vector $\vec{\nu}^{t-1}$ of virtual radio resources requested by the tenants in the previous epoch $t-1$, a vector $\vec{l}^t$ containing the current loads $l_i^t$ of the tenants, the epoch $t$ and the number of available radio resources at the neutral-host $n^t$. Formally, this is defined as

$$x^t := (t, \vec{\nu}^{t-1}, \vec{l}^t, n^t) \in X. \tag{6.2}$$

Note that $x^t$ is known to the neutral-host since it either contains information maintained by itself or obtained from the tenants every time they request resources.

The reward function of the neutral-host reflects the requirements of Section 6.2.1 and is defined as

$$r(x^{t+1}, p^t \mid x^t) = f\left(\frac{n^t - \sum_{i \in I} \nu_i^t}{n^t}\right) g(p^t, n^t, \vec{\nu}^t). \tag{6.3}$$

In the first factor on the right hand side, the function $f$ has the Gaussian form

$$f(k) = \begin{cases} w^+ e^{-\frac{k^2}{\sigma^2}}, & k > 0, \\ w^- e^{-\frac{k^2}{\sigma^2}}, & k \leq 0, \end{cases} \tag{6.4}$$

which expresses how effectively the available resources are utilized by the tenants, rewarding a close match of supply and demand. Ideally, resource under-utilization ($n^t - \sum_{i \in I} \nu_i^t > 0$) is to be avoided, as it leaves room to allocate more resources to tenants and increase their satisfaction. Excess demand ($n^t - \sum_{i \in I} \nu_i^t < 0$) should also be avoided, as the proportional allocation in rule (6.1) gives some tenants fewer RBs than the amount they requested at this price, leading to a decrease in their satisfaction. Both aspects are reflected in (6.4) – the highest reward comes only when demand fully meets the supply. The argument of $f$ in (6.3) is normalized by the number of available

RBs $n^t$ to make the reward function behave in the same way regardless of the amount of available spectrum. The parameters $w^+$ and $w^-$ introduce bias to indicate whether pricing decisions leading to excess demand of magnitude $|k|$ are preferable over decisions leading to resource underutilization by the same amount or vice versa. When $w^- = w^+$ there is no bias. When $w^- > w^+$, the neutral-host achieves higher rewards for pricing decisions leading to excess demand. Similarly, when $w^- < w^+$, decisions that lead to the underutilization of resources are rewarded higher. The greater the difference between $w^+$ and $w^-$, the higher the bias. Finally, the parameter $\sigma$ indicates how small the supply-demand mismatch $(n^t - \sum_{i\in I} v_i^t)$ should become to get a high reward. A small $\sigma$ attenuates the reward more sharply for a given mismatch, forcing the neutral-host to develop more accurate pricing policies.

In the second factor of (6.3), the function $g$ has the form

$$g(p,n,\vec{v}) = \min\Big(1, \frac{p\sum_{i\in I} u_i(\vec{v})}{C(n)}\Big)^{\delta}, \qquad \delta \geq 1 \tag{6.5}$$

which acts as a scaling factor reflecting the neutral-host's losses. When the revenue $p^t \sum_{i\in I} u_i^t(\vec{v}^t)$ is at least equal to the neutral-host's cost $C(n)$, the right hand side in (6.5) attains the highest possible value 1 and the neutral-host receives a full reward. If it instead incurs losses, the reward is scaled down by $g < 1$, forcing the host to influence the price decision that could raise its revenue to offset the costs. The higher the value of $\delta$, the more aggressively the neutral-host scales down the reward. Note that $g$ attains the maximum value 1 regardless of how big a profit is made by the neutral-host, so the exact value of the profit has no impact to the host's reward (and behavior). This is in line with Section 6.2.1 since the host's goal is to avoid losses, efficiently utilize the resources and provide service differentiation for the tenants.

With the reward function (6.3), the behavior of the neutral-host is defined by a policy $\pi$, which maps the states to a probability distribution over the actions $\pi : X \rightarrow Pr(A)$. With the mild assumptions stated in Section 6.2.2 and the structure of the state in (6.2), the state transitions from $x^t$ to $x^{t+1}$ given the action $a^t$ satisfy the Markov property and thus, applying a policy $\pi$ to this MDP defines a Markov chain. We denote expectations over this chain by $\mathbb{E}_\pi$. We define the return from a state $x^t$ as the sum of the discounted future reward: $R^t = \sum_{\tau=t}^{\infty} \gamma^{(\tau-t)} r(x^\tau, a^\tau)$ for a discounting factor $\gamma \in [0,1]$ [116]. The goal of the neutral-host then is to find a pricing policy that will maximize its expected returns from the start state $\mathbb{E}_\pi\left[R^0\right]$ over an infinite horizon.

### 6.2.4 Deep Reinforcement Learning Solution

Reinforcement learning is a common way to solve MDP problems of the kind presented in Section 6.2.3, where an exact model describing the dynamics of the environment (e.g., tenant behavior and network traffic in our case) is not available. Q-learning [137] is an obvious algorithm to use for this purpose. The idea underlying Q-learning is that we have an action-value function $Q^{\pi}$ which describes the expected future return after taking the action $a^t$ in some state $x^t$ and following policy $\pi$ from that point on, i.e.,

$$Q^{\pi}(x^t, a^t) = \mathbb{E}_{\pi}\left[R^t | x^t, a^t\right]. \tag{6.6}$$

This function can be expressed through a recursive relationship known as the Bellman equation:

$$Q^{\pi}(x^t, a^t) = \mathbb{E}_{r^t, x^{t+1} \sim X}\left[r(x^t, a^t) + \gamma Q^{\pi}(x^{t+1}, \pi(x^{t+1}))\right]. \tag{6.7}$$

Q-learning is an off-policy method in that the policy $\pi$ used to estimate the discounted future reward is different from the policy used for the action of the learning agent in a state transition. The policy used for the estimation of the discounted future reward of Q-learning is the greedy policy $\pi(x) = \arg\max_a Q(x,a)$ whereas an exploration policy is employed for the state transitions (e.g., take random actions).

Though a straightforward choice, Q-learning cannot be directly applied to our problem for several reasons. Firstly, it uses a table to store its Q-values. When the state space of the problem is continuous or very large (as in our problem due to the range of possible values for $l_i$, $\nu_i$ and $n$), calculating $Q^{\pi}$ using a table becomes challenging. To overcome this, we can rely on function approximators [24] parametrized by $\theta^Q$. These parameters can be optimized by minimizing the loss:

$$L(\theta^Q) = \mathbb{E}_{\pi'}\left[(Q(x^t, a^t | \theta^Q) - y^t)^2\right], \tag{6.8}$$

where

$$y^t = r(x^t, a^t) + \gamma Q(x^{t+1}, \pi(x^{t+1}) | \theta^Q). \tag{6.9}$$

In addition to the large state space, we also have to deal with a continuous action space (the announced price) which needs to be discretized in order to use Q-learning. However, how to discretize the prices is not obvious or straightforward [24], since the price range and its interpretation can be environment dependent. Using a policy gradient actor-critic algorithm (e.g., [128]) is a way to overcome this problem. Such algorithms maintain a parametrized actor function $\pi(x|\theta^{\pi})$ that estimates an action policy and a parametrized critic function $Q(x,a|\theta^Q)$ that estimates the Q-values of

action-state pairs through the Bellman equation, as in Q-learning. The actor policy is improved at each step by performing a gradient descent considering the estimated values of the critic. Recent works (e.g., [83, 125, 39, 55]) show that using deep neural networks as the function approximators for the estimation of actors and critics can produce better results compared to using linear approximators, when the learning task presents similar complexity to the one considered here in terms of its dimensionality. Among others, it has been shown that higher rewards can be attained (avoiding local optima) and that in some cases convergence speed can be improved.

In view of the above, we choose to use a state-of-the-art deep reinforcement learning actor-critic algorithm called DDPG [83]. The use of deep neural network approximators allows DDPG to scale to high-dimensional state spaces and operate over continuous action spaces, making it an ideal candidate for Iris. One of its key features is the use of replay buffers (a type of cache) to sample prior transitions $(x^t, a^t, x^{t+1}, a^{t+1})$ which are used to train the neural networks. It also uses a technique called batch normalization that improves the effectiveness of the learning process when using features with different units and ranges (e.g., RBs and time). Finally, it uses a technique that employs slow-changing copies of the actor and critic networks, called target networks, which are used for calculating $y^t$. This has been shown to greatly improve the stability of the learning method.

**Algorithm 1** Iris Dynamic Pricing Mechanism Outline

**procedure** DYNAMICPRICE
  $t \leftarrow 0$
  Receive initial network state $x^0$
*loop*:
  Choose a price $p^t$ given the policy of the actor
  $a^t \leftarrow p^t + \varepsilon$, where $\varepsilon$ is exploration noise
  Execute action $a^t$ (`announce price to tenants`)
  Collect the radio resource requests of tenants $\vec{v}^t$ and distribute the allocated RBs $u_i^t(\vec{v}^t), \quad \forall i \in I$
  Calculate the reward $r^t$ and transition to the state $x^{t+1}$
  Update the actor-critic parameters $\theta^Q$ and $\theta^\pi$ (`DDPG`)
  $t \leftarrow t + 1$
  **goto** *loop*.
**end procedure**

Algorithm 1 gives an outline of the dynamic pricing mechanism in Iris. A new price $p^t$ is selected at each epoch $t$ (`line 5`) using the policy indicated by the actor $\pi(x|\theta^\pi)$. Some exploration noise $\varepsilon$ is added to the price to allow the agent to explore other states. The price is announced to tenants (`line 7`) and their radio resource requests are collected in return. Based on these requests, Iris neutral-host allocates the radio resources following the rule in (6.1) (`line 8`); then calculates the reward $r^t$ and transitions into a new state $x^{t+1}$ (`line 9`). The parameters of the actor and the critic network are updated based on the DDPG algorithm (`line 10`), which is the training step, and a new epoch $t+1$ begins. Note that the learning process of Iris never stops, allowing the pricing mechanism to re-train and adapt to the new environments as the tenants change their valuations for the radio resources over time, etc.

## 6.3 Iris System Architecture

Having described the dynamic pricing mechanism, the key component of Iris, we now present its overall system architecture design and implementation.

### 6.3.1 Design

Our design builds on the observation that the small-cell infrastructure sharing capability offered by a neutral-host is a particular albeit compelling use case of the broader RAN slicing in the 5G context. However, a vanilla RAN slicing system would be insufficient to address the specific needs of shared spectrum management and indoor small-cell environments. In the following, we highlight how we address these needs in our design (schematic shown in Figure 6.1). It embraces the C-RAN design paradigm, with BBUs centralized in a virtualized BBU pool located in an edge cloud (e.g., in the basement of the indoor space) and RRUs deployed throughout the building in a planned manner. The RRUs are connected to the BBUs over high speed channels (e.g., 10-Gigabit Ethernet or Fiber). This approach places most of the RAN processing on the edge cloud which allows the system to scale better as BBU resources can be adaptively provisioned depending on the number of tenants and the spectrum availability. It also lowers the form factor of the RRUs, making their deployment easier and discreet from a building aesthetics viewpoint.

Each tenant is allocated a *Virtual RAN Controller*, deployed as a VNF over the edge cloud. The controllers interface with the BBUs following a message-based communi-

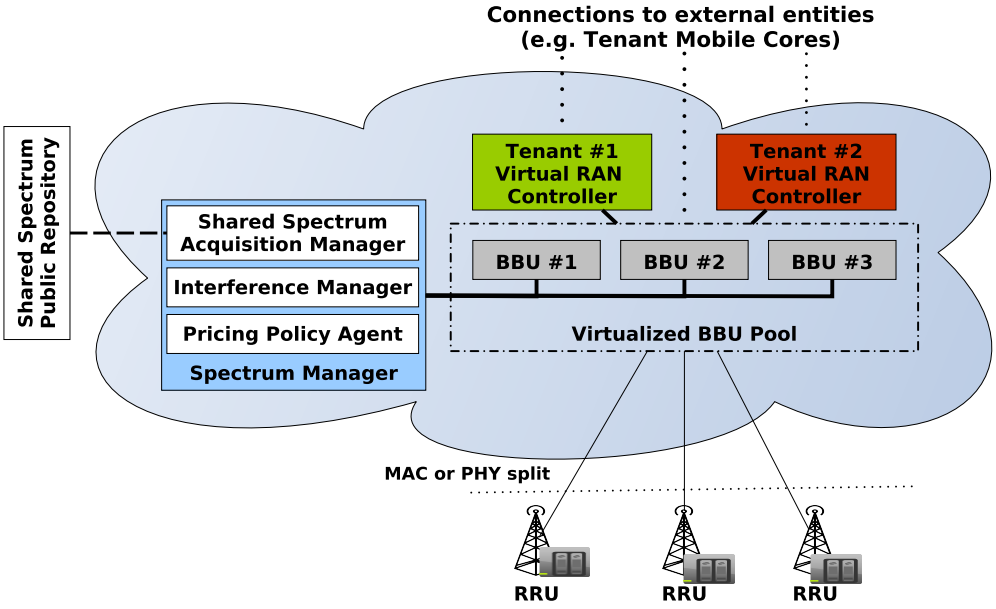



Figure 6.1: Schematic of Iris neutral-host system architecture.

cation and provide tenant-specific functions such as schedulers and mobility managers, as well as an agent for the allocation of shared spectrum (discussed shortly).

At the heart of Iris lies the *spectrum manager*, a centralized controller managed by the neutral-host. This controller informs the BBUs about the amount and type of available shared and privately owned spectrum and about its valid allocations, depending on the access rights of tenants, distinguishing in particular, between tenants operating exclusively over shared spectrum and tenants that can also use their own private licensed spectrum. A shared spectrum acquisition manager acquires the shared spectrum in a demand driven manner through a public repository (e.g., SAS in the CBRS context). Moreover, this controller manages interference among small cells. Due to the system's C-RAN based design, the VNF of the spectrum manager co-exists with the virtualized BBU pool over the same edge cloud, simplifying its coordination with the BBUs through low-latency and high bandwidth channels and enabling the use of advanced interference management techniques like CoMP [63].

**Shared spectrum allocation process in Iris.** Crucially, the spectrum manager hosts a pricing policy agent responsible to decide on the dynamic shared spectrum prices. The functionality of the Iris dynamic pricing mechanism (Section 6.2) is distributed among three distinct agents as illustrated in Figure 6.2.

The *pricing policy agent* initiates the shared spectrum allocation process in each epoch, deciding on the price for each cell using the deep reinforcement learning al-

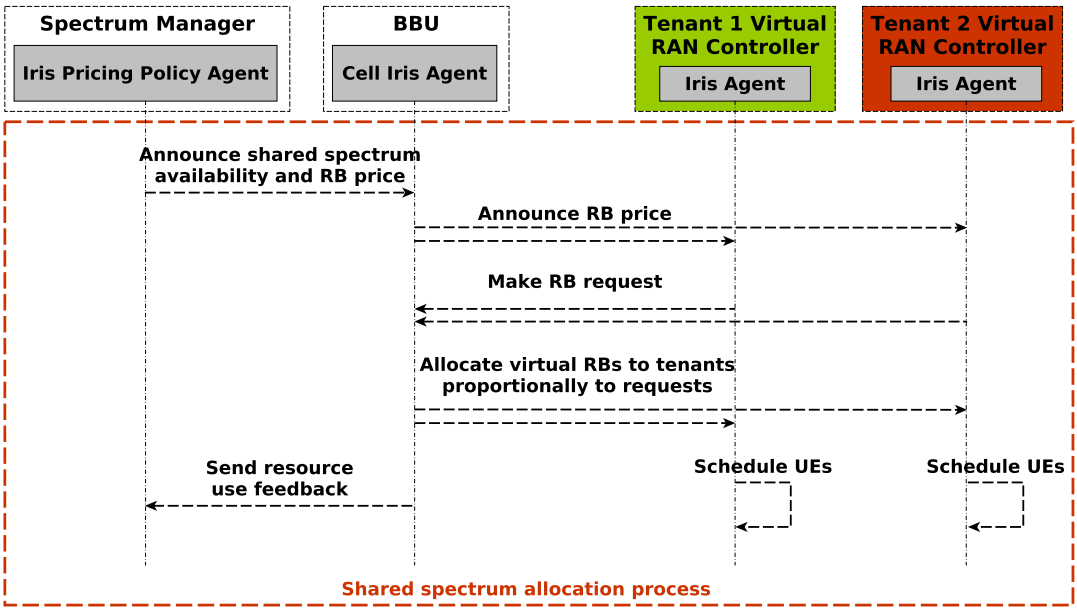



Figure 6.2: Iris agents involved in dynamic pricing mechanism.

gorithm described earlier in Section 6.2.4. The pricing policy agent announces the spectrum availability and the cell specific prices to the respective *cell agents* residing in the BBUs, which in turn convey the price to the *tenant agents* residing in the tenants' virtual RAN controllers. Each tenant considers the announced price along with its traffic load at the cell in question to decide on quantity of resource to be requested as dictated by its internal policy. Such tenant requests are aggregated at the cell agent, which distributes the available shared spectrum to the tenants proportionally to their requests following rule (6.1) and notifies the pricing policy agent about the new state of the cell, i.e., the allocated resources, the load of the tenants etc. The schedulers running within the virtual RAN controllers of the tenants use the allocated resources to serve the traffic of their UEs as per the tenants' internal policies. Once the shared spectrum allocation process is complete, the pricing policy agent is further trained using the feedback from the cells and decides on a new price for the upcoming epoch.

### 6.3.2 Implementation

Following the design described above, we developed a prototype implementation of Iris, considering LTE as the RAT. To realize RAN slicing (i.e., multi-tenancy within each BBU in the virtualized BBU pool), we leveraged Orion to enable the isolation of the virtual control planes of tenants and the virtualization of the radio resources revealed to them through the abstractions of vRRBs and vRRB Pools provided by the

Orion `Hypervisor`. As already explained in Section 5.3.4, Orion is in turn built on top of the OAI LTE platform. OAI has built-in C-RAN support offering three different functional splits: lower-PHY, higher-PHY and nFAPI [129]. Although in principle any of these functional splits could be used in Iris, the Orion implementation is only compatible with the first two. Between them, considering their differences in fronthaul bandwidth requirements (1Gbps with lower-PHY versus 280Mbps for higher-PHY for a 20MHz carrier) [91, 28], we opt for the higher-PHY functional split. Moreover, to allow the C-RAN capabilities of OAI to work with the slicing features provided by Orion, we had to modify the OAI code to synchronize the `Hypervisor` threads responsible for the communication with the control planes of tenants to the threads sending the I/Q data to the RRUs.

**Edge Cloud Deployment.** To realize the Iris system design, we created an OpenStack edge cloud deployment composed of 5 commodity compute nodes (4 Xeon CPUs @3.2GHz and 8GB RAM each), which were optimized for real-time operation (disabled CPU C-states, low-latency Linux kernel, no CPU frequency scaling). Moreover, the compute nodes were instructed by the OpenStack controller to use CPU pinning, i.e., dedicated CPU cores that were not shared with other VNFs. For the RRUs, we used low-cost SDRs (USRP B210), interfaced with compute nodes used for the communication with the BBUs (over Gigabit Ethernet) and for running the lower part of the PHY operations. We used small factor PCs (UP board with 4GB of RAM and Intel Atom x5 Z8350 CPUs @1.92GHz) as RRU compute nodes (shown in Figure 6.3).

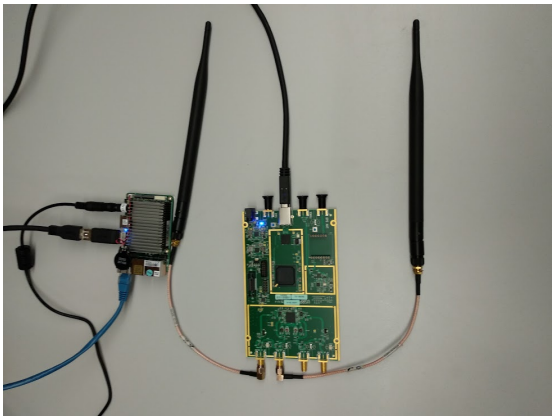


Figure 6.3: RRU equipment used for the prototype, composed of an SDR unit (right) and a basic compute unit (left).

**Spectrum Manager.** We implemented a prototype Python-based spectrum manager to

host the Iris pricing policy agent, employing an existing implementation of the DDPG algorithm [5] that uses Tensorflow for the training of the deep neural networks. Given our equipment constraints, we used a Tensorflow flavor that supports CPU execution (GPU-based variants could be more computationally efficient). Regarding the parameters of DDPG, we retained the default values provided in the aforementioned implementation. The neural networks of both the actor and the critic had two hidden layers with 400 and 300 units, respectively. For the representation of the state and action space of the dynamic pricing mechanism, we employed the OpenAI Gym [23] reinforcement learning toolkit.

In a full implementation, shared spectrum support should be introduced through carrier aggregation. However, given that this functionality is not currently supported by OAI, we simulated scaling the available shared spectrum up/down using a contiguous band of spectrum, scaled by the spectrum manager through signaling messages to the cell agents, restricting their operations to a portion of the spectrum.

**Cell Agents.** The cell agents of Iris are introduced by modifying the Orion`Hypervisors` in the BBUs. First, the radio resource allocation scheme of Iris following rule (6.1) was introduced into the radio resource manager of the `Hypervisor`. Moreover, the `Hypervisors` were interfaced with the spectrum manager using Google Protocol Buffers [2] and ZeroMQ [1]. Finally, the protocol used for the communication of the `Hypervisors` with the virtual control planes of tenants was extended to support the messages required for the operation of the Iris tenant agents, i.e., for the shared spectrum price announcements and the radio resource requests of tenants.

**Tenant Agents.** On the tenant side, we leveraged the virtual control plane implementation of Orion, which we extended with the Iris tenant agents. In these agents, we implemented a generic parameterizable dis-utility function that can be customized by each tenant to decide on the allocation of shared spectrum based on the announced price and its load. With each tenant we associated a dis-utility function of the form

$$\bar{U}(p,d,b) = \Big(\alpha(\max(0,d-b))^{\gamma_d} + (pb)^{\gamma_p}\Big)^{1/\gamma_p}. \tag{6.10}$$

All arguments of the function refer to the same epoch. Specifically, $p$ stands for the price announced by the pricing policy agent, $d$ expresses the tenant's traffic load in terms of radio RBs and $b$ represents the number of requested RBs. The parameters $\alpha$, $\gamma_d$ and $\gamma_p$ characterize the individual tenant behavior; the settings of these param-

eters determine the sensitivity of the dis-utility function to the current load or price and can therefore allow simulation of different tenant behaviors and reactions to price changes (as elaborated further in Section 6.4). An API is provided in the controllers of the tenants for these parameters can be modified on-the-fly, allowing the tenants to dynamically change their shared spectrum allocation policy. Raising the sum in (6.10) to the power $1/\gamma_p$ expresses the value of the dis-utility in units of "cost", bearing the same interpretation for all tenants, regardless of the value of $\gamma_p$ employed in each of them. This allows introducing the notion of "total dis-utility" calculated as the sum of dis-utilities over all tenants. Through (6.10) and given the price $p^t$ announced by the Iris pricing policy agent and the level of traffic load $l_i^t$, corresponding to $d(l_i^t)$ RBs, the agent of each tenant $i$ requests from the Iris cell agent the amount of RBs that minimizes its dis-utility i.e., $\arg\min_b \bar{U}_i(p, d(l_i^t), b)$. When mapping the load of a tenant from bytes to RBs, the current implementation employs a static correspondence, assuming a 64-QAM modulation scheme allowing each RB to serve on average 80 bytes, i.e. $d(l_i^t) = l_i^t/80$.

It is important to note that although we implemented the form in (6.10) to realize different tenant behaviors for our evaluations, our design (specifically the dynamic pricing mechanism) is completely agnostic to actual tenant utility functions and does not make any assumptions about their nature.

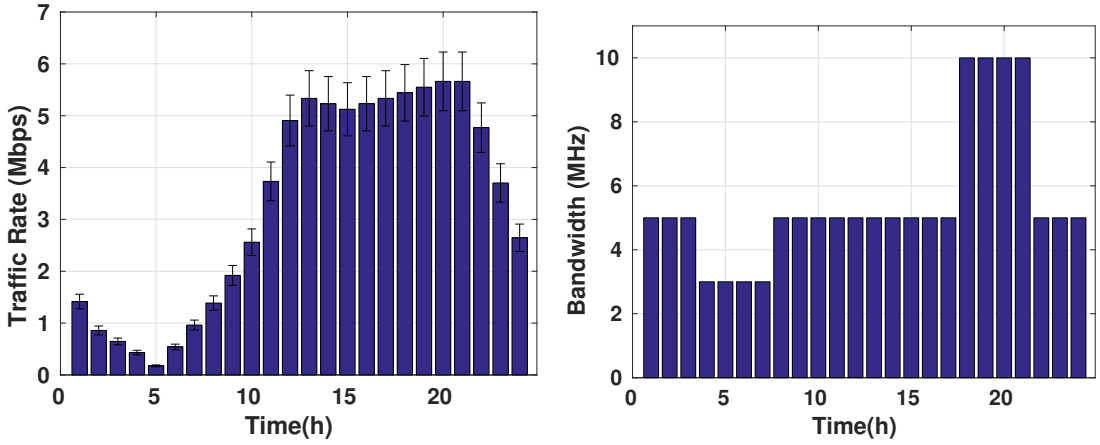
## 6.4 Experimental Evaluation

### 6.4.1 Evaluation Setup

We use the prototype implementation of Iris (Section 6.3.2) in our evaluations. The default experimental setup corresponds to 4 tenants per cell. For experiments with a single small cell, real UEs (LTE smartphones and dongles), one per tenant, and the D-ITG traffic generator [22] were used to generate a simulated aggregate UDP traffic for the tenant. Scenarios with multiple small cells were realized using simulated UEs, due to the complexity of managing the experimental setup.

To simulate the traffic loads of tenants, we used the daily aggregate traffic pattern presented in [136] for an entertainment area, which is one of the representative indoor environments. We assume that the aggregate incoming hourly traffic of each tenant follows a normal distribution, with mean and variance depending on the particular hour in the day considered, as illustrated in Fig. 6.4a. The shared spectrum announced

by the spectrum manager is based on the spectrum availability profile in Figure 6.4b. The idea behind this profile is that the available shared spectrum is re-adjusted (with some delay) by the spectrum manager to approximately match the traffic load. While some of the experiments span the whole day and use the full profiles in Figure 6.4, others focus on a particular hour and employ the traffic and spectrum values for that hour. The default evaluation configuration considers the conditions applicable at 3pm and a cell with 5MHz of available shared spectrum, emulating a CBRS-like service using LTE band 7. Small-cells use a Single-Input Single-Output transmission mode, which for 5MHz spectrum corresponds to a max throughput of 16Mbps.




(a) Traffic load

(b) Spectrum availability

Figure 6.4: Daily traffic profile of tenants and spectrum availability profile over a day at the neutral-host.

For the dynamic pricing mechanism, the presented results correspond to the price range $[0, p_{max}]$, with $p_{max} = 2500$. For the shared spectrum acquisition cost we employ the linear cost function

$$C(n) = p_c n, \tag{6.11}$$

where $p_c$ is the cost incurred to the small cell for the acquisition of a single RB. We set $p_c = 0.25 p_{max} = 625$, unless explicitly stated otherwise. The epoch duration is set to 30ms (chosen based on results of Section 6.4.4). The parameters of (6.4) are configured empirically, with the bias parameters set to $w^+ = 9.7 \times 10^3$ and $w^- = 10^4$ to promote solutions avoiding resource underutilization and $\sigma = 27$, allowing the pricing policy agent to attain a high reward only when the supply-demand mismatch is in the order of a few tens of RBs.

For the shared spectrum requests of the tenants, we created the 4 profiles listed in

Table 6.2: Tenant profiles with different parameterizations of the generic dis-utility function and the resulting behaviors.

| Profile # | $\alpha$ | $\gamma_p$ | $\gamma_d$ | Effect |
|---|---|---|---|---|
| 1 | $3.5 \times 10^8$ | 2 | 1 | The tenant is only willing to cover its load when the price is low. Otherwise, it buys a minimum amount of resources to maintain a presence in the network. |
| 2 | $2 \times 10^9$ | 2 | 1 | The tenant is willing to fully cover its load for low to medium prices. For high prices, it only covers part of its load. |
| 3 | 0.203 | 1 | 2 | In periods of high load, tenant is willing to buy a large amount of resources even at high prices. In other times, the tenant will queue its traffic until the load increases enough to buy in bulk. |
| 4 | $1.1 \times 10^5$ | 2 | 2 | Tenant always tries to provide a medium level of service, asking a price-dependent fraction of its load. |

Table 6.2, that configure the dis-utility function in (6.10) to simulate various tenant behaviors. The expected effects of each configuration to the behavior of the tenants (for the price range considered) are explained in Table 6.2. Unless explicitly stated otherwise, tenants were assigned profiles in a cyclic manner, i.e. tenant 1 to profile 1, tenant 2 to profile 2, tenant 5 to profile 1 etc.

### 6.4.2 Characterizing Iris Shared Spectrum Management

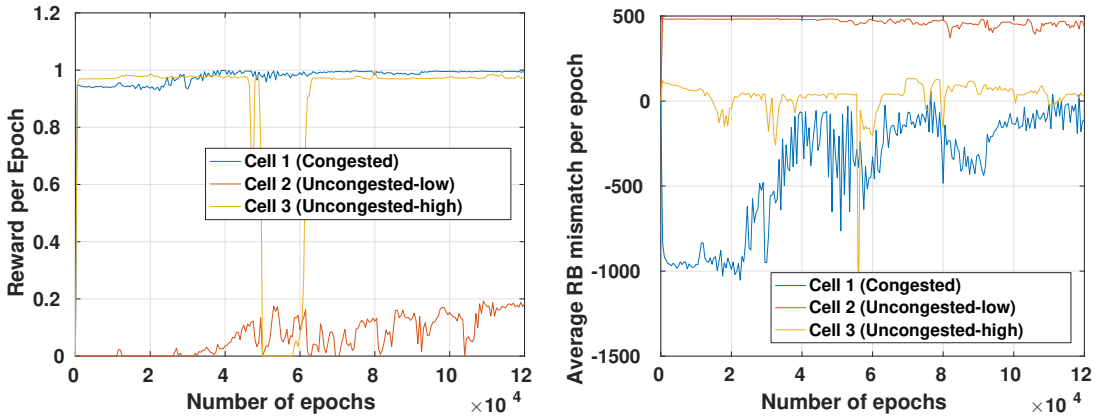
**Learning behavior for various traffic loads.** We evaluate the learning behavior of the pricing policy agent for four tenants over three cells, with each cell having a different aggregate traffic load. We consider a congested cell (Cell 1), corresponding to the conditions at 3pm from the daily traffic profile of Figure 6.4, an uncongested cell (Cell 2) experiencing low traffic load, corresponding to the conditions at 8am from the daily traffic profile and a cell with high traffic loads (Cell 3) that are not leading to congestion, corresponding to the conditions at 11am from the daily traffic profile.

The results in Figure 6.5 depict the reward obtained by the pricing policy agent (normalized over the maximum possible reward) and the average mismatch between the RBs available during an epoch and those collectively requested by all tenants (i.e., $\sum_{i \in I} v_i^t$). A zero mismatch corresponds to an exact match between demand and supply. On the other hand, a negative mismatch signifies a collective tenant request that led to excess demand and a positive mismatch signifies underutilization, due either to high price or low load.

As we can observe, in the congested cell (cell 1) case, the agent begins with a

very high RB mismatch, which drops by more than half after the first 40000 epochs and constantly gets closer to 0 as time progresses, indicating that the pricing policy agent is effectively learning how to control the requests of the tenants. This is also apparent by the reward of the agent that is close to 1 (the max), meaning that the agent both utilizes the radio resources, while it covers its expenses for the acquisition of the shared spectrum.

For cell 2, the mismatch is always positive and close to 500. This is expected, since the load of the tenants is very low and the demand can never match the supply. Due to the very low load, it is infeasible for the neutral-host to fully cover its expenses for the acquisition of the shared spectrum, regardless of the pricing policy it follows. This is reflected in its reward that is scaled down by (6.5). However, despite this and after the first 40000 epochs, the neutral-host agent discovers a pricing policy that improves its revenue, which is reflected into the improvement of the obtained reward (getting close to 0.2).




(a) Neutral-host reward

(b) Resource block mismatch

Figure 6.5: Learning behavior of pricing policy agent for cells with different levels of congestion and loads.

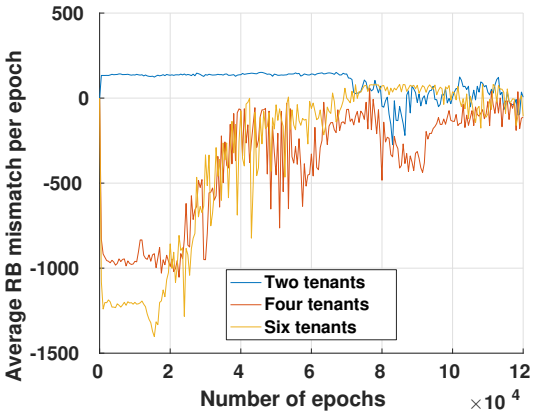
Finally, in the case of cell 3, the agent presents a stable behavior, with its reward (and RB mismatch) remaining relatively static throughout the experiment. This is because the aggregate traffic of the tenants requires an amount of RBs that is roughly equal to the RBs that are available in the system. Therefore, as long as the pricing policy agent does not announce very high prices, the spectrum will be almost fully utilized. However, for a short period of time after the first 45000 epochs, the RB mismatch undergoes a sudden increase (also reflected in the agent's reward). The root cause of this behavior lies in the exploration noise of the reinforcement learning pro-

cess, as presented in Algorithm 1. This noise makes the agent diverge from the pricing policy it already knows to explore new states that could potentially lead to better results. This can have a negative impact to the tenants, due to their sensitivity to the announced prices, which is reflected into the sudden degradation in the RB mismatch and the decrease in the reward.

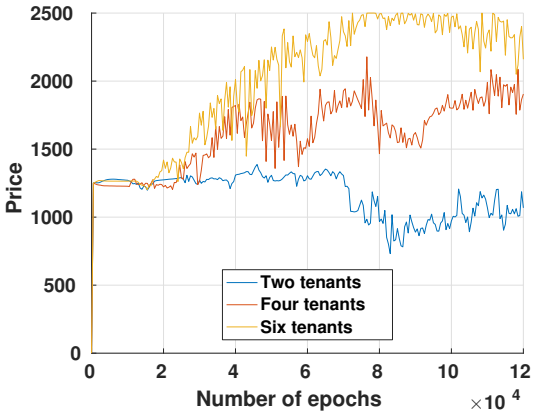
**Different number of tenants.** We evaluate the learning behavior of the dynamic pricing mechanism as the number and behavior of active tenants vary. (Recall that the behavior of each tenant depends on its index, as explained in Section 6.4.1). We consider three cases with two, four and six tenants and a congested cell with the aggregate traffic generated by all tenants being 16Mbps (all tenants generating equal levels of traffic). The average RB mismatch is illustrated in Figure 6.6a. In the case of four and six tenants the system begins with a negative mismatch, while for two tenants with a positive mismatch. This is related to the effect of neutral-host's pricing choices to the tenants (Figure 6.6b), given their loads and shared spectrum allocation profiles. For two tenants, the initial prices (around 1300) are considered high, leading to the underutilization of the resources, despite the cell congestion. On the other hand, for four and six tenants and given the increased competition for the resources, the price is low, leading to excess demand. In all cases, the agent discovers an appropriate pricing policy to minimize the mismatch, showing the adaptiveness of the proposed mechanism. For four and six tenants this is translated into high and very high prices correspondingly, while for the two tenants into a price reduction.

**Effect of reward function.** In the next experiment, we explore how the behavior of the learning process can be affected by the reward function. We consider three different variants of the reward function; the one we used in our default setup, an unbiased reward function ($w^+ = w^- = 10^4$) and the default biased version of the function with the parameter $\sigma$ set to a high value ($\sigma = 750$).

The results for the effect of these configurations on the RB mismatch are in Figure 6.7a. The variant with the large $\sigma$ fails to provide acceptable results until very late in the experiment. This is because for a large $\sigma$, the reward attained by the agent for any type of demand expressed by the tenants (both high and low) is very similar, making the differentiation of good and bad pricing decisions difficult. The other two variants of the reward function yield similar results and present similar behavior. However, as it can be seen in the zoomed area of the figure, the biased version tends to favor pricing



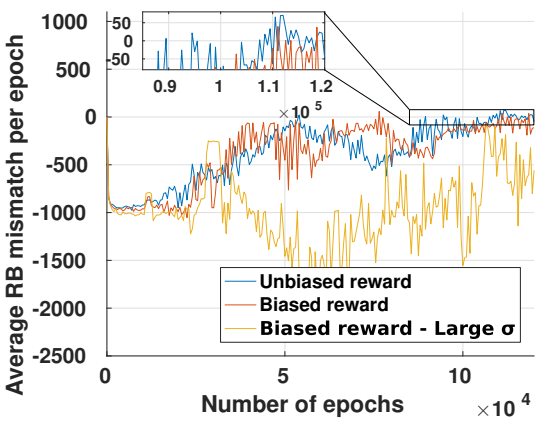

(a) Learning behavior for different number of tenants.



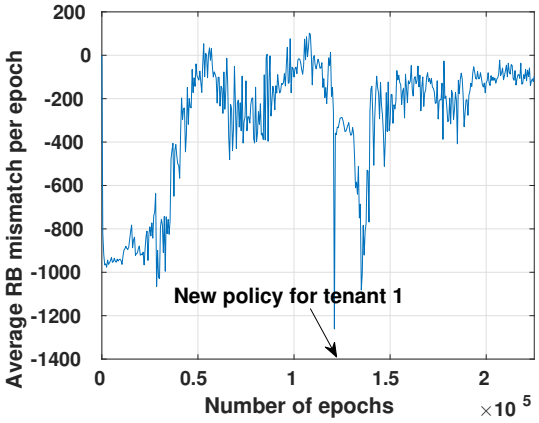

(b) Price decision for different number of tenants.

Figure 6.6: Behavior of Iris dynamic pricing mechanism for varying number of tenants.

policies that lead to a slight excess in demand (negative mismatch) rather than leaving part of the shared spectrum underutilized, like in the unbiased flavor (frequent positive mismatches).




(a) Effect of reward function parameters.




(b) Adaptation to dynamic policy changes.

Figure 6.7: Behavior of Iris dynamic pricing mechanism under different reward function parameters and dynamic tenant policy changes.

**Effect of dynamic policy changes.** This experiment demonstrates the adaptiveness of the dynamic pricing mechanism when tenants make policy changes dynamically. According to the scenario, the experiment runs for 120000 epochs using the default tenant profiles. When this period elapses, and once the agent has identified an appro-

priate pricing policy for the given load, the API of tenant 1 is used to dynamically change the tenant's policy into profile #2 (Table 6.2). This leads to a temporary failure of the agent to appropriately price the available radio resources, which is translated into a major RB mismatch, as depicted in Figure 6.7b. However, after about 30000 epochs (150000 epochs in the experiment), the neutral-host agent manages to re-adapt to the new behavior of tenant 1.

**Effect of shared spectrum acquisition cost.** Here, we evaluate the effect of the shared spectrum acquisition cost to the pricing decisions. For this, we consider two cases with a different acquisition price $p_c$. The first case employs the default value $p_c = 0.25 p_{max} = 625$, while the second case uses a very high acquisition price $p_c = 0.75 p_{max} = 1850$.

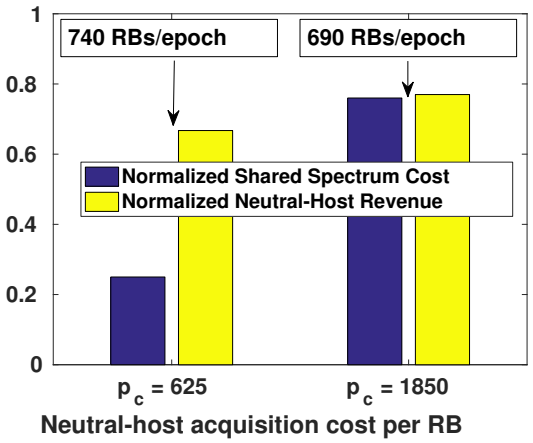



Figure 6.8: Impact of spectrum acquisition cost.

We compare the normalized revenue of the neutral-host against the normalized cost for acquiring the spectrum by the public repository. The normalization is performed in terms of the maximum revenue achievable by the neutral-host for the same period (i.e. selling all resources at price $p_{max}$). The results are in Figure 6.8. For the low acquisition price, the revenue of the neutral-host is much higher than the spectrum cost. Due to the high load (traffic at 3pm) experienced by the tenants and the network congestion, tenants are willing to buy the RBs in prices much higher than $p_c = 0.25 p_{max}$. However, for the high acquisition cost, the valuation of the tenants for the available resources is not high enough to cover the costs of the neutral-host. As a result, the neutral-host agent, driven by its reward function, learns an alternative pricing policy that makes the tenants buy less RBs on average (690 vs. 740 in the low price case) but at higher prices allowing the host to recover its spectrum acquisition cost.

### 6.4.3 Comparison with Alternative Approaches

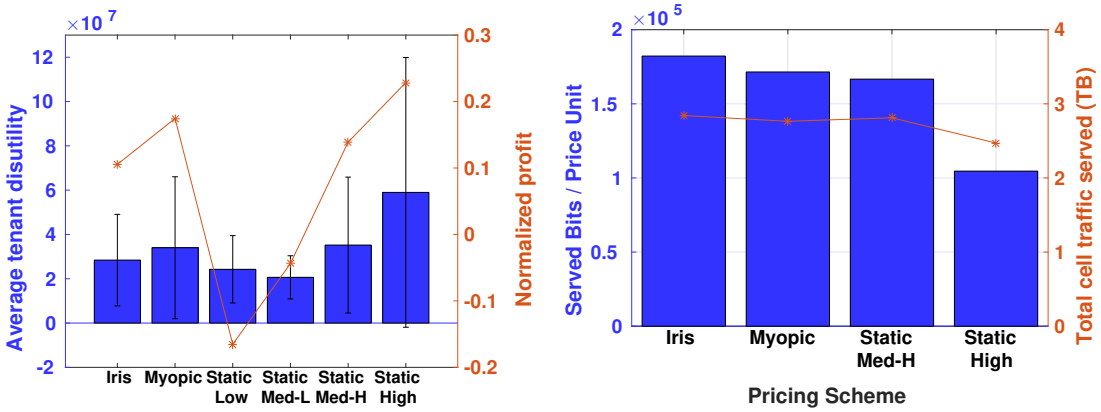
We now compare the performance of the Iris dynamic pricing mechanism with alternative approaches in terms of the benefits provided to tenants. We consider the traffic generated for a whole day (using the full profile presented in Figure 6.4) for four tenants with their behaviors defined in Table 6.2. We are interested in the performance of the neutral-host agent at day 0 of training, i.e., the agent is evaluated against other schemes without any prior training. This worst-case scenario is important to benchmark the effectiveness of the neutral-host agent's operation in volatile environments, where tenants can frequently change their valuations.

One alternative that could be viewed as a variant of the optimal solution is an unrealistic myopic pricing scheme in which the neutral-host knows the dis-utility functions of the tenants. Using this knowledge, it determines at each epoch, the price to charge the tenants by minimizing the sum of tenant dis-utilities, subject to the resource availability constraint and the requirement that the neutral-host experiences no losses, i.e.,

$$\begin{aligned} \underset{p,\vec{\nu}}{min} \quad & \sum_{i\in I} \bar{U}_i(p,d,\nu_i) \\ \text{s.t.} \quad & p\sum_{i\in I}\nu_i \geq C(n), \\ & \sum_{i\in I}\nu_i \leq n, \\ & \nu_i \geq 0, \quad \forall i \in I \end{aligned} \tag{6.12}$$

The neutral-host allocates the resources myopically during each epoch (in the sense that it views each epoch in isolation) so that it does not incur losses even in the short-term. A side-effect of this is that under very low traffic load, the neutral-host forces the tenants to buy more resources than they actually need, to balance the acquisition cost for the spectrum. It is noted that the comparative evaluation does not consider as a baseline the unrealistic but "optimal" solution which optimizes the allocation of resources considering the network dynamics (traffic, spectrum cost, tenant behaviors) throughout the whole day. The complex modeling requirements accounting for the dependency across epochs, the high computational complexity of obtaining the optimal solution and the fact that this needs to be performed over and over again in time makes it impractical even as a baseline.

In addition to the myopic scheme outlined above, we also consider four static pricing schemes, where the price announced by the agent during each epoch $t$ is fixed to $p_{max}/8 \approx 312$ (Static Low), $3p_{max}/8 \approx 937$ (Static Med-L), $5p_{max}/8 \approx 1562$ (Static




(a) Total tenant disutility and neutral-host profit. (b) Total traffic served and bits per price unit.

Figure 6.9: Comparison of Iris with alternative approaches.

Med-H) and $7p_{max}/8 \approx 2187$ (Static High) correspondingly, to capture the whole range of possible prices.

We begin by looking at the average dis-utility of the tenants for each pricing scheme, as well as the total normalized profit made by the neutral-host (Figure 6.9a). The profit is normalized by the maximum possible revenue of the neutral-host, i.e., selling all the available resources at the max price of $p_{max}$. In terms of the dis-utility, we can observe that Iris performs worse than two of the static pricing schemes with the lower prices (Static Low and Static Med-L). However, through the profit results, we also observe that with the low static pricing schemes the neutral-host experiences losses, something that could have been avoided if the neutral-host dynamically adapted its prices. This means that while the low static prices can improve the satisfaction of tenants, they also disincentivize the neutral-host to provide its service in the first place.

The results are opposite for the myopic and the higher static pricing schemes (Med-H and High) in that with these schemes the neutral-host obtains a higher profit at the expense of a higher tenant dis-utility compared to Iris . For the static pricing schemes, this is due to the inability of the pricing mechanism to adapt to the traffic loads, charging high prices even at times of no congestion (e.g., 1am-10am when the traffic load is low or 6pm-9pm when there is abundance of spectrum). For the myopic scheme, however, this is due to the neutral-host agent forcing tenants to buy resources not needed to myopically recover its spectrum acquisition cost within each epoch. These behaviors are better seen in the hourly breakdown of the tenants' dis-utilities and corresponding prices decided by the neutral-host as shown in Figure 6.10. The Iris dynamic pricing mechanism manages to draw a balance between the needs of the tenants and the

neutral-host more effectively, learning the right pricing policy that keeps the tenants as satisfied as possible, but without incurring losses that would disincentivize the neutral-host.

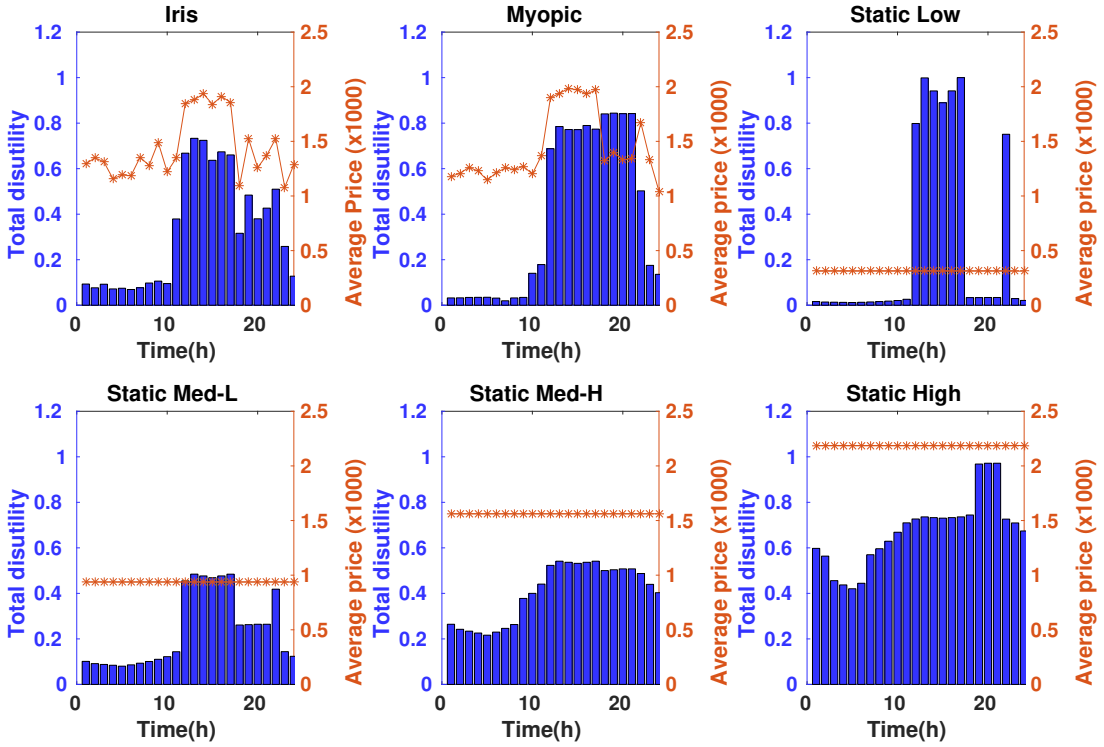
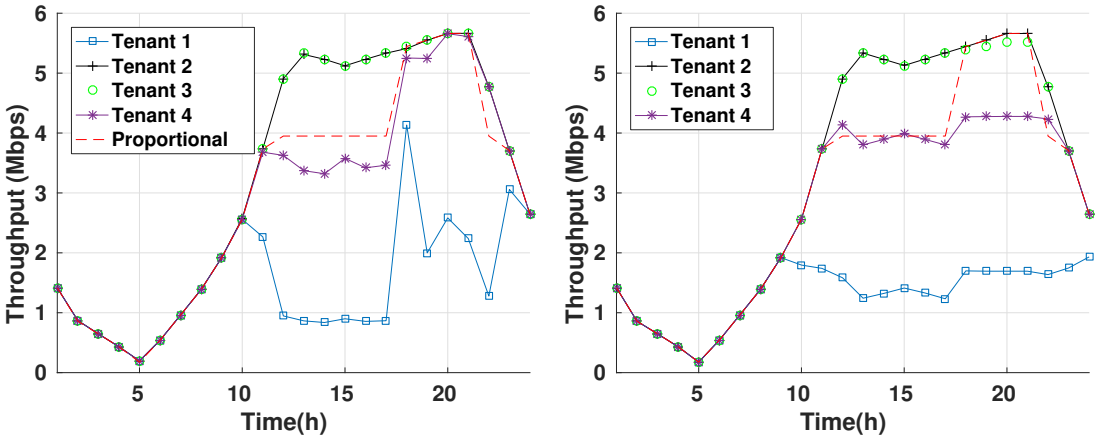



Figure 6.10: Hourly breakdown of tenants' total disutility and average price selected by neutral-host.

In terms of the offered service, we measure the total traffic served by a cell throughout the day and calculate the average bits per price unit that the tenants bought for each pricing scheme. We omit the static Low and Med-L schemes from this comparison, given the losses they incur to the neutral-host. The results are in Figure 6.9b. As we can observe, Iris offers the cheapest service, benefiting from the adaptiveness of its pricing scheme. Note that, although the same adaptiveness is also offered by the myopic scheme, the fact that tenants might be forced to buy unwanted resources raises the overall service cost. Another interesting observation is that, the total traffic served in the High static pricing scheme is significantly lower than that of Iris. This is because, due to the high prices, the tenants avoid buying radio resources despite the availability (evident from the fact that the other pricing schemes served more traffic with the same overall amount of resources).

Finally, we compare the service differentiation offered by Iris against the myopic scheme and the spectrum allocation policy proposed in [72]. The latter allocates RBs to




(a) Service differentiation in Iris vs proportional load allocation.

(b) Service differentiation in myopic allocation scheme.

Figure 6.11: Service differentiation among tenants with Iris and other approaches.

the tenants proportionally to their load, so it can be viewed as a purely load dependent but pricing agnostic scheme. For this result, the myopic scheme can act as a baseline, since in this case the neutral-host is aware of the dis-utility functions of the tenants and thus optimally distributes the resources among them. The results appear in Figure 6.11. As we can observe, Iris provides service differentiation among tenants, with results that are close to that of the myopic scheme. For the proportional scheme, we can see that no differentiation can be achieved (since every tenant generates the same traffic load). This can have a negative impact to the tenants' satisfaction, since the tenants that value the available spectrum the most end up getting less resources than they would like during hours that the network experiences congestion (e.g., 12-6pm).

### 6.4.4 Iris Deployment Feasibility

Here we examine the system requirements for deploying Iris. We begin with the average execution time required for a single training step of the pricing policy agent, i.e., the time required to calculate the new parameters of the actor and the critic functions by DDPG. This time is very significant, as it dictates the time granularity of the dynamic pricing epoch, i.e., an epoch cannot be shorter than the agent training time. We measure the execution time for a varying number of tenants with results shown in Figure 6.12. As we can observe, the execution time increases linearly with the number of tenants and remains below 22ms even for 8 tenants. This linear effect is correlated with the computational complexity introduced by the linear increase in the number of input

layer units in the neural networks of the actor and critic as the number of tenants grow – each tenant adds one load $l_i$ and one request $\nu_i$ feature to the input layer. When setting the epoch duration, the additional overhead introduced by the message exchanges between the various Iris agents should also be taken into account. We therefore chose 30ms for the epoch duration in all the earlier experiments. It should however be noted that offloading the training computations to GPUs (rather than using CPU as in our current implementation) can potentially lead to significant reductions in the execution time, which in turn allows the duration of the epoch to lower and Iris to learn faster.

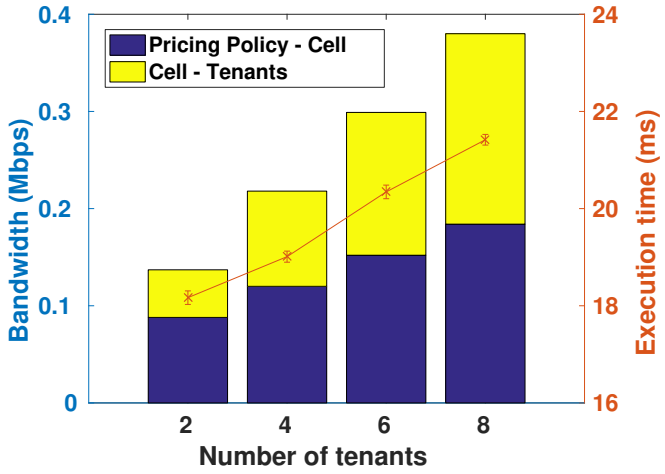



Figure 6.12: Execution time for single training step of Iris and bandwidth requirements for Irisagents message exchanges.

We also measure the bandwidth requirements for the message exchanges of Iris, which includes both the messages for the communication between the pricing policy and cell agents as well as the messages for communication between cell and tenant agents. As illustrated in Figure 6.12, the bandwidth requirements of Iris are minimal (less than 0.4Mbps) for all practical deployment scenarios of up to 8 tenants and posing a negligible overhead to the edge cloud deployment.

## 6.5 Conclusions

We have presented Iris, a system architecture for neutral-host indoor small-cells based on shared spectrum. The design of Iris follows a C-RAN approach that simplifies the control and deployment of the underlying infrastructure and leverages Orion for providing full isolation for tenants in terms of radio resource management. At the core of Iris lies a novel dynamic pricing radio resource allocation mechanism for shared spectrum.

This mechanism employs deep reinforcement learning to discover pricing policies that allow tenants to request shared spectrum resources on demand, ensuring the differentiation of their services based on their valuation of the spectrum, while avoiding losses for the neutral-host due to the acquisition cost of the shared spectrum from a public repository. Using our prototype implementation of Iris developed for LTE, we have conducted extensive experimental evaluations to characterize the dynamic pricing mechanism of Iris under different conditions, show the benefits of the Iris approach compared to alternative approaches and examine its deployment feasibility.

# Chapter 7

# Conclusions and Future Work

## 7.1 Conclusions

The emergence of 5G is expected to bring a wave of new technological and architectural changes that are designed to greatly improve the performance of mobile networks in terms of throughput, latency, scalability etc. At the same time, 5G mobile networks are envisioned to become multi-service environments, catering the needs of mobile network operators, verticals and over-the-top service providers by integrating inherent support for the dynamic deployment of services with a diverse set of performance requirements.

Unfortunately, the one-size-fits-all approach and the rigidness in the design of the conventional mobile network architecture proves to be a significant bottleneck for the evolution towards this vision. For the network to evolve, flexibility in the creation, management and control of the network components is of paramount importance. The key into enabling this flexibility is the introduction of programmability and virtualization capabilities for the mobile network functions. The focus of this thesis was on providing three relevant solutions in the context of the RAN, considering the challenges that arise by the idiosyncrasies of this part of the mobile network architecture compared to the core.

While each of the individual solutions focused on addressing different problems, the resulting systems are highly-complementary and contribute in the bigger picture of creating a next generation RAN that is flexible and adaptable to a wide-range of use cases. Big attention was given on creating realizable system designs, something that is evident both from the concrete prototype implementations that complement them and from the extensive evaluation results that indicate the feasibility and performance

of the proposed approaches. The high-level conclusions for each of these solutions is presented in the following subsections.

### 7.1.1 Software-Defined Radio Access Networks

The first contribution of this thesis was the FlexRAN platform, which is a concrete prototype implementation of an SD-RAN architecture. FlexRAN was specifically designed to take into consideration the challenges that appear in the RAN regarding the control of the radio resources and the stringent time constraints presented by a number of critical control functions. The main advantage of FlexRAN is that it provides a flexible control plane designed with support for real-time RAN control applications, flexibility to realize various degrees of coordination among RAN infrastructure entities, and programmability to adapt control over time and easier evolution to the future following SDN/NFV principles. Evaluation results reveal the feasibility of adopting an SD-RAN design for many practical deployment scenarios. The benefits of the SD-RAN architecture are further demonstrated by exploiting the FlexRAN prototype to implement a number of prominent use cases that are being considered in the context of both 4G and 5G, including Mobile Edge Computing, inter-cell interference management and RAN sharing.

### 7.1.2 RAN Slicing

The second contribution of this thesis attempted to address the problem of bringing virtualization and slicing capabilities to the RAN in a way that ensures the efficiency in the allocation of the resources and the full functional and performance isolation of the co-existing slices. Towards this direction, the Orion RAN slicing system was proposed, which enables the dynamic virtualization of base stations and the flexible customization of slices to meet their respective service needs. Orion is based on a `Hypervisor` component that is the key enabler of the RAN virtualization. Among others, the `Hypervisor` builds on set of abstractions that are specifically introduced for ensuring the isolated distribution of the radio resources to slices. It also leverages and extends the primitives of FlexRAN to allow the introduction of multiple virtual control planes, each corresponding to a different slice. To complement the basic design of Orion, an additional extension was also considered to cater the needs of OTT service providers. The evaluation results and the case studies that were performed using a concrete prototype implementation, revealed both the feasibility of the proposed approach

and the benefits that it can bring compared to the state-of-the-art.

### 7.1.3 Multi-tenancy in Indoor Small-cell Deployments with Shared Spectrum

The final contribution of this thesis built on the RAN slicing capabilities introduced by Orionand introduced a system called Iristhat was specifically designed for the emerging use case of multi-tenancy in indoor small-cell deployments with shared spectrum. The main focus of this part of the thesis was on designing a radio resource allocation mechanism that considers the idiosyncrasies of shared spectrum and allows tenants to differentiate their services without revealing private information about their business model, while at the same time ensuring that the neutral-host does not experience losses from the acquisition of the shared spectrum. For that, a dynamic pricing mechanism was proposed in the form of an MDPs and a deep reinforcement learning algorithm was employed to solve it. The dynamic pricing mechanism was embedded in a C-RAN design and the extensive evaluation results that were provided over a prototype testbed implementation demonstrated the feasibility and practicality of the proposed allocation mechanism in a realistic setting.

## 7.2 Limitations and Future Work

This section summarizes the limitations and future work opportunities in relation to the contributions made in this thesis.

### 7.2.1 FlexRAN: A Software-Defined Radio Access Network Platform

**State consistency –** Based on the current design of FlexRAN , it is possible for RAN applications that are deployed and concurrently running on top of the platform to modify the same state. Therefore, a conflict resolution mechanism ensuring state consistency becomes valuable when third-party network applications need to be supported. For example, such a mechanism should prohibit the deployment of multiple applications that may simultaneously issue scheduling decisions for the same resource blocks, effectively leading to conflicts.

**Control delegation –** Another significant issue of FlexRAN is related to the mechanism of control delegation. Currently the VSF code that is pushed to the agents by the controller needs to be compiled against the processor architecture of the target agent using the programming language of the FlexRAN agent API (currently in C). This cross-compilation process can be cumbersome and error-prone. Introducing a high-level domain-specific language that would make the development of VSFs technology-agnostic could greatly simplify the control delegation mechanism, especially in cases where the underlying infrastructure is heterogeneous in terms of the hardware employed for the agent implementation.

**Security concerns –** Another issue that needs to be further considered is related to the security concerns raised by the VSF updation mechanism. Depending on the environment in which FlexRAN operates, the VSF updation mechanism could be potentially exploited in cases when the development and deployment of third-party applications is allowed. Developing a sandbox environment with controlled permissions for the execution of the VSFs in line with the discussion in Section 4.3.3.1 is a very important topic for further research.

**Northbound API –** The current implementation of FlexRAN does not employ any high-level abstractions in the northbound API and instead reveals raw information to the applications through the RIB. This can make the development of control and management applications more challenging and time consuming. Specifying RAN abstractions and primitives that can simplify the development of applications is an interesting topic for future work.

**Scale and scope –** Improving the scalability of FlexRAN for wide area settings by introducing another layer of control and broadening its scope to go beyond the control and management of the radio resources in the cellular RAN by considering other domains like the core network and multi-RAT settings would provide a more holistic SDN solution for future mobile networks.

### 7.2.2 Orion: A RAN Slicing System

**RAN-as-a-Service –** The virtualization capabilities enabled by Orion can be seen as a concrete step towards realizing the so-called RAN-as-a-Service (RaaS) paradigm, where virtual RAN instances can be dynamically created over a cloud infrastructure. However, apart from virtualization, another significant aspect of the RaaS paradigm

is the capability to perform a dynamic functional split of the RAN in terms of control and data plane operations, enabling a more flexible composition of RAN instances. Extending the design of Orion to accommodate this need, while retaining its virtualization capabilites is a natural next step.

**Multi-RAT virtualization –** 5G networks are expected to span multiple RATs, including emerging technologies not just in the 3GPP space, like 5G NR and Narrowband-IoT, but also WiFi, which in the future is expected to become more similar to LTE (e.g. with 802.11ax). Moreover, the 5G core is expected to become access agnostic. In this setting, it is vital for RAN virtualization solutions to be able to accommodate multiple RATs. Orion is currently designed for single-RAT settings, however investigating the feasibility of extending its capabilities for multi-RAT is a significant topic for future work. This presents an additional outstanding challenge as it is unclear whether multiple RATs can be multiplexed over the same possibly specialized hardware or each needs its own dedicated hardware; the answer to this question might depend on the set of RATs under consideration.

**Slicing the UE –** The current design of Orion assumes a 1:1 mapping of UEs to slices with the additional capability of supporting OTT services through the extensions considered in Section 5.6. However, in some cases, multi-slice capabilities might still be desirable, like for example allowing a device to be connected to two different slices at the same time for work and personal use correspondingly. Perhaps the most important challenge when trying to introduce such multi-slice capabilities is related to the need to provide guarantees regarding the conflict-free operation of slices. For example, dealing with mobility and power management in such settings is not straightforward, as the roles and the responsibilities of the different slices are not clear. Introducing a mechanism to resolve such conflicts and regulate the operation of multi-slice environments is an interesting topic for future research.

### 7.2.3 Multi-Tenancy in a Neutral Host Environment

**Spectrum manager –** While the focus of this work was on the dynamic pricing mechanism provided by the spectrum manager of Iris, designing this entity in its entirety presents a number of additional challenges. First of all, the decisions regarding the acquisition and release of shared spectrum are not always obvious. Different cells in the indoor space might present different characteristics regarding their traffic loads. There-

fore, deciding on the proper amount of spectrum to obtain from a public repository is not trivial and needs to take into consideration the demands of *all* the small-cells of the deployment. Properly monitoring the small-cells and accurately forecasting their traffic patterns could help on making such decisions. It should be noted that this problem can become even more challenging when considering cases where the shared spectrum is accessible via a market (e.g. managed by the public repository) and demand dictates the cost of a block of spectrum. This dynamism needs to also be factored into any system that relies on such spectrum.

Another more technical challenge is related to the assignment of the shared spectrum to the small-cells. Depending on the frequency bands that are available, there might exist different constraints regarding the transmission power that a small-cell can use. This could have a direct effect in the coverage of the small-cells (and correspondingly their spectral efficiency). The spectrum manager should take into consideration such constraints when assigning the spectrum, as they could have a significant impact to the overall network performance.

**Strategic tenants –** The evaluation of this work assumes that tenants participating in the allocation mechanism provided by Iris do not act as price makers (e.g. strategically deciding not to buy resources in some epoch with the goal of reducing future prices) and instead their behavior is dictated by deterministic dis-utility functions. However, in scenarios where the tenants develop strategic behavior, we no longer have time-invariant transition rates from the point of view of any entity participating in the system (neutral-host or tenant), which can make the problem of solving the MDP harder. In order to improve the neutral-host's efficiency, learning rate and adjustability to the ever-changing network dynamics, we could view the dynamic pricing problem in the context of a multi-agent reinforcement learning framework like [86]. In this, both the neutral-host and the tenants could have their own strategic goals regarding the utilization of the radio resources and the adjustment of the price and could be using a semi-centralized learning process to reach their goal. Optimizing the dynamic pricing mechanism of Iris for such cases is left as future work.

**Multi-RAT settings –** The main focus of Iris was in enabling multi-tenancy in indoor small-cell environments with shared spectrum. The practicality of the system could be further enhanced by also considering unlicensed spectrum in the context of both 3GPP-based radio access technologies (e.g. LTE-U, MulteFire), but also technologies like WiFi, as this could greatly increase the capacity of the network. However, the

opportunistic nature of the spectrum introduces additional constraints in terms of its allocation to the tenants, since its availability can change at an even finer time granularity compared to the scenario of the shared spectrum (e.g. every few ms). Investigating the applicability of the proposed dynamic pricing mechanism in such cases and identifying ways to improve its performance is another interesting topic for future work.

# Appendix A

# Work and Publications

## A.1 Work related to FlexRAN

The FlexRAN SD-RAN platform that was presented in Chapter 4 was reported in the following publication:

1. **Xenofon Foukas, Navid Nikaein, Mohamed M. Kassem, Mahesh K. Marina, and Kimon 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 (CoNEXT)*, ACM, 2016.

A demo of the FlexRAN platform was also presented in a number of venues:

1. ACM MobiCom 2017

2. IEEE 5G Berlin Summit (held in conjunction with IEEE CSCN 2016)

3. All Things Cellular 2016 (held in conjunction with ACM MobiCom 2016)

The FlexRAN platform is publicly available as open source software. More information about the platform, the FlexRAN protocol specifications and demonstration information (video and live demos) can be found at:

- `http://networks.inf.ed.ac.uk/flexran/`

## A.2 Work related to Orion

The Orion RAN slicing system that was presented in Chapter 5 was reported in the following publication:

1. **Xenofon Foukas, Mahesh K. Marina, and Kimon Kontovasilis**. "Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture." In In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom)*, ACM, 2017.

A demo of Orion was also presented in a number of venues:

1. ACM MobiCom 2017 (**Best demo award**)
2. BT/Lime Microsystems Hackathon (**Finalist project**)
3. IEEE 5G Thessaloniki Summit 2017
4. Cambridge Wireless "The Time is Ripe for Innovations" event on 5G enabling technologies (Nov 2017)

## A.3 Other works

The main motivation and background of this thesis was also reported as part of the following publications:

1. **Xenofon Foukas, Georgios Patounas, Ahmed Elmokashfi, and Mahesh K. Marina**. "Network Slicing in 5G: Survey and Challenges." IEEE Communications Magazine 55, no. 5 (2017).
2. **Xenofon Foukas, Mahesh K. Marina, and Kimon Kontovasilis**. "Software Defined Networking Concepts." Software Defined Mobile Networks (SDMN): Beyond LTE Network Architecture (2015).

# Appendix B

# FlexRAN Protocol Specification

This appendix lists the FlexRAN protocol specification.

## B.1 FlexRAN header

All messages begin with a FlexRAN header:

```
flex_header {
    uint8_t version; /* The protocol version used */
    uint16_t type; /* The type of the message */
    uint32_t xid; /* Transaction id to pair req and reply */
};

enum flex_type {
    /* Discovery and maintenance messages */
    FLPT_HELLO,
    FLPT_ECHO_REQUEST,
    FLPT_ECHO_REPLY,
    /* Error reporting messages */
    FLTP_ERROR,
    /* Time indication messages*/
    FLPT_SF_TRIGGER,
    /* eNB configuration messages */
    FLPT_GET_ENB_CONFIG_REQUEST,
    FLPT_GET_ENB_CONFIG_REPLY,
    FLPT_GET_UE_CONFIG_REQUEST,
    FLPT_GET_UE_CONFIG_REPLY,
    FLPT_GET_LC_CONFIG_REQUEST,
    FLPT_GET_LC_CONFIG_REPLY,
    FLPT_SET_CELL_CONFIG,
```

```
    FLPT_SET_UE_CONFIG,
    /* Statistics and measurement messages */
    FLPT_STATS_REQUEST,
    FLPT_STATS_REPLY,
    /* Controller command messages */
    FLPT_DL_MAC_CONFIG,
    FLPT_UL_MAC_CONFIG,
    /* Asynchronous messages */
    FLPT_UL_SR_INFO,
    FLPT_DL_RACH_INFO,
    FLPT_UE_STATE_CHANGE,
    /* Control delegation messages */
    FLPT_CONTROL_LOAD_REQUEST,
    FLPT_CONTROL_LOAD_REPLY,
    FLPT_DELEGATE_CONTROL,
    FLPT_RECONFIGURE_AGENT
};
```

## B.2 Common Structures

### Cell configuration related structures and enums

```
enum flex_hopping_mode {
    FLHM_INTER = 0,
    FLHM_INTERINTRA,
};

enum flex_phich_resource {
    FLPR_ONE_SIXTH = 0,
    FLPR_HALF,
    FLPR_ONE,
    FLPR_TWO,
};

enum flex_phich_duration {
    FLPD_NORMAL = 0,
    FLPD_EXTENDED,
};

struct flex_si_config {
    uint16_t sfn;  /* Frame number to apply the SI config */
```

```
    uint16_t sib1_length;  /* The length of SIB1 in bytes */
    uint8_t si_window_length; /* SI Scheduling window in SF */
    uint8_t nrSI;  /* The number of SI messages */
    struct prp_si_message si_message[nrSI];  /* List of SI messages
       to send. The index identifies the type of message:  0 - SIB1,
        1..31 - Six, 32..63 - PCCH */
};

struct flex_si_message {
    uint16_t periodicity;  /* Periodicity of SI msg in frames */
    uint16_t length;  /* The length of the SI message in bytes */
};

enum flex_ul_cyclic_prefix_length {
    FLUCPL_NORMAL = 0,
    FLUCPL_EXTENDED,
};

enum flex_dl_cyclic_prefix_length {
    FLDCPL_NORMAL = 0,
    FLDCPL_EXTENDED,
};

enum flex_duplex_mode {
    FLDM_TDD = 0,
    FLDM_FDD,
};

enum flex_qam {
    FLEQ_MOD_16QAM = 0,
    FLEQ_MOD_64QAM,
};
```

This structure contains all information regarding the current configuration of a cell

```
struct flex_cell_config {
    uint32_t phy_cell_id;  /* The PCI of this cell */
    uint32_t cell_id;  /* The PLMN cell id of this cell */
    uint8_t pusch_hopping_offset;  /*PUSCH resources in RBs for
       hopping */
    uint16_t hopping_mode;  /* One of the FLHM_* */
    uint8_t n_sb;  /* The number of subbands */
    uint16_t phich_resource;  /* The number of resource element
```

```
        groups used for PHICH. One of FLPR_* */
    uint8_t phich_duration;  /* One of the FLPD_* */
    uint8_t init_nr_PDCCH_OFDM_sym;  /*See TS 36.211, section 6.9*/
    struct prp_si_config si_config;  /* The SI configuration */
    uint8_t ul_bandwidth;  /* The UL transmission bandwidth RBs */
    uint8_t dl_bandwidth;  /* The DL transmission bandwidth RBs */
    uint16_t ul_cyclic_prefix_length;  /* One of the FLUCPL_* */
    unit16_t dl_cyclic_prefix_length;  /* One of the FLDCPL_* */
    uint8_t antenna_ports_count;  /* Number of cell antenna ports */
    uint8_t duplex_mode;  /* One of the FLDM_* */
    uint8_t subframe_assignment;  /*TDD DL/UL subframe assignment*/
    uint8_t special_subframe_patterns;  /* TDD only. See TS 36.211,
        table 4.2.1 */
    uint8_t mbsfn_set;  /* Boolean value */
    uint8_t mbsfn_subframe_config_rfperiod[5];  /* The MBSFN radio
        frame period. Optional */
    uint8_t mbsfn_subframe_config_rfoffset[5];  /* The radio frame
        offset. Optional */
    uint8_t mbsfn_subframe_config_sfalloc[5];  /*Bitmap indicating
        the MBSFN subframes. Optional */
    uint8_t prach_config_index;  /* See TS 36.211, section 5.7.1 */
    uint8_t prach_freq_offset;  /* See TS 36.211, section 5.7.1 */
    uint8_t ra_response_window_size;  /* Duration of RA response
        window in SF */
    uint8_t mac_contention_resolution_timer;  /*Timer used for RA*/
    uint8_t max_HARQ_Msg3Tx;  /* See TS 36.321 */
    uint8_t n1PUCCH_AN;  /* See TS 36.213, section 10.1 */
    uint8_t deltaPUCCH_shift;  /* See TS 36.211, section 5.4 */
    uint nRB_cqi;  /* See TS 36.211, section 5.4 */
    uint8_t srs_subframe_config;  /* See TS 36.211, table
        5.5.3.3-1,2 */
    uint8_t srs_bw_config;  /* See TS 36.211, section 5.5.3.2 */
    uint8_t srs_mac_up_pts;  /* Boolean value. See TS 36.211,
        section 5.5.3.2. TDD only */
    uint8_t enable_64QAM;  /* One of the FLEQ_* */
    uint8_t carrier_index;  /* Carrier component index */
};
```

## UE configuration related structures and enums

```
struct flex_drx_config {
        /* See TS 36.321 */
```

```
    uint8_t on_duration_timer;  /* Timer in SF */
    uint16_t drx_inactivity_timer;  /* Timer in SF */
    uint8_t drx_retransmission_timer; /* Timer in SF */
    uint16_t long_DRX_cycle; /*Long DRX cycle in SF */
    uint16_t long_DRX_cycle_start_offset;  /*Long DRX cycle offset*/
    uint16_t short_DRX_cycle;  /* Short DRX cycle in SF */
    uint8_t drx_short_cycle_timer; /* Timer in subframes */
};

enum flex_meas_gap_config_pattern {
    FLMGCP_GP1 = 0,
    FLMGCP_GP2,
    FLMGCP_OFF,
};

struct flex_sps_config {
    uint16_t semi_persistent_sched_interval_UL;  /* SPS UL
       scheduling interval in SF */
    uint16_t semi_persistent_sched_interval_DL;  /* SPS DL
       scheduling interval in SF */
    uint8_t num_of_conf_SPS_proc;  /* Number of SPS HARQ processes,
       see TS 36.321 */
    uint8_t n1_PUCCH_AN_persistent_list_size;  /* The size of the
       list. Ignored when prp_sps_config is used */
    uint16_t n1_PUCCH_AN_element[4];  /* See TS 36.213. Ignored when
        prp_sps_config is used as part of PRPT_SET_UE_CONFIG */
    uint8_t implicit_release_after;  /* number of empty
       transmissions before release of SPS */
};

enum flex_setup_release_action {
    FLSRA_SETUP = 0,
    FLSRA_RELEASE,
};

struct flex_sr_config {
    uint8_t sr_action;  /* Indicates if SR config should be changed
       or released. One of the FLSRA_* values */
    uint8_t sched_interval;  /* SR scheduling interval in SF */
    uint8_t dsr_trans_max;  /* See TS 36.213 */
};
```

```
struct flex_cqi_config {
    uint8_t cqi_action;  /* Indicates if CQI config should be
       changed or released. One of the FLSRA_* values */
    uint8_t cqi_sched_interval;  /* CQI scheduling interval in SF */
    uint8_t ri_sched_interval;  /* RI scheduling interval in SF */
};

struct flex_ue_capabilities {
    uint8_t half_duplex;  /* Only half-duplex support. FDD operation
       . Boolean value */
    uint8_t intra_SF_hopping;  /* Support of intra-subframe hopping.
        Boolean value */
    uint8_t type2_sb_1;  /* UE support for type 2 hopping with n_sb
       >1 */
    uint16_t ue_category;  /* The UE category */
    uint8_t res_alloc_type1;  /* UE support for resource allocation
       type 1 */
};

enum flex_ue_transmission_antenna {
    FLUTA_NONE = 0,
    FLUTA_OPEN_LOOP,
    FLUTA_CLOSED_LOOP,
};

enum flex_aperiodic_cqi_report_mode {
    FLACRM_RM12 = 0,
    FLACRM_RM20,
    FLACRM_RM22,
    FLACRM_RM30,
    FLACRM_RM31,
    FLACRM_NONE,
};

enum flex_tdd_ack_nack_feedback_mode {
    FLTANFM_BUNDLING = 0,
    FLTANFM_MULTIPLEXING,
};

struct flex_scell_config {
    uint8_t carrier_index;  /*Identifier of the carrier component*/
    uint8_t scell_index;  /* Index of this SCell (RRC SCellIndex) */
```

```
    uint8_t use_ccs;  /* Indicates if cross-carrier scheduling is
        used by this SCell. Boolean value */
    uint8_t sched_cell_index;  /* Index of the cell responsible for
        scheduling this SCell */
    uint8_t pdsch_start;  /* Starting OFDM symbol of PDSCH data
        region for this SCell */
};

struct flex_ue_config {
    uint16_t rnti;  /* The RNTI of the UE */
    uint8_t drx_config_present;  /* Boolean. Indicates if the drx
        structure is valid */
    struct flex_drx_config drx_config;  /* The DRX configuration */
    uint16_t time_alignment_timer;  /* Timer in SF. Controls the
        synchronization status of the UE, not the timing advance
        procedure. See TS 36.321 */
    uint8_t meas_gap_config_pattern;  /* Measurement gap
        configuration. One of FLMGCP_*. See TS 36.133 */
    uint8_t meas_gap_config_sf_offset;  /* Measurement gap offset,
        if applicable */
    uint8_t sps_config_present;  /* Boolean value. Indicates if the
        SPS configuration is valid */
    struct flex_sps_config sps_config;  /* The SPS configuration */
    uint8_t sr_config_present;  /* Boolean value. Indicates if the
        SR configuration is valid */
    struct flex_sr_config sr_config;  /* The SR configuration */
    uint8_t cqi_config_present;  /* Boolean value. Indicates if the
        CQI configuration is valid*/
    struct flex_cqi_config;  /* The CQI configuration */
    uint8_t transmission_mode;  /* The UE transmission mode. See TS
        36.213 */
    uint64_t ue_aggregated_max_bitrate_UL;  /* Aggregated bit-rate
        of non-gbr bearer per UE. See TS 36.413 */
    uint64_t ue_aggregated_max_bitrate_DL;  /* Aggregated bit-rate
        of non-gbr bearer per UE. See TS 36.413 */
    struct flex_ue_capabilities capabilities;  /* The UE
        capabilities */
    uint8_t ue_transmission_antenna;  /* One of FLUTA_* values */
    uint8_t tti_bundling;  /* Boolean value. See TS 36.321 */
    uint8_t max_HARQ_Tx;  /* The max HARQ retransmission for UL. See
         TS 36.321 */
    uint8_t beta_offset_ACK_index;  /* See TS 36.213 */
```

```
    uint8_t beta_offset_RI_index;  /* See TS 36.213 */
    uint8_t beta_offset_CQI_index;  /* See TS 36.213 */
    uint8_t ack_nack_simultaneous_trans;  /* Boolean. See TS 36.213,
        Section 8.2 */
    uint8_t simultaneous_ack_nack_cqi;  /* Boolean. See TS36.213,
       Section 10.1 */
    uint8_t aperiodic_cqi_rep_mode;  /*One of the FLACRM_* values*/
    uint8_t tdd_ack_nack_feedback;  /*One of the FLTANFM_* values*/
    uint8_t ack_nack_repetition_factor;  /* See TS36.213, section
       10.1 */
    uint8_t extended_bsr_size;  /* Boolean. Extended BSR size */
    uint8_t ca_support;  /* Boolean. Support for CA */
    uint8_t cross_carrier_sched_support;  /* Boolean */
    uint8_t pcell_carrier_index;  /* Index of primary cell */
    uint8_t nr_scells;  /* Number of secondary cells */
    struct flex_scell_config scell_config[nr_scells];  /* Secondary
       cells configuration */
    uint8_t scell_deactivation_timer;  /* Deactivation timer for
       secondary cell */
};
```

## Logical channels configuration related structures and enums

```
enum flex_lc_direction {
    FLLCD_UL = 0,
    FLLCD_DL,
    FLLCD_BOTH,
};

enum flex_qos_bearer_type {
    FLQBT_NON_GBR = 0,
    FLQBT_GBR,
};

struct flex_lc_config_element {
    uint8_t lcid;  /* The logical channel id */
    uint8_t lcg;  /* The LC group (0..3) the LC is mapped to. 4
       means no LCG association */
    uint8_t direction;  /* The LC direction. One of the FLLCD_*
       values */
    uint8_t qos_bearer_type;  /* GBR or NGBR bearer. One of the
       FLQBT_* values */
```

```
    uint8_t qci;  /* The QCI defined in TS 23.203, coded as defined
       in TS 36.413 (one less than the actual QCI value) */
    uint64_t e_RAB_max_bitrate_UL;  /* In bps. GBR only */
    uint64_t e_RAB_max_bitrate_DL;  /* In bps. GBR only */
    uint64_t e_RAB_guaranteed_bitrate_UL;  /* In bps. GBR only */
    uint64_t e_RAB_guaranteed_bitrate_DL;  /* In bps. GBR only */
};

struct flex_lc_ue_config {
    uint16_t rnti;  /* The RNTI identifying the UE */
    uint8_t nr_lc;  /* The number of logical channels */
    struct flex_lc_config[rn_lc];  /* The list of LC configurations
       for the UE */
};
```

## Statistics related structures and enums

### Types of statistics related enums

```
/* Types of statistics requested by the controller */
enum flex_stats_type {
    FLST_COMPLETE_STATS = 0,
    FLST_CELL_STATS,
    FLST_UE_STATS,
};

/* Report frequency for the requested statistics */
enum flex_stats_report_freq {
    FLSRF_ONCE = 0,
    FLSRF_PERIODICAL,
    FLSRF_CONTINUOUS,  /* Report any occurring change of stats */
    FLSRF_OFF,  /* Stops periodic reports for defined types */
};

/* Flags for cell statistics */
enum flex_cell_stats_type {
    FLCST_NOISE_INTERFERENCE = 1 << 0,  /* Noise and interference */
};

/* Flags for UE-related statistics */
enum flex_ue_stats_type {
    FLUST_BSR = 1 << 0,  /* Buffer status report */
    FLUST_PRH = 1 << 1,  /* Power headroom */
```

```
    FLUST_RLC_BS = 1 << 2,  /* Buffer status of RLC LCs */
    FLUST_MAC_CE_BS = 1 << 3,  /* Buffer status for MAC CE */
    FLUST_DL_CQI = 1 << 4,  /* DL CQI report */
    FLUST_PBS = 1 << 5,  /* Paging buffer status. Valid only when
       rnti = p-rnti */
    FLUST_UL_CQI = 1 << 6,  /* UL CQI report */
};
```

### UE related statistics

```
/* RLC buffer status for a specific logical channel of a UE */
struct flex_rlc_bsr {
    uint8_t lc_id;  /* The id of the LC */
    uint32_t tx_queue_size;  /* Transmission queue size (bytes) */
    uint16_t tx_queue_hol_delay;  /*Tx Head of line delay in ms*/
    uint32_t retransmission_queue_size;  /* The size of the re-tx
       queue in bytes */
    uint16_t retransmission_queue_hol_delay;  /* Head of line delay
       of re-tx in ms */
    uint16_t status_pdu_size;  /* The current size of the pending
       STATUS message in bytes */
};

/* Flags for MAC CEs */
enum flex_ce_type {
    FLPCET_TA = 1 << 0,  /* Timing Advance */
    FLPCET_DRX = 1 << 1,  /* DRX control */
    FLPCET_CR = 1 << 2,  /* Contention Resolution */
    FLPCET_CA = 1 << 3,  /* CA activation/deactivation */
};

/* Type of DL CSI report */
enum flex_csi_type {
    FLCSIT_P10 = 0,
    FLCSIT_P11,
    FLCSIT_P20,
    FLCSIT_P21,
    FLCSIT_A12,
    FLCSIT_A22,
    FLCSIT_A20,
    FLCSIT_A30,
    FLCSIT_A31,
};
```

```
/* CSI type P10 */
struct flex_csi_p10 {
    uint8_t wb_cqi;  /* The reported wideband CQI value */
};

/* CSI type P11 */
struct flex_csi_p11 {
    uint8_t wb_cqi[2];  /* Wideband CQI value per codeword. */
    uint8_t wb_pmi;  /* The reported matrix index. */
};


/* CSI type P20 */
struct flex_csi_p20 {
    uint8_t wb_cqi;  /*The wideband CQI value per codeword*/
    uint8_t sb_cqi;  /* The CQI in the current subband */
    uint8_t bandwidth_part_index;  /* The bandwidth part for which
       CQI is reported */
    uint8_t sb_index;  /* The preferred subband in the current
       bandwidth part. */
};

/* CSI type P21 */
struct flex_csi_p21 {
    uint8_t wb_cqi[2];  /* Wideband CQI value per codeword. */
    uint8_t wb_pmi;  /*Wideband precoding matrix index.*/
    uint8_t sb_cqi[2];  /* CQI for up to two codewords for the
       preferred subband */
    uint8_t bandwidth_part_index;  /* The bandwidth part for which
       CQI is reported */
    uint8_t sb_index;  /* The preferred subband in the current
       bandwidth part. */
};

/* CSI type A12 */
struct flex_csi_a12 {
    uint8_t wb_cqi[2];  /* Wideband CQI value per codeword. */
    uint8_t sb_pmi[13];  /* The pmi conditioned on the current
       subband for aperiodic  higher-layer selected subbands. */
};
```

```
/* CSI type A22 */
struct flex_csi_a22 {
    uint8_t wb_cqi[2];  /* Wideband CQI value per codeword. */
    uint8_t sb_cqi[2];  /* The CQI for up to two codewords for the
       preferred subband */
    uint8_t wb_pmi;  /* Wideband precoding matrix index. */
    uint8_t sb_pmi;  /* The PMI conditioned on current subband */
    uint8_t sb_list[6];  /*Preferred-M subbands reported by the UE*/
};

/* CSI type A20 */
struct flex_csi_a20 {
    uint8_t wb_cqi;  /* Wideband CQI value per codeword */
    uint8_t sb_cqi;  /* The CQI in the current subband */
    uint8_t sb_list[6];  /*Preferred-M subbands reported by the UE*/
};

/* CSI type A30 */
struct flex_csi_a30 {
    uint8_t wb_cqi;  /* Wideband CQI value per codeword */
    uint8_t sb_cqi[13];  /*CQI conditioned on preferred-M subbands*/
};

/* CSI type A31 */
struct flex_csi_a31 {
    uint8_t wb_cqi[2];  /* Wideband CQI value per codeword. */
    uint8_t sb_cqi[13][2];  /* The CQI for up to two codewords
       conditioned on the  preferred -M subbands. */
    uint8_t wb_pmi;  /* Wideband precoding matrix index. */
};

/* The CSI report of the UE for a specific servCellIndex */
struct flex_dl_csi {
    uint8_t serv_cell_index;  /*Definition according to TS 36.331*/
    uint8_t ri;  /* The last received rank indicator */
    uint16_t type;  /* Type of CSI report. A FLCSIT_* value */
    uint8_t report[0];  /* CSI report based on the type */
};

/* The full DL CQI report for all CC of a UE */
struct flex_dl_cqi_report {
    uint16_t sfn_sn;  /* The SFN/SF in which the CQI was received */
```

```
    uint16_t nr_reports;  /* Number of reports for the given UE */
    struct flex_dl_csi csi_report[nr_reports];  /* UE CSI report */
};

/* Paging message info */
struct flex_paging_info {
    uint8_t paging_index;  /* An index used to identify the
       scheduled message. Must be used for scheduling decision */
    uint16_t paging_message_size;  /* The size of paging message */
    uint8_t paging_subframe;  /* The subframe in which the message
       needs to be scheduled */
    uint8_t carrier_index;  /* The carrier where the message should
       be scheduled */
};

/* Report for the paging buffer status */
struct flex_paging_buffer_report {
    uint8_t nr_paging_info;  /* The number of pending paging
       messages. Valid only if RNTI is a P-RNTI */
    struct flex_paging_info paging_list[nr_paging_info];  /* A list
       of pending paging messages. Valid only if RNTI is a P-RNTI */
};

/* The type of UL CQI */
enum flex_ul_cqi_type {
    FLUCT_SRS = 0,
    FLUCT_PUSCH,
    FLUCT_PUCCH_1,
    FLUCT_PUCCH_2,
    FLUCT_PRACH,
};

/* UL CQI report for a specific UE for a given report type */
struct flex_ul_cqi {
    uint8_t type;  /* Type of report. One of the FLUCT_* values */
    uint16_t sinr[100];  /* The SINR measurements in dB based on the
        resource given in type. For PUCCH only first index used. For
        PRACH first 6 indices used. For PUSCH and SRS each index is
       1 RB. */
    uint8_t serv_cell_index;  /* Definition in TS 36.331 */
};
```

```
/* Full UL CQI report for a specific UE */
struct flex_ul_cqi_report {
    uint16_t sfn_sn;  /* The SFN/SF in which the CQI information was
        received */
    uint16_t nr_reports;  /* Number of reports for the given UE */
    struct flex_ul_cqi cqi_meas_list[nr_reports];  /* A list of
       reports for the given UE */
};

/* Statistics report for a specific UE */
struct flex_ue_stats_report {
    uint16_t rnti;  /* The RNTI of the UE. Could also be a P-RNTI (
       all other flags disabled) */
    uint32_t flags;  /* A bitmap of FLUST_* values for the elements
       of this report */
    uint8_t bsr[4];  /* BSR for the 4 LCGs (values 0..63). Value 64
       no update of buffer status. */
    uint8_t phr;  /* Power headroom. Valid only if flag is set */
    uint16_t nr_lc;  /* Number of LCs for buffer status report.
       Valid only if flag is set */
    struct flex_rlc_bsr rlc_reports[nr_lc];  /* Buffer status for
       each LC in the RLC. Valid only if flag is set */
    uint8_t pedning_mac_ces;  /* A bitmap of FLCET_* flags showing
       the pending CEs for MAC transmission. Valid only if proper
       flag is set */
    struct flex_dl_cqi_report dl_cqi_report;  /* DL CQI report.
       Valid only if proper flag is set */
    struct flex_paging_buffer_report pbr;  /* Status of paging
       buffer report. Valid only if rnti = p-rnti and flag is set */
    struct flex_ul_cqi_report ul_cqi_report;  /* UL CQI report.
       Valid if proper flag is set */
};
```

### Cell related statistics

```
/* The UL noise and interference report for a UE */
struct flex_noise_interference_report {
    uint16_t sfn_sf;  /* The SFN and SF in which the info was
       received. Bit 0-3 SF and 4-13 SFN */
    uint16_t rip;  /* Received interference in dBm. See TS 36.214 */
    uint16_t tnp;  /* Thermal noise power in dBm. See TS 36.214 */
};
```

```
/* The full statistics report for a specific cell */
struct flex_cell_stats_report {
        uint16_t carrier_index;                 /* The index of the
            CC */
        uint32_t flags;                                 /* A bitmap
            of FLCST_* values for the elements of this report */
        struct flex_noise_interference_report noise_inter_report;
                /* A report on the noise and interference of the
            cell. Optional, only if flag is set */
};
```

## Controller commands related structures and enums

### DL MAC config structures and enums

```
enum flex_dci_format {
    FLDCIF_1 = 0,
    FLDCIF_1A,
    FLDCIF_1B,
    FLDCIF_1C,
    FLDCIF_1D,
    FLDCIF_2,
    FLDCIF_2A,
    FLDCIF_2B,
    FLDCIF_3,
    FLDCIF_3A
};

enum flex_vrb_format {
    FLVRBF_LOCALIZED = 0,
    FLVRBF_DISTRIBUTED,
};

enum flex_ngap_val {
    FLNGV_1 = 0,
    FLNGV_2,
};

/*UE DCI info for the DL */
struct flex_dl_dci {
    uint16_t rnti;  /* The RNTI identifying the UE */
    uint8_t res_alloc;  /* Type of resource allocation */
    uint32_t rb_bitmap;  /* Bitmap for RBs allocation to the UE */
```

```
    uint8_t rb_shift;  /* See TS 36.213 section 7.1.6.2 */
    uint8_t nr_of_tbs;  /* The number of transport blocks */
    uint16_t tbs_size[2];  /* The size of TBs in size */
    uint8_t mcs[2]; /*MCS of each TB. See TS 36.213 section 7.1.7*/
    uint8_t ndi[2];  /* New data indicator in each TB. Boolean */
    uint8_t rv[2];  /* Redundancy version in each TB */
    uint8_t cce_index;  /* CCE index used to send the DCI */
    uint8_t aggr_level;  /* The aggregation level */
    uint8_t precoding_info;  /* 2 antenna ports:0..6, 4 antenna
       ports:0..50 */
    uint8_t format;  /* DCI format. One of the FLDCIF_* values */
    int8_t tpc;  /* See TS 36.213, sec 5.1.1.1 */
    uint8_t harq_process;  /* The number of the HARQ process */
    uint8_t dai;  /* Only for TDD */
    uint8_t vrb_format; /*TS 36.213, sec 7.1.6.3. A FLVRBF_* value*/
    uint8_t tb_swap;  /* Boolean value. TB to codeword swap flag.
       See TS 36.21, section 5.3.3.1.5 */
    uint8_t sps_release;  /* SPS release. Boolean value */
    uint8_t pdcch_order;  /* Boolean value. Indicates if PDCCH is
       for PDCCH order */
    uint8_t preamble_index;  /* Preamble index. Only valid if
       pdcch_order=1 */
    uint8_t prach_mask_index;  /* PRACH mask index. Valid if
       pdcch_order = 1 */
    uint8_t n_gap;  /* The value of N_GAP. One of FLNGV_* values */
    uint8_t tbs_idx;  /* The TBS index for Format 1A */
    uint8_t dl_power_offset; /*Format 1D. See TS 36.213, sec 7.1.5*/
    int8_t pdcch_power_offset;  /* DL PDCCH power boosting in dB */
    uint8_t cif_present;  /* Boolean value. CIF field indication */
    uint8_t cif;  /* CIF field for cross carrier scheduling */
};

/* Parameters for RLC PDU creation */
struct flex_rlc_pdu {
    uint8_t logical_channel_id;  /* The logical channel ID */
    uint16_t size;  /* Maximum length of the RLC PDU in bytes */
};

/* UE DL conf information */
struct flex_dl_data {
    uint16_t rnti;  /* The RNTI of the UE */
    struct flex_dl_dci dl_dci; /*DL DCI configured for the UE. Could
```

```
        also indicate PDCCH order or SPS release of format 3/3A*/
    uint8_t ce_bitmap[2];  /* The CEs scheduled for transmission in
       this TB. A bitmap composed of FLPCET_* flags */
    uint8_t nr_rlc_pdu;  /* Number of RLC PDUs to be built */
    struct flex_rlc_pdu rlc_pdu_list[nr_lrc_pdu][2];  /* A list of
       parameters for the creation of RLC PDUs per TB */
    uint8_t serv_cell_index;  /*Definition according to TS 36.331*/
    uint8_t act_deact_ce;  /* Hex content of MAC CE for Activation/
       Deactivation in CA */
};

/* RAR configuration */
struct flex_dl_rar {
    uint16_t rnti;  /* The C-RNTI identifying the UE */
    uint32_t grant;  /* 20-bit UL grant. See TS 36.213, sec 6.2 */
    struct flex_dl_dci rar_dci;  /* DL DCI configured for the RAR */
    uint8_t carrier_index;  /* The carrier index for the RAR */
};

enum flex_broadcast_type {
    FLBT_BCCH = 0,
    FLBT_PCCH,
};

/* paging/broadcast configuration */
struct flex_dl_broadcast {
    uint8_t type;  /* Broadcast message type. A FLBT_* value */
    uint8_t index;  /* Broadcast message index. 0 - SIB1, 1..31 -
       Six, 32..63 - PCCH (obtained from flex_paging_info) */
    struct flex_dl_dci broad_dci; /*DL DCI for BCCH/PCCH*/
    uint8_t carrier_index;  /* Index of the carrier for broadcast */
};

struct flex_pdcch_ofdm_sym_count {
    uint8_t carrier_index;  /* Unique carrier identifier */
    uint8_t num_pdcch_ofdm_symbols;  /*PDCCH size in OFDM symbols*/
};
```

### UL MAC config structures and enums

```
struct flex_ul_dci {
    uint16_t rnti;  /* The RNTI identifying the UE */
    uint8_t rb_start; /*The first RB allocated to UE. See TS 36.213,
```

```
        sec 8.1*/
    uint8_t rb_len;  /*RBs allocated to UE. See TS 36.213, sec 8.1*/
    uint16_t tb_size;  /* Size of the transport block in bytes */
    uint8_t mcs;  /* The MCS of each TB */
    uint8_t ndi;  /* New data indicator field (0-1) */
    uint8_t cce_index;  /* CCE index used to send the DCI */
    uint8_t aggr_level;  /* The aggregation level */
    uint8_t ue_tx_antenna_selection;  /* See TS 36.212, section
       5.3.3.2. Values 0,1,3. 3 means off */
    uint8_t hopping;  /* Hopping enabled flag. Boolean */
    uint8_t n_2_dmrs;  /* Cyclic shift (0..7) */
    int8_t tpc; /*Tx power control command. TS 36.213, sec 5.1.1.1*/
    uint8_t cqi_request;  /* Aperiodic CQI req flag. Boolean */
    uint8_t ul_index;  /* UL index. TDD only */
    uint8_t dai;  /* DL assignment index. TDD only */
    uint8_t freq_hopping;  /* Frequency hopping bits. See TS 36.213,
        sec 8.4 */
    int8_t pdcch_power_offset;  /* PDCCH power boosting in dB */
    uint8_t cif_present;  /* Boolean value. CIF field indication */
    uint8_t cif;  /* CIF field for cross carrier scheduling */
    uint8_t serv_cell_index;  /*Definition according to TS 36.331*/
};

struct flex_ul_phich {
    uint16_t rnti;  /* The RNTI identifying the UE */
    uint8_t phich;  /*ACK(0) or NACK(1) passed to UE in the PHICH*/
    uint8_t serv_cell_index;  /*Definition according to TS 36.331*/
};
```

## Asynchronous messages related structures and enums

### RACH reception structures and enums

```
struct flex_rach {
    uint16_t rnti;  /*Allocated t-crnti based on the RACH preamble*/
    uint16_t estimated_size;  /* The estimated minimum size of the
       UL message in bits, based on the RACH preamble */
    uint8_t carrier_index; /*Carrier that RACH req was received*/
};
```

### UE state change structures and enums

```
enum flex_ue_state_change_type {
```

```
    FLUESC_UPDATED = 0,
    FLUESC_ACTIVATED,
    FLUESC_DEACTIVATED,
    FLUESC_MOVED,
};
```

### Time indication related structures and enums

```
enum flex_harq_status {
    FLHS_ACK = 0,
    FLHS_NACK,
    FLHS_DTX,
};

struct flex_dl_info {
        uint16_t rnti;  /* The rnti identifying the UE */
        uint8_t harq_process_id;  /* The id of the HARQ process */
    uint8_t nr_harq_status;  /*Size of the HARQ status list (TBs) */
    uint8_t harq_status[2];     /* HARQ status for the above process
       . One of FLHS_* values */
    uint8_t serv_cell_index;  /* definition according to 36.331 */
};

enum flex_reception_status {
    FLRS_OK = 0,
    FLRS_NOT_OK,
    FLRS_NOT_VALID,
};

struct flex_ul_info {
    uint16_t rnti; /* The rnti identifying the UE */
    uint16_t ul_reception[11];  /* The amount of data in bytes in
       the MAC SDU received in this subframe for the given logical
       channel. */
    uint8_t reception_status;  /* One of the FLRS_* status values */
    int8_t tpc;  /* Tx power control. See TS 36.213, sec 5.1.1.1 */
    uint8_t serv_cell_index;  /* Definition according to 36.331 */
};
```

## B.3 Discovery and maintenance messages

### Hello message

The hello message contains no body, i.e. it is composed only by the header of the message with the type set to `FLPT_HELLO`. Future versions of the protocol might also support a Hello message body for additional functionality.

```
struct flex_hello {
    struct flex_header header;
};
```

### Echo request message

The echo request message consists of the message header and a variable length data field. This datafield could be a zero-length field in case the message is used to check for liveness, a timestamp when used for latency or a field of various lengths when used to measure bandwidth.

```
struct flex_echo_request {
    struct flex_header header;
    uint8_t data[0];  /* A variable-length data field */
};
```

### Echo reply message

The echo reply message consists of the message header and the unmodified data field that was used in the corresponding request message.

```
struct flex_echo_reply {
    struct flex_header header;
    uint8_t data[0];  /* The unmodified variable-length data field
        of the req*/
};
```

## B.4 Error reporting messages

### Error message

The error message is used to notify the controller for any problems that might have occurred to the eNB. Each error message has a type, a code and a data field. The type

field describes the high-level type of the error, while the code describes the specific error based on the type. The data is a variable-length field that is defined based on the specific type and code of a message and contains in most cases the message that caused the error.

```
struct flex_error_msg {
    struct flex_header header;
    uint16_t type;  /* The high-level type of the error */
    uint16_t code;  /* The code for the specific error type */
    uint8_t data[0]; /*Variable-length data based on type and code*/
};
```

## B.5 eNB configuration messages

### eNB configuration request message

The eNB configuration request message contains no body, i.e. it is composed only of the message header with the type set to `FLPT_GET_ENB_CONFIG_REQUEST`. The controller expects to receive a message of type `FLPT_GET_ENB_CONFIG_REPLY` as a reply to this message.

### UE configuration request message

The UE configuration request message contains no body, i.e. it is composed only of the message header with the type set to `FLPT_GET_UE_CONFIG_REQUEST`. The controller expects to receive a message of type `FLPT_GET_UE_CONFIG_REPLY` as a reply to this message.

### Logical channels configuration request message

The logical channels configuration request message contains no body, i.e. it is composed only of the message header with the type set to `FLPT_GET_LC_CONFIG_REQUEST`. The controller expects to receive a message of type `FLPT_GET_LC_CONFIG_REPLY` as a reply to this message.

### eNB configuration reply message

The eNB configuration reply message consists of the message header, the eNB ID and a list of cell configurations for all the cells supported by the eNB.

```
struct flex_enb_config_reply {
    struct flex_header header;
    uint32_t eNB_id;  /* The id of the eNB */
    uint8_t nr_cells;  /* Number of cells (carriers) */
    struct flex_cell_config cell_config[nr_cells];  /* Config of
        each cell. */
};
```

### UE configuration reply message

The UE configuration reply message `FLPT_GET_UE_CONFIG_REPLY` consists of the message header and a list of the UE configurations of all the UEs currently associated with the eNB.

```
struct flex_ue_config_reply {
    struct flex_header header;
    uint16_t nr_ue;  /* The number of UEs in the list */
    struct flex_ue_config ue_config[nr_ue];  /* A list of UE
        configurations */
};
```

### Logical channels configuration reply message

The logical channels configuration reply message `FLPT_GET_LC_CONFIG_REPLY` consists of the message header and a list of logical channel configurations for all the associated UEs

```
struct flex_lc_config_reply {
    struct flex_header header;
    uint16_t nr_ue;  /* The number of UEs in the list */
    struct flex_lc_ue_config lc_ue_config[nr_ue];  /* A list of
        logical channel configs for the connected UEs */
};
```

### Cell configuration set message

The cell configuration set message FLPT_SET_CELL_CONFIG consists of the message header and a list of cell configurations for the eNB.

```
struct flex_cell_config_update {
    struct flex_header header;
    uint16_t nr_cells;  /*      The number of cells with updated
        configurations */
    struct flex_cell_config cell_configs[nr_cells];  /* The cell
        configuration updates */
};
```

### UE configuration set message

The UE configuration set message FLPT_SET_UE_CONFIG consists of the message header and a list of UE configurations for updates.

```
struct flex_ue_config_update {
    struct flex_header header;
    uint16_t nr_ue;  /* The number of UEs with updated
        configurations */
    struct flex_ue_config ue_configs[nr_ue];  /* The UE
        configuration updates */
};
```

## B.6 Statistics and measurement report messages

### Statistics request message

The statistics request message FLPT_STATS_REQUEST consists of the message header, the type of statistics requested and a body based on the request type.

```
struct flex_stats_request {
    struct flex_header header;
    uint8_t type;  /* One of the FLST_* values */
    uint8_t body[0];    /* The body of the request. Different based
        on the type. 5.6.1.1 - 5.6.1.3 structs */
};
```

#### Complete stats request

```
struct flex_complete_stats_request {
    uint8_t report_frequency;  /* One of the FLSRF_* values */
    uint16_t sf;  /* Period of reporting in SFs. Valid only in
       PERIODICAL report frequency */
    uint32_t cell_report_flags; /* A bitmap of FLCST_* flags */
    uint32_t ue_report_flags;  /* A bitmap of FLUST_* flags */
};
```

#### Cell stats request

```
struct flex_cell_stats_request {
    uint16_t nr_cells;
    uint16_t cell_list[nr_cells];  /* The IDs of the cells */
    uint32_t flags;
};
```

#### UE stats request

```
struct flex_ue_stats_request {
    uint16_t nr_ue;
    uint16_t rnti_list[nr_ue];  /* The RNTIs of the UEs */
    uint32_t flags;
};
```

### Statistics reply message

The statistics reply message `FLPT_STATS_REPLY` consists of the message header, one list of UE statistics reports and one list of cell statistics reports

```
struct flex_stats_reply {
    struct flex_header header;
    uint8_t nr_ue_stats;  /* Number of UE reports */
    struct flex_ue_stats_report ue_reports[nr_ue_stats];  /* A list
       of the UE reports */
    uint8_t nr_cell_stats;  /* Number of cell reports */
    struct flex_cell_stats_report cell_reports[nr_cell_stats];  /* A
        list of cell reports */
};
```

## B.7 Controller command messages

### Configure DL MAC for next TTI

The command message FLPT_DL_MAC_CONFIG configures the state of the MAC for the DL for a specific SFN and SF. The message consists of the message header, the SFN and the SF in which the command is to be applied and three lists: one with configurations for random access responses, one with configurations for UE data and one for broadcast and paging data. Finally, the message contains the number of OFDM symbols used for the PDCCH channel of each carrier component.

```
struct flex_dl_mac_config {
    struct flex_header header;
    uint16_t sfn_sf;  /* The SFN and SF in which the configuration
       is to be applied. Bit 0-3 SF and 4-13 SFN */
    uint8_t nr_ue_data;  /* The number of UE data configuration
       elements */
    struct flex_dl_data dl_ue_data[nr_ue_data];  /* The UE data
       config elements */
    uint8_t nr_rar_data;  /* The number of RAR configuration
       elements */
    struct flex_dl_rar dl_rar[nr_rar_data];  /* The RAR config
       elements */
    uint8_t nr_broadcast_data;  /* The number of broadcast data
       configuration elements */
    struct flex_dl_broadcast dl_broadcast_list[nr_broadcast_data];
       /* The broadcast config elements */
    uint8_t nr_cc;  /* The number of CCs for CA */
    struct flex_pdcch_ofdm_sym_count ofdm_sym[nr_cc];  /* A list
       with OFDM symbol counts for each CC */
};
```

### Configure UL MAC for next TTI

The command message FLPT_UL_MAC_CONFIG configures the state of the MAC for the UL for a specific SFN and SF. The message consists of the message header, the SFN and the SF in which the command is to be applied and two lists: one for UL UE data configurations (DCI 0) and one for PHICH information transmissions.

```
struct flex_ul_mac_config {
    struct flex_header header;
```

```
    uint16_t sfn_sf;  /* The SFN and SF in which the configuration
       is to be applied. Bit 0-3 SF and 4-13 SFN */
    uint8_t nr_ue_data;  /* The number of UL UE data configuration
       elements */
    struct flex_ul_dci ul_dci_list[nr_ue_data];  /* The UE config
       elements (DCI 0) */
    uint8_t nr_phich_data;  /* The number of PHICH information */
        struct flex_ul_phich ul_phich_list[nr_phich_data];  /* The
           PHICH config elements */
};
```

## B.8 Asynchronous messages

These are messages sent by the agents to the controller to notify for events occurring at the eNBs.

### Scheduling request reception

The message `FLPT_UL_SR_INFO` provides scheduling request reception information that will be used by the scheduler. The message consists of the message header, the SFN and the SF in which the scheduling request was received and a list of the SRs received during that subframe.

```
struct flex_ul_sr_info {
    struct flex_header header;
    uint16_t sfn_sf;  /* The SFN and SF in which the configuration
       is to be applied. Bit 0-3 SF and 4-13 SFN */
    uint8_t nr_sr;  /* The number of SRs received */
    uint16_t sr_list[nr_sr];  /* A list of RNTIs with scheduling
       requests */
};
```

### RACH request reception

The message `FLPT_DL_RACH_INFO` provides information about the RACH process to the controller to be used in order to generate the RAR. The message consists of the message header, the SFN and the SF in which the RACH preamble was received and a list of the detected RACHs.

```
struct flex_dl_rach_info {
    struct flex_header header;
    uint16_t sfn_sf;  /* The SFN and SF in which the configuration
       is to be applied. Bit 0-3 SF and 4-13 SFN */
    uint8_t nr_rach;  /* The number of RACHs received */
    struct flex_rach rach_list[nr_rach];  /* The list of detected
       RACHs */
};
```

### UE state change

The UE state message FLPT_UE_STATE_CHANGE provides information about the state changes of the UE. The message consists of the message header, the type of state change and a body of type flex_ue_config containing the new state of the UE. This config does not provide a delta update (all fields of the config should be checked for changes). In the case of a message of type FLUESC_DEACTIVATED, the config is empty.

```
struct flex_ue_state_change {
        struct flex_header header;
        uint8_t type;  /* The type of the state change. One of the
           FLUESC_* values */
        struct flex_ue_config config;  /* The body of the message
           based on type. */
};
```

## B.9 Control delegation messages

These messages are used to allow the controller to assume or delegate control of one or more cells to the eNB agent as well as to inform the controller of the capability of the agent to deal with the commands issued by the controller within the designated time limits.

### Control load request

The control load request message contains no body. It is a message that consists only of a header with the type set to FLPT_CONTROL_LOAD_REQUEST. The controller expects a message of type FLPT_CONTROL_LOAD_REPLY from the agent.

### Control load reply

The control load reply message FLPT_CONTROL_LOAD_REPLY consists of a header and a list of statistics about control load for each cell of the eNB.

```
struct flex_control_load_reply {
    struct flex_header header;
    uint8_t nr_cells;  /* Number of cells reported */
    struct flex_control_load_stats load_stats[nr_cells];  /* A list
       of control load statistics for the cells */
};
```

### Control delegation

The control delegation message FLPT_DELEGATE_CONTROL consists of a header, an enumerator describing the type of delegation (e.g. dl/ul scheduler of the MAC, mobility manager in RRC etc), the delegated function payload in binary format and the name of the delegated function.

```
struct flex_control_delegation {
    struct flex_header header;
    uint32_t delegation_type;  /* The delegation type that should be
        performed by the agent. An enum of FLCDT_* values */
    uint64_t n_bytes;  /* The number of bytes in the payload of the
       delegated functions */
    uint8_t payload[n_bytes];  /* The delegated functions pushed by
       the controller */
    uint16_t name_length;  /* The length of the function's name */
    char name[name_length];  /* The delegated function name */
};
```

### Policy reconfiguration

The policy reconfiguration message FLPT_RECONFIGURE_AGENT consists of a header, and a string describing the policy changes using YAML syntax. The policy changes could be in the form of loading one of the delegated functions or changing the value of parameters for certain functions. The structure of the policy change string is described in section 2.

```
struct flex_agent_reconfiguration {
    struct flex_header header;
```

```
    uint16_t policy_length; /* The length of the policy change
        string */
    char policy[policy_length];/*The policy changes in YAML syntax*/
};
```

## B.10 Time indication messages

These messages are triggered by the eNB to signal the controller for specific events and to maintain its synchronization with the eNB.

### Subframe trigger

The subframe trigger message `FLPT_SF_TRIGGER` is triggered by the end of every TTI and is used to provide timing information to the controller in terms of SFN and SF as well as to provide UL/DL information updates (HARQ status, UL reception status). In the current protocol version the eNB is always expected to send these messages to the controller in each TTI unless MAC control has been delegated to the local controller. The message consists of a header, the SFN and the SF for the time indication and a list of reports about the status of DL HARQ processes and UL receptions for the reported subframe.

```
struct flex_dl_trigger {
    struct flex_header header;
    uint16_t sfn_sf;  /* The SFN and SF reported. Bit 0-3 SF and
        4-13 SFN */
    uint8_t nr_dl_info;  /* The number of elements in the DL
        information list */
    struct flex_dl_info dl_info_list[nr_dl_info];  /* The DL
        information list */
    uint8_t nr_ul_info;  /* The number of elements in the UL
        information list */
    struct flex_ul_info dl_info_list[nr_ul_info];  /* The UL
        information list */
};
```

# Appendix C

# DDPG Algorithm

Here, for completeness, we provide the details of the DDPG algorithm proposed in [83] and adjusted for the work presented in Chapter 6.

The execution of the algorithm is broken down into episodes and runs in an infinite loop. In each episode, a different random process is initialized for the action exploration. For the case of Iris, an episode corresponds to $H = 24$ periods and thus the total number of epochs for each episode is $He$, where according to the model of Section 6.2, $e$ is the number of epochs in a single period.

**Algorithm** DDPG algorithm

Randomly initialize critic network $Q(x, \alpha|\theta^Q)$ and actor $\pi(x|\theta^\pi)$ with weights $\theta^Q$ and $\theta^\pi$.
Initialize target network $Q'$ and $\pi'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\pi'} \leftarrow \theta^\pi$
Initialize replay buffer $R$
Initialize episode $m \leftarrow 0$
Receive initial observation state $x^0$
*loop*:
Initialize a random process $\mathcal{N}$ for action exploration
**for** $t = 0, \ldots, He$ **do**
    Select action $a^t = \pi(x^t|\theta^\pi) + \mathcal{N}^t$ according to the current policy and exploration noise
    Execute action $a^t$ and observe reward $r^t$ and new state $x^{t+1}$
    Store transition $(x^t, a^t, r^t, x^{t+1})$ in $R$
    Sample a random minibatch of $N$ transitions $(x^t, a^t, r^t, x^{t+1})$ from $R$
    Set $y^t = r^t + \gamma Q'(x^{t+1}, \pi'(x^{t+1}|\theta^{\pi'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y^i - Q(x^i, a^i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\pi} J \approx \frac{1}{N}\sum_i \nabla_a Q(x, a|\theta^Q)|_{x=x^t, a=\pi(x^t)} \nabla_{\theta^\pi} \pi(x|\theta^\pi)|_{x^t}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\pi'} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi'}$$

**end for**
$m \leftarrow m + 1$ ▷ Move to the next episode
$x^0 \leftarrow x^t$ ▷ Use last state as initial state of next episode
**goto** *loop*

# Bibliography


[1] http://zeromq.org/.

[2] https://developers.google.com/protocol-buffers/, 2016.

[3] http://dash.edgesuite.net/dash264/TestCases/2a/qualcomm/1/MultiResMPEG2.mpd, 2016.

[4] http://dash.edgesuite.net/akamai/streamroot/050714/Spring_4Ktest.mpd, 2016.

[5] DDPG implementation. https://github.com/stevenpjg/ddpg-aigym, 2018.

[6] 3GPP. Architecture enhancements for dedicated core networks. TS 23.707, 2017.

[7] 3GPP. Architecture Enhancements to Facilitate Communications with Packet Data Networks and Applications. TS 23.682, 2017.

[8] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures. TS 36.213, 2017.

[9] 5G Americas. Multi-operator and neutral host small cells: Drivers, architecture, planning and regulation, Dec 2016.

[10] I. F. Akyildiz et al. SoftAir: A software defined networking architecture for 5G wireless systems. *Computer Networks*, 85:1–18, 2015.

[11] H. Ali-Ahmad et al. CROWD: an SDN approach for DenseNets. In *Second European Workshop on Software Defined Networks (EWSDN)*, pages 25–31. IEEE, 2013.

[12] X. An et al. On end to end network slicing for 5G communication systems. *Transactions on Emerging Telecommunications Technologies*, 2016.

[13] A. Apostolaras et al. Evolved User Equipment for Collaborative Wireless Backhauling in Next Generation Cellular Networks. In *12th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 408–416. IEEE, 2015.

[14] M. Arslan et al. Software-Defined Networking in Cellular Radio Access Networks: Potential and Challenges. *Communications Magazine, IEEE*, 53(1):150–156, 2015.

[15] U. Author. IEEE 802.1q – IEEE standard for local and metropolitan area networks virtual bridged local area networks. *IEEE Computer Society*, 2005.

[16] A. Banerjee et al. Scaling the LTE Control-Plane for Future Mobile Access. In *Proceedings of the 11th ACM CoNEXT*, page 19. ACM, 2015.

[17] M. Bansal et al. OpenRadio: A Programmable Wireless Dataplane. In *Proceedings of the 1st workshop on Hot topics in Software Defined Networks*, pages 109–114. ACM, 2012.

[18] N. Baranasuriya et al. QProbe: Locating the bottleneck in cellular communication. In *Proceedings of the 11th ACM CoNEXT*, page 33. ACM, 2015.

[19] D. Bega et al. Optimising 5G infrastructure markets: The business of network slicing. In *INFOCOM 2017*, pages 1–9. IEEE, 2017.

[20] C. Bernardos et al. An Architecture for Software Defined Wireless Networking. *Wireless Communications, IEEE*, 21(3):52–61, 2014.

[21] F. Boccardi et al. Five Disruptive Technology Directions for 5G. *Communications Magazine, IEEE*, 52(2):74–80, 2014.

[22] A. Botta, A. Dainotti, and A. Pescapé. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, 56(15):3531–3547, 2012.

[23] G. Brockman et al. OpenAI Gym, 2016.

[24] L. Busoniu et al. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC press, 2010.

[25] P. Caballero et al. Multi-Tenant Radio Access Network Slicing: Statistical Multiplexing of Spatial Loads. *IEEE/ACM Transactions on Networking*, 25(5):3044–3058, 2017.

[26] P. Caballero et al. Network slicing games: Enabling customization in multi-tenent networks. In *Proceedings of IEEE INFOCOM 2017*. IEEE, 2017.

[27] L. Cano et al. Cooperative infrastructure and spectrum sharing in heterogeneous mobile networks. *IEEE Journal on Selected Areas in Communications*, 34(10):2617–2629, 2016.

[28] C.-Y. Chang et al. FlexCRAN: A flexible functional split framework over ethernet fronthaul in Cloud-RAN. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2017.

[29] T. Chen et al. SoftMobile: Control Evolution for Future Heterogeneous Mobile Networks. *Wireless Communications, IEEE*, 21(6):70–78, 2014.

[30] I. Chih-Lin et al. Recent Progress on C-RAN Centralization and Cloudification. *Access, IEEE*, 2:1030–1039, 2014.

[31] China Mobile. C-RAN: the road towards green RAN. *White Paper, ver*, 2, 2011.

[32] I. P. Chochliouros et al. A Novel Architectural Concept for Enhanced 5G Network Facilities. In *MATEC Web of Conferences*, volume 125, page 03012. EDP Sciences, 2017.

[33] B. Chun et al. Planetlab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[34] Cisco Visual Networking Index. Global mobile data traffic forecast update, 2016-2021, 2017.

[35] X. Costa-Pérez et al. Radio Access Network Virtualization for Future Mobile Carrier Networks. *IEEE Communications Magazine*, 51(7):27–35, 2013.

[36] S. Costanzo et al. OpeNB: A framework for Virtualizing Base Stations in LTE Networks. In *IEEE International Conference on Communications (ICC)*, pages 3148–3153. IEEE, 2014.

[37] DASH Industry Forum. DASH Reference Client. http://dashif.org/reference/players/javascript/v2.4.1/samples/dash-if-reference-player/index.html, 2017.

[38] S. Deb et al. Algorithms for Enhanced Inter Cell Interference Coordination (eICIC) in LTE HetNets. *IEEE/ACM Transactions on Networking (TON)*, 22(1):137–150, 2014.

[39] Y. Duan et al. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[40] Ericsson. A Vision of the 5G Core: Flexibility for New Business Opportunities, 2016.

[41] Ericsson. 5G Systems - Enabling the Transformation of Industry and Society, Jan 2017.

[42] ETSI. Open Source MANO. https://osm.etsi.org/, 2017.

[43] Eurecom. Demo at MWC 2017:5G Cloud RAN slice. https://insights.ubuntu.com/event/mobile-world-congress-2017/, 2017.

[44] J. O. Fajardo et al. Introducing mobile edge computing capabilities through distributed 5G cloud enabled small cells. *Mobile networks and applications*, 21(4):564–574, 2016.

[45] FCC. FCC Rule Making on 3.5 GHz Band / Citizens Broadband Radio Service, April 2015.

[46] X. Foukas et al. FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks. In *Proceedings of the 12th ACM CoNEXT*, pages 427–441. ACM, 2016.

[47] X. Foukas et al. Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture. In *Proceedings of the 23rd ACM MobiCom*, pages 127–140. ACM, 2017.

[48] L. Foundation. Netem linux network emulator. https://wiki.linuxfoundation.org/networking/netem, 2017.

[49] T. Frisanco et al. Infrastructure Sharing and Shared Operations for Mobile Network Operators from a Deployment and Operations View. In *Proceedings of IEEE NOMS 2008*, pages 129–136. IEEE, 2008.

[50] F. Fu and U. C. Kozat. Stochastic game for wireless network virtualization. *IEEE/ACM Transactions on Networking (ToN)*, 21(1):84–97, 2013.

[51] P. C. Garcés et al. RMSC: A Cell Slicing Controller for Virtualized Multitenant Mobile Networks. In *IEEE 81st Vehicular Technology Conference (VTC Spring)*, pages 1–6. IEEE, 2015.

[52] A. A. Gebremariam et al. Resource pooling via dynamic spectrum-level slicing across heterogeneous networks. In *14th IEEE annual consumer communications and networking conference (CCNC)*, 2017.

[53] I. Giannoulakis et al. The emergence of operator-neutral small cells as a strong case for cloud computing at the mobile edge. *Transactions on Emerging Telecommunications Technologies*, 27(9):1152–1159, 2016.

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

[55] S. Gu et al. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[56] A. Gudipati et al. SoftRAN: Software Defined Radio Access Network. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 25–30. ACM, 2013.

[57] A. Gudipati et al. RadioVisor: A Slicing Plane for Radio Access Networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 237–238. ACM, 2014.

[58] S. Ha et al. Tube: Time-dependent pricing for mobile data. *ACM SIGCOMM Computer Communication Review*, 42(4):247–258, 2012.

[59] J. He and W. Song. AppRAN: Application-Oriented Radio Access Network Sharing in Mobile Networks. In *Proceedings of IEEE International Conference on Communications (ICC)*, pages 3788–3794. IEEE, 2015.

[60] M. Helsley. LXC: Linux container tools. *IBM devloperWorks Technical Library*, page 11, 2009.

[61] P. Hintjens. *ZeroMQ: Messaging for Many Applications*. O'Reilly Media, Inc, 2013.

[62] Y. C. Hu et al. Mobile Edge Computing - A Key Technology Towards 5G. *ETSI White Paper*, 11, 2015.

[63] R. Irmer, H. Droste, P. Marsch, M. Grieger, G. Fettweis, S. Brueck, H.-P. Mayer, L. Thiele, and V. Jungnickel. Coordinated multipoint: Concepts, performance, and field trial results. *IEEE Communications Magazine*, 49(2):102–111, 2011.

[64] ITU. ITU-R M.[IMT-2020.TECH PERF REQ] - Minimum requirements related to technical performance for IMT-2020 radio interface(s), Feb 2017.

[65] M. Jiang et al. Network slicing management & prioritization in 5G mobile systems. In *Proceedings of 22th European Wireless Conference*, pages 1–6. VDE, 2016.

[66] X. Jin et al. Softcell: Scalable and Flexible Cellular Core Network Architecture. In *Proceedings of the ninth ACM CoNEXT*, pages 163–174. ACM, 2013.

[67] V. Joseph and G. de Veciana. NOVA: QoE-Driven Optimization of DASH-based Video Delivery in Networks. In *Proceedings of IEEE INFOCOM*, pages 82–90. IEEE, 2014.

[68] M. I. Kamel et al. LTE Wireless Network Virtualization: Dynamic Slicing via Flexible Scheduling. In *Vehicular Technology Conference (VTC Fall), 2014 IEEE 80th*, pages 1–5. IEEE, 2014.

[69] J. Khun-Jush et al. Licensed shared access as complementary approach to meet spectrum demands: Benefits for next generation cellular systems. In *ETSI Workshop on reconfigurable radio systems*, 2012.

[70] M. G. Kibria et al. Resource allocation in shared spectrum access communications for operators with diverse service requirements. *EURASIP Journal on Advances in Signal Processing*, 2016(1):83, 2016.

[71] M. G. Kibria et al. Heterogeneous networks in shared spectrum access communications. *IEEE Journal on Selected Areas in Communications*, 35(1):145–158, 2017.

[72] M. G. Kibria et al. Shared spectrum access communications: A neutral host micro operator approach. *IEEE Journal on Selected Areas in Communications*, 35(8):1741–1753, 2017.

[73] B.-G. Kim et al. Dynamic pricing for smart grid with reinforcement learning. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 640–645. IEEE, 2014.

[74] Y. H. Kim et al. Slicing the Next Mobile Packet Core Network. In *11th International Symposium on Wireless Communications Systems (ISWCS)*, pages 901–904. IEEE, 2014.

[75] A. Kivity et al. kvm: the Linux virtual machine monitor. In *Proceedings of the Linux symposium*, volume 1, pages 225–230, 2007.

[76] R. Kokku et al. NVS: A Substrate for Virtualizing Wireless Resources in Cellular Networks. *IEEE/ACM Transactions on Networking (TON)*, 20(5):1333–1346, 2012.

[77] R. Kokku et al. CellSlice: Cellular Wireless Resource Slicing for Active RAN Sharing. In *2013 Fifth International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–10. IEEE, 2013.

[78] A. Ksentini and N. Nikaein. Toward enforcing network slicing on ran: Flexibility and resources abstraction. *IEEE Communications Magazine*, 55(6):102–108, 2017.

[79] L. E. Li et al. Toward Software-Defined Cellular Networks. In *Proceedings of 2012 European Workshop on Software Defined Networking (EWSDN)*, pages 7–12. IEEE, 2012.

[80] N. Li et al. Optimal demand response based on utility maximization in power networks. In *Power and Energy Society General Meeting, 2011 IEEE*, pages 1–8. IEEE, 2011.

[81] Y. Li et al. Content-Aware Playout and Packet Scheduling for Video Streaming over Wireless Links. *IEEE Transactions on Multimedia*, 10(5):885–895, 2008.

[82] C. Liang and F. R. Yu. Wireless Virtualization for Next Generation Mobile Cellular Networks. *IEEE Wireless Communications*, 22(1):61–69, 2015.

[83] T. P. Lillicrap et al. Continuous control with deep reinforcement learning. In *ICLR 2016*, 2016.

[84] Linux Foundation. ONAP. `https://www.onap.org/`, 2017.

[85] Z. Liu et al. Pricing data center demand response. *ACM SIGMETRICS Performance Evaluation Review*, 42(1):111–123, 2014.

[86] R. Lowe et al. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6382–6393, 2017.

[87] J. Luo et al. Multi-carrier waveform based flexible inter-operator spectrum sharing for 5G systems. In *IEEE International Symposium on Dynamic Spectrum Access Networks (DYSPAN)*, pages 449–457. IEEE, 2014.

[88] T. Magedanz et al. Prototyping new concepts beyond 4G–the Fraunhofer Open5GCore. *it-Information Technology*, 57(5):314–320, 2015.

[89] R. Mahindra et al. Network-Wide Radio Access Network Sharing in Cellular Networks. In *Proceedings of 21st IEEE International Conference on Network Protocols (ICNP)*, pages 1–10. IEEE, 2013.

[90] T. Mahmoodi and S. Seetharaman. Traffic Jam: Handling the Increasing Volume of Mobile Data Traffic. *IEEE Vehicular Technology Magazine*, 9(3):56–62, 2014.

[91] N. Makris et al. Experimental evaluation of functional splits for 5G Cloud-RANs. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2017.

[92] I. Malanchini et al. Generalized Resource Sharing for Multiple Operators in Cellular Wireless Networks. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 803–808. IEEE, 2014.

[93] M. Matinmikko et al. Spectrum sharing using licensed shared access: the concept and its workflow for LTE-advanced networks. *IEEE Wireless Communications*, 21(2):72–79, 2014.

[94] M. Matinmikko et al. Micro Operators to Boost Local Service Delivery in 5G. *Wireless Personal Communications*, 95(1):69–82, 2017.

[95] N. McKeown et al. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[96] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[97] Microsoft Test Content. DASH Sample Video. `http://wams.edgesuite.net/media/MPTExpressionData02/BigBuckBunny_1080p24_IYUV_2ch.ism/manifest(format=mpd-time-csf)`, 2017.

[98] Mobile Experts. CBRS: New Shared Spectrum Enables Flexible Indoor and Outdoor Mobile Solutions and New Business Models, March 2017.

[99] Mobile Experts. Cisco Vision 5G: Thriving Indoors, March 2017.

[100] M. Moradi et al. SoftMoW: Recursive and Reconfigurable Cellular WAN Architecture. In *Proceedings of the 10th ACM CoNEXT*, pages 377–390. ACM, 2014.

[101] MulteFire Alliance. MulteFire release 1.0 technical paper: A new way to wireless, 2017.

[102] A. Nakao et al. Demo at ITU FG IMT-2020 Workshop:Softwarized LTE in FLARE network slices. `http://www.itu.int/en/ITU-T/Workshops-and-Seminars/201612/Pages/Programme.aspx`, Dec 2016.

[103] A. Nakao et al. End-to-end Network Slicing for 5G Mobile Networks. *Journal of Information Processing*, 25:153–163, 2017.

[104] NGMN-Alliance. 5G White Paper, Feb 2015.

[105] N. Nikaein. FlexRAN tutorial on RAN sharing. `https://gitlab.eurecom.fr/mosaic-5g/mosaic-5g/wikis/ran-sharing`, 2017.

[106] N. Nikaein et al. OpenAirInterface: A flexible platform for 5G research. *ACM SIGCOMM Computer Communication Review*, 44(5):33–38, 2014.

[107] N. Nikaein et al. Network store: Exploring slicing in future 5G networks. In *Proceedings of 10th ACM International Workshop on Mobility in the Evolving Internet Architecture (MobiArch'15)*, pages 8–13, Sep 2015.

[108] Nokia. Network Sharing: Delivering mobile broadband more efficiently and at lower cost. `http://resources.alcatel-lucent.com/asset/200192`, 2014.

[109] Ofcom. 3.8 GHz to 4.2 GHz Band: Opportunities for Innovation, April 2016.

[110] Ofcom. Communications Market Report, Aug 2017.

[111] D. O'Neill, M. Levorato, A. Goldsmith, and U. Mitra. Residential demand response using reinforcement learning. In *First IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 409–414. IEEE, 2010.

[112] OpenAirInterface Software Alliance. Openair-cn repository. `https://gitlab.eurecom.fr/oai/openair-cn`, 2017.

[113] P. Pahalawatta et al. Content-Aware Resource Allocation and Packet Scheduling for Video Transmission over Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 25(4), 2007.

[114] K. I. Pedersen et al. A Flexible 5G Frame Structure Design for Frequency-Division Duplex Cases. *IEEE Communications Magazine*, 54(3):53–59, 2016.

[115] K. Pentikousis et al. MobileFlow: Toward Software-Defined Mobile Networks. *Communications Magazine, IEEE*, 51(7):44–53, 2013.

[116] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[117] Z. A. Qazi et al. KLEIN: A Minimally Disruptive Design for an Elastic Cellular Core. In *Proceedings of the Symposium on SDN Research*, page 2. ACM, 2016.

[118] Qualcomm. Making 5G NR a reality, Dec 2016.

[119] R. Riggio. Demo: The EmPOWER Mobile Network Operating System. In *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, pages 87–88. ACM, 2016.

[120] P. Rost et al. Mobile Network Architecture Evolution Toward 5G. *IEEE Communications*, 54(5), 2016.

[121] P. Rost et al. Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks. *IEEE Communications magazine*, 2017.

[122] G. Salami et al. LTE indoor small cell capacity and coverage comparison. In *IEEE 24th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*, pages 66–70. IEEE, 2013.

[123] K. Samdanis et al. Service Boost: Towards on-demand QoS enhancements for OTT apps in LTE. In *21st IEEE International Conference on Network Protocols (ICNP)*, pages 1–6. IEEE, 2013.

[124] K. Samdanis et al. From Network Sharing to Multi-Tenancy: The 5G Network Slice Broker. *IEEE Communications*, 54(7), 2016.

[125] J. Schulman et al. High-dimensional continuous control using generalized advantage estimation. *ICLR 2016*, 2016.

[126] V. Sciancalepore et al. Mobile Traffic Forecasting for Maximizing 5G Network Slicing Resource Utilization. *IEEE INFOCOM*, 2017.

[127] R. Sherwood et al. FlowVisor: A Network Virtualization Layer. *OpenFlow Switch Consortium, Tech. Rep*, pages 1–13, 2009.

[128] D. Silver et al. Deterministic policy gradient algorithms. In *ICML*, 2014.

[129] Small Cell Forum. nFAPI and FAPI specifications, May 2017.

[130] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia*, pages 62–67, 2011.

[131] A. Syed and J. Van der Merwe. Proteus: A Network Service Control Platform for Service Evolution in a Mobile Software Defined Infrastructure. In *Proceedings of the 22nd ACM MobiCom*, pages 257–270. ACM, 2016.

[132] T. Taleb et al. Lightweight Mobile Core Networks for Machine Type Communications. *IEEE Access*, 2:1128–1137, 2014.

[133] T. Taleb et al. EASE: EPC as a Service to Ease Mobile Core Network Deployment over Cloud. *IEEE Network*, 29(2):78–88, 2015.

[134] K. Tsagkaris et al. SON Coordination in a Unified Management Framework. In *77th IEEE Vehicular Technology Conference (VTC Spring)*, pages 1–5. IEEE, 2013.

[135] B. Wang et al. Multimedia Streaming via TCP: An Analytic Performance Study. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 4(2):16, 2008.

[136] H. Wang et al. Understanding mobile traffic patterns of large scale cellular towers in urban environment. In *Proceedings of the 2015 Internet Measurement Conference*, pages 225–238. ACM, 2015.

[137] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[138] Z. Wen, D. O'Neill, and H. Maei. Optimal demand response using device-based reinforcement learning. *IEEE Transactions on Smart Grid*, 6(5):2312–2324, 2015.

[139] W. Wu et al. PRAN: Programmable Radio Access Networks. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 6. ACM, 2014.

[140] X. Xie et al. piStream: Physical Layer Informed Adaptive Video Streaming Over LTE. In *Proceedings of the 21st ACM MobiCom*, pages 413–425. ACM, 2015.

[141] M. Yang et al. OpenRAN: A Software-Defined RAN Architecture via Virtualization. In *ACM SIGCOMM computer communication review*, volume 43, pages 549–550. ACM, 2013.

[142] V. Yazıcı et al. A New Control Plane for 5G Network Architecture with a Case Study on Unified Handoff, Mobility, and Routing Management. *IEEE Communications Magazine*, 52(11):76–85, 2014.

[143] Y. Zaki et al. LTE Mobile Network Virtualization. *Mobile Networks and Applications*, 16(4):424–432, 2011.

[144] J. Zander and P. Mähönen. Riding the data tsunami in the cloud: myths and challenges in future wireless access. *IEEE Communications Magazine*, 51(3):145–151, 2013.

[145] L. Zhao et al. LTE virtualization: From Theoretical Gain to Practical Solution. In *Proceedings of the 23rd International Teletraffic Congress*, pages 71–78. International Teletraffic Congress, 2011.

[146] K. Zhu and E. Hossain. Virtualization of 5G cellular networks as a hierarchical combinatorial auction. *IEEE Transactions on Mobile Computing*, 15(10):2640–2654, 2016.