Softwarized Approaches for the Open RAN of NextG Cellular Networks

A Dissertation Presented
by

Leonardo Bonati

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Engineering

Northeastern University
Boston, Massachusetts

August 2022

To my family.

Abstract

Softwarized Approaches for the Open RAN of NextG Cellular Networks

by

Leonardo Bonati

Doctor of Philosophy in Computer Engineering

Northeastern University, August 2022

Dr. Tommaso Melodia, Advisor ♦ Dr. Stefano Basagni, Advisor

The 5th and 6th generations of cellular networks (5G and 6G), also known as NextG, will bring unprecedented flexibility to the wireless cellular ecosystem. Because of a typically closed and rigid market, the telco industry has incurred high costs and non-trivial obstacles for delivering new services and functionalities that satisfy the requirements and the demands of NextG networks. To break this trend the industry is now moving toward *open* architectures based on *softwarized* approaches, which afford network operators *flexible control* and unprecedented *adaptability to heterogeneous conditions*, including traffic and application requirements. Now, by simply expressing a high-level intent, operators will be able to instantiate bespoke services *on-demand* on a generic hardware infrastructure, and to adapt such services to the current network conditions. The separation of control functions from the hardware fabric, and the introduction of open and standardized control interfaces, will ultimately enable the definition and use of *softwarized control loops*, which will bring embedded intelligence and real-time analytics to effectively realizing the vision of *autonomous and self-optimizing networks*.

This dissertation work focuses on the design, prototyping and experimental evaluation of *softwarized approaches for the Open Radio Access Network (RAN) of NextG cellular networks*. We analyze the architectural enablers, challenges, and requirements for a *programmatic zero-touch control* of the network elements and propose practical solutions for its realization. We prototype solutions by leveraging open-source software implementations of cellular protocol stacks and frameworks, and heterogeneous virtualization technologies. The contributions of this work include (i) OpenRAN Gym, the first publicly-available research platform for the design, prototyping, and experimentation at scale of *data-driven O-RAN solutions*; (ii) CellOS, a *zero-touch cellular operating system* for the automatic generation and execution of distributed network optimization control programs, and (iii) OrchestRAN, a network intelligence orchestration framework for Open RAN that automates

the deployment of data-driven inference and control solutions. The effectiveness of our solutions in achieving superior control and performance of the RAN is demonstrated at scale on state-of-the-art experimental facilities, including software-defined radio-based laboratory setups and open access experimental wireless platforms, such as Colosseum, Arena, and the POWDER and COSMOS platforms from the PAWR program.

Table of Contents

List of Figures

List of Tables

List of Acronyms

3GPP 3rd Generation Partnership Project
4G 4th generation
5G 5th generation
5GC 5G Core
ADC Analog-to-Digital Converter
AERPAW Aerial Experimentation and Research Platform for Advanced Wireless
AI Artificial Intelligence
AM Acknowledged Mode
AMF Access and Mobility Management Function
AoA Angle of Arrival
AP Access Point
API Application Programming Interface
APN Access Point Name
ARA Agriculture and Rural Communities
ASK Amplitude Shift Keying
AUSF Authentication Server Function
BS Base Station
BSS Business Support System
CaaS Connectivity-as-a-Service
CCaaS Cellular Connectivity-as-a-Service
CDD Cyclic Delay Diversity
CDF Cumulative Distribution Function
CDN Content Distribution Network
CellOS Cellular Operating System
CIR Channel Impulse Response
CN Core Network
COMAC Converged Multi-Access and Core
CoMP Coordinated Multi-point
CORD Central Office Re-architected as a Datacenter
CORNET COgnitive Radio NETwork
COSMOS Cloud Enhanced Open Software Defined Mobile Wireless Testbed for City-Scale Deployment
COTS Commercial Off-the-Shelf
CPU Central Processing Unit
CQI Channel Quality Information
CR Cognitive Radio
CRC Cyclic Redundancy Check
CSI Channel State Information
CU Central Unit
DAC Digital-to-Analog Converter
DARPA Defense Advanced Research Projects Agency
DAS Distributed Antenna System
DL Downlink

DL-SCH Downlink Shared Channel
DNN Deep Neural Network
DQN Deep Q-Network
DRL Deep Reinforcement Learning
DRS Discovery Reference Signal
DU Distributed Unit
E-UTRAN Evolved Universal Terrestrial Access Network
EI Enrichment Information
eMBB Enhanced Mobile Broadband
eNB evolved Node Base
EPC Evolved Packet Core
EPS Evolved Packet System
ETSI European Telecommunications Standards Institute
FaaS Function-as-a-Service
FAPI Functional Application Platform Interface
FCC Federal Communications Commission
FDD Frequency Division Duplexing
FED4FIRE+ Federation 4 Future Internet Research and Experimentation Plus
FIR Finite Impulse Response
FIT Future Internet of Things
FPGA Field Programmable Gate Array
gNB Next Generation Node Base
GPU Graphics Processing Unit
GTP GPRS Tunneling Protocol
GTP-C GPRS Tunnelling Protocol Control Plane
GTP-U GPRS Tunnelling Protocol User Plane
GUI Graphical User Interface
GW Gateway
HARQ Hybrid Automatic Repeat reQuest
HMAC Keyed-Hash Message Authentication Code
HSS Home Subscription Server
IBSPC Inter-base Station Power Control
ICMP Internet Control Message Protocol
IMSI International Mobile Subscriber Identity
IoT Internet of Things
IP Infrastructure Provider
JT Joint Transmission
K-S Kolmogorov–Smirnov
KPI Key Performance Indicator
KPM Key Performance Measurement
KVM Kernel-based Virtual Machine
LTE Long Term Evolution
LXC Linux Container
MAC Medium Access Control
MANO Management and Orchestration
MCC Mobile Cloud Computing
MCHEM Massive Channel Emulator
MCS Modulation and Coding Scheme
MEC Multi-access Edge Computing
MFC Mobile Fog Computing
MGEN Multi-Generator
MIMO Multiple Input, Multiple Output
MISO Multiple Input, Single Output
ML Machine Learning

MME Mobility Management Entity
mmWave millimeter wave
MRT Maximum Ratio Transmission
MSIN Mobile Subscription Identification Number
MTC Machine-type Communications
MU-MIMO Multi-user MIMO
MVNO Mobile Virtual Network Operator
NAS Network Attached Storage
NFV Network Function Virtualization
NFVI Network Function Virtualization Infrastructure
NIC Network Interface Card
NO Network Operator
NRDZ National Radio Dynamic Zone
NRF Network Repository Function
NSA Non Standalone
NSE Network Slicing Engine
NSF National Science Foundation
NSM Network Service Mesh
NSSF Network Slice Selection Function
OAI OpenAirInterface
OAI-CN OpenAirInterface (OAI) Core Network
OAI-RAN OpenAirInterface Radio Access Network
OAM Operations, Administration and Maintenance
OMEC Open Mobile Evolved Core
ONAP Open Network Automation Platform
ONF Open Networking Foundation
ONOS Open Networking Operating System
OOM Open Network Automation Platform (ONAP) Operations Manager
OPNFV Open Platform for Network Function Virtualization (NFV)
ORBIT Open-Access Research Testbed for Next-Generation Wireless Networks
OSC O-RAN Software Community
OSM Open Source Management and Orchestration
OSS Operations Support System
PAWR Platforms for Advanced Wireless Research
PBCH Physical Broadcast Channel
PCA Principal Component Analysis
PCCaaS Private Cellular Connectivity-as-a-Service
PCEF Policy and Charging Enforcement Function
PCFICH Physical Control Format Indicator Channel
PCRF Policy and Charging Rules Function
PDCCH Physical Downlink Control Channel
PDCP Packet Data Convergence Protocol
PDF Probability Density Function
PDSCH Physical Downlink Shared Channel
PF Proportional Fair
PGW Packet Gateway
PHICH Physical Hybrid ARQ Indicator Channel
PHY Physical
PL Partial Linearization
PMCH Physical Multicast Channel
PMI Precoding Matrix Indicators
POWDER Platform for Open Wireless Data-driven Experimental Research
PPO Proximal Policy Optimization
PRACH Physical Random Access Channel

PRB Physical Resource Block
PSS Primary Synchronization Signal
PUCCH Physical Uplink Control Channel
PUSCH Physical Uplink Shared Channel
QAM Quadrature Amplitude Modulation
QCI Quality of Service (QoS) Class Identifier
QoE Quality of Experience
QoS Quality of Service
QPSK Quadrature Phase Shift Keying
RACH Random Access Channel
RAN Radio Access Network
RAT Radio Access Technology
RB Resource Block
RBG Resource Block Group
REC Radio Edge Cloud
REE Reconfigurable Edge Element
RENEW Reconfigurable Eco-system for Next-generation End-to-end Wireless
RF Radio Frequency
RIC RAN Intelligent Controller
RLC Radio Link Control
RMR RIC Message Router
RPS Reconfigurable Protocol Stack
RR Round Robin
RRC Radio Resource Control
RRU Remote Radio Unit
RU Radio Unit
S1AP S1 Application Protocol
SA Standalone
SC2 Spectrum Collaboration Challenge
SDAP Service Data Adaptation Protocol
SDK Software Development Kit
SDN Software-defined Networking
SDR Software-defined Radio
SEBA SDN-Enabled Broadband Access
SGD Stochastic Gradient Descent
SGW Service Gateway
SINR Signal-to-Interference-plus-Noise Ratio
SISO Single Input, Single Output
SLA Service Level Agreement
SM Service Model
SMF Session Management Function
SMO Service Management and Orchestration
SNR Signal-to-Noise-Ratio
SRN Standard Radio Node
SRS Sounding Reference Signal
SSH Secure Shell
SSS Secondary Synchronization Signal
TB Transport Block
TDD Time Division Duplexing
Telco Telecommunications Company
TFT Traffic Flow Template
TGEN Traffic Generator
TIP Telecom Infra Project
TM Transparent Mode

TTI Transmission Time Interval
UAS Unmanned Aerial System
UAV Unmanned Aerial Vehicle
UDM Unified Data Management
UDP User Datagram Protocol
UDR Unified Data Repository
UE User Equipment
UHD USRP Hardware Driver
UL Uplink
UL-SCH Uplink Shared Channel
UM Unacknowledged Mode
UPF User Plane Function
UPS Uninterruptible Power Supply
URLLC Ultra Reliable and Low Latency Communication
USIM Universal Subscriber Identity Module
USRP Universal Software Radio Peripheral
VIM Virtualization Infrastructure Manager
VM Virtual Machine
VNF Virtual Network Function
VoLTE Voice over LTE
VoNR Voice over NR
vRAN Virtualized RAN
WF Wired-first
ZSM Zero-touch Network and Service Management

Chapter 1

Introduction

The 5th and 6th generations of cellular networks (5G and 6G), also known as NextG, will enable unparalleled technological advancements in the networking hardware and software ecosystems [8]. Based *openness* and *programmability*, this novel paradigm will overcome issues such as spectrum crunch and resource-scarcity that have haunted previous cellular generations for years. This will ultimately usher applications like telemedicine, virtual reality, high-resolution video streaming, and private cellular networking, just to name a few. By unbridling the sheer power of these applications, NextG networks will bring unprecedented flexibility to the wireless cellular ecosystem, fostering unrivaled cellular networking-based innovation [9].

The journey to achieve this vision, however, is still beset by many research and development challenges. Previous cellular generations are characterized by an *inflexible and monolithic infrastructure*, where hardware and software components are *plug-and-play* with little or no reconfiguration capabilities. These "black-box" approaches are incapable of meeting the heterogeneity and variability of the novel services envisioned for NextG networks, and the strict requirements of their applications [10]. The lack of full control of the vast amount of available resources and network parameters makes it hard to *adapt network operations to real-time traffic conditions and requirements*, resulting in ineffective resource management, sub-optimal performance, and inability to implement connectivity-as-a-service technologies such as private cellular networking [11]. The inflexibility of current approaches is even more harmful in NextG scenarios, where densification of deployments and the need for directional communications call for fine-grained network control [12–14], resources are scarce and spectrum availability and energy consumption are strictly regulated [15].

Flexibility for real-time and swift adaptation is therefore key to accommodate the requirements of future cellular wireless applications. To achieve this unprecedented level of dynamic control, the industry is now moving toward *open* architectures based on *softwarized* approaches, which will allow operators to instantiate bespoke services *on-demand* on a generic hardware infrastructure—and to adapt such services to the current network conditions—by simply expressing a high-level intent. Through *disaggregation*, functionalities that were implemented by previous cellular generations on monolithic and specialized devices will be *split* across multiple components—possibly provided by different vendors—interconnected through well-defined *open interfaces*, also used to capture and expose Key Performance Measurements (KPMs) and network analytics. The separation of control functions from the hardware fabric, and the introduction of standardized control interfaces, will ultimately enable the definition and use of *softwarized and programmable control loops*, which will

bring embedded intelligence and real-time analytics to effectively realizing the vision of *autonomous and self-optimizing networks*. However, achieving the ambitious objective of realizing the *softwarized* and *open* Radio Access Network (RAN) envisioned for NextG networks is not trivial as novel solutions are required to handle the increasing densification and scale of cellular deployments, adapt to diverse wireless environments, and react to changing traffic conditions and requirements, whilst avoiding causing outages to the network services.

This dissertation work focuses on the design, prototyping and experimental evaluation of *softwarized approaches for the Open RAN of NextG cellular networks*. We analyze the architectural enablers, challenges and requirements for a *programmatic zero-touch control* of the very many network elements and propose practical solutions for its realization. We prototype solutions by leveraging open-source software implementations of cellular protocol stacks and frameworks, and heterogeneous virtualization technologies, including the srsRAN and OpenAirInterface cellular implementations, the O-RAN framework, and the LXC and Docker virtualization technologies. The contributions of this work include: (i) the first demonstration of O-RAN data-driven control loops in a *large-scale experimental testbed using open-source, programmable* RAN and RAN Intelligent Controller (RIC) components through *xApps* and *dApps* of our design; (ii) CellOS, a *zero-touch cellular operating system* that automatically generates and executes distributed control programs for the *simultaneous optimization of heterogeneous control objectives on multiple network slices* starting from a high-level intent expressed by the operators; (iii) QCell, a Deep Q-Network-based framework for the *self-optimization of slice resources and policies*; (iv) OpenRAN Gym, the first publicly-available research platform for the design, prototyping, and experimentation at scale of *data-driven O-RAN solutions*; (v) SCOPE, an open and softwarized platform for *large-scale prototyping and experimentation* of NextG solution, and for the *automated data-collection* of RANs KPMs; (vi) ColO-RAN, a first-of-its-kind development platform for Deep Reinforcement Learning (DRL)-based xApps for closed-loop control in O-RAN on large-scale experimental platforms; (vii) OrchestRAN, a network intelligence orchestration framework for Open RAN systems that leverages O-RAN applications and open interfaces to provide operators with an automated orchestration tool for deploying flexible data-driven inference and control solutions, and (viii) SteaLTE, the first realization of *private cellular connectivity-as-a-service* in NextG networks through wireless steganography. The effectiveness of our solutions in achieving superior control and performance of the RAN is demonstrated on state-of-the-art experimental facilities, including software-defined radio-based laboratory setups and open access experimental wireless platforms, such as Colosseum, Arena, and the POWDER and COSMOS platforms from the U.S. PAWR program [5].

The remaining of this work is organized as follows. Chapter 2 (based on [523]) provides an overview of the state-of-the-art of open, programmable and virtualized cellular networks, focusing on architectural enablers, frameworks and technologies. Chapter 3 (derived from [528,532,542,545,547]) describes the main software tools and experimental platforms developed and used throughout this work. Chapter 4 (based on [526, 530]) demonstrates the potential and effectiveness of data-driven control loop enabled by the O-RAN architecture. Chapter 5 (which combines [521, 543]) proposes approaches for the distributed zero-touch optimization of softwarized cellular networks. Chapter 6 (based on [546]) discusses the intelligent orchestration of network components in the Open RAN. Chapter 7 (derived from [540]) analyzes the concept of private cellular connectivity-as-a-service and proposes viable solutions to achieve private communications in NextG networks. Finally, Chapter 8 concludes this work.

Chapter 2

Open, Programmable and Virtualized Cellular Networks: The State-of-the-Art

Researchers from both industry and academia agree that the practical realization of 5th generation (5G) systems needs a radical overhaul of all plug-and-play approaches in favor of new, agile and open paradigms for network deployment, control and management. In this context, revolutionary and innovative networking solutions based upon *programmability*, *openness*, *resource sharing* and *edgefication* are welcome to the cellular arena [16, 17]. New networking principles such as Software-defined Networking (SDN) [18], network virtualization [19], and Multi-access Edge Computing (MEC) [20] have demonstrated that dynamic network control and agile management (e.g., frequency planning, user scheduling, mobility management, among others) is possible. Similarly, the emergence of network slicing and cloud Radio Access Network (RAN) technologies have made it clear that infrastructure sharing not only maximizes resource utilization, but also opens new market opportunities (e.g., differentiated services, infrastructure leasing, Connectivity-as-a-Service (CaaS)), *thus representing a desirable solution for network operators and infrastructure providers alike* [21, 22].

Following the growing interest in softwarization and virtualization technologies, the 5G ecosystem has witnessed the exponential growth of dedicated solutions for 5G applications [23]. These solutions include software and hardware tailored to specific tasks [24] and full-fledged multitasking frameworks spanning the whole infrastructure [25]. Despite their diversity in structure and purpose, the majority of these solutions has two important aspects in common: they are *open-source* and fully *programmable*. *These two aspects together are bringing unprecedented flexibility to 5G systems, making them accessible to a much broader community of researchers and developers.*

Just a few years ago, the majority of researchers had no access to actual cellular networks. When they did, access was limited to individual network components or functionalities. Today, the software-defined paradigm as made popular by the GNU Radio libraries [26], has been easily adopted by software bundles such as OpenAirInterface (OAI) [24] and srsRAN [27] for swift instantiation of fully-functional cellular networks on commercial Software-defined Radio (SDR) devices. Software frameworks such as O-RAN [28, 29], which run on "white-box" servers, allow reconfiguration and optimization of network and transceiver functionalities. These new software and hardware components have radically changed the way the research community and the telecom industry plan,

deploy, and interact with cellular systems. Prototyping, testing, and deploying new algorithms and protocols for cellular networks enjoys now unprecedented ease and time to market. The advantage of this revolutionary approach is twofold: (i) *openness* allows researchers to evaluate and analyze their solutions on real-world setups (see Chapter 3), and enables telecom operators to directly interact and control networking equipment (see Chapter 4 and [11]). Also, (ii) *programmability* fosters the design of novel and advanced algorithms that optimize network performance by efficiently and dynamically allocating network resources and controlling software and hardware functionalities, even in real time, if appropriate. For instance, telecom operators such as Rakuten are leveraging microservices to separate the user and control planes in their network deployments, thus endowing them with unprecedented flexibility [30]. Programs like Platforms for Advanced Wireless Research (PAWR) by the U.S. National Science Foundation [5], are bringing programmable wireless testing infrastructure at scale to broad communities of researchers—thus creating a fertile ground for software-based open innovation.

The race to the open-source and programmable Holy Grail has generated a plethora of heterogeneous software and hardware components and frameworks, whose functionality, scope, and interoperability with other solutions are often obscure and hard to assess. *This chapter, which was published as [523], organizes the multiplicity of solutions into the appropriate building blocks of the open-source and programmable 5G ecosystem. We detail how each components fits into a 5G network, highlight the interactions among solutions, and unfold their capabilities and functionalities, highlighting strengths and limitations. Our survey provides the first cohesive and exhaustive recount and taxonomy of open, programmable, and virtualized solutions for 5G networks.* As most frameworks and devices serve specific purposes in the 5G architectures, we also provide usage directives and how-to guidelines to combine different components into full-fledged open-source 5G systems. With respect to previous survey efforts [31, 32] we provide extensive details and commentary on the architecture of softwarized 5G networks, their building blocks, the software frameworks developed so far, and their interactions.

Figure 2.1 provides a visual guide to how the topics surveyed in this chapter relate to one another, as well as to the structure of the remainder of this chapter. Section 2.1 provides a bird's-eye view of the architecture of 5G systems, describing its components and technologies. Sections 2.2 and 2.3 introduce and describe open-source solutions for the RAN and Core Network (CN) portions of the infrastructure, respectively. General open-source frameworks inclusive of both RAN and CN functionalities are discussed in Section 2.4. Virtualization and management frameworks are provided in details in Section 2.5. Section 2.6 describes software-defined hardware platforms for open-source radio units, highlighting their features and their suitability for 5G applications. Section 2.7 presents a variety of experimental testbeds allowing instantiation of softwarized 5G networks and testing of new solutions. Finally, in Section 2.8 we conclude this chapter by identifying limitations of the current 5G open-source ecosystem and discuss the road ahead, with its unanswered research questions.

2.1 Architectural Enablers of 5G Cellular Networks

Mobile networks are transitioning from monolithic architectures, based on dedicated "*black-box*" hardware with proprietary firmware and software, to disaggregated deployments based on open-source software that runs on generic SDR or "agnostic" computing devices [33–35]. This trend is not new to

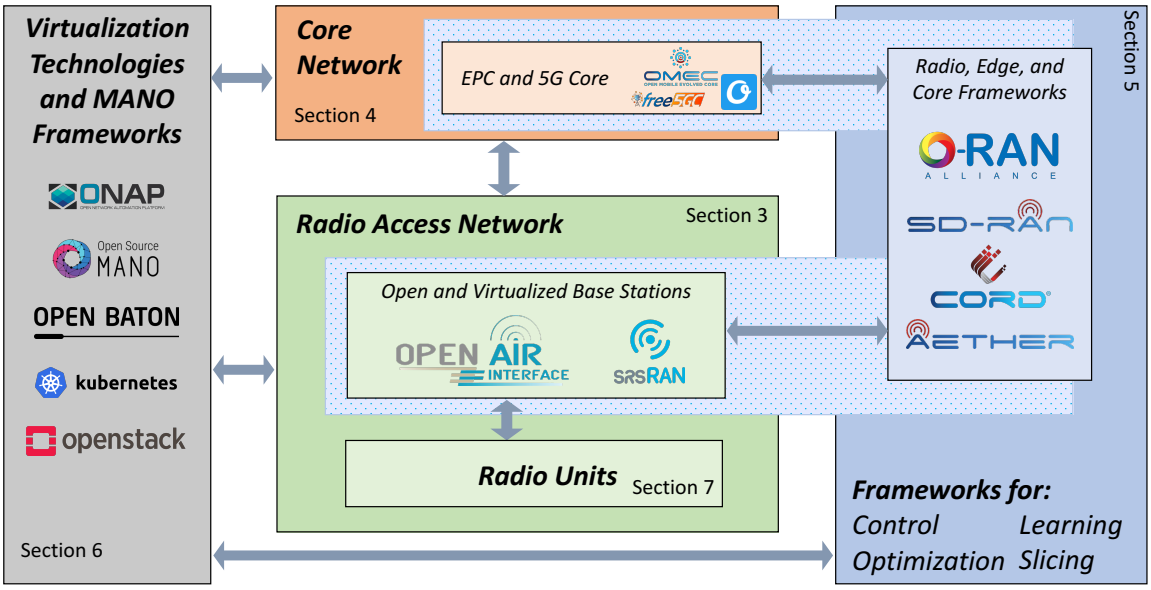


Figure 2.1: The main building blocks of open-source, programmable and virtualized 5G networks with their components and technologies.

cellular networking, as it has been part of the general discussion around 4th generation (4G) cellular networks. However, while software-based design represents a relatively recent evolution in the context of 4G networks, 5G specifications have foreseen the flexible deployment of agile, softwarized services already in their early stages, with their application to key infrastructure components such as the core, the RAN and the edge cloud [36]. This "flexibility-by-design" puts 5G networks in the privileged position to meet the requirements of heterogeneous traffic classes, mobility and advanced applications through design that is unified, open and dynamically changeable.

In this section we provide an overview of 4G and 5G cellular network architectures, as well as their main components and building blocks (Figure 2.1). We start by describing radio access and core network elements and general deployment paradigms. We then discuss the architectural and technology enablers such as Software-defined Networking (SDN), Network Function Virtualization (NFV), network slicing, MEC, and intelligent networks. Our aim is to provide a reference architecture to map the different open-source software libraries and frameworks surveyed in this chapter to specific network functionalities.

2.1.1 Architecture of 4G and 5G Cellular Networks

Figure 2.2 provides a high-level overview of the 4G and 5G cellular architectures, along with some of the open-source software frameworks envisioned as their components.

Cellular networks consist of a Radio Access Network (RAN) and a Core Network (CN). Even though this separation remains unaltered in 4G and 5G deployments, the actual implementation and configuration of these core components differ greatly. Particularly, they comply with the 3rd

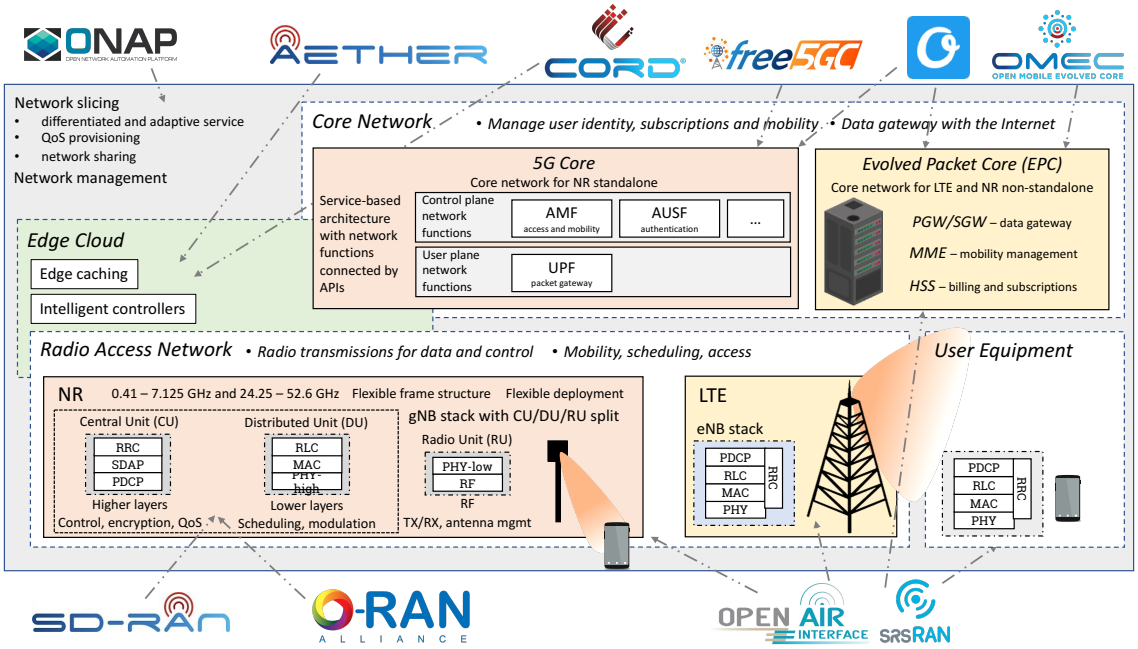


Figure 2.2: Cellular network architecture.

Generation Partnership Project (3GPP) Long Term Evolution (LTE) and NR[1] specifications for the RAN, and the Evolved Packet Core (EPC) and 5G Core (5GC) for the CN, respectively.[2] Figure 2.2 highlights the differences, in terms of flexibility, between the deployments of 4G (in the yellow boxes) and 5G (in the orange boxes) networks. For the CN, the 4G EPC has multiple components that have been traditionally executed on dedicated hardware, and only recently have transitioned to software-based deployments. The 5GC, instead, has been designed according to a service-based approach from the get-go. The EPC servers are split into multiple virtual network functions providing specific functionalities. They are connected to each other through open and standardized interfaces. A similar separation principle has been considered for the 5G RAN, now designed to provide a functional split among heterogeneous parts of the base stations (e.g., control, computing and radio units), with different layers of the protocol stack instantiated on different elements located in different parts of the network.

LTE and EPC. The LTE RAN is composed of evolved Node Bases (eNBs), i.e., the LTE base stations, which provide wireless connectivity to the mobile User Equipments (UEs). The eNBs are generally deployed as a single piece of equipment on dedicated hardware components, and are networked together and to the core network. LTE operates on a frame structure with 10 subframes of 1 ms per frame, and 12 to 14 OFDM symbols for each subframe. The maximum carrier bandwidth is 20 MHz. Up to 5 carriers can be aggregated for a total of 100 MHz [39].

[1] Although initially introduced as "New Radio" in [37], NR has lost its original meaning in the latest 3GPP specifications [36] where it now refers to the 5G RAN.

[2] Notice that, while LTE has been originally associated with 4G networks, its evolution (e.g., LTE-A) will be part of the air interface of 5G networks, together with NR [38].

The LTE protocol stack for the user plane (also known as Evolved Universal Terrestrial Access Network (E-UTRAN), bottom right corner of the RAN box in Figure 2.2) consists of:

- *The Packet Data Convergence Protocol (PDCP) layer,* which implements security functionalities (e.g., ciphering of packets), performs header compression, and takes care of the end-to-end packet delivery between the eNB and the UE [40].

- *The Radio Link Control (RLC) layer,* which provides data link layer services (e.g., error correction, packet fragmentation and reconstruction). It supports three different configurations: The Transparent Mode (TM), to simply relay packets between the Medium Access Control (MAC) and PDCP layers; the Unacknowledged Mode (UM), for buffering, segmentation, concatenation and reordering, and the Acknowledged Mode (AM), for retransmitting packets via a ACK/NACK feedback loop [41].

- *The MAC layer,* which performs scheduling, interacts with RLC to signal transmissions, forwards the transport blocks to the physical layer, and performs retransmissions via Hybrid Automatic Repeat reQuest (HARQ) [42].

- *The Physical (PHY) layer,* which takes care of channel coding, modulates the signal, and performs transmissions in an OFDM-based frame structure [43].

These layers also perform control plane functionalities, which concern measurement collection and channel quality estimation. Additionally, the Radio Resource Control (RRC) layer manages the life cycle of the eNB to UE connection, and it is a point of contact with the core network for control functionalities.

The main components of the EPC (in the top right corner of Figure 2.2) are: (i) the Packet Gateway (PGW) and Service Gateway (SGW), which are packet gateways to and from the Internet; (ii) the Mobility Management Entity (MME), which handles handovers and the UE connection life cycle from the core network point of view, and (iii) the Home Subscription Server (HSS), which manages subscriptions and billing [44].

NR. The 3GPP NR RAN represents quite the evolution of the 4G LTE, especially in terms of protocol stack, functionalities and capabilities. First, it supports a wider range of carrier frequencies, which include part of the millimeter wave (mmWave) spectrum [45]. Second, the frame structure, while still OFDM-based, is more flexible, with a variable number of symbols per subframe, the option to use much larger bandwidths than LTE (up to 400 MHz per carrier), and the integration of signals and procedures to manage directional transmissions at mmWaves [46]. Third, the 5G RAN can be connected either to the 4G EPC (*non-standalone configuration*) or to the new 5GC (*standalone configuration*). Finally, the NR base stations (Next Generation Node Bases (gNBs)) allows distributed deployment, with different parts of the protocol stack in different hardware components.

The NR protocol stack (bottom left corner of Figure 2.2) features a new layer on top of the PDCP, i.e., the Service Data Adaptation Protocol (SDAP) layer [47], which manages the Quality of Service (QoS) of end-to-end flows, and maps them to local resources in the gNB-UE link. The design of the remaining layers has been updated to support the aforementioned NR features [48–52].

CU/DU Split and the Virtualized RAN Architecture. The main innovation introduced by NR comes from the possibility of splitting the higher layers of the 3GPP stack (PDCP, SDAP, and RRC) and the lower layers (RLC, MAC, and PHY) into two different logical units, called *Central Unit (CU)* and the *Distributed Unit (DU)*, which can be deployed at separate locations. Moreover, the lower part of the physical layer can be separated from the DU in a standalone *Radio Unit (RU)*. The CU, DU and RU are connected through well-defined interfaces operating at different data rates and latency (with tighter constraints between the DU and RU).

This architecture, proposed by 3GPP in [53], enables the Virtualized RAN (vRAN) paradigm. Specifically, the antenna elements (in the RU) are separated from the baseband and signal processing units (in the DU and CU), which are hosted on generic, even multi-vendor, hardware. If the interfaces between the different RAN components are open, the 5G deployment follows the Open RAN model, which defines open and standardized interfaces among the elements of the disaggregated RAN [54]. A notable example of Open RAN is currently being promoted by the O-RAN Alliance [11]. This consortium has defined a set of interfaces between CU, DU, RU, and a RAN Intelligent Controller (RIC) that can be deployed at the edge of the network (see also Section 2.4.1).

The 5G Core. Openness and flexibility have guides the design of the 5GC, now realized according to a service-based approach [55]. Control and user plane core functionalities have been split into multiple network functions [56]. The 3GPP has also defined interfaces and Application Programming Interfaces (APIs) among the network functions, which can be instantiated on the fly, enabling elastic network deployments and network slicing (Section 2.1.3). The User Plane Function (UPF) is a user plane gateway to the public Internet that acts as mobility anchor and QoS classifier for the incoming flows. On the control plane side, most of the MME functions (e.g., mobility management) are assigned to the Access and Mobility Management Function (AMF). The Session Management Function (SMF) allocates IP addresses to the UEs, and orchestrates user plane services, including the selection of which UPF a UE should use. For a detailed overview of all 5G core functions the reader is referred to [55, 57].

2.1.2 Enabling Technologies for Softwarized 5G Cellular Networks

5G networks will embody heterogeneous network components and technologies to provide unprecedented performance levels and a unique experience to subscribers. Managing the integration of such a menagerie of technologies, controlling such variegate infrastructure and orchestrating network services and functionalities is clearly no trivial feat. To solve this management and control problem, 5G networks have borrowed widespread and well-established processes and architectures from the cloud-computing ecosystem, where softwarization and virtualization are merged together to abstract services and functionalities from the hardware where they are executed. In the following, we introduce two of these technologies and how they integrate with future 5G systems.

Softwarization and Software-defined Networking. In order to integrate hardware components produced by multiple vendors with different functionalities and configuration parameters, 5G systems rely on *softwarization*. This technology concept grew in popularity in the second decade of the 21st century thanks to the Software-defined Networking (SDN) architectural paradigm and the widespread adoption of the now well-established OpenFlow protocol. As shown in Figure 2.3, SDN leverages softwarization to decouple network control from the forwarding (or data) plane, thus

separating routing and control procedures from specialized hardware-based forwarding operations. By decoupling the functions of these two planes, network control dynamics can be directly programmed in software with an abstract view of the physical infrastructure. Then, a centralized network controller runs the network intelligence, retains a global view of the network, and makes decisions on policies regarding automated network optimization and management, among others.

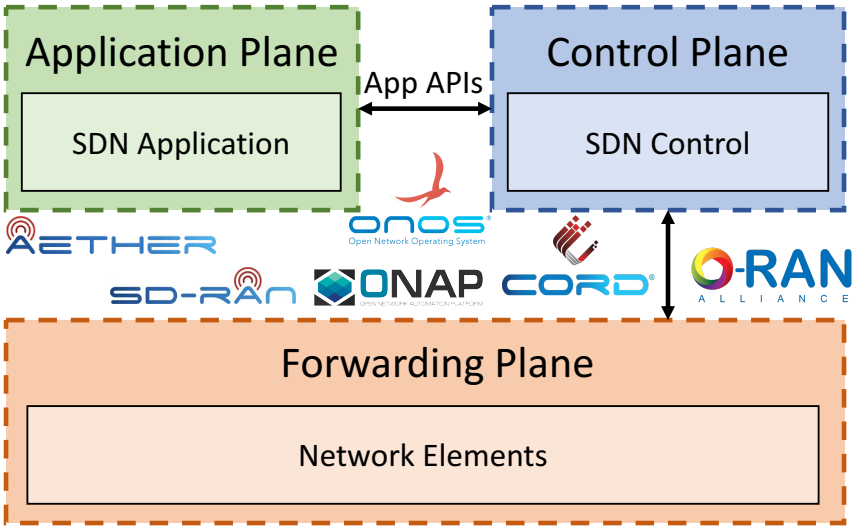


Figure 2.3: High-level overview of the SDN architecture.

The fundamental principle of SDN, namely, the separation of data and control, has been adopted by 5G networks to detach RAN and edge hardware components from their networking and service capabilities. In fact, 5G systems takes softwarization to a broader and comprehensive application range, where it is leveraged to effectively put into practice RAN disaggregation, where RUs operate as basic transceivers and all control and processing operations are performed in software through open interfaces and APIs. This is witnessed by the plethora of open-source SDN solutions for mobile networks, also shown in Figure 2.3, which include Open Networking Operating System (ONOS) [17], Central Office Re-architected as a Datacenter (CORD) [58], O-RAN [11], Open Network Automation Platform (ONAP) [25], Aether [59], and SD-RAN [60].

Network Function Virtualization. NFV brings scalable and flexible management and orchestration to softwarized networks. This is achieved by virtualizing network services and functionalities and decoupling them from the hardware where they are executed. Each functionality is implemented in software via Virtual Network Functions (VNFs), which are executed on Virtual Machines (VMs) instantiated on general-purpose hardware. One of the main advantages of NFV is that each VNF provides atomic functionalities. Therefore, multiple VNFs can be combined together to create more complex and customized network services. Figure 2.4 depicts the main components of the NFV architecture.

These are: (i) the network orchestrator, which instantiates and manages the VMs on the physical infrastructure and the services they run; (ii) the VNFs, which are executed on the VMs and implement the network services, and (iii) the Network Function Virtualization Infrastructure (NFVI),

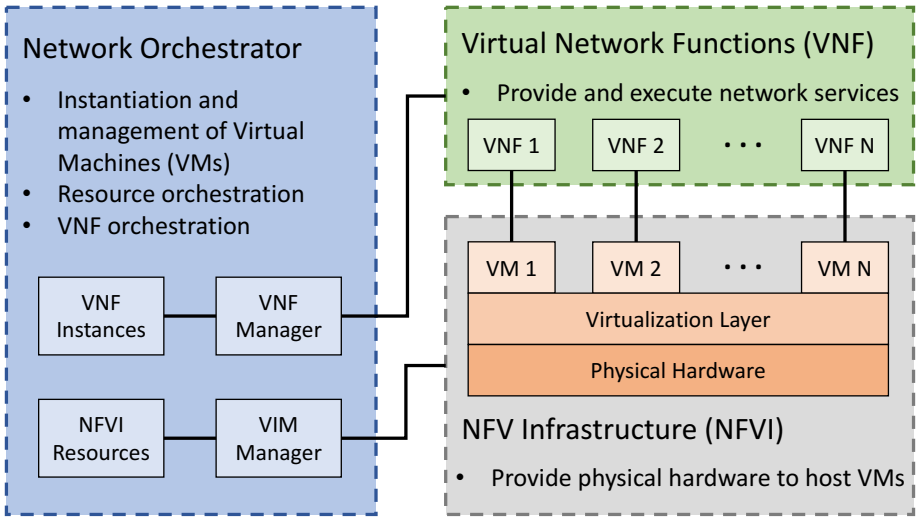


Figure 2.4: High-level overview of the NFV architecture.

which consists of the general purpose physical hardware hosting the VMs deployed by the network orchestrator.

An example of open-source network virtualization project is Open Platform for NFV (OPNFV), which facilitates the adoption and development of a common NFVI [61]. OPNFV also provides testing tools, compliance and verification programs to accelerate the transition of enterprise and service provider networks to the NFV paradigm.

2.1.3 RAN and Core Network Slicing

The whole 5G network design is rooted in softwarization, virtualization and sharing principles. This strategic design choice paved the way toward a new generation of more efficient, dynamic and profitable networks. Such a revolution has also been made possible by the concepts of *network slicing*.

Network slicing is a multi-tenancy virtualization technique where network functionalities are abstracted from hardware and software components, and are provided to the so-called *tenants* as *slices* [13]. The physical infrastructure (e.g., base stations, optical cables, processing units, routers, etc.) is shared across multiple tenants, each of which may receive one or more slices. Each slice is allocated a specific amount of physical resources and operates as an independent virtual network instantiated on top of the physical one. Although tenants have full control over their slices and the resources therein, they cannot interact with other slices, a concept known as *slice isolation* or *orthogonality* [12].

Each slice provides specific functionalities covering the RAN and the core portions of the network. For example, tenants can be granted RAN slices instantiated on selected base stations providing CaaS (e.g., for private cellular networking) to mobile users [62]. They can also instantiate network slices dedicated to specific services, users and applications. Such a flexible approach makes it possible to instantiate slices dedicated to resource-hungry applications, such as virtual and augmented reality,

while simultaneously controlling another slice carrying low-priority traffic generated by browsing activities. An example of practical interest is shown in Figure 2.5, depicting how slicing technologies enable infrastructure sharing and support the instantiation of multiple slices embedding different infrastructure components. The figure also lists relevant and well-established open-source software projects for effective instantiation, control and configuration of network slices in different portions of the infrastructure (see also Sections 2.4 and 2.5).

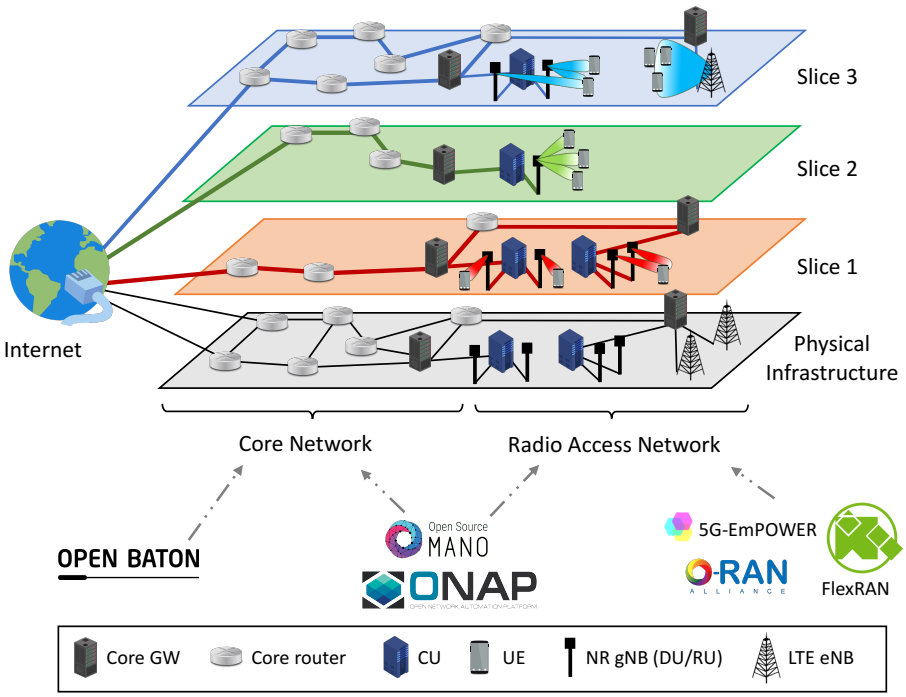


Figure 2.5: An example of RAN and CN slicing.

The benefits of network slicing include: (i) each slice can be reserved to handle specific traffic classes with diverse security requirements and is allocated with a different amount of resources, thus enabling *service differentiation* at the infrastructure level; (ii) slicing is controlled by software components, which enable real-time and on-demand instantiation, reconfiguration, and revocation of network slices to adapt to time-varying traffic demand and/or fulfill Service Level Agreements (SLAs), and (iii) underutilized resources can be leased to Mobile Virtual Network Operators (MVNOs) in the form of network slices, thus maximizing resource utilization and generating new profit opportunities for infrastructure providers [63].

In this context, it is worth mentioning how Operations Support Systems (OSSs) and Business Support Systems (BSSs) will play a vital role in the 5G ecosystem and will determine the success of network slicing. Network slices must be orchestrated, instantiated and revoked dynamically, must satisfy SLA agreements and should be robust against failures and outages. OSSs will serve as a tool to guarantee the fulfillment of services by facilitating closed-loop control and management of network slices. At the same time, operators and infrastructure owners providing slicing services to verticals must generate diversified offers with slices dedicated to services that reflect verticals'

needs. BSSs will be necessary to control such a diversified environment and to implement dynamic billing and pricing mechanisms for each slice. Examples of initiatives confirming this trend are Open APIs and Open Digital Architecture led by the industry association TM Forum [64], ONAP [65] and the open-source project OpenSlice [66]. Since these benefits affect both business and performance aspects of 5G networks, slicing has become pivotal to 5G systems. In this context, the open-source community has led to the development of a variety of solutions to integrate slicing algorithms into the 5G ecosystem [22].

Section 2.4 will survey the most relevant and well-established open-source projects enabling the delivery and handling of network slicing technologies for network RAN and core.

2.1.4 Multi-access Edge Computing

5G systems will leverage advanced and high-performance signal processing and transmission techniques for the highest data rates and QoS possible. However, these technologies alone are not enough to meet the stringent throughput and latency requirements of many 5G applications. For instance, tactile applications as those for virtual and augmented reality, rely upon near real-time processing and interaction with the environment. To be effective these technologies require sub-millisecond transmission times [67]. Furthermore, they involve a significant amount of computation (e.g., augmented reality processors hinge on GPU acceleration), which would result in excessive delay and poor user experience if performed in data centers from a distant cloud. With network technologies of the past meeting the strict constraints of these practical applications was almost considered an utopian task. 5G networks, on the other hand, leverage a simple, yet effective, approach to network architecture design based on softwarization principles to bring services and functionalities to the edge of the network [20].

In this context, Multi-access Edge Computing (MEC) (the technology formerly known as Mobile Edge Computing) has been identified as the solution of the needed *edgification* process. MEC introduces an innovative design shift where essential components of the architecture (both hardware and software) are moved closer to users. By building on edge computing, content caching, NFV and SDN, MEC provides an effective solution to the latency and throughput demands of 5G applications [68]. MEC (i) moves content and functionalities to the edge, meaning that data only sporadically needs to traverse the CN, thus resulting in low latency and in an offloaded core, and (ii) enables localized service provisioning such as private cellular networking, Internet of Things (IoT) data collection and processing at the edge for health and environmental monitoring, and augmented reality for education, telesurgery and advanced industrial applications [69]. There have been several proposals on how to enable MEC in 5G networks. Solutions include those from European Telecommunications Standards Institute (ETSI), which defines the term MEC in [70], and from the 3GPP, which has introduced open interfaces (in Releases 15 and 16) to integrate MEC apps with the softwarized 5G core [71]. The interested readers are referred to [72, 73] for analysis and reviews of different MEC architectures. Open-source MEC frameworks and their enablers will be discussed in Section 2.4.

2.1.5 Intelligence in the Network

Another key component of the 5G ecosystem is the application of machine learning and artificial intelligence-based technologies to network optimization [74]. The scale of 5G deployments makes

traditional optimization and manual configuration of the network impossible. Therefore, automated, data-driven solutions are fundamental for self-organizing 5G networks. Additionally, the heterogeneity of use cases calls for a tight integration of the learning process to the communication stack, which is needed to swiftly adapt to quickly changing scenarios.

Learning techniques for 5G networks have been proposed for different applications. Use cases range from forecasting traffic demands to scale CN resources [75], to predicting HARQ feedback [76] to reduce latency in Ultra Reliable and Low Latency Communication (URLLC) flows, to beam adaptation in mmWave vehicular networks [77]. A summary of results from applying deep learning techniques can be found in [78, 79].

Notably, telecom operators have embraced the deployment of machine learning techniques for self-managed and self-optimized networks. The integration of machine learning in real deployments, however, faces several architectural and procedural challenges [80]. This is primarily because real-time network telemetry and data need to be collected and aggregated to allow the data intensive learning operations of training and inference. The previously discussed MEC paradigm has been proposed as an architectural enabler for applying machine learning to networking, with intelligent controllers deployed at the edge of the network and integrated to the RAN. A software-based framework that implements this paradigm is O-RAN [11], which envisions a RAN Intelligent Controller (RIC) interfaced with gNBs and eNBs, for monitoring, learning, and performing closed-loop actuation. We discuss the O-RAN architecture in detail in Section 2.4.1.

2.2 The Radio Access Network

This section describes the open-source libraries and frameworks for 4G and 5G cellular networks to deploy a software-defined RAN. The most relevant of these open-source software frameworks and their features are listed in Table 2.1.

Table 2.1: Open-source RAN software.

RAN Software	eNB	gNB	SDR UE	COTS UE	License	Main Contributor(s)
OpenAirInterface [81]	yes	yes	yes	yes	OAI Public License v1.1	OAI Software Alliance, EURECOM
srsRAN [82]	yes	yes	yes	yes	GNU AGPLv3	Software Radio Systems
Radisys [28, 83]	no	yes, (O-RAN)	no	N/A	Apache v2.0, O-RAN Software License v1.0	Radisys

2.2.1 OpenAirInterface

The OpenAirInterface Radio Access Network (OAI-RAN) [24, 84] provides software-based implementations of cellular base stations, UEs and core network (OAI-CN; see Section 2.3).

The OAI-RAN source code is written in C to guarantee real-time performance, and is distributed under the OAI Public License [85], a modified version of the Apache License v2.0 that allows patent-owning individuals and companies to contribute to the OAI source code while keeping their patent rights. Both base station and UE implementations are compatible with Intel x86 architectures running the Ubuntu Linux operating system. (An experimental version for the CentOS 7 is under

development.) Several kernel- and BIOS-level modifications are required for these implementations to achieve real-time performance, including installing a low-latency kernel, and disabling power management and CPU frequency scaling functionalities.

eNB Implementation. At the physical layer, the eNB can operate in Frequency Division Duplexing (FDD) and Time Division Duplexing (TDD) configurations with 5, 10, and 20 MHz channel bandwidths, corresponding to 25, 50, and 100 Physical Resource Blocks (PRBs). As for the transmission modes, it supports Single Input, Single Output (SISO), transmit diversity, closed-loop spatial multiplexing, Multi-user MIMO (MU-MIMO), and 2×2 Multiple Input, Multiple Output (MIMO). Channel quality reports are sent through standard-compliant Channel Quality Informations (CQIs) and Precoding Matrix Indicatorss (PMIs). Finally, OAI-RAN also supports HARQ at the MAC layer.

In Downlink (DL), OAI-RAN implements synchronization signals used by UEs to acquire symbol and frequency synchronization (Primary Synchronization Signal (PSS) and Secondary Synchronization Signal (SSS)), and channels that carry information on the DL configuration used by the eNB (Physical Broadcast Channel (PBCH)) and on the DL control channel (Physical Control Format Indicator Channel (PCFICH)). The OAI-RAN eNB also implements the Physical Downlink Control Channel (PDCCH), which carries scheduling assignments of the UEs and DL control information, and the Physical Downlink Shared Channel (PDSCH), which transports data intended for specific UEs. Finally, ACKs/NACKs for the data received in uplink from the UEs are sent through the Physical Hybrid ARQ Indicator Channel (PHICH), while broadcast and multicast services are provided through the Physical Multicast Channel (PMCH).

In Uplink (UL), it supports the Physical Random Access Channel (PRACH), which is used by UEs to request an UL allocation to the base station, as well as channels carrying reference signals from the UE to the eNB (Sounding Reference Signal (SRS) and Discovery Reference Signal (DRS)). Data from the UEs to the eNB is carried by the Physical Uplink Shared Channel (PUSCH), while the Physical Uplink Control Channel (PUCCH) is used to transmit UL control information. Modulations up to 64 Quadrature Amplitude Modulation (QAM) and 16 QAM are supported in DL and UL, respectively.

The E-UTRAN stack of the eNB implements the MAC, RLC, PDCP, and RRC layers and provides interfaces to the core network with support for IPv4 and IPv6 connectivity (see Section 2.1.1 for a detailed description of these layers). As for the MAC layer scheduling, OAI-RAN implements a channel-aware proportional fairness algorithm commonly used in commercial cellular networks, as well as greedy and fair round-robin scheduling algorithms.

The eNB can be interfaced with both commercial and open-source EPCs (e.g., OAI-CN and Open5GS; Section 2.3), and with a number of SDRs, including Ettus Research [86] B-series Universal Software Radio Peripherals (USRPs), e.g., USRP B210, and X-series, e.g., USRP X310 (see Section 2.6 for a comprehensive description of compatible hardware platforms). However, to the best of our knowledge, at the time of this writing the eNB application executed over USRPs X-series appears to be less than fully stable. Both Commercial Off-the-Shelfs (COTSs) smartphones and SDRs can be used as UEs. However, OAI privileged the development of the eNB application rather than the UE one, which may result in connectivity issues between the two.

OAI-RAN also includes a simulation environment implementing layer-2 and layer-3 functionalities only, without the need to interface with any external SDR device. Being transparent to layer-1 procedures, the simulation environment provides a useful tool to evaluate the performance of

algorithms and protocols at the upper-layers. Finally, NR-compliant applications for base stations (both for Non Standalone (NSA) and Standalone (SA) modes), UEs and core network (5GC) have recently being released to the community [24, 87].

Sample Use Cases. OpenAirInterface has witnessed recent widespread adoption by both academia and industry. For instance, Kaltenberger et al. leveraged OAI to build a Cloud-RAN Massive MIMO testbed with Remote Radio Units (RRUs) built from commodity hardware [88]. Foukas et al. proposed Orion [4] and FlexRAN [3], two RAN-oriented centralized network virtualization solutions based on OAI. Liu et al. implemented a learning-assisted network slicing solution for cyber-physical systems on OAI [89], while D'Alterio et al. leveraged OAI to prototype and experimentally evaluate the performance of a Unmanned Aerial Vehicle (UAV)-based eNB [535].

Fujitsu is integrating and testing OAI in commercial units of its proprietary infrastructure [90], while WindyCitySDR is leveraging OAI to create low-bandwidth mobile phone data networks [91]. InterDigital and SYRTEM are developing mmWave software solutions and devices based on the OAI implementation [92, 93].

The full potential of an open and softwarized approach to cellular networking is demonstrated in Section 3.2 and in Chapters 5 and 4. Specifically, a softwarized automatic optimization framework with RIC functionalities, called CellOS, is instantiated on a network with eNBs featuring an enriched version of OAI. In the experimental setting 3 eNBs serve a total of 9 UEs (COTS smartphones). Figure 2.6 compares the throughput achieved by the CellOS-driven automatic user scheduling optimization to that achieved by the OAI-RAN proportional-fairness and greedy scheduling algorithms.

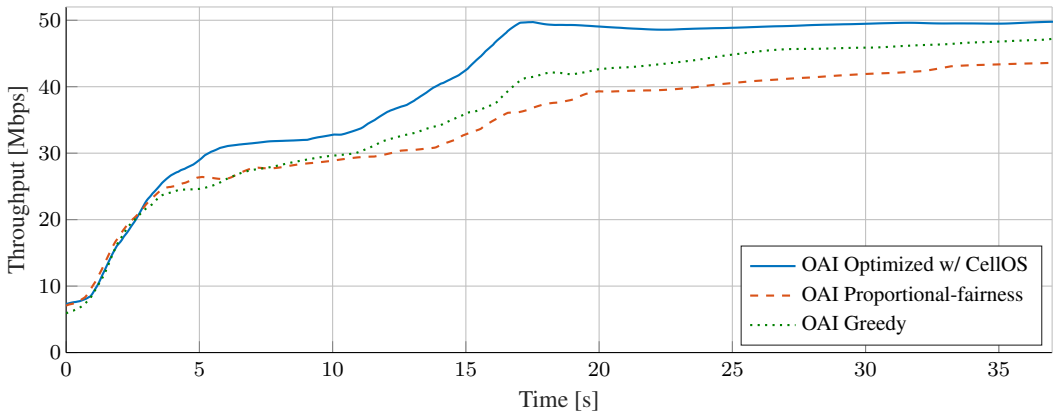


Figure 2.6: Software-defined optimization with OpenAirInterface and CellOS.

The CellOS optimized approach increases the network throughput significantly, and reduces the convergence time to stable high throughput with respect to the other schedulers (see Chapter 5.1). This simple yet effective experiment shows the importance of gaining access and reconfiguring via software network parameters and protocols (the scheduling algorithm, in this case). Without open and programmable software such as OAI, it would have been unfeasible to change scheduling policies baked into hardware components and improve network performance swiftly and automatically.

2.2.2 srsRAN

Similarly to OAI-RAN, srsRAN[3] provides software implementations of LTE eNB, UE, and EPC [27, 95] (discussed in Section 2.3). The software suite is written in the C and C++ programming languages and it is distributed under the GNU AGPLv3 license [96]. srsRAN is compatible with the Ubuntu and Fedora Linux distributions. It does not require any kernel- or BIOS-level modifications to achieve real-time performance (unlike OAI; Section 2.2.1). (Disabling CPU frequency scaling is recommended.)

eNB Implementation. At the physical layer, the eNB implementation supports FDD configurations with channel bandwidths of 1.4, 3, 5, 10, 15, and 20 MHz, corresponding to configurations from 6 to 100 PRBs. The available transmission modes are single antenna, transmit diversity, Cyclic Delay Diversity (CDD), and closed-loop spatial multiplexing. The channels supported in DL and UL are the same of OAI (Section 2.2.1), with modulations up to 256 QAM.

Similar to OAI, the E-UTRAN stack of srsRAN eNB implements the MAC, RLC (TM, AM, and UM modes are supported), PDCP, and RRC layers (Section 2.1.1). The eNB interfaces with the CN through the S1 Application Protocol (S1AP) and GPRS Tunneling Protocols (GTPs) interfaces, and supports IPv4 connectivity. It can be used to serve both COTS and SDR UEs, which can be implemented through srsRAN UE application. Finally, implementations of both SA and NSA NR gNBs have been recently released to the research community.

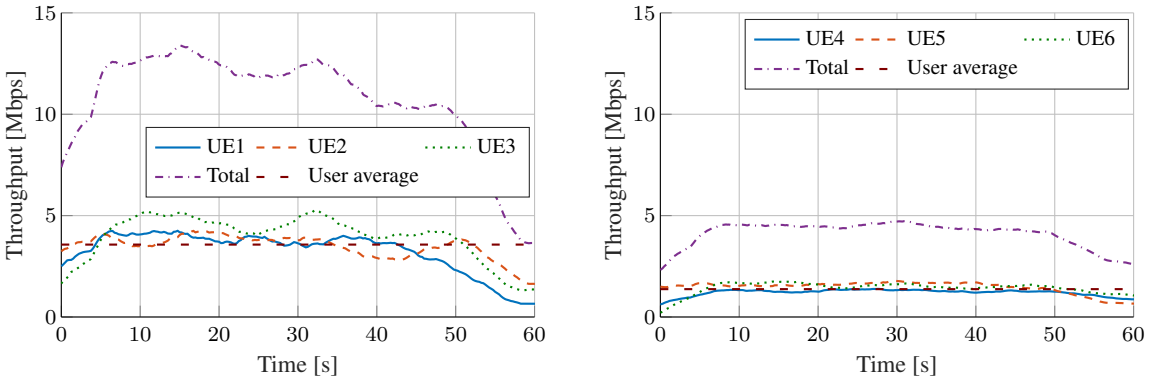
UE Implementation. The UE implementation features PHY, MAC, RLC, PDCP, and RRC layers as the ones of the eNB. Additionally, srsUE also includes a Network Attached Storage (NAS) layer that manages control plane communication between UE and CN, and a Gateway (GW) layer. The latter supports IPv4 and IPv6 connectivity, and is used to create a virtual interface on the machine that runs the user application to tunnel IP packets from/to the RF front-end.

To authenticate users and CN, the UE application supports both soft and hard Universal Subscriber Identity Module (USIM) cards. These are meant to contain values to uniquely identify the UE in the network, such as the International Mobile Subscriber Identity (IMSI), the authentication key (K), the operator code (OP), and the phone number. The soft USIM can work with both the XOR [97] and Milenage [98] authentication algorithms, and the previously mentioned UE authentication values are stored in a configuration file. The hard USIM, instead, requires a physical SIM card that needs to be programmed with the user parameters discussed above through a smart card reader/programmer. This SIM card, then, needs to be connected to the host computer that runs the UE application, for instance, through the same smart card reader/programmer.

As RF front-end, both eNB and UE applications are compatible with several of the boards that will be described in Section 2.6, including USRP B- and X-series (i.e., USRP B210, B205mini-i, and X310), as well as limeSDR [99], and bladeRF [100]. To analyze some of the eNB and UE capabilities in a controlled environment, srsRAN provides utilities to simulate dynamics such as uncorrelated fading channels, propagation delays, and Radio-Link failures between eNBs and UEs. Finally, at the time of this writing, srsRAN is working toward NR compatibility. An initial support of NR at the MAC, RLC, RRC, and PDCP layers is included in the latest releases of the code.

[3] srsRAN was previously known as srsLTE. The renaming happened in April 2021 [94].

Sample Use Cases. Several recent works have been using srsRAN to investigate the security of LTE networks. Bui and Widmer proposed OWL, an srsRAN-based framework to capture and decode control channel of LTE devices [101]. OWL is then leveraged by Meneghello et al. in [102], and Trinh et al. in [103] to fingerprint LTE devices through machine-learning approaches. Kim et al. designed LTEFuzz, a tool for semi-automated testing of the security of LTE control plane procedures [104], while Rupprecht et al. carried out a security analysis of LTE layer 2 [105]. Yang et al. proposed and evaluated SigOver, an injection attack that performs signal overshadowing of the LTE broadcast channel [106]. Singla et al. designed an enhanced paging protocol for cellular networks robust to privacy and security attacks [107]. The National Institute of Standards and Technology (NIST) built OpenFirst on top of srsRAN, a platform for first responders to test and validate LTE technologies focused on public safety communications [108], while Ferranti et al. experimentally evaluated the performance of a UAV-based eNB leveraging srsRAN [538]. Sl-EDGE proposes an optimization-



(a) Slice 1, premium service

(b) Slice 2, regular service

Figure 2.7: Softwarized per-slice QoS differentiation on srsRAN (with SCOPE) using Sl-EDGE and CellOS.

based MEC slicing framework to instantiate slice services on heterogeneous devices at the edge of the network (see Appendix A).

One use case of particular interest is that of RAN slicing. CellOS proposes a RIC which controls resources of the eNBs of the network by interfacing with different RAN software, such as OAI and srsRAN (see Section 5.1). In this case, the source code of srsRAN is extended and included into the SCOPE framework to achieve differentiated service through 5G slicing technologies (see Section 3.2). Figure 2.7, shows experimental results in which 2 eNBs running srsRAN are serving 6 COTS UEs. First, the resource allocation for two slices, serving premium and regular users of the cellular network, is computed through Sl-EDGE (see Appendix A), then this is applied by CellOS (see Section 5.1). The network eNBs allocate 80% of spectrum resources to slice 1 (premium service in Figure 2.7a), and the remaining 20% to slice 2 (regular service in Figure 2.7b). As expected, the throughput of the premium UEs of the slice 1 outperforms that of the regular users of slice 2, with average gains of more than 2.5x.

2.2.3 Radisys Open-source RAN Contributions

Radisys is a 4G/5G vendor that contributes to a number of open-source software consortia, including O-RAN and several Open Networking Foundation (ONF) initiatives [83].

As part of O-RAN (Section 2.4.1), Radisys provides an open-source implementation of the 3GPP NR stack for the gNB DU [109]. At the time of this writing, this does not represent a complete solution that can be deployed to run real-world experiments (as with OAI and srsRAN), as it lacks integration with open-source CU and RU implementations. However, this represents a key first step toward the availability of an open-source 5G gNB based on the CU/DU split principle described in Section 2.1.1.

The currently available open-source code, licensed according to the Apache License v2.0, provides a complete implementation of the MAC and RLC layers. The Radisys release also provides a layer that manages the operations of the DU and interfaces it with the CU, the RU and external controllers, when available. The codebase is aligned with Release 15 of the 3GPP NR specifications. The NR MAC uses the Functional Application Platform Interface (FAPI) to interact with a scheduler, adapted from an LTE implementation. The RLC layer supports the TM, UM and AM modes (see Section 2.1.1 for details on these modes of operation). Additionally, Radisys has open-sourced a full implementation for 4G eNBs, licensed with AGPLv3 [110]. However, this implementation concerns the firmware of a specific Qualcomm chipset FSM9955, thus representing a solution for 4G small cell hardware vendors rather than an alternative to srsRAN and OAI.

2.3 Core Network

In this section, we describe the main open-source solutions for the cellular core networks, i.e., the EPC and the 5G Core. A summary of the solutions discussed in this section is shown in Table 2.2.

Table 2.2: Open-source CN software.

CN Software	EPC	5G Core	License	Main Contributor
OpenAirInterface [81]	yes	under development	Apache v2.0	OAI Software Alliance, EURECOM
srsRAN [82]	yes	no	GNU AGPLv3	Software Radio Systems
OMEC [111]	yes	compatible	Apache v2.0	ONF, Intel, Deutsche Telekom, Sprint, AT&T
free5GC [112]	no	yes	Apache v2.0	free5GC
Open5GS [113]	yes	yes	GNU AGPLv3	Open5GS

2.3.1 Evolved Packet Core

Implementations of the 4G EPC, discussed in details in Section 2.1.1, typically include components for the Mobility Management Entity (MME), the Home Subscription Server (HSS), the Service Gateway (SGW), and the Packet Gateway (PGW).

The MME is responsible for control messages to establish connection with the UEs, paging and mobility procedures. It includes the NAS signaling and security features, as well as tracking area list management, PGW/SGW selection, UE authentication, and reachability procedures. It also takes care of bearer management, i.e., a tunnel between UE and PGW in the case of EPC, and between UE and UPF in the case of 5GC [55, 114]. Moreover, it supports protocols for control plane signaling between EPC and E-UTRAN, reliable message-level transport service. Tunneling protocols for User Datagram Protocol (UDP) control messaging are also provided, as well as protocols for authentication, authorization and charging of UEs.

The HSS implements the user database, and stores information on the subscribers, e.g., identity and key. It is also responsible for user authentication. It provides interfaces for user provisioning in the HSS database, as well as interfaces to connect to the MME.

The SGW and PGW components carry packets through the GTP for both user and control planes, i.e., through the GPRS Tunnelling Protocol User Plane (GTP-U) and GPRS Tunnelling Protocol Control Plane (GTP-C), which use UDP as transport protocol. Packet routing and forwarding, IP address allocation to UEs, and paging are also supported. Open-source implementations of the LTE EPC are provided by OpenAirInterface (with OAI Core Network (OAI-CN)), srsRAN (with srsEPC), Open5GS, and Open Mobile Evolved Core (OMEC).

Table 2.3: Implemented EPC interfaces.

Interface	OpenAirInterface [81]	srsRAN [82]	Open5GS [113]	OMEC [111]
MME / HSS				
S1-MME	x	x	x	x
S6a	x	x	x	x
S10	x	-	-	-
S11	x	x	x	x
SGW / PGW				
S1-U	x	x	x	x
S5 / S8	-	x	x	x
S11	x	x	x	x
SGi	x	x	x	x
Gx	-	-	x	x

A summary of the most relevant 3GPP interfaces implemented by each of these EPC softwares is shown in Table 2.3:

- *S1-MME:* it enables the flow of the S1-AP control application protocol between E-UTRAN and MME.
- *S6a:* it connects MME and HSS, it is used for user authentication and authorization and to transfer user subscriptions.
- *S10:* control interface among different MMEs.
- *S11:* control plane interface between MME and SGW used to manage the Evolved Packet System (EPS).
- *S1-U:* it enables the flow of user plane data between E-UTRAN and SGW.
- *S5/S8:* it provides user plane tunneling management, and control services between SGW and PGW.
- *SGi:* it connects the PGW to the Internet.
- *Gx:* it allows the transfer of QoS policies and charging rules from the Policy and Charging Rules Function (PCRF) to the Policy and Charging Enforcement Function (PCEF) in the PGW.

OAI-CN. OAI-CN is written in the C and C++ programming languages, and it is distributed under the Apache License v2.0 [24]. It is compatible with Intel x86 architectures running the Ubuntu Linux operating system. Kernel modifications similar to those discussed in Section 2.2.1 for OAI-RAN are required to guarantee real-time capabilities. Dynamic QoS with establishment, modification and removal of multiple dedicated bearers, and policy-based QoS update are also features implemented by the OAI-CN MME. Traffic Flow Template (TFT) operations, such as fault detection, filter rules, and IP-filters are also provided. Finally, implicit (e.g., service request failures) and explicit (e.g., bearer resource and delete commands) congestion indicators are supported, along with multi-Access Point Name (APN), paging, and restoration procedures.

srsEPC. The EPC implementation included in the srsRAN software suite, namely, srsEPC, is written in C++ and distributed under the GNU AGPLv3 license [27]. It is compatible with the Ubuntu and Fedora Linux operating systems. The HSS supports the configuration of UE-related parameters in the form of a simple textual csv file. UE authentication is supported by XOR and Milenage authentication algorithms. srsEPC enables per-user QoS Class Identifier (QCI) and dynamic or static IP configurations.

OMEC. This is a high performance open-source implementation of LTE Release 13 EPC developed by the ONF together with telecom operators and industry partners, such as Intel, Deutsche Telekom, Sprint, and AT&T [111]. OMEC is built using a NFV architecture to sustain scalability in large-scale scenarios such as those of 5G and IoT applications. It is distributed under the Apache License v2.0, and offers connectivity, billing and charging features. OMEC can be used as a standalone EPC, or integrated in larger frameworks, such as Converged Multi-Access and Core (COMAC) (see Section 2.4.3).

Sample Use Cases. Sevilla et al. developed CoLTE, a Community LTE project to bring cellular connectivity in rural areas that are not covered by traditional cellular service [115]. CoLTE interfaces commercial eNBs (BaiCellsNova-233) with OAI-CN, modified to include features such as user billing and accounting. CoLTE is currently porting its implementation from OAI-CN to Open5GS. Core network functions for 5G NR, instead, are being developed by bcom starting from the OAI-CN implementation [116]. Moreover, OAI-CN is used as EPC inside the Magma framework (see Section 2.4.3). Haavisto et al. use NextEPC (now Open5GS) to deploy a 5G open-source RAN [16]; Lee et al. interface it with srsRAN to study security concerns of modern cellular networks [117].

2.3.2 5G Core

Two projects that are currently providing an open-source 5GC are free5GC and Open5GS. Both these software implementations support the following relevant 3GPP interfaces:

- *N1/N2:* connect the AMF to the UE and RAN, respectively. They are used for session and mobility management.
- *N3/N4/N6:* connect the UPF to the RAN, SMF, and data network, respectively. They support user plane functions.
- *N8:* connects the Unified Data Management (UDM) and the AMF. It enables user authorization procedures.

- *N10/N11:* connect the SMF to the UDM and AMF, respectively. They handle subscription and session management requests.

- *N12/N13:* connect the Authentication Server Function (AUSF) to the AMF and UDM, respectively. They enable authentication services.

free5GC. This 5GC implementation is distributed under the Apache License v2.0 [112]. It is written in the Go programming language, and it is compatible with machines running the Ubuntu Linux operating system. This implementation, which was initially based on NextEPC (now Open5GS), supports the management of user access, mobility, and sessions (AMF and SMF), and the discovery of the services offered by other network functions (Network Repository Function (NRF)). It also includes network functions to select which network slices to allocate to UEs (Network Slice Selection Function (NSSF)), to manage, store and retrieve user data (UDM and Unified Data Repository (UDR)), to perform UEs authentication within the network (AUSF). Functions for the operation, administration and management of the core network (Operations, Administration and Maintenance (OAM)), and to perform network orchestration, among others, are also included.

Open5GS. This project offers a joint implementation of EPC and 5GC written in C and distributed under the AGPLv3 license [113].[4] It is compatible with a variety of Linux distributions, such as Debian, Ubuntu, Fedora, and CentOS, as well as FreeBSD and macOS. Open5GS is compliant with the Release 16 of 3GPP and, differently from other core networks, it supports the delivery of voice calls through the LTE and NR networks instead of relying on traditional circuit switching networks. This is achieved by leveraging Voice over LTE (VoLTE) and Voice over NR (VoNR) solutions, respectively.

In addition to the EPCs that are evolving toward the 5G architecture (Section 2.3.1), an open-source implementation of the 5GC is currently being developed by Hewlett Packard Enterprise (HPE) [119]. Software implementations of the data plane of the 5GC can also benefit from the acceleration introduced by domain-specific, platform-independent packet-processing languages such as P4 [120], which is a protocol-independent programming language that instructs the networking hardware on how to process packets. Several P4-based UPF implementations have been proposed commercially and in the literature. Most of their software is not open-source [121–123].

2.4 RAN and Core Frameworks

This section describes several open-source frameworks that operate both in the RAN and CN domains. While the software described in Sections 2.2 and 2.3 performs specific functions (e.g., eNB, UE, or CN), the frameworks that will be introduced in the following paragraphs are more general and broad in scope, and interact with individual components in the RAN and CN for control, management, and coordination.

Table 2.4 compares the features, license and availability of the different frameworks and projects that will be presented throughout this section.

[4] Open5GS was previously known as NextEPC [118]. The renaming happened in 2019.

Table 2.4: Open frameworks and projects.

Framework	Main Focus	Status	License	Main Members
		Mobile		
O-RAN [28]	Virtualized, intelligent RAN	available	Apache v2.0, O-RAN Software License v1.0	O-RAN Alliance w/ operators
COMAC [124]	Agile service delivery at the edge	available	Apache v2.0	ONF
SD-RAN [125]	CU/DU control and user planes	available	Apache v2.0	ONF
Aether [126]	5G/LTE, Edge-Cloud-as-a-Service (ECaaS)	under development		ONF
Magma [127]	CN Orchestration	available	BSD	Facebook
OpenRAN [128]	Programmable, disaggregated RAN w/ open interfaces	closed source		TIP
Radio Edge Cloud [129]	O-RAN RIC automated configuration / integration testing blueprint	available	Apache v2.0	Akraino
Aerial [130]	SDK for GPU-accelerated 5G vRAN	early access	proprietary	NVIDIA
		Slicing		
5G-EmPOWER [131]	Centralized controlled for heterogeneous RAN	available	Apache v2.0	FBK (in the framework of multiple EU projects)
FlexRAN [132]	Real-time controller for software-defined RAN	available	MIT License	Mosaic5G Consortium
		Edge		
CORD [133]	Data center for network edge	available	Apache v2.0	ONF, AT&T, Google, Telefonica
LL-MEC [134]	Low-latency MEC and network slicing	available	Apache v2.0	Mosaic5G Consortium
LightEdge [135]	MEC services	available	Apache v2.0	FBK (in the framework of multiple EU projects)

2.4.1 O-RAN

The O-RAN Alliance is an industry consortium that promotes the definition of an open standard for the vRAN, with two goals [11]. The first is the integration of machine learning and artificial intelligence techniques in the RAN, thanks to intelligent controllers deployed at the edge [80]. The second is the definition of an agile and open architecture, enabled by well-defined interfaces between the different elements of the RAN. Since all O-RAN components must expose the same APIs, it is easy to substitute components with others offering alternative implementations of the same functionalities. This allows O-RAN-based 5G deployments to integrate elements from multiple vendors, thus opening the RAN market to third-party entities providing new functionalities and diversified services. Moreover, it makes it possible to adopt COTS hardware, in an effort to promote flexibility and reduce costs. Eventually, following the trend started with cloud-native infrastructures, the O-RAN Alliance also aims at promoting open-source software as part of the consortium effort. In this section, we provide a succint overview of the O-RAN framework and functionalities. Interested readers are referred to [527] for a detailed survey on the O-RAN architecture, open interfaces, algorithms, security, and research challenges.

The O-RAN Architecture. Figure 2.8 illustrates the high-level architecture and the interfaces of O-RAN [11]. With respect to Figure 2.2, O-RAN concerns the edge and the 4G and 5G RANs. It is composed by a non-real-time and a near-real-time RAN Intelligent Controller (RIC), and by the eNBs and gNBs. The service management and orchestration framework (top of Figure 2.8) operates

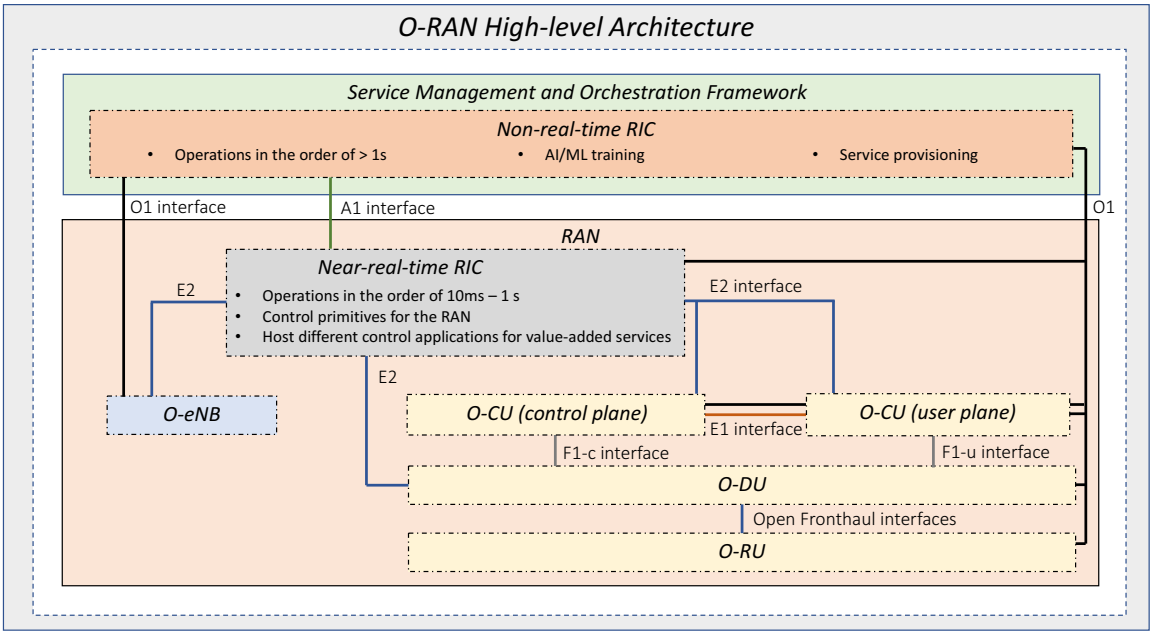


Figure 2.8: O-RAN high-level blocks and interfaces.

a non-real-time RIC, which performs control decisions with a granularity higher than one second. For example, it can provision the different functions of O-RAN, and train learning algorithms over data provided by the RAN, among others. A near-real-time RIC, instead, performs a control loop with a much tighter timing requirement (with a decision interval as short as 10 ms), relying on different start, stop, override, or control primitives in the RAN, e.g., for radio resource management. These APIs can be used by different applications installed on the near-real-time RIC (named xApps), which can be developed by third-party entities and pulled from a common marketplace. For example, through the near-real-time RIC and its xApps, an operator can control user mobility processes (e.g., handovers), allocate networking resources according to predicted paths for connected vehicles and UAVs, perform load balancing and traffic steering, and optimize scheduling policies [136]. The near-real-time RIC can also leverage machine learning algorithms trained in the non-real-time RIC. The remaining components of the O-RAN architecture concern the CU/DU/RU into which 5G gNBs are split [137], and the 4G eNBs [39] (bottom of Figure 2.8). The CU is further split into a control plane CU and a user plane CU. Among the different options investigated by the 3GPP, O-RAN has selected split 7-2x for the DU/RU split [138], in which coding, modulation and mapping to resource elements are performed in the DU, and the inverse FFT, the cyclic prefix addition and digital to analog conversion are carried out in the RU. Precoding can be done in either of the two.

O-RAN Interfaces. O-RAN is in the process of standardizing the interfaces between each of the components in Figure 2.8. The two RICs interact using the A1 interface, while the non-real-time RIC uses the O1 interface to interact with the RUs and legacy 4G eNBs. The A1 interface [139] allows the non-real-time RIC to provide (i) policy-based guidance to the near-real-time RIC, in case it senses that its actions are not fulfilling the RAN performance goals; (ii) manage machine

learning models, and (iii) provide enrichment information to the near-real-time RIC, for example from RAN-external sources, to further refine the RAN optimization. The O1 interface, instead, has operation and management functions, and strives at being compatible with existing standards to permit a seamless integration with existing management frameworks (Section 2.5) [140]. For example, it relies on the IETF Network Configuration Protocol (NETCONF) [141] and on several 3GPP-defined APIs. The non-real-time RIC uses O1 to (i) provision management, fault supervision, and performance assurance services; (ii) perform traces collection; (iii) start up, register, and update physical equipment, and (iv) manage communication surveillance services.

The near-real-time RIC exposes the E2 interface [142] to multiple elements, i.e., the CU, the DU and the eNB. This interface only concerns control functionalities, related to the deployment of near-real-time RIC control actions to the nodes terminating the E2 interface, and to the management of the interaction of the RIC and these nodes [143].

The E1 and F1 interfaces comply to the specifications from 3GPP. The E1 interface runs between the control and user plane CUs, and its main functions concern trace collection for specific UEs, and bearer setup and management [144]. The F1 interface operates between the CUs and DUs [145]. It has two different versions, one for the control plane and one for the user plane. F1 transports signaling and data between CUs and DUs, to carry out RRC procedures and PDCP-RLC packet exchange. Finally, the interface toward the RU is developed by the Open Fronthaul initiative inside O-RAN [138]. This interface carries compressed IQ samples for the data plane, and control messages for beamforming, synchronization, and other physical layer procedures.

O-RAN Deployment Options. O-RAN envisions different strategies for the deployment of its architecture in *regional* and *edge* cloud locations and at operator-owned cell sites [136]. Each facility could either run O-Cloud, i.e., a set of containers and virtual machines executing the O-RAN code with open interfaces, or be a proprietary site, which still uses the O-RAN open APIs, but could run closed-source code. Both cases are illustrated in Figure 2.9, which depicts the six different O-RAN deployment combinations indicated in [136].

In Scenario A all the components except the RUs are deployed at the edge of the network, co-located in the same data center that terminates the fronthaul fiber connectivity. Other alternatives foresee the RICs and CUs co-located at a regional cloud facility, with DUs and RUs at the edge or on cell sites. The preferred deployment solution, however, is Scenario B, which deploys the RIC in the regional cloud, CUs and DUs at the edge, and only the RUs in the operator cell sites [136].

The O-RAN Software Community. Besides concerning standardization activities, the O-RAN Alliance has established a *Software Community* in collaboration with the Linux Foundation for contributing open-source 5G software that is compliant with the O-RAN specifications [11]. The first two O-RAN releases date November 2019 (Amber release [146]) and June 2020 (Bronze release [147]), and feature contributions from major vendors and operators, including AT&T, Nokia, Ericsson, Radisys and Intel. These releases include Docker containers (which will be discussed in Section 2.5.1) and the source code for multiple O-RAN components:

- The non-real-time RIC, with the A1 interface controller, and the possibility of managing machine learning and artificial models in the RAN.

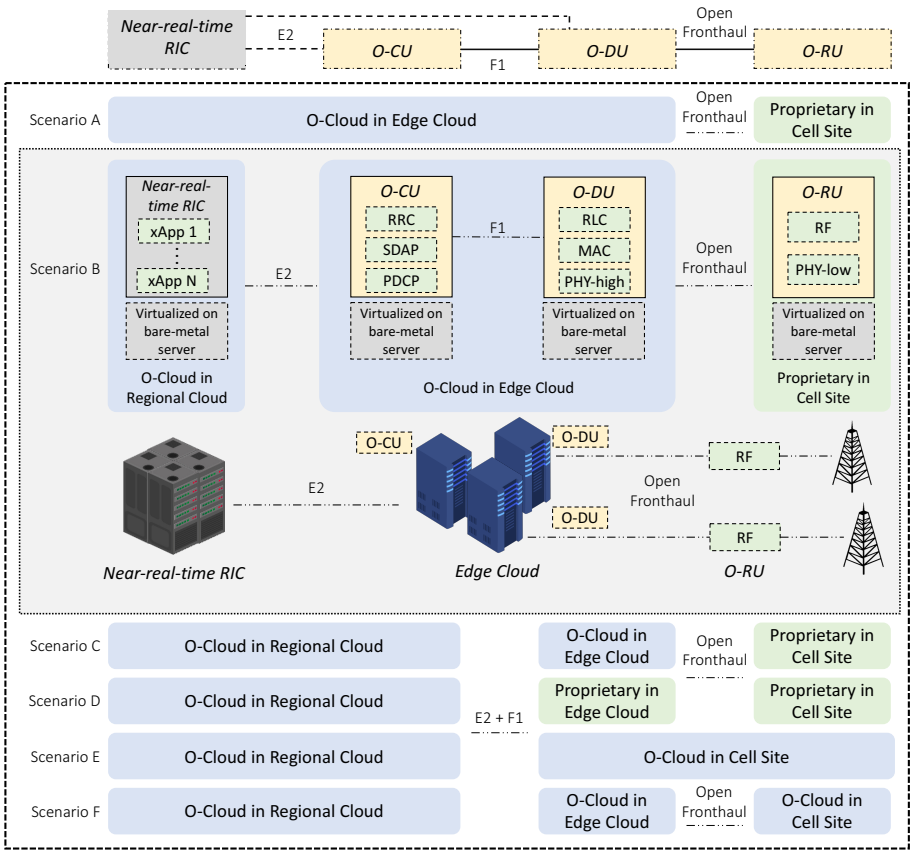


Figure 2.9: Logical (top) and physical (bottom) deployment options for O-RAN.

- The platform for the near-real-time RIC, with multiple applications, such as admission control, UE manager, performance and measurement monitor, which talk to the DU through the E2 interface.
- The DU, as previously discussed in Section 2.2.3, and an initial version of the fronthaul library.
- A framework for operation, administration, and maintenance, and the virtualization infrastructure.

Moreover, a simulator has been developed to test the functionalities of the different interfaces. Other releases, e.g., D (July 2021) and E (December 2021) continue these efforts adding further support to interfaces, controllers, and RAN elements. Interest readers are referred to [527] for a comprehensive survey on O-RAN.

Expected Use Cases. The full realization of the O-RAN vision will revolutionize not only the modus operandi and business of telecom operators [136], but also the world of those scientists, researchers and practitioners that will be able to run a modern, open-source, full-fledged 5G control infrastructure in their lab and investigate, test and eventually deploy all sorts of algorithms (e.g., AI-inspired) for cellular networks at scale. Researchers will be able to deploy and use the open-source software

provided by the O-RAN community to develop, test, and evaluate real-time RAN control applications. The O-RAN open-source suite will enable 5G networking in a standardized environment, thus allowing reproducibility and easing future extensions. Moreover, deploying third-party xApps in the RIC (in collaboration with telecom operators) could enable experimentation running directly on O-RAN-compliant cellular networks. These tests could start in labs to be then they scaled up to larger deployments in the operator network, using the RICs of both scenarios as a *trait d'union*.

At the time of this writing, however, O-RAN is not at production-level. Therefore, future releases will attempt to complete integration of the different RAN components with the RICs. A parallel development effort is also being led by the SD-RAN project [125], aiming at an open-source, 3GPP-compliant RAN integrated with the O-RAN RIC and interfaces. According to [60], the reference code will include the DU and CU, interoperable with third-party RUs, and a near-real-time RIC based on ONOS [17].

2.4.2 Open Networking Foundation Frameworks

The Open Networking Foundation (ONF) is a consortium of several telecom operators that contribute open-source code and frameworks used for the deployment of their networks. Specific examples include OMEC (Section 2.3.1) and the aforementioned SD-RAN [60] and ONOS [17]. The ONF generally distinguishes between *Component Projects*, which are frameworks and/or software that serve a specific purpose, and *Exemplar Platforms*, which combine several Component Projects in a deployable, proof-of-concept reference design. The different projects developed by the ONF are characterized by modular design, facilitating the integration of component projects, and providing the means to incorporate new open-source projects. Additionally, some of them (e.g., ONOS and Trellis) integrate P4 packet processing pipelines. Table 2.5 summarizes the dependencies across different Component Projects and Exemplar Platforms.

Table 2.5: ONF frameworks interactions.

	Aether [126]	COMAC [124]	CORD [133]	SD-RAN [125]	OMEC [111]	ONOS [17]
Aether [126]	-	x	x	x	x	x
COMAC [124]	x	-	x	x	x	x
CORD [133]	x	x	-	-	-	-
SD-RAN [125]	x	x	-	-	-	x
OMEC [111]	x	x	-	-	-	-
ONOS [17]	x	x	-	x	-	-

Component Projects. The ONF currently overlooks the development of 10 open-source component projects, concerning SDN, transport networks, programmable networking hardware, and mobile networks [148]. In the last category, besides OMEC and SD-RAN, notable efforts include CORD [58], which is an open-source project (also part of the Linux Foundation portfolio) for deploying and managing edge cloud facilities for the MEC (Section 2.1.4). The CORD framework is based on multiple software solutions that, together with reference hardware design, realize a reference MEC architecture based on SDN, NFV and cloud-native solutions. CORD aims at (i) reducing deployment costs by using commodity hardware, and (ii) enabling innovative services, thanks to well-defined APIs for accessing edge computing facilities and multi-domain security. Moreover, CORD can be

easily extended to address the heterogeneous requirements of different markets. In particular, two CORD architectures specific for mobile and residential services have been spawned off into two Exemplar Platforms (SDN-Enabled Broadband Access (SEBA) [149] and COMAC [124]). CORD is one of the ONF projects with the largest number of contributions by the open-source community. It includes detailed installation, operation and development guides [150], and a set of repositories with its source code [151].

Another project related to software-defined mobile networks is ONOS [17], an open-source operating system for networking projects. While it has mostly been used for SDN deployments in wired networks, ONOS will provide a common substratum for SD-RAN and several Exemplar Platforms, such as Aether and COMAC, described next.

Exemplar Platforms. An Exemplar Platform is given by extending a Component Project to implement a specific target or by combining and integrating multiple projects that can be deployed as a proof of concept. Among those currently available the following ones concern cellular and mobile networks [152].

- *COMAC,* which extends CORD into a platform that targets the integration of multiple access and CN technologies, including 4G and 5G cellular networks, broadband, fiber and cable networks, and Wi-Fi deployments [124]. The framework provides a common data plane in the core, which aggregates user data to and from different access technologies, and the possibility of managing users' subscriptions and identities with a single management platform. COMAC is based on the SEBA platform (a lightweight multi-access technology platform, which provides high-speed links from the edge of the network to the backbone of the infrastructure), and on multiple Component Projects, such as OMEC, for the mobile core and edge, and CORD for the broadband subscriber management. Moreover, it will exploit O-RAN (with the SD-RAN implementation) for the control plane of the mobile cellular access. The first COMAC release [153] provides instructions on how to configure the different software components to actually set up the overall platform (except for the SD-RAN portion that will be made available in future releases). Additionally, a self-contained COMAC-in-a-Box can be used to install the whole platform on a single server or virtual machine, to run end-to-end tests through an emulated data-plane (based on the OAI simulator, introduced in Section 2.2.1) and the virtualized core and management environments [154].

- *Aether,* for streamlined deployment of private enterprise cellular networks [126]. It combines three main elements, namely, a control and orchestration interface to the RAN, an edge cloud platform (the Aether edge), with support for cloud computing APIs, and a central cloud (the Aether core), for orchestration and management [59]. The Aether project will build and integrate several ONF efforts, including SD-RAN, ONOS, CORD and OMEC. At the time of this writing, the source code and the deployment pipeline are not publicly available. When the code will be released, besides providing an opportunity for private 5G networks, Aether could be effectively used to deploy and manage integrated RAN-edge testbeds for 5G research and innovation.

2.4.3 Other Frameworks and Projects

Along with O-RAN and the ONF solutions, several open-source communities (e.g., from 5G European projects) and companies have released frameworks and projects targeting connectivity, slicing and core-related functionalities. A few noteworthy examples are presented next.

5G-EmPOWER. 5G-EmPOWER [155] is an operating system for heterogeneous RAN architectures. It consists of an open-source and reprogrammable software platform abstracting the physical RAN infrastructure and providing high-level APIs to control RAN functionalities. The code of the platform is released under the Apache License v2.0 [156].

5G-EmPOWER embraces the SDN philosophy to decouple control and data planes. This separation is obtained in practice via two main components, i.e., a centralized controller and a set of agents. The centralized controller (i) acts as an operating system with complete visibility of the physical infrastructure and its functionalities, and (ii) orchestrates the agents' actions via control directives sent through the OpenEmpower protocol [155]. In turn, agents (i) run on each network element; (ii) abstract the underlying RAN-specific protocol implementations (e.g., LTE, Wi-Fi) to the controller, and (iii) modify the underlying protocol parameters according to the controller's directives.

5G-EmPOWER currently supports several mobile Radio Access Technologies (RATs) such as LTE via srsRAN, Wi-Fi, and LoRa. The 5G NR is not supported yet. Integration of diverse RATs is obtained through agents embedding specialized *wrappers*, one for each RAT. While the general architecture of the agent is RAT-independent, the wrapper is RAT-specific. For instance, new RATs (e.g., 5G NR) can be integrated by implementing new wrappers. Despite the current lack of support for 5G NR, 5G-EmPOWER already integrates relevant 5G-related technologies such as RAN slicing. Specifically, it provides software components that allow the instantiation of customized and isolated RAN slices on top of a shared physical infrastructure. Each RAN slice is created from a Slice Descriptor specifying SLAs and users belonging to each slice. A slice resource manager and a hypervisor are, then, in charge of admitting/revoking RAN slices and provisioning them with a certain amount of resources necessary to meet the corresponding SLA.

FlexRAN. FlexRAN leverages abstraction and softwarization technologies to develop a RAT-independent RAN management platform [3]. FlexRAN embraces SDN principles to decouple control and data planes. The control plane is orchestrated by a real-time centralized controller, which controls a set of agents, one for each network element. FlexRAN implements a set of REST APIs in JSON format describing the northbound interface of FlexRAN. These APIs are used by the agents to interface with base stations, thus enabling control of the protocol stack and functionalities of the base stations (i.e., MAC, RRC, PDCP).

FlexRAN directly interfaces with OAI. As such, it does not support 5G NR communications yet. However, the northbound REST APIs can be used to specify and reconfigure slicing policies and requirements, providing support for 5G technologies such as RAN slicing. The FlexRAN code is available upon request and released as part of the Mosaic5G project under MIT license [157].

Magma. Magma is a framework developed by the Facebook Connectivity initiative for simplifying the deployment of cellular networks in rural markets [127]. Notably, its goal is to avoid dependence on a specific access technology (i.e., cellular or Wi-Fi) or on a generation of 3GPP core networks.

Moreover, it avoids vendor lock-in for telecom operators, while offering advanced automation and federation capabilities. The latter is particularly relevant in rural and under-developed scenarios, as it allows the pooling of resources from multiple network operators. Magma is made up of three main components:

- *An access gateway,* which interfaces the access network to the CN. The current Magma release supports an LTE EPC, and has been tested as termination point for the S1 interface of some commercial LTE base stations (see sections 2.2 and 2.3 for more details).

- *A cloud-based orchestrator,* which monitors the operations of the wireless network and securely applies configuration changes. It exposes an analytics interface providing control and traffic flow information.

- *A federation gateway,* which is a proxy between the Magma core running in the access gateway and the network operator 3GPP-compliant core. This proxy exposes the 3GPP interfaces to the different CN components, thus bridging the local mobile deployment with the operator backbone.

LL-MEC. LL-MEC is an open-source MEC framework for cellular systems compliant with 3GPP and ETSI specifications [158]. This framework merges SDN, edge computing and abstraction principles to provide an end-to-end platform where services requested by mobile users are executed on edge nodes of the network. LL-MEC consists of two main components: the Edge Packet Service controlling core network elements (e.g., routers and gateways) via OpenFlow APIs, and the Radio Network Information Service interfacing the data plane and physical RAN elements (e.g., eNBs) via the FlexRAN protocol [3]. Aside from MEC capabilities, LL-MEC supports network slicing for differentiated services applications with diverse latency and throughput requirements. The LL-MEC code is available upon request and released as part of the Mosaic5G project under Apache License v2.0 [157].

LightEdge. LightEdge is a MEC platform for 4G and 5G applications compliant with ETSI MEC specifications [159]. LightEdge allows network operators to provide MEC services to mobile users through cloud-based applications. The framework provides a Service Registry summarizing services and applications registered to the MEC platform. LightEdge also includes modules and libraries for real-time information exchange across applications and services, and to perform traffic steering to and from the cellular network. LightEdge supports multiple eNBs and is compatible with several open-source projects such as srsRAN, Open5GS, and srsEPC. The LightEdge code is available from the project repository under the Apache License v2.0 [160].

OpenRAN. The Telecom Infra Project (TIP) OpenRAN project aims at developing fully programmable RAN solutions that leverage general purpose hardware platforms and disaggregated software [161, 162]. Albeit being closed source, OpenRAN implements open interfaces among the various elements of an NR-enabled cellular network, such as CU, DU, and RU (see Figure 2.2). One of the projects spawned from OpenRAN is OpenRAN 5G NR, for defining a white-box gNB platform [128, 163]. This platform allows equipment manufacturers to build flexible 5G RAN solutions for seamless multi-vendor support.

Akraino REC. Akraino Radio Edge Cloud (REC) is a blueprint to support and meet the requirements of the O-RAN RIC (Section 2.4.1) [129, 164, 165]. It is part of the Telco Appliance blueprint family [166]. Its features include automated configuration and integration testing to facilitate the management and orchestration of the virtualized RAN. The blueprint is made up of modular building blocks and provides an abstraction of the underlying hardware infrastructure, allowing O-RAN RIC to run on top of it, and to seamlessly interface with the provided APIs.

NVIDIA Aerial. NVIDIA Aerial is a set of Software Development Kits (SDKs) that allows to build Graphics Processing Unit (GPU)-accelerated software-defined, cloud-native applications for the 5G vRAN [130]. At the time of this writing, Aerial provides two main SDKs: cuBB and cuVNF.

- *The cuBB SDK* provides a highly-efficient 5G signal processing pipeline that runs PHY-layer operations directly on a GPU to deliver high throughput.
- *The cuVNF SDK* provides optimized input/output and packet placement in which 5G packets are directly sent to the GPU from compatible Network Interface Cards (NICs). Packets can be read and written directly from the NIC to the GPU memory bypassing the Central Processing Unit (CPU), thereby reducing latency. This SDK also allows developers to implement additional VNFs, e.g., deep packet inspection, firewall and vRAN.

2.5 Open Virtualization and Management Frameworks

Besides RAN and CN software, virtualization and management frameworks have an important role in the management and deployment of end-to-end, carrier-grade networks. Several communities and consortia have led the development of open-source frameworks that have been deployed at scale by major telecom operators for the management of their physical and virtual infrastructure [29, 167, 168].

ETSI has defined a set of common features that an NFV Management and Orchestration (MANO) framework should have, mainly for orchestrating network functions [169, 170]. Figure 2.10 depicts where these NFV components fit in the 5G ecosystem. Table 2.6 summarizes the main differences between the frameworks that will be described later in this section, i.e., ONAP, Open Source Management and Orchestration (OSM), and Open Baton. NFV orchestrators are in charge of

Table 2.6: Comparison among different VNF orchestrators.

	ONAP [25]	OSM [171]	Open Baton [172]
Community	Linux Foundation w/ operators	ETSI w/ operators	Fraunhofer, FOKUS, TU Berlin
Compliance w/ ETSI MANO	in progress	yes	yes
External APIs	REST APIs (for external controllers, OSS/BSS, etc.)		Java SDK
Network Services	VNFs and PNFs	VNFs and PNFs	VNFs
Infrastructure	Virtual Machines	Virtual Machines	Virtual Machines
	Containers w/ Kubernetes and Docker	Containers w/ Kubernetes	Containers w/ Docker

provisioning network services, i.e., combinations of physical and virtual network functions that can be chained together with a specific topology, managing their creation and life-cycle [173]. Notably, during the initialization of a network service, a basic configuration (0-day) is applied by default. Then, the MANO framework advertises the actual configuration for the function or

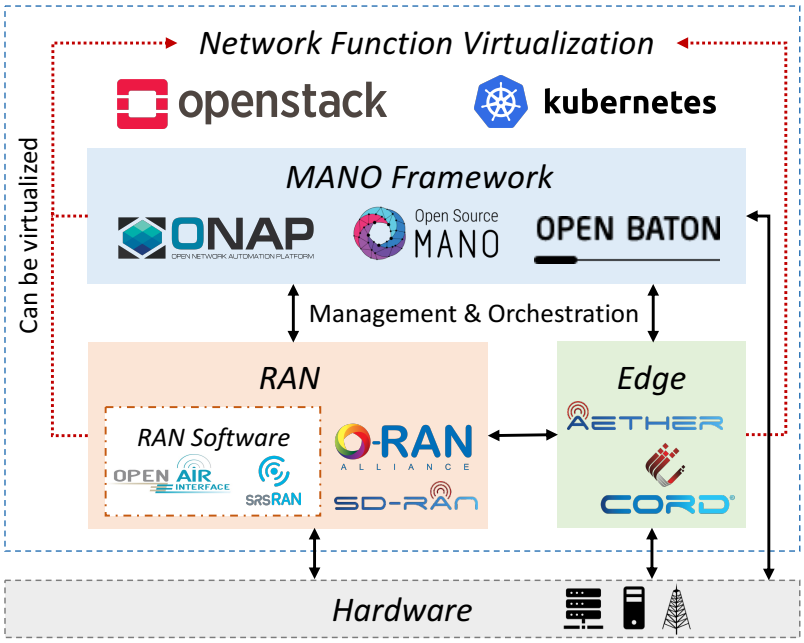


Figure 2.10: High-level relationship among MANO, RAN, and edge frameworks, and virtualization components.

service (1-day). Finally, updates (2-day configurations) can be deployed at a later stage. These operations are performed in concert by the different components of the NFV orchestrator. Following the ETSI architecture [169, 170], an NFV orchestrator is composed (i) of a subsystem that manages the virtualization infrastructure (e.g., Virtualization Infrastructure Manager (VIM) frameworks, such as OpenStack, Kubernetes, and Docker) and the connections to the physical hardware; (ii) of the actual MANO framework, and (iii) of the collection of VNFs that it manages. These frameworks are equipped with southbound and northbound APIs to interact with other cellular infrastructure components, as shown in Figure 2.10, such as edge frameworks for governing the RAN environment (e.g., Aether and CORD), and RAN frameworks (see Section 2.4 for more details). The latter include O-RAN and SD-RAN (described in Sections 2.4.1 and 2.4.2, respectively), which execute functions such as bringing intelligence to the network (Section 2.1.5) and interacting with open-source software, such as OAI and srsRAN (Section 2.2). These, in turn, focus on the CU/DU split introduced by 5G NR, and act as cellular base stations.

In the remainder of this section we discuss virtualization techniques and VIMs (Section 2.5.1) and we describe popular MANO frameworks, such as ONAP, OSM, and Open Baton (Sections 2.5.2, 2.5.3, and 2.5.4, respectively).

2.5.1 Virtualization Techniques

The NFV paradigm decouples the services deployed in a network from the hardware infrastructure on which they run. Applications are packaged into hardware-independent virtual machines, which can be instantiated on different physical machines. This way, NFV eliminates the need for hardware

dedicated to each network function and enables scalability of network services.

NFV, whose high-level architecture is depicted in Figure 2.11, provides many different ways to decouple applications and services, also known as VNFs, from the general-purpose infrastructure on which they run, thus improving scalability and portability. The most common approaches are:

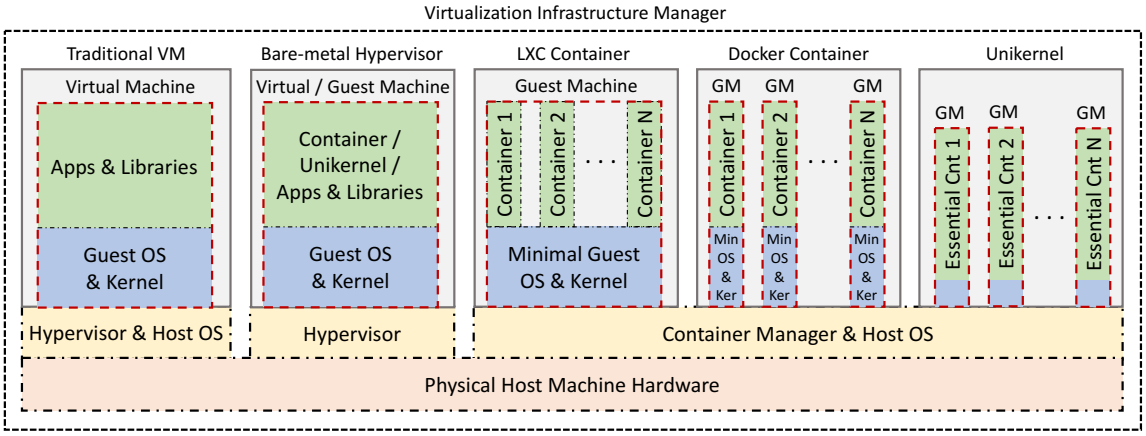


Figure 2.11: High-level NFV architecture.

(i) traditional VMs; (ii) bare-metal hypervisors; (iii) containers, and (iv) unikernels.[5] In the NFV paradigm, a hypervisor can be used to create/run VMs, containers, and unikernels, as well as to manage their resource allocation over the physical hardware. Finally, VIMs are leveraged to control the NFV infrastructure at a higher level.

Traditional Virtual Machines. A traditional VM emulates a computer operating system through a guest operating system and kernel (Figure 2.11). To provide machine-level isolation, the VM requires the virtualization of the hardware of the physical machine on which it runs (called "host machine"). This task is taken care of by the hypervisor, which is *hosted* by the operating system of the physical machine and coordinates resource allocation between host and virtual machines. In general, traditional VMs are considered a resource-heavy approach because of the many hardware virtualization requirements (e.g., virtual disk, CPU, network interfaces, etc.) that are needed to run the VNFs.

Bare-metal Hypervisors. The approach of a bare-metal hypervisor VM is similar to that of traditional VMs, although the hypervisor *directly runs* on the *bare-metal hardware* of the host, without requiring a host operating system (Figure 2.11). Additionally, a bare-metal hypervisor can be used to run and manage a *container or unikernel* (described next) instead of a full-fledged VM.

Containers. Containers are virtual environments that package a specific code and its dependencies to run applications and services in a virtualized way. They are isolated from each other (through *namespace isolation*) and share access to the operating system and kernel of the physical machine

[5] Another virtualization solution that has recently been proposed is that of *serverless computing*, or *Function-as-a-Service (FaaS)*, in which the virtualization platform only spawns computing resources for specifying functions invoked by complex applications, without the need to manage a container or VM. In this chapter, however, we focus primarily on the most common approaches in the NFV domain.

on which they run. They only require a minimal guest operating system instead of the heavy and resource-wasteful hardware virtualization required by VMs (Figure 2.11). Containers can be maintained both by a container manager interfaced with the operating system of the host machine, or by a hypervisor directly running on a bare-metal host machine. The most widespread open-source container virtualization systems are Linux Container (LXC) [174] and Docker [175] (Figure 2.11):

- *LXC* was the first major implementation of the modern containers. It leverages control groups and namespace isolation to create virtual environments with separated networking and process space.

- *Docker* enables the creation of containers, and uses virtualization at the operating system level to deploy them on the physical machine. Differently from LXC, on which it was initially based upon, Docker breaks applications, services, and dependencies into modular units and layers inside each container. Additionally, these layers can be shared among multiple containers, increasing the efficiency of Docker container images. Compared to LXC, Docker containers lack some UNIX functionalities and subsystems.

Unikernels. Unikernels are minimal, lightweight, specialized images built with the sole purpose of running specific applications (Figure 2.11). They compile application services and dependencies into the executable virtual images, without including unnecessary components that would be, instead, included by a generic operating system. This way, unikernels achieve better performance than traditional containers and virtual machines. Since unikernels only include the software components that are needed to run the application of interest, they also improve the security of the system by exposing fewer functionalities that can be attacked by malicious entities.

Examples of unikernel systems are: (i) ClickOS [176], IncludeOS [177] and OSv [178], which focus on high-performance, low-latency, and secure applications; (ii) MirageOS [179], which includes several libraries that are then converted to kernel routines upon image compilation, and (iii) UniK [180], which deals with compilation and orchestration of unikernel images.

Unikernels applications for cellular networks include the following. Wu et al. that integrates Android system libraries into OSv to offload mobile computation for Mobile Cloud Computing (MCC) and Mobile Fog Computing (MFC) [181]. Valsamas et al. propose a content distribution platform for 5G networks that is based on unikernels such as ClickOS, OSv, and MirageOS [182]. A performance comparison of the IncludeOS unikernel and Docker containers instantiated as VNF for 5G applications is carried out in [183].

Hypervisors. A hypervisor is software that creates and runs virtual machines, the *guest machines*, on a physical computer, called the *host machine*. Key tasks of an hypervisor include (i) providing isolation between virtual/guest machine and the host machine; (ii) managing allocation/reallocation of resources, such as CPU, memory and storage, to the guest machines, and (iii) scheduling of resources among host and guest machines.

There are two types of hypervisors: type 1 and type 2 hypervisors. The former are referred to as *bare-metal hypervisors* and manage the guest operating system by running them directly on the host hardware, thus acting as an operating system for the host machine. Examples of hypervisors of type 1 are Xen [184] and VMware ESXi [185]. Type 2, instead, are referred to as *hosted hypervisors* and run on top of the host operating system as a software layer or application. Examples of hypervisors

of type 2 are Linux Kernel-based Virtual Machine (KVM) [186], BSD bhyve [187], and Oracle VirtualBox [188].

Virtualization Infrastructure Managers. A VIM is in charge of control and management of the NFV infrastructure and its resources, such as storage, computation, and networking resources, and coordinates the instantiation of virtual guest machines on the hardware of the physical host machines. The VIM is part of MANO frameworks, such as those described in the remaining of this section. Examples of VIMs are OpenStack [189] and Kubernetes [190]:

- *OpenStack* is a cloud computing software platform capable of controlling a plethora of heterogeneous resources, such as compute, storage and networking resources. Among its very many features, it can act as a VIM, managing the network infrastructure, virtual machines, containers, unikernels, VNF services and applications.

- *Kubernetes* provides automatic deployment, scaling, and management of virtual machines, containers, unikernels, and their applications through a set of primitives. Kubernetes abstracts and represents the status of the system through a series of objects. These are persistent entities that describe the VNF or applications that are running on the Kubernetes-managed cluster, their available resources, and the policies on their expected behavior. In the past few years, a number of projects able to interact with Kubernetes have been launched to solve complex problems at the layers 2 and 3 of the protocol stack. These projects include Istio [191] and Network Service Mesh (NSM) [192]. Istio mesh services carry out traffic management, policy enforcement and telemetry collection tasks. NSM interfaces with Kubernetes APIs to support advanced use cases and facilitates the adoption of new cloud native paradigms. Specifically, it allows network managers to seamlessly perform tasks such as adding radio services, requesting network interfaces, or bridging multiple layer 2 services.

2.5.2 The Open Network Automation Platform

ONAP is an NFV framework developed as a project of the Linux Foundation, with AT&T, China Mobile, Vodafone, China Telecom, Orange, Verizon and Deutsche Telekom as main mobile operator supporters. ONAP is deployed in several commercial cellular networks, and vendors like Ericsson, Nokia, Huawei and ZTE, among others, provide ONAP support and integration in their products [193, 194]. Therefore, ONAP represents one of the most advanced software-based solutions for commercial cellular networks, actively maintained and developed to meet production-level quality standards and satisfy new emerging requirements [195].

ONAP handles the design, creation, and life cycle management of a variety of network services. Network operators can use ONAP to orchestrate the physical and virtual infrastructure deployed in their networks, in a vendor-agnostic way [25]. In addition to common NFV orchestrator functionalities (e.g., automated policy-driven management of the virtualization infrastructure and of the network services), ONAP provides a design framework to model network applications and services as well as a framework for data analytics to monitor the services for healing and scaling. Additionally, ONAP provides a number of reference designs, i.e., *blueprints*. These can be used to deploy the ONAP architecture, depicted at a high-level in Figure 2.12, in specific markets or for specialized use cases (i.e., 5G networks or Voice over LTE deployments). They have been tested in combination with their typical hardware configurations.

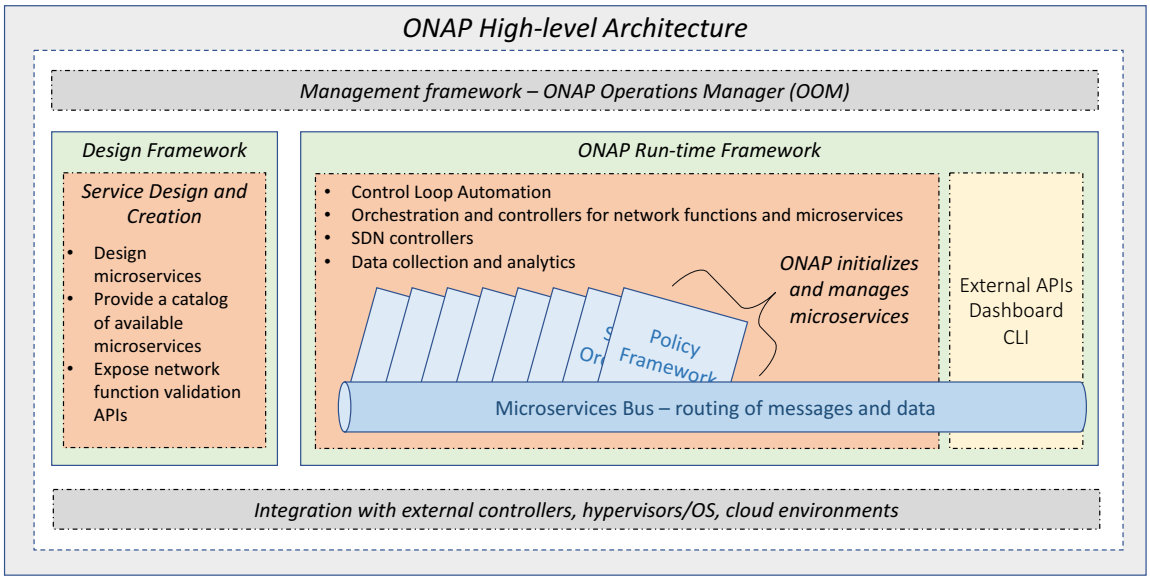


Figure 2.12: High-level architecture of the ONAP framework.

The main components of the ONAP architecture [25], depicted in Figure 2.12, are: (i) the Management Framework; (ii) the Design Framework, and (iii) the Run-time Framework. The management framework, called ONAP Operations Manager (OOM), orchestrates and monitors the lifecycle of the ONAP components. The OOM leverages Kubernetes (Section 2.5.1), and Consul [196], which enables service control, discovery, configuration, and segmentation. Among its functionalities the most noteworthy are: (i) component deployment, dependency manager, and configuration; (ii) real-time health monitoring; (iii) service clustering and scaling, and (iv) component upgrade, restart, and deletion.

The design framework allows to create network services with a declarative modeling language, which makes it possible to specify requirements and functionalities of each service. It allows to model resources, services, products and their management and control functions, through a set of common specifications and policies. Additionally, it includes service design and creation modules for the definition, simulation, and certification of systems assets, processes and policies. Finally, this module provides a database of existing services, and APIs for the validation of network functions.

ONAP run-time framework is made up of several software frameworks for most of its management and orchestration functionalities. In the run-time domain, a microservices bus allows communication and routing of messages and data among the different network functions initialized and managed by ONAP. The run-time framework dispatches and terminates microservices, using an automated control loop, and collects data and analytics from the platform. The run-time component exposes APIs, a dashboard and a command-line tools with a unified interface to control the network infrastructure. Finally, a southbound layer (gray box at the bottom of Figure 2.12) can be used for the integration with external controllers, operating systems and cloud environments, while northbound APIs are offered to OSS/BSS, big data, and other relevant services.

Integration with 5G Networks. Besides representing a general framework for the management and orchestration of mobile networks, ONAP offers some key features that are relevant to 5G deployments. The main requirements that operators have identified are the need to support a hybrid infrastructure, with both physical and virtual appliances, edge automation, with the cloud geographically distributed in different edge locations, and real time analytics, which would enable closed-loop automation.

The Dublin release (June 2019) has introduced a first iteration on the implementation of a 5G blueprint. The most noteworthy feature is the joint discovery of virtual and physical network functions. As previously mentioned (sections 2.1.1 and 2.2) the 5G RAN will not only deal with software components, but will also be in charge of real time signal and RF processing through physical hardware equipment. As such, ONAP also includes discovery and integration procedures to gain awareness of both these *virtual* and *physical network functions*, and to properly manage them. Additionally, the Dublin release includes preliminary support for network slicing. Preliminary evaluation studies for the feasibility of network slicing in ONAP are discussed in [197, 198]. Finally, the ONAP 5G blueprint currently supports a dynamic configuration and optimization of 5G network parameters. This relies on a data collection platform, which transfers in a matter of minutes relevant Key Performance Indicators (KPIs) from the network edge to central processing facilities, and analyzes these data to automatically apply optimizations, or scale resources as needed.

Starting from the Frankfurt release (June 2020), ONAP has been adding support for end-to-end network slicing and service orchestration. Additionally, it allows network designers to define control loops without having to wait for the next official ONAP release. One of the most noteworthy features introduced by the Frankfurt release is the harmonization effort toward O-RAN compatibility through the O1 and A1 interfaces. This effectively aims at defining the specifications for managing the elements of the 5G RAN, such as CU, DU, RU (Section 2.4.1 and Figure 2.8). Future releases will further integrate ONAP and O-RAN, making it easier for telecom operators to deploy an integrated O-RAN/ONAP solution [199].

2.5.3 Open-Source NFV Management and Orchestration

Open Source Management and Orchestration (OSM) is a MANO framework developed by a set of network providers, including Telefonica, BT and Telenor. The community also counts cloud and open-source entities, such as Amazon and Canonical. Similar to ONAP, the OSM framework is well developed and deployed in major cellular networks.

The goals and general architecture of OSM are introduced in [171], and shown at a glance in Figure 2.4. Overall, the framework is an end-to-end network service orchestrator, tailored for deployment in mobile networks. Figure 2.13 describes the OSM architecture and its interactions with the network functions and VIM it manages, following the typical NFV orchestrator structure described in the introduction to this section. The main logical components of OSM are:

- *The Information Model.* It performs the modeling of network functions, services, and slices into templates called packages. This is enabled by a well-defined information model provided by the ETSI MANO framework [169]. Similarly to the design component of ONAP, this allows telecom operators to analyze the requirements of the network and model the resources that need to be deployed for functions, services, and slices.

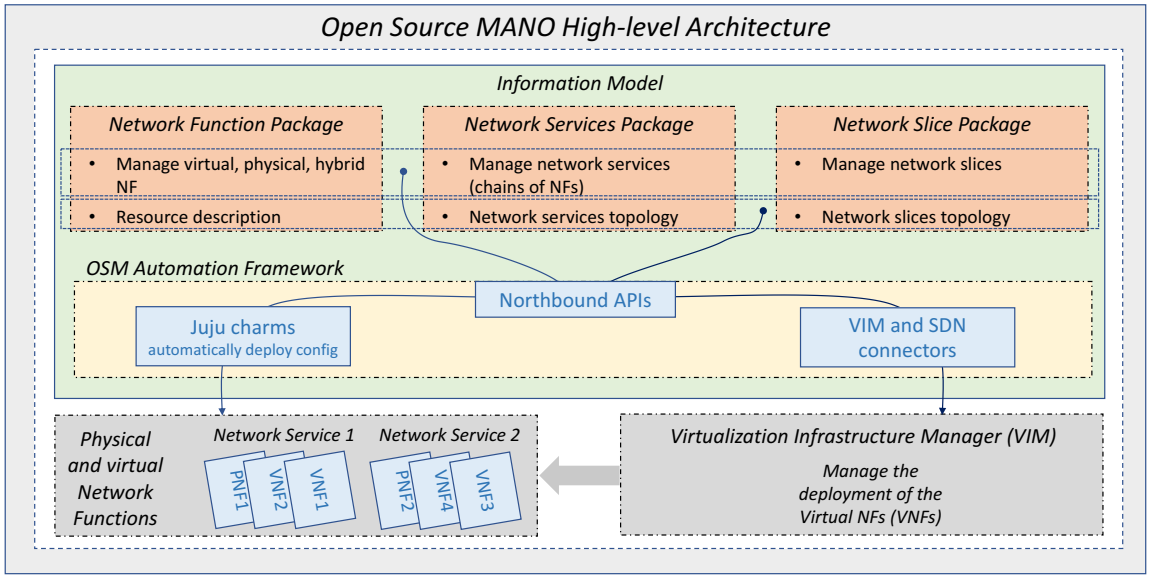


Figure 2.13: High-level architecture of the OSM framework.

- *The OSM Automation Framework.* It automates the life cycle of network services, from instantiation to scaling and, eventually, deletion. This is done by applying the information model to the actual deployed infrastructure, as shown in Figure 2.13, through a northbound interface that the automation framework exposes to the different modeling components. The 0-day, 1-day and 2-day configurations of the actual services and functions are done through Juju Charms [200] (i.e., tools to define, configure, and deploy services on cloud and bare-metal systems).

Similarly to ONAP, OSM has southbound and northbound APIs that can be exploited by other external services, such as other orchestrators and OSS/BSS, respectively [201].

Integration with 5G Networks. OSM has published in December 2018 a 5G-ready release, which added the possibility of managing both virtual and physical network functions, network slicing, and a policy-based closed control loop, and extended the analytics and interface frameworks. Several 5G European projects have used and/or contributed to OSM. Metro-Haul focused on the design of an SDN-based optical transport infrastructure for 5G networks, and developed an OSM component for the management of the infrastructure in a distributed wide area network [202]. Similarly, 5G Tango has developed multiple components for OSM, including an emulator for the virtual infrastructure manager, and network slicing capabilities, while discussing and proposing possible extensions of the MANO concept for 5G into more advanced frameworks [168, 203]. The 5GCity and 5G-MEDIA projects have used OSM as NFV orchestrator for management frameworks of networks for smart cities and media distribution over a Content Distribution Network (CDN), respectively [204, 205]. Finally, 5G-TRANSFORMER has integrated OSM in its network slicing framework for the management of computing resources [206].

The author of [207] investigates how to integrate OSM and OAI-CN, to facilitate the deployment of fully-software-based solutions in testbeds and edge locations. Finally, [208] uses OSM to

experiment with dynamic virtual network function placing in a 5G vehicular scenario.

2.5.4 Open Baton

Open Baton [209] is an open-source project jointly developed by Fraunhofer FOKUS and TU Berlin aimed at providing a modular and reconfigurable framework for the orchestration of network services. The framework focuses on NFV management and is fully-compliant with the ETSI NFV MANO specification. Its source code is available online under Apache License v2.0 [210].

Open Baton provides a full-fledged ecosystem to instantiate and handle atomic VNFs, as well as to compose them to create more complex network services. The framework has been designed to operate over a virtualized infrastructure. For this reason, Open Baton features drivers to directly interface with most VIMs, with specific support for OpenStack [189] (see Section 2.5.1).

Besides VNF orchestration, Open Baton also provides support for multi-tenancy applications through network slicing and MEC [172]. Specifically, Open Baton features a Network Slicing Engine (NSE), a Java-based external software component that interacts with Open Baton via dedicated SDKs. The NSE allows network operators to specify QoS requirements for each network slice (e.g., minimum bandwidth for a target traffic class) in a clean and simple way via minimal JSON or YAML configuration files. Through Open Baton's VIM drivers, these configuration files are, then, dispatched to the VIM, which ultimately guarantees that each slice meets the set QoS requirements.

2.6 Software-defined Radio Support for Open-source Radio Units

The open-source software described throughout this chapter can be mostly executed on commodity hardware, except for the signal processing related to the physical layer, which generally runs on the Field Programmable Gate Arrays (FPGAs) of the SDRs. In this section, we discuss the main SDR solutions compatible with the software suites described in Section 2.2. These platforms are pivotal in enabling researchers to deploy and experiment with end-to-end networks, even though they may not have access to carrier-grade hardware deployed by the major telecom operators.

A summary of the capabilities of each SDR is shown in Table 2.7. There we can find powerful SDRs that can act as rooftop base stations, such as the USRP N310, and cell towers, such as the USRP X310 or arrays of Iris SDRs. Smaller SDR models, such as USRPs B210, bladeRF/2.0 micro, and LimeSDR, instead, are powerful enough to operate as small cells, while, the ultra compact and lightweight USRP B205mini-i can act as a Distributed Antenna System (DAS) node. A description of the capabilities of each of these SDRs will be given in the following paragraphs.

USRP. The Universal Software Radio Peripheral (USRP) are SDR solutions produced by National Instruments/Ettus Research for designing, prototyping and testing wireless protocols and systems [86]:

- *USRP B210:* It is a full-duplex SDR with two transmit receive channels. It covers a frequency range from 70 MHz to 6 GHz with a real-time bandwidth of up to 56 MHz. It connects to the host computer through a USB 3.0 interface, and is compatible with OAI and srsRAN discussed in Section 2.2.

Table 2.7: Capabilities of SDRs and their integration with RAN software.

SDR	TX/RX Channels	Frequency Range	Instantaneous Bandwidth (up to)	RAN Software	Target
bladeRF	1	[300MHz, 3.8GHz]	28 MHz	OAI, srsRAN	DAS node, small cell
bladeRF 2.0 micro	2	[47 MHz, 6 GHz]	56 MHz	OAI, srsRAN	DAS node, small cell
Iris	2	[50 MHz, 3.8 GHz]	56 MHz	OAI	DAS node, small cell, tower
LimeSDR	4 TX, 6 RX	[100 kHz, 3.8 GHz]	61.44 MHz	OAI, srsRAN	DAS node, small cell
USRP B205mini-i	1	[70 MHz, 6 GHz]	56 MHz	srsRAN	DAS node
USRP B210	2	[70 MHz, 6 GHz]	56 MHz	OAI, srsRAN	DAS node, small cell
USRP N310	4	[10 MHz, 6 GHz]	100 MHz	OAI	DAS node, small cell, tower, rooftop
USRP X310	up to 2 (daughterboards)	[DC, 6 GHz] (daughterboards)	160 MHz (daughterboards)	OAI, srsRAN	DAS node, small cell, tower

- *USRP B205mini-i:* It is a full-duplex SDR with a frequency range from 70 MHz to 6 GHz, and an instantaneous bandwidth of up to 56 MHz. Similar to USRP B210, it connects to the host computer through a USB 3.0 interface. It is compatible with srsRAN.

- *USRP X310:* It is an SDR with two daughterboard slots, which enable up to two full-duplex transmit/receive chains. The covered frequency range and instantaneous bandwidth vary according to the specific daughterboard model (from DC to 6 GHz, and up to 160 MHz). This USRP can connect to a host computer through a range of interfaces such as 1 Gigabit Ethernet, dual 10 Gigabit Ethernet, PCIe Express, and ExpressCard. It is compatible with both OAI and srsRAN;

- *USRP N310:* It is a full-duplex networked SDR with four transmit/receive chains. It covers the [10 MHz, 6 GHz] frequency range with an instantaneous bandwidth of up to 100 MHz. It can be connected to a host computer through 1 Gigabit Ethernet, 10 Gigabit, or Xilinx Aurora over two SFP+ ports, and is compatible with the OAI software.

bladeRF. The bladeRF denotes a series of full-duplex SDR devices produced by Nuand [100]. They are available with different form factors and FPGA chips, and are compatible with both OAI and srsRAN. They and connect to the host computer through a USB 3.0 interface.

- *bladeRF:* This SDRs comes in two different configurations, i.e., bladeRF x40 with a 40KLE Cyclone IV FPGA, and bladeRF x115 with a 115KLE Cyclone IV FPGA. Regardless of the specific FPGA chip, the bladeRF SDR has a single transmit/receive chain, which covers the [300 MHz, 3.8 GHz] frequency range with up to 28 MHz of instantaneous bandwidth.

- *bladeRF 2.0 micro:* This model is equipped with two transmit/receive chains and supports 2×2 MIMO operations. It is available with different FPGA chips, i.e., 49KLE Cyclone V FPGA chip (bladeRF xA4), and 301KLE Cyclone V FPGA chip (bladeRF xA9). It covers the [47 MHz, 6 GHz] frequency range with an instantaneous bandwidth of 56 MHz.

LimeSDR. This SDR is produced by Lime Microsystems [99], has four transmit and six receive chains, and supports 2×2 MIMO operations. It covers the [100 kHz, 3.8 GHz] frequency range with an instantaneous bandwidth of up to 61.44 MHz. The LimeSDR is compatible with both OAI and srsRAN. Other versions of this SDR include the LimeSDR Mini, which has two channels instead of

Table 2.8: Testbeds for Cellular Network Experimentation.

Testbed	Technology available	Open-source Software	Framework	Scenario
AERPAW	5G and CR for UASs	RAN & Core / under development		City-scale outdoor
ARA	5G, CR, massive MIMO, mmWave, optical and low-earth-orbit satellite	under development		City-scale outdoor
Arena	5G, CR, massive MIMO	RAN & Core	O-RAN RIC	Large-scale office
Colosseum	5G, CR	RAN & Core	O-RAN RIC	Large-scale network emulator
CORNET	5G, CR	RAN & Core	N/A	Large-scale indoor
COSMOS	5G, mmWave, CR, optical switching	RAN & Core	O-RAN components	Indoor, city-scale outdoor
Drexel Grid	5G, CR	RAN & Core	N/A	Large-scale indoor
FIT testbeds	5G, CR, IoT, NFV	RAN & Core	OSM	Large-scale indoor
IRIS	5G, CR, Wi-Fi, WiMAX, cloud-RAN, NFV, S-band	RAN & Core	N/A	Indoor
NITOS	5G, CR, Wi-Fi, WiMAX	RAN & Core	N/A	Large-scale indoor and outdoor, office
POWDER	5G, CR, massive MIMO, Network Orchestration	RAN & Core	O-RAN RIC	Indoor, city-scale outdoor
5TONIC	5G NFV, network orchestration	N/A	OSM	Data center

four, and the LimeSDR QPCIe, which enables 4×4 MIMO configurations instead of the 2×2 of the standard model. Support for these LimeSDR models has not been explicitly reported by either OAI or srsRAN.

Iris. This is a networked SDR device with two transmit/receive chains, produced by Skylark Wireless [211]. It works in the $[50\ \mathrm{MHz}, 3.8\ \mathrm{GHz}]$ frequency range and supports an instantaneous bandwidth of up to $56\ \mathrm{MHz}$. This SDR connects to the host computer through a 1 Gigabit Ethernet interface, and is compatible with OAI. It can be combined with additional hardware platforms provided by Skylark to boost its performance, while multiple Iris SDRs can be grouped in arrays to enable massive MIMO operations (see Argos [212], and the POWDER PAWR testbed [6, 213] described in Section 2.7).

2.7 Testbeds

In this section we describe a number of testbeds that can be used to instantiate softwarized 5G networks by leveraging the open-source utilities, frameworks and hardware components described in this chapter. An overview of the capabilities of each testbed is given in Table 2.8.

Platforms for Advanced Wireless Research. The objective of the NSF-funded Platforms for Advanced Wireless Research (PAWR) program is to enable experimental investigation of new wireless devices, communication techniques, networks, systems, and services in real wireless environments through several heterogeneous city-scale testbeds [5]. The following are the PAWR platforms awarded so far and that are being built.

- *POWDER.* The combination of the Platform for Open Wireless Data-driven Experimental Research (POWDER) [6, 214] and of the Reconfigurable Eco-system for Next-generation End-to-end Wireless (RENEW) [213, 215] provides a testbed, namely POWDER-RENEW (or

simply, POWDER), which covers an area of $6\ \mathrm{km}^2$ of the University of Utah campus in Salt Lake City, UT. Its objective is to foster experimental research for a range of heterogeneous wireless technologies, including 5G, RAN, network orchestration, and massive MIMO technologies. POWDER is equipped with cutting-edge compute, storage, and cloud resources, as well as state-of-the-art SDRs.

Besides allowing users to install their own software suites, the testbed offers a series of ready-to-use "profiles," which are instantiated on its bare-metal machines. These include open-source RAN software, e.g., OAI and srsRAN, coupled with different EPC solutions, as well as components of the frameworks previously discussed, including the O-RAN RIC. Finally, the POWDER platform has been used to demonstrate automated optimization of 5G networks (see Section 5.1).

- *COSMOS.* The Cloud Enhanced Open Software Defined Mobile Wireless Testbed for City-Scale Deployment (COSMOS) [216–218] is being deployed in the densely-populated neighborhood of West Harlem, New York City, NY. Upon completion, it will cover an area of $2.59\ \mathrm{km}^2$. This testbed focuses on providing ultra-high-bandwidth and low-latency wireless communications, and it will have edge-computing capabilities. Among others, COSMOS will allow researchers to experiment with mmWave and optical switching technologies.

  COSMOS includes the Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT) [219], a platform with a number of USRPs X310, compatible with the open-source RAN solutions described in Section 2.2. The platform served as Open Test and Integration Center during the O-RAN plugfest, a proof of concept for demonstrating the potentials of multi-vendor interoperability of cellular networks [220].

- *AERPAW.* The Aerial Experimentation and Research Platform for Advanced Wireless (AERPAW) [221–223] is the first-ever wireless platform to allow large-scale Unmanned Aerial System (UAS) experimentation for 5G technologies and beyond. AERPAW is being deployed the North Carolina Research Triangle and, upon completion, it will include flying aerial base stations to provide cellular connectivity to ground users.

- *ARA.* The Agriculture and Rural Communities (ARA) platform implements a wireless living lab for smart and connected rural communities. It will deploy edge and cloud equipment in central Iowa, covering an area with a diameter of over $60\ \mathrm{km}$. Upon completion, ARA will enable researchers to experiment low-latency wireless applications in rural environemnts [224].

As publicly funded testbeds, the PAWR platforms will be accessible to the wireless research community for experimental use. As such, each platform implements the fundamental requirements that will ensure reproducibility of experiments, interoperability with other platforms, programmability, open access to the research community, and sustainability.

Colosseum is the world's most powerful wireless network emulator. It is housed at Northeastern University Innovation Campus in Burlington, MA. It is composed of 21 server racks with 256 USRP X310 SDRs, half of which are controllable by experimenters, while the other half is allocated to Colosseum Massive Channel Emulator (MCHEM). Through MCHEM scenarios, researchers can seamlessly emulate entire virtual worlds with up to $65{,}536$ concurrent wireless channels with realistic

characteristics, such as path loss and fading. This way, Colosseum assures the ultimate reproducibility of experiments in the sub-6 GHz spectrum band (see Section 3.4).

Arena is an indoor testbed composed of 24 SDR USRPs (16 USRPs N210 and 8 USRPs X310) stacked up in a radio rack, and controlled by 12 Dell PowerEdge R340 high-performance machines in a server rack. Servers and SDRs are connected through dual 10 Gigabit Ethernet connections to guarantee rapid and low-latency radio control and communication. The USRPs are connected to a grid of 8×8 antennas hung off the ceiling of a 2240 ft^2 dynamic indoor office space through same-length cables that guarantee equal signal delays across the whole testbed. Moreover, Arena SDRs are fully synchronized through 4 Octoclock clock distributors to enable massive MIMO applications, among others. The Arena testbed, which will be discussed in details in Section 3.5, allows researchers to experimentally evaluate wireless protocols and solutions for indoor 5G deployments in an office-like environment.

5TONIC includes data center and equipment for 5G virtual network experimentation [225]. It is composed of a NFV infrastructure with high-performance servers and workstations running network orchestration and virtualization functions and a number of SDR platforms and devices. 5TONIC allows users to run complex NFV and orchestrations frameworks such as OSM. 5TONIC has been used for NFV MANO [226] and mmWave applications [227].

FED4FIRE+ is a Horizon 2020 [228] project to foster experimentally-driven research in the future Internet ecosystem [229]. Among others, it includes a number of *federated* testbeds for wireless, 5G, IoT, cloud, and big data applications. Below, we describe the testbeds of the FED4FIRE+ project targeting open-source cellular networks research and compatible with the OAI and srsRAN RAN software tools (Section 2.2).

- *NITOS* is a Future Internet Facility composed of an outdoor, an indoor, and an office testbed with both SDRs and commercial nodes [230]. The outdoor testbed comprises nodes with Wi-Fi, WiMAX, and LTE capabilities, while the indoor and the office testbeds are made up of of Icarus Wi-Fi nodes [231] deployed in an isolated environment. NITOS has been used for MANO [232], 5G distributed spectral awareness [233], and MEC applications [234, 235], among others.

- The *IRIS* testbed focuses on Cloud-RAN, NFV, and SDN experimental research [236]. The testbed includes a number of ceiling-mounted SDR devices supporting Wi-Fi, WiMAX, and 4G/5G technologies, as well as S-band transceivers.

COgnitive Radio NETwork (CORNET) is a testbed of 48 SDR nodes deployed in a four-story building in the Virginia Tech campus (Blacksburg, VA) that enables experiments on dynamic spectrum access and Cognitive Radio (CR) research [237]. CORNET allows users to perform 5G experimental research by leveraging open-source software, such as OAI and srsRAN, or by emulating cellular signals through COTS equipment. Among other applications, CORNET has been used to evaluate link adaptation in cellular systems [238].

Future Internet of Things (FIT) is a French project for large-scale testbeds for wireless communications [239]. It includes: (i) FIT Wireless, which targets indoor Wi-Fi, CR, and 5G applications (through OAI, for instance); (ii) FIT IoT-Lab, which concerns IoT-related experimentation, and (iii) FIT Cloud, which supports the other two by enabling SDN and NFV research through OSM, among other frameworks.

Drexel Grid is a testbed made up of 24 SDR devices (20 USRPs N210 and 4 USRPs X310) hung off the ceiling of a dedicated indoor room to evaluate diverse 5G and CR wireless technologies [240,241]. The USRPs X310 of the testbed can be used with open-source RAN software such as OAI and srsRAN (Section 2.2). Additionally, this testbed includes a channel emulator with simulated nodes to evaluate wireless systems in a controlled and repeatable environment.

2.8 Softwarized 5G: Limitations and Road Ahead

This chapter aimed at investigating how the "softwarization of everything"—that is pervasive to current trends in computing, communication and networking—got its way into cellular networks, and in particular how it has not only revolutionized their fourth generation, but has also established a radically new way, both technical and commercial, to usher in the 5G era successfully.

Our overview focused on the most recent advances in the open-source and reprogrammable 5G ecosystem. We listed and discussed a variety of heterogeneous, yet modular software and hardware components. In particular, we illustrated how their expected evolution is key to transitioning from the traditional black box approach of cellular network management to those white box principles that will bring both research and industry communities to swift innovation, shorter time-to-market and overall higher customer satisfaction.

By way of conclusion, we finally intend to point out that despite operators, vendors and scientists are paying considerable attention to the new software-defined technologies described in this chapter, these solutions are not ready for prime time on commercial 5G networks just yet. Indeed, the road to celebrate this marriage needs overcoming a few show stoppers, which we describe below.

- **Keep pace with the standards.** The cellular network community faces constant pressure to keep up with the specifications/technologies being introduced by new communication, networking and even programming standards. A notable example are the NR and mmWave communication technologies introduced as 5G enablers by 3GPP and are currently being deployed in *closed source* commercial networks. The RAN software libraries described in Section 2.2 have not yet completed the development of the support for NR, as such task requires a considerable effort in terms of coding and testing. In this domain, open-source network simulators have, so far, provided a more controlled development environment for the development and assessment of 5G solutions [242–245]. The testing of real-world 5G software, especially for mmWaves, is indeed extremely complex due to the lack of accessible open hardware for the software to run, which precludes testing important components, such as beam management. The platforms described in Section 2.6 are optimized for carriers below 6 GHz, even though early prototypes of open mmWaves boards are currently being developed [246–249]. Similarly, most of the testbeds described in Section 2.7 focus on sub-6 GHz deployments, with only a few (e.g., COSMOS) considering an extension to mmWaves.

The road ahead for the development of high-band 5G software-defined solutions, therefore, lies in a more concerted, joint software development effort, and in hardware platforms that can keep up with the requirements of the software community. Furthermore, the current lack of a mature code base for 3GPP NR RAN software libraries hinders the development of more advanced features even in the sub-6 GHz spectrum, such as URLLC support (e.g., with mini slots [49]) and multi-connectivity [36].

- **Latency and scalability issues.** The scalability of an SDR-based system, both in terms of processing and computing requirements, depends on the number of signal processing operations, which are generally proportional to the available bandwidth [250]. As the next generation wireless systems will deal with larger and larger bandwidths for higher data rates [36], the implementation of the radio stack and its software processing chains will have to be extremely efficient, robust, and count on powerful and reliable hardware. Moreover, considering the tight latency and throughput requirements of many 5G use cases, the integration of software and hardware needs to be seamless, to deliver the best possible performance. In this regard, although virtualized solutions add unprecedented flexibility to the network, they also come with new challenges. Specifically, these solutions rely upon resource sharing (which limits and regulates resource utilization among different processes [251]), and virtualization requires additional interactions between the virtualized and bare-metal environments [67]. Together, these aspects introduce additional latency which might not be tolerable for many 5G application and services.

- **Limited contributions for RAN open-source software.** The same large telecom operators and vendors driving the development of open-source CN and MANO frameworks are not showing the same level of attention to RAN-related projects. RAN efforts have indeed mostly come from academia or from smaller companies, with limited manpower and resources. As some sophisticated digital signal processing and implementations of the lower layers of the RAN stack are often source of intellectual property and product-bearing revenues for telecom businesses, major vendors and operators are not encouraged to release their solutions as open-source. Recognizing this limitation, the OpenAirInterface Software Alliance has licensed the OAI RAN implementation with a permissive license, which allows contributors to retain intellectual property claims (see Section 2.2.1). Additionally, the O-RAN Alliance is moving encouraging first steps toward an openly softwarized RAN (see Section 2.4.1), even though the current development efforts do not include also an open-source software for the radio front-ends. Therefore, the wireless community should aim at increasing the support toward the development of complete and Open RAN and radio software libraries, increasing the number of active contributors to the currently available open-source RAN projects.

- **Lack of robust, deployable, and well-documented software.** As of now, most of the frameworks and libraries described in Sections 2.2-2.5 cannot be used in actual networks, as their open-source component is either incomplete, requires additional integration and development for actual deployment, or lacks robustness. Moreover, to reach the quality of commercial solutions, the open-source community should aim at delivering well-documented, easy-to-deploy, and robust software, specifying all dependencies and additional software components that guarantee the correct and efficient functioning of the system. For example, the container-driven

development model as used in cloud-native computing could be adopted to simplify and expedite the software deployment process.

- **Need for secure open-source software.** Heightened attention to software development following best practices for robustness and security is sorely required [252], to guarantee privacy, integrity, and security to the end users of softwarized networks. Openness already facilitates useful scrutiny of the code. Audits and reviews from the open-source community can help prevent bugs and/or security holes, whose existence needs to be responsibly disclosed to the project maintainers [253]. Appropriate security, especially "by design," however, is still lacking. The exposure of APIs to third party vendors (e.g., for the RIC apps), for instance, could introduce new vulnerabilities in the network, in case the APIs are not properly securely designed, and contain weaknesses that can be exploited by attackers. It is clear that the security of the open-source software that will be eventually deployed in 5G and beyond systems must be a key concern for the developers and telecom ecosystem. The wireless community, thus, should follow the best practices developed over the years by other open-source communities (e.g., the Linux kernel), that constantly make it possible to tighten the security of open-source products [254].

All these road blocks are currently preventing, or considerably slowing down, the widespread and painless application of several of the softwarized solutions presented in this chapter. It is now the task of the wireless research and development community to transform these challenges into the opportunity to innovate further in the direction of truly realizing open, programmable, and virtualized cellular networks.

Chapter 3

OpenRAN Gym: Data Collection and Experimentation in O-RAN

The next generations of cellular networks will follow the Open RAN paradigm, which promotes openness, virtualization, programmability, and data-driven control loops in the mobile environment (see Chapter 2). This will help network operators support new bespoke services on the same physical infrastructure, thanks to the flexibility and reconfigurability of software-based deployments and algorithmic control. Open RAN will also decrease operational costs because of the increased efficiency enabled by virtualized and open ecosystems.

In this context, the O-RAN Alliance has developed specifications to apply the Open RAN paradigm to prevailing radio access technologies including 3GPP LTE and NR networks [255]. O-RAN specifications introduce standardized interfaces that connect new, O-RAN-specific network nodes to key RAN elements, such as the NR CUs, DUs, RUs, and the LTE O-RAN-compliant eNBs [527]. To enable programmatic closed-loop control of the RAN, O-RAN also introduces two so-called RICs. The near-real-time (or near-RT) RIC is connected through the E2 interface to the RAN network functions (i.e., the CU and the DU) to enable control loops that operate at timescales between 10 ms and 1 s [256]. The non-real-time (non-RT) RIC, instead, is part of the service management and orchestration framework and operates at timescales larger than 1 s [257]. This component connects to one or multiple near-RT RICs through the A1 interface, used to distribute policies, external information, and to manage Artificial Intelligence (AI) and Machine Learning (ML) models. These models define intelligent network control strategies that are then run on the RICs in the form of applications—namely, xApps and rApps—that can be provided by RAN vendors, operators or third-party entities. xApps run in the near-RT RIC, while rApps run in the non-RT RIC. Finally, the Service Management and Orchestration (SMO) connects to the RAN, and to the O-RAN elements of the RAN, through the O1 interface, used for management and orchestration, and to the O-RAN virtualization platform (the O-Cloud) via the O2 interface.

The open and disaggregated O-RAN architecture enables the practical deployment of AI/ML solutions at scale. For instance, AI/ML algorithms can perform inference and traffic forecasting or configure RAN nodes based on run-time conditions and traffic requirements. The O-RAN specifications [258] discuss the typical workflow for the development and testing of AI/ML in the RAN [528]. This involves multiple steps, including: (i) *data collection*, to create rich datasets to

capture the characteristics of the environment where the AI/ML solution will be deployed (e.g., wireless channel, user distributions and requirements) as well as various indicators of network performance under different configurations; (ii) *AI/ML model design*, with a selection of the model inputs and outputs, and *training and testing*, to understand its limits and effectiveness; (iii) *model deployment* as xApps or rApps; (iv) *model fine-tuning* with live data from the RAN, to adapt it to the production environment, and, finally, (v) the actual *inference, forecasting and/or control* of the RAN.

In this chapter, derived from [528, 529, 532, 542, 545, 547], we give an overview of the main software tools and experimental wireless platforms developed and used throughout this work. OpenRAN Gym, an open toolbox for data collection and experimentation with AI/ML in O-RAN is described in Section 3.1. SCOPE, an open and softwarized prototyping platform for NextG systems is described in Section 3.2. ColO-RAN, the first-of-its-kind open, large-scale, experimental O-RAN framework for training and testing AI/ML solutions for next-generation RANs is detailed in Section 3.3. Colosseum, the world's largest wireless network emulator with hardware-in-the-loop is described in Section 3.4. Arena, an SDR-based indoor testbed for sub-6 GHz spectrum research is described in Section 3.5. Finally, conclusions are drawn in Section 3.6.

3.1 OpenRAN Gym Overview

In this section, we introduce OpenRAN Gym, an open toolbox to develop O-RAN-compliant AI/ML solutions, deploy them as xApps on the O-RAN near-real-time RIC, and test them on large-scale softwarized RANs controlled by the RIC. We first give an overview of OpenRAN Gym and its core components, discussing how they enable design and testing workflows of ML-based xApps. Then, we demonstrate how two xApps designed with OpenRAN Gym can be used to control a large-scale RAN instantiated on the Colosseum wireless network emulator through the SCOPE framework (see Section 3.2), and managed by an O-RAN near-real-time RIC provided by the ColO-RAN framework (see Section 3.3). Finally, showcase how OpenRAN Gym experiments can be ported to heterogeneous experimental wireless platforms and testbeds. Specifically, we demonstrate how solutions developed on Colosseuem can be ported to the Arena testbed (see Section 3.5), and to the POWDER and COSMOS platforms of the PAWR program (see Section 2.7). To the best of our knowledge, this is the first open toolset for the end-to-end design and experimentation of data-driven O-RAN xApps on large-scale experimental platforms.

Previous experimental work has focused on the development of data-driven solutions and xApps for specific use cases [259, 260], on the description of the AI/ML capabilities of O-RAN [261, 262], on interoperability testing [220], and on orchestration [546]. Compared to the state of the art, OpenRAN Gym enables an end-to-end workflow for the design and testing of AI/ML solutions as xApps in the O-RAN ecosystem. By doing so, it empowers users with a first-of-its-kind open and publicly-available O-RAN-compliant toolbox that will unleash the potential of data-driven applications for next generation cellular networks. OpenRAN Gym aims at creating a thriving community of researchers and developers contributing to it with open source software components for experimental O-RAN-driven AI/ML research.[1]

The remainder of this section is organized as follows. Section 3.1.1 details the building blocks of the OpenRAN Gym architecture. A practical description of OpenRAN Gym data collection and

[1] The software components of OpenRAN Gym are publicly-available and accessible at `https://openrangym.com`

control framework, and O-RAN control architecture is given in Sections 3.1.2 and 3.1.3. Section 3.1.4 presents the xApp design and testing workflow, and provides an example of large-scale RAN control using xApps developed with OpenRAN Gym on Colosseum. Finally, Section 3.1.5 showcases exemplary results obtained by instantiating OpenRAN Gym on different experimental wireless platforms.

3.1.1 OpenRAN Gym Architecture

The architecture of OpenRAN Gym is shown in Figure 3.1. It includes: (i) one or multiple publicly-

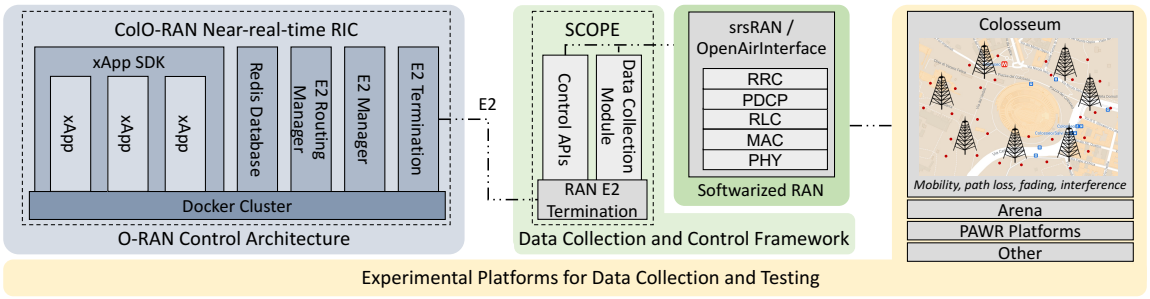


Figure 3.1: OpenRAN Gym architecture.

accessible *experimental platforms* for data collection and testing (e.g., Colosseum, Arena, the platforms of the PAWR program [5]); (ii) a *softwarized RAN* (e.g., implemented through srsRAN or OpenAirInterface); (iii) a *data collection and control framework* with APIs to control the cellular stacks and extract statistics from them (e.g., SCOPE), and (iv) an *O-RAN control architecture* with interfaces to connect to the RAN and control it through AI/ML solutions (e.g., ColO-RAN). The platform-independence of OpenRAN Gym allows users to collect data, design and train solutions in heterogeneous environments before deploying them on production networks. In this way, several evaluation and fine-tuning iterations can be performed in controlled setups to guarantee that the final AI/ML model behaves as expected.

At the time of this writing, OpenRAN Gym supports Arena, Colosseum, and the POWDER and COSMOS testbeds of the PAWR program as experimental platforms. In the current version of OpenRAN Gym, the softwarized RAN is based on srsRAN [27], which implements the full stack of a 3GPP eNB and controls the USRPs X310 of the Standard Radio Nodes (SRNs) that act as radio front-ends. The srsRAN protocol stack is extended by the SCOPE framework, which enhances it with additional networking and control functionalities. As discussed in Section 3.2, this component leverages the emulation capabilities of Colosseum—which acts as a *wireless data factory*—to enable automated, large-scale data collection campaigns to create datasets with tens of hours of RAN experiments and in different wireless and traffic conditions [526, 528, 543]. Finally, the O-RAN control architecture is implemented through ColO-RAN, which extends and adapts the O-RAN Software Community (OSC) near-real-time RIC to the Colosseum environment, and connects SCOPE-enabled base stations to the near-real-time RIC through the O-RAN E2 interface. As discuss in Section 3.3, ColO-RAN provides an instance of an O-RAN-compliant near-real-time RIC, together with an *xApp SDK* that allows users to design and test AI/ML-based xApps (Figure 3.1).

Table 3.1: Main SCOPE slicing and scheduling configuration parameters.

Name	Description
`network-slicing`	Enable/disable network slicing
`slice-allocation`	Define the base station slice allocation
`slice-scheduling-policy`	Set the scheduling policy for each slice
`slice-users`	Assign UEs to slices

3.1.2 Data Collection and Control Framework

OpenRAN Gym leverages SCOPE as the data collection and control framework. SCOPE—described in Section 3.2—provides a development environment to design, prototype and test adaptive solutions for cellular networking, and for large-scale data collection of RAN Key Performance Measurements (KPMs). It builds on top of srsRAN [27], and extends it with novel functionalities (e.g., network slicing and additional scheduling policies), open APIs to control the RAN configuration at run time (e.g., the resources allocated to each slice), and data collection capabilities. SCOPE has powerful data collection capabilities when used in combination with Colosseum, which makes it possible to automatically perform large-scale data collection campaigns in realistic wireless environments. Examples include data collection campaigns with tens of hours of experiments, and in setups with up to 49 nodes (7 base stations and 42 users) in sliced RANs with different QoS requirements, scheduling policies, slicing resources, and in different emulation scenarios [526, 528, 543]. SCOPE has been extended to incorporate a RAN-side E2 termination (based on the OSC DU [263]) to connect to the O-RAN near-real-time RIC. This allows xApps running on the RIC to interface with the SCOPE APIs and control the functionalities of the base stations at run time.

In the remaining of this section, we will show how to configure the main parameters of the SCOPE base stations, and how to start SCOPE on Colosseum. It is worth mentioning that even if we specifically focus on the implementation for the Colosseum network emulator (provided as a ready-to-use container image, namely `scope-e2`), these procedures can be ported to different platforms with minor adaptations.

3.1.2.1 Starting SCOPE

SCOPE allows users to configure the base stations through JSON files. The main parameters to set up the network slicing and scheduling functionalities (see Table 3.1) are as follows.[2]

- `network-slicing`: this parameter enables/disables the network slicing capabilities of the SCOPE base station.
- `slice-allocation`: sets the Resource Block Groups (RBGs) of each slice. It takes as input {`slice:[first_rbg, last_rbg],...`}, e.g., {`0:[0,3],1:[5,7]`} assigns RBGs 0-3 to slice 0 and RBGs 5-7 to slice 1.
- `slice-scheduling-policy`: sets the scheduling policy the base station uses for each slice, e.g., `[2,0]` assigns policy 2 to slice 0 and policy 0 to slice 1. The possible values correspond

[2] A comprehensive description of all parameters of the SCOPE APIs configuration files can be found at `https://github.com/wineslab/colosseum-scope`.

to the scheduling policies supported by SCOPE (0 for round-robin, 1 for waterfilling, 2 for proportionally fair).

- `slice-users`: assigns UEs to the slices. It takes as input {`slice:[ue1,ue2],...`}, e.g., {`0:[2,5],1:[3,4]`} assigns UEs 2, 5 to slice 0, and UEs 3, 4 to slice 1. The UE ID corresponds to the i-th SRN allocated to the experimenter (the base station is assumed as the first SRN).

After saving the above parameters in a JSON-formatted configuration file[3] (e.g., named `radio.conf`), experiments can be started through the commands shown in Listing 3.1. This command, executed on

```
#!/bin/bash
cd radio_api/
python3 scope_start.py --config-file radio.conf
```

Listing 3.1: Commands to start SCOPE applications.

the SRNs assigned to the user, takes care of starting base station, core network and UEs applications.

At experiment run time, the SCOPE APIs can be used to reconfigure the base station, e.g., to modify the amount of RBGs allocated to each slice, or their scheduling policy (see [542, Section 3.3]). Finally, RAN KPMs are automatically logged by the base stations and saved into CSV-formatted files. These files can either be used on-the-fly (e.g., to perform online inference or model training) or retrieved at a later time from Colosseum data storage (e.g., for offline training or data processing).

3.1.3 O-RAN Control Architecture

The O-RAN control architecture component used by OpenRAN Gym is provided by ColO-RAN, an open development environment to design, train and test data-driven O-RAN-compliant solutions at scale [528]. ColO-RAN—detailed in Section 3.3—offers a minimalist implementation of the OSC near-real-time RIC, that can be instantiated on Colosseum through Docker containers, as well as scripts for the automated deployment of the RIC components. ColO-RAN includes containers for the O-RAN *E2 termination, E2 manager* and *E2 routing manager* that handle the communications within the RIC and with the RAN nodes (e.g., SCOPE base stations), a *Redis database* that records the RAN nodes connected to the RIC, and an *xApp SDK* (Figure 3.1). The latter provides software tools to design and test AI/ML-based xApps for run-time RAN inference and/or control.

At a high level, ColO-RAN xApps are made of two building blocks, shown in Figure 3.2: (i) the *Service Model (SM) connector*, which handles the communications to/from the near-real-time RIC (e.g., to communicate with the base stations), ASN.1 message encoding/decoding, and queries to the RIC Redis database, and (ii) the *data-driven logic unit* that performs tasks based on KPMs received from the RAN at run time, e.g., traffic prediction and/or control of the base stations. Notice that at the time of this writing, ColO-RAN xApps use custom SMs. Standard-compliant SMs are part of our future work.

In the remainder of this section, we will show how to instantiate the ColO-RAN near-real-time RIC on Colosseum (Section 3.1.3.1), connect the SCOPE base station to the RIC through

[3] An example of a SCOPE configuration file can be found at `https://github.com/wineslab/colosseum-scope/blob/main/radio_api/radio_interactive.conf`.

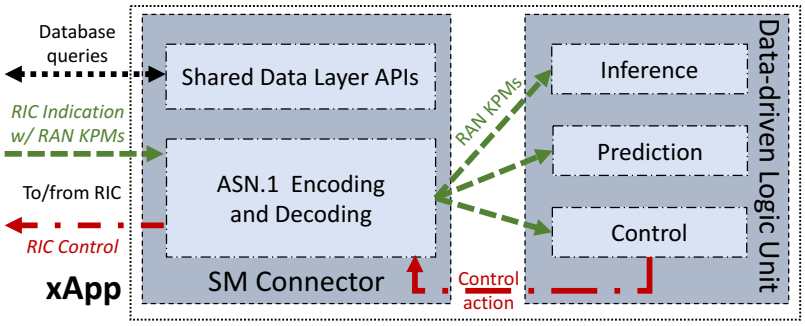


Figure 3.2: ColO-RAN xApp.

the O-RAN E2 termination (Section 3.1.3.2), and start a sample xApp that interfaces with the base station (Section 3.1.3.3). Even though we focus on the implementation for the Colosseum environment (provided in a ready-to-use container image, namely `coloran-near-real-time-ric`), these procedures can be ported to different platforms with minor adjustments.

3.1.3.1 Starting the ColO-RAN Near-real-time RIC

The Docker images of the ColO-RAN near-real-time RIC can be built and started as containers through the provided `setup-ric.sh` script and the commands of Listing 3.2. This script (adapted from [264]) takes as argument the network interface used by the RIC to communicate with the RAN (e.g., the `col0` interface in Colosseum). First, base Docker images, used to build the RIC images, are

```
#!/bin/bash
cd setup-scripts/
./setup-ric.sh col0
```

Listing 3.2: Commands to set up the ColO-RAN near-real-time RIC.

imported. Then, the four images composing the near-real-time RIC are built, and their IP addresses and ports (defined in the `setup-lib.sh` script) are configured. These images include: (i) `e2term`, which is the endpoint of the E2 messages on the RIC ("E2 Termination" in Figure 3.1); (ii) `e2mgr`, which manages the messages to/from the E2 interface ("E2 Manager"); (iii) `e2rtmansim`, which leverages the RIC Message Router (RMR) protocol to route the E2 messages inside the RIC ("E2 Routing Manager"), and (iv) `db`, which maintains a database of the RAN nodes connected to the RIC ("Redis Database"). After the building process completes, the Docker images are initialized as containers on a Colosseum SRN, and listen for incoming connections from RAN nodes implementing an E2 termination endpoint. The container logs can be read through the `docker logs` command, e.g., `docker logs e2term -f` shows the logs of the E2 termination (`e2term`) container.

3.1.3.2 Connecting the SCOPE Base Station to ColO-RAN

After the ColO-RAN neat-RT RIC is started following the steps of Section 3.1.3.1, the SCOPE base station (set up in Section 3.1.2.1) can be connected to it. To this aim, the SCOPE base station runs an instance of the O-RAN E2 termination, which we adapted from the OSC DU implementation [263].

After connecting to the near-real-time RIC, this component can exchange messages with the xApps running therein. Specifically, the E2 termination of the base station can: (i) receive *RIC Subscription* messages from the xApps; (ii) send periodic RAN KPMs to the xApps through *RIC Indication* messages; (iii) receive *RIC Control* messages from the xApps, and (iv) interface with the SCOPE APIs to modify the scheduling and slicing configurations of the base station based on the received xApp control.

Listing 3.3 shows the steps to initialize the E2 termination on the SCOPE base station. The E2

```
1 #!/bin/bash
2 cd colosseum-scope-e2/
3 ./build_odu.sh clean
4 ./run_odu.sh
```

Listing 3.3: Commands to build and start the SCOPE E2 termination process.

termination is first built through the `build_odu.sh` script of line 3, which also sets the IP address and port of the near-real-time RIC to connect to, as well as the network interface used for the connection to the RIC. Then, the E2 termination can be started through the `run_odu.sh` script (line 4), which initializes the E2 termination and connects it to the near-real-time RIC. The successful connection of base station and near-real-time RIC can be verified by reading the logs of the `e2term` container (through the command `docker logs e2term -f`, see Section 3.1.3.1). This log shows the association messages between the RIC and the base station, together with the ID of the connected base stations (e.g., `gnb:311-048-01000501`).

3.1.3.3 Initializing a Sample xApp

After starting the near-real-time RIC, and connecting the SCOPE base station to it, the sample xApp provided as part of ColO-RAN can be initialized. This can be done through the `setup-sample-xapp.sh` script and the commands shown in Listing 3.4. The script takes as input the ID of the RAN node the

```
1 #!/bin/bash
2 cd setup-scripts/
3 ./setup-sample-xapp.sh gnb:311-048-01000501
```

Listing 3.4: Commands to build the ColO-RAN sample xApp Docker image, and to start and configure the xApp container.

xApp subscribes to (e.g., the base station), which can be read in the logs of the ColO-RAN `e2term` Docker container (see Section 3.1.3.2). It then builds the Docker image of the sample xApp, and starts it as a Docker container on the near-real-time RIC.

After the xApp container (named, for instance `sample-xapp`) has been started, the xApp process can be run with the commands shown in Listing 3.5. By running these commands, the xApp

```
1 #!/bin/bash
2 docker exec -it sample-xapp /home/sample-xapp/run_xapp.sh
```

Listing 3.5: Commands to run the ColO-RAN sample xApp process.

subscribes to the RAN node specified in Listing 3.4 (through a RIC Subscription message), and triggers periodic reports (sent through RIC Indication messages) of RAN KPMs from the node.

After performing these steps, the ColO-RAN sample xApp logs on file the KPMs received from the RAN node. Users of OpenRAN Gym can add custom intelligence (e.g., through AI/ML agents) to the xApp by modifying the template scripts in the `setup/sample-xapp/` directory, and rebuilding the xApp Docker image through the steps described in this section.

3.1.4 xApp Design and Testing Workflow

The steps of the workflow to develop a data-driven xApp in OpenRAN Gym on Colosseum, shown in Figure 3.3, are described next.

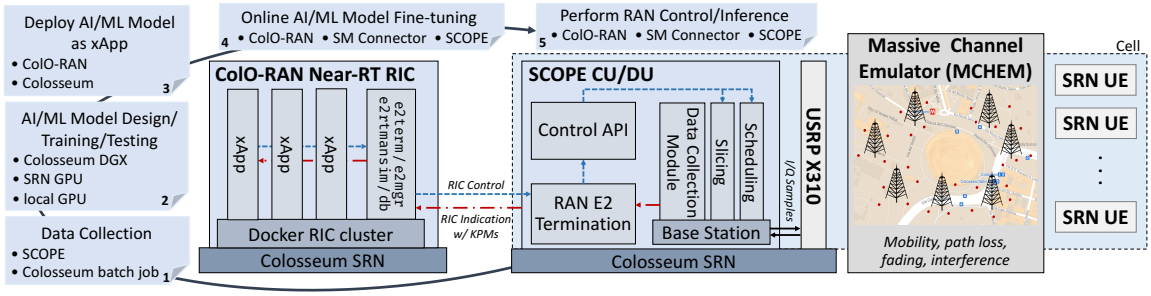


Figure 3.3: OpenRAN Gym xApp design and testing workflow on Colosseum.

1) **Data collection**. Data to train the AI/ML model that will be included in the xApp needs to be collected. In Colosseum, this can be done by using SCOPE to instantiate a large-scale cellular network with multiple base stations and UEs (see Section 3.1.2). During the experiment, the base stations log the KPMs relative to the performance of the served UEs, thus creating CSV-formatted datasets representative of the RAN statistics. Colosseum then transfers these datasets to its internal data storage, making them accessible to the users after the experiment ends. Data should be representative of a variety of environments and conditions, so that the model can adapt to diverse channel conditions and traffic requirements. Using OpenRAN Gym within Colosseum, this can be achieved by running several experiments that emulate different wireless and traffic scenarios (e.g., through SCOPE cellular scenarios (see Section 3.2.3).

However, manually running hundreds of experiments at scale and in different scenarios is not trivial, and it may take a long time. To facilitate this task, users can leverage Colosseum *batch jobs*, which take care of automatically running scheduled experiments (or jobs) configured through JSON files (see Section 3.4.3). In this way, users can schedule experiments in different scenarios and configuration in advance, and then retrieve the generated data from Colosseum data storage.

2) **AI/ML model design, training and testing**. After collecting data in the desired wireless environments, the model can be designed. This step involves selecting the AI/ML techniques that the model will use, which data it should take as input, the reward function of the model, and the set of actions produced as output (e.g., to perform inference or control of the RAN).

After the model has been designed, it can be trained and tested offline on the data collected in step 1.[4] These phases, which usually benefit from GPU-enabled environments, can either be carried out locally (on the user's own GPUs), or on the GPUs of the SRNs. As another option to train and

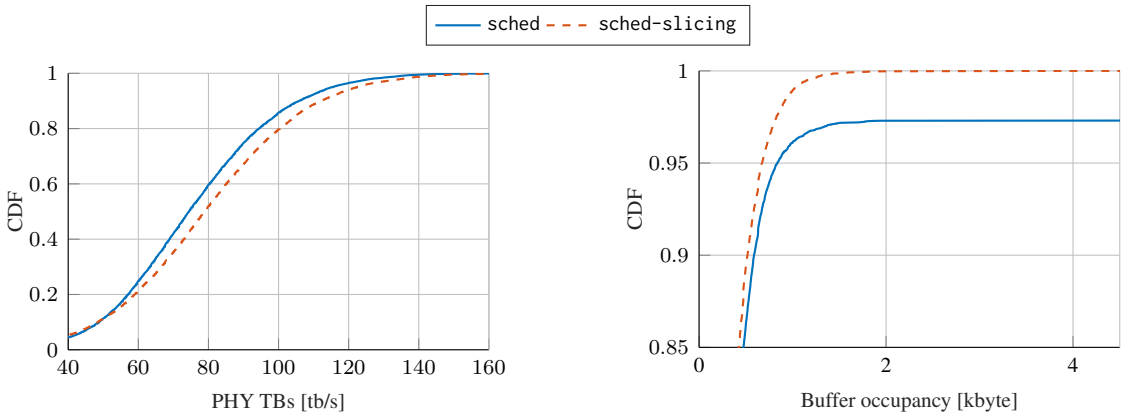
[4]It is worth mentioning that the O-RAN specifications do not permit the deployment of AI/ML models that have not been pre-trained offline. This is to shield the RAN from poor performance or outages [258].

test AI/ML solutions, Colosseum has recently added two NVIDIA DGX servers with A100 GPUs as part of its planned extensions. These novel servers significantly increase Colosseum's computational capabilities, making it, together with OpenRAN Gym, a key tool to develop data-driven solutions to control Open RAN systems.

3) **Deploy the AI/ML model as an xApp**. After the model has been designed, trained, and tested, it can be deployed as an xApp on the ColO-RAN near-real-time RIC. This can be done through the procedures described in Section 3.1.3.3. Specifically, the trained AI/ML model can be included in the ColO-RAN sample xApp (as the *data-driven logic unit* of Figure 3.2) modifying the template scripts in the `setup/sample-xapp/` directory. Then, the Docker image of the xApp with the user-defined logic is built with the commands of Listing 3.4, and instantiated on the ColO-RAN near-real-time RIC through the commands of Listing 3.5.

4) **Online AI/ML model fine-tuning**. Upon startup, the xApp interfaces with the SCOPE base station through the ColO-RAN near-real-time RIC and the O-RAN E2 termination. First, the xApp subscribes to SCOPE CU/DU by sending it RIC Subscription messages. Then, it triggers periodic reports (tunable based on the experiment requirements (see Section 3.3)) of RAN KPMs from the base station, sent through RIC Indication messages. The xApp may also use the KPMs in these report messages to fine-tune the model online, which allows it to adapt to the actual production environment where it will be deployed. Once the model has undergone this additional phase of online training, the xApp Docker image can be updated to save the newly trained weights of the model.

5) **Perform RAN control/inference**. Now the xApp can be used in the production infrastructure to perform run-time inference and control of the RAN. This latter step is achieved by having the xApp transmitting the actions computed by the AI/ML model (e.g., to modify the parameters and configuration of the base station) through RIC Control messages. These are sent to the base station through the O-RAN E2 interface, where they are processed by the CU/DU the xApp is subscribed to. At the CU/DU side, these messages trigger the SCOPE control APIs (see Figure 3.3) that apply the newly received configuration to the protocol stack of the base station at run time.



(a) MTC transmitted packets (b) URLLC buffer occupancy

Figure 3.4: Comparison of xApps developed with OpenRAN Gym.

We now showcase an example of xApps designed and tested with OpenRAN Gym, and used to control a cellular network with 7 base stations and 42 UEs (6 UEs per base station) instantiated

on Colosseum. Each base station serves UEs with different QoS requirements, divided on three network slices: Enhanced Mobile Broadband (eMBB), Machine-type Communications (MTC), and URLLC slice. We follow the workflow described in this section to design two Deep Reinforcement Learning (DRL)-based xApps—trained on a collected dataset with 3.4 GB of RAN traces, and more than 73 hours of experiments—that control the configuration of the base stations at run time using RAN KPMs, as discussed in Section 3.3. One xApp (named `sched`) manages the scheduling policies of the base station; the other (`sched-slicing`) also allocates the amount of resources available to each slice.

Figure 3.4 illustrates the Cumulative Distribution Function (CDF) of the RAN when the two xApps are instantiated on the ColO-RAN near-real-time RIC and used to control the network. Statistics on the transmitted Transport Blocks (TBs) for the MTC slice are shown in Figure 3.4a; on the downlink buffer occupancy of the URLLC slice in Figure 3.4b. In this example, both xApps aim at maximizing the transmit rate of the MTC traffic, and at minimizing the time packets remain in the base station queues for the URLLC traffic. By acting on an additional action set (i.e., the slice resource allocation), the `sched-slicing` xApp achieves better performance both in terms of transmitted packets and of buffer occupancy.[5]

3.1.5 Traveling Containers

In this section, we showcase some experimental results obtained from running OpenRAN Gym and its components across a set of heterogeneous testbeds. We ported the SCOPE and ColO-RAN near-RT RIC containers from Colosseum to the Arena, POWDER, and COSMOS testbeds. A description of the compute node and radio setups used in these testbeds (also summarized in Table 3.2) follows. Since the capabilities offered by the different testbeds can be substantially different (e.g., number

Table 3.2: Compute node and radio setups used across the different testbeds.

Testbed	Compute Node	Processor	CPU Cores	RAM [GB]	SDR
		Base Station (BS) / UE			
Arena	Dell EMC PowerEdge R340	Intel Xeon E-2146G	6	32	USRP X310
Colosseum	Dell EMC PowerEdge R730	Intel Xeon E5-2650	48	128	USRP X310
COSMOS	Asus server	Intel i7-4790	4	16	USRP B210
POWDER (BS)	Dell EMC PowerEdge R740	Intel Xeon Gold 6126	24	98	USRP X310
POWDER (UE)	Intel NUC 8559	Intel 7-8559U	4	32	USRP B210
		Near-RT RIC			
Arena	Dell EMC PowerEdge R340	Intel Xeon E-2146G	6	32	N/A
Colosseum	Dell EMC PowerEdge R730	Intel Xeon E5-2650	48	128	N/A
COSMOS	Supermicro 1028U-TRT+	Intel Xeon E5-2698	16	251	N/A
POWDER	Dell EMC PowerEdge R740	Intel Xeon Gold 6126	24	98	N/A

of available over-the-air nodes), for the sake of consistency, and to fairly compare results, we run experiments with one cellular base station and up to three UEs, and one near-RT RIC node. We divide the spectrum of the base stations into up to three network slices, and statically assign the UEs to them (e.g., based on the SLA between users and their network operator). Downlink UDP traffic

[5] A detailed evaluation of OpenRAN Gym xApps, including their orchestration, and control of large-scale experimental networks can be found in Section 3.3 and in Chapter 6.

generated through the iPerf3 tool is leveraged to evaluate the network performance. Finally, the base stations—implemented through SCOPE—connect to ColO-RAN near-RT RIC through the E2 interface standardized by O-RAN.

POWDER. We instantiated both the ColO-RAN near-RT RIC and the SCOPE base station on Dell EMC PowerEdge R740 compute nodes with Intel Xeon Gold 6126 processor, 24 CPU cores and 98 GB memory. The UEs were instantiated on Intel NUC 8559 nodes with Intel i7-8559U processor, 4 CPU cores, and 32 GB RAM. The radio front-end of the base station was implemented through a USRP X310, while USRP B210 were used for the UEs. As this testbed does not natively support the LXC virtualization technology, the OpenRAN Gym container images were transferred from Colosseum to the compute nodes through the `scp` utility, instantiated on Ubuntu Linux images loaded on the bare-metal servers of the testbed.

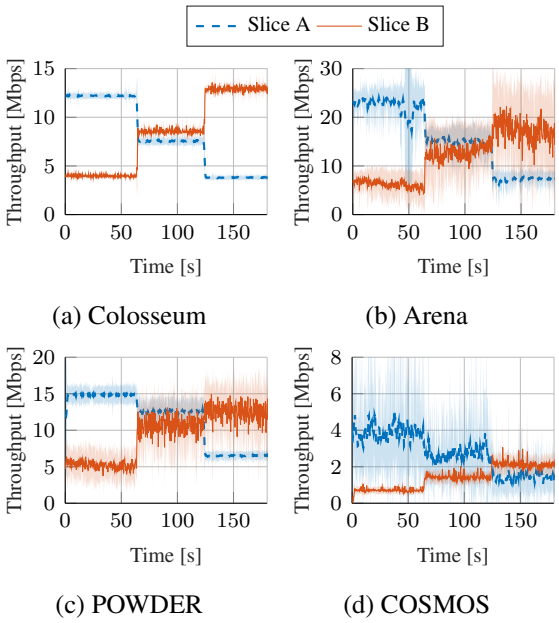
COSMOS. In this case, the near-RT RIC was instantiated on a Supermicro 1028U-TRT+ server with an Intel Xeon E5-2698 processor, 16 CPU cores and 251 GB memory. Base station and UE, instead, were virtualized on Asus servers with Intel i7-4790 processor, 4 CPU cores, and 16 GB memory driving USRP B210 SDRs. Similarly to what done for POWDER, as the LXC virtualization technology is not directly supported by this testbed, the container images were transferred from Colosseum through the `scp` utility, and instantiated on Ubuntu Linux images loaded on the bare-metal nodes available on the testbed.

Arena. All applications were run on Dell EMC PowerEdge R340 servers with Intel Xeon E-2146G processor, 6 CPU cores, and 32 GB memory. In this case, the OpenRAN Gym LXC containers are instantiated directly on the bare-metal nodes of the testbed, which leverage USRP X310 SDRs as radio front-ends. On this testbed, the UEs are implemented through commercial smartphones.

Colosseum. To mimic the same deployment scenario used in the other testbeds, in Colosseum we considered cellular nodes deployed in a static Radio Frequency (RF) scenario without user mobility. In this case the LXC containers of RIC, base station, and UEs directly run on Colosseum bare-metal nodes, i.e., Dell EMC PowerEdge R730 servers with Intel Xeon E5-2650 processor, 48 CPU cores, and 128 GB memory. All the cellular nodes leverage USRP X310 SDRs as radio-front ends.

3.1.5.1 Experimental Results

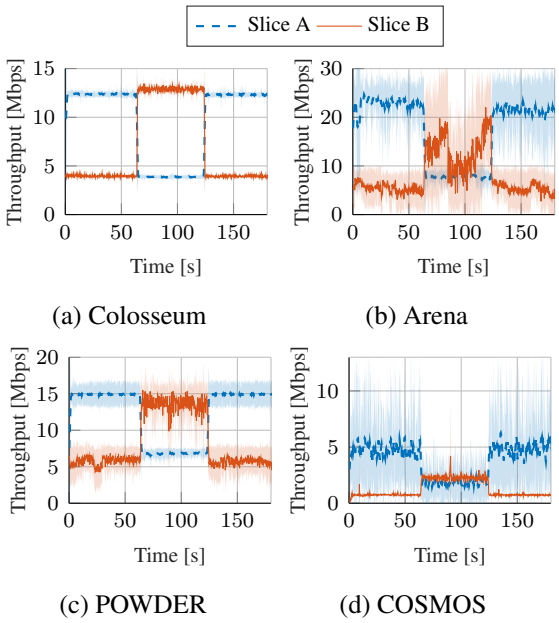
To showcase the flexibilty of OpenRAN Gym in dynamically reconfiguring the spectrum allocated to the network slices across different testbeds, Figures 3.5 and 3.6 show the overall throughput of each network slice varying the resources allocated to them, in terms of RBGs. 95% confidence intervals are also represented by the shaded areas in the figures. For both figures, the percentage of RBGs allocated to each slice of the base station—which uses a 10 MHz configuration—is dynamically changed through the SCOPE APIs according to the following configuration (also summarized in Table 3.3). In Figure 3.5, the two network slices, i.e., slice A and B in the figure, are allocated the following RBGs percentage: (i) 75% to slice A and 25% to slice B in the first minute; (ii) 50% to each slice in the second minute, and (iii) 25% to slice A and 75% to slice B in the third minute. In Figure 3.6, instead they are allocated the following RBGs percentage: (i) 75% to slice A and 25% to slice B in the first minute; (ii) 25% to slice A and 75% to slice B in the second minute, and (iii) 75% to slice A and 25% to slice B in the third minute. In both these figures, the throughput varies proportionally to the specific allocation of slice resources, in which slices with more RBGs achieve higher throughput values. These values then change during the experiment as RBGs are



(a) Colosseum (b) Arena

(c) POWDER (d) COSMOS

Figure 3.5: Overall slice throughput varying the percentage of RBGs allocated to each slice over time according to the configuration reported in Table 3.3.



(a) Colosseum (b) Arena

(c) POWDER (d) COSMOS

Figure 3.6: Overall slice throughput varying the percentage of RBGs allocated to each slice over time according to the configuration reported in Table 3.3.

Table 3.3: Slicing configuration, expressed as percentage of RBGs, used in Figures 3.5 and 3.6.

Figure	Slice	First Minute	Second Minute	Third Minute
Figure 3.5	Slice A Slice B	75% RBGs 25% RBGs	50% RBGs 50% RBGs	25% RBGs 75% RBGs
Figure 3.6	Slice A Slice B	75% RBGs 25& RBGs	25% RBGs 75% RBGs	75% RBGs 25% RBGs

dynamically reallocated to the slices. We notice that even if the throughput differs across the various testbeds because of the different capabilities and environments they offer—with Arena achieving the highest performance due to the use of commercial smartphones as the UEs—the overall trends are consistent across the different setups.

We now showcase an instance in which the ColO-RAN near-RT RIC is leveraged to control a softwarized RAN implemented through SCOPE. LXC containers for both applications are deployed on the testbeds mentioned above, whose specifications are summarized in Table 3.2. Figure 3.7 shows the evolution in time of the throughput of the three network slices (namely, slice A, B, and C) implemented by the SCOPE base station. Initially, the slices are allocated a fixed RBG configuration, and no control is performed by the RIC. Then, at around second 150, an xApp that prioritizes one of the network slices (slice A in the figure) is instantiated on the near-RT RIC. As a result, the xApp dynamically reallocates the amount of RBGs of each slice, which reflects on the performance of the slices of the RAN. Similar to the previous case, slices with a larger amount of RBGs allocated to them achieve higher throughput values. Overall, even in this case results are consistent across the

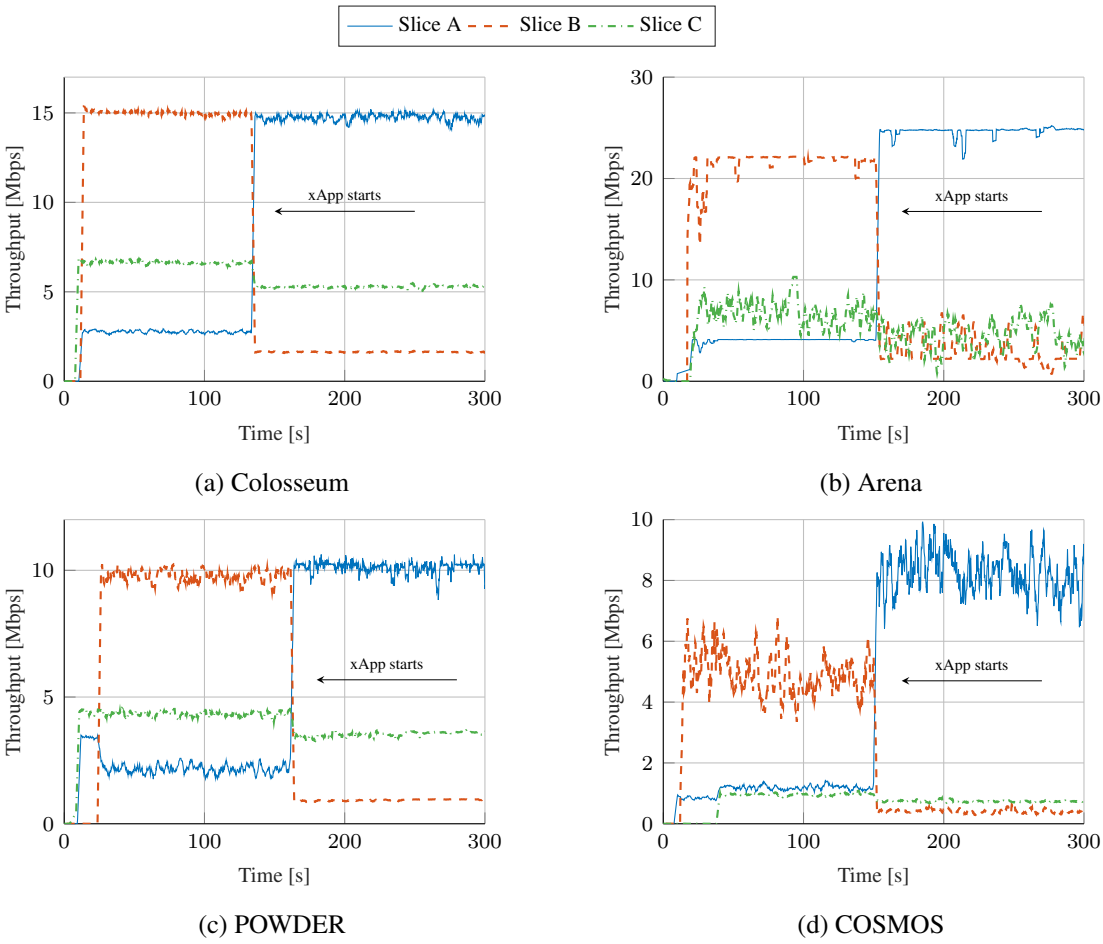


Figure 3.7: Slice throughput when the SCOPE RAN is controlled by ColO-RAN near-RT RIC. At around second 150, xApp to prioritize the amount of resources (i.e., RBGs) allocated to slice A is instantiated on the near-RT RIC.

different testbeds.

Now we show some timing statistics on the average amount of time taken to transfer the SCOPE and ColO-RAN LXC images from Colosseum to the Arena, COSMOS, and POWDER platforms. All the image transfers were performed through the `scp` utility. In both cases, these timing statistics were derived using the hardware of Table 3.2. We used the compute nodes listed in the "base station (BS)/UE" section of the table for the SCOPE LXC image/container (in the case of the POWDER platform, in which different compute nodes were are listed for base station and UE, the base station node was used), and the compute nodes in "near-RT RIC" section of the table for ColO-RAN. In the tables that will be described next, we consider the following LXC images:

- *SCOPE w/ E2*: this is the SCOPE LXC image with the O-RAN E2 termination to interface with the near-RT RIC.
- *ColO-RAN near-RT RIC, prebuilt*: this is the ColO-RAN LXC image in which the Docker

containers of the RIC, and sample xApp (described in Section 3.1.3) have been built a priori.

- *ColO-RAN near-RT RIC, to build*: this is the ColO-RAN LXC image with the scripts to build the Docker containers of the RIC and sample xApp from scratch.

Table 3.4 shows the average time required to transfer the LXC images from Colosseum to the other platforms. Times span from as low as ~ 1.5 minutes to as high as almost 6 minutes, depending on

Table 3.4: Average time to transfer the LXC images from Colosseum to specific testbeds. The size of each image is listed in brackets.

Testbed	SCOPE w/ E2 (1.7 GB)	ColO-RAN near-RT RIC, prebuilt (6.5 GB)	ColO-RAN near-RT RIC, to build (1.6 GB)
Arena	1 m 27.413 s	5 m 41.487 s	1 m 25.002 s
COSMOS	1 m 28.631 s	5 m 39.704 s	1 m 27.352 s
POWDER	1 m 30.787 s	5 m 43.704 s	1 m 28.546 s

the size of each image—also listed in the table—and capabilities of the testbeds. However, transfer times are consistent across the different testbeds.

Finally, Table 3.5 shows the times taken to instantiate LXC containers from the images transferred from Colosseum. In this case, we notice some difference among the times achieved on the different

Table 3.5: Average time to start as a container the LXC image exported from Colosseum on specific testbeds. The size of each image is listed in brackets.

Testbed	SCOPE w/ E2 (1.7 GB)	ColO-RAN near-RT RIC, prebuilt (6.5 GB)	ColO-RAN near-RT RIC, to build (1.6 GB)
Arena	0.887 s	1 m 11.483 s	46 m 18.110 s
COSMOS	25.463 s	2 m 34.905 s	26 m 4.410 s
POWDER	30.139 s	2 m 55.654 s	21 m 11.220 s

testbeds. For instance, Arena is significantly faster than COSMOS and POWDER in instantiating the SCOPE container—completing the instantiation in less than 1 s—and the prebuilt ColO-RAN container (instantiation in approximately 1 minute). This is mainly due to the fact that Arena allows users to instantiate applications on the bare-metal nodes directly. This removes the latency of the extra virtualization layer of the other two testbeds, in which the LXC containers are nested inside the virtualized architecture the users are given access to. When it comes to building the Docker containers of the ColO-RAN near-RT RIC (see Section 3.1.3) from scratch, instead, POWDER and COSMOS are significantly faster than Arena, taking approximately half the time to complete the same operations. This is mainly due to the superior compute capabilities of the nodes of these two testbeds (24 core CPU server on POWDER, and 16 core server on COSMOS vs. 6 core CPU server on Arena). Nonetheless, this building operation needs to be completed only once, as the compiled ColO-RAN LXC image can be saved to be used in subsequent experiments, with instantiation times sensibly lower (slightly above 1 minute for Arena, and below 3 minutes for POWDER and COSMOS).

3.2 SCOPE: Softwarized Prototyping of NextG Systems

In this section, we address the key issue of facilitating platform-independent design by presenting SCOPE (for *Softwarized Cellular Open Prototyping Environment*), a development environment tailored to the design, prototyping and testing of solutions for the softwarized NextG cellular RAN. SCOPE consists of a virtualized container and an emulation environment with the following features:

- *Open and portable implementation*. SCOPE includes an open-source implementation of a 3GPP-compliant softwarized cellular base station. The cellular base station adds novel capabilities to the srsRAN base implementation (described in Section 2.2.2), such as RAN slicing, and MAC and PHY-layer functions. These allow, for instance, to run multiple virtual networks on top of the same physical infrastructure, with the option to select a different scheduling policy in each one of them. Fine-tuning the Modulation and Coding Scheme (MCS) of each mobile subscriber, and implementing downlink power-control schemes are also possible. These functionalities can either be controlled directly through the SCOPE open-source implementation, or through a set of APIs, also enabling real-time reconfiguration of softwarized network elements. To make SCOPE platform-independent, we developed a ready-to-deploy LXC instance of SCOPE to be deployed on LXC-enabled Linux machines. (See Section 2.5.1 for more details on this virtualization technique.)

- *Data collection capabilities*. SCOPE includes a data collection module for automatically recording the performance of the network. Collected data can be used at run time, e.g., to design adaptive solutions, or offline, e.g., to facilitate the design, training and testing of ML/AI algorithms.

- *Prototyping RF and traffic scenarios.* To further facilitate NextG experimental research, we developed a set of real-world RF and traffic scenarios—namely, *SCOPE scenarios*—that can be run through the Colosseum emulator of Section 3.4. SCOPE users can leverage them to test algorithms at scale under diverse channel conditions (e.g., position, mobility), network deployments (e.g., rural, urban) and traffic.

- *Repeatability, reproducibility and replicability at scale.* All SCOPE scenarios are executed in a deterministic way. This means that while channel coefficients and traffic change over time during an experiment, all experiments executed with the same scenario will experience the very same channel and traffic conditions. In this way, our system can be used to *prototype and fine-tune* solutions by experimenting at scale in repeatable environments before testing them in the field.

We demonstrate the effectiveness of SCOPE as a prototyping environment by: (i) analyzing the statistical properties of SCOPE cellular scenarios representative of diverse urban environments, and (ii) showcasing exemplary ML and optimization applications. We also provide an example of how SCOPE can be used to port solutions prototyped on Colosseum to real-world heterogeneous testbeds, namely the indoor Arena (see Section 3.5), and the outdoor POWDER (described in Section 2.7).

The remainder of this section is organized as follows. Section 3.2.1 presents an overview of how to use SCOPE for wireless experiments. The open-RAN implementation of SCOPE and its capabilities are described in Section 3.2.2. The SCOPE emulation environment is presented

in Section 3.2.3. Section 3.2.4 showcases use cases of SCOPE, including ML and optimization applications, and demonstrates the portability of SCOPE to different testbeds. Finally, related works are surveyed in Section 3.2.5.

3.2.1 Experimenting with SCOPE

The lifetime of an experiment with SCOPE is illustrated in Figure 3.8. For enhanced clarity, we provide a step-by-step summary of how to use SCOPE on Colosseum. However, SCOPE can be instantiated on any LXC-enabled testbed (Section 3.2.4.3).

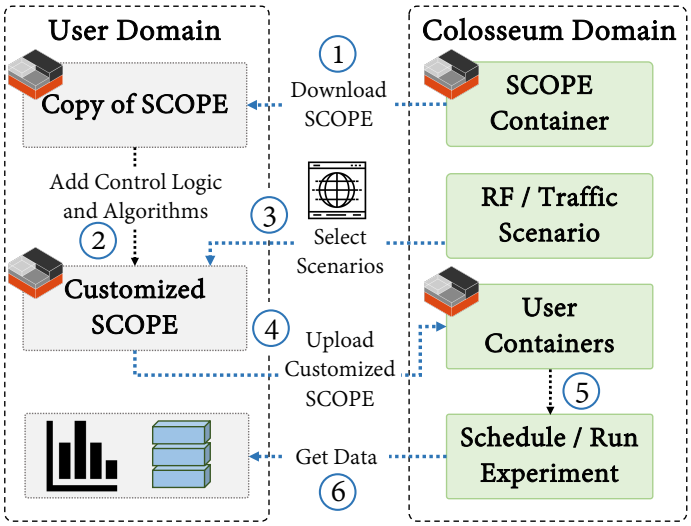


Figure 3.8: High-level lifetime of SCOPE experiments.

Users first download a copy of the SCOPE container (Section 3.2.2) that includes the code to instantiate base stations and UEs, as well as our SCOPE APIs (Section 3.2.2.3) for controlling key functions of the softwarized base station at run time (step 1 in the figure). Then, on their local machine (User Domain in the figure), SCOPE users add the desired control logic and algorithms to the container either via the SCOPE APIs or by interfacing directly with its open-source code. In this way, a "customized" instance of SCOPE is created (step 2). Users can now select RF and traffic scenarios for their experiment (Section 3.2.3.2) from the set of available SCOPE scenarios, through a dedicated Graphical User Interface (GUI) (step 3). After, researchers can upload their customized container (i.e., SCOPE with user-defined control logic) to Colosseum (step 4). They can then schedule an experiment (through a web GUI) specifying parameters such as the number of nodes (step 5).

In the case of Colosseum, each node, or SRN, consists of a GPU-endowed server connected to one USRP X310. SRNs are fully programmable and serve as virtualized environments running LXC. This makes it possible to use them as either compute-only (e.g., edge or cloud servers) or compute-and-transmit (e.g., UE or base station) nodes.

As soon as the configuration phase is complete, Colosseum deploys the containers on the selected SRNs, starting the experiment. Network run-time metrics are saved in CSV format in the metrics

and performance dataset. Users can either query the generated dataset at run time (e.g., using it as a feedback loop) or download it at the end of the experiment to refine the control logic (step 6).

3.2.2 SCOPE Virtualized Container

We designed SCOPE[6] to facilitate the development of new, adaptive solutions for NextG softwarized cellular systems. A SCOPE container, realized through LXC, is a flexible and ready-to-use prototyping toolkit for users to effortlessly instantiate platform-independent softwarized networks.

The main components of a SCOPE container are: (i) a softwarized cellular protocol stack of base stations, UEs and core network (Section 3.2.2.1); (ii) a data collection module for artificial intelligence and machine learning applications (Section 3.2.2.2), and (iii) a set of open APIs for users to interface with and control the other two components in real time (Section 3.2.2.3).

Users can instantiate SCOPE on any LXC-enabled Linux system (e.g., Colosseum, POWDER and Arena, as discussed in sections 3.2.3 and 3.2.4.3), and control the base stations and their configuration in real time with just a few lines of code through a set of high-level open APIs, or by modifying directly the open-source code. The APIs are directly interfaced with the data collection module, thus providing SCOPE users with a feedback loop to monitor the performance and state of the network and adapt the control strategy accordingly.

3.2.2.1 Softwarized Protocol Stack

The SCOPE base stations, UEs, and core network are based on the open cellular software srsRAN (described in Sections 2.2.2 and 2.3). SCOPE advances the programmability and virtualization capabilities of srsRAN by considerably extending its functionalities to include network slicing, additional MAC-layer scheduling policies, and the ability of fine-tuning PHY-layer parameters at run time. The most relevant functionalities introduced by SCOPE are described in the remaining of this section.

Network Slicing. The SCOPE implemtation of network slicing supports the coexistence of multiple slices tailored to specific traffic classes and UEs on the same shared infrastructure. Our implementation makes it possible to slice the spectrum available at each base station and to dictate the resource allocation for each slice of the network. This is achieved by specifying how many downlink RBGs (and, thus, PRBs) are allocated to each slice. For example, SCOPE makes it possible to allocate more resources to slices associated with high data rate traffic, while giving less resources to those slices that have lower priority or no strict QoS requirements. Selecting the exact portion of the spectrum to allocate to a given slice is also possible. This is achieved through an allocation matrix that specifies which of the available RBGs should be allocated to the slice. Optionally, this allocation matrix can be periodically reloaded at run time to dynamically modify the slice resource allocation.

To facilitate the setup and instantiation of network slices, SCOPE APIs enable the users to specify the association between UEs and slices according to QoS requirements. The APIs also allow to assign different scheduling policies to each slice, and to modify them at run time. This makes it possible to define slice-specific control strategies. This feature is useful to evaluate how specific resource allocations affect different slices and the services they provide.

[6] SCOPE has been publicly released to the research community: `https://github.com/wineslab/colosseum-scope`.

MAC-layer Scheduling. By default, srsRAN implements a round-robin scheduling algorithm. This limits the degrees of freedom researchers can enjoy in their experiments. SCOPE implements two additional, fairer MAC-layer scheduling algorithms: *waterfilling* and *proportionally fair*. Both are implemented by first computing the amount of downlink PRBs required by the users based on the data they request, the MCS and the transport block size. Then, resources are granted and allocated according to the users requirements and slice capabilities (e.g., with an allocation proportional to the user request in case of the proportionally fair scheduler). Additional scheduling algorithms can be implemented and plugged in by SCOPE users by modifying the provided resource allocation routines. Scheduling policies can be reconfigured at run time for either the whole network or for selected slices via the SCOPE APIs.

By combining scheduling and slicing capabilities, users can control the performance of each UE and the service levels, and reconfigure the network at run time, if necessary. This effectively enables Telecommunications Companies (Telcos) to offer different levels of service and subscription to UEs.

PHY-layer Capabilities. At the PHY layer SCOPE offers the ability to fine-tune per-user downlink transmission power and MCS (both in uplink and downlink). The former is obtained by selecting the percentage of the maximum power to be assigned to downlink signals of the selected UEs. The latter is an integer number in the range $[0, 28]$ that can be used to change modulation scheme and coding rates of downlink and uplink transmissions on a per-user basis. Current available choices for the modulation are Quadrature Phase Shift Keying (QPSK), 16 QAM and 64 QAM. The relationship between MCS index, modulations and coding rate is defined by Tables 7.1.7.1-1, 7.1.7.1-1A and Table 8.6.1-1 in [265].

3.2.2.2 Data Collection Module

Artificial Intelligence is rapidly reshaping the way we design and operate cellular networks. Unfortunately, one of the issues that has plagued the research community for years is the almost complete lack of large-scale datasets to train ML models. Business and privacy concerns often keep Telcos from publicly releasing their datasets, thus slowing down innovation and advancements in the field from the research community. Recently, the release of open-source software for cellular networks has enabled researchers to generate their own datasets. However, this is no easy task as testbeds are usually small-scale and can only model network behavior in limited setups. This makes it hard to train models that can be applied to diverse network deployments and conditions.

SCOPE aims at overcoming these limitations by including a data collection module for the creation of large-scale datasets over a wide number of realistic RF and traffic scenarios. Used on experimental testbeds such as Colosseum, SCOPE provides an effective tool for data collection and experimentation of ML/AI solutions in large-scale cellular networks. The main advantage over existing platforms is that SCOPE combines hardware-in-the-loop execution of cellular procedures with the reconfigurability of the RF channels provided by Colosseum. In this way, while other platforms are representative only of their physical deployment (e.g., location of radios, channel conditions), SCOPE (combined with Colosseum) supports data collection from a virtually infinite number of scenarios and channel conditions. This facilitates the design and testing of ML/AI solutions that are not tied to specific network deployments.

Detailed statistics (e.g., throughput, MCS, buffer size, slice PRBs) on the performance of each base station and UE are periodically logged by the SCOPE data collection module and stored in a CSV-formatted dataset. As we will discuss in Section 3.2.2.3, data generated by this module can be accessed via the SCOPE APIs at run time, allowing the implementation of closed-loop optimization and data-driven control routines.

3.2.2.3 Open APIs

A sample of relevant SCOPE Python APIs is shown in Table 3.6. They facilitate the run-time reconfiguration of network slicing, scheduling and PHY-layer parameters (Section 3.2.2.1), among others. They also allow SCOPE users to query the metrics and performance dataset (Section 3.2.2.2).

Table 3.6: Sample of relevant SCOPE Python APIs.

Function	Description
`enable_slicing`	Enables/disables network slicing globally
`set_slice_users`	Set UE-slice associations
`get_slice_users`	Get UE-slice associations
`set_slice_resources`	Sets resources allocated to the slice
`set_slice_scheduling`	Sets slice scheduling policy
`set_slice`	Sets slice scheduling and/or resources
`set_scheduling`	Sets global scheduling of the BS
`set_mcs`	Sets UE downlink/uplink MCS
`set_power`	Sets scaling factor for UE downlink signals
`read_metrics`	Reads metrics from dataset
`get_metric`	Returns value of a specific metric

For instance, network slicing can be enabled/disabled globally for the whole network (*enable_slicing*). If enabled, SCOPE APIs allow to set/get the UEs associated to each slice of the network (*set/get_slice_users*). The behavior of each slice can be configured in terms of allocated PRBs—which reflects the portion of spectrum available to the slice—and scheduling policy (*set_slice_resources*, *set_slice_scheduling*). Additionally, PRBs and scheduling of each slice can also be jointly set (*set_slice*). Besides allowing users to set the scheduling policy for each slice, SCOPE APIs also allow to set the global scheduling policy for the whole network (*set_scheduling*).

The PHY-layer configuration can be tuned by setting the downlink/uplink MCS of selected UEs, e.g., all the UEs of a certain slice (*set_mcs*), which directly impacts on the signal modulation and coding rate (see Section 3.2.2.1 and [265]). Additionally, the power level of signals for selected UEs can be tuned as well (*set_power*). Finally, SCOPE APIs allow interaction with the dataset generated by the data collection module of the base stations (Section 3.2.2.2). Specifically, they allow to read and get specific metrics values for any target time window (*read_metrics* and *get_metric*).

Listing 3.6 shows an example where SCOPE APIs are used to dynamically assign resources to each slice according to run-time performance read from the dataset.

While a SCOPE experiment is running (line 3), the user calls the SCOPE *read_metrics* API to read the performance metrics of each slice from the generated dataset and for the specified *time_window* (line 4). Metrics are stored in a dictionary (*wnd_metrics*) that can be accessed iteratively (line 6). Users can call the SCOPE *set_mcs* API to set a specific *mcs_level* for all UEs of the slice (line 9). Note that SCOPE APIs also enable the selection of MCS levels for each UE.

```
1  import scope_api as sc, time
2
3  while experiment_running:
4    wnd_metrics = sc.read_metrics(time_window)
5
6    for slice_id, slice_metrics in wnd_metrics.items():
7      slice_users = slice_metrics['ue']
8      slice_rbg = slice_metrics['rbg']
9      sc.set_mcs(slice_users, mcs_level, 'dl')
10
11     if slice_metrics['buffer'] > threshold:
12       sc.set_slice(slice_id, 'proportionally', slice_rbg + 2)
13     else:
14       sc.set_slice(slice_id, 'round-robin', slice_rbg - 2)
15
16   time.sleep(timeout)
```

Listing 3.6: Example of SCOPE APIs.

Listing 3.6 also shows an example of how SCOPE users can implement control logic policies. For example, users can change scheduling and network slicing policies when the metrics reported in SCOPE dataset meet certain conditions. For instance, if the size of the transmission buffer (*slice_metrics['buffer']*) is above/below a threshold (lines 11 and 13), the resources of each slice, e.g., scheduling policy and allocated RBGs, can be changed accordingly (*set_slice*, lines 12 and 14). Finally, the algorithm waits for a *timeout* (line 16) before reading the metrics from the dataset again.

We note that the SCOPE APIs are a tool provided to users to facilitate the reconfiguration of a variety of network parameters. They are not the only way to access lower-layer information and capabilities. Developers can still customize and modify the open-source cellular code and access parameters not currently available via the SCOPE APIs.

3.2.3 SCOPE Emulation Environment

In this section, we detail SCOPE cellular emulation environment, executed through Colosseum. A system overview is shown in Figure 3.9 where we distinguish between two main parts: (i) *user domain*, and (ii) *Colosseum domain*.

- *User Domain.* The user domain, running on the user local machine, is where researchers download the SCOPE container and use its APIs to implement their custom algorithms (Section 3.2.2). Here, SCOPE users interface with Colosseum to run their experiments and test their devised solutions in various emulation setups. Specifically, users access Colosseum via a dedicated web GUI [266], select the SCOPE scenarios they want to run to evaluate their solutions, and eventually visualize and process the obtained results. Indeed, SCOPE allows to collect large amounts of data (e.g., throughput, transmission queue status, CQI, PRBs allocated to each network slice, to name a few) that can be leveraged to train ML/AI models, or to design novel optimization and heuristic solutions for cellular applications.

- *Colosseum Domain.* The operations executed on Colosseum can be divided into two different phases: (i) the *experiment configuration*, and (ii) the *experiment execution*.

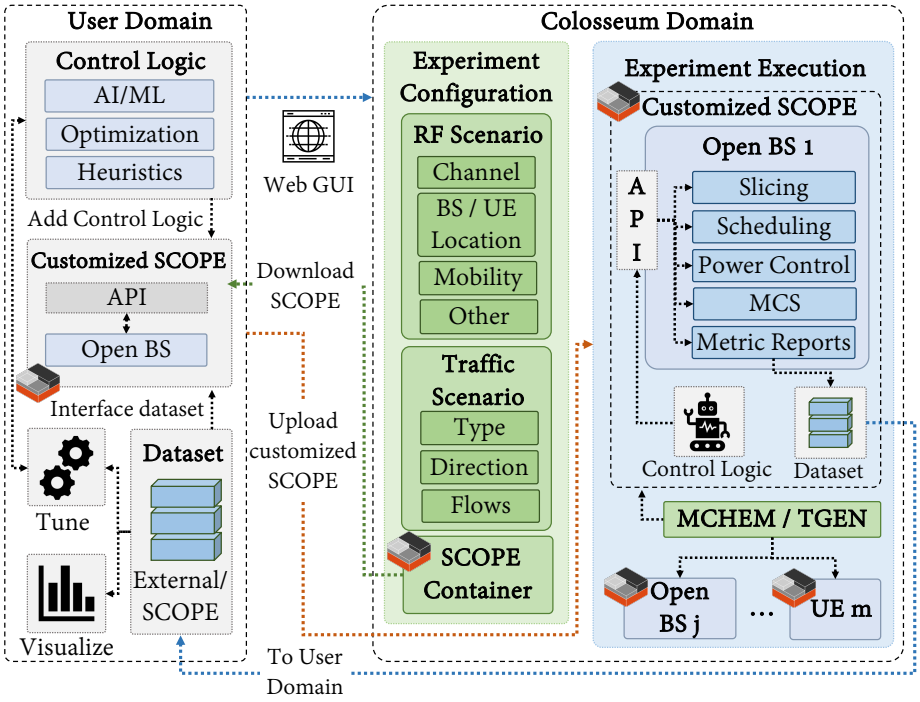


Figure 3.9: SCOPE emulation environment.

During the experiment configuration, users setup the experiment to run on Colosseum. They select the desired RF and traffic scenarios and the duration of the experiment. Among others, they can specify which nodes act as base stations and which as UEs. After creating a customized instance of SCOPE with user-defined control logic, users can upload it on Colosseum. They now enter the experiment execution phase, where the specified experiments are actually run.

Experiments can be scheduled and started through a web GUI. Once the experiment begins, the user-customized SCOPE container is automatically deployed on the corresponding SRNs, and SCOPE RF and traffic scenarios of choice are executed by Colosseum MCHEM and Traffic Generator (TGEN) (Section 3.2.3.1). User-defined control logic, e.g., any script using SCOPE APIs to implement custom ML and/or optimization algorithms, is also started at run time and the dataset module acts as a feedback loop on the network performance and can be used to evaluate the impact of control decisions on each node.

When the experiment is over, the scenario execution ends and all the results collected during the experiment—stored in the metrics and performance dataset—are transferred to the user directory on Colosseum. This makes it possible to process results and save the dataset for future applications.

3.2.3.1 Creating Cellular Scenarios

This section provides details on the inner workings of SCOPE cellular scenarios. An overview of ready-to-use sample scenarios will be given in Section 3.2.3.2. Each scenario (Figure 3.10) consists of two macroblocks: the *RF scenario* and the *traffic scenario*.

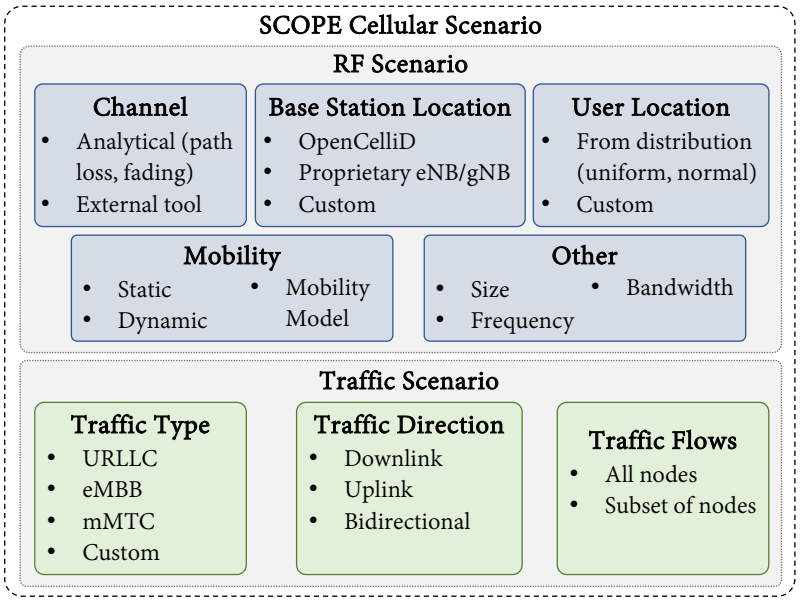


Figure 3.10: SCOPE cellular scenario components.

The ***RF Scenario*** specifies the *channel conditions* that each node experiences in the experiment. For each SRN, the scenario defines channel impulse responses that model path loss, fading and multipath effects. These channel coefficients are updated every millisecond and can be generated in different ways. Colosseum supports scenarios with channel coefficients generated via analytical models, ray tracing software or obtained via real-world measurements/channel sounders. Coefficients are then fed to MCHEM, which applies the corresponding channel taps to signals to/from each SRN. (See Section 3.4.1 for more details on Colosseum and MCHEM.)

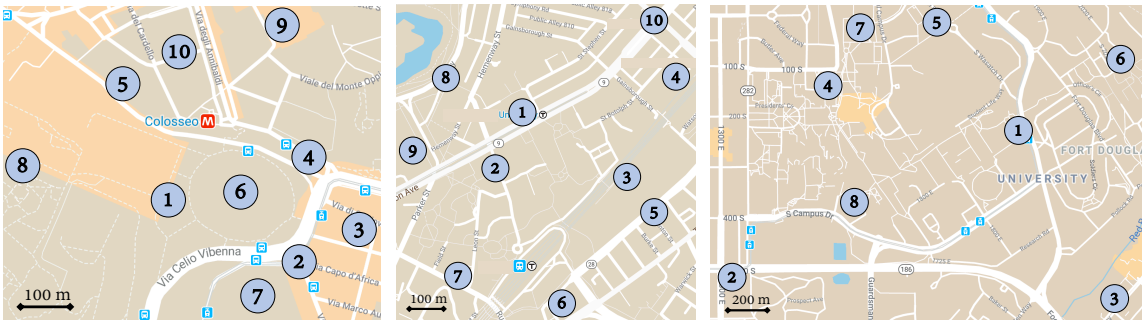
Since channel conditions vary based on the *location* of the nodes, RF scenarios specify the position of each node for each instant of time (Figure 3.10). This makes it possible to run experiments on different cellular deployment configurations that mimic real-world cellular topologies. The location of the base stations can be specified via GPS coordinates, which can be either generated randomly, through statistical models, or derived from open-source 4G/5G datasets such as OpenCelliD [267]. Similarly, the location of UEs can be specified via GPS coordinates according to diverse probability distributions (e.g., uniform, normal), or historical data from Telcos, if available. Additionally, scenarios can specify *user mobility*, which in SCOPE can be static, i.e., the users do not move for the entire duration of the experiment, or dynamic, in which case the mobility model and speed can also be specified. Finally, RF scenarios allow to select the size of the emulated environment and RF frequency.

The ***Traffic Scenario*** specifies and configures the traffic flows among base stations and UEs (Figure 3.10). Traffic scenarios are handled by the Colosseum TGEN, which is built on top of Multi-Generator (MGEN), an open-source software that generates and controls realistic TCP/UDP traffic [268]. MGEN supports a variety of different classes of traffic with diverse QoS requirements, probability distributions, data rates and types of service. In this way, MGEN can be used to generate URLLC, eMBB and MTC traffic, which SCOPE can assign to different slices of the network and

control independently (Section 3.2.2).

3.2.3.2 SCOPE Cellular Scenarios

In this section, we give an overview of sample SCOPE cellular scenarios in three different urban setups: (i) Rome, Italy; (ii) Boston, U.S., and (iii) Salt Lake City, U.S. (POWDER scenario). SCOPE cellular scenarios have been thoroughly designed leveraging the toolchain developed and validated by DARPA for the Colosseum network emulator. The very same toolchain has been used to create the scenarios of the Spectrum Collaboration Challenge, a $2M competition to foster collaboration in the wireless spectrum [269]. For the Rome and Boston scenarios, the locations of the base stations reflect real cell tower deployments extracted from the OpenCelliD database [267]. In the POWDER scenario, they mirror those of the rooftop base stations deployed in the Salt Lake City platform [270]. Each scenario includes from 8 to 10 base stations and up to 40 UEs, whose location and mobility can be selected by SCOPE users when setting up an experiment. We considered the following UE distribution configurations: (i) *close* (UEs are randomly distributed within $20\ \mathrm{m}$ from the serving base station); (ii) *medium* ($50\ \mathrm{m}$), and (iii) *far* ($100\ \mathrm{m}$). We implemented three different mobility configurations: (i) *static*, in which UEs do not move for the entire duration of the experiment; (ii) *moderate*, in which they move at an average speed of $3\ \mathrm{m/s}$, and (iii) *fast*, in which their average speed is $5\ \mathrm{m/s}$. In all the cases with mobility, UEs follow a random waypoint mobility model.



(a) Rome scenario (b) Boston scenario (c) POWDER scenario

Figure 3.11: Large-scale cellular scenario maps. The numbered blue circles mark the locations of the base stations on the map.

A graphical overview of SCOPE sample cellular scenarios is given in Figure 3.11, in which the numbered blue circles represent the locations of the cellular base stations on the map:

- The *Rome scenario* captures the dynamics of the city center of Rome, Italy. A total of 50 nodes are involved: 10 base stations and 40 UEs. This is the most dense scenario we developed in Colosseum and it covers an area of $0.5\ \mathrm{km}^2$ (Figure 3.11a).

- The *Boston scenario* captures the dynamics of downtown Boston, U.S. A total of 50 nodes are involved: 10 base stations and 40 users. This scenario covers an area of $0.95\ \mathrm{km}^2$ (Figure 3.11b).

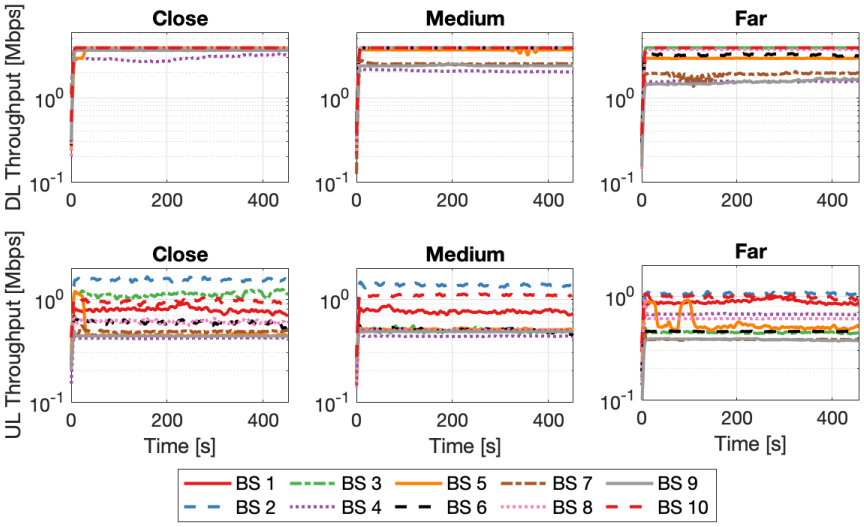


Figure 3.12: Downlink (DL) and uplink (UL) throughput in the Rome static scenario.

- The *POWDER scenario* mirrors the setup of the rooftop base stations deployed in the POWDER platform in Salt Lake City, U.S. [270]. A total of 40 nodes are involved: 8 base stations and 32 UEs. This scenario is the sparsest with an area of 3.6 km^2 (Figure 3.11c).

Along with the RF scenario, we designed relevant traffic scenarios (Section 3.2.3.1). Given the ever-increasing popularity of video streaming platforms, we leveraged TGEN and MGEN to generate dedicated traffic scenarios that model uplink and downlink video streaming traffic flows among UEs and base stations.

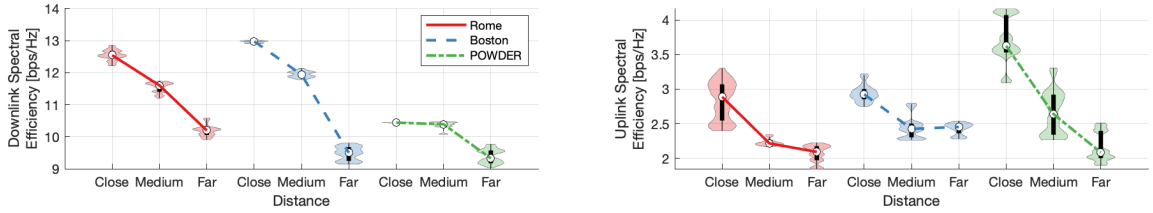
Scenario Analysis. We now show results obtained by executing SCOPE in the above scenarios focusing on (i) providing insights on how different topologies, distributions and mobility patterns affect network performance, and (ii) showcasing the statistical properties of SCOPE experiments.

To better highlight the major differences among the above scenarios, we consider the default SCOPE configuration where all UEs generate the same type of traffic, belong to the same slice, and are served via a round-robin scheduling algorithm (see Section 3.2.4 for more use cases). For each scenario, we measured downlink/uplink throughput and spectral efficiency for different distances and mobility configurations.

- *Overview of a SCOPE experiment.* To give a better understanding of what running SCOPE experiments looks like, we first show results pertaining single experiment runs (one for each distance among base stations and UEs) in the Rome scenario with static UEs. The considered distances are: *close*, *medium*, and *far*, as described earlier in this section. The measured downlink and uplink throughput is shown in Figure 3.12. As the distance among base stations and UEs increases, the gap among the downlink throughput of different base stations (top part of Figure 3.12) becomes larger. This is due to channel artifacts, e.g., path loss and fading, which become more significant at greater distances among UEs and base stations. Similarly, the uplink throughput (bottom part of Figure 3.12) decreases as the distance between base stations and UEs increases.

- *Statistical analysis.* To illustrate the statistical properties of SCOPE experiments, and demonstrate how results do not vary significantly across multiple experiment repetitions. we performed more than 60 repetitions ($>$ 10 hours) varying the distance among base stations and UEs, and the mobility of UEs.

To present our results, we resort to violin plots, which show both the Probability Density Function (PDF) and distribution of the data measurements across several realizations (shaded areas in the figures), as well as their median (white dots). The black boxes show the 95% confidence intervals.
a) *Static case.* Results for static cellular scenarios are shown in Figure 3.13. In this case, UEs do not move but they are placed at different distances from the base stations (*close*, *medium* and *far*).

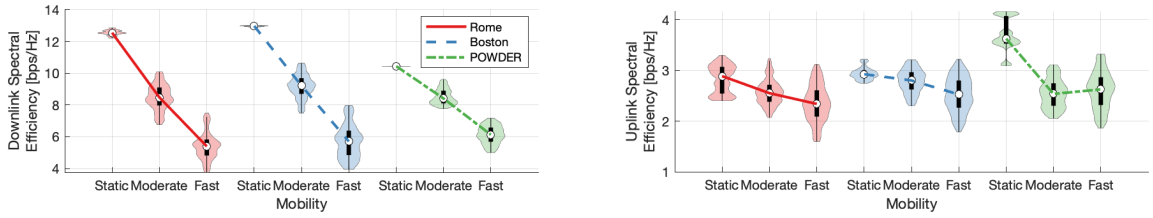


(a) Downlink spectral efficiency

(b) Uplink spectral efficiency

Figure 3.13: Spectral efficiency in the static scenarios.

Metrics for downlink and uplink performance are shown in Figures 3.13a and 3.13b, respectively. As expected, the spectral efficiency decreases for both downlink and uplink as the UEs are placed further away from the base stations. However, despite few outliers, in all cases the data are distributed around the median (white dots in the figures), and exhibit tight 95% confidence intervals (black boxes).
b) *Dynamic case.* Figure 3.14 shows the above metrics in the case of mobile nodes (*static*, *moderate*, and *fast* UE mobility).



(a) Downlink spectral efficiency

(b) Uplink spectral efficiency

Figure 3.14: Spectral efficiency in the dynamic scenarios.

The downlink performance is shown in Figure 3.14a; Figure 3.14b shows the uplink performance. Since the higher the speed, the more likely UEs are to move away from the serving base station, we observe a drop in performance as the speed increases. However, as for the static case (Figure 3.13), data are distributed around the median (white dots) with small 95% confidence intervals (black boxes). This demonstrates that several realizations of the same experiments achieve comparable results.

3.2.4 SCOPE Use Cases

In this section, we discuss relevant SCOPE use cases and examples of interest to the research community ranging from ML (Section 3.2.4.1) to traditional optimization control techniques (Section 3.2.4.2). We also demonstrate how SCOPE can be ported seamlessly on heterogeneous testbeds (Section 3.2.4.3). These include Arena (which will be detailed in Section 3.5) and POWDER.

3.2.4.1 Machine Learning

Our first use case is that of a researcher utilizing SCOPE to implement ML-based control algorithms for cellular networks (see Figure 3.15). In the user domain, ML/AI algorithms are designed and

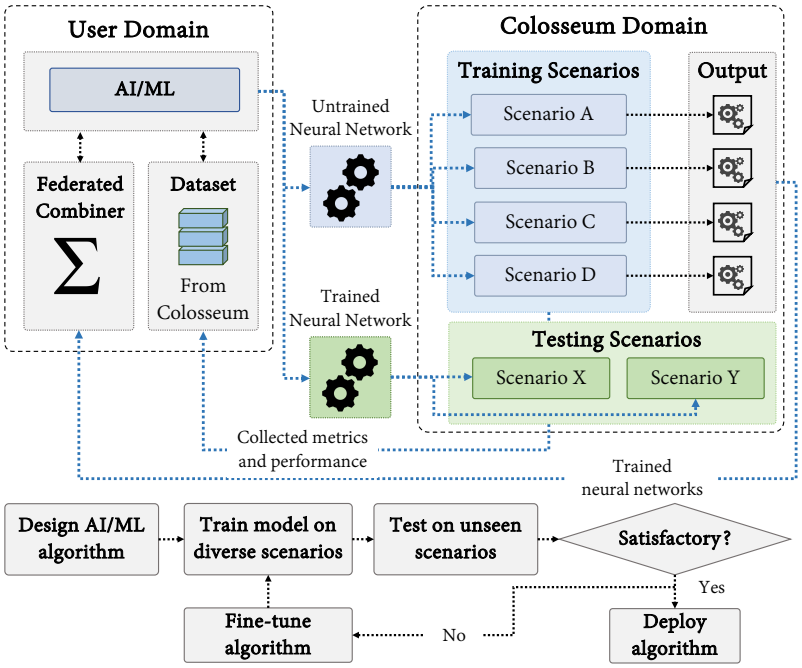


Figure 3.15: Machine learning use case.

interfaced with SCOPE through its open APIs (Section 3.2.2). The resulting version of the SCOPE container, which includes the user-defined control logic, is then transferred to Colosseum, where it is used to run experiments. Being able to run experiments on multiple scenarios enables a variety of ML applications. For example, researchers can train multiple copies of the same *neural network* on a subset of the available scenarios (*training scenarios* in Figure 3.15) leveraging the *metrics and performance dataset* generated by SCOPE. Then, they can use federated learning techniques to combine weights and develop more general models [271].

After the training is completed, the neural networks can be tested on a completely different and unseen set of scenarios (*testing scenarios* in the figure). This makes it possible to validate the generalization capabilities of the trained model and eventually fine-tune its weights, if necessary. Finally, after the devised algorithms work as expected, the model can be exported from SCOPE and deployed on production cellular networks or any other LXC-enabled testbed.

SCOPE for Deep Reinforcement Learning. To provide a practical example of how SCOPE can be used to prototype machine learning algorithms, we implemented a simple yet effective DRL algorithm [79, 272] using SCOPE container and scenarios. As shown in Figure 3.16, we trained a Deep Q-Network (DQN) agent—a well-established DRL solution for problems with discrete actions [273]—that reads periodically the metrics stored in the base station dataset and adapts slicing and scheduling strategies at run time to maximize the network throughput. In this case, each base station of the network hosts a dedicated DRL agent.

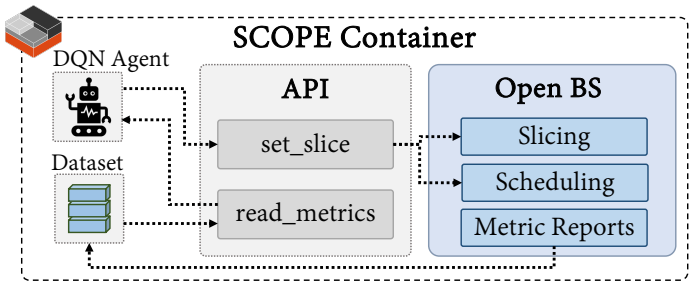


Figure 3.16: Deep Q-Network agent.

We considered a network with two different slices. The agent is required to select how many PRBs to allocate to each slice and which scheduling algorithm should be used to serve the UEs of the slice. The actions taken by the agent are then enforced via SCOPE (Figure 3.16), which reconfigures the base stations in real time. The state of the agent is generated by periodically reading the dataset entries corresponding to the most recent 10 s of the experiment (this frequency can be tuned as appropriate). These are then fed to the encoding portion of an autoencoder trained to create a latent representation (and thus with lower dimension) of the state of the system [274]. Due to space limitations, and since this is not the focus of this section, we refrain from providing a detailed description of the DQN and autoencoder implementations.

We consider two different control configurations. In the first one, the agent makes decisions on the scheduling policy of each slice only. In the second, the agent controls both slicing and scheduling policies. First, we report the downlink throughput measured during the training of

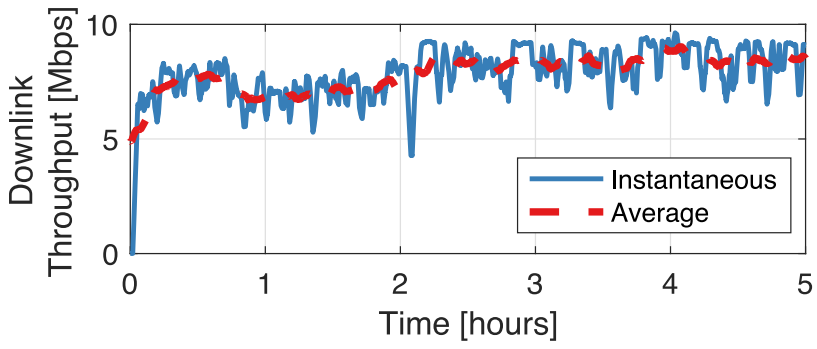
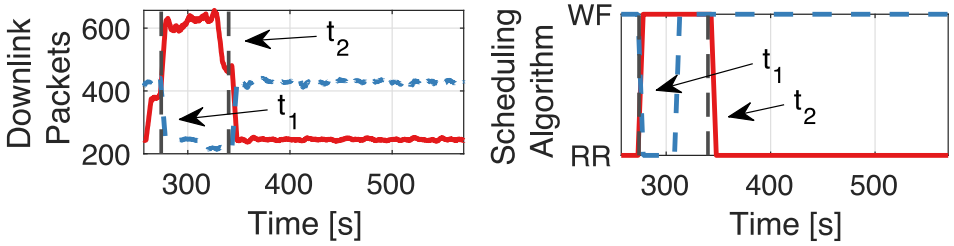


Figure 3.17: Downlink throughput of the DQN agent as a function of the training time with scheduling decisions only.

the agent (Figure 3.17) under the first configuration. Then, we show the number of packets to be transmitted to the UEs, PRBs and scheduling policy of each slice, and spectral efficiency for the two configurations (Figures 3.18 and 3.19). Finally, we compare the trained DQN agent with the case in which static scheduling policies are adopted by the network (Figure 3.20).

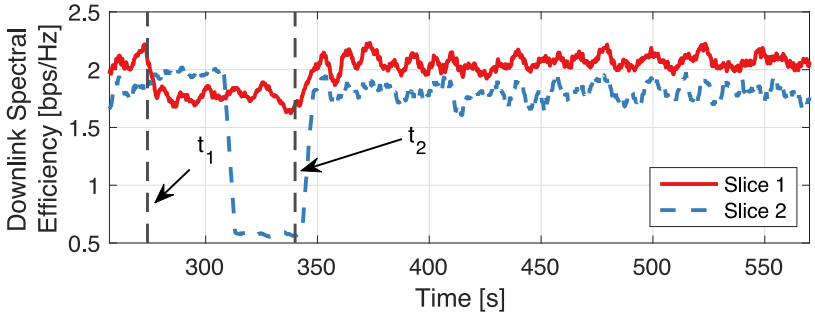
Recall that, at the beginning of the training, the agent is initialized with random weights. As shown in Figure 3.17, this means that actions computed in the first few epochs are taken at random and are generally sub-optimal. As the training goes on, the agent learns how to effectively select strategies that achieve higher throughput values. After 2 hours of training, for instance, the agent is already capable of selecting actions that result in improved performance.

Figure 3.18 shows the results obtained when we test the DQN agent previously trained to make scheduling decisions for each slice only (i.e., no decisions on resource allocation).



(a) Downlink packets

(b) Scheduling algorithm

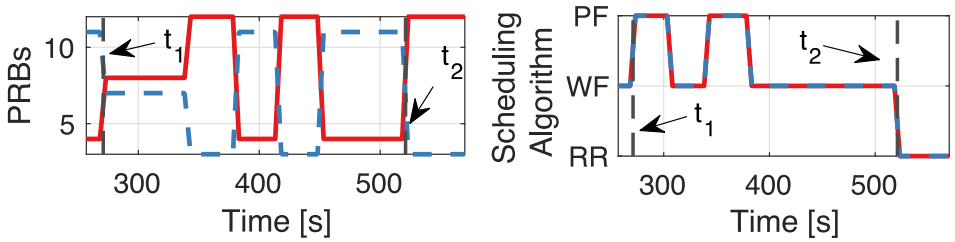


(c) Downlink spectral efficiency

Figure 3.18: Machine learning use case with scheduling decision policies.

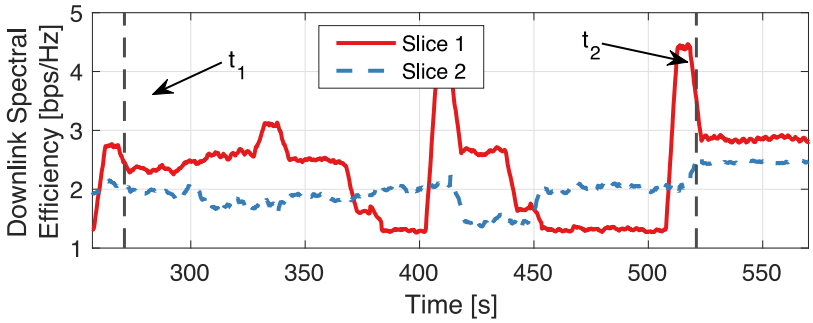
At the beginning of the testing experiment, the trained DQN agent selects the round-robin (RR) and waterfilling (WF) scheduling algorithms for slices 1 and 2, respectively. As the number of packets for the UEs of slice 1 increases (solid red line in Figure 3.18a), the agent changes the scheduling policy of the slice from round-robin to a fairer waterfilling (t_1, Figure 3.18b). This action allows the network to maintain a good level of spectral efficiency (Figure 3.18c). On the other hand, the scheduling algorithm of slice 2, which sees a 20 s decrease in the packet arrivals for the UEs (dashed blue line in Figure 3.18a) is changed from waterfilling to round-robin, and changed back to waterfilling shortly after. Finally, as the burst of packets for the UEs of slice 1 ends, the scheduling policy of the slice is changed back to round-robin (t_2). We observe that the brief decrease in the spectral efficiency of slice 2 (Figure 3.18c) corresponds to a short time window with fewer packet arrivals (see Figure 3.18a).

Results obtained by testing on Colosseum the DQN agent trained to control both scheduling and slicing policies are shown in Figure 3.19. The DQN agent makes decisions on both PRB allocation, which affects the resources of the slices (Figure 3.19a), and scheduling policy (Figure 3.19b) of the two slices. Both decisions reflect on the downlink spectral efficiency (Figure 3.19c), which the agent tries to balance between the two slices.



(a) PRB allocation

(b) Scheduling algorithm



(c) Downlink spectral efficiency

Figure 3.19: Machine learning use case with scheduling and slicing decision policies.

When the testing experiment starts, both slices are served via the waterfilling scheduling algorithm (Figure 3.19b). Then, at time instant t_1, the DQN agent observes the state of the network and modifies the resource allocation of each slice. As a result, the PRBs of slice 1 are increased, while those of slice 2 decreased (Figure 3.19a). Additionally, the agent selects the proportionally fair (PF) scheduling algorithm for both slices. On the one hand, this causes the downlink spectral efficiency of slice 1 to increase (Figure 3.19c). On the other hand, slice 2 achieves the same spectral efficiency utilizing fewer resources. Similar decisions are made throughout the experiment to balance the spectral efficiency of the two slices. Finally, at time t_2, the DQN reallocates the PRBs of the two slices (11 PRBs for slice 1, and 4 for slice 2, recall that the base station uses 15 PRBs), and sets their scheduling policies to round-robin (Figure 3.19b). This results in the two slices achieving similar, i.e., fair, levels of spectral efficiency.

Finally, Figure 3.20 compares the network performance obtained running the DQN agent with scheduling decisions with that of the network running round-robin, waterfilling and proportionally fair scheduling policies. We notice that the sequential decision making of the DQN agent allows it to improve the overall downlink throughput with respect to fixed scheduling policies, thus adapting to the varying channel and traffic conditions.

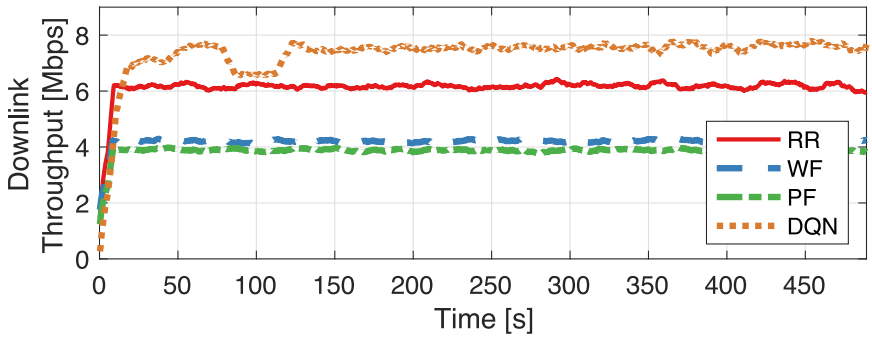


Figure 3.20: Downlink throughput of the DQN agent with scheduling decisions vs. fixed scheduling policies.

3.2.4.2 Optimization and Heuristics

A major issue with many optimization algorithms and heuristics is that they often rely upon analytical models, assumptions and approximations that do not accurately reflect the real network behavior. SCOPE can help researchers refine these models and assessing their accuracy in real-world applications. A possible use case is shown in Figure 3.21. Similar to the ML use case (Section 3.2.4.1),

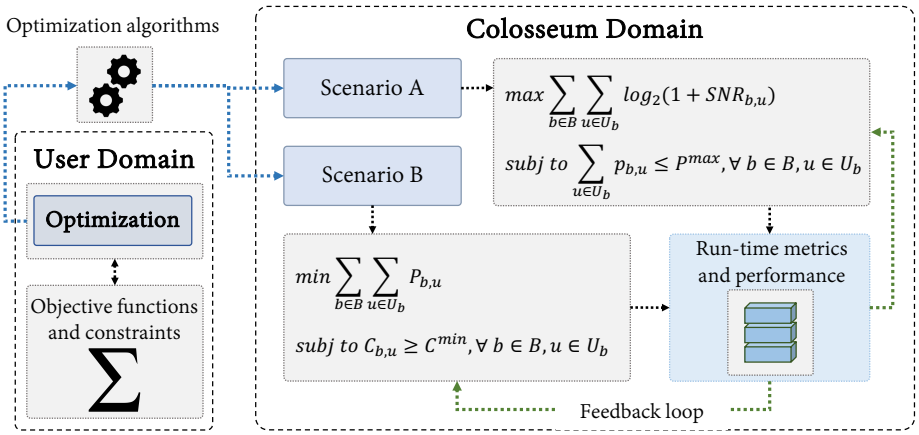


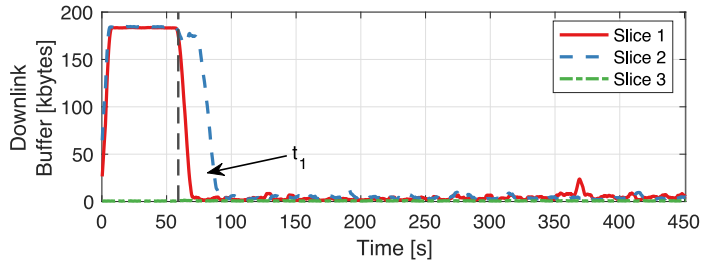
Figure 3.21: Optimization use case.

SCOPE can be customized to include optimization- and heuristic-based control logic. This control logic can leverage SCOPE to interface with the base stations and control their configuration at run time. The customized container can then be uploaded to Colosseum and used to test different optimization/heuristic objectives. At the experiment run time, the user algorithms can leverage SCOPE metrics and performance dataset to have a feedback on the behavior of the network. Decisions can be made on the policies of the network (e.g., scheduling and slicing), with subsequent run-time reconfiguration of the base stations. Finally, when the user policies reach a satisfactory behavior, they can be reliably deployed on commercial cellular networks.

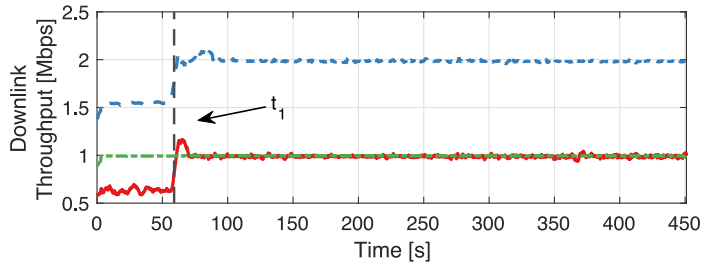
For the sake of illustration, in Figure 3.22 we show the downlink buffer size and throughput obtained by implementing a heuristic algorithm on SCOPE. We consider the case in which base stations serve three slices with different classes of traffic, UEs and QoS requirements: (1) one UE

generating URLLC traffic with low-latency requirements; (2) two UEs with eMBB traffic with high data rates, and (3) one UE generating MTC traffic with loose latency and data rate requirements.

The heuristic algorithm periodically reads data from SCOPE dataset and tunes the slice resources accordingly. Slice 1 (URLLC) is given additional resources when the base station has more packets to transmit to the UEs (i.e., when the size of the transmission buffer increases) to enable prompt communications. On the other hand, slice 2 (eMBB) is allocated more resources when the available bandwidth of the slice saturates. Finally, due to its loose performance requirements, slice 3 (MTC) is served with fixed scheduling policies and is allocated two PRBs for the entire duration of the experiment.



(a) Downlink transmit buffer



(b) Downlink throughput

Figure 3.22: Optimization use case for different classes of traffic: URLLC (slice 1), eMBB (slice 2), and MTC (slice 3).

As soon as the experiment starts, there is a surge in the traffic of slices 1 and 2, which causes an increase in the downlink buffer (Figure 3.22a). Following the above policy, resources of both slices are increased at time t_1 to prevent congestion of the transmission buffers. Additionally, the scheduling policy of slice 2 is also changed from round-robin to a fairer waterfilling. Because of this, at time t_1 we observe a prompt decrease in the buffer size of both slices and an increase of throughput (Figures 3.22a and 3.22b, respectively).

3.2.4.3 SCOPE Portability

In this section we illustrate how SCOPE can be ported to different testbeds. We start by prototyping an instance of SCOPE with 8 base stations serving 32 UEs on Colosseum. We then port it to Arena—an indoor office testbed that will be discussed in Section 3.5—and to POWDER—an outdoor large-scale platform (described in Section 2.7) that is part of the U.S. PAWR program [5]. In both

testbeds the base station serves 2 UEs located at an average distance of 4.5 m for Arena, and 345 m for POWDER. In Arena the antennas of all devices (USRPs X310) are at the same height (hung off the ceiling); in POWDER the base station (USRP X310) is located on the rooftop of a 30 m-tall building and serves ground-level UEs (USRPs B210).

Since SCOPE uses virtualized LXC containers, porting it to different testbeds involves transferring the container to the target testbed, and configuring LXC to bridge the interfaces of the host machine to the container [275]. As in SCOPE the actual communication with the hardware radio is left to srsRAN, different SDRs can be used by installing the required drivers [276].

In this experiment, we measure the average downlink spectral efficiency of the base stations. Results are shown in Figure 3.23.

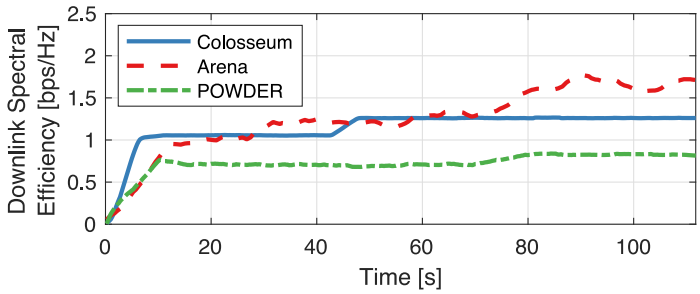


Figure 3.23: Downlink spectral efficiency of SCOPE ported on three different testbeds: Colosseum, Arena and POWDER.

We notice that SCOPE *achieves performances showing similar trends in three very different experimental facilities*: a wireless emulator, an indoor testbed and a large-scale outdoor platform. We also observe that SCOPE not only adapts to diverse environments, but also to different numbers of UEs. This demonstrates the *feasibility and effectiveness of prototyping solutions with SCOPE on Colosseum before testing them in the field.*

3.2.5 Related Work

We are aware of only a handful of works concerning network slicing implementations for srsRAN-based cellular networks. Garcia-Aviles et al. propose a multi-slice service-orchestration framework [277] and implement it on a small-scale prototype [278], while Ayala-Romero et al. devise a deep learning approach for joint allocation of computational and radio resources [279]. Furthermore, Appendix A proposes a MEC framework for resource orchestration on heterogeneous network slices, while Appendix B demonstrates optimal network slicing solutions for small-scale 5G network deployments.

A proof of concept of RAN slicing on the 5G-EmPOWER platform is given by Coronado et al. [280]. Although this work offers interesting insights, the provided implementation considers only a single base station and two UEs. Koutlia et al. describe an experimental testbed with slicing support [281]. The focus of this work is primarily on policy enforcement, slice provisioning and admission control. Moreover, slices are statically allocated and they cannot be reconfigured at run time.

Finally, a centralized and cloud-based slicing framework based on network function virtualization is developed by Marinova et al. [282]. While network parameters can be reconfigured at run time, this comes at the cost of restarting and redeploying the network base stations and subsequent re-attachment of UEs. This operation requires a network downtime tens of seconds long, hardly acceptable given the sub-millisecond latency requirements of NextG networks. Furthermore, no open APIs are offered to interface user-defined control logic.

SCOPE positions itself as a prototyping platform for sub-6 GHz NextG solutions that complements the testbeds detailed in Section 2.7 by providing open APIs for real-time reconfiguration of cross-layer RAN functionalities along with a portable and platform-independent containerized implementation. If instantiated on such testbeds, SCOPE is able to target indoor and outdoor wireless deployments by seamlessly adapting to their underlying physical infrastructure.

Looking at the broader NextG scene, in the last few years several industry consortia have been focusing on developing solutions to redesign and revolutionize cellular networks. The most noteworthy of these efforts is O-RAN, which disaggregates network functionalities and enables their virtualized execution on different hardware components [11]. This is done through O-RAN RAN Intelligent Controller (RIC), which enables centralized control of the RAN at different time scales and granularities (see Chapter 2). This component can also host third-party applications, called *xApps*, which interact with the RAN APIs to control the 3GPP CUs and DUs [53]. It is worth mentioning that SCOPE is *O-RAN-ready*. Indeed SCOPE APIs can seamlessly interface with the RAN by using similar routines and structures as the ones defined for O-RAN xApps, and control the network elements at run time (see Chapter 4).

3.3 ColO-RAN: Data-driven xApps for the Open RAN

Intelligent, dynamic network optimization via add-on software xApps is clearly a key enabler for future network automation. However, it also introduces novel practical challenges concerning, for instance, the deployment of data-driven ML control solutions at scale. Domain-specific challenges stem from considering the constraints of standardized RANs, the very nature of the wireless ecosystem and the complex interplay among different elements of the networking stack. These challenges, all yet to be addressed in practical RAN deployments, include:

1) Collecting datasets at scale. Datasets for ML training at scale need to be collected and curated to accurately represent the intrinsic randomness and behavior of real-world RANs.

2) Testing ML-based control at scale. Even if ML algorithms are trained on properly collected data, it is necessary to assess their robustness at scale, especially when considering closed-loop control, to prevent poorly-designed data-driven solutions from causing outages or sub-optimal performance.

3) Designing efficient ML agents with unreliable input and constrained output. In production systems, real-time collection of data from the RAN may be inconsistent (e.g., with varying periodicity) or incomplete (e.g., missing entries), and control actions may be constrained by standard specification.

4) Designing ML agents capable of generalizing. Agents should be able to generalize and adapt to unseen deployment configurations not part of the training set.

5) Selecting meaningful features. Features should be accurately selected to provide a meaningful representation of the network status without incurring into dimensionality issues.

To address these key challenges, in this section we describe the design of DRL-based xApps for closed-loop control in O-RAN and their testing with ColO-RAN, a first-of-its-kind softwarized pipeline for large-scale experimental platforms. Based on this experience, we review and discuss key insights in the domain of ML design for O-RAN networks. Our contributions are as follows:

- We introduce ColO-RAN, a first-of-its-kind open, large-scale, experimental O-RAN framework for training and testing ML solutions for next-generation RANs. It combines O-RAN components, a softwarized RAN framework (i.e., SCOPE, see Section 3.2), and Colosseum, the world's largest, open, and publicly-available wireless network emulator based on SDRs (see Section 3.4). ColO-RAN leverages Colosseum as a *wireless data factory* to generate large-scale datasets for ML training in a variety of RF environments, taking into account propagation and fading characteristics of real-world deployments. The ML models are deployed as xApps on the near-real-time RIC, which connects to RAN nodes through O-RAN-compliant interfaces for data collection and closed-loop control. ColO-RAN is the first platform that enables wireless researchers to deploy ML solutions on a full-stack, fully virtualized O-RAN environment which integrates large-scale data collection and DRL testing capabilities with SDRs. Moreover, the lightweight, containerized implementation of ColO-RAN is easily portable to other experimental platforms. *ColO-RAN and the dataset created for this section are publicly available to the research community.*[7]

- We develop three xApps for closed-loop control of RAN scheduling and slicing policies, and for the online training of DRL agents on live production environments. We propose an innovative xApp design based on the combination of a unified interface to the near-real-time RIC for data and control messaging, and a data-driven unit with an autoencoder with the DRL agent. This simplifies the design and prototyping of xApps, which share the same interface but are equipped with different intelligent logic. In addition, the combination of the autoencoder and of the DRL agents based on an actor-critic setup with Proximal Policy Optimization (PPO) improves the resilience and robustness to real, imperfect network telemetry, as well as the effectiveness of the policy selection, and the training convergence. We then utilize ColO-RAN to provide insights on the performance of the DRL agents for adaptive RAN control at scale. We train the autoencoders and agents over a 3.4 GB dataset with more than 73 hours of live RAN performance traces, and perform one of the first evaluations of DRL agents autonomously driving a programmable, software-defined RAN with 49 nodes. The results of our large-scale experimental evaluation include new understandings of data analysis, feature selection, and modeling of control actions for DRL agents, and insights on design strategies to train ML algorithms that generalize and operate even with unreliable data.

- We analyze the trade-offs of training of DRL agents on live networks using Colosseum and Arena (a publicly-available indoor testbed for spectrum research described in Section 3.5) with commercial smartphones. We profile the RAN performance during the DRL exploration phase and after the training, showing how an extra online training step adapts a pre-trained model to

[7]The ColO-RAN source code is available at `https://github.com/wineslab/colosseum-near-rt-ric` and `https://github.com/wineslab/colosseum-scope-e2`. The dataset is available at `https://github.com/wineslab/colosseum-oran-coloran-dataset`.

deployment-specific parameters, fine-tuning its weights at the cost of a temporary performance degradation in the online exploration phase.

- We discuss and review the lessons learned on multiple levels. First, we consider the *system-level insights* that we gathered while building the ColO-RAN framework and while defining the data and control pipelines that support DRL-based control on a live RAN. Second, we summarize the takeaways on the design of effective *machine-learning-based RAN control*, spanning from the design to the deployment of DRL agents for RAN control. Lessons learned from our work highlight (i) the importance of end-to-end experimental frameworks for the data collection, training, and testing of intelligent RAN control solutions; (ii) the effectiveness of adaptive control policies over static configurations, even if the latter are optimized; (iii) the impact of different design choices of DRL agents on end-to-end network performance, and (iv) the trade-offs associated to online DRL training in wireless environments.

We believe that these insights and the research infrastructure developed in this work can catalyze, promote and further the deployment of ML-enabled control loops in next generation networks.

The rest of this section is organized as follows. Section 3.3.1 describes the development of ML solutions in O-RAN-based networks. Section 3.3.2 introduces ColO-RAN, and Section 3.3.3 presents the xApp, DRL agent design, and the data collection campaign for offline training. Large-scale evaluation and lessons learned are discussed in Sections 3.3.4 and 3.3.5. Section 3.3.6 reviews related work. Finally, Section 3.3.7 reviews the main lessons learned.

3.3.1 Machine Learning for the Open RAN

The deployment of machine learning models in wireless networks is a multi-step process (Figure 3.24). It involves a data collection step, the design of the model, its offline or online training and deployment

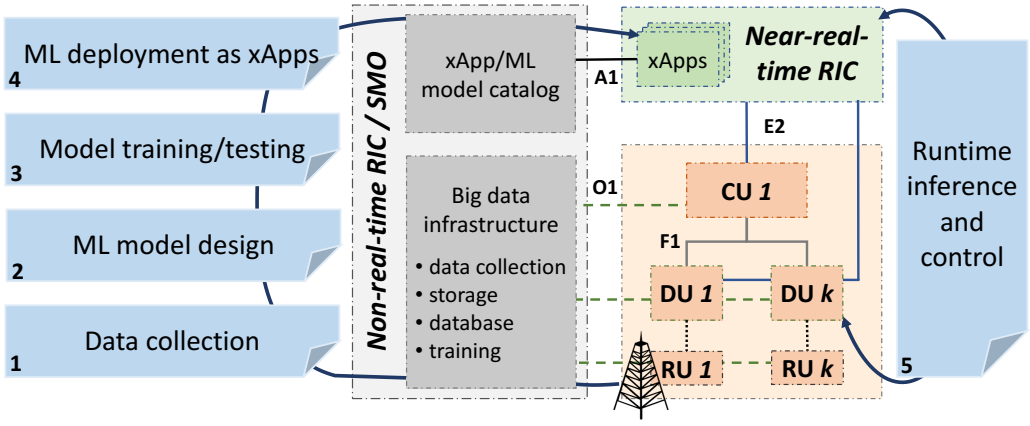


Figure 3.24: The O-RAN architecture and the workflow for the design, development and deployment of ML applications in next generation wireless networks.

for runtime inference and control. The O-RAN architecture, also shown in Figure 3.24, has been developed to aid the overall deployment process, focusing on open interfaces for data collection

and deployment steps. In the following, we describe the O-RAN architecture, and discuss how it facilitates training and deploying ML models in the RAN.

3.3.1.1 ML Pipelines in O-RAN

The O-RAN specifications include guidelines for the management of ML models in cellular networks. Use cases and applications include QoS optimization and prediction, traffic steering, handover, and radio fingerprinting [283]. The specifications describe the ML workflow for O-RAN through five steps (Figure 3.24): (1) data collection; (2) model design; (3) model training and testing; (4) model deployment as xApp, and (5) runtime inference and control.

First, data is collected for different configurations and setups of the RAN (e.g., large/small scale, different traffic, step 1). Data is generated by the RAN nodes, i.e., CUs, DUs and RUs, and streamed to the non-real-time RIC through the O1 interface, where it is organized in large datasets. After enough data has been collected, a ML model is designed (step 2). This entails the following: (i) identifying the RAN parameters to input to the model (e.g., throughput, latency, etc.); (ii) identifying the RAN parameters to control as output (e.g., RAN slicing and scheduling policies), and (iii) the actual ML algorithm implementation. Once the model has been designed and implemented, it is trained and tested on the collected data (step 3). This involves selecting the model hyperparameters (e.g., the depth and number of layers of the neural network) and training the model on a portion of the collected data until a (satisfactory) level of convergence of the model has been reached. After the model has been trained, it is tested on an unseen portion of the collected data to verify that it is able to generalize and react to potentially unforeseen situations. Then, the model is packaged into an xApp ready to run on the near-real-time RIC (step 4). After the xApp has been created, it is deployed on the O-RAN infrastructure. In this phase, the model is first stored in the xApp catalog of the non-real-time RIC, and then instantiated on demand on the near-real-time RIC, where it is interfaced with the RAN through the E2 interface to perform runtime inference and control based on the current network conditions (step 5).

3.3.2 Enabling Large-scale ML Research with O-RAN and Colosseum

The ML pipeline described in Section 3.3.1.1 involves a number of critical steps whose execution requires joint access to *comprehensive datasets* and *testing facilities at scale*, still largely unavailable to the research community. In fact, even major telecom operators or infrastructure owners might not be able to dedicate (parts of) their extensive commercial networks to training and testing of ML algorithms. This stems from the lack of adequate solutions to separate testing from commercial service and to prevent performance degradation. As a consequence, researchers and innovators are constrained to work with small ad hoc datasets collected in contained lab setups, resulting in solutions that hardly generalize to real-world deployments [284].

To address this limitation, here we introduce ColO-RAN, a large-scale research infrastructure built upon the Colosseum network emulator to train, deploy, and test state-of-the-art wireless ML solutions. We first introduce the implementation of the ColO-RAN virtualized O-RAN infrastructure on Colosseum (Section 3.3.2.1) and of the xApps we designed (Section 3.3.3). Then, we finally describe the scenario for data collection that we use to illustrate the usage of ColO-RAN (Section 3.3.3.3).

3.3.2.1 O-RAN-based Colosseum Machine Learning Infrastructure

Besides enabling large-scale data collection, Colosseum also provides a hybrid RF and compute environment for the deployment of ColO-RAN, a complete end-to-end ML infrastructure. ColO-RAN provides researchers with a ready-to-use environment to develop and test ML solutions, following the steps of Figure 3.24 (Section 3.3.1.1). These include the deployment on a 3GPP-compliant RAN, testing in heterogeneous emulated environments, and an O-RAN-compliant infrastructure. With respect to other open source implementations of the O-RAN infrastructure, ColO-RAN features a more lightweight footprint (e.g., it does not require a full Kubernetes deployment, contrary to the OSC RIC), and it can be ported to other testbeds, e.g., Arena [522], with minimal changes, thanks to its virtualized and container-based implementation. As a further contribution, this platform has been made openly available to the research community.

The software, compute and networking components of our end-to-end infrastructure are shown in Figure 3.25. The SMO (left) features three compute nodes to train large ML models, 64 Terabyte

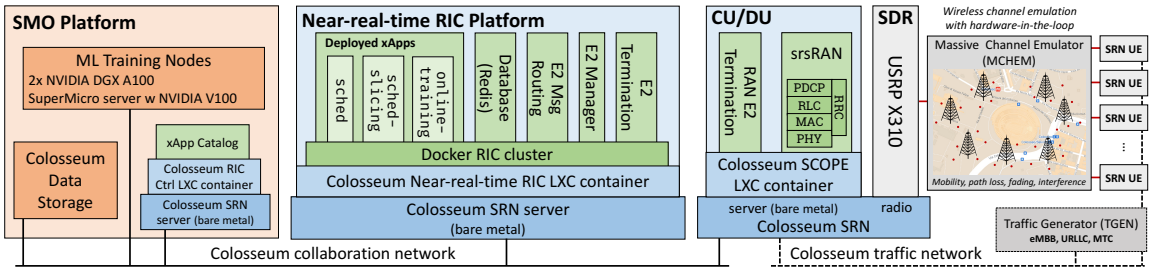


Figure 3.25: Integration of the O-RAN infrastructure in Colosseum.

of storage for models and datasets, and the xApp catalog. The near-real-time RIC (Figure 3.25, center) provides E2 connectivity to the RAN and support for multiple xApps interacting with the base stations. It is implemented as a standalone LXC that can be deployed on a Colosseum SRN.[8] It includes multiple Docker containers for the *E2 termination* and *manager*, the *E2 message routing* to handle messages internal to the RIC, a *Redis database*, which keeps a record of the nodes connected to the RIC, and the *xApps* (Section 3.3.3). The implementation of the near-real-time RIC is based on the Bronze release of the OSC [147]. The OSC near-real-time RIC was adapted into a minimal version, which does not require a Kubernetes cluster, and can fit in a lightweight LXC container. We also extended the OSC codebase to support concurrent connections from multiple base stations and xApps, and to provide improved support for encoding, decoding and routing of control messages.

The near-real-time RIC connects to the RAN base stations through the E2 interface (Figure 3.25, right). The base stations leverage a joint implementation of the 3GPP DUs and CUs. These nodes run our SCOPE framework co-located with srsRAN [27]. Specifically, SCOPE can tune certain capabilities of the srsRAN base stations at run-time, thus modifying their configuration. Examples of this are the amount of resources, expressed as number of PRBs, to allocate to each slice, or the scheduling policy that each base stations should adopt on each slice. Additionally, we extended SCOPE to access relevant network KPMs and forward them to the near-real-time RIC through a custom, standard-compliant E2 termination. The latter extends the capabilities of the E2 termination

[8] https://github.com/wineslab/colosseum-near-rt-ric

Table 3.7: Catalog of the three developed xApps.

xApp	Functionality	Input (Observation)	Output (Action)	ML Models	Utility (Reward)
`sched-slicing`	Single-DRL-agent for joint slicing and scheduling control	Rate, buffer size, PHY TBs (DL)	PRB and scheduling policy for each slice	DRL-base, DRL-reduced-actions, DRL-no-autoencoder	Maximize rate for eMBB, PHY TBs for MTC, minimize buffer size for URLLC
`sched`	Multi-DRL-agent per-slice scheduling policy selection	Rate, buffer size, PRB ratio (DL)	Scheduling policy for each slice	DRL-sched	Maximize rate for eMBB and MTC, PRB ratio for URLLC
`online-training`	Train DRL agents with online exploration	Rate, buffer size, PHY TBs (DL)	Training action (PRB and scheduling)	Trained online by the xApp itself	Based on specific training goals

in [285], making it possible to reconfigure base stations directly from the near-real-time RIC and to perform an automatic and periodic data reporting and collection.[9] The E2 termination allows the setup procedure and registration of the base stations with the near-real-time RIC. Our implementation also features two custom SMs (as discussed next) for trigger-based or periodic reporting, and control events in the base stations. This effectively enables data-driven real-time control loops between the base stations and the xApps. The RAN supports network slicing with 3 slices for different QoS: (i) eMBB, representing users requesting video traffic; (ii) MTC for sensing applications, and (iii) URLLC for latency-constrained applications. Hence, we characterize our slices based on the type of traffic requested by its users, rather than based on a feature of the RAN itself. As slices represent a specific type of service the operators agree to provide to their subscribers (e.g., as part of SLAs), they are pre-instantiated on the base stations, and users are statically assigned to one of such slices based on the purchased service level. For each slice, the base stations can adopt 3 different scheduling policies independently of that of the other slices, namely, the Round Robin (RR), the Wired-first (WF), and the Proportional Fair (PF) scheduling policies. These policies were selected as they represent popular scheduling strategies in wireless deployments [286]. It is worth noticing that we do not directly control the scheduling of the users (i.e., which specific user should be scheduled), for which we would need tighter control loops implemented directly at the base stations [530]. Instead, our control involves the type of scheduling policy run by the base stations for each slice of the network. Finally, the base stations connect to the RF frontends (USRPs X310) that perform signal transmission and reception.

3.3.3 xApp Design for DRL-based Control

The xApps deployed on the near-real-time RIC are the heart of the O-RAN-based RAN control loops. We developed three xApps to evaluate the impact of different ML strategies for closed-loop RAN control (Table 3.7). Each xApp can receive data and control RAN nodes with two custom SMs, which resemble the O-RAN KPM and RAN control SMs [287]. The control actions available to the xApps are the selection of the slicing policy (the number of PRB allocated to each slice) and of the scheduling policy (which scheduler is used for each slice).

[9] `https://github.com/wineslab/colosseum-scope-e2`

The xApps have been developed by extending the OSC basic xApp framework [288], and include two components (Figure 3.26).

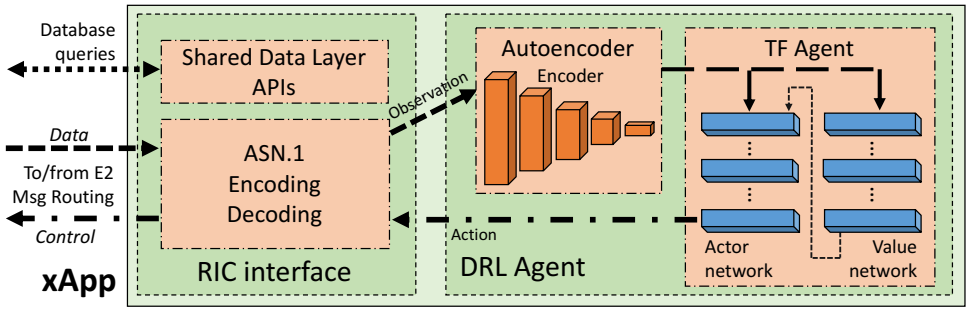


Figure 3.26: Structure of a ColO-RAN xApp.

The first is the interface to the RIC, which implements the SM and performs ASN.1 encoding/decoding of RAN data and control. The second is the ML infrastructure itself, which includes one or more autoencoders and DRL agents. For these, we used TensorFlow 2.4 [289] and the TF-Agents library [290], and we used the neural network architectures described in Section 3.3.3.2.

3.3.3.1 DRL Agent Design

Agent Architecture. The DRL agents considered in this section have been trained using the PPO algorithm [291]. PPO is a well-established on-policy DRL architecture that uses an actor-critic configuration where the *actor* network takes actions according to current network state, and the value network (or *critic*) scores the actions taken by the actor network by observing the reward obtained when taking an action in a specific state of the environment. By leveraging this architecture, the PPO algorithm decouples the action taking process from the evaluation of achieved rewards. This is extremely important to ensure that the actor network can learn an unbiased policy (i.e., a mapping between state and actions) where the actor network selects an action because it is effective in the long run and not only because it occasionally results in high instantaneous rewards that are instead inefficient in the majority of cases.

It is worth mentioning that the actor-critic setup is also important because PPO is an on-policy architecture, which means that the training procedure uses a memory buffer that contains data that is collected by using actions that are taken with the most current version of the actor network. If compared to off-policy algorithms (such as DQNs), which use a memory buffer that store experience collected at any time by the DRL agent, PPO only uses data that is fresh and does not contain experiences from the past, meaning that the memory buffer is emptied every time the actor network is updated during the training phase. This approach is usually slower then others, but together with the actor-critic setup it has been shown to be one of the most efficient and reliable DRL architectures in the literature [291].

Pre-processing Observations via Autoencoders. One of the main causes of slow training of DRL agents is the use of observations with high dimensionality that result in actor and critic networks with many parameters and large state space. Indeed, the RAN produces an extremely large amount of

data which not always provide meaningful insights on the actual state of the system due to redundant information and outliers. To reduce the size of the observation fed to the DRL agent, mitigate outliers and provide a high-quality yet high-level representation of the state of the system, we resort to autoencoders, as also shown in Figure 3.26. Specifically, before being fed to the DRL agents, the data produced by the RAN is processed by the encoding portion of an autoencoder for dimensionality reduction (whose impact on DRL-based control is investigated in Section 3.3.4.2).

Although autoencoders might have several implementations according to the specific applications, autoencoders for dimensionality reduction have an hourglass architecture with an encoder and a decoder components. The former produces a lower dimension representation of the input data (i.e., latent representation) which - if trained properly - can be accurately reconstructed by the decoder portion of the autoencoder with negligible error. The decoder is the specular image of the encoder and the goal of this architecture is to create a reduced version of the input data that contains only relevant information, yet it is accurate enough to be able to reconstruct the original data without any loss. To further reduce the complexity of the DRL agents, we perform feature selection on the metrics that are observed by the agents (see Section 3.3.4 for more details).

Observations, Actions, and Rewards for Each Agent. As mentioned before, although our xApps share the same high-level architecture shown in Figure 3.26, each xApp embeds a different DRL agent whose configuration varies according to the specific goal of the xApp. Specifically, each DRL agent observes different metrics of the RAN, takes diverse actions and aims at maximizing different rewards. The configurations considered in this section are summarized in Table 3.7 and discussed in the following:

- **`sched-slicing` xApp**: this xApp is designed to simultaneously select slicing and scheduling policies for a single base station and all slices (eMBB, MTC, and URLLC). For this xApp we trained three DRL models: a baseline model (DRL-base) able to select any feasible actions (i.e., slicing and scheduling policies), an agent that explores a reduced set of actions (DRL-reduced-actions) and an agent where input data is not processed by the autoencoder but is fed directly to the agent (DRL-no-autoencoder). In this case, the reward of the DRL agent is configured to jointly maximize the rate of the eMBB slice, maximize the number of transmitted packets of the MTC slice, and minimize the buffer size (i.e., a proxy for the data transmission latency) of the URLLC slice.

- **`sched` xApp**: this xApp includes three DRL agents that act in parallel and are responsible for selecting the scheduling policy for each individual slice (DRL-slice). Each agent has been trained using slice-specific rewards. Specifically, the eMBB and MTC agents aim at maximizing the data rate of the controlled slice, while the URLLC agents aims at maximizing the ratio between the number of PRBs being requested by each user and how many are effectively allocated by the scheduler (i.e., the higher the ratio, the faster a user is served, and the lower the latency).

- **`online-training` xApp**: this xApp represents a variation of the above two xApps where the embedded DRL agents are fine-tuned in an online fashion by updating the pre-trained weights according to live data from the RAN by performing exploration steps on the online RAN infrastructure itself. While this is not recommended by O-RAN [283], it specializes the trained model to the specific deployment.

It is worth noticing that while agents for the eMBB and MTC slices respectively optimize the user throughput and transmitted packets—which can be measured at the base stations directly—the URLLC slice aims at maintaining the transmission buffer queues of the base stations as low as possible, or transmit as many URLLC packets as possible. These metrics are a proxy for the latency of the service provided to the users in practice, for which we do not have a direct measure at base station protocol stack.

3.3.3.2 Training the DRL Agents

DRL agents are trained on the dataset described in Section 3.3.3.3, where at each training episode we select RAN data from different base stations to remove dependence on a specific wireless environment (Section 3.3.5) and facilitate generalization.

Following O-RAN specifications, training is performed offline on the dataset. In our case, this is achieved by randomly selecting instances in which the network reaches the state s_1 that results from the combination of the previous state s_0 and the action to explore a_0.

In our experiments, the actor and critic networks of all DRL agents have been implemented as two fully-connected neural networks with 5 layers with 30 neurons each and an hyperbolic tangent activation function. The encoder consists of 4 fully-connected layers with 256, 128, 32 and 3 neurons and a rectified linear activation function. Moreover, the input size of the autoencoder is a matrix of size $(10, 3)$, where 3 represents the input KPMs relevant to the specific DRL agent (as specified in Table 3.7) and 10 represents the number of independent measurements of such KPMs. For all models, the learning rate is set to 0.001.

With respect to the `online-training` xApp, we leverage TensorFlow `CheckPoint` objects to save and restore a pre-trained model for multiple consecutive rounds of training. In this way, the training services in the xApp can restore an agent trained on an offline dataset using it as starting point for the online, live training on the RAN. We discuss the trade-offs involved in this operation in Section 3.3.5.

3.3.3.3 Large-scale Data Collection for ColO-RAN

To train the DRL agents for the ColO-RAN xApps we performed large-scale data collection experiments on Colosseum. The parameters for the scenario are summarized in Table 3.8.

Table 3.8: Configuration parameters for the considered scenario.

Parameter	Value
Number of nodes	$N_{\text{BS}} = 7$, $N_{\text{UE}} = 42$
RF parameters	DL carrier $f_d = 0.98$ GHz, UL carrier $f_u = 1.02$ GHz, bandwidth $B = 10$ MHz (50 PRBs)
Schedulers	RR, WF, PF
Slices	eMBB, MTC, URLLC (2 UEs/BS/slice)
Traffic profiles	Slice-based: 4 Mbps/UE for eMBB, 44.6 kbps/UE for MTC, 89.3 kbps/UE URLLC Uniform: 1.5 Mbps/UE for eMBB, MTC, URLLC

The large-scale RF scenario mimics a real-world cellular deployment in downtown Rome, Italy, with the positions of the base stations derived from the OpenCelliD database [267]. We instantiated a softwarized cellular network with 7 base stations through the SCOPE framework. Each base

station operates on a 10 MHz channel (50 PRBs) which can be dynamically assigned to the 3 slices (i.e., eMBB, MTC, URLLC). Additionally, we considered two different TGEN traffic scenarios: slice-based traffic and uniform traffic. In slice-based traffic, users are distributed among different traffic profiles (4 Mbps constant bitrate traffic to eMBB users, and 44.6 kbps and 89.3 kbps Poisson traffic to MTC and URLLC, respectively). The uniform traffic is configured with 1.5 Mbps for all users. The training of the DRL agents on the offline dataset has been performed with slice-based traffic. Finally, the base stations serve a total of 42 users equally divided among the 3 slices.

In our data collection campaign, we gathered 3.4 GB of data, for a total of more than 73 hours of experiments. In each experiment, the base stations periodically report RAN KPMs to the non-real-time RIC. These include metrics such as throughput, buffer queues, number of PHY TBs and PRBs. The complete dataset features more than 30 metrics that can be used for RAN analysis and ML training.[10]

3.3.4 DRL-based xApp Evaluation

Learning strategies for RAN control are coded as xApps on ColO-RAN. This section presents their comparative performance evaluation. Feature selection based on RAN KPMs is described in Section 3.3.4.1. The experimental comparison of the different DRL models is reported in Section 3.3.4.2.

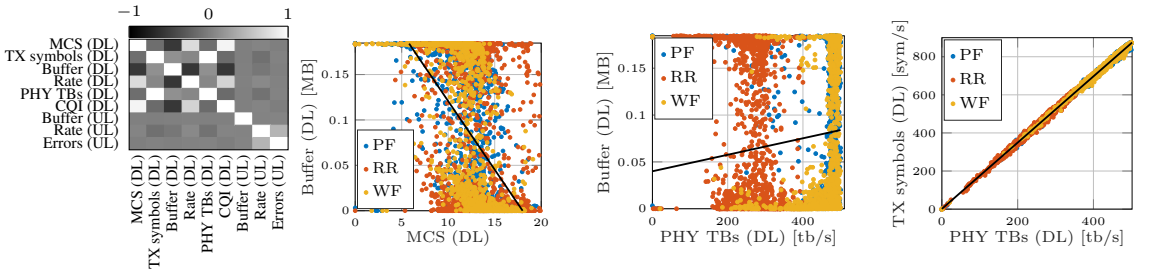
3.3.4.1 RAN KPM and Feature Selection

O-RAN is the first architecture to introduce a standardized way to extract telemetry and data from the RAN to drive closed-loop control. However, O-RAN does not indicate which KPMs should be considered for the design of ML algorithms. The O-RAN E2SM KPM specifications [287] allow the generation of more than 400 possible KPMs, listed in [292, 293]. More vendor-specific KPMs may also be reported on E2. These KPMs range from physical layer metrics to base station monitoring statistics. Therefore, the bulk set of data may not be useful to represent the network state for a specific problem. Additionally, reporting or collecting all the metrics via the E2 or O1 interfaces introduces a high overhead, and a highly dimensional input may lead to sub-optimal performance for ML-driven xApps [294].

Therefore, a key step in the design process of ML-driven xApps is the selection of the features that should be reported for RAN closed-loop control. In this context, the availability of large-scale, heterogeneous datasets and wireless data factories is key to enable feature selection based on a combined expert- and data-driven approach. To better illustrate this, in Figures 3.27 and 3.28 we report a correlation analysis for several metrics collected in the dataset described in Section 3.3.3.3. The correlation analysis helps us identify the KPMs that provide a meaningful description of the network state with minimal redundancy.

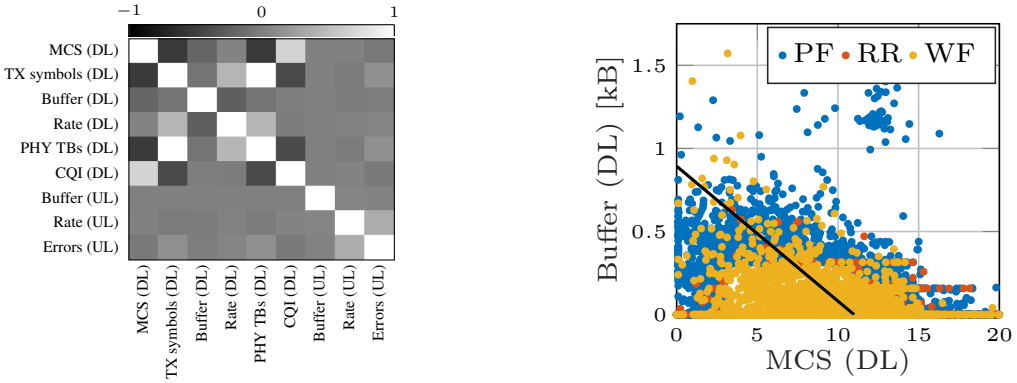
Correlation Analysis. Figure 3.27a shows the correlation matrix of 9 among the 30 UE-specific metrics in the dataset for the eMBB slice. While downlink and uplink metrics exhibit a low correlation, most downlink KPMs positively or negatively correlate with each other (the same holds for uplink KPMs). For example, the downlink MCS and buffer occupancy have a negative correlation (-0.56). This can also be seen in the scatter plot of Figure 3.27b: as the MCS increases, it is less likely to

[10] The dataset is available at `https://github.com/wineslab/colosseum-oran-coloran-dataset`.



(a) Correlation matrix (b) MCS vs. buffer size (c) Number of PHY TB vs. buffer size (d) Number of PHY TB vs. TX symbols

Figure 3.27: Correlation analysis for the eMBB slice with 36 PRBs and the slice-based traffic profile. The solid line is the linear regression fit of the data.



(a) Correlation matrix (b) MCS vs. buffer size in downlink

Figure 3.28: Correlation analysis for the URLLC slice with 11 PRBs and the slice-based traffic profile. The solid line is the linear regression fit of the data.

have a high buffer occupancy, and vice versa. Similarly, the number of TBs and symbols in downlink have a strong positive correlation (0.998), as also shown in Figure 3.27d. Two downlink metrics that do not correlate well, instead, are the number of TBs and the buffer occupancy. Indeed, the amount of data transmitted in each TB varies with the MCS and therefore cannot be used as indicator of how much the buffer will empty after each transmission. Additionally, as shown in Figure 3.27c, the three scheduling policies have a different quantitative behavior, but they all show a low correlation.

eMBB vs. URLLC. The correlation among metrics also depends on the RAN configuration and slice traffic profile. This can be seen by comparing Figure 3.27, which analyzes the eMBB slice with 36 PRBs, and Figure 3.28, which uses telemetry for the URLLC slice with 11 PRBs. With the slice-based traffic, the URLLC users receive data at a rate that is an order of magnitude smaller than that of the eMBB users. As a consequence, the load on the URLLC slice (represented by the buffer occupancy of Figure 3.28b) is lower, and the buffer is quickly drained even with lower MCSs. Consequently, the correlation among the buffer occupancy and the MCS (-0.2) is lower with respect to the eMBB slice. This further makes the case for collecting datasets that are truly representative of a wireless RAN deployment, including heterogeneous traffic and diverse applications.

Summary. Figure 3.27 and 3.28 provide insights on which metrics can be used to describe the RAN status. Since the number of downlink symbols and TBs, or the MCS and the buffer occupancy for the eMBB slice are highly correlated, using them to represent the state of the network only increases the dimensionality of the state without introducing additional information. Conversely, the buffer occupancy and the number of TBs enrich the representation with low redundancy. Therefore, the DRL agents for the xApps in this section consider as input metrics the number of TBs, the buffer occupancy (or the ratio of PRB granted and requested, which has a high correlation with the buffer status), and the downlink rate.

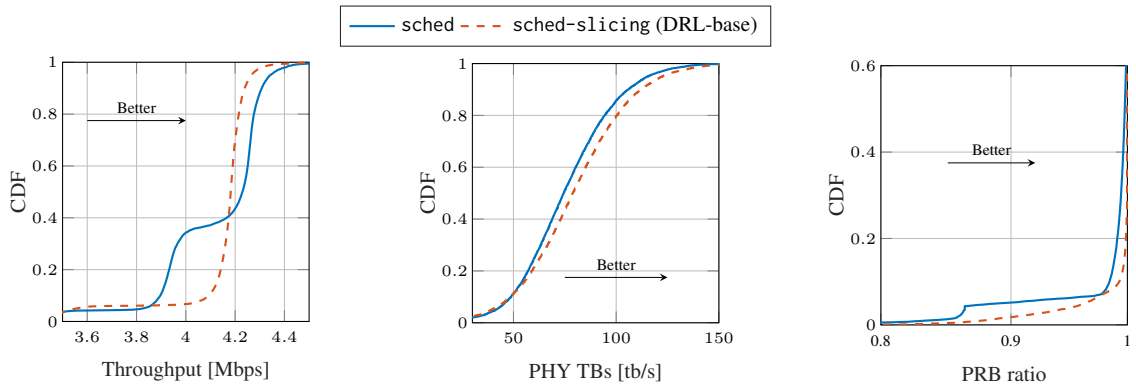
3.3.4.2 Comparing Different DRL-based RAN Control Strategies

Once the input metrics have been selected, the next step in the design of ML applications involves the selection of the proper modeling strategy [283]. In this section, we consider ML models for sequential decision making, and thus focus on DRL algorithms.

Control Policy Selection. In this context, it is clearly crucial to properly select the control knobs, i.e., the RAN parameters that need to be controlled and adapted automatically, and the action space, i.e., the support on which these parameters can change. To this end, Figure 3.29 compares the performance for the `sched` and `sched-slicing` xApps, which perform different control actions. The first assumes a fixed slicing profile and includes three DRL agents that select the scheduling policy for each slice, while the second jointly controls the slicing (i.e., number of PRBs allocated to each slice) and scheduling policies with a single DRL agent. For this comparison, the slicing profile for the `sched` xApp evaluation matches the configuration that is chosen most often by the `sched-slicing` agent, and the source traffic is slice-based. The CDFs of Figure 3.29 show that the joint control of slicing and scheduling improves the relevant metric for each slice, with the most significant improvements in the PRB ratio and in the throughput for the users below the 40th percentile. This shows that there exist edge cases in which adapting the slicing profile further improves the network performance with respect to adaptive schedulers with a static slice configuration, even if the fixed slicing configuration is the one that is chosen most often by the `sched-slicing` xApp.

DRL Agent Design. To further elaborate on the capabilities of `sched-slicing`, in Figure 3.30 we compare results for different configurations of the DRL agent of the xApp, as well as for a static baseline without slicing or scheduling adaptation, using the slice-based traffic. The slicing profile for the static baseline is the one chosen most often by the `sched-slicing` xApp. The results of Figure 3.30 further highlight the performance improvement introduced by adaptive, closed-loop control, with the DRL-driven control outperforming all baselines.

Additionally, this comparison spotlights the importance of careful selection of the action space for the DRL agents. By constraining or expanding the action space that the DRL agents can explore, the xApp designer can bias the selected policies. Consider the DRL-base and DRL-reduced-actions agents (see Table 3.7), whose difference is in the set of actions that the DRL agent can explore. Notably, the DRL-reduced-actions agent lacks the action that results in the policy chosen most often by the DRL-base agent. Compared to the most common action chosen by the DRL-reduced-actions agent (36 PRB for eMBB, 9 for MTC, 5 for URLLC), the most likely policy of DRL-base agent favors the URLLC over the MTC slice (11 vs. 3 PRBs). This is reflected in the performance metrics for the different slices. Notably, DRL-reduced-actions fails to maintain a small buffer and high PRB



(a) eMBB throughput (b) MTC PHY TBs (c) URLLC PRB ratio

Figure 3.29: Comparison between the `sched` and `sched-slicing` xApps, with the slice-based traffic profile. The slicing for the `sched` xApp is fixed and based on the configuration chosen with highest probability by the `sched-slicing` xApp (36 PRBs for eMBB, 3 for MTC, 11 for URLLC).

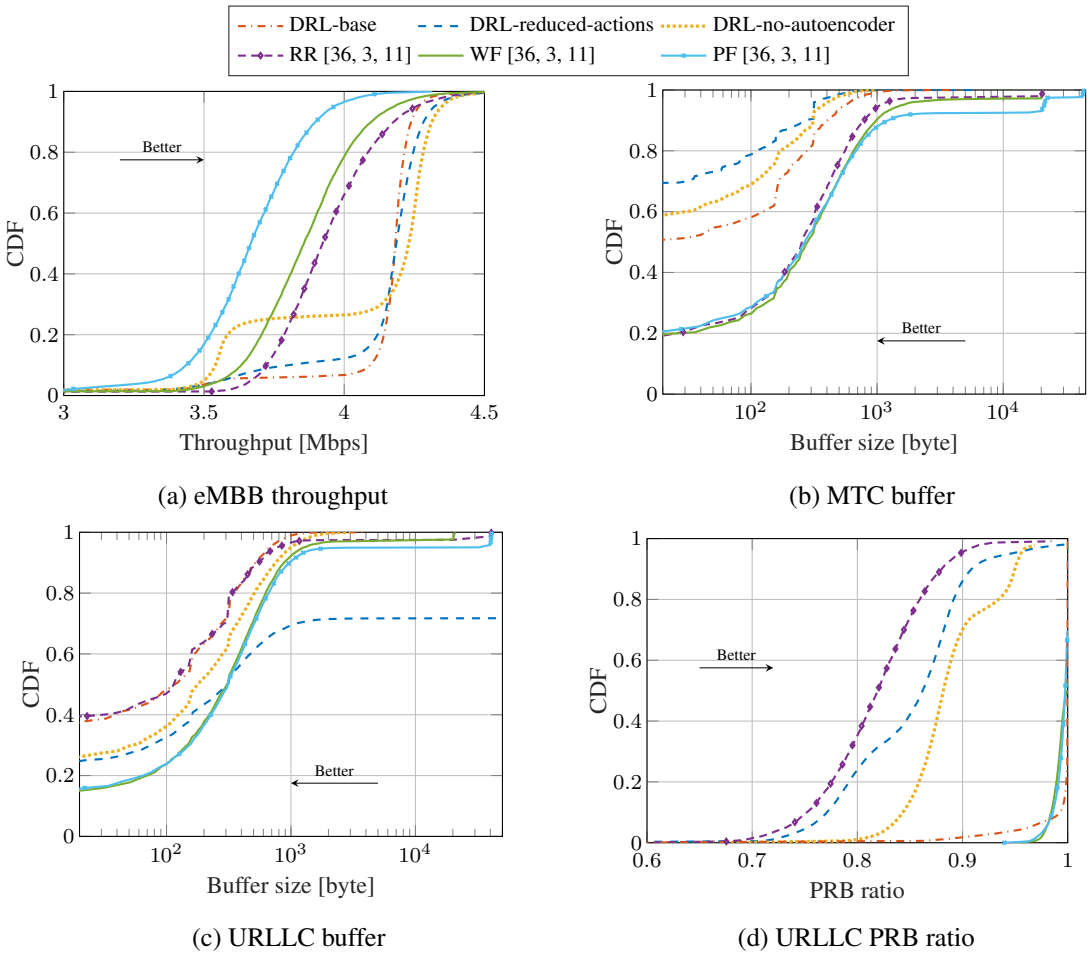
ratio for the URLLC slice (Figures 3.30c and 3.30d), but achieves the smallest buffer occupancy for the MTC traffic.

Autoencoder. Finally, the results of Figure 3.30 show the benefit of using an autoencoder, as the DRL-base and DRL-reduced-actions agents generally outperform the DRL-no-autoencoder agent. The autoencoder decreases the dimensionality of the input for the DRL agent, improving the mapping between the network state and the actions. Specifically, the autoencoder used in this section reduces a matrix of $T = 10$ input vectors with $N = 3$ metrics each to a single N-dimensional vector. Second, it improves the performance with online inference on real RAN data. One of the issues of operating ML algorithms on live RAN telemetry is that some entries may be reported inconsistently or may be missing altogether. To address this, we train the autoencoder simulating the presence of a random number of zero entries in the training dataset. This allows the network to be able to meaningfully represent the state even if the input tensor is not fully populated with RAN data.

Control Loop Performance. ColO-RAN is able to perform control loops compliant with the OSC specifications, i.e., between $10\,\mathrm{ms}$ and $1\,\mathrm{s}$. As an example, the average round-trip-time from when the base stations transmit the KPM reports to the RIC to when they receive the control actions computed by the DRL-based xApps equals to $114.27\ \mathrm{ms}$, with a variance of $2.653\ \mathrm{ms}$.

3.3.5 Online Training for DRL-driven xApps

The last set of results presents an analysis of the trade-offs associated with training DRL agents on a live network in an online fashion. These include the evaluation of the time required for convergence, the impact of the exploration process on the RAN performance, and the benefits involved with this procedure. To do this, we load on the `online-training` xApp a model pre-trained on the offline dataset with the slice-based traffic profile. The same model is used in the DRL-reduced-actions agent. We deploy the `online-training` xApp on a ColO-RAN base station and further continue the training with online exploration, using the uniform traffic profile (with the same constant bitrate



(a) eMBB throughput

(b) MTC buffer

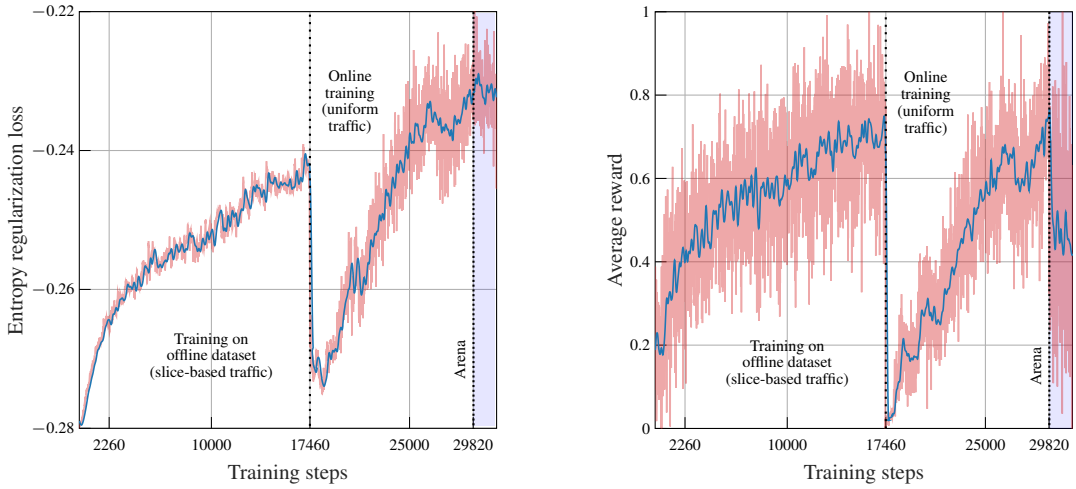
(c) URLLC buffer

(d) URLLC PRB ratio

Figure 3.30: Comparison between the different models of the `sched-slicing` xApp and baselines without DRL-based adaptation. For the latter, the performance is based on the slicing configuration chosen with highest probability by the best-performing DRL agent, and the three scheduler policies.

traffic for each user). Additionally, we leverage the containerized nature of ColO-RAN to deploy it on Arena [522], a publicly available indoor testbed, and perform training with one SDR base station and three smartphones.

Convergence. Figures 3.31 and 3.32 show how quickly the pre-trained agent adapts to the new environment. Figure 3.31a reports the entropy regularization loss as a function of the training step of the agent. This metric correlates with the convergence of the training process: the smaller the absolute value of the entropy, the more likely the agent has converged to a set of actions that maximize the reward in the long run [295]. We stop the training when this metric (and the average reward, Figure 3.31b) plateaus, i.e., at step 17460 for the offline training and step 29820 for the online training on Colosseum. The loss remains stable when transitioning from the Colosseum to the Arena online training, while it increases (in absolute value) when switching traffic profile at step 17460. This shows that the agent generalizes better across different channel conditions than



(a) Entropy regularization loss

(b) Reward

Figure 3.31: Metrics for the training on the offline dataset and the online training on Colosseum and Arena. The Arena configuration uses LTE band 7. Notice that the Arena deployment considers 3 users per base station, contrary to the 6 users per base station of Colosseum, thus the absolute average reward decreases.

source traffic profiles. The same trend is observed for the average reward (Figure 3.31b), although the transition from Colosseum to Arena halves the reward (as this configuration features 3 instead of 6 users for each base station). While Colosseum online training requires 30% fewer steps than the initial offline training, it also comes with a higher wall-clock time as offline exploration allows the instantiation of multiple parallel learning environments. Because of this, the Colosseum DGX supports the simultaneous exploration of 45 network configurations. Instead, online training can explore one configuration at a time, leading to a higher wall-clock time.

Figure 3.32 reports the evolution of the distribution of the actions chosen by the DRL agent for the Colosseum offline and online training. Three histograms for steps 2260, 17460 (end of offline training) and 29820 (end of online training) are also highlighted in the plot on the right. During training, the distribution of the actions evolves from uniform (in yellow) to more skewed, multi-modal distributions at the end of the offline training (in orange) and online training (in red). Additionally, when the training on the new environment begins, the absolute value of the entropy regularization loss increases (Figure 3.31a), and, correspondingly, the distribution starts to change, until convergence to a new set of actions is reached again.

Impact of Online Training on RAN Performance. Achieving convergence with a limited number of steps is particularly important for online training, as the performance of the RAN may be negatively affected during the training process. Figure 3.33 reports the CDF for the user throughput during training and after, when the agent trained online is deployed on the `sched-slicing` xApp. The performance worsens when comparing the initial training step, which corresponds to the agent still using the actions learned during offline training, with an intermediate step, in which it is exploring random actions. Once the agent identifies the policies that maximize the reward in the new environment (in this case, with the uniform source traffic profile), the throughput improves.

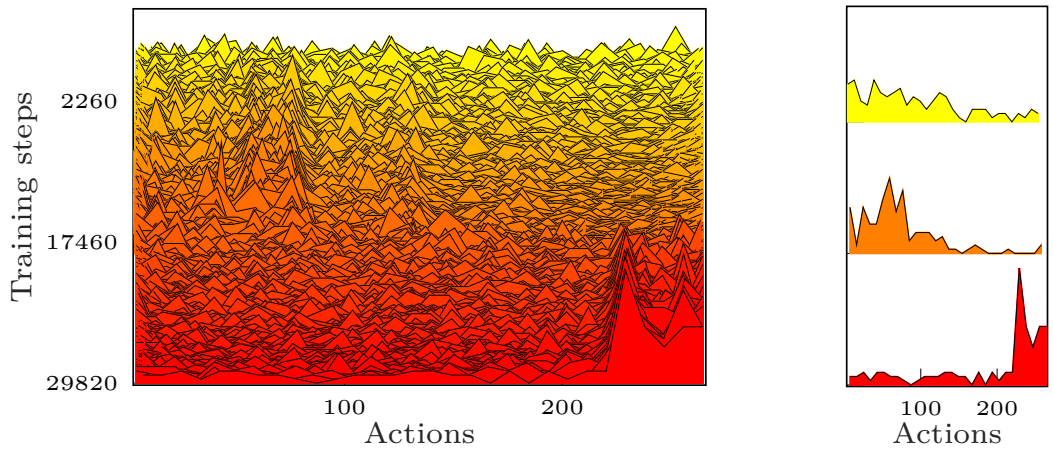


Figure 3.32: Distribution of the actions during the training on the offline dataset and the online training on Colosseum. The offline training stops at step 17460.

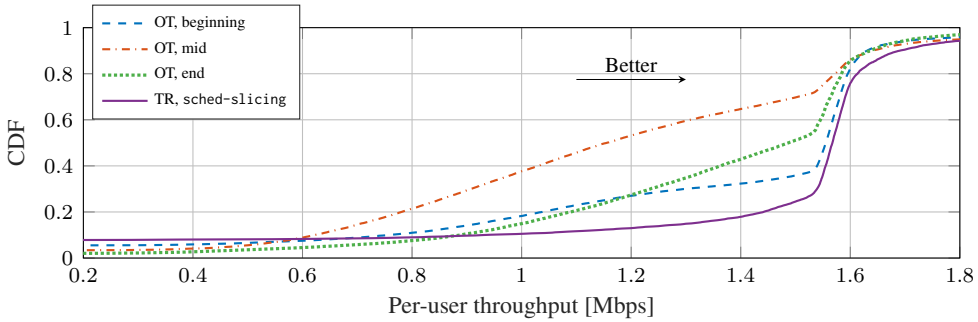


Figure 3.33: CDF of the throughput for the eMBB slice during the online training (OT) and with the trained agent (TR) with the uniform traffic profile.

The best performance, however, is achieved with the trained agent, which does not perform any exploration. Figure 3.34 further elaborates on this by showing how the online training process increases the throughput variability for the two eMBB users. Therefore, performing online training on a production RAN may be something a telecom operator cannot afford, as it may temporarily lead to disservices or reduced quality of service for the end users. In this sense, testbeds such as Colosseum can be an invaluable tool for two reasons. First, they provide the infrastructure to test pre-trained ML algorithms—and ColO-RAN enables any RAN developer to quickly onboard and test their xApps in a standardized O-RAN platform. Second, they allow online training without affecting the performance of production environments.

Adaptability. The main benefit of an online training phase is to allow the pre-trained agent to adapt to updates in the environment that are not part of the training dataset. In this case, the agent trained by the `online-training` xApp adapts to a new configuration in the slice traffic, i.e., the uniform traffic profile. Figure 3.35 compares the cell throughout for the agent before/after the online training, with the slice-based (Figure 3.35a) and the uniform traffic (Figure 3.35b). Notably, the online agent

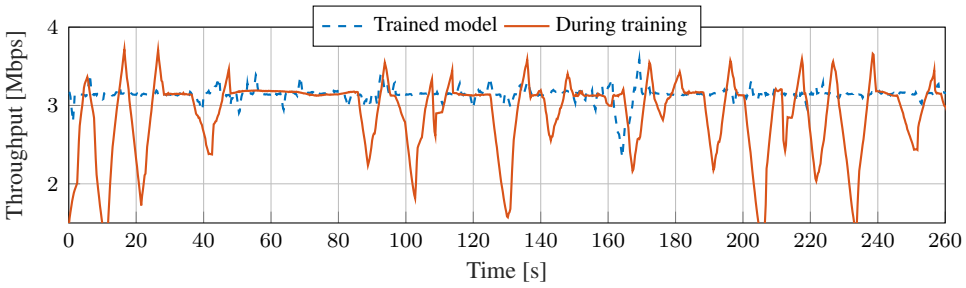
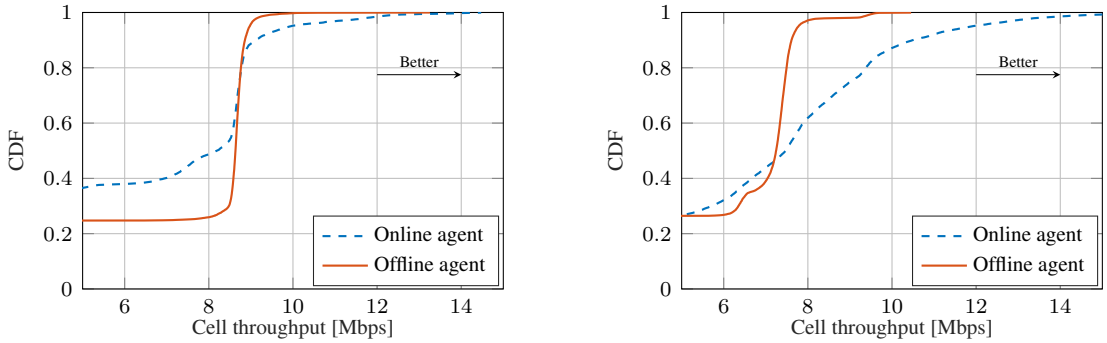


Figure 3.34: eMBB slice throughput during training and with the trained model.



(a) Slice-based source traffic

(b) Uniform source traffic

Figure 3.35: Throughput comparison between the offline- and online-trained models with two source traffic patterns. The offline agent is the DRL-base for the `sched-slicing` xApp.

achieves a throughput comparable with that of the agent trained on the offline dataset with slice-based traffic, showing that—despite the additional training steps—it is still capable of selecting proper actions for this traffic profile. This can also be seen in Figure 3.36, which shows that the action selected most often grants the most PRBs to the eMBB slice (whose users have a traffic one order of magnitude higher than MTC and URLLC).

The online agent, however, outperforms the offline-trained agent with the uniform traffic profile, with a gap of 2 Mbps in the 80th percentile, demonstrating the effectiveness of the online training to adapt to the updated traffic. The action profile also changes when comparing slice-based and uniform traffic, with a preference toward more balanced PRB allocations.

Summary. These results show how online training can help pre-trained models evolve and meet the demands of the specific environment in which they are deployed, at the cost, however, of reduced RAN performance during training. This makes the case for further research in this area, to develop, for example, smart scheduling algorithms that can alternate training and inference/control steps according to the needs of the network operator. Additionally, we showed that models pre-trained on Colosseum can be effective also in over-the-air deployments, making the case for ColO-RAN as a platform to train and test O-RAN ML solutions in a controlled environment.

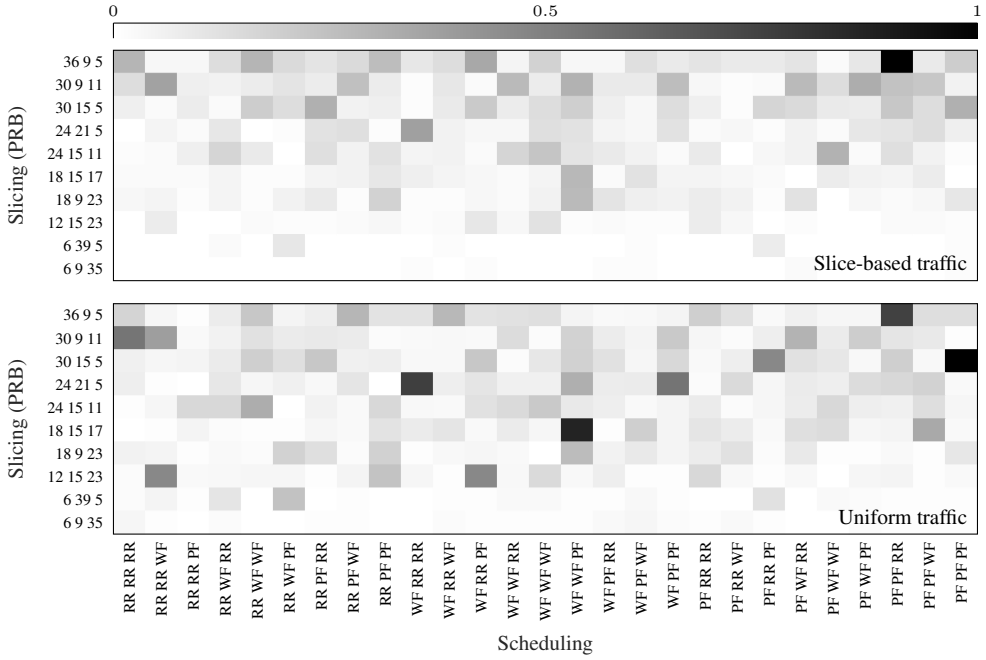


Figure 3.36: Probability of selecting a slicing/scheduling combination for the online-trained agent with two different source traffic patterns. For each tuple, the first element refers to the PRB (scheduling) for the eMBB slice, the second for the MTC slice, and the third for the URLLC slice.

3.3.6 Related Work

The application of ML to wireless networks has received considerable attention in recent years. Existing works span the full protocol stack, with applications to channel modeling, PHY and MAC layers, ML-based routing and transport, and data-driven applications [296–298].

Several papers review the potential and challenges of ML for wireless networks, discussing open issues and potential solutions. Kibria et al. highlight different areas in which ML and big data analytics can be applied to wireless networks [298]. Sun et al. [299] and Gunduz et al. [300] review the key learning techniques that researchers have applied to wireless, together with open issues. Similarly, Chen et al. focus on artificial neural network algorithms [301]. Other reviews can be found in [284, 302]. While these papers present a clear overview of open problems associated with learning in wireless networks, and sometimes include some numerical evaluations [303, 304], they do not provide results based on an actual large-scale deployment, as this section does, thus missing key insights on using real data, with imperfections, and on using closed-loop control on actual radios.

When it comes to cellular networks, ML has been applied throughout the 3GPP protocol stack. Perenda et al. automatically classify modulation and coding schemes [305]. Their approach is robust with respect to modulation parameters that are not part of the training set—a typical problem in wireless networks. Again, at the physical layer, Huang et al. investigate learning-based link adaptation schemes for the selection of the proper MCS for eMBB in case of preemptive puncturing for URLLC [306]. Others apply ML to 5G network management and KPM prediction [80, 307, 308]. These papers, however, do not close the loop through the experimental evaluation of the control

action or classification accuracy on real testbeds and networks. Chuai et al. describe a large-scale, experimental evaluation on a production network, but the evaluation is limited to a single performance metric [309].

DRL has recently entered the spotlight as a promising enabler of self-adaptive RAN control. Nader et al. consider a multi-agent setup for centralized control in wireless networks, but not in the context of cellular networks [310]. Wang et al. use DRL to perform handover [311]. Other papers analyze the theoretical performance of DRL agents for medium access [312] and user association [313]. Mollahasani et al. evaluate actor-critic learning for scheduling [314], and Zhou et al. applies Q-learning to RAN slicing [315]. Chinchali et al. apply DRL to user scheduling at the base station level [316]. Differently from these papers, we analyze the performance of DRL agents with a closed loop, implementing the control actions on a software-defined testbed with an O-RAN compliant infrastructure to provide insights on how DRL agents impact a realistic cellular network environment. Finally, [317, 526] consider ML/DRL applications in O-RAN, but provide a limited evaluation of the RAN performance without specific insights and results on using ML.

3.3.7 Lessons Learned

The design of ColO-RAN and of its xApps has allowed us to collect a number of key insights into the development of end-to-end ML and AI pipelines for O-RAN and next-generation wireless networks. The lessons learned include (i) *system-level* takeaways, related to the practical implementation of the end-to-end ML pipeline described in Section 3.3.1 in multiple large scale experimental testbeds (i.e., Colosseum and Arena), and (ii) insights on the *design of ML algorithms* for wireless network control and on their *online* adaptation. The main lessons learned are summarized as follows:

- The overall framework—combined with the capabilities of Colosseum and Arena—makes it possible to collect datasets, and to test ML routines, at scale. This has required the integration and adaptation of multiple components, including open source software developed by different (and often siloed) projects, such as the OSC and srsRAN, and custom components such as the xApps developed for ColO-RAN. The implementation of ColO-RAN bridges the gap among multiple projects, providing a streamlined tool for AI/ML-based RAN control.

- End-to-end ML integration requires open interfaces for the data collection and the connectivity between the RIC and the RAN, which need to comply with O-RAN specifications while at the same time adapt to the specific use case (i.e., slicing and scheduling selection). To this end, we equipped ColO-RAN with an O-RAN-compliant E2 interface at the RAN, and developed custom SMs on top, which can be easily extended to study and demonstrate other use cases.

- ColO-RAN strikes a balance between portability and performance. The interfaces and the closed-loop control need to comply with the near-real-time time scale, thus the RAN reporting loop and xApp inference need to be tuned to align to such constraints. At the same time, however, the ColO-RAN RIC has been adapted to be instantiated in a constrained environment (i.e., a single Colosseum SRN or Arena server with LXC containers) and to be an easy-to-deploy solution that can simplify and accelerate research in the O-RAN space.

- The processing and inference based on near-real-time live RAN data requires a careful design of the data ingestion pipelines at the ColO-RAN RIC. Notably, we leveraged autoencoders

to aggregate metrics from multiple reporting periods and/or users in a slice, and to handle the lack of data for users which may disconnect or move in the scenario. The autoencoders are trained to handle different zero-padding configurations, thus they accurately represent the RAN state even with missing data. This also makes it possible to provide the DRL agent with a fixed and limited input size, independent on events in the RAN.

- The collection and sharing of data among different components of a networked system, including AI-based xApps, also requires a proper design of the state for each control application. We showed with a large scale wireless dataset that a proper data analysis helps identifying correlation among RAN KPMs and avoid unnecessarily increase the ML algorithm input space. ColO-RAN has been designed to support data collection and analysis across different testbeds and scenarios, thus simplifying the end-to-end ML design process.

- ColO-RAN demonstrates the effectiveness of intelligent control for the RAN through results from a large-scale comparative performance evaluation of multiple xApps. Adaptive policies effectively improve the performance of tenants with different (and often orthogonal) traffic requirements, a key element for next-generation cellular networks.

- Finally, ColO-RAN makes it possible to train and test the same xApps on different wireless environments, e.g., Colosseum and Arena. This allowed us to evaluate online training steps, which make it possible to adapt and fine-tune the performance of a pre-trained ML algorithm to new events and conditions, not part of the training set. We showed however that this has a cost in terms of degraded RAN performance during the exploration phase, prompting the study and development of smart scheduling solutions. In addition, the performance evaluation indicated that models pre-trained on Colosseum data are effective also when deployed on an over-the-air testbed, making hardware-based emulation a viable and powerful step toward the creation of large-scale wireless experimental datasets.

ColO-RAN and the dataset collected for this work are publicly available and will enable O-RAN-based experiments in Colosseum. We believe that this end-to-end experimental infrastructure, together with the insights and lessons summarized here, will enable further research and development in AI and ML solutions for the Open RAN.

3.4 Colosseum: Large-Scale Wireless Network Emulation Through Hardware-in-the-Loop Experimentation

In this section, we present Colosseum—the world's largest wireless network emulator with hardware in-the-loop—as a platform that is *for the first time* available to the research community.[11] Originally built by the Defense Advanced Research Projects Agency (DARPA) and by the Johns Hopkins University Applied Physics Laboratory to support the Spectrum Collaboration Challenge (SC2) [269, 318–327], Colosseum is being expanded and operated by the Institute for the Wireless Internet of Things[12] at Northeastern University through an National Science Foundation (NSF) grant, which also

[11] `https://www.colosseum.net`
[12] `https://www.northeastern.edu/wiot`

made it publicly available to the research community. With its 256 SDRs and 128 remotely accessible compute nodes and GPUs, Colosseum provides the capabilities to test full-protocol stack solutions at scale with real hardware devices and in emulated—yet realistic—environments with complex channel interactions (e.g., path loss, fading, multipath). Besides its experimentation capabilities, Colosseum can be used as an AI playground and wireless data factory to create large-scale datasets and train/test solutions in a safe and controlled environment (see Section 3.2 and Chapter 4). We provide an overview of the architectural components and emulation capabilities of Colosseum in Sections 3.4.1 and 3.4.2. We show how to use Colosseum in Section 3.4.3, including practical use cases in Section 3.4.4. Finally, we describe the planned extensions to Colosseum in Section 3.4.5.

3.4.1 Colosseum Architecture

A high-level representation of the architecture of Colosseum is shown in Figure 3.37. Colosseum comprises 128 SRNs, the MCHEM, the RF scenario server, the TGEN, and the management infrastructure. The SRNs, which can be controlled remotely to perform experiments, are divided in four quadrants and are synchronized in time and frequency through hierarchical OctoClock clock distributors. Each SRN is a state-of-the-art server with 48-core Intel Xeon E5-2650 CPUs and an NVIDIA Tesla K40m GPU, and drives an NI/Ettus USRP X310. Each X310 is equipped with two UBX-160 daughterboards that operate between 10 MHz and 6 GHz. Colosseum allows multiple concurrent users to automatically deploy softwarized containers—implemented via LXCs—on the bare-metal SRNs (Figure 3.37, left). Containers can be used to run a variety of protocol stacks and to control parameters and configurations at different layers of these stacks.

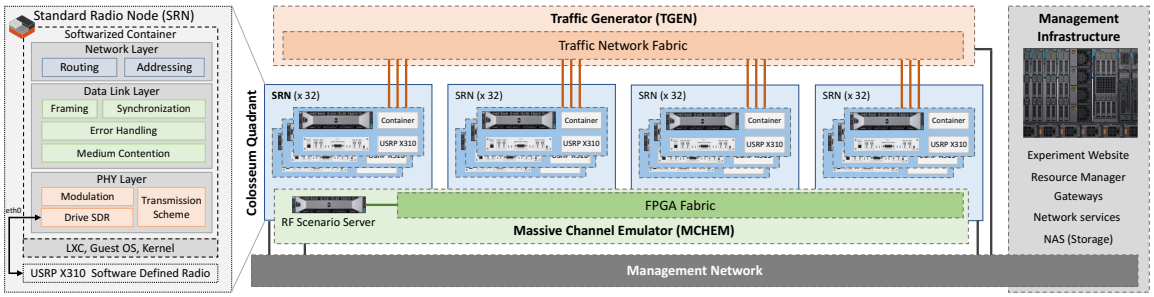


Figure 3.37: Colosseum architecture.

MCHEM is tasked with the channel emulation in Colosseum. It comprises four interconnected quadrants, each with 4 NI ATCA 3671 FPGA modules and 16 Virtex-7 690T FPGAs, and it drives an array of 128 USRPs X310 interconnected in a one-to-one fashion to the USRPs of the SRNs (see Figure 3.38).

When an RF transmission occurs in Colosseum, the signals generated by the USRPs at the SRN side (e.g., signal x_1 in Figure 3.38) get transmitted to the corresponding USRPs X310 of MCHEM, which perform RF to baseband and analog-to-digital conversions. The digital signals are then forwarded to the FPGAs of MCHEM that process them through Finite Impulse Response (FIR) filters. Filters are composed of 512 pre-computed complex-valued taps that capture the characteristics

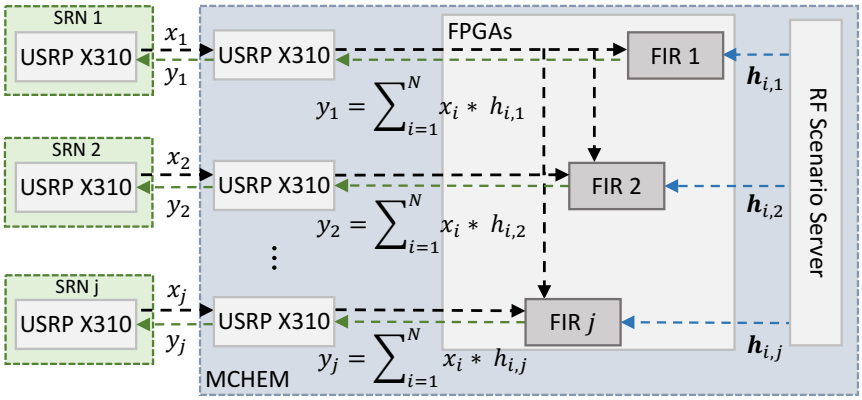


Figure 3.38: FPGA-based RF scenario emulation in Colosseum.

of the channel, i.e., the Channel Impulse Response (CIR), between any pair of SRNs.[13] As depicted in Figure 3.38, the FIR filters load the vector $\boldsymbol{h}_{i,j}$ of the 512-tap CIRs among receive node j and every transmit node i, with $i, j \in \{1, ..., N\}$ set of SRNs of a specific experiment. These channel taps are then applied to signal x_i through a convolution operation. The resulting signal $y_j = \sum_{i=1}^{N} x_i * h_{i,j}$ (i.e., the transmitted signals x_i processed with the CIR of the emulated channel between nodes i and j) is then sent to SRN j. In this way, Colosseum aggregates all the received signals after passing them through the corresponding CIR and transmits the results y_j to all the SRNs of the experiment (and not only to the intended receiver), thus capturing and emulating effects of real wireless channels, such as the interference among nodes and superimposition of signals from multiple nodes [328].

The RF scenario server (see Figure 3.37 and 3.38) maintains a catalog of the Colosseum RF scenarios and feeds their channel taps to the channel emulator at run time. Scenarios make it possible to emulate effects of the wireless channel, including path loss and fading in terrains up to $1\ \mathrm{km}^2$ and with up to $80\ \mathrm{MHz}$ bandwidth. The modular architecture of MCHEM and the independence of its USRPs allow Colosseum to concurrently emulate different scenarios on different experiments so that multiple users can operate on the system at the same time. As we will discuss in Section 3.4.3, the specific scenario to run can be selected by the user through a specialized control interface.

Similarly to how MCHEM emulates the RF scenarios, TGEN—based on the U.S. Naval Research Laboratory's MGEN [268]—takes care of emulating IP traffic flows between the SRNs. The set of traffic flows that can be generated by TGEN, together with their characteristics (e.g., packet rate, size and distribution), constitutes a *traffic scenario*. Once a traffic scenario starts, packets are delivered to the SRNs, which handle them through the user-defined protocol stack (e.g., by transmitting them through a cellular of Wi-Fi stack, see Section 3.4.4). Finally, the Colosseum management infrastructure hosts a variety of auxiliary services (Figure 3.37). These include: (i) the website through which the users reserve resources and start experiments; (ii) the resource manager, which allocates resources to the users; (iii) gateways for user and management access to Colosseum; (iv) a $900\ \mathrm{TB}$ NAS to store LXC images, experiment data and logs, and (v) various network services, including services to provide time synchronization across the whole testbed.

[13] Due to the high computational complexity required to generate scenarios, and to the large space to store them, only 4 channel taps are non-zero-valued.

3.4.2 Experimental Scenarios in Colosseum

Experimental scenarios, i.e., RF and traffic scenarios, are at the core of Colosseum. RF scenarios enable the emulation of up to 65,535 wireless channels in diverse environments with an emulation area up to $1\,\mathrm{km}^2$. They give users full control over the wireless environment, allowing them to capture and reproduce the desired channel effects while enabling reproducibility and repeatability of experiments at scale. This emulation is carried out by MCHEM through FPGA-based FIR filters, which capture and reproduce the CIR of the emulated wireless environment (see Section 3.4.1). The CIR can be derived in many different ways: for instance, it can be computed through well-known channel modeling equations, performing field measurements [539], or leveraging high-accuracy software tools, such as ray-tracers [541]. In this way, scenarios not only model the channel between transmitter and receiver pairs, but they also model effects typical of the wireless propagation environment, such as interference and superposition of the signals generated by multiple nodes. This ultimately guarantees a *high-fidelity* channel emulation, ensuring that the emulated environments are as close as possible to real-world deployments.

3.4.2.1 Examples of Available Experimental Scenarios

We now provide a sample of large-scale scenarios that are available on Colosseum at the time of this writing: the *Alleys of Austin* and the *SC2 Championship Event (SCE) Qualification* scenarios—developed by DARPA for SC2—and a set of *cellular scenarios* developed as part of the SCOPE framework (see Section 3.2).

- *Alleys of Austin.* This scenario is set in the urban area of downtown Austin, TX, and involves 50 nodes divided in 5 groups, or *squads*. Every group consists of 9 pedestrian users moving in a row formation, and a UAV circling on top of them. The center frequency of this scenario is set to 1 GHz. The duration of the scenario is 15 minutes, divided in three stages of 5 minutes each. In the first stage, nodes exchange voice traffic. In the second stage, they transmit images and videos in addition to the voice data of the previous phase. Finally, in the third stage, the transmission rate of the nodes significantly increases.

- *SCE Qualification.* This scenario concerns 10 nodes and has center frequency set to 1 GHz. The duration of the scenario is 10 minutes. The Signal-to-Noise-Ratio (SNR) among nodes is initially set to 20 dB and decreases by 5 dB every 2 minutes of the scenario. Finally, in the last 2 minutes, the scenario center frequency shifts to 1.1 GHz, and the SNR is reset to 20 dB. As for the traffic, nodes exchange UDP packets at constant bitrate.

- *Cellular Scenarios.* Colosseum includes a number of scenarios designed for cellular networking experimentation developed as part of the SCOPE framework, which will be described in Section 3.2. They involve 8 to 10 base stations serving four mobile users each. The locations of the base stations—derived from the OpenCelliD database [267]—match those of real-world cellular deployments in Rome, Italy, in Boston, MA, and in the POWDER testbed of Salt Lake City, UT (described in Section 2.7). See Section 3.2.3.2 for a detailed description of these scenarios.

3.4.2.2 Colosseum as a Wireless Data Factory

Colosseum scale and emulation capabilities allow it to create large-scale datasets, namely acting as a wireless data factory. Its unique emulation capabilities—both in terms of RF and traffic scenarios—allow users to configure an experiment once and then seamlessly run it in different environment, channel and traffic conditions. Moreover, once set up, experiments can be run automatically in *batches*, allowing users to collect data in bulk into wireless datasets (see Section 3.4.3). These capabilities are fundamental when designing ML algorithms, which need to be trained on large amounts of data and on different conditions to be able to generalize and adapt to unforeseen situations.

3.4.3 Operational Modes of Colosseum

Colosseum allows users to operate in either (i) *interactive mode*, in which users operate the SRNs manually, or (ii) *batch mode*, in which experiments are carried out automatically.

Interactive Mode. The workflow of an interactive experiment in Colosseum is shown in Figure 3.39. As a first step, the users reserve a number of SRNs through Colosseum website. In this step, they

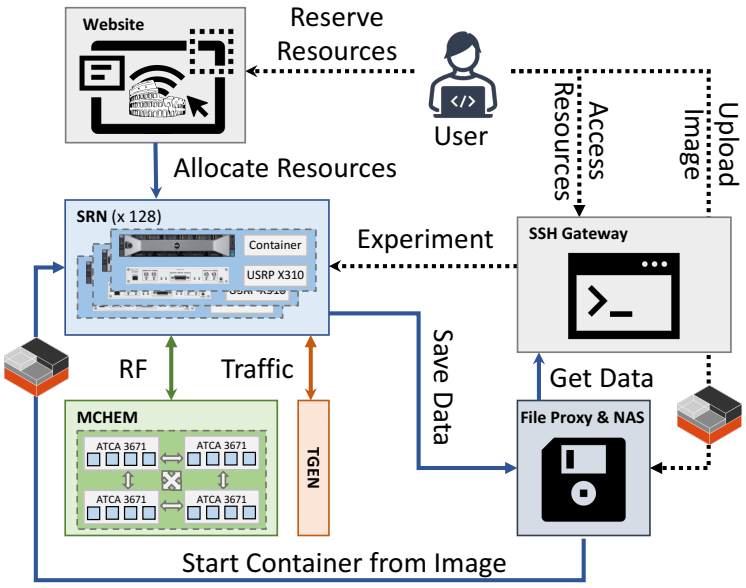


Figure 3.39: Workflow of a Colosseum interactive experiment.

can specify which LXC image to load on each SRN (i.e., either one of Colosseum-provided images or an image they previously created/uploaded to the system). To guarantee a fair use of the testbed, Colosseum implements a token-based resource allocation system, where occupying resources costs a certain amount of tokens per hour.[14]

After resources have been reserved, Colosseum automatically instantiates LXC containers on the SRNs. Next, users log into Colosseum gateway through the Secure Shell (SSH) protocol and access the SRNs allocated to them. This is where the main part of Colosseum interactive experiments

[14]Colosseum users are divided into teams, i.e., sets of users belonging to the same research group or organization. Each team is provided with a token budget that automatically resets on a weekly basis.

takes place. Users can start RF and traffic scenarios by using `colosseumcli`—an API to interface with some auxiliary services of Colosseum, e.g., MCHEM and TGEN—and control the USRPs X310 through softwarized protocol stacks instantiated on the SRNs (see Section 3.4.4 for exemplary Colosseum use cases). Through `colosseumcli`, users can also make a *snapshot* of the container running on the SRNs. This creates an LXC image from the container and saves it on Colosseum NAS for later use. Finally, when the experiment ends, resources are deallocated and the content of the `/logs` directory of each container are saved on the NAS, from where they can be retrieved through the file proxy server.

Batch Mode. While the interactive mode is more suitable for designing, prototyping and troubleshooting solutions, the batch mode can be used to automatically run experiments in bulk, e.g., to benchmark solutions or perform data collection. Batch jobs are configured through `json`-formatted files, in which users can specify the experiment duration, which RF and traffic scenarios to run, the number of SRNs to allocate and which LXC images to instantiate on them. Additional parameters—handled by the user-defined programs—can also be passed. Once scheduled, batch jobs are inserted in a queue and kicked off based on resource availability. Once a batch job starts, Colosseum takes care of instantiating the containers and to run the specified emulation scenario. Then, user-defined startup scripts are invoked on the SRNs and the actual experiment is carried out. Once the batch job ends, data and logs are copied from the `/logs` directory of the SRN containers to the NAS, analogously to what happens in interactive mode.

3.4.4 Use Cases of Colosseum

This section illustrates a set of Colosseum use cases. Examples concerning cellular networking are shown in Section 3.4.4.1, applications using a software-defined Wi-Fi protocol stack are showcased in Section 3.4.4.2, while spectrum sharing capabilities are discussed in Section 3.4.4.3. Finally, the application of UAVs to wireless networking is demonstrated in Section 3.4.4.4.

3.4.4.1 Cellular Networking

Softwarization and virtualization will play a fundamental role in 5G and beyond cellular networks. Telecom operators will deploy open-source custom solutions on a "white-box" agnostic RAN where services, e.g., the base stations, will be instantiated on-demand. This not only endows the network with flexibility by design, but it also allows real-time optimization of the users' service based on the current network conditions and traffic demand (see Chapters 2 and 4). Although open-source approaches bring unprecedented performance improvements, they require a reliable development environment and at scale evaluation before they are deployed on the commercial infrastructure. More importantly, this testing phase needs to account for different environments, wireless channel and traffic conditions. With its unique emulation capabilities, Colosseum allows the research community to reliably prototype and benchmark solutions for future cellular networks. To showcase these capabilities, we instantiate a cellular network with 4 base stations and 24 users through srsRAN, which has been discussed in Section 2.2.2. (More advanced applications, including the instantiation of O-RAN compliant softwarized cellular networks on Colosseum, will be discussed in details in Chapter 4, and in Sections 3.2 and 5.2.) Figure 3.40 shows the network downlink throughput when the users request traffic through `iPerf3` at different rates: low traffic (1 Mbps per user), medium

traffic (2 Mbps), and high traffic (4 Mbps). In this case, the base stations use a 10 MHz channel bandwidth.

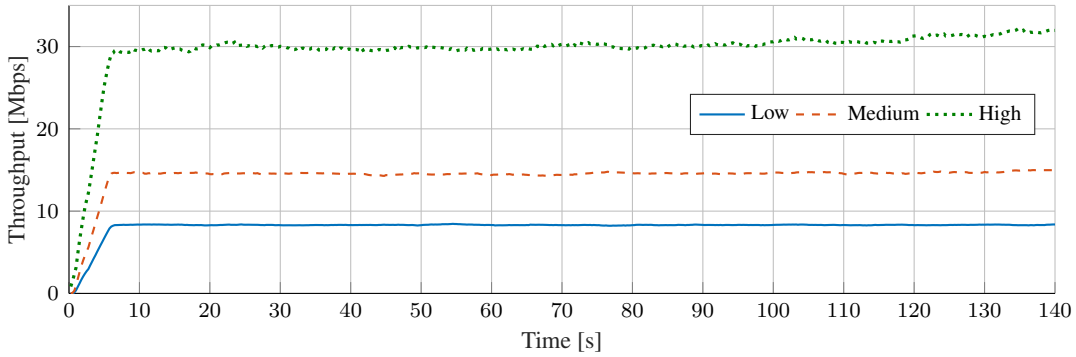
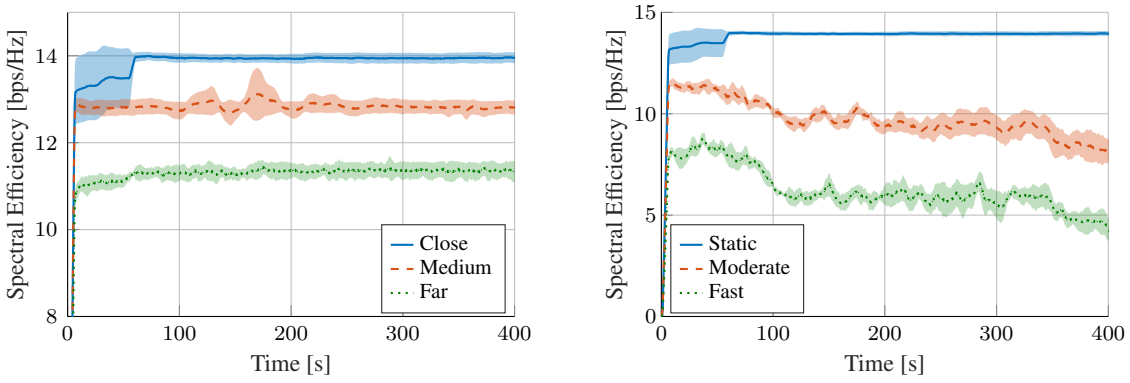


Figure 3.40: Downlink throughput with different traffic conditions.

Figure 3.41 shows how Colosseum scenarios can be used to study the impact of deployment distance and user mobility in cellular networks. We evaluate a network with 10 base stations and 40 users in close proximity to the base stations. Users request downlink video traffic at a rate of 1 Mbps through one of Colosseum traffic scenarios. Specifically, Figure 3.41a shows the downlink



(a) Varying distance

(b) Varying speed

Figure 3.41: Downlink spectral efficiency of the network varying the distance between users and base stations, and speed of the users.

spectral efficiency of the network for different deployment distances between users and base stations: close (users are deployed within 20 m from the base stations), medium (50 m), and far (100 m). Figure 3.41b, instead, depicts the same metric for different configurations of user mobility: static (no mobility), moderate (users move at a speed of 3 m/s), and fast (5 m/s). In this case, the users are deployed in close proximity to the base stations. We notice that although the spectral efficiency decreases with the increased mobility/distance between users and base stations, the network performance shows tight 95% confidence intervals (shaded areas). This confirms the reliability and repeatability of Colosseum emulation.

3.4.4.2 Wi-Fi

Wi-Fi has become an indispensable and ubiquitous wireless technology to provide home networking and public/hotspot Internet connectivity, as well as supporting a wide range of IoT applications. To support the ever-increasing number of devices, the Federal Communications Commission (FCC) has recently opened an additional 1.2 GHz of spectrum in the 6 GHz band [329]. This portion of the spectrum, intended for unlicensed use, has been embraced in the IEEE 802.11ax amendment, namely Wi-Fi 6 [330]. Large-scale experimentation is a core enabler for research and development in these emerging spectrum bandwidths.

To showcase how Colosseum can facilitate these operations, we instantiate a Wi-Fi network through an open-source GNU Radio-based implementation of the IEEE 802.11a/g/p standard [331]. We deploy 50 Wi-Fi transceivers in a Colosseum urban scenario and measure the Signal-to-Interference-

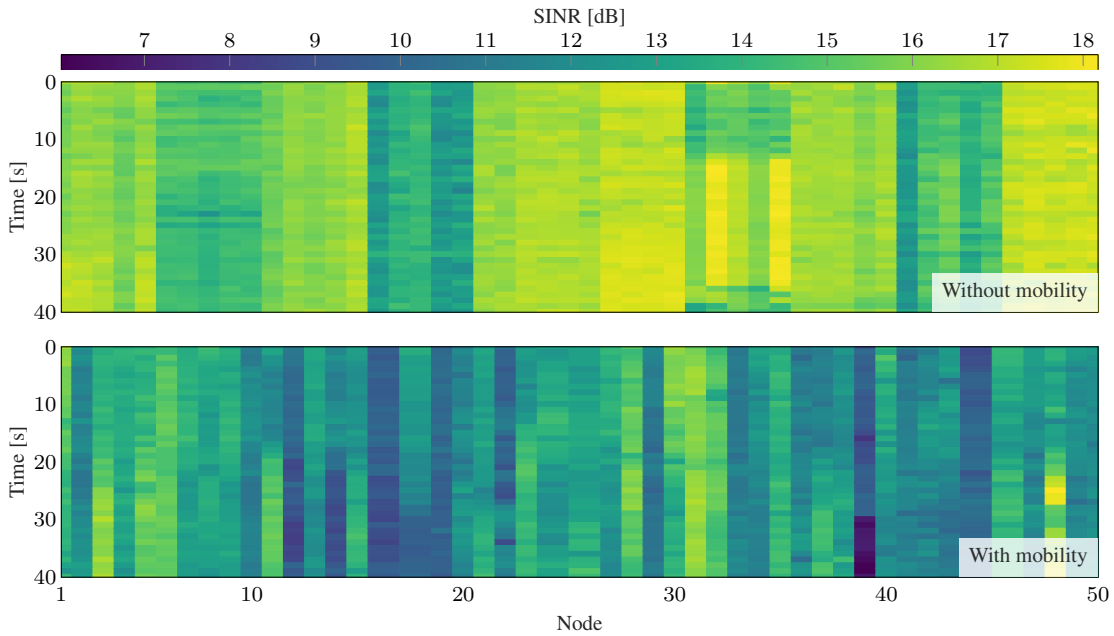


Figure 3.42: SINR measured by Wi-Fi nodes on Colosseum in a scenario without node mobility (top) and with node mobility (bottom).

plus-Noise Ratio (SINR) in the case nodes do not move (Figure 3.42, top), and in the case they move with an average speed of $3\ \mathrm{m/s}$ (Figure 3.42, bottom). We notice that when node mobility is implemented, the SINR degrades by 3.18 dB on average because of the rapidly-changing channel conditions.

3.4.4.3 Spectrum Sharing

In recent years, the widespread use of wireless communications has fostered technological advancement and motivated the development and deployment of innovative services for mobile users. These phenomena have resulted in a sharp increase in the number of wireless devices and traffic demand. This has exacerbated the issue of the lack of spectrum to support them, also known as *spectrum crunch* [332, 333].

In most cases, spectrum is either allocated to government agencies for research and military uses (e.g., radars, law enforcement), to the scientific community (e.g., for radio astronomy, atmospheric sensing), or licensed to operators, which gain exclusive access to spectrum frequencies in exchange for very high licensing fees. Only in a handful of cases, such as the Industrial, Scientific and Medical (ISM) band, spectrum is allocated to non-commercial and experimental applications. One of the main drawbacks of such a static allocation is that unlicensed portions of the spectrum are extremely congested, while licensed ones might experience lower load conditions due to the lack of activity.

In this context, one research topic that has gained momentum is spectrum sharing, whose ultimate goal is to devise solutions that allow multiple technologies to share portions of the spectrum. To this end, SDRs have gained increasing attention in the community as a promising solution to allow unlicensed users to fill the so-called *spectrum holes* left by licensed users. Although the literature is characterized by several works that leverage optimization and data-driven solutions in SDRs, one of the main limitations of such works is the lack of large-scale experimentation that demonstrates their effectiveness in heterogeneous network topologies, traffic and channel conditions [213].

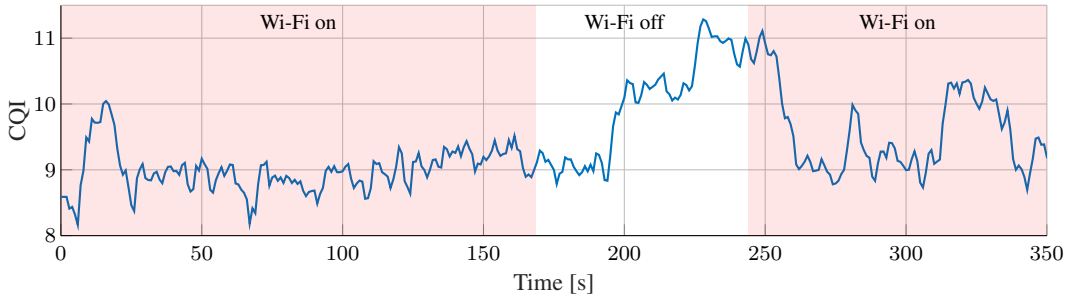
Colosseum fills this gap and allows users to instantiate large-scale networks with nodes running heterogeneous wireless protocol stacks (e.g., cellular, Wi-Fi) on the same spectrum bandwidth. This enables researchers to design, implement and evaluate novel spectrum sharing solutions on heterogeneous and diverse network deployments and conditions without causing harmful interference to licensed users. To provide an example of this application, we deploy on the same portion of the spectrum a cellular base station serving 5 cellular users, and two Wi-Fi nodes. Figure 3.43 shows the average CQI and percentage of uplink errors experienced by the cellular users with and without Wi-Fi transmissions active. We notice that the interference generated by Wi-Fi communications causes the CQI of the cellular users to rapidly drop (Figure 3.43a), and the percentage of uplink errors to increase (Figure 3.43b) when Wi-Fi transmissions are ongoing.

3.4.4.4 Unmanned Aerial Vehicles

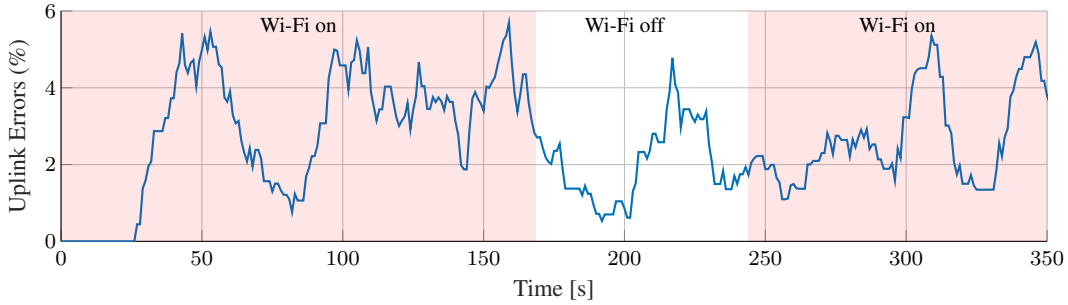
The growing commercialization of UAVs, along with their greater affordability and increasing popularity, has recently spawned the interest of telecom operators and equipment providers toward the use of drones for networking applications. Examples of such applications include the use of UAVs to create swarm of flying networks [334, 335, 536], to promptly aid in the aftermath of disaster scenarios [336, 337, 525], and to bring connectivity to remote areas [534, 538]. However, security concerns, strict flying regulations, and the cost of the equipment itself impose additional challenges to the development and testing of solutions at scale.

Colosseum has the potential of facilitating the evaluation of such solutions by providing the means to emulate large-scale SDR-enabled UAV networks. To showcase these capabilities, we instantiate a network with 30 nodes in the Alleys of Austin scenario (see Section 3.4.2.1). Nodes are divided in three groups, or *squads* (see Figure 3.44a). Each squad is composed of a UAV (marked with "U" in the figure) and 9 ground relay nodes ("R"), which need to deliver data (e.g., video and voice traffic) to specific members of the squad.

At each instant of time, specific nodes act as relays (aerial or ground) for data from a squad to the intended destination (e.g., nodes in other squads), while coping with inter-user and inter-squad interference. Figure 3.44b shows the average SINR when two different algorithms are used to select the relay nodes: SINR-based maximum weighted matching, and random allocation. We notice that



(a) Average CQI of cellular users



(b) Average percentage of uplink errors of cellular users

Figure 3.43: Average cell CQI and percentage of uplink errors with and without Wi-Fi traffic. Time periods without Wi-Fi traffic are marked with a red shaded area.

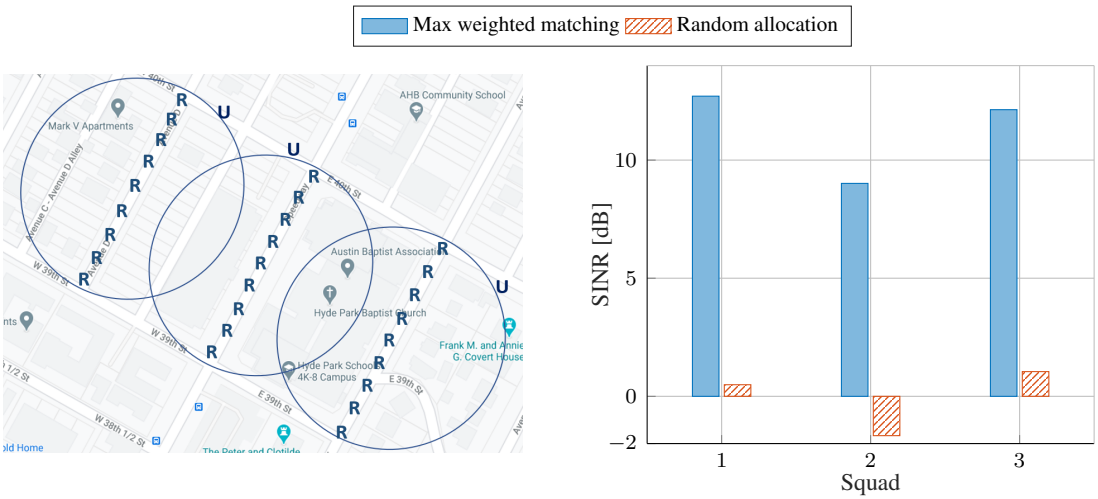
when the maximum weighted matching algorithm is used, the squads experience a higher SINR than when the random allocation strategy is enforced. This is due to the fact that the former algorithm elects as relays those nodes that have better channel conditions (and thus a higher SINR) toward the remaining squad members. This algorithm also takes advantage of the UAVs as aerial relays, which are in line-of-sight condition with the ground nodes.

3.4.5 Evolution of Colosseum

While Colosseum represents an invaluable tool for the evaluation of large-scale wireless networking scenarios, it also needs to evolve to match the needs of communication technologies in the years to come. The following sections describe planned extensions of Colosseum that will add new emulation capabilities (Section 3.4.5.1) and AI components (Section 3.4.5.2).

3.4.5.1 Expansions to Support NRDZ and mmWave Research

A steady trend in the wireless industry has been the push for the allocation of additional spectrum for communications, with spectrum sharing [21] and the adoption of new frequency bands, e.g., millimeter waves [46, 338]. As the usage of the spectrum expands into new frontiers, it becomes important to study and understand how different spectrum users can safely coexist. Along this line, the NSF has recently published a “Dear Colleague Letter” to explore the feasibility of establishing National Radio Dynamic Zones (NRDZs) in the United States. NRDZs are meant as safe platforms



(a) Network topology (b) Average squad SINR

Figure 3.44: Left: network topology of an experiment with 30 nodes (ground relays, "R", and UAVs, "U") in the Alleys of Austin scenario. The circles mark the trajectory of the UAVs. Right: average SINR of three squads in Alleys of Austin scenario when different algorithms are used for the selection of relay nodes.

for experiments that would not be allowed under FCC spectrum regulations because of chances to generate harmful interference toward incumbent users. Colosseum is naturally positioned to implement an NRDZ, as—by being an emulator—it avoids any interference to external devices. Additionally, its programmability allows experimenters to test multiple scenarios, different protocol stacks, and different traffic patterns (see Section 3.4.2).

As part of the NRDZ activities, Colosseum will be expanded to support new spectrum bands, scenarios, and power levels. Colosseum already supports very granular coordinated spectrum sharing across heterogeneous networks, and will act as a "management and optimizer" system for protection, observation, validation and automation of spectrum sharing. These capabilities will be extended by adding two new quadrants (see Figure 3.45) and by improving the current MCHEM emulation.

As of today, there are two limitations in Colosseum that prevent it to act as a tool to design and study NRDZs as defined above. First, Colosseum was designed to emulate terrains up to $1{\times}1\,\mathrm{km}$. This limitation is a consequence of the maximum propagation delay that can be emulated by MCHEM, which is embedded in the design of its FPGAs. Second, Colosseum was designed to emulate sub-6 GHz omnidirectional transmissions. However, it cannot emulate directional transmissions and higher frequency bands.

To go beyond today's limitations, we are deploying an additional, reduced version of a Colosseum quadrant as a development environment. This environment includes a dedicated MCHEM quadrant that will be the basis for the development of new channel emulation code, as well as the testing of new hardware for the user radios and SRNs. Notably, two extensions for MCHEM will be introduced first in this environment, and then scaled to the full Colosseum (Figure 3.45). The first will allow longer delays for high-fidelity emulation of a 100×100 km grid and terrains representative of NRDZs. The second will introduce the modeling of beamforming-based, directional communications to emulate mmWave networks. Finally, to support the bandwidth requirements of 5G and beyond cellular networks, we will create a new Colosseum quadrant able to emulate very large baseband

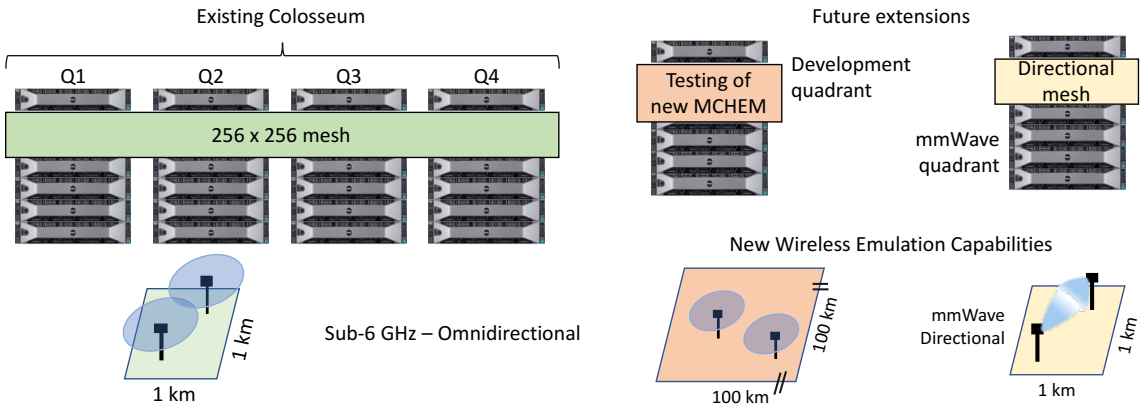


Figure 3.45: Current components of Colosseum and proposed extensions for the NRDZ program.

bandwidth (up to 2 GHz of bandwidth).

3.4.5.2 AI Jumpstart Integration

Colosseum is currently being expanded through the MassTech "AI Jumpstart" program [339]. This program aims at jump-starting Massachusetts firms interested in deploying AI to enhance their businesses, connecting industry practitioners with world-class facilities and researchers. Part of the AI Jumpstart equipment will be integrated in the Colosseum environment to enable research and development of AI-based wireless networking solutions. Examples of use cases are the fast and efficient training of large-scale wireless datasets collected on Colosseum (see Chapter 4); real-time, AI-driven 5G signal processing for the full 5G protocol stack [340], and model-free adaptation and control of large-scale wireless networks.

The new equipment includes two NVIDIA DGX A100 nodes—among the most powerful AI compute solutions on the market today—capable of delivering up to 10 petaFLOPS [341]. These nodes include 8 GPUs, state-of-the-art AMD CPUs, 1 TB of RAM, and 10 Mellanox ConnectX-6 network cards, each capable of sustaining a 200 Gbps link. An additional large memory node enables memory-intensive workloads, with 3 TB of RAM. These machines are connected through a dedicated Mellanox Infiniband switch, which can sustain an aggregated traffic of up to 16 Tbps. This system will be fully meshed with the compute and wireless resources of Colosseum, with dedicated Nomad-based orchestration and load balancing capabilities [342].

3.5 Arena: An Indoor Platform for Spectrum Research

In this section we give a succint overview Arena, an open-access wireless testing platform based on an indoor 64-antenna ceiling grid connected to programmable SDRs for sub-6 GHz spectrum research. A comprehensive description of the Arena testbed, including further details on hardware/software configuration, experimental capabilities and additional use cases, can be found in [522].

The Arena testbed is located in the open-space laboratory on the fourth floor of the Northeastern University Interdisciplinary Science & Engineering Complex and provides researchers and practition-

ers with the ability of testing medium- and short-range solutions over a real-world indoor wireless environment. The main characteristics of Arena can be summarized as follows:

- **Real-time real-channel evaluation platform**. Arena is an open-access, remotely accessible testing platform that can be used to prototype, develop, and experimentally evaluate new emerging technologies running real-time experiments leveraging the testbed SDRs and real indoor wireless channel characteristics.

- **Fully-synchronized testbed**. Arena leverages a 24-SDR rack to drive a total of 64 transmit/receive antennas deployed on a ceiling grid layout covering an overall area of $2240\ \mathrm{ft}^2$. The radios are synchronized via clock distributors, and they connect to the antennas using identical equal-length cables, ensuring full symbol-level synchronization throughout the whole testbed. This enables applications such as massive MIMO, cooperative multi-point MIMO, and synchronized distributed systems.

- **Repeatable, flexible, and scalable indoor experiments**. The 64-antenna grid of Arena provides a plethora of possible network topologies and the scale to foster new technology development. Its design ensures unchanged locations throughout the experiments and guarantees the integrity of the collected experimental data. A server rack of 12 identical compute machines drives the radios and guarantees the same computational power to each one of them toward a fair evaluation. The three-tier design of Arena—made of the server rack, the radio rack, and the ceiling grid—solves SDR deployment issues typical of laboratory setups, such as antenna orientation, cable non-linearities, and, ultimately, guarantees experiment repeatability.

The rest of this section is organized as follows. In Section 3.5.1, we outline the testbed design and system architecture, while we describe the Arena hardware and software components in Section 3.5.2. Finally, in Section 3.5.3, we describe how to perform experiments on the testbed, while the experimental capabilities of Arena are showcased in Section 3.5.4 through sample use cases.

3.5.1 Testbed Design and System Architecture overview

Arena provides researchers with a software control framework and radio hardware to evaluate wireless development on a multitude of different radio configurations, topologies, and channel conditions. Furthermore, it offers the capability of scaling up the testing environment by just changing a few lines of code, with the ultimate guarantee of reproducible experiments. Arena is based on a three-tiered architecture, the server rack, the radio rack, and the antenna grid.

The Server Rack. The server rack consists of 12 servers individually accessible through a top-of-the-rack gateway responsible for authenticating users (see Figure 3.46). The platform is remotely accessible through the College of Engineering (COE) Gateway. Servers are in charge of driving the SDRs performing the transmit and receive baseband processing operations. They are identical to each other both on hardware capabilities, kernel version, operating system, and installed software, to guarantee uniform computational power and fair operations across the whole testbed. Moreover, servers employ the network file system protocol to mount user disk spaces and to maintain consistency of new software deployments and files across the whole rack. Servers connect to SDRs through a dedicated PCIExpress network card, offering two additional network interfaces at $10\ \mathrm{Gbps}$ each.

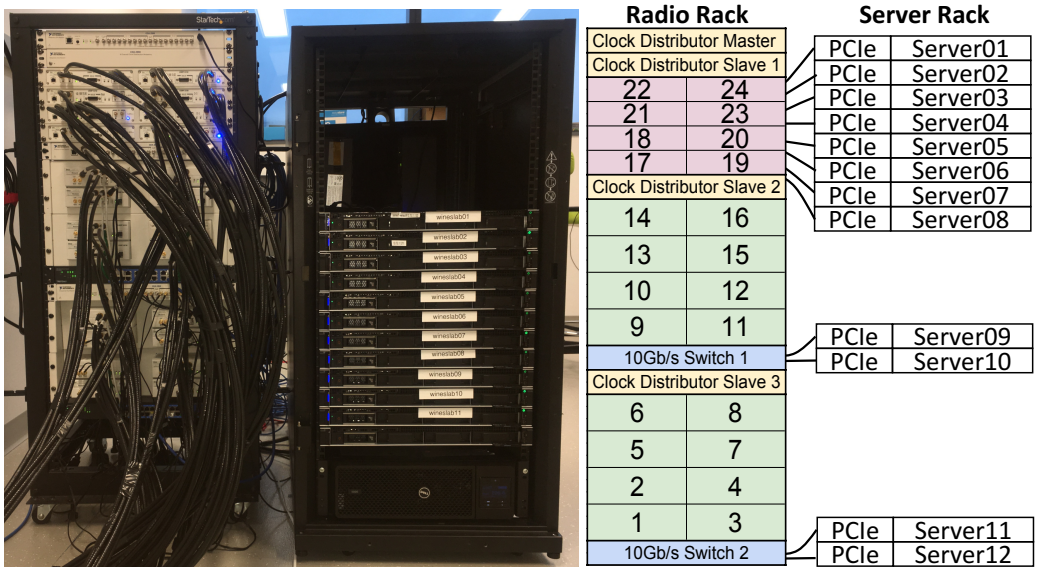


Figure 3.46: Radio and server rack physical and logical configuration.

They are also connected to the gateway through standard 1 Gbps interfaces. Each server connects to radios in one of two possible ways: *one-to-one* driving, and *one-to-four* driving, depending on the model of the radio to be driven.

- *One-to-one driving.* Eight of the 12 servers are connected one-to-one to individual USRP X310 SDRs via a dedicated 10 Gigabit network interface. The USRP X310 is a high-performance, scalable SDR platform whose hardware architecture combines two extended-bandwidth daughterboard slots covering DC–6 GHz with up to 120 MHz of baseband bandwidth. The 10 Gigabit interface provides high bandwidth and a low-latency connection between servers and SDRs.

- *One-to-four driving.* The remaining four servers are organized in pairs, each pair driving eight USRP N210 SDRs through a TRENDnet TEG-30284 10 Gigabit switch, for a total of 16 USRPs N210. The USRP N210 is a high-bandwidth, high-dynamic range radio designed to operate from DC to 6 GHz. The USRPs N210s are driven over 1 Gigabit Ethernet interfaces through the 10 Gigabit switches.

The Radio Rack. The radio rack is composed of 16 USRPs N210, 8 USRPs X310, 4 OctoClock clock distributors, and two 10-Gigabit switches (see Figure 3.46, left). The 24 SDRs of the testbed house 32 daughterboards, each having one TX/RX and one RX2 antenna frontends. The SDRs in the rack are logically organized into three groups of eight radios each. Each group is provided with time and frequency synchronization by an OctoClock clock distributor (for a total of three clock distributors). These are in turn synchronized to a main clock distributor able to generate time and frequency signals. To connect the main clock distributor to the three secondary clock distributors, and the latter to the SDRs, we use 54 identical and same-length cables. These cables guarantee clear reference signals and identical delays across all the 24 radios, which is essential for full synchronization across the whole rack.

With the goal of load balancing the network traffic between the two 10 Gigabit network interfaces of each server, the radio rack networking has been configured to partition the SDRs into sub-networks. Specifically, each USRP X310 is on its own private sub-network, while the USRPs N210 are grouped into four different 4-device sub-networks. Given the baseband processing capabilities of USRPs X310 (200 MSamples/s) and of USRPs N210 (25 MSamples/s), this configuration aims at load balancing the traffic over the two 10 Gigabit/s interfaces of the servers. This guarantees that the bandwidth of these interfaces suffices to drive up to four USRPs N210 or one USRP X310 through the USRP Hardware Driver (UHD) (32 bit/Sample).

The Antenna Grid. Each SDR houses one or two radio frequency daughterboards with two antenna frontends, namely TX/RX and RX2. The input ports connect to one ceiling antenna through a 100 ft low-attenuation coaxial cable. Overall, each daughterboard connects to one *antenna pair* formed by one transmitting and one receiving antenna. In this way, the USRPs X310 connect to a total of four antennas, while the USRPs N210 connect to two antennas. The antenna grid floor plan layout is shown in Figure 3.47. The 64 antennas are deployed across 8 rails, each hosting four equidistant

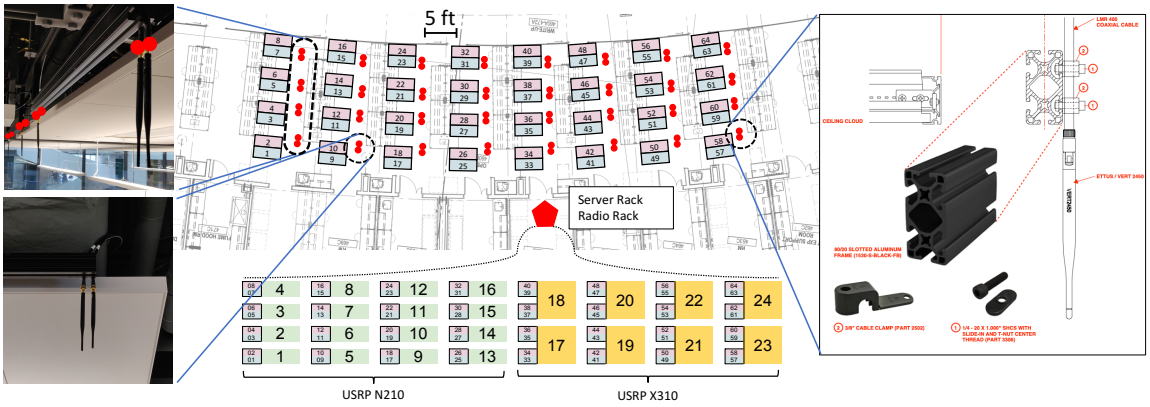


Figure 3.47: Arena antenna grid layout.

antenna pairs (8 antennas). With a spacing of 5 ft between *antenna pairs* and 12 ft between rails, Arena covers an overall deployment area of 2240 ft^2. Being antennas hung off the ceiling, Arena offers unique line-of-sight conditions with respect to similar size indoor testbeds, yet preserving the wireless channel characteristics typical of office-like environments. Its grid layout eases topology changes as well as long-distance communications without requiring physical relocation of the radios. Finally, in spite of the multitude of radio testing topologies that the 64-antenna grid offers, additional topologies can easily be arranged due to the sliding rails that allow each antenna to be relocated for application-specific scenarios, as illustrated in Figure 3.47, left.

3.5.2 Hardware and Software Components

In this section, we provide a detailed description of Arena's design choices, as well as of its hardware and software configuration.

3.5.2.1 Server and Radio Rack Configuration

The server rack is composed of 12 servers, a gateway, and a switch. The servers are Dell EMC PowerEdge R340 machines, namely `wineslab01-12`, running Ubuntu Linux, and provide the computational power to drive the 24 SDRs. Specifically, each server has a 6-core (12-threads each) Intel Xeon E-2186G processor with $3.80\,\mathrm{GHz}$ base frequency (max $4.70\,\mathrm{GHz}$) and four $8\,\mathrm{GB}$ DDR4-2666 RAM with $2666\,\mathrm{MT/s}$ speed.

Since high-speed connection and low-latency control are the keys to efficient software baseband processing, each server employs an additional Intel X520 Dual Port 10 Gigabit DA/SFP+ network card to communicate with the radios. This additional network card adds two network interfaces leveraging the SFP+ technology to establish a fast and reliable link to the radios and an aggregate data-rate of $20\,\mathrm{Gbps}$.

A set of open-source software tools has been pre-installed on the servers to communicate with the SDRs and drive them. Among these, there are GNU Radio 3.7.13, srsRAN, UHD 3.14, Python 2.7, and Python 3.5, which are ready to be used by any user to run wireless experiments. A standard Gigabit Ethernet interface connects each server to the WiNES Lab gateway through a 24-Port Netgear GS324 Gigabit Ethernet switch. The gateway is implemented on a Dell Precision 5820 machine running Ubuntu Linux and implements server access control and network security features, as well as it provides Internet access to each of the 12 servers.

Powering the server rack might require a high power supply, which potentially varies over time. To this end, the power supply system is based on two APC Metered Rack Power Distribution Units (PDU) AP7811B and a Dell $5000\,\mathrm{VA}$ $208\,\mathrm{V}$ Smart Uninterruptible Power Supply (UPS). This protects all the server rack devices from power spikes and surges and provides approximately one hour of emergency power in case of outages. Moreover, the UPS is powered through an emergency power receptacle, active even in case of a power outage in the building, as a second level of power outage protection.

The radio rack houses 16 USRP N210, 8 USRP X310, 4 OctoClock clock distributors, and two 10-Gigabit switches. USRPs are experimental hardware radio platforms completely controllable through software programs. They embed a FPGA, Analog-to-Digital Converters (ADCs), and Digital-to-Analog Converters (DACs) and are particularly suitable to design, test, prototype, and deploy wireless radio communication systems and protocols (see Section 2.6).

3.5.2.2 Grid Configuration

One of the highlight design choices of Arena is the 64-antenna ceiling grid concerning an 8×8 array that covers an overall area of $2240\,\mathrm{ft}^2$, where each antenna point is cabled to the radio rack. Arena is based on 64 American Wire Gauge (AWG) RG8-CMP [343] low-attenuation, fireproof 100 ft long SMA-to-SMA connection cables that have been specifically designed for indoor applications. Using 64 same-length cables guarantees equal delays in symbol transmission and reception across the whole testbed, despite the antenna location. Guaranteeing same delays all the way to the antennas is crucial for transmission schemes such as MIMO, where different antenna connector length might compromise the transmission synchronization. The cable length has been selected as the shortest one connecting the farthest antenna point to the rack to reduce signal power loss, which equals to $6.8\,\mathrm{dB}$ and $11\,\mathrm{dB}$ over $100\,\mathrm{ft}$ length at $2.4\,\mathrm{GHz}$ and $5.0\,\mathrm{GHz}$, respectively, which can

be however compensated with transmission and reception gains. Guaranteeing consistent delays across the whole testbed comes at the price of having tens of extra feet of cables to bundle. These have been professionally coiled across the ceiling avoiding loops that might result in undesirable electromagnetic effects.

For over-the-air communications, Arena features $3\,\mathrm{dBi}$ gain omnidirectional dualband VERT2450 antennas for transmission and reception (see Figure 3.47). These toroidal-radiation dipole antennas are optimized to work in the 2.4-2.5 GHz and 4.9-5.9 GHz operating frequency bands and offer a $50\ \Omega$ nominal impedance [344].

The 64 antennas are mounted on eight $15\ \mathrm{ft}$ rails hung off the ceiling. Rails have a 1.5×3 inches rectangular T-slotted profile with six open slots on each of its sides and one on the front side. Each rail hosts four *antenna pairs* spaced $5\ \mathrm{ft}$ from each other, while same pair antennas are spaced 1 inch only to mimic a transceiver radio device. The rail's robust structure provides enough strength to support the weight of eight AWG cables each, while their modular design permits antennas relocation along their whole length. In fact, antenna locations can be effortlessly adjusted by just sliding them along rails as shown in Figure 3.47.

3.5.3 Life-cycle of an Experiment

The Arena access diagram is shown in Figure 3.48. Upon getting an account on Arena, users can

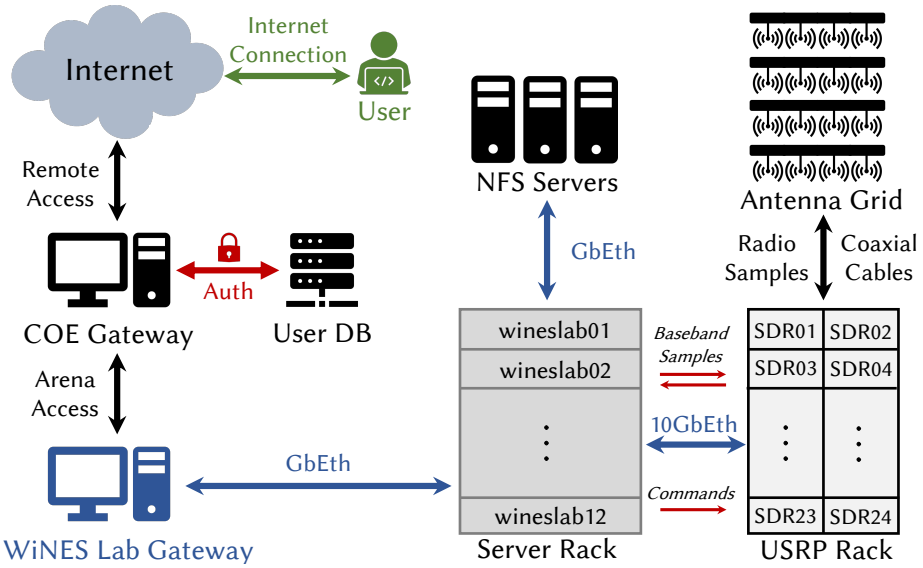


Figure 3.48: Arena access system diagram.

authenticate via SSH to the WiNES Lab Gateway through the COE Gateway[15]. At their first access, users will be allocated dedicated network disk space, accessible through any machine under the COE domain via the network file system protocol. Once logged into the WiNES Lab Gateway, it is possible to Secure Shell (SSH) to any of the 12 Arena servers.[16]

[15] `gateway.coe.neu.edu`
[16] `wineslab01-12.coe.neu.edu`

Since each server drives a subset of the available radios, the server selection can be made upon the user's experiment of choice. In this way, users can access one, some, or all of the 12 servers depending on their needs. Specifically, accessing servers `wineslab01-08` will allow driving one and only one USRP X310 per server (SDRs 17-24), servers `wineslab09-10` can be accessed to drive up to 8 USRPs N210 (SDRs 9-16), while servers `wineslab11-12` can be accessed to drive the remaining 8 USRPs N210 (SDRs 1-8).

Despite users being able to install their own software of choice on the network disk space allocated to them, basic development and testing software, such as Python, GNU Radio, and srsRAN, has already been installed on the servers and it is accessible by all users. On Arena, performing real-time wireless experiments is as simple as running pre-compiled software such as GNU Radio transmit an receive programs (e.g., `benchmark_tx.py` and `benchmark_rx.py`), or srsRAN core network and base station programs (e.g., `srsepc` and `srsenb`) and specify the desired radio to be driven by command line tool (e.g., `--args="addr=192.168.10.2"` to drive SDR 1) for USRP N210, while explicit radio addressing is not needed for USRP X310, which are one-to-one driven by servers `wineslab01-08`. The UHD, a user-space library that runs on a general-purpose processor, handles the baseband samples between the SDR and the control host. The control host software (e.g., GNU Radio), will report network status and measured metrics to the user such as transmitted, received, and correctly decoded packets, overall network throughput, and nodes interference levels, which can be saved on network disk or external drive for further analysis. To terminate an experiment session, the user can simply log out from the servers in use and from the WiNES Lab gateway.

Ultimately, Arena's real-time testing capabilities can be leverages to experiment with simple point-to-point links, test multi-hop transmissions, evaluate cellular network performance, or implement MIMO communication schemes, both for real-time research validation and live demonstrations.

3.5.4 Experimental Capabilities

While Arena can be leveraged to test point-to-point transmission techniques, such as narrowband and OFDM, employing diverse modulations, transmission powers, and other physical layer parameters, in this section we focus on providing some experimental use case scenarios where we test more complex communication schemes as well as evaluate some published work on Arena.

MIMO Capabilities. Among Arena highlights is its full-testbed symbol-level synchronization. This can be employed to implement distributed MIMO transmission schemes with the goal, for instance, of increasing the SINR of a wireless communication link. We implemented a 4-Multiple Input, Single Output (MISO) transmitter employing Maximum Ratio Transmission (MRT) beamforming at SDRs 1-2-3-4, and evaluated its performance against single antenna (SISO) transmission toward several single antenna receivers [345, 346] for the same overall output power. Figure 3.49 compares the received throughput for the two transmission schemes at 11 different receiver locations across the testbed. During this set of experiments, changing the network topology was as easy as addressing different receiver devices from GNU Radio, and did not require any SDR/antenna relocation.

Ad Hoc Networks. We also demonstrated ad hoc wireless networks implementing an instance of WNOS [347]. WNOS is a wireless network operating system for ad hoc networks that provides automated network control, interfacing the network designer with a simple control interface. WNOS takes network control programs defined on a centralized abstraction of the network, and automatically

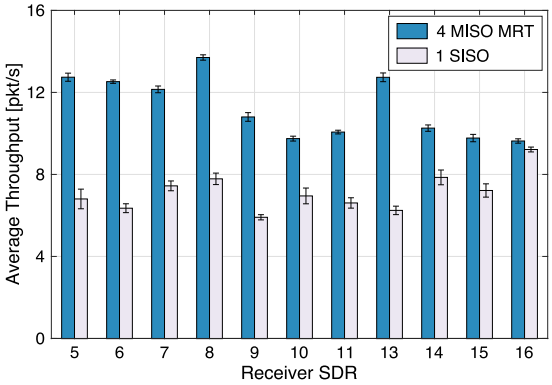


Figure 3.49: Average 4-MISO and SISO comparison at 11 different receivers. 95% confidence intervals are shown.

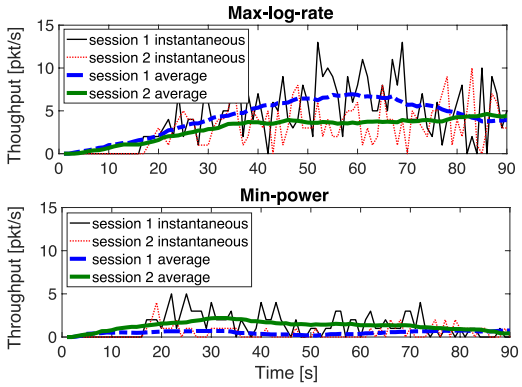


Figure 3.50: Single run and average results for two different control problems on a 14-node ad hoc network, using WNOS.

generates distributed cross-layer control programs based on distributed optimization theory. These are, then, executed by each individual network node on an abstract representation of the radio hardware. We implement a 14-nodes WNOS prototype on Arena, where two source nodes intend to deliver data to two destinations through 12 relay nodes, in a wireless multi-hop fashion. As source, relay, and destination nodes, we use SDRs 3-11, SDRs 1-2-5-6-7-9-10-13-14-15, and SDRs 8-16, respectively. We leverage WNOS to dictate two different network behaviors, namely *max-rate* and *min-power* (see [347] for details). The network performance for the two traffic sessions under the two different control problems is shown in Figure 3.50.

Cellular Networks. Arena can be used to easily evaluate cellular network scenarios with a high degree of realism. We herein showcase the implementation of a multi-cell LTE network and evaluate its performance for two different scenarios: high inter-cell interference and low inter-cell interference. In the former, two eNBs are located in close proximity to one another and have almost completely overlapping coverage areas, while in the latter, the two eNBs are located further away from each other. For both scenarios, each eNB serves three mobile subscribers continuously requesting data at the maximum rate. We used srsRAN, discussed in Section 2.2.2, to implement the cellular network. We implemented the eNBs on USRPs X310, specifically, we employed SDRs 21 and 22 for the high interference scenario, and SDRs 17 and 24 for the low interference one. As UE, we used Samsung Galaxy S5 commercial cellular phones. Figure 3.51 shows the total network throughput, as well as the average per-user throughput for the two deployments. Results report how inter-cell interference negatively affects the user average throughput and the sum network performance.

Cognitive Radio Networks. Arena can also be used to implement spectrum-sensing-based network solutions such as cognitive radios. Concepts like spectrum sharing, opportunistic spectrum access, and spectrum management typically rely on information gathering by idle listening on the wireless channel [348–350]. We herein show how Arena can be leveraged to gather information about Wi-Fi activity in the surrounding environment. Specifically, we leveraged an IEEE 802.11 GNU Radio implementation [331] to implement an SDR-based Wi-Fi receiver at SDRs 13-14-15-16-17-18-19-

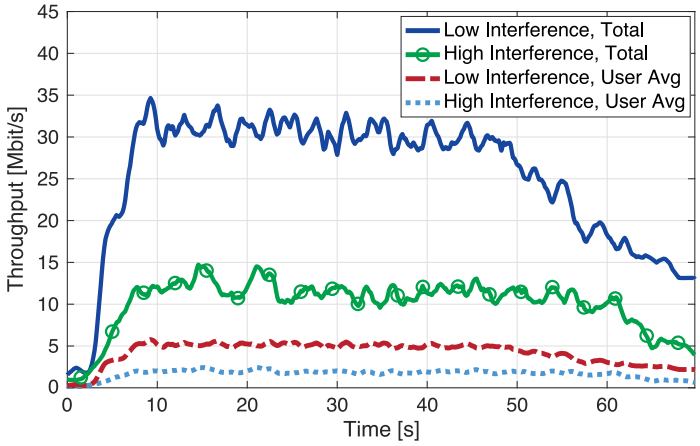


Figure 3.51: Network throughput in high and low inter-cell interference scenarios using 2 eNBs and 6 UEs with srsRAN.

20-21-22-23-24 and sense the Wi-Fi activity on Wi-Fi channel 6. Figure 3.52 illustrates the heatmap of the sensed Wi-Fi activity across the employed devices.

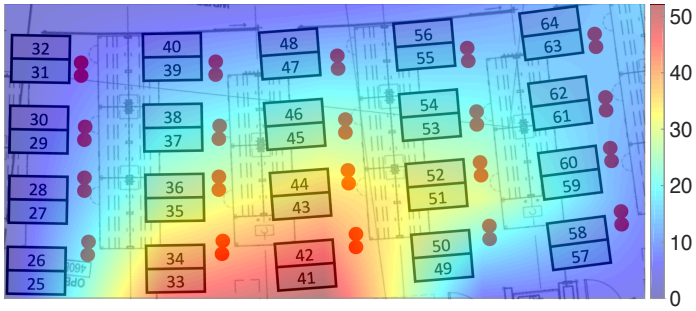


Figure 3.52: Heatmap of sensed Wi-Fi packets per second at different locations of Arena.

3.6 Conclusions

In this chapter, we presented OpenRAN Gym, the first publicly-available research platform for data-driven O-RAN experimentation at scale. OpenRAN Gym enables end-to-end design and testing of data-driven O-RAN xApps. We described the main components of OpenRAN Gym, detailing configuration options and procedures for experimenting at scale on Colosseum. We then gave an overview of the OpenRAN Gym xApp design and testing workflow, from user instantiation of their AI/ML models as xApps on the near-real-time RIC, to performing inference, prediction and/or control of base stations using live KPMs from the RAN. We provided an example of two xApps designed with OpenRAN Gym and used to control a large-scale O-RAN-managed network deployed on Colosseum. Additionally, we demonstrated how experiments and solutions designed with OpenRAN Gym on Colosseum can be ported to a set of heterogeneous testbeds, such as Arena, and the POWDER and COSMOS platforms of the U.S. National Science Foundation PAWR program.

We described in details the various building blocks of OpenRAN Gym. SCOPE—the data collection and control framework of OpenRAN Gym—is a development environment for softwarized and virtualized cellular networks that provides: (i) a ready-to-use portable open-source cellular container with flexible 5G-oriented functionalities; (ii) data collection tools, such as dataset generation functions for recording cellular performance and metrics, and for facilitating data analysis; (iii) a set of APIs to control and reprogram key functionalities of the full cellular stack at run time, without requiring redeploying the network, and (iv) an emulation environment with diverse cellular scenarios closely matching real-world deployments for precise prototyping NextG network solutions. We showcased SCOPE usage in the Colosseum network emulator and demonstrated its flexibility by porting it to real-world testbeds, both indoor and outdoor. Researchers can use SCOPE to design, implement and test novel control solutions on large-scale real-world cellular scenarios with different topologies, mobility patterns, channel and traffic characteristics.

ColO-RAN—forming the O-RAN control architecture of OpenRAN Gym—is a first-of-its-kind open, large-scale, experimental O-RAN framework for training and testing ML solutions for next-generation RANs. It combines O-RAN components, a softwarized RAN framework, and Colosseum, the world's largest, open, and publicly-available wireless network emulator with hardware-in-the-loop. ColO-RAN leverages Colosseum as a *wireless data factory* to generate large-scale datasets for ML training in a variety of RF environments, taking into account propagation and fading characteristics of real-world deployments. The ML models are deployed as O-RAN xApps on the near-real-time RIC, which connects to RAN nodes through O-RAN-compliant interfaces for data collection and closed-loop control.

Then, we detailed two of the experimental platforms for data colleciton and testing leveraged by OpenRAN Gym: Colosseum and Arena. Colosseum is the world's largest publicly accessible wireless network emulator with hardware-in-the-loop. The unique emulation capabilities and scale of Colosseum enable the design, prototyping, and testing of wireless solutions in a host of scenarios and channel conditions. We illustrated the architecture, emulation capabilities and operational modes of Colosseum, and we provide examples of large-scale experimentation in Colosseum, considering cellular, Wi-Fi, spectrum sharing and UAV scenarios.

Arena is an open-access wireless testing platform for sub-6 GHz spectrum research. The scale and deployment characteristics of Arena allow researchers to test medium- and short-range solutions at scale in an indoor real-world wireless environment. We discussed the architecture and design choices of Arena, then showcased exemplary applications and technologies that can be evaluated on Arena, such as MIMO transmission schemes, ad hoc networks, cellular, and cognitive radio networks. OpenRAN Gym has been made publicly-available to the research community, and opened up for community contributions and additions.

Chapter 4

O-RAN-based Control of Softwarized Cellular Networks

The *O-RAN Alliance*—a consortium of industry and academic institutions—is working toward realizing the vision of NextG cellular networks, where telecom operators use standardized interfaces to control multi-vendor infrastructures and deliver high performance services to their subscribers [143]. To achieve this goal, O-RAN embraces and promotes the 3GPP functional split, where base station functionalities are virtualized as network functions and divided across multiple network nodes, and proposes the RIC, a new architectural component that provides a centralized abstraction of the network, allowing operators to implement and deploy custom control plane functions (see Section 2.4.1).

While the O-RAN architectural vision is gaining momentum among researchers, the challenges of implementing it for data-driven, open, programmable and virtualized NextG networks are still largely to be dealt with. Important architectural questions are yet to be answered, including (i) the exact functionalities and parameters to be controlled by each network component; (ii) where to place network intelligence; (iii) how to validate and train data-driven control loop solutions, and (iv) how AI agents can access data and analytics from the RAN while minimizing the overhead of moving them from the RAN to the storage and inference locations. To answer these questions, in this chapter, which was derived from [526, 530], we provide the following contributions.

- We discuss how data-driven, closed-control loop solutions can be implemented in NextG RANs. We focus on the *opportunities* offered by the O-RAN architecture, including functional split and open interfaces, and on their role in advancing intelligent and programmable networks.

- Differently from prior work [317, 523], we investigate the *limitations* of the current O-RAN specifications and the *challenges* associated with deploying data-driven policies at different nodes of the RAN.

- We discuss how large-scale experimental testbeds will play a key role by providing researchers with heterogeneous and large datasets, critical to the success of data-driven solutions for cellular networks. We focus on the three PAWR platforms available at the time of this writing, i.e., POWDER, COSMOS, and AERPAW (described in Section 2.7), and on Colosseum and

Arena (described in Sections 3.4 and 3.5, respectively), which can all be used to generate massive datasets under a variety of network configurations and RF conditions.

- We provide the first demonstration of an O-RAN data-driven control loop in a *large-scale experimental testbed using open-source, programmable* RAN and RIC components. We deploy O-RAN on the Colosseum network emulator and use it to control multiple network slices instantiated on 4 SDR base stations serving 40 SDR UEs.

- We develop a set of DRL agents as RIC xApps to optimize key performance metrics for different network slices through *data-driven closed-control loops*. Experimental results show that our DRL approach outperforms other control strategies improving spectral efficiency by up to 20% and reducing buffer occupancy by up to 37%. We released the DRL agents and the 7 GB dataset used to train them.[1]

- We introduce the concept of *dApps*, distributed applications that complement xApps/rApps by implementing RAN intelligence at the CUs and DUs to address real-time use cases outside the timescales supported by the RICs.

The remainder of this chapter is organized as follows. We discuss how intelligent control schemes can be embedded in the O-RAN architecture in Section 4.1. We demonstrate the potential our O-RAN-based closed-control loops in Section 4.2. Finally, we introduce the concept of dApps to enable real-time control of the RAN in Section 4.3, and we draw our conclusions in Section 4.4.

4.1 Intelligent Wireless Architectures

Openness, programmability, and disaggregation are key enablers of data-driven applications. However, they are only the first step toward the seamless integration of AI and ML-based control loops in cellular networks. Typically, data-driven approaches involve several steps, ranging from data collection and processing, to training, model deployment and closed-control and testing.

This section illustrates how O-RAN is steering NextG deployments to bring intelligence to the network, by defining a practical architecture for the swift execution of data-driven operations, and discusses extensions to control procedures not currently considered by O-RAN.

Data Handling and Training Procedures. The effectiveness of data-driven approaches heavily depends on how data is handled, starting from data collection and aggregation at the RAN (where data is generated) to the point where it is processed for model training and inference. However, collecting and moving large amounts of data might result in significant overhead and latency costs. Hence, data-driven architectures must cope with tradeoffs between centralized approaches—providing a comprehensive view of the state of the network at the cost of overhead and latency—and distributed ones—which operate at the edge only, gather data from a small number of sources while enjoying low latency [80].

In this context, the O-RAN ML specifications introduce standardized interfaces (e.g., O1) to collect and distribute data across the entire infrastructure as well as operational guidelines for the deployment of ML and AI solutions in the network [283]. These include practical considerations

[1] https://github.com/wineslab/colosseum-oran-commag-dataset

on how, where and when models can be trained, tested and eventually deployed in the network. First, AI/ML models are made available to operators via a marketplace system similar to that of the well-established NFV MANO architecture, where models are stored in a catalog together with details on their control objectives, required resources, and expected inputs and outputs. Second, data-driven solutions must be trained and validated offline to avoid causing inefficiencies—or even outages—to the RAN. Indeed, since AI/ML techniques usually rely upon a randomized initialization, O-RAN requires all ML models to be trained and validated offline before their deployment [283]. As we will discuss next, albeit shielding the network from unwanted behavior, this requirement also limits the effectiveness of such approaches, especially the online ones. Online AI/ML techniques could still be used in O-RAN compliant architectures by allowing models to be trained with offline data in the non-real-time RIC, and then perform online learning in the near-real-time RIC. The smaller time-scale of the control loops of the latter would in fact allow the online training pipeline to be fed with data collected in real time.

Open Wireless Data Factories. Data-driven approaches aim at autonomously managing the network requiring little to no human intervention. Training and testing algorithms and data-driven closed-control loop policies require large amounts of data gathered in diverse scenarios, with varying traffic patterns, requirements and user behaviors, so that the resulting policy is effective when deployed in real networks.

Access to the massive amounts of data needed for training, however, is usually a privilege that only telecom operators enjoy. Owing to privacy and competition concerns, operators seldom share such data openly with the research community. As a consequence, researchers and practitioners are often constrained to rely on datasets collected in small laboratory setups, which seldom capture the variety and scale of real cellular deployments. In the context of intelligent networking for NextG cellular systems, large-scale wireless testbeds are needed for developing, training and testing new data-driven solutions, serving as *open wireless data factories* for the community. Examples of such open platforms—which would facilitate massive data collection in realistic and diverse wireless deployments [523]—are the city-scale platforms of the PAWR program, which have been described in Section 2.7, Colosseum, detailed in Section 3.4, and Arena, described in Section 3.5.

Control Loops. Figure 4.1 portraits how intelligence can be embedded at different layers and entities of a disaggregated cellular network together with the challenges and limitations of doing so. Each

Control and learning objective	Scale	Input data	Timescale	Architecture	Challenges and limitations
Policies, models, slicing	> 1000 devices	Infrastructure-level KPIs	Non real-time > 1 s	Service Management and Orchestration (SMO) *non real-time RIC*	Orchestration of very many near real-time RICs and CUs/DUs/RUs
User Session Management e.g., load balancing, handover	> 100 devices	CU-level KPIs e.g., number of sessions, PDCP traffic	Near real-time 10-1000 ms	A1, O1, *Near real-time RIC*, E2, gNB, *CU*	Process streams from multiple CUs and sessions
Medium Access Management e.g., scheduling policy, RAN slicing	> 100 devices	MAC-level KPIs e.g., PRB utilization, buffering	Near real-time 10-1000 ms	F1, *DU*	Operate at small time scales, make decisions involving several DUs/UEs
Radio Management e.g., resource scheduling, beamforming	~10 devices	MAC/PHY-level KPIs e.g., PRB utilization, channel estimation	Real-time < 10 ms	Open FH, Mobile devices, *RU*	Deployment of AI/ML models at the DU is not supported
Device DL/UL Management e.g., modulation, interference, blockage detection	1 device	I/Q samples	Real-time < 1 ms		Require device- and/or RU-level standardization

Figure 4.1: Learning-based closed-control loops in an O-RAN architecture.

closed-control loop optimizes RAN parameters and operations by running at different timescales, with different number of UEs, and using different sources for the input data. The O-RAN Alliance is also looking into how to standardize the data-driven workflows for these control loops. As of this writing, O-RAN only considers non and near-real-time loops, while real-time loops are left for future studies. Figure 4.1 also depicts the additional inference timescale below $1\ \mathrm{ms}$ to process raw I/Q samples and perform AI-driven PHY layer tasks, currently not part of O-RAN as it would require device- and/or RU-level standardization.

To better highlight the potential and limitations of the approach proposed by O-RAN, in the following we analyze each control loop individually, highlighting the role of each network component. Finally, we discuss how the current O-RAN architecture can be extended to realize the control loops and applications illustrated in Figure 4.1.

4.1.1 Non-Real-time Control Loop

The O-RAN Alliance defines *non-real-time* any control loop that operates on a *timescale of at least one second.* As shown in Figure 4.1, this involves the coordination between the *non-real-time* and *near-real-time* RIC through the A1 interface. This control loop manages the orchestration of resources at the infrastructure level, making decisions and applying policies that impact thousands of devices. These actions can be performed using data-driven optimization algorithms processing data from multiple sources, and inference models deployed on the non-real-time RIC itself.

Practical examples of non-real-time data-driven control include instantiating and orchestrating network slices, as well as selecting which pre-trained inference models in the catalog should be deployed to accomplish operator intents, and deciding in which near-real-time RIC these models should be executed. Said decisions can be made according to a variety of factors, ranging from computational resources and data availability to minimum performance requirements to comply with Service Level Agreements. Moreover, since the non-real-time RIC is endowed with Service Management and Orchestration capabilities, this control loop can also handle the association between the near-real-time RIC and the DUs/CUs. This is particularly useful in virtualized systems where DUs and CUs are dynamically instantiated on-demand to match the requests and load of the RAN. However, non-real-time loops are challenging to actuate in practice because of the very many interactions among the non-real-time RIC and the network elements, which require tight coordination, data collection and orchestration capabilities.

4.1.2 Near-Real-time Control Loops

Near-real-time control loops operate on a timescale *between* $10\ \mathrm{ms}$ *and* $1\ \mathrm{s}$. As shown in Figure 4.1, they run between the near-real-time RIC and two components of the gNBs: the CU and the DU. Because one near-real-time RIC is associated to multiple gNBs, these control loops can make decisions affecting up to thousands of UEs, using user-session aggregated data and MAC/PHY layer KPIs. ML-based algorithms are implemented as external applications, i.e., *xApps*, and are deployed on the near-real-time RIC to deliver specific services such as inference, classification, and prediction pipelines to optimize the per-user quality of experience, controlling load balancing and handover processes, or the scheduling and beamforming design. Challenges of near-real-time control loops

include the need to promptly make decisions in a matter of tens or hundreds of milliseconds for each of the several CUs and DUs controlled by the RIC.

4.1.3 Real-time Control Loops

A crucial component of the operations of a cellular network involves actions at a *sub-*10 ms*—or even sub-ms—timescale.* In O-RAN, these operations are labeled as *real-time control loops,* and mainly concern interactions between elements in the DU. Control loops at a similar timescale could also be envisioned to operate between the DU and the RU, or at the UEs. However, as deploying ML solutions at the DU is not currently supported, these loops are left for future extensions of the O-RAN specifications. We propose a practical way to achieve such real-time control of the RAN through the concept of *dApps*, which we discuss in Section 4.3.

Finally, data-driven approaches at the lower layers of the protocol stack or at the device, i.e., involving sub-ms timescales, are extremely powerful and can be used for data-driven scheduling decisions [316] and for feedback-less detection of PHY layer parameters (e.g., modulation and coding scheme, and interference recognition) [351]. Overall, the fact that device-/RU-level standardization is required for sub-ms loops makes it very challenging to realize them in practice, thus limiting their applicability.

4.2 Scheduling Control in Sliced 5G Networks through O-RAN RIC

This section showcases an example of a data-driven closed-loop control implemented using the O-RAN Software Community near-real-time RIC and an open cellular stack on Colosseum (Figure 4.2). We demonstrate the feasibility of a closed-control loop where DRL agents running in xApps on the near-real-time RIC select the best-performing scheduling policy for each RAN slice.

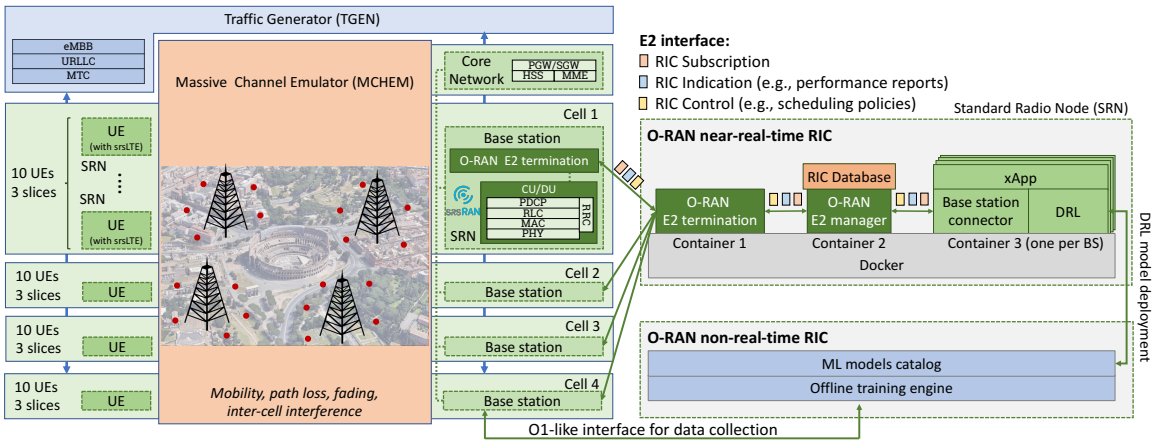


Figure 4.2: O-RAN integration in Colosseum.

Experimental Scenario. We have emulated a 5G network with 4 base stations and 40 UEs (Figure 4.2, left) in the dense urban scenario of Rome, Italy. The locations of the base stations have been extracted from OpenCelliD (a database of real-world cellular deployments) and cover an area

of $0.11\ \mathrm{km}^2$. (See Section 3.2.3.2 for a detailed description of this urban scenario.) Downlink and uplink frequencies have been set to 0.98 and $1.02\ \mathrm{GHz}$, respectively; the channel bandwidth to $3\ \mathrm{MHz}$. While these parameters might be atypical for 5G, their choice depends on the Colosseum environment. We note, however, that this does not affect our findings on how data-driven solutions improve the RAN performance.

We consider a multi-slice scenario in which UEs are statically assigned to a slice of the network and request three different traffic types, i.e., high capacity eMBB, URLLC, and MTC. This reflects the case, for instance, of telecom operators providing different levels of service to different devices (e.g., MTC service to IoT-enabled devices, or URLLC to devices for time-critical applications). The base stations serve each slice with a dedicated—and possibly different—scheduling policy, selecting among Proportionally Fair (PF), Waterfilling (WF), and Round Robin (RR). We also consider the case where the number of PRBs allocated to each slice varies over time [12, 537].

We used the SCOPE framework of Section 3.2, which is based on srsRAN (described in Section 2.2.2), to implement our softwarized cellular network. Although this framework is not yet fully compliant with the NR specifications, we are confident that our DRL-based approach enabled by O-RAN can be easily extended to future NR-compliant versions of this (or of any other) software where base stations expose control interfaces to the network. For ease of prototyping, we co-located the core network on the same SRN that also runs the base station application. For the purposes of this chapter, this setup is equivalent to deploying the core network on a dedicated SRN (see Figure 4.2). The scenario we considered concerns pedestrian user mobility with time-varying path-loss and channel conditions. Traffic among base stations and UEs is generated through the Colosseum TGEN, configured to send different traffic types to UEs of different slices, i.e., eMBB ($1\ \mathrm{Mbps}$ constant bitrate traffic), URLLC (Poisson traffic, with $10\ \mathrm{pkt/s}$ of $125\ \mathrm{bytes}$) and MTC (Poisson traffic, with $30\ \mathrm{pkt/s}$ of $125\ \mathrm{bytes}$). For each base station, the UE-slice allocation is as follows: eMBB and URLLC slices serve 3 UEs each, while MTC slices serve 4 UEs. We embedded the DRL agents into xApps running in the near-real-time RIC (right of Figure 4.2), for a total of 12 DRL agents running in parallel and making decisions with a time granularity of $500\ \mathrm{ms}$. Agents connect with the network base stations through the O-RAN E2 interface. This interface is composed of two elements: the application protocol, and the SM [142]. The former defines the set of messages that the near-real-time RIC and the RAN nodes can exchange, and the procedures for the RAN node subscription to the RIC. The SM, instead, defines which parameters of the RAN nodes can be controlled by the RIC to achieve a given closed-loop control objective. Specifically, the E2 interface exposes analytics and the scheduler policy selection using a custom SM. As shown in Figure 4.2, xApps interface with the base stations through the O-RAN E2 manager, which ultimately connects with the base stations via the E2 interface. Other components of the RIC include the RIC database, which keeps entries on the connected base stations, the training engine, and the ML model catalog, which deploys the DRL model chosen by the telecom operator on the near-real-time RIC. Finally, messages internal to the RIC are managed by the RIC Message Router, a library which associates message types to destination endpoints.

DRL Agent Training. To train our DRL agents we generated some $7\ \mathrm{GB}$ of training data of various performance metrics (e.g., throughput and bit error rate), system state information (e.g., transmission queue size, signal-to-interference-plus-noise ratio, and channel quality information) and resource allocation strategies (e.g., slicing and scheduling policies) by running a total of 89

hours of experiments on Colosseum. Each DRL agent has been trained via the PPO algorithm to manage a single slice for fine-grained and flexible control of the whole cellular network. Agents have been trained under network configurations obtained by varying the distance between base stations and UEs and the mobility of the UEs. Testing has been performed in the most challenging setup, which includes the random mobility of the UEs. Although the training is performed with the same topology configuration, we notice that our agents are topology-independent, as each of them controls a single slice for a given base station. Specifically, agents process the performance metrics received by the base station they are controlling—which possibly expresses the performance of several UEs—through an encoder. This allows them to cast the dimensionality of the input data to a fixed size and to process it regardless of the number of active UEs of the slice. As a consequence, the DRL agents do not need to be aware of the number of UEs and base stations in the network, which makes our approach general and scalable. Through the RIC Indication messages sent via the O-RAN E2 interface (Figure 4.2), the agent is fed real-time performance measurements of the slice it controls. These messages generate an overhead of 72 $\mathrm{bytes/s}$ per UE. Data goes through an encoder for dimensionality reduction and is then used by the agent to identify the state of the system. The agent uses a fully connected neural network with 5 layers and 30 neurons each to determine the best scheduling policy for the corresponding slice. This policy is then signaled to the corresponding base station through RIC Control messages sent via the E2 interface. The reward of the agents depends on the specific slice and the corresponding KPI requirements. Specifically, eMBB and MTC agents have been trained to maximize the throughput of UEs; the URLLC agent has been trained to minimize latency by allocating resources (i.e., PRBs) as quickly as possible. To comply with O-RAN directives, we have trained the DRL agents offline in the non-real-time RIC, which also performs the initial data-collection and deploys the model in the near-real-time RIC. We have then tested them on the emulated Colosseum scenario.

Experimental Results. Figure 4.3 shows the CDF of the downlink spectral efficiency of the eMBB slice. We compare the performance of the network when DRL agents dynamically select the best

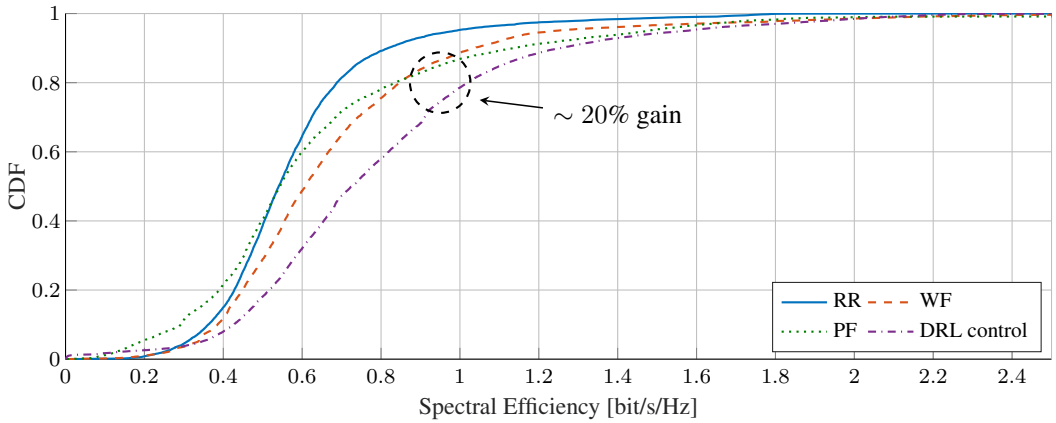


Figure 4.3: Downlink spectral efficiency of the eMBB slice for different scheduling policies and with DRL control.

scheduling strategy among RR, PF and WF against the case where scheduling strategies are fixed

over time. Our results clearly indicate that data-driven optimization outperforms fixed policies by delivering gains in spectral efficiency that are up to 20% higher than that of the best performing static policy. This is due to the fact that eMBB traffic requires high data-rates and DRL agents are capable of dynamically adapting scheduling decisions to the current network state and traffic demand.

Figure 4.4 shows the ratio of PRBs allocated to UEs of the URLLC slice for different scheduling policies. We observe that RR is not even capable of satisfying the requirements of the URLLC

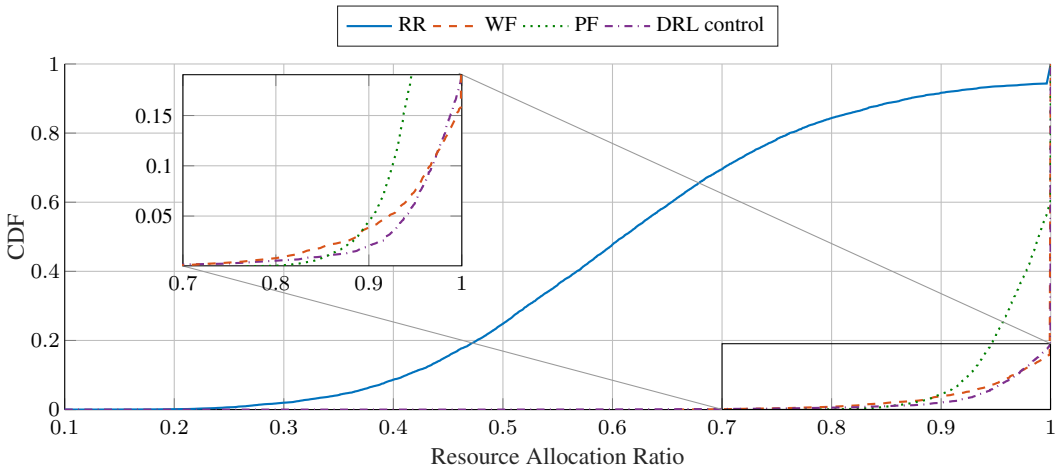


Figure 4.4: PRB allocation ratio of the URLLC slice for different scheduling policies and with DRL control.

UEs. The DRL agent achieves the best performance, ensuring that all UEs are granted the resource they request. Our results also show that using DRL obtains a downlink buffer occupancy at the base station that is 37%, 5% and 17% less of that of RR, WF and PF, respectively.

Figure 4.5 shows the CDF of the downlink buffer size for the URLLC slice under different scheduling policies. Low buffer size indicates timely data delivery to requesting UEs; higher buffer

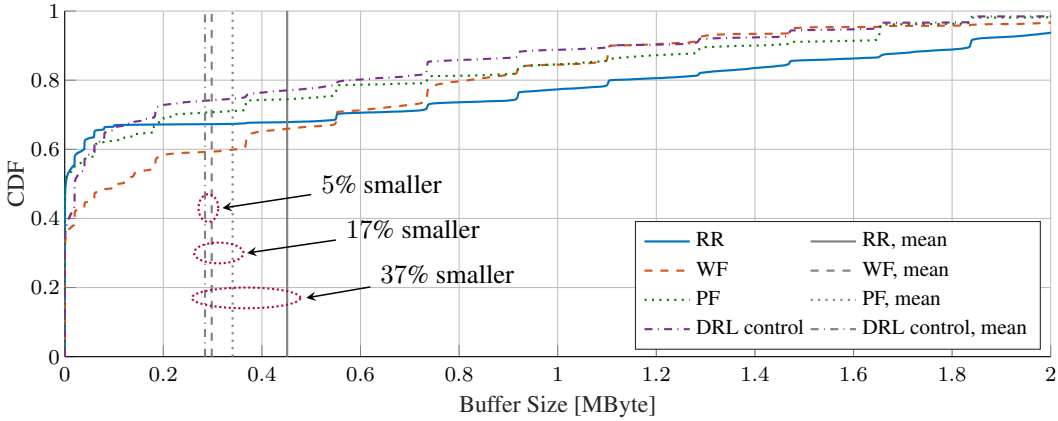


Figure 4.5: Downlink buffer size of the URLLC slice for different scheduling policies and with DRL control.

size results in higher latency due to packets waiting in the queue. Results show that DRL agents serve the UEs faster than the static policies, resulting in a lower latency. Particularly, the average

buffer size of the URLLC slice when using DRL control is 37%, 5% and 17% smaller than that of the RR, WF and PF scheduling policies, respectively. The DRL agent also significantly outperforms the WF policy between the 50th and 90th percentiles.

Figure 4.6 depicts how often DRL agents select specific scheduling policies as a function of the number of PRBs of each slice. The bigger the circle, the higher the probability of selecting a given policy. We observe that MTC and eMBB DRL agents select WF with 99% probability. Also,

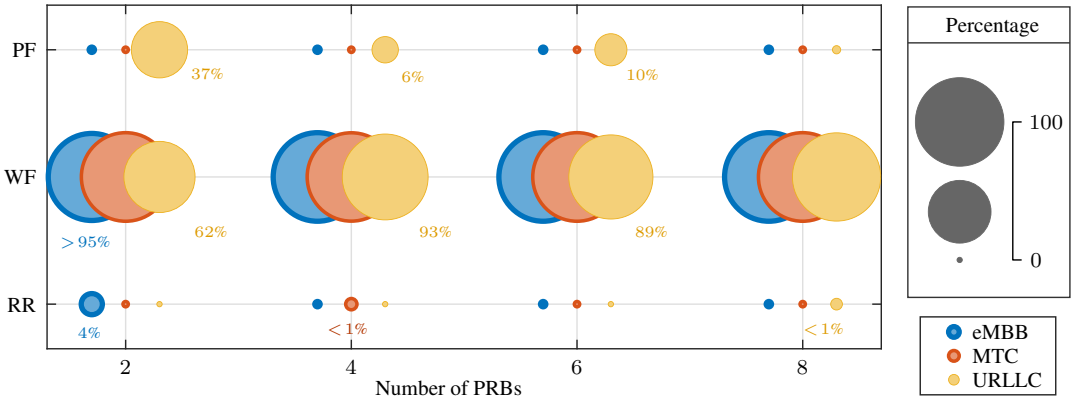


Figure 4.6: DRL action selection distribution vs. number of slice PRBs. Values $> 99\%$ (big circles) or $< 0.5\%$ (small circles) are omitted.

eMBB agents select RR with 4% probability when only a few PRBs are allocated to the slice. On the contrary, URLLC DRL agents are likely to select both PF and WF scheduling policies even when more PRBs are available. These results show that adapting control strategies to current network state and traffic requirements is essential to achieve remarkable performance improvements (Figure 4.3 and 4.5). DRL agents dynamically select the best performing scheduling strategy based on available resources and current network state, providing performance gains simply unattainable with static scheduling policies.

4.3 dApps: Extending O-RAN for Real-time Inference and Control

The RICs, xApps and rApps will eventually realize the vision of self-organizing networks, where components autonomously detect ongoing changes in channel, network, and traffic state, and react to meet minimum QoS requirements and to comply with SLAs. Practical control examples include resource allocation [352], network slicing [353], handover and mobility management [354], and spectrum coexistence [355].

Although these examples can benefit from the instantiation and execution of network intelligence, we are still far from the vision of fully-automated and intelligent cellular networks. Indeed, limiting the execution of control applications to the near-real-time and non-real-time RICs prevents the use of data-driven solutions in cases where control decisions and inference must be made in *real time*, or within temporal windows shorter than the 10 ms supported by near-real-time control loops [262, 526]. Two practical examples are user scheduling and beam management applications. Scheduling requires making decisions at sub-ms timescales (e.g., to perform puncturing and preemption to support URLLC traffic with tolerable latency values as low as 1 ms). Similarly, beam management involves

beam sweeping via reference signals transmitted within bursts that are 5 ms long (half the duration of a 5G NR frame) [340].

Unfortunately, the near-real-time RIC and xApps might struggle in accomplishing these procedures because they have limited access to low-level information (e.g., transmissions queues, I/Q samples, beam directionality) and/or incur high latency to obtain this information. For example, beam management would require the transmission of reference signals (or, as proposed in [340], I/Q samples) from the DU/RU to the RIC over the E2 interface. This would result in increased overhead and delay due to propagation, transmission, switching, and inference latency, which might prevent real-time (i.e., < 10 ms) execution. Moreover, since I/Q samples might contain sensitive user-related data (e.g., packet payload), they cannot be transmitted to the RIC out of privacy and security concerns and, therefore, they are processed at the gNB directly. For these reasons, such procedures (and any procedure that requires real-time execution, or handles sensible data) are typically run directly at the DU/RU, usually via closed and proprietary hardware and software implementations—referred to as the "vendor's secret sauce" by many in the industry. While hardware-based implementations can satisfy the above temporal requirements and deliver high performance, they are ultimately inflexible, hard to upgrade, and not scalable. For example, updates of hardware functionalities (e.g., after a new 3GPP release) require hardware or (whenever possible) firmware updates at the DUs/RUs.

As of today, the O-RAN architecture is primarily focused on offering softwarized programmatic and AI-based control to the higher layers of the RAN protocol stack, with limited flexibility for the lower layers hosted at DUs/RUs. However, prior work has already demonstrated how running AI at the edge of the network—with a specific focus on the PHY and MAC layers of the DUs/RUs—can provide major performance benefits. Moreover, recent works have shown that AI at the edge can significantly improve network performance by leveraging traditionally available KPMs (e.g., throughput, SINR, channel quality information, latency) [353, 356, 357], as well as by processing in parallel (thus not affecting demodulation and decoding procedures) I/Q samples collected at the PHY layer that carry detailed information on channel conditions and spatial information of the received waveforms [340, 351]. Although the O-RAN specifications have identified a few use cases that could benefit from running intelligence at gNBs directly, these use cases are left for future studies [527].

The goal of this section is to foster a discussion on enabling network intelligence at the edge in the O-RAN ecosystem. We introduce the notion of *dApps*, custom and distributed applications that complement xApps/rApps by implementing RAN intelligence at the CUs/DUs to address real-time use cases outside the timescales of the current RICs (see Figure 4.1). dApps receive real-time data and KPMs from the RUs (e.g., frequency-domain I/Q samples), DUs (e.g., buffer size, QoS levels), and CUs (e.g., mobility, radio link state), as well as Enrichment Information (EI) from the near-real-time RIC, and use it to execute real-time inference and control of lower-layer functionalities. We build on already available logical components and propose an extension of the O-RAN architecture—requiring minimal modification to the specifications—to include the concept of dApps. Finally, we discuss challenges specific to dApps and their instantiation, and provide preliminary results—obtained through experiments on the Colosseum testbed [526, 545]—that demonstrate the benefits of executing network intelligence through dApps.

4.3.1 Why dApps?

dApps are distributed applications that complement xApps/rApps and enable the execution of RAN intelligence at CUs/DUs to support real-time use cases requiring tighter timescales than those implemented by the RICs. This section identifies the advantages of dApps and discusses relevant use cases and applications that would benefit from them.

Reduced Latency and Overhead. Moving networking functionalities and services to the edge is undeniably one of the most efficient ways to reduce latency. The near-real-time RIC brings the network control closer to the edge, but it primarily executes in cloud facilities [527]. Therefore, data needs to travel from the DUs to the xApps in the near-real-time RIC, and the output of the inference needs to be sent back to the DUs/RUs. This results in increased latency and overhead over the E2 interface to support data collection, inference and control. This problem can be mitigated by executing procedures that require real-time access at the CUs/DUs directly via dApps, which substantially reduces both latency and overhead (in Section 4.3.4 we show a 3.57$\times$ reduction in overhead compared to the case where intelligence can only run at the near-real-time RIC).

AI at the Edge. While just a few years ago AI (and specifically ML) was associated with data centers with hundreds of GPUs, nowadays there is plenty of evidence on the feasibility of training and executing AI on resource-constrained edge nodes with a limited footprint [358]. GPUs are now smaller, more powerful, cheaper, and widely available. Technological advances in AI have resulted in procedures and techniques (e.g., pruning [358]) that make it possible to compress ML-solutions by 27$\times$ and reduce inference times by 17$\times$ while resulting in a negligible accuracy loss of 1%.

Controlling MAC- and PHY-layer Functionalities. Another important aspect is related to the feasibility of controlling lower-layer functionalities of the MAC and PHY layers. These include fundamental procedures related to scheduling, modulation, coding and beamforming, which all operate at sub-ms timescales and require real-time execution. As a consequence, while xApps can be used to select which scheduling policy to use at the DU (e.g., round-robin), they cannot allocate resource elements to UEs in real time at the sub-frame level (e.g., to perform puncturing and preemption for URLLC traffic). Moreover, many PHY-layer functionalities operate in the I/Q domain. Recent advances demonstrate how core functionalities of this layer (e.g., beamforming, modulation recognition, channel equalization, radio-frequency fingerprinting-based authentication) can be executed in software with increased flexibility, reduced complexity, and higher scalability by processing the I/Q samples directly [340]. Because of these tight time constraints and security concerns, xApps and rApps—which operate far from the DUs—unlike dApps, are not suitable to make decisions on these functionalities.

Access DU/CU Data and Functionalities in Real Time. dApps executing at the DU/CU also make it possible to access control- and user-plane data that is either unavailable at the near-real-time RIC, or available but not with a sub-ms latency. This includes real-time access to user-plane information (e.g., I/Q samples, data packets), handover-related mobility information, dual-connectivity between 5G NR and 4G, among others. By executing at the DUs/CUs, dApps will be able to access UE-specific metrics and data to deliver higher performance services tailored to individual UE requirements, and instantaneous channel and network conditions.

Extensibility and Reconfigurability. The majority of currently available DUs and RUs leverage hardware-based implementations of MAC and PHY functionalities [527] that strongly limit their extensibility and reprogrammability. On the contrary, the integration of dApps within the O-RAN ecosystem offers the ideal platform for software-based implementations of the above functionalities, and thus facilitates their extension and real-time reconfiguration. In this context, the O-RAN Alliance is developing standardized interfaces to support hardware acceleration in O-RAN [527], which is a first step toward the integration of AI within DUs and RUs.

4.3.2 Challenges and Open Issues

Despite the above advantages, bringing intelligence to the edge comes with several challenges:

- *Resource availability.* First and foremost, AI solutions require computational capabilities to quickly and reliably perform inference. For this reason, the DUs must be equipped with enough computational power to support the network intelligence. In this context, GPUs, CPUs, FPGAs and hardware acceleration will play a key role in the success of dApps.

- *Softwarized ecosystem.* Similar to the RIC, CUs/DUs will need a container-based platform to support the simultaneous instantiation, execution, and lifecycle management of dApps. At the same time, dApps must operate without halting or delaying the real-time execution of gNB functionalities. In this context, hardware acceleration will be pivotal in guaranteeing that dApps execute reliably and fast.

- *Standardized interfaces for DUs/CUs.* The execution of intelligence at the edge requires interfaces between DUs, CUs, and dApps that offer similar functionalities to those currently available to the RICs and other O-RAN components. In this way, DUs can expose supported control and data collection capabilities to CUs and the near-real-time RIC. This requirement is key to make sure that dApps are platform-independent and that they all follow the same specifications, and enable the interactions with other O-RAN components and applications.

- *Orchestration of the intelligence.* The introduction of dApps comes with additional diversity and complexity. Indeed, there is a need to design orchestration solutions for O-RAN systems that can determine how to split and distribute the intelligence according to data availability, control timescales, geographical requirements and network workload, while satisfying operator intents and SLAs. Hence, the near-real-time RIC and its xApps will play an important role in determining which control and inference tasks are executed via dApps at CUs/DUs, and which at the near-real-time RIC via xApps.

- *Friction from vendors.* Traditionally, gNB components host a substantial part of vendor's intellectual property (e.g., schedulers, beamforming, queue management). Enabling third-party applications at DUs and CUs will inevitably reduce the value of such intellectual property. Although the introduction of dApps may foster competitiveness and innovation, it might inevitably find friction from vendors. Another concern is often related to the monolithic development approach of RAN vendors, which would prevent the execution of third-party components such as dApps. Nonetheless, the xApp paradigm has already shown that it is possible to separate the RAN state machine between gNB nodes and the RICs for control in the

near- or non-real-time timescales. This shows that monolithic, inflexible approaches are not the only option, and a similar approach can be adopted to implement dApps while satisfying real-time control latency constraints thanks to their distributedness.

4.3.3 Proposed Architecture

In this section, we discuss a possible architectural extension (shown in Figure 4.7) that enables the execution of dApps while requiring minimal changes to the already existing O-RAN architecture.

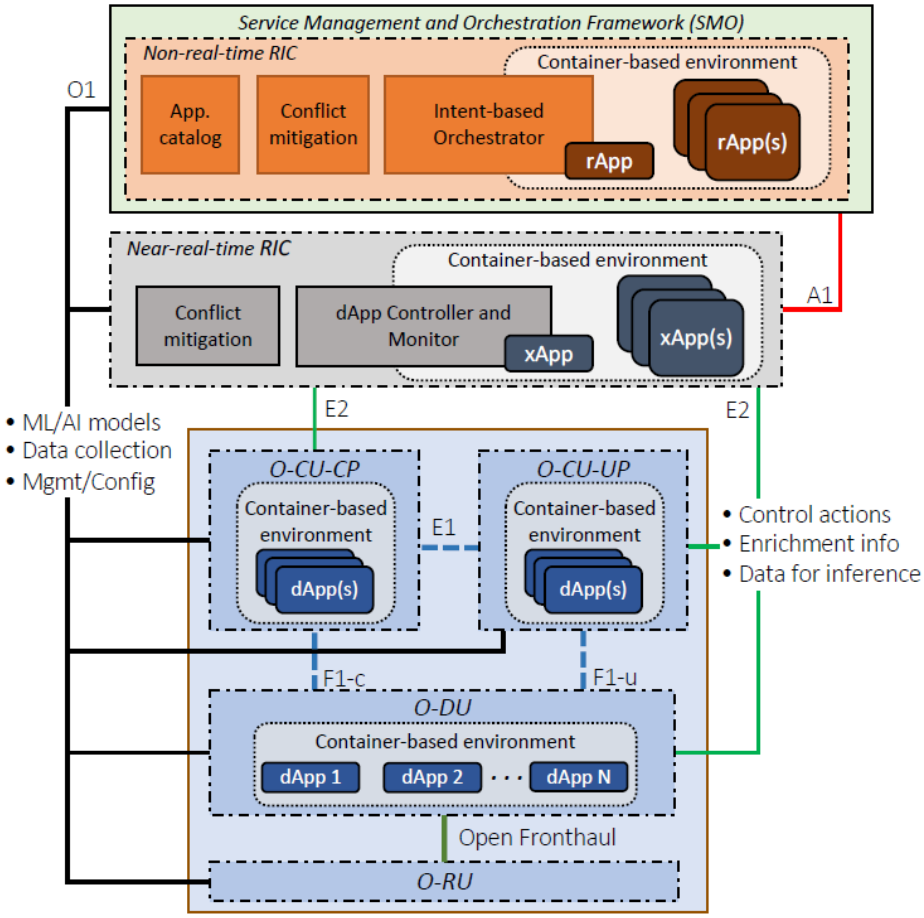


Figure 4.7: dApps and proposed extension to the O-RAN architecture.

dApps as Softwarized Containers. Similarly to xApps and rApps, dApps are based upon a containerized architecture. This makes it possible to: (i) seamlessly manage the lifecycle of dApps, i.e., deployment, execution and termination; (ii) facilitate the integration and use of new (or updated) functionalities included in newly-released O-RAN specifications via software updates; (iii) provide an abstraction level where the CUs, DUs, and RUs advertise the tunable parameters and functionalities (similarly to what is already envisioned for xApps and the E2 interface), thus making it possible to deploy dApps that are tailored to control specific parameters; (iv) achieve hardware-independent implementations of dApps, which can be offered as standalone O-RAN applications in a marketplace,

and (v) facilitate the development and use of AI-based solutions for the lower layers of the protocol stack. This approach also requires a resource manager in place that allows containers to access and share the physical resources (e.g., CPUs, GPUs, memory) available in the RAN nodes.

Leveraging O-RAN Interfaces. The O-RAN interfaces currently available can be extended and used to support the deployment, execution and management of dApps:

- *Real-time user-plane interface.* Currently, the O-RAN specifications do not envision data-driven control based on analysis and inference of user-plane data, including I/Q samples and data packets. These, however, can be the basis for several data-driven use cases, discussed in Section 4.3.4. Therefore, dApps executing at the DU receive (i) waveform samples in the frequency domain from the RU over the O-RAN Fronthaul interface (processed in parallel to decoding procedures to not halt ongoing communications), as well as (ii) transport blocks, or RLC packets that are already locally available at the DU. Similarly, dApps executing at the CU will be able to perform inference on locally available data pertaining to the PDCP or to the SDAP.

- *Real-time control-plane interface.* dApps can receive data for online inference/control by adapting and extending the SMs defined for the E2 interface. As of today, this feature is not available, as it requires an extension to the E2 interface to let dApps extract relevant KPMs using an adapted E2SM KPM within a latency of 10 ms to support real-time execution.

- *dApps deployment.* Similarly to xApps, dApps are dispatched via the O1 interface.

- *Enrichment information.* Similar to how xApps receive EI from the non-real-time RIC via the A1 interface, dApps can receive EI from the near-real-time RIC via the E2 interface. In this case, xApps process data from one or more gNBs, and send EI to the dApps, which use it to make decisions on control operations. For example, a DU can receive traffic forecasts from the near-real-time RIC, and use this information to control scheduling, MCS, and beamforming of the RUs.

Extending Conflict Mitigation to dApps. The O-RAN specifications already envision conflict mitigation components that ensure that the same parameters (or functionalities) are controlled by at most one O-RAN application at any given time. The introduction of dApps will increase the system complexity and further emphasize the importance of conflict detection and mitigation. Indeed, dApps will require conflict mitigation components operating at stricter timescales (compared to those currently envisioned by O-RAN) to identify conflicts between dApps operating at sub-ms timescales.

Intent-based O-RAN Apps Orchestrator. The abundance of O-RAN applications (e.g., rApps, xApps, dApps) will require automated solutions capable of determining which applications should be executed and where.

For this reason, we envision an orchestration module in charge of managing these applications. This component should reside in the non-real-time RIC and can run either as an rApp, or as a standalone component (it can also be hosted in the service management and orchestration domain). This module converts goals and requirements of the operator (e.g., expressed in YAML/XML/JSON formats) into a set of O-RAN applications that constitute a fabric of intelligent modules to meet the

desired intent. Then, it dispatches them from the application catalog where they reside to the RAN location where they are executed, thus creating a complex ecosystem of chained applications that cooperate to achieve the operator intent.

To achieve this, the orchestrator needs to understand the intent specified by the operator, and compute the optimal configuration and set of applications to instantiate and where [546]. This is performed by ensuring that applications are executed only at network nodes: (i) where input data can be made available within the required timescale; (ii) that can actually control the required parameters and functionalities, and (iii) with enough physical resources (e.g., CPUs/GPUs/FPGAs) to support the required applications. For example, if an operator wants to perform real-time beam detection and traffic forecasting for a set of gNBs, the orchestrator needs to deploy a dApp that executes at the DU (where the I/Q samples are available through the Open Fronthaul interface) to perform beam detection, and an xApp at the near-real-time RIC (that receives traffic-related KPMs from the CUs via the E2 interface) to perform traffic forecasting.

dApp Controller and Monitor. As shown in Figure 4.7, this component is hosted in the near-real-time RIC, and is in charge of controlling and monitoring dApps executing at the gNBs. Specifically, it ensures that dApps meet the desired QoS levels and are in line with the operator intent. As a possible extension, this component can also convert an xApp into multiple atomic dApps dispatched and executed at the gNB components to provide a finer control of the RAN procedures. In this case, the dispatchment can be coordinated by the non-real-time RIC, and performed via the O1 interface.

4.3.4 Use Cases and Results

In this section, we discuss relevant use cases that would benefit from the introduction of dApps in the O-RAN ecosystem, and present preliminary results.

Beam Management. dApps can be used to extend the beam management capabilities of NR gNBs. The 3GPP specifies a set of synchronization and reference signals to evaluate the quality of specific beams, and to allow the UE and the RAN to use intelligent algorithms [340, 359] that select the best combination of transmit and receive beams.

These techniques, however, require a dedicated implementation on RAN components, which is often provided by vendors as a black box. In this case, xApps and rApps can only embed logic to control high-level parameters for beam management, e.g., select and deploy a codebook at the RU based on KPMs or coarse channel measurements. On the contrary, dApps can truly open the ecosystem to custom beam management logic, in which the dApp itself selects the beams to use and/or explore, rather than only providing high-level policy guidance.

For example, in [340] we introduced DeepBeam, a beam management framework that leverages deep learning on the I/Q samples to infer the Angle of Arrival (AoA) and which beam is the transmitter using in a certain codebook. DeepBeam is thus an example of a data-driven algorithm that cannot be deployed at the RICs, as it requires access to user-plane I/Q samples for inference. This approach is an ideal candidate for deployment in a dApp, as it requires access to information that can be easily exposed by a DU in real time (i.e., the frequency-domain waveform samples), but cannot be transferred to another component of the network without (i) violating control latency constraints, (ii) exposing sensitive user data; and (iii) increasing the traffic on the E2 or O1 interface excessively. As an example, Figure 4.8 reports the data rate (and time) needed to transfer the I/Q

samples required to perform inference with the DeepBeam convolutional neural networks from a DU to the near-real-time RIC. DeepBeam can perform inference and classify the transmit beam and

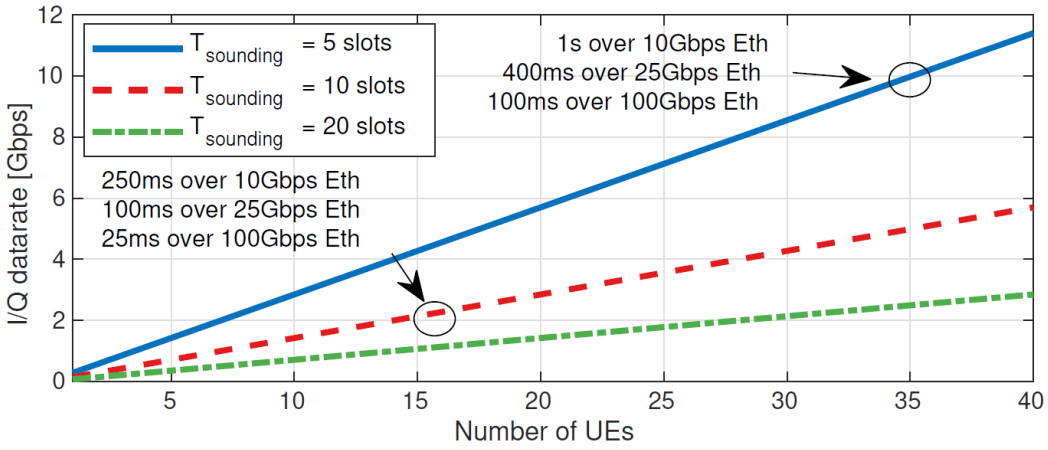


Figure 4.8: Data rate and latency to perform I/Q-based beam management over E2 interface. In most cases, latency is higher than 10 ms (required to perform real-time beam management).

AoA using any kind of samples (e.g., from packets or sounding signals) [340]. As a reference, in this case we consider the number of samples that can be collected through 3GPP NR SRSs. We use 3GPP-based parameters and assume that each SRS uses 3300 subcarriers (i.e., the full bandwidth available to NR UEs), 2 symbols in time, a periodicity T_{sounding} of 5, 10, or 20 slots, and that each UE monitors 3 uplink beams. The I/Q samples have 9 bits, and we assume numerology 3 (i.e., slots of 125 $^{-}$s). The results show that it would be impractical to transfer the required amount of samples because of timing (i.e., no real-time control) and of the data rate required, which can reach more than 100 Gbps in certain configurations.

Supporting Low-latency Applications. Another application of practical relevance is that of dApps to support real-time and low-latency applications by, for example, controlling RAN slicing and scheduling decisions. Indeed, the timescale at which dApps operate is appropriate to access UE-specific information from the DU in real time (e.g., buffer size, MCS profile, instantaneous SINR), and to make decisions on the RAN slicing and resource allocation strategies based on QoS requirements and network conditions.

To showcase the benefits of dApps and their integration with other O-RAN components, we trained a set of ML solutions for O-RAN applications. Specifically, we trained two DRL agents that process input data from the RAN (i.e., downlink buffer occupancy, throughput, traffic demand) to control the scheduling and RAN slicing policies of the gNBs (due to space limitations, details are omitted and can be found in [526]). The gNBs, which are deployed on the Colosseum platform [545], implement network slices associated to different traffic types, i.e., eMBB, MTC, and URLLC traffic. The agents aim at (i) maximizing the throughput for the eMBB slice, (ii) maximizing the number of transmitted packet for MTC, and (iii) reducing the service latency for URLLC. Moreover, we also trained two forecasting models to predict the UE traffic demand and the transmission buffer occupancy.

We consider the case where the DRL agents and the forecasters can run either at the near-real-time RIC as xApps, or at the DUs as dApps. In the former case, data for inference is received from the E2 interface, while in the latter data is locally available at the dApp. Figure 4.9 shows the impact that running intelligence at the dApps has on the overhead over the E2 interface as a function of the total number of deployed xApps and dApps. We consider three different configurations. In one

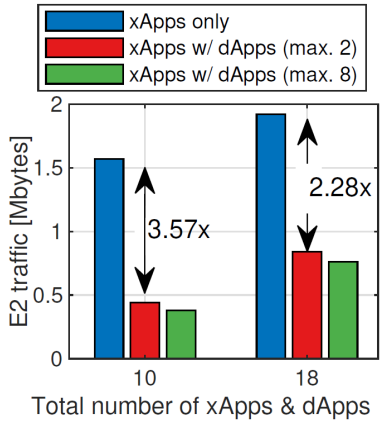


Figure 4.9: E2 traffic analysis.

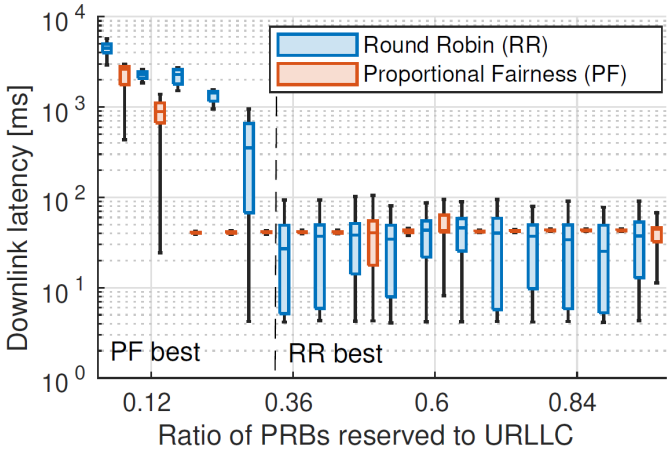


Figure 4.10: URLLC slice end-to-end latency for different RAN slicing and schedulers.

configuration, the intelligence can only run at the xApps; in the other two, the ML solutions can be executed either through xApps or dApps, with the DUs supporting at most 2 and 8 concurrent dApps. Figure 4.9 shows that dApps halve the traffic over the E2 interface, with a traffic reduction up to 3.57× with respect to the case with only xApps. Notice that two or more xApps (e.g., a DRL agent and a forecasting model) can share the same input data received over the E2 interface. For this reason, the traffic over E2 does not linearly grow with the number of xApps.

To further demonstrate the importance of being able to control RAN behavior in real time, we ran extensive data collection campaigns on Colosseum [526, 545], and demonstrated the impact of selecting different RAN slicing (i.e., the ratio of PRBs reserved exclusively to URLLC traffic) and scheduling strategies (i.e., RR and PF) on the application-layer latency of URLLC traffic. Although these campaigns do not involve any AI-based decision-making, the results reported in Figure 4.10 demonstrate the importance of joint slicing and scheduling control to support URLLC use cases (e.g., industrial automation, remote control, real-time streaming). For example, when less than 30% of resources are reserved for the URLLC traffic, selecting the PF scheduling algorithm ensures the lowest latency. On the contrary, RR works best when more PRBs are reserved to URLLC communications, with end-to-end latency values as low as 4 ms. These results show that achieving ultra-low latency still requires decisions made at the DUs directly (e.g., via dApps) to ensure that a tolerable end-to-end latency level is guaranteed despite rapidly changing channel and network conditions (e.g., buffer size, traffic load).

4.4 Conclusions

In this chapter we provided a path and a demonstration of the feasibility of integrating closed-control loops in cellular networks. We first reviewed key enablers, namely, virtualization, disaggregation, openness and reprogrammability of NextG cellular networks, using O-RAN as an exemplary technology. We then discussed which data-driven control loops can be implemented, their timescale, and whether the current O-RAN architecture supports them. We showed how large-scale experimental testbeds can be used to develop and validate data-driven algorithms by deploying a DRL-based O-RAN RIC on Colosseum. Our results show that using closed-control loops can provide a strong foundation toward the full realization of future generation, data-driven, autonomous, and self-optimizing cellular networks. Finally, we proposed to extend the O-RAN capabilities even further with the concept of *dApps*, distributed O-RAN applications for real-time control of the RAN that run at the DU/CU, and complement the control capabilities provided by xApps and rApps.

Chapter 5

Zero-touch Distributed Optimization of Softwarized Cellular Networks

Previous generations of cellular networks rely on proprietary and inflexible hardware and software solutions produced and maintained by few vendors. These closed architectures generally require manual configuration, preventing Telcos from being able to fully control resources such as spectrum, computing and transmission power to optimize network performance [360–362]. Remedies to this fundamental limitation have been piecemeal, often based on offline solutions for frequency assignment and network planning [363, 364]. Optimizing time-sensitive network functionalities also rests on heuristics often engraved in the hardware fabric [365, 366]. As of today, autonomous optimization of network parameters and swift and flexible control of real-time requirements of lower layer protocols are a territory that is largely uncharted.

Through SDN, Telcos are breaking the imposed vendor lock-in by leaving the static and monolithic RAN architecture in favor of using a dynamically programmable, i.e., *softwarized*, *Open RAN* for rapid and innovative network deployments [360, 361, 367–369, 523]. Although the benefits of such an open and multi-vendor approach have been showcased widely [220], how to fully embed softwarization in the future NextG infrastructure remains unsettled. This issue is further exacerbated by the increasing densification of cellular deployments and users, which makes non-automated control ineffective, if feasible at all. This is witnessed by recent works on cellular and wireless SDN clearly lamenting that the swift dynamics of these networks generate an overwhelming amount of signaling traffic, hardly bearable by traditional softwarized controllers [3, 35, 347, 370]. As a consequence, current hardware implementations and traditional centralized softwarized approaches do not allow timely optimization of network behavior and the increasingly needed superior network performance [371, 372].

Telcos are extremely sensitive to these issues. For example, the European Telecommunications Standards Institute (ETSI) formed the Zero-touch Network and Service Management group to define fully-automated—*zero-touch*—paradigms to provide flexibility to the highly decentralized technology of future wireless [373]. Similarly, the O-RAN Alliance and the Linux Foundation are promoting and building O-RAN and ONAP (see Sections 2.4.1 and 2.5.2). We observe that, although these approaches foresee network optimization as pivotal, they do not directly implement it. As of now, this is left to the wits of the Telco and to the best of our knowledge there is no *zero-touch* solution

yet to perform it dynamically.

In this chapter, which combines [521,543], we describe softwarized frameworks for the distributed zero-touch optimization of NextG cellular networks starting from a high-level intent expressed by the Telcos. We illustrate CellOS—based on traditional optimization techniques—in Section 5.1, and QCell—based on data-driven optimization—in Section 5.2. Then, we overview works closely related to these frameworks in Section 5.3 and we draw our conclusions in Section 5.4.

5.1 CellOS: Zero-touch Softwarized Open Cellular Networks

This section contributes to the efforts toward *automated softwarization* and *self-optimization* of future 5G networks by proposing CellOS (as in *Cellular Operating System*), the first *zero-touch* software framework for NextG cellular networks. Like an operating system interfacing hardware and software functions (whence the name), CellOS flexibly bridges SDN with cross-layer distributed optimization techniques for the cellular architecture. We push the SDN paradigm beyond the traditional separation of control and data planes, in that we also decouple control from optimization, adding further and unprecedented flexibility. Responding fully to ETSI requirements and industry interests, CellOS enables *zero-touch control and optimization* of low-level network functionalities by providing Telcos with an efficient, automated, modular, and flexible network control platform. Specifically, CellOS (i) allows Telcos to define centralized and high-level control objectives (e.g., "maximize network throughput") without requiring expertise in cross-layer optimization theory or knowledge of network specifics; (ii) provides a general *virtual network abstraction* that shields the Telco from the complexity of a sophisticated framework by abstracting network infrastructure and parameters, including those known at *run-time only* (e.g., user-to-base station associations and channel information); (iii) automatically converts high-level control directives into *distributed cross-layer control programs* to be executed at each network edge element, and (iv) enables *zero-touch optimization of distinct control objectives on different network slices coexisting on the same infrastructure* [374].

Figure 5.1 illustrates the overall structure of CellOS, exemplified for the 3GPP network architecture. The upper-left side of the figure depicts the high level APIs that the Telcos can use to define

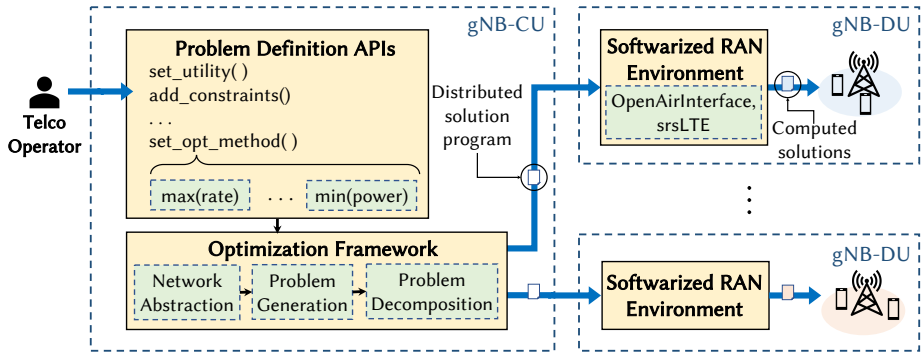


Figure 5.1: CellOS at a glance as instantiated for the 3GPP architecture.

the network control objectives. On the bottom we indicate the components of the framework for

automatic generation of the optimization problems and their decomposition into control programs. In a 3GPP scenario this unit corresponds to the CU, a logical node primarily concerned with control decisions at larger time-scales. On the right, we describe the softwarized RAN that will execute the generated programs. In the 3GPP context, this task would be carried out by the DU, a logical node that makes time-sensitive decisions involving the lower layers of the protocol stack, and that is interfaced with the CU.

We have prototyped CellOS on heterogeneous LTE-compliant testbeds. We have chosen two different implementations of the LTE stack, namely—OAI and srsRAN, discussed in Sections 2.2.1 and 2.2.2, respectively—to show that our framework is not tied to any specific RAN infrastructure. Our experiments consider a variety of scenarios with multiple base stations and users to show that CellOS optimizes the network performance by swiftly adapting to varying network configurations and settings. We also show the gains in performance that CellOS can bring to RAN implementations for cellular networks, such as OAI and srsRAN, as well as to MAC-layer scheduling algorithms commonly used in cellular networks, i.e., proportional fairness, greedy, and round-robin scheduling algorithms. Results of the comparative performance evaluation of CellOS and prevailing baseline solutions show that using our framework remarkably improves key performance metrics, such as throughput (up to 86%), energy efficiency (up to 84%) and user fairness (up to 29%). We also show that CellOS is transparent to the use of network slicing technologies [13, 524, 537], enabling Telcos to simultaneously optimize different network functions on distinct network slices. To the best our knowledge this is the first such demonstration, paving the way to the independent management of optimized network slices in 5G systems. Finally, and for the first time, we provide evidence of the potentials of *zero-touch optimization* in a *softwarized RAN* ecosystem by testing CellOS on the long-range open-source POWDER PAWR 5G platform [5, 6]. Our results show that CellOS seamlessly interacts with the LTE protocol stack by optimizing resource allocation strategies, successfully increasing the average throughput by 23%.

The remainder of the section is organized as follows. Section 5.1.1 presents CellOS in the 3GPP context, and a succinct overview of its main components. Details of its architecture are provided in Section 5.1.2. An example of CellOS operations is given in Section 5.1.3. An LTE-compliant prototype of CellOS is illustrated in Section 5.1.4. Finally, section 5.1.5 reports the performance evaluation of CellOS on various testbeds, including a laboratory setup, the Arena testbed (see Section 3.5), and the POWDER PAWR 5G platform [5, 6], using both the OAI and srsRAN RAN implementations with multiple base stations and users.

5.1.1 CellOS in a Nutshell

A bird's-eye view of the CellOS architecture is shown in Figure 5.1. In line with the 3GPP *functional split* [375], CellOS is partitioned in CU and DU modular units to decouple the definition of network control procedures (at the CU) from their execution (at the DU). CellOS main components are the interface to the Telco (providing the *Problem Definition APIs*) and the automatic *Optimization Framework* at the CU, and the *Softwarized RAN Environment* at the DU.

By means of a rich variety of APIs, the Telco sets the network control objective through high level, highly descriptive directives (e.g., "maximize throughput"), providing few key parameters (e.g., the number of base stations). That is all the TO needs to specify, as CellOS abstracts the underlying network structure, hiding lower-level details to the Telco and mapping network elements such as base

stations and UEs into virtual ones (*Network Abstraction* block of our Optimization Framework). As soon as the desired control objective is specified, CellOS converts it into a set of mathematical expressions that are used to define a centralized optimization problem, namely, the analytical representation of the optimization objective and of its constraints (*Problem Generation* block in Figure 5.1). The generated problem is then *automatically* decomposed into a set of distributed sub-problems, one for each of the edge elements (e.g., base stations). This is done by the *decomposition engine*, a core component of the *Problem Decomposition* block. Based on rigorous mathematical techniques, the centralized problem is partitioned both horizontally (decoupling variables belonging to different elements) and vertically (decoupling variables from different layers of each element's protocol stack). The obtained sub-problems are then automatically converted into executable programs that are individually dispatched to each element (*distributed solution programs*, in the *Softwarized RAN Environment*). Finally, each base station updates the distributed solution program with the real-time network parameters gathered from the RAN software stacks, and runs it through its local solver. It is worth mentioning that CellOS is independent of any specific RAN and can be interfaced with any other current or future 5G softwarized cellular stack. Finally, since CellOS edge elements have access to network real-time information by interfacing with the RAN software stacks (e.g., OAI, srsRAN), they update the received distributed control programs, adapting to the network time-varying dynamics, such as user arrival/departure, and mobility.

5.1.2 CellOS Architecture

In this section, we describe in details the components of the CellOS architecture, depicted in Figure 5.2.

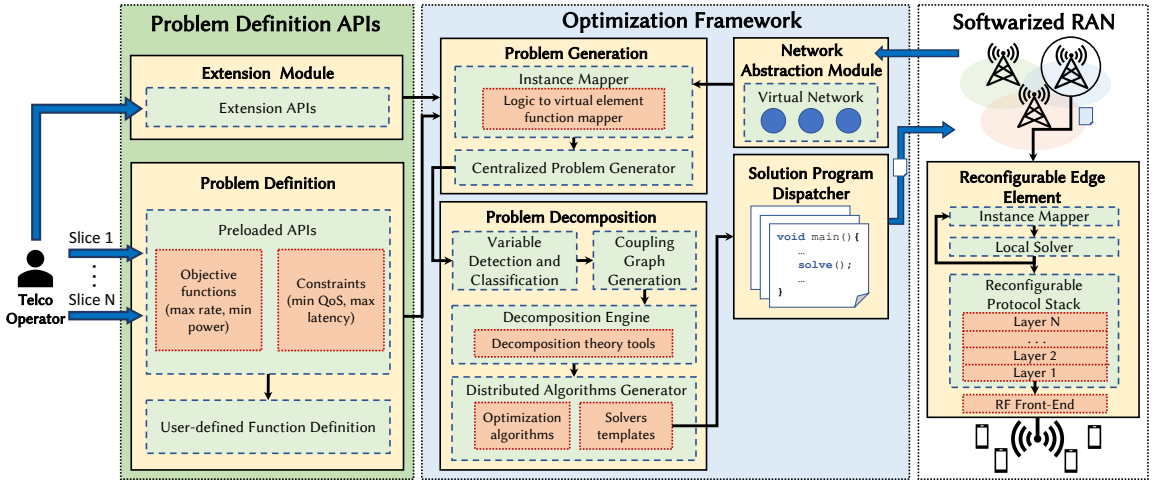


Figure 5.2: The CellOS architecture.

5.1.2.1 Problem Definition APIs

CellOS defines a rich set of APIs to specify general high-level information about the desired network configuration and optimization. These APIs include functions to add base stations and for setting per-

user requirements (e.g., minimum rate guarantees). The network control objective can be specified through a simple textual string, e.g., *max(rate)* to maximize the network rate, *min(power)* to minimize the overall power consumption.

```
1  from cellos import Network, Engine
2
3  # Network instantiation
4  nwk = Network(bs_num)
5  slices = nwk.get_slices()
6
7  # Optimization problem and optional constraints definition
8  nwk.set_utility('min(power)', slices[0])
9  nwk.add_constraints({'user_min_rate': [slices[0].get_users(), rate]})
10
11 # Optimization engine initialization
12 eng = Engine()
13 eng.set_opt_method('sub-gradient')
14 nwk.initialize_engine(eng)
```

Listing 5.1: CellOS API example.

An example of CellOS APIs and of the few lines of code needed to program a network objective are shown in Listing 5.1. In this example, the Telco instantiates a new network with a number `bs_num` of base stations (line 4), and gets the network slices instantiated in the network (line 5). An optimization problem aiming at minimizing the transmission power over a specific network slice (`slices[0]`) is then simply set in line 8, with constraints for guaranteeing a minimum rate defined in line 9. It is worth noting that existing slices of the network, active subscribers, and associations of the two, are known a priori by the Telco, and stored, for instance, in the cellular core network. We observe that very few lines of code are needed for the Telco to set the network goal, after which no further interaction is required. This is because CellOS, dovetailed with the ETSI *zero-touch* principles [373], hides all low-level network details (e.g., channel status, position of mobile users) from the Telco through the *network abstraction module* (Section 5.1.2.2), and also automatically defines and distributively solves the optimization problem corresponding to the set control objective.

While specifying the objective function in textual form is enough for CellOS to properly work, experienced Telcos can define tailor-made custom and more advanced objective functions, optimization techniques, and solvers through an *extension module*. This provides additional APIs for custom mathematical expressions and optimization constraints, and it also allows the Telco to select specific optimization techniques and solvers, as well as to achieve fine-grained control of network parameters and functionalities. These are then fed to the optimization framework in a way similar to the preloaded APIs. As of now, CellOS allows to specify functions expressed as linear combination of capacity, SINR, power, and energy efficiency terms, which already enables Telcos to formulate a large number of wireless networking optimization problems [376].

5.1.2.2 Optimization Framework

The heart of CellOS resides in its Optimization Framework, which: (i) Converts the high-level centralized code into an optimization problem; (ii) decomposes it into sub-problems; (iii) creates and

maintains an abstraction of the network, and (iv) dispatches the solution problems to the Softwarized RAN.

Problem Generation. In order to transform high-level specifications into an optimization problem, CellOS first pairs high-level abstraction directives (control objective and constraints) with available network elements (e.g., base stations and users). This is accomplished by the *instance mapper module* that maps physical network elements to their virtual representation, and converts the control objective defined using high-level CellOS APIs (Section 5.1.2.1) into machine-understandable code. For example, *max(sum(log(rate)))* is converted into $\max \sum_{u \in \mathcal{U}} \log(r_u)$, where $\mathcal{U}$ is the set of UEs and r_u their transmit rate. The generated utility is kept as general as possible by using symbolic placeholders in lieu of parameters whose value will only be known at run-time (e.g., UE-base station associations, channel coefficients, interfering signals, etc.). In so doing, our Optimization Framework is UE-agnostic. It is the base stations that, at run-time, replace the symbolic placeholders with their current value. Specifically, base stations interfaced with CellOS expose parameters and variables that can be tuned and optimized. Thus, placeholders of the generated problems always match physical network capabilities.

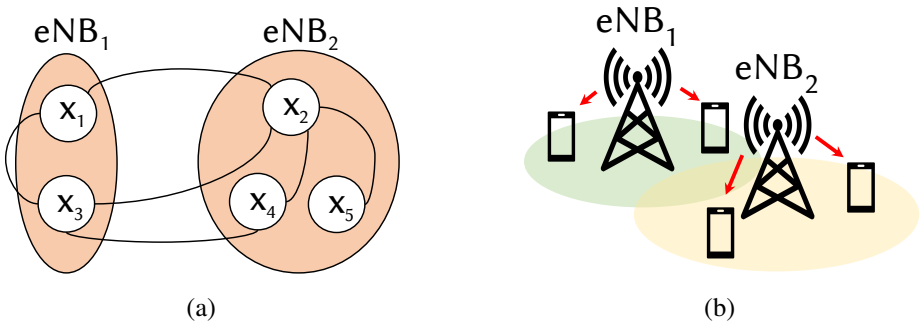
Problem Decomposition. This component of the Optimization Framework partitions the centralized problem into multiple sub-problems, one for each network element, to be solved distributively at each base station. In general, the centralized network control problem can be formalized as the following network utility maximization problem

$$\underset{\mathbf{x} \in \mathcal{X}}{\text{maximize}} \;\; f(\mathbf{x}) \tag{CEN}$$

$$\text{subject to} \;\; g_i(\mathbf{x}) \leq h_i(\mathbf{x}), \quad \forall i \in \mathcal{I} \tag{5.1}$$

where $\mathbf{x}$ represents the optimization variables (e.g., scheduling policies or transmission power levels), $\mathcal{X}$ is the strategy space (i.e., the set of all feasible strategy combinations), $f(\cdot)$ is the network-wide objective function (e.g., the overall capacity or the total energy efficiency of the network). Inequality (5.1) represents the set $\mathcal{I}$ of constraints (e.g., the transmission power must be bounded by some constant value; each PRB can be allocated to one UE only, etc.). The biggest challenge in solving (CEN) is that both objective function and constraints are, in general, coupled to different edge elements and to different layers of each element protocol stack. Because of this tight coupling, generating distributed sub-problems that can be locally solved by each base station becomes challenging. To address this challenge, CellOS first automatically identifies coupled variables and then applies rigorous decomposition to generate new sub-instances of (CEN) that are automatically assembled into uncoupled distributed programs to be executed at each base station. This is accomplished performing the following (Figure 5.2): *variable detection and classification*, *coupling graph generation*, decomposition (through the *decomposition engine*), and *distributed algorithms generation*.

- *Variable Detection and Classification.* CellOS starts by identifying the optimization variables of the network control problem. This is done by parsing the generated objective function expression looking for symbolic placeholders introduced therein. For instance, in (CEN) CellOS detects $\mathbf{x}$ to be the set of optimization variables of the problem. Then, it determines which layer of the protocol stack houses which variable, e.g., power belongs to the PHY layer,



(a) (b)

Figure 5.3: (a) Coupling graph for $f(\mathbf{x}) = x_2(x_4 + x_5) + x_3(x_4 + x_1/x_2)$; (b) Network scenario considered in Section 5.1.3.

scheduling to the MAC layer, and so on. CellOS then identifies to which base station each variable belongs to. As a result, each variable is assigned to a specific base station and to one of its protocol stack layers.

- *Coupling Graph Generation.* After detecting and classifying problem variables, CellOS organizes their coupling in a graph $G = (V, E)$, where V is the set of variables of the network control problem, which are joined by an edge in E only if they are coupled. Similarly to what done in the previous step, coupling among variables is detected through a symbolic parser. As an example, a coupling graph for $f(\mathbf{x}) = x_2(x_4 + x_5) + x_3(x_4 + \frac{x_1}{x_2})$ is shown in Figure 5.3a. Variables $\{x_i\}_{i=1,3}$ and $\{x_j\}_{j=2,4,5}$ belong to eNB_1 and eNB_2 (Figure 5.3b), respectively.

  Figure 5.3a shows that coupling is not limited to variables of a single eNB, but it might also involve those controlled by other eNBs.

- *Decomposition Engine.* Variable detection/classification and coupling graphs are preliminary to automated problem decomposition, which we perform by using well-established techniques, including duality theory [377] and decomposition via Partial Linearization [371] (additional ones can be implemented through the *extension module* of Figure 5.2). Decomposability is achieved introducing auxiliary variables (e.g., Lagrangian multipliers, penalization terms, and aggregate interference functions) that remove coupling across optimization variables and generate objective functions and constraints with separable terms in the sense of [377]. Unfortunately, coupling in cellular networks involves heterogeneous network elements and different layers of the protocol stack, resulting in optimization problems whose utility or constraints are rarely separable. For this reason, it is classified into *horizontal coupling* and *vertical coupling*. The former reflects dependencies among different network elements (e.g., among interfering base stations and their subscribers). The latter, instead, concerns *cross-layer* dependencies among different layers of the protocol stack of the same element (e.g., MAC policies affect transmission power and modulation strategies at the PHY layer). Coupling makes centralized control of cellular networks extremely challenging as (i) the number of variables of the problem grows exponentially with the number of network elements, resulting in high computational and time complexity; (ii) the TO needs to be fully aware of the underlying network topology, the traffic demand, and the Channel State Information (CSI)

for each individual UE and base station, and (iii) centralized approaches require real-time information exchange between each network element and the centralized controller, imposing high signaling overhead and latency. It is worth to point out that such network real-time information is not known at CellOS controller, but only at the edge elements. Due to the fast changing network dynamics, though, the time required to signal local information to the controller, compute a centralized solution, and adopt it at the edge elements might exceed the coherence time of the found solution. *Such solutions, may refer to an old network state and be obsolete, thus resulting in poor performance.* This makes distributed solutions highly desirable, if not mandatory. Even though distributed algorithms might not always guarantee globally optimal solutions, they usually manage to compute locally optimal ones with significantly lower computational complexity, while ensuring run-time performance [371, 372].

We point out that this work does not focus on proposing new decomposition theories. *Our aim, instead, is to automatically generate distributed optimization programs based on a high-level objective, irrespective of the decomposition method used.*

- *Distributed Algorithms Generator.* The final step to achieve distributed control of the cellular network is to generate *distributed solution programs* which can be executed and solved by each base station via standard optimization solvers. This task is performed by the *distributed algorithms generator* unit of CellOS Optimization Framework (Figure 5.2). As mentioned, the Optimization Framework is not cognizant of the value of parameters that are known at run-time only. Accordingly, the *distributed solution programs* contain symbols in place of these parameters. Each base station will then replace these symbols with their actual value at run-time, and associate optimization variables to the served UEs. The *instance mapper* module has been designed to perform this task (Figure 5.2). This is one of the most important features of CellOS as it makes the solution program generation process (i) fully automated; (ii) independent of network configuration, and (iii) self-adapting to compute parameters at run-time based on current network conditions.

Dispatcher and Abstraction Module. The last two components of the Optimization Framework are the *solution program dispatcher* and the *network abstraction module*. The *dispatcher* utilizes sockets to transfer the generated distributed solution programs to each network base station, which will execute and solve them to achieve the desired network objective.

The *network abstraction module* creates a high-level representation of the network infrastructure, hiding low-level, hardware/software details from the Telco. This abstraction allows the *problem generation* to automatically convert directives and constraints given through the CellOS APIs into mathematical expressions and utility functions (Section 5.1.2.2).

5.1.2.3 Softwarized RAN

The third main component of the CellOS architecture (Figure 5.2) is in charge of running the distributed solution programs at each network element so as to reach the global network objective requested by the TO. Once the dispatcher has delivered the programs, the *instance mapper* component of the Reconfigurable Edge Element (REE) replaces the symbolic placeholders in the program with their corresponding run-time values. This component is capable of dynamically adapting solution programs to current network conditions, such as arrival/departure of UEs, handovers, and CSI. At

the end of this mapping procedure each program is executed by the *local solver* and a solution is computed. As mentioned above, CellOS uses decoupling terms (e.g., Lagrangian multipliers) to allow individual base stations to coordinate with each other. Relevant parameters are iteratively updated and exchanged among the coupled REEs through already available inter-base station interfaces (e.g., X2/Xn interfaces of cellular networks).

Since all the decisions are made locally at the base stations, at most $|\mathcal{U}|\,(|\mathcal{N}|+1)$ variables need to be exchanged at each iteration, where $\mathcal{U}$ is the set of users, $\mathcal{N}$ are the available transmission channels, and $|\cdot|$ denotes the cardinality operator. As we will demonstrate in Section 5.1.5.3 through experimental results, this overhead is negligible if compared to that of centralized approaches, which need to gather local information at the central controller. Because of this very limited signaling overhead, our framework effectively self-adapts to the network fast changing behavior. Upon computing optimal solutions for each local network control problem (e.g., transmission and scheduling policies), these are used by each REE through the Reconfigurable Protocol Stack (RPS), which controls MAC and PHY layers, among others.

5.1.3 CellOS in Action: An Example

We consider the scenario depicted in Figure 5.3b, where two interfering eNBs in the set $\mathcal{B}$ share two channels and serve two UEs each. Here, $\mathcal{U}_b$ is the set of users u served by eNB $b \in \mathcal{B}$. We consider a downlink cross-layer optimization problem where each eNB has a transmission power budget $P^{\max}$, and that the UEs request a minimum capacity $C^{\min}$. The optimization variables of this problem concern MAC and PHY layers, namely, user scheduling and transmission power allocation. In this example, we assume that the Telco uses CellOS to maximize the network capacity. The Telco first instantiates a network with two base stations (`nwk = Network(2)`). Then the following network control objective is set on the slice controlled by the Telco: `nwk.set_utility('max(capacity)', slices[0])`.

On the other hand, CellOS needs to perform a more complex set of operations to reach the objective specified so succinctly by the TO. Let $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2)$ represent the network scheduling profile, where $\mathbf{y}_b = (y_{b,1,n}, y_{b,2,n})_{n=1,2}$ is the scheduling profile for eNB $b \in \{1,2\}$. Let $y_{b,u,n}$, instead, represent the scheduling variable such that $y_{b,u,n} = 1$ if user u is scheduled for downlink transmission on channel $n \in \mathcal{N} = \{1,2\}$ and $y_{b,u,n} = 0$, otherwise. Similarly, $\mathbf{p} = (\mathbf{p}_1, \mathbf{p}_2)$ represents the network power allocation profile, where $\mathbf{p}_b = (p_{b,1,n}, p_{b,2,n})_{n=1,2}$ is the power allocation profile for eNB b, and $p_{b,u,n}$ represents the downlink transmission power from b to user u on channel n. Let $C_{b,u,n}(\mathbf{y}, \mathbf{p})$ be the capacity for UE u served by eNB b on channel n, expressed as

$$C_{b,u,n}(\mathbf{y}, \mathbf{p}) = B \log_2 \left(1 + \frac{g_{b,u,n} y_{b,u,n} p_{b,u,n}}{N + g_{b',u,n} \sum\limits_{u' \in \mathcal{U}_{b'}} p_{b',u',n} y_{b',u',n}} \right), \tag{5.2}$$

where B is the employed bandwidth, N is the background noise power, and $g_{b,u,n}$ is the channel gain coefficient computed by u and sent to b, as part of standard cellular networks signaling procedures between user and base station (e.g., LTE PUCCH).

The centralized network control problem can be expressed as the following Capacity Maximization Problem (CMP)

$$\underset{\mathbf{y},\mathbf{p}\in\mathcal{X}}{\text{maximize}} \quad \sum_{b\in\mathcal{B}}\sum_{u\in\mathcal{U}_b}\sum_{n=1}^{2} C_{b,u,n}(\mathbf{y},\mathbf{p}), \tag{CMP}$$

$$\text{subject to} \quad \sum_{n=1}^{2} C_{b,u,n}(\mathbf{y},\mathbf{p}) \geq C^{\min}, \quad \forall b\in\mathcal{B}, u\in\mathcal{U}_b \tag{5.3}$$

$$\sum_{u\in\mathcal{U}_b}\sum_{n=1}^{2} p_{b,u,n} \leq P^{\max}, \quad \forall b\in\mathcal{B} \tag{5.4}$$

$$\sum_{n=1}^{2} y_{b,u,n} \leq 1, \quad \forall b\in\mathcal{B}, \forall u\in\mathcal{U}_b \tag{5.5}$$

where (5.3) represents the users' minimum capacity constraint, (5.4) enforces eNBs' power budget, and (5.5) guarantees that each eNB allocates each channel to a single UE only.

The main challenges in decomposing (CMP) are: (i) it is a Mixed Integer Non-Linear Programming problem, which is NP-hard in general [378], and (ii) both (5.2) and (5.3) are coupled among different eNBs.

CellOS recognizes $\mathbf{y}$ and $\mathbf{p}$ to be the problem optimization variables and associates them to the MAC and PHY layers, respectively. Now, the *problem decomposition* module understands which variables belong to which eNB and creates a coupling graph similar to that in Figure 5.3a. This is, then, used to detect the aggregate interference term in the capacity expression (5.2). Accordingly, it defines the following auxiliary function

$$h_{b,u,n}(\mathbf{y}_{-b},\mathbf{p}_{-b}) = \sum_{b'\in\mathcal{B}\setminus\{b\}} g_{b',u,n} \sum_{u'\in\mathcal{U}_{b'}} p_{b',u'n} y_{b',u',n} \tag{5.6}$$

where $\mathbf{y}_{-b} = \mathbf{y}\setminus\{\mathbf{y}_b\}$ and $\mathbf{p}_{-b} = \mathbf{p}\setminus\{\mathbf{p}_b\}$ are the scheduling and power allocation variables of the eNBs belonging to $\mathcal{B}\setminus\{b\}$. At this point, new auxiliary variables are introduced to rewrite (CEN) as

$$\underset{\mathbf{y},\mathbf{p},\mathbf{i}}{\text{maximize}} \quad \sum_{b\in\mathcal{B}}\sum_{u\in\mathcal{U}_b}\sum_{n=1}^{2} C_{b,u,n}(\mathbf{y}_b,\mathbf{p}_b,\mathbf{i}_b), \tag{DCMP}$$

$$\text{subject to} \quad \sum_{n=1}^{2} C_{b,u,n}(\mathbf{y}_b,\mathbf{p}_b,\mathbf{i}_b) \geq C^{\min}, \quad \forall b\in\mathcal{B}, u\in\mathcal{U}_b \tag{5.7}$$

$$i_{b,u,n} \geq h_{b,u,n}(\mathbf{y}_{-b},\mathbf{p}_{-b}), \forall b\in\mathcal{B}, u, n=1,2 \tag{5.8}$$

Constraints (5.4), (5.5)

CellOS can now use duality optimization tools to generate the following Lagrangian dual function

$$
\begin{aligned}
L(\boldsymbol{\lambda}, \boldsymbol{\mu}, \mathbf{i}, \mathbf{y}, \mathbf{p}) &= \sum_{b \in \mathcal{B}} \sum_{u \in \mathcal{U}_b} \sum_{n=1}^{2} C_{b,u,n}(\mathbf{y}_b, \mathbf{p}_b, \mathbf{i}_b) \\
&- \sum_{b \in \mathcal{B}} \sum_{u \in \mathcal{U}_b} \lambda_{b,u} \left(C^{\min} - \sum_{n=1}^{2} C_{b,u,n}(\mathbf{y}_b, \mathbf{p}_b, \mathbf{i}_b) \right) \\
&- \sum_{b \in \mathcal{B}} \sum_{u \in \mathcal{U}_b} \sum_{n=1}^{2} \mu_{b,u,n} \left(h_{b,u,n}(\mathbf{y}_{-b}, \mathbf{p}_{-b}) - i_{b,u,n} \right), \qquad (5.9)
\end{aligned}
$$

where $\boldsymbol{\lambda} = (\lambda_{b,u,n})$ and $\boldsymbol{\mu} = (\mu_{b,u,n})$ are the non-negative Lagrangian multipliers used in constrained optimization [377].

We observe that problems (CMP) and (DCMP), and the Lagrangian dual function (5.9) all aim at solving the centralized control problem (CEN). However, the advantage of using (5.9) is that function $L(\boldsymbol{\lambda}, \boldsymbol{\mu}, \mathbf{i}, \mathbf{y}, \mathbf{p})$ is written with separable variables, meaning that it can be split into $|\mathcal{B}|$ sub-problems locally solvable by each eNB.

Finally, CellOS dispatches the generated distributed solution programs to the eNBs that populate them with network run-time information (e.g., users' channel coefficients), and compute optimized solutions through their *local solver*.

It is worth noting that the procedures detailed in Sections 5.1.2.1 and 5.1.2.2 need to be executed only once per control problem specified by the Telco and that they take very little time to be performed, e.g., 0.03 s for the example of this section (more details on the scalability of CellOS automatic procedures will be given in Section 5.1.5.3).

5.1.4 CellOS Prototype

In this section, we discuss the prototypes of CellOS, which have been built based on the OAI and srsRAN open-source RAN implementations. The OAI-based prototype is illustrated in Figure 5.4.

The *CellOS Controller* performs the functionalities of the Problem Definition APIs and of the Optimization Framework. Particularly, it creates and maintains the network abstraction, generates the optimization problem based on the directives from the Telco, and performs the problem decomposition. In our experiments the decomposition process is obtained through Lagrangian duality theory [377] and decomposition via Partial Linearization [371].

Multiple *eNB Controllers*, one for each base station, are connected to the CellOS Controller through a Gigabit Ethernet connection. These controllers use interior-point and sub-gradient algorithms [377] to solve the received distributed programs, and set the parameters to be used with the RF front-ends they are connected to. Each of these controllers drives an Ettus Research USRP B210, which serves UEs over LTE frequencies. As UEs we used a set of heterogeneous COTS cellular phones (Samsung Galaxy S5, S6 and S7, and Apple iPhone 6s).

In this prototype, CellOS interfaces with the LTE protocol stack implementation offered by *OpenAirInterface*, i.e., an open-source software-based experimental platform for LTE implementations [24]. OAI features LTE RAN applications along with Evolved Packet Core components. As OAI does not directly allow per-user power control, or optimized PRB allocation—key essential requirements of many network control objectives—we have extended its functionalities by significantly

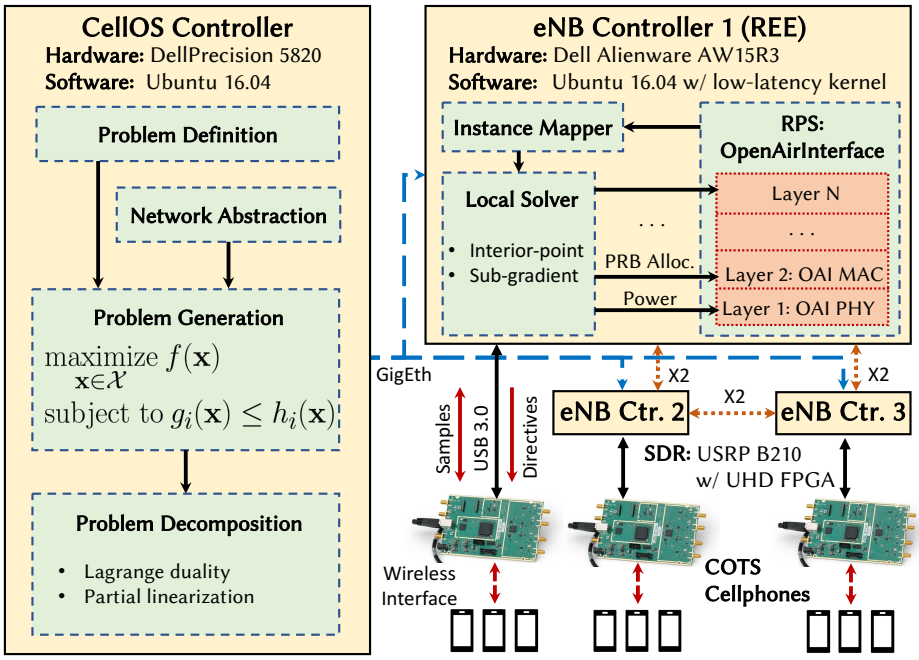


Figure 5.4: OAI-based CellOS prototype.

modifying its core implementation. Specifically, power control is obtained by amplitude-modulating the downlink data signal intended for a specific UE. PRB allocation, instead, is based on an optimized waterfilling-like fair scheduling algorithm [379], which has low-complexity, thus complying with LTE strict timing requirements. Because of the PRB short time duration it is of utmost importance to compute the PRB allocation very quickly to guarantee compliance with LTE and promptly serve the UEs. According to our scheme, PRBs are allocated only to those UEs whose downlink transmission buffer is not empty.

A similar approach has been followed for the *srsRAN*-based prototype—implemented through the SCOPE framework of Section 3.2—which leverages USRPs X310 in place of USRPs B210. This time, each eNB controller connects to the SDR through a $10\ \mathrm{Gbit/s}$ PCI Express network card.

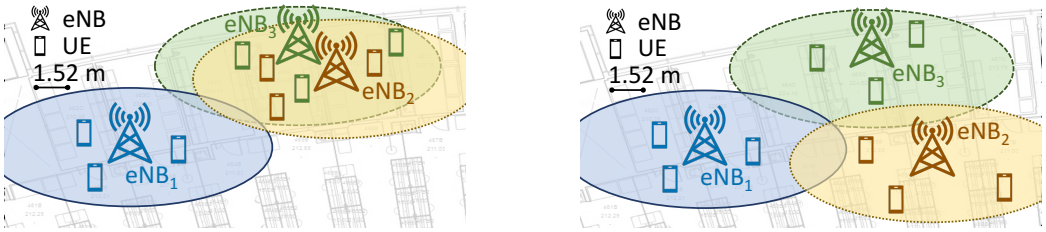
5.1.5 Experimental Evaluation

The effectiveness of CellOS in automatically creating distributed optimization programs from high-level directives, and in managing the network infrastructure to reach different control objectives, is demonstrated via experimentation on various LTE-compliant testbeds. We describe our testbed in Section 5.1.5.1, we introduce the investigated performance metrics in Section 5.1.5.2, and present our experimental results in Section 5.1.5.3.

5.1.5.1 Network Scenarios and Testbed Settings

To demonstrate its platform-independence, we test CellOS over different software and hardware platforms, using OAI and srsRAN, as well as heterogeneous software-defined radios and testbeds.

The OAI-based prototype of Section 5.1.4 has been used in a testbed composed of 3 eNBs and up to 9 UEs. Each eNB uses a 10 MHz channel bandwidth corresponding to 50 PRBs. For this prototype we consider the two indoor scenarios depicted in Figure 5.5: (i) A high interference scenario, where two eNBs are in line-of-sight conditions and have largely overlapping coverage areas (Figure 5.5a), and (ii) a low interference scenario where eNBs are in non-line-of-sight conditions and their coverage areas only partially overlap with each other (Figure 5.5b).



(a) High interference (b) Low interference

Figure 5.5: The CellOS laboratory setup.

The high interference scenario represents those crowded environments (e.g., university campuses, concert halls or convention centers) where several femtocells are deployed in a crowded region to balance the traffic load of a macrocell farther away. In this case, while the interference among macro- and femtocells is small, femtocells with overlapping coverage areas are subject to significant inter-cell interference. In the low interference scenario, instead, eNBs are located far away from each other and, thus, are less subject to inter-cell interference and the subsequent performance degradation.

The srsRAN-based prototype is evaluated on a low-interference setup on the Arena testbed (see Section 3.5). We instantiated 3 LTE eNBs on USRPs X310 whose antennas are connected to the ceiling of a 208.1 m^2 office space. A set of Dell EMC PowerEdge R340 servers are used to drive the USRPs through 10 Gigabit Ethernet connections. This set of experiments shows that CellOS can *simultaneously obtain different control objectives on multiple network slices.* This represents the scenario in which multiple MVNOs share the same edge elements, or that of a single Telco wishing to set diverse control problems on each network slice. To demonstrate the benefits of automatic optimization of the *Open RAN*, we finally instantiate CellOS on the long-range open-source 5G POWDER platform [6], which is the combination of the Platform for Open Wireless Data-driven Experimental Research (POWDER) and Reconfigurable Eco-system for Next-generation End-to-end Wireless (RENEW), and part of the Platforms for Advanced Wireless Research (PAWR) [5].

We assess CellOS performance by letting UEs download a file stored on our local server for 60 s. *It is worth mentioning that it only took CellOS* 1.43 s *and 8 lines of code (see Listing 5.1) to automatically generate the evaluated distributed control programs* (more details on the scalability of these operations will be given in Section 5.1.5.3)

5.1.5.2 Performance Metrics

CellOS has been evaluated against the following metrics.

- *Sum throughput of the network*, defined as

$$S = \sum_{b \in \mathcal{B}} \sum_{u \in \mathcal{U}_b} S_{b,u}, \quad \forall b \in \mathcal{B}, u \in \mathcal{U}_b \tag{5.10}$$

where $\mathcal{B}$ and $\mathcal{U}_b$ are the sets of the eNBs b and of UEs u they are serving, and $S_{b,u}$ is the throughput offered to $u \in \mathcal{U}_b$ by b.

- *Normalized transmission power* of the base stations to the UEs. To analyze the impact of power control policies on the transmission power of eNBs, we show the transmission power of the base stations normalized to their maximum transmission power. Let $P_b^{\max}$ and $P_b^{\min}$ be the maximum and minimum power levels of base station b, the normalized transmission power is defined as

$$P_{b,u}^N = \frac{P_{b,u} - P_b^{\min}}{P_b^{\max} - P_b^{\min}}, \quad \forall b \in \mathcal{B}, u \in \mathcal{U}_b \tag{5.11}$$

  where $P_{b,u} \in \{P_b^{\min}, P_b^{\max}\}$ is the power used by eNB $b \in \mathcal{B}$ to transmit to its user $u \in \mathcal{U}_b$.

- *Global energy efficiency*, defined as the amount of information per unit of energy the eNBs transmit to their subscribers:

$$EE = \frac{\sum_{b \in \mathcal{B}} \sum_{u \in \mathcal{U}_b} S_{b,u}}{\sum_{b \in \mathcal{B}} \sum_{u \in \mathcal{U}_b} P_{b,u}}, \quad \forall b \in \mathcal{B}, u \in \mathcal{U}_b \tag{5.12}$$

  where $P_{b,u}$ is the power used by eNB b to transmit to its user u.

- *System fairness*, measured through Jain's equation [380]. Given users $u \in \mathcal{U} = \bigcup_{b \in \mathcal{B}} \mathcal{U}_b$, Jain's fairness index $\mathcal{J}$ is defined as

$$\mathcal{J} = \frac{(\sum_{b \in \mathcal{B}} \sum_{u \in \mathcal{U}_b} S_{b,u})^2}{|\mathcal{U}| \sum_{b \in \mathcal{B}} \sum_{u \in \mathcal{U}_b} S_{b,u}^2}, \quad \forall b \in \mathcal{B}, u \in \mathcal{U}_b. \tag{5.13}$$

5.1.5.3 Experimental Results

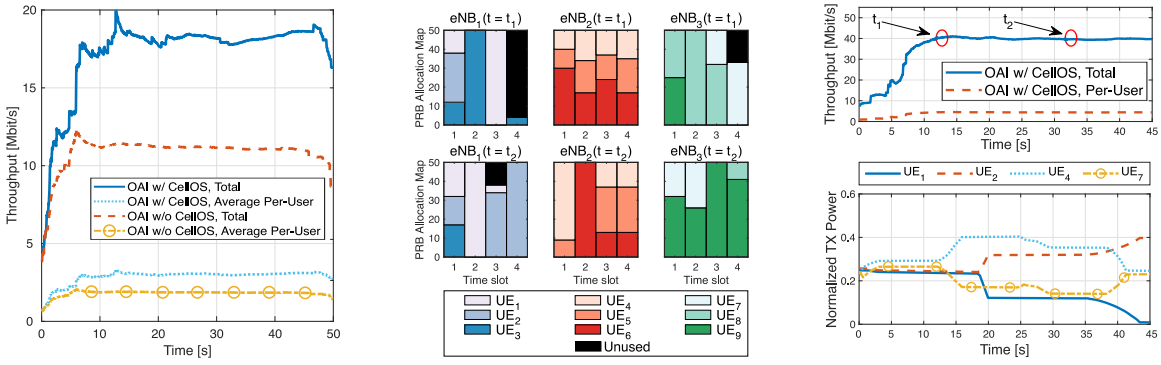
Table 5.1: Summary of experimental setup.

Figure	Optimization Problem	Scenario	RAN Software	Testbed
Figure 5.6	max(rate)	High Interference	OAI	Laboratory Setup
Figure 5.7	min(power)	High Interference	OAI	Laboratory Setup
Figure 5.8	max(sum(log(rate)))	High Interference	OAI	Laboratory Setup
Figure 5.9	max(sum(log(rate)))	Low Interference	OAI	Laboratory Setup
Figure 5.10	max(rate)	Low Interference	OAI	Laboratory Setup
Figure 5.11a	max(rate)	Slicing	srsRAN	Arena
Figure 5.11b	min(power)	Slicing	srsRAN	Arena
Figure 5.12	max(rate), min(power), max(sum(log(rate)))	Controller Time	N/A	Arena
Figure 5.13	max(rate), min(power), max(sum(log(rate)))	Local Solver Time	N/A	Arena
Figure 5.14	max(rate), min(power), max(sum(log(rate)))	Signaling Overhead	N/A	Arena
Figure 5.15a	max(rate)	Long-range	srsRAN	POWDER

CellOS has been evaluated against the metrics of Section 5.1.5.2 in a variety of network configurations (i.e., high and low interference, with and without network slicing), and on different testbeds, including a laboratory setup, the Arena testbed (see Section 3.5), and the POWDER PAWR 5G platform [5, 6].

To fully appreciate the effects of the automatic optimization procedures introduced by CellOS, we consider a cellular network implemented through OAI and srsRAN and we compare the achieved network performance with and without CellOS. Moreover, we also compare the performance achieved by state-of-the-art scheduling algorithm commonly used in commercial cellular networks, *i.e., proportional fairness, greedy, and round-robin,* to that achieved by CellOS-managed networks. A summary of our experimental setup is shown in Table 5.1.

High Interference Scenario. Figure 5.6 presents results obtained when optimizing throughput (network control objective of *max(rate)*) in the high interference scenario in Figure 5.5a. We start by evaluating the throughput gains brought to OAI by CellOS zero-touch approach. Average total and per-user throughput are shown in Figure 5.6a. We observe that CellOS brings significant benefits to the network performance, with improvements as high as 75% (63% on average). This is because of the interplay between the optimized per-user power control and scheduling determined by CellOS and executed locally by the Softwarized RAN. Indeed, CellOS automatic optimization procedures allow the eNBs to serve UEs with an optimized resource allocation and power-controlled signals, which significantly reduces the inter-cell interference while guaranteeing a minimum rate to UEs. To provide further insights on the resource allocation procedures automatically executed by each eNB, we investigated the network throughput, and power and PRBs allocated to the users during an experiment run of the *max(rate)* solution program (Figures 5.6b and 5.6c, respectively). For clarity, only the power for four users is shown. As time progresses, the throughput (both total and per-user) plateaus out to a stable value, which is a consequence of local optimality of the solution program that successfully limits interference. Power is changed for the individual user in time, also responding to optimization requirements and reflecting current network conditions. Figure 5.6b depicts the PRBs allocated to UEs at time instants t_1 and t_2 of Figure 5.6c. We observe that the eNBs adapt the PRB allocation in *real-time* to satisfy user requests while achieving the set network objective. In fact, time



(a) OAI w/ and w/o CellOS (b) PRBs at time t_1 and t_2 (c) Throughput and power

Figure 5.6: Throughput maximization in the high interference scenario on the OAI-based prototype.

slots with unassigned PRBs may even occur, without compromising the eNB ability of satisfying its subscribers requirements.

To show that different network control objectives produce different results, we investigate throughput and power determined by CellOS for power minimization (control objective of *min(power)*), while guaranteeing a minimum per-user data rate of $1\,\mathrm{Mbit/s}$ (Figure 5.7). As expected, the achieved

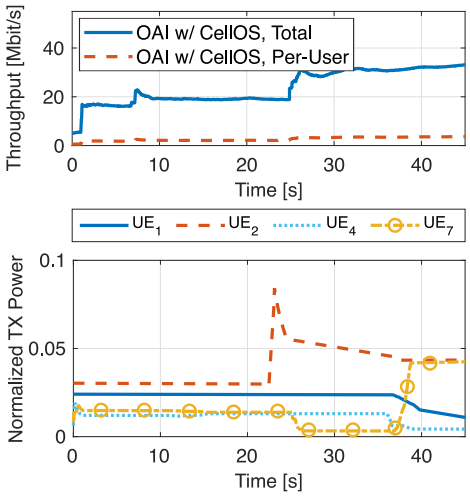
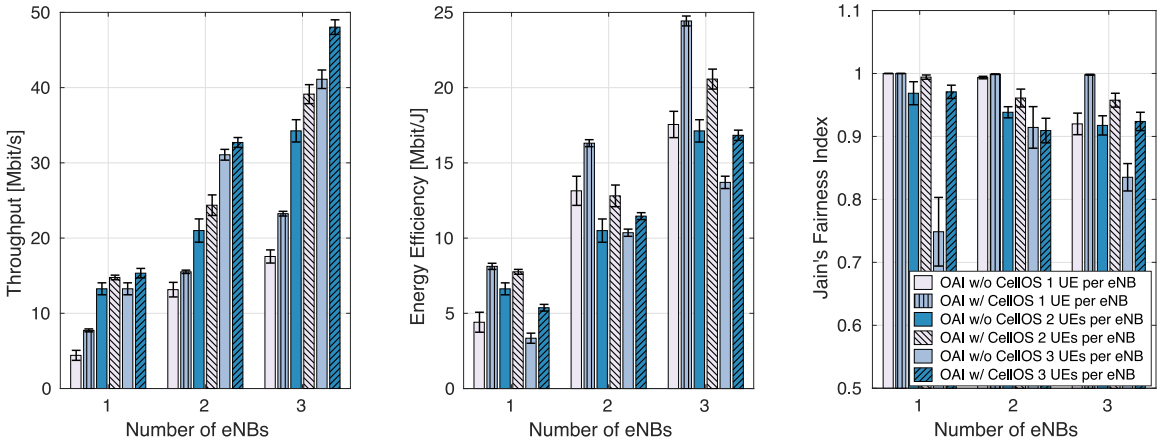


Figure 5.7: Power minimization in the high interference scenario on the OAI-based prototype.

throughput is lower than that of the *max(rate)* control program (Figure 5.6c). This is due to the normalized transmission power of the eNBs being remarkably lower than that in Figure 5.6c (up to one order of magnitude). We notice, though that UEs achieve an average throughput of $2.63\,\mathrm{Mbit/s}$, which satisfies the constraint on their minimum rate.

The next set of experiments concerns the performance of OAI with and without CellOS in scenarios with varying number of eNBs and UEs. The network control objective requires to maximize throughput while explicitly accounting for fairness, namely, is set to *max(sum(log(rate)))*. Scenarios with one base station consider only eNB_3, while Scenarios with two base stations concern eNB_2 and eNB_3, i.e., the base stations with overlapping cells (see Figure 5.5a). Results concerning sum throughput, energy efficiency and fairness are shown in Figure 5.8.

The throughput comparison is shown in Figure 5.8a, where we can see that OAI with CellOS always outperforms OAI without CellOS. In Figure 5.8b, we evaluate energy efficiency, pivotal in large-scale networks [381]. As expected, since our framework achieves a higher throughput with a lower power expenditure, the network is more energy efficient when managed by CellOS. System fairness is shown in Figure 5.8c. We notice that, in general, CellOS improves user fairness, with increases up to 29%. Improvements are more evident in scenarios with higher number of eNBs and UEs, as optimization techniques are more effective in those more dense scenarios with higher interference. Specifically, since in these scenarios suboptimal algorithm solutions generate inefficient resource allocation policies, optimal ones are required the most. Indeed, CellOS optimized resource allocation, and its ability to fine-tune the power directed to the served UEs allows the base stations to contain the interference directed to other eNBs, thus increasing the network performance.



(a) Throughput (b) Energy efficiency (c) Fairness

Figure 5.8: Sum-log-rate maximization in the high interference scenario on the OAI-based prototype w/ and w/o CellOS.

Low Interference Scenario. These experiments concern 3 eNBs and 9 UEs in low interference conditions (Figure 5.5b). Results on throughput and on the allocated normalized power are shown in Figure 5.9. In this scenario CellOS is required to optimize the network control objective *max(sum(log(rate)))*. As expected, performance is better than in the high interference scenario

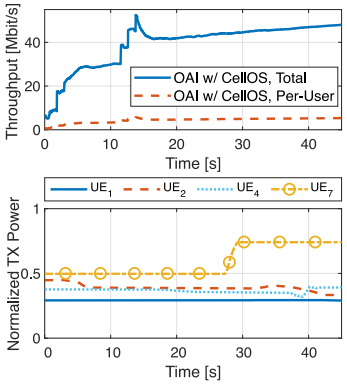


Figure 5.9: Sum-log-rate maximization in the low interference scenario on the OAI-based prototype w/ CellOS.

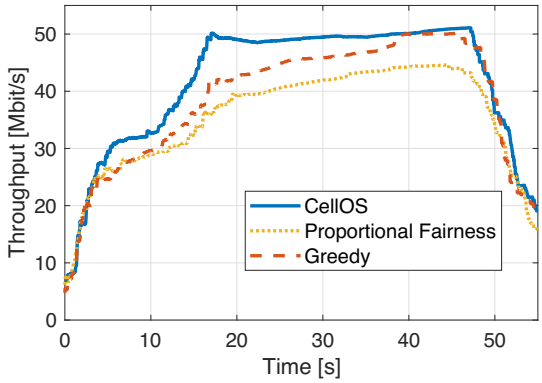


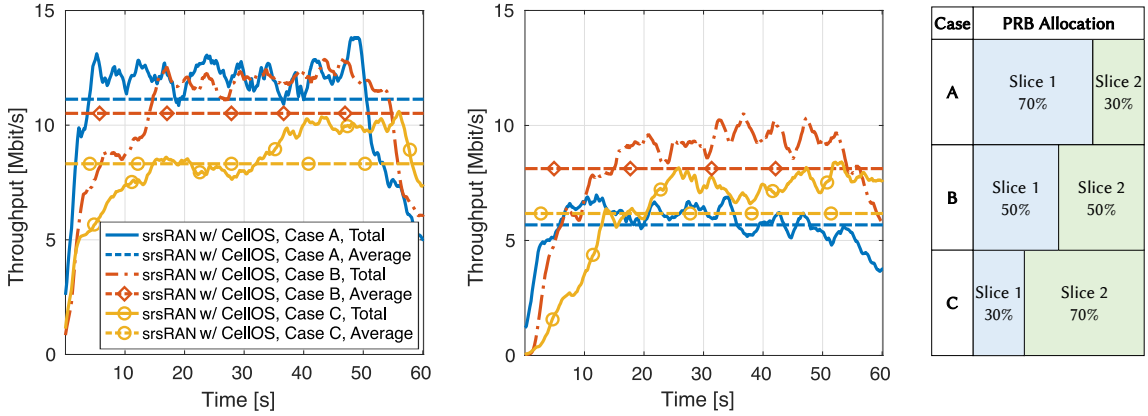
Figure 5.10: Rate maximization in the low interference scenario: OAI w/ CellOS vs. OAI w/ *proportional fairness* [1] and OAI w/ *greedy* [2] scheduling policies.

because of the lower interference level, that allows the eNBs to use higher power without disrupting each other transmissions. In Figure 5.10, we compare CellOS rate maximization with two well-known state-of-the-art scheduling algorithms: the *proportional fairness* algorithm, that is the *de facto standard* in cellular networks [1, 366], and the *greedy* algorithm [2]. We notice that CellOS outperforms the proportional fairness algorithm because of this overarching optimization approach to network management. The greedy approach, instead, obtains throughput levels similar to those of CellOS, albeit with a significant delay. Indeed, because of its optimized MAC-layer procedures,

which allow the network base stations to mindfully allocate resources to the served UEs, CellOS achieves said throughput level after only few seconds from the system start and maintains it until the UEs finish downloading data.

Network Slicing. This set of experiments concerns 3 eNBs instantiated on the USRPs X310 of the Arena testbed (see Section 3.5). The eNBs serve 9 COTS UEs.

We target a scenario in which multiple MVNOs lease infrastructure resources from an Infrastructure Provider (IP). The IP, which owns the physical equipment (e.g., the base stations), allocates slices of the network to MVNOs following, for instance, the approaches described in Appendices A and B. Since MVNOs act independently from one another, with different subscribers and requirements (e.g., quality of service), they may need to optimize different control programs on their slice of the network. Considering this, and cognizant of current 5G cellular networks trends, we designed CellOS to handle different network slicing configurations.



(a) (b) (c)

Figure 5.11: Optimization of different control programs on different slices on the srsRAN-based prototype instantiated on the Arena testbed: (a) Throughput of Slice 1 (*max(rate)*); (b) throughput of Slice 2 (*min(power)*); (c) PRB allocation.

Figure 5.11 showcases the unique ability of CellOS in implementing different control strategies for different network slices, *simultaneously optimizing different control programs on different network slices,* namely, Slice 1 and Slice 2, on each eNB. Specifically, Slice 1, which is allocated to MVNO 1, aims at maximizing the network throughput, while Slice 2, allocated to MVNO 2, minimizes the power consumption. The network sum and average throughput achieved by this per-slice behavior are shown in Figure 5.11. In our experiments, the two slices were allocated different percentages of the available PRBs (see Figure 5.11c): first 70% to Slice 1 and 30% to Slice 2 (Case A of Figure 5.11), then 50% to each slice (Case B), and finally a 30%–70% allocation was used (Case C). Figure 5.11a shows the throughput of Slice 1 in the three cases. Figure 5.11b presents that of Slice 2. As expected, the throughput of the *max(rate)* control program instantiated by MVNO 1 on Slice 1 increases with the resources allocated to the slice. On the contrary, the throughput performance of the *min(power)* control program instantiated by MVNO 2 on Slice 2 does not increase with the resources allocated to the slice. All three configurations of Figure 5.11b converge toward 7 $\mathrm{Mbit/s}$. This is due to the fact that this control problem aims at reaching the minimum per-user rate constraint set by the

Telco without consuming all available network resources. By looking at Figure 5.11, we notice that CellOS managed to independently optimize different control problems on different slices of the network (*max(rate* on Slice 1, and *min(power)* on Slice 2). This demonstrates that CellOS provides softwarized MVNOs with independent control of all resources in their leased network slice while sharing the same physical network infrastructure.

CellOS Scalability. We now evaluate the scalability of CellOS in terms of time and operations required by the controller to generate distributed solution programs, and by the REEs to solve them. Finally, we compare the overhead generated by CellOS REEs to that of state-of-the-art solutions, such as FlexRAN [3] and Orion [4]. The results presented in this section have been obtained by

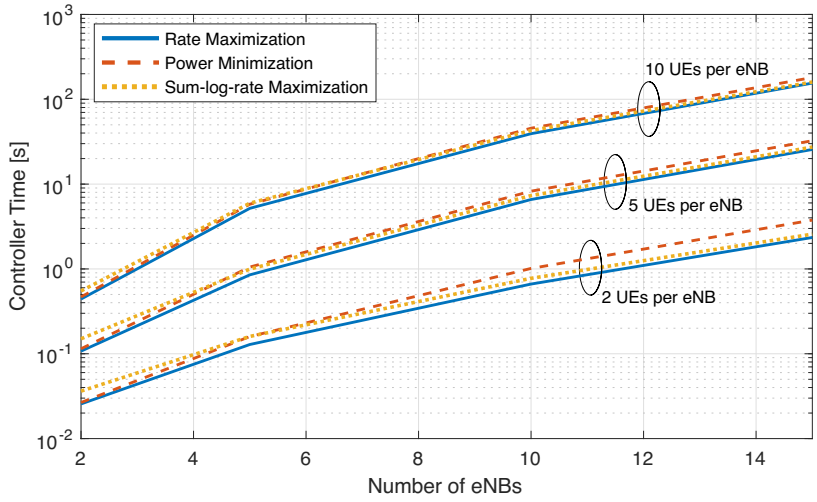


Figure 5.12: Scalability of CellOS controller operations as a function of the number of eNBs, UEs and for different network control problems.

executing CellOS on a single CPU of a Dell EMC PowerEdge R340 server of the Arena testbed (see Section 3.5). The server is equipped with an Intel Xeon E-2146G processor with 3.5 GHz base frequency and 32 GB DDR4-2666 RAM.

Figure 5.12 shows the time needed by CellOS controller to generate the distributed solution programs starting from the Telco directives as a function of the number of network eNBs, UEs, and for different network control problems. This includes the time to perform: (i) the problem definition procedures, which interpret the Telco high level directives; (ii) the generation of the centralized version of the problem based on an abstraction of the network, and (iii) the problem decomposition operations, which divide the centralized problem into sub-problems to be distributively solved by the softwarized RAN. We notice that, even though the computation time increases with the number of users and base stations, these operations are executed once per control problem. Also, recall that the generated problems utilize symbolic placeholders and do not require knowledge of real-time parameters. For this reason, all operations can be performed offline, and computation times are thus negligible if compared to the typical service times of cellular networks.

Figure 5.13 shows the time needed by CellOS REEs to solve the distributed problems automatically generated by the controller (Section 5.1.2) for different numbers of base stations and

UEs in the network. Different control problems require different solution times. For instance, the

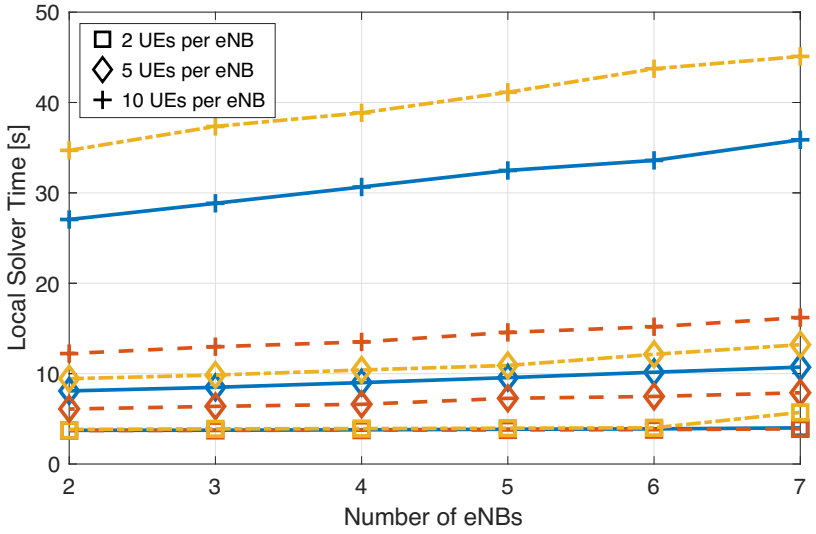


Figure 5.13: Scalability of CellOS local solver operations as a function of the number of eNBs, UEs and for different network control problems: (i) rate maximization (solid lines); (ii) sum-log-rate maximization (dot-dashed lines), and (iii) power minimization (dashed lines).

power minimization problem, whose objective function is a linear function in the transmission power variables, is solved more rapidly than the rate and sum-log-rate maximization problems, whose utility functions are non-linear because of logarithmic and fractional terms, which increase the problem complexity. As a consequence, the execution time of each problem strongly depends on the complexity of the underlying objective function to be optimized. It is worth noticing that the times of both Figures 5.12 and 5.13 can be considerably reduced if executed on high-performance equipment, as the one typically used in commercial cellular network deployments.

The signaling overhead generated by each CellOS REE is evaluated in Figure 5.14 against that generated by other well-established software-defined cellular control frameworks such as FlexRAN [3] and Orion [4]. Since CellOS executes the optimization problems locally at each REE, its overhead stems from the REEs exchanging $|\mathcal{U}|\,(|\mathcal{N}|+1)$ optimization variables and Lagrangian multipliers. These are the only information required to converge to a distributed problem solution (Section 5.1.2.3). These variables are represented by real numbers encoded as 32-bit floating point numbers. Figure 5.14 shows that the signaling overhead generated by CellOS REEs is significantly lower than that of prevailing state-of-the-art centralized approaches. Even when managing a single network base station, as it is the case of Figure 5.14, previous approaches must exchange a massive amount of local information with the central controller, thus generating large signaling and latency.

Experiment on the POWDER Platform. We demonstrate the platform- and RAN-independence of CellOS by running *long-range* experiments on one of the PAWR wireless platforms [5]. Specifically, we leverage the POWDER platform (see Section 2.7) and the 5G implementation of srsRAN to deploy a NR gNB and 2 UEs in an authentic outdoor wireless environment [6]. The gNB employs a USRP X310 located on the rooftop of a 28.75 m-tall building, while we use ground-level USRPs B210 as UEs. The gNB utilizes a reduced channel bandwidth of 15 PRBs (corresponding to 3 MHz)

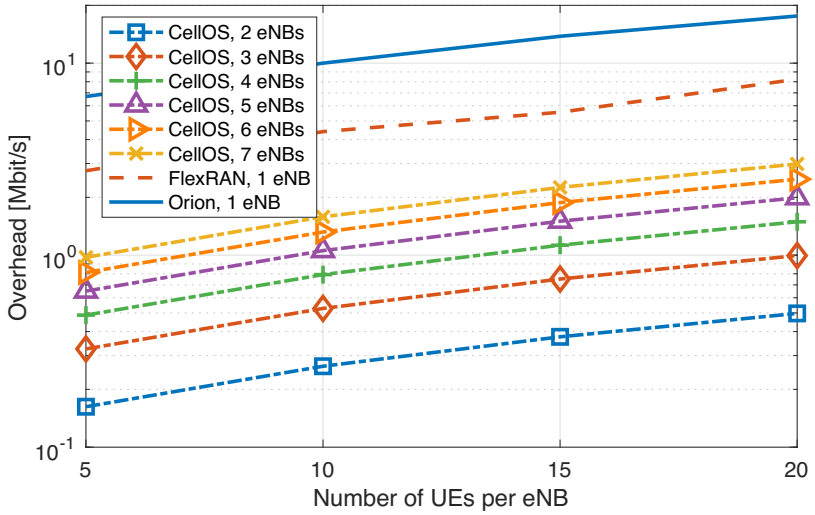
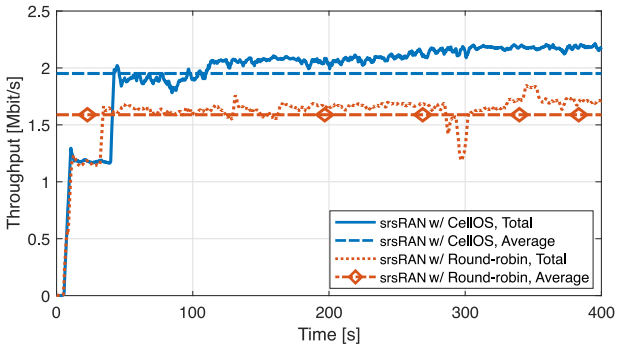


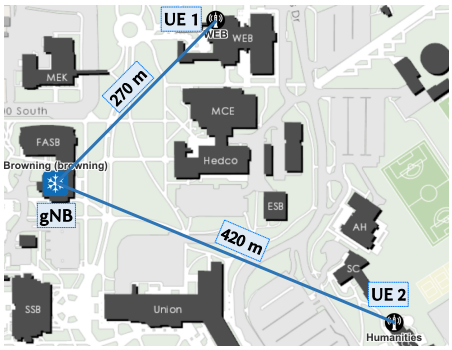
Figure 5.14: Signaling overhead: CellOS vs. FlexRAN [3, Figure 7] and Orion [4, Figure 13a].

to reach the two UEs distant 270 m and 420 m, respectively (see Figure 5.15b). In this case, the UEs download a file from a local server for 400 s.

Figure 5.15a shows the throughput gains achievable by running CellOS rate maximization on top of srsRAN, which uses a *round-robin* scheduler when instantiated without CellOS. Albeit the



(a) srsRAN w/ CellOS rate maximization vs. srsRAN w/ the round-robin scheduling algorithm



(b) Long-range experiment area

Figure 5.15: Long-range experiments on the POWDER PAWR platform [5, 6].

reduced bandwidth and increased gNB-UEs distance result in a lower total throughput than that of the previous experiments, we notice that CellOS significantly improves the network performance because of its zero-touch approach to optimization, which allows to optimize the resources allocated to the UEs, and bring gains as high as 86% (23% on average). To the best of our knowledge, this is the first demonstration of *zero-touch optimization* on a long-range open-source 5G testbed. Such instantiation gives evidence of the potential of the *softwarized Open RAN* approach cellular networks are moving toward.

5.2 QCell: Self-optimization of Softwarized NextG Networks Through Deep Q-Learning

In this section, we leverage data-driven techniques to push further the zero-touch optimization of NextG cellular networks. Differently to approaches based on traditional optimization techniques, AI/ML-based solutions are generally faster to react to network changes, and adapt more easily to unpredictable—and unforeseen—network dynamics and traffic demand.

To this aim, we introduce QCell, a DQN-based model-free framework for cellular network self-optimization that dynamically: (i) Allocates resources (e.g., bandwidth) to multiple slices of the network, and (ii) determines the optimal scheduling policy for each network slice. QCell is able to adapt in real time to varying network conditions and traffic demand by querying the performance at the Base Stations (BSs), quickly adapting to the heterogeneous and time-varying context of 5G cellular networks. The modular design of QCell allows it to either execute as a standalone distributed framework on each network BS, or as part of larger frameworks such as O-RAN (e.g., as an xApp, see Section 2.4.1).

We build a prototype of QCell on the SCOPE framework of Section 3.2, which is based on srsRAN (see Section 2.2.2). Finally, we train and test QCell on Colosseum, the world's largest wireless network emulator with hardware-in-the-loop, which allows to design, prototype and test wireless solutions at scale in a variety of emulated network scenarios and conditions (see Section 3.4). Our experimental results involving 25 SDRs over a variety of different network scenarios show that QCell significantly improves the throughput of cellular users up to 37.6%, and decreases the size of downlink transmission queues by up to 11.9%, thus decreasing the service latency.

The remainder of this section is organized as follows. Section 5.2.1 provides some background knowledge on DQNs. The QCell architecture is detailed in Section 5.2.2. Finally, Section 5.2.3 presents our QCell prototype and experimental results.

5.2.1 Deep Q-Networks: A Primer

A Deep Q-Network (DQN) is a DRL technique used to determine policies π from high-dimensional inputs by leveraging reinforcement learning techniques [272, 382]. In a DQN model, an *agent* interacts with an *environment* through a sequence of *observations* of the environment (formed by a series of *states*), *actions*, and *rewards*. A sequence of states, actions, and rewards that lead to a terminal state of the system is called an *episode*. The agent's purpose is to choose the action that maximizes the cumulative future rewards.

The optimal state-action value function $Q^*(s, a)$, *aka* the *Q-function,* of a policy π, namely, $Q^\pi(s, a)$, measures the expected return—or sum of discounted rewards—that is obtained by taking action a from state s, and then following the policy π thereafter. $Q^*(a, s)$ is defined as the maximum return that is obtainable by taking action a in state s, and then following the optimal policy. This is approximated with a Deep Neural Network (DNN), which produces the cumulative future reward r_t at time t, discounted by a factor $\gamma \in [0, 1]$.

$$Q^*(s, a) = \left[r + \gamma \max_{a'} Q^*(s', a') \right], \tag{5.14}$$

where s' is the next state, and a' is the action taken from it.

To cope with instability due to the correlation between consecutive episodes, the *experience replay* D_t at time step t needs to be considered [272]. This is a set of agent's experiences $\{e_1, \ldots, e_t\}$ that randomizes over data, where $e_i = (s_i, a_i, r_i, s_{i+1})$ is the set of state, action, reward, and next state. At each time step t of the learning phase, the agent performs an action a_t, which causes the system to transition to state s_t. The experience vector $e_t = (s_t, a_t, r_t, s_{t+1})$ is, then, stored in the experience replay D_t. To reduce the correlation between Q-function value Q and the optimal Q^*, a second Q-Network—namely target Q-Network—is introduced. The structure of the target Q-Network is the same as that of the original Q-Network, while its weights are updated periodically to those of the original Q-Network. Finally, the Q-learning update at iteration t leverages the following loss function:

$$L_t(\theta_t) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_t^-) - Q(s, a; \theta_t)\right)^2\right], \tag{5.15}$$

where θ_t (θ_t^-) are the weights of the original (target) Q-Network.

5.2.2 QCell Architecture

The QCell architecture consists of a distributed multi-agent optimization framework. A DQN agent runs on each cellular BS and makes control decisions on its configuration, e.g., scheduling and slicing policies. These decisions are based on the real-time performance of the network, i.e., the network state, signaled through messages exchanged among the agents of the interfering BSs. In this way, agents are able to learn the best action to optimize the performance of their own BS, while keeping the interference to the other BSs as low as possible.

Figure 5.16 shows the architecture of the QCell DQN framework. This is formed by three components: (i) the agent; (ii) the BS protocol stack, and (iii) the BS connector.

Each *agent* is equipped with two different encoders (A and B in the figure) used to reduce the dimensionality of the input data, a reward calculator, and a DQN. At each time instant t, the agent reads the network state (s_t), which reflects the performance of the BS on which QCell is deployed, and of the UEs served by it. The feed-forward multi-layer neural network encoders are used to normalize the dimensionality of the input, which depends on the number of users served by the BS. Both encoders A and B are fed with the current network state and retain as output the information relevant to the DQN. The output of encoder A (ν_t) represents the state of the QCell BS (regardless of the state of each user). This is sent to the interfering BSs by the *BS connector*, along with the reward of the action taken at time step $t-1$. The output of encoder B (ϕ_t) is used to set the input of the DQN to a given dimension, while retaining more than 85% of the original dataset variance through the Principal Component Analysis process [383].

The *reward calculator* receives the metrics sent from the interfering BSs from the BS connector and computes the reward for the current time step t. This is the linear combination of the variation of the metrics of the users served by the BS, and the rewards received in input by the interfering QCell BSs.

The DQN is formed by a 4-layer deep convolutional neural network. It takes as input the output of encoder B and the state signaled by the other QCell BSs (received by the BS connector), and

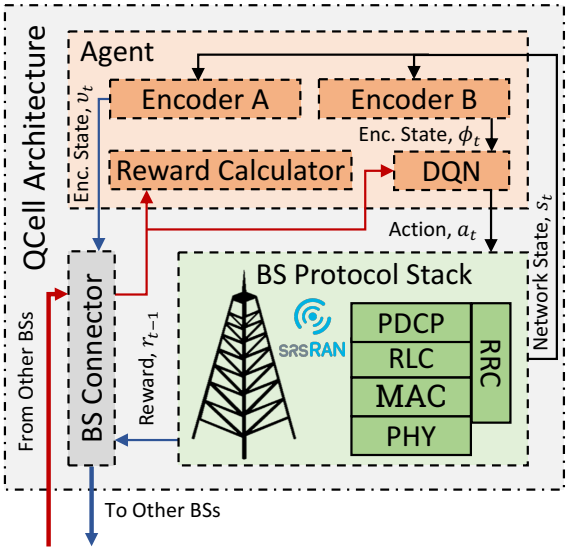


Figure 5.16: The QCell architecture.

returns as output a set of Q-values, each of which is associated with a different action and network configuration. Among the possible returned actions, the one associated with the highest Q-value (a_t) is implemented at the BS. For the sake of improving the efficiency of the training, the DQN has been decoupled from the encoders, and the three components have been trained separately [382]. The action space of each agent is defined by a finite set of possible network configurations. For each network slice, actions are the combination of a scheduling policy (we considered the round-robin, waterfilling and proportionally fair scheduling policies) and resource allocation of the slice, i.e., the number of RBGs allotted to the slice (see Appendix B).

The reward function of each agent is shown in Algorithm 1. This is a function of the average variation of the metrics of the users the BS is serving, and of those sent by the interfering QCell BSs. Specifically, the goal of the QCell agents is to simultaneously optimize the downlink throughput of the users and the size of their downlink transmission buffer, i.e., the buffer containing the data the BS needs to transmit to each user in downlink, which directly reflects on the service latency.

The *BS connector* allows the communication between the BSs of the network. It receives the state encoded by encoder A (ν_t) and the previous-step reward (r_{t-1}), and sends them to the interfering BSs of the network. Similarly, it also receives analogous information from the interfering BSs and forwards it to the *reward calculator* and to the DQN.

Finally, the *BS protocol stack* implements the softwarized cellular BS, including layers such as PHY, MAC, RLC, PDCP, and RRC (see Chapter 2). This enables the communication with the UEs of the network and can be implemented through open-source software solutions, e.g., srsRAN (see Section 5.2.3.1). Regardless of the specific implementation, this element provides the network state to encoders A and B of the QCell agent, and the reward obtained from the last time step to the BS connector.

Algorithm 1 QCell agent reward function at time t.

```
Output Reward r_t;
Set My reward r_t^m = 0;
for all users u ∈ U do
    for all metrics m do
        sign ← (m_t − m_{t−1}) / |m_t − m_{t−1}|;
        if sign = −1 then
            sign ← −1.2;
        end if
        variation z ← |m_t − m_{t−1}| / m_{t−1};
        if z ⩽ 0.01 then
            r_t^m ← r_t^m + sign;
        else if z ⩽ 0.05 then
            r_t^m ← r_t^m + 5 · sign;
        else if z ⩽ 0.1 then
            r_t^m ← r_t^m + 10 · sign;
        else
            r_t^m ← r_t^m + 20 · sign;
        end if
    end for
end for
r_t^m ← r_t^m / |U|;
send(r_t^m);
other BSs' rewards r_{t−1}^{oth} ← receive_rewards();
r_t ← 0.8 · r_t^m + 0.1 · sum(r_{t−1}^{oth});
```

5.2.2.1 QCell Example

We will now give a high-level example of the QCell algorithm workflow. (The detailed implementation of the QCell algorithm can be found in [543].) We consider a scenario with 3 interfering BSs, namely a, b, and c (see Figure 5.17).

As the first step, the two encoders (A and B in the figure) read the network state s_t^a, and the BS connector reads the previous reward r_{t-1}^a (step 1 in Figure 5.17). Encoder A encodes the state information of the BS into ν_t (step 2). This is sent to the agents of the interfering QCell BSs from the BS connector, together with the information about the reward of the agent at time $t-1$, i.e., r_{t-1}^a (step 3, outgoing blue arrow). At the same time, this component receives the signaling messages from the interfering BSs (step 3, incoming red arrow). These messages contain the network state and previous-step reward of the QCell agents of the nearby BSs. In step 4, the received rewards (r_{t-1}^b and r_{t-1}^c) are used to compute the final agent reward according to Algorithm 1. The received states (s_t^b and s_t^c), instead, are fed to the DQN together with the output ϕ_t produced by encoder B (step 5). Finally, in step 6, the agent action at time t, a_t, is computed by the DQN.

5.2.3 Experimental Results

In this section, we describe the network emulator we leveraged to develop our QCell prototype, the DQN training process, and the obtained experimental results.

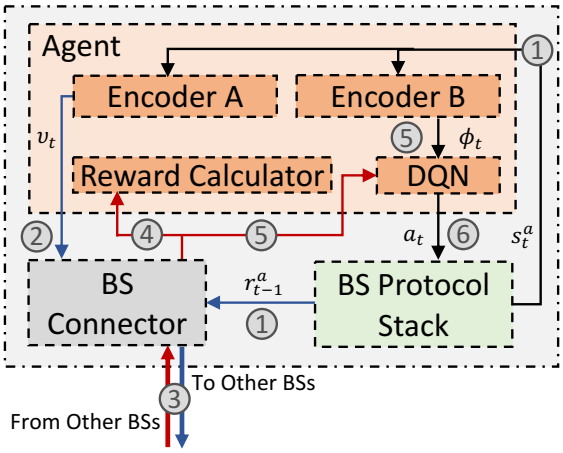


Figure 5.17: Example of the QCell algorithm workflow.

5.2.3.1 QCell Prototype

We prototyped QCell on Colosseum, the world's largest wireless network emulator with hardware-in-the-loop (see Section 3.4), and implemented 3GPP-compliant BSs and UEs through the SCOPE framework (see Section 3.2). Specifically, we leveraged SCOPE to deploy 3 cellular BSs and up to 22 UEs on Colosseum SRNs. For each network BS, we considered 2 network slices and 3 different scheduling policies: round-robin, waterfilling, and proportionally fair scheduling policies. The downlink center frequencies of the BSs were set to 1000.0 MHz, 1002.5 MHz, and 1005.0 MHz (i.e., partially interfering but not completely overlapped), while the channel bandwidth to 3 MHz, corresponding to 15 Physical Resource Blocks (8 RBGs).

Traffic among BSs and UEs was generated through `iPerf3`, which allows to measure the network performance through TCP/UDP packet flows. To account for realistic—yet unpredictable—traffic conditions, we varied randomly the traffic rate and the length of the packets between BSs and UEs in $[3, 10]$ Mbps and $[3, 10]$ kB, respectively.

As for the DQN, we selected the behavioral distribution through an ϵ-greedy policy. At every time step, the environment state is defined by the state of every agent, which is determined by the metrics of the UEs served by the BSs. We considered the following metrics collected at each BS: (i) slice resource ratio, as the fraction of RBGs assigned to the slice the UE belongs to; (ii) granted resource ratio, as the ratio between the resources granted and required by the UEs; (iii) channel quality information, which gives an indication of the quality of the channel, as perceived by the UEs; (iv) downlink throughput; (v) downlink buffer size, as the amount of data the BSs need to transmit to the UEs at any given moment, and (vi) modulation and coding scheme.

The agent actions are composed of the fraction of RBGs to allocate to each network slice, and the scheduling policy to use at the BS. A sample of the action space adopted in our QCell prototype is shown in Table 5.2. We recall that the total number of RBGs used in our prototype is 8. Finally, the agent reward at time t is computed as the linear combination of the variation of the metrics of the UEs served by the current BS, and the metrics sent by the agents of the interfering QCell BSs (see Algorithm 1).

A diagram of the QCell prototype we implemented on Colosseum is shown in Figure 5.18. We

Table 5.2: Sample action space of QCell prototype.

Action	Fraction RBGs Slice 1	Fraction RBGs Slice 2	Scheduling Policy
0	1/2	1/2	Round-robin
1	1/2	1/2	Waterfilling
2	1/2	1/2	Proportionally Fair
3	1/4	3/4	Round-robin
4	1/4	3/4	Waterfilling
5	1/4	3/4	Proportionally Fair
6	3/4	1/4	Round-robin
7	3/4	1/4	Waterfilling
8	3/4	1/4	Proportionally Fair

Table 5.3: QCell hyperparameters.

Hyperparameter	Value
Minibatch size	32
Replay memory size	10000
Target network update frequency	35
Discount factor, γ	0.9
Learning rate	0.01
Initial exploration value for ϵ	1
Final exploration value for ϵ	0.05
Final exploration iteration	10000

deployed 3 softwarized BSs and 22 UEs on Colosseum SRNs, and emulated a urban scenario through MCHEM. Each BS runs a *QCell agent*. The agent reads the *run-time performance* of the BS (e.g., network conditions, traffic demand, and level of service provided to the UEs), which forms the network state. It then encodes this state and signals it, together with the previous reward, to the interfering BSs (which are also running QCell) through the *BS connector*. At the same time, this component receives the signaling messages from the interfering BSs and forwards them to the *QCell agent*. Finally, the agent leverages this information, together with the network state, to compute the optimal action (i.e., slicing and scheduling policies) through the DQN, and re-configures the BS at run time.

5.2.3.2 QCell Training

To let QCell interact dynamically with a diversified environment, in our training we periodically reinitialize the experiments on Colosseum, randomly assigning UEs to the network BSs. QCell periodically reads the up-to-date network performance collected at the BS by srsRAN and modifies slicing and scheduling policies used to serve the UEs, if necessary.

Relevant QCell hyperparameters are shown in Table 5.3. We set the *minibatch size* to 32, which determines the number of training cases over which the Stochastic Gradient Descent (SGD) algorithm is computed. The *replay memory size* (set to 10000) regulates the number of training cases considered by the SGD algorithm, while the *target network update frequency*, measured as the number of network parameter updates, is set to 35. The *discount factor* γ of the Q-learning update, and the *learning rate* of the SGD algorithm are set to 0.9 and 0.01, respectively. Finally, the *initial* and *final exploration* ϵ values, which enable QCell to self-adapt to unseen network scenarios, are set to 1 and 0.05, respectively, and the *final exploration iteration* to 10000.

Our training consisted of almost 16000 episodes, for a total of 150 hours of training. Figure 5.19 shows the loss value evolution over the training episodes. As the training goes on, we notice a decrease in the average loss value after around 1900 episodes. This metric, then, continues to decrease until the end of the training, indicating convergence of QCell learning process.

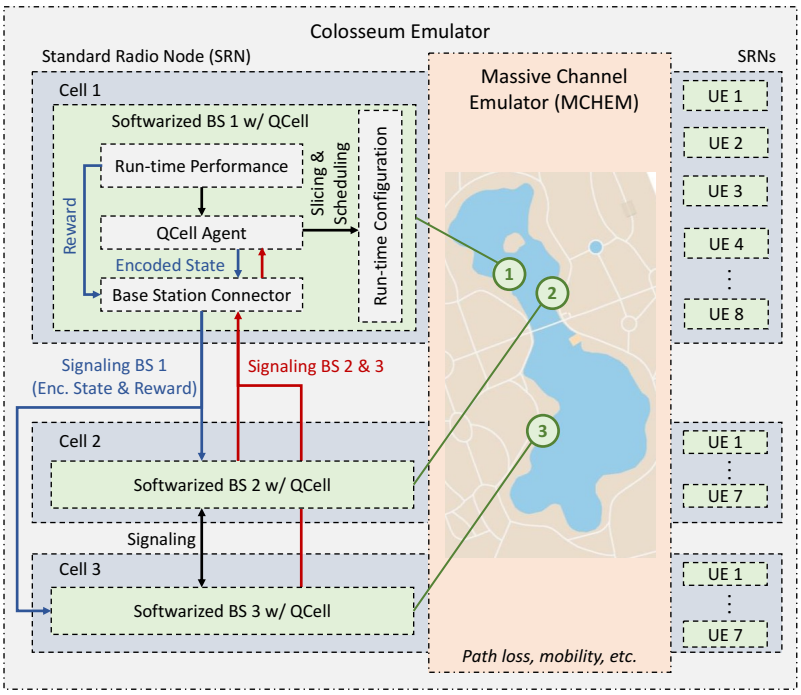


Figure 5.18: The QCell prototype implementation on Colosseum.

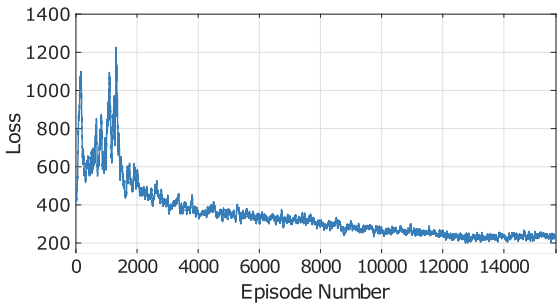


Figure 5.19: Loss value vs. number of episodes.

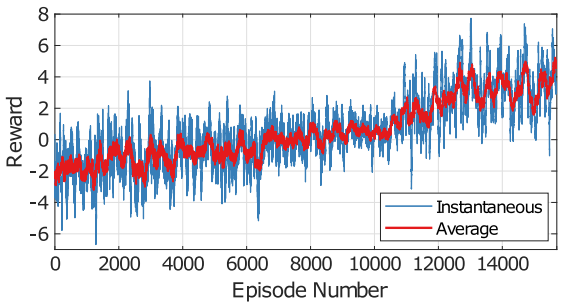


Figure 5.20: Reward value vs. number of episodes.

Figure 5.20 shows the reward of QCell agents, both instantaneous and average, over the various episodes of the training. We can see that the agent reward keeps increasing, on average, for the whole duration of the training, except for the final 3000 episodes, in which the reward oscillates around similar value intervals. This suggests convergence of the QCell algorithm, which is able to get the average optimal reward for each episode. Moreover, at the end of the training, the reward assumes positive values, demonstrating that QCell can efficiently optimize the performance of the network.

5.2.3.3 QCell Testing

To test QCell, we ran over 17 hours of experiments on the Colosseum testbed (see Section 5.2.3.1). We compared the network performance in terms of downlink throughput and buffer queue length

with and without QCell. In the case without QCell, we considered both the average of all the static configurations (see Table 5.2), and the best performing static configuration (action 7 in Table 5.2).

Figure 5.21 depicts the downlink throughput with and without QCell in the above-mentioned cases. We notice that QCell is able to significantly improve the user throughput compared to both average and best static configurations. This is due to the tight interactions among the BS agents, which allow to dynamically reconfigure the network at run time based on the current network conditions and performance.

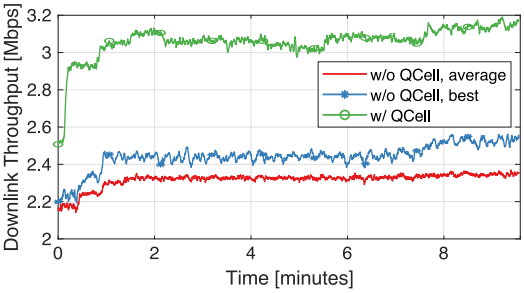


Figure 5.21: Throughput w/ and w/o QCell.

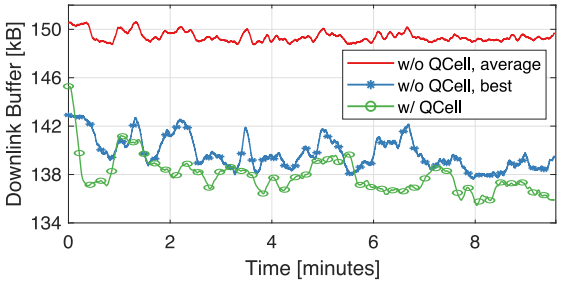


Figure 5.22: Buffer occupancy w/ and w/o QCell.

Figure 5.22 shows the average downlink buffer occupancy with and without QCell. In general, the occupancy of the downlink buffer is reduced when QCell is active, implying a faster service to the UEs. This is due to the superior behavior of QCell, which leverages the UE feedback collected at the BSs to efficiently adapt to the varying network dynamics.

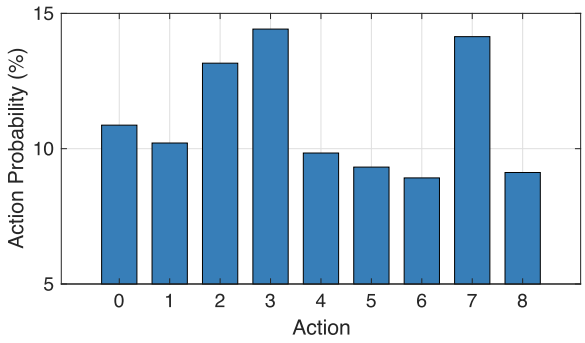


Figure 5.23: Agent action probability distribution.

The action probability distribution of QCell agents is depicted in Figure 5.23. This metric conveys how often a certain action is selected by the agents after the training phase has been completed. We observe that the actions (reported in Table 5.2) are chosen with similar probabilities by QCell, which enacts the best-performing configuration based on the specific network scenario and traffic demand.

Finally, we analyzed the frequency at which QCell modifies the BS configuration after the training phase has been completed. Results confirmed the overall stability of QCell optimization framework, which varied the configuration of the BSs every 101.45 s, on average (corresponding to a 0.44 coefficient of variation).

5.3 Related Work

Recent years have heralded SDN as the technology that would inherently endow the *monolithic* Internet architecture with much needed *flexibility*. The largest part of SDN work focuses on the *programmability* of wired networks, with few works exploring scenarios comprising wireless devices [3, 35, 347, 384–387].

Guan et al. proposed WNOS, a wireless network operating system featuring network virtualization and distributed solution of optimization problems [347]. Although this work is the most similar to CellOS, it only focuses on infrastructure-less ad hoc networks with static nodes. For this reason, it is not suitable to handle mobile and dynamic cellular scenarios. An effort to explicitly take mobility into account is made by Bertizzolo et al. with SwarmControl, a distributed control framework for the self-optimization of drone networks [536].

ONAP and O-RAN are two infrastructure-oriented automation platforms with the ambition of "orchestrating" many network functions [388, 389]. They offer Telcos network abstractions to specify system details and traffic policies. However, optimization policies and algorithms must be explicitly programmed.

Adaptations of the SDN paradigm to cellular networks have been proposed by Li et al. (CellSDN [386]), Bernardos et al. (SDWN [390]), and by Bradai et al. (CSDN [387]). CellSDN proposes a control-oriented operating system focused on cellular network management and subscriber policies rather than on performance optimization. Works like SDWN and CSDN, instead, describe general frameworks to optimize network utilization and performance leveraging edge network information.

Few works have addressed the interplay between the SDN architecture and that of networks including LTE explicitly. Gudipati et al. envision SoftRAN as an abstraction of all eNBs in a geographical area as a single virtual base station to perform operations including metrics optimization [385]. This centralized approach, however, can hardly address heterogeneous optimization problems in the dense, flexible and rapidly growing architecture of 5G cellular networks. Foukas et al. propose FlexRAN [3] and Orion [4] as centralized controllers coordinating various LTE agents, and supporting network slicing, respectively. These systems, though, neglect optimization, and their centralized nature may result in limited scalability and reduce the performance in dense scenarios. OpenRadio, by Bansal et al., develops a programmable wireless data plane providing programming interfaces on PHY and MAC layers [384]. Optimization, however, is left to the wits of the TO. We notice that all the mentioned solutions for cellular networks propose programmable protocol stack implementations where the optimization procedures need to be *manually designed* and there is no way to perform them *dynamically* or *automatically*.

Machine learning-based solutions for optimized cellular performance concern specific aspects of the network architecture [304, 524]. For instance, Li et al. devise a DRL framework for resource management in network slicing in which actions are defined as radio resource allocations [391]. Although simplifying system design, this choice significantly increases the policy space, making it harder to find optimal solutions. The work by Chinchali et al. [316] and the work discussed in Chapter 4 present learning-based schedulers that adapt to traffic demand and different reward functions set by the telecom operators.

Overall, both CellOS and QCell take an unconventional approach to self-optimization of softwarized cellular networks by making decisions on slicing and scheduling policies of the BSs starting from high-level directives of the Telcos. As demonstrated through extensive experimental campaigns,

these approaches self-adapts to varying network conditions and traffic, significantly improving key performance metrics of next generation cellular networks.

5.4 Conclusions

In this chapter, we described distributed frameworks, namely CellOS and QCell, for the zero-touch self-optimization of the next-generation cellular Open RAN: CellOS—based on traditional optimization techniques—*automatically* converts the high-level operator intent into distributed solution programs to be run at the base stations to simultaneously optimize heterogeneous objectives on different network slices, and QCell—which leverages DQN techniques—follows a multi-agent approach for dynamic slicing and scheduling allocation to BSs at run time, adapting to current network and traffic conditions.

Experimental results of both frameworks—prototyped through 3GPP-compliant software tools (e.g., OAI, srsRAN), and on open-access wireless platforms, i.e., the indoor Arena testbed (see Section 3.5) and the outdoor POWDER platform from the PAWR program (see Section 2.7) in the case of CellOS, and the Colosseum network emulator (see Section 3.4) in the case of QCell—show remarkable performance improvements in metrics such as network throughput, transmission power and queue size, and service latency.

Chapter 6

Orchestration in the Open RAN

The O-RAN architecture makes it possible to bring automation and intelligence to the network through ML and AI, which will leverage the enormous amount of data generated by the RAN—and exposed through the O-RAN interfaces—to analyze the current network conditions, forecast future traffic profiles and demand, and implement closed-loop network control strategies to optimize the RAN performance. For this reason, how to design, train and deploy reliable and effective data-driven solutions has recently received increasing interest from academia and industry alike, with applications ranging from controlling RAN resource and transmission policies [353, 356, 357, 392–396, 521, 526, 543], to forecasting and classifying traffic and KPIs [307, 340, 397–399], thus highlighting how these approaches will be foundational to the Open RAN paradigm. However, how to deploy and manage, i.e., *orchestrate*, intelligence into softwarized cellular networks is by no means a solved problem for the following reasons:

- *Complying with time scales and making input available.* Adapting RAN parameters and functionalities requires control loops operating over time scales ranging from a few milliseconds (i.e., real-time) to a few hundreds of milliseconds (i.e., near-real-time) to several seconds (i.e., non-real-time) [400, 526]. As a consequence, the models and the location where they are executed need to be selected to be able to retrieve the necessary inputs and compute the output within the appropriate time constraints [401, 526]. For instance, while IQ samples are easily available in real time at the RAN, it is extremely hard to make them available at the near-real-time and non-real-time RICs within the same temporal window, making the execution of models that require IQ samples as input on the RICs ineffective.

- *Choosing the right model.* Each ML/AI model is designed to accomplish specific inference and/or control tasks and requires well-defined inputs in terms of data type and size. One must make sure that the most suitable model is selected for a specific Network Operator (NO) request, and that it meets the required performance metrics (e.g., minimum accuracy), delivers the desired inference/control functionalities, and is instantiated on nodes with enough resources to execute it.

- *Conflict mitigation.* One must also ensure that selected ML/AI models do not conflict with each other, and that the same parameter (or functionality) is controlled by only a single model at any given time.

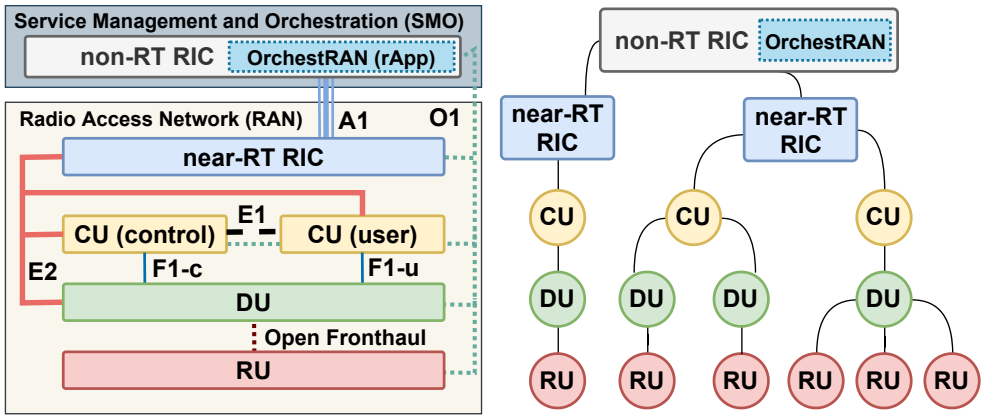


Figure 6.1: O-RAN reference architecture and interfaces (left). Representation of an O-RAN network architecture as a tree graph (right).

For these reasons, *orchestrating* network intelligence in the Open RAN presents unprecedented and unique challenges that call for innovative, automated and scalable solutions. In this chapter, which was published as [546], we address these challenges by presenting OrchestRAN, an automated intelligence orchestration framework for the Open RAN. OrchestRAN follows the O-RAN specifications and operates as an rApp executed in the non-real-time RIC (Figure 6.1) providing automated routines to: (i) Collect control requests from NOs; (ii) select the optimal ML/AI models to achieve NOs' goals and avoid conflicts; (iii) determine the optimal execution location for each model complying with timescale requirements, resource and data availability, and (iv) automatically embed models into O-RAN applications that are dispatched to selected nodes, where they are executed and fed the required inputs.

To achieve this goal, we have designed and prototyped novel algorithms embedding preprocessing variable reduction and branching techniques that allow OrchestRAN to compute orchestration solutions with different complexity and optimality trade-offs, while ensuring that the NOs intents are satisfied. We evaluate the performance of OrchestRAN in orchestrating intelligence in the RAN through numerical simulations, and by prototyping OrchestRAN on ColO-RAN [528], an O-RAN-compliant large-scale experimental platform developed on top of Colosseum, the world's largest wireless network emulator with hardware in-the-loop [545]. Experimental results on an O-RAN-compliant softwarized network with 7 cellular base stations and 42 users demonstrate that OrchestRAN enables seamless instantiation of O-RAN applications with diverse timescale requirements at different O-RAN components. OrchestRAN automatically selects the optimal execution locations for each O-RAN application, thus moving network intelligence to the edge with up to 2.6$\times$ reduction of control overhead over the O-RAN E2 interface. To the best of our knowledge, this is the first large-scale demonstration of an O-RAN-compliant network intelligence orchestration system.

The remainder of this chapter is organized as follows. OrchestRAN, its building blocks and procedures are presented in Section 6.1. Section 6.2 formulates the intelligence orchestration problem and shows its NP-hardness, while Section 6.3 presents low-complexity and scalable algorithms to solve it. The performance of OrchestRAN is assessed numerically and experimentally in Sections 6.4 and 6.5, respectively. Finally, Section 6.6 surveys works related to OrchestRAN, and Section 6.7

concludes the chapter.

6.1 OrchestRAN

As illustrated in Figure 6.1, OrchestRAN is designed to be executed as an *rApp* at the non-real-time RIC. Its architecture is illustrated in Figure 6.2. At a high-level, first NOs specify their intent by

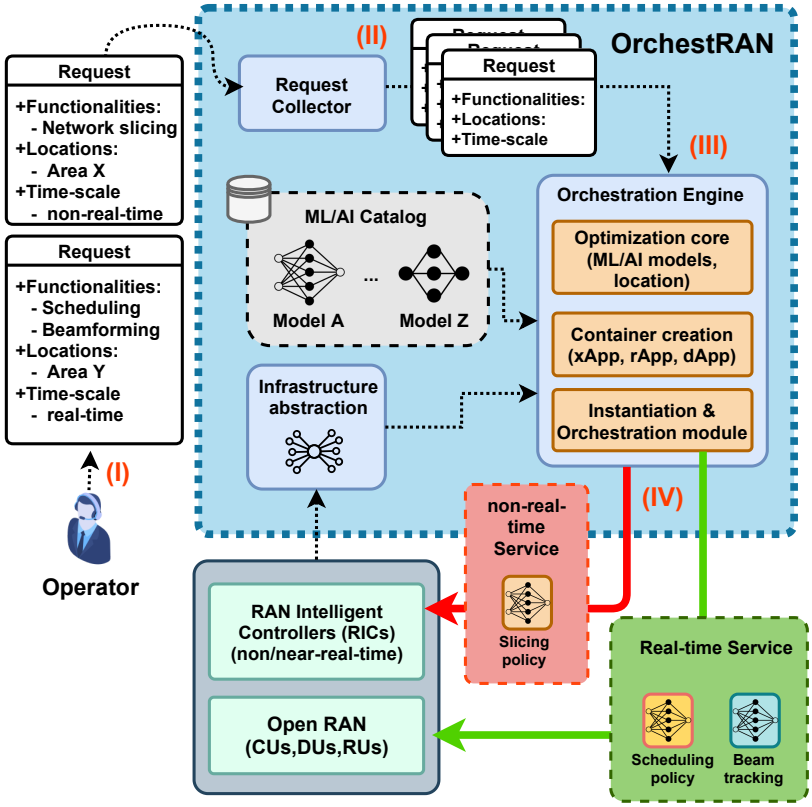


Figure 6.2: System design of OrchestRAN and main procedures.

submitting a control request to OrchestRAN (step I). This includes the set of functionalities they want to deploy (e.g., network slicing, beamforming, scheduling control, etc.), the location where functionalities are to be executed (e.g., RIC, CU, DU) and the desired time constraint (e.g., delay-tolerant, low-latency). Then, requests are gathered by the *Request Collector* (step II, Section 6.1.3) and fed to the *Orchestration Engine* (step III, Section 6.1.4) which: (i) Accesses the *ML/AI Catalog* (Section 6.1.2) and the *Infrastructure Abstraction* module (Section 6.1.1) to determine the optimal orchestration policy and models to be instantiated; (ii) automatically creates containers with the embedded ML/AI models in the form of O-RAN applications, and (iii) dispatches such applications at the locations determined by the Orchestration Engine.

6.1.1 The Infrastructure Abstraction Module

This module provides a high-level representation of the physical RAN architecture, which is divided into five separate logical groups: non-real-time RICs, near-real-time RICs, CUs, DUs and RUs. Each group contains a different number of nodes deployed at different locations of the network. Let $\mathcal{D}$ be the set of such nodes, and $D = |\mathcal{D}|$ be their number.

The hierarchical relationships between nodes can be represented via an undirected graph with a tree structure such as the one in Figure 6.1 (right). Specifically, leaves represent nodes at the edge (e.g., RUs/DUs/CUs), while the non-real-time RIC is the root of the tree.[1] For any two nodes $d', d'' \in \mathcal{D}$, we define variable $c_{d',d''} \in \{0, 1\}$ such that $c_{d',d''} = 1$ if node d' is reachable from node d'' (e.g., there exist a communication link such that node d' can forward data to node d''), $c_{d',d''} = 0$ otherwise. In practical deployments, it is reasonable to assume that nodes on different branches of the tree are unreachable. Moreover, for each node $d \in \mathcal{D}$, let ρ_d^ξ be the total amount of resources of type $\xi \in \Xi$ dedicated to hosting and executing ML/AI models and their functionalities, where Ξ represents the set of all resource types. Although we do not make any assumptions on the specific types of resources, practical examples may include the number of CPUs, GPUs, as well as available disk storage and memory. In the following, we assume that each non-real-time RIC identifies an independent networking domain and the set of nodes $\mathcal{D}$ includes near-real-time RICs, CUs, DUs and RUs controlled by the corresponding non-real-time RIC only.

6.1.2 The ML/AI Catalog

In OrchestRAN, the available pre-trained data-driven solutions are stored in a *ML/AI Catalog* consisting of a set $\mathcal{M}$ of ML/AI models. Let $\mathcal{F}$ be the set of all possible control and inference functionalities (e.g., scheduling, beamforming, capacity forecasting, handover prediction) offered by such ML/AI models—hereafter referred to simply as "models".

Let $M = |\mathcal{M}|$ and $F = |\mathcal{F}|$. For each model $m \in \mathcal{M}$, $\mathcal{F}_m \subseteq \mathcal{F}$ represents the subset of functionalities offered by m. Accordingly, we define a binary variable $\sigma_{m,f} \in \{0, 1\}$ such that $\sigma_{m,f} = 1$ if $f \in \mathcal{F}_m$, $\sigma_{m,f} = 0$ otherwise. We use ρ_m^ξ to indicate the amount of resources of type $\xi \in \Xi$ required to instantiate and execute model m. Let $\mathcal{T}$ be the set of possible input types. For each model $m \in \mathcal{M}$, $t_m^{\text{IN}} \in \mathcal{T}$ represents the type of input required by the model (e.g., IQ samples, throughput and buffer size measurements).

Naturally, not all models can be equally executed everywhere. For example, a model m performing beam alignment [340], in which received IQ samples are fed to a neural network to determine the beam direction, can only execute on nodes where IQ samples are available. While IQ samples can be accessed in real-time at the RU, they are unlikely to be available at CUs and the RICs without incurring in high overhead and transmission latency. For this reason, we introduce a suitability indicator $\beta_{m,f,d} \in [0, 1]$ which specifies how well a model m is suited to provide a specific functionality $f \in \mathcal{F}$ when instantiated on node d. Values of $\beta_{m,f,d}$ closer to 1 mean that the model is well-suited to execute at a specific location, while values closer to 0 indicate that the model performs poorly. We also introduce a performance score $\gamma_{m,f}$ measuring the performance of the model with respect to $f \in \mathcal{F}$. Typical performance metrics include classification/forecasting accuracy, mean squared error

[1] Coexisting CUs/DUs/RUs are modeled as a single logical node with a hierarchy level equal to that of the hierarchically highest node in the group.

and probability of false alarm. A model can be instantiated on the same node multiple times to serve different NOs or traffic classes. However, due to limited resources, each node d supports at most $C_{m,d} = \min_{\xi \in \Xi}\{\lfloor \rho_d^\xi / \rho_m^\xi \rfloor\}$ instances of model m, where $\lfloor \cdot \rfloor$ is the floor operator.

6.1.3 Request Collector

OrchestRAN allows NOs to submit requests specifying which functionalities they require, where they should execute, and the desired performance and timing requirements. Without loss of generality, we assume that each request is feasible. The Request Collector of OrchestRAN is in charge of collecting such requests. A request i is defined as a tuple $(\mathcal{F}_i, \boldsymbol{\pi}_i, \boldsymbol{\delta}_i, \mathcal{D}_i^{\text{IN}})$, with each element defined as follows:

- *Functions and locations.* For each request i, we define the set of functionalities that must be instantiated on the nodes as $\mathcal{F}_i = (\mathcal{F}_{i,d})_{d \in \mathcal{D}}$, with $\mathcal{F}_{i,d} \subseteq \mathcal{F}$. Required functionalities and nodes are specified by a binary indicator $\tau_{i,f,d} \in \{0, 1\}$ such that $\tau_{i,f,d} = 1$ if request i requires functionality f on node d, i.e., $f \in \mathcal{F}_{i,d}$, $\tau_{i,f,d} = 0$ otherwise. We also define $\mathcal{D}_i = \{d \in \mathcal{D} : \sum_{f \in \mathcal{F}_i} \tau_{i,f,d} \geq 1\}$ as the subset of nodes of the network where functionalities in $\mathcal{F}_i$ should be offered;

- *Performance requirements.* For any request i, $\boldsymbol{\pi}_i = (\pi_{i,f,d})_{d \in \mathcal{D}_i, f \in \mathcal{F}_{i,d}}$ indicates the minimum performance requirements that must be satisfied to accommodate i. For example, if f is a beam detection functionality, $\pi_{i,f,d}$ can represent the minimum detection accuracy of the model. We do not make any assumptions on the physical meaning of $\pi_{i,f,d}$ as it reasonably differs from one functionality to the other.

- *Timing requirements.* Some functionalities might have strict latency requirements that make their execution at nodes far away from the location where the input is generated impractical or inefficient. For this reason, $\delta_{i,f,d} \geq 0$ represents the maximum latency request i can tolerate in executing f on d;

- *Data source.* For each request i, the NO also specifies the subset of nodes whose generated (or collected) data must be used to deliver functionality f on node d. This set is defined as $\mathcal{D}_i^{\text{IN}} = (\mathcal{D}_{i,f,d}^{\text{IN}})_{d \in \mathcal{D}_i, f \in \mathcal{F}_{i,d}}$, where $\mathcal{D}_{i,f,d}^{\text{IN}} \subseteq \mathcal{D}$. This information is paramount to ensure that each model is fed with the proper data generated by the intended sources only. For any tuple (i, f, d) we assume that $c_{d,d'} = 1$ for all $d' \in \mathcal{D}_{i,f,d}^{\text{IN}}$.

In the remaining of this chapter, we use $\mathcal{I}$ to represent the set of outstanding requests with $I = |\mathcal{I}|$ being their number.

6.1.4 The Orchestration Engine

As depicted in Figure 6.3, once requests are submitted to OrchestRAN, the Orchestration Engine selects the most suitable models from the ML/AI Catalog and the location where they should execute (step I). Then, OrchestRAN embeds the models into containers (e.g., Docker containers of dApps, xApps, rApps) (step II) and dispatches them to the selected nodes (step III). Here, they are fed data from the RAN and execute their functionalities (step IV). The selection of the models and of their

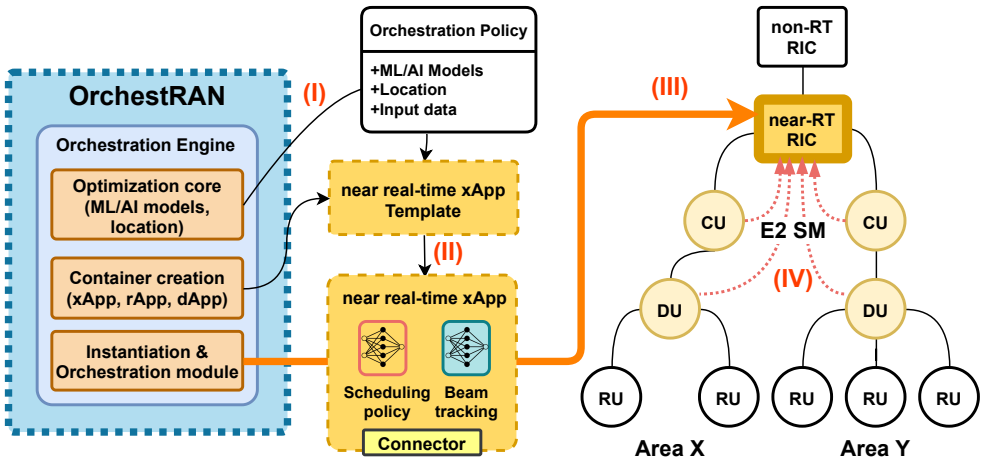


Figure 6.3: An example of creation and dispatchment of an xApp on the near-real-time RIC via OrchestRAN.

optimal execution location is performed by solving the *orchestration problem* discussed in detail in Sections 6.2 and 6.3. This results in an *orchestration policy*, which is converted into a set of O-RAN applications that are dispatched and executed at the designated network nodes, as discussed next.
Container creation, dispatchment and instantiation. To embed models in different O-RAN applications, containers integrate two subsystems, which are automatically compiled from descriptive files upon instantiation. The first is the model itself, and the second is an application-specific *connector*. This is a library that interfaces with the node where the application is running (i.e., with the DU in the case of dApps, near-real-time RIC for xApps, and non-real-time RIC for rApps), collects data from $\mathcal{D}_i^{\text{IN}}$ and sends control commands to nodes in $\mathcal{D}_i$. Once the containers are generated, OrchestRAN dispatches them to the proper endpoints specified in the orchestration policy, where are instantiated and interfaced with the RAN to receive input data. For example, xApps automatically send an E2 subscription request to nodes in $\mathcal{D}_i^{\text{IN}}$, and use custom SMs to interact with them over the E2 interface [402] (see Figure 6.3).

6.2 The Orchestration Problem

Before formulating the orchestration problem, we first discuss important properties of Open RAN systems.

- *Functionality outsourcing.* Any functionality that was originally intended to execute at node d' can be outsourced to any other node $d'' \in \mathcal{D}$ as long as $c_{d',d''} = 1$. As we will discuss next, the node hosting the outsourced model must have access to the required input data, have enough resources to instantiate and execute the outsourced model, and must satisfy performance and timing requirements of the original request.

- *Model sharing.* The limited amount of resources, especially at DUs and RUs, calls for efficient resource allocation strategies. If multiple requests involve the same functionalities on the same group of nodes, an efficient approach consists in deploying a single model that can be shared across all requests.

For the sake of clarity, in Figure 6.4 (left) we show an example where a request can be satisfied by instantiating models m_1 and m_2 on d', and a second one that can be accommodated by instantiating models m_1 and m_3 on d''. Figure 6.4 (right) shows an alternative solution where m_1 (common to

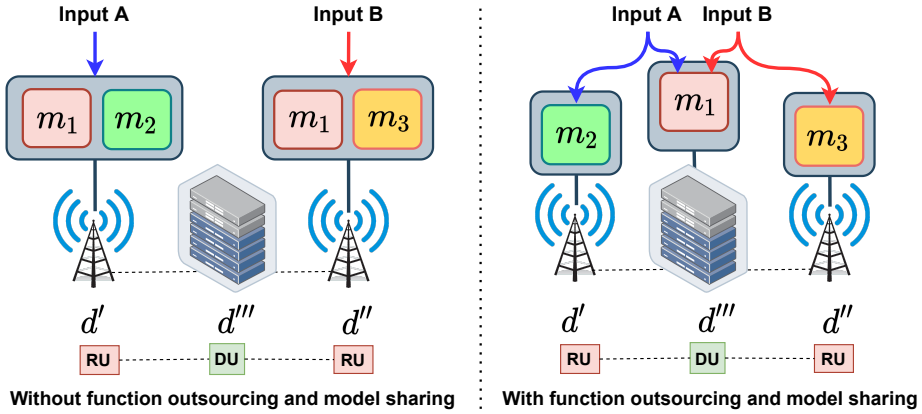


Figure 6.4: Example of function outsourcing and model sharing in Open RAN.

both requests) is *outsourced* to d''' and it is shared between the two requests, with a total of three deployed models, against the four required in Figure 6.4 (left). In the next section, we also discuss the case where model sharing or function outsourcing are nonviable.

6.2.1 Formulating the Orchestration Problem

Let $x^{i,f,d}_{m,k,d'} \in \{0,1\}$ be a binary variable such that $x^{i,f,d}_{m,k,d'} = 1$ if functionality f demanded by request i on node d is provided by instance k of model m instantiated on node d'. In the following, we refer to the variable $\mathbf{x} = (x^{i,f,d}_{m,k,d'})_{i,f,d,m,k,d'}$ as the *orchestration policy*, where $i \in \mathcal{I}, f \in \mathcal{F}, (d,d') \in \mathcal{D}\times\mathcal{D}, m \in \mathcal{M}, k = 1 \dots C_{m,d'}$.

- *Conflict avoidance.* For any tuple (i,f,d) such that $\tau_{i,f,d} = 1$, we assume that OrchestRAN can instantiate at most one model to avoid multiple models controlling the same parameters and/or functionalities. As mentioned earlier, this can be achieved by either instantiating the model at d, or by outsourcing it to another node $d' \neq d$. The above requirement can be formalized as follows:

$$\sum_{m\in\mathcal{M}} \sigma_{m,f} \sum_{d'\in\mathcal{D}} \sum_{k=1}^{C_{m,d'}} c_{d,d'} x^{i,f,d}_{m,k,d'} = y_i \tau_{i,f,d} \tag{6.1}$$

  where $y_i \in \{0,1\}$ indicates whether or not i is satisfied. Specifically, (6.1) ensures that: (i) For any tuple (i,f,d) such that $\tau_{i,f,d} = 1$, function f is provided by one model only, and (ii) $y_i = 1$ (i.e., request i is satisfied) if and only if OrchestRAN deploys models providing all functionalities specified in $\mathcal{F}_i$.

- *Complying with the requirements.* An important aspect of the orchestration problem is guaranteeing that the orchestration policy $\mathbf{x}$ satisfies the minimum performance requirements $\boldsymbol{\pi}_i$ of each request i, and that both data collection and execution procedures do not exceed

the maximum latency constraint $\delta_{i,f,d}$. These requirements are captured by the following constraints.

Quality of models. For each tuple (i, f, d) such that $\tau_{i,f,d} = 1$, NOs can specify a minimum performance level $\pi_{i,f,d}$. This can be enforced via the following constraint

$$\chi_{i,f,d} \sum_{m \in \mathcal{M}} \sum_{d' \in \mathcal{D}} \sum_{k=1}^{C_{m,d'}} c_{d,d'} x_{m,k,d'}^{i,f,d} A_{m,f,d} \geq \chi_{i,f,d} y_i \pi_{i,f,d} \tag{6.2}$$

where $A_{m,f,d} = \beta_{m,f,d}\, \gamma_{m,f}\, \sigma_{m,f}$, and the performance score $\gamma_{m,f}$ is defined in Section 6.1.2. In (6.2), $\chi_{i,f,d} = 1$ if the goal is to guarantee a value of $\gamma_{m,f}$ higher than a minimum performance level $\pi_{i,f,d}$, and $\chi_{i,f,d} = -1$ if the goal is to keep $\gamma_{m,f}$ below a maximum value $\pi_{i,f,d}$.

Control-loop time-scales. Each model m requires a specific type of input t_m^{IN} and, for each tuple (i, f, d), we must ensure that the time needed to collect such input from nodes in $\mathcal{D}_{i,f,d}^{\text{IN}}$ does not exceed $\delta_{i,f,d}$. For each orchestration policy $\mathbf{x}$, the *data collection time* can be formalized as follows:

$$\Delta_{i,f,d}(\mathbf{x}) = \sum_{m \in \mathcal{M}} \sigma_{m,f} \sum_{d' \in \mathcal{D}} \sum_{k=1}^{C_{md'}} x_{m,k,d'}^{i,f,d} \sum_{d'' \in \mathcal{D}_{i,f,d}^{\text{IN}}} c_{d',d''} \Theta_{m,d',d''}^{i,f,d} \tag{6.3}$$

where $\Theta_{m,d',d''}^{i,f,d} = \left(\frac{s_{t_m^{\text{IN}}}}{b_{d',d''} |\mathcal{D}_{i,f,d}^{\text{IN}}|} + T_{d',d''} \right)$, $s_{t_m^{\text{IN}}}$ is the input size of model m measured in bytes, $b_{d',d''}$ is the data rate of the link between nodes d'' and d', and $T_{d',d''}$ represents the propagation delay between nodes d' and d''. Let T_m^{exec} be the time to execute model m on node d'. For any tuple (i, f, d), the *execution time* under orchestration policy $\mathbf{x}$ is

$$\Delta_{i,f,d}^{EXEC}(\mathbf{x}) = \sum_{m \in \mathcal{M}} \sigma_{m,f} \sum_{d' \in \mathcal{D}} T_{m,d'}^{exec} \sum_{k=1}^{C_{m,d'}} x_{m,k,d'}^{i,f,d} \tag{6.4}$$

By combining (6.3) and (6.4), any orchestration policy $\mathbf{x}$ must satisfy the following constraint for all (i, f, d) tuples:

$$\Delta_{i,f,d}(\mathbf{x}) + \Delta_{i,f,d}^{EXEC}(\mathbf{x}) \leq \delta_{i,f,d}\, \tau_{i,f,d} \tag{6.5}$$

- *Avoiding resource over-provisioning.* We must guarantee that the resources consumed by the O-RAN applications do not exceed the resources ρ_d^ξ of type ξ available at each node (i.e., ρ_d^ξ). For each $d \in \mathcal{D}$ and $\xi \in \Xi$, we have

$$\sum_{m \in \mathcal{M}} \rho_m^\xi \sum_{k=1}^{C_{md}} z_{m,k,d} \leq \rho_d^\xi \tag{6.6}$$

where $z_{m,k,d} \in \{0, 1\}$ indicates whether instance k of model m is associated to at least one model on node d. Specifically, let

$$n_{m,k,d} = \sum_{i \in \mathcal{I}} \sum_{f \in \mathcal{F}_i} \sum_{d' \in \mathcal{D}} x_{m,k,d}^{i,f,d'} \tag{6.7}$$

be the number of tuples (i, f, d') assigned to instance k of model m on node d ($n_{m,k,d} > 1$ implies that m is shared). Notice that (6.6) and (6.7) are coupled one to another as $z_{m,k,d} = 1$ if and only if $n_{m,k,d} > 0$. This conditional relationship can be formulated by using the following big-M formulation [403]

$$n_{m,k,d} \geq 1 - M(1 - z_{m,k,d}) \tag{6.8}$$
$$n_{m,k,d} \leq M z_{m,k,d} \tag{6.9}$$

where $M \in \mathbb{R}$ is a real-valued number whose value is larger than the maximum value of $n_{m,k,d}$, i.e., $M > IFD$ [403].

- *Problem formulation.* For any request i, let $v_i \geq 0$ represent its value. The goal of OrchestRAN is to compute an orchestration policy $\mathbf{x}$ maximizing the total value of requests being accommodated by selecting (i) which requests can be accommodated; (ii) which models should be instantiated; and (iii) where they should be executed to satisfy request performance and timescale requirements. This can be formulated as

$$\max_{\mathbf{x},\mathbf{y},\mathbf{z}} \quad \sum_{i \in \mathcal{I}} y_i v_i \tag{6.10}$$
$$\text{subject to} \quad \text{Constraints (6.1), (6.2), (6.5), (6.6), (6.8), (6.9)}$$
$$x_{m,k,d'}^{i,f,d} \in \{0, 1\} \tag{6.11}$$
$$y_i \in \{0, 1\} \tag{6.12}$$
$$z_{m,k,d} \in \{0, 1\} \tag{6.13}$$

where $\mathbf{x}$ is the orchestration policy, $\mathbf{y} = (y_i)_{i \in \mathcal{I}}$ and $\mathbf{z} = (z_{m,k,d})_{m \in \mathcal{M}, k=1,\ldots,C_{m,d}, d \in \mathcal{D}}$. A particularly relevant case is that where $v_i = 1$ for all $i \in \mathcal{I}$, i.e., the goal of OrchestRAN is to maximize the number of satisfied requests.

- *Disabling model sharing.* Indeed, model sharing allows a more efficient use of the available resources. However, out of privacy and business concerns, NOs might not be willing to share O-RAN applications. In this case, model sharing can be disabled in OrchestRAN by guaranteeing that a model is assigned to one request only. This is achieved by adding the following constraint for any $m \in \mathcal{M}$, $d' \in \mathcal{D}$ and $k = 1, .., C_{m,d'}$

$$\sum_{i \in \mathcal{I}} \sum_{d \in \mathcal{D}} \sum_{f \in \mathcal{F}_{i,d}} x_{m,k,d'}^{i,f,d} \leq 1 \tag{6.14}$$

6.3 Solving the Orchestration Problem

BILP problems such as Problem (6.10) can be optimally solved via Branch-and-Bound (B&B) techniques [404], readily available within well-established numerical solvers, e.g., CPLEX, MATLAB, Gurobi. However, due to the extremely large number N_{OPT} of optimization variables, these solvers might still fail to compute an optimal solution in a reasonable amount of time, especially in large-scale deployments. Indeed, $N_{\mathrm{OPT}} = |\mathbf{x}|+|\mathbf{y}|+|\mathbf{z}| \approx |\mathbf{x}|$, where $|\mathbf{x}| = \mathcal{O}(IFD^2MC_{\max})$, $|\mathbf{y}| = \mathcal{O}(I)$, $|\mathbf{z}| = \mathcal{O}(MDC_{\max})$, and $C_{\max} = \max_{m\in\mathcal{M},d\in\mathcal{D}}\{C_{m,d}\}$. For example, a deployment with $D = 20$, $M = 13$, $I = 10$, $F = 7$ and $C_{\max} = 3$ involves $\approx 10^6$ optimization variables.

6.3.1 Combating Dimensionality via Variable Reduction

To mitigate the "curse of dimensionality" of the orchestration problem, we have developed two preprocessing algorithms to reduce the complexity of Problem (6.10) while guaranteeing the optimality of the computed solutions. We leverage a technique called *variable reduction* [405]. This exploits the fact that, due to constraints and structural properties of the problem, there might exist a subset of *inactive* variables whose value is always zero. These variables do not participate in the optimization process, yet they increase its complexity. To identify those variables, we have designed the following two techniques.

- *Function-aware Pruning (FP).* It identifies the set of inactive variables $\mathbf{x}_{-}^{\mathrm{FP}} = \{x_{m,k,d'}^{i,f,d} : \tau_{i,f,d} = 0 \vee \sigma_{m,f} = 0, \forall i \in \mathcal{I}, f \in \mathcal{F}, (d,d') \in \mathcal{D}\times\mathcal{D}, m \in \mathcal{M}, k = 1,\dots,C_{,m,d}\}$, which contains all the $x_{m,k,d}^{i,f,d}$ variables such that either (i) $\tau_{i,f,d} = 0$, i.e., request i does not require function f at node d, or (ii) $\sigma_{m,f} = 0$, i.e., model m does not offer function f;

- *Architecture-aware Pruning (AP).* This procedure identities those variables whose activation results in instantiating a model on a node that cannot receive input data from nodes in $\mathcal{D}_{i,f,d}^{\mathrm{IN}}$. Indeed, for a given tuple (i,f,d) such that $\tau_{i,f,d} = 1$, we cannot instantiate any model on a node d' such that $c_{d,d'} = 0$, i.e., the two nodes are not connected. The set of these inactive variables is defined as $\mathbf{x}_{-}^{\mathrm{AP}} = \{x_{m,k,d'}^{i,f,d} : c_{d,d'} = 0, \forall i \in \mathcal{I}, f \in \mathcal{F}, (d,d') \in \mathcal{D}\times\mathcal{D}, m \in \mathcal{M}, k = 1,\dots,C_{,m,d}\}$.

Once we have identified all inactive variables, Problem (6.10) is cast into a lower-dimensional space where the new set of optimization variables is equal to $\tilde{\mathbf{x}} = \mathbf{x} \setminus \{\mathbf{x}_{-}^{\mathrm{FP}} \cup \mathbf{x}_{-}^{\mathrm{AP}}\}$, which still guarantees the optimality of the solution [405]. The impact of these procedures on the complexity of the orchestration problem will be investigated in Section 6.4.

6.3.2 Graph Tree Branching

Notice that $|\mathbf{x}| = \mathcal{O}(IFD^2MC_{\max})$, i.e., the number of variables of the orchestration problem grows quadratically in the number D of nodes. Since the majority of nodes of the infrastructure are RUs, DUs and CUs, it is reasonable to conclude that these nodes are the major source of complexity. Moreover, O-RAN systems operate following a cluster-based approach where each near-real-time RIC controls a subset of CUs, DUs and RUs of the network only, i.e., a *cluster*, which have none (or limited) interactions with nodes from other clusters.

These two intuitions are the rationale behind the low-complexity and scalable solution proposed in this section, which consists in splitting the infrastructure tree into smaller subtrees—each operating as an individual cluster—and creating sub-instances of the orchestration problem that only accounts for requests and nodes regarding the considered subtree. The main steps of this algorithm are:

- *Step I*: Let C be the number of near-real-time RICs in the non-real-time RIC domain. For each cluster c, the c-th subtree $\mathcal{D}_c \subseteq \mathcal{D}$ is defined such that $\mathcal{D} = \bigcup_{c=1}^{C} \mathcal{D}_c$ and $\bigcap_{c=1}^{C} \mathcal{D}_c = d^{\text{root}}$, with d^{root} being the non-real-time RIC. A variable $\alpha_{d,c} \in \{0, 1\}$ is used to determine whether a node $d \in \mathcal{D}$ belongs to cluster c (i.e., $\alpha_{d,c} = 1$) or not (i.e., $\alpha_{d,c} = 0$). Since, $\bigcap_{c=1}^{C} \mathcal{D}_c = d^{\text{root}}$, we have that $\sum_{c=1}^{C} \alpha_{d,c} = 1$ for any $d \in \mathcal{D} \setminus \{d^{\text{root}}\}$;
- *Step II*: For each subtree $\mathcal{D}_c$ we identify the subset $\mathcal{I}_c \subseteq \mathcal{I}$ such that $\mathcal{I}_c = \{i \in \mathcal{I} : \sum_{f \in \mathcal{F}} \sum_{d \in \mathcal{D}_c} \tau_{i,f,d} \geq 1\}$ contains all the requests that involve nodes belonging to cluster c only;
- *Step III*: We solve Problem (6.10) via B&B considering only requests in $\mathcal{I}_c$ and nodes in $\mathcal{D}_c$. The solution is a tuple $(\mathbf{x}_c, \mathbf{y}_c, \mathbf{z}_c)$ specifying which models are instantiated and where ($\mathbf{x}_c$), which requests are satisfied in cluster c ($\mathbf{y}_c$) and what instances of the models are instantiated on each node of $\mathcal{D}_c$ ($\mathbf{z}_c$).

Remark. This branching procedure might compute solutions with *partially satisfied requests*. These are requests that are accommodated on a subset of clusters only, which violates Constraint 6.1. However, as we will show in Section 6.4, this procedure is scalable as each subtree $\mathcal{D}_c$ involves a limited number of nodes only, and we can solve each lower-dimensional instance of Problem (6.10) in parallel and in less than 0.1 s.

6.4 Numerical Evaluation

To evaluate the performance of OrchestRAN in large-scale scenarios, we have developed a simulation tool in MATLAB that uses CPLEX to execute optimization routines. For each simulation, NOs submit $R = 20$ randomly generated requests, each specifying multiple sets of functionalities and nodes, as well as the desired timescale. Unless otherwise stated, we consider a single-domain deployment with 1 non-real-time RIC, 4 near-real-time RICs, 10 CUs, 30 DUs and 90 RUs. For each simulation, the number of network nodes is fixed, but the tree structure of the infrastructure is randomly generated. We consider the three cases shown in Table 6.1, where we limit the type of nodes that can be included in each request. Similarly, we also consider the three cases in Table 6.2.

Table 6.1: Controllable nodes.

Case / Requested Nodes	Non-real-time RIC	Near-real-time RIC	CU	DU	RU
All nodes (ALL)	✓	✓	✓	✓	✓
Edge and RAN (ER)	✗	✓	✓	✓	✓
RAN only (RO)	✗	✗	✓	✓	✓

For each case, we specify the probability that the latency requirement $\delta_{i,f,d}$ for each tuple (i, f, d) is associated to a specific timescale. The combination of these 6 cases covers relevant Open RAN applications.

Table 6.2: Request timescale cases and probabilities.

Case / Time scale	TTI-level (≤ 0.01 s)	Sub-second (≤ 1 s)	Long (> 1 s)
Delay-Tolerant (DT)	0.2	0.2	0.6
Low Latency (LL)	0.2	0.6	0.2
Ultra-Low Latency (ULL)	0.6	0.4	0

The ML/AI Catalog consists of $M = 13$ models that provide $F = 7$ different functionalities. Ten models use metrics from the RAN (e.g., throughput and buffer measurements) as input, while the remaining three models are fed with IQ samples from RUs. The input size $s_{t_m^{\text{IN}}}$ is set to 100 and 1000 bytes for the metrics and IQ samples, respectively. For the sake of illustration, we assume that $\beta_{m,f,d} = \sigma_{m,f}$, $\pi_{i,f,d} = \tau_{i,f,d}$ and $C_{m,d} = 3$ for all $m \in \mathcal{M}$, $i \in \mathcal{F}$, $f \in \mathcal{F}$ and $d \in \mathcal{D}$. The execution time of each model is equal across all models and nodes and set to $T^{exec}_{m,d} = 1$ ms. The available bandwidth $b_{d,d'}$ is 100 Gbps between non-real-time RIC and near-real-time RIC, 50 Gbps between near-real-time RICs and CUs, 25 Gbps between CUs and DUs, and 20 Gbps between DUs and RUs, while the propagation delay $T_{d,d'}$ is set to $[10, 10, 5, 1]$ ms, respectively. The resources ρ_d available at each node are represented by the number of available CPU cores, and we assume that each model requests one core only, i.e., $\rho_m = 1$. The number of cores available at non-real-time RICs, near-real-time RICs, CUs, DUs and RUs are 128, 8, 4, 2, and 1, respectively. Results presented in this section are averaged over 100 independent simulation runs.

Computational complexity. Figure 6.5 shows the number of optimization variables and computation time of our algorithms with varying network size. At each simulation run, we consider a single

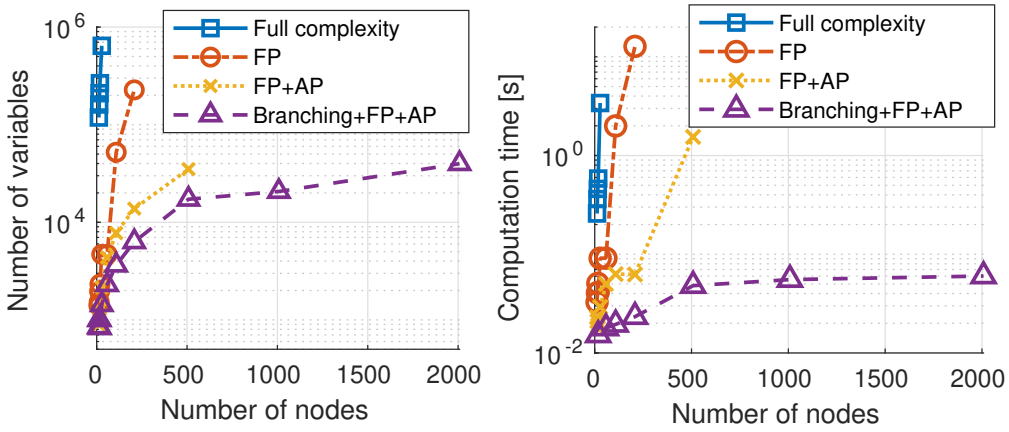


Figure 6.5: Number of variables and computation time for different network size.

non-real-time RIC and a randomly generated tree graph that matches the considered size. As expected, the number of variables and the complexity increase with larger networks. This can be mitigated by using our FP and AP pre-processing algorithms, which reduce the number of optimization variables while ensuring the optimality of the computed solution. Their combination allows computation of optimal solutions in 0.1 s and 2 s for networks with 200 and 500 nodes, respectively. Figure 6.5 also shows the benefits of branching the optimization problem into sub-problems of smaller size (Section 6.3.2). Although the branching procedure might produce partially satisfied requests, it

results in a computation time lower than 0.1 s even for instances with 2000 nodes, providing a fast and scalable solution for large-scale applications.

Acceptance ratio. Figure 6.6 (left) shows the acceptance ratio for different cases and algorithms. The number of accepted requests decreases when moving from loose timing requirements (i.e.,

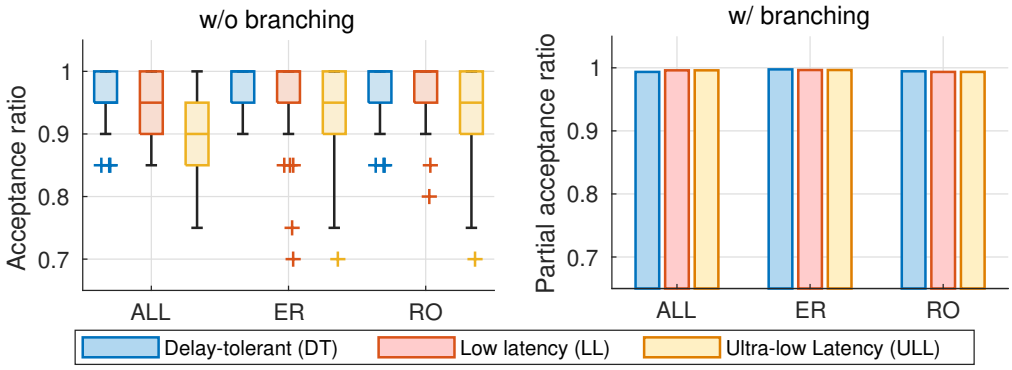


Figure 6.6: (Left) Ratio of accepted requests w/ model sharing but w/o branching; (Right) Ratio of partially accepted requests w/ model sharing and branching.

Delay-tolerant (DT)), to tighter ones (i.e., Low Latency (LL) and Ultra-Low Latency (ULL)). For example, while 95% of requests are satisfied on average for the DT configuration, we observe ULL instances in which only 70% of requests are accepted. Indeed, Transmission Time Interval (TTI)-level services may only be possible at the DUs/RUs which, however, have limited resources and cannot support the execution of many concurrent O-RAN applications. In Figure 6.6 (right), we show the probability that a request is partially accepted when considering the branching algorithm. Specifically, it shows that branching results in ≈99% of requests being partially satisfied on one subtree or more. This means that in the case where not enough resources are available to accept the entirety of the request, OrchestRAN can satisfy portions of it. Thus, requests that would be otherwise rejected can be at least partially accommodated.

Advantages of model sharing. Figure 6.7 shows the resource utilization with and without model sharing (left) and the corresponding resource utilization saving (right). As expected, model sharing always results in lower resource utilization and uses 2× less resources than the case without model sharing. Figure 6.8 shows the acceptance ratio when model sharing is disabled, and by comparing it with Figure 6.6 (left)—where model sharing is enabled—we notice that model sharing also increases the acceptance ratio. Specifically, model sharing accommodates at least 90% of requests in all cases, while this number drops to ≈70% when model sharing is disabled.

To better understand how OrchestRAN orchestrates intelligence, Figure 6.9 shows the distribution of models across the different network nodes for the ER case (see Table 6.1) with different timing constraints. Requests with loose timing requirements (DT) result in ≈45% of models being allocated in the RICs. Instead, stringent timing constraints (LL and ULL) result in ≈70% of models being instantiated at CUs, DUs, and RUs.

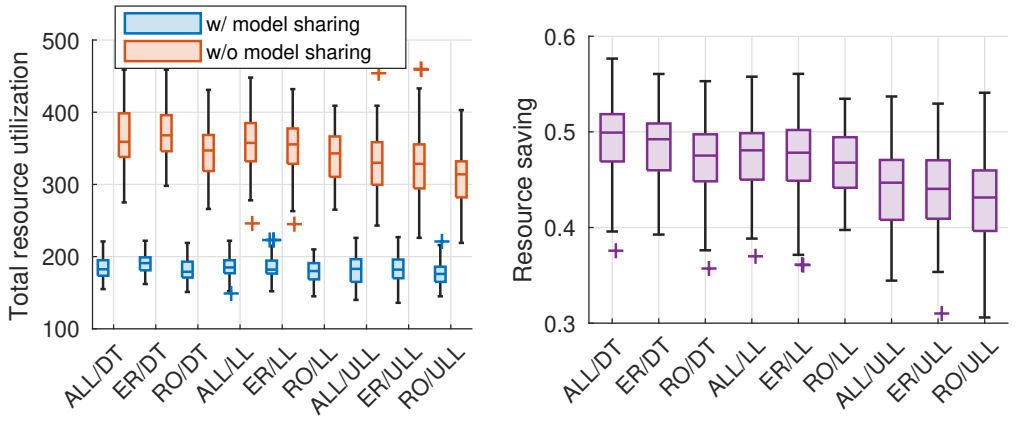


Figure 6.7: Resource utilization and saving with and without model sharing.

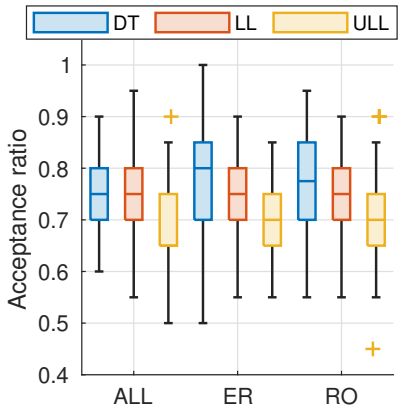


Figure 6.8: Percentage of accepted requests w/o model sharing for different cases.

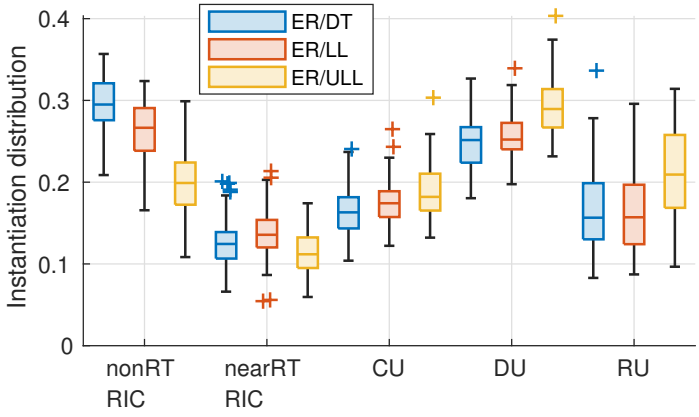


Figure 6.9: Distribution of model instantiation for different cases.

6.5 Prototype and Experimental Evaluation

To demonstrate the effectiveness of OrchestRAN, we leveraged ColO-RAN [528], an O-RAN-compliant large-scale experimental platform developed on top of the Colosseum wireless network emulator [545]. Colosseum includes 128 computing servers (i.e., SRNs), each controlling a USRP X310 SDR, and a MCHEM emulating wireless channels between the SRNs to reproduce realistic and time-varying wireless characteristics (e.g., path-loss, multi-path) under different deployments (e.g., urban, rural, etc.) [545]. We leverage the publicly available tool SCOPE [542] to instantiate a softwarized cellular network with 7 base stations and 42 UEs (6 UEs per base station) on the Colosseum city-scale downtown Rome scenario, and to interface the base stations with the O-RAN near-real-time RIC through the E2 interface. SCOPE, which is based on srsRAN [27], implements open APIs to reconfigure the base station parameters (e.g., slicing resources, scheduling policies, etc.) from O-RAN applications through closed-control loops, and to automatically generate datasets from RAN

statistics (e.g., throughput, buffer size, etc.). Users are deployed randomly and generate traffic belonging to 3 different network slices configured as follows: (i) slice 0 is allocated an eMBB service, in which each UE requests 4 Mbps constant bitrate traffic; (ii) slice 1 a MTC service, in which each UE requests Poisson-distributed traffic with an average rate of 45 kbps, and (iii) slice 2 to a URLLC service, in which each UE requests Poisson-distributed traffic with an average rate of 90 kbps. We assume 2 UEs per slice, whose traffic is handled by the base stations, which use a 10 MHz channel bandwidth with 50 PRB.

The high-level architecture of the OrchestRAN prototype on Colosseum is shown in Figure 6.10. OrchestRAN runs in an LXC embedding the components of Figure 6.2. For each experiment,

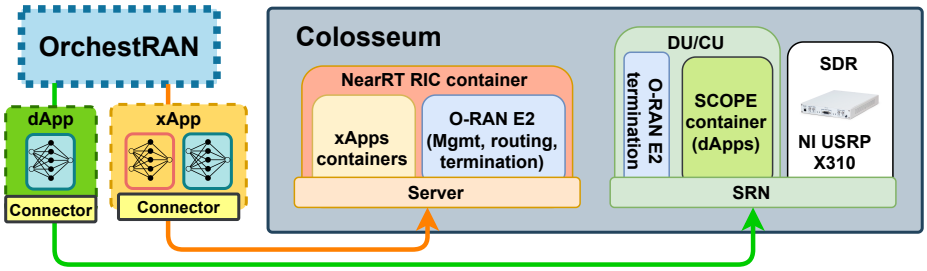


Figure 6.10: OrchestRAN prototype architecture on Colosseum and integration with O-RAN and SCOPE components.

we randomly generate a new set of control requests every 4 minutes. The Orchestration Engine computes the optimal orchestration policy and embeds the models within O-RAN applications that are dispatched to the nodes where they are executed. We consider the case where models can run at the near-real-time RIC (as xApps) or at the DU (as dApps via SCOPE).

We used SCOPE to generate datasets on Colosseum and train 4 ML models that constitute our ML/AI Catalog. Models $M1$ and $M2$ have been trained to forecast throughput and transmission buffer size. Models $M3$ and $M4$ control the parameters of the network to maximize different rewards through PPO-based DRL agents (see Table 6.3). Specifically, $M3$ consists of three DRL agents, each

Table 6.3: DRL agents in the ML/AI catalog.

Model	Reward Slice 0	Reward Slice 1	Reward Slice 2	Action
M3	max(throughput)	max(TX packets)	max(PRB ratio)	Scheduling
M4	max(throughput)	max(TX packets)	min(buffer size)	Scheduling, RAN slicing

making decisions on the scheduling policies of one slice only. The three agents aim at maximizing the throughput of slice 0, the number of transmitted packets of slice 1, and the ratio between the allocated and requested PRBs (i.e., the *PRB ratio* which takes values in $[0, 1]$) of slice 2, respectively. Model $M4$, instead, consists of a single DRL agent controlling the scheduling and RAN slicing policies (i.e., how many PRBs are assigned to each slice) to *jointly* maximize the throughput of slice 0 and the number of transmitted packets of slice 1, and to minimize the buffer size of slice 2. Each model requires one CPU core, and we consider three configurations: (i) "RIC only", in which models can be executed via xApps at the near-real-time RIC only; (ii) "RIC + lightweight DU", in which DUs have 2 cores each to execute up to two dApps concurrently; and (iii) "RIC + powerful

DU", in which DUs are equipped with 8 cores. In all cases, the near-real-time RIC has access to 50 cores. Overall, we ran more than 95 hours of experiments on Colosseum.

Experimental results. Figure 6.11 (left) shows the probability that models are executed at the near-real-time RIC for different configurations and number of requests. As expected, in the "RIC

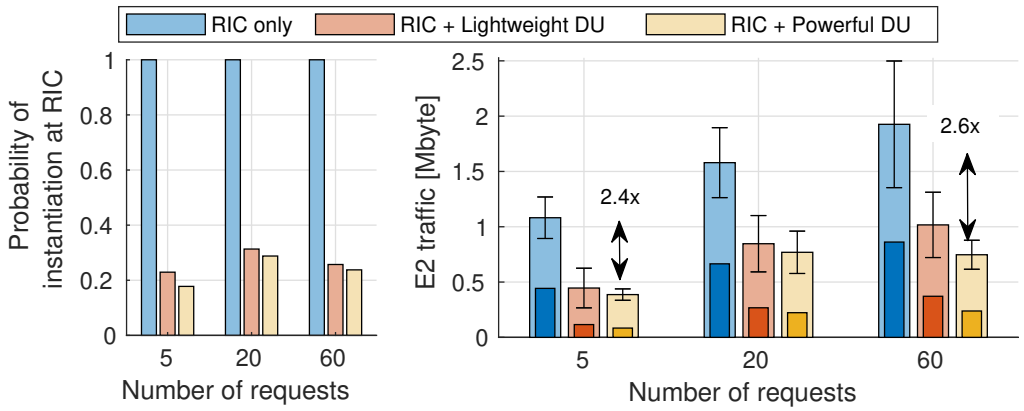


Figure 6.11: (Left) Probability of instantiating O-RAN applications at the near-real-time RIC; (Right) Traffic over O-RAN E2 interface for different configurations. Dark bars represent traffic related to payload only.

only" case, all models execute as xApps at the near-real-time RIC, while both "RIC + lightweight DU" and "RIC + powerful DU" cases result in ≈25% of models executing at the RIC. The remaining 75% of the models are executed as dApps at the DUs. Figure 6.11 (right) shows the traffic in Mbyte over the E2 interface between the near-real-time RIC and the DUs for the different configurations. This includes messages to set up the initial subscription between the near-real-time RIC and the DUs, messages to report metrics from the DUs to the RIC (e.g., throughput, buffer size), and control messages from the RIC to the DUs (e.g., to update scheduling and RAN slicing policies). Results clearly show that ≈40% of the E2 traffic transports payload information (dark bars), while the remaining 60% consists of overhead data. Although the initial subscription messages exchanged between the near-real-time RIC and the DUs are sent in all considered cases, running models as dApps at the DUs still results in up to 2.6× less E2 traffic if compared to the "RIC only" case.

Finally, we showcase the impact of the real-time execution of OrchestRAN on the network performance. We focus on DU 7, and in Figure 6.12 (top) we show the location and time instant at which OrchestRAN instantiates the four models on the near-real-time RIC and on DU 7 for a single experiment. The impact on the network performance of the different orchestration policies is shown in Figure 6.12 (center and bottom). Since $M1$ and $M2$ perform forecasting tasks only, the figure only reports the evolution of the metrics used to reward the DRL agents $M3$ and $M4$ (see Table 6.3) for different slices. We notice that OrchestRAN allows the seamless instantiation of dApps and xApps, controlling the same DU without causing any service interruptions. Moreover, although $M3$ and $M4$ share the same reward for slices 0 and 1, $M4$ can also make decisions on the network slicing policies. Thus, it provides a higher throughput for slice 0 (≈10% higher than $M3$), and a higher number of transmitted packets for slice 1 (≈2× higher than $M3$) (Figure 6.12 (center)). Similarly, in the case of slice 2, $M3$ aims at maximizing the PRB ratio, while $M4$ at minimizing the size of the transmission buffer, which results in $M3$ and $M4$ computing different control policies for slice 2. As

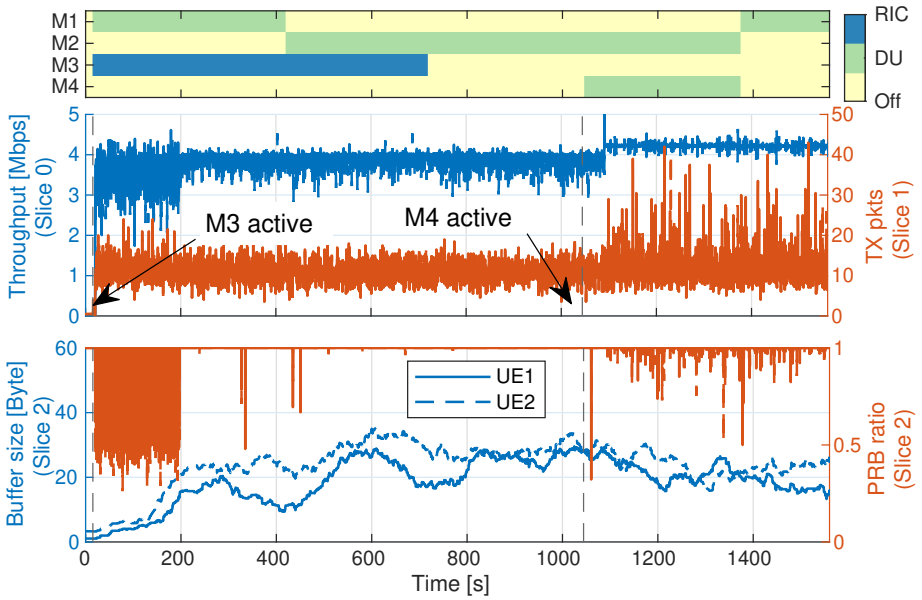


Figure 6.12: (Top) Dynamic activation of O-RAN applications at near-real-time RIC and DU 7; (Center and bottom) Performance comparison for different deployments of O-RAN applications and network slices. Solid lines and dashed lines refer to traffic for $\mathrm{UE}_{1,i}$ and $\mathrm{UE}_{2,i}$ of Slice i.

shown Figure 6.12 (bottom), although $M3$ converges to a stable control policy that results in a PRB ratio $\approx$1, its buffer size is higher than that of $M4$. Conversely, the buffer size of slice 2 decreases once $M4$ is instantiated with a decrease in the PRB ratio.

6.6 Related Work

The application of ML/AI algorithms to cellular networks is gaining momentum as a promising and effective way to design and deploy solutions capable of predicting, controlling, and automating the network behavior under dynamic conditions. Relevant examples include the application of Deep Learning and DRL to predict the network load [80, 307, 395], classify traffic [397, 406, 407], perform beam alignment [340, 398], allocate radio resources [356, 392, 542], and deploy service-tailored network slices [353, 393–396, 408, 526]. It is clear that ML/AI techniques will play a key role in the transition to intelligent networks, especially in the O-RAN ecosystem [261]. However, a relevant challenge that still remains unsolved is how to bring such intelligence to the network in an efficient, reliable and automated way, which is ultimately the goal of this chapter.

In [409], Ayala-Romero et al. present an online Bayesian learning orchestration framework for intelligent virtualized RANs where resource allocation follow channel conditions and network load. The same authors present a similar framework in [357], where networking and computational resources are orchestrated via DRL to comply with SLAs while accounting for the limited amount of resources. Singh et al. present GreenRAN, an energy-efficient orchestration framework for NextG that splits and allocates RAN components according to the current resource availability [410]. In [411], Chatterjee et al. present a radio resource orchestration framework for 5G applications where network

slices are dynamically re-assigned to avoid inefficiencies and SLA violations. Relevant to our work are the works of Morais et al. [412] and Matoussi et al. [413], which present frameworks to optimally disaggregate, place and orchestrate RAN components in the network to minimize computation and energy consumption while accounting for diverse latency and performance requirements. Although the above works all present orchestration frameworks for NextG systems, they are focused on orchestrating RAN resources and functionalities, rather than network intelligence, which represents a substantially different problem.

In the context of orchestrating ML/AI models in NextG systems, Baranda et al. [414, 415] present an architecture for the automated deployment of models in the 5Growth MANO platform [416], and demonstrate automated instantiation of models on demand. The closest to our work is the work of Salem et al. [401], which proposes an orchestrator to select and instantiate inference models at different locations of the network to obtain a desirable balance between accuracy and latency. However, [401] is not concerned with O-RAN systems, but focuses on data-driven solutions for inference in cloud-based applications.

Besides the differences highlighted in the previous discussion, OrchestRAN differs from the above works in that it focuses on the Open RAN architecture and is designed to instantiate *both* inference and control solutions complying with O-RAN specifications. Moreover, OrchestRAN allows model sharing across multiple requests to efficiently reuse available network resources. We prototyped and benchmarked OrchestRAN on Colosseum. To the best of our knowledge, this is the first large-scale demonstration of a network intelligence orchestration system tailored to O-RAN architecture and networks.

6.7 Conclusions

In this chapter, we presented OrchestRAN, a novel network intelligence orchestration framework for Open RAN systems. OrchestRAN is based upon O-RAN specifications and leverages the RIC xApps and rApps and O-RAN open interfaces to provide NOs with an automated orchestration tool for deploying data-driven inference and control solutions with diverse timing requirements. OrchestRAN has been equipped with orchestration algorithms with different optimality/complexity trade-offs to support non-real-time, near-real-time and real-time applications. We assessed OrchestRAN performance and presented an O-RAN-compliant prototype by instantiating a cellular network with 7 base stations and 42 UEs on the Colosseum network emulator. Our experimental results demonstrate that OrchestRAN achieves seamless instantiation of O-RAN applications at different network nodes and time scales, and reduces the message overhead over the O-RAN E2 interface by up to $2.6\times$ when instantiating intelligence at the edge of the network.

Chapter 7

Private Communications in Softwarized Cellular Networks

The *softwarization* of the RAN is being heralded as the core of NextG cellular networks [63, 388, 389, 417, 521, 523, 526]. Enabling virtualization technologies, softwarization will allow IPs to create *virtual networks* on top of their physical infrastructure, each assigned to a different infrastructure *slice* [12, 13, 537]. This fundamental innovation will concretely realize the long-standing vision of *Cellular Connectivity-as-a-Service (CCaaS)*, where the IP assigns physical resources (e.g., spectrum, power, base stations, etc.) to each MVNO according to their requirements [374, 418]. CCaaS is envisioned to provide unparalleled levels of Quality of Experience (QoE) to mobile users, as well as usher in new business opportunities between IPs and MVNOs [419, 420].

In this chapter, which was published as [540], we leverage RAN softwarization and network slicing to concretely realize *Private Cellular Connectivity-as-a-Service (PCCaaS)*, pushing the CCaaS innovation to the realm of *private* networking. Through PCCaaS the IPs can instantiate and deploy *private network slices* sharing the virtualized infrastructure with other (public) slices. In this chapter we use the word *private* to identify slices whose existence is known only to selected users that can exchange sensitive data embedding it *covertly and undetectably* into *primary traffic*, used as decoy.

The opportunities and applications of PCCaaS are multifold. For instance, with PCCaaS law enforcement agencies could leverage the ubiquitous connectivity offered by extant cellular infrastructure and use private slices to establish undetectable communications with undercover agents in the field. Similarly, law enforcement could deploy tiny IoT devices as "bugs," collecting audio and video content and communicating it covertly. Such devices would pose as regular IoT sensors and conceal sensitive covert information on top of innocuous primary traffic, e.g., temperature readings. An example of PCCaaS is shown in Figure 7.1.

The IP instantiates three slices whose profile is communicated to the cellular base station. Two slices are public (in red). These slices are for public users of the infrastructures (e.g., cellular subscribers). The third slice is private. In this slice, private users exchange data that is of non-sensitive nature (decoy traffic, in blue). This is their primary traffic. They also exchange *covert* traffic (in orange), which is hidden into the primary data. The challenge of PCCaaS is that of fooling a malicious eavesdropper to believe that the private users are only exchanging decoy traffic, namely, in

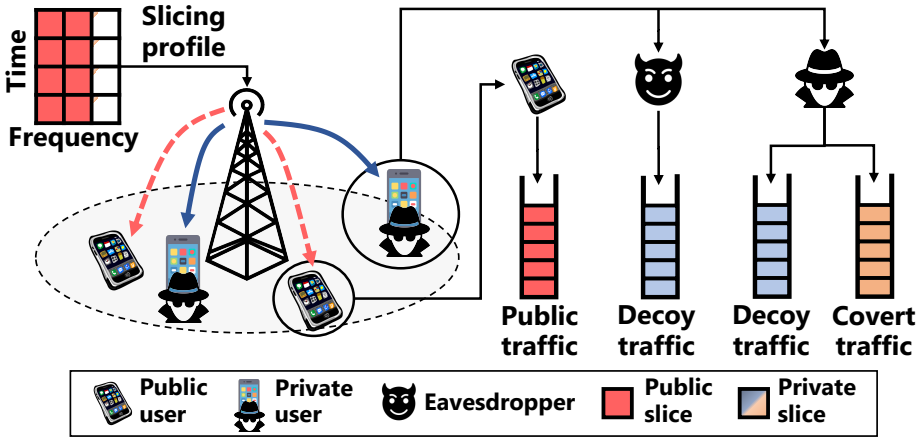


Figure 7.1: Private cellular connectivity-as-a-service.

allowing the eavesdropper to capture only their primary traffic.

Clearly, no form of effective data encryption is the solution to realizing the vision of PCCaaS. First of all, not all devices have the necessary resources to support the execution of power-hungry and computationally complex encryption algorithms. The IoT scenario described above is a typical example. Furthermore, encrypted traffic is still subject to jamming: An adversarial user that is capable to detect the transmission of sensitive information could prevent its intended recipient to receive it. The key question is therefore how to ensure that communication of sensitive data is not only secure, but also *undetectable*, independently of encryption.

One of the key challenges in realizing PCCaaS is that data transmitted over wireless channels cannot be easily hidden. To address this problem, *wireless steganography* directly operates on RF waveforms by applying "hand-crafted" tiny displacements to the I/Q symbols being transmitted, also known as *primary* symbols [421–428]. While a steganographic receiver (the private user of Figure 7.1) can decode the covert information by translating the "dirty" I/Q symbols to a corresponding covert bit sequence, public users would be able to decode primary symbols only.

Figure 7.2 illustrates a practical example of wireless steganography where covert data are embedded into a QPSK-modulated signal.

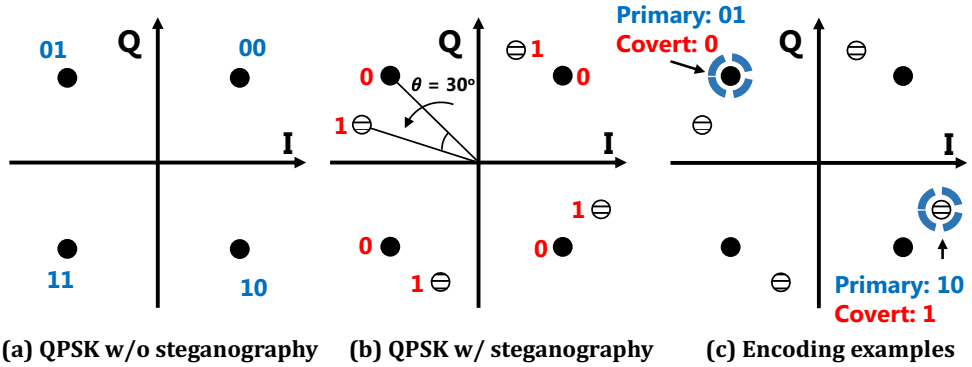


Figure 7.2: Wireless steganography over a QPSK modulation.

Specifically, Figure 7.2a shows the set of QPSK symbols used to form the primary message. Let us assume that the transmitter sends the primary symbol "01" to deceive adversaries, but at the same time it wants to embed a covert message in it through wireless steganography. To achieve this, the transmitter can rotate the phase of the I/Q symbols of Figure 7.2a by an angle θ to send the covert bit "1" (Figure 7.2b), while symbols with no rotations correspond to the bit "0" (Figure 7.2c) [429]. (More sophisticated schemes are described in Section 7.1.2.)

Despite recent advances, wireless steganography has not yet found widespread application in networking. We believe this is because existing approaches operate only at the physical layer, which is insufficient to make PCCaaS systems possible.

In this chapter we leverage steganography as the core of a ready-to-use *full-stack* approach to PCCaaS-based networking. Our system, that for testing purposes has been realized on open-source LTE implementations, is called SteaLTE to indicate the *stealthy*, private nature of the networking it enables to satisfy PCCaaS requirements. SteaLTE achieves:

- *End-to-end Reliability and Security*. We design a *full-stack steganographic system* leveraging proven reliable data transfer techniques. These are integrated to a *steganographic mutual authentication* mechanism where legitimate parties authenticate each other before exchanging confidential information.

- *Adaptive Traffic Embedding*. Covert data need to be embedded over *primary* traffic, which is inherently varied and unpredictable. Clearly, a large covert data packet cannot be embedded on a small primary packet, or cannot be transmitted at all in the absence of primary traffic. This requires the process of embedding covert traffic to be *flexible enough* to deal with such unpredictability. To this purpose, SteaLTE features a covert packet generator component that creates and embeds covert packets that seamlessly adapt to primary data traffic, generating "dummy" primary traffic on-demand, if necessary.

- *Standard compliance*. To successfully operate over existing cellular networks, PCCaaS must adhere to standard protocol implementations of 4G/5G systems. SteaLTE has been designed to seamlessly integrate with cellular systems without disrupting primary communications or affecting their performance.

- *Undetectability*. A goal of PCCaaS is to make covert data communications undetectable, concealing them from eavesdroppers and jammers. To this aim, we design a stochastic steganography scheme that embeds covert transmissions by mimicking wireless channel noise. We show that SteaLTE reduces the Kolmogorov–Smirnov (K-S) distance from the "clean" (i.e., without covert data) distribution by 4.8x, improving undetectability with respect to previous solutions [430].

Our LTE-compliant prototype of SteaLTE—the first for PCCaaS-based cellular networking—has been evaluated through experiments over indoor and outdoor testbeds (including the POWDER platform from the PAWR program [5, 6]) on scenarios with varying parameters, including topology, traffic patterns, mobility and link ranges. Our results show that, overall, the SteaLTE covert throughput is comparable to the primary throughput, that it minimally affects primary transmissions imposing $< 6\%$ loss of primary throughput, and that, even in challenging outdoor settings, effectively delivers covert data on links up to 852 ft long.

The rest of chapter is organized as follows. The SteaLTE system design is presented in Section 7.1. Its prototype over LTE-compliant implementations is described in Section 7.2. Section 7.3 reports results from our testbed-based experimental evaluation of SteaLTE. A review of previous work on the topic is surveyed in Section 7.4. Conclusions are drawn in Section 7.5.

7.1 SteaLTE Design

In this section we describe SteaLTE, providing details on its *covert communications* (Section 7.1.1), *transmitter* and *receiver design* (sections 7.1.2 and 7.1.3), and the mechanisms to enable *undetectable covert communications* (Section 7.1.4).

7.1.1 Covert Communications: Formats and Operations

This section describes the packet format and the operations that allow SteaLTE to enable PCCaaS-based reliable and secure covert communications.

7.1.1.1 Packet format

The structure of SteaLTE covert packets is illustrated in Figure 7.3. Each packet consists of three elements: a *header*, a *payload* and the *Cyclic Redundancy Check (CRC)*.

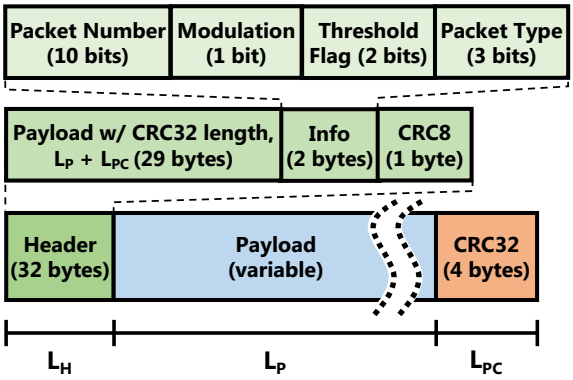


Figure 7.3: SteaLTE covert packet structure.

Table 7.1: SteaLTE covert packet types.

Type	Flag	Description
0	000	ACK
1	001	NACK
2	010	Data
3	011	Source / Destination, Packet Info
4	100	Challenge
5	101	Challenge Response
6	110	Authentication ACK
7	111	Reserved

The **header** consists of 32 bytes carrying information on how to decode a received covert packet. For packet detection and demodulation, the header is modulated through a fixed covert modulation known by the receiver. Its structure is as follows: (i) A 29-byte field with the length of the covert payload (L_P) and the CRC (L_{PC}); (ii) a 2-byte *info* field with information on how to demodulate the covert packet, and (iii) a 1-byte CRC8 field to detect errors on the header. The *info* field contains:

- *Packet Number.* 10 bits uniquely identifying a packet, also used to request packet retransmission (Section 7.1.1.2).
- *Modulation.* a bit flag indicating the modulation used to encode payload and CRC. SteaLTE chooses between two covert modulation schemes depending on the quality of the wireless

channel. This field can be extended to account for additional covert modulation schemes (see also Section 7.1.2.1).

- *Threshold Flag.* 2 bits to instruct the receiver on how to demodulate and decode covert data. This field is paramount for our undetectability scheme (Section 7.1.4).
- *Packet Type.* A 3-bit field to discern among data and control packets. The different packet types and their flags are shown in Table 7.1. Packet types 0 and 1 are `ACK` and `NACK` control packets sent by the receiver to give feedback on the covert transmission (Section 7.1.1.2). Packets carrying covert data are of type 2. Packets of type 3 carry information on source and destination of covert packets and on the total number of packets in the current transmission. Each source and destination address is encoded by 5 bytes containing the corresponding Mobile Subscription Identification Number (MSIN) (i.e., the telephone number commonly used to identify mobile subscribers). Upon receiving an uplink covert transmission, the BS maps the destination MSIN to the corresponding IMSI, which uniquely identifies the UE. It then relays the covert message to the receiver via a downlink covert transmission or forwards it to the SteaLTE BS serving the receiver, as for regular voice traffic). Packets of type 4, 5 and 6 are used for the mutual authentication of covert transmitters and receivers (Section 7.1.1.3). Packet type 7 is reserved for future use.

Payload and CRC32. The variable-size packet payload carries sensitive user data to be transmitted covertly. This field adapts to the size of primary packets to improve the efficiency of covert communications (Section 7.1.2.1). To ease reception, the length of this field is included in the packet header (Figure 7.3). The packet ends with a 4-byte CRC32 field utilized for error detection and to ensure the integrity of covert transmissions.

7.1.1.2 Reliable covert communications

SteaLTE provides built-in reliability through standard reliable data transfer mechanisms. These include error detection, receiver-to-transmitter feedback (positive and negative acknowledgments), packet sequence numbers, timeouts, and retransmissions [431]. Error detection is performed through two Cyclic Redundancy Check (CRC) codes: a CRC8 code is used to protect the header of the packet and a CRC32 code for the packet payload (see Figure 7.3).

7.1.1.3 Mutual Authentication

SteaLTE implements a scheme for the mutual authentication of BSs and UEs through covert challenge/response operations. After standard cellular attachment procedures are completed, the BS sends a randomly-generated *challenge* to the UE using a type 4 packet (Table 7.1). Upon receiving this packet the UE computes the Keyed-Hash Message Authentication Code (HMAC) from the BS challenge and key (which has been pre-shared), and sends the HMAC result as the *challenge response* (packet of type 5). After receiving the response from the UE, the BS compares it with the expected HMAC result. If the two match, the BS considers the UE *authenticated*. To notify the successful end of the UE authentication procedures, the BS sends an *authentication* `ACK` message to the UE (packet of type 6). If the challenge response is not received, the BS retransmits the challenge to the UE. After a certain number of unresponded attempts, or in case of erroneous response, the BS considers the

UE *not authenticated* and will avoid any covert communication with it. When the UE receives the authentication ACK from the BS, it follows a similar procedure to *authenticate* the BS.

7.1.2 Transmitter Design

This section presents the main components of SteaLTE transmitter: the *Covert Packet Generator*, the *Covert Modulator*, and the *Covert Embedder*. In the reminder of the chapter, *orange-colored* blocks with dashed lines denote system components of SteaLTE. All other colors identify standard cellular components that do not require hardware or software modifications.

Figure 7.4 provides a high-level overview of the building blocks of a SteaLTE transmitter.

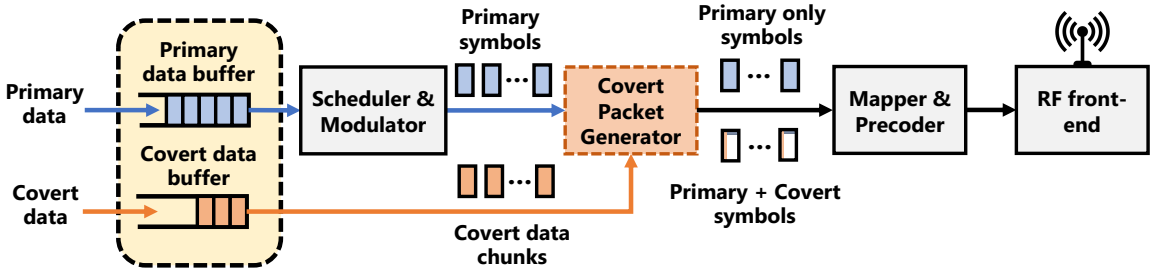


Figure 7.4: High-level SteaLTE transmitter design.

Primary and covert data streams are separate and independent from one another. Primary data are processed through standard scheduling and signal processing procedures (e.g., modulation) of the primary system. These data result in a sequence of primary symbols that are fed to the SteaLTE *covert packet generator* (Section 7.1.2.1). After the covert symbols have been embedded in the primary symbols, they are mapped and precoded according to standard cellular procedures (Section 7.2), and transmitted through the RF front-end.

7.1.2.1 Covert Packet Generator

This block reads covert data from the covert data buffer, and embeds it in the modulated primary symbols. This is achieved by executing the following three steps (see Figure 7.5): (i) verifying that there are enough primary symbols to embed a complete covert packet; (ii) generating covert symbols to be transmitted, and (iii) embedding them into primary ones.

The covert packet generator starts by verifying if the number of primary symbols L_{PS} is large enough to accommodate at least $L_{min} = 36$ bytes, which are required for the covert packet header ($L_H = 32$ bytes) and the CRC32 field ($L_{PC} = 4$ bytes). In the positive, it generates the covert packet payload and CRC32 field. The length of the payload and of the CRC32 are included in the packet header, as described in Section 7.1.1.1. The packet is then modulated through the *covert modulator* according to set *covert modulation parameters* (also in the header). Finally, the resulting covert modulated symbols are embedded in the primary symbols through the *covert embedder*. If $L_{PS} \leq L_{min}$ no covert data are embedded in the primary traffic. Note that the adaptive structure of the covert packets allows to embed variable size covert data on top of *time-varying* and *unpredictable* primary traffic. This feature makes SteaLTE transparent to primary traffic dynamics, thus enabling the integration of SteaLTE with any softwarized cellular system.

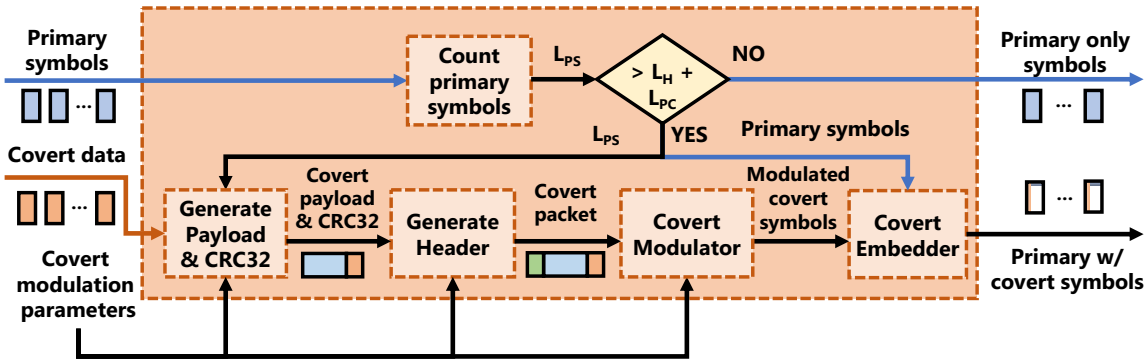


Figure 7.5: Covert packet generator block overview.

Covert Modulator. This block is in charge of encoding covert packets into covert symbols that can be embedded into primary transmissions (Figure 7.5). Several approaches are possible for covert embedding of data through wireless steganography. Figure 7.6 illustrates three examples of 2 covert bits to be added on top of a primary QPSK constellation.

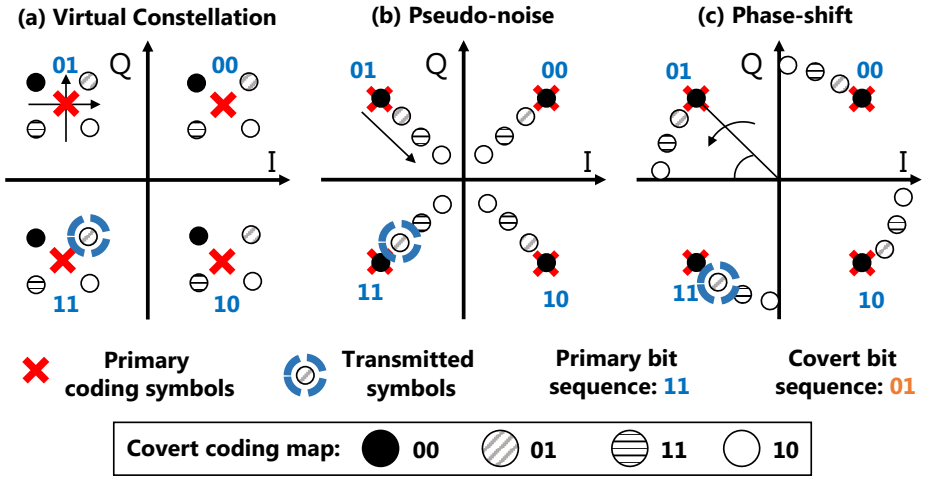


Figure 7.6: Approaches to wireless steganography.

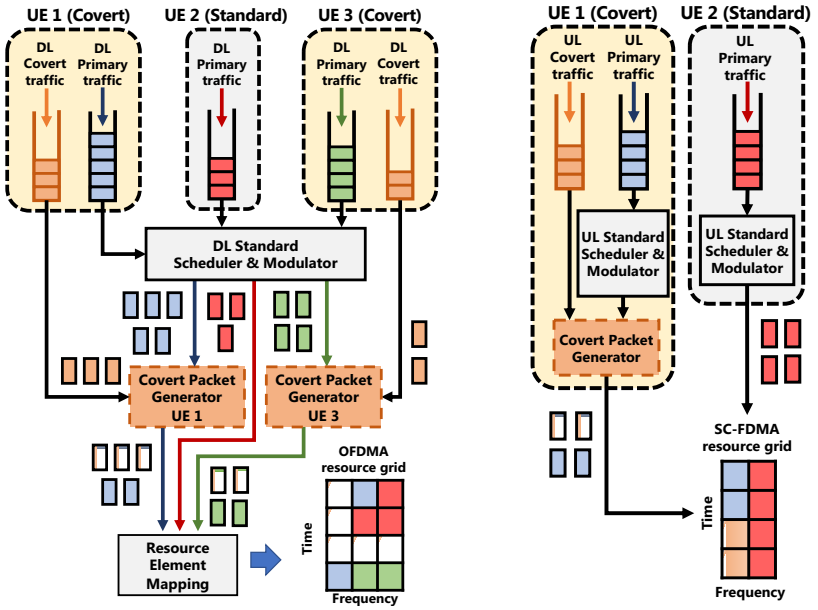
The first approach generates a "dirty" QPSK constellation around each primary symbol (Figure 7.6a) mimicking a hierarchical constellation on top of the primary QPSK constellation [422]. The second one introduces a hierarchical Amplitude Shift Keying (ASK) modulation manipulating the amplitude of the primary symbols (Figure 7.6b) such that different amplitude values encode different covert bit sequences [430]. The third approach modifies the phase offset of the primary symbols (Figure 7.6c) in a way that each phase rotation encodes a specific bit sequence [429]. As SteaLTE is not tied to any specific steganographic procedure its covert modulator supports any of these approaches. In the following we assume that the covert modulator block implements the approach depicted in Figure 7.6b [430], which we call M_C-ASK, where M_C is the number of symbols in the

covert constellation. The advantages of this approach include that it is robust against phase rotations introduced by fading, that it supports high-order modulation schemes (for high covert data rates), and that it can be seamlessly integrated with OFDM systems such as those used in the latest generations of cellular networks. The covert modulator receives the covert packets together with the set of covert modulation parameters, which specify the modulation order, the corresponding coding map, and the packet type (Figure 7.5). The coding map uniquely associates covert packets (i.e., bit sequences) to modulated covert symbols. Covert symbols are, then, embedded in primary symbols through the *covert embedder* block.

Covert Embedder. Once covert symbols have been generated, they are embedded by the covert embedder (Figure 7.5). This procedure modulates the amplitude (and phase) of the primary symbols based on the covert symbols to embed [430]. The output is a sequence of primary symbols with embedded covert data. The symbols are then processed by mapping and precoding blocks and transmitted through the RF front-end (Figure 7.4).

7.1.2.2 Downlink and Uplink Procedures

SteaLTE runs seamlessly on both downlink and uplink transmissions (Figure 7.7), and does not depend on specific MAC strategies (e.g., TDD/FDD, OFDMA/SC-FDMA).



(a) Downlink (DL) transmitter (b) Uplink (UL) transmitter

Figure 7.7: High-level SteaLTE downlink and uplink transmitter design.

Figure 7.7a shows the downlink transmitter design at the BS. In this example, the BS is serving three subscribers: two covert users (UE 1 and UE 3), and a standard user (UE 2). After scheduling the primary transmissions through typical cellular procedures, the BS generates the covert packet to embed on the primary traffic of UE 1 and UE 3 (Section 7.1.2.1). Then, the data for all users

is mapped on the cellular resource grid (e.g., the OFDMA grid in case of LTE downlink), and transmitted.

The high-level uplink transmitter design is shown in Figure 7.7b, where two users are connected to a SteaLTE BS: UE 1 (covert user), and UE 2 (standard user). After completing the mutual authentication procedures, UE 1 generates and embeds the covert packets in the primary uplink traffic to send to the BS. Then, it maps the primary uplink transmission with embedded covert data on the cellular resource grid (e.g., SC-FDMA grid in case of LTE networks). On the other hand, UE 2, which is not aware of the ongoing covert communications, schedules its uplink transmission according to standard cellular procedures.

7.1.3 Receiver Design

Figure 7.8 shows the receiver design. Primary data are processed as per standard cellular procedures. Covert data follow a separate receive chain with two main components: the *Covert Packet Detector* and the *Covert Payload Demodulator*.

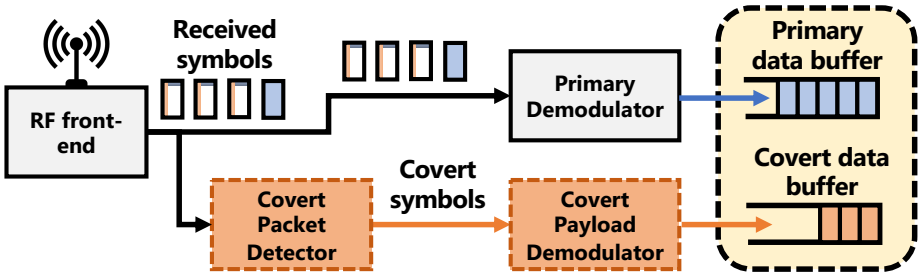


Figure 7.8: High-level SteaLTE receiver design.

7.1.3.1 Covert Packet Detector

This block detects the presence of covert packets demodulating the covert packet header (first L_H bytes of the covert packet). As mentioned in Section 7.1.1.1, the covert header is modulated through a 2-ASK modulation. Thus, if the CRC8 check passes (Section 7.1.1.2), the receiver assumes a covert packet has been received.

Upon detecting a covert packet, the covert packet detector reads the length $L_P + L_{PC}$ of the covert payload and CRC32 fields, the packet number and the modulation parameters in the info field of the header (Figure 7.3). Finally, it extracts the symbols corresponding to the encoded $L_P + L_{PC}$ bytes of the covert packet, that will be demodulated by the *covert demodulator* block of Section 7.1.3.2.

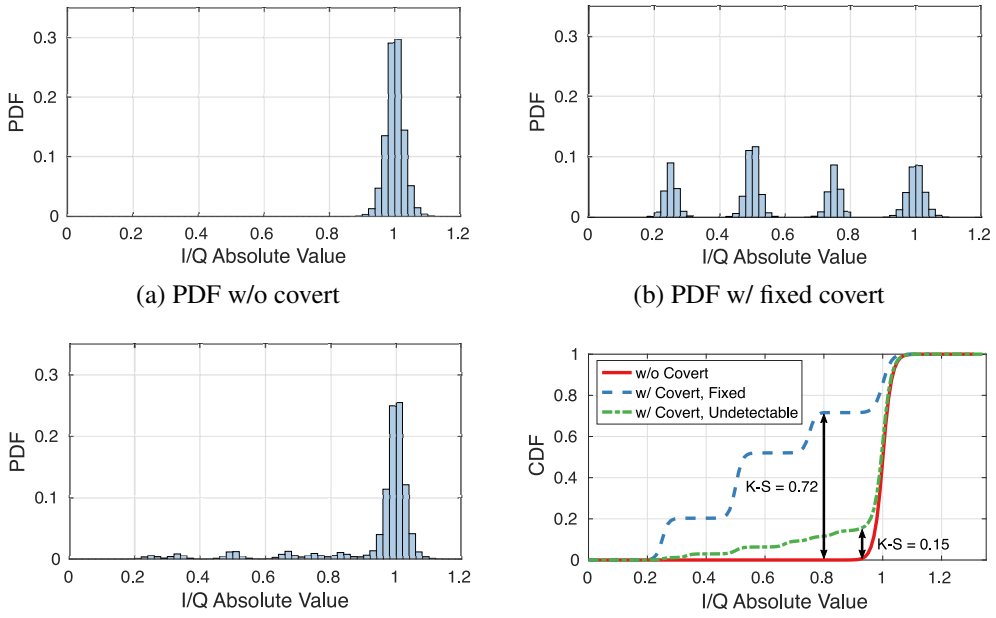
7.1.3.2 Covert Demodulator

This block extracts the encoded covert information from each packet. As shown in Figure 7.3, covert modulation parameters necessary to demodulate covert packets, such as employed covert modulation, packet length and the packet type are specified in the header. This way, the demodulator block can reconstruct the decoding map and use it to demodulate the received symbols into covert data (e.g., bit sequence). Since all the covert modulation parameters are specified in the packet

header, no further interaction is required between transmitter and receiver. Section 7.1.4 (below) shows that this approach also enables time-varying coding/decoding mappings that make covert transmissions undetectable and secure against eavesdroppers. Finally, the received CRC32 value is checked (Section 7.1.1.2). If the check passes, the data are saved, otherwise a retransmission will be requested.

7.1.4 Undetectable Covert Communications

Steganography is not immune from attacks. For instance, through steganalysis [432] an eavesdropper may analyze the statistical properties of captured I/Q samples and infer the presence of a covert slice. For example, let us consider the case of primary QPSK transmissions where SteaLTE embeds covert data through a 4-ASK covert modulation [430]. Figure 7.9 shows the PDF of the I/Q samples captured through the testbed described in Section 7.3.1 in three different cases: primary-only transmissions are shown in Figure 7.9a; primary with *fixed*, i.e., detectable, 4-ASK covert transmissions in Figure 7.9b, and primary with SteaLTE *undetectable* covert transmissions in Figure 7.9c. We also show the CDF of all cases in Figure 7.9d. The Kolmogorov–Smirnov (K-S) distance is also shown to measure the similarity of the CDFs: the smaller the distance, the better.



(a) PDF w/o covert

(b) PDF w/ fixed covert

(c) PDF w/ undetectable covert

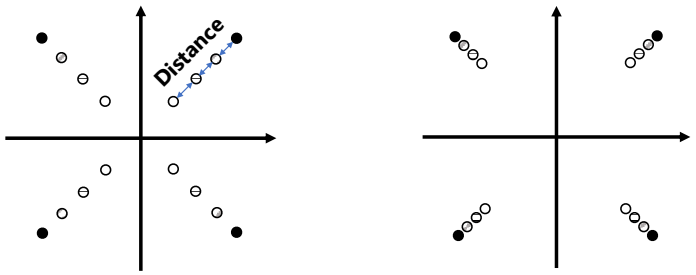
(d) CDF and K-S distance

Figure 7.9: PDF and CDF of received I/Q samples.

Figure 7.9a shows the PDF of the absolute value of the captured primary I/Q samples without any covert transmission. As expected, the PDF assumes a Gaussian distribution with mean 1 due to noise and fading. Figure 7.9b shows the PDF when covert data are embedded through a 4-ASK covert modulation [430]. Comparing Figure 7.9b with the primary-only case of Figure 7.9a, we notice that the absolute value of the captured samples no longer exhibits a Gaussian PDF centered around 1, but multiple bell-shaped Gaussian curves centered at 0.25, 0.5, 0.75, and 1. This is also illustrated in

Figure 7.9d where the CDF of the I/Q samples resembles a step function with a K-S distance with the primary-only case equal to 0.72. Such statistical behavior is not surprising: this result is inherited from the 4-ASK covert scheme in Figure 7.6b, whose operation results in 4 possible covert points per primary symbol with amplitude equal to 0.25, 0.5, 0.75, and 1. Steganalysis can easily identify such an abnormal statistical pattern, thus revealing the ongoing covert transmissions to the eavesdropper. For steganographic communications to be undetectable, they must statistically behave like primary ones, which is possible by reducing their K-S distance.

For this reason, SteaLTE implements a mechanism that mimics I/Q displacements introduced by channel noise by randomizing the covert embedding procedures. Rather than utilizing a *fixed distance* between covert symbols (as done in [430]), SteaLTE *randomly changes* this distance, providing the *first-of-its-kind undetectability mechanism*. This process is illustrated in Figure 7.10, where we show 4 possible configurations of a 4-ASK covert constellation.



(a) Flag = 01 (b) Flag = 11

Figure 7.10: Examples of covert 4-ASK constellations with different distances and threshold flag values.

Table 7.3: Example of 4-ASK threshold and distances.

Flag	Distance
00	0.333
01	0.25
10	0.165
11	0.0825

To decode covert ASK messages, the receiver must be aware of the distance between covert symbols. As a consequence, randomizing the transmitted covert constellation could potentially undermine the receiver's covert demodulation procedures. To overcome this problem, SteaLTE packets carry a *threshold flag* field (see Section 7.1.1.1) instructing the receiver on the covert constellation used by the transmitter. The value of this flag is changed on a per-packet basis, thus reducing the probability of successful steganalysis attacks. In our current implementation, this field consists of 2 bits encoding the 4 different distance configurations shown in Table 7.3. However, it is straightforward to increase the size of this field to further enhance undetectability.

The effectiveness of our approach is depicted in Figure 7.9c, where we show the PDF of SteaLTE undetectable covert messages. The absolute value of the captured I/Q samples is centered around 1, while the remaining small peaks are hardly distinguishable from the noise of the wireless channel. The same behavior can also be observed in the CDF of the primary I/Q samples shown in Figure 7.9d. When compared with the CDF of the primary-only LTE transmissions (solid line), the CDF of the covert signal not only does not show the steps observed with fixed displacements (dashed line), but it also results in a 4.8x shorter K-S distance.

7.2 SteaLTE Prototype

We prototyped SteaLTE on COTS NI USRPs B210 and X310 SDRs. Our implementation is based on the LTE-compliant srsRAN open-source software (see Section 2.2.2). We remark that as SteaLTE follows a software-defined approach, it is not bound to LTE technology, and it can be easily extended to future 5G-and-beyond cellular networks.

We extended srsRAN to allow SteaLTE to *embed*, *encode*, and *decode* covert data on the downlink and uplink LTE primary traffic. Specifically, we enhanced the PDSCH and PUSCH procedures at the PHY-layer. The PDSCH carries the downlink data sent by the eNB to the UEs, and the random access response messages if the PDSCH is mapped to the Random Access Channel (RACH). The PUSCH carries the uplink data that the UEs transmit to the eNB, and ACKs and NACKs for PDSCH data. Figure 7.11 depicts the modified structure of the SteaLTE covert transmitter, i.e., the PDSCH at the eNB downlink side or the PUSCH at the UE uplink side.

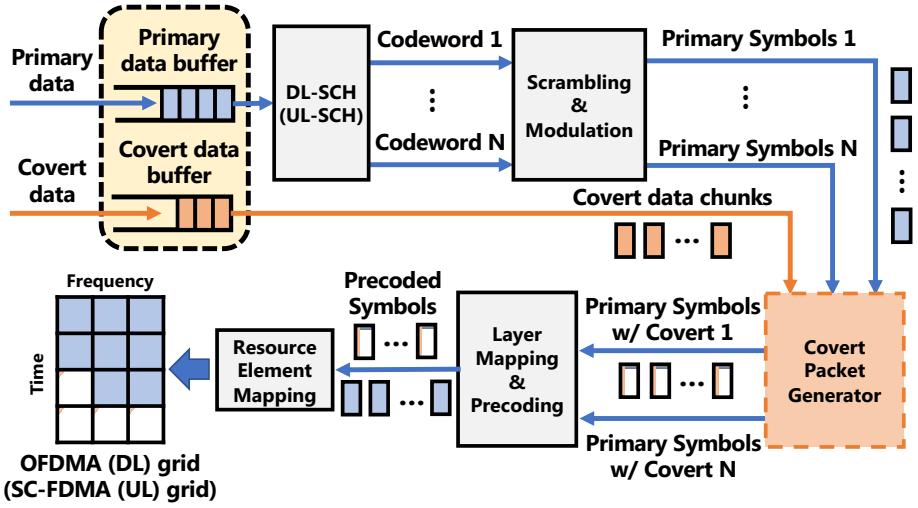


Figure 7.11: The SteaLTE covert transmitter.

When there is primary data to transmit (either in downlink to the UE or uplink to the eNB), this is converted into *codewords* through the Downlink Shared Channel (DL-SCH) (or the Uplink Shared Channel (UL-SCH)), which performs transport channel encoding operations. Then, the resulting codewords are *scrambled* and *modulated* into *primary symbols* (for illustration purposes, in our experiments we adopt a QPSK modulation for the primary traffic that carries covert data). After these operations the SteaLTE *covert packet generator* (Section 7.1.2.1) modifies the amplitude of the resulting primary symbols, thus applying a second (covert) modulation to them. This way, the *covert data chunks* are embedded on the primary symbols. At this point, the complex-modulated *primary symbols with embedded covert* are mapped into layers for spatial multiplexing, and then precoded, as per LTE specifications. Finally, the resulting *precoded symbols* are mapped into resource elements to be transmitted via OFDMA (downlink), or SC-FDMA (uplink).

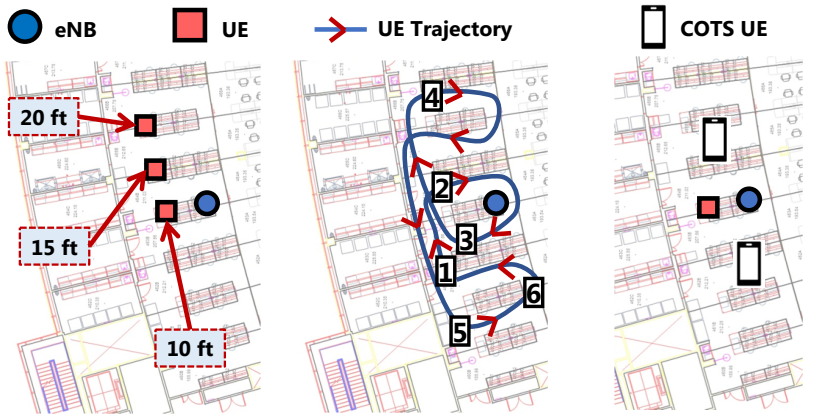
7.3 Experimental Evaluation

We report results from experimental campaigns for evaluating the performance of SteaLTE. Setups are described in Section 7.3.1. Results are shown and discussed in Section 7.3.2.

7.3.1 Experimental Setup

We evaluated SteaLTE on indoor and outdoor platforms.

Indoor scenarios. For our indoor experiments we leveraged the indoor Arena testbed (see Section 3.5). We instantiate LTE-compliant eNBs and UEs on NI USRP X310 SDRs, USRP B210 and COTS Xiaomi Redmi Go smartphones. In all configurations the eNB uses a 10 MHz channel bandwidth corresponding to 50 PRBs. These devices are used in the indoor testbed configurations depicted in Figure 7.12.



(a) Static scenario (b) Dynamic scenario (c) PCCaaS scenario

Figure 7.12: Indoor testbed setup and experiment configuration on the Arena testbed.

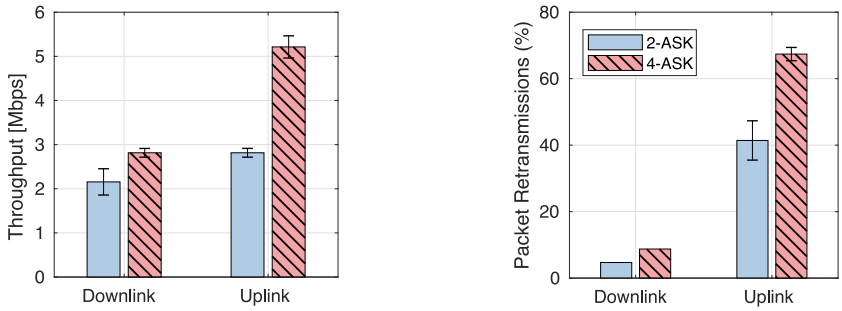
The *static scenario* comprises one eNB and three UEs that are statically placed 10 ft, 15 ft and 20 ft away from the eNB (Figure 7.12a). In this configuration all devices are USRPs X310. In this scenario eNB and UE exchange primary traffic generated through `iPerf3` [433], a software tool for network performance evaluation with TCP and UDP traffic. For experiments with UDP traffic the bitrate varies in the set $\{0.75, 2.5, 5\}$ Mbps. We also use traffic generated by a user-initiated speed test.
The *dynamic scenario* is made up of the eNB and one UE traveling a distance of 190 ft around the eNB as illustrated in Figure 7.12b. Measurements are collected at six different location in the UE journey. In this configuration the eNB is a USRP X310 and the mobile device is a USRP B210. Primary traffic is generated through the *ping* software utility, which uses Internet Control Message Protocol (ICMP) echo request and reply messages.
The *PCCaaS scenario* comprises two slices, one public and one private, with one eNB (USRP X310), one UE using the private slice for covert traffic (USRP X310), and two COTS Xiaomi Redmi Go smartphones transmitting data over the public slice. All users are statically positioned as in Figure 7.12c within 10 ft from the eNB. In this scenario, primary traffic concerns YouTube videos streamed by the users.

Outdoor scenario. For outdoor, long-range testing we ported SteaLTE to the PAWR POWDER platform (see Section 2.7) [5, 6]. We use the NR version of srsRAN to instantiate one outdoor 5G base station (gNB) and one UE. The gNB is located on the rooftop of a 95 ft-tall building and is realized by a USRP X310. The UE is statically positioned at ground-level. It is implemented through a USRP B210. The distance between gNB and UE is 852 ft. The gNB uses a 3 MHz-bandwidth (15 PRBs). Covert data are embedded through a 2-ASK modulation. Primary traffic between gNB and UE is generated through the *ping* utility. In all scenarios covert data are images and text files.

7.3.2 Experimental Results

In this section we report the results of the performance evaluation of SteaLTE in each of the considered scenarios. For each scenario, we describe the investigated metrics and their relevance, and illustrate the corresponding experimental results. Plots include 95% confidence intervals (not shown if $< 1\%$).

Indoor Static Scenario. In the static scenario of Figure 7.12a we start by measuring the performance of SteaLTE to deliver covert data by investigating throughout (data delivery over time) and the percentage of packets that needs to be retransmitted. Figure 7.13 shows the downlink and uplink covert throughput and retransmissions for both 2-ASK and 4-ASK covert modulations under TCP primary traffic.



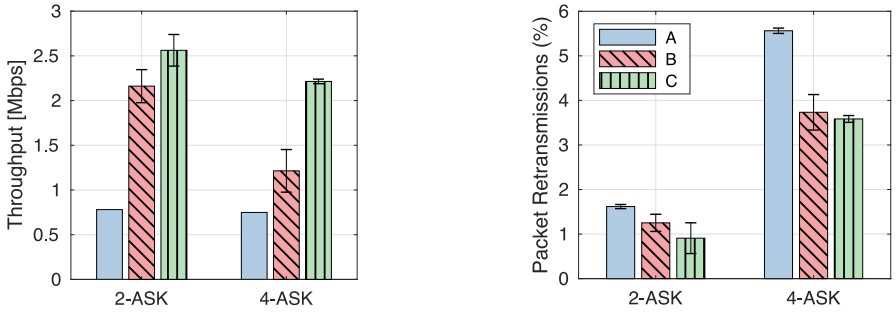
(a) Throughput (b) Retransmissions

Figure 7.13: Downlink and uplink covert performance with TCP primary traffic for different covert modulations.

Being of higher order, 4-ASK obtains a higher throughput than 2-ASK (Figure 7.13a). However, this comes at the cost of transmission errors, leading to a higher percentage of covert packet retransmissions (Figure 7.13b). This is because of the higher resilience to errors of the 2-ASK modulation, which requires 38% less retransmissions than the 4-ASK case (uplink).

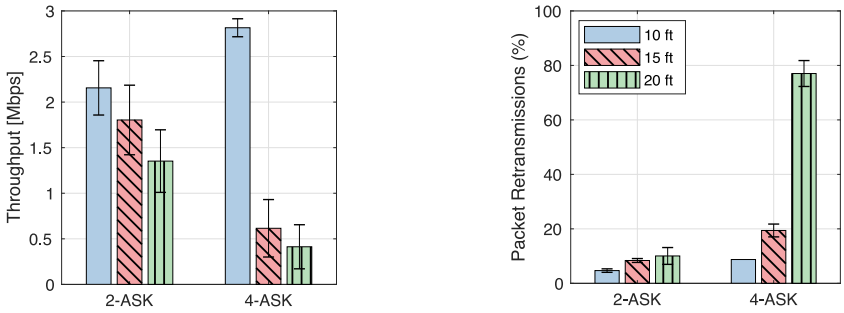
Figure 7.14 depicts the covert throughput (Figure 7.14a) and percentage of covert packet retransmissions (Figure 7.14b) under UDP primary traffic. The `iPerf3` UDP bitrate was set as follows: (A) 0.75 Mbps; (B) 2.5 Mbps, and (C) 5 Mbps.

Differently from TCP traffic (see downlink performance in Figure 7.13a), the covert throughput is higher when data are embedded through a 2-ASK modulation. This is because, unlike TCP, UDP streams data at the specified bitrate, as it does not implement reliable data transfer. This behavior can be further observed in Figure 7.14b, which shows a much higher (close to 3x) number of covert retransmissions in case of 4-ASK modulation.



(a) Throughput (b) Retransmissions

Figure 7.14: Downlink covert performance with UDP primary traffic for different traffic profiles and covert modulations.



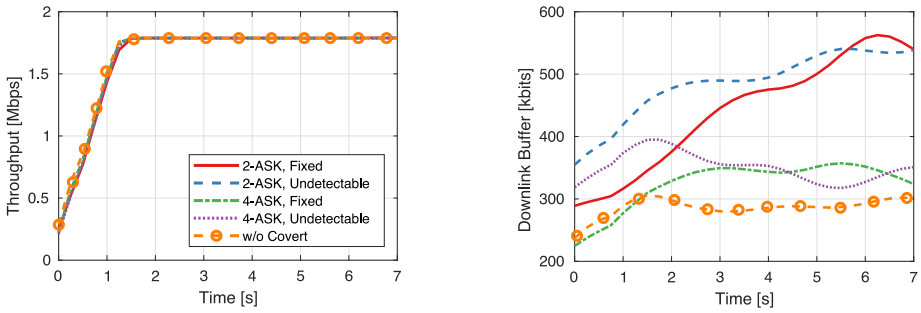
(a) Throughput (b) Retransmissions

Figure 7.15: Downlink covert performance with TCP primary traffic for different covert modulations and distances between eNB and UE.

Figures 7.15a and 7.15b show the covert throughput and the percentage of packet retransmissions when TCP primary traffic is exchanged between eNB and UE, as a function of their distance (Figure 7.12a). We notice that for short distances (i.e., $10\,\mathrm{ft}$) 4-ASK provides the highest performance. However, as the distance increases ($\geq 15\,\mathrm{ft}$), modulating the covert message through 2-ASK yields a better performance, both for throughput and retransmissions. This is because of the higher robustness to errors of this modulation compared to the 4-ASK modulation.

We now investigate the impact of the undetectability schemes presented in Section 7.1.4 on the performance of primary transmissions. Their effectiveness in concealing covert data has been shown in Figure 7.9. Here we show results for metrics that indicate that these schemes do not have a significant impact over the quality of transmission and on channel quality: throughput, the number of bytes that are to be transmitted in the downlink and the SINR. The results shown in Figures 7.16 and 7.17 refer to UE-generated traffic according to a speed test application at $1.8\,\mathrm{Mbps}$.

Figures 7.16a and 7.16b show the downlink primary throughput and transmission buffer size averaged over 10 independent experiments. Figure 7.16a clearly indicates that embedding covert messages—both fixed and undetectable—does not noticeably affect primary traffic. In Figure 7.16b we notice a slight increase of the size of the downlink buffer queue when covert communications happen, especially for the 2-ASK undetectability schemes. This suggests that a higher number of



(a) Downlink throughput (b) Downlink buffer size

Figure 7.16: Impact of SteaLTE on speed test primary traffic: downlink throughput and buffer size.

retransmissions is needed for the eNB to deliver the primary traffic to the UEs, with an impact on the number of resources that the eNB uses to communicate to the users. Yet again, in this scenario, this does not translate in a noticeable degradation of the throughput.

Figure 7.17 shows the distribution of the downlink buffer size (top) and the SINR measured by the user (bottom). These two results show that the statistical distributions of these two metrics do not

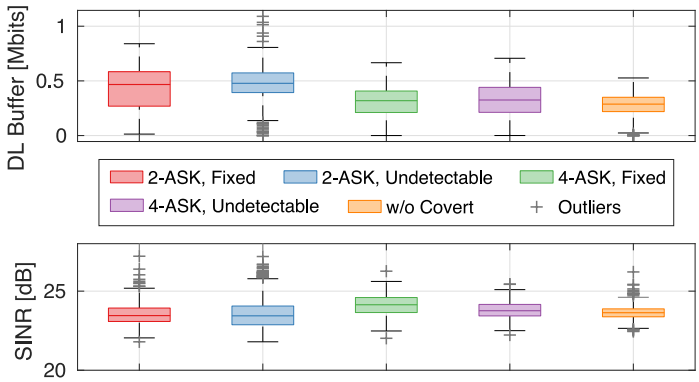
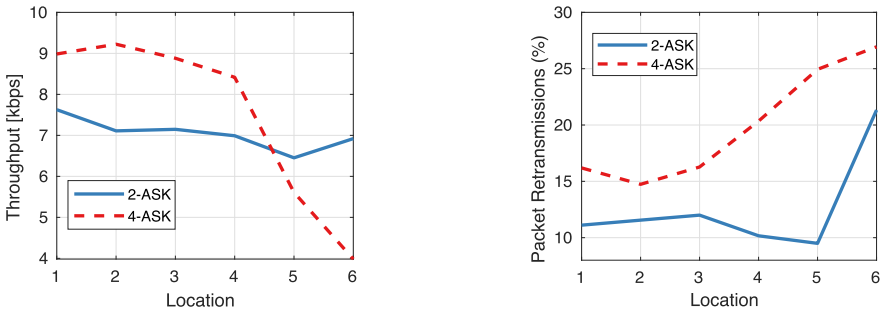


Figure 7.17: Impact of SteaLTE on speed test primary traffic: downlink buffer size and SINR.

vary significantly in the case when SteaLTE is used (independently of the undetectability scheme and modulation used) and the case when it is not. Particularly, the average difference among the downlink buffer size in the two cases never exceeds 106.5 kbit. Also, the difference among SINR is within 0.15 dB, indicating that embedding covert data does not increase noise perceptively.

Indoor Dynamic Scenario. The throughput and retransmission performance of SteaLTE for the different user locations of Figure 7.12b and covert modulations is shown in Figure 7.18.

As the distance between eNB and UE increases (as for locations 4, 5 and 6—Figure 7.12b), the covert throughput (Figure 7.18a) for both 2-ASK and 4-ASK modulations decreases, while the percentage of packet retransmissions increases (Figure 7.18b). As noticed before, in general, 2-ASK outperforms 4-ASK because of its higher robustness to channel impairments, which are more noticeable at larger distances. Despite the performance degradation, SteaLTE still manages to enable secret communications between covert transmitter and receiver in presence of mobility.



(a) Throughput (b) Retransmissions

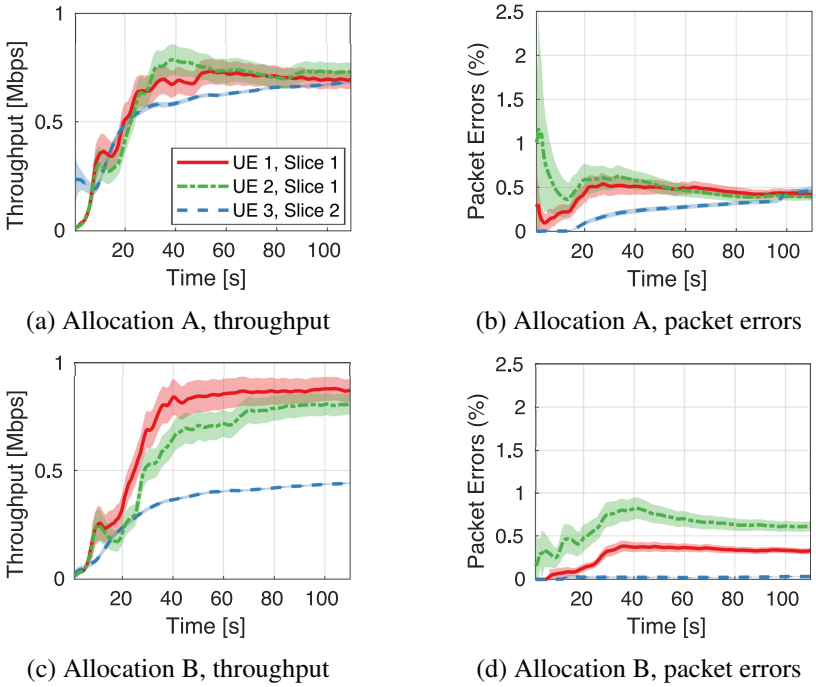
Figure 7.18: Covert performance with ICMP echo reply primary traffic for different covert modulations in presence of UE mobility.

Indoor PCCaaS scenario We now set to investigate the effectiveness of SteaLTE for instantiating private network slices on a shared cellular infrastructure, which serves the need of those critical applications requiring rapid instantiation of private and secure cellular networks. In this set of experiments, we instantiate two slices: a primary-only slice (Slice 1) serving covert-agnostic UEs 1 and 2 (for which we use smartphones), and a SteaLTE private slice (Slice 2) for covert communications between UE 3 and the eNB (both USRPs X310). The covert data of UE 3 is embedded through 4-ASK modulation. We consider two different network slicing allocations: (A) spectrum resources are evenly split among the two slices, and (B) 70% of the resources are allocated to Slice 1 and 30% to Slice 2. For these scenarios we investigate the primary throughput and the percentage of packets erroneously received for both slicing configurations when all users stream videos from YouTube. Results are shown in Figure 7.19.

We notice that in both allocations, the primary throughput of all users is not impacted by the presence of covert communications. The percentage of packet errors (Figures 7.19b and 7.19d) is negligible in both allocations (i.e., below 0.5% on average, with a peak of 2.5%). As a result, Figures 7.19a and 7.19c show that the throughput level achieved by all users is enough for rate-demanding applications, thus confirming the low impact of SteaLTE on primary communications.

Outdoor Scenario. SteaLTE has been tested also on a long-rage link (> 850 ft) in the outdoor scenario provided by the POWDER platform [6]. Results on throughput and retransmission percentage are shown in Figure 7.20. Specifically, Figure 7.20a shows downlink covert throughput and retransmissions and Figure 7.20b depicts the same metrics for the primary traffic (with and without covert).

Figure 7.20a shows that SteaLTE is capable of delivering covert data despite the severe path-loss, multi-path and fading conditions experienced over the long-range link, thus demonstrating its suitability for traditional (mostly outdoors) cellular applications. As for the indoor results above, Figure 7.20b confirms that embedding covert data into primary traffic has negligible effect on its throughput performance. The primary throughput degradation over the long-range link is merely 5.59%, whereas the packet retransmission percentage increases from 2.6% to 3.2% (0.6% increase) when covert traffic is embedded.



(a) Allocation A, throughput

(b) Allocation A, packet errors

(c) Allocation B, throughput

(d) Allocation B, packet errors

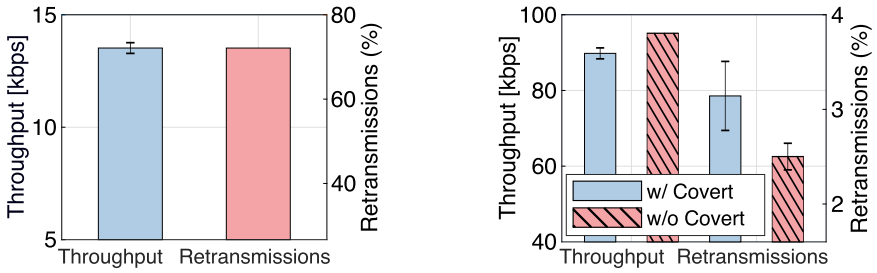
Figure 7.19: Throughput and percentage of packet errors on primary traffic for different resource allocations of the private and standard slices.

7.4 Related Work

Wireless steganography has been frequently used for covert communications among parties. Differently from approaches where covert data are embedded in the packet control fields (e.g., checksum [434], flags [435, 436], and padding fields [437], among others [438, 439]), wireless steganography introduces tiny displacements in the I/Q constellation plane that can be controlled to encode covert information. Typical methods include frequency/phase shifts [423, 429], I/Q imbalance [440], superimposing noisy constellations [422, 430, 441, 442] or training and preamble sequences manipulations [429, 443]. These approaches, however, lack reliability as they are prone to demodulation errors and rarely support long-range communications, quintessential for many communication systems.

These reliability issues have been partially addressed at the higher layers of the protocol stack. Hamdaqa and Tahvildari describe a steganographic system for Voice-over-IP (VoIP) that encodes covert information by carefully delaying packet transmissions [444]. Although this approach is highly reliable and undetectable, it operates over large temporal windows, which considerably limits the achievable covert rate. Nain and Rajalakshmi develop a steganographic communication system that hides information over chip sequences of IEEE 802.15.4 networks integrating error-coding techniques to mitigate errors [445]. Although being reliable, this method only achieves low transmission rates. Overall, previous solutions either achieve *low covert throughput*, or are highly detectable through steganalysis [432, 446], or lack practical implementations demonstrating their effectiveness and feasibility.

SteaLTE is the missing answer to the high throughput, reliability, and undetectability requirements



(a) Covert (b) Primary

Figure 7.20: Long-range experiments on the POWDER platform.

of PCCaaS applications characterized by mobility, time-varying channels and large distances. As a reliable end-to-end steganographic system SteaLTE: (i) achieves high throughput through wireless steganography; (ii) provides reliable and channel-resilient communications through a combination of error-coding, retransmissions and adaptive covert modulation schemes, and (iii) can be seamlessly integrated in 3GPP-compliant cellular systems.

7.5 Conclusions

This chapter proposes SteaLTE, the first practical PCCaaS-enabling system for softwarized cellular networks. SteaLTE supports reliable, undetectable, high-throughput, and long-range covert communications. We have prototyped SteaLTE and implemented it on LTE-compliant testbeds, including indoor settings and the PAWR POWDER platform (ourdoors). We have extensively evaluated the performance of SteaLTE under diverse traffic profiles, distance and mobility patterns, highlighting the feasibility of undetectable transmissions and their negligible impact on primary traffic. Results over the multiplicity of the considered scenarios show that, in the vast majority of our experiments SteaLTE achieves a throughput of covert traffic that is at least 90% of the throughput of the primary traffic, affecting the latter negligibly ($< 6\%$ loss).

Chapter 8

Conclusions

This dissertation work focused on the design, prototyping and experimental evaluation of *softwarized approaches for the Open RAN of NextG cellular networks*. First, we gave an overview of the state-of-the-art on open, programmable, and virtualized cellular networks (Chapter 2). We discussed the architectural enablers, frameworks, and technologies of NextG networks, highlighting challenges and future directions in achieving a fully-softwarized and flexible control of the network. In Chapter 3, we detailed the main wireless experimental platforms used throughout this work. We described (i) Colosseum, the world's largest wireless network emulator, which enables hardware-in-the-loop experimentation at scale under different emulated wireless and traffic conditions; (ii) SCOPE, an open and softwarized platform for *large-scale prototyping and experimentation* of NextG solution, and for the *automated data-collection* of RANs KPIs; and (iii) Arena, an indoor testbed for spectrum research.

Then, in Chapter 4, we provided the first demonstration of O-RAN data-driven control loops in a *large-scale experimental testbed*—for which we leveraged the Colosseum testbed—*using open-source, programmable* RAN and RIC components through *xApps* of our design. In Chapter 5, we proposed zero-touch solutions to achieve automatic control and self-optimization of the very many elements of NextG cellular networks. We proposed CellOS, a *zero-touch cellular operating system* that automatically generates and executes distributed control programs for *simultaneous optimization of heterogeneous control objectives on multiple network slices* starting from a high-level intent expressed by the operators, and QCell, a Deep Q-Network-based framework for the *self-optimization of slice resources and policies*. In Chapter 6, we presented OrchestRAN, a framework for the intelligent orchestration of network components in the Open RAN. Finally, in Chapter 7, we presented SteaLTE, the first realization of *private cellular connectivity-as-a-service* in NextG networks through wireless steganography.

We prototyped the proposed solutions by leveraging open-source software implementations of cellular protocol stacks and frameworks, and heterogeneous virtualization technologies, including the srsRAN and OpenAirInterface cellular implementations, the O-RAN framework, and the LXC and Docker virtualization technologies. The effectiveness of our solutions in achieving superior control and performance of the RAN is demonstrated on state-of-the-art experimental facilities, including software-defined radio-based laboratory setups and open access experimental wireless platforms, such as Colosseum, Arena, and the POWDER platform from the U.S. PAWR program.

References

[1] T. Bu, L. E. Li, and R. Ramjee, "Generalized Proportional Fair Scheduling in Third Generation Wireless Data Networks," in *Proceedings of IEEE INFOCOM*, Barcelona, Spain, April 2006.

[2] X. Wu, R. Srikant, and J. R. Perkins, "Scheduling Efficiency of Distributed Greedy Scheduling Algorithms in Wireless Networks," *IEEE Transactions on Mobile Computing*, vol. 6, no. 6, pp. 595–605, June 2007.

[3] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks," in *Proc. of ACM SIGCOMM Conf. on emerging Networking EXperiments and Technologies (CoNEXT)*, Irvine, CA, USA, December 2016.

[4] X. Foukas, M. Marina, and K. Kontovasilis, "Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture," in *Proc. of ACM Intl. Conf. on Mobile Computing and Networking (MobiCom)*, Snowbird, UT, USA, October 2017.

[5] Platforms for Advanced Wireless Research (PAWR). https://www.advancedwireless.org. Accessed July 2020.

[6] J. Breen, A. Buffmire, J. Duerig, K. Dutt, E. Eide, A. Ghosh, M. Hibler, D. Johnson, S. K. Kasera, E. Lewis *et al.*, "POWDER: Platform for Open Wireless Data-driven Experimental Research," *Computer Networks*, vol. 197, pp. 1–18, October 2021.

[7] Q. Liu and T. Han, "DIRECT: Distributed Cross-Domain Resource Orchestration in Cellular Edge Computing," in *Proceedings of ACM MobiHoc*, Catania, Italy, July 2019.

[8] S. Parkvall, E. Dahlman, A. Furuskar, and M. Frenne, "NR: The New 5G Radio Access Technology," *IEEE Communications Standards Magazine*, vol. 1, no. 4, pp. 24–30, December 2017.

[9] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta, and P. Popovski, "Five Disruptive Technology Directions for 5G," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 74–80, May 2014.

[10] D. Martín-Sacristán, J. F. Monserrat, J. Cabrejas-Penuelas, D. Calabuig, S. Garrigas, and N. Cardona, "On the Way Towards Fourth-generation Mobile: 3GPP LTE and LTE-Advanced," *Springer EURASIP Journal on Wireless Communications and Networking*, vol. 2009, no. 1, p. 354089, August 2009.

[11] O-RAN Alliance White Paper, "O-RAN: Towards an Open and Smart RAN," https://www.o-ran.org/resources, 2018.

[12] S. D'Oro, F. Restuccia, and T. Melodia, "The Slice Is Served: Enforcing Radio Access Network Slicing in Virtualized 5G Systems," in *Proceedings of IEEE INFOCOM*, Paris, France, May 2019.

[13] S. D'Oro, F. Restuccia, and T. Melodia, "Toward Operator-to-Waveform 5G Radio Access Network Slicing," *IEEE Communications Magazine*, vol. 58, no. 4, pp. 18–23, April 2020.

[14] M. Zambianco and G. Verticale, "Interference Minimization in 5G Physical-Layer Network Slicing," *IEEE Transactions on Communications*, March 2020.

[15] N. Bhushan, J. Li, D. Malladi, R. Gilmore, D. Brenner, A. Damnjanovic, R. T. Sukhavasi, C. Patel, and S. Geirhofer, "Network Densification: The Dominant Theme for Wireless Evolution into 5G," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 82–89, 2014.

[16] J. Haavisto, M. Arif, L. Lovén, T. Leppänen, and J. Riekki, "Open-source RANs in Practice: an Over-The-Air Deployment for 5G MEC," in *Proc. of IEEE European Conf. on Networks and Communications (EuCNC)*, Valencia, Spain, June 2019.

[17] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an Open, Distributed SDN OS," in *Proc. of ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, Chicago, Illinois, USA, 2014.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 38, no. 2, pp. 69—-74, March 2008.

[19] N. M. M. K. Chowdhury and R. Boutaba, "A Survey of Network Virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, April 2010.

[20] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, November 2017.

[21] L. Zhang, M. Xiao, G. Wu, M. Alam, Y. Liang, and S. Li, "A Survey of Advanced Techniques for Spectrum Sharing in 5G Networks," *IEEE Wireless Communications*, vol. 24, no. 5, pp. 44–51, October 2017.

[22] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5G Network Slicing Using SDN and NFV: A Survey of Taxonomy, Architectures and Future Challenges," *Computer Networks*, vol. 167, p. 106984, February 2020.

[23] S. Sun, M. Kadoch, L. Gong, and B. Rong, "Integrating Network Function Virtualization with SDR and SDN for 4G/5G Networks," *IEEE Network*, vol. 29, no. 3, pp. 54–59, June 2015.

[24] F. Kaltenberger, A. P. Silva, A. Gosain, L. Wang, and T.-T. Nguyen, "OpenAirInterface: Democratizing Innovation in the 5G Era," *Computer Networks*, no. 107284, May 2020.

[25] ONAP, "Architecture Overview," https://www.onap.org/wp-content/uploads/sites/20/2019/07/ONAP_CaseSolution_Architecture_062519.pdf, 2019.

[26] GNU Radio Project. GNU Radio. http://www.gnuradio.org. Accessed July 2020.

[27] I. Gomez-Miguelez, A. Garcia-Saavedra, P. Sutton, P. Serrano, C. Cano, and D. Leith, "srsLTE: An Open-source Platform for LTE Evolution and Experimentation," in *Proceedings of ACM WiNTECH*, New York City, NY, USA, October 2016.

[28] O-RAN. https://o-ran.org. Accessed July 2020.

[29] Open Networking Foundation. Open Network Automation Platform (ONAP). https://onap.org. Accessed July 2020.

[30] Rakuten, "How Elegant Software Can Make 5G Networks More Resilient," https://rakuten.today/blog/5g-network-reliability-lightreading.html, 2020, accessed July 2020.

[31] 5G Americas Whitepaper, "The Status of Open Source for 5G," https://www.5gamericas.org/wp-content/uploads/2019/07/5G_Americas_White_Paper_The_Status_of_Open_Source_for_5G_Feb_2019.pdf, 2019, accessed August 2020.

[32] T. Chen, M. Matinmikko, X. Chen, X. Zhou, and P. Ahokangas, "Software Defined Mobile Networks: Concept, Survey, and Research Directions," *IEEE Communications Magazine*, vol. 53, no. 11, pp. 126–133, November 2015.

[33] M. Condoluci and T. Mahmoodi, "Softwarization and Virtualization in 5G Mobile Networks: Benefits, Trends and Challenges," *Computer Networks*, vol. 146, pp. 65–84, December 2018.

[34] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, August 2018.

[35] D. Thembelihle, M. Rossi, and D. Munaretto, "Softwarization of Mobile Network Functions Towards Agile and Energy Efficient 5G Architectures: A Survey," *Wireless Communications and Mobile Computing*, vol. 2017, pp. 1–21, November 2017.

[36] 3GPP, "NR and NG-RAN Overall Description," TS 38.300, V15.0.0, 2018.

[37] ——, "Study on Scenarios and Requirements for Next Generation Access Technologies," TS 38.913, V14.1.0, 2018.

[38] ATIS - on behalf of 3GPP, "3GPP 5G Candidate For Inclusion In IMT-2020: Submission 1 (SRIT)," ITU Document 5D/1216-E, June 2019.

[39] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.300, 04 2017, version 14.2.0. [Online]. Available: http://www.3gpp.org/DynaReport/36300.htm

[40] ——, "Evolved Universal Terrestrial Radio Access (E-UTRA); Packet Data Convergence Protocol (PDCP) Specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.323, 03 2017, version 14.2.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2439

[41] ——, "Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Link Control (RLC) Protocol Specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.322, 03 2017, version 14.0.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2438

[42] ——, "Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) Protocol Specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.321, 04 2017, version 14.2.1. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2437

[43] ——, "Evolved Universal Terrestrial Radio Access (E-UTRA); LTE Physical Layer; General Description," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.201, 03 2017, version 14.1.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2424

[44] ——, "Network Architecture," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 23.002, 3 2018, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/23002.htm

[45] S. Rangan, T. S. Rappaport, and E. Erkip, "Millimeter-Wave Cellular Wireless Networks: Potentials and Challenges," *Proceedings of the IEEE*, vol. 102, no. 3, pp. 366–385, March 2014.

[46] M. Giordani, M. Polese, A. Roy, D. Castor, and M. Zorzi, "A Tutorial on Beam Management for 3GPP NR at mmWave Frequencies," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 173–196, February 2019.

[47] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA) and NR; Service Data Adaptation Protocol (SDAP) Specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 37.324, 03 2018, version 15.1.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3282

[48] ——, "NR; Physical Layer; General Description," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.201, 01 2018, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/38201.htm

[49] ——, "NR; Medium Access Control (MAC) Protocol Specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.321, 01 2018, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/38321.htm

[50] ——, "NR; Radio Link Control (RLC) Protocol Specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.322, 01 2018, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/38322.htm

[51] ——, "NR; Packet Data Convergence Protocol (PDCP) Specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.323, 01 2018, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/38323.htm

[52] ——, "NR; Radio Resource Control (RRC); Protocol Specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.331, 01 2018, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/38331.htm

[53] ——, "Study on CU-DU Lower Layer Split for NR," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 38.816, 01 2018, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/38816.htm

[54] D. Brake, "A U.S. National Strategy for 5G and Future Wireless Innovation," https://itif.org/publications/2020/04/27/us-national-strategy-5g-and-future-wireless-innovation, April 2020.

[55] 3GPP, "System Architecture for the 5G System (5GS)," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 23.501, 3 2020, version 16.4.0. [Online]. Available: http://www.3gpp.org/DynaReport/23501.htm

[56] ——, "Study on Enhancements to the Service-Based Architecture," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 23.742, 12 2018, version 16.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/23742.htm

[57] J. Kim, D. Kim, and S. Choi, "3GPP SA2 Architecture and Functions for 5G Mobile Communication System," *ICT Express*, vol. 3, no. 1, pp. 1–8, April 2017.

[58] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow, "Central Office Re-architected as a Data Center," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 96–101, October 2016.

[59] ONF White Paper, "Aether - Enterprise-5G/LTE-Edge-Cloud-as-a-Service," https://www.opennetworking.org/wp-content/uploads/2020/02/Aether-white-paper.pdf, February 2020.

[60] ——, "ONF's Software-Defined RAN Platform Consistent with the O-RAN Architecture," https://www.opennetworking.org/wp-content/uploads/2020/03/SD-RAN-White-Paper.pdf, February 2020.

[61] Open Platform for NFV (OPNFV). https://www.opnfv.org. Accessed July 2020.

[62] S. D'Oro, F. Restuccia, T. Melodia, and S. Palazzo, "Low-Complexity Distributed Radio Access Network Slicing: Algorithms and Experimental Results," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2815–2828, December 2018.

[63] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, and H. Bakker, "Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, May 2017.

[64] TM Forum. https://www.tmforum.org. Accessed July 2020.

[65] ONAP, "ONAP 5G Blueprint Overview," 2019. [Online]. Available: https://www.onap.org/wp-content/uploads/sites/20/2019/07/ONAP_CaseSolution_5G_062519.pdf

[66] OpenSlice Repositories. https://github.com/openslice. Accessed July 2020.

[67] Z. Xiang, F. Gabriel, E. Urbano, G. T. Nguyen, M. Reisslein, and F. H. P. Fitzek, "Reducing Latency in Virtual Machines: Enabling Tactile Internet for Human-Machine Co-Working," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1098–1116, May 2019.

[68] F. Giust, V. Sciancalepore, D. Sabella, M. C. Filippou, S. Mangiante, W. Featherstone, and D. Munaretto, "Multi-Access Edge Computing: The Driver Behind the Wheel of 5G-Connected Cars," *IEEE Communications Standards Magazine*, vol. 2, no. 3, pp. 66–73, March 2018.

[69] A. Kropp, R. Schmoll, G. T. Nguyen, and F. H. P. Fitzek, "Demonstration of a 5G Multi-access Edge Cloud Enabled Smart Sorting Machine for Industry 4.0," in *Proc. of IEEE Annual Consumer Communications Networking Conf. (CCNC)*, Las Vegas, NV, USA, 2019.

[70] ETSI, "Multi-access Edge Computing (MEC); Framework and Reference Architecture," ETSI GS MEC 003 V2.1.1, https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf, 2019, accessed July 2020.

[71] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, K. W. Wen, K. Kim, R. Arora, A. Odgers, L. M. Contreras, and S. Scarpina, "MEC in 5G networks," ETSI White Paper No. 28, https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf, June 2018, accessed July 2020.

[72] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, Third quarter 2017.

[73] R. Bruschi, R. Bolla, F. Davoli, A. Zafeiropoulos, and P. Gouvas, "Mobile Edge Vertical Computing over 5G Network Sliced Infrastructures: An Insight into Integration Approaches," *IEEE Communications Magazine*, vol. 57, no. 7, pp. 78–84, July 2019.

[74] C. Jiang, H. Zhang, Y. Ren, Z. Han, K. Chen, and L. Hanzo, "Machine Learning Paradigms for Next-Generation Wireless Networks," *IEEE Wireless Communications*, vol. 24, no. 2, pp. 98–105, April 2017.

[75] I. Alawe, A. Ksentini, Y. Hadjadj-Aoul, and P. Bertin, "Improving Traffic Forecasting for 5G Core Network Scalability: A Machine Learning Approach," *IEEE Network*, vol. 32, no. 6, pp. 42–49, November 2018.

[76] N. Strodthoff, B. Göktepe, T. Schierl, C. Hellge, and W. Samek, "Enhanced Machine Learning Techniques for Early HARQ Feedback Prediction in 5G," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 11, pp. 2573–2587, November 2019.

[77] A. Asadi, S. Müller, G. H. Sim, A. Klein, and M. Hollick, "FML: Fast Machine Learning for 5G mmWave Vehicular Communications," in *Proc. of IEEE Intl. Conf. on Computer Communications (INFOCOM)*, Honolulu, HI, USA, April 2018.

[78] C. Zhang, P. Patras, and H. Haddadi, "Deep Learning in Mobile and Wireless Networking: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224–2287, March 2019.

[79] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, May 2019.

[80] M. Polese, R. Jana, V. Kounev, K. Zhang, S. Deb, and M. Zorzi, "Machine Learning at the Edge: A Data-driven Architecture with Applications to 5G Cellular Networks," *IEEE Transactions on Mobile Computing*, vol. 20, no. 12, pp. 3367–3382, December 2021.

[81] OpenAirInterface Software Alliance. OpenAirInterface (OAI). https://openairinterface.org. Accessed July 2020.

[82] Software Radio Systems. srsLTE. https://srsran.com. Accessed November 2021.

[83] Radisys. https://radisys.com. Accessed July 2020.

[84] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A Flexible Platform for 5G Research," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 33–38, October 2014.

[85] OpenAirInterface (OAI) Public License. https://www.openairinterface.org/?page_id=698. Accessed July 2020.

[86] Ettus Research. Universal Software Radio Peripheral (USRP). https://www.ettus.com/products. Accessed July 2020.

[87] F. Kaltenberger, G. de Souza, R. Knopp, and H. Wang, "The OpenAirInterface 5G New Radio Implementation: Current Status and Roadmap," in *Proc. of ITG Workshop on Smart Antennas (WSA)*, Vienna, Austria, April 2019.

[88] F. Kaltenberger, X. Jiang, and R. Knopp, "From Massive MIMO to C-RAN: The OpenAirInterface 5G Testbed," in *Asilomar Conf. on Signals, Systems, and Computers (ACSSC)*, Pacific Grove, CA, USA, October 2017.

[89] Q. Liu, T. Han, and N. Ansari, "Learning-Assisted Secure End-to-End Network Slicing for Cyber-Physical Systems," *arXiv preprint arXiv:1910.13537 [cs.NI]*, October 2019.

[90] Fujitsu demonstrates the power of OpenAirInterface. (2019, March) https://www.openairinterface.org/?news=fujitsu-demonstrates-the-power-of-openairinterface.

[91] WindyCitySDR. http://www.windycitysdr.com/home. Accessed July 2020.

[92] InterDigital MWC18: 5G Air Interface. (2018, February) https://www.interdigital.com/presentations/mwc18-5g-air-interface.

[93] SYRTEM. http://www.syrtem.com. Accessed July 2020.

[94] Software Radio Systems. The srsLTE Project is Evolving. https://www.srsran.com/srslte-srsran. Accessed November 2021.

[95] A. Puschmann, P. Sutton, and I. Gomez, "Implementing NB-IoT in Software - Experiences Using the srsLTE Library," *arXiv preprint arXiv:1705.03529 [cs.NI]*, May 2017.

[96] GNU Affero General Public License Version 3. https://www.gnu.org/licenses/agpl-3.0.en.html. Accessed July 2020.

[97] 3GPP, "3G Security; Cryptographic Algorithm Requirements," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 33.105, 6 2018, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/33105.htm

[98] ——, "3G Security; Specification of the MILENAGE Algorithm Set: An Example Algorithm Set for the 3GPP Authentication and Key Generation Functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm Specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 35.206, 10 2018, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/35206.htm

[99] Lime Microsystems. LimeSDR. https://limemicro.com/products/boards/. Accessed July 2020.

[100] Nuand. bladeRF. https://www.nuand.com. Accessed July 2020.

[101] N. Bui and J. Widmer, "OWL: A Reliable Online Watcher for LTE Control Channel Measurements," in *Proc. of ACM Workshop on All Things Cellular: Operations, Applications and Challenges (ATC)*, New York City, NY, USA, October 2016.

[102] F. Meneghello, M. Rossi, and N. Bui, "Smartphone Identification via Passive Traffic Fingerprinting: A Sequence-to-Sequence Learning Approach," *IEEE Network*, vol. 34, no. 2, pp. 112–120, February 2020.

[103] H. D. Trinh, A. F. Gambin, L. Giupponi, M. Rossi, and P. Dini, "Mobile Traffic Classification through Physical Channel Fingerprinting: A Deep Learning Approach," *arXiv preprint arXiv:1910.11617 [eess.SP]*, October 2019.

[104] H. Kim, J. Lee, E. Lee, and Y. Kim, "Touching the Untouchables: Dynamic Security Analysis of the LTE Control Plane," in *Proc. of IEEE Symp. on Security and Privacy (SP)*, San Francisco, CA, USA, May 2019.

[105] D. Rupprecht, K. Kohls, T. Holz, and C. Pöpper, "Breaking LTE on Layer Two," in *Proc. of IEEE Symp. on Security and Privacy (SP)*, San Francisco, CA, USA, May 2019.

[106] H. Yang, S. Bae, M. Son, H. Kim, S. M. Kim, and Y. Kim, "Hiding in Plain Signal: Physical Signal Overshadowing Attack on LTE," in *Proc. of USENIX Security Symp.*, Santa Clara, CA, USA, August 2019.

[107] A. Singla, S. R. Hussain, O. Chowdhury, E. Bertino, and N. Li, "Protecting the 4G and 5G Cellular Paging Protocols against Security and Privacy Attacks," in *Proc. of Sciendo Privacy Enhancing Technologies*, Montreal, QC, Canada, July 2020.

[108] NIST. OpenFirst. https://www.nist.gov/ctl/pscr/openfirst. Accessed July 2020.

[109] Radisys. Radysis O-RAN DU. https://gerrit.o-ran-sc.org/r/admin/repos/o-du/l2. Accessed July 2020.

[110] ——. Open-Source 4G RAN Software for Qualcomm FSM9955 Chipset. https://www.radisys.com/OpenRadisys-4G-RAN-Software. Accessed July 2020.

[111] Open Networking Foundation. Open Mobile Evolved Core (OMEC). https://opennetworking.org/omec. Accessed July 2020.

[112] free5GC. https://free5gc.org. Accessed July 2020.

[113] Open5GS. https://open5gs.org. Accessed July 2020.

[114] 3GPP, "General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 23.401, 3 2020, version 16.6.0. [Online]. Available: http://www.3gpp.org/DynaReport/23401.htm

[115] S. Sevilla, M. Johnson, P. Kosakanchit, J. Liang, and K. Heimerl, "Experiences: Design, Implementation, and Deployment of CoLTE, a Community LTE Solution," in *Proc. of ACM Intl. Conf. on Mobile Computing and Networking (MobiCom)*, Los Cabos, Mexico, October 2019.

[116] bcom. https://5g.labs.b-com.com/. Accessed July 2020.

[117] G. Lee, J. Lee, J. Lee, Y. Im, M. Hollingsworth, E. Wustrow, D. Grunwald, and S. Ha, "This is Your President Speaking: Spoofing Alerts in 4G LTE Networks," in *Proc. of ACM Intl. Conf. on Mobile Systems, Applications, and Services (MobiSys)*, Seoul, South Korea, June 2019.

[118] NextEPC. https://nextepc.org. Accessed July 2020.

[119] Hewlett Packard Enterprise, "HPE Speeds Up 5G Adoption with Cloud Native 5G Core Software Stack, Available as-a-Service," March 2020. [Online]. Available: https://www.hpe.com/us/en/newsroom/press-release/2020/03/hpe-speeds-up-5g-adoption-with-cloud-native-5g-core-software-stack-available-as-a-service.html

[120] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-Independent Packet Processors," *SIGCOMM Computer Communication Review*, vol. 44, no. 3, p. 87–95, July 2014.

[121] R. Ricart-Sanchez, P. Malagon, P. Salva-Garcia, E. C. Perez, Q. Wang, and J. M. Alcaraz Calero, "Towards an FPGA-Accelerated Programmable Data Path for Edge-to-core Communications in 5G Networks," *Journal of Network and Computer Applications*, vol. 124, pp. 80–93, December 2018.

[122] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, and Q. Wang, "P4-NetFPGA-based Network Slicing Solution for 5G MEC Architectures," in *Proc. of ACM/IEEE Symp. on Architectures for Networking and Communications Systems (ANCS)*, Cambridge, United Kingdom, September 2019.

[123] Kaloom, "The Kaloom 5G User Plane Function (UPF)," https://www.mbuzzeurope.com/wp-content/uploads/2020/02/Product-Brief-Kaloom-5G-UPF-v1.0.pdf, accessed July 2020.

[124] Open Networking Foundation. Converged Multi-Access and Core (COMAC). https://opennetworking.org/comac. Accessed July 2020.

[125] ——. SD-RAN. https://opennetworking.org/sd-ran. Accessed July 2020.

[126] ——. Aether. https://opennetworking.org/aether. Accessed July 2020.

[127] Facebook Connectivity. Magma. https://connectivity.fb.com/magma. Accessed July 2020.

[128] OpenRAN 5G NR. https://telecominfraproject.com/5gnr. Accessed July 2020.

[129] Akraino Radio Edge Cloud. https://www.lfedge.org/projects-old/akraino/release-1/telco-appliance-radio-edge-cloud. Accessed July 2020.

[130] NVIDIA. Aerial SDK. https://developer.nvidia.com/aerial-sdk. Accessed July 2020.

[131] Fondazione Bruno Kessler (FBK). 5G-EmPOWER. https://5g-empower.io. Accessed July 2020.

[132] FlexRAN. http://mosaic-5g.io/flexran. Accessed July 2020.

[133] Open Networking Foundation. Central Office Re-architected as a Datacenter (CORD). https://opennetworking.org/cord. Accessed July 2020.

[134] LL-MEC. http://mosaic5g.io/ll-mec. Accessed July 2020.

[135] LightEdge. https://lightedge.io. Accessed July 2020.

[136] O-RAN Alliance White Paper, "O-RAN Use Cases and Deployment Scenarios," https://static1.squarespace.com/static/5ad774cce74940d7115044b0/t/5e95a0a306c6ab2d1cbca4d3/1586864301196/O-RAN+Use+Cases+and+Deployment+Scenarios+Whitepaper+February+2020.pdf, February 2020.

[137] 3GPP, "NG-RAN; Architecture Description," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.401, 01 2018, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/38401.htm

[138] O-RAN Fronthaul Working Group, "Control, User and Synchronization Plane Specification - v2.00," ORAN-WG4.CUS.0-v02.00 Technical Specification, 2019.

[139] O-RAN Working Group 2, "A1 Interface: General Aspects and Principles - v1.00," ORAN-WG2.A1.GA&P-v01.00 Technical Specification, 2019.

[140] O-RAN Working Group 1, "O-RAN Operations and Maintenance Interface - v2.00," O-RAN-WG1.O1-Interface-v02.00 Technical Specification, 2019.

[141] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," Internet Requests for Comments, RFC Editor, RFC 6241, June 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6241.txt

[142] O-RAN Working Group 3, "O-RAN Near-Real-time RAN Intelligent Controller Architecture & E2 General Aspects and Principles - v1.00," ORAN-WG3.E2GAP-v01.00 Technical Specification, 2020.

[143] O-RAN Working Group 1, "O-RAN Architecture Description - v1.00," O-RAN-WG1-O-RAN Architecture Description - v01.00.00 Technical Specification, 2020.

[144] 3GPP, "NG-RAN; E1 General Aspects and Principles," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.460, 01 2019, version 16.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/38460.htm

[145] ——, "NG-RAN; F1 General Aspects and Principles," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.470, 01 2018, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/38470.htm

[146] O-RAN Software Community. Amber Release. https://wiki.o-ran-sc.org/pages/viewpage.action?pageId=14221337. Accessed July 2020.

[147] ——. Bronze Release. https://wiki.o-ran-sc.org/pages/viewpage.action?pageId=14221635. Accessed July 2020.

[148] Open Networking Foundation. Component Projects List. https://wiki.opennetworking.org/display/COM/Component+Projects. Accessed July 2020.

[149] ——. SDN-Enabled Broadband Access (SEBA). https://opennetworking.org/seba. Accessed July 2020.

[150] ——. OpenCORD Guide. https://guide.opencord.org. Accessed July 2020.

[151] ——. OpenCORD Repositories. https://github.com/opencord. Accessed July 2020.

[152] ——. Exemplar Platform List. https://wiki.opennetworking.org/display/COM/Exemplar+Platforms. Accessed July 2020.

[153] ——. COMAC Release. https://guide.opencord.org/profiles/comac/release-notes.html. Accessed July 2020.

[154] ——. COMAC in a Box. https://guide.opencord.org/profiles/comac/install/ciab.html. Accessed July 2020.

[155] E. Coronado, S. N. Khan, and R. Riggio, "5G-EmPOWER: A Software-Defined Networking Platform for 5G Radio Access Networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 715–728, April 2019.

[156] Smart Networks and Services (SENSE) Research Unit at Fondazione Bruno Kessler (FBK). 5G-EmPOWER Repositories. https://github.com/5g-empower. Accessed July 2020.

[157] Mosaic5G Community. Mosaic5G and FlexRAN Projects Repositories. https://gitlab.eurecom.fr/mosaic5g. Accessed July 2020.

[158] N. Nikaein, X. Vasilakos, and A. Huang, "LL-MEC: Enabling Low Latency Edge Applications," in *Proc. of IEEE Intl. Conf. on Cloud Networking (CloudNet)*, Tokyo, Japan, October 2018.

[159] E. Coronado, Z. Yousaf, and R. Riggio, "LightEdge: Mapping the Evolution of Multi-Access Edge Computing in Cellular Networks," *IEEE Communications Magazine*, vol. 58, no. 4, pp. 24–30, April 2020.

[160] LightEdge. LightEdge Repositories. https://github.com/lightedge. Accessed July 2020.

[161] G. Brown, "TIP OpenRAN: Toward Disaggregated Mobile Networking," https://cdn.brandfolder.io/D8DI15S7/as/qc19tk-54bsw-305pae/TIP_OpenRAN_-Heavy_Reading_May_2020-_White_Paper.pdf, May 2020, accessed July 2020.

[162] OpenRAN. https://telecominfraproject.com/openran. Accessed July 2020.

[163] Telecom Infra Project, "OpenRAN 5G NR Base Station Platform Requirements Document," https://cdn.brandfolder.io/D8DI15S7/as/q688zl-7ly8w8-4xfk0m/TIP_OpenRAN_5GNR_Requirements_Document.pdf, 2020, accessed July 2020.

[164] Akraino Radio Edge Cloud Blueprint. https://wiki.akraino.org/pages/viewpage.action?pageId=6128402. Accessed July 2020.

[165] Akraino Edge Stack Repositories. https://github.com/akraino-edge-stack. Accessed July 2020.

[166] Akraino Telco Appliance Blueprint Family. https://wiki.akraino.org/display/AK/Telco+Appliance+Blueprint+Family. Accessed July 2020.

[167] G. M. Yilma, F. Z. Yousaf, V. Sciancalepore, and X. Costa-Perez, "On the Challenges and KPIs for Benchmarking Open-Source NFV MANO Systems: OSM vs ONAP," *arXiv preprint arXiv:1904.10697 [cs.NI]*, April 2019.

[168] P. Trakadas, P. Karkazis, H. C. Leligou, T. Zahariadis, F. Vicens, A. Zurita, P. Alemany, T. Soenen, C. Parada, J. Bonnet, E. Fotopoulou, A. Zafeiropoulos, E. Kapassa, M. Touloupou, and D. Kyriazis, "Comparison of Management and Orchestration Solutions for the 5G Era," *Journal of Sensor and Actuator Networks*, vol. 9, no. 1, January 2020.

[169] C. Rotsos, D. King, A. Farshad, J. Bird, L. Fawcett, N. Georgalas, M. Gunkel, K. Shiomoto, A. Wang, A. Mauthe, N. Race, and D. Hutchison, "Network Service Orchestration Standardization: A Technology Survey," *Computer Standards & Interfaces*, vol. 54, pp. 203–215, November 2017.

[170] L. Mamushiane, A. A. Lysko, T. Mukute, J. Mwangama, and Z. D. Toit, "Overview of 9 Open-Source Resource Orchestrating ETSI MANO Compliant Implementations: A Brief Survey," in *Proc. of IEEE Wireless Africa Conference (WAC)*, Pretoria, South Africa, August 2019.

[171] Open Source MANO End User Advisory Group, "OSM Scope, Functionality, Operation and Integration Guidelines," https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_Scope_and_Functionality.pdf, February 2019.

[172] G. A. Carella, M. Pauls, T. Magedanz, M. Cilloni, P. Bellavista, and L. Foschini, "Prototyping NFV-based Multi-access Edge Computing in 5G Ready Networks with Open Baton," in *Proc. of IEEE Conf. on Network Softwarization (NetSoft)*, Bologna, Italy, July 2017.

[173] A. J. Gonzalez, G. Nencioni, A. Kamisiński, B. E. Helvik, and P. E. Heegaard, "Dependability of the NFV Orchestrator: State of the Art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3307–3329, Fourth quarter 2018.

[174] Canonical. Linux Container (LXC). https://linuxcontainers.org. Accessed July 2020.

[175] Docker. https://www.docker.com. Accessed July 2020.

[176] J. Martins, M. Ahmed, C. Raiciu, and F. Huici, "Enabling Fast, Dynamic Network Processing with ClickOS," in *Proc. of ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, Hong Kong, China, August 2013.

[177] A. Bratterud, A.-A. Walla, H. Haugerud, P. E. Engelstad, and K. Begnum, "IncludeOS: A Minimal, Resource Efficient Unikernel for Cloud Services," in *Proc. of IEEE Intl. Conf. on Cloud Computing Technology and Science (CloudCom)*, Vancouver, BC, Canada, December 2015.

[178] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har'El, D. Marti, and V. Zolotarov, "OSv: Optimizing the Operating System for Virtual Machines," in *Proc. of USENIX Annual Technical Conf.*, Philadelphia, PA, USA, June 2014.

[179] Xen and Linux Foundation. MirageOS. https://github.com/mirage/mirage. Accessed July 2020.

[180] UniK. https://github.com/solo-io/unik. Accessed July 2020.

[181] S. Wu, C. Mei, H. Jin, and D. Wang, "Android Unikernel: Gearing Mobile Code Offloading Towards Edge Computing," *Future Generation Computer Systems*, vol. 86, pp. 694–703, April 2018.

[182] P. Valsamas, S. Skaperas, and L. Mamatas, "Elastic Content Distribution Based on Unikernels and Change-Point Analysis," in *Proc. of European Wireless Conf. (EW)*, Catania, Italy, May 2018.

[183] J. B. Filipe, F. Meneses, A. U. Rehman, D. Corujo, and R. L. Aguiar, "A Performance Comparison of Containers and Unikernels for Reliable 5G Environments," in *Proc. of IEEE Intl. Conf. on the Design of Reliable Communication Networks (DRCN)*, Coimbra, Portugal, March 2019.

[184] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, October 2003.

[185] VMware. ESXi. https://www.vmware.com/products/esxi-and-esx.html. Accessed July 2020.

[186] Linux Kernel-based Virtual Machine (KVM). https://www.linux-kvm.org. Accessed July 2020.

[187] BSD. bhyve. https://bhyve.org. Accessed July 2020.

[188] Oracle. VirtualBox. https://www.virtualbox.org. Accessed July 2020.

[189] OpenStack Project. OpenStack. https://www.openstack.org. Accessed July 2020.

[190] Linux Foundation. Kubernetes. https://kubernetes.io. Accessed July 2020.

[191] Istio. https://istio.io. Accessed July 2020.

[192] Network Service Mesh (NSM). https://networkservicemesh.io. Accessed July 2020.

[193] Linux Foundation, "ONAP Dublin Release," https://www.linuxfoundation.org/press-release/2019/07/onap-doubles-down-on-deployments-drives-commercial-activity-across-open-source-networking-stack-with-dublin-release, July 2019.

[194] ——, "ONAP Frankfurt Release," https://www.onap.org/announcement/2020/06/18/onaps-6th-release-frankfurt-available-now-most-comprehensive-secure-and-collaborative-software-to-accelerate-5g-deployments, June 2020.

[195] A. Kapadia, *ONAP Demystified: Automate Network Services with ONAP*. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2018.

[196] Consul, "Service Mesh and Microservice Networking," https://www.datocms-assets.com/2885/1536681707-consulwhitepaperaug2018.pdf, August 2018.

[197] F. Slim, F. Guillemin, A. Gravey, and Y. Hadjadj-Aoul, "Towards a Synamic Adaptive Placement of Virtual Network Functions Under ONAP," in *Proc. of IEEE Conf. on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Berlin, Germany, 2017.

[198] V. Q. Rodriguez, F. Guillemin, and A. Boubendir, "5G E2E Network Slicing Management with ONAP," in *Proc. of IEEE Conf. on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, Paris, France, February 2020.

[199] ONAP. Mobility Standards Harmonization with ONAP. https://wiki.onap.org/display/DW/MOBILITY+STANDARDS+HARMONIZATION+WITH+ONAP. Accessed July 2020.

[200] Canonical. Juju Charms. https://jaas.ai. Accessed July 2020.

[201] Open Source MANO End User Advisory Group, "OSM Deployment and Integration," https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_Deployment_and_Integration.pdf, February 2020.

[202] R. Casellas, R. Martínez, R. Vilalta, and R. Muñoz, "Metro-Haul: SDN Control and Orchestration of Disaggregated Optical Networks with Model-Driven Development," in *Proc. of IEEE Intl. Conf. on Transparent Optical Networks (ICTON)*, July 2018.

[203] T. Soenen, W. Tavernier, M. Peuster, F. Vicens, G. Xilouris, S. Kolometsos, M. Kourtis, and D. Colle, "Empowering Network Service Developers: Enhanced NFV DevOps and Programmable MANO," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 89–95, May 2019.

[204] C. Colman-Meixner, P. Diogo, M. S. Siddiqui, A. Albanese, H. Khalili, A. Mavromatis, L. Luca, A. Ulisses, J. Colom, R. Nejabati, and D. Simeonidou, "5G City: A Novel 5G-Enabled Architecture for Ultra-High Definition and Immersive Media on City Infrastructure," in *Proc. of IEEE Intl. Symp. on Broadband Multimedia Systems and Broadcasting (BMSB)*, Valencia, Spain, June 2018.

[205] S. Rizou, P. Athanasoulis, P. Andriani, F. Iadanza, G. Carrozzo, D. Breitgand, A. Weit, D. Griffin, D. Jimenez, U. Acar, and O. P. Gordo, "A Service Platform Architecture Enabling Programmable Edge-To-Cloud Virtualization for the 5G Media Industry," in *Proc. of IEEE Intl. Symp. on Broadband Multimedia Systems and Broadcasting (BMSB)*, Valencia, Spain, June 2018.

[206] A. de la Oliva, X. Li, X. Costa-Perez, C. J. Bernardos, P. Bertin, P. Iovanna, T. Deiss, J. Mangues, A. Mourad, C. Casetti, J. E. Gonzalez, and A. Azcorra, "5G-TRANSFORMER: Slicing and Orchestrating Transport Networks for Industry Verticals," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 78–84, August 2018.

[207] T. Dreibholz, "Flexible 4G/5G Testbed Setup for Mobile Edge Computing Using OpenAirInterface and Open Source MANO," in *Springer Web, Artificial Intelligence and Network Applications (WAINA)*, March 2020.

[208] C. Tranoris, S. Denazis, L. Guardalben, J. Pereira, and S. Sargento, "Enabling Cyber-Physical Systems for 5G Networking: A Case Study on the Automotive Vertical Domain," in *Proc. of the IEEE/ACM Intl. Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*, Gothenburg, Sweden, 2018.

[209] Open Baton. Open Baton Project. https://openbaton.github.io. Accessed July 2020.

[210] ——. Open Baton Repositories. https://github.com/openbaton. Accessed July 2020.

[211] Skylark Wireless. https://www.skylarkwireless.com. Accessed July 2020.

[212] Argos. http://argos.rice.edu/. Accessed July 2020.

[213] R. Doost-Mohammady, O. Bejarano, and A. Sabharwal, "Good Times for Wireless Research," *Computer Networks*, vol. 188, pp. 1–9, April 2021.

[214] Platform for Open Wireless Data-driven Experimental Research (POWDER). https://www.powderwireless.net. Accessed July 2020.

[215] Reconfigurable Eco-system for Next-generation End-to-end Wireless (RENEW). https://renew.rice.edu. Accessed July 2020.

[216] Cloud Enhanced Open Software Defined Mobile Wireless Testbed for City-Scale Deployment (COSMOS). http://www.cosmos-lab.org. Accessed July 2020.

[217] D. Raychaudhuri, I. Seskar, G. Zussman, T. Korakis, D. Kilper, T. Chen, J. Kolodziejski, M. Sherman, Z. Kostic, X. Gu, H. Krishnaswamy, S. Maheshwari, P. Skrimponis, and C. Gutterman, "Challenge: COSMOS: A City-Scale Programmable Testbed for Experimentation with Advanced Wireless," in *Proceedings of ACM MobiCom*, London, United Kingdom, September 2020.

[218] M. Kohli, T. Chen, M. B. Dastjerdi, J. Welles, I. Seskar, H. Krishnaswamy, and G. Zussman, "Open-Access Full-Duplex Wireless in the ORBIT and COSMOS Testbeds," *Computer Networks*, 2021.

[219] Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT). https://www.orbit-lab.org. Accessed July 2020.

[220] O-RAN Alliance Conducts First Global Plugfest to Foster Adoption of Open and Interoperable 5G Radio Access Networks. (2019, December) https://static1.squarespace.com/static/5ad774cce74940d7115044b0/t/5dfba8fb1326ae1bcf4a8b6f/1576773884092/O-RAN-2019.12.19-EC-C-PR-on-2019-Plugfest-v1.0.pdf.

[221] Aerial Experimentation and Research Platform for Advanced Wireless (AERPAW). https://www.aerpaw.org. Accessed July 2020.

[222] V. Marojevic, I. Guvenc, M. Sichitiu, and R. Dutta, "An Experimental Research Platform Architecture for UAS Communications and Networking," in *Proc. of IEEE Vehicular Technology Conference (VTC2019-Fall)*, Honolulu, HI, USA, September 2019.

[223] M. Sichitiu, I. Guvenc, R. Dutta, V. Marojevic, and B. Floyd, "AERPAW Emulation Overview," in *Proc. of ACM Intl. Workshop on Wireless Network Testbeds, Experimental evaluation & CHaracterization (WiNTECH)*, London, United Kingdom, September 2020.

[224] H. Zhang, Y. Guan, A. Kamal, D. Qiao, M. Zheng, A. Arora, O. Boyraz, B. Cox, T. Daniels, M. Darr, D. Jacobson, A. Khokhar, S. Kim, J. Koltes, J. Liu, M. Luby, L. Nadolny, J. Peschel, P. Schnable, A. Sharma, A. Somani, and L. Tang, "ARA: A Wireless Living Lab Vision for Smart and Connected Rural Communities," in *Proceedings of ACM WiNTECH*, New Orleans, LA, USA, October 2021.

[225] 5TONIC. https://www.5tonic.org. Accessed July 2020.

[226] B. Nogales, I. Vidal, D. R. Lopez, J. Rodriguez, J. Garcia-Reinoso, and A. Azcorra, "Design and Deployment of an Open Management and Orchestration Platform for Multi-Site NFV Experimentation," *IEEE Communications Magazine*, vol. 57, no. 1, pp. 20–27, 2019.

[227] M. J. Roldan, P. Leithead, and J. Mack, "Experiments and Results of a mmW Transport Platform to Enable 5G Cloud RAN Lower Layer Splits," in *Proc. of IEEE Long Island Systems, Applications and Technology Conf. (LISAT)*, Farmingdale, NY, USA, 2018.

[228] Horizon 2020. https://ec.europa.eu/programmes/horizon2020. Accessed July 2020.

[229] Federation 4 Future Internet Research and Experimentation Plus (FED4FIRE+). https://www.fed4fire.eu. Accessed July 2020.

[230] NITOS. https://nitlab.inf.uth.gr/NITlab/nitos. Accessed July 2020.

[231] Icarus Node. https://nitlab.inf.uth.gr/NITlab/hardware/wireless-nodes/icarus-nodes. Accessed July 2020.

[232] P. Karamichailidis, K. Choumas, and T. Korakis, "Enabling Multi-Domain Orchestration using Open Source MANO, OpenStack and OpenDaylight," in *Proc. of IEEE Intl. Symp. on Local and Metropolitan Area Networks (LANMAN)*, Paris, France, July 2019.

[233] K. Chounos, N. Makris, and T. Korakis, "Enabling Distributed Spectral Awareness for Disaggregated 5G Ultra-Dense HetNets," in *Proc. of IEEE 5G World Forum (5GWF)*, Dresden, Germany, September 2019.

[234] N. Makris, V. Passas, C. Nanis, and T. Korakis, "On Minimizing Service Access Latency: Employing MEC on the Fronthaul of Heterogeneous 5G Architectures," in *Proc. of IEEE Intl. Symp. on Local and Metropolitan Area Networks (LANMAN)*, Paris, France, July 2019.

[235] V. Passas, N. Makris, V. Miliotis, and T. Korakis, "Pricing Based MEC Resource Allocation for 5G Heterogeneous Network Access," in *Proc. of IEEE Global Communications Conf. (GLOBECOM)*, Waikoloa, HI, USA, December 2019.

[236] IRIS. http://iristestbed.eu. Accessed July 2020.

[237] COgnitive Radio NETwork (CORNET). https://cornet.wireless.vt.edu. Accessed July 2020.

[238] R. M. Rao, V. Marojevic, and J. H. Reed, "Analysis of Non-Pilot Interference on Link Adaptation and Latency in Cellular Networks," in *Proc. of IEEE Vehicular Technology Conf. (VTC2019-Spring)*, Kuala Lumpur, Malaysia, April 2019.

[239] Future Internet of Things (FIT). https://fit-equipex.fr. Accessed July 2020.

[240] Drexel Grid SDR Testbed. https://research.coe.drexel.edu/ece/dwsl/research/drexel-grid-sdr-testbed. Accessed July 2020.

[241] K. R. Dandekar, S. Begashaw, M. Jacovic, A. Lackpour, I. Rasheed, X. R. Rey, C. Sahin, S. Shaher, and G. Mainland, "Grid Software Defined Radio Network Testbed for Hybrid Measurement and Emulation," in *Proceedings of IEEE SECON*, Boston, MA, USA, June 2019.

[242] M. Mezzavilla, M. Zhang, M. Polese, R. Ford, S. Dutta, S. Rangan, and M. Zorzi, "End-to-End Simulation of 5G mmWave Networks," *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2237–2263, April 2018.

[243] N. Patriciello, S. Lagen, B. Bojovic, and L. Giupponi, "An E2E Simulator for 5G NR Networks," *Simulation Modelling Practice and Theory*, vol. 96, no. 101933, November 2019.

[244] S. Pratschner, B. Tahir, L. Marijanovic, M. Mussbah, K. Kirev, R. Nissel, S. Schwarz, and M. Rupp, "Versatile Mobile Communications Simulation: The Vienna 5G Link Level Simulator," *Springer EURASIP Journal on Wireless Communications and Networking*, vol. 2018, no. 1, p. 226, September 2018.

[245] E. J. Oughton, K. Katsaros, F. Entezami, D. Kaleshi, and J. Crowcroft, "An Open-Source Techno-Economic Assessment Framework for 5G Deployment," *IEEE Access*, vol. 7, pp. 155 930–155 940, October 2019.

[246] Pi-Radio. Fully-digital mmWave Front-ends. https://www.pi-rad.io/home/product. Accessed July 2020.

[247] A. Dhananjay, K. Zheng, J. Haarla, L. Iotti, M. Mezzavilla, D. Shasha, and S. Rangan, "Calibrating a 4-channel Fully-Digital 60 GHz SDR," in *Proc. of ACM Intl. Workshop on Wireless Network Testbeds, Experimental evaluation & CHaracterization (WiNTECH)*, London, United Kingdom, September 2020.

[248] M. Polese, F. Restuccia, A. Gosain, J. Jornet, S. Bhardwaj, V. Ariyarathna, S. Mandal, K. Zheng, A. Dhananjay, M. Mezzavilla, J. Buckwalter, M. Rodwell, X. Wang, M. Zorzi, A. Madanayake, and T. Melodia, "MillimeTera: Toward A Large-Scale Open-Source mmWave and Terahertz Experimental Testbed," in *Proc. of ACM Workshop on Millimeter-Wave Networks and Sensing Systems (mmNets)*, Los Cabos, Mexico, 2019.

[249] J. Haarla, V. Semkin, K. Zheng, A. Dhananjay, M. Mezzavilla, J. Ala-Laurinaho, and V. Viikari, "Characterizing 60 GHz Patch Antenna Segments for Fully Digital Transceiver," in *Proc. of the 14th European Conference on Antennas and Propagation (EuCAP)*, Copenhagen, Denmark, 2020.

[250] R. Akeela and B. Dezfouli, "Software-defined Radios: Architecture, State-of-the-Art, and Challenges," *Computer Communications*, vol. 128, pp. 106–125, 2018.

[251] M. Savi, M. Tornatore, and G. Verticale, "Impact of Processing-Resource Sharing on the Placement of Chained Virtual Network Functions," *IEEE Transactions on Cloud Computing*, 2019.

[252] C. Cowan, "Software Security for Open-source Systems," *IEEE Security Privacy*, vol. 1, no. 1, pp. 38–45, January 2003.

[253] V. Piantadosi, S. Scalabrino, and R. Oliveto, "Fixing of Security Vulnerabilities in Open Source Projects: A Case Study of Apache HTTP Server and Apache Tomcat," in *Proc. of IEEE Conf. on Software Testing, Validation and Verification (ICST)*, Xi'an, China, 2019.

[254] P. Wang, J. Krinke, K. Lu, G. Li, and S. Dodier-Lazaro, "How Double-Fetch Situations turn into Double-Fetch Vulnerabilities: A Study of Double Fetches in the Linux Kernel," in *Proc. of USENIX Security Symp. (USENIX Security)*, Vancouver, BC, Canada, August 2017.

[255] O-RAN Working Group 1, "O-RAN Architecture Description 5.00," O-RAN.WG1.O-RAN-Architecture-Description-v05.00 Technical Specification, July 2021.

[256] O-RAN Working Group 3, "O-RAN Near-RT RAN Intelligent Controller Near-RT RIC Architecture 2.00," O-RAN.WG3.RICARCH-v02.00, March 2021.

[257] O-RAN Working Group 2, "O-RAN Non-RT RIC Architecture 1.0," O-RAN.WG2.Non-RT-RIC-ARCH-TS-v01.00 Technical Specification, July 2021.

[258] ——, "O-RAN AI/ML workflow description and requirements 1.03," O-RAN.WG2.AIML-v01.03 Technical Specification, July 2021.

[259] M. Dryjański, Ł. Kułacz, and A. Kliks, "Toward Modular and Flexible Open RAN Implementations in 6G Networks: Traffic Steering Use Case and O-RAN xApps," *Sensors*, vol. 21, no. 24, pp. 1–14, December 2021.

[260] D. Johnson, D. Maas, and J. Van Der Merwe, "Open Source RAN Slicing on POWDER: A Top-to-Bottom O-RAN Use Case," in *Proceedings of ACM MobiSys*, Virtual Conference, June 2021.

[261] H. Lee, J. Cha, D. Kwon, M. Jeong, and I. Park, "Hosting AI/ML Workflows on O-RAN RIC Platform," in *Proceedings of IEEE GLOBECOM Workshops*, Taipei, Taiwan, December 2020.

[262] A. S. Abdalla, P. S. Upadhyaya, V. K. Shah, and V. Marojevic, "Toward Next Generation Open Radio Access Network–What O-RAN Can and Cannot Do!" *IEEE Network Magazine*, May 2022.

[263] O-RAN Software Community. O-DU L2 Repository. https://github.com/o-ran-sc/o-du-l2. Accessed December 2021.

[264] David Johnson. POWDER RIC Profile Repository. https://gitlab.flux.utah.edu/johnsond/ric-profile. Accessed December 2021.

[265] 3GPP, "Evolved universal terrestrial radio access (E-UTRA); physical layer procedures," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.213, 7 2020, version 16.2.0. [Online]. Available: http://www.3gpp.org/DynaReport/36213.htm

[266] Colosseum. https://www.colosseum.net. Accessed July 2020.

[267] Unwired Labs, "OpenCelliD," accessed September 2021. [Online]. Available: https://opencellid.org

[268] U.S. Naval Research Laboratory. MGEN Traffic Emulator. https://www.nrl.navy.mil/Our-Work/Areas-of-Research/Information-Technology/NCS/MGEN. Accessed September 2021.

[269] DARPA. Spectrum Collaboration Challenge. https://www.darpa.mil/program/spectrum-collaboration-challenge. Accessed September 2021.

[270] POWDER Deployment. (2021) https://www.powderwireless.net/area.

[271] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, 2019.

[272] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level Control through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, February 2015.

[273] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[274] W. Wang, Y. Huang, Y. Wang, and L. Wang, "Generalized autoencoder: A neural network framework for dimensionality reduction," in *Proc. of IEEE/CVF CVPR Workshops*, 2014, pp. 490–497.

[275] Canonical. (2021) LXD Network Configuration. https://linuxcontainers.org/lxd/docs/master/networks.

[276] Software Radio Systems. (Accessed May 2021) srsLTE (srsRAN) Repository. https://github.com/srsran/srsRAN.

[277] G. Garcia-Aviles, M. Gramaglia, P. Serrano, and A. Banchs, "POSENS: A Practical Open Source Solution for End-to-End Network Slicing," *IEEE Wireless Communications*, vol. 25, no. 5, pp. 30–37, October 2018.

[278] G. Garcia-Aviles, M. Gramaglia, P. Serrano, F. Gringoli, S. Fuente-Pascual, and I. Labrador Pavon, "Experimenting with Open Source Tools to Deploy a Multi-service and Multi-slice Mobile Network," *Computer Communications*, vol. 150, pp. 1–12, January 2020.

[279] J. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Pérez, A. Banchs, and J. Alcaraz, "VrAIn: A Deep Learning Approach Tailoring Computing and Radio Resources in Virtualized RANs," in *Proceedings of ACM MobiCom*, Los Cabos, Mexico, October 2019.

[280] E. Coronado, S. Khan, and R. Riggio, "5G-EmPOWER: A Software-Defined Networking Platform for 5G Radio Access Networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 715–728, June 2019.

[281] K. Koutlia, R. Ferrús, E. Coronado, R. Riggio, F. Casadevall, A. Umbert, and J. Pérez-Romero, "Design and Experimental Validation of a Software-Defined Radio Access Network Testbed with Slicing Support," *Wireless Communications and Mobile Computing*, pp. 1–17, June 2019.

[282] S. Marinova, T. Lin, H. Bannazadeh, and A. Leon-Garcia, "End-to-end Network Slicing for Future Wireless in Multi-region Cloud Platforms," *Computer Networks*, vol. 177, pp. 1–10, August 2020.

[283] O-RAN Working Group 2, "O-RAN AI/ML Workflow Description and Requirements - v1.01," Technical Specification, 2020.

[284] J. Wang, C. Jiang, H. Zhang, Y. Ren, K.-C. Chen, and L. Hanzo, "Thirty Years of Machine Learning: The Road to Pareto-Optimal Wireless Networks," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1472–1514, Third quarter 2020.

[285] O-RAN Software Community. O-DU-L2 Documentation. https://docs.o-ran-sc.org/projects/o-ran-sc-o-du-l2/en/latest/index.html. Accessed July 2021.

[286] F. Capozzi, G. Piro, L. Grieco, G. Boggia, and P. Camarda, "Downlink Packet Scheduling in LTE Cellular Networks: Key Design Issues and a Survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 678–700, June 2013.

[287] O-RAN Working Group 3, "O-RAN Near-Real-time RAN Intelligent Controller E2 Service Model (E2SM) KPM 1.0," ORAN-WG3.E2SM-KPM-v01.00.00 Technical Specification, February 2020.

[288] O-RAN Software Community. xApp Framework. https://wiki.o-ran-sc.org/display/ORANSDK/xAppFramework. Accessed July 2021.

[289] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," March 2016.

[290] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo, "TF-Agents: A library for Reinforcement Learning in TensorFlow," https://github.com/tensorflow/agents, 2018. [Online]. Available: https://github.com/tensorflow/agents

[291] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347 [cs.LG]*, July 2017.

[292] 3GPP, "5G Performance Measurements," Technical Specification (TS) 28.552, June 2021, version 17.3.1.

[293] ——, "Performance Measurements Evolved Universal Terrestrial Radio Access Network (E-UTRAN)," Technical Specification (TS) 32.425, June 2021, version 17.1.0.

[294] M. Sakurada and T. Yairi, "Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction," in *Proceedings of the 2nd Workshop on Machine Learning for Sensory Data Analysis*, Gold Coast, Australia QLD, Australia, December 2014.

[295] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement Learning with Deep Energy-Based Policies," in *Proc. of the 34th Intl. Conf. on Machine Learning*, Sydney, NSW, Australia, July 2017.

[296] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to Communicate: Channel Auto-encoders, Domain Specific Regularizers, and Attention," in *Proceedings of IEEE ISSPIT*, Limassol, Cyprus, December 2016.

[297] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Wanna Make Your TCP Scheme Great for Cellular Networks? Let Machines Do It for You!" *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 265–279, January 2021.

[298] M. G. Kibria, K. Nguyen, G. P. Villardi, O. Zhao, K. Ishizu, and F. Kojima, "Big Data Analytics, Machine Learning, and Artificial Intelligence in Next-Generation Wireless Networks," *IEEE Access*, vol. 6, pp. 32 328–32 338, 2018.

[299] Y. Sun, M. Peng, Y. Zhou, Y. Huang, and S. Mao, "Application of Machine Learning in Wireless Networks: Key Techniques and Open Issues," *IEEE Communication Surveys & Tutorials*, vol. 21, no. 4, pp. 3072–3108, June 2019.

[300] D. Gunduz, P. de Kerret, N. D. Sidiropoulos, D. Gesbert, C. R. Murthy, and M. van der Schaar, "Machine Learning in the Air," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 10, pp. 2184–2199, October 2019.

[301] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial Neural Networks-Based Machine Learning for Wireless Networks: A Tutorial," *IEEE Communication Surveys & Tutorials*, vol. 21, no. 4, pp. 3039–3071, July 2019.

[302] C. Jiang, H. Zhang, Y. Ren, Z. Han, K.-C. Chen, and L. Hanzo, "Machine Learning Paradigms for Next-Generation Wireless Networks," *IEEE Wireless Communications*, vol. 24, no. 2, pp. 98–105, April 2017.

[303] Y. Fu, S. Wang, C.-X. Wang, X. Hong, and S. McLaughlin, "Artificial Intelligence to Manage Network Traffic of 5G Wireless Networks," *IEEE Network*, vol. 32, no. 6, pp. 58–64, November 2018.

[304] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L.-C. Wang, "Deep Reinforcement Learning for Mobile 5G and Beyond: Fundamentals, Applications, and Challenges," *IEEE Vehicular Technology Magazine*, vol. 14, no. 2, pp. 44–52, June 2019.

[305] E. Perenda, S. Rajendran, G. Bovet, S. Pollin, and M. Zheleva, "Learning the Unknown: Improving Modulation Classification Performance in Unseen Scenarios," in *Proceedings of IEEE INFOCOM*, Virtual Conference, May 2021.

[306] Y. Huang, T. Hou, and W. Lou, "A Deep-Learning-based Link Adaptation Design for eMBB/URLLC Multiplexing in 5G NR," in *Proceedings of IEEE INFOCOM*, Virtual Conference, May 2021.

[307] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "DeepCog: Optimizing Resource Provisioning in Network Slicing with AI-based Capacity Forecasting," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 361–376, February 2020.

[308] J. Wang, J. Tang, Z. Xu, Y. Wang, G. Xue, X. Zhang, and D. Yang, "Spatiotemporal Modeling and Prediction in Cellular Networks: A Big Data Enabled Deep Learning Approach," in *Proceedings of IEEE INFOCOM*, Atlanta, GA, USA, May 2017, pp. 1–9.

[309] J. Chuai, Z. Chen, G. Liu, X. Guo, X. Wang, X. Liu, C. Zhu, and F. Shen, "A Collaborative Learning Based Approach for Parameter Configuration of Cellular Networks," in *Proceedings of IEEE INFOCOM*, Paris, France, April 2019.

[310] N. Naderializadeh, J. J. Sydir, M. Simsek, and H. Nikopour, "Resource Management in Wireless Networks via Multi-Agent Deep Reinforcement Learning," *IEEE Transactions of Wireless Communications*, vol. 20, no. 6, pp. 3507–3523, June 2021.

[311] Z. Wang, L. Li, Y. Xu, H. Tian, and S. Cui, "Handover Control in Wireless Systems via Asynchronous Multiuser Deep Reinforcement Learning," *IEEE Internet Things Journal*, vol. 5, no. 6, pp. 4296–4307, December 2018.

[312] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep Reinforcement Learning for Dynamic Multichannel Access in Wireless Networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 2, pp. 257–265, June 2018.

[313] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep Reinforcement Learning for User Association and Resource Allocation in Heterogeneous Cellular Networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 11, pp. 5141–5152, November 2019.

[314] S. Mollahasani, M. Erol-Kantarci, M. Hirab, H. Dehghan, and R. Wilson, "Actor-Critic Learning Based QoS-Aware Scheduler for Reconfigurable Wireless Networks," *IEEE Transactions on Network Science and Engineering*, pp. 1–10, April 2021.

[315] H. Zhou, M. Elsayed, and M. Erol-Kantarci, "RAN Resource Slicing in 5G Using Multi-Agent Correlated Q-Learning," in *Proceedings of IEEE PIMRC*, Virtual Conference, September 2021.

[316] S. Chinchali, P. Hu, T. Chu, M. Sharma, M. Bansal, R. Misra, M. Pavone, and S. Katti, "Cellular Network Traffic Scheduling with Deep Reinforcement Learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, April 2018.

[317] S. Niknam, A. Roy, H. S. Dhillon, S. Singh, R. Banerji, J. H. Reed, N. Saxena, and S. Yoon, "Intelligent O-RAN for Beyond 5G and 6G Wireless Networks," *arXiv:2005.08374 [eess.SP]*, May 2020.

[318] P. Tilghman, "AI Will Rule the Airwaves: A DARPA Grand Challenge Seeks Autonomous Radios to Manage the Wireless Spectrum," *IEEE Spectrum*, vol. 56, no. 6, pp. 28–33, May 2019.

[319] R. L. Yuan and K. M. Schmidt, "Defense Advanced Research Projects Agency Spectrum Collaboration Challenge at APL: Introduction," vol. 35, no. 1, pp. 2–3, 2019.

[320] D. Coleman, K. R. McKeever, M. L. Mohr, L. R. Orraca Rosario, K. E. Parker, and M. L. Plett, "Overview of the Colosseum: The World's Largest Test Bed for Radio Experiments," *Johns Hopkins APL Technical Digest*, vol. 35, no. 1, pp. 4–11, 2019.

[321] A. S. Freeman, A. T. Plummer, Jr., J. N. Kraus, M. L. Plett, K. E. Parker, and D. M. Coleman, "Software Project Management for the Defense Advanced Research Projects Agency Spectrum Collaboration Challenge," *Johns Hopkins APL Technical Digest*, vol. 35, no. 1, pp. 12–21, 2019.

[322] A. T. Plummer, Jr. and K. P. Taylor, "Development and Operations on the Defense Advanced Research Project Agency's Spectrum Collaboration Challenge," *Johns Hopkins APL Technical Digest*, vol. 35, no. 1, pp. 22–33, 2019.

[323] J. W. Mok, A. L. Hom, J. J. Uher, and D. M. Coleman, "The Resource Manager for the Defense Advanced Research Projects Agency Spectrum Collaboration Challenge Test Bed," *Johns Hopkins APL Technical Digest*, vol. 35, no. 1, pp. 34–41, 2019.

[324] D. A. White, Jr., J. E. Annis, and F. F. Johnson, "Standard Radio Nodes in the Defense Advanced Research Projects Agency Spectrum Collaboration Challenge," vol. 35, no. 1, pp. 42–48, 2019.

[325] K. J. Yim, K. R. McKeever, and D. R. Barcklow, "Incumbent Radio Systems in the Defense Advanced Research Projects Agency Spectrum Collaboration Challenge Test Bed," vol. 35, no. 1, pp. 49–57, 2019.

[326] P. D. Curtis, A. T. Plummer, Jr., J. E. Annis, and W. J. La Cholter, "Traffic Generation System for the Defense Advanced Research Projects Agency Spectrum Collaboration Challenge," *Johns Hopkins APL Technical Digest*, vol. 35, no. 1, pp. 58–68, 2019.

[327] D. Barcklow, L. Bloch, S. Sweeney, B. Ahr, W. La Cholter, S. Berhanu, H. Bisyir, J. Cook, and D. Coleman, "Radio Frequency Emulation System for the Defense Advanced Research Projects Agency Spectrum Collaboration Challenge," *Johns Hopkins APL Technical Digest*, vol. 35, no. 1, pp. 69–78, 2019.

[328] A. Chaudhari and M. Braun, "A Scalable FPGA Architecture for Flexible, Large-Scale, Real-Time RF Channel Emulation," in *Proceedings of IEEE ReCoSoC*, Lille, France, July 2018.

[329] FCC. (2020, April) FCC Adopts New Rules for the 6 GHz Band, Unleashing 1,200 MHz of Spectrum for Unlicensed Use. https://docs.fcc.gov/public/attachments/DOC-363945A1.pdf. Accessed September 2021.

[330] IEEE 802.11 Wireless LAN Working Group, "IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 1: Enhancements for High-Efficiency WLAN," *IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020)*, pp. 1–767, May 2021.

[331] B. Bloessl, M. Segata, C. Sommer, and F. Dressler, "Performance Assessment of IEEE 802.11p with an Open Source SDR-based Prototype," *IEEE Transactions on Mobile Computing*, vol. 17, no. 5, pp. 1162–1175, May 2018.

[332] P. Yang, L. Kong, and G. Chen, "Spectrum Sharing for 5G/6G URLLC: Research Frontiers and Standards," *IEEE Communications Standards Magazine*, vol. 5, no. 2, pp. 120–125, June 2021.

[333] G. Naik and J.-M. J. Park, "Coexistence of Wi-Fi 6E and 5G NR-U: Can We Do Better in the 6 GHz Bands?" in *Proceedings of IEEE INFOCOM*, Vancouver, BC, Canada, May 2021.

[334] S. Mohanti, C. Bocanegra, J. Meyer, G. Secinti, M. Diddi, H. Singh, and K. Chowdhury, "AirBeam: Experimental Demonstration of Distributed Beamforming by a Swarm of UAVs," in *Proceedings of IEEE MASS*, Monterey, CA, USA, November 2019.

[335] J. Buczek, L. Bertizzolo, S. Basagni, and T. Melodia, "What is A Wireless UAV? A Design Blueprint for 6G Flying Wireless Nodes," in *Proceedings of ACM WiNTECH 2021*, New Orleans, LA, January 2022.

[336] A. Coletta, G. Maselli, M. Piva, and D. Silvestri, "DANGER: A Drones Aided Network for Guiding Emergency and Rescue Operations," in *Proceedings of ACM Mobihoc*, Virtual Conference, October 2020.

[337] C. Rottondi, F. Malandrino, A. Bianco, C. F. Chiasserini, and I. Stavrakakis, "Scheduling of Emergency Tasks for Multiservice UAVs in Post-disaster Scenarios," *Computer Networks*, vol. 184, pp. 1–13, January 2021.

[338] S. Rangan, T. S. Rappaport, and E. Erkip, "Millimeter-Wave Cellular Wireless Networks: Potentials and Challenges," *Proceedings of the IEEE*, vol. 102, no. 3, pp. 366–385, March 2014.

[339] The Innovation Institute at the MassTech Collaborative. (2020, February) AI Jumpstart. https://innovation.masstech.org/projects-and-initiatives/collaborative-research-matching-grant-program/ai-jumpstart. Accessed September 2021.

[340] M. Polese, F. Restuccia, and T. Melodia, "DeepBeam: Deep Waveform Learning for Coordination-Free Beam Management in mmWave Networks," in *Proceedings of ACM Mobihoc*, Shanghai, China, July 2021.

[341] NVIDIA. NVIDIA DGX A100—The Universal System for AI Infrastructure. https://www.nvidia.com/en-us/data-center/dgx-a100. Accessed September 2021.

[342] HashiCorp. Nomad. https://www.nomadproject.io. Accessed September 2021.

[343] A. R.-C. Datasheet. (2019) https://rubimages-liberty.netdna-ssl.com/spec/RG8-CMP%20Specification.pdf.

[344] W. E. Co. (2019) https://kb.ettus.com/images/9/9e/ettus_research_vert2450_datasheet.pdf.

[345] T. Lo, "Maximum Ratio Transmission," *IEEE Transactions on Communications*, vol. 47, no. 10, pp. 1458–1461, October 1999.

[346] E. G. Larsson, O. Edfors, F. Tufvesson, and T. L. Marzetta, "Massive MIMO for Next Generation Wireless Systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 186–195, February 2014.

[347] Z. Guan, L. Bertizzolo, E. Demirors, and T. Melodia, "WNOS: Enabling Principled Software-Defined Wireless Networking," *IEEE/ACM Transactions on Networking*, vol. 29, no. 3, pp. 1391–1407, June 2021.

[348] R. Zhang and Y. Liang, "Exploiting Multi-Antennas for Opportunistic Spectrum Sharing in Cognitive Radio Networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 1, pp. 88–102, February 2008.

[349] I. Akyildiz, W. Lee, M. Vuran, and S. Mohanty, "A Survey on Spectrum Management in Cognitive Radio Networks," *IEEE Communications Magazine*, vol. 46, no. 4, pp. 40–48, April 2008.

[350] D. Niyato and E. Hossain, "Competitive Spectrum Sharing in Cognitive Radio Networks: A Dynamic Game Approach," *IEEE Transactions on Wireless Communications*, vol. 7, no. 7, pp. 2651–2660, July 2008.

[351] T. O'Shea and J. Hoydis, "An Introduction to Deep Learning for the Physical Layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, Oct. 2017.

[352] H. Zhang, H. Zhou, and M. Erol-Kantarci, "Team Learning-Based Resource Allocation for Open Radio Access Network (O-RAN)," in *Proceedings of IEEE ICC*, Seoul, South Korea, May 2022.

[353] S. Bakri, P. A. Frangoudis, A. Ksentini, and M. Bouaziz, "Data-Driven RAN Slicing Mechanisms for 5G and Beyond," *IEEE Transactions on Network and Service Management*, July 2021.

[354] O. Orhan, V. N. Swamy, T. Tetzlaff, M. Nassar, H. Nikopour, and S. Talwar, "Connection Management xAPP for O-RAN RIC: A Graph Neural Network and Reinforcement Learning Approach," in *Proceedings of IEEE ICMLA*, 2021, pp. 936–941.

[355] Y. Shi, P. Rah imzadeh, M. Costa, T. Erpek, and Y. E. Sagduyu, "Deep Reinforcement Learning for 5G Radio Access Network Slicing with Spectrum Coexistence," *TechRxiv preprint TechRxiv.16632526*, September 2021.

[356] S. Shen, T. Zhang, S. Mao, and G.-K. Chang, "DRL-Based Channel and Latency Aware Radio Resource Allocation for 5G Service-Oriented RoF-mmWave RAN," *Journal of Lightwave Technology*, vol. 39, no. 18, pp. 5706–5714, September 2021.

[357] J. A. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, A. Banchs, and J. J. Alcaraz, "vrAIn: Deep Learning based Orchestration for Computing and Radio Resources in vRANs," *IEEE Transactions on Mobile Computing*, December 2020.

[358] T. Jian, Y. Gong, Z. Zhan, R. Shi, N. Soltani, Z. Wang, J. G. Dy, K. R. Chowdhury, Y. Wang, and S. Ioannidis, "Radio Frequency Fingerprinting on the Edge," *IEEE Transactions on Mobile Computing*, March 2021.

[359] A. Zhou, X. Zhang, and H. Ma, "Beam-forecast: Facilitating Mobile 60 GHz Networks via Model-driven Beam Steering," in *Proceedings of IEEE INFOCOM*, Atlanta, GA, USA, May 2017.

[360] Radisys. (2019) Open RAN - Enabling the O-RAN of the Future Today. hub.radisys.com/white-papers/open-ran-enabling-the-o-ran-future.

[361] Telecom Infra Project. (2019) OpenRAN: The Next Generation of Radio Access Networks. https://telecominfraproject.com/wp-content/uploads/OpenRAN-v11082019-vFinal.pdf.

[362] S. Kumar, E. Hamed, D. Katabi, and L. E. Li, "LTE Radio Analytics Made Easy and Accessible," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 211–222, August 2014.

[363] D. Gonzalez G., M. Garcia-Lozano, S. Ruiz Boqué, and D. S. Lee, "Optimization of Soft Frequency Reuse for Irregular LTE Macrocellular Networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 5, pp. 2410–2423, May 2013.

[364] I. Siomina and D. Yuan, "Analysis of Cell Load Coupling for LTE Network Planning and Optimization," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 2287–2297, June 2012.

[365] L. Korowajczuk, *LTE, WiMAX and WLAN Network Design, Optimization and Performance Analysis*. Chichester, United Kingdom: John Wiley & Sons, 2011.

[366] R. Margolies, A. Sridharan, V. Aggarwal, R. Jana, N. K. Shankaranarayanan, V. A. Vaishampayan, and G. Zussman, "Exploiting Mobility in Proportional Fair Cellular Scheduling: Measurements and Algorithms," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 355–367, February 2016.

[367] S. Mair. (2019) AT&T 2019 5G Recap: New Year, New Ways for AT&T Customers to Connect. https://about.att.com/innovationblog/2020/01/2019_5g_recap.html.

[368] O-RAN Alliance. (2020) The O-RAN Alliance and the Telecom Infra Project (TIP) Reach New Level of Collaboration for OpenRadio Access Networks. https://static1.squarespace.com/static/5ad774cce74940d7115044b0/t/5e54df89a726e63147f0b569/1582620555237/2020-02-25_O-RAN-TIP-PR-v1.0.pdf.

[369] ——. (2020) O-RAN Alliance Continues to Grow as Global Operators and Suppliers Reach Across Borders to Collaborate on Open Innovation in Radio Access Networks. https://static1.squarespace.com/static/5ad774cce74940d7115044b0/t/5e4ed59178b98159a8b1f881/1582224786007/2020-02-20_O-RAN+progress+PR_v1.0.pdf.

[370] D. Lynch, M. Fenton, D. Fagan, S. Kucera, H. Claussen, and M. O'Neill, "Automated Self-optimization in Heterogeneous Wireless Communications Networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 419–432, February 2019.

[371] G. Scutari, F. Facchinei, P. Song, D. P. Palomar, and J. S. Pang, "Decomposition by Partial Linearization: Parallel Optimization of Multi-agent Systems," *IEEE Transactions on Signal Processing*, vol. 62, no. 3, pp. 641–656, February 2014.

[372] G. Scutari, F. Facchinei, and L. Lampariello, "Parallel and Distributed Methods for Constrained Nonconvex Optimization—Part I: Theory," *IEEE Transactions on Signal Processing*, vol. 65, no. 8, pp. 1929–1944, April 2016.

[373] ETSI ZSM. (2019, August) Zero-touch Network and Service Management Reference Architecture. https://www.etsi.org/deliver/etsi_gs/ZSM/001_099/002/01.01.01_60/gs_ZSM002v010101p.pdf.

[374] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, March 2018.

[375] 3GPP, "Release description; Release 15," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 21.915, 10 2019, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/21915.htm

[376] S. D'Oro, P. Mertikopoulos, A. L. Moustakas, and S. Palazzo, "Interference-Based Pricing for Opportunistic Multicarrier Cognitive Radio Systems," *IEEE Transactions on Wireless Communications*, vol. 14, no. 12, pp. 6536–6549, December 2015.

[377] D. P. Bertsekas, *Convex Optimization Algorithms*. Nashua, NH, USA: Athena Scientific Belmont, 2015.

[378] J. Lee and S. Leyffer, *Mixed Integer Nonlinear Programming*. New York, NY, USA: Springer Science & Business Media, 2011.

[379] G. Scutari, D. Palomar, and S. Barbarossa, "The MIMO Iterative Waterfilling Algorithm," *IEEE Transactions on Signal Processing*, vol. 57, no. 5, pp. 1917–1935, May 2009.

[380] R. K. Jain, D. M. W. Chiu, and E. R. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System," *Eastern Research Laboratory, Digital Equipment Corporation: Hudson, MA, USA*, pp. 2–7, September 1984.

[381] 3GPP, "Telecommunication management;Study on system and functional aspects of energy efficiency in 5G networks," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 32.972, 9 2019, version 16.1.0. [Online]. Available: http://www.3gpp.org/DynaReport/32972.htm

[382] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, November 2017.

[383] H. Abdi and L. J. Williams, "Principal Component Analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[384] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "OpenRadio: A Programmable Wireless Dataplane," in *Proceedings of ACM HotSDN*, Helsinki, Finland, August 2012.

[385] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "SoftRAN: Software Defined Radio Access Network," in *Proceedings of ACM HotSDN*, Hong Kong, China, August 2013.

[386] L. E. Li, M. Z. Mao, and J. Rexford. (2012) CellSDN : Software-defined Cellular Networks. ftp://ftp.cs.princeton.edu/techreports/2012/922.pdf.

[387] A. Bradai, K. Singh, T. Ahmed, and T. Rasheed, "Cellular Software Defined Networking: A Framework," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 36–43, June 2015.

[388] The Linux Foundation. (2018) Open Network Automation Platform Architecture. https://www.onap.org/wp-content/uploads/sites/20/2018/11/ONAP_CaseSolution_Architecture_112918FNL.pdf.

[389] O-RAN Alliance. (2018, October) O-RAN: Towards an Open and Smart RAN. https://www.o-ran.org/s/O-RAN-WP-FInal-181017.pdf.

[390] C. J. Bernardos, A. de la Oliva, P. Serrano, A. Banchs, L. M. Contreras, H. Jin, and J. C. Zuniga, "An Architecture for Software Defined Wireless Networking," *IEEE Wireless Communications Magazine*, vol. 21, no. 3, pp. 52–61, June 2014.

[391] R. Li, Z. Zhao, Q. Sun, I. Chih-Lin, C. Yang, X. Chen, M. Zhao, and H. Zhang, "Deep Reinforcement Learning for Resource Management in Network Slicing," *IEEE Access*, vol. 6, pp. 1–4, November 2018.

[392] J. Du, C. Jiang, J. Wang, Y. Ren, and M. Debbah, "Machine Learning for 6G Wireless Networks: Carrying Forward Enhanced Bandwidth, Massive Access, and Ultrareliable/Low-Latency Service," *IEEE Vehicular Technology Magazine*, vol. 15, no. 4, pp. 122–134, December 2020.

[393] A. Okic, L. Zanzi, V. Sciancalepore, A. Redondi, and X. Costa-Pérez, "π-ROAD: A Learn-as-You-Go Framework for On-Demand Emergency Slices in V2X Scenarios," in *Proceedings of IEEE INFOCOM*, Vancouver, BC, Canada, May 2021.

[394] D. Bega, M. Gramaglia, A. Garcia-Saavedra, M. Fiore, A. Banchs, and X. Costa-Perez, "Network Slicing Meets Artificial Intelligence: An AI-based Framework for Slice Management," *IEEE Communications Magazine*, vol. 58, no. 6, pp. 32–38, June 2020.

[395] N. Salhab, R. Langar, R. Rahim, S. Cherrier, and A. Outtagarts, "Autonomous Network Slicing Prototype Using Machine-Learning-Based Forecasting for Radio Resources," *IEEE Communications Magazine*, vol. 59, no. 6, pp. 73–79, June 2021.

[396] J. Mei, X. Wang, K. Zheng, G. Boudreau, A. B. Sediq, and H. Abou-zeid, "Intelligent Radio Access Network Slicing for Service Provisioning in 6G: A Hierarchical Deep Reinforcement Learning Approach," *IEEE Transactions on Communications*, vol. 69, no. 9, pp. 6063–6078, September 2021.

[397] U. Paul, J. Liu, S. Troia, O. Falowo, and G. Maier, "Traffic-profile and Machine Learning Based Regional Data Center Design and Operation for 5G Network," *Journal of Communications and Networks*, vol. 21, no. 6, pp. 569–583, December 2019.

[398] A. Klautau, P. Batista, N. González-Prelcic, Y. Wang, and R. W. Heath, "5G MIMO Data for Machine Learning: Application to Beam-selection Using Deep Learning," in *Proceedings of IEEE Information Theory and Applications Workshop (ITA)*, San Diego, CA, USA, February 2018.

[399] C. Luo, J. Ji, Q. Wang, X. Chen, and P. Li, "Channel State Information Prediction for 5G Wireless Communications: A Deep Learning Approach," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 227–236, March 2018.

[400] J. M. DeAlmeida, L. DaSilva, C. B. Bonato Both, C. G. Ralha, and M. A. Marotta, "Artificial Intelligence-Driven Fog Radio Access Networks: Integrating Decision Making Considering Different Time Granularities," *IEEE Vehicular Technology Magazine*, vol. 16, no. 3, pp. 137–148, September 2021.

[401] T. S. Salem, G. Castellano, G. Neglia, F. Pianese, and A. Araldo, "Towards Inference Delivery Networks: Distributing Machine Learning with Optimality Guarantees," in *Proceedings of IEEE MedHocNet*, June, Ibiza, Spain 2021.

[402] O-RAN Working Group 3, "O-RAN Near-Real-time RAN Intelligent Controller E2 Service Model 1.0," Technical Specification, February 2020.

[403] R. Raman and I. E. Grossmann, "Modelling and Computational Techniques for Logic Based Integer Programming," *Computers & Chemical Engineering*, vol. 18, no. 7, pp. 563–578, July 1994.

[404] L. A. Wolsey, *Integer Programming*. John Wiley & Sons, September 2020.

[405] X. Li, Q. Zhai, J. Zhou, and X. Guan, "A Variable Reduction Method for Large-Scale Unit Commitment," *IEEE Transactions on Power Systems*, vol. 35, no. 1, pp. 261–272, January 2020.

[406] Y. Li, B. Liang, and A. Tizghadam, "Robust Online Learning against Malicious Manipulation and Feedback Delay with Application to Network Flow Classification," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2648–2663, August 2021.

[407] T. N. Weerasinghe, I. A. Balapuwaduge, and F. Y. Li, "Supervised Learning Based Arrival Prediction and Dynamic Preamble Allocation for Bursty Traffic," in *Proceedings of IEEE INFOCOM Workshops*, Paris, France, April 2019.

[408] H. Chergui and C. Verikoukis, "OPEX-Limited 5G RAN Slicing: An Over-Dataset Constrained Deep Learning Approach," in *Proceedings of IEEE ICC*, Dublin, Ireland, June 2020.

[409] J. A. Ayala-Romero, A. Garcia-Saavedra, X. Costa-Perez, and G. Iosifidis, "Bayesian Online Learning for Energy-Aware Resource Orchestration in Virtualized RANs," in *Proceedings of IEEE INFOCOM*, Vancouver, BC, Canada, May 2021.

[410] R. Singh, C. Hasan, X. Foukas, M. Fiore, M. K. Marina, and Y. Wang, "Energy-Efficient Orchestration of Metro-Scale 5G Radio Access Networks," in *Proceedings of IEEE INFOCOM*, Vancouver, BC, Canada, May 2021.

[411] S. Chatterjee, M. J. Abdel-Rahman, and A. B. MacKenzie, "On Optimal Orchestration of Virtualized Cellular Networks with Statistical Multiplexing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 1, pp. 310–325, January 2022.

[412] F. Z. Morais, G. M. de Almeida, L. Pinto, K. V. Cardoso, L. M. Contreras, R. d. R. Righi, and C. B. Both, "PlaceRAN: Optimal Placement of Virtualized Network Functions in the Next-generation Radio Access Networks," *arXiv:2102.13192 [cs.NI]*, February 2021.

[413] S. Matoussi, I. Fajjari, S. Costanzo, N. Aitsaadi, and R. Langar, "5G RAN: Functional Split Orchestration Optimization," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, pp. 1448–1463, July 2020.

[414] J. Baranda, J. Mangues-Bafalluy, E. Zeydan, L. Vettori, R. Martínez, X. Li, A. Garcia-Saavedra, C. Chiasserini, C. Casetti, K. Tomakh, O. Kolodiazhnyi, and C. J. Bernardos, "On the Integration of AI/ML-based Scaling Operations in the 5Growth Platform," in *Proceedings of IEEE NFV-SDN*, Leganes, Spain, November 2020.

[415] J. Baranda, J. Mangues-Bafalluy, E. Zeydan, C. Casetti, C. F. Chiasserini, M. Malinverno, C. Puligheddu, M. Groshev *et al.*, "Demo: AIML-as-a-Service for SLA management of a Digital Twin Virtual Network Service," in *Proceedings of IEEE INFOCOM Workshops*, Vancouver, BC, Canada, May 2021.

[416] X. Li, A. Garcia-Saavedra, X. Costa-Perez, C. J. Bernardos, C. Guimarães, K. Antevski, J. Mangues-Bafalluy, J. Baranda, E. Zeydan, D. Corujo *et al.*, "5Growth: An End-to-End Service Platform for Automated Deployment and Management of Vertical Services over 5G Networks," *IEEE Communications Magazine*, vol. 59, no. 3, pp. 84–90, March 2021.

[417] M. Moradi, W. Wu, L. E. Li, and Z. M. Mao, "SoftMoW: Recursive and Reconfigurable Cellular WAN Architecture," in *Proceedings of ACM CoNEXT*, Sydney, Australia, December 2014.

[418] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Pérez, "Resource Sharing Efficiency in Network Slicing," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 909–923, September 2019.

[419] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Pérez, "Optimising 5G Infrastructure Markets: The Business of Network Slicing," in *Proceedings of IEEE INFOCOM*, Atlanta, GA, USA, May 2017.

[420] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network Slicing in 5G: Survey and Challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, May 2017.

[421] F. Y. Shih, *Digital Watermarking and Steganography: Fundamentals and Techniques*. CRC Press, April 2017.

[422] A. Dutta, D. Saha, D. Grunwald, and D. Sicker, "Secret Agent Radio: Covert Communication through Dirty Constellations," in *Proceeding of Springer IH*, Berkeley, CA, USA, May 2012.

[423] S. Grabski and K. Szczypiorski, "Steganography in OFDM Symbols of Fast IEEE 802.11n Networks," in *Proceedings of the IEEE CS Security and Privacy Workshops*, San Francisco, CA, USA, May 2013.

[424] K. Szczypiorski and W. Mazurczyk, "Hiding Data in OFDM Symbols of IEEE 802.11 Networks," in *Proceedings of IEEE MINES*, Nanjing, Jiangsu, China, November 2010.

[425] T. Kho, "Steganography in the 802.15.4 Physical Layer," *U.C. Berkeley*, December 17 2007.

[426] E. Zielinska and K. Szczypiorski, "Direct Sequence Spread Spectrum Steganographic Scheme for IEEE 802.15.4," in *Proceedings of IEEE MINES*, Shanghai, China, November 2011.

[427] B. A. Bash, D. Goeckel, D. Towsley, and S. Guha, "Hiding Information in Noise: Fundamental Limits of Covert Wireless Communication," *IEEE Communications Magazine*, vol. 53, no. 12, pp. 26–31, December 2015.

[428] D. Kahn, "The History of Steganography," in *Proceedings of Springer IH*, Cambridge, United Kingdom, May 1996.

[429] J. Classen, M. Schulz, and M. Hollick, "Practical Covert Channels for WiFi Systems," in *Proceedings of IEEE CNS*, Florence, Italy, September 2015.

[430] S. D'Oro, F. Restuccia, and T. Melodia, "Hiding Data in Plain Sight: Undetectable Wireless Communications Through Pseudo-Noise Asymmetric Shift Keying," in *Proceedings of IEEE INFOCOM*, Paris, France, April 2019.

[431] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-down Approach*, 7th ed. Pearson, 2017.

[432] Z. Xia, X. Wang, X. Sun, and B. Wang, "Steganalysis of Least Significant Bit Matching Using Multi-order Differences," *Wiley Security and Communication Networks*, vol. 7, no. 8, pp. 1283–1291, August 2014.

[433] iPerf3. (2021) https://iperf.fr.

[434] Z. Liu, Y. Jiang, and P. Qian, "A Data-Hiding Method Based on TCP/IP Checksum," in *Advances in Computer Science and its Applications*. Springer, 2014, pp. 1039–1044.

[435] S. J. Murdoch and S. Lewis, "Embedding covert channels into TCP/IP," in *Proceedings of Springer IH 2005*, Barcelona, Spain, June 2005.

[436] K. Ahsan and D. Kundur, "Practical Data Hiding in TCP/IP," in *Proceeding of the ACM IH&MMSec*, Juan-les-Pins, France, December 2002.

[437] I. Grabska and K. Szczypiorski, "Steganography in Long Term Evolution Systems," in *Proceedings of the IEEE CS Security and Privacy Workshops*, San Jose, CA, USA, May 2014.

[438] D. Martins and H. Guyennet, "Steganography in MAC Layers of 802.15.4 Protocol for Securing Wireless Sensor Networks," in *Proceedings of IEEE MINES*, Nanjing, Jiangsu, China, November 2010.

[439] A. M. Mehta, S. Lanzisera, and K. S. Pister, "Steganography in 802.15.4 Wireless Communication," in *Proceedings of IEEE ANTS*, Mumbai, India, December 2008.

[440] K. Sankhe, F. Restuccia, S. D'Oro, T. Jian, Z. Wang, A. Al-Shawabka, J. Dy, T. Melodia, S. Ioannidis, and K. Chowdhury, "Impairment Shift Keying: Covert Signaling by Deep Learning of Controlled Radio Imperfections," in *Proceedings of IEEE MILCOM*, Norfolk, VA, USA, November 2019.

[441] P. Cao, W. Liu, G. Liu, X. Ji, J. Zhai, and Y. Dai, "A Wireless Covert Channel Based on Constellation Shaping Modulation," *Hindawi Security and Communication Networks*, vol. 2018, pp. 1–15, January 2018.

[442] V. Kumar, J.-M. Park, T. C. Clancy, and K. Bian, "PHY-layer Authentication Using Hierarchical Modulation and Duobinary Signaling," in *Proceedings of IEEE ICNC*, Honolulu, HI, USA, February 2014.

[443] Z. Hijaz and V. S. Frost, "Exploiting OFDM Systems for Covert Communication," in *Proceedings of IEEE MILCOM*, San Jose, CA, USA, October 2010.

[444] M. Hamdaqa and L. Tahvildari, "ReLACK: A Reliable VoIP Steganography Approach," in *Proceedings of IEEE SSIRI*, Jeju Island, South Korea, June 2011.

[445] A. K. Nain and P. Rajalakshmi, "A Reliable Covert Channel Over IEEE 802.15.4 Using Steganography," in *Proceedings of IEEE WF-IoT*, Reston, VA, USA, December 2016.

[446] S. Grabski and K. Szczypiorski, "Network Steganalysis: Detection of Steganography in IEEE 802.11 Wireless Networks," in *Proceedings of IEEE ICUMT*, Almaty, Kazakhstan, September 2013.

[447] N. Bizanis and F. A. Kuipers, "SDN and Virtualization Solutions for the Internet of Things: A Survey," *IEEE Access*, vol. 4, pp. 5591–5606, 2016.

[448] ETSI, "5G End to End Key Performance Indicators (KPI)," https://tinyurl.com/ETSIStandard, 2018.

[449] ——, "MEC in 5G Networks," https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf, 2018.

[450] N. Van Huynh, D. Thai Hoang, D. N. Nguyen, and E. Dutkiewicz, "Optimal and Fast Real-Time Resource Slicing With Deep Dueling Neural Networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1455–1470, June 2019.

[451] G. Castellano, F. Esposito, and F. Risso, "A Distributed Orchestration Algorithm for Edge Computing Resources with Guarantees," in *Proceedings of IEEE INFOCOM*, Paris, France, April 2019.

[452] A. Ndikumana, N. H. Tran, T. M. Ho, Z. Han, W. Saad, D. Niyato, and C. S. Hong, "Joint Communication, Computation, Caching, and Control in Big Data Multi-Access Edge Computing," *IEEE Transactions on Mobile Computing*, vol. 19, no. 6, pp. 1359–1374, June 2020.

[453] M. Bouet and V. Conan, "Mobile Edge Computing Resources Optimization: A Geo-clustering Approach," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 787–796, June 2018.

[454] M.-H. Lin, J. G. Carlsson, D. Ge, J. Shi, and J.-F. Tsai, "A Review of Piecewise Linearization Methods," *Hindawi Mathematical Problems in Engineering*, September 2013.

[455] M. Elf, C. Gutwenger, M. Jünger, and G. Rinaldi, *Branch-and-Cut Algorithms for Combinatorial Optimization and Their Implementation in ABACUS*. Springer Berlin Heidelberg, 2001, pp. 157–222.

[456] A. Huang, "Similarity Measures for Text Document Clustering," in *Proceedings of the New Zealand Computer Science Research Student Conference*, Christchurch, New Zealand, April 2008.

[457] R. Xu and D. C. Wunsch, "Survey of Clustering Algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, May 2005.

[458] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Now Publishers Inc., 2011.

[459] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the Linear Convergence of the ADMM in Decentralized Consensus Optimization," *IEEE Transactions on Signal Processing*, vol. 62, no. 7, pp. 1750–1761, April 2014.

[460] H. Holma and A. Toskala, *LTE for UMTS: OFDMA and SC-FDMA Based Radio Access*. Wiley, 2009.

[461] DASH Industry Forum, "dash.js," https://tinyurl.com/ojd2bba, 2019.

[462] T. Stockhammer, "Dynamic Adaptive Streaming Over HTTP: Standards and Design Principles," in *Proceedings of ACM MMSys*, San Jose, CA, USA, February 2011.

[463] FFmpeg Developers, "ffmpeg tool," http://ffmpeg.org/, 2019.

[464] ETSI, "MEC Support for Network Slicing," https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=53580, 2019.

[465] ——, "MEC Deployments in 4G and Evolution Towards 5G," https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp24_MEC_deployment_in_4G_5G_FINAL.pdf, 2019.

[466] V. Mancuso, P. Castagno, M. Sereno, and M. A. Marsan, "Slicing Cell Resources: The Case of HTC and MTC Coexistence," in *Proceedings of IEEE INFOCOM*, Paris, France, April 2019.

[467] S. Mandelli, M. Andrews, S. Borst, and S. Klein, "Satisfying Network Slicing Constraints via 5G MAC Scheduling," in *Proceedings of IEEE INFOCOM*, Paris, France, April 2019.

[468] G. Garcia-Aviles, M. Gramaglia, P. Serrano, and A. Banchs, "POSENS: A Practical Open Source Solution for End-to-end Network Slicing," *IEEE Wireless Communications*, vol. 25, no. 5, pp. 30–37, October 2018.

[469] Q. Zhang, F. Liu, and C. Zeng, "Adaptive Interference-aware VNF Placement for Service-customized 5G Network Slices," in *Proceedings of IEEE INFOCOM*, Paris, France, April 2019.

[470] B. Han, V. Sciancalepore, D. Feng, X. Costa-Perez, and H. D. Schotten, "A Utility-Driven Multi-Queue Admission Control Solution for Network Slicing," in *Proceedings of IEEE INFOCOM*, Paris, France, April 2019.

[471] H. Halabian, "Distributed Resource Allocation Optimization in 5G Virtualized Networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 627–642, March 2019.

[472] Y. Sun, M. Peng, S. Mao, and S. Yan, "Hierarchical Radio Resource Allocation for Network Slicing in Fog Radio Access Networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3866–3881, April 2019.

[473] S. Agarwal, F. Malandrino, C.-F. Chiasserini, and S. De, "Joint VNF Placement and CPU Allocation in 5G," in *Proceedings of IEEE INFOCOM*, Honolulu, HI, USA, April 2018.

[474] P. Caballero, A. Banchs, G. de Veciana, and X. Costa-Pérez, "Network Slicing Games: Enabling Customization in Multi-tenant Networks," in *Proceedings of IEEE INFOCOM*, Atlanta, GA, USA, May 2017.

[475] M. Jiang, M. Condoluci, and T. Mahmoodi, "Network Slicing in 5G: An Auction-based Model," in *Proceedings of IEEE ICC*, Paris, France, May 2017.

[476] S. Misra and N. Saha, "Detour: Dynamic Task Offloading in Software-Defined Fog for IoT Applications," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1159–1166, May 2019.

[477] T. X. Tran and D. Pompili, "Joint Task Offloading and Resource Allocation for Multi-server Mobile-edge Computing Networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, January 2018.

[478] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal Computation Task Scheduling for Mobile-edge Computing Systems," in *Proceedings of IEEE ISIT*, Barcelona, Spain, July 2016.

[479] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation Offloading in Multi-Access Edge Computing Using a Deep Sequential Model Based on Reinforcement Learning," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 64–69, May 2019.

[480] A. Al-Shuwaili and O. Simeone, "Energy-efficient Resource Allocation for Mobile Edge Computing-based Augmented Reality Applications," *IEEE Wireless Communications Letters*, vol. 6, no. 3, pp. 398–401, June 2017.

[481] M. Erol-Kantarci and S. Sukhmani, "Caching and Computing at the Edge for Mobile Augmented Reality and Virtual Reality (AR/VR) in 5G," in *Ad Hoc Networks*. Springer, 2018, pp. 169–177.

[482] L. Tang and S. He, "Multi-user Computation Offloading in Mobile edge Computing: A Behavioral Perspective," *IEEE Network*, vol. 32, no. 1, pp. 48–53, January 2018.

[483] S. He, J. Ren, J. Wang, Y. Huang, Y. Zhang, W. Zhuang, and S. Shen, "Cloud-Edge Coordinated Processing: Low-Latency Multicasting Transmission," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1144–1158, May 2019.

[484] S.-R. Yang, Y.-J. Tseng, C.-C. Huang, and W.-C. Lin, "Multi-Access Edge Computing Enhanced Video Streaming: Proof-of-Concept Implementation and Prediction/QoE Models," *IEEE Trans. on Vehicular Technology*, vol. 68, no. 2, pp. 1888–1902, February 2018.

[485] D. T. Hoang, D. Niyato, D. N. Nguyen, E. Dutkiewicz, P. Wang, and Z. Han, "A Dynamic Edge Caching Framework for Mobile 5G Networks," *IEEE Wireless Communications*, vol. 25, no. 5, pp. 95–103, October 2018.

[486] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Cooperative Content Caching in 5G Networks with Mobile Edge Computing," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 80–87, June 2018.

[487] A. Kaloxylos, "A Survey and an Analysis of Network Slicing in 5G Networks," *IEEE Communications Standards Magazine*, vol. 2, no. 1, pp. 60–65, March 2018.

[488] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, March 2017.

[489] Ericsson, "Ericsson Mobility Report," https://www.ericsson.com/49da93/assets/local/mobility-report/documents/2020/june2020-ericsson-mobility-report.pdf, June 2020.

[490] National Institute of Standards and Technology (NIST), "Spectrum Crunch," https://www.nist.gov/topics/advanced-communications/spectrum-crunch, 2019.

[491] A. Nakao, P. Du, Y. Kiriha, F. Granelli, A. A. Gebremariam, T. Taleb, and M. Bagaa, "End-to-end Network Slicing for 5G Mobile Networks," *Journal of Information Processing*, vol. 25, pp. 153–163, 2017.

[492] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan, "CellSlice: Cellular Wireless Resource Slicing for Active RAN Sharing," in *Proceedings of IEEE COMSNETS*, Bangalore, India, January 2013.

[493] S. D'Oro, L. Galluccio, P. Mertikopoulos, G. Morabito, and S. Palazzo, "Auction-based Resource Allocation in OpenFlow Multi-tenant Networks," *Computer Networks*, vol. 115, pp. 29–41, 2017.

[494] E. Hossain, M. Rasti, H. Tabassum, and A. Abdelnasser, "Evolution Toward 5G Multi-tier Cellular Wireless Networks: An Interference Management Perspective," *IEEE Wireless Communications*, vol. 21, no. 3, pp. 118–127, 2014.

[495] S. D'Oro, A. Zappone, S. Palazzo, and M. Lops, "A Learning Approach for Low-Complexity Optimization of Energy Efficiency in Multicarrier Wireless Networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 5, pp. 3226–3241, 2018.

[496] V. Jungnickel, K. Manolakis, W. Zirwas, B. Panzner, V. Braun, M. Lossow, M. Sternad, R. Apelfrojd, and T. Svensson, "The Role of Small Cells, Coordinated Multipoint, and Massive MIMO in 5G," *IEEE Communications Magazine*, vol. 52, no. 5, pp. 44–51, 2014.

[497] 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*, vol. 49, no. 2, pp. 102–111, February 2011.

[498] J. Lee, Y. Kim, H. Lee, B. L. Ng, D. Mazzarese, J. Liu, W. Xiao, and Y. Zhou, "Coordinated Multipoint Transmission and Reception in LTE-advanced Systems," *IEEE Communications Magazine*, vol. 50, no. 11, 2012.

[499] D. Boviz and Y. E. Mghazli, "Fronthaul for 5G: Low Bit-rate Design Enabling Joint Transmission and Reception," in *Proceedings of IEEE GLOBECOM Workshops*, December 2016.

[500] W. Nam, D. Bai, J. Lee, and I. Kang, "Advanced Interference Management for 5G Cellular Networks," *IEEE Communications Magazine*, vol. 52, no. 5, pp. 52–60, 2014.

[501] K. Samdanis, S. Wright, A. Banchs, A. Capone, M. Ulema, and K. Obana, "5G Network Slicing - Part 1: Concepts, Principles, and Architectures," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 70–71, May 2017.

[502] C.-Y. Chang, N. Nikaein, and T. Spyropoulos, "Radio Access Network Resource Slicing for Flexible Service Execution," in *Proceedings of IEEE INFOCOM Workshops*, 2018.

[503] O. Sallent, J. Perez-Romero, R. Ferrus, and R. Agusti, "On Radio Access Network Slicing from a Radio Resource Management Perspective," *IEEE Wireless Communications*, vol. 24, no. 5, pp. 166–174, 2017.

[504] E. Dahlman, S. Parkvall, and J. Skold, *4G: LTE/LTE-Advanced for Mobile Broadband.* Academic Press, 2013.

[505] R. Ferrus, O. Sallent, J. Perez-Romero, and R. Agusti, "On 5G Radio Access Network Slicing: Radio Interface Protocol Features and Configuration," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 184–192, 2018.

[506] P. Zhao, H. Tian, S. Fan, and A. Paulraj, "Information Prediction and Dynamic Programming-Based RAN Slicing for Mobile Edge Computing," *IEEE Wireless Communications Letters*, vol. 7, no. 4, pp. 614–617, August 2018.

[507] P. Caballero, A. Banchs, G. de Veciana, and X. Costa-Pérez, "Multi-Tenant Radio Access Network Slicing: Statistical Multiplexing of Spatial Loads," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 3044–3058, 2017.

[508] A. A. Gebremariam, M. Chowdhury, M. Usman, A. Goldsmith, and F. Granelli, "SoftSLICE: Policy-based Dynamic Spectrum Slicing in 5G Cellular Networks," in *Proceedings of IEEE ICC*, May 2018.

[509] Y. Jia, H. Tian, S. Fan, P. Zhao, and K. Zhao, "Bankruptcy Game Based Resource Allocation Algorithm for 5G Cloud-RAN Slicing," in *Proceedings of IEEE WCNC*, April 2018.

[510] O. Narmanlioglu and E. Zeydan, "Learning in SDN-based Multi-tenant Cellular Networks: A Game-theoretic Perspective," in *Proceedings of IFIP/IEEE International Symposium on Integrated Network and Service Management*, May 2017.

[511] H. Ryoo and N. Sahinidis, "Global Optimization of Nonconvex NLPs and MINLPs with Applications in Process Design," *Computers & Chemical Engineering*, vol. 19, no. 5, pp. 551 – 566, 1995.

[512] C. A. Floudas and V. Visweswaran, "Quadratic Optimization," in *Handbook of Global Optimization*. Springer, 1995, pp. 217–269.

[513] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[514] X. Huang, G. Xue, R. Yu, and S. Leng, "Joint Scheduling and Beamforming Coordination in Cloud Radio Access Networks With QoS Guarantees," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 7, pp. 5449–5460, 2016.

[515] P. Vo, M. Nguyen, T. Le, and N. Tran, "Slicing the Edge: Resource Allocation for RAN Network Slicing," *IEEE Wireless Communications Letters*, vol. 7, no. 6, December 2018.

[516] V. Sciancalepore, X. Costa-Perez, and A. Banchs, "RL-NSB: Reinforcement Learning-Based 5G Network Slice Broker," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1543–1557, Aug 2019.

[517] A. Devlic, A. Hamidian, D. Liang, M. Eriksson, A. Consoli, and J. Lundstedt, "NESMO: Network Slicing Management and Orchestration Framework," in *Proceedings of IEEE ICC Workshops*, May 2017.

[518] A. Ksentini and N. Nikaein, "Toward Enforcing Network Slicing on RAN: Flexibility and Resources Abstraction," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 102–108, June 2017.

[519] B. Han, J. Lianghai, and H. D. Schotten, "Slice as an Evolutionary Service: Genetic Optimization for Inter-Slice Resource Management in 5G Networks," *IEEE Access*, vol. 6, pp. 33 137–33 147, June 2018.

[520] R. Mahindra, M. A. Khojastepour, H. Zhang, and S. Rangarajan, "Radio Access Network Sharing in Cellular Networks," in *Proceedings of IEEE ICNP*, October 2013.

List of Publications

Journals

[521] L. Bonati, S. D'Oro, L. Bertizzolo, E. Demirors, Z. Guan, S. Basagni, and T. Melodia, "CellOS: Zero-touch Softwarized Open Cellular Networks," *Computer Networks*, vol. 180, pp. 1–13, October 2020.

[522] L. Bertizzolo, L. Bonati, E. Demirors, A. Al-shawabka, S. D'Oro, F. Restuccia, and T. Melodia, "Arena: A 64-antenna SDR-based Ceiling Grid Testing Platform for Sub-6 GHz 5G-and-Beyond Radio Spectrum Research," *Computer Networks*, vol. 181, pp. 1–17, November 2020.

[523] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead," *Computer Networks*, vol. 182, pp. 1–28, December 2020.

[524] S. D'Oro, L. Bonati, F. Restuccia, and T. Melodia, "Coordinated 5G Network Slicing: How Constructive Interference Can Boost Network Throughput," *IEEE/ACM Transactions on Networking*, vol. 29, no. 4, pp. 1881–1894, August 2021.

[525] L. Ferranti, S. D'Oro, L. Bonati, F. Cuomo, and T. Melodia, "HIRO-NET: Heterogeneous Intelligent RObotic Network for Internet sharing in Disaster Scenarios," *IEEE Transactions on Mobile Computing*, 2021.

[526] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia, "Intelligence and Learning in O-RAN for Data-driven NextG Cellular Networks," *IEEE Communications Magazine*, vol. 59, no. 10, pp. 21–27, October 2021.

[527] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges," *arXiv:2202.01032*, February 2022.

[528] ——, "ColO-RAN: Developing Machine Learning-based xApps for Open RAN Closed-loop Control on Programmable Experimental Platforms," *IEEE Transactions on Mobile Computing*, pp. 1–14, July 2022.

[529] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "OpenRAN Gym: AI/ML Development, Data Collection, and Testing for O-RAN on PAWR Platforms," *arXiv:2207.12362*, pp. 1–12, July 2022.

[530] S. D'Oro, M. Polese, L. Bonati, H. Cheng, and T. Melodia, "dApps: Distributed Applications for Real-time Inference and Control in O-RAN," *IEEE Communications Magazine*, November 2022.

Conference Proceedings

[531] L. Ferranti, S. D'Oro, L. Bonati, E. Demirors, F. Cuomo, and T. Melodia, "HIRO-NET: Self-Organized Robotic Mesh Networking for Internet Sharing in Disaster Scenarios," in *Proceedings of IEEE WoWMoM*, Washington, D.C., USA, June 2019.

[532] L. Bertizzolo, L. Bonati, E. Demirors, and T. Melodia, "Arena: A 64-Antenna SDR-Based Ceiling Grid Testbed for Sub-6 GHz Radio Spectrum Research," in *Proceedings of ACM WiNTECH*, Los Cabos, Mexico, October 2019.

[533] ——, "Demo: Arena: A 64-Antenna SDR-Based Ceiling Grid Testbed for Sub-6 GHz Radio Spectrum Research," in *Proceedings of ACM WiNTECH*, Los Cabos, Mexico, October 2019.

[534] L. Bertizzolo, M. Polese, L. Bonati, A. Gosain, M. Zorzi, and T. Melodia, "mmBAC: Location-Aided mmWave Backhaul Management for UAV-Based Aerial Cells," in *Proceedings of ACM mmNets*, Los Cabos, Mexico, October 2019.

[535] F. D'Alterio, L. Ferranti, L. Bonati, F. Cuomo, and T. Melodia, "Quality Aware Aerial-to-Ground 5G Cells through Open-Source Software," in *Proceedings of IEEE GLOBECOM*, Waikoloa, HI, USA, December 2019.

[536] L. Bertizzolo, S. D'Oro, L. Ferranti, L. Bonati, E. Demirors, Z. Guan, T. Melodia, and S. Pudlewski, "SwarmControl: An Automated Distributed Control Framework for Self-Optimizing Drone Networks," in *Proceedings of IEEE INFOCOM*, Virtual Conference, July 2020.

[537] S. D'Oro, L. Bonati, F. Restuccia, M. Polese, M. Zorzi, and T. Melodia, "Sl-EDGE: Network Slicing at the Edge," in *Proceedings of ACM Mobihoc*, Virtual Conference, October 2020.

[538] L. Ferranti, L. Bonati, S. D'Oro, and T. Melodia, "SkyCell: A Prototyping Platform for 5G Aerial Base Stations," in *Proceedings of IEEE SwarmNet*, Virtual Conference, August 2020.

[539] M. Polese, L. Bertizzolo, L. Bonati, A. Gosain, and T. Melodia, "An Experimental mmWave Channel Model for UAV-to-UAV Communications," in *Proceedings of ACM mmNets*, London, United Kingdom, September 2020.

[540] L. Bonati, S. D'Oro, F. Restuccia, S. Basagni, and T. Melodia, "SteaLTE: Private 5G Cellular Connectivity as a Service with Full-stack Wireless Steganography," in *Proceedings of IEEE INFOCOM*, Virtual Conference, May 2021.

[541] M. Tehrani-Moayyed, L. Bonati, P. Johari, T. Melodia, and S. Basagni, "Creating RF Scenarios for Large-Scale, Real-Time Wireless Channel Emulators," in *Proceedings of IEEE MedComNet*, Virtual Conference, June 2021.

[542] L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "SCOPE: An Open and Softwarized Prototyping Platform for NextG Systems," in *Proceedings of ACM MobiSys*, Virtual Conference, June 2021.

[543] B. Casasole, L. Bonati, S. D'Oro, S. Basagni, A. Capone, and T. Melodia, "QCell: Self-optimization of Softwarized 5G Networks through Deep Q-learning," in *Proceedings of IEEE GLOBECOM*, Madrid, Spain, December 2021.

[544] T. Melodia, S. Basagni, K. R. Chowdhury, A. Gosain, M. Polese, P. Johari, and L. Bonati, "Tutorial: Colosseum, the World's Largest Wireless Network Emulator," in *Proceedings of ACM MobiCom*, New Orleans, LA, USA, October 2021.

[545] L. Bonati, P. Johari, M. Polese, S. D'Oro, S. Mohanti, M. Tehrani-Moayyed, D. Villa, S. Shrivastava, C. Tassie, K. Yoder, A. Bagga, P. Patel, V. Petkov, M. Seltser, F. Restuccia, A. Gosain, K. R. Chowdhury, S. Basagni, and T. Melodia, "Colosseum: Large-Scale Wireless Experimentation Through Hardware-in-the-Loop Network Emulation," in *Proceedings of IEEE DySPAN*, Virtual Conference, December 2021.

[546] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "OrchestRAN: Network Automation through Orchestrated Intelligence in the Open RAN," in *Proceedings of IEEE INFOCOM (accepted)*, Virtual Conference, May 2022.

[547] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "OpenRAN Gym: An Open Toolbox for Data Collection and Experimentation with AI in O-RAN," in *Proceedings of IEEE WCNC Workshop on Open RAN Architecture for 5G Evolution and 6G*, Austin, TX, USA, April 2022.

Editorials

[548] L. Bonati, S. Basagni, and T. Melodia, "Editorial: Advances in Experimental Wireless Platforms and Systems," *Computer Networks*, vol. 203, pp. 1–2, February 2022.

Patent Applications

[549] T. Melodia, L. Bonati, S. D'Oro, and F. Restuccia, "Private 5G Cellular Connectivity as a Service Through Full-Stack Wireless Steganography," Patent Application Publication U.S. 2021/0352053 A1, November 2021.

[550] S. D'Oro, T. Melodia, F. Restuccia, and L. Bonati, "Methods for Multi-Access Edge Computing Network Slicing in 5G Networks," Patent Application Publication U.S. 2022/0022044 A1, January 2022.

[551] T. Melodia, S. D'Oro, M. Polese, and L. Bonati, "Intelligence and Learning in O-RAN for 5G and 6G Cellular Networks," Patent Application Publication U.S. 2022/0167236 A1, May 2022.

Appendix A

Sl-EDGE: Network Slicing at the Edge

It is now clear that advanced softwarization and virtualization paradigms such as network slicing will be the *cornerstone* of 5G networks [13] and the Internet of Things [447]. Indeed, by sharing a common underlying physical infrastructure, NOs can dynamically deploy multiple "slices" tailored for specific services (e.g., video streaming, augmented reality), as well as requirements (e.g., low latency, high throughput, low jitter) [448], avoiding the static—thus, inefficient—network deployments that have plagued traditional hardware-based cellular networks. To further decrease latency, increase throughput, and provide improved services to their subscribers, NOs have recently started integrating *MEC* technologies [72], which are expected to become essential to reach the sub-1 ms latency requirements of 5G. MEC will be so quintessential that the European Telecommunications Standards Institute has identified MEC as "*one of the key pillars for meeting the demanding key performance indicators of 5G*" and "*[as playing] an essential role in the transformation of the telecommunications business*" [449].

Despite the clear advantages of network slicing and MEC, the truth of the matter is that we cannot have one without the other. Indeed, slicing networking resources only, e.g., spectrum and PRBs *cannot suffice to satisfy the stringent timing and performance requirements of 5G networks*. Real-time wireless video streaming, for example, requires at the same time (i) networking resources (e.g., PRBs) to broadcast the video, (ii) computational resources to process and transcode the video, as well as (iii) storage resources to locally cache the video. The key issue that sets MEC slicing apart from traditional slicing problems is that MEC resources are usually *coupled*, meaning that slicing one resource usually leads to a *performance degradation* in another type of resource.

We clarify this issue in Figure A.1, where we show an experiment (testbed described in Section A.6) where we instantiate 1 slice for video streaming (S1) and 2 slices for video transcoding (S2 and S3). S1 starts at $t = 0$ s, while S2 and S3 start at $t = 75$ s. Figure A.1 clearly shows that when S2 and S3 start, the performance of S1 plummets. This is because the computational resources allocated for S2 and S3 cause the video buffer (blue line) to drop from $\sim$30 seconds to $\sim$10 seconds, which in turn causes highly-jittered bitrate (red line). As soon as S2 and S3 end at $t = 190$ s, buffer size and video bitrate sharply increase and stabilize. This demonstrates that slices requiring both computation and networking resources (S1) are inevitably impacted by slices running on the same node that only require computation (S2 and S3). Therefore, *taking into account coupling among slices is a compelling necessity to guarantee appropriate performance when designing edge slicing algorithms*.

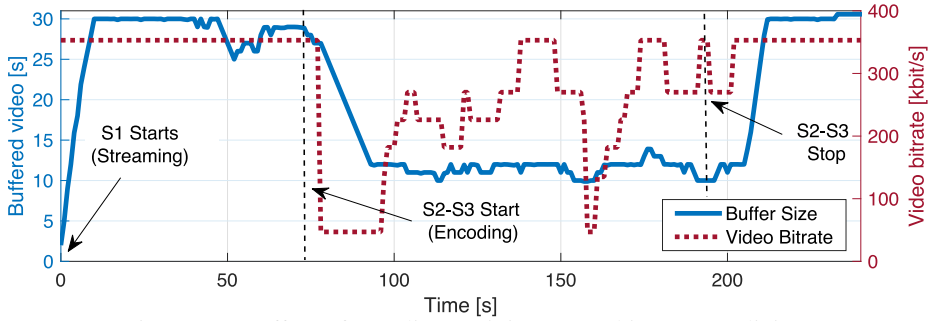


Figure A.1: Effect of coupling on joint networking-MEC slicing.

Extensive research efforts have already explored MEC algorithms for cellular networks and network slicing [7, 450–452]. The key limitation of prior work, however, is the assumption that network slicing and MEC are distinct problems. As we demonstrated in Figure A.1, this is hardly the case in practical scenarios. However, the already rich literature on network slicing—discussed in details in Section A.7—has not yet considered this fundamental aspect. This makes the joint MEC/slicing paradigm a largely unexplored problem. We also point out that due to the massive scale envisioned for 5G and IoT applications, centralized algorithms become prohibitive. Thus, a core issue is not only to account for resource coupling, but also to devise new slicing algorithms that *provide highly efficient and scalable slicing strategies*.

Contributions. In this chapter, which was published as [537], we design, analyze and experimentally evaluate the unified MEC slicing framework, called Sl-EDGE, that allows network operators to instantiate heterogeneous slice services (e.g., video streaming, caching, 5G network access) on edge devices. In a nutshell, we make the following contributions:

- We mathematically model and discuss coupling relationships among networking, storage and computation resources at each edge node (Section A.2). We formulate the *Edge Slicing Problem* (ESP) as a Mixed Integer Linear Programming (MILP) problem (Section A.3);

- We propose three novel slicing algorithms to address the ESP, each having different optimality and computational complexity. Specifically, we present (i) a centralized optimal algorithm for small network instances (Section A.3); (ii) an approximation algorithm that leverages virtualization concepts to reduce complexity with close-to-optimal performance (Section A.4.1), and (iii) a low-complexity algorithm where slicing decisions are made at the edge nodes with minimal overhead (Section A.4.2);

- We extensively evaluate the performance of the three slicing algorithms through simulation, and compare Sl-EDGE with DIRECT [7], to the best of our knowledge the state-of-the-art slicing framework for MEC 5G applications (Section A.5). Our results show that, by taking into account coupling among heterogeneous resources, Sl-EDGE (i) instantiates slices 6x more efficiently then the algorithm proposed in [7], while satisfying resource availability constraints, and (ii) can be implemented with a distributed approach while getting 25% close to the optimal solution;

- We prototype and demonstrate Sl-EDGE on a testbed composed by 24 software-defined radios. Experimental results demonstrate that Sl-EDGE instantiates heterogeneous slices providing LTE connectivity to smartphones, video streaming over Wi-Fi, and *ffmpeg* video transcoding. It achieves an instantaneous throughput of 37 $\mathrm{Mbit/s}$ over LTE links, and a 1.2 $\mathrm{Mbit/s}$ video streaming bitrate, with an overall CPU utilization of 83% (Section A.6).

The remainder of this chapter is organized as follows. An overview of Sl-EDGE and its architecture is presented in Section A.1. Sl-EDGE system model is discussed in details in Section A.2. Section A.3 presents the centralized optimal algorithm that Sl-EDGE uses to efficiently allocate networking and MEC slices in small-scale networks, while approximation algorithms for large-scale networks are described in Section A.4. Section A.5 illustrates the performance of Sl-EDGE through numerical simulations, and Section A.6 presents a prototype of Sl-EDGE. Section A.7 surveys the literature on the topic and highlights major differences between Sl-EDGE and the state-of-the-art. Finally, concluding remarks are given in Section A.8.

A.1 Sl-EDGE at a Glance

Sl-EDGE is a slicing framework for MEC-enabled 5G systems. Its key advantage is that it provides a fast, flexible and efficient deployment of joint networking and MEC slices. The three-tiered architecture of Sl-EDGE is illustrated in Figure A.2.

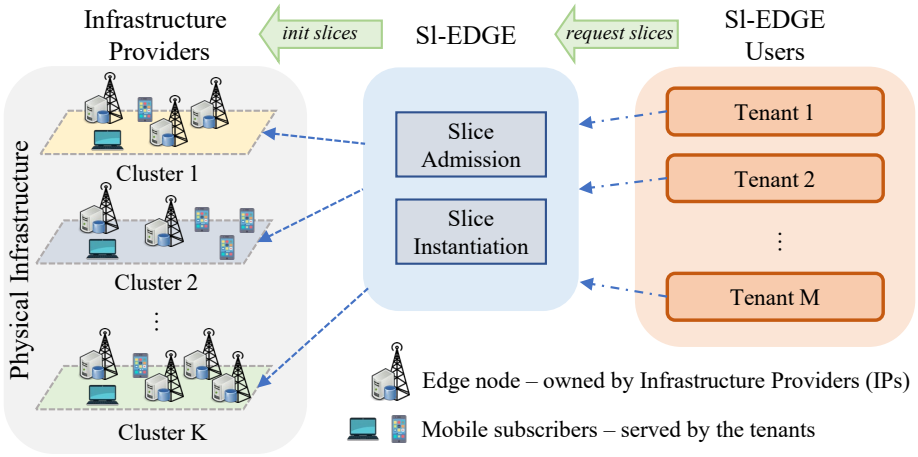


Figure A.2: The three-tier architecture of Sl-EDGE.

The *physical infrastructure* consists of a set of MEC-enabled networking edge nodes (e.g., base stations, access points, IoT gateways)—referred to as *MEC hosts* [449]—controlled by one or more IPs. MEC hosts are located at the network edge and simultaneously provide networking, storage and computational services (*e.g.*, Internet access, video content delivery, caching).

Sl-EDGE Users are both mobile and virtual NOs and Service Providers—referred to as the *tenants*—willing to rent portions of the infrastructure to provide services to their subscribers. Tenants access Sl-EDGE to visualize relevant information such as position of MEC hosts, covered areas and a list of networking and MEC services that can be instantiated on each host (e.g., 5G/Wi-Fi

connectivity, caching, computation). Whenever tenants need to provide these services, they submit slice requests to obtain networking, storage or computation resources. Received slice requests are collected and processed by Sl-EDGE which (i) determines the requests to be accommodated by using centralized (Section A.3) and distributed algorithms (Section A.4); (ii) instantiates slices by allocating the available resources to each admitted slice, and (iii) notifies to the admitted tenants the list of the resources allocated to the slice.

A.2 System Model

Let $\mathcal{D}$ be the set of deployed MEC-enabled networking devices, or *edge nodes*. Edge nodes provide both wireless connectivity and MEC services to a limited portion of the network. Therefore, they can be clustered into K *clusters* located in different geographical areas [385, 452, 453]. Let $\mathcal{K} = \{1, 2, \dots, K\}$ be the set of these K independent clusters, and let $\mathcal{D}_k$ be the set of edge nodes in cluster $k \in \mathcal{K}$. Each edge node $d \in \mathcal{D}_k$ is equipped with a set of networking, storage and computational capabilities, usually measured in terms of number of PRBs [12], megabytes, and billion of instructions per second (GIPS) [452], respectively. Let $z \in \mathcal{T} = \{\mathcal{N}, \mathcal{S}, \mathcal{C}\}$ represent the resource type, *i.e.*, networking ($\mathcal{N}$), storage ($\mathcal{S}$), and computing ($\mathcal{C}$). Moreover, let $\boldsymbol{\rho}_d = (\rho_d^z)_{z\in\mathcal{T}} \in \mathbb{R}^3_{\geq 0}$ be the set of resources available at each edge node d. An example of the physical infrastructure and its clustered structure is shown in Figure A.3.

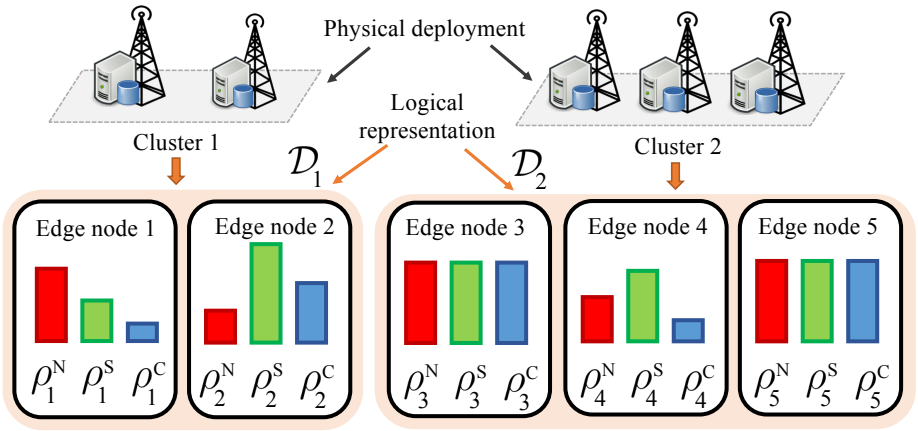


Figure A.3: System model example with $K = 2$ clusters with edge node sets $\mathcal{D}_1 = \{1, 2\}$ and $\mathcal{D}_2 = \{3, 4, 5\}$, respectively.

Let $\mathcal{R} = \{\mathcal{R}^{\mathcal{N}}, \mathcal{R}^{\mathcal{S}}, \mathcal{R}^{\mathcal{C}}\}$ be the set of slice requests submitted to Sl-EDGE, with $\mathcal{R}^{\mathcal{N}}, \mathcal{R}^{\mathcal{S}}, \mathcal{R}^{\mathcal{C}}$ being the set for networking, storage, and computing slice requests, respectively. Each request $r \in \mathcal{R}^z$ of type z is associated to a *value* $v_r^z > 0$ used by the IP to assess the importance—or monetary value—of r. Also, we define the K-dimensional *request demand array* $\boldsymbol{\tau}_r = (\tau_{r,k}^z)_{k\in\mathcal{K}, z\in\mathcal{T}}$, where $\tau_{r,k}^z \geq 0$ represents the amount of resources of type z requested by r in cluster k. Without loss of generality, we assume that $\sum_{k\in\mathcal{K}} \tau_{r,k}^z > 0$ for all $r \in \mathcal{R}$.

A.2.1 Resource Coupling and Collateral Functions

To successfully slice networking and MEC resources, it is paramount to understand the underlying dynamics between resources of different nature. To this purpose, let us consider two simple but extremely effective examples.

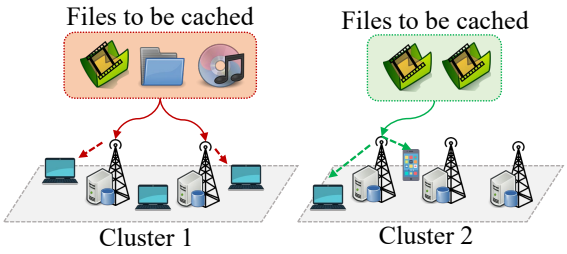


Figure A.4: Content caching.

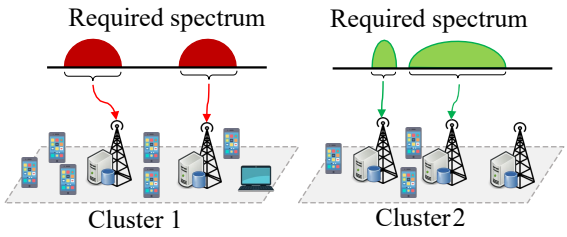


Figure A.5: 5G networking.

Content Caching. A tenant instantiates a storage slice (Figure A.4) providing caching services to subscribers, i.e., $r \in \mathcal{R}^{\mathcal{S}}$, and specifies how many megabytes ($\tau^{\mathcal{S}}_{r,k}$) should be allocated in each cluster $k \in \mathcal{K}$. The content to be cached should be first (i) transmitted and then (ii) processed by storage edge nodes. Therefore, storage activities related to caching procedures not only utilize storage resources, but also require networking and computational resources.

5G Networking. A tenant wants to provide cellular services to mobile subscribers (Figure A.5). Hence, it submits a networking slice request r of type $\mathcal{N}$ and specifies the clusters to be included in the slice and the amount of spectrum resources ($\tau^{\mathcal{N}}_{r,k}$) needed in each cluster. Edge nodes providing connectivity must (i) perform channel estimation and baseband signal processing procedures, and (ii) locally cache or buffer the data to transmit. Therefore, the allocation of resources of type $\mathcal{N}$ entails resources of type $\mathcal{C}$ and $\mathcal{S}$.

These two examples suggest that heterogeneous resources are tightly intertwined, thus motivating the need for new slicing algorithms that account for these intrinsic relationships. To incorporate coupling within the Sl-EDGE framework, we introduce the concept of *collateral functions*.

Let us consider the case where, to instantiate a slice of type $z \in \mathcal{T}$, x resources must be allocated on edge node $d \in \mathcal{D}_k, k \in \mathcal{K}$. For any resource type $t \in \mathcal{T} \setminus \{z\}$, we define the collateral function $\alpha^{z \to t}_{d,k}(x) : \mathbb{R} \to \mathbb{R}$. This function (i) reflects coupling among heterogeneous resources, and (ii) determines how many resources of type t should be allocated on edge node d when allocating x resources of type z. Of course, $\alpha^{z \to z}_{d,k}(x) = x$.

In this chapter, we model resource coupling as an increasing linear function of x. This way, the number of resources of type z needed to instantiate x resources of type t on a given edge node d can be evaluated as $\alpha^{t \to z}_{d,k}(x) = A^{t \to z}_{d,k} x$, with $A^{t \to z}_{d,k}$ being measured in units of type z per unit of type t, *e.g.*, GIPS per megabyte.[1] However, the more general case where $\alpha^{t \to z}_{d,k}(x)$ is a non-linear function can be easily related to the linear case by using well-established and accurate piece-wise

[1] We assume that $A^{t \to z}_{d,k}$ differs among edge nodes, but is uniform across services of type t. When different services of type t have different values of $A^{t \to z}_{d,k}$ (e.g., video encoding and file compression might require a different number of GIPS to process the same data) we can extend $\mathcal{T}$ by adding service-specific classes with different $A^{t \to z}_{d,k}$ values.

linearization techniques [454]. For any $k \in \mathcal{K}$ and $d \in \mathcal{D}_k$, let $\mathbf{A}_{d,k}$ be the *collateral matrix* for edge node d. Such a matrix can be written as

$$\mathbf{A}_{d,k} = \begin{pmatrix} 1 & A_{d,k}^{\mathcal{S}\to\mathcal{N}} & A_{d,k}^{\mathcal{C}\to\mathcal{N}} \\ A_{d,k}^{\mathcal{N}\to\mathcal{S}} & 1 & A_{d,k}^{\mathcal{C}\to\mathcal{S}} \\ A_{d,k}^{\mathcal{N}\to\mathcal{C}} & A_{d,k}^{\mathcal{S}\to\mathcal{C}} & 1 \end{pmatrix}. \tag{A.1}$$

A.3 Edge Slicing Problem and its Optimal Solution

The key targets of Sl-EDGE are to (i) maximize profits generated by infrastructure slice rentals, and (ii) allow location-aware and dynamic instantiation of slices in multiple clusters, while (iii) avoiding over-provisioning of resources to avoid congestion and poor performance. We formalize the above three targets with the edge slicing optimization problem (ESP) introduced below.

$$\underset{\mathbf{y},\boldsymbol{\sigma}}{\text{maximize}} \sum_{k\in\mathcal{K}} \sum_{z\in\mathcal{T}} \sum_{r\in\mathcal{R}^z} v_r^z y_r^z \tag{ESP}$$

$$\text{subject to} \sum_{d\in\mathcal{D}_k} \sigma_{r,d}^z = \tau_{r,k}^z y_r^z,\ \forall z \in \mathcal{T}, k \in \mathcal{K}, r \in \mathcal{R}^z \tag{A.2}$$

$$\sum_{r\in\mathcal{R}^z} \sum_{t\in\mathcal{T}} \alpha_{d,k}^{t\to z}(\sigma_{r,d}^t) \le \rho_{d,k}^z,\ \forall z \in \mathcal{T}, k \in \mathcal{K}, d \in \mathcal{D}_k \tag{A.3}$$

$$y_r^z \in \{0,1\},\ \ \forall z \in \mathcal{T}, r \in \mathcal{R}^z \tag{A.4}$$

$$\sigma_{r,d}^z \ge 0,\ \forall z \in \mathcal{T}, r \in \mathcal{R}^z, k \in \mathcal{K}, d \in \mathcal{D}_k \tag{A.5}$$

where $\mathbf{y} = (y_r^z)_{z\in\mathcal{T}, r\in\mathcal{R}^z}$ and $\boldsymbol{\sigma} = (\sigma_{r,d}^z)_{z\in\mathcal{T}, r\in\mathcal{R}^z, d\in\mathcal{D}}$ respectively are the *slice admission* and *resource slicing* policies. Quantity y_r^z is a binary variable such that $y_r^z = 1$ if request r is admitted, and $y_r^z = 0$ otherwise. Similarly, $\sigma_{r,d}^z$ represents the amount of resources of type z that are assigned to r on edge node d.

One can easily verify that Problem (ESP) meets the previously mentioned requirements, since it (i) aims at maximizing the total value of the admitted slice requests; (ii) guarantees that each admitted slice obtains the required amount of resources in each cluster (Constraint (A.2)), and (iii) prevents resource over-provisioning on each edge node (Constraint (A.3)).

Given the presence of both continuous and 0-1 variables, (ESP) belongs to the class of MILPs problems, well-known to be hard to solve. More precisely, (ESP) is NP-hard even in the case of requests having the same value and edge nodes belonging to a single cluster (see [537, Theorem 1]).

Problem (ESP) can be solved by means of efficient and well-established exact Branch-and-Cut (B&C) algorithms. Even though the worst-case complexity of such algorithms is exponential, B&C leverages structural properties of the problem to confine the search space, thus reducing the time needed to compute an optimal solution. The B&C procedure—not reported here for the sake of space—can be found in [455]. We now focus on how to overcome some of the limitations of B&C. Specifically, B&C suffers from high computational complexity, and requires a centralized entity with perfect knowledge, both of which are unacceptable in large-scale and dynamic networks.

A.4 Approximation Algorithms

We now design two approximation algorithms for (ESP) whose primary objective is to (i) reduce the computational complexity of the problem, and (ii) minimize the overhead traffic traversing the network. In Section A.4.1 and Section A.4.2, we present the implementation of the two algorithms, and further discuss their optimality, complexity and overhead.

A.4.1 Decentralization Through Virtualization

One of the main sources of complexity in Problem (ESP) is the large number of optimization variables $\mathbf{y}$ and $\boldsymbol{\sigma}$. However, we notice that $R = \sum_{z \in \mathcal{T}} |\mathcal{R}^z|$, where $|\cdot|$ is the set cardinality operator. On the contrary, the number of $\boldsymbol{\sigma}$ variables is $\mathcal{O}(RD)$, with D being the total number of edge nodes in the infrastructure. While R is generally limited to a few tens of requests, the number D of edge nodes deployed in the network might be very large. However, a big portion of these edge nodes are equipped with hardware and software components that are either similar or exactly the same. Thus, we can leverage similarities among edge nodes to reduce the complexity of Problem (ESP) while achieving close-to-optimal solutions and reduced control overhead.

Edge nodes with similar collateral functions will behave similarly. However, being similar in terms of α only does not suffice to determine whether or not two edge nodes are similar. In fact, nodes with similar α might have a different amount of available resources. For this reason, we leverage the concept of *similarity functions* [456].

Definition 1. Let $\Delta(d', d'') : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ be a function to score the similarity between edge nodes d' and d''. Two edge nodes $d', d'' \in \mathcal{D}$ are said to be ϵ-*similar* with respect to Δ if $\Delta(d', d'') \leq \epsilon$, for any $\epsilon \in \mathbb{R}_{\geq 0}$. If $\Delta(d', d'') = 0$, we say that d' and d'' are identical.

Through ϵ-similarity, we can first determine which edge nodes inside the same cluster are similar, and then abstract their physical properties to generate a *virtual edge node*. For the sake of generality, in this chapter we do not make any assumption on $\Delta(\cdot)$ (interested readers are referred to [457] for an exhaustive survey on the topic). However, the impact of $\Delta(\cdot)$ and ϵ on the overall system performance will be first discussed in Section A.4.1.1, and then evaluated in Section A.5.

Now we present V-ESP, an approximation algorithm that leverages virtualization concepts to compute a solution to Problem (ESP). The main steps of V-ESP are as follows:

- *Step 1 (virtual edge node generation):* for each cluster k, we build the $|\mathcal{D}_k| \times |\mathcal{D}_k|$ *similarity matrix* $\mathcal{S}_k$. For any real $\epsilon \geq 0$, element $s_{d',d''} \in \mathcal{S}_k$ indicates whether or not d' and d'' are ϵ-similar. That is, $s_{d',d''} = 1$ if $\Delta(d', d'') \leq \epsilon$, and $s_{d',d''} = 0$ otherwise. We partition the set $\mathcal{D}_k$ into $G_k \geq 1$ independent subsets that contain similar edge nodes only. The partition is generated such that $\bigcup_{g=1}^{G_k} \mathcal{D}_{k,g} = \mathcal{D}_k$ and $\mathcal{D}_{k,j} \cap \mathcal{D}_{k,i} = \emptyset$ for any $i, j = 1, 2, \ldots, G_k, i \neq j$.

  Each non-singleton subset is converted into a virtual edge node. Specifically, for each non-singleton subset $\mathcal{D}_{k,g}$, we define a virtual edge node $\tilde{d}_g$ whose available resources are equal to the sum of the available resources of all edge nodes in the subset, i.e., $\rho^z_{\tilde{d}_g,k} = \sum_{d \in \mathcal{D}_{k,g}} \rho^z_{d,k}$. The collateral function of the virtual edge node d^v_g is constructed as $\alpha^{t \to z}_{\tilde{d}_g,k} = f(\mathcal{D}_{k,g}, t, z)$, where $f(\cdot)$ is a function that generates a virtualized collateral function for virtual edge node $\tilde{d}_g$ discussed in Section A.4.1.1. We show an example of virtualization procedure in Figure A.6.

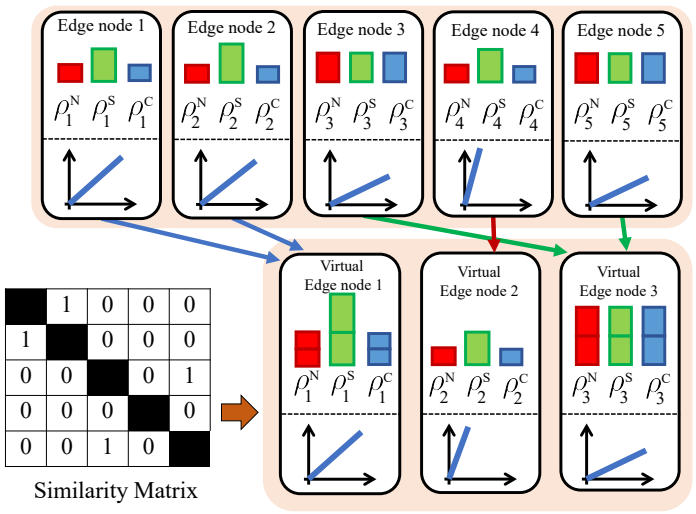


Figure A.6: An example of the virtual edge node generation in Step 1. The similarity matrix determines which edge nodes can be aggregated. Similar edge nodes (*i.e.*, $\{1,2\}$ and $\{3,5\}$) are aggregated into virtual ones. Edge node 4 is not aggregated as it has similar resources to $\{1,2\}$, but different collateral function.

- *Step 2 (virtual edge nodes advertisement):* each cluster k advertises to Sl-EDGE the set $\mathcal{D}_k = (\tilde{d}_g)_{g=1,\dots,G_k}$ of G_k virtualized edge nodes, as well as their virtual collateral functions $(\alpha^{t\to z}_{\tilde{d}_g,k})$ and available resources $(\rho^z_{\tilde{d}_g,k})$.

- *Step 3 (solve virtualized ESP):* Sl-EDGE solves Problem (ESP) with virtualized edge nodes through B&C. Slice admission and resource slicing policies $(\tilde{\mathbf{y}}^*, \tilde{\mathbf{oe}}^*)$ are computed and each cluster receives the 2-tuple $(\tilde{\mathbf{y}}^*, \tilde{\mathbf{oe}}^*_{\mathbf{k}})$, with $\tilde{\boldsymbol{\sigma}}^*_k = (\tilde{\sigma}^{z*}_{r,\tilde{d}_g})_{z\in\mathcal{T},r\in\mathcal{R},g=1,\dots,G_k}$ being the resource allocation policy over the virtualized edge nodes of cluster k.

- *Step 4 (virtualized edge node resource allocation):* upon receiving $(\tilde{\mathbf{y}}^*, \tilde{\mathbf{oe}}^*_{\mathbf{k}})$, cluster k solves G_k Linear Programming (LP) problems in parallel, one for each virtual edge node g. These LPs are formulated as follows:

$$\text{find} \quad \boldsymbol{\sigma}_{k,g} \tag{A.6}$$

$$\text{subject to} \quad \sum_{d\in\mathcal{D}_{k,g}} \sigma^z_{r,d} = \tilde{\sigma}^{z*}_{r,\tilde{d}_g}, \ \forall r \in \mathcal{R} \tag{A.7}$$

$$\text{Constraints (A.3), (A.5)}$$

  which can be optimally solved by computing *any* feasible resource allocation policy that satisfies all constraints.

- *Step 5 (slicing policies construction):* let $\boldsymbol{\sigma}^*_{k,g} = (\boldsymbol{\sigma}^*_{k,d})_{d\in\mathcal{D}_{k,g}}$ be the optimal solution of the g-th instance of (A.6). The resource slicing policy $\boldsymbol{\sigma}^*_k$ for cluster k is constructed by stacking all G_k individual solutions computed by individual clusters, *i.e.*, $\boldsymbol{\sigma}^*_k = (\boldsymbol{\sigma}^*_{k,g})_{g=1,\dots,G_k}$. The final slice admission and resource slicing policies are $(\tilde{\mathbf{y}}^*, \mathbf{oe}^*)$ with $\boldsymbol{\sigma}^* = (\boldsymbol{\sigma}^*_k)_{k\in\mathcal{K}}$.

Through V-ESP, each cluster exposes $G_k \leq |\mathcal{D}_k|$ virtual edge nodes only, rather than $|\mathcal{D}_k|$ (Steps 1-2). Thus, virtualization reduces the number of edge nodes and thus the number of variables in (ESP). Moreover, since virtualization leaves the structure of the slicing problem unchanged, we efficiently solve Step 3 through the same B&C techniques used for (ESP). In addition, while Steps 3-5 are executed whenever a new slicing policy is required (e.g., tenants submit new slice requests or the slice rental period expires), Steps 1-2 are executed only when the structure of the physical infrastructure changes (e.g., edge nodes are turned on/off or are subject to hardware modifications). This way, we can further reduce the overhead. In short, V-ESP splits the computational burden among the NO (Step 3) and the edge nodes (Steps 1-2 and 4), which jointly provides the high efficiency typical of centralized approaches while enjoying the reduced complexity of decentralized algorithms.

A.4.1.1 Design Aspects of Virtualization

Step 1 relies on ϵ-similarity to aggregate edge nodes and reduce the search space. Intuitively, the higher the value of ϵ, the smaller the set of virtual edge nodes generated in Step 1, the faster Sl-EDGE computes solutions in Step 3. However, large ϵ values might group together edge nodes with different available resources and collateral functions. In this case, (i) Step 1 might produce virtual edge nodes that poorly reflect physical edge nodes features, and (ii) solutions computed at Step 3 might not be feasible when applied to Step 4. Thus, there is a trade-off between accuracy and computational speed, which will be the focus of Section A.5.4.

Another aspect that influences the efficiency and feasibility of solutions generated by the V-ESP algorithm is the function $f(\cdot)$, which transforms collateral functions of similar edge nodes into an aggregated collateral function. Recall that $f(\cdot)$, which can be represented as a collateral matrix (A.1), must mimic the actual behavior of physical edge nodes belonging to the same subset g. To avoid overestimating the capabilities of virtual edge nodes, and producing infeasible solutions, the generic element of the virtual collateral matrix (A.1) for virtual edge node $\tilde{d}_g$ is set to $A^{z\to t}_{\tilde{d}_g,k} = \max_{d\in\mathcal{D}_{k,g}}\{A^{z\to t}_{d,k}\}, \forall z, t \in \mathcal{T}$. Although this model underestimates the capabilities of physical edge nodes and may admit fewer requests than the optimal algorithm, it always produces feasible solutions in Steps 3 and 4.

A.4.2 Distributed Edge Slicing

In this section we design a distributed edge slicing algorithm for Problem (ESP) such that clusters can locally compute slicing strategies. We point out that making (ESP) distributed is significantly challenging. In fact, both utility function and constraints are coupled with each other through the optimization variables $\boldsymbol{\sigma}$ and $\mathbf{y}$. This complicates the decomposition of the problem into multiple independent sub-problems.

In order to decouple the problem into multiple independent sub-problems, we introduce the auxiliary variables $\mathbf{y}_k = (y^z_{r,k})_{z\in\mathcal{T},r\in\mathcal{R}^z}$ such that $y^z_{r,k} = y^z_r$ for any request r and cluster k. Thus,

Problem (ESP) can be rewritten as

$$\underset{\boldsymbol{\sigma},\mathbf{y}}{\text{maximize}} \ \frac{1}{|\mathcal{K}|}\sum_{k\in\mathcal{K}}\sum_{z\in\mathcal{T}}\sum_{r\in\mathcal{R}^z} v_r^z y_{r,k}^z \tag{D-ESP}$$

$$\text{subject to} \ \sum_{d\in\mathcal{D}_k}\sigma_{r,d}^z = \tau_{r,k}^z y_{r,k}^z, \ \forall z\in\mathcal{T}, k\in\mathcal{K}, r\in\mathcal{R}^z \tag{A.8}$$

$$y_{r,k}^z = y_{r,m}^z, \ \ \forall z\in\mathcal{T}, (k,m)\in\mathcal{K}^2, r\in\mathcal{R}^z \tag{A.9}$$

$$y_{r,k}^z \in \{0,1\} \ \ \forall z\in\mathcal{T}, r\in\mathcal{R}^z \tag{A.10}$$

$$\text{Constraints (A.3), (A.5)}$$

where $\mathbf{y} = (y_{r,k}^z)_{z\in\mathcal{T},r\in\mathcal{R}^z,z\in\mathcal{T}}$, while Constraint (A.9) guarantees that different clusters admit the same requests.

Problem (D-ESP) is with separable variables with respect to the K clusters. That is, Problem (D-ESP) can be split into K sub-problems, each of them involving only variables controlled by a single cluster. To effectively decompose Problem (D-ESP) we leverage the Alternating Direction Method of Multipliers (ADMM) [458]. The ADMM is a well-established optimization tool that enforces constraints through quadratic penalty terms and generates multiple sub-problems that can be iteratively solved in a distributed fashion.

The augmented Lagrangian for Problem (D-ESP) can be written as follows:

$$\begin{aligned} L(\boldsymbol{\sigma},\mathbf{y},\boldsymbol{\lambda},\rho) = &\sum_{k\in\mathcal{K}}\sum_{z\in\mathcal{T}}\sum_{r\in\mathcal{R}^z} v_r^z y_{r,k}^z \\ &- \sum_{z\in\mathcal{T}}\sum_{r\in\mathcal{R}^z}\sum_{k\in\mathcal{K}}\sum_{m\in\mathcal{K}} \lambda_{r,k,m}^z (y_{r,k}^z - y_{r,m}^z) \\ &- \frac{\rho}{2}\sum_{z\in\mathcal{T}}\sum_{r\in\mathcal{R}^z}\sum_{k\in\mathcal{K}}\sum_{m\in\mathcal{K}} (y_{r,k}^z - y_{r,m}^z)^2 \end{aligned} \tag{A.11}$$

where $\boldsymbol{\lambda} = (\lambda_{r,k,m}^z)$ are the so-called *dual variables*, and $\rho > 0$ is a step-size parameter used to regulate the convergence speed of the distributed algorithm [458].

Let $k \in \mathcal{K}$, we define $\mathbf{y}_{-k} = (\mathbf{y}_m)_{m\in\mathcal{K}\setminus\{k\}}$ which identifies the slice admission policies taken by all clusters except for cluster k. Similarly, we define $\boldsymbol{\sigma}_{-k} = (\boldsymbol{\sigma}_m)_{m\in\mathcal{K}\setminus\{k\}}$. Problem (D-ESP) can be solved through the following ADMM-based iterative algorithm

$$\{\mathbf{y}_k,\boldsymbol{\sigma}_k\}(t+1) = \underset{\mathbf{y}_k,\boldsymbol{\sigma}_k}{\text{argmax}}\, L(\boldsymbol{\sigma}_k,\mathbf{y}_k,\boldsymbol{\sigma}_{-k}(t),\mathbf{y}_{-k}(t),\boldsymbol{\lambda}(t),\rho) \tag{A.12}$$

$$\lambda_{r,k,m}^z(t+1) = \lambda_{r,k,m}^z(t) + \rho(y_{r,k}^z(t+1) - y_{r,m}^z(t+1)) \tag{A.13}$$

where each cluster sequentially updates $\mathbf{y}_k$ and $\boldsymbol{\sigma}_k$, while the dual variables $\boldsymbol{\lambda}$ are updated as soon as all clusters have updated their strategy according to (A.12). To update (A.12) each cluster solves the following quadratic problem

$$\underset{\boldsymbol{\sigma}_k,\mathbf{y}_k}{\text{maximize}} \sum_{z\in\mathcal{T}}\sum_{r\in\mathcal{R}^z} \tilde{v}_{r,k}^z(\mathbf{y}_{-k}(t-1),\boldsymbol{\lambda}(t-1)) y_{r,k}^z - 2\rho(y_{r,k}^z)^2 \tag{DC-ESP}$$

$$\text{subject to Constraints (A.3), (A.5), (A.8), (A.9),}$$

where $\tilde{v}^z_{r,k}$ is the adjusted value of request r defined as

$$\tilde{v}^z_{r,k}(\mathbf{y}_{-k}(t), \boldsymbol{\lambda}(t)) = v^z_{r,k} - \sum_{m \in \mathcal{K} \setminus \{k\}} \left(\lambda^z_{r,k,m}(t) - \lambda^z_{r,m,k}(t)\right) + \rho\, \phi_{r,k}(\mathbf{y}_{-k}(t)) \tag{A.14}$$

and $\phi_{r,k}(\mathbf{y}_{-k}(t)) = \sum_{m \in \mathcal{K} \setminus \{k\}} y^z_{r,m}(t)$ is used by cluster k to obtain the number of clusters that have accepted request r.

The advantages of Problem (DC-ESP) are that (i) clusters do not need to advertise the composition of the physical infrastructure to the IP or to other clusters, and (ii) it can be implemented in a distributed fashion. Indeed, at any iteration t, the only parameters needed by cluster k to solve (A.12) are the dual variables $\boldsymbol{\lambda}(t-1)$ and the number $\phi_{r,k}(\mathbf{y}_{-k}(t))$ of clusters that admitted the request r at the previous iteration.

It has been shown that ADMM usually enjoys linear convergence [459], but improper choices of ρ might generate oscillations. To overcome this issue and achieve convergence, we implemented the approach proposed in [458, Eq. (3.13)] where ρ is updated at each iteration of the ADDM. The optimality and convergence properties of DC-ESP will be extensively evaluated in Figures A.10 and A.11.

A.5 Numerical Results

We now assess the performance of the three slicing algorithms described in Section A.3 and Section A.4 by (i) simulating a MEC-enabled 5G network, and by (ii) comparing the algorithms with the recently published DIRECT framework [7].

We consider a scenario where edge nodes provide mobile subscribers with 5G NR connectivity as well as storage and computation MEC services, such as caching and video decoding. We assume that (i) edge nodes share the same NR numerology—more precisely, networking resources are arranged over an OFDM-based resource grid with 50 PRBs, and (ii) edge nodes are equipped with hardware components with up to 1 Terabyte of storage capabilities and a maximum of 200 GIPS. We fix the number of PRBs for each edge node, while we randomly generate the amount of computation and storage resources at each simulation run. To simulate a realistic scenario with video transmission, storage and transcoding applications, collateral matrices in (A.1) are generated by randomly perturbing the following matrix $\mathbf{A}^0 = [1, 0.0382, 0.1636; 26.178, 1, 0.0063; 0.49, 0.15, 1]$ at each run. To give an example, processing a data rate of 15.264 Mbit/s (equivalent to LTE 16-QAM with 50 PRBs) requires 24.4224 GIPS (*e.g.*, turbo-decoding) [460], which results in $A^{\mathcal{N}\to\mathcal{C}}_{d,k} = 0.49$ GIPS/PRB. Similarly, a 1-second long compressed FullHD 30fps video approximately occupies 500 kB and requires 80 GIPS to decode, thus $A^{\mathcal{C}\to\mathcal{S}}_{d,k} = 0.0062$ MB/GIPS. We assume that the physical infrastructure consists of $K = 5$ MEC-enabled edge clusters, each containing the same number of edge nodes but equipped with different amounts of available resources and collateral functions. We model $\Delta(\cdot)$ as the *cosine similarity function* [456] and, unless otherwise stated, the aggregation threshold is set to $\epsilon = 0.1$. Slice requests and the demanded resources are randomly generated at each run.

In the following, we refer to the optimal B&C algorithm in Section A.3 as O-ESP. Similarly, the two approximation algorithms proposed in Section A.4.1 and Section A.4.2 are referred to as V-ESP and DC-ESP respectively.

A.5.1 The Impact of Coupling on MEC-enabled 5G Systems

DIRECT [7] provides an efficient distributed slicing algorithm for networking and computing resources in MEC-enabled 5G networks. Although this approach does not account for the case where edge nodes provide both networking and MEC functionalities, it represents the closest work to Sl-EDGE.

Moreover, DIRECT does not explicitly slice storage resources. Thus, to perform a fair comparison, we consider the case where tenants do not request any storage resource. Let $D_c = 75$ be the total number of edge nodes in the network. We let tenants randomly generate slice requests to obtain networking and computational resources. Results are shown in Figure A.7, where any positive value indicates resource over-provisioning.

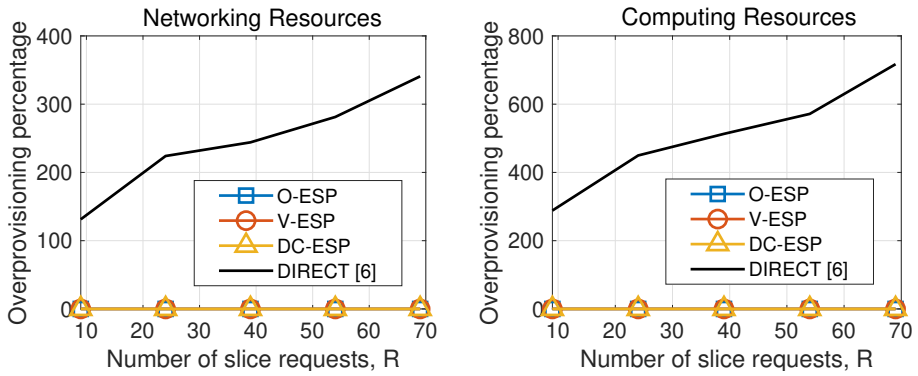


Figure A.7: Over-provisioning of networking and computational resources of Sl-EDGE and DIRECT [7].

Figure A.7 shows that Sl-EDGE never produces over-provisioning slices. Conversely, since DIRECT does not account for coupling among heterogeneous resources on the same edge node, it always incurs in over-provisioning, allocating up to 6x more resources than the available ones. These results show that *already existing solutions, which perform well in 5G systems with networking and MEC functionalities decoupled at different points of the network, cannot be readily applied to scenarios where resources are simultaneously handled by edge nodes—which strongly motivates the need for approaches such as Sl-EDGE.*

A.5.2 Maximizing the Number of Admitted Slices

Let us now focus on the scenario where the IP owning the infrastructure aims at maximizing the number of slice requests admitted by Sl-EDGE—to maximize resource utilization, for instance. Although each slice request r comes with an associated (monetary) value $v_r > 0$, this can be achieved by resetting the value of each request to $v_r = 1$ in Problem (ESP).

Figure A.8 reports the performance of Sl-EDGE as a function of the total number R of generated slice requests for different values of the number of edge nodes D_c. We notice that the number of admitted slices increases as the slice requests that are submitted to Sl-EDGE increase (left-side plot). However, Figure A.8 (center) clearly shows that the percentage of admitted slices rapidly decreases as R increases (only 10 requests are admitted by O-ESP when $D_c = 75$ and $R = 70$). This is due to the scarcity of resources at edge nodes, which prevents the admission of a large number of slices.

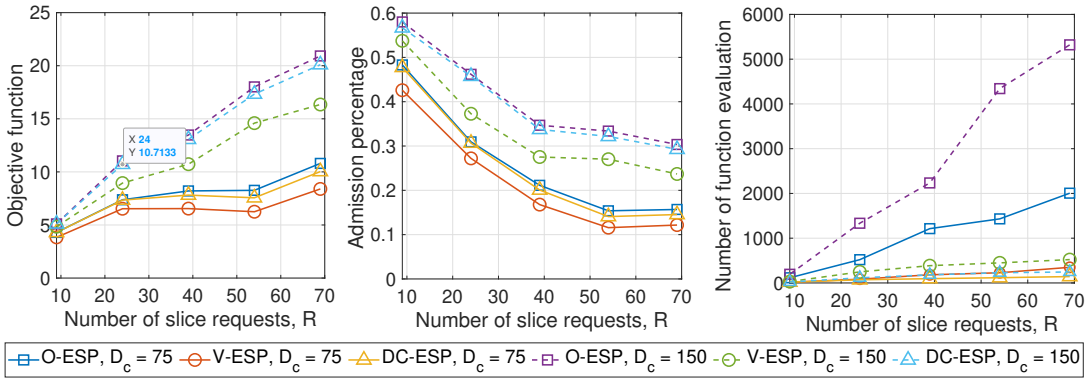


Figure A.8: Sl-EDGE performance when maximizing the number of admitted slice requests.

Thus, IPs should either provide edge nodes with more resources, or increase the number of deployed edge nodes. Figure A.8 (left), indeed, shows that denser deployments of edge nodes (*i.e.*, $D_c = 150$) allows more slices to coexist on the same infrastructure.

Finally, the right-hand side plot of Figure A.8 shows the computational complexity of the three algorithms measured as the number of function evaluations needed to output a solution. As expected, the complexity of all algorithms increases as both R and D_c increase. Moreover, O-ESP, a fully centralized algorithm, has the highest computational complexity. V-ESP and DC-ESP, reduced-complexity versions of O-ESP, instead show lower complexity. However, V-ESP and DC-ESP admit approximately 10% and 16% fewer requests than O-ESP, respectively, due to their non-optimality.

A.5.3 Maximizing the Profit of the Infrastructure Provider

Let us now consider the case of slice admission and instantiation for profit maximization (Figure A.9). In this case, Sl-EDGE selects the slice requests to be admitted to maximize the total (monetary) value of the admitted slices. Similarly to Figure A.8, Figure A.9 (center) shows that increasing R reduces the percentage of admitted slices.

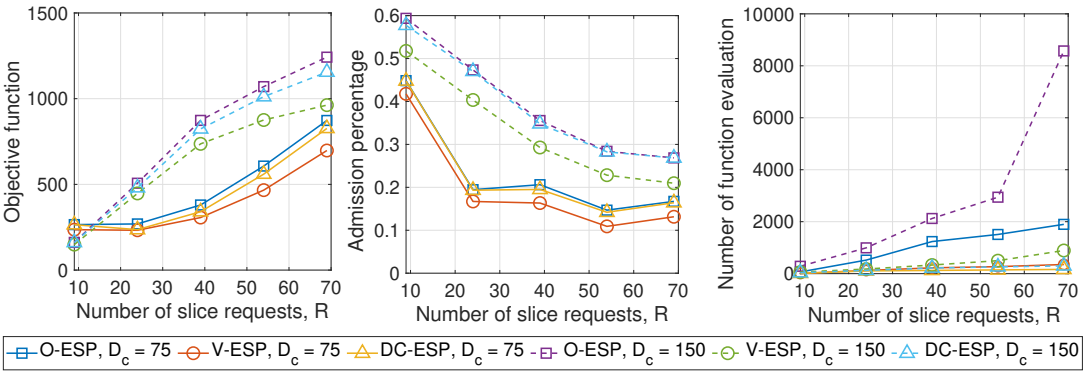


Figure A.9: Sl-EDGE performance maximizing the profit of the IP.

When compared to the problem described in Section A.5.2, this profit maximization problem

differs because (i) even if the number of edge nodes is small (i.e., $D_c = 75$), profit maximization produces profits that rapidly increase with R, and (ii) the percentage of admitted requests steeply decreases as R increases. Indeed, the higher the number of requests, the higher the probability that slices with high value are submitted by tenants. In this case, Sl-EDGE prioritizes more valuable requests at the expenses of others.

A.5.4 Impact of ϵ on the V-ESP Algorithm

Finally, we investigate the impact of different choices on the performance of the V-ESP algorithm. Recall that ϵ controls the number of edge nodes that are aggregated into virtual edge nodes (Section A.4.1). The higher the value of ϵ, the higher the percentage of edge nodes that are aggregated, and the smaller the number of virtual edge nodes generated by Sl-EDGE.

Figure A.10 shows the computational complexity of V-ESP as a function of ϵ for different values of the number of deployed edge nodes D_c. As expected, ϵ impacts neither O-ESP nor DC-ESP,

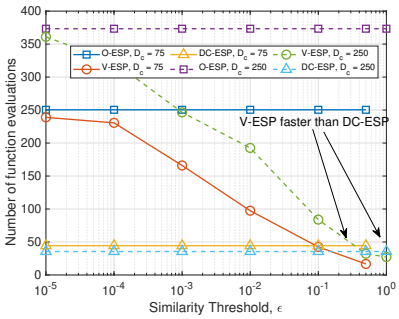


Figure A.10: Computational complexity of the proposed algorithms as a function of the similarity parameter ϵ.

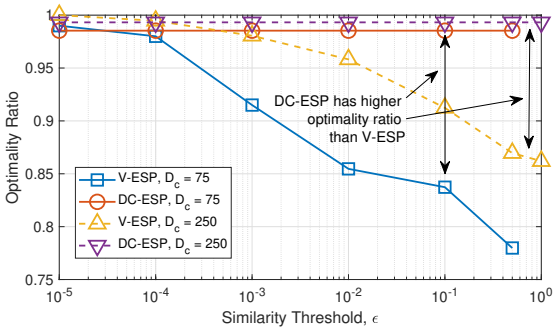


Figure A.11: Optimality ratio of the algorithms proposed in Section A.4 as a function of the similarity parameter ϵ.

whereas its impact on V-ESP is substantial. Indeed, larger values of ϵ reduce the number of physical edge nodes in the network, which are instead substituted by virtual edge nodes (one per aggregated group). This reduction eventually results in decreased computational complexity. Surprisingly, Figure A.10 also shows that V-ESP enjoys an even lower computational complexity than that of the distributed DC-ESP when $\epsilon \approx 1$. Recall that V-ESP centrally determines an efficient slicing strategy over virtualized edge nodes, and these strategies are successively enforced by each cluster. This means that V-ESP can compute an efficient slicing policy as rapidly as DC-ESP while avoiding any coordination among different clusters. Overall, Figure A.10 shows that V-ESP (green dashed line) computes a solution 7.5x faster than O-ESP (purple dashed line) when ϵ is high.

However, Figure A.11 shows that reduced computation complexity comes at the expense of efficiency. Indeed, the optimality ratio (*i.e.*, the distance of the output of any approximation algorithm from the optimal solution of the problem) decreases as ϵ increases up to a maximum of 25% loss with respect to the optimal. Although the optimality ratio for $\epsilon = 0.1$ is high (i.e., 92% and 84% for $D_c = 75$ and $D_c = 250$, respectively), clearly a trade-off between computational complexity and efficiency should be considered.

A.6 Sl-EDGE Prototype

We prototyped Sl-EDGE on the *Arena* testbed (see Section 3.5). We leveraged 14 USRPs of the above-mentioned testbed (10 USRPs N210 and 4 USRPs X310) to prototype Sl-EDGE. In our testbed, an edge node consists of one USRP and one server: the former provides networking capabilities, while the latter provides storage and computing resources. The testbed configuration adopted to prototype and evaluate Sl-EDGE performance is shown in Figure A.12.

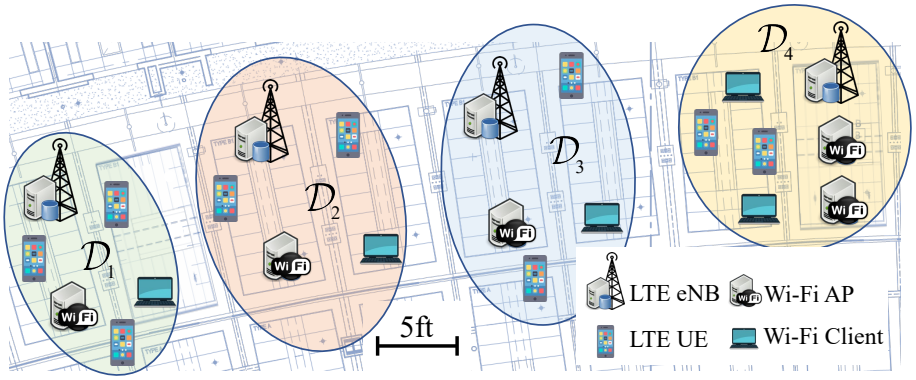


Figure A.12: Experimental Sl-EDGE setup on the Arena testbed.

We used the *SCOPE* framework of Section 3.2 to implement LTE *networking* slices. Since LTE and NR resource block grids are similar, we are confident that our findings remain qualitatively valid for 5G scenarios. We leveraged SCOPE to instantiate 4 eNBs on USRPs X310, while we employed 9 COTS cellular phones (Samsung Galaxy S5) as users. Each user downloads a data file from one of the rack servers, which are used as caching nodes with *storage* capabilities. We consider three tenants demanding an equal number of LTE network slices (i.e., LS_1, LS_2 and LS_3) at times $t_0 = 0$ s, $t_1 = 40$ s, and $t_2 = 80$ s. Each tenant controls a single slice only and serves a set of UEs located in different clusters as shown in Table A.1 (right).

Table A.1: Per-cluster admitted PRBs in LS_1, and UE association.

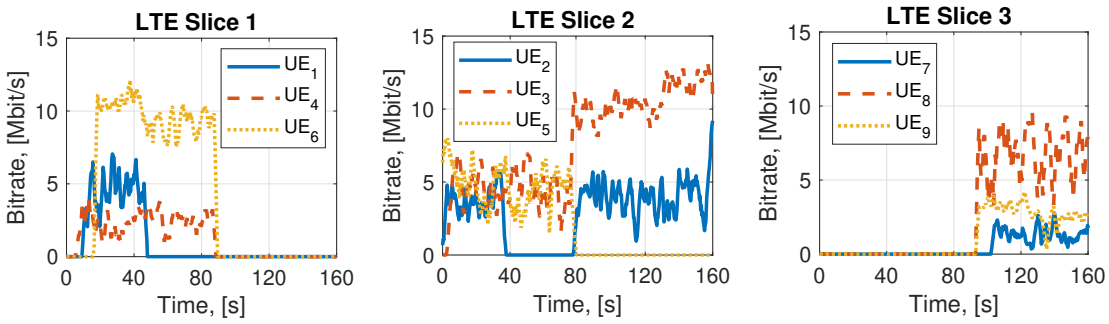
	$t_0 = [0, 40]$s	$t_1 = [40, 80]$s	$t_2 = [80, 160]$s
$\mathcal{D}_1$	24	0	0
$\mathcal{D}_2$	0	0	0
$\mathcal{D}_3$	24	24	0
$\mathcal{D}_4$	42	24	0

LS_1	LS_2	LS_3
UE_1	UE_2	UE_7
-	UE_3	UE_8
UE_4	UE_5	UE_9
UE_6	-	-

To test Sl-EDGE abilities to handle slices involving both *networking and computation capabilities*, we also implemented a video streaming slice where edge nodes stream a video file stored in an Apache instance through the *dash.js* reference player [461] running on the Chrome web browser. DASH allows real-time adaptation of the video bitrate, according to the client requests and the available resources [462]. Each streaming video was sent to the receiving server of the rack through USRPs N210 acting as SDR-based Wi-Fi transceivers (Wi-Fi Access Points (APs) and Clients in Figure A.12), using the GNU Radio-based IEEE 802.11a/g/p implementation [331]. At the same time, the edge node performed transcoding of video chunks using *ffmpeg* [463]. We point out that

each SDR can play multiple roles in the cluster (e.g., USRPs X310 can act as Wi-Fi transceiver/LTE eNB), and the actual role is determined at run-time based on the slice types allocated to each tenant.

A demonstration of the operations of Sl-EDGE in the scenario of Figure A.12 is shown in Figures A.13, A.14 and A.15. Overall, the prototype of Sl-EDGE allocates and supports 11 heteroge-



(a) (b) (c)

Figure A.13: Instantiation of LTE network slices.

neous slices simultaneously: 3 for cellular connectivity, 3 for video streaming over Wi-Fi, and 5 for computation with the *ffmpeg* transcoding. Bitrate results for the LTE slices and individual UEs are reported in Figure A.13, where we show that Sl-EDGE provides an overall instantaneous throughput of 37 $\mathrm{Mbit/s}$.

It is worth pointing out that the throughput of each LTE slice, and thus each UE, varies according to the amount of resources allocated to the tenants. An example is shown in Table A.1 (left), where we report the output of Sl-EDGE O-ESP algorithm (*i.e.*, the amount of PRBs allocated to LTE Slice 1 (LS_1)) in each cluster. Such an allocation impacts the throughput of UEs attached to slice LS_1. As an example, in Figure A.13 we notice that $\mathrm{UE}_6 \in \mathcal{D}_4$ is allocated 42 PRBs at t_0, 24 PRBs at t_1, and 0 PRBs at t_2 and approximately achieves a throughput of $12\mathrm{Mbit/s}$, $8\mathrm{Mbit/s}$ and $0\mathrm{Mbit/s}$, respectively.

The video streaming application from Figure A.14 involves 5 tenants that share 3 non-overlapping channels, allocated in any cluster $\mathcal{D}_i, i \in \{1, 2, 3, 4\}$. To avoid co-channel interference, Sl-EDGE

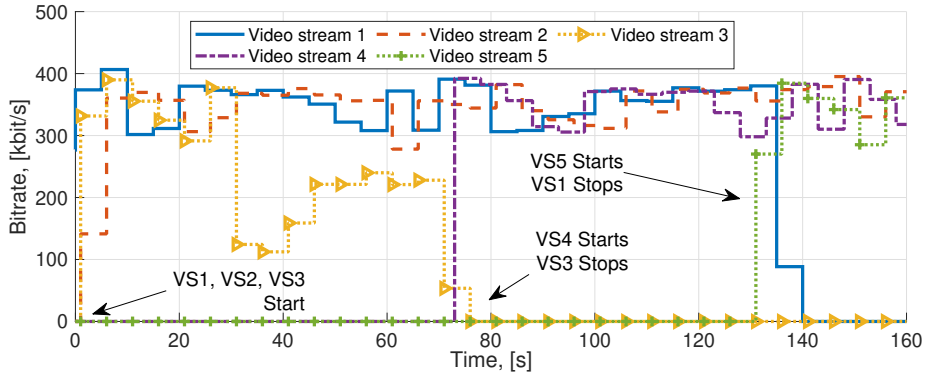


Figure A.14: Dynamic instantiation of video streaming slices.

only admits 3 flows at any given time. As Figure A.14 shows, during the first 70 seconds of the

experiment only the slices for tenants 1, 2 and 3 are admitted, while tenant 4 needs to wait for tenant 3 to stop the video streaming before being granted a slice. Similarly, the slice for tenant 5 starts at time $t = 140$ s, when the flow of tenant 1 stops. Meanwhile, the tenants submit requests for computation slices to transcode the videos with ffmpeg, which compete with the SCOPE and GNU Radio slices necessary for LTE connectivity and video streaming in the 3 LTE eNBs and 5 Wi-Fi APs of the 4 clusters. Moreover, in each server one of the 6 cores is reserved to the operating system exclusively and is never allocated to tenants. Figure A.15 shows that Sl-EDGE limits the total CPU utilization to

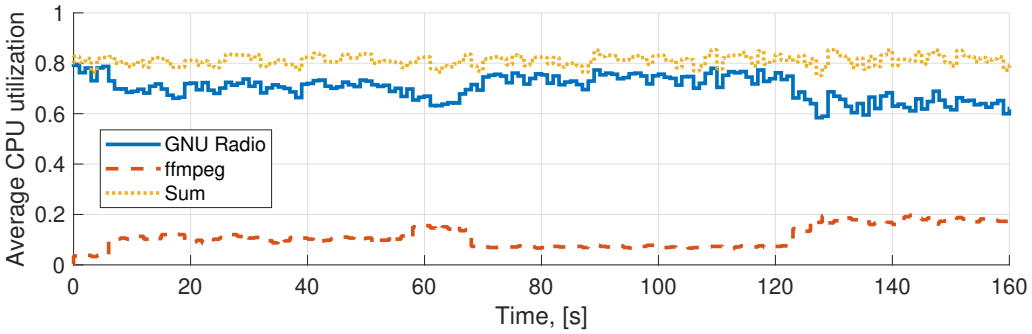


Figure A.15: CPU utilization for networking and transcoding services.

83%, which demonstrates that Sl-EDGE avoids over-provisioning of available resources.

A.7 Related Work

Motivated by the ever-increasing interests from both NOs and standardization entities [449, 464, 465], network slicing and multi-access edge computing technologies have recently become "all the rage" in the wireless research community [4, 12, 63, 466–469]. Lately, we have seen a deluge of algorithms to efficiently slice portions of the network and instantiate service-specific slices. These solutions leverage optimization [470–473, 521], game-theory [62, 474, 475] and machine learning [450] tools.

Concurrently, MEC has demonstrated to be an effective methodology to significantly reduce latency. This paradigm has been successfully used to provide task offloading [476–479], augmented reality [480–482], low-latency video streaming [483, 484], and caching [485, 486], among others. We refer the interested reader to the following surveys on network slicing and MEC [20, 72, 374, 487, 488].

These works are extremely effective when the two technologies are considered as independent entities operating on the same infrastructure. However, they are prone to resource over-provisioning when both technologies coexist on the same edge nodes and share the same resources. Ndikumana et al. [452] consider the allocation of heterogeneous resources for task offloading problems in MEC ecosystems, while in [473] Agarwal et al. consider the problem of jointly allocating CPUs and virtual network functions for network slicing applications. Similarly, in [450] Van Huynh et al. account for slice networking, computation, and storage resources by designing a deep dueling neural network that determines which slices to admit, maximizes the network provider's reward, and avoids resource over-provisioning. Although [450] accounts for resource heterogeneity, the authors only focus on the slicing admission control problem, and do not consider how to partition the resources of each slice among the nodes of the network. A heterogeneous resource orchestration framework

for edge computing ecosystems called *Senate* is presented by Castellano et al. in [451]. Senate leverages Distributed Orchestration Resource Assignment (DORA) and election algorithms to allow the instantiation of multiple virtual network functions on the same infrastructure. However, this work focuses on the wired portion of the edge network only.

The closest work to Sl-EDGE is [7], where Liu and Han proposed a distributed slicing framework for MEC-enabled wireless networks called DIRECT. Despite being extremely successful in slicing networks with MEC resources residing in dedicated servers close to the base stations, DIRECT does not account for the case where both networking and MEC resources coexist on the same edge node.

Sl-EDGE separates itself from prior work as it makes a substantial step forward toward the coexistence of network slicing and MEC. Indeed, we consider the challenging case of edge nodes jointly providing wireless network access and MEC functionalities to mobile users. Furthermore, we model the intrinsic coupling among heterogeneous resources on edge nodes, and design Sl-EDGE, a slicing framework that leverages such a coupling to instantiate heterogeneous slices on the same physical infrastructure.

A.8 Conclusions

In this chapter, we have presented Sl-EDGE, a unified MEC slicing framework that instantiates heterogeneous slice services (e.g., video streaming, caching, 5G network access) on edge devices. We have first shown that the problem of optimally instantiating joint network-MEC slices is NP-hard. Then, we have proposed distributed algorithms that leverage similarities among edge nodes and resource virtualization to instantiate heterogeneous slices faster and within a short distance from the optimum. We have assessed the performance of our algorithms through extensive numerical analysis and on a 64-antenna testbed with 24 software-defined radios. Results have shown that Sl-EDGE instantiates slices 6x more efficiently then state-of-the-art MEC slicing algorithms, and that Sl-EDGE provides at the same time highly-efficient slicing of joint LTE connectivity, video streaming over Wi-Fi, and ffmpeg video transcoding.

Appendix B

Coordinated 5G Network Slicing

The sheer number of mobile subscriptions worldwide—predicted to be around 8.9 billions by the end of 2025 [489]—will generate amounts of traffic that currently commercially-available wireless infrastructures and spectrum bands are not able to support [490]. Critically, traditional one-size-fits-all resource allocation policies will not enable *dynamic, effective and efficient* radio access strategies, which motivates the already increasing demand for novel solutions to design and deploy faster, lower-latency wireless cellular connections (see Chapters 2, 4 and 5).

To address the above issues, RAN slicing has been recently welcomed as a promising approach by the academia and industry alike [4, 12, 63, 307, 450, 466–468, 470, 491, 492]. This technology allows multiple MVNOs to share the same physical infrastructure—ultimately realizing a game-changing vision that completely overturns the traditional model of single ownership of all network resources. Although a similar concept is already widely applied in the context of cloud computing by companies such as Amazon and Microsoft [493], *RAN slicing is an intrinsically different problem as: (i) spectrum is a scarce resource for which over-provisioning is not possible (see Appendix A), and (ii) interference jeopardizes isolation across slices belonging to different MVNOs, thus resulting in performance degradation if not handled properly* [12, 13].

As shown in Figure B.1, in RAN slicing applications each MVNO controls a separate "slice" of the network. Slices can be assigned/revoked by the IP which determines the slices to be admitted to the system and how many resources each slice should receive. Once RAN slicing policies have been defined, a key problem is how to allocate the spectrum Resource Blocks (RBs) as prescribed by the slicing policy. This problem, also referred to as the RAN slicing enforcement problem (RSEP) [12], ensures that if an MVNO has been assigned a slice of 15% of the spectrum resources, such MVNO receives approximately 15% of the available RBs.

The design and evaluation of *RAN slicing enforcement* algorithms is paramount to implement in practice the slicing policy of the IP. Moreover, to be effective, RAN slicing enforcement algorithms must facilitate interference-mitigating strategies such as Inter-base Station Power Control (IBSPC) [12, 494, 495], MIMO [496], and Coordinated Multi-point (CoMP) [496–498] schemes such as Joint Transmission (JT) [499, 500]. Therefore, *it becomes imperative to design effective and efficient slicing enforcement algorithms assigning the same (or similar in time/frequency) RBs to the same MVNOs when BSs interfere among themselves.*

To illustrate the above point, we consider the cellular network scenario depicted in Figure B.1.

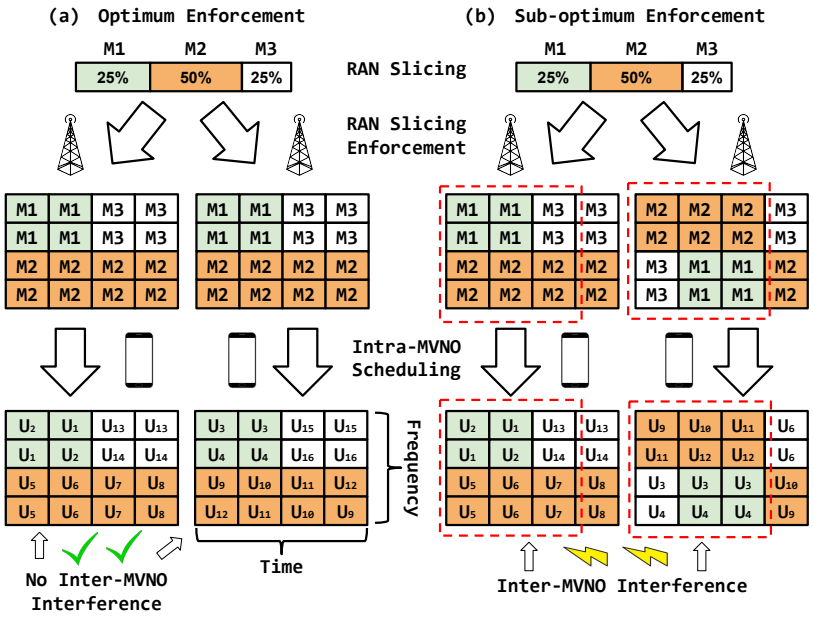


Figure B.1: Optimum and Sub-optimum RAN Slicing Enforcement.

Here, the IP administers two BSs (assumed to be close enough to interfere with each other) and 16 RBs (i.e., 4 frequency units during 4 time units). We consider the case where three MVNOs, namely M1, M2 and M3, have been assigned the following slice: $\mathrm{M1} = 25\%, \mathrm{M2} = 50\%, \mathrm{M3} = 25\%$, on both the BSs. Figure B.1a shows an optimum slicing enforcement, represented as two *RB allocation matrices* (RBAMs), where inter-MVNO interference is absent (i.e., MVNOs control the same RBs at the two BSs). In this case, MVNOs have maximum flexibility and can easily mitigate interference between their cellular users residing in the two BSs by using IBSPC. Conversely, Figure B.1b shows sub-optimum RBAMs causing inter-MVNO interference during 12 RBs, which will likely result in performance degradation due to poor interference management.

To further demonstrate the negative impact of inter-MVNO interference, we ran a series of experiments on the LTE-compliant testbed described in Section B.6. Similarly to Figure B.1, in such experiments we deploy two LTE base stations and instantiate two RAN slices controlled by MVNOs M_1 and M_2, respectively. Each slice is assigned with 50% of the available RBs and serves a set of cellular users (i.e., commercial LTE smartphones). In Figure B.2, we report the network throughput of the network. Specifically, we compare measured throughput with (RBs are allocated to minimize inter-MVNO interference as in Figure B.1a) and without (Figure B.1b) slice isolation. It is easy to notice that slice isolation considerably improves network throughput and provides a throughput gain up to 3 Mbps. *In Section B.6, we show how our algorithms considerably improve network throughput with respect to slicing enforcement algorithms that do not enforce isolation across slices, such as the one in Figure B.1b.*

Although the problem of RAN slicing has attracted large interest [4, 63, 307, 374, 450, 466–470, 487, 501], only few works have tackled the issue of physical-level allocation of spectrum resources to MVNOs [12]. This is not without a reason; the design of slicing enforcement algorithms presents the following unique challenges, which are substantially absent in traditional RAN resource allocation

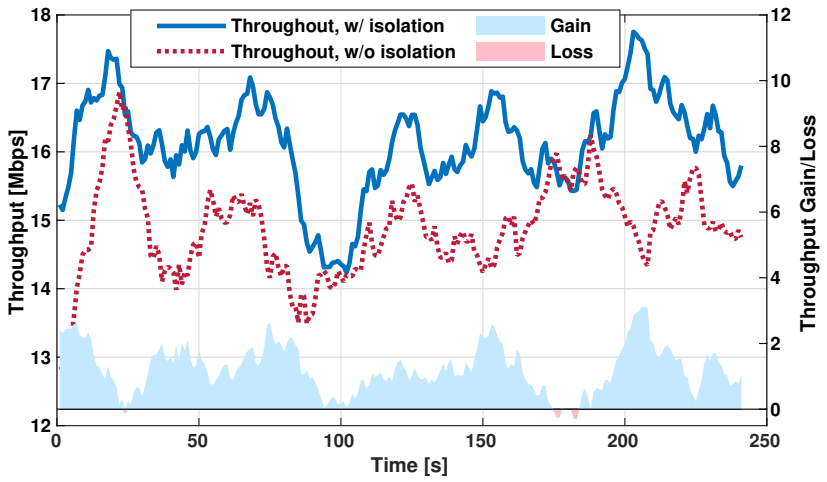


Figure B.2: Impact of coordination-based slicing on network throughput. Lines represents throughput measurements, shaded areas indicate gains and losses.

scenarios:

1. *Enabling of advanced 5G technologies*: 5G systems will heavily rely upon advanced signal processing and RF transmission technologies such as IBSPC, JC, CoMP and MIMO. This techniques considerably improve network performance, but require coordination among BSs in proximity. For this reason, the RB allocation should facilitate and foster such coordination;

2. *Isolation:* As demonstrated in Figure B.2, to increase efficiency, orthogonality among each RAN slice must be ensured. That is, each RB should be allocated to only one MVNO to avoid interference and other performance-degrading factors [4, 502, 503];

3. *Contract Compliance*: MVNOs stipulate contracts with the IP to obtain control over the amount of resources specified by the slicing policy. In other words, the RSEP must guarantee that if an MVNO has been assigned 30% of spectrum resources and it is paying to get them, then it should also receive 30% of the total RBs available.

In this chapter, which was published as [524], we design, analyze and experimentally evaluate RAN slicing enforcement algorithms that address the three critical challenges mentioned above. Specifically, we make the following contributions:

- We formulate the *RAN slicing enforcement problem* (RSEP), and show that it is NP-hard. Therefore, we propose approximation and heuristic solutions tailored for different network scales, optimality and timing requirements;

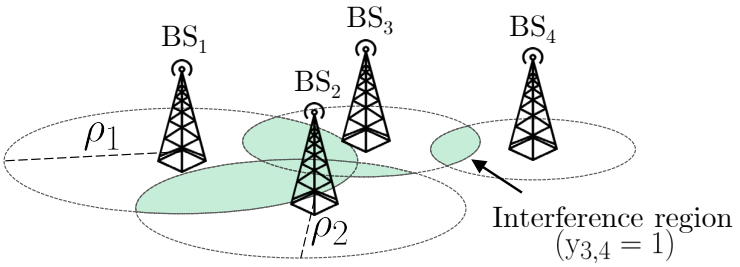
- We show via simulations that the computation time of the proposed algorithms can be as low as few hundreds of microseconds without considerably impacting the overall efficiency of the computed solutions. We also show that by enforcing slice orthogonality, the proposed approach reduces inter-MVNO interference, thus effectively doubling the overall SINR of the network if compared to traditional RAN slicing enforcement approaches;

- We demonstrate the effectiveness of the proposed algorithms through experimental evaluation on a LTE-compliant testbed composed of 2 LTE base stations and 6 COTS users. Results show that our approach outperforms other slicing techniques that do not enforce isolation across RAN slices. Specifically, our algorithms compute slicing strategies that lead to SINR and throughput improvements up to 27%.

The remainder of this chapter is organized as follows. The considered RAN model is illustrated in Section B.1; Section B.2 introduces the RSEP problem, and Section B.3 presents optimal, approximated and heuristic solutions to the RSEP. Section B.4 presents two effective complexity reduction techniques to further speed-up the computation of enforcement strategies. The performance of the proposed algorithms are assessed numerically and experimentally in Sections B.5 and B.6, respectively. Finally, Section B.7 surveys the literature on the topic, while concluding remarks are given in Section B.8.

B.1 System Model and RAN Slicing Overview

We consider the RAN shown in Figure B.3, consisting of a set $\mathcal{B} = \{1, 2, \ldots, B\}$ of B BSs associated with a *coverage area* $\rho_b,\ b \in \mathcal{B}$. Two BSs b and b' are *interfering* (or *adjacent*) if $\rho_b \cap \rho'_b \neq \emptyset$, i.e., their coverage areas overlap. We define $\mathbf{Y} = (y_{b,b'})_{b,b' \in \mathcal{B}}$ as a symmetric adjacency matrix where $y_{b,b} = 0$ for all $b \in \mathcal{B}$, $y_{b,b'} = 1$ if BSs b and b' interfere with each other, and $y_{b,b'} = 0$ otherwise. An illustrative example of the adjacency matrix is shown in Figure B.3.



$\mathbf{Y}$

	BS1	BS2	BS3	BS4
BS1	0	1	1	0
BS2	1	0	1	0
BS3	1	1	0	1
BS4	0	0	1	0

Figure B.3: An illustrative example with 4 BSs and their corresponding adjacency matrix $\mathbf{Y}$. Green areas show interference (or adjacency) regions where coverage areas overlap.

The RAN is administered by an IP, who periodically rents virtual RAN slices built on top of the underlying physical network $\mathcal{B}$ to a set $\mathcal{M} = \{1, 2, \ldots, M\}$ of M MVNOs. Without loss of generality, we assume RAN slices are valid for T consecutive time slots [12]. We can utilize the T parameter to model different scenarios with different temporal scales. As an example, large T values can be used to model environments with slow-varying dynamics (e.g., cellular networks in rural areas during nighttime), and small T values are used to model environments with fast-varying dynamics (e.g., urban areas during daytime) requiring frequent update of RAN slice enforcement policies to adapt to mobility and channel dynamics.

In line with 5G NR and LTE standards, we assume that spectrum resources are represented as RBs, where each RB corresponds to the minimum scheduling unit [504]. We also consider an OFDMA channel access scheme where RBs are organized as in a time-frequency *resource grid* with

N_{RB} subcarriers and T temporal slots. Thus, the set of available resources at each BS is $\mathcal{R}$, with $|\mathcal{R}| = N_{RB} \cdot T$, where each RB in $\mathcal{R}$ can be represented as a 2-tuple (n, t) with $n = 1, 2, \ldots, N_{RB}$ and $t = 1, 2, \ldots, T$. We assume that all BSs share the same resource grid structure, the case where this assumption is relaxed is considered in [14].

The interaction between MVNOs and the IP can be summarized as illustrated in Figure B.4. First,

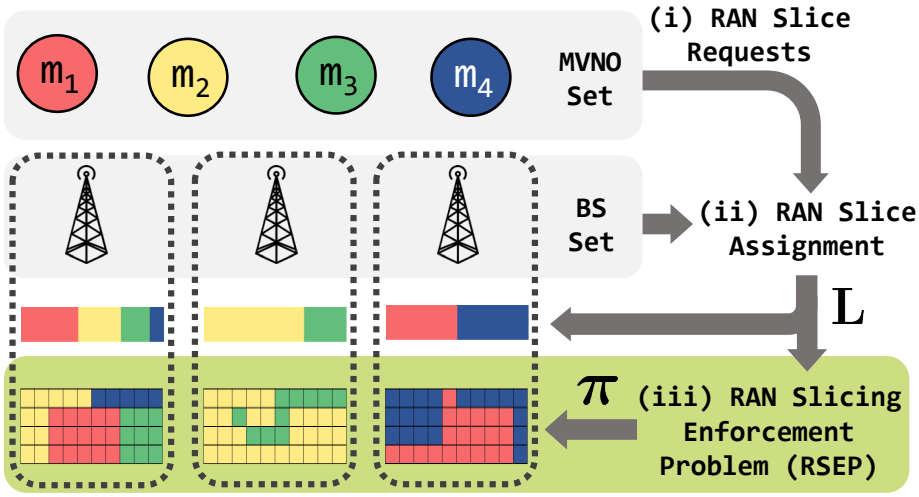


Figure B.4: The RAN slicing architecture.

(i) MVNOs' *RAN slice requests* are collected by the IP. Then, (ii) the IP determines which requests should be admitted to the system, and generates a *slicing profile* $\mathbf{L} = (L_{m,b})_{m \in \mathcal{M}, b \in \mathcal{B}}$ where $L_{m,b}$ represents the amount of resources that the IP should allocate to MVNO $m \in \mathcal{M}$ on BS b in the time span $0 \leq t \leq T$ (i.e., *RAN Slice Assignment*), and (iii) computes a slicing enforcement policy $\boldsymbol{\pi}$ that allocates RBs among the MVNOs such that all admitted requests are satisfied (i.e., *RAN Slice Enforcement Problem*).

Problem (ii) has been already extensively investigated in the literature [4, 374, 474, 487, 501, 505–510]. For this reason, in this chapter we instead address point (iii) by showing how the IP can compute an efficient slicing enforcement policy $\boldsymbol{\pi}$ that satisfies the three requirements described at the beginning of this Appendix, i.e., enabling of advanced 5G technologies, isolation and contract compliance.

In this chapter, We tackle the problem from an IP's point of view which has no access to mobile users' location, demanded traffic and channel conditions. Thus, we consider the case where MVNOs submit slice requests that reflect the current state of the network and, for privacy and business concerns, do not share the above information with the IP. Accordingly, MVNOs requests a target number of RBs that guarantees satisfaction of service-level agreements and meet desired key performance indicators. A summary of the used notation in this Appendix is shown in Table B.1.

B.2 The RAN Slicing Enforcement Problem (RSEP)

For any given slicing profile $\mathbf{L}$ and BS b, we identify the subset $\mathcal{M}_b \subseteq \mathcal{M}$ of MVNOs that include BS b in their RAN slice as $\mathcal{M}_b = \{m \in \mathcal{M} : L_{m,b} > 0\}$.

Table B.1: Summary of Notation.

Variable	Description
$\mathcal{B}$	Set of base stations (BSs)
ρ_b	Coverage area of base station $b \in \mathcal{B}$
$\mathcal{M}$	Set of Mobile Virtual Network Operators (MVNOs)
$\mathbf{Y}$	Adjacency matrix
$y_{b,b'}$	Adjacency indicator
N_{RB}	Number of available subcarriers
T	Number of temporal slots within the slicing window
$\mathcal{R}$	Set of the available resources at each BS, $\lvert\mathcal{R}\rvert = N_{RB} \cdot T$
$\mathbf{L}$	RAN Slicing profile
$\mathcal{M}_b$	Set of MVNOs including BS b in their RAN slice
$x_{m,b,n,t}$	RB allocation indicator
$\boldsymbol{\pi}$	Slicing enforcement policy
$\mathit{\Pi}$	Set of all feasible slicing enforcement policies $\boldsymbol{\pi}$

Let $x_{m,b,n,t} \in \{0,1\}$ be the *RB allocation indicator* such that $x_{m,b,n,t} = 1$ if RB $(n,t) \in \mathcal{R}$ is allocated to MVNO m, $x_{m,b,n,t} = 0$ otherwise. Also, let $\boldsymbol{\pi} = (\boldsymbol{\pi}_b)_{b\in\mathcal{B}}$ be the *slicing enforcement policy*, where $\boldsymbol{\pi}_b = (\boldsymbol{\pi}_{m,b})_{m\in\mathcal{M}}$ and $\boldsymbol{\pi}_{m,b}$ represents the set of RBs on BS b that are allocated to MVNO m. In more detail, for any RB $(n,t) \in \mathcal{R}$, we have that $(n,t) \in \boldsymbol{\pi}_{m,b} \iff x_{m,b,n,t} = 1$. Hence, the set $\mathit{\Pi}$ of all feasible slicing enforcement policies $\boldsymbol{\pi}$ can be defined as:

$$\begin{aligned}\mathit{\Pi} = \{\boldsymbol{\pi} = (\boldsymbol{\pi}_{m,b})_{m\in\mathcal{M},b\in\mathcal{B}} : |\boldsymbol{\pi}_{m,b}| = L_{m,b} \wedge \\ \boldsymbol{\pi}_{m,b} \cap \boldsymbol{\pi}_{m',b} = \emptyset \; \forall m \neq m', m, m' \in \mathcal{M},\ b \in \mathcal{B}\}\end{aligned} \tag{B.1}$$

To properly formulate the RSEP, we now introduce the concept of linked RBs.

Definition 2 (Linked RB)**.** A given RB (n,t) on the resource grid is *linked* to MVNO m on two interfering BS b and b' if and only if $x_{m,b,n,t} = x_{m,b',n,t} = 1$ and $y_{b,b'} = 1$.

Linked RBs indicate those RBs that have been assigned to the same MVNO on adjacent BSs. Specifically, a linked RB allows the corresponding MVNO to simultaneously access a specific spectrum portion in the same time slot from two or more BSs. This is relevant because (i) linked RBs enable 5G advanced transmission schemes (e.g., distributed beamforming, MIMO, CoMP transmissions and power control) among nearby BSs; (ii) as shown in Figure B.1, linked RBs can be used to deploy fully-orthogonal RAN slices that do not interfere with each other, and hence (iii) linked RBs do not generate inter-MVNO interference, thus avoiding any need for centralized coordination or distributed coordination among MVNOs.

It is clear that the maximization of the number of simultaneously linked RBs addresses the three issues identified at the beginning of this Appendix. Thus, we focus our attention on this approach. By leveraging the concept of linked RBs, for each MVNO m we define the number of linked RBs associated to interfering BSs b and b', i.e., $y_{b,b'} = 1$, as follows:

$$n_{b,b',m} = y_{b,b'} \cdot |\boldsymbol{\pi}_{m,b} \cap \boldsymbol{\pi}_{m,b'}|, \tag{B.2}$$

where the relationship $n_{b,b',m} = n_{b',b,m}$ always holds for all $b, b' \in \mathcal{B}$ and $m \in \mathcal{M}$.

For each MVNO $m \in \mathcal{M}$, the total number N_m of linked RBs on the corresponding RAN slicing profile $\mathbf{L}$ is

$$N_m = \frac{1}{2} \sum_{b\in\mathcal{B}} \sum_{b'\in\mathcal{B}\setminus\{b\}} y_{b,b'} \cdot n_{b,b',m}, \tag{B.3}$$

where the $1/2$ factor is introduced to avoid counting the same RBs twice, and $n_{b,b',m}$ is defined in (B.2).

With (B.2) and (B.3) at hand, we can formally define the RSEP as follows:

$$\underset{\boldsymbol{\pi} \in \mathit{\Pi}}{\text{maximize}} \sum_{m \in \mathcal{M}} N_m \tag{RSEP}$$

In a nutshell, the objective of Problem RSEP is to compute a feasible slicing enforcement policy $\boldsymbol{\pi}$ that maximizes the total number of linked RBs while guaranteeing that such policy does not violate the feasibility constraint $\boldsymbol{\pi} \in \mathit{\Pi}$. Moreover, Figure B.1 shows that the formulation in Problem RSEP is particularly well-suited for RAN slicing problems. This is because it satisfies MVNOs requirements in terms of number of obtained RBs, helps orthogonality among slices through the reduction of inter-MVNO interference, and enables coordination-based 5G communications such as CoMP, JT and beamforming. In Sections B.5 and B.6, we will demonstrate how increasing the number of linked RBs of the system improves key performance metrics such as throughput and SINR.

B.3 Addressing the RSEP Problem

To solve Problem RSEP, we need to compute a slicing enforcement policy by exploring the feasible set $\mathit{\Pi}$ searching for a solution that maximizes the number of linked RBs. However, the formulation in Problem RSEP does not in itself provide any intuitions on how a solution can be computed. For this reason, now we: (i) reformulate Problem RSEP by using the RB allocation indicators introduced in Section B.2; (ii) show that the resulting problem is NP-hard, and (iii) present a number of algorithms to address and solve Problem RSEP.

B.3.1 Optimal Solution

By using the definition of the RB allocation indicator $x_{m,b,n,t} \in \{0, 1\}$ and from (B.1), (B.3) can be reformulated as

$$N_m = \frac{1}{2} \sum_{t=1}^{T} \sum_{n=1}^{N_{RB}} \sum_{b \in \mathcal{B}} \sum_{b' \in \mathcal{B} \setminus \{b\}} y_{b,b'} x_{m,b,n,t} x_{m,b',n,t} \tag{B.4}$$

Let us consider the matrices $\mathbf{B} = \mathbf{Y} \otimes \mathbf{I}_{N_{RB} \cdot T}$ and $\mathbf{Q} = \mathbf{I}_M \otimes \mathbf{B}$, where $\otimes$ stands for Kronecker product and $\mathbf{I}_k$ is the $k \times k$ identity matrix. From (B.4), it can be easily shown that

$$\sum_{m \in \mathcal{M}} N_m = \frac{1}{2} \mathbf{x}^\top \mathbf{Q} \mathbf{x}.$$

Accordingly, Problem RSEP can be reformulated as

$$\underset{\boldsymbol{x}}{\text{maximize}} \ \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} \tag{RSEP-QP}$$

$$\text{subject to} \sum_{t=1}^{T}\sum_{n=1}^{N_{RB}} x_{m,b,n,t} = L_{m,b},\ \forall b \in \mathcal{B},\ \forall m \in \mathcal{M} \tag{C1}$$

$$\sum_{m\in\mathcal{M}} x_{m,b,n,t} \leq 1, \qquad \forall (n,t) \in \mathcal{R},\ \forall b \in \mathcal{B} \tag{C2}$$

$$x_{m,b,n,t} \in \{0,1\}, \forall (n,t) \in \mathcal{R}, \forall b \in \mathcal{B}, \forall m \in \mathcal{M} \tag{C3}$$

where $\mathbf{x} = (x_{m,b,n,t})_{m,b,n,t}$ is a $MBN_{RB}T \times 1$ column array.

In Problem RSEP-QP, Constraint (C1) ensures that all MVNOs receive the assigned number of RBs, while Constraint (C2) guarantees that each RB is allocated to one MVNO only. Finally, Constraint (C3) expresses the boolean nature of the RB allocation indicator. We point out that problems RSEP and RSEP-QP are equivalent. Indeed, the latter is a reformulation of the former in terms of the RB allocation indicator. However, we have been able to show that the RSEP can be modeled as a 0-1 (or binary) Quadratic Programming (QP) problem. Thus, Problem RSEP-QP—and thus Problem RSEP—is NP-Hard (see [524, Theorem 1].

The indefiniteness of the $\mathbf{Q}$ matrix prevents the application of well-established results on quadratic functions where positive/negative definiteness guarantees the existence of a unique global solution a priori.

To find an optimal solution to Problem RSEP-QP it is possible to apply spatial Branch and Bound (sB&B) techniques for non-linear non-convex problems. These algorithms have been shown to globally solve these class of problems by iteratively generating convex relaxations whose accuracy is refined at each iterations [511]. Although sB&B makes it possible to globally solve Problem RSEP-QP, its complexity is still too high to be effectively employed in real-world 5G network deployments where the number of base stations and users is extremely large.

The objective of the following two subsections is to address the above issues and design algorithms that can compute effective RAN slice enforcing policies with low computational complexity.

B.3.2 Approximated Solution

Let $V = M \cdot B \cdot N_{RB} \cdot T$, and let us consider the following transformed version of Problem RSEP-QP

$$\underset{\boldsymbol{x}}{\text{maximize}} \ \frac{1}{2}\mathbf{x}^\top(\mathbf{Q} + 2\lambda\mathbf{I}_V)\mathbf{x} - \lambda \boldsymbol{e}^\top\mathbf{x} \tag{RSEP-EQ}$$

$$\text{subject to} \ \text{(C1), (C2)}$$

$$0 \leq x_{m,b,n,t} \leq 1, \forall (n,t) \in \mathcal{R}, \forall b \in \mathcal{B}, \forall m \in \mathcal{M} \tag{C4}$$

where $\lambda \in \mathbb{R}$ is a real-valued parameter whose relevance to Problem RSEP-EQ will be explained in Theorem 1 (proof reported in [524, Section V-B]), and $\boldsymbol{e}^\top = (1, 1, \ldots, 1)$. The following theorem holds.

Theorem 1. *There exists $\lambda \in \mathbb{R}$ such that Problem RSEP-EQ is equivalent to Problem RSEP-QP. Also, let z^* be the largest (positive) eigenvalue of $\mathbf{Q}$. For any $\lambda \geq -z^*$, Problem RSEP-EQ is a quadratic convex problem over the unit hypercube.*

Algorithm 2 RSEP-MLF

```
Input $\mathcal{B}; \mathcal{M}; \mathbf{Y}; \mathbf{L}$;
Output A MLF RBs allocation $\mathbf{x}^G = (x^G_{m,b,n,t})_{m,b,n,t}$;
Set $x^G_{m,b,n,t} = 0$ for all $m \in \mathcal{M}, b \in \mathcal{B}, (n,t) \in \mathcal{R}$;
Compute the linking index $\mathbf{l} = (l_m)_{m \in \mathcal{M}}$;
$\mathcal{M}^G \leftarrow$ Sort $\mathcal{M}$ by $l_m$ in decreasing order;
while $\mathcal{M}^G \neq \emptyset$ do
    for each BS $b \in \mathcal{B}$ do
        Update $x^G_{m,b,n,t}$ by allocating $L_{\mathcal{M}^G(1),b}$ subsequent RBs to MVNO $m$ on BS $b$;
    end for
    $\mathcal{M}^G \leftarrow \mathcal{M}^G \setminus \{\mathcal{M}^G(1)\}$;
end while
```

Remarks. Theorem 1 shows that we can relax the binary constraint of Problem RSEP-QP with a penalty term λ. When λ is large enough, RSEP-EQ and RSEP-QP are equivalent and produce the same solutions. Otherwise, equivalence does not hold and solutions computed by RSEP-EQ might substantially deviate from the optimal ones computed by RSEP-QP.

In general, local and global solutions of convex quadratic maximization problems (and the corresponding concave quadratic minimization problems) lie on the vertices of the feasibility set [512]. Since the vertex space is considerably smaller than the complete feasibility set $\mathit{\Pi}$ considered in Problem RSEP-QP, Problem RSEP-EQ is easier to solve when compared to Problem RSEP-QP. Specifically, approaches such as cutting plane and extreme point ranking methods [512] can be used to efficiently solve Problem RSEP-EQ.

B.3.3 Heuristic Solution

Although Problem RSEP-EQ has lower complexity than Problem RSEP-QP, in the worst case it still requires exponential time with respect to the number of vertices, which spurred us to design polynomial-time algorithms.

Given Problem RSEP-QP maximizes the number of shared RBs, we can allocate as many linked RBs as possible to those MVNOs that request the highest amount of RBs on multiple interfering BSs. Indeed, MVNOs that request the greatest number of resources on different interfering BSs are also expected to produce a high number of linked RBs. Accordingly, for each MVNO m we define the *linking index* l_m as

$$l_m = \sum_{b \in \mathcal{B}} \sum_{b' \in \mathcal{B} \setminus \{b\}} \min\{L_{m,b}, L_{m,b'}\} y_{b,b'} \tag{B.5}$$

The linking index is used to sequentially allocate RBs to those MVNOs with the highest linking index. We refer to this procedure as the *Most Linked First* (MLF) procedure, which is illustrated in Algorithm 2 and works as follows:

1. we generate set $\mathcal{M}^G = \mathcal{M}$ in ascending order of l_m. Specifically, for any $m, k \in \mathcal{M}^G$, $m < k$ if $l_m \geq l_k$;

2. we start allocating RBs on all BSs in sequential order to the first MVNO in $\mathcal{M}^G$, i.e., the MVNO whose linking index l_m is the highest among all MVNOs in $\mathcal{M}$. When all RBs are allocated to the considered MVNO, say m', we remove it from $\mathcal{M}^G$ and we set $l_{m'} = 0$;

3. if $\mathcal{M}^G = \emptyset$, we stop. Otherwise, we re-execute Step 2 until all MVNOs are assigned to the required RBs.

Line 4 requires to compute (B.5) which has complexity $\mathcal{O}(MB^2)$, while Line 5 has complexity $\mathcal{O}(M \log M)$. The `while` loop at Line 6 has complexity $\mathcal{O}(N_{RB}BM)$. Thus, the complexity of MLF is $\mathcal{O}(C)$, where $C = \max\{MB^2, M \log M, N_{RB} \cdot B \cdot M\}$.

B.3.4 Improved RSEP-MLF

The greedy algorithm RSEP-MLF enjoys fast convergence time at the price of sub-optimal performance [12]. Thus, we design a novel heuristic that reduces the gap between RSEP-QP and RSEP-MLF while keeping the computational complexity as low as possible. To this end, we present RSEP-IMLF, which improves upon RSEP-MLF by iteratively adjusting the RB allocation strategy to increase the number of linked RBs.

First, note that the RSEP is shift-invariant with respect to the indexing of the RB (n) and temporal slot (t). This is because, for any given solution $\mathbf{x}^*$, the solution $\mathbf{x}$ with $x_{m,r,1,t} = x^*_{m,r,N_{RB},t}$ and $x_{m,r,N_{RB},t} = x^*_{m,r,1,t}$ for all m, r and t is clearly still equivalent to $\mathbf{x}^*$ as it produces the same number of linked RBs as $\mathbf{x}^*$. In general, we can extend this result to any reshape procedure that maintains the cardinality of $\mathcal{R}$ equal to $N_{RB} \cdot T$. We show this in Figure B.5, where the original RB grid (left) contains $N_{RB} \cdot T = 12$ RBs and has 6 linked RBs. Figure B.5 shows that by reshaping the resource grid into a row vector does not change the amount of linked RBs.

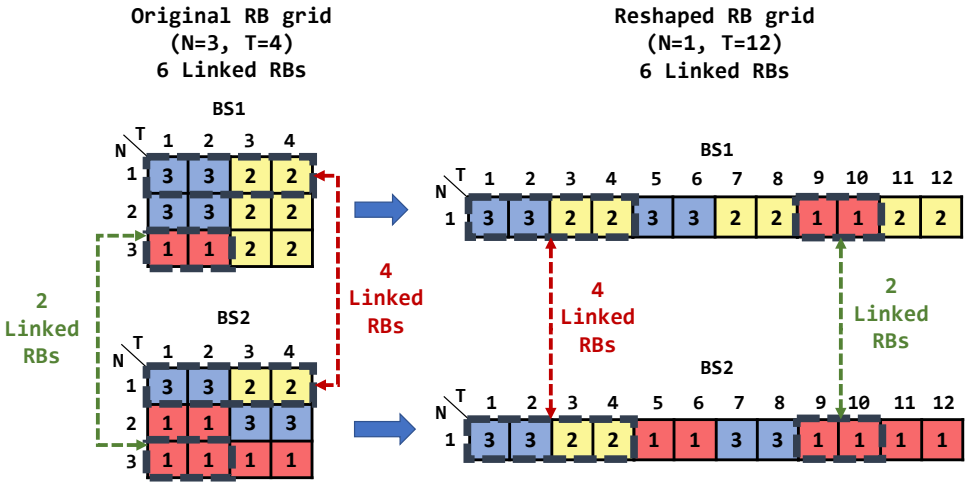


Figure B.5: An illustrative example of a RB grid reshaping with $B = 2$ BSs, $M = 3$ MVNOs and 6 linked RBs. The original RB grid is shown on the left, while the reshaped grid is shown on the right.

Another important notion is the concept of *RB allocation matrix* (RBAM). Let us consider the reshaped RB grid $\mathcal{R} \in \mathbb{R}^{N_{RB}T \times 1}$, the RBAM is represented by the matrix $\boldsymbol{\sigma} = (\boldsymbol{\sigma}_b)_{b \in \mathcal{B}}$ where

$\boldsymbol{\sigma}_b(\mathbf{x}^*) = (\sigma_{b,\tau})_{\tau\in\mathcal{R}} : \mathcal{X} \to \mathcal{R}$. Henceforth, b and τ will represent rows and columns of $\boldsymbol{\sigma}$, respectively. For any slicing enforcement solution $\mathbf{x} \in \mathcal{X}$, the RBAM builds a map between each RB in $\mathcal{R}$ and the MVNO that has been assigned with that RB on BS b. Let $M_{b,\tau}(\mathbf{x}^*)$ be the MVNO that RB τ has been assigned to, i.e., the MVNO m such that $x_{m,b,\tau} = 1$. Accordingly, we set $\sigma_{b,\tau} = M_{b,\tau}(\mathbf{x}^*)$. An example of a possible RBAM with $B = 4$ BSs and $M = 5$ MVNOs is shown in Figure B.6.

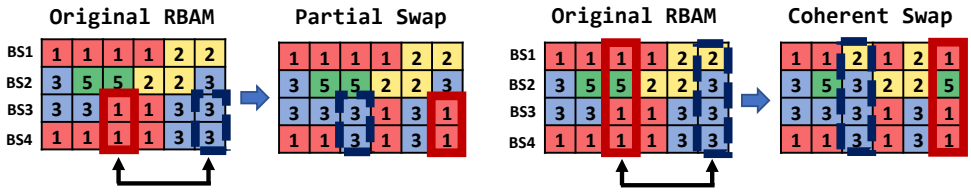


Figure B.6: An illustrative example of a RB allocation matrix (RBAM) and swapping procedures with $B = 4$ BSs and $M = 5$ MVNOs.

We are now ready to introduce the concept of *swapping*. Specifically, we say that two columns τ_1 and τ_2 of the RBAM are *coherently swapped* when all their corresponding entries σ_{b,τ_1} are replaced with those of σ_{b,τ_2} and *vice versa*. On the contrary, two columns are *partially swapped* when only a portion of entries is replaced among two columns. An example of a coherent swap is shown in the right side of Figure B.6, where the third and sixth columns are swapped. Instead, the left side shows a partial swap where only the two bottom elements of the columns of the RBAM are swapped.

By leveraging the concepts of reshaping, RBAM and swapping, we can finally develop an improved version of RSEP-MLF (i.e., RSEP-IMLF, see Algorithm 3) which works as follows:

1. Compute a slicing enforcement solution $\mathbf{x} \in \mathcal{X}$ via RSEP-MLF and derive the corresponding RBAM $\boldsymbol{\sigma}$. Define $\mathcal{B}^* = \mathcal{B}$;
2. Select the row $b_0 \in \mathcal{B}^*$ in $\boldsymbol{\sigma}$ with the smallest number of distinct MVNOs and remove it from $\mathcal{B}^*$, i.e., $\mathcal{B}^* = \mathcal{B}^* \setminus \{b_0\}$;
3. Pick the row $b^* \in \mathcal{B}^*$ that shares the highest number of linked RBs with b_0;
4. Select at random two columns τ_1 and τ_2. Perform a single-row partial swap on the RBAM $\boldsymbol{\sigma}$ by swapping the two elements (b_0, τ_1) and (b_0, τ_2). If the partial swap has improved the number of linked RBs, we update the RBAM accordingly. This step is repeated at most I_S times, where $I_S > 0$ is a parameter specifying the maximum number of trials RSEP-IMLF tries to improve upon the current slicing enforcement strategy;
5. If $\mathcal{B}^* = \emptyset$, we stop. Otherwise we re-execute Step 2).

The rationale behind RSEP-IMLF is to compute a sub-optimal solution fast, and then iteratively try to increment the total number of linked RBs by testing a limited amount of swapping combinations. As discussed in Section B.3.3, the complexity of Step 1 is $\mathcal{O}(C)$, where $C = \max\{MB^2, M\log M, N_{RB} \cdot B \cdot M\}$. Step 2 is executed once, and its complexity is $\mathcal{O}(B)$, while the complexity of Step 3 is $\mathcal{O}(BI_S)$. Accordingly, the overall complexity of RSEP-IMLF is $\mathcal{O}(C + B + BI_S) = \mathcal{O}(C + BI_S)$, meaning that RSEP-IMLF contributes to the overall complexity of RSEP-MLF with an additional linear complexity term.

Algorithm 3 RSEP-IMLF

```
Input $\mathcal{B}; \mathcal{M}; \mathbf{Y}; \mathbf{L}$;
Output A RBs allocation $\mathbf{x}^G = (x^G_{m,b,n,t})_{m,b,n,t}$;
Set $\mathcal{B}^* \leftarrow \mathcal{B}$;
$\mathbf{x}^* \leftarrow$ A MLF RB allocation computed through Algorithm 2;
$\boldsymbol{\sigma} \leftarrow$ The RBAM for $\mathbf{x}^*$;
$b_0 \leftarrow$ The row of $\boldsymbol{\sigma}$ with the smallest number of distinct MVNOs;
while $|\mathcal{B}^*| \neq 0$ do
    $b^* \leftarrow$ The row that shares the highest number of linked RBs with $b_0$;
    while $i \leq I_S$ do
        $(\tau_1, \tau_2) \leftarrow$ Selects two columns at random;
        $\boldsymbol{\sigma}^* \leftarrow$ A copy of $\boldsymbol{\sigma}$ with elements $(b_0, \tau_1)$ and $(b_0, \tau_2)$ swapped;
        if number of linked RBs has improved then
            $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}^*$;
        end if
        $i \leftarrow i + 1$;
    end while
    $\mathcal{B}^* \leftarrow \mathcal{B}^* \setminus \{b_0\}$;
end while
```

B.3.5 Fairness Aspects

As shown in (RSEP-QP), we aim at maximizing the number of linked RBs of the system without considering how these RBs are distributed across the different slices. On the one hand, this makes it possible to assign RBs in a way that reduces interference and maximizes inter-slice isolation. On the other hand, different MVNOs can be assigned with different number of linked RBs, thus resulting in unfair linked RBs distribution. Although this problem is out of the scope of this chapter, we believe that the problem is extremely challenging and it is worth of investigation. For this reason, here we discuss two different approaches that might effectively solve the above problem and generate more fair enforcement policies. The simplest approach would be to introduce a new constraint guaranteeing that a minimum number N_{min} of linked RBs is allocated to each MVNO (i.e., $N_m \geq N_{min}$). From (B.4), this approach would result in a quadratic constraint that might make the problem unfeasible if N_{min} is too large. Another approach, would be to adapt the objective function of the RSEP problem via a α-fairness utility. If compared to the previous approach, this formulation would avoid unfeasibility of the problem, but would introduce a strong non-linearity in the objective function that would eventually result in higher computational complexity.

B.4 Speeding-up RSEP-QP and RSEP-EQ

Although Problems RSEP-QP and RSEP-EQ have exponential complexity, two intuitions help reduce their complexity by leveraging specific structural properties of the RSEP.

B.4.1 Sparsity

Let $\mathbf{x}^{\mathrm{OPT}}$ be an optimal solution to either Problem RSEP-QP or RSEP-EQ. If $L_{m,b} = 0$ for a given MVNO m on BS b, then $x^{\mathrm{OPT}}_{m,b,n,t} = 0$ for all n and t. Furthermore, we notice that the complexity of many optimization problems strongly depends on the number of non-zero entries (i.e., the sparsity) of the $\mathbf{Q}$ matrix [513]. Thus, we reduce the complexity of the two problems by inducing sparsity through two transformations. Let m' and b' such that $L_{m',b'} = 0$, for both RSEP-QP and RSEP-EQ we generate a reduced matrix $\tilde{\mathbf{Q}}$ where we set $Q_{m',b',n,t} = 0$ for all $(n,t) \in \mathcal{R}$. For RSEP-QP, it suffices to replace the $\mathbf{Q}$ matrix with $\tilde{\mathbf{Q}}$. To keep equivalence between RSEP-QP and RSEP-EQ, the objective function of RSEP-EQ is rewritten as

$$\frac{1}{2}\mathbf{x}^\top(\tilde{\mathbf{Q}} + 2\lambda\tilde{\mathbf{I}}_V)\mathbf{x} - \lambda \tag{B.6}$$

where $\tilde{\mathbf{I}}_V$ is the identity matrix where we set to zero those entries corresponding to the 2-tuple (m', b').

Note that the two above transformations generate equivalent problems to RSEP-QP and RSEP-EQ and do not impact the optimality of the computed solutions. In fact, Constraint (C1) requires $\sum_{t=1}^{T}\sum_{n=1}^{N_{RB}} x_{m',b',n,t} = 0$ when $L_{m',b'} = 0$. Since $x_{m',b',n,t} \in \{0,1\}$, we have that $x_{m',b',n,t} = 0$ for all n and t associated to the 2-tuple (m', b'). That is, at the optimal solution, $x_{m',b',n,t} = 0$ independently of the value of $q_{m',b',n,t}$.

B.4.2 RB Aggregation

Let $K = \mathrm{GCD}(\mathbf{L})$ be the greatest common divisor (GCD) among all of the elements in the $\mathbf{L}$ matrix. We show that Problems RSEP-QP and RSEP-EQ are equivalent to solve the same problems with a scaled RB grid, when given conditions on K, T and N_{RB} are satisfied. Specifically, if $K > 1$ and either the number N_{RB} of RBs or the number T of time slots are proportional to K, the available resources can be aggregated in groups of K RBs, and each of such groups can be seen as a single aggreagated RB. We refer to such a property as *aggregability* of the RSEP, whose definition is as follows, and its relevance to our problem will be shown in Theorem 2 (proof reported in [524, Section VI-B]).

Definition 3 (Aggregable RSEP)**.** The RSEP is said to be *aggregable* if $N_{RB} \pmod K = 0$ or $T \pmod K = 0$, where $K = \mathrm{GCD}(\mathbf{L}) > 1$ and $A \pmod B$ is the A modulo B operator.

In the first case, we scale the number of RBs as $\tilde{N}_{RB} = N_{RB}/K$. In the second case, we scale the number of time slots as $\tilde{T} = T/K$. That is, for each BS $b \in \mathcal{B}$, the set $\mathcal{R}_b$ of available RBs at b is replaced with an aggregated version of cardinality $|\tilde{\mathcal{R}}_b| = N_{RB}T/K$ where K RBs are grouped together to create a single RB. We refer to this low-dimensional RSEP as the *aggregated RSEP*.

Theorem 2. *Let the RSEP be aggregable, it is possible to compute an optimal solution to the RSEP by solving the aggregated RSEP.*

B.5 Numerical Analysis

We now assess the performance of the algorithms proposed in Section B.3. To this end, we simulate an LTE FDD system with 1.4 MHz channel bandwidth, which is divided into 72 subcarriers organized

into 6 PRBs. Each PRBs consists of 12 subcarriers and 7 symbols. Time is divided into discrete time slots called *sub-frames*. Each sub-frame is formed of two PRBs, lasts 1 ms, and is the minimum scheduling unit in LTE. Groups of $N_{SF} = 10$ sub-frames constitute a *frame*.

In our analysis, each RB corresponds to one sub-frame, therefore we consider a total of $N_{RB} = 6$ RBs per time slot. Let $N_F \in \mathbb{N}$ be the number of frames within the slicing enforcing window. It follows that $T = N_F \cdot N_{SF}$. Unless stated otherwise, we assume that both the interference matrix $\mathbf{Y}$ and the slicing profile matrix $\mathbf{L} = (L_{m,b})_{m \in \mathcal{M}, b \in \mathcal{B}}$ defined in Section B.1 are generated at random at each simulation run.

In order to evaluate the benefits of the proposed approach, in the following of this section we compare our algorithms with slice-unaware schemes that do not leverage information on network topology and interference to instantiate RAN slices. We refer to this method as the *w/o isolation* case case where RBs are assigned to requesting MVNOs in a round-robin fashion with complexity $\mathcal{O}(1)$.

The simulator is implemented in MATLAB and is interfaced with IBM CPLEX optimization toolbox. Specifically, CPLEX is used to solve RSEP-QP and RSEP-EQ, while the two heuristics RSEP-MLF and RSEP-IMLF have been implemented in MATLAB only. Results were averaged over 1000 independent simulation runs.

B.5.1 Convergence Time Analysis

Figure B.7 shows the convergence time of the four algorithms presented in Section B.3 as a function of the number M of MVNOs when $N_F = 2$. As expected, the algorithm with the slowest convergence

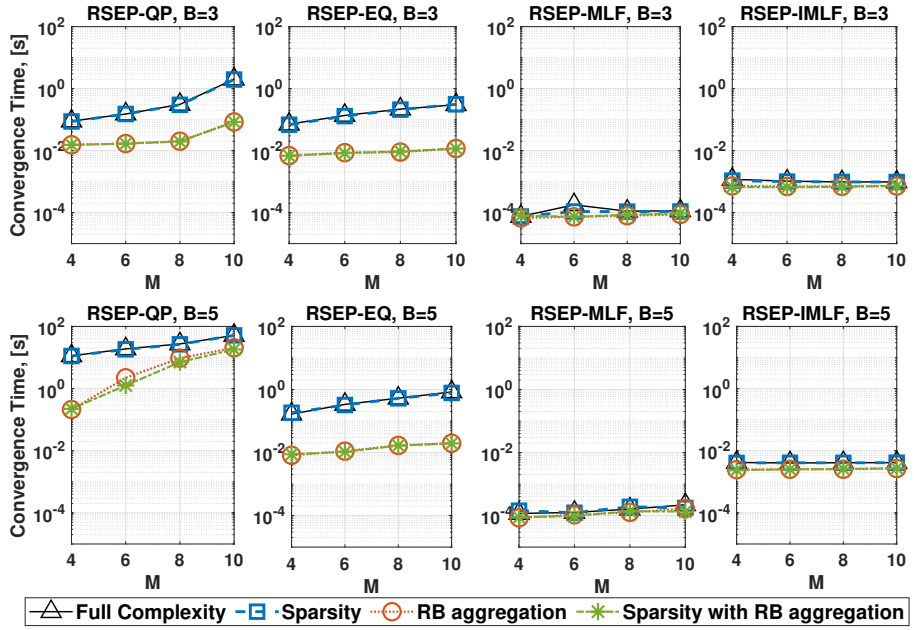


Figure B.7: Convergence time (in seconds) of the three proposed solutions as a function of M considering different computational time reduction techniques.

time is the optimal algorithm RSEP-QP, while the fastest algorithm is RSEP-MLF. Interestingly

enough the convergence time of both RSEP-QP and RSEP-EQ increases as the number of MVNOs in the network grows, a behavior that is not exhibited by the two heuristic algorithms RSEP-MLF and RSEP-IMLF whose convergence time only slightly increases as a function of M.

Figure B.7 also shows the impact of the sparsity and RB aggregation mechanisms in Sections B.4.1 and B.4.2 on the overall convergence time. It can be observed that the techniques presented in Section B.4 can effectively reduce the computation time of all the four algorithms. Moreover, we show that RB aggregation is the technique that produces the best performance improvement in terms of convergence time.

We point out that RSEP-QP requires approximately $100s$ *to compute an optimal solution when* $M = 10$ *and* $B = 5$ *and RSEP-EQ only requires* $1s$*. On the contrary, RSEP-IMLF computes a solution within few milliseconds, while RSEP-MLF computes the solution in less than a millisecond.*

It is worth to point out that Figure B.7 reveals how the reduction in terms of convergence time brought by sparsity can not be appreciated in small-scale scenarios. For this reason, we have further investigated the impact of sparsity in large-scale networks and the obtained results are presented in Figure B.8. Our results show that sparsity can effectively reduce the computation time by several tens of seconds, and the gain increases as both M and B increase. From Figure B.8 we can conclude that sparsity is a complexity reduction technique that best performs in large scale network deployments.

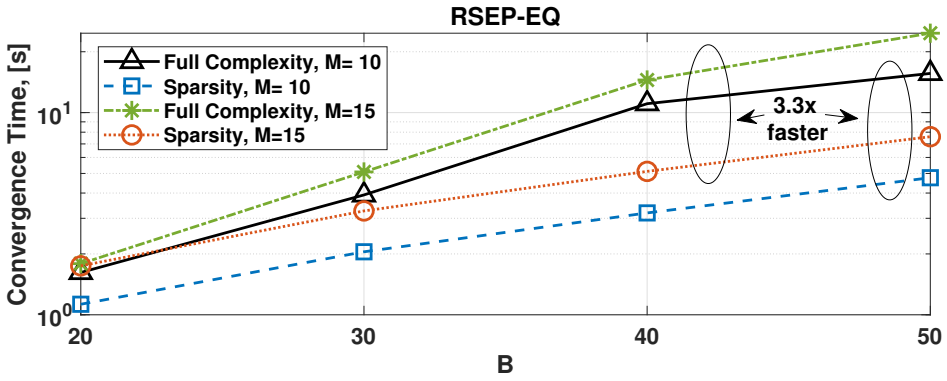


Figure B.8: Convergence time (in seconds) of RSEP-EQ as a function of B considering different number M of MVNOs.

B.5.2 Optimality-gap Analysis

Another crucial aspect is the optimality-gap between the optimal solution computed by RSEP-QP and those computed through RSEP-EQ/RSEP-MLF algorithms. Although Theorem 1 shows that (under some conditions) Problem RSEP-EQ is equivalent to Problem RSEP-QP, we can not guarantee that the solution computed by RSEP-EQ is a global optimum. Indeed, the solver might get stuck in one of the local maximizers, thus effectively preventing the computation of an actual global maximizer.

For this reason, in Figure B.9 we investigate the Optimality-gap of RSEP-EQ, RSEP-MLF and RSEP-IMLF with respect to an optimal solution computed by RSEP-QP. This performance metric is defined as one minus the ratio between the utility function achieved by any of the aforementioned

approximation and heuristic algorithms and that achieved by RSEP-QP. The closer to zero is the gap, the closer to optimality is the solution computed by approximation and heuristic algorithms.

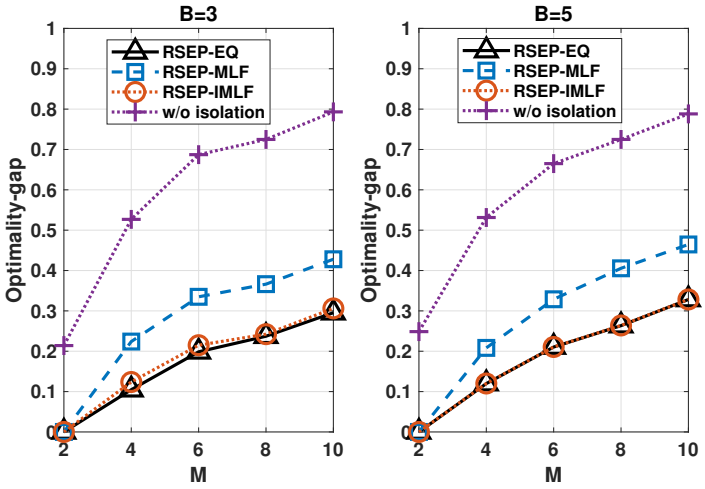


Figure B.9: Optimality-gap of RSEP-EQ, RSEP-MLF and RSEP-IMLF as a function of M considering different number B of BSs.

Figure B.9 shows that the Optimality-gap increases as the number of MVNOs and BSs in the network increases. Intuitively, this is because the feasibility set increases as M and/or B increase. Given that local maximizers of RSEP-EQ lie on the vertices of the feasibility set, greater values of M and B produce a greater number of local maximizers, thus the probability of getting stuck in a local maximizer increases as well. Notice that although RSEP-MLF is negligibly affected by the number of BSs B, it achieves poor performance if compared to RSEP-EQ. It is worth to mention that RSEP-IMLF is perhaps the most efficient algorithm which effectively trades-off between optimality and computational complexity. Indeed, Figs. B.9 and B.7 show that RSEP-IMLF can compute RAN slice enforcement strategies that achieve the same performance as RSEP-EQ in the order of few milliseconds. Furthermore, the w/o isolation case employs a round-robin scheme that, although being fast, results in very high gap.

B.5.3 Linked RBs and SINR Analysis

Figure B.10 shows the impact of M on the percentage of linked RBsRBs of the system when $N_F = 10$, $B = 5$ and $T = 100$. As expected, RSEP-EQ and RSEP-IMLF always perform better than RSEP-MLF in terms of number of linked RBs. Moreover, Figure B.10 illustrates that the number of linked RBs decreases as the number M of MVNOs increases. This is because, when more MVNOs include the same BS to their slices, it is harder to guarantee that all MVNOs will receive the corresponding amount of RBs jointly with a large number of linked RBs.

As demonstrated in Figure B.10, and if compared to the traditional approach where inter-slice isolation is not enforced, our approach increases the percentage of RBs that can be used to perform coordination-based transmissions. A major question, however, is whether or not the enforcement strategies presented in this chapter can actually bring performance gains in terms of throughput

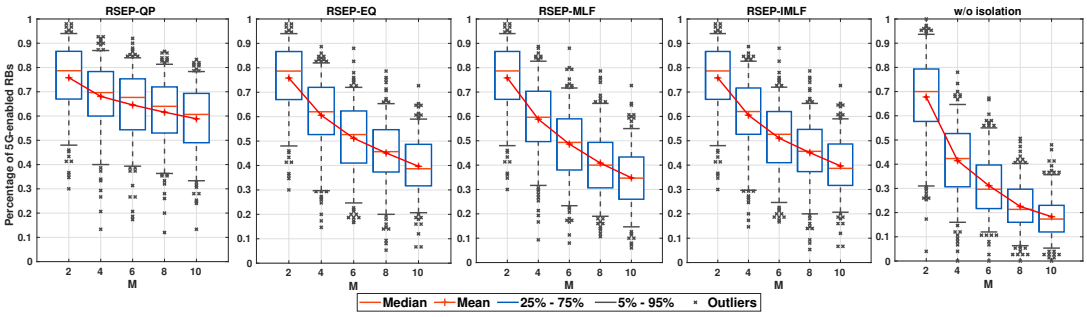


Figure B.10: Percentage of linked RBs as a function of M and different RAN slicing enforcement policies ($B = 5$).

and interference mitigation when applied to real-world 5G networks. To answer such an interesting question, at each simulation run we have generated a random network topology with B BSs and M independent user sets each consisting of 10 cellular users randomly deployed within the simulated area. In other words, we assume that each MVNO requests a RAN slice to serve 10 cellular users. Channel gain coefficients between BSs and cellular users are computed through the well-established free-space path loss model.

Recall that the slicing profile $\mathbf{L}$ is randomly generated at each simulation run. For each $\mathbf{L}$ we allocate RBs to MVNOs by running different RAN slicing enforcement algorithms. Then, for each MVNO we compute the optimal downlink transmission policy that maximizes the rate of the system [514] by determining (i) which user should be scheduled in each RB, (ii) how much power to allocate to each transmission, and (iii) whether or not a user should be served by multiple neighbouring BSs through CoMP transmissions.

Our results are reported in Figure B.11, where we show the average SINR for different RAN slicing enforcement algorithms as a function of the number of MVNOs and BSs. In general,

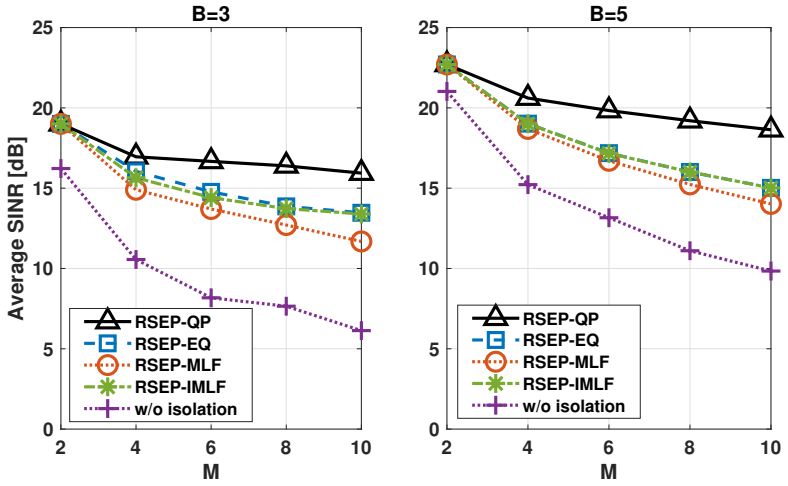


Figure B.11: Average SINR achieved by the proposed algorithms as a function of M considering different number B of BSs.

Figure B.11 shows that our approach always improves the SINR of cellular users by providing gains up to 2 times and an effective SINR gain up to 10 dB. Despite the optimality ratio decreases when large values of B and M are considered, Figure B.11 shows that cellular users still experience higher SINR values if compared to traditional RAN slicing algorithms where isolation across slices is not enforced. This results show the effectiveness of our approach even in the case of sub-optimal RAN slice enforcement policies.

B.5.4 Bandwidth and Time-scale Analysis

We now investigate the impact of different bandwidth configurations and time-scale requirements. Specifically, we consider the case of a resource grid with a 20 MHz bandwidth, $N_{RB} = 100$ RBs and $T = 5$s. In this configuration, we let $M = 5$ MVNOs change the slicing profile $\mathbf{L}$ at a slower frequency if compared to the case considered in previous sections. Since we have already demonstrated that complexity reduction techniques presented in Section B.4 effectively reduce computation times, we will present results obtained by applying both sparsity and RB aggregation techniques to our approach. Also, given that the considered scenario presents a very high number of RBs, we investigate the impact of slice requests profiles on the complexity of the problem. Specifically, we consider 5 different cases where MVNOs are allowed to submit requests where the number of requested RBs must be a multiple of $\xi \in \{2, 5, 10, 20, 50\}$, which is equivalent to submitting requests with an instantaneous minimum bandwidth equal to $0.4, 1, 2, 4, 10$ MHz for each subframe. We refer to ξ as the minimum RB request block size. Indeed, the minimum request requirement does not hold if the MVNO is not willing to request any RB on a particular BS.

In Figure B.12, we show the convergence time and optimality-gap of our algorithms for different values of ξ. As expected, large values of ξ facilitate RB aggregation (see Section B.4.2) and

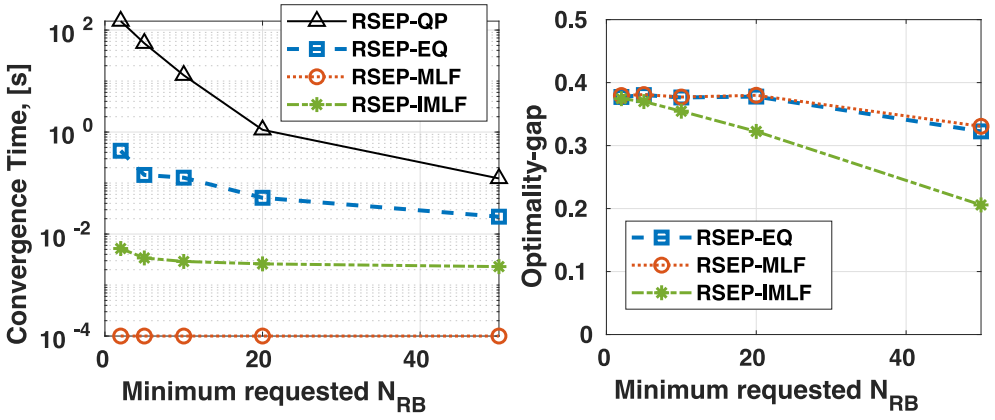


Figure B.12: Convergence time (s) and optimality-gap of our algorithms as a function of the minimum RB request block size ξ.

considerably reduce the computation time of all solutions, especially for RSEP-QP which experiences a reduction in the computation time by a factor $1000\times$ when moving from $\xi = 2$ to $\xi = 50$. Similar considerations apply to the optimality-gap as well which decreases as the value of ξ decreases as

well. These results show that aggregation of RBs not only reduces the complexity of the problem, but it also results in more efficient solutions.

B.6 Experimental Evaluation

The objective of this section is to experimentally demonstrate that the benefits of our approach are not restricted to simulation scenarios only, but they also apply to real cellular network deployments. For this reason, in Sections B.6.1 and B.6.2 we first describe the testbed and the scenario considered in our experiments. Then, we discuss the obtained results in Section B.6.3.

B.6.1 Experimental Setup

To demonstrate the superior performance of our algorithms, we have instantiated a standard-compliant LTE cellular network on the Arena testbed (see Section 3.5). To deploy a standard-compliant LTE network, we leveraged the srsRAN open-source software which offer LTE-compliant eNB and UEs protocol stack implementations, as well as an Evolved Packet Core application (see Sections 2.2.2 and 2.3). In this section we discuss implementation details and report results obtained through our srsRAN-based prototype. However, we would like to remark that our solutions can be seamlessly ported with minimal changes to other open-source software platforms such as OpenAirInterface (OAI) (see Section 2.2.1).

We deployed two standard-compliant eNBs on Arena USRPs X310 serving 6 COTS UEs (Xiaomi Redmi Go). The deployed LTE network is shown in Figure B.13, where UE_{xy} is served by eNB_x, with $x \in \{1, 2\}, y \in \{1, 2, 3\}$.

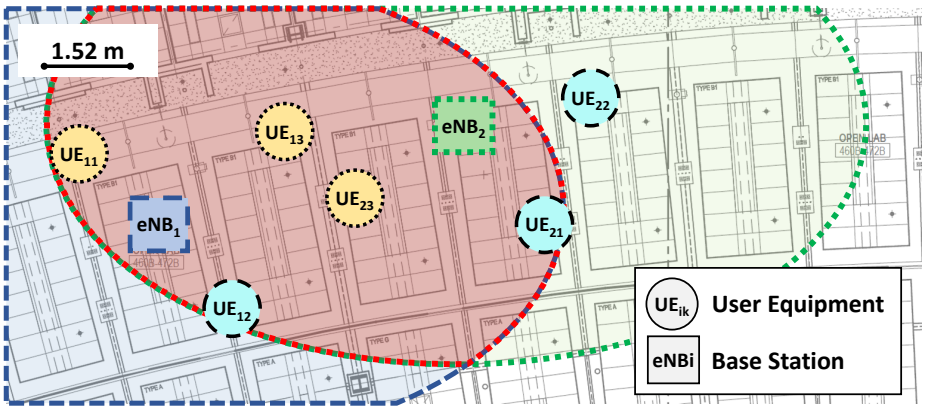


Figure B.13: Experimental Setup on the Arena testbed.

We deploy the LTE network in FDD mode in the LTE Band 7 with a bandwidth equal to 10MHz and 50 RBs. As shown in Figure B.13, we consider 2 MVNOs serving UE_{11}, UE_{13}, UE_{23} (slice 1) and UE_{12}, UE_{21}, UE_{22} (slice 2), respectively. The two eNBs are deployed approximately 6 meters apart and their coverage areas partially overlap. For example, UE_{13} and UE_{23}—which are both associated to slice 1—experience severe interference due to the proximity to adjacent eNBs.

This setup is particularly well-suited to showcase the performance gains brought by interference reduction of our approach.

B.6.2 SCOPE-based Prototype

As srsRAN does not directly provide RAN slicing functionalities, we interfaced our algorithms with the SCOPE framework of Section 3.2. In each experiment, the IP receives RAN slice requests generated by a set of MVNOs. Each RAN slice request specifies which eNBs should be included in the slice, and the number of RBs that should be assigned to the slice on each eNB. Upon reception of these requests, the IP executes one of the RAN slicing enforcement algorithms proposed in this chapter to assign the available RBs to the requesting MVNOs such that the number of linked RBs is maximized. The solution of the algorithm is then converted into a set of B configuration files (i.e., the `config.txt` files). Each configuration file is associated to individual eNBs and specifies which RBs should be assigned to each slice. These files are then dispatched to the corresponding eNB and processed by a *config file parser module*. RAN slicing enforcement information is then fed to a *RAN slicing module* that instantiates RAN slices and exclusively assigns RBs to each of them according to the configuration specified in the `config.txt` file. Finally, RBs assigned to each RAN slice are used by individual MVNOs to schedule UEs downlink transmissions performed by Arena USRPs. This is achieved by assigning each UEs to one slice only. This association is implemented by assigning a unique identifier (i.e., an integer number) to each slice and associating the unique international mobile subscriber identity (IMSI) of each UEs to a slice identifier. This way, the *UEs scheduler* module can schedule UEs belonging to a specific slice on RBs that have been assigned to that slice only.

B.6.3 Experimental Results

We consider the case where two MVNOs lease eNB resources (i.e., RBs) to instantiate RAN slices. Our experiments aim at evaluating two critical performance parameters, i.e., network throughput and SINR experienced by UEs. To showcase the effectiveness of our algorithms, we compared the optimal RSEP-QP method presented in Section B.3.1, with the traditional one (i.e., *w/o isolation*) in which RAN slices are instantiated without leveraging network topology information and without enforcing slice isolation. We ran 10 experiments on the testbed presented in Section B.6.1. At each experiment run we generate a random slicing profile $\mathbf{L}$ in MATLAB. To compensate for inaccurate synchronization of our experimental equipment, slicing profile $\mathbf{L}$ are generated with $K = 9$, which we have experienced to be a large enough value to ensure that most RBs are synchronized across adjacent BSs. Mobile users perform a 2-minute long speed-test (which ensures that transmission buffers are constantly backlogged with downlink packets and slices are always active) and report both throughput and SINR measurements. To provide a fair comparison between different approaches, for each $\mathbf{L}$ we compute RAN slicing enforcement policies by using the method presented in this chapter (i.e., RSEP-QP) and traditional ones (i.e., *w/o isolation*). Also, to avoid time-varying performance degradation introduced by Internet connectivity, which would result in an unfair comparison between the two methods, the speed-test server is locally hosted on the Arena testbed.

The average network throughput over the 10 experiments is reported in Figure B.14. Our results clearly show that our approach (i.e., RSEP-QP) outperforms traditional interference-agnostic

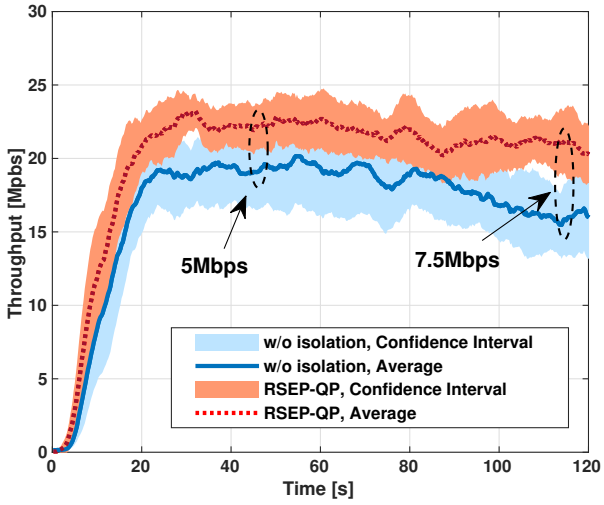
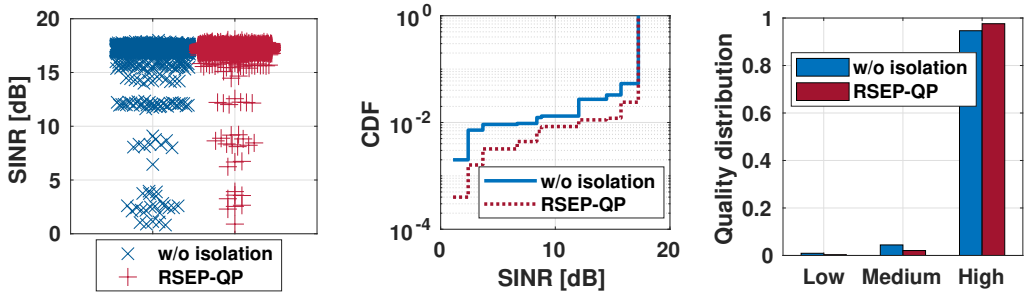


Figure B.14: Experimental throughput comparison.

approaches and increases throughput by approximately 27% (approximately 5Mbps gain) with peak throughput gains up to 7.5Mbps.

In Figure B.15 we analyze SINR measurements reported by UEs under the two considered methods (Figure B.15a). The Cumulative Distribution Function (CDF) of the SINR shown in



(a) SINR measurements (b) CDF (c) SINR Quality

Figure B.15: Experimental SINR analysis.

Figure B.15b clearly demonstrates that traditional approaches are subject to poor SINR performance due to high interference across heterogeneous RAN slices. Our approach, instead, effectively reduces such interference and improves the SINR experienced by UEs. This can be easily noticed in Figure B.15c where we show the ratio of users reporting Low ($\text{SINR} \leq 5\text{dB}$), Medium ($5\text{dB} < \text{SINR} \leq 17\text{dB}$) and High ($\text{SINR} > 17\text{dB}$) SINR. Figure B.15c shows that our approach results in a larger portion of UEs reporting higher SINR if compared to traditional approach which, instead, shows higher percentage of UEs reporting low and medium SINR levels.

B.7 Related Work

The problem of determining how many resources each RAN slice should receive, also known as RAN slicing [4,501,505], has received significant interest from the research community over the last years; for excellent surveys on recent work on the topic the reader may refer to [374, 487]. Theoretical tools, ranging from optimization [467,506–508,515], auctions [474], game theory [62,509,510] and artificial intelligence [307,516] have been proposed. However, such work does not address how to actually deploy RAN slices on top of the underlying physical network.

This key aspect has stimulated the research community to research the enforcement of RAN slicing policies. Prior work [4,63,505,517,518] virtualizes the available resources to create "pools" that are then shared and allocated among the MVNOs. This approach, however, may be ineffective in scenarios where *fine-grained control of physical-layer resources is required*, for example, to enable IBSPC, CoMP and beamforming.

Recent work has focused on addressing the RAN slicing enforcement problem from a resource allocation perspective. In [466], Mancuso *et al.* present a stochastic model to predict the impact of different enforcement policies on the overall performance of a sliced cell. Chang *et al.* [502] propose a partitioning algorithm that allocates the available RBs to each requesting MVNO by simultaneously maximizing the percentage of satisfied MVNOs while allocating the minimum amount of RBs. Similarly, Han *et al.* [519] consider genetic algorithms to assign the available RBs to the MVNOs such that a long-term utility is maximized. However, [466, 502, 519] analyze the problem considering a network with a single BS, and thus cannot be applied in multi-cell networks where MVNOs request different amounts of resources on different BSs. The authors in [503, 520] identify fine-grained RB management as a promising approach to guarantee orthogonality and reduce inter-MVNO interference, thus deploying highly-efficient 5G networks. However, [503] does not provide any algorithm to enforce slicing policies to maximize network efficiency, while [520] does not consider interference among BSs when allocating RBs.

B.8 Conclusions

In this chapter, we have investigated the challenging and timely problem of RAN slicing enforcement in 5G networks. First, we have formulated the resource slicing enforcement problem (RSEP) and shown its NP-hardness. Then, we have proposed three approximation and heuristic algorithms that render the problem tractable and scalable as the problem increases in complexity. Finally, we have evaluated the algorithms through simulations, and demonstrated their effectiveness through experimental analysis on a testbed composed by 2 LTE base stations and 6 cellular users. Results conclude that our algorithms are scalable and provide near-optimal performance. Moreover, our solutions effectively enforce RAN slicing policies by satisfying MVNOs requirements, reducing inter-MVNO interference, and providing throughput and SINR gains up to 27% and 100%, respectively.

Acknowledgments

The journey leading to the completion of my Ph.D. dissertation work has been quite long and exciting. I will not hide it has also been frustrating and exhausting at times—but it was surely worth pursuing. I believe to have grown quite a bit since the day I joined the Wireless Networks and Embedded Systems Lab—WiNES Lab, for friends—at the time housed at "140 The Fenway," and now part of the Institute for the Wireless Internet of Things at Northeastern University. Here, I would like to spend a few words to thank the people that have been part of this journey.

First of all, this would not have been possible without my advisors, Stefano Basagni and Tommaso Melodia—mentioned in no particular order—who entrusted me with increasing responsibility and always pushed me to aim for more. Their vision, support, and the autonomy they granted me in my own research have been key in the development of this work. I am also grateful to Kaushik Chowdhury, part of my Ph.D. Committee, who provided me with precious suggestions. Very special thanks also go to Salvatore D'Oro, who patiently—maybe too patiently at times—guided me every step of the way, and to Michele Polese, who matched Salvatore's guidance in the last couple of years. I have been extremely fortunate to work with—and learn from—them. Thank you also to Francesco Restuccia, Pedram Johari, Emrecan Demirors, Zhangyu Guan, and Cristian Cassella for the interesting conversations, advice, or words of encouragement they provided me throughout the years.

Thank you to all my other co-authors from Northeastern University, who helped me in my endeavors, entrusted me with their projects, or provided me with useful insights: Lorenzo Bertizzolo, Ludovico Ferranti, Miead-Tehrani Moayyed, Amani Al-shawabka, Davide Villa, Hai Cheng, Subhramoy Mohanti, Chinenye Tassie, Shweta Shrivastava, and Abhimanyu Gosain. And thank you to my co-authors outside Northeastern: Bernardo Casasole and Antonio Capone from the Polytechnic University of Milan; Francesco D'Alterio and Francesca Cuomo from Sapienza University of Rome; Scott Pudlewski from AFRL, and Michele Zorzi from University of Padova. I would also like to thank all my colleagues from WiNES Lab (or just visiting) that I have not mentioned yet, and that made this experience definitely more enjoyable: Raffaele Guida, Sara Falleni, Andrea Lacava, Jennie Rodowicz, Luca Baldesi, Daniel Uvaydov, Deniz Unal, Neagin Neasamoni Santhi, Kerem Enhos, Clifton Robinson, Maria Tsampazi, Matteo Bordin, Neil Dave, John Buczek, Nan Cen, Liyang Zhang, Jiacheng Shi, Paolo Testolina, Eugenio Moro, Mattia Lecci, Pietro Brach del Prever, Alessandro Talamonti, Anna Sofia Argiró, Marco Casarotto, Claudia Greco, Lorenzo Ghiro, Dario Bega, Claudio Fiandrino, Luca Antonelli, and Aditya Sur. A big thank you also goes to the Colosseum team for all the help—and shared sufferings—in maintaining the world's largest wireless network emulator (because yes, I even found myself involved in the management of an experimental wireless platform!): Davide Villa, Andrea Lacava, Michele Polese, Pedram Johari, Salvatore D'Oro, Francesco Restuccia,

Alberto Munio Gracia, Mattia Lecci, Miead-Tehrani Moayyed, Eric Anderson, Kurt Yoder, Michael Seltser, Paresh Patel, Ventz Petkov, and Ajeet Bagga. Thank you also to the administrative staff of the Institute who took care of the many organizational tasks over the years, allowing me to focus on my research work: Fiona Morgan, Neko Smith, William Hart, Rosy Trovato, Graham Waters, Spencer Vu, and Reena Cash. Finally, I would like to thank my M.S. (and B.S.) advisor from University of Padova, Michele Rossi, for putting me in touch with Stefano and Tommaso for a certain Ph.D. opportunity.

On the personal side, I would like to thank the few friends that I made along the years. Some old friends from home: Luca, Andrea and Valentina (thank you both for really making an effort to keep in touch with me!), Fabio, Francesco (and Beatrice), Giacomo, Matteo (and Elisa), Rebecca, Camilla, Joy, Riccardo, Marianna, Leonardo, Paolo, Lorenzo, Attilio, Luca, and Mariano. And—of course—some friends from more recent years: Sara, Pasquale, Salvatore, Elisa (thank you for baking me cookies that time!), Fiona, Stella, Jennie, Michele, Giulia, Pedram (also for enduring my bad jokes), Ludovico, Raffaele, Lorenzo, Emrecan, Luca, Davide, Andrea, Zamora, Paolo, Eugenio, Pietro, Francesco (and Lucia), Enrico, Monika, Bernardo, Deniz, Daniel, Neagin, Matteo, Luca, Pietro, Giuseppe, Flavius, Bernard, Gabriel, Farah, Sila, Antea, Michele, Nicholas, Miead, and Volkan. On the family side, I would like to thank all my aunts, uncles, and cousins, among which are Alberto, Paola, Valentina, Saverio, Marina, Giacomo (and Sara), Monica, Francesco, Tommaso, Matilde, and Luca.

Last but surely not least, very special thanks go to my parents, Sonia and Andrea, who love me very much and have always supported me, and to my grandmother, Bruna, for the very same reasons. I know the path I chose to pursue so many years ago was not easy for you either.