

DESIGN, IMPLEMENTATION AND EXPERIMENTAL EVALUATION OF
A NETWORK-SLICING AWARE MOBILE PROTOCOL STACK

by

GINÉS GARCÍA AVILÉS

A dissertation submitted by in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in

Telematic Engineering

Universidad Carlos III de Madrid

Advisor: Pablo Serrano Yáñez-Mingot
Co-Advisor: Marco Gramaglia

June 2021

*Design, Implementation and Experimental Evaluation of a Network-Slicing aware
Mobile Protocol Stack*

Prepared by:

Ginés García Avilés, Universidad Carlos III de Madrid

contact: gines.garcia2@gmail.com

Under the advice of:

Pablo Serrano Yáñez-Mingot, University Carlos III of Madrid

Marco Gramaglia, University Carlos III of Madrid

Telematic Engineering Department, Universidad Carlos III de Madrid

This work has been supported by:



Unless otherwise indicated, the content of this thesis is distributed under a Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA).

“Talk is cheap. Show me the code.”

Linus Torvalds

“Hmm...it was working for sure”

Software Developers

“I did not change anything”

Software Developers

Acknowledgements

The typical phrases are those that always state what you need in a way that the majority of people will understand it, so here is mine: "I was never a brilliant student, but with effort and dedication I was able to achieve what I set out to do." As I said, a mere cliché. The truth is that this process has not been easy, not difficult, funny, but not always. What I can say is that I am not the same person who started a PhD., not only because of I am four years older but for the intensity I was living each one of them.

This story starts with an internship at IMDEA Network Institute at the Software department, whose main head was Rebeca De Miguel. There, together with Diego Lucero, Diego Sierra and Patricia Callejo I became part of the "Rebecarios" team lead by one of the best software engineers I have ever met, Christian Sanchez. I would like to say thank you to all of them for all the hard work and all the funny moments we shared.

After the internship, I started the PhD program at IMDEA Networks Institute, where the last years have passed away too fast. I would like to start giving all my sincere appreciation to my supervisors Pablo Serrano and Marco Gramaglia because they guided me through this jungle called research to achieve the main goals of the program. These years working with Pablo have been amazing. I have been able to overcome all the different challenges, having always his support. Thanks for your help not only in the pure research field but at personal level to properly overcome the difficulties found during these years. Marco has been fundamental during this PhD. Thanks to his advice, I have been able to learn, evolve and effectively get the best out from every situation. Thanks a lot for everything Marco.

Over these years, I have met a lot of amazing people and I would like to thank all of them the time we have spend together: coffee breaks, football matches at the university dreaming with the idea of winning the tournament or discussing about cryptocurrencies during lunch time: UC3M 1st floor: Patri, Donato, Sergio Gonzalez, Tamurejo, Nuria, Cris, Kiril, Milan, Stefano, Winnie, Pedro; UC3M pro lab: Borja, Victor, Antonio, Jose and Jorge. IMDEA: Joan, Pablo, Moha. Thanks guys, It was an amazing experience to share these years with you.

Moreover, I would like to mention the people who have accompanied me during all my walks, with their unconditional support and always showing a smile regardless of anything else. My parents and my brothers. To my father, for always listening to me and for trying to bring me peace when I need it most. To my mother, for making everything always easier, listening and understanding everything regardless of the situation. To my brother Adrián, for listening to my thousand crazy stories and always remind me what is important when I need it most. To my brother Álvaro, for always having a moment to share with me and for sometimes showing me points of view that I didn't know they exist.

Finally, I would like to thank the person with whom I have shared all this path and I hope to be able to share many more, Noelia, who helped me to believe in myself and, above all, she believed in me before I did. Thanks for your touch, love, kisses and crazy Fridays with "il nonno" pizza.

Published Content

Published Content

This thesis is based on the following published papers:

[1] **Gines Garcia-Aviles**, Carlos Donato, Marco Gramaglia, Pablo Serrano, Albert Banchs. *ACHO: A framework for flexible re-orchestration of virtual network functions*. Computer Networks Journal, volume 180, October 24, 2020.

Link: <https://doi.org/10.1016/j.comnet.2020.107382>

- This work is part of this thesis and its content reported in Sections 6.3 and 7.3.
- The role of the author of the thesis in this work is the design, implementation and experimental evaluation of the framework for flexible orchestration proposed at this work.

[2] Marco Gramaglia, Pablo Serrano, Albert Banchs, **Gines Garcia-Aviles**, Andres Garcia-Saavedra, Ramon Perez. *The case for serverless mobile networking*. The International Federation of Information Processing (IFIP) Networking Conference, June 22-25, 2020.

Link: <https://ieeexplore.ieee.org/abstract/document/9142747>

- This work is part of this thesis and its content reported in Section 4.2.
- The role of the author of the thesis in this work is focused on contributing to the main concepts, as well as contributions in the design of the proposed approach.

[3] **Gines Garcia-Aviles**, Marco Gramaglia, Pablo Serrano, Francesco Gringoli, Sergio Fuente-Pascual, Ignacio Labrador Pavon. *Experimenting with open source tools to deploy a multi-service and multi-slice mobile network*. Computer Communications Journal, volume 150, January 15, 2020.

Link: <https://doi.org/10.1016/j.comcom.2019.11.003>

- This work is part of this thesis and its content reported in Sections 6.2 and 7.2.
- The role of the author of the thesis in this work is focused on the design, implementation and experimentation with regarding of the concepts proposed in the

paper excluding the local breakout, designed and implemented by one of the authors of the paper.

[4] **Gines Garcia-Aviles**, Marco Gramaglia, Pablo Serrano, Albert Banchs. *POSENS: A Practical Open Source Solution for End-to-End Network Slicing*. Published at IEEE Wireless Communications, volume 25, October, 2018.

Link: <https://doi.org/10.1109/MWC.2018.1800050>

- This work is part of this thesis and its content reported in Sections 6.1 and 7.1.
- The role of the author of the thesis in this work is focused on the design, implementation and experimentation of an end-to-end network slicing solution based on Open Source software solutions.

Additional Research Merits

This section provides information about additional research works I have (co)authored, being all of them related with this thesis.

[5] **Gines Garcia-Aviles**, Marco Gramaglia, Pablo Serrano, Marc Portoles, Albert Banchs & Fabio Maino. *SEMPER: A Stateless Traffic Engineering Solution for WAN Based on MP-TCP*. IEEE International Conference on Communications, Kansas City (MO, USA) October 20-24, 2018.

Link: <https://doi.org/10.1109/ICC.2018.8422991>

- The role of the author of the thesis in this work is focused on the design, implementation and experimentation of a traffic engineering solution based on Multipath TCP.

[6] Sébastien Henri, **Gines Garcia-Aviles**, Pablo Serrano, Albert Banchs & Patrick Thiran. *Protecting against Website Fingerprinting with Multihoming*. Proceedings on Privacy Enhancing Technologies Journal, volume 2020 issue 2, 01 April, 2020.

Link: <https://doi.org/10.2478/popets-2020-0019>

- The role of the author of the thesis in this work is focused on the implementation and experimentation of a Multipath TCP scheduler whose main concern is to prevent traffic fingerprinting by exploiting path diversity.

[7] Marco Gramaglia, Ignacio Labrador Pavon, Francesco Gringoli, **Gines Garcia-Aviles**, Pablo Serrano. *Design and Validation of a Multi-service 5G Network with QoE-aware Orchestration*. Proceedings of the 12th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization (ACM WINTECH 2018),

2 November 2018, New Delhi, India

Link: <https://doi.org/10.1145/3267204.3267216>

- The role of the author of the thesis in this work is focused on the design, implementation and experimentation with regarding of the concepts proposed in the paper.

[8] Pablo Serrano, Marco Gramaglia, Dario Bega, David Gutierrez-Estevez, **Gines Garcia-Aviles** & Albert Banchs. *The path toward a cloud-aware mobile network protocol stack*. Transactions on Emerging Telecommunications Technologies Journal, volume 29, 19 April, 2018.

Link: <https://doi.org/10.1002/ett.3312>

- The role of the author of the thesis in this work is focused on contributing to the main concepts proposed in the paper.

[9] Winnie Nakimuli, and Giada Landi, and Ramon Perez, and Matteo Pergolesi, and Marc Molla, and Christos Ntogkas, and **Gines Garcia-Aviles**, and Jaime Garcia-Reinoso, and Mauro Femminella, and Pablo Serrano, and others. *Automatic deployment, execution and analysis of 5G experiments using the 5G EVE platform*. IEEE 3rd 5G World Forum (5GWF), pages 372-377, 12 September, 2020.

Link: <https://doi.org/10.1109/5GWF49715.2020.9221060>

- The role of the author of the thesis in this work is focused on the design, implementation and evaluation of different components of the proposed experimentation platform.
- [-] **Gines Garcia-Aviles**, et all *Cloud-native Radio Access Networks*.
- The role of the author of the thesis in this work is focused on the design, implementation and experimentation of a cloud-native RAN.

Abstract

With the arrival of new generation mobile networks, we currently observe a paradigm shift, where monolithic network functions running on dedicated hardware are now implemented as software pieces that can be virtualized on general purpose hardware platforms. This paradigm shift stands on the softwarization of network functions and the adoption of virtualization techniques. Network Function Virtualization (NFV) comprises softwarization of network elements and virtualization of these components. It brings multiple advantages: *(i)* Flexibility, allowing an easy management of the virtual network functions (VNFs) (deploy, start, stop or update); *(ii)* efficiency, resources can be adequately consumed due to the increased flexibility of the network infrastructure; and *(iii)* reduced costs, due to the ability of sharing hardware resources. To this end, multiple challenges must be addressed to effectively leverage of all these benefits.

Network Function Virtualization envisioned the concept of *virtual network*, resulting in a key enabler of 5G networks flexibility, Network Slicing. This new paradigm represents a new way to operate mobile networks where the underlying infrastructure is "sliced" into logically separated networks that can be customized to the specific needs of the tenant. This approach also enables the ability of instantiate VNFs at different locations of the infrastructure, choosing their optimal placement based on parameters such as the requirements of the service traversing the slice or the available resources. This decision process is called orchestration and involves all the VNFs withing the same network slice. The orchestrator is the entity in charge of managing network slices. Hands-on experiments on network slicing are essential to understand its benefits and limits, and to validate the design and deployment choices. While some network slicing prototypes have been built for Radio Access Networks (RANs), leveraging on the wide availability of radio hardware and open-source software, there is no currently open-source suite for end-to-end network slicing available to the research community. Similarly, orchestration mechanisms must be evaluated as well to properly validate theoretical solutions addressing diverse aspects such as resource assignment or service composition.

This thesis contributes on the study of the mobile networks evolution regarding its softwarization and cloudification. We identify software patterns for network function virtualization, including the definition of a novel mobile architecture that squeezes the

virtualization architecture by splitting functionality in atomic functions.

Then, we effectively design, implement and evaluate of an open-source network slicing implementation. Our results show a per-slice customization without paying the price in terms of performance, also providing a slicing implementation to the research community. Moreover, we propose a framework to flexibly re-orchestrate a virtualized network, allowing on-the-fly re-orchestration without disrupting ongoing services. This framework can greatly improve performance under changing conditions. We evaluate the resulting performance in a realistic network slicing setup, showing the feasibility and advantages of flexible re-orchestration.

Lastly and following the required re-design of network functions envisioned during the study of the evolution of mobile networks, we present a novel pipeline architecture specifically engineered for 4G/5G Physical Layers virtualized over clouds. The proposed design follows two objectives, resiliency upon unpredictable computing and parallelization to increase efficiency in multi-core clouds. To this end, we employ techniques such as tight deadline control, jitter-absorbing buffers, predictive Hybrid Automatic Repeat Request, and congestion control. Our experimental results show that our cloud-native approach attains $> 95\%$ of the theoretical spectrum efficiency in hostile environments where state-of-the-art architectures collapse.

Table of Contents

Acknowledgements	VII
Published Content	IX
Abstract	XIII
Table of Contents	XV
List of Tables	XIX
List of Figures	XXI
List of Acronyms	XXV
I Introduction, motivation and challenges	1
1. Introduction	3
1.1. Motivation	3
1.2. Challenges and Contributions	4
1.3. Thesis overview	5
2. Background	7
2.1. Mobile Networks	7
2.1.1. The 4 th Generation of mobile networks	8
2.1.2. The 5 th Generation of mobile networks	9
2.2. Network slicing	11
2.2.1. Properties and challenges	11
2.3. Virtualization of Network functions	13
2.3.1. Properties and challenges	13
2.4. Software Defined Networking	14

3. Open Source Virtual Network Functions	17
3.1. Radio Access Networks	17
3.2. Core Network	18
3.3. Management and Orchestration	18
3.4. Network Slicing	19
4. Software patterns for NFV	21
4.1. Towards a cloud-aware mobile network protocol	23
4.1.1. The quest for cloudification	24
4.1.2. Re-designing VNFs internals	26
4.1.3. The need for performance indicators	28
4.2. The case for Serverless mobile networking	29
4.2.1. Serverless Mobile Architectures	32
4.2.2. Challenges to Address	35
4.3. Flexible re-orchestration of VNFs	39
4.3.1. Advantages of a fine-grained re-orchestration	41
4.3.2. State of the art solutions	41
5. Summary Part I	45
II Bringing Network Slicing to softwarized mobile networks	47
6. Design and Implementation	49
6.1. POSENS, a Practical Open Source Solution for end-to-end Network Slicing	49
6.1.1. Design of POSENS	52
6.2. Experimenting with open source tools to deploy a multi-service and multi-slice mobile network	55
6.2.1. Novel services considered	55
6.2.2. Access Network	57
6.2.3. Local breakout	59
6.2.4. Core Network	62
6.2.5. Management and Orchestration	64
6.2.6. Functions beyond 3GPP	67
6.3. ACHO: A framework for flexible re-orchestration of virtual network functions	70
6.3.1. ACHO: A suite for flexible 5G networking	71
6.3.2. Re-configuration of VNFs, a context-based approach	71
6.3.3. Baseline 5G implementation	73
6.3.4. New MANO functionality	75
6.3.5. Re-orchestrable VNFs	76
6.3.6. ACHO adoption strategies	78

7. Experimental evaluation	81
7.1. End-to-end Network Slicing implementation	81
7.1.1. Testbed Description	81
7.1.2. Independence between slices	82
7.1.3. Throughput performance	83
7.1.4. Slice customization and orchestration	84
7.1.5. Compatibility with commercial equipment	84
7.2. Multi-service and multi-slice deployment evaluation	85
7.2.1. Testbed Description	85
7.2.2. Slicing-aware MAC Scheduling	86
7.2.3. Service Creation Time	86
7.2.4. VNF re-location	88
7.2.5. Low latency through LB	89
7.3. Flexible orchestration with ACHO	90
7.3.1. Testbed description	90
7.3.2. VNF relocation delay	91
7.3.3. Performance under re-orchestration	93
8. Summary Part II	97
III Bringing cloud nativeness to softwarized mobile networks	99
9. Designing a Cloud-native Radio Access Networks	101
9.1. Fundamentals of 4G and 5G	106
9.2. LTE and NR pipeline diagnosis	109
9.3. Cloud-native RAN	112
9.3.1. DSP forethead	114
9.3.2. Data workers	115
9.3.3. Early HARQ	117
9.3.4. Congestion control	119
10.Experimental evaluation	121
10.1. Testbed description	122
10.2. Downlink	122
10.3. Uplink (Early HARQ)	123
10.4. Capacity region	124
10.5. Multiple vDUs	126
11.Summary Part III	127

12. Conclusions and future work	129
12.1. Conclusions	129
12.2. Future Work	130
References	131

List of Tables

3.1. Recent software contributions for network slicing.	19
7.1. Service Creation Time KPI	87
7.2. VNF relocation delays obtained by ACHO and by <i>OpenStack</i>	92
9.1. LTE & NR Channels	107
10.1. Cloud-native Parametrization	121

List of Figures

2.1. Simplified 4G architecture overview	8
2.2. Simplified 5G architecture overview	9
2.3. Network slicing definition	12
2.4. Software Defined Network controller architecture.	14
4.1. Different approaches to softwarization, from monolithic elements (top) to a completely <i>cloudified</i> network (bottom).	22
4.2. The transition from a traditional to a pipelined network protocol stack. Same subscript means that the same functionality is fulfilled.	26
4.3. Performance degradation achieved by elastic computation: performance is not degraded by the same relative amount as resources are reduced.	28
4.4. Major transitions in the adoption of softwarization.	30
4.5. Mobile network architecture evolution.	32
4.6. The liquid scalability (top) and an empirical evaluation (bottom)	34
4.7. The simplified threading architecture of the srsLTE software (left) and a possible serverless design of the software stack (right).	36
4.8. eBPF-enhanced data path vs the traditional iptables-based approach.	37
4.9. Full VNF relocation	43
6.1. Different RAN slicing options.	50
6.2. Design of POSENS: changes introduced at the UE and the eNodeB.	53
6.3. General scenario showing a legacy GTP tunnel (top, red dotted/solid lines) and a GTP with LB (bottom blue line). The LB introduces additional interface IF for routing traffic locally.	60
6.4. TEID table maintained by the LB threads: it associates TEID numbers to the IP address of each UE. It is used for crafting GTP tunnel return packets.	61
6.5. Sketched view of the 5G Core, as in [10]	62
6.6. The OSM architecture (Adapted from [11])	64
6.7. The relations between our implemented architecture and the ETSI - 3GPP domains	65
6.8. NS Descriptor snippet.	66

6.9. VNF Descriptor snippet.	67
6.10. The Orchestration architecture	68
6.11. the context relocation performed with ACHO	72
6.12. Representation of the SMF context	74
6.13. Representation of the MAC context	74
6.14. MANO Implementation and new Interfaces. ACHO creates new interfaces in the reference points defined by ETSI and acts on the underlying virtual or physical NFs (both c-plane and u-plane) to provide a fast re-location. . .	74
7.1. A multi-slice network architecture.	82
7.2. Independence between slices	83
7.3. Total and per-slice throughput performance	84
7.4. Independent Service Function Chaining	85
7.5. (a) Obtained throughput with variable resources shares vs share	86
7.6. (b) Obtained throughput with variable resources shares vs time	86
7.7. Latency evaluated for the URLLC scenario.	89
7.8. The Physical testbed setup.	91
7.9. The Network Slice setup employed in the experimental evaluation, consisting of 3 slices.	92
7.10. SFC amendment. Flow A (top), B (middle) and C (bottom).	94
7.11. Relocation of the IoT gateway across edge clouds.	94
7.12. UPF migration from the central cloud to the edge cloud, under different configurations.	95
7.13. On-demand Radio resources assignment	96
9.1. Virtualized Radio Access Network (RAN) architecture [12]	102
9.2. Baseline digital signal processor (DSP) pipeline parallelization. One subframe every transmission time interval (TTI, 1 ms)	103
9.3. Decoding time of one transport block in a CPU core.	104
9.4. Two Virtualized Distributed Units (vDUs) competing for computing resources. The Uplink (UL)/Downlink (DL) data load of vDU 1 is the highest possible while vDU 2's is variable.	104
9.5. Dedicated workers for UL/DL data processing tasks. <i>Forethread</i> coordinates subframe processing.	105
9.6. Computing allowance, split into two phases, impose a hard deadline on UL/DL data workers.	105
9.7. <i>Early HARQ</i> predicts the <i>decodability</i> of unfinished UL data tasks.	106
9.8. Congestion controller adapts the generation of data tasks to the available computing capacity.	106
9.9. Subframes and PHY radio channels (FDD).	107

9.10. LTE and NR PHY pipeline: DSP job n	108
9.11. vPhysical Layer (PHY) pipeline parallelization for $M = 4$ and four DSP workers in a worker-time grid. Colored cells represent the computing budget of a worker to process a DSP job. Because each job must be completed within 3 ms ($M = 4$), worker 1 <i>must</i> be available to process job $n + 3$, which makes worker 4 always idle (dashed cells) and it is hence <i>redundant</i> . Therefore, the maximum pipeline depth is 3, i.e., 3 parallel workers.	110
9.12. Throughput performance for both uplink and downlink (top). Central Processing Unit (CPU) time required by different PHY layer functions (bottom). Different uplink/downlink load (relative to the maximum) and channel conditions (Signal-to-noise ratio (SNR)).	111
9.13. A 4G/5G PHY pipeline for cloud environments.	113
9.14. DL-Data worker operation.	116
9.15. UL-Data worker operation	117
9.16. Mean extrinsic magnitude for each iteration of a turbodecoder. Dot/line indicate the average value across multiple PUSCH TBs with different MCS, TBS and SNR. Error bars indicate the standard deviation.	119
10.1. Downlink throughput (left) and vPHY buffering (right) with saturating DL load. Comparison between the baseline and cloud-native approach with different λ settings under different computing capacities.	122
10.2. Early Hybrid Automatic Repeat Request (E-HARQ) accuracy and false positive rate for different γ settings, and for different combinations of Modulation and Coding Scheme (MCS), Transport Block (TB) size, SNR, and computing capacity.	123
10.3. Maximum computing latency supported by cloud-native's E-HARQ given a target accuracy.	124
10.4. Capacity region of the cloud-native solution and our baseline. Different computing capacity settings equal to $k \cdot c_0$, where c_0 is the nominal encoding/decoding capacity of a <i>dedicated</i> Intel Xeon core @ 1.9GHz. . .	125
10.5. Network capacity for a variable number of vDUs contending for computing resources.	126

List of Acronyms

3GPP	3rd Generation Partnership Project
ACHO	Adaptive slice re-Configuration using Hierarchical Orchestration
ACK	Acknowledgement
AIMD	Additive-Increase/Multiplicative-Decrease
AMF	Access Management Function
API	Application Programming Interface
AR	Augmented Reality
ARQ	Automatic Repeat Request
ASIC	Application-Specific Integrated Circuit
AUSF	Authentication Server Function
BBU	Baseband Unit
BPF	Berkeley Packet Filter
BS	Base Station
C-RAN	Cloud-RAN
CB	Code Block
CN	Core Network
CNF	Cloud-Native Network Function
c/e	Context/Execution
CP	Cyclic Prefix
CPRI	Common Public Radio Interface
CPU	Central Processing Unit
CQI	Channel Quality Indicator
CRC	Cyclic Redundancy Check
CU	Central Unit
DCI	Downlink Control Information
DL-SCH	Downlink Shared Channel

DL	Downlink
DPDK	Data Plane Development Kit
DSP	digital signal processor
DU	Distributed Unit
E-HARQ	Early Hybrid Automatic Repeat Request
eBPF	enhanced Berkeley Packet Filter
eDECOR	Enhanced Dedicated Core Network
eMBB	Enhanced Mobile Broadband
eNB	Evolved Node B
EPC	Evolved Packet Core
FaaS	Function as a Service
FDD	Frequency Division Duplex
FEC	Forward Error Correction
FFT	Fast Fourier Transformation
FG	Forwarding Graph
gNB	5G Next Generation NodeB
GTP	GPRS Tunneling Protocol
GUI	Graphical User Interface
GW	Gateway
HARQ	Hybrid Automatic Repeat Request
HSS	Home Subscriber Server
HTTP	Hypertext Transfer Protocol
IFFT	Inverse Fast Fourier Transformation
IoT	Internet of Things
KPI	Key Performance Indicator
KVM	Kernel-Based Virtual Machine
LB	Local Breakout
LCM	Life Cycle Management
LDPC	Low Density Parity Check
LLR	Log-Likelihood Ratio
LTE	Long Term Evolution

LXC	LinuX Containers
MAC	Medium Access Control Layer
MANO	Management and Network Orchestration
MCS	Modulation and Coding Scheme
MEC	Mobile Edge Computing
MIB	Master Information Block
MIMO	Multiple-Input Multiple-Output
MME	Mobility Management Entity
mMTC	Massive Machine-type Communications
MVSF	Minimal Viable Subframe
NACK	Negative Acknowledgement
NAS	Non-access stratum
NEF	Network Exposure Function
NEST	Network Slice Template
gNB	Next generation Node B
NF	Network Function
NFV	Network Function Virtualization
NFVI	Network Function Virtualisation Infrastructure
NFVO	NFV Orchestrator
NGMN	Next Generation Mobile Networks Alliance
NIC	Network Interface Card
NR	New Radio
NRF	Network Repository Function
NS	Network Service
NSaaS	Network Slice as a Service
NSD	Network Slice Descriptor
NWDAF	Network Data Analytic Function
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiplexing Access
ONAP	Open Networking Automation Platform
OSM	Open Source MANO

P-GW	Packet data network Gateway
PBCH	Physical Broadcast Channel
PCFICH	Physical Control Format Indicator Channel
PDCCH	Physical Downlink Control Channel
PDCP	Packet Data Convergence Protocol
PDN	Packet Data Network
PDSCH	Physical Downlink Shared Channel
PHY	Physical Layer
PNF	Physical Network Function
PRACH	Physical Random Access Channel
PRB	Physical Resource Blocks
PSS	Primary Synchronisation Signal
PUCCH	Physical Uplink Control Channel
PUSCH	Physical Uplink Shared Channel
QoS	Quality of Service
QR	Quick Response
RA	Random-Access procedure
RAM	Random-access memory
RAN	Radio Access Network
RB	Resource Block
RBG	Resource Block Group
REST	Representational state transfer
RLC	Radio Link Control
RRC	Radio Resource Control
RRH	Remote Radio Head
RTT	Round-Trip Time
RU	Radio Unit
S-GW	Serving Gateway
S1AP	S1 Application Protocol
SaaS	Software as a Service
SBA	service based architecture

SC-FDMA	single-carrier FDMA
SDN	Software Defined Networking
SDR	Software Defined-Radio
SF	Subframe
SFC	Service Function Chain
SIB0	System Information Block Type 0
SIB1	System Information Block Type 1
SIB2	System Information Block Type 2
SMF	Session Management Function
SNR	Signal-to-noise ratio
SR-IOV	Single Root I/O Virtualization
SSS	Secondary Synchronisation Signal
TB	Transport Block
TEID	Tunnel endpoint identifier
TTI	Transmission Time Interval
UCI	Uplink Control Information
UDM	User Data Management
UDP	User Datagram Protocol
UE	User Equipment
UL	Uplink
UPF	User Plane Function
URLLC	Ultra-Reliable Low Latency Communications
vDU	Virtualized Distributed Unit
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VNFD	Virtual Network Function Descriptor
VNFM	Virtual Network Function Manager
VPN	Virtual Private Network
VR	Virtual Reality
vRAN	Virtualized RAN

PART I

INTRODUCTION, MOTIVATION AND CHALLENGES

1

Introduction

Technologies have drastically changed our world, becoming a fundamental part in our society. Within this digital change, telecommunications networks are becoming more and more important due to their intrinsic ability of providing access to digital services from all over the world. Currently, millions of people are producing and/or consuming substantial amounts of load in the providers' infrastructure using network-enabled services.

The 5th generation of mobile telecommunications networks are expected to change the way users see and approach mobile networks, evolving from a simple enabler to a complete manageable service inside a complex service-oriented ecosystem where service providers, telecommunications operators and users coexist.

This evolution is subject to technical, service and operational requirements, which means that there will be a large variety of requirements that must be correctly managed by the network. Having multiple individual deployments is not an option, because new generation networks are also expected to be energy efficient and cost effective. This situation motivates the need for flexible and scalable network, to properly ensure that the mobile network will be adapted to specific use cases (in terms of bandwidth, latency, ...).

1.1. Motivation

While specification and documentation activities regarding the 5th generation mobile systems (5G) have progressed at a rapid speed, practical experiments have evolved at a slower pace. Apart from the obvious reasons (the relatively high development costs), another (and partly related) issue that precluded the prototype of mobile system is the unavailability of resources: mobile equipment is typically expensive, and the frequency bands are licensed. Because of this, operators were the only groups performing experimentation with cellular systems, while the academia was restricted to technologies based on the ISM band (see e.g. [13] for a recent survey on experimentation with 802.11). One technology that helped this ISM-based experimentation is Software Defined-Radio (SDR), which enables researchers to implement all layers of a wireless protocol stack.

The use of SDR has enabled the implementation of novel access schemes and communication paradigms (such as e.g. cognitive radio, full-duplex), but prototypes have been limited to the lower layers of the protocol stack (i.e., the MAC layer), the main reason being the lack of interest on developing complete systems over these prototype boards, given the high development costs. These costs are further exacerbated for the case of mobile systems, as these are characterized by relatively complex protocol stacks (in contrast to e.g. TCP/IP). The situation, though, has recently changed with the emergence of software platforms based on SDR that enable instantiating a complete mobile network, with Eurecom's OpenAirInterface (OAI) [14] and srsLTE [15] among the most popular initiatives.

The availability of these open projects has leveled up the playground, with now more players (academics, SMEs) being able to develop novel enhancements for cellular systems. This helps accelerating the development of solutions for 5G networking, which will boost the running of field trials under realistic conditions and bring better understanding about the performance of future mobile systems.

Despite the above, the gap between theory and practice for 5G networking is still large. In particular, while network slicing has been acknowledged as a key technology to efficiently support services with very diverse requirements [16] or *Softwarization* of networks trend is becoming reality, there are little experimental "hands on" reports on the use of this technologies in practice.

1.2. Challenges and Contributions

As described above, 5th generation of networks prototypes progress at a slower pace than specification activities currently does. In this thesis we will focus on study the required mechanisms for bringing network slicing and flexibility to 5G networks from an experimental point of view. During this process, we will tackle the main challenges associated with these novel technologies and propose novel solutions to effectively overcome the issues we found. Hence, with this work we aim to contribute filling the gap by effectively design and implement a fully virtualized network-slicing aware mobile protocol stack.

This thesis focus on investigating the benefits of a network slicing-aware protocol stack and the design of a novel cloud-native Radio Access Network (RAN). This thesis summarizes in 4 publications, 1 published in *IEEE Wireless communications*, 1 published on *Computer Communications Journal*, 1 published on *Computer Networks Journal* and 1 published on The International Federation of Information Processing (IFIP) Networking Conference.

Other 6 works have been published on different conference/journals such as

Proceedings on Privacy Enhancing Technologies Journal, IEEE International Conference on Communications or *ACM WINTECH*. In detail,

- **Network slicing implementation:** A real implementation of network slicing inside a 4G LTE protocol stack. The implementation attacks all the challenges of network slicing from the design and implementation point of view, finally accomplishing the expected requirements.
- **Novel Orchestration mechanism:** Placement and orchestration mechanisms have been widely studied theoretically, but here we provide an experimental evaluation of the living approaches and propose a novel context-based orchestration to efficiently achieve the required network flexibility on 5G networks.
- **Cloud-native RAN:** Design of a cloud-native RAN which is designed work on shared cloud environments. The proposed design outperforms the baseline approach and increases reliability and resiliency when computing fluctuations appear.
- **Experimental outcomes:** During this work, we have been producing experimental results as well as software components to enrich and speed up research in a more experimental field.

1.3. Thesis overview

This thesis is organized as follows:

- Part I we first introduce the main technologies we used during this thesis in Chapter 2. Then, in Chapter 3 we present the main open source solutions available for 5G prototyping. Lastly, Chapter 4 is dedicated to the study of the mobile networks evolution, resulting in novel approaches and concepts to properly overcome the future network requirements.
- Part II is dedicated to perform the design and implementation of a complete end-to-end network slicing network architecture, a 5G network prototype and the design and implementation of a framework for flexible orchestration (Chapter 6). Then, we perform an experimental evaluation of the proposed approaches (Chapter 7).
- Part III, where we present a novel cloud-native RAN whose design is able to effectively deal with non-stable cloud environments. In Chapter 9, we design the novel pipeline architecture, while Chapter 10 is dedicated to the performance evaluation.

2

Background

the 5th Generation (5G) of mobile networks magnifies how important telecommunications are within our society. This change is seen as the transition from a simple enabler for communications to a complex ecosystem of service providers, operators and users managing limited instances of the network. This novel vision of the network requires a drastic re-design of the network, whose main objective is to properly accommodate services with many diverse requirements.

In this section, we introduce the main concepts of mobile networks, 4th and 5th generation of mobile networks as well as the main novelties of the latter.

2.1. Mobile Networks

Mobile networks or cellular networks are communications infrastructures providing wireless connectivity to end nodes (i.e. mobile phones). These infrastructures consist on multiple transceivers (also known as Base Station (BS)) providing "cells", name that represents a coverage area of a specific BS. Those coverage areas can be used for transmitting/receiving content from the network and neighbouring cells usually uses different frequency sets to avoid interfering each other. The main features of mobile networks are:

- Increased capacity compared with single large transmitter, because the same frequency can be used in different cells.
- Extended coverage, because we can continue adding BS to extend coverage without being limited by the horizon.
- End devices consume less power, because BS are closer.

In the following sections, we will take a look at the main characteristics and components regarding the 4th and 5th generation of mobile networks (Sections 2.1.1 and 2.1.2 respectively)

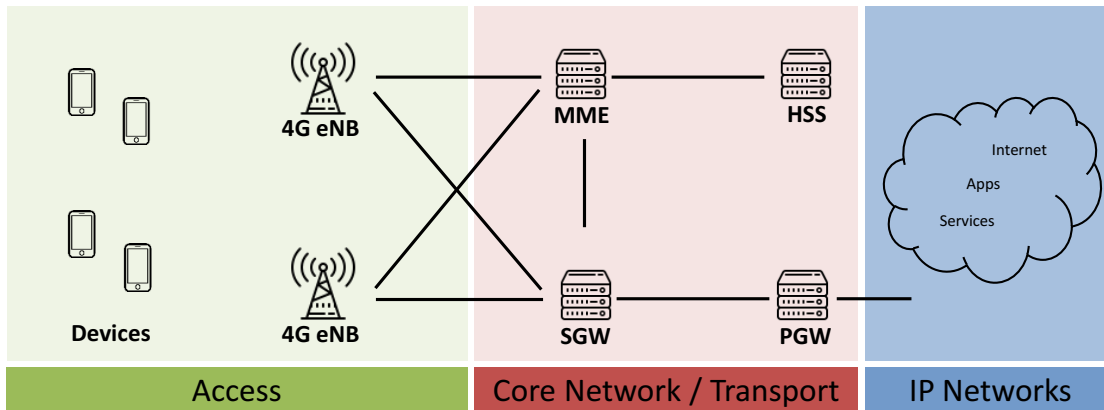


Figure 2.1: Simplified 4G architecture overview

2.1.1. The 4th Generation of mobile networks

4G stands for fourth generation of mobile networks. The most important challenge addressed by 4G was the integration, providing an IP based mobile broadband that allowed users to access streaming services, gaming or voice over IP calls. Next, we will take a look at the main components of the 4G infrastructure.

4G Access:

In mobile networks, the radio access is one of the most important parts of the infrastructure. This part is in charge of providing the air interface at which the devices (User Equipment (UE)) will be connected to. Traditional Radio Access Network (RAN) consists on two components: the *Baseband Unit (BBU)*, which comprises all the computer intensive functions to properly perform signal processing; the *Remote Radio Head (RRH)*, for digital to signal (and vice-versa) conversion, ensuring that signals are correctly transmitted (frequency, power gain, etc.). A BBU and up to three RRHs conforms what we call Evolved Node B (eNB), being the standard Common Public Radio Interface (CPRI) the optical interface connecting these components. UEs and eNBs are depicted in green left part of Figure 2.1.

4G Core Network:

The Evolved Packet Core (EPC) is the anchor point for the traffic coming from the users connected to the eNB and whose main purpose is to reach services at the network. The EPC was designed with a "flat architecture", where few nodes are handling traffic and there is a split between user data (user plane) and signalling (control plane) to facilitate operator the dimension of each part independently. Figure 2.1 shows at the red section the a basic architecture of an EPC. At the upper part we have the modules in charge of managing the control plane, while in the lower part the user plane will be handled. Let's

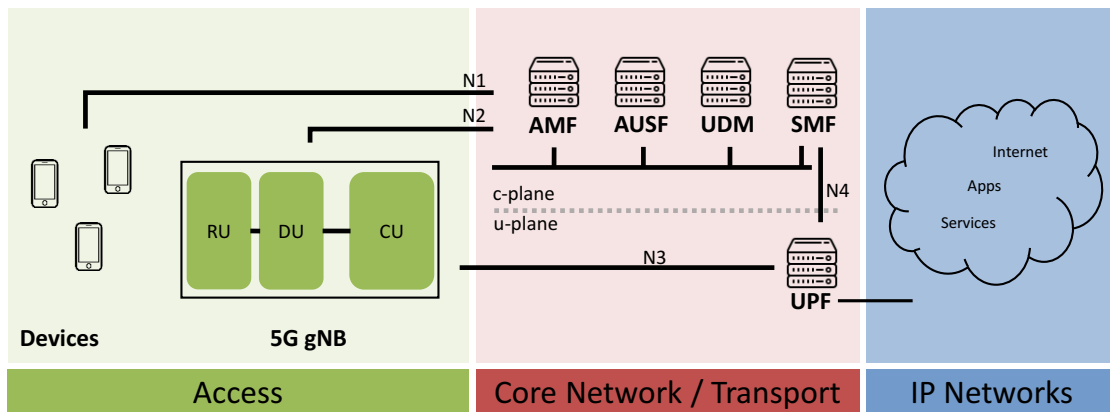


Figure 2.2: Simplified 5G architecture overview

now check each component separately:

- **Home Subscriber Server (HSS):** This component is essentially a database where all the information required for user authentication is stored.
- **Mobility Management Entity (MME):** This component is in charge of handling all the control plane messages coming from eNBs (i.e. S1 Application Protocol (S1AP)).
- **Serving Gateway (S-GW):** The S-GW is the first in charge of handling traffic between the UE and the external network, connecting the eNB to the EPC.
- **Packet data network Gateway (P-GW):** This component interconnects the EPC with external IP networks. It also performs the IP address allocation for UEs.

2.1.2. The 5th Generation of mobile networks

5G Networks will change the way in which cellular connectivity is provided. High data rates (500+ Mbps), extensive coverage (10+ Tbps/Km²) low latencies (<5 ms) and high network flexibility are just few of the target Key Performance Indicators (KPIs) to be fulfilled by the next generation mobile networks [17]. However, this results in major changes, some of them related to how networks are implemented and deployed to make them "cloud enabled". To clearly identify the novelties introduced by 5G, we next introduce the main building blocks of the architecture.

5G Access:

Although the main purpose of the radio access will remain, the arrival of 5G has changed the functionality and how the RAN components are split within the Next generation Node B (gNB) (Figure 9.1), which is the main component in the access network.

This component is the evolution of the eNB in 4G, but in this case its functionality has been split into three blocks: (i) *Radio Unit (RU)*, which manages the digital front end and implements parts of the Physical Layer (PHY) layer (also called low PHY); (ii) *Distributed Unit (DU)*, placed close to the RU and implements high PHY, Radio Link Control (RLC) and Medium Access Control Layer (MAC); (iii) *Central Unit (CU)*, in charge of implementing Radio Resource Control (RRC) and Packet Data Convergence Protocol (PDCP) layers.

Regarding the Interfaces, N2 and N3 are the control and user plane interfaces respectively. The gNB uses this interfaces to correctly communicate with the 5G Core Network (CN). Moreover, the N1 interface is a transparent interface created by the gNB to allow UEs to transmit non radio signalling to the 5G CN.

5G Core network:

The 5G core network is an evolution of the previous 4G core network architectures. In this case, The 5G CN relies on an service based architecture (SBA), which means that the 5G CN will be a set of interconnected Network Functions (NFs) offering and consuming services from each other. This approach opened the adoption of cloud-based architectures such as softwarization and virtualization, automatic deployment and scaling, etc. Moreover it also includes design enhancements for natively support network slicing (see Section 2.2). Next, we will introduce the 5G CN components depicted in Figure 9.1:

- **Access Management Function (AMF)**: manages authentication and mobility between device and network. Using the N1 interface, the UE sends to this component connection and session related information.
- **Authentication Server Function (AUSF)**: Performs part of the authentication (together with AMF and User Data Management (UDM)).
- **UDM**: Similarly to AUSF, it performs parts of the authentication process in collaboration with AUSF and AMF
- **Session Management Function (SMF)**: Manages sessions, IP allocation and control policies.
- **User Plane Function (UPF)**: This component basically manages routing and packet forwarding, as well as Quality of Service (QoS) policies application.

Management and Orchestration

As previously revealed, 5G network architectures softwarization will bring a huge amount of services with stringent requirements that must be accurately guided to satisfy their requirements. Management and orchestration will deal with the challenges coupled with high variety of requirements scenarios. Complex challenges such as deployment across multiple domains, efficient resources provisioning when using shared infrastructures or optimal usage of the allocated resources as well as simple ones such as service deployment will be tackled by the management and orchestration system.

2.2. Network slicing

Albeit 5G entails a huge step forward due to the addition of novel technologies, the undeniable revolution comes with the flexibility and programmability of the network. Flexibility requires being hardware-independent, and here is when virtualization of network functions and network slicing [18] come into play.

Network slicing is a network architecture that enables multiplexing of virtualized and independent logical networks on the same shared physical infrastructure (Figure 2.3). A *network slice* consists of a set of resources assigned to a *tenant* to provide a specific *service*. Those resources are both network resources (e.g., spectrum, link capacities), and cloud resources (i.e., the infrastructure required to run the Virtual Network Functions (VNFs)). Tenants could be mobile network operators providing Enhanced Mobile Broadband (eMBB), or third party *verticals* [19] that use a slice specifically tailored to their needs (e.g., Ultra-Reliable Low Latency Communications (URLLC)). To satisfy the service requirements of each tenant, a different network slice will be instantiated to provide the corresponding service. This ability to provide highly customizable services over the same shared infrastructure will increase the revenue opportunities, and drastically reduce the costs of 5G networking, due to the improvements in efficiency.

2.2.1. Properties and challenges

The advantages of network slicing are clear [16], and there is a wide consensus among the industrial and standardization communities on the need to adopt this technology. The main properties of network slicing can be summarized as follows:

- **Isolation:** Network slices must have their own resources. Moreover, the connectivity within a slice must be managed from inside while connectivity between slices must be granted/removed from outside of the network slices.
- **Assurance:** Network slices must assure a specific performance with a specific level of trust, being the latter how trusted the ability of the network slice to provide a certain quality of service.

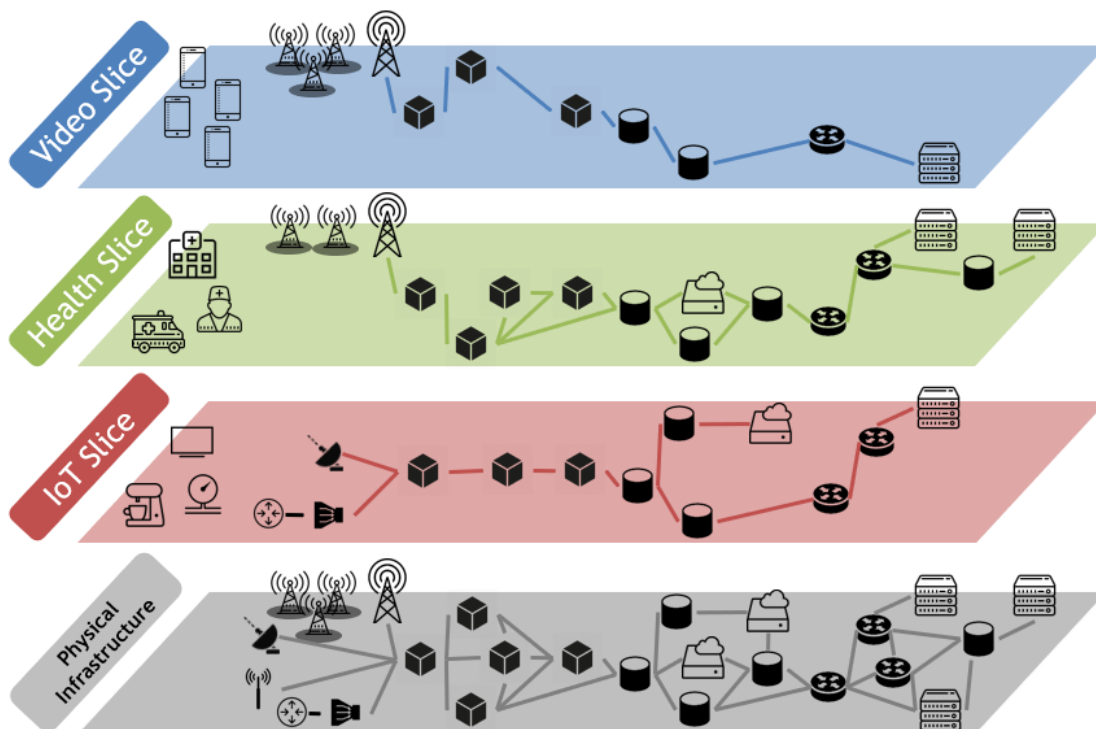


Figure 2.3: Network slicing definition

- **Scalability:** Network slices must be able to automatically being reconfigured/scaled at minimal effort.

The idea of creating logical networks on top of common infrastructures lead to multiple challenges, going from security and privacy to physical limitations due to different regulations because of the geographical location of the infrastructure. The design Challenges of network slicing can be grouped as follows:

- **Resource sharing:** The resources sharing is one of the key issues of network slicing. Sharing can be a simple static partitioning or an elastic dynamic sharing of the resources. The last option is more efficient due to the dynamic nature of the load but it requires advanced techniques to correctly manage these resources.
- **Dynamic management of slices:** Management of slices, including service function chain amendments, is another challenging procedure that, in this case orchestrators, must take care of. The challenge here is to efficiently accommodate multiple slices while keeping isolation both in terms of connectivity and performance.
- **Partial/complete slice isolation:** keeping isolation between slices is vital in sliced environment. Isolation could be achieved by isolating control and data plane of different slices. Nonetheless, part of the control plane could be shared among slices without breaking the isolation policy.

However, we lack a thorough experimental validation of its effectiveness, e.g., on the gains when using different mechanisms for orchestrations, or under different traffic scenarios. While there are implementations for some of the the enablers for network slicing, to the best of our knowledge there is no solution that implements end-to-end network slicing. More specifically, virtualization is a mature technology that has been extensively used for the wired elements, with technologies such as e.g. OpenStack¹ and Kubernetes² for virtual machine and containers management, respectively. However, the situation is less mature for the wireless access part, Orion [20] being among the few proposals to implement slicing at the RAN that have been tested in practice.

The inclusion of network slicing impacted both functional splits (see Section 2.2) and management and orchestration adding a new level of complexity. In Section 6.1 we will provide a detailed explanation about the impact of network slicing on RAN functional splits and in Section 6.3 we will cover the management and orchestration challenges of network slicing.

2.3. Virtualization of Network functions

Network Function Virtualization (NFV) is the natural step following the softwarization of network functions trend, where former physical boxes will now evolve to software components virtualized on shared physical infrastructures. This new operational approach will allow operators to easily update or modify network capabilities on demand while reducing costs.

2.3.1. Properties and challenges

NFV relies on virtualization mechanisms to run network functions, breking dependencies of network services from proprietary hardware. The usage of shared infrastructures results in a more efficient usage of the data center resources. Cloud technologies such as virtual machines or containers can effectively host network functions. NFV has the following advantages:

- **Of-the-shelf hardware:** The usage of virtual networ instances allows service providers to run network functions on commodity instead of dedicated hardware.
- **Efficiency:** a virtualization infrastructure can start/stop NFVs on-demand following the workload, which results in an efficient power consumption.
- **Flexibility:** Having the ability to start/stop/modify virtual instances as well as the service function chain that they conform, increases the flexibility of the deployment.

¹<https://www.openstack.org>

²<https://kubernetes.io>

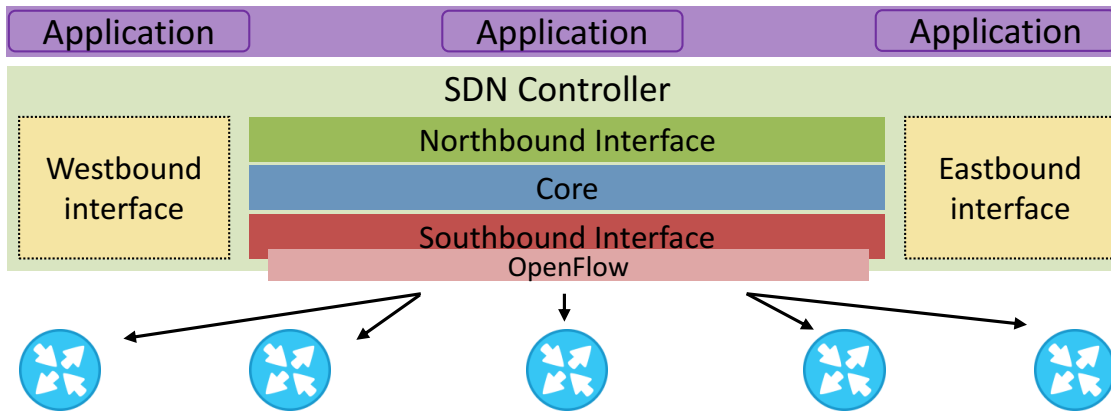


Figure 2.4: Software Defined Network controller architecture.

One key aspect of NFV is the softwarization of the network functions, which essentially brings the advantages of software components to network elements that were monolithic and difficult to modify. Moreover, is one of the enablers of network slicing (Section 2.2), where virtual network are created on top of physical shared infrastructures.

The challenges of NFV are not directly related to the concept itself, but with the integration and coordination of the different actors, together with the performance of certain network functions that have to deal with tons of packets per second. When applied to cloud environments, NFV brings as well scalability, resiliency and fault-tolerance to virtual network environments. As we will discuss in Part II and Part III, virtualization of network functions lead to different challenges that we will tackle along this thesis.

2.4. Software Defined Networking

Software Defined Networking (SDN) is a networking architectural approach to improve network management and flexibility. SDN implements a network abstraction to effectively separate control from data forwarding functions. The network control functions will now be centrally implemented in software applications called SDN "controllers" (see Figure 2.4), enabling the programming of network behaviours.

Looking at Figure 2.4, an SDN controller can be split in, at least, three main components. First, the *north interface* which is used for the communication between the SDN controller and the services an applications running on the network. This interface exposes functionality usually via Representational state transfer (REST) Application Programming Interface (API). Then we have the *core* of the controller is the framework where all the networking functionality is implemented (i.e. protocols, libraries, etc.). Finally, we have the *south interface*, which allows the communication with the SDN switches. OpenFlow [21] is an example of southbound interface and it defines how SDN controllers should interact with the network elements to properly make adjustments in the

network. Optionally, we could find the west/east interfaces that usually enables horizontal communication with other SDN controllers (usually for coordination purposes).

When SDN is combined with an NFV infrastructure, NFV enables the creation of virtual network instances (i.e. virtual switches) while SDN is in charge of configuring and controlling their behaviour. We can usually find virtualized SDN controllers managing virtualized network switches.

3

Open Source Virtual Network Functions

There are several recent initiatives to prototype mobile networks in software but in this work, we focus on open-source solutions. There exist different open-source components for each architectural block we want to build (see Section 2.1). At the RAN side, most of them are built on the GNU Radio development suite and the Ettus Research USRP Software Defined-Radio (SDR) platforms, and running on standard Linux-based computing equipment (Intel x86 PC architectures)¹. We next provide a short review of the current open-source solutions ecosystem.

3.1. Radio Access Networks

Concerning **the RAN part**, three of the most popular software solutions to run LTE over SDR are Eurecom's OpenAirInterface (OAI),² openLTE,³ and srsLTE.⁴ OAI [22] provides an implementation of a subset of LTE Release 10 elements, including the UE and the eNB. Its performance is considered relatively good, although it is also acknowledged that the code structure is complex and difficult to customize. It is also worth mentioning that the eNB and UE RAN are licensed under a specific OAI Public License.

openLTE is an open-source project providing an implementation of LTE specifications, which includes a C library, Octave code for testing downlink and uplink Physical Random Access Channel (PRACH) functionalities, GNU Radio applications for downlink functionalities, both simulated and using hardware platforms, and a simple implementation of an eNB using USRP. While its code is considered as relatively well organized, documented and would result easier to modify, it does not provide an UE, and many features are still unstable or under development.

¹We note that there are complete commercial products such as the Amari LTE 100 (a fully software-based Long Term Evolution (LTE) BS solution), its closed license makes it unsuitable for research activities.

²<http://www.openairinterface.org/>

³<http://openlte.sourceforge.net/>

⁴<https://github.com/srsLTE/srsLTE>

Finally, the srsLTE [15] open-source project provides a platform for LTE Release 8 experimentation, designed for maximum modularity. The RAN part provides a complete UE application and a complete eNB application. The project is more recent than OAI, and in general the source code is considered easy to customize, although it also consumes more CPU resources than the other alternatives. The code is provided under an AGPL v3.0 license.

Given that our aim is to implement and evaluate novel solutions and not an efficient software to put in production, code modularity and re-usability are determinant factors when selecting a platform, and therefore we decided to design our solutions as extensions of the srsUE and the srsENB (the applications for the UE and eNB, respectively).

3.2. Core Network

Concerning **the core network**, apart from commercial solutions such as OpenEPC(also supporting shared source code licensing), two of the most relevant solutions are the ones associated with the same initiatives mentioned above. Firstly, srsLTE has very recently released srsEPC, a light-weight CN implementation, including the MME, HSS, P-GW and S-GW, under the same license. Secondly, OAI also provides the same elements for a basic EPC solution, in this case released under a standard Apache v2.0 license. Given that when we started our work only the latter was available, we used OAI CN as the software solution for the CN.

One additional advantage of our POSENS, which contrasts Enhanced Dedicated Core Network (eDECOR) is interoperability: our solution works with any implementation supporting the S1AP protocol (we confirmed that POSENS is compatible with different different commercial EPCs, whose names we cannot disclosure due to confidentiality agreements).

3.3. Management and Orchestration

Finally, concerning **MANO**, it has received a lot of attention from both the open-source community and the enterprises [23]. There is a wide range of fully-fledged Management and Network Orchestration (MANO) tools such as Open Baton,⁵ OSM,⁶ that provide the required functionalities related to the VNF life-cycle management, including their scaling on a virtual infrastructure. They rely on a Virtual Infrastructure Manager (VIM), a software that is more mature, as it has been already employed in production cloud computing environments since many years. Among VIMs, we can list

⁵<https://openbaton.github.io/>

⁶<https://osm.etsi.org/>

Table 3.1: Recent software contributions for network slicing.

Work	Base	Purpose	Main Feature	Limitations	Open
[24]	srsLTE	RAN Slicing	Multiple, per tenant, eNB virtualization	Implementation only up to MAC layer.	No
[25]	OAI	RAN Slicing	Thorough evaluation of slices utilization	RAN Slicing only.	No
[26]	OAI	RAN Slicing	SDN-based RAN slicing	CN not included.	Upon request
[20]	OAI	E2E slicing	Core Network handling multiple slices	Single Slice UE only.	No
POSENS	srsLTE	E2E slicing	Slice aware shared RAN.	One RAN split available.	Yes

solutions such as OpenStack⁷ or OPNFV.⁸ However, as key required features such as per-tenant orchestration are not available with existing open-source solutions, we decided to implement POSENS MANO using a dedicated software that directly leverages on the VIM APIs.

3.4. Network Slicing

We finish this section by reviewing state-of-the-art solutions on Network Slicing that have recently appeared, which we list in Table 3.1. Available solutions are either considering the RAN only [24–26] or neglecting the UE role [20] at their implementations.

While each of these solutions provides specific elements to implement a 5G mobile network, only recently researchers started putting pieces together to implement end-to-end network slicing [4, 20, 27]. Among very recent efforts available in the literature that target an objective similar to ours we have the one in [28, 29], which base their implementation on OAI instead of SRSLTE, as we do. Finally, mention some ongoing initiative such as [30, 31], that aim at providing a living lab for wireless network researchers. In this work, we limit our research to small scale, locally hosted, deployments.

⁷<https://www.openstack.org>

⁸<https://www.opnfv.org/>

4

Software patterns for NFV

5G mobile networks will be characterized by a variety of services imposing a diversity of requirements¹. To efficiently support this diversity, we need a change of paradigm in the provisioning of Network Functions (NFs), moving from the traditional vision where Physical Network Functions (PNFs) are tightly coupled with the hardware substrate running them, to the new vision where Virtual Network Functions (VNFs) run over instantiations of a general-purpose infrastructure. It is envisioned that this transition will introduce a tremendous improvement in flexibility, adaptability and reconfigurability, similar to the one that happened when transitioning from circuit-based to packet-based networking.

By *softwarizing* the operation of the network, VNFs (e.g., schedulers, databases, gateways) run as software components over a set of shared resources (antennas, links, servers, etc.), and can be dynamically provisioned as needed. As illustrated in Figure 4.1, a possible transition from the traditional vision to a fully softwarized network is the “Softwarization of Elements.” That is, legacy PNFs are ported to a software implementation running in virtual containers.

This approach indeed improves the flexibility of the network: these monolithic programs run over shared computational resources, allowing, e.g., their re-instantiation on-demand, the reduction of development cycles and easier reconfiguration in general. Still, softwarization poses a number of challenges such as the efficient resource assignment to VNFs. This problem has traditionally been investigated from the network management perspective [35], but a very little effort has been done from the VNF design point of view. As VNFs are not designed to run over shared resources (that is, with virtualization in mind), there is the risk that any variation in these resources would cause service disruption: this potential *sensitivity* of VNFs to e.g. resource shortages, updates in the infrastructure, or container migrations, may preclude their wide use in future

¹This has been repeatedly argued by now, in a number of position papers such as, e.g., [32], in SDOs such as 3rd Generation Partnership Project (3GPP) [33], and in industry fora such as Next Generation Mobile Networks Alliance (NGMN) [34]

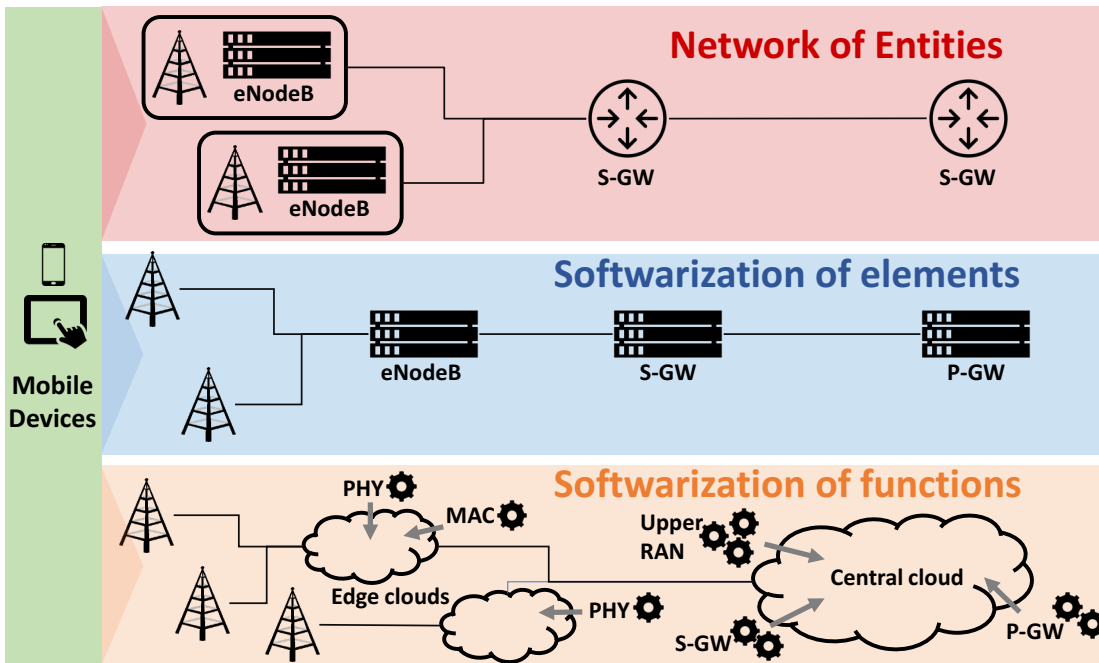


Figure 4.1: Different approaches to softwarization, from monolithic elements (top) to a completely *cloudified* network (bottom).

networks. Furthermore, given the indivisibility of these pieces of software, the assignment of programs to execution nodes (known as network embedding [36]) has been usually performed with a relatively coarse level of granularity, which hinders an efficient use of the resources.

A different approach to softwarization, which addresses this last issue, is depicted in Figure 4.1 as “Softwarization of Functions”: monolithic VNFs are decomposed into smaller functions that can be instantiated and chained as required. This approach provides a finer granularity to the mobile network operation, allowing thus a better flexibility and therefore a more efficient use of resources. This transition from a *network of elements* to a *network of functions* further complicates network management, as the minimum requirements (e.g., in terms of delay and bandwidth) on the function chaining are increasingly heterogeneous. However, there is little when none research effort on reducing and simplifying these requirements (especially in terms of delay) by modifying the interactions between different functions.

As discussed above, softwarization has brought a notable amount of novelties that have an unprecedented impact on the way network management has to be performed. However, the current protocol stack and its composing blocks can be considered “legacies” of the past: the way in which physical network functions are designed has not changed much since early 3GPP Releases. Indeed, the network functions addressed by current softwarization efforts — e.g., Serving Gateway (S-GW), Packet data network Gateway

(P-GW), or Radio Access Network (RAN) upper layers — have a practically direct mapping with the ones defined in 3GPP Release 8 [37], which was published almost ten years ago.

For these reasons, we need a change of paradigm in the design of the network functions, to efficiently support all the novel features that network softwarization and cloudification bring. In other words, the time is ripe for a new class of *cloud-aware protocol stacks*, which embrace softwarization as the fundamental design criteria.

4.1. Towards a cloud-aware mobile network protocol

Recent trends in network softwarization have indeed considered the transition steps described in Fig 4.1. However, although the landscape of software implementations of networking stack has grown in the recent years on both OpenSource [14, 15] and Commercial [38] sides, all of them (especially RAN) are, to the best of our knowledge, a “standard” porting of the legacy functionality that allow its execution on general purpose hardware. Engineers and researchers working on these projects have indeed cared about the overall performance of the system, striving to reduce the resource usage footprint (mostly in terms of CPU and memory). Still they did not make any further consideration on improving the protocol stack to favor its execution as pure software. And while we only consider above current RAN softwarization efforts coming from the open source community or from small enterprise solutions, also the available products in the most important network equipment vendors do not consider the opportunities we are discussing.

As a matter of fact, not considering a cloud-aware approach for the design of VNFs has direct implications on the proper dimensioning of the cloud. Most of the OpenSource solutions, for instance, require low latency kernels to correctly perform baseband processing, and rely on resource over-provisioning to avoid timing constraint violations that, in turn, result into frames dropped, poor user quality of experience, and even disconnections. Over-provisioning may be effective when dealing with laboratory deployments, but it results practically infeasible and extremely inefficient when real scenarios come into play.

Our Vision

We propose to follow a new research line for the design of software mobile network architectures that aims way beyond the existing strategies such as the functional splits already proposed in different fora (e.g., by 3GPP [39] and the Small Cell Forum [40]). We believe that making the protocol stack *cloud-aware*, as we will describe in Section 4.1.1, requires significant changes to the existing functions or even the design of new ones.

We thus advocate for a complete re-design of the mobile network protocol stack with the goal of achieving a *cloud-aware protocol stack*. We identify two possible research

lines that can follow this approach, detailing examples of how they can be applied in the context of a *cloudified* network.

4.1.1. The quest for cloudification

The advantages brought by a cloud-driven VNFs design will fuel the research community. In fact, while researchers have devoted so far just a little attention to solve the problems involved by this approach, we believe that the relative maturity of current software initiatives (and the recent increase in the pace of their updates) provides the means for research in this area to bloom.

In this thesis, we argue that future, fully softwarized and cloudified mobile networks will necessarily build on *cloud-aware protocol stacks*. We believe that both network management and the resulting overall performance will benefit from *making VNFs aware* of being executed in environments such as virtual machines or containers, running on shared resources. In Chapter 9 and 10 we design, implement and experimentally evaluate a cloud-native RAN solution. In this section, we discuss the main challenges to achieve this vision, while in the next section we describe three implementations of functionality that builds on this *cloud-awareness*, assessing how they will improve performance in a softwarized network both qualitative and quantitatively.

This approach entails two main challenges, namely *(i)* redefining the interactions between VNFs, relaxing as much as possible their temporal and logical connections, and *(ii)* support an elastic operation, to efficiently cope with changing input loads while running in an infrastructure of resources that is not over-provisioned. We detail the functional requirements of these novel design strategies in what follows, before discussing why they will also require the formal definition of novel Key Performance Indicators.

Given the high flexibility provided by the Network Function Virtualization (NFV) approach, the deployment of such *cloud-aware protocol stack* does not have a direct implication on the provided telecommunication service *per se*. The re-definition of the interactions among VNFs allows for a more flexible service orchestration, while the re-design of VNF internals may be easily provided by a code refactoring in a much faster way than the current tightly coupled HW-SW PNF approach. While having a *cloud-aware protocol stack* will benefit any kind of telecommunication service, this may be particularly relevant for the extreme ones. For example, a mission critical VNF can be optimized to reduce its memory footprint, while low latency services may exploit especially tailored orchestration patterns involving edge computing facilities, as we describe in the following section.

Re-thinking network interactions

Future network architectures will heavily rely on the flexible function decomposition and allocation [41]. That is, the former monolithic PNFs are split into interconnected modules that, concatenated, provide the same functionality: e.g., a physical Evolved Node B (eNB) is split into Physical Layer (PHY), MAC, Radio Link Control (RLC) and Packet Data Convergence Protocol (PDCP) software implementations running in different execution containers, which can be located in different nodes of the cloudified network.

This approach (depicted in the bottom part of Figure 4.1) provides several advantages, as it allows heterogeneous deployments for different services (i.e., massive machine type communication, enhanced mobile broadband), which are tailored to their specific requirements. For example, depending on the latency, bandwidth, and/or computational requirements of the service, it may be better to locate certain VNFs towards the edge of the cloud rather than in a central location. How to place VNFs across the cloud is a network orchestration problem, which is constrained by the split into modules described above. However, this typical NF decomposition for the RAN protocol stack was not designed for its cloudification, and therefore the potential gains are limited. We next discuss this issue in more detail.

One key assumption of network stack designs is that certain functions are implemented in the same physical space (maybe on a different chip, but surely on the same hardware). So, non-ideal links with non-negligible delays are a problem for physical network elements that need to be decomposed into several network functions. Interfaces among them, thus, were designed considering communication links spanning some microns of silicon, and not several miles of fiber as in the case of, e.g., Cloud-RAN (C-RAN) [42].

In this way, the possible inter-dependencies between these functions are overlooked, as the delivery of information between them is practically immediate. However, as we have argued above, to fully benefit from a network-wide orchestration of a cloudified stack, VNFs should support their execution on different nodes. But the design of traditional protocol stacks do not support such flexible placement of VNFs, as those with heavy inter-dependencies may introduce very high coordination overheads, or may not be even possible due to infeasible network requirements. These limitations severely constrain network orchestration, which compromises the overall gains obtained from the flexible function allocation. This is flagrant for e.g., the introduction of centralized RAN functions, where long delays in the information exchange between radio access points and the central cloud result in serious performance deterioration.

Because of the above, the full protocol stack (and, in particular, the RAN) has to be re-designed with the goal of leveraging the benefits of the flexible function decomposition and allocation, so as to cope with non-ideal communication (i.e., non-zero and varying delay, limited throughput) between the nodes in the cloud. Specifically, a *cloud-aware protocol stack* should relax as much as possible, or even completely remove, the logical

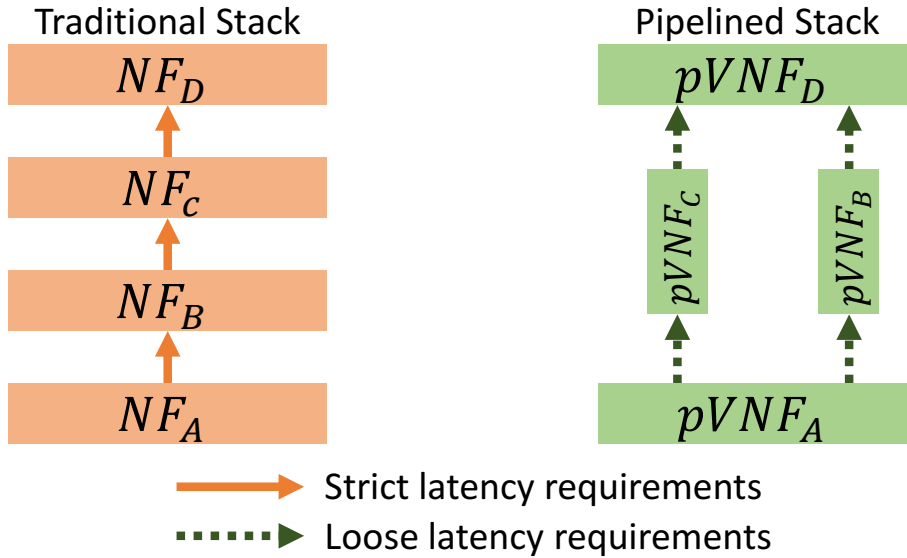


Figure 4.2: The transition from a traditional to a pipelined network protocol stack. Same subscript means that the same functionality is fulfilled.

and temporal dependencies between VNFs, to enable their parallel execution and provide a higher flexibility in their placement. We refer to this approach as *pipelined network stack*, which is illustrated in Figure 4.2 (right), showing its differences as compared to the traditional stack (left): the challenge is to define new pipelined and virtual versions of the traditional NFs (marked as pipelined VNF, pVNF).

4.1.2. Re-designing VNFs internals

One of the most immediate and appealing advantages of a cloudified network is the possibility of reducing costs, by adapting and re-distributing resources following (and even anticipating) temporal and spatial traffic variations. However, it is also likely that in certain occasions the resource assignment across the cloud cannot cope with the existing traffic due to some peaks of resource demands. This is particularly true for C-RAN deployments, that have to deal with demand loads known to be highly variable [43]. In this scenario, allocating resources based on peak requirements would be highly inefficient, as this design jeopardizes multiplexing gains in particular when cloud resources may be scarce (e.g., a “flash crowd” at an edge cloud): here any temporal shortage might result in a system failure. VNFs, instead, shall efficiently use the resources they are assigned with. Thus, they have to become *elastic*, i.e., adapt their operation when temporal changes in the resources available occur, in the same way they have a long-established manner of dealing with outages such e.g. channel errors. Therefore, to fully exploit the benefits of softwarizing the network operation, the network function design has to take the potential scarcity into account, and be prepared to react accordingly. This challenge is addressed

in Chapter 9 and 10, where we propose a cloud-native RAN design that effectively adapts to temporal changes in cloud environments.

In the context of wireless communications, the concept of elasticity usually refers to a graceful performance degradation when the spectrum becomes insufficient to serve all users. However, in the framework of a cloudified operation of mobile networks that has to deal with elasticity under resource shortages, we also need to consider other kinds of resources that are native to the cloud environment such as computational, memory, and storage assets available to the containers the VNFs are bound to. This has hardly been a problem for traditional network functions, that were designed to run over a given hardware substrate with exclusive access to the resources, and requires the definition of novel interfaces that will provide the amount and type of available cloud resources at a given point in time, just like, e.g., the accessible spectrum is a parameter for a RAN function.

Elasticity has also been considered by non-VNFs cloud operators, but our concept deviates very much from theirs: the time scales involved in RAN functions are significantly more stringent than the ones required by e.g., a Big Data platform or a web server back-end. Another key difference is that resources are way more scattered in our scenario (e.g. they are distributed across the “edge clouds”), which reduces the possibility of damping peaks by aggregating resources.

To better illustrate the benefits of elasticity in the cloudified mobile network operation context, we first consider the notion of “computational outage” [44], i.e., the unavailability of the required resources to perform the expected operation. In a traditional, non-elastic operation, there is a 1-to-1 mapping between outages and performance loss, as Figure 4.3 illustrates: if the resources are not available 20% of the time, there is a 20% performance degradation, as the function is unable to operate under any shortage. In contrast, an elastic design supports what we refer hereafter as *graceful* performance degradation, which causes that the VNF still functions under a resource shortage, this resulting in the “gains” illustrated in the Figure. Making a protocol stack *cloud-aware* through elastic VNFs requires hence a paradigm shift in their design, moving away from the tight hardware-software co-design that we discussed before, to a flexible operation in which the amount of available resources is an additional parameter.

To fully take advantage of elastic VNFs, a detailed analysis of their operation is required: first, a thorough assessment of the resources consumed during execution, including statistics about temporal variations over time; second, a characterization of the correlations between VNFs operations, to serve as input for the orchestration algorithm, so it could e.g. dynamically assign resources to resilient VNFs and quickly “rescue” them when outages happen.²

²Indeed, the quest for cloudification will end up with novel orchestration algorithms.

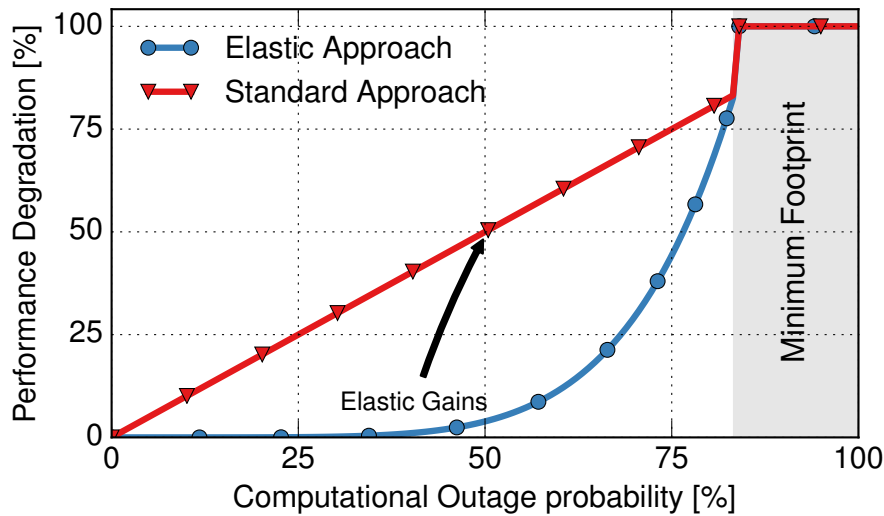


Figure 4.3: Performance degradation achieved by elastic computation: performance is not degraded by the same relative amount as resources are reduced.

4.1.3. The need for performance indicators

Another consequence of a *cloud-aware* network stack is the need for a new framework to (i) quantify its behavior, (ii) assess its design and (iii) serve as input for the orchestration algorithm, which would decide where to place the VNFs depending on their elasticity. In this section we discuss about how to evaluate the benefits of the two strategies proposed above, especially focusing on the elastic VNF design, as it has more cross-fertilization opportunities.

The cloud computing community has indeed long worked on the definition of elasticity, this generally being defined as the ability to provision and de-provision resources to match the demand at each time instant as closely and efficiently as possible [45, 46].³ Building on this definition, it is possible to assess the degree of elasticity of e.g. a certain system, quantifying thus its ability to match the demand. However, as discussed before, the relatively coarse times scales of the traditional cloud operation prevents a direct mapping to our vision, where VNFs operate on much shorter ones. In the following, we provide some considerations on the required space of metrics to quantitatively characterize these new VNFs.

In general, the resources supporting the execution of a VNF are a heterogeneous set (e.g., Central Processing Unit (CPU), Random-access memory (RAM), spectrum, transport bandwidth), thus care should be taken when measuring them or varying their

³Elasticity is also related to resiliency, scalability, and efficiency, but with key differences: resiliency is the ability to recover from failures or to adjust easily to them, but it does not deal with efficiency; scalability is the ability to meet a larger load demand by adding a proportional amount of resources, but it does not consider temporal aspects of how fast and how often scaling actions can be performed. Finally, although a better elasticity should result in a higher efficiency, the opposite is not necessarily true.

availability performing experiments, to perform fair comparisons. A VNF should be characterized by its *minimum footprint*, defined as the minimum combination of resources needed to provide any output. During its regular operation, the *footprint* is the set of percentages of time the resources are occupied because of the execution of a function.

Under ideal circumstances, i.e., no shortage or variation of the resources available, a cloud-aware VNF has to operate as reliably as a traditional network function. With this being the benchmark, we can define *reliability* as the % of time that a VNF is providing the expected output. However, while in the traditional approach this reliability referred to the availability of a communication resource (e.g. “five nines reliability”), in our vision there are more categories of resources that impact performance apart from congestion or link degradation.

Arguably, the most distinct feature of an elastic VNF is how it relates the above two points, i.e., the shape of the function that maps the available resources to the obtained outputs. This *degradation function* characterizes the way in which performance degrades as resources lack, and depending on its actual shape we could characterize different quantitative behaviours: e.g., a *graceful degradation* might be defined by a % decrement of resources causing the same or a smaller % of reduction in performance.

Additional aid in achieving resiliency can be obtained via orchestration mechanisms, that horizontally (up/down) or vertically (in/out) scale the containers executing the VNFs, but usually require a relatively long time scale to operate. For that reason, not all VNFs could easily “be rescued” in cases of low resource availability. The *rescuability* of a VNFs is hence its capability of overcoming an outage by providing a limited degradation, until new resources are available.

4.2. The case for Serverless mobile networking

As discussed in the previous section, there is a wide consensus among the research and industrial communities that future mobile networks will be software networks, due to flexibility and cost reasons (in fact, some functionality such as the Evolved Packet Core is already provided in specialized software running over general-purpose hardware). However, we still lack the ability to match the network demand at any point in time, i.e., a technology that is *i*) re-configurable over very fast periods, and *ii*) very granular, to reduce the cost of inaccuracies in the re-configurations in e.g. the access network [47].

A similar problem has already been tackled by the cloud computing community, which has continuously provided faster and more scalable solutions over the last decade. Additionally, these solutions have also made the system more flexible and open, enabling the appearance of new business models. In what follows, we will provide a quick overview of the evolution of cloud computing technologies and a review of the current status of network functions softwarization.

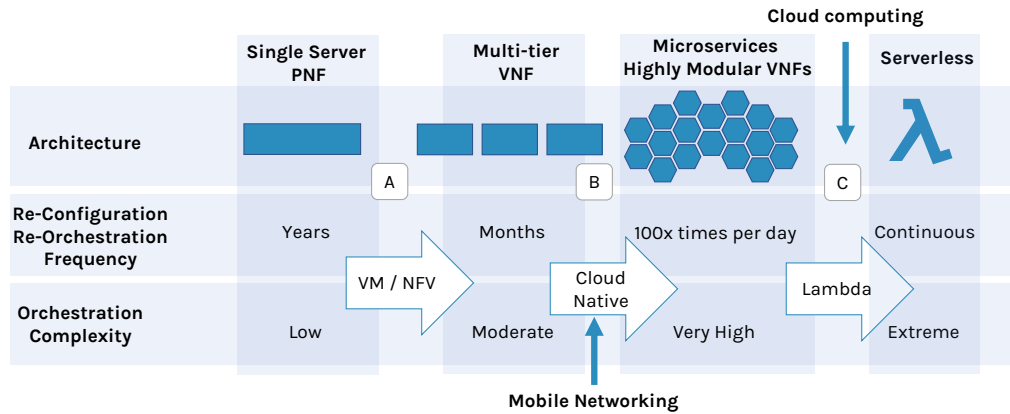


Figure 4.4: Major transitions in the adoption of softwarization.

Evolution of Cloud Computing

One major achievement in cloud computing took place in the early 2000s, with the appearance of new virtualization solutions such as Xen, VMWare or Kernel-Based Virtual Machine (KVM). With these technologies, that efficiently exploited the novel virtualization extensions supported by the hardware, a new way of providing services “conquered” the cloud computing environment. It consisted of a more modular architecture that supported a higher re-configuration frequency but also required a higher management complexity. This achievement is marked with an ‘A’ in Fig. 4.4, where we illustrate the different transitions that we considered along three dimensions: architecture, re-configuration frequency, and complexity. For this first transition, the figure illustrates how the architecture evolved from monolithic functions to modular ones, supporting a change of operational timescales from years to months, but also increased complexity in the operation.

The second transition that we identify is marked with a ‘B’ in Fig. 4.4 and happened in the early 2010s. It was caused by the arrival of the so-called microservices paradigm [48], introduced by software architects to support a much finer granularity. This paradigm supports, for instance, that a database server can be split into many tailored microservices, each one fulfilling a specific functionality such as e.g., an account manager or the data storage system. This transition is driven by the availability of new virtualization technologies, such as Docker and Linux Containers (LXC), which allow the deployment and scaling of small virtual applications in a much more lightweight fashion, enabling also new coding practices such as DevOps [48].

Finally, the last transition that we can identify is the one marked with a ‘C’ in Fig. 4.4, namely, serverless architectures [49]. This recent paradigm, also known as Function as a Service (FaaS), is an extremely liquid approach to scalability and resource usage. With this approach, a tenant creates calls to functions, i.e., the minimum building block of a software component, which are served by the infrastructure provider. In this way,

the software component becomes both platform- and server-independent, as the different functions of the same program could be served by different providers.

Evolution of Mobile Networking

There is currently a huge research effort on the softwarization of the mobile network. Among other efforts, 5G mobile communications are working towards the introduction of a fully softwarized architecture [50]. However, as compared to the cloud evolution, the telecommunications world is still half-way in this transition, despite the adoption of technologies such as Software Defined Networking (SDN) and Network Function Virtualization (NFV) which have helped towards the “softwarization” of network architectures, and the architectural trend towards their “modularization,” with a clean separation between the control and the user planes. That is, the trend is to split, already at the architectural level, the formerly monolithic nodes into several smaller logical entities. Thus, the Next generation Node B (gNB) can be split into centralized and distributed units, denoted as gNB-CU and gNB-DU, respectively [51], while core network components grow both in number and in functionality.

As depicted in Fig. 4.4, the telco world is lagging in the adoption of novel software paradigms. We are approx. at mark ‘B’ of cloud computing, with achievements such as:

- The standardization of 3GPP Release 15 [52], which specifies the service based architecture (SBA). This represents a new paradigm for the 5G Core Network and is driven by the trend towards the modularization of the network. With this approach, the formerly static interface between different elements has evolved into a flexible bus, which hosts HTTP REST primitives between modules.
- The concept of Cloud-Native Network Function (CNF), which is making its way into the current technology. In fact, there are already proposals for the design of cloud-native Virtual Network Functions (VNFs). However, they are in a very early stage and mostly involve Core Network VNFs only. We believe that the softwarization paradigm change shall involve all domains, including the most challenging one such as the Radio Access Network (RAN) (as exemplified in Section 4.2.2).

Despite these achievements, we are still in the middle of this transition as the cloud-native paradigm has not been fully adopted into operational networks. This is caused by the poor agility of the current state-of-the-art solutions, and the fact that current VNFs are not truly agnostic to the underlying NFV infrastructure. While dynamic cloud resources orchestration algorithms are currently under study [53], the VNFs that will be running on such resources are still not optimized for this type of operation.

So even if the efforts towards the cloud-native transition of the NFV are still ongoing, the research community shall prepare for the next transition. This will introduce a

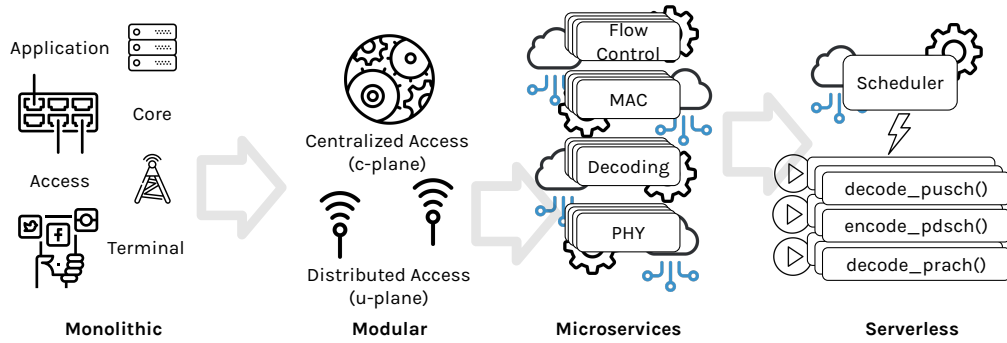


Figure 4.5: Mobile network architecture evolution.

complete re-design of the whole mobile protocol stack, which will certainly facilitate a dynamic resource orchestration and assignment, allowing hence higher efficiency.

We then advocate for a mobile protocol stack that is (i) more efficient in terms of both resource and time granularity scalability, and (ii) capable to elastically adapt to the instantaneous demand. We claim that with such a protocol stack, the deployment and operational costs of the network will be reduced to their minimum. Given that this flexible and on-demand operation of the network closely resembles the current operation of cloud computing platforms, it should also follow similar principles, hence the name *Serverless Mobile Architectures*. We next formally introduce this concept and its potential advantages.

4.2.1. Serverless Mobile Architectures

We start by describing what we mean by serverless mobile network architectures. To this aim, we illustrate in Fig. 4.5 the evolution of the different architectures to support a mobile service. Note that throughout this section, we use RAN functions as examples, as they provide the most difficult scenario for serverless architecture given their tight execution constraints. The leftmost subfigure depicts the traditional **monolithic** paradigm (e.g., 4G networks), where functions are implemented in specialized pieces of equipment. In this case, software and hardware are tightly coupled, and it is not uncommon that different functions are indissolubly associated to the same piece of equipment, e.g., the Serving Gateway (S-GW) and Packet data network Gateway (P-GW).

The next subfigure illustrates a **modular** network architecture, represented by the Cloud-RAN (C-RAN) paradigm, where some control functionality traditionally associated with the antenna (i.e., the scheduling algorithm) is re-located to a central server. This change constitutes a shift from the monolithic approach, with some functions “freed” from their traditional association to monolithic pieces of hardware. These functions are now logically different pieces of software, whose execution can be placed in different parts of the network. As discussed before, this approach is also tackled by the architectural

work, with the definition of standard interfaces among well-defined elements (e.g., the F1 interface between the gNB-CU and gNB-CUs).

The **microservices** architecture pushes the modular paradigm further, by decomposing the building blocks into sub-modules. Note that this is a logical division and that the actual implementation of the architecture needs to accommodate e.g. specific use-case requirements, this eventually resulting in fewer or more pieces of software.

For the case of the RAN, this results in the protocol stack now being logically divided into physical layer processing, decoding, encoding, MAC, flow control, etc., each of them running in an independent execution environment and connected through synchronization APIs. This allows an easier scaling over a finer resource assignment strategy, which eventually leads to better resource utilization. Furthermore, some very recent proposals are pushing for microservice-based core network functions [54], showing that this increased modularity in the VNF design is catching momentum.

Given that said, we advocate for a **serverless** mobile architecture composed by atomic functions that can run independently on a cloud infrastructure. This independence contrasts with the tight coupling across functions in the other architectures, with strict timing considerations between modules. In a serverless approach, functions are disaggregated from the main scheduling logic and executed in the most appropriate server available. As Fig. 4.5 illustrates, for the case of User Plane Functions we envision, for instance, that the decoding of different Modulation and Coding Scheme could be made by different functions that could run in different executors, provided that some “loose synchronization” is guaranteed.

4.2.1.1. Advantages

Introducing the serverless operation as explained above brings several advantages to the network operation. Next, we build and extend the reasoning introduced in [49] to motivate the advantages of serverless mobile networking.

No server management: in the serverless paradigm, the functionality carried out by a VNF is broken into very fine execution environments (i.e., functions) that do not need to directly undergo into the classic lifecycle management (instantiation, run-time and decommissioning), but rather be scaled according to the real load and with a very fast pace in a “message broker” fashion. By moving this complexity to the network orchestration, this allows increasing the commodification of the network with a clear separation between the infrastructure and the services orchestrated therein.

No idling: operators usually provision the network based on the peak load. This is very inefficient at all network layers: at the access level, needless to say, but also at more centralized levels in which VMs or containers may be underused or even idling in trough

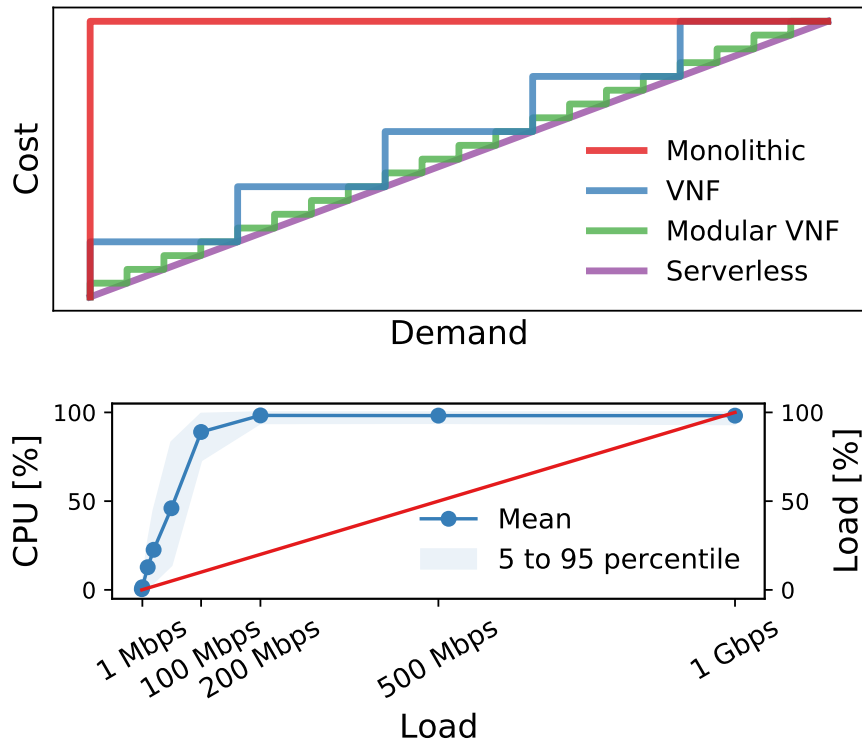


Figure 4.6: The liquid scalability (top) and an empirical evaluation (bottom)

loads. With the serverless paradigm, execution engines are spawned and operated just when and where they are needed. This is key for minimizing resource wastage in the network operation.

Liquid scalability: this is achieved by providing the highest *modularization* level. That is, specific functions of a VNF can be scaled according to the real demand, avoiding the scaling of the full VNF instead, and achieving the liquid scalability depicted in Figure 4.6 (top). We believe that this will be one of the fundamental pillars for the sustainable operation of the next-generation networks. In [47], the authors quantified the cost in terms of resource overhead of deploying and operating the infrastructure needed to support multi-service networks. This study showed that the efficiency (i.e., the number of resources used by a not multi-tenant network compared to a multi-tenant one) is very low, and just with a very dynamic network reconfiguration (such as the ones envisioned here) it is possible to improve these figures.

Figure 4.6 (bottom) depicts an empirical evaluation of the liquid scalability concept, by evaluating the Central Processing Unit (CPU) footprint of a state-of-the-art VNF (OpenVSwitch [55]), running inside a KVM Virtual Machine in Linux. This setup summarizes the “VNF” approach sketched in the upper part of Figure 4.6 as it can be

scaled on a VM-basis (new Virtual Machines (VMs) can be added or removed according to the load), this being the only way of scaling up or down according to the load. We can observe that this way of softwarizing clearly falls short when the objective is finer scalability. We can observe that while growing the offered load (in our experiments we span 4 orders of magnitude, from 100 Kbps to 1 Gbps) the resource footprint utilization has a clear “on-off” behavior. Indeed, the CPU utilization shortly hits close to 100 % utilization with offered loads that barely reach the 10% of the total. This means that there is still a lot of room for improvement in the software design of VNF with respect to IT resources consumption.

Pay-per-use network: although the pricing model behind the network slicing paradigm is not clear yet, it is to be expected that, at least for the software part, it will follow a classic approach in which tenants are charged on the number of CPUs, the amount of memory and bandwidth used. With a serverless approach, instead, tenants can be charged on a specific usage basis (i.e., number of times and duration of each function), allowing for a richer pricing model.

Customization: current mobile network technology provides only limited customization. For example, the currently envisioned resource models in 3GPP [56] target the Network Slice as a Service (NSaaS) paradigm, which is the telecommunication counterpart of the well-known Software as a Service (SaaS) paradigm employed in the cloud computing world. Under this model, service providers (or tenants) are allowed to select from an operator Network Slice portfolio one of the available templates (e.g., Enhanced Mobile Broadband). However, this provides limited customizability to tenants: the network provider still handles most of the management part.

4.2.2. Challenges to Address

To achieve the above advantages, the serverless paradigm needs to provide: *(i)* new VNFs that allow for the wire speed execution VNFs while minimizing the number of resources needed for their operation, *(ii)* a new environment for the execution of such challenging VNFs, with minimal overhead, and *(iii)* a new Management and Orchestration framework that is capable to manage the rocketed complexity of the paradigm.

Challenge #1: New VNFs

The current way of implementing VNFs is still very bound to the traditional way of implementing network functions. Current solutions do not embrace modularization: many commercial products are softwarized but very bounded to the hardware platform, while open source initiatives are practically mere translations of hardware functionality

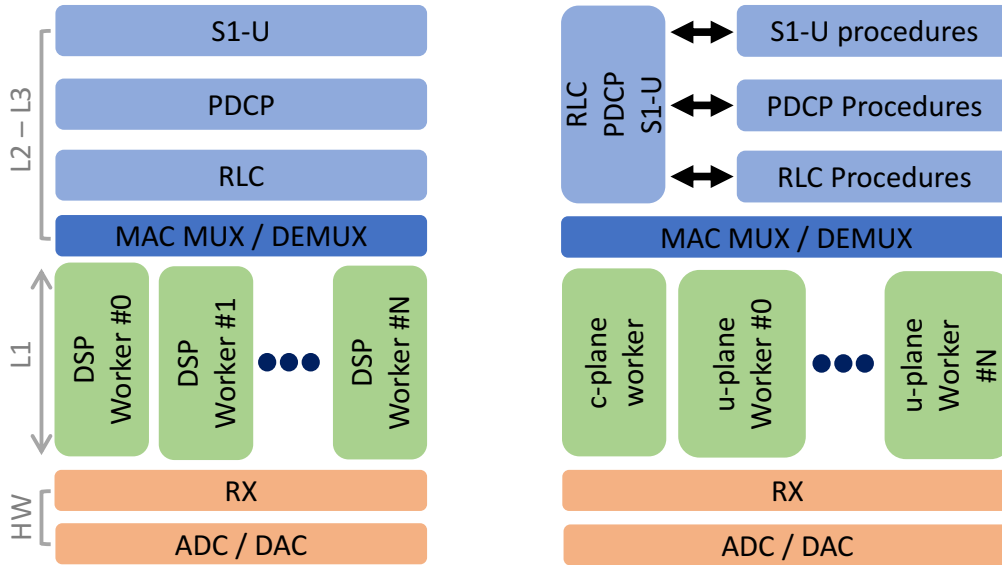


Figure 4.7: The simplified threading architecture of the *srsLTE* software (left) and a possible serverless design of the software stack (right).

into software modules. To adopt the serverless approach, we need to change how VNFs are designed. To illustrate our point, in the previous section we focus on the user plane part of the RAN, but the same paradigm could be easily applied to other network functions. Our motivation is that, as the radio functions are the most resource-consuming ones (considering resources of all kinds: spectrum, transport network, and computational resources [57]), we expect that the transition towards high modularity will be especially beneficial for such functions.

We take as exemplary case study a well-known open source implementation of a 4G-LTE RAN stack, namely, *srsLTE* [15].⁴ We illustrate on the left part of Fig. 4.7 its threading architecture [58], which follows a *classical* layered architecture for a great extent and has remarkable modularity, in particular considering that the software has been designed for stand-alone operation. Still, the division is quite coarse, and there are additional issues that would prevent the use of a serverless approach, e.g., the physical layer does not distinguish between control and data channels, which are processed sequentially.

In this way, while the *srsLTE* design is perfectly valid for the working conditions initially considered by its developers, the architecture would benefit from a different design such as the one suggested in the right-hand part of Fig. 4.7. Here, the control and data plane processing are placed in different modules: while the control plane functionality on the L1 is “fixed” and has to be performed independently of the load, the user plane could be placed and executed in different threads (or even containers) to scale them according

⁴<https://github.com/srsLTE/srsLTE>

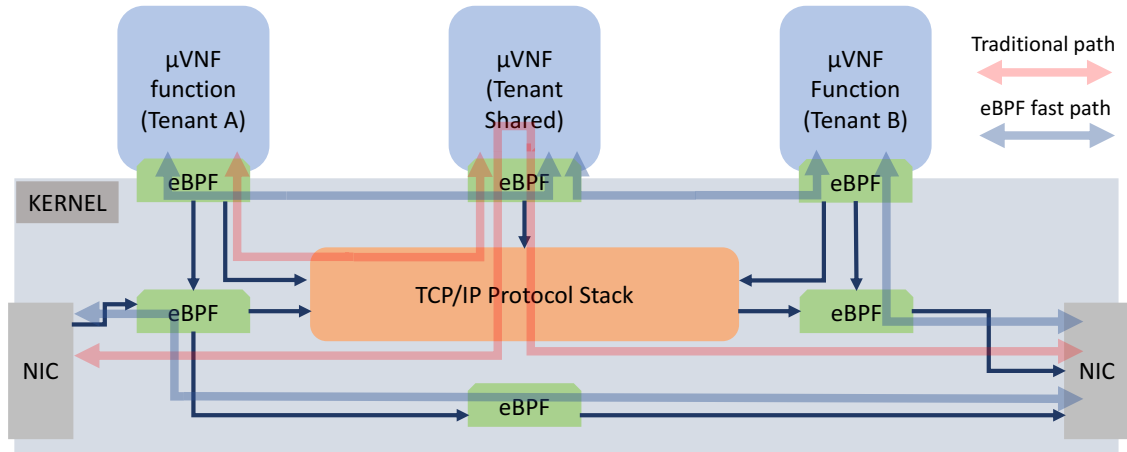


Figure 4.8: eBPF-enhanced data path vs the traditional iptables-based approach.

to the load. Alternatively, the software can be modified to support intelligent resource assignment schemes [58].

However, the current state of the art of networking (i.e., the Linux kernel) can hardly support this approach, as it is heavily API-based and requires fast inter-process communications. Moreover, current network functions (especially RAN functions) exhibit two fundamental characteristics that make them different when compared to other cloud computing applications: *i*) they impose very high load on the CPU (i.e., encoding and decoding of wireless frames) and *ii*) they have very stringent timing requirements as usually communication protocols are time-partitioned and need time synchronization. While there are CPU intensive services running in the cloud (e.g., Netflix video transcoding) these are not real-time. These timing constraints were barely a problem in the (now old) Physical Network Function (PNF) paradigm, but it is a challenge to address since the arrival of novel technologies such as C-RAN.

Making tasks aware of their execution time is usually not conceived as a problem in general cloud computing systems, which barely have to provide near to real-time outputs. The Linux kernel system provides tools to address this problem, but real-time software usually runs in rugged embedded devices for industrial purposes (e.g. robotics) or in dedicated data centers for fast pace trading in stock exchanges, not in the cloud. So, VNFs shall be re-designed to also take advantage of e.g., the real-time kernel primitives or used jointly with the elastic network function design as investigated in [57]. In Part II we re-define multiple network services as well as in Part III, a complete re-design of the RAN is performed to properly fit the ideas introduced in Sections 4.1.2 and 4.1.1.

Challenge #2: Scalable interconnections

One key requirement for novel deployments, given the trends of mobile data consumption, is the ability to operate at wire speed. Being extremely fast on the data plane has been one of the main goals of research in wireless communications. However, the original virtualization platforms were not designed with this goal in mind. To address this issue, the most common approach to achieve high performance has been *kernel bypassing*, through technologies such as Data Plane Development Kit (DPDK) and Single Root I/O Virtualization (SR-IOV). These solutions skip the traditional Linux kernel networking stack based on `iptables` and enable a fast data path between the hardware and the application logic residing in the user-space. However, while this approach indeed increases the speed of the data processing, this hard link between the network card (i.e. the hardware layer) and the user-space reverts this approach to the traditional monolithic one, that leveraged on a tight hardware-software link. This hinders programmability as all the data traffic is offloaded from the kernel to the user-space, creating a vendor-specific link between hardware and software.

The above approach makes the management of the VNF very machine-dependent, so it is only valid for scenarios with a relatively small number of VNFs. Furthermore, it also has several drawbacks that make it difficult to integrate into a highly-dynamic scenario with a much larger number of software components (for each tenant, slice, and service, there might be multiple software functions). This would drastically increase the complexity of this kernel bypassing approaches, eventually resulting in two hardware infrastructure layers, namely, the kernel and the networking bypass, to manage the computing resources and the network platform, respectively.

We propose instead to integrate the data path back into the kernel. To this aim, i.e., to avoid the limitations from building on `iptables`, we propose to extend the Linux kernel networking stack with the adoption of enhanced Berkeley Packet Filter (eBPF) [59]. These are pieces of code that can be dynamically injected into the kernel at run-time through a programmable interface. This approach allows managing the VNFs running on top of the kernel holistically, controlling all the aspects such as their CPU, memory, etc. in a unified way. eBPFs fit with a lightweight container-based approach, enabling strong security policies among them.

We illustrate our proposal in Fig. 4.8, where the bottleneck caused by the slow `iptables`-based design of the TCP-IP stack is removed thanks to the use of eBPFs. With these, a fast mesh of interconnected elements is created, providing an extremely scalable network infrastructure within the same host. Moreover, as Berkeley Packet Filters (BPFs) are (i) executed in native code (e.g., x64), (ii) if needed, placed close to the Network Interface Card (NIC) to ensure fast reaction times, and (iii) programmed with fast memory mapping to the VNFs, the highest possible speed is achieved, comparable to DPDK according to recent studies [60].

Challenge #3: Precise orchestration algorithms

The serverless paradigm aims at the most efficient service provisioning, by accurately adjusting the resources deployed at any point in time to the actual demand. To benefit from this paradigm, it is essential to accurately estimate the demand required by a service and to forecast its envisioned resource consumption, to boost the multiplexing gains. To support this type of management, two main building blocks are required: (i) technical solutions to support flexible and fast resource re-orchestration at the finest granularity, and (ii) Big Data techniques that operate on historical data and anticipate future trends. The former should be achieved with the first two challenges (i.e., the use of functions instead of VMs and the integration of eBPFs into the kernel), while the latter requires the design of new techniques, as we discuss next.

We propose to use data-driven techniques to accurately characterize the future demand trends for a given service, this supporting a proactive and fine-grained orchestration of the network. This proactive management contrasts the current state of the art of resource orchestration, which is usually a reactive process based on load thresholds to trigger the scaling of VMs, and typical prediction algorithms that operate on very coarse and aggregated data. Moreover, the main goal of existing orchestration algorithms is usually fault-tolerance or self-healing, and not a higher efficiency. An efficiency-driven orchestration framework would provide an accurate forecast of demand, both in time and across resources (e.g., antennae, edge data centers, core clouds), to enable efficient provisioning of network services. Given the complexity of this type of operation, we believe that such orchestration is only possible if empowered with deep learning solutions [61].

The orchestration would work as follows. By starting from the historical demand of a given tenant/service, a deep learning architecture would provide the intelligent back-end for the Management and Orchestration of the network. By employing a deep learning technique, orchestration decisions are applied to the underlying NFV infrastructure that is used to host the VNF. This approach is currently being investigated by ETSI ENI [62] from the architectural point of view. In Section 4.3, we study the advantages of flexible orchestration of VNFs, the challenges to be tackle and how the available solutions deal with the identified challenges.

4.3. Flexible re-orchestration of VNFs

Network orchestration [63,64] can be defined as the coordination between the hardware elements of a given deployment and the software modules running on top of them. Current orchestration solutions only support a static and coarse-grained operation: once instantiated, it is hard to modify the resources associated to a specific network slice (e.g., re-locate a Virtual Network Function (VNF) to the edge), or to support a fine-grained re-configuration (e.g., scale-up just the flows belonging to a specific network slice). We

define a **flexible network orchestration** as the one supporting a dynamic and fine grained operation. These characteristics would enable the so-called elastic orchestration of network slices [65] which, in turn, would improve the resource utilisation in the network.

As defined above, a re-orchestration solution is flexible only if it is both dynamic and fine-grained, features which are currently unavailable with existing orchestration solutions (we discuss these solutions in Section 4.3.2).

Some of these advantages obtained with a dynamic re-orchestration are:

Adapting to user mobility. Low-latency services such as tactile or vehicular communications require that VNFs affecting latency are as close as possible to the user, to minimize the delay between these functions and the user. When a user moves to a new location, VNFs should move as well to keep close to the user's new location. This requires the ability to relocate those functions without disrupting the ongoing service.

Service enhancements in run-time. Flexible re-location also gives the ability to re-compose a service provided by a given slice, to add, substitute, remove, or relocate VNFs in the chain. This enables introducing a variety of features during run-time operation, such as, e.g., adding or relocating a firewall, or replacing a more efficient (but slower) video encoder by a quicker but less efficient one to adapt to changes in the measured delay while keeping quality of experience.

Improved de/scaling. Resource scaling refers to the ability to assign resources as needed. While the *traditional* vertical and horizontal scaling could provide this feature to some extent, the use of relocatable VNFs introduces an additional level of flexibility without disrupting the service: when a VNF runs out of resources, it can be relocated to a different location with more resources. When few functions are running in different locations, this allows to relocate them in a single resource and deactivate the unused nodes, saving resources by implementing infrastructure on-demand schemes [66].

Resilient operation. The ability to relocate VNFs in real-time enables novel methods to provide resiliency. In case of service disruption due to, e.g., the congestion of a node, or a hardware failure, it would be possible to relocate the required functions seamlessly trigger their "activation", thus providing resilience against impairments of various kinds.

4.3.1. Advantages of a fine-grained re-orchestration

The transition to a more modular and software-based architecture such as the one in the 3GPP Release 15 [67] opens the door for a more precise resource management. Also, the Application Programming Interface (API)-based control of the core network function (the so-called service based architecture (SBA) architecture) allows for an easier way of re-configuring functions, following the slice needs. Among the features that such fine-grained re-orchestration capabilities would enable, we have:

Per-slice re-configuration. Network slices (or Sub Network Slices [68]) are the “least common multiple” when it comes to service management. As VNFs can be shared among slices [68], they should expose APIs that enable the per-slice configuration. Besides the per-slice parameter re-configuration, the orchestration framework shall also support operations such as join and split, i.e., grouping into the same virtual instance (a Virtual Machine or a container) a group of VNFs belonging to different slices, and vice-versa.

Joint parameters and resource configuration. Modifying a parameter of a given VNFs may have an impact on its resource footprint, and also on the one from other VNFs, both from the network resources perspective (i.e., more or different frequency bands) and the computational (i.e., more Central Processing Unit (CPU)). An orchestration algorithm shall be able to assess the impact of a change of parameters on the underlying infrastructure and act accordingly.

Access network re-configuration. While the core network functions already have incorporated softwarization principles since the standardization, access network functions are more “grounded” in a less flexible architecture, which is partly also due to their need to comply with stringent timing requirements. Just very recently, industrial fora such as Open RAN [69] started to advocate for a finer programmable management of the radio access. Such concepts shall be incorporated in the orchestration framework.

4.3.2. State of the art solutions

Despite the advantages discussed above, the technology currently available does not support a flexible re-orchestration of VNFs. In the following, we review the state of the art, highlighting the most relevant initiatives and contributions.

General VNF placement and orchestration problems

There is a bulk of literature available on the problems of VNF placement and orchestration, summarized by a number of surveys, e.g., [70–72]. In general, the different proposals can be classified depending on various axes: (*i*) the variables to be optimized,

e.g., power, cost, latency; *(ii)* if the optimization is mono- or multi-objective; and *(iii)* whether the matching of physical and virtual resources is carried out in an offline manner (gathering inputs, requirements, etc.) or in an online manner, following, e.g., the crossing of a threshold, or a periodic trigger. It should be noted, though, that even if the approaches falling into this latter category are referred to as “dynamic” in [71], these solutions are not tested in scenarios considering quick variations over time (see e.g. [73]).

In fact, despite this remarkable amount of previous work, actually few proposals deal with the implementation of such algorithms on real Virtual Infrastructure Managers (VIMs), with the use of real-life traces being among the most common approaches for the performance evaluation. Furthermore, for those proposals performing a real-life evaluation, they typically rely on existing orchestration technologies that, as discussed in the next section, lack both the dynamism and granularity required for what we refer to as a flexible re-orchestration (i.e., dynamic and fine-grained).

General-purpose orchestration technologies

Existing Network Function Virtualization (NFV) Management and Network Orchestration (MANO) software solutions such as, e.g., Open Source MANO (OSM) [74] or the Open Networking Automation Platform (ONAP) [75], are continuously evolving solutions used in many fields to manage the VNF lifecycle (design, configuration, termination, etc.). Their interactions with the underlying infrastructure (to instantiate, connect, and terminate virtual resources) are done through a VIMs, a software element that abstracts the complexity of the cloud. To enable the discussed advantages of a flexible orchestration of network slices, this VIM has to support *(i)* flexible relocation of virtual resources, and *(ii)* their fine-grained re-configuration.

On the one hand, a fine-grained orchestration is tough: state-of-the-art orchestration platforms only allow to re-configure very basic parameters such as the IP address of the VNF, while other fine-grained parameters (such as the ones described in the Information Model [76]) are left to the implementation of each VNF.

On the other hand, VNF re-location technologies are also lacking in terms of dynamism. Although existing VIMs can relocate a Virtual Machine (VM) from one compute node to another, this operation has notable limitations:

- They especially target limited parts of a VM such as its memory. These techniques use an iterative process [77] that starts from the memory pages that were the least frequently accessed, keep updating them until the ones that are the most used are moved. While relocating memory, usually a significant part of the VM has to be kept in a fixed location (e.g., a NAS hosting the disks). As a result, the relocation capability is limited.

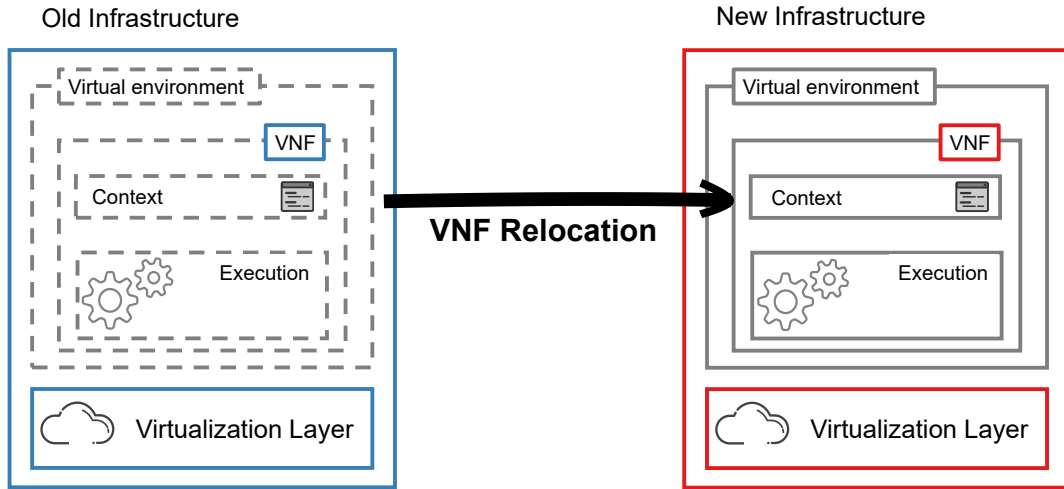


Figure 4.9: Full VNF relocation

- The relocation of a VM is limited within the boundaries of a single datacenter of a single VIM. These limitations are acceptable for cloud computing environments, which typically focus on very high reliability and therefore VMs are only relocated in case of, e.g., disk failures or programmed maintenance, but are inadequate for dynamic scenarios such as the use cases discussed above, involving the movement of VNFs across the network to reduce latency or to improve efficiency across datacenters. As a matter of fact, the topic of migration over WAN links (which is a relevant scenario for networking purposes, e.g., migration to edge cloud) is currently overlooked by the bulk of available literature, as also confirmed by the authors of [78]. Among the more than 200 works reviewed there, just a handful deal with migration over long distances and none of them provide experimental results.

As discussed, state of the art orchestration platforms provide some methods to relocate VMs. For instance, OpenStack provides live-migration tools [79]; however, their use precludes fast VNF re-location. Their operation is sketched in Figure 4.9: the full Virtual Machine has to be copied to the targeted destination. This includes common data such as the guest kernel, libraries, and the file system structure [77]. Furthermore, as these techniques are very expensive in terms of exchanged data, they are only available between the same NFV infrastructure point of presence (i.e., the infrastructure controlled by the same VIM instance), excluding thus the migration among VIMs.

Moreover, commercial products such as *VMware* implement sophisticated techniques that can perform live migrations in very short times, by incrementally copying the memory of running virtual instances. However, these methods require very high bandwidth and a very short latency between the endpoints, as well as a shared disk image. The technical report from *VMware* [80] declares migration times in the order of tenths of seconds over

a 10 Gbps Ethernet connection. All these requirements preclude their use in our target scenario, which should support re-locations between endpoints relatively “far away” (e.g., different datacenters). Similar techniques are also employed in the context of containers, e.g., Voyager [81]. Besides being substantially lighter than a VM, this technique however still has to copy all the memory and the disk used by the container, making it unsuitable for far re-locations.

Similar considerations apply for other virtualization platforms that are particularly optimized for the VNF migration. For instance, `unikernels` can perform live migration within few milliseconds [82], but they are currently not part of any large scale NFV infrastructure deployment and therefore they are not integrated into commonly used MANO platforms such as ONAP or OSM. Because of this, they lack the required infrastructure management capabilities, and therefore they are unsuited to accommodate basic features such as i.e., re-orchestration triggers in a seamless way.

Ad hoc solutions

Enabling flexible re-orchestration in softwarized network deployment received attention by the research community in the last few years. The work most closely related to ours is `SENATUS` [83], a framework that internally leverages on state of the art VIMs; because of this, this framework yields to very poor performance, in particular in challenging (i.e., very dynamic) scenarios, as we quantify in Section 7.3.

Finally, if we consider more targeted solutions that also specifically include the access network, the available material is even less. The most remarkable solution is Orion [20], which allows for a per-slice re-configuration of radio resources but the software is not freely available.

5

Summary Part I

In this part, we have introduced 4th and 5th generation of mobile networks with their main features and building blocks. Then, we moved to the key enablers that provide the required flexibility to 5G network infrastructures. Network Function Virtualization (NFV) is a network architecture concept that uses virtualization mechanisms to decouple network functions from dedicated hardware components. Then, we talked about network slicing, which defines a network architecture that enables the ability of running multiple logical virtual networks on the same physical infrastructure, keeping characteristics such as isolation. Lastly Software Defined Networking (SDN), which propose a network architecture approach where management and flexibility are improved by splitting control and data plane.

Next, we have identified the *cloudification* and *softwarization* trends that mobile networks will follow during their evolution, clearly identifying a parallelism between cloud and mobile networks evolution. Moreover, we have proposed the *serverless* mobile architectures, where we propose going one step further on networks cloudification by composing a mobile architecture with atomic functions that can run independently on a cloud infrastructure, removing the tight coupling in other architectures.

We have identified mainly three challenges: (i) Softwarization of network functions; (ii) the need for flexible orchestration; and (iii) cloud-aware virtual Radio Access Network (RAN). In Part II we will go through softwarization of network functions and flexible orchestration, going one step forward to solve this challenge. Then, in Part III, we will face the design of a cloudified RAN, going again one step further in the resolution of this challenge.

PART II

BRINGING NETWORK SLICING TO SOFTWAREZIZED MOBILE NETWORKS

6

Design and Implementation

Given the current network softwarization trends (see Chapter 2), the benefits of network slicing and cloud-aware architectures are clear, and there is a wide consensus among the industrial and standardization communities on the need to adopt this technologies. However, we lack of experimental validations on its effectiveness e.g., on the gains when using different orchestration mechanisms, or under different traffic scenarios or different virtualization infrastructures.

While there are implementations for some of the enables such as network slicing, to the best of our knowledge, there is no open source solution that comprises neither end-to-end network slicing nor cloud-native architectural approach. More specifically, virtualization is a mature technology that has been extensively used for wired elements, with technologies such as e.g. OpenStack and Kubernetes for virtual machine and containers management, respectively. However, the situation is less mature for the wireless access part, being Orion [20] among the few proposals to implement slicing at the Radio Access Network (RAN) that have been tested in practice, without any alternative when talking about cloud-native Radio Access Network (RAN) approaches.

In the following sections, we fill this gap with the design, implementation and experimental evaluation of an end-to-end network slicing solution together with a complete framework to provide flexible orchestration, following the path towards a cloud-aware mobile protocol stack (see Section 4.1)

6.1. POSENS, a Practical Open Source Solution for end-to-end Network Slicing

The architectural design of network slicing has been widely studied in the literature and standardisation institutions. These works precisely expose how network slicing must be implemented to effectively provide the benefits it was thought for. In the literature, there are mainly three architectural options that have been proposed for RAN Network

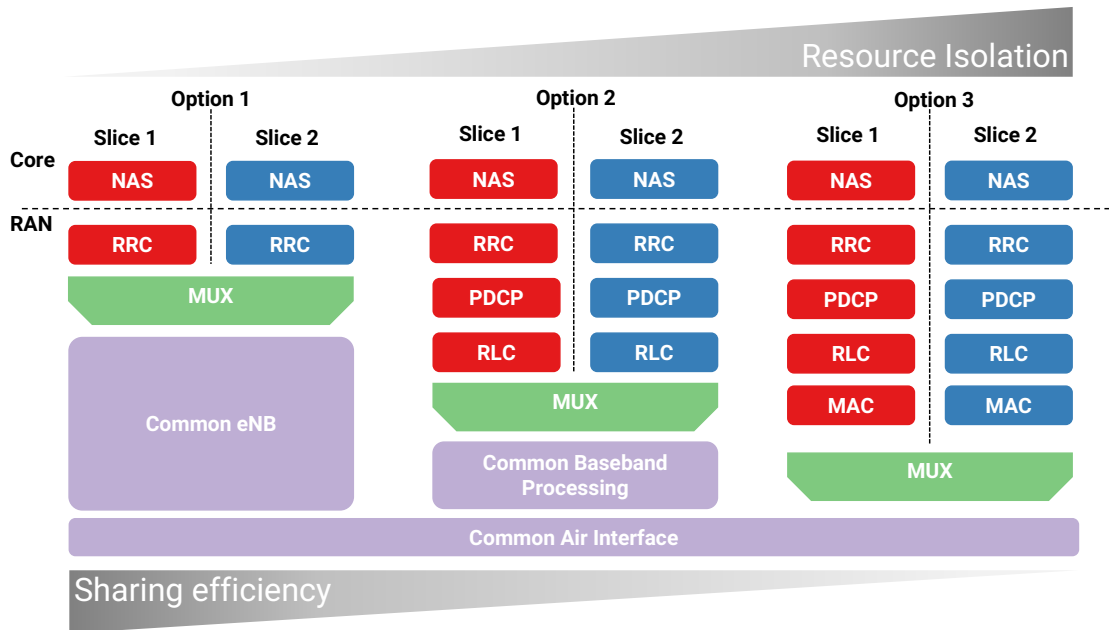


Figure 6.1: Different RAN slicing options.

slicing. These options are presented in order of “increasing depth” in Fig. 6.1, where the deeper the slicing (the “MUX” block represents this depth), the less functions are shared by different tenants.

The leftmost option (“**Option 1**”) is the so-called **slice-aware shared RAN**, which basically consists in sharing the complete RAN, and then each tenant is responsible for its Core Network (CN). With this option, the same User Equipment (UE) can use different slices, and therefore connect to different CNs. This solution, which can be considered as the “basic” solution to support network slicing, provides relatively little isolation across tenants, but also leads to the highest potential gains in terms of efficiency. This solution can be related with some current proposals such as 3rd Generation Partnership Project (3GPP) Long Term Evolution (LTE) Enhanced Dedicated Core Network (eDECOR) [84], introduced to support the instantiation of dedicated CNs. However, eDECOR requires introducing changes to the CN and new signalling messages for the connection setup (something that , as we will explain in the following sections, POSENS does not). We also remark that 3GPP RAN3 Working Group [85] is considering a functional split performed at this level. This approach nicely fits with the shared RAN slicing option, in which multiple network slices are handled by a centralized unit.

While with this option the RAN is shared to a large extent by different slices, the core instances are completely independent among tenants, allowing per-tenant configuration, orchestration and (cloud) resource assignment.

The central option in the figure (“**Option 2**”) is the **slice-specific Radio Bearer** configuration. With this option, the slicing goes deeper in the network stack, and basically only cell-specific functionality are shared, i.e., the Physical Layer (PHY) and Medium Access Control Layer (MAC) layers in the user plane, and the Radio Resource Control (RRC) in the control plane. This configuration increases the resource isolation between tenants, at the price of a higher complexity at the MAC layer (for instance, to fully exploit this resource isolation, slice-aware scheduling algorithms are required).

Finally, the last option (“**Option 3**”) is the so-called **slice-specific RAN**. In this case, only the air interface is shared among network slices, while all the other functionality is instantiated specifically for each tenant. This configuration provides the maximum degree of freedom, given that each network slice can be customized down to the physical layer. However, this option also requires a tight synchronization between the multi-tenancy policies implemented by a common part, and the per-slice (dedicated) implementation.

This option could be particularly useful in scenarios where different radio access technologies coexist within the same shared spectrum, e.g., 4G and 5G. Since it may be very challenging to dynamically reallocate spectrum resources at a fine time granularity, this option may potentially harm resource utilization and limit the potential multiplexing gains.

The above options can be regarded as a “roadmap” to enable a fully configurable protocol stack to support any network slicing option, where each option presents a different trade-off between efficiency, isolation and complexity. For the first release of POSENS, we decided to implement “**Option 1**,” which can bring the maximum efficiency gains and provides end-to-end slicing, in this way providing researchers with a tool to experiment with different algorithms and mechanisms. Although options 2 and 3 provide a higher degree of isolation between slices, “Option 1” already enables key features without requiring the complexity of more advanced RAN schema.¹

More specifically, this option readily supports experimentation on fundamental research items in 5G, such as *(i)* per-tenant Service Function Chain (SFC) creation and management, as the network slices flows go through chains that contains instantiations of different Virtual Network Functions (VNFs), or *(ii)* per-tenant orchestration, as different tenants can implement their own Management and Network Orchestration (MANO) using their preferred software on their cloud, enforcing thus service specific management and orchestration policies regardless of other tenants’ ones.

¹While Options 2 and 3 require very tight synchronization among slices, this is not an issue for Option 1 since it employs a conventional RAN stack that already provides the required synchronization.

6.1.1. Design of POSENS

After studying the available state of the art alternatives (Section 3), we now dive into the design decisions and details towards the proposed network slicing solution. POSENS provides an implementation of all the modules needed for an end-to-end network slicing-aware mobile network. This includes elements belonging to all the realms of a mobile network (UE, RAN and CN), plus an orchestration framework. Still, the most important enabler of an end-to-end network slicing setup is RAN slicing.

In the following, we describe the design of our solution to support RAN slicing. This solution consists on introducing a number of changes and new modules to the srsLTE UE and Evolved Node B (eNB) implementations. The resulting software architecture, for the case of two slices, is illustrated in Fig. 6.2, where each slice is depicted with a different color (we consider the case of two slices for simplicity, but the software can be easily extended to support more slices). We will also assume for simplicity that each slice is associated with a different tenant.

As previously discussed, we decided to implement in our first release of POSENS the “**Option 1**” for RAN slicing, where slices are multiplexed and demultiplexed at the Packet Data Convergence Protocol (PDCP) layer. This option has the additional advantage of requiring less changes in the eNB software implementation, which is the main cause of instabilities in a Software Defined-Radio (SDR)-based testbeds. The cornerstones of the solution are the “slice coalescer” modules, located at the PDCP layer and above. These modules forwarding the control and data layer information for each slice over the common communication channel. Another key feature of our implementation is that each slice at the UE has its own RRC module, and does not require any additional functionality inside the CN. Conversely, at the eNB there is only one RRC module, as with its default behaviour is capable of managing multiple Non-access stratum (NAS) from various users simultaneously. In what follows, we provide a more detailed description of the enhancements required by our solution, by describing the behaviour of the UE and the eNB.

User Equipment

The UE plays a fundamental role in the network slice selection procedure. As depicted in Fig. 6.2, one slice performs a full radio configuration of all the RAN layers (including PHY and MAC), while the other one relies on the RRC configuration parameters set by the first slice and prepares the PDCP entities and the Radio Link Control (RLC) channel managers (Acknowledged mode for the u-plane and Transparent mode for signaling messages).

Once the UE has been powered on, the (unmodified) PHY performs the usual cell search (following the configuration provided within the Master Information Block (MIB),

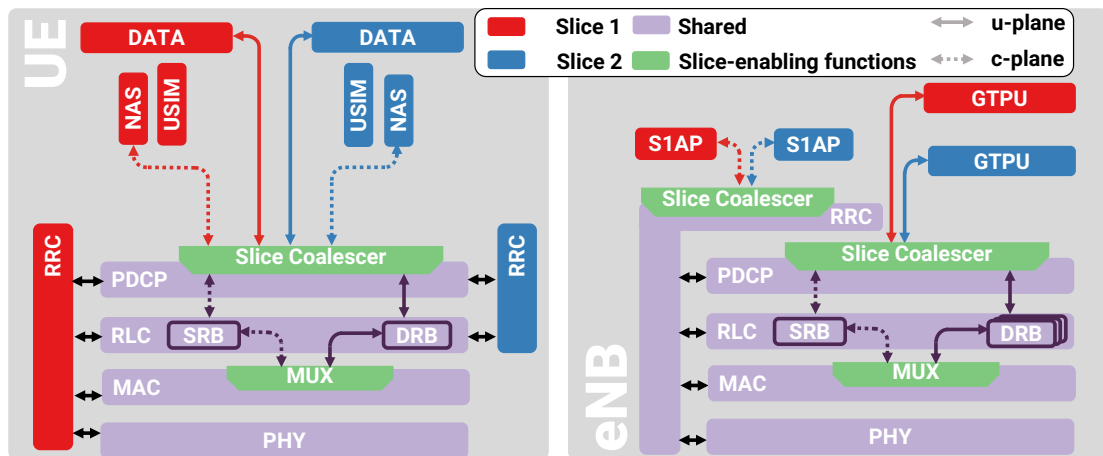


Figure 6.2: Design of POSENS: changes introduced at the UE and the eNodeB.

System Information Block Type 0 (SIB0), System Information Block Type 1 (SIB1) and System Information Block Type 2 (SIB2) messages) and synchronization. Then, the RRC module corresponding to the first slice sets up the initial connection with the eNB by performing the Random-Access procedure (RA) to get an initial Uplink (UL) grant, i.e., a valid configuration for PDCP, RLC, MAC and PHY. This configuration is shared across slices, and therefore subsequent RRC modules (corresponding to other slices) will not request it. This motivates that the `RRConnectionSetup` message that arrives during the RA process has to be stored within the PDCP module, for subsequent slices to be able to establish their session with the CN.

Following the initial UL grant, the NAS protocol of the first slice establishes a session with the CN, generating a `RRConnectionSetupComplete` message nesting the initial NAS messages in the same packet. The selection of different slices happens in a sequential fashion, after the first slice RRC has configured the wireless link, the subsequent slices are configured using the reception of a `RRConnectionSetupComplete` as triggering event.

That is, upon a `RRConnectionSetupComplete` the PDCP sends to the next slice a previously stored `RRConnectionSetup` message, containing the details of the RRC channel. This, in turn, triggers the NAS authentication procedure in the new slice. Each time a slice finishes its NAS configuration, the RRC calls a `slice_configured` function within the PDCP, including the `(slice_id, IP address)` tuple of the slice, which will support the proper forwarding of information within the module (this is only needed for receiving information).

Besides coordinating the c-plane, the slice coalescer in the UE also has to multiplex and demultiplex the u-plane. This is achieved by exploiting the data multiplexer available at the MAC for the UL:

This function demultiplexes the data coming from the lower layers and forwards to the appropriate slice instance at the PDCP on a per-destination IP prefix basis.

eNodeB

The changes in the eNB are the counterpart of the ones introduced in the UE. That is, the slice coalescer handles the multiplexing and demultiplexing of the c- and u-plane. The multi-slice updates in the eNB are less elaborated than the ones in the UE, as the eNB is already capable of handling parallel authentications coming from different UEs. We remark that multiple authentications coming from the same multi-slice UE (such as the one described in Section 6.1.1) can be considered as atomic operations, as they happen sequentially.

This simplifies the required enhancements at the eNB, as there are no concurrent NAS procedures for the same UE running simultaneously, and therefore each one can use the same inner data structure available at the RRC. In this way, we use a flag to mark the slice under configuration, which enables forwarding the control traffic to the corresponding CN via the appropriate S1 Application Protocol (S1AP) interface.

Like with the UE implementation, the u-plane multiplexing happens in the PDCP, following an IP-prefix matching approach, i.e., data traffic is forwarded to the right CN by considering the source address of IP packets.

Core and MANO

The main enabler of our slicing solution is the RAN slicing. Therefore, to allow an easier experimentation with unmodified software solutions, we did not tackle CN VNFs, leveraging on a vanilla implementation such as the one provided in the OpenAirInterface suite. Similar considerations hold for the MANO part: one of our objectives is to allow the open experimentation of MANO algorithms on top of the POSENS stack.

The MANO of VNFs, a fundamental part of the future 5G Networks, is being standardized by the 3GPP SA5 and will leverage on the already consolidated elements of the ETSI Network Function Virtualization (NFV) MANO architecture [86]. We include in POSENS a baseline implementation of this MANO functionality, which builds on top of an open source Virtual Infrastructure Manager (VIM) (OpenStack), and provides a per-slice orchestration (which is the functional role played by the VNF Manager and the NFV Orchestrator in the ETSI architecture) through an ad-hoc Java software. This implementation leverages directly on the Nova and Neutron Application Programming Interfaces (APIs) to provide a lightweight version of the VNFM-Vi and Or-Vi reference points defined by ETSI.

6.2. Experimenting with open source tools to deploy a multi-service and multi-slice mobile network

As we exposed at the beginning of this chapter, while network slicing has been acknowledged as a key technology to efficiently support services with very diverse requirements [16], there are few open solutions implementing it and, consequently, little experimental “hands on” reports on the use of this technology in practice.

In this section, we contribute to filling this gap by reporting on the development and validation of a multi-slice 5G network prototype, each slice serving a different application, and by showcasing several features such as the *reallocation of virtual network features* or the use of *Local Breakout (LB)* to minimize delays [87]. We describe the hardware used, the software installed, and how the different building blocks are connected, providing in this way researchers and practitioners with a “how to” guide while reporting on our experiences and best practices for prototyping. We believe our results provide valuable information to boost the development of further 5G prototype initiatives.

6.2.1. Novel services considered

The deployed network provides two services over two network slices, with a focus on aspects like QoS/QoE-aware control, Network Function Virtualizations (NFVs) and orchestration. The motivation is to feature two different network slices on the cloud infrastructure: one with a reduced latency service and another one with a mobile broadband service. We show how an ETSI NFV Management and Network Orchestration (MANO) platform can be used to deploy, manage and orchestrate different services on different network slices, this including the dynamic re-orchestration of a particular Network Service (NS) forwarding graph, and the placement of certain Virtual Network Functions (VNFs) in the appropriate host. In both cases, QoS/QoE aspects trigger the re-orchestration function. The software produced for this work is partially based on [4] and it is mostly available on GitHub ² ³.

The two network slices are: (i) a Reduced Latency slice (i.e., Ultra-Reliable Low Latency Communications (URLLC)), used to read real-time physical measurements triggered by Quick Response (QR) labels, and (ii) an Enhanced Mobile Broadband (eMBB) slice, which serves contextual captions to streaming media, according to the user profile and surrounding context. Both slices are deployed on the same Evolved Node B (eNB) and share the same spectrum. For the Core Network (CN) part, each tenant (i.e., a service) runs its own instance of the protocol stack (i.e., mobility management, gateways and the upper layers), performing the QoE/QoS policies needed by each scenario. ⁴

²<https://github.com/wnlUc3m>

³<https://github.com/bsnet>

⁴Although we deploy a multi-tenancy case where each service belongs to a different tenant, other

Service 1: Video Streaming through the eMBB slice

The first service is hosted by an eMBB slice that has been designed to provide a service consisting of enriching a video streaming signal with context-based add-ons (e.g., subtitles or other graphical elements) depending on the user preferences (i.e., color, language), other conditions (e.g., hearing impairments) and surroundings (e.g., ambient noise). These profiles or environmental conditions can be understood here as QoS/QoE influence factors: a final user, for instance, can generate a trigger to explicitly request the service according to its preferences; or certain QoS metrics could automatically activate the service without an explicit user request.

The additional video features are activated by means of MANO procedures that dynamically add the necessary VNFs to the forwarding graph without interrupting the video streaming. Besides the possible real-life applications of this service, our goal is to demonstrate three different orchestration-related functionalities:

(i) *On-boarding of the network service itself*, i.e., the deployment of the necessary VNFs driven by service descriptors where the operational layout and requirements are defined; (ii) *Dynamic update of the NS Forwarding Graph (FG)* depending on QoS/QoE measurements (QoS/QoE-awareness); (iii) *Placement of VNFs to specific compute nodes*, to simulate the placement of Network Functions (NFs) in the edge or core cloud depending on the service requirements.

Service 2: Augmented Reality through an URLLC slice

The Reduced Latency use case consists of an Android application that performs Augmented Reality (AR) using QR codes. In a real environment, those codes may be distributed in an industrial area close to the equipment where measurements of interest are obtained (e.g., pipes flow or pressure, electric measurements, tank levels, etc.). On each QR code decoding, the mobile terminal requests the corresponding information that it displays to the user on top of the captured image.

To keep delays between the User Equipment (UE) and the information server low, the latter shall be located close to the user, e.g. in the same element hosting the eNB, or in an edge cloud. To this end we deploy within this application a *local breakout* component that can route selected traffic directly towards an edge cloud. With this feature, information flows are processed locally and incur into smaller delays.

multi-tenancy scenarios can be supported with our solution.

6.2.2. Access Network

The mobile network architecture employed in our deployment relies on a well-known open-source software implementations of the LTE Stack: `srsLTE` [15] an open source implementation of a UE and the eNB. In the following Sections we describe how we extended the software architecture to support the two key features of our solution, namely, (i) support for multi-slice in the radio part, providing isolation and resource differentiation across services, and (ii) support for local breakout, which enables a more efficient implementation of latency-critical services.

While the functions in the Core Network run as VNFs (as we will describe below), the network functions for the radio part run as Physical Network Functions (PNFs). That is, the cloudification of RAN function is currently a very hot research topic [88]. Still, the porting to real systems of real cloud RAN software is in its early stages. Our implementation of the slice-aware RAN is based on `srsLTE` [15],

Although it is a full software implementation of a 3GPP Radio Access Network (RAN) suite, also `srsLTE` has a limited integration in the state of the art virtualization technologies such as Virtual Machines and Containers. Similar considerations apply for the other major open source RAN platform: `OAI` [22].

The `srsLTE` suite used for this implementation defines different classes for the different tasks that have to be fulfilled in the RAN: (i) a common library that performs the encoding/decoding operations, up to the MAC; (ii) the Radio Link Control (RLC), Radio Resource Control (RRC) and the Packet Data Convergence Protocol (PDCP) layers; and (iii) specific modules for the UE (the UE module for the data plane of the UE) and the eNB (S1 Application Protocol (S1AP), that manages the connectivity for the core control plane, and Gateway (GW) for the GW connectivity).

We provide two alternative approaches for the RAN slicing solutions. The first option is `POSENS` (see Section 6.1), an end-to-end network slicing solution based on `srsLTE` described in Section 6.2.2.1. Then, we provide an alternative solution based on traffic differentiation described in Section 6.2.2.2.

6.2.2.1. RAN slicing implementation

To cover the network slicing RAN requirement for our prototype, one of the alternatives is the usage of `POSENS` (see Section 6.1). `POSENS` implements the so-called “Slice-aware shared RAN” scheme as it is defined in [16] and represented in Figure 6.1 (Option 1), which basically covers the requirements for this mobile network prototype.

Summarising, `POSENS` modifies the higher layers of the network protocol stack (i.e., PDCP and RRC) on both the eNB and UE. Namely, `POSENS` provides per slice differentiation for common c-plane and u-plane procedures as follows: for the c-plane, it allows two registrations against two different Non-access stratum (NAS) instances, so

it duplicates (i.e. an instance for each slice) the UE module and the RRC layer, which triggered two NAS connectivity requests. On the other hand, it multiplexes and demultiplexes at PDCP module, according to the final destination address of the traffic. That is, by triggering two NAS registration procedures, the UE obtains two valid IP addresses from each slice Serving Gateway (S-GW) and creates two virtual network "TUN" interfaces (one per slice). Therefore, the UE can connect to two different slices at the same time.

A sketch of the proposed implementation is depicted in Figure 6.2. The network slicing enabling modules, marked in green, provide the (de)multiplexing functionality described above, discriminating the traffic (both control and user plane) belonging to each slice between the slice specific modules (among them the NAS, which is part of the core functionality explained in Section 6.2.4) and the shared ones. This allows for a differentiation of different flows in the core part, while keeping the radio shared, enabling enhanced orchestration capabilities such as the one described in Section 6.2.5.

6.2.2.2. An alternative implementation for RAN slicing

Differentiating traffic at the MAC layer is a problem that has received a broad attention in the literature in the past [89]. However, the introduction of network slicing into Medium Access Control Layer (MAC) schedulers requires modifications that go beyond the metrics that are usually controlled by state of the art solutions such as the delay budget or the minimum throughput.

As already discussed by [20], network slicing aware solutions need also to take into account other factors such as the amount of resources allotted to each slice, which are usually not considered.

In order to differentiate the resources given to each slice, for simplicity we decided to introduce modifications to the Scheduler that is in charge of assigning resources to the different UEs attached (in the following, we assume that different UE are assigned to different slices). Our motivation is to demonstrate the feasibility of a slice-aware scheduler, by enforcing different resource assignment to tenants according to, e.g., some pre-defined requirements (the means to convey these requirements are out of our scope). For instance, if on a 5G radio access point two tenants are assigned with concurrent slices, but one has more stringent requirements than the other, it shall be granted a larger share of resources to fulfil them.

The scheduler is in charge of the allocation for shared time-frequency resources among users at every time instant. In `srsLTE`, this module is located in the eNB part and assigns Uplink (UL) and Downlink (DL) resources according to the scheduling policy. Thus, it determines to which user the shared resources (time and frequencies) for each Transmission Time Interval (TTI) (in our case, 1 ms) should be allocated for reception of Downlink Shared Channel (DL-SCH) transmissions (We refer the reader to [15] for a

thorough description of the `srsLTE` platform)

By default, `srsLTE` implements a Round Robin scheduler: commonly used in LTE networks, UEs are given resources sequentially. When all the UE have been assigned with Resource Block (RB) at least once, it starts over. In this way, this scheduling results very simple and does not take into account e.g. the Channel Quality Indicator (CQI) reported by UEs. While this has some disadvantages in terms of spectral efficiency (as UEs with instantaneous poor channel conditions are scheduled), it is easy to be implemented and provides good fairness in the terms of RBs allocation.

We decided to modify the Round Robin scheduler to implement a **weighted Round Robin**, where the assignment is performed by weighting the time slot assignment following the priority assigned to each slice. For this purpose, we define a vector of weights $\mathbf{W} = [w_1, w_2, \dots, w_n]$, where w_i represents the relative weight of slice i among the n slices served by the RAN.

For instance, $\mathbf{W} = [1, 1, 1]$ means that the system is serving $n = 3$ slices with equal weight, while $\mathbf{W} = [2, 1]$ represents a system with $n = 2$ slices, with the first one obtaining twice the resources than the second.

If we let $l = \sum_1^n w_i$ and $\hat{\mathbf{W}}$ be an array that holds the cumulative sum of \mathbf{W} , we assign RBs in the following way. Being t_i the incremental TTI number we compute t as its modulo base l . Finally the scheduled slice i is $i = \arg \min_i \hat{w}_i - t$. In this way we assure that upon every cycle of l TTIs, we assign resources to slices according to their relative weight \mathbf{W} . In Section 7.2 we evaluate the behaviour of the proposed solution.

6.2.3. Local breakout

Enforcing network slicing at the RAN level (as discussed in Section 6.2.2) allows to implement different per-slice data traffic management policies as early as the traffic flows leaves the Radio. This is of particular importance when the considered slices have stringent latency requirements and their data flows shall be handled at the edge. One promising solution to this problem is Local Breakout (LB).

The `local breakout` functionality could be deployed in different manners: for instance by directly modifying the network configuration to support different Packet Data Networks (PDNs) on the UE. However, deploying it as an independent VNF results a more flexible solution, for at least two reasons: *(i)* it can be maintained and upgraded separately, and *(ii)* it would work (in principle) with any eNB, as it does not require any NF-specific Application Programming Interface (API). This point makes the life-cycle management of the LB software much easier, allowing to deploy it on demand and at run-time. We next present the main characteristics of our LB implementation.

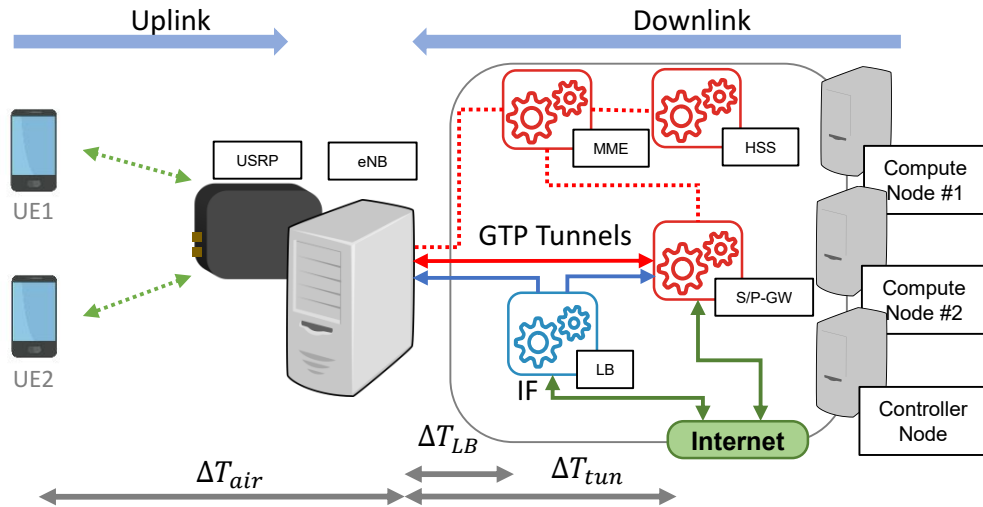


Figure 6.3: General scenario showing a legacy GTP tunnel (top, red dotted/solid lines) and a GTP with LB (bottom blue line). The LB introduces additional interface IF for routing traffic locally.

GTP-tunnel and LB

Differently than in a Wireless Local Area Network, traffic generated by mobile users is not routed at the device that terminates the air interface: it is, in fact, delivered inside a GPRS Tunneling Protocol (GTP) tunnel to the remote Gateway controlled by the user's service provider (S-GW/ Packet data network Gateway (P-GW) for Long Term Evolution (LTE), User Plane Function (UPF) for 5G) where it starts its journey to the destination (top tunnel in red/solid line in Figure 6.3). Return traffic is first received by the gateway that tunnels it to the specific eNB at which the destination UE is connected. This method facilitates accounting for roaming users but introduces inefficiencies at the routing level.

To avoid these issues, a LB device can be set up to intercept GTP packets earlier (bottom tunnel in blue/solid line). We report in the following the LB architecture focusing on the mechanisms for transparently intercepting and conveniently routing selected traffic locally.

6.2.3.1. LB implementation

As shown in Figure 6.3, the LB node has to *(i) inspect tunneled packets to extract those matching specific rules and forward them through the new interface IF;* and *(ii) push packets received from IF into the tunnel.*

As uplink users' packets are embedded in User Datagram Protocol (UDP) datagrams, the LB can easily access both the fixed length GTP header and the original IP packet. The GTP header is 8 byte long and contains the Tunnel endpoint identifier (TEID),

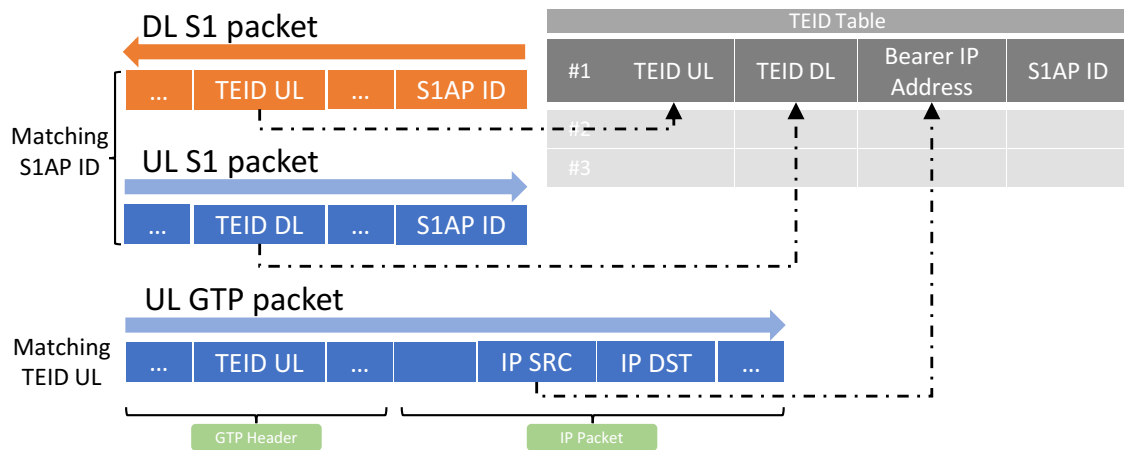


Figure 6.4: TEID table maintained by the LB threads: it associates TEID numbers to the IP address of each UE. It is used for crafting GTP tunnel return packets.

a 32-bit value that identifies the bearer to which the inner IP packet is addressed and that is generated by the eNB/Mobility Management Entity (MME) when the UE node connects to the network (or when it requests a new service). We implemented the tasks for intercepting and pushing packets into the tunnel and for determining the TEID values as three main threads that refer to a common TEID table for storing/fetching TEID values:

S1 sniffer thread. When a UE requests a new service, the eNB and the MME exchange a couple of S1 packets that carry new TEID values: one in downlink with the TEID decided by the MME; followed by one in uplink carrying the TEID chosen by the eNB. The knowledge of the latter is fundamental to the LB for pushing downlink packets received from the IF interface into the tunnel. We show in Figure 6.4 how the S1 thread correlates the two S1 packets by using the common S1AP-ID field to create a new row in the TEID table with the corresponding TEID values. Because of the complexity in dissecting S1 traffic, the S1 thread forks a `tshark` process and uses a pipe for receiving the TEID data from it. It is worth noting that at this stage the IP address of the UE is not yet known: it will be discovered by the Uplink thread as we explain next.

Uplink thread. To inspect all GTP packets going to the S-GW/P-GW, the Uplink thread installs a rule in the netfilter framework of the LB kernel that matches UDP packets addressed to the remote gateway. Setting `NFQUEUE`⁵ as target allows the Uplink thread to receive all packets and decide which must be stopped, stripped by their GTP header and injected through interface IF.

When a UE bearer transmits an uplink data packet for the first time, the thread adds

⁵https://www.netfilter.org/projects/libnetfilter_queue

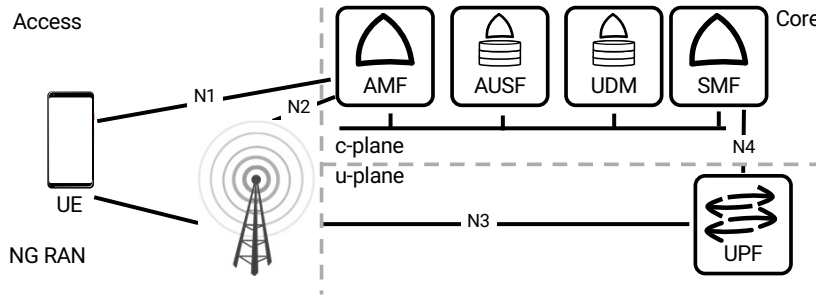


Figure 6.5: Sketched view of the 5G Core, as in [10]

the source IP address found in the packet inside the TEID table: to this end, it looks up the corresponding row by searching the TEID UL value extracted from the GTP packet, as shown at the bottom of Figure 6.4.

Downlink thread. This thread receives packets from the IF interface and pushes them into the GTP tunnel. It first uses the destination IP address of the packet to look up in the TEID table the value of the TEID DL field: it then crafts a new GTP packet by copying the TEID DL value in the header and concatenating the IP packet in the GTP payload. If no TEID DL value is found, the thread simply drops the packet.

6.2.4. Core Network

The transition towards 5G and the application of new concepts of network softwarization has pushed the standardization efforts towards a cleaner designed of the core network with respect to the 4G/LTE. More specifically (since Rel. 14, the last 4G one), the main CN modules implement a split between the control plane and the user plane, distinguishing between the Network Function devoted to c-plane functionality and the one that handles u-plane.

Therefore, in our implementation, we decided to leverage on this new design paradigm, to provide an implementation of some selected functionality of the 5G Core, which follows the c-/u-plane split discussed above. Some of the code has been adapted from existing open source projects, while other components which were not available as open source have been implemented from scratch. Also this part is publicly available in our repository.

We next describe the baseline 5G Core architecture that we have developed that includes: (i) the interface towards a multi-slice capable access network (both in the UE and eNB, such as the one described in Section 6.2.2), to support end-to-end network slicing, (ii) a *modularized* implementation of the CN, as mandated by recent 3GPP standards, and (iii) their interfaces towards the MANO framework.

Although there are a number of initiatives providing CN functionality (e.g., the ones

included in OAI and srsLTE, or NextEpc⁶), in our testbed we implemented most of our solutions from scratch, to take full advantage of the novel architecture designed by 3GPP, including the following features: (i) a clean control/user plane split and, (ii) a service based architecture (SBA) for the c-plane function. Specifically, we have implemented all the CN modules illustrated in Figure 6.5, including the c-plane interfaces for the SBA (which are mandated by 3GPP).

The implementation of the SBA allows for a consumer / producer communication that enables the modularization of the core network functions. The implementation of the Access Management Function (AMF), Authentication Server Function (AUSF) and User Data Management (UDM) functions is done in Python 3 and inspired by the highly-modular design of srsLTE, as detailed next.

User Plane Function (UPF)

This function provides the encapsulation, decapsulation, and forwarding to the Packet Data Network. Its context consists on the current rules applied to encapsulate/decapsulate packets and to forward them. This function has already been built to enable the standard OpenFlow protocol to deliver the c-plane rules (i.e., the context) to the function, which simplifies the implementation of the context extraction and installation primitives, as they are very similar to the existing OpenFlow primitives. We have implemented the UPF module building on the Open Source, OpenFlow-capable Lagopus switch.⁷

Session Management Function - SMF

This c-plane function controls and configures the UPF instances on the u-plane through the N4 interface. Thus, the context here also consists of the rules to encapsulate/decapsulate/forward packets, in this case for all the UPF functions controlled by the Session Management Function (SMF). For the implementation of this module, we leverage available SDN-capable implementations, enriching them with mobile network functionality, and employing a Ryu Controller⁸ to implement the N4 interface between the UPF and the SMF. One key feature of our implementation is that the context information is stored in a separated object, thus facilitating its extraction and installation.

Other Core functions

In our testbed we also implemented other Core network functions that implement some selected functionality for our setup. In order to provide the basic authentication

⁶<http://nextepc.org/>

⁷<http://www.lagopus.org>

⁸<https://ryu-sdn.org/>

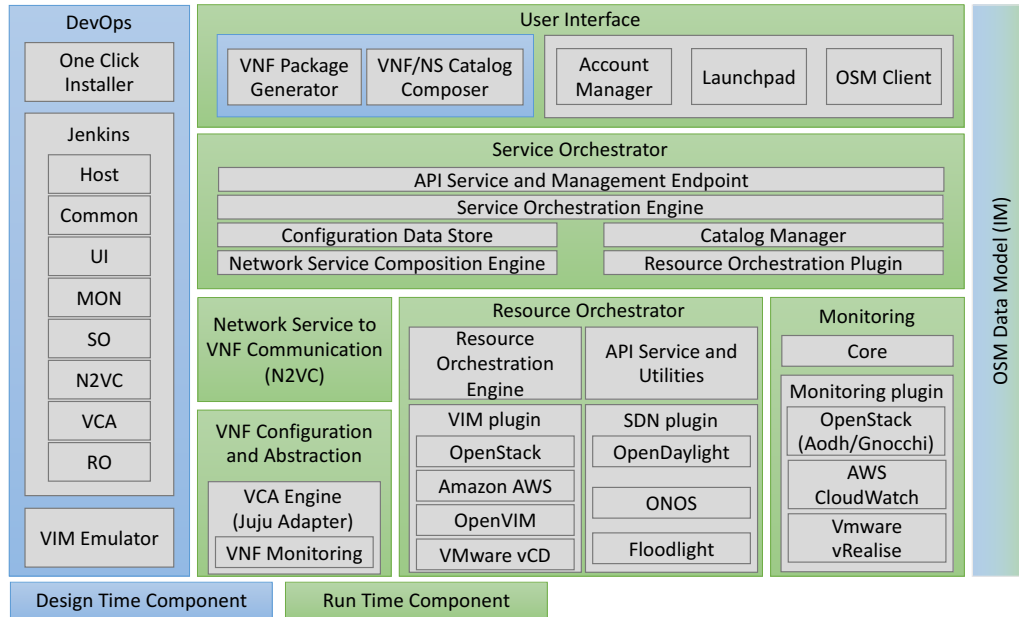


Figure 6.6: The OSM architecture (Adapted from [11])

functions we developed a tailored Python prototype of the AUSF, which retrieves user information from the UDM modules that holds all the basic user information such as the private keys or the IMEI.

While the two modules above mostly deals with the authentication procedures, we also had to implement a lightweight version of the AMF which deals with the selection of the radio access from the core perspective. While this functionality is needed for the overall behaviour of the network, in our scenario is trivially linking the `srsLTE` software with the core modules to provide connectivity.

6.2.5. Management and Orchestration

The MANO Engine

In order to implement the features envisioned by the testbed architecture described above, we rely on an orchestration architecture based on the Open Source MANO (OSM) orchestrator and the OpenStack Virtual Infrastructure Manager (VIM). OSM is one of the leading solutions for the implementation of a fully-fledged mobile network orchestrator that includes several components for the automatic Life Cycle Management (LCM). Still, to provide the enhanced functionality needed, substantial changes to the OSM code and architecture are provided.

Figure 6.6 shows the OSM architecture, which is composed by several modules that performing the different functionality needed by an orchestration solution. Next, we describe the additional modules that are required for supporting the specific features

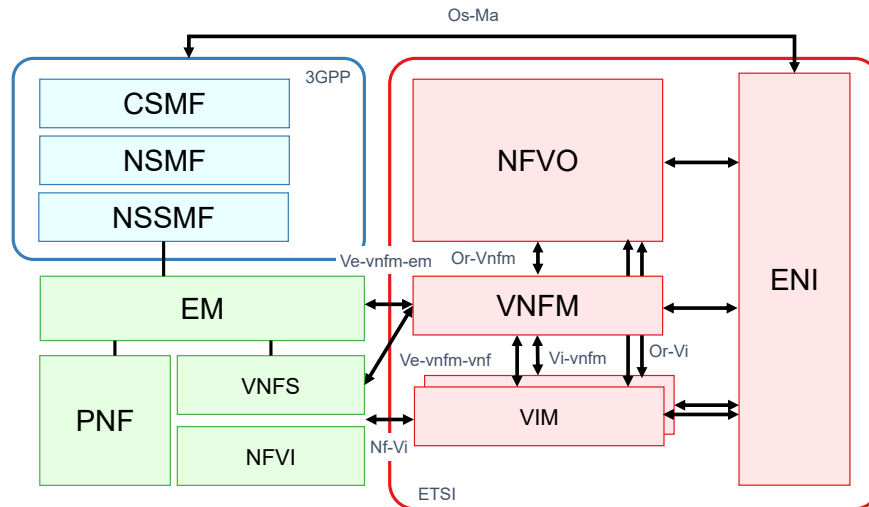


Figure 6.7: The relations between our implemented architecture and the ETSI - 3GPP domains

needed for our purposes:

- Resource Orchestrator:** This module provides the hooks towards different VIMs supported by OSM. In the context of the testbed, we use OpenStack as the main VIM, to manage the local Network Function Virtualisation Infrastructure (NFVI) deployment. However, we could leverage also on other APIs: the Amazon AWS EC2 one, to demonstrate the feasibility of a large-scale deployment over public clouds and the Kubernetes one, to possibly include container-based VNFs.
- Network Service to VNF communications:** We extend this API (that is used to configure the VNFs in a chain) to allow more specific configurations (such as the ones needed by the radio ones, that are PNFs) and support the VNF relocation.
- Orchestration module:** the specific algorithms that trigger the network function relocation or the local breakout reside on top of the Service Orchestration module, to use the API specifically designed for that purpose.

ETSI NFV compliancy

The diagram represented in Figure 6.7 shows the framework of the testbed based on the well-known ETSI NFV [90] and 3GPP MANO architectures [91] that are also building our one. We populated the specific modules of the architecture with our algorithms, as specified next.

The **onboarding** functionality encompasses all the baseline operation available in a 5G MANO system. The automatic onboarding of the two network slices is a seamless

Listing 1 Network Slice Descriptor snippet.

```

vld:
# Networks for the VNFs
- id: management
  name: management
  short-name: management
  type: ELAN
  mgmt-network: "true"
  vim-network-name: public
  vnfd-connection-point-ref:
    # Specify the constituent VNFs
    # member-vnf-index-ref -
    #   entry from constituent vnf
    # vnfd-id-ref - VNFD id
    # NGINX
    - vnfd-id-ref: cirros_vnfd_s11
      member-vnf-index-ref: "1"
      vnfd-connection-point-ref: vnf-mgmt1
      ip-address: 192.168.200.185
      # 245_UPPER_LAYERS
    - vnfd-id-ref: cirros_vnfd_s11
      member-vnf-index-ref: "1"
      vnfd-connection-point-ref: vnf-mgmt2
      ip-address: 192.168.200.186
      # 245_UPF
    - vnfd-id-ref: cirros_vnfd_s11
      member-vnf-index-ref: "1"
      vnfd-connection-point-ref: vnf-mgmt3
      ip-address: 192.168.200.187
  
```

Figure 6.8: NS Descriptor snippet.

operation that allows (i) verticals to define the requirements associated with each slice (in this case, high bandwidth for the eMBB and very low latency for the URLLC) and (ii) the deployment of the VNF in the cloud to fulfil the set of requirements. This part has specific elements of novelties for the blueprinting. Specifically, we implemented it by using the YAML file descriptors that are used for the definition of VNFs in OSM. More specifically, two kinds of descriptors have to be created: one for the virtual networks (i.e., the virtual links that span one or more Virtual Machine (VM) and connect VNFs among them) called Network Slice Descriptor (NSD), see Figure 6.8) and one for the virtual appliances that run on top of them, called Virtual Network Function Descriptor (VNFD), see Figure 6.9).

Listing 2 VNF Descriptor snippet.

```
- id: upf
  name: upf
  description: UPF
  count: 1
  # Flavour of the VM to be instantiated for the VDU
  vm-flavor:
    vcpu-count: 1
    memory-mb: 512
    storage-gb: 5
  # Image including the full path
  image: "ubuntu"
  interface:
    # Specify the external interfaces
    # There can be multiple interfaces defined
    - name: eth0-mgmt3
      type: EXTERNAL
      virtual-interface:
        type: VIRTIO
      external-connection-point-ref: vnf-mgmt3
      position: 1
    - name: eth1-in10_0_0_3
      type: EXTERNAL
      virtual-interface:
        type: VIRTIO
      external-connection-point-ref: in10_0_0_3
      position: 2
```

Figure 6.9: VNF Descriptor snippet.

6.2.6. Functions beyond 3GPP

The network functions described in Section 6.2.4 provide eventually IP connectivity towards the so-called PDN that is attached to the UPF. However, in order to provide novel 5G services (such as the ones described in Section 6.2.1), additional functions besides the 3GPP ones shall be instantiated and orchestrated by a MANO framework. For instance, an eMBB slice may need a Content Distribution Network to be deployed or an URLLC slice may need to leverage on Mobile Edge Computing capabilities.

Enriched video service

Specifically, the main feature of the eMBB network slice deployed in this work is to provide a service that may add context-based add-ons on a live video streaming signal, depending on the user profile, preferences, and certain environmental conditions. These parameters are defined here as QoS/QoE influence factors, and serve as VNFs

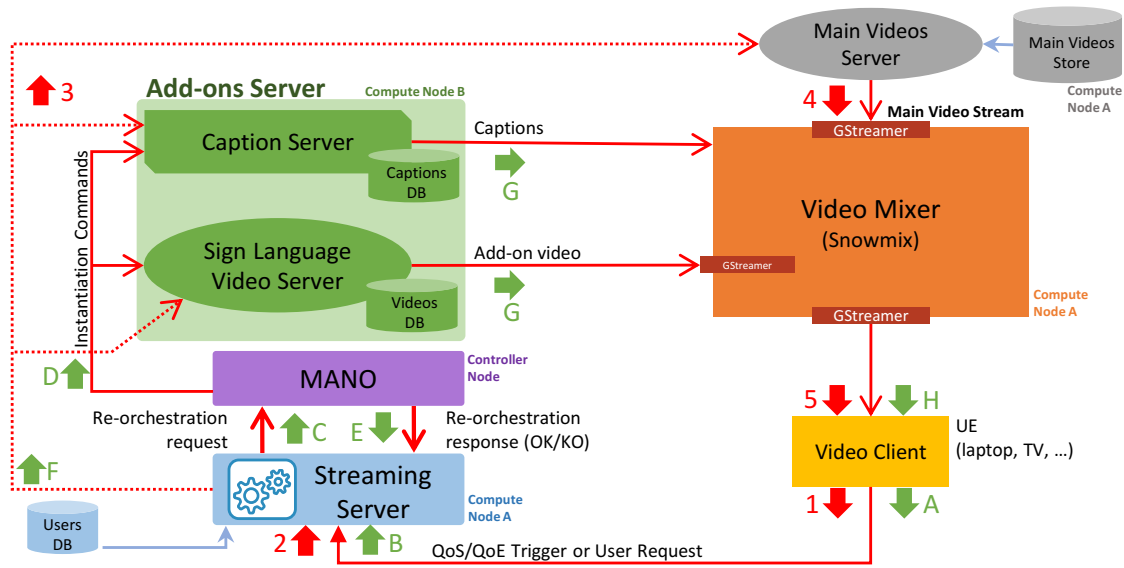


Figure 6.10: The Orchestration architecture

re-orchestration triggers.

The add-ons can be subtitles, generated depending on the subscriber preferences (language, colour, size, position, etc.) or other support videos, with the sign language translation (e.g., for people with a hearing impairment). These add-ons could be explicitly requested by the user at any time during the video playback or triggered by environmental conditions (e.g., excessive ambient noise), so a good synchronization between add-ons and the main video signal is a must.

The MANO architecture and the orchestrated network functions to provide this service are depicted in Figure 6.10. In the Figure, the red arrows labelled from 1 to 5 show the initial interactions, where the user request the playback of a certain video from a client (yellow box on the right). This is just a regular video service providing the requested video to the user, where the Streaming Server block (bottom left) is acting just as a proxy for the Main Videos Server (where the video files are actually stored). The video stream pass through the Video Mixer block (orange box), which although it does not mix the video with any other content, it helps to prevent disruptions when add-ons are inserted.

When add-ons are requested by the user (green arrows, labelled from A to H), the Streaming Server communicates with the MANO block, which instantiates the Add-ons Server (green box). This server is split into two internal blocks: the Sign Language Videos Server (which is used to provide the sign language videos) and the Captions Server (for providing subtitles). Once the appropriate Add-on Server is instantiated, the MANO informs the Streaming Server block, which sends control commands towards the Add-ons Server (dashed line). The selected add-on (video and/or subtitle) is injected into the Video Mixer block which delivers the mixed video signal to the final user.

The Video Mixer block is implemented using *Snowmix*, an open-Source and very-flexible command line tool for dynamically mixing live audio and video feeds which supports overlaying video, images, texts and graphic elements as well as mixing audio [92]. All these components have been embedded in a special-purpose VNF, deployed on one of the compute nodes. The Add-ons Server is split in two VNFs, corresponding to the two blocks in the Figure, i.e., the Captions Server and the Sign Language Videos Server. Each VNF contains a database with the necessary caption and video files, and a service running over TomCat [93]. The service exposes a Representational state transfer (REST) API used by the Streaming Server block to execute the necessary control commands once the VNF instances are up and running. The Caption Server communicates with the Video Mixer block using a Snowmix-specific protocol which makes it possible to specify where and how the add-ons are placed within the video. The Sign Language Videos Server uses *ffmpeg* [94] to stream the add-on videos into the mixer.

The Main Videos Server is also an independent VNF, running over the same compute node as previously mentioned. It contains a database with all the possible videos the final user could access. The Streaming Server is an independent VNF which performs three primary functions:

1. It works as a server for the end user. Specifically, it embeds an Hypertext Transfer Protocol (HTTP) server to what the final user can access (using a general purpose web browser) to request the video playbacks and the available add-ons.
2. It decodes the user's HTTP requests, implementing the logic to trigger the instantiation of the necessary add-on server (through a request to the MANO) and sending the necessary control commands to these instances, and also, to the Main Videos Server.
3. It maintains a good synchronization between the main video stream and the add-ons. Add-ons can be requested at any time by the final user, so they must appear properly synchronized with the main video and with a minor delay.

The MANO block (purple box in Figure 6.10) represents our implementation of the MANO framework. Although not explicitly represented in Figure 6.10, it also communicates with the underlying ETSI NFVI MANO components (i.e., NFV Orchestrator (NFVO), Virtual Network Function Manager (VNFM) and VIM, embedded in the OSM platform) to orchestrate and manage the virtual resources. This process is executed on the Controller Node. Finally, regarding the End-User equipment (yellow box in the Figure), it consist of two different elements, namely, the video player and a web browser with a Graphical User Interface (GUI). The former is based on the *ffplay* video player [94], while the later is developed for the end-user to control the service.

Latency-triggered re-orchestration

Both network slices are orchestrated with all their VNFs (namely the higher layers of the RAN stack, the eNB) instantiated and running in the central cloud. The MANO system continuously collects data about the network parameters of the virtual network (i.e. latency, throughput, available and used radio resources). This is especially important for the URLLC slice, which may have very stringent requirements on the E2E latency between the UE and the server running in the cloud. For this reason, the delay is constantly monitored to avoid operational glitches caused by a sudden delay increase due to external factors, such as additional congestion in the radio and transport, or internal ones, like a high number of UEs connected to the server. In these cases, the orchestration framework triggers (i) a relocation of the NFs and Virtual Reality (VR) application to the edge cloud, to benefit from the reduced latency or (ii) the offloading of some selected flows through Local Breakout.

6.3. ACHO: A framework for flexible re-orchestration of virtual network functions

Based on the analysis of the state of the art orchestration solutions (Section 4.3.2), we conclude that there is no practical open source tool for flexible orchestration of Virtual Network Functions (VNFs) in a mobile network architecture. This motivated the design and implementation of Adaptive slice re-Configuration using Hierarchical Orchestration (ACHO), a framework that provides the following novelties as compared vs. the state of the art:

- Firstly, ACHO targets mobile networking, a more heterogeneous scenario with very diverse network functions with very different requirements (e.g., access network vs. core network functions).
- ACHO provides a clear methodology to adapt existing VNFs, which follows the recent architectural trends of 5G networking, and is aligned with the ongoing standardization efforts.
- ACHO specifies two novel interfaces to support and dynamic and fine-grained orchestration, which can be easily implemented with existing off-the-shelf orchestrators.
- ACHO also provides a fully-featured implementation of 5G VNFs and orchestration elements, which can be easily downloaded, customized, and tested with off-the-shelf hardware.

- Finally, we discuss different implementation strategies, also aligned with existing standardization efforts, to maximize the practicality of ACHO.

6.3.1. ACHO: A suite for flexible 5G networking

We next present ACHO (Adaptive slice re-Configuration using Hierarchical Orchestration), a software framework consisting of an implementation of 5G Functionality (the most critical 3GPP Rel. 15 Core Network Functions, depicted in Fig. 6.5 and marked with SBA in Fig. 6.14), the radio access network functions and the Management and Network Orchestration (MANO) modules to handle them. ACHO provides a full network-slicing aware solution that includes all the MANO modules to enable a flexible re-orchestration of the mobile network. Some of the network components of ACHO have been adapted from existing open source projects (e.g., srsLTE), while other components that were not available as open source have been implemented from scratch. As a result, the software codebase provided by ACHO is very complete and has no match in the landscape of the open source mobile networking initiatives.

6.3.2. Re-configuration of VNFs, a context-based approach

As discussed in Section 4.3.2, a plethora of orchestration algorithms rely on dynamically migrating a VNF *on the fly*. However, very few of them deal with the actual implementation of the migration mechanism, with the work of [83] being among the notable exceptions, but providing very poor performance. This motivates the design of the ACHO framework.

The key enabler of ACHO is a clean split between the *context* of a network function and its *execution* engine, which we refer to as the Context/Execution (c/e) split. The context is defined by the current values of all the variables employed by the function, while the engine is the part responsible for the actual execution of the function. In this way, when relocating a network function, it is sufficient to move to the new location just the context, which contains the “state” of the function. Therefore, we can instantiate a new function engine in the new location and feed it with the data corresponding to the context extracted from the previous location. The proposed strategy is depicted in Fig. 6.11 and it can be compared with Fig 4.9, where a full VNF relocation is performed.

By moving the context of a VNF only, we reduce the amount of information that has to be moved to the bare minimum, without incurring into large penalties as done by VNF unaware solutions [77]. For instance, the tests performed in [95] show how the total amount of data transferred is almost a linear function of the Virtual Machine (VM) size.

The idea of splitting the context of a function from its execution engine has also been proposed by some works in the literature, most notably OpenNF [96] and Split Merge [97]. These papers propose the fast relocation of VNFs by moving the least

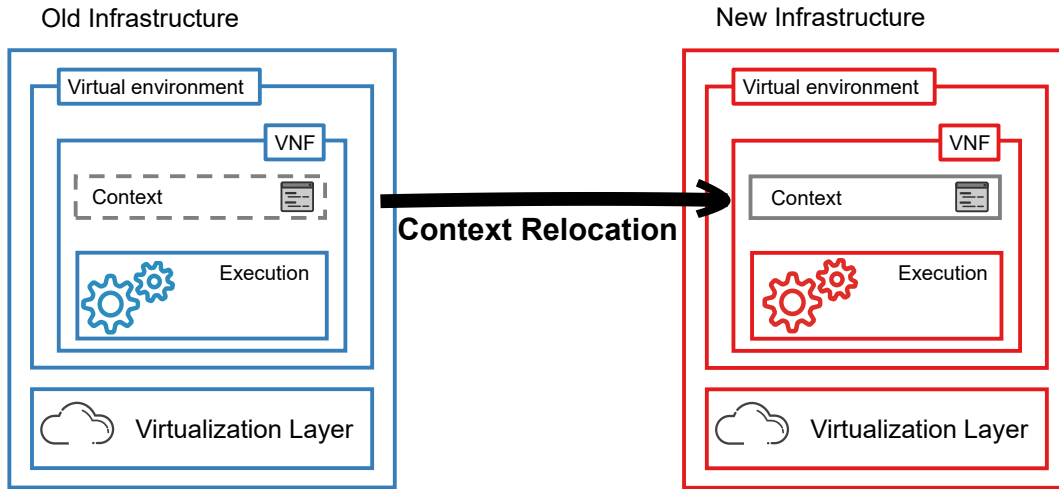


Figure 6.11: the context relocation performed with ACHO

amount of information between different virtualization environments. While these cases are relatively similar to ours, these solutions lack two key features that preclude their use in mobile networks: (i) the considered VNFs do not constitute part of the “3GPP ecosystem,” and (ii) they lack an interface with a modern orchestrator, which is required to enable must-have features of mobile networks e.g., network slicing.

A similar idea is also currently included in the 3GPP specification [98], to relocate the information related to a specific User Equipment (UE) between different instances of a network function, in particular for load balancing purposes or to keep providing connectivity when a specific function is decommissioned. Still, this procedure is available for the core functions only.

Next, we introduce the meaning of "context" inside the Network Function Virtualization paradigm and discuss in detail how the proposed VNF migration can take place.

6.3.2.1. The VNF context

As discussed above, by introducing the *c/e* split, ACHO trades flexibility (i.e., the orchestration framework needs to know what kind of VNFs are running), with the compactness of the exchanged data, which is the bare minimum data representing the internal state of a VNF. The context is specific to each VNF but it is independent of the execution environment: for instance, we implemented ACHO for a VM-based deployment, but it can work with containers or *unikernels*. A context may comprise the specific rules of a firewall, or the information of the authenticated UE for an Authentication Server Function (AUSF). To support this, network functions need to be re-implemented to enable a clear separation between the context and the engine, a re-implementation that is specific to each function. Furthermore, these re-implemented VNF have to

expose this new capability, which could be achieved by e.g. extending the Network Exposure Function (NEF) to support c/e split through an Application Programming Interface (API).

An example of the context extracted from the Session Management Function (SMF) Network Function is depicted in Fig. 6.12. Given that the SMF is in charge of handling the end user session, routing them from the base station to the User Plane Function (UPF), the context of this function includes all the required information to re-install the relocated flow into the new VNF. Fig. 6.12 provides a JSON representation of the context, but binary formats such as e.g. Google PBF could also be used. The context does not contain information about the resources utilized in the underlying infrastructure (i.e., number of Central Processing Units (CPUs), amount of RAM) that are left to the MANO framework by using the standard technologies (e.g., the Virtual Network Function Descriptors (VNFDs)). A full description of the implementation of such relocatable functions is provided in Section 6.3.5.

Thus, according to operator-defined re-orchestration triggers (which can be computed from QoS metrics), the MANO pulls the context from the source VNF and injects it in the destination VNF (details on the specific interfaces are provided in Section 6.3.4). Hence, in the SMF case discussed here, all the information related to the Next generation Node B (gNB) and Gateway, including the tunnel id for each user is moved to the new location. Hence, the target VNF can immediately start serving the UE traffic from the new location.

Analogously, we depict in Fig. 6.13 the context used in our implementation of the MAC scheduler, which can perform re-orchestration based on per-slice. We use it to enforce isolation across UEs belonging to different slices, changing the Resource Blocks (RBs) allocation according to the number of served slices. This allows for a very granular per-slice (hence partial) re-orchestration, as all the parameters handled by ACHO are related to specific slice instances, as we also show with our proof of concept results in Section 7.3.

Hence, along with the re-implementation of these functions, we also need the means to transfer of the context from one location to another, i.e., instantiate the engine in the new location, feed it with the context, and update the corresponding communication paths. ACHO provides an open-source and practical implementation of this functionality, demonstrating that it is indeed feasible to relocate VNFs without disrupting ongoing services.

6.3.3. Baseline 5G implementation

To illustrate the benefits of the c/e split we need a working baseline implementation of certain 5G functionality, neither supported by current versions of 3GPP (i.e., 4G/LTE) nor existing open-source implementations. We next describe the baseline architecture that we have developed, which includes: (*i*) a multi-slice capable access network (both in the

```

1  {
2      "enb_tun_ip_addr": "192.168.10.12",
3      "gw_tun_ip_addr": "192.168.10.10",
4      "enb_tun_hw_addr": "xx:xx:xx:xx:xx:xx",
5      "gw_tun_hw_addr": "yy:yy:yy:yy:yy:yy",
6      "external_src_mac": "zz:zz:zz:zz:zz:zz",
7      "external_dst_mac": "cc:cc:cc:cc:cc:cc",
8      "ue_ip_addr": "172.16.0.30",
9      "teid": "11111111"
10 }

```

Figure 6.12: Representation of the SMF context

```

1  {
2      "nsl_id": "1",
3      "rnti": ["1", "2", "3"],
4      "start_rb_id": 1,
5      "stop_rb_id": 20,
6  }

```

Figure 6.13: Representation of the MAC context

UE and Evolved Node B (eNB)), to support end-to-end network slicing, and (ii) a *modular* implementation of the Core Network (CN), as mandated by recent 3GPP standards.

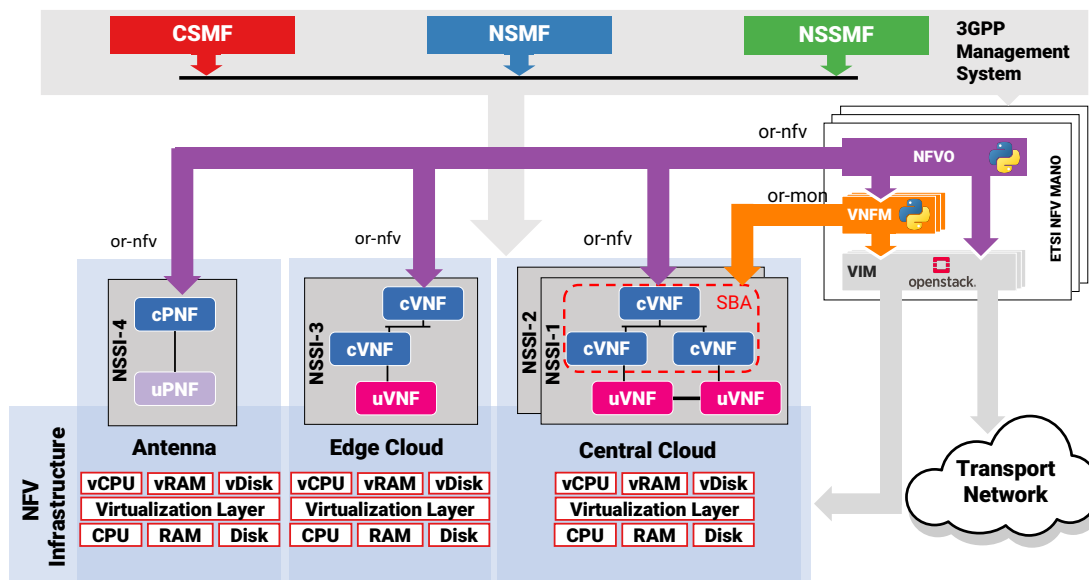


Figure 6.14: MANO Implementation and new Interfaces. ACHO creates new interfaces in the reference points defined by ETSI and acts on the underlying virtual or physical NFs (both c-plane and u-plane) to provide a fast re-location.

Radio Access Network Our Radio Access Network (RAN) implementation is based on the open source software suite srsLTE [15], which is extended to support multiple slices. This *Multi-slice RAN* builds on a modified version of srsLTE to support multiple slices on the same radio VNFs (the full implementation details of the baseline are available in Section 6.1), that we have extended to support fine-grained reconfiguration of radio resources (see Section 6.3.5).

Core Network ACHO employs an ad-hoc version of the CN functionality that has been specifically implemented for this purpose, as VNFs shall implement the c/e split paradigm. Moreover, our implementation is fully modular and follows the service based architecture (SBA). More specifically, the implementation of the Access Management Function (AMF), AUSF, User Data Management (UDM) and UPF functions is done in Python 3, and is detailed in Section 6.3.5.

6.3.4. New MANO functionality

The adoption of the c/e split requires novel MANO functionality, to enable the relocation of network functions, and new interfaces between the management and orchestration layers and the VNFs, to extract and install the contexts.

Hierarchical management and orchestration

To implement the relocation of VNFs within a running slice, we design a system that supports this functionality, following the recent efforts from the 3GPP [68] and ETSI [99]. Our design is illustrated in Fig. 6.14 and follows a hierarchical structure, with the following two main components:

(i) **A 3GPP management system** (top of the figure, as defined by [68]), which provides the entry point towards the business layers (i.e., the tenants that request a specific communication service) and manages services in the underlying network. We implemented these parts as Python modules, which includes the mapping of two communication services (namely, Enhanced Mobile Broadband (eMBB) and Massive Machine-type Communications (mMTC)) into two Network Function chains. In ACHO, we implement a reduced subset of the ones already defined by 3GPP. Namely, we logically select the VNFs that belong to each slice (including the sharing policies) and create their logic topology.

(ii) **An ETSI NFV MANO system** (top right) in charge of the central part of the network lifecycle management (i.e., instantiation, runtime, and termination). To implement this part, we have developed a composite implementation of the ETSI NFV MANO [99] stack. Specifically, we employ a base-line OpenStack as the Virtual

Infrastructure Manager (VIM), and then developed the other modules (i.e., Virtual Network Function Manager (VNFM) and NFV Orchestrator (NFVO)) as ad-hoc modules, in `Python`. Basically, we leverage OpenStack to trigger the instantiation of different VMs in our infrastructure, by using its API. Also, the interfaces towards the VNFs are implemented using `Python`.

New interfaces

We designed two new interfaces: one to extract and install the context, and another one to estimate network conditions, which is needed to support decisions about VNFs relocations. We denote these interfaces as `or-nfv` and `or-mon`, respectively (see Fig. 6.14). These interfaces can be considered as part of the already defined ETSI MANO reference points `or-vfnm`, `ve-vnfm-vnf` and `or-vi`, although other extensions may be considered. They are described next:

(i) **or-nfv**: This interface is used to extract and push the context of the VNFs. This interface is used by the Orchestrator (the NFVO), which is in charge of all the operational logic of a Network Slice. In particular, when deciding to relocate a function, the NFVO first extracts the context of the network function and then re-orchestrates this function, by pushing the context into the function available at the new location. This interface is similar to the one already included in the 5G system between the management service and the core network functions. This interface [100], connects the capabilities provided by the NEF and Network Repository Function (NRF) to extract and set configuration parameters from the network functions. In our implementation, following the current trends in network softwarization, this interface is implemented through a Representational state transfer (REST) API.

(ii) **or-mon**: This interface connects the VNFM with the VNFs through the SBA, and serves to monitor the Cloud-Native Network Functions (CNFs), to trigger a relocation when performance falls below a given target (although the VIM has some monitoring capabilities, they typically circumscribe to the Virtual Machines and not the VNFs). This interface is also similar to the one defined by 3GPP between the Network Data Analytic Function (NWDAF) available in the core and the management system. However, in our implementation (based on a REST API), we extend its focus by targeting different metrics (e.g. latency, in addition to load) and also including access functions.

6.3.5. Re-orchestrable VNFs

The proposed *c/e* split can be applied to any VNF, provided it implements the interfaces described above to extract and install the context. To show this, we have implemented different VNFs following the *c/e* split and thus making them “re-orchestrable.” We note that the *c/e* split nicely fits with the Software Defined Networking

(SDN) approach, which is an easy way to extract and inject the context from and to a VNF (i.e., in traditional SDN, the context of a switch are its forwarding rules). Thus, to implement the VNFs, for simplicity we have selected the Open Source **Lagopus** switch⁹ as basis for our implementation (alternatives such as ONOS [101] may be used for larger deployments). The diversity of the chosen functions shows the generality of our approach and the ability to apply it to any network function:

UPF: This function provides the encapsulation, decapsulation, and forwarding to the Packet Data Network. The implementation of this module follows the c/e split and includes the corresponding interfaces with our MANO system to extract and install the context. The context consists of the current rules applied to encapsulate/decapsulate GTP packets and to forward them. We have implemented the UPF module building on the **Lagopus** switch.

SMF: This c-plane function controls and configures the UPF instances on the u-plane through the N4 [67] interface. Thus, the context here also consists of the rules to encapsulate/decapsulate/forward packets, in this case for all the UPF functions controlled by the SMF. For the implementation of this module, we leverage available SDN-capable implementations, enriching them with mobile network functionality, and employing a **Ryu** Controller¹⁰ to implement the N4 interface between the UPF and the SMF, as well as the interface enabling the context management.

IoT broker: The Internet of Things (IoT) broker acts as middleware between the sensors connected to a mobile network and a data sink that may be located in a central location. We have implemented this module in **Python** from scratch, including specific libraries for the handling of traffic flows from the sensors. Our lightweight and flexible implementation allows to dynamically transfer the broker context to a new location, which is particularly suitable for Mobile Edge Computing (MEC) deployments, as it allows moving the broker functionality across different edge infrastructures.

Firewall: This network function forwards IP packets from an ingress to an egress port following a set of firewall rules. The context of this network function thus consists of these rules. We have implemented the u-plane part of this function as a **Lagopus** switch, and the c-plane part as an extended **Ryu** controller. The latter gathers the rules, which are stored as **Python** objects, and provides them to the MANO system through the corresponding primitives.

⁹<http://www.lagopus.org>

¹⁰<https://ryu-sdn.org/>

MAC scheduler: One of the main functions of Medium Access Control Layer (MAC) layer in LTE is the scheduling, which basically consist of assigning a given amount of resources to different users. The context of this function is, therefore, the amount of available resources, and the different users requesting them. Our implementation includes an interface at MAC layer level to enable a dynamic resource management: each time a user gets authenticated, the MAC layer notifies the orchestrator, which replies with the amount of resources to be assigned to this user. We use ACHO just on selected events, to allow enough stability on the radio link. Although the ACHO mechanism does not impose any constraint on the frequency of re-orchestration, each re-orchestration imposes a price in terms of resource re-allocation. Finally, by employing ACHO at the network edge, allows for a better network slice isolation, as demonstrated in Section 7.3.

6.3.6. ACHO adoption strategies

The adoption of ACHO in the state of the art architecture requires fundamentally two new features: (i) the introduction of new relocatable functions (Section 6.3.5) and (ii) their interaction with the MANO (Section 6.3.4). Indeed, they require an important re-structure of the current network implementation strategies, but we believe that the advantages brought by our approach (i.e., the possibility of a fast re-orchestration of network functions) will certainly be considered in the upcoming transition to novel paradigms such as the cloud-native network functions [102].

Still, the changes from the architectural perspective are limited and, in some cases, even already partially targeted by the current standardization work. Summarizing, the new architectural interfaces shall be able to expose:

- **Network parameters:** as discussed in Section 6.3.4, ACHO envisions a new interface between the VNF and the MANO domains, that is used to perform extraction and injection of the context to and from virtual appliances. This kind of approach is totally aligned with current trends of network softwarization, which propose a profound restructuring of interfaces with an API based approach.
- **Network resource models:** the *context* of a VNF is tightly bound with its internal state, which is represented by a set of parameters usually associated to different granularity levels: per user (such as the bearer information), per user group or slice (such as the IoT broker) or globally to the VNF (like the eNB configuration). All these aspects are discussed in Section 6.3.5.

Having this in mind, we next propose two implementation strategies that are aligned with the current efforts by SDOs.

- **Transparent mode:** While the network functions shall provide an API to extract and inject their *context*, its definition may be actually up to the vendor. Therefore,

the data blob comprising the context of a network function at a certain point in time can be transparently handled by the MANO through the *or-nfv* interface, which simply transfers it to another location. Then the consistency is provided internally by the VNF vendor. This is the strategy used in our implementation discussed in Section 7.3.

- **Exposed mode:** defining the parameters that are used by a VNF is a task that has already been carried out by 3GPP SA5 for management purposes. For instance, [103] defines such parameters list for every network function defined in the 5G Core and Radio Access Network (RAN). Thus, *context* can be exposed following a standardized approach, to enable inter-vendor migration and enhanced management functionality at the MANO side (e.g., extract the context from one VNF and split it into several virtual appliances).

7

Experimental evaluation

This chapter provides an extensive step-by-step experimental evaluation of the mechanisms introduced in Chapter 6. The main objective is to properly characterize each of the proposed mechanisms and their performance within a real deployment.

First, we start by evaluating the main enabler towards a slicing-aware protocol stack, the end-to-end network slicing implementation in Section 7.1. Then, in Section 7.2 we show the results of evaluating a multi-service deployment based on *POSENS* and implementing different mechanisms to properly accomplish low latency and flexibility. Finally, after identify limitations at the evaluated mechanisms to provide flexibility, in Section 7.3 we perform a complete evaluation of the mechanisms proposed in Section 6.3.

7.1. End-to-end Network Slicing implementation

We next evaluate *POSENS* by deploying the proposed implementation in a real testbed to first, validate the correct behaviour of the implementation, being then able to quantify the benefits of the proposed solution. More precisely, we evaluated the slices independence, performance in terms of throughput, ability of slice customisation and compatibility with commercial equipment.

7.1.1. Testbed Description

To properly evaluate *POSENS* (details in Section 6.1), we have deployed a testbed consisting of one UE implementing two slices, and one eNB connected to two different CN (one per slice). The testbed architecture is depicted in Figure 7.1.

The UE runs over an Ettus USRP B210 board connected to an HP OMEN laptop, running Ubuntu Linux 16.04. The eNB runs over another B210 board, connected to a Intel NUC running the same Linux distribution. The TX and RX ports of one B210 board are connected to the RX and TX ports, respectively, of the other board, using coaxial cables with SMC connectors to prevent any interference.

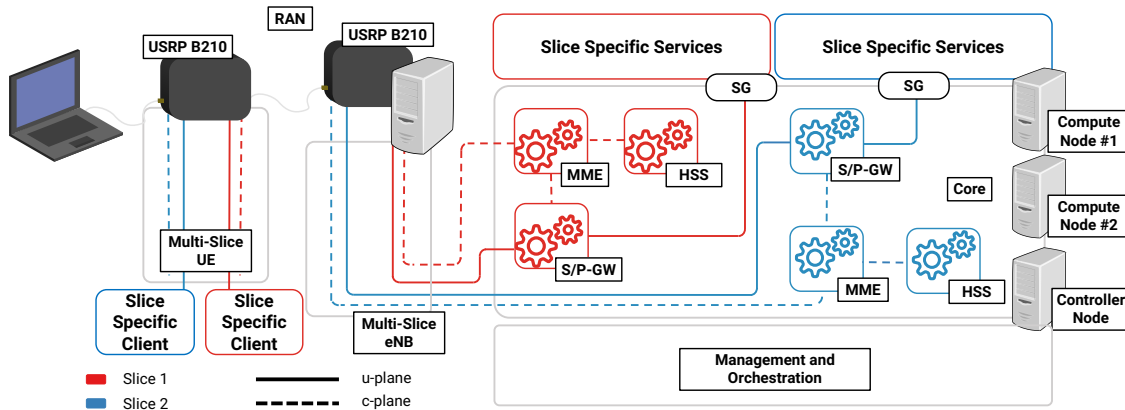


Figure 7.1: A multi-slice network architecture.

To implement the CN, we run two instances of the OpenAirInterface CN implementation, which contains the Mobility Management Entity (MME), Home Subscriber Server (HSS), Serving Gateway (S-GW), and Packet data network Gateway (P-GW). The OAI-CN VNFs are packaged in Ubuntu 16.04 VM, running in an OpenStack managed cloud composed of three compute nodes and one controller node.

Before performing the actual validation of POSENS, we first conduct an extensive evaluation of the best RAN (i.e., srsLTE) parameters that lead to the most reliable and stable configuration. To find a good trade-off between RAN performance (in terms of throughput) and stability, we set the channel bandwidth to 10MHz and a RX gain of 60 dB for the UE and 60 dB for the eNB. We used the LTE channel 7 (centered around 2600 MHz).

7.1.2. Independence between slices

We first validate that the slices can run simultaneously and independently, in this way supporting e.g., experimentation in scenarios with multiple slices, each one potentially re-configured in real-time. To this aim, the experiment starts with two configured slices, each one implementing a periodic request-response service between the UE and a server. We emulate that these servers are relatively far away by introducing an extra delay of 100 ms via the `tc` command. Then, after 20 s , the server of the second slice is moved to the eNB, simulating e.g. the use of a MEC-like solution. We represent the obtained performance in terms of average Round-Trip Time (RTT) across 10 repetitions in Fig. 7.2.

As the figure illustrates, at the beginning of the experiment both slices experience the same RTT of approx. 120 ms , with a few outliers across experiments. The re-allocation of the server in the second slice has an obvious impact on performance, with the RTTs immediately reduced to approx. 20 ms , while the performance with the first

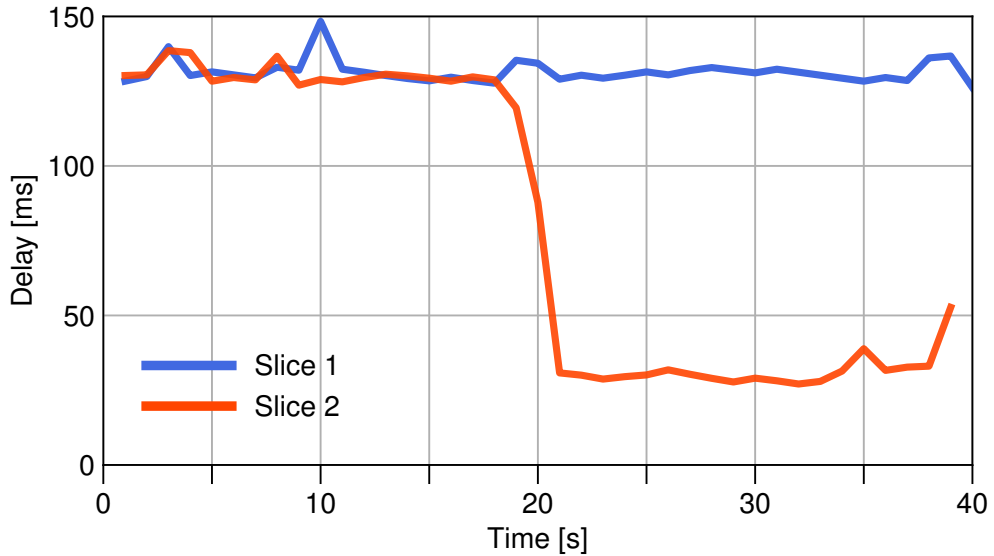


Figure 7.2: Independence between slices

slice remains unaltered. With this experiment, we thus confirm that researchers could prototype scenarios where different services are provided with different slices, and each service could be independently modified without altering the others.

7.1.3. Throughput performance

We next assess quantitatively the performance of our solution, to analyze if the overall efficiency is degraded because of the use of slicing, and if the slices are fairly sharing the available resources. To this aim, we start our experiment with both slices configured, but only one (“Slice 1”) performing a TCP download from a server. Then, after 20 s, another download is performed on the second slice (“Slice 2”), from a different server. We illustrate the per-slice throughput and the total throughput (“Aggregated”) averaged over windows of 1 s in Fig. 7.3. We also represent in the figure the throughput performance when no slicing is done, i.e., both the UE and the eNB use the vanilla version of srsLTE (“Single Slice”).

The figure illustrates two main results: first, there is practically no difference in total throughput between our implementation and the use of the vanilla version of srsLTE, which confirms the efficiency of the developed solution. Second, when both slices are active, they fairly share the medium, each one obtaining approximately 50% of the total throughput (we repeated the experiment several times and in all cases the performance was very similar).

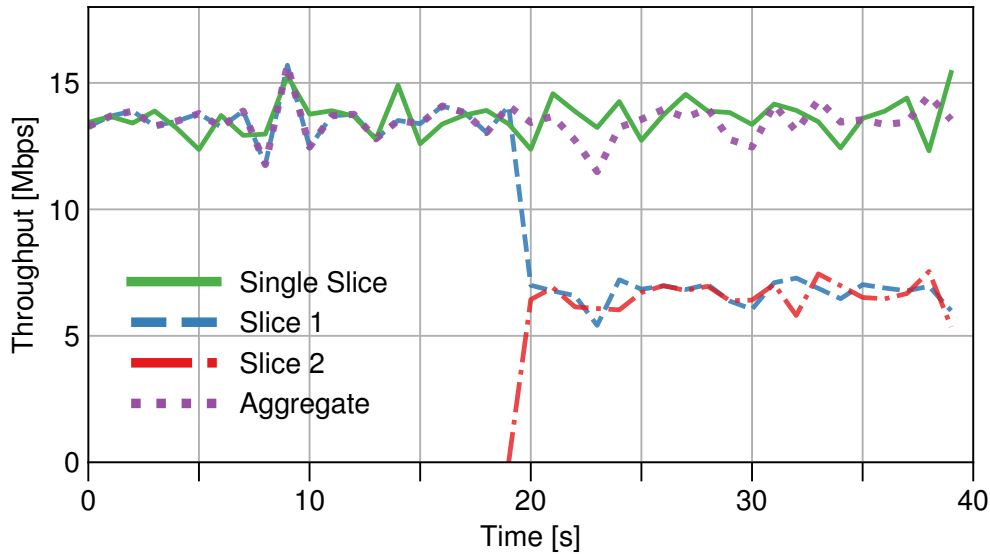


Figure 7.3: Total and per-slice throughput performance

7.1.4. Slice customization and orchestration

We next show how our solution supports a per-slice orchestration and customization of services, as well as the adjustment of the resources that a slice consumes. We demonstrate this capability by modifying in real-time the chain of VNFs that build a service. In particular, we insert two additional user plane functions into an operating slice: a traffic shaper and a firewall. Our experiment works as follows. We start with two slices serving downlink traffic to the UE, fairly sharing the channel as illustrated in Fig. 7.4. Then, after 20 s, we add into “Slice 2” a firewall function to block incoming connections and a traffic shaper function to limit the bandwidth to 2 Mbps. As the figure illustrates, the effect is immediate and “Slice 1” receives a higher throughput. We also confirmed that connections were blocked immediately. This shows that, even though our slicing solution cannot allocate RAN resources directly, it can control the overall resource consumption (including RAN) as long as terminals employ some congestion-aware sending mechanism.

7.1.5. Compatibility with commercial equipment

In this section, we confirm that our solution is compatible with commercial equipment. To this aim, we perform a connectivity test using a Nexus-5 phone, equipped with a Sysmocom programmable SIM card¹. To support this test, we slightly modified the hardware setup, attaching an antenna to the eNB Software Defined-Radio (SDR) card, and using isolation hardware (Ramsey electronics shielded enclosures) to prevent interference.

¹<http://shop.sysmocom.de/products/sysmousim-sjs1>

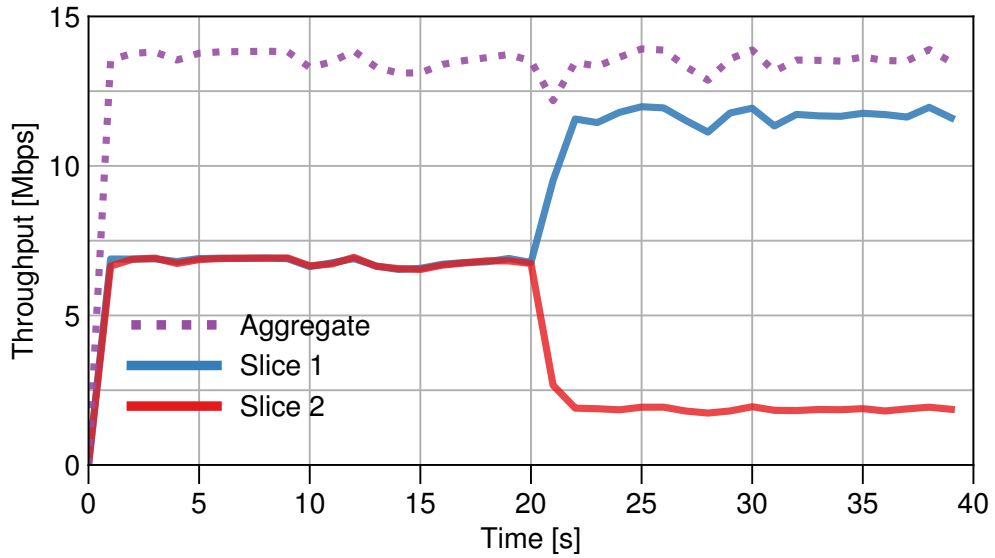


Figure 7.4: Independent Service Function Chaining

We confirmed that POSENS supports both modified UEs and commercial UEs, namely, slice-aware and slice-unaware UEs. In this way, we support scenarios where several UEs can be attached to the same slice (e.g., eMBB), and only a few UEs, in need of specific services, require the instantiation of a different slice (e.g., an Ultra-Reliable Low Latency Communications (URLLC) service). This further extends the applicability of our solution, opening it to a very wide range of testing scenarios.

7.2. Multi-service and multi-slice deployment evaluation

Once we have validated the baseline network slicing approach, we move forward to build a complete multi-service and multi-slice deployment where we conduct a complete experimental evaluation of the services introduced in Section 6.2.

7.2.1. Testbed Description

To conduct the experimental evaluation, the scenario is similar to the one depicted in Figure 7.1 but including the newly implemented mechanisms as well as the services to be tested. As we mention in the previous section, the full bandwidth achievable with this configuration is roughly 15 Mbps with good channel conditions.

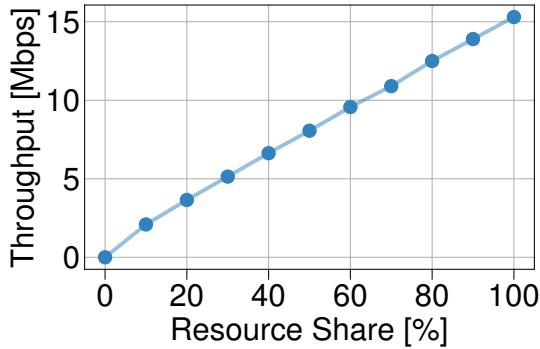


Figure 7.5: (a) Obtained throughput with variable resources shares vs share

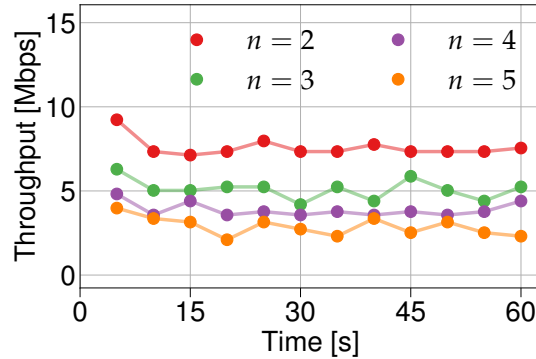


Figure 7.6: (b) Obtained throughput with variable resources shares vs time

7.2.2. Slicing-aware MAC Scheduling

In this section we test the implementation introduced in Section 6.2.2.2. To correctly evaluate the proposed scheduling scheme, we emulate the presence of other slices by leaving the resource blocks originally assigned to them blank. We then run an Iperf in Downlink (DL) direction with TCP traffic. Figures 7.5 and 7.6 show the correctness of our software implementation in terms of throughput for different sharing options and over time respectively.

In the first experiment we create 2 slices (i.e., $n = 2$) and assign an increasing share to the first slice (the one that holds the User Equipment (UE)) by modifying the vector \mathbf{W} . The results, shown in Figure 7.5, demonstrate the effectiveness of our implementation in correctly enforcing the amount of resources assigned to a slice. In the second experiment, we generate a variable number of slices and assign to them the same amount of resources. The results, depicted in Figure 7.6, show how our implementation can maintain the throughput around the assigned level for all the duration of our tests (60s).

Although this implementation, available in our repository, is not meant to implement a sophisticated scheduling policy with strong mathematical guarantees, it is indeed a first step towards the open implementation of one of the RAN slicing options depicted in Figure 6.1.

7.2.3. Service Creation Time

In this section we present the results achieved in our implementation with respect to the service creation time, one of the most important parameters targeted by the major stakeholders in the 5G environment such as 5GPP [104]. Therefore in Table 7.1 we summarize the timings that we measured in our setup. Intervals are obtained by taking timings of the Open Source MANO (OSM) or OpenStack primitives that are used to perform the different actions, grouped by the main lifecycle management primitives

Table 7.1: Service Creation Time KPI

Phase 1. Onboarding		
Time Components	What	Time
1.01	Network Slice Template (NEST)	7 minutes
1.02	Network Service Descriptor (NSD)	2 minutes
1.03	VNF package (VNFD)	5 minutes
Phase 2. Instantiation, Configuration and Activation		
Time Components	What	Time
2.01	Instantiate Network Slice (NSI)	30 seconds
2.02	Instantiate & Activate Network Service (NS)	45 seconds
2.03	Instantiate & Configure VNFs in service chain (VNF)	10 seconds
2.04	Configure other NFVI elements	10 seconds
2.05	Configure SDN infrastructure	5 seconds
Phase 3. Modification		
Time Components	What	Time
3.01	Modify Network Slice configuration	1.5 minutes
3.02	Modify Network Service configuration	2 minutes
3.03	Detect triggering condition	<1 second
3.04	Modify VNF configuration in service chain	<1 second

(onboarding, instantiation and modification).

Onboarding

Here we measure the times needed to actually create the service by uploading the YAML descriptors detailed in Section 6.2.5. Under the Network Slice Template (NEST) we group the two sub-timings devoted for the Network Slice Descriptor (NSD) and the Virtual Network Function Descriptor (VNFD). In our setup, which is limited by the available hardware capabilities, we can successfully create the two services in around 7 minutes. This is completely in line with the requirements set by e.g., 5G-PPP (“from 90 hours to 90 minutes”) [104].

Instantiation, Configuration and Activation

In this group we measure the timings needed for the activation of the single Virtual Network Functions (VNFs) that build the service function chain providing the services. Here (also due to the reduced size of the Virtual Machines involved), the timings are reduced to seconds. Also, some time is needed to configure other Network Function Virtualisation Infrastructure (NFVI) infrastructure such as the transport network. We remark that the timings obtained here are a subset of the Onboarding one. Onboarding requires more time for internal sanity checks performed by OSM before considering the operation completed.

Modification

As discussed in Section 6.2.5, we implemented in the Network Orchestration algorithms to perform e.g a VNF relocation. While the timings related to the detection of the triggering conditions and the decision to enforce a new orchestration policy are negligible (sub-seconds), the time needed to prepare the network for such re-orchestration is around a couple of minutes: similarly to onboarding, in fact, this operation requires the preparation of Virtual Machines (VMs) and additional sanity checks.

7.2.4. VNF re-location

One of the objectives of this multi-service deployment is to showcase advanced orchestration functionality, like a service-aware adaptive allocation of functions to different network nodes using VNF mobility concepts. Regarding this, we envisioned the possibility of relocating the VNFs between different compute nodes, which are acting as edge and central cloud nodes in a real 5G network.

For our particular case, we provide this feature by using the so-called “live-migration” functionality offered by OpenStack (which is our default choice for the OSM Virtual Infrastructure Manager (VIM)). Namely, we performed two different experiments, which we detail next.

The first one, based on the so-called “block live-migration”, just needs a network connection from the source node to the destination. Basically, the complete virtual machine is transmitted without service interruption. In our specific case, VNF instances can be quite big (tens of GBs once instantiated), so it would be necessary to have a very high bandwidth and dedicated network connection to achieve fast migration times. With a common network connection like the ones we have in our testbed the process takes times in the range of minutes, as discussed before. Moreover, while very high bandwidths may be achievable within the same datacenter, having them available in different geographical locations may be questionable. Furthermore, this approach also reduces the VM processing power during the migration time, so we considered this is not an acceptable approach for a future 5G network.

The second possible approach is called “shared storage live migration”. As the name states, it is based on using a shared storage which is accessible from both, source and destination host. In this case, performance has been acceptable (in the range of few milliseconds, see entry 3.04 in Table 7.1), without any service interruption.

Although the performance is quite good using the latter approach, this shared storage node is a drawback in itself. This procedure has well-known inconveniences, among others:

- (i) adding shared storage nodes requires to design the network topology according to this;
- (ii) the shared storage node becomes single point of failure;
- (iii) the network itself becomes a single point of failure;

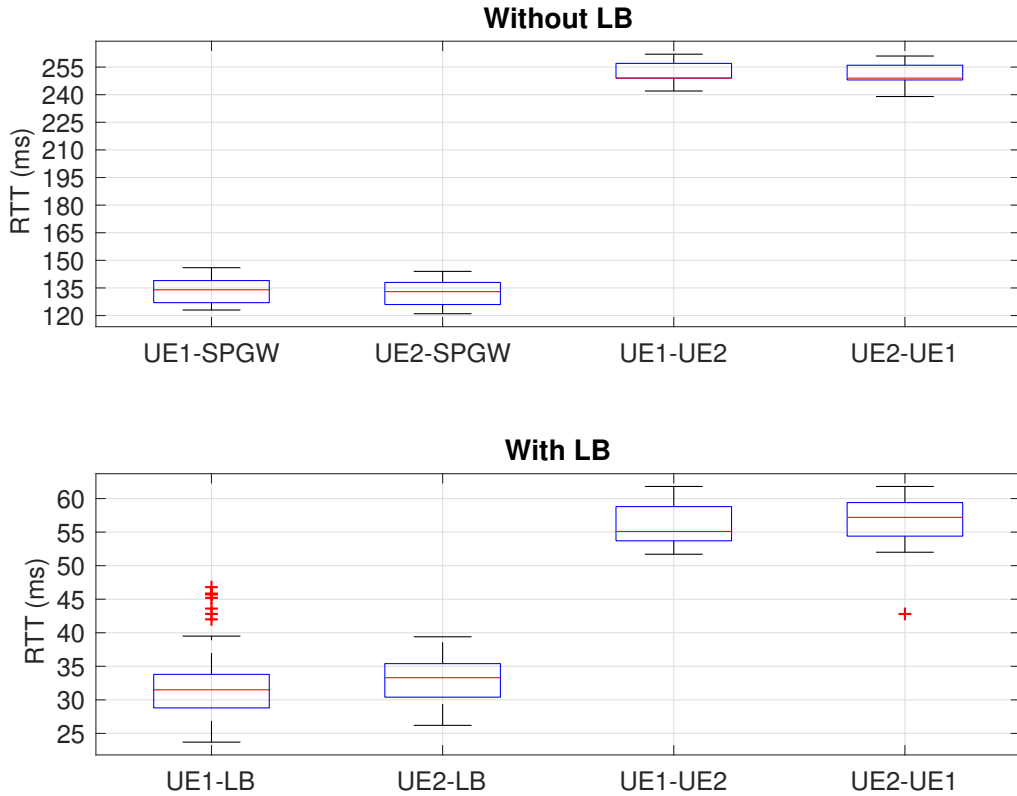


Figure 7.7: Latency evaluated for the URLLC scenario.

(iv) the network security shall be improved as the shared storage should be placed in a separate secured network, and

(v) even with the good performance results experienced in our demo case, the network latency could impact performance, especially for certain scenarios which require very low latency.

However, these problems may be solved in different ways. They range from reducing the size of the VMs to be migrated to the usage of containers [105] or unikernels [82]. Another possibility is the improvement of the shared storage option to tackle the known weaknesses, e.g., adding redundancy to the single points of failure and improving the security.

7.2.5. Low latency through LB

Continuing with the evaluation, this section is intended for the Local Breakout (LB) implementation introduced in Section 6.2.3. The experiments were performed by running the corresponding VNF in the slice providing the low latency service. We attached to our setup the multi-slice UE (UE1) presented in Section 7.1 and an additional single slice UE (UE2) that connected to the low latency slice only.

In one scenario, we co-located the Augmented Reality (AR) server with the gateway,

and compared it with another scenario where the AR server was placed together with the LB VNF. An additional delay of 100 ms was added between the eNB and the gateway, to emulate the distances between edge and central cloud.

The box and whiskers plot in Figure 7.7 shows the delay for a UE to server and UE to UE communications. The latter is not related to the provided Ultra-Reliable Low Latency Communications (URLLC) service, but it is useful to assess the performance of e.g., machine to machine communications. As expected, the latency with the LB VNF activated is much less than the normal case, with a median value of around 30 ms and 135 ms for the UE to Server case with or without LB respectively. The gap is even higher for the UE to UE case, with median latency figures of 55 ms and 255 ms respectively.

7.3. Flexible orchestration with ACHO

After evaluating the available state-of-the-art strategies and tools for flexible orchestration (see Section 7.2), we identified some limitations we overcome by means of Adaptive slice re-Configuration using Hierarchical Orchestration (ACHO) (Section 6.3). We now evaluate the novel mechanisms proposed by ACHO to properly validate and quantify their performance.

We evaluate the performance obtained by ACHO under a set of different metrics: (i) Virtual Network Function (VNF) **relocation delays** (Section 7.3.2), and (ii) **the re-orchestration of VNFs** discussed (Section 7.3.3).

7.3.1. Testbed description

To evaluate the performance of ACHO, we have deployed a testbed consisting of an access network and three datacenters, one acting as “central cloud” and two acting as “edge clouds,” which run the components presented in the previous section. Over this setup, three services (Enhanced Mobile Broadband (eMBB), Massive Machine-type Communications (mMTC), and Ultra-Reliable Low Latency Communications (URLLC)) are provided as illustrated in Fig. 7.9. Arrows serve to indicate the four re-orchestrations that we perform and are described in Section 7.3.3. We remark that, since the same User Equipment (UE) may connect to the same attachment point for different slices, the mobility management and authentication procedures can be shared across slices, and so are the Access Management Function (AMF), Authentication Server Function (AUSF) and User Data Management (UDM) functions (the “Shared Functions” in Fig. 7.9). This relies on the network function sharing functionality, which is mandated by 3GPP [67].

The experimental setup is depicted in Figure 7.8 and is an evolution of the testbed used in the previous section (Figure 7.1). This testbed is entirely composed of commodity hardware, which shows that ACHO does not have any particular hardware requirement. The access network consists of a physical UE and virtual UEs. The physical UE runs

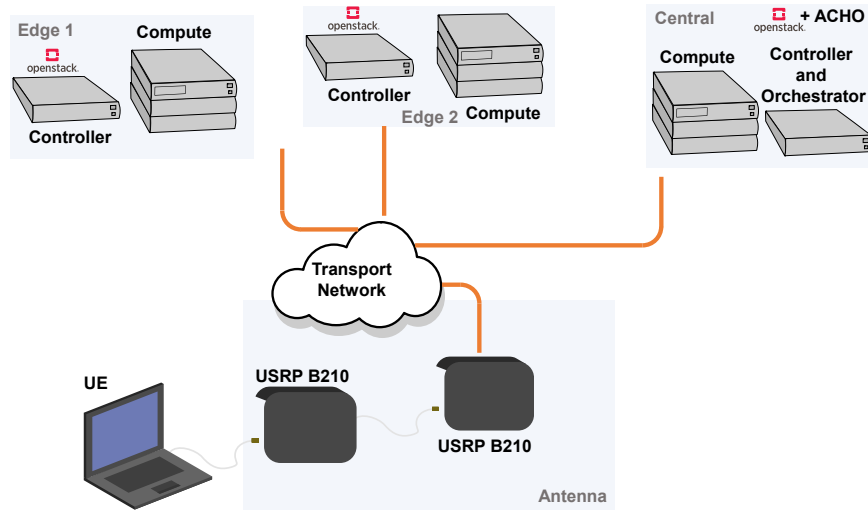


Figure 7.8: The Physical testbed setup.

in a laptop with Ubuntu 16.04, and the radio link is implemented by two Ettus USRP B210 Software Defined-Radio (SDR) cards cross-connected with RF cables. The multi-slice Evolved Node B (eNB) software runs in an Intel NUC with an Ubuntu 18.04. The same machine hosts the Virtual Radio Access Network (RAN) software for the mMTC deployment.

The datacenters run *OpenStack*, with one controller node that manages the virtual links connecting the VNFs. They are hosted in Ubuntu 16.04 servers, each server equipped with two network cards: one acting as the provider network (i.e., carrying the 3GPP Network traffic), and the other carrying the control and management traffic. The transport network connecting the different datacenters consists of four *Northbound Networks Zodiac FX* Openflow-enabled switches. To emulate long-distance links (i.e., between edge and cloud), we use the Linux traffic shaper `tc`.

7.3.2. VNF relocation delay

We start our evaluation by focusing on the delay to perform a relocation of a VNF, which is defined as the time elapsed between the Management and Network Orchestration (MANO) taking the relocation decision, and the moment in which the VNF is up and running in the new location.²

We measure the relocation delay for three of the VNFs described in Section 6.3.5: the UPF and the firewall (FW), each one running in a `nano` instance, and the IoT broker, which runs in a `small` instance (these Virtual Machine (VM) flavors are inspired by the Amazon EC2 service). For all the considered VNFs, we evaluate the relocation

²Note that we do not consider “live-migrations,” since available orchestrators such as *OpenStack* can only perform this type of migration when disk or memory is shared across locations, something unfeasible in the scenarios we consider (e.g., a VNF relocated to a different and possibly far node).

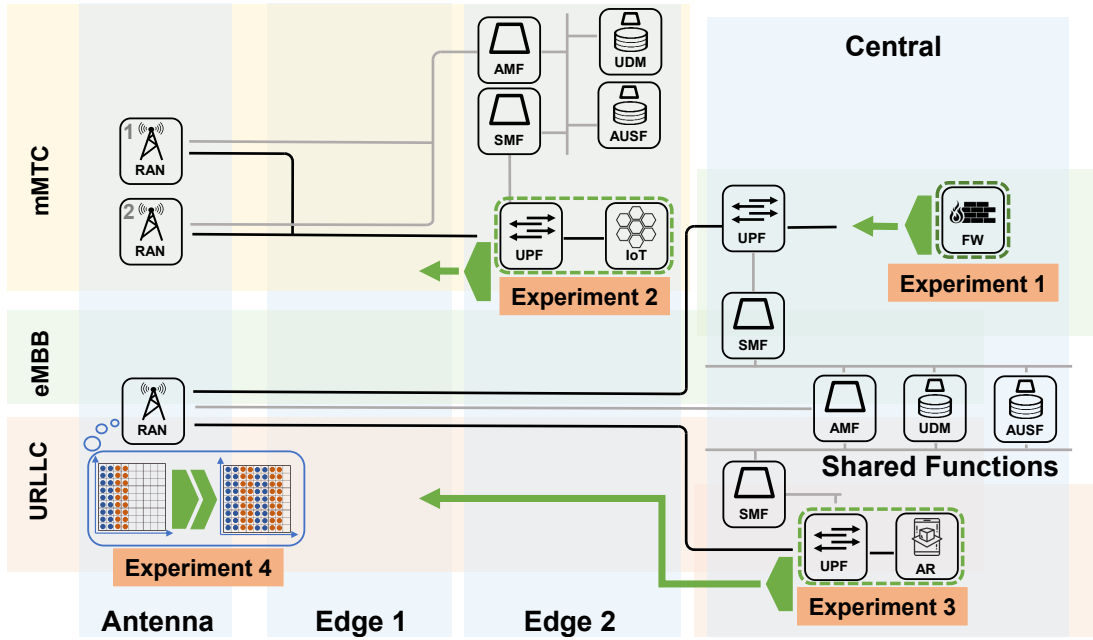


Figure 7.9: The Network Slice setup employed in the experimental evaluation, consisting of 3 slices.

VNF	ACHO				OpenStack
	Run	Pool	Cached	Non-C.	
UPF	70 ms	28.3 s	1 m 11.2 s	2 m 29.3 s	74 m 40 s
IoT br.	72 ms	28.8 s	1 m 5.7 s	2 m 29.2 s	89 m 35 s
FW	71 ms	27.7 s	1 m 3.3 s	2 m 27.3 s	59 min 48 s

Table 7.2: VNF relocation delays obtained by ACHO and by *OpenStack*.

delay incurred when using two different orchestration platforms: (i) ACHO, with four different configurations (discussed below), and (ii) the one obtained with *OpenStack* live migration. We provide the resulting relocation delays, corresponding to the average of 5 repetitions, in Table 7.2. This comparison allows us to quantify what are the advantages of a lightweight solution like ACHO with respect to a heavy migration technique such as the one provided by *OpenStack*. This scenario, which reflects a typical central cloud to edge cloud migration, cannot be properly handled directly through the Virtual Infrastructure Manager (VIM).

That is, the results confirm that *OpenStack* results extremely slow as compared with ACHO, for all the configurations. These configurations are: (i) *already running* (Run in the table), where the engine is already bootstrapped, (ii) *pool*, where the engine is already created in the new location, but not started; (iii) *cached*, where the target engine has already been started in the destination machine in the past; and (iv) *non-cached* (Non-C.), where the image of the engine is available at the new destination but has to be

created and bootstrapped for the first time.

OpenStack results order of magnitude slower than any of these configurations, as moving a VNF requires moving a full copy of the engine (including memory and disks). This is the main showstopper for the direct application of the live migration in an environment such as the one depicted here, in which a flat Network Function Virtualization (NFV) infrastructure may not be available. In contrast, delays are much smaller with ACHO, which furthermore enables having an engine already running in the destination node, thus making the relocation delay almost negligible (note that delays could be further reduced by employing more lightweight engines, such as, e.g., Containers or Unikernels). These results are also aligned with the ones provided in [106].

We finalize this section by analyzing the relocation delay of *SENATUS* [83], the orchestration framework closest to our proposal (as we discussed in Section 4.3.2). *SENATUS* leverages the native OpenStack Application Programming Interfaces (APIs) to perform a full snapshot of the image running the VNF before moving it to the new location, which requires the service to be stopped during the migration. Using similar images to the ones reported in [83]³, we obtained migration times of approx. 130 s, a performance comparable to ACHO’s *non-cached* configuration (in both cases, the image has to be created and bootstrapped for the first time). We note, however, that ACHO supports a “make before break” paradigm, as it only needs to stop the VNF in the old location when starting the context transfer, and not before. As a result, ACHO can re-orchestrate VNFs without any perceptible service interruption (as we confirm next), while *SENATUS* would incur in a service disruption during this 130 s interval.

7.3.3. Performance under re-orchestration

Next, we evaluate the impact of re-orchestration on performance. To this aim, we have performed four experiments:

Experiment 1: *Service function chain re-orchestration.* One key feature of ACHO is the ability to seamlessly modify the function chain of a service already running, i.e., adding or removing a VNF. We tested this feature in the eMBB network slice by adding a new firewall function to support a new requirement. Using the interface *or-nfv* described in Section 6.3.4, injecting the state is an atomic operation decoupled from the execution environment of the network function.

Our experiment starts with the eMBB slice serving three TCP flows, namely A, B, and C. After 30 s, we enforce a new policy by adding a firewall function into the slice and injecting the firewall rules (as context) through the *or-nfv* interface. These rules match flow A, which is immediately interrupted without affecting the rest of the flows of this

³*SENATUS* is evaluated using CirrOS images, which by default do not have a context as they do not run a proper VNFs. So we could not test ACHO’s mechanism against this setup.

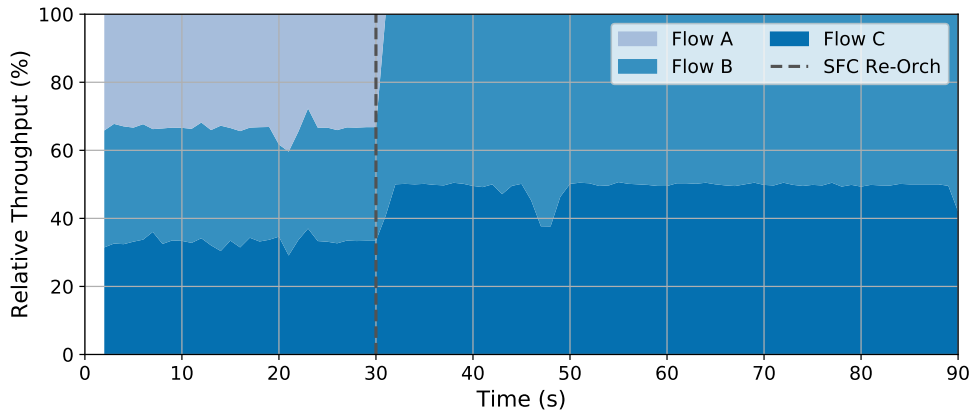


Figure 7.10: SFC amendment. Flow A (top), B (middle) and C (bottom).

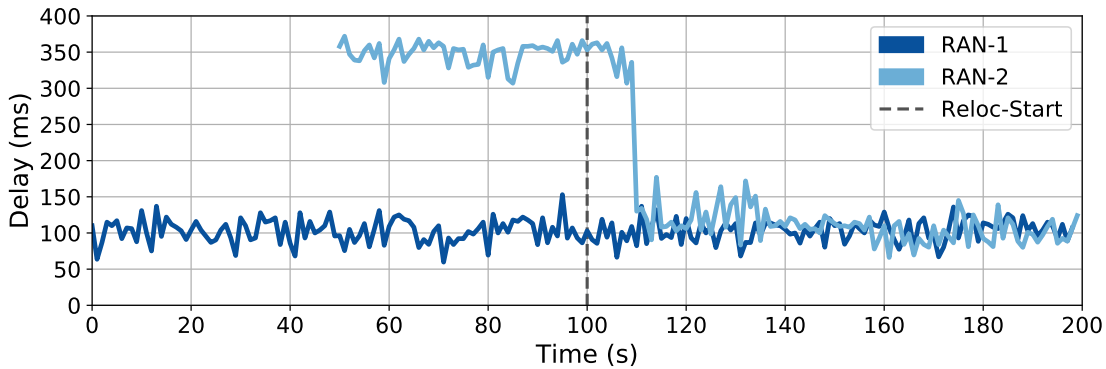


Figure 7.11: Relocation of the IoT gateway across edge clouds.

slice. We plot the throughput obtained by each flow in Fig. 7.10, which illustrates that re-orchestration does not disrupt the performance of the ongoing services.

Experiment 2: Follow-the-load VNF relocation. Next, we consider an mMTC service in a scenario with two RANs and two edge clouds. Initially, all the UEs are connected to RAN 1, which is closest to Edge 1 and therefore both the User Plane Function (UPF) and the Internet of Things (IoT) Broker application are orchestrated there. This results in a Round-Trip Time (RTT) for the application of approx. 100 ms, as Fig. 7.11 shows. Then, at time $t=50$ s, half the UEs are moved to RAN 2, which is farther away from Edge 1, this resulting in RTTs of approx. 370 ms. This performance degradation is detected by the MANO via the `or-mon` interface, which reacts by instantiating new UPF and IoT Broker in Edge 2 and, once these are available, relocating the context of those UEs that moved into them. This whole process (i.e., creating a new VM with the VNF image and, once ready, copy the context) takes approx. 30 s (which corresponds to the “pool” strategy in Table 7.2) and the service is never disrupted, nor for the UEs that stay in RAN 1 nor for

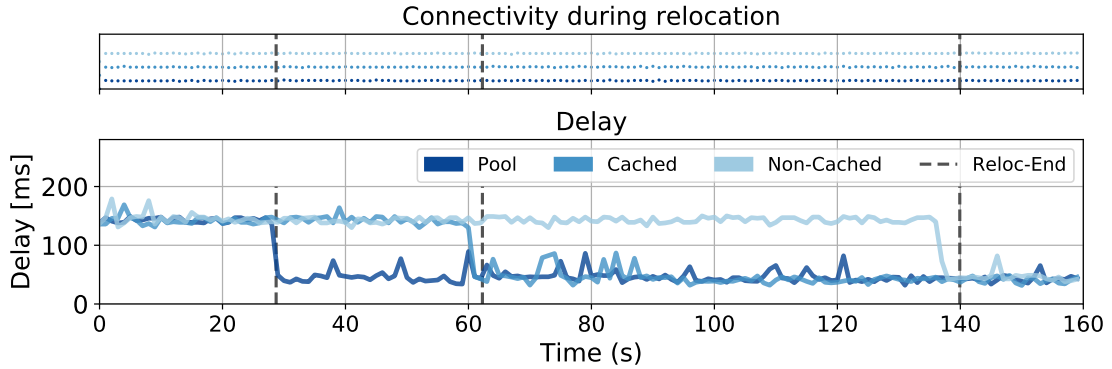


Figure 7.12: UPF migration from the central cloud to the edge cloud, under different configurations.

those that move to RAN 2. These results show the ability of ACHO to flexibly relocate only selected parts of a context.

Experiment 3: Bringing VNF closer to users. Next, we demonstrate the ability of ACHO to relocate VNFs inside the same slice. To this aim, we consider the URLLC slice, supporting a 600 kbps application that experiences a delay of approx. 150 ms. At some point, the MANO marks this delay as excessive and triggers a re-orchestration of the slice. This re-orchestration involves the relocation of the UPF and the low latency application (i.e., augmented reality in this case, marked as AR in Fig. 7.9), bringing both of them closer to the UE (i.e., from the central to the edge cloud). We analyze the resulting performance using three of the ACHO strategies discussed in Section 7.3.2, namely, Pool, Cached and Non-C. To this aim, we depict in Fig. 7.12 the performance since the MANO triggered the re-allocation in terms of connectivity (i.e., frames received, top subplot), and delay (bottom subplot).

The results confirm that (i) the re-orchestration is performed seamlessly towards the application, which perceives no disruption (i.e., no frames are lost), (ii) performance in terms of latency improves due to the relocation of the VNFs, (iii) migration delays (time between $t = 0$ and the thick black ticks in the figure) are in line with those presented in Section 7.3.2, with the “pool” strategy providing the smallest latency and the “non-cached” the largest one.

Experiment 4: On-demand radio resources assignment. In this experiment, we consider two users (UE1 and UE2) of a video streaming services. Each user requests at the beginning of the experiment a low quality video, therefore the orchestrator assigns the same amount of resources to each of them. At time $t=30$ s, UE1 requests a higher quality video (720p) and the orchestrator reacts by assigning more resources to that flow. Similarly, at time $t=60$ s UE2 requests a higher quality video, triggering a similar re-

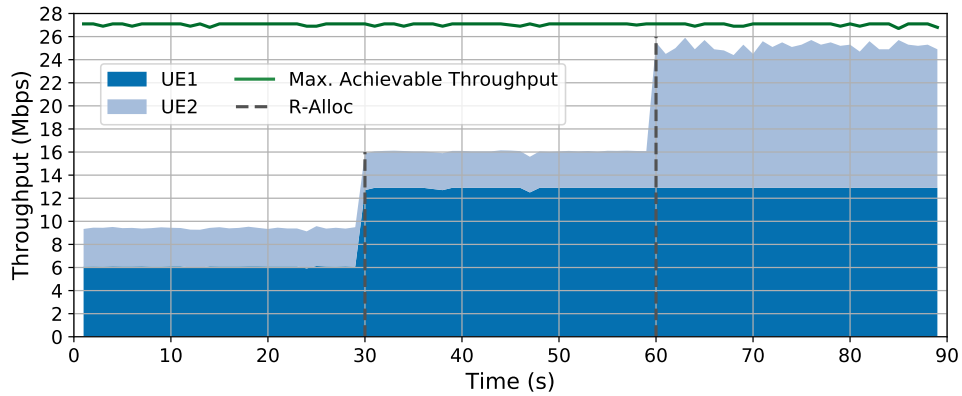


Figure 7.13: On-demand Radio resources assignment

configuration. We provide in Fig. 7.13, the resulting throughput obtained by each user.

The insights about the above reconfiguration are provided next. The eNB is configured with a bandwidth of 10 MHz of bandwidth, which translates into 16 Resource Block Groups (RBGs) of 3 Physical Resource Blocks (PRB), and 1 RBG of 2 PRBs. The initial assignment is 4 RBGs per *UE*, which supports the transmission of a 480p video. Then, at time $t=30$ s ($t=60$ s) the orchestrator assigns four more RBGs to *UE1* (to *UE2*) to support the transmission of a 720p video. As the Fig. 7.13 confirms, we can dynamically assign resources to UE with strong guarantees on their isolation (i.e., increasing the bandwidth for one UE does not affect the other).

8

Summary Part II

Part II contains the details for bringing network slicing to softwarized mobile networks and it is split into two chapters. Chapter 6 is dedicated to the study of the design alternatives for the required components as well as the implementation decisions with the corresponding explanation. Then, Chapter 7 dedicated to experimentally validate the proposed solutions as well as characterize their performance.

More specifically, Sections 6.1 and 7.1 are dedicated to introduce an end-to-end network slicing implementation, the design decisions we made to implement it and a complete validation and experimental evaluation respectively.

Then, in Sections 6.2 and 7.2, we moved to a complete deployment based on the aforementioned slicing implementation, where we deployed multiple novel services with diverse requirements that each network slice must satisfy. Moreover, we included an orchestrator to add a basic flexibility to the multi-service and multi-slice deployment.

Lastly, after identifying the limitations on the current orchestration mechanisms, in Sections 6.3 we propose a novel framework enable flexible orchestration based on a context/execution split to correctly manage virtual network functions, performing its evaluation in 7.3.

PART III

BRINGING CLOUD NATIVENESS TO SOFTWAREZIZED MOBILE NETWORKS

9

Designing a Cloud-native Radio Access Networks

As described in Section 4.2, network functions softwarization continues its path to finally conquer the very last mile of the network, the Radio Access Network (RAN). The virtualization of radio access networks (RANs), based hitherto on monolithic appliances over Application-Specific Integrated Circuits (ASICs), will become the spearhead of next-generation mobile systems beyond 5G [107,108].

Initiatives such as the carrier-led O-RAN alliance [69]¹ or Rakuten’s greenfield deployment in Japan [109]² have spurred the market—and so the research community—to find novel solutions that import the flexibility and cost-efficiency of network function virtualization (NFV) [110–114] into the very far edge of mobile networks [107,108].

Compared to purpose-built RAN hardware, Virtualized RANs (vRANs) pose several advantages [108], such as:

1. Leverage off-the-shelf platforms, which are more cost-efficient over the long-term due to economies of scale;
2. Harmonize the ecosystem, which helps reduce costs;
3. Leverage software development practices that shorten development cycles; and
4. Enable seamless integration of cloud technologies, which lowers barriers for competition and innovation.

Fig. 9.1 shows the architecture of a vRAN, with Base Stations (BSs) split into a Central Unit (CU), hosting the highest layers of the stack; a Distributed Unit (DU), hosting the Physical Layer (PHY); and a Radio Unit (RU), hosting basic radio functions such as amplification or sampling [115].

As depicted by the figure, vRANs shall rely on cloud platforms, comprised of pools of shared computing resources (mostly Central Processing Units (CPUs), but also *shared* hardware accelerators brokered by an abstraction layer³), to host virtualized functions

¹<https://www.o-ran.org/>

²https://www.altiostar.com/5g_cloud_native/

³See, e.g., **bbdev** (https://doc.dpdk.org/guides/prog_guide/bbdev.html)

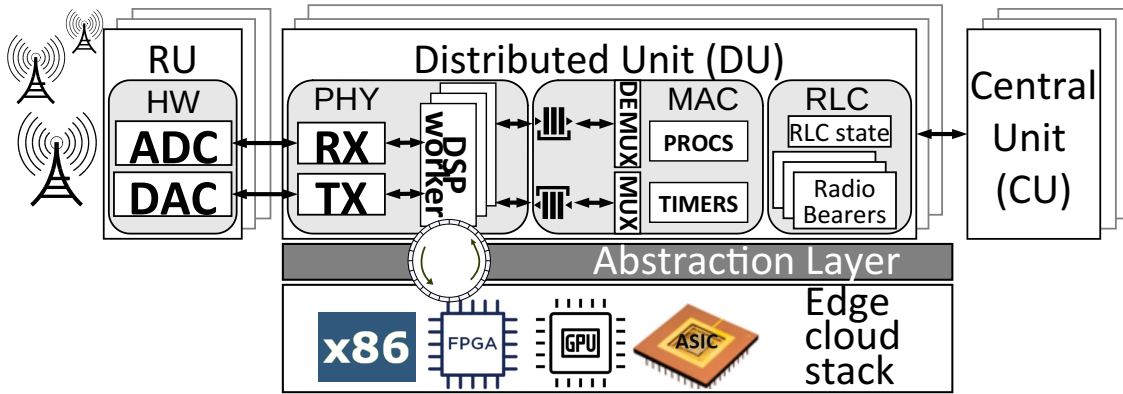


Figure 9.1: Virtualized RAN architecture [12]

such as the PHY [12].

However, while CUs are amenable to virtualization in regional clouds, Virtualized Distributed Units (vDUs)—namely, the vPHY therein—require fast and predictable computation in edge clouds [12, 108, 109]. Clouds provide a harsh environment for DUs because they trade off the predictability supplied by dedicated platforms for higher flexibility and cost-efficiency [111, 114].

Indeed, research has shown that resource contention in cloud infrastructure, even when placing virtual functions on separate cores, may lead to up to 40% of performance degradation compared to dedicated platforms [112, 114]—the so-called *noisy neighbor* problem.

This is certainly an issue for traditional network functions such as virtual switches, firewalls, Virtual Private Networks (VPNs), or even CUs, which *only* suffer a performance degradation that is proportional to the computing fluctuations caused by resource contention. For contemporary 4G/5G PHY pipelines, nevertheless, such fluctuations are simply catastrophic, as we demonstrate next. Consequently, *the main challenge for DU virtualization is to design a virtual PHY processor that preserves carrier-grade performance in cloud platforms at the edge.*

The problem

Fig. 9.2 illustrates the operation of a digital signal processor (DSP) that is used commonly to implement a 4G/5G PHY [15, 108, 116]. We focus on Frequency Division Duplex (FDD) where Uplink (UL) and Downlink (DL) transmissions occur concurrently in different frequency bands, which has been the most successful scenario in LTE. Moreover, to simplify our design, we focus on 5G’s baseline numerology ($\mu = 0$ in 3GPP TS 38.211), which yields one Transmission Time Interval (TTI) per subframe. In this context, every TTI, a *worker* initiates a *job* comprised of a *pipeline* of *tasks*: (i) processing an UL subframe, (ii) scheduling UL and DL grants, and (iii) compiling a DL subframe.

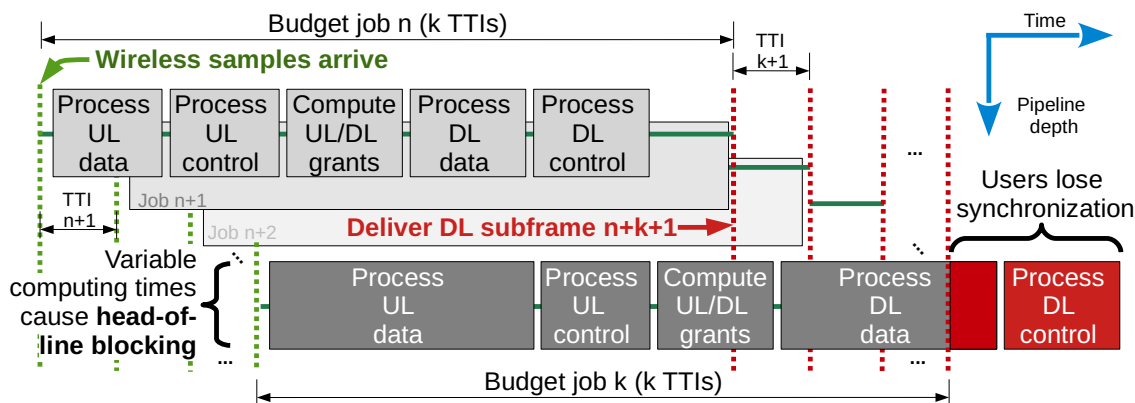


Figure 9.2: Baseline DSP pipeline parallelization. One subframe every transmission time interval (TTI, 1 ms)

These are compute-intensive tasks and hence processing a job within a TTI is challenging. For instance, Fig. 9.3 depicts the time it takes for a general-purpose CPU to decode a transport block of data from one user—just one out of many operations when processing an UL subframe—encoded with a mild modulation and coding scheme (3GPP MCS index 17) in a 20-MHz channel, and enduring different Signal-to-noise ratio (SNR) settings between 5 a 35 dB.

However, processing a DSP job *every* TTI is vital to preserve synchronization and process control information. To give workers some slack, *pipeline parallelization* is commonly used, that is, a pool of workers processes multiple jobs in parallel as shown in Fig. 9.2. In this way, with a pool of k workers, each job n gets a computing budget of roughly k TTIs to process UL subframe n (corresponding to the n^{th} TTI) and compile DL subframe $n + k + 1$ (carrying DL signals during the $(n + k + 1)^{\text{th}}$ TTI).

Albeit this approach is the basis for most of the work conducted to date [58, 117–119], we argue that *it is not sufficient* to attain carrier-grade performance over clouds. We show this with an experiment with two DUs, virtualized over Linux containers and sharing 5 Intel Xeon x86 cores @ 1.9GHz.

In this experiment, vDU 1 transmits and receives as much data as possible. Conversely, vDU 2 transmits and receives traffic following a random process with different parameters, which generate normally-distributed computing workload with the mean and variance shown at the bottom of Fig. 9.4: the higher the load variance of vDU 2, the larger the fluctuations of the computing capacity available for vDU 1. Fig. 9.4 (top) depicts the throughput of vDU 1 in yellow (“Baseline”) as a function of the workload produced by vDU 2. The figure shows that the performance of vDU 1, which uses the baseline pipeline introduced above, quickly deteriorates in the presence of computing capacity fluctuations.

Our analysis in Section 9.2 studying the latent factors behind such performance collapse can be summarized as follows:

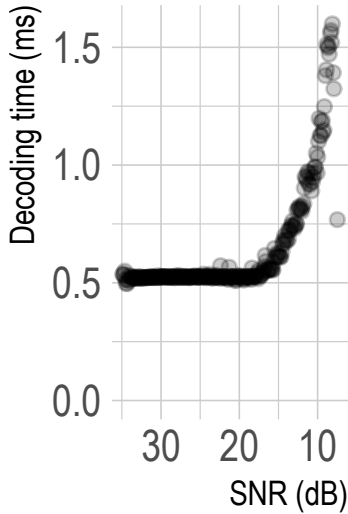


Figure 9.3: Decoding time of one transport block in a CPU core.

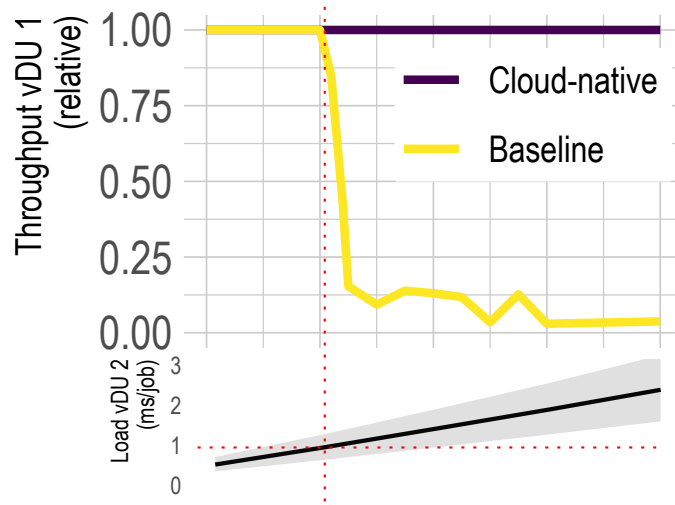


Figure 9.4: Two vDUs competing for computing resources. The UL/DL data load of vDU 1 is the highest possible while vDU 2's is variable.

1. The PHY has tight inter-task deadlines, which *constraints the maximum pipeline depth* ($k = 3$, usually) and, hence, the computing budget of each DSP job (~ 3 ms, usually);
2. Inter-task dependencies hinder task parallelization and causes *head-of-line-blocking* (the scheduler needs feedback from UL tasks, DL tasks need grants, etc.); and
3. The time incurred by data processing tasks depends on the users' behavior, channel dynamics, and cloud computing fluctuations, which gives in to *unreliability*.

The latter is shown at the bottom of Fig. 9.2, which illustrates an example where data processing tasks take longer time. As a consequence, the execution of the remaining tasks gets delayed (head-of-line blocking), which plays havoc with the ability of the vPHY processor to execute a DSP job every TTI if the pipeline depth is not sufficiently high (unreliability).

The solution

The obvious solutions applied today in the market [107–109], namely, *dedicated* hardware acceleration and aggressive over-dimensioning, diminish the very reasons that make virtualization appealing for the RAN in the first place: flexibility and cost-efficiency.

On the one hand, research has shown that clouds require 5x more resources than dedicated platforms to attain similar performance guarantees in real mobile networks [120]. On the other hand, dedicated accelerators make vDUs more expensive and power-hungry than their pure hardware counterparts [121]—let alone the fact that the much-longed hardware/software decoupling is not achieved.

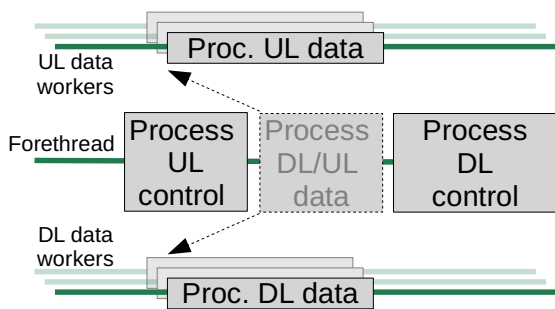


Figure 9.5: Dedicated workers for UL/DL data processing tasks. *Forethread* coordinates subframe processing.

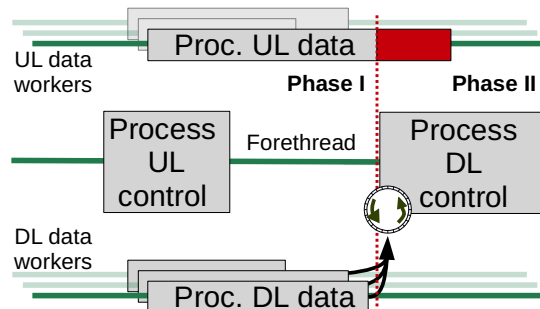




Figure 9.6: Computing allowance, split into two phases, impose a hard deadline on UL/DL data workers.

Conversely, the research community either (i) strives to expedite the processing time of some operations involved in DSP tasks (usually forward error correction, by exploiting SIMD instructions, intra-task parallelization, and other software optimizations) [122]; or (ii) resorts to compute-aware radio scheduling methods [57], which can only track slow and predictable dynamics.

These solutions certainly help but *they do not tackle the core problem: a DSP pipeline architecture that is unable to accommodate cloud computing fluctuations*. Even if hardware accelerators are shared *à la cloud*, as shown in Fig. 9.1, queueing in the abstraction layer brokering access to the accelerators fall into similar issues [12]. Hence, we must find *new* solutions to enable carrier-grade vRANs without compromising the advantages of virtualization.

In this work, we take a step back, look at the PHY's architecture as a whole, and design a supple DSP processor that is suitable for clouds. Our design follows two objectives: (i) *resiliency* upon deficit of computing resources; and (ii) *task parallelization* to efficiently exploit multi-core clouds. To this end, we rely upon the four techniques illustrated in Figures 9.5, 9.6, 9.7 and 9.8:

- (Fig. 9.5) We parallelize UL and DL data processing tasks with *dedicated workers*, in addition to performing pipeline parallelization. These are coordinated by a *forethread*, and are initiated as early as possible every DSP job to maximize their computing allowance.
- (Fig. 9.6) We grant UL and DL data workers a *fixed time budget* to accomplish their tasks and terminate them upon hard deadlines (red mark)—this ensures a *minimum viable subframe* every TTI that preserves synchronization. Moreover, a ring buffer  allows us to absorb jitter when encoding DL data.
- (Fig. 9.7) We deploy *predictors*  to infer the decodability of UL data upon reaching the forethread's deadline. This allows us to provide additional slack to UL data workers

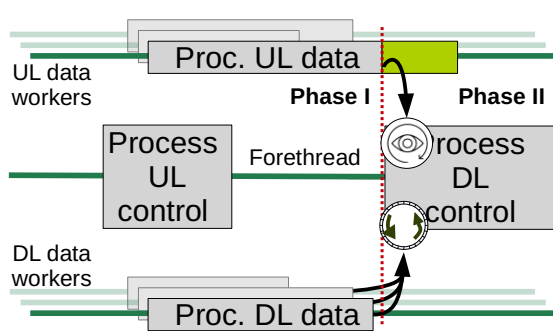


Figure 9.7: *Early HARQ* predicts the *decodability* of unfinished UL data tasks.

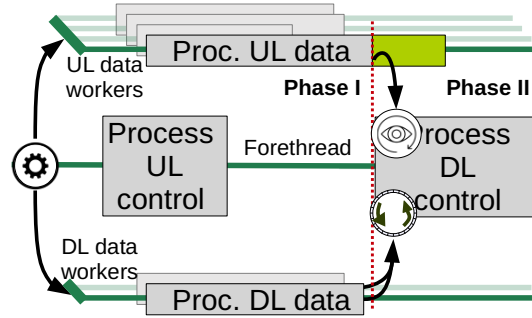


Figure 9.8: Congestion controller adapts the generation of data tasks to the available computing capacity.

and, hence, to minimize resource wastage.

(Fig. 9.8) We integrate a simple congestion control mechanism \odot to dynamically adapt the rate of workload (grants) to the availability of computing capacity.

In this way, in coalition with DSP task optimizations such as those in [122], the proposed cloud-native approach takes a firm step forward to rid of dedicated hardware. As shown by the purple line in Fig. 9.4, our proposed cloud-native approach sustains maximum throughput despite severe fluctuations in computing capacity. The details of our design and is presented in Section 9.3, while a thorough experimental assessment is exposed in Chapter 10.

9.1. Fundamentals of 4G and 5G

In this section, we first introduce the fundamentals of 4G LTE and 5G New Radio (NR) that are relevant to understand this work (Section 9.1). Then, we present the details of the baseline PHY pipeline architecture introduced earlier (Section 9.1).

A primer in LTE & NR PHY

We next review characteristics of 4G LTE and 5G NR PHYs that are relevant for our work. The interested reader may find more details in [123, 124] and references therein.

NR adopts Orthogonal Frequency Division Multiplexing Access (OFDMA) with Cyclic Prefix (CP) for both DL and UL transmissions, which enables fine-grained scheduling over a time-spectrum *grid*, and Multiple-Input Multiple-Output (MIMO) techniques. While LTE also adopts OFDMA in the DL, it relies on single-carrier FDMA (SC-FDMA) for UL, a linearly precoded flavor of OFDMA that reduces peak-to-average power ratio in mobile terminals.

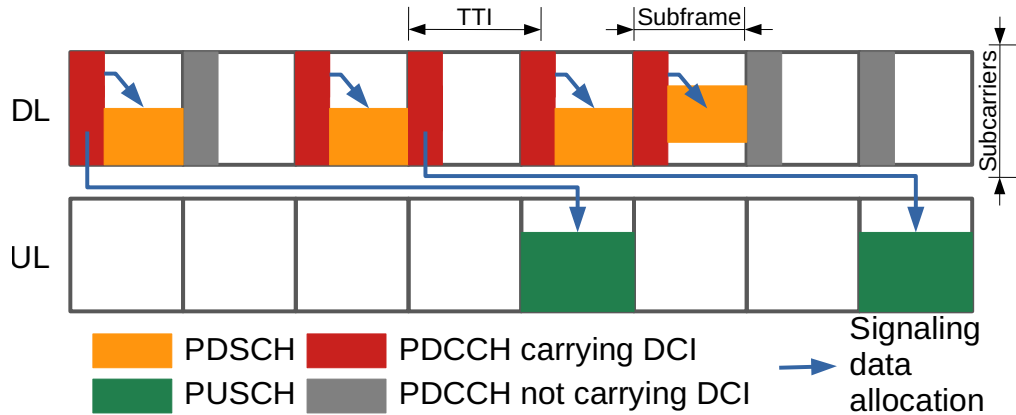


Figure 9.9: Subframes and PHY radio channels (FDD).

An example of the radio operation for FDD is depicted in Fig. 9.9. In the time domain, each frame is comprised of 10 Subframes (SFs), each with 1-ms duration. Each SF comprises 2 slots, each with 7 (with normal CP) or 6 (with extended CP) Orthogonal Frequency Division Multiplexing (OFDM) symbols for LTE; and one or more slots, each with 14 (with normal CP) or 12 (with extended CP) OFDM symbols for NR, depending of the (flexible) OFDM numerology employed, which is configurable in 5G. Accordingly, a TTI, which specifies the time resolution for scheduling, is equal to 1 ms in LTE and configurable to one or multiple slots in NR. Without loss in generality, we will assume a TTI is equal to 1 ms (baseline numerology in NR) to simplify our explanations. In the spectrum domain, SFs comprise a number of subcarriers with inter-subcarrier spacing equal to 15 kHz in LTE and variable, between 15 KHz and 240 kHz, for NR. LTE and NR support different bandwidth configurations up to 20 MHz and 100 MHz, respectively.

The time-spectrum grid is divided into *channels* which, although LTE and NR map into mildly different formats and grid allocations, are conceptually similar. Table 9.1 provides a summary of these PHY channels.

Downlink Channels and Signals	
PDSCH	Physical DL Shared Channel: <i>Carries user data, higher-layer user information and paging, as indicated in PDCCH.</i>
PDCCH	Physical DL Control Channel: <i>Carries resource assignments and UL scheduling grants.</i>
PBCH	Physical Broadcast Channel: <i>Carries basic information about the BS, e.g., bandwidth.</i>
PHICH	(LTE only) Physical HARQ Indicator Channel: <i>Carries UL Hybrid-ARQ feedback.</i>
PCFICH	(LTE only) Physical Control Format Indicator Channel: <i>Indicates the format used for PDCCH and PHICH.</i>
PSS/SSS	Primary/Secondary Synchronization Signals: <i>Signals used for synchronization and BS identity.</i>
Uplink Channels and Signals	
PUSCH	Physical UL Shared Channel: <i>Carries user data as indicated in PDCCH and, optionally, UL Control Information (UCI).</i>
PUCCH	Physical UL Control Channel: <i>Carries UCI including feedback (HARQ, channel quality, etc.) and scheduling requests.</i>

Table 9.1: LTE & NR Channels

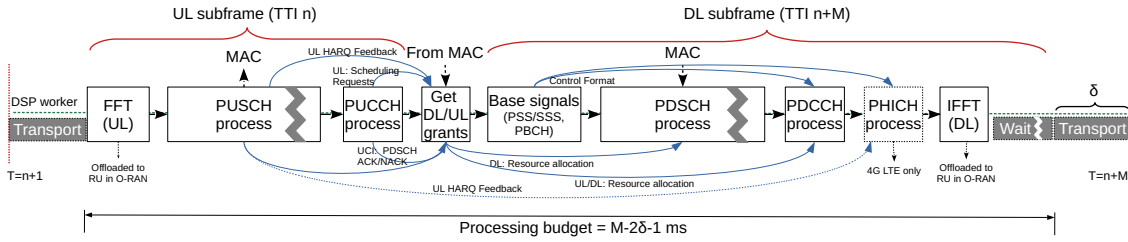


Figure 9.10: LTE and NR PHY pipeline: DSP job n

A Physical Resource Blocks (PRB), comprised of 12 subcarriers and 1 slot, is the smallest resource unit that can be allocated; and a Transport Block (TB) carries data using a variable number of PRBs. The size of a TB depends on the state of data buffers and the selected Modulation and Coding Scheme (MCS), which in turn depends on the SNR. Every TTI, Physical Downlink Shared Channel (PDSCH) (for DL) and/or Physical Uplink Shared Channel (PUSCH) (for UL) carry one TB (or two, in some MIMO settings) for each user.

The grid allocation of TBs are indicated by DL/UL *grants*, computed by the Medium Access Control Layer (MAC) scheduler, and carried by PDCCH's Downlink Control Information (DCI), as shown in Fig. 9.9.

Hybrid Automatic Repeat Request (HARQ), combining Forward Error Correction (FEC) and Automatic Repeat Request (ARQ), is used for error control. To this end, explicit feedback is received from the users in Uplink Control Information (UCI) messages carried by PUSCH or Physical Uplink Control Channel (PUCCH), and TBs are encoded with low-density parity-check codes (NR) or turbo codes (LTE).

Baseline DSP pipeline

A vPHY processor hosts a set of DSP workers. Every TTI, an idle worker handles DSP job n , which consists of the pipeline shown in Fig. 9.10 [15, 108, 116]:

1. Process uplink subframe n (received during TTI n):

- 1.1. First, wireless samples corresponding to the n^{th} UL SF are transformed into OFDM symbols by performing Fast Fourier Transformation (FFT) and CP removal.⁴
- 1.2. Then, the PUSCH and PUCCH channels are demodulated, decoded and processed, providing UL TBs (data) and UL feedback (DL HARQ, channel quality, scheduling requests) to the MAC layer.

2. Compute UL/DL grants carried by DL SF $n + M$: Schedule resources for DL data and UL requests, considering HARQ and channel quality feedback. The

⁴ This task may be offloaded to the RU in O-RAN.

choice of M establishes the pipeline depth of the vPHY (more later).

3. Process downlink subframe $n + M$:

- 3.1. First, the base signals are encoded and modulated into the DL SF template, including Primary Synchronisation Signal (PSS)/Secondary Synchronisation Signal (SSS), Physical Broadcast Channel (PBCH), and, in case of LTE, Physical Control Format Indicator Channel (PCFICH).
- 3.2. Then, the PDSCH and the Physical Downlink Control Channel (PDCCH) are processed, encoded, and modulated, according to the UL/DL grants.
- 3.3. Finally, the encoded OFDM symbols corresponding the DL SF $n + M$ are converted into wireless samples by adding CP and performing Inverse Fast Fourier Transformation (IFFT). The corresponding wireless samples are then sent to the radio transmission chain, with sub-ms transportation delay δ , at time $n + M$ (note the “wait” at the end).⁴

Existing vRAN solutions, often built on top of Intel’s FlexRAN, expedite the execution of individual tasks using SIMD extensions and intra-task parallelization, or they are simply offloaded into hardware accelerators [122].

The choice of M sets a maximum allowance of $M - 2\delta - 1$ ms for each worker to finish a DSP job. This is because DL SF $n + M$ must be delivered to the RU precisely at time $n + M$. Therefore, this approach enables a maximum pipeline depth of $M - 1$ workers processing DSP jobs in parallel: DSP worker 1 processes DSP job n (i.e., DL SF n and UL SF $n + M$), worker 2 processes DSP job $n + 1$ (i.e., DL SF $n + 1$ and UL SF $n + M + 1$), and, in general, worker $M - 1$, processes job $n + M - 2$ (i.e., DL SF $n + M - 2$ and UL SF $n + 2M - 2$). Fig. 9.11 shows an example with $M = 4$ (the usual case): note that, because each worker must finish up in ~ 3 ms, there can only be a maximum of 3 workers processing DSP jobs in parallel at any given time.

9.2. LTE and NR pipeline diagnosis

The design presented in Section 9.1 is not suited for clouds *without the assistance of dedicated hardware acceleration or aggressive over-dimensioning* [108], which compromise flexibility and cost-efficiency [121]. We have identified the following issues.

Timing constraints. PHY tasks have hard time constraints. In LTE, the latency between PUSCH reception and delivery of HARQ feedback shall be within 4 ms. NR has other constraints: K_0 —delay between DL allocation (in PDCCH) and the associated data delivery (within PDSCH); and K_3 —delay between ACK/NACK reception in UL UCI and the corresponding DL re-transmission (PDSCH).

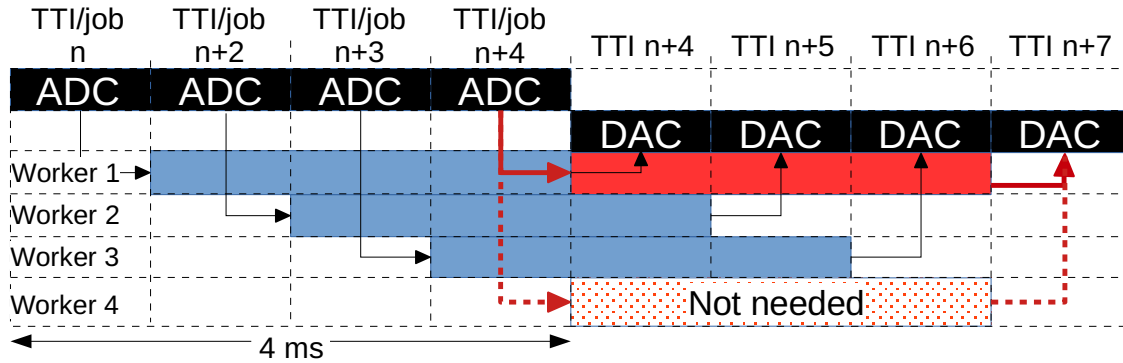


Figure 9.11: vPHY pipeline parallelization for $M = 4$ and four DSP workers in a worker-time grid. Colored cells represent the computing budget of a worker to process a DSP job. Because each job must be completed within 3 ms ($M = 4$), worker 1 *must* be available to process job $n + 3$, which makes worker 4 always idle (dashed cells) and it is hence *redundant*. Therefore, the maximum pipeline depth is 3, i.e., 3 parallel workers.

Though these timings are tunable, they are configured at longer timescales by the CU. As a consequence, these timing settings impose a deadline to process each DSP job and, hence, caps the maximum pipeline depth at the vPHY. In practice, setting $M = 4$ is the usual choice [125] (mandatory for LTE), which enables up to 3 useful parallel workers as in Fig. 9.11.

Head-of-line blocking. Regardless of individual task optimizations [122], different PHY processing tasks (such as PDSCH encoding, PUSCH decoding, or PUCCH decoding) are strongly coupled.

For instance, DL resource allocations must be computed before actually mapping the PDSCH into the resource grid; all UL TBs in PUSCH must be decoded before computing UL scheduling grants; and so on.

As a result, known implementations (e.g., Samsung’s vDU [108], `srsRAN`⁵ and `OpenAirInterface`⁶) perform each DSP job in a single-thread pipeline (Fig. 9.10), or in a multi-thread pipeline where each thread has to wait and be executed in a precise order [116], which boils down to Fig. 9.10 again. *This prevents task parallelization and causes head-of-line blocking, which averts an efficient use of multi-core clouds to expedite job executions.*

Unreliability. As hinted in our toy experiment shown in Fig. 9.4, the computing time required by DSP tasks highly depends on the instantaneous availability of computing resources. We note moreover that the most compute-intensive tasks also depend on the context, that is, on the data load (rate of TBs to decode/encode) and on the mobility

⁵<http://www.srsran.com/>

⁶<https://www.openairinterface.org/>

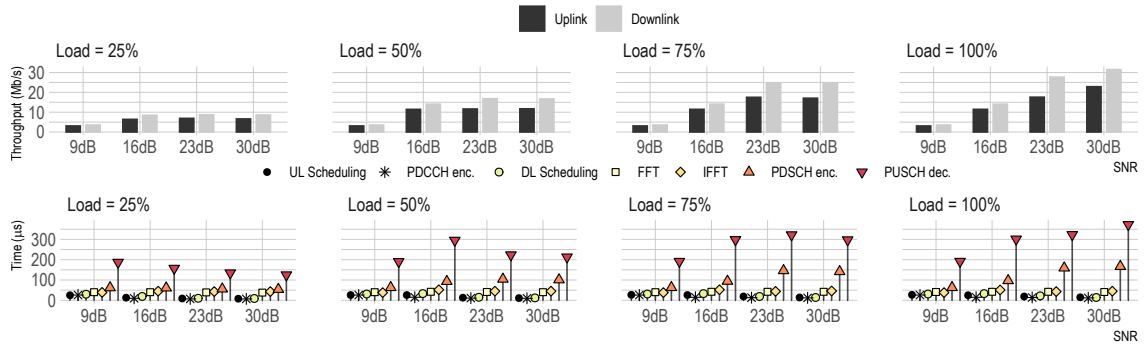


Figure 9.12: Throughput performance for both uplink and downlink (top). CPU time required by different PHY layer functions (bottom). Different uplink/downlink load (relative to the maximum) and channel conditions (SNR).

patterns of the users (signal quality) [58], which can induce very quick fluctuations in the demand for computing resources.

To illustrate this, we deploy our baseline vDU, implemented in `srsRAN` [15], processing downlink and uplink traffic over one Intel i7 core in a 10-MHz band. Fig. 9.12 depicts the achieved throughput in both the uplink and downlink (top subplots), and the median time incurred by the CPU to perform DSP tasks (bottom subplots). We take these measurements for different load intensities (relative to the capacity in UL and DL, respectively) and average SNR indicating the channel quality for both UL and DL, and adapt the MCS to minimize the workload issued by the decoder (differently to our results in Fig. 9.3).

The results yield two observations. First, encoding TBs for PDSCH and (especially) decoding TBs from PUSCH are the two tasks that consume CPU time the most, which is not surprising as it has been observed before [15]. Second, while the CPU time of the rest of tasks (and others not shown in the figure to reduce clutter) remains practically constant,⁷ the time required to process PDSCH and PUSCH highly depends on the context; that is, on the SNR—and so on the mobility patterns of the users, and on the load—and hence on the behaviour of the users.

This gives in upon head-of-line blocking, causing unreliability, as there is no guarantee that wireless samples encoding a minimally functional subframe will be ready for transmission every TTI (see Fig. 9.2).

Conclusions: Optimization techniques exploiting AVX SIMD instructions (e.g., using Intel’s FlexRAN platform) and intra-task parallelization as in [122] certainly help to relieve these issues, but *they do not tackle the fundamental problem at its core: a*

⁷To be precise, the processing time of these tasks *does* vary with the number of users (e.g., scheduling becomes more complex); however, our results (not shown here due to space constraints) indicate that this variability is negligible compared to that of PUSCH and PDSCH processing tasks.

flawed PHY pipeline architecture that cannot accommodate the fluctuations in processing times intrinsic to cloud infrastructure. Note that even if shared pools of hardware accelerators are used *à la cloud* to reduce processing times, queuing in the abstraction layer brokering access to the accelerators across multiple vDUs incur in similar head-of-line blocking issues [12]. Needless to say, aggressive over-dimensioning or *dedicated* hardware accelerators [108, 126] can deal with the aforementioned issues but they compromise cost and flexibility, the very reasons Network Function Virtualization (NFV) is of interest for next-generation RANs in the first place [121].

Conversely, we propose a pipeline design that exploits multi-core clouds efficiently to maximize performance and is resilient to cloud computing fluctuations to attain reliability.

9.3. Cloud-native RAN

To overcome the issues introduced in Section 9.2, we present a novel cloud-native vPHY architecture. Our solution is specifically engineered for 4G LTE and 5G NR workloads that are virtualized over clouds of shared resources with constrained and/or fluctuating computing capacity.

Our approach pursues two objectives: (i) *resiliency* in the presence of computing fluctuations; and (ii) *task parallelization* in multi-core clouds to maximize performance. To this end, we rely upon four main techniques:

1. The baseline design presented earlier exploits *pipeline parallelism* but not *task parallelism*. Pipeline parallelism helps to improve throughput (DSP jobs per unit of time) but not latency (of each DSP job). Parallelizing data processing tasks in **UL/DL data workers**, as illustrated in Fig. 9.5, alleviates the head-of-line blocking they cause to other important tasks, executed by the **forethread**, such as processing control channels or performing IFFT.
2. The time *budget* to process each job n , associated with UL SF n and DL SF $n + M$, goes between time $n + 1 + \delta$ and $n + M - \delta$, which constitutes a fixed allowance of $M - 2\delta - 1$ ms. The wireless samples of DL SF $n + M$ must be delivered to the RU at time $n + M$ (and not before; note the “wait” in Fig. 9.10), that is, *there is no benefit in underusing this budget to complete a job*.

To use it more efficiently, we divide it into two phases as depicted in Fig. 9.6. **Phase I** processes the received UL SF and issues tasks to parallel data workers. **Phase II** uses up the budget to harvest the labor of data workers and build the respective DL SF in time. In this way, data tasks unaccomplished within Phase I due to sudden cloud fluctuations may be ignored to guarantee (at least) a *minimal viable subframe (MVSF)* every TTI.

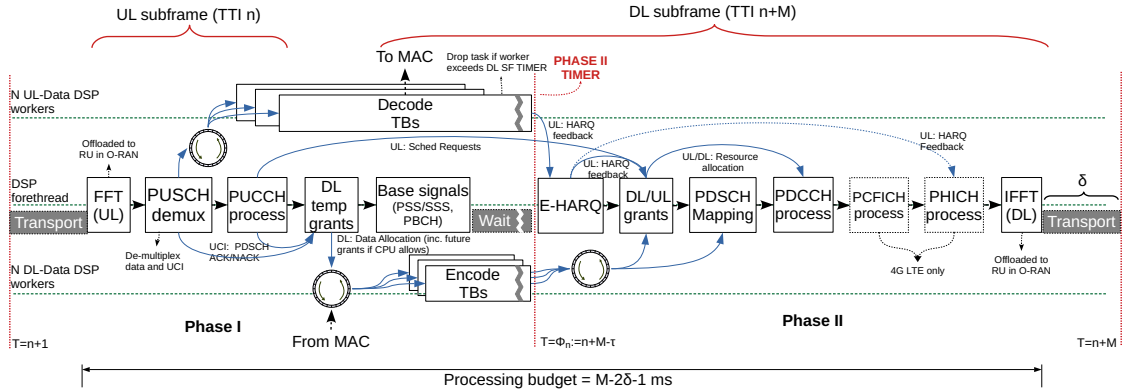


Figure 9.13: A 4G/5G PHY pipeline for cloud environments.

3. To mitigate the impact of Phase I's deadline over UL data tasks during fast computing fluctuations, we implement an **Early Hybrid Automatic Repeat Request (E-HARQ)** procedure that predicts the *decodability* of unfinished UL data workers (Fig. 9.7). This provides UL workers extra budget without compromising the burden of the forethread, i.e., building a DL SF that encodes HARQ feedback therein.
4. We introduce a simple **congestion control** mechanism that adapts the generation of work to UL/DL data workers to slow fluctuations in computing capacity (Fig. 9.8).

The detailed design is depicted in Fig. 9.13. In contrast to the baseline design, where each DSP job is handled by a single worker (see Fig. 9.10), the proposed cloud-native pipeline enables a *minimum number* of three threads processing DSP tasks in parallel:

1. *DSP forethread*: One thread in charge of (i) performing a minimum subset of tasks, computationally cheap and roughly deterministic; and (ii) coordinating the remaining DSP job workers (see middle thread in Fig. 9.13);
2. *UL-Data workers*: One or more threads in charge of PUSCH decoding heavy-lifting (top in Fig. 9.13); and
3. *DL-Data workers*: One or more threads in charge of the bulk of PDSCH processing tasks (bottom in Fig. 9.13).

Decoupling compute-intensive data processing tasks in this way (top and bottom threads in Fig. 9.13) effectively washes off head-of-line blocking that affects critical control tasks performed by the DSP forethread (middle thread of Fig. 9.13) and allows increasing task parallelization.

Once described the main characteristics of the proposed cloud-native RAN, in the following sections we will dig into the details of each part of the pipeline, as well as the mechanisms introduced in the previous section.

9.3.1. DSP forethread

As mentioned above, our cloud-native solution divides the time budget of a DSP job into two phases, as depicted in Fig. 9.13:

- **Phase I:** In this phase, the DSP forethread undertakes the most basic tasks to process the received UL SF n and other UL-independent tasks to process DL SF $n + M$. Specifically, Phase I comprises the following sequence of tasks:

1. FFT (in case of Cloud RAN).
2. **(modified task)** If PUSCH is received, demultiplex UL TBs from UCI and buffer encoded TBs for external processing by an UL-Data worker (see Section 9.3.2).
3. Process PUCCH/UCI if present, and encode base signals (PSS/SSS/PBCH) for the corresponding DL SF $n + M$.
4. **(new task)** Compute *temporary* DL grants depending on the availability of DL data, radio resources, and *computing capacity*. To this end, we employ a simple congestion control mechanism that regulates the flow of grants it issues, which indirectly controls the demand of computing resources (details in Section 9.3.4). These DL grants are temporary, and are stored in a separate buffer for external proceeding by parallel DL-Data workers (see Section 9.3.2), because only those processed before the beginning of Phase II will be actually granted in this job (in DL SF $n + M$).
5. **(new task)** *Snooze* up until time $\Phi_n := n + M - \tau$, where τ is the processing time incurred by Phase II plus transportation (δ ms).⁸ *This step is paramount because it lets us maximize the budget share used by data workers, yet it gives the forethread the leeway to generate an Minimal Viable Subframe (MVSF) that preserves connectivity if data workers take longer than Φ during cloud fluctuations or sudden radio changes.*

- **Phase II:** The forethread awakes at time Φ_n to finish up DL SF $n + M$. To this end, the forethread has $\tau - \delta$ ms of the budget to perform the following sequence of tasks:

1. **(new task)** *E-HARQ* makes a *prediction* upon the decodability of TBs yet unprocessed by UL-Data workers. This prediction is used by the MAC scheduler to schedule feedback and re-transmissions for the users,

⁸ τ and δ are rather predictable (see Section 9.2) and can be estimated beforehand.

and lets us give more computing time budget to *promising* UL TBs that were not decoded on time. More details in Section 9.3.2.

2. **(modified task)** Given the state of finished UL-Data workers, predicted decodability of unfinished UL TBs (see Section 9.3.2), and the actual amount of grants that the DL-Data workers managed to encode on time (see Section 9.3.2), the final UL/DL grants are computed by the MAC scheduler. Similarly to its DL counterpart, UL grants are ruled by a simple congestion control algorithm, detailed in Section 9.3.4. In the worst case, when no worker provided input to the buffer of encoded TBs (e.g., during cloud capacity fluctuations), no DL grants are allocated and the SF becomes an MVSF with the only responsibility of preserving connectivity with the users.
3. If DL grants are provided, the encoded TBs are modulated and mapped into the radio resource grid.
4. All the remaining control channels are processed.
5. IFFT (in case of Cloud RAN).

Dividing the time budget of each DSP job into two phases allows us to improve performance and attain reliability. First, the idle time at the end of each DSP job in our reference design (Fig. 9.10), is in our design used by different workers to process data channels, which helps to increase performance (Fig. 9.13). Second, imposing a hard deadline to data workers, which separates Phase I and II, guarantees a MVSF every job, which provides reliability upon computing fluctuations or sudden changes on radio conditions. In contrast, in the baseline pipeline of Fig. 9.10, head-of-line blocking and computing fluctuations bar any guarantee to accomplish DSP jobs within the time budget, hence becoming a source of unreliability and poor performance.

9.3.2. Data workers

Downlink Performance

DL-Data workers are in charge of encoding DL TBs issued by the forethread's *temporary* scheduler, which are stored in a buffer once encoded successfully. Then, during Phase II, the forethread computes the final DL grants based on those that have already been encoded and, hence, are stored in the buffer. This allows us to decouple data encoding tasks from the rest of the pipeline.

It is expected that the temporary scheduler issues an amount of DL grants such that the amount of workload they generate (by the encoder) can be carried out within Phase I (before time Φ_n), as illustrated in Fig. 9.14's scenario (1). We achieve this by employing a rate controller that is introduced later in Section 9.3.4. However, during cloud computing

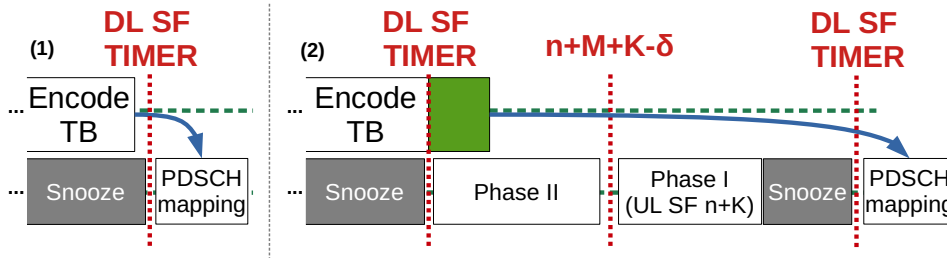


Figure 9.14: DL-Data worker operation.

fluctuations, the DL grants computed by job n 's temporary scheduler may be ready only at job $n + K$, for some integer $K > 0$. Therefore, K represents the amount of time that DL data is buffered by the vPHY, which allows us to absorb computing fluctuations appropriately and hence to preserve resiliency during events of computing volatility.

With the above, not only do we attain resiliency, we also carry out work that is useful at all times, which increases performance. In contrast, the baseline pipeline must start over each (useless) encoding task every time the budget is violated, which wastes resources.

Uplink

As shown in Fig. 9.13, as soon as OFDM symbols arrive, the DSP forethead deposits in a buffer the UL TBs being transported into PUSCH. Then, (coded) TBs are assigned to idle UL-Data workers, i.e., *to decode TBs is the main and only task of UL-Data workers*.

Both types of decoders managed by UL-Data workers in 4G and 5G employ an iterative algorithm to decode data, and a *stopping criteria* to decide on the decodability of the TB [127, 128]. Then, based on such decision, feedback is provided to the DSP forethead so grants for UL re-transmissions (or just new UL data) are allocated.

As illustrated in Fig. 9.15, there are 4 possible scenarios:

1. *TB is decoded successfully before time Φ_n* : At time Φ_n , the UL-Data DSP worker signals an Acknowledgement (ACK) to the UL MAC scheduler, which allocates HARQ feedback accordingly.
2. *TB can be decoded successfully after Φ_n* : In this case, at time Φ_n , the DSP forethead uses E-HARQ to *predict* the positive outcome of the decoding task executed by the UL-Data DSP worker, and signals an ACK to the UL MAC scheduler, like before.
3. *TB is declared undecodable before Φ_n* : At Φ_n , the UL-Data DSP worker signals a Negative Acknowledgement (NACK) to the UL MAC scheduler, and a grant to retransmit the TB is computed accordingly.
4. *TB is declared undecodable after Φ_n* : At Φ_n , E-HARQ *predicts* the negative outcome of the worker and signals a NACK to the UL MAC scheduler, like before.

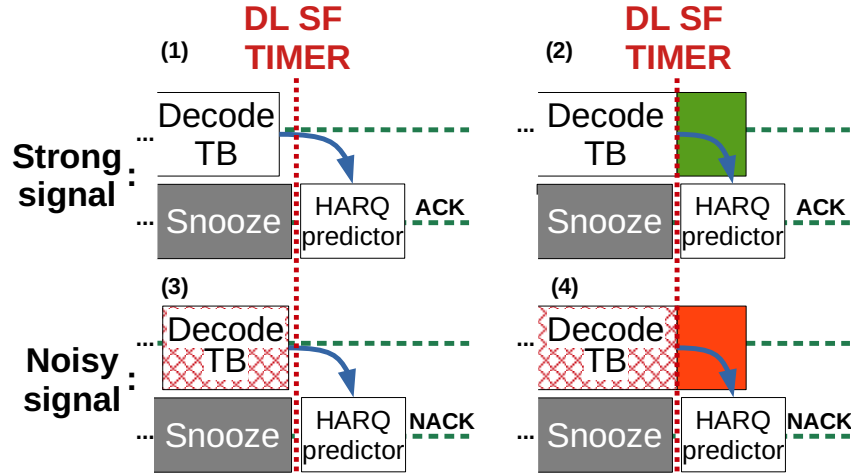


Figure 9.15: UL-Data worker operation

It is expected that cases (1) and (3) occur frequently if cloud resources have been provisioned appropriately and/or our congestion control mechanism (see Section 9.3.4) adapts sufficiently fast. However, due to the volatility inherent to cloud environments, cases (2) and (4) will also occur, i.e., UL-Data workers may not finish their task by time Φ_n . Simply discarding such TBs is a waste of expensive radio resources. Moreover, the availability/lack of computing resources has no impact on the decodability of the TBs, only on the time required to finish the task. To avoid this, the first task executed by the DSP forethread at the beginning of Phase II, at time Φ_n , is E-HARQ, with the goal of *predicting* the decodability of TBs for UL-Data DSP tasks unaccomplished by time Φ , which leads us to the next section.

9.3.3. Early HARQ

E-HARQ is recently receiving attention in the context of low-latency communications. The approach is usually to design appropriate stopping criteria for the iterative algorithms employed by turbodecoders [128] and Low Density Parity Check (LDPC) decoders [129], or to predict the decodability of the data to send HARQ feedback early [130]. As mentioned before, our solution leverages this technique to provide extra time budget to UL workers.

Let $S_{w,t_n} \in \mathcal{S}$ denote the *state* of an UL-Data worker w at time t in DSP job n , where \mathcal{S} is the state space of the decoder. At time Φ_n , the DSP forethread observes the state of each *unfinished* worker, i.e., S_{w,Φ_n} for all $w \in \mathcal{W}_u$ where \mathcal{W}_u is the set of active UL-Data workers, and apply a rule $\Pi(S_{w,\Phi_n}) \in \{\text{UNDECODABLE}, \text{DECODABLE}, \text{UNKNOWN}\}$ to decide upon the decodability, undecodability, or uncertain decodability of the TB.

Once E-HARQ infers the decodability of the TB, it signals the UL MAC scheduler so it delivers the appropriate UL HARQ feedback to the users and/or schedules re-

transmissions, accordingly, as *if the workers had finished their task*. UNKNOWN TBs are treated by the UL MAC scheduler as UNDECODABLE TBs; however, this information is useful for congestion control, as we will show in Section 9.3.4. If $\Pi(S_w, \Phi_n) = \text{DECODABLE}$, the UL-Data worker w is allowed to continue up till a maximum number of decoding iterations. Usually, Cyclic Redundancy Check (CRC) validation is used as a stopping criterion. Otherwise, TB is discarded and the UL-Data worker w becomes idle again and ready to receive new work.

Our approach to design the rule Π and \mathcal{S} draws on different ideas from prior work on turbo and LDPC codes. The key idea behind rests upon the concept of *extrinsic information*, which spawns organically by *belief propagation* algorithms used by both turbo and LDPC codes. We refer the reader to [131] for detailed information about these coding techniques. In a nutshell, belief information is encoded into Log-Likelihood Ratio (LLR), $L_b := \ln \frac{\text{Prob}(b=+1|\text{input})}{\text{Prob}(b=-1|\text{input})}$, where “input” refers to all the inputs of each decoding node i in a decoder, and b represents the information symbol (bit). The key to iterative decoding is the sequence of *a posteriori LLRs* of the information symbols, $\vec{L}_{x_i}^{(k)}$, which is exchanged every iteration k between the decoding nodes so each node takes advantage of the information computed by the others.

To improve the bit estimations every iteration, the different nodes need to exchange belief information *that do not originate from themselves*. The original concept of extrinsic information was conceived precisely to identify the information components that depend on redundant information introduced by the incumbent code. Such extrinsic LLRs $\vec{L}_{e_i}^{(k)}$ are used to transform a posteriori LLRs into *a priori LLRs* $\vec{L}_{a_i}^{(k+1)}$ used as an input in the next decoding iteration. For instance, a turbo decoder, consisting of two convolutional decoding nodes, computes two sets of extrinsic LLR vectors every iteration as follows:

$$\vec{L}_{e_i}^{(k)} = \vec{L}_{x_i}^{(k)} - \vec{L}_x - \vec{L}_{a_n}^{(k)}, \quad \forall i \neq n \in \{1, 2\}$$

where \vec{L}_x is the sequence of input symbols to the decoder.

Extrinsic LLRs can be good estimators of the decodability of the Code Blocks (CBs) that constitute a TBs. We define $S_w, \Phi_n := \{K, \vec{L}_{e_1}, \dots, \vec{L}_{e_D}\}$, where $D, K \in \mathbb{N}$ are, respectively, the number of decoding nodes and the number of iterations completed by w by then. $\vec{L}_{e_i} := [\bar{L}_{e_i}^{(1)}, \dots, \bar{L}_{e_i}^{(K)}]$ is a K -dimensional vector comprised of the mean magnitude of *extrinsic* LLRs at every iteration $k = \{1, \dots, K\}$, i.e., $\bar{L}_{e_i}^{(k)} = \frac{1}{N} \sum_{b=1}^N |L_{e_i,b}^{(k)}|$, where N is the length of the CB being decoded and $L_{e_i,b}$ is the extrinsic LLR of bit b .

Fig. 9.16 shows the mean \vec{L}_{e_i} of both decoding nodes in a turbo decoder for decodable CBs (colored with a blue-green-yellow gradient) and for undecodable CBs (colored in red).⁹ The color of decodable CBs indicate the maximum number of iterations required till CRC validates their successful decoding. We can observe that the mean extrinsic

⁹Dataset available at [hidden to respect the author’s anonymity].

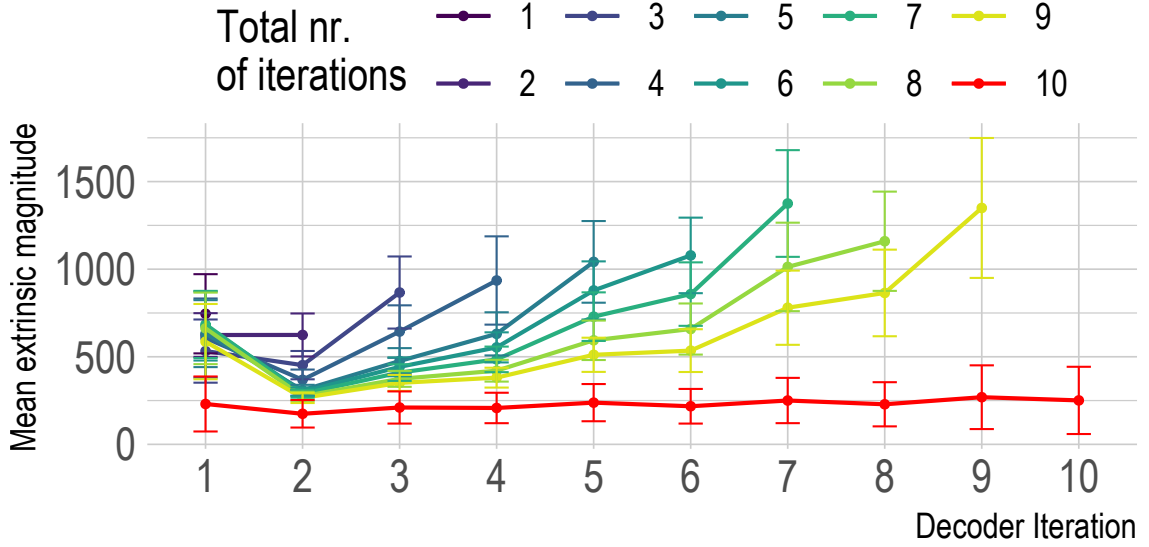


Figure 9.16: Mean extrinsic magnitude for each iteration of a turbodecoder. Dot/line indicate the average value across multiple PUSCH TBs with different MCS, TBS and SNR. Error bars indicate the standard deviation.

magnitude of *undecodable* CBs (in red) is small and rather steady across iterations. In contrast, the mean extrinsic magnitude of decodable CBs grow over iterations. That is, there exist patterns of extrinsic information as it propagates across iterations in the decoder that are distinguishable between decodable CBs and undecodable CBs.

In light of the above, we implement a simple rule $\Pi_{\vec{\gamma}}$, parametrized with $\vec{\gamma} = [\gamma_1, \gamma_2]$ and detailed in Algorithm 1, that poses minimal overhead to the DSP architect. Note that, in case we do not have enough extrinsic information to make a prediction because the decoder has not completed any iteration, we resort to simple predictive models that only rely on SNR estimates such as that in [132].

9.3.4. Congestion control

For a vPHY processor operating in clouds, it is important to adapt the demand for computing resources to the system capacity. This is a fundamentally different paradigm to that of RANs using dedicated hardware. The proposed cloud-native solution, generates temporary DL grants, and hence workload to the DL-Data workers during Phase I, and UL grants, and hence workload to the UL-Data workers during Phase II. To achieve the above goal, the computation of these grants (and so the resulting workload) based on the data buffers, the channel quality—like regular schedulers—and *the available computing capacity*.

Adapting a flow of requests (encoding/decoding tasks, in this case) to a server capacity (cloud platform) falls into the realm of *congestion control*. Hence, we resort to mechanisms

Algorithm 1 E-HARQ rule $\Pi_{\vec{\gamma}}$

```

1: if  $K < 1$  then
2:   if  $\mathcal{P}(\text{MCS}, \text{SNR}, N) > \epsilon$  then ▷ Using [132]
3:      $\Omega \leftarrow \text{UNKNOWN}$ 
4:   else
5:      $\Omega \leftarrow \text{DECODABLE}$ 
6:   end if
7: else
8:   if  $(\bar{L}_{e_D}^{(K)} < \gamma_1) \parallel (\bar{L}_{e_D}^{(K)} - \bar{L}_{e_D}^{(2)} < \gamma_2)$  then
9:      $\Omega \leftarrow \text{UNDECODABLE}$ 
10:  else
11:     $\Omega \leftarrow \text{DECODABLE}$ 
12:  end if
13: end if
14: Return  $\Omega \in \{\text{UNDECODABLE}, \text{DECODABLE}, \text{UNKNOWN}\}$ 

```

amply used in networking protocols such as TCP. Similarly to TCP flows, schedulers integrate an UL and a DL *congestion window* that regulate the flow of respective grants that can be generated: $cwnd_{\text{UL}}$ and $cwnd_{\text{DL}}$, respectively.

We adopt a conventional Additive-Increase/Multiplicative-Decrease (AIMD) algorithm where the congestion window additively increases by α PRBs every DSP job n ($cwnd^{(n+1)} = cwnd^{(n)} + \alpha$) as long as congestion is not detected or the maximum PRB capacity is reached, and multiplicatively decreases by a backoff factor β ($cwnd^{(n+1)} = cwnd^{(n)} \cdot \beta$) if congestion is detected. Of course, congestion is inferred differently for UL and DL:

- For DL grants, congestion is signalled if the aggregate amount of PRBs stored in the DL-Data buffer exceeds λ times the PRB capacity of the vDU per TTI;
- For UL grants, congestion is signalled every time E-HARQ declares the decodability of a TB as UNKNOWN.

10

Experimental evaluation

To validate and evaluate our pipeline design, we have implemented the designed cloud-native solution into srsRAN’s stack [15] with the parameters shown in Table 10.1, and use its implementation of the baseline architecture shown in Fig. 9.10 as our benchmark.

We next:

- Validate cloud-native’s approach and assess its performance with different settings (congestion control, E-HARQ) in both downlink and uplink scenarios (Section 10.2 and Section 10.4);
- Assess the proposed E-HARQ under different network conditions and computing capacity settings (Section 10.3);
- Evaluate cloud-native RAN in scenarios with a variable number of vDUs contending for shared computing resources (Section 10.5).

Table 10.1: Cloud-native Parametrization

Parameter	Value
Pipeline parallelization depth (M)	4
Nr. DL-DSP workers	3 per forethread
Nr. UL-DSP workers	3 per forethread
UL-DSP buffer size	8 grants
DL-DSP buffer size	25 grants
Phase II budget (τ)	1.3 ms
UL Congestion Control (α, β)	3, 0.9
DL Congestion Control (α, β, λ)	3, 0.9, 2
E-HARQ (γ_1, γ_2)	600, 200

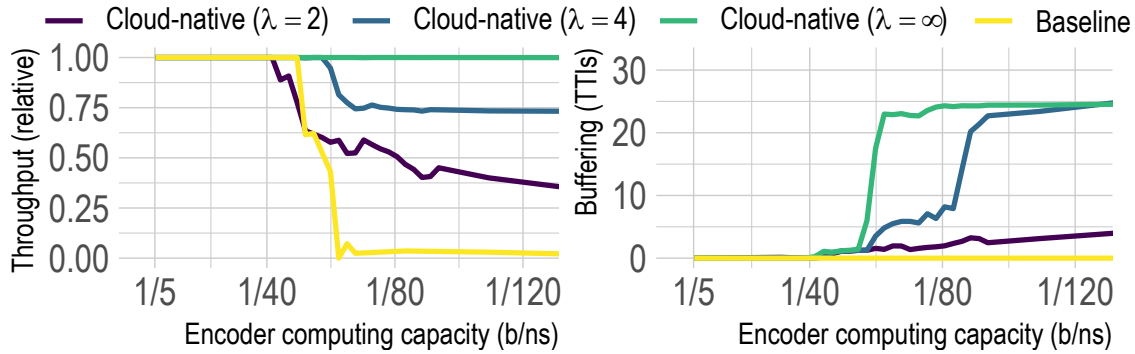


Figure 10.1: Downlink throughput (left) and vPHY buffering (right) with saturating DL load. Comparison between the baseline and cloud-native approach with different λ settings under different computing capacities.

10.1. Testbed description

Similarly to all the evaluations performed in this work, we deployed a real scenario based on off-the-shelf Intel(R) Xeon(R) CPU D-1528 server with six x86 cores @ 1.90GHz, and Linux containers to host vDU instances. To avoid uncontrolled effects, we disable hyperthreading and apply CPU shielding to allocate five cores to host vDU instances; the remaining core is reserved for the OS and other processes (e.g., for monitoring). Both vDU and User Equipment (UE) run over an Ettus USRP board connected similarly to the testbeds used in Sections 7.1.1, 7.2.1 and 7.3.1

10.2. Downlink

We begin by assessing the performance with downlink traffic. To this end, we set up a scenario comprised of one vDU transmitting as much load as possible with high SNR, and measure throughput (rate of bits successfully decoded by the receiver) and buffering (time elapsed between scheduling a data grant for encoding and actually delivering it over the air). Fig. 10.1 compares the performance of our baseline and the cloud-native approach with different settings of λ , for different amount of computing resources allocated to the vDU's encoder, measured as the number of bits that are processed by time unit, between 8 b/ μ s and 200 b/ μ s. As a reference, a *dedicated* Intel Xeon core @ 1.9GHz provides an encoding performance of 200 b/ μ s.

We first note that the baseline approach is *inelastic*: its throughput collapses suddenly when its DSP jobs exceed the available processing time budget (see Fig. 9.10). In marked contrast, the cloud-native solution preserves high throughput irrespective of the available computing capacity, providing *elasticity* upon cloud computing fluctuations. Indeed, λ serves to choose the desired trade-off between data buffering at the vPHY (shown by the right-most subplot) and throughput. When $\lambda = \infty$, effectively

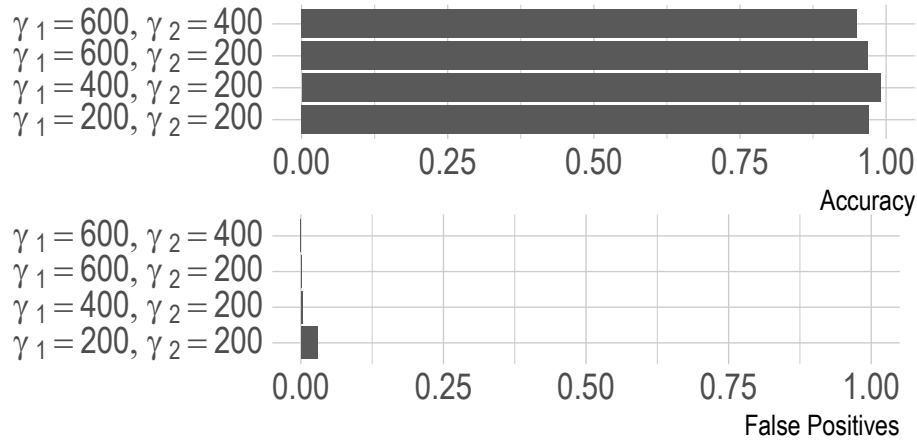


Figure 10.2: E-HARQ accuracy and false positive rate for different γ settings, and for different combinations of MCS, TB size, SNR, and computing capacity.

disabling congestion control, the cloud-native solution is greedy and maintains maximum throughput irrespective of the computing capacity but incurs high buffering. Lower values of λ help reduce buffering at the cost of throughput upon cloud computing fluctuations.

10.3. Uplink (Early HARQ)

Different combinations of SNR, MCS, TB size, and computing capacity have different effects on the E-HARQ's performance. To assess this, we test different Π_γ (see Algorithm 1) and show in Fig. 10.2 its accuracy (ratio of predictions made right, at the top), and its ratio of false positives (bottom subplot). The latter is relevant because a false positive, i.e., acknowledging a TB that is not decodable, incurs substantially higher cost because the TB has to be recovered by higher layers such as RLC or even TCP (if RLC does not use ARQ). This exercise lets us explore the parameter space of our E-HARQ approach and choose the most appropriate setting. Although $\{\gamma_1 = 600, \gamma_2 = 400\}$ reaches a negligible false positive ratio ($\sim 0.04\%$) and high accuracy (95%), we have selected $\{\gamma_1 = 400, \gamma_2 = 200\}$ because it attains 99% accuracy while keeping a low ratio of false positives as well ($\sim 0.1\%$).

Certainly, the actual performance is determined by the (extrinsic) information available when Phase II starts in each DSP job. To get additional insight on this, we present in Fig. 10.3 the minimum computing capacity required by the decoder to attain a given target of accuracy from 50% to 99% for 3 different SNR regimes (low, with SNR below 20 dB, medium, with SNR up to 25 dB, and high), and 6 different MCS indexes (14 to 19) from 3GPP.¹ As a baseline to compare against, we plot with a red line

¹A higher MCS index allows higher data bit count per PRB by either reducing the coding rate or increasing the modulation level. See 3GPP TS 36.213.

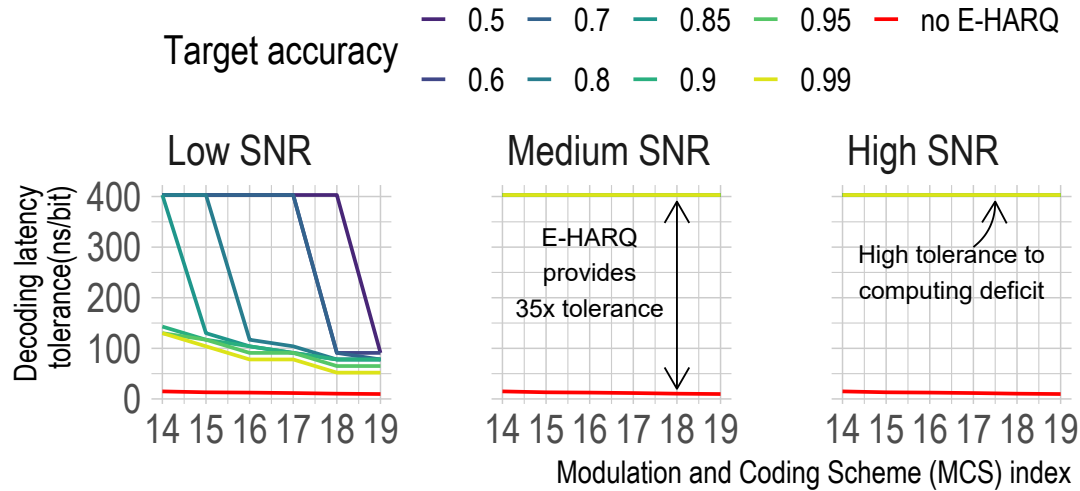


Figure 10.3: Maximum computing latency supported by cloud-native's E-HARQ given a target accuracy.

the computing capacity required to run up to 10 decoder iterations in our time budget, which is the usual threshold to decide on the decodability of a TB when E-HARQ is not employed. As a reference, a *dedicated* Intel Xeon core @ 1.9GHz provides a decoding performance of ~ 80 b/ μ s per iteration.

To attain high accuracy during medium and high SNR regimes, we only require sufficient computing resources so our decoder can process ~ 2.5 bits/ μ s (400 ns per bit). This is true for all MCSs under medium SNR except for MCS 19, which tightens its computing capacity requirement to ~ 10 bit/ μ s. During low SNR regimes, we need to process bits at a rate higher than ~ 20 bit/ μ s if we target 99% accuracy. The rest of MCS indexes under evaluation alternate between high and mild processing requirements depending on the accuracy target and MCS during low SNR conditions. Remarkably, our E-HARQ mechanism alone lets us reduce by 35x the amount of computing capacity required to carry out the task.

10.4. Capacity region

We now characterize empirically the *capacity region* of one vDU under high SNR regimes. To this end, we measure DL and UL throughput for a wide set of UL/DL network load combinations. We also vary the amount of computing resources allocated to the encoder and the decoder by scaling them down to $k \cdot c_0$, where c_0 is the nominal encoding/decoding capacity of a *dedicated* Intel Xeon core @ 1.9GHz, and $k \geq 1$. Our results are shown in Fig. 10.4 for the cloud-native solution, tuned with and without E-HARQ and $\lambda = \{2, \infty\}$, and for our baseline approach. To ease visualization, we only present the region envelop and a few experimental points around it.

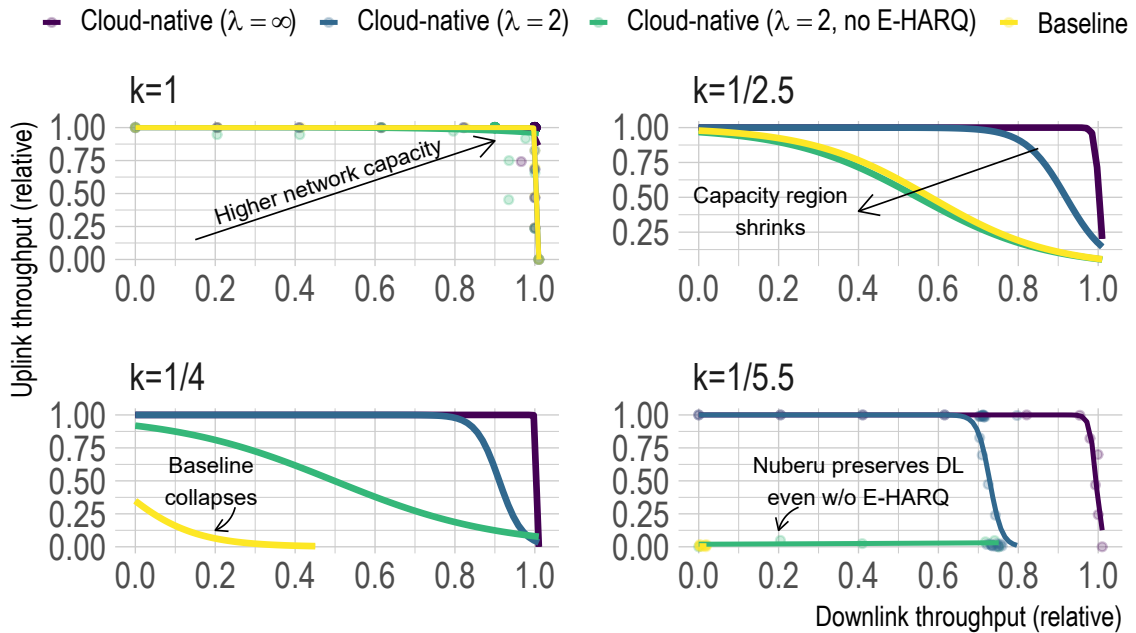


Figure 10.4: Capacity region of the cloud-native solution and our baseline. Different computing capacity settings equal to $k \cdot c_0$, where c_0 is the nominal encoding/decoding capacity of a *dedicated* Intel Xeon core @ 1.9GHz.

Obviously, when we have dedicated and over-dimensioned computing capacity ($k = 1$), the networking capacity region shows a rectangular shape where uplink and downlink reach maximum spectral efficiency. However, if we reduce the system's computing capacity by $k = 1/2.5$, the Cloud-native approach with $\lambda = \infty$ (in purple line) preserves the highest theoretical spectrum efficiency.² Moreover, with $\lambda = 2$ (in dark blue line) provides a much higher maximum aggregate throughput than our baseline approach (yellow line). Most of this capacity gain comes from E-HARQ, as it can be seen from the fact that our solution falls to the baseline's performance when E-HARQ is disabled (green line). As we reduce the amount of computing resources ($k = 1/5$ and $k = 1/5.5$), the baseline's capacity region keeps shrinking while the cloud-native approach sustains high downlink capacity (even if E-HARQ is disabled) as well as uplink (unless E-HARQ is disabled and the computing conditions are the harshest, i.e., $k = 1/5.5$). Remarkably, with $k = 1/5.5$, which represents an 80% computing resource reduction over the best scenario for the baseline, the cloud-native approach reaches $\sim 95\%$ of the maximum theoretical spectrum efficiency.

²As shown by our earlier experiment, this comes at the cost of buffering at the vPHY (results omitted due to space constraints).

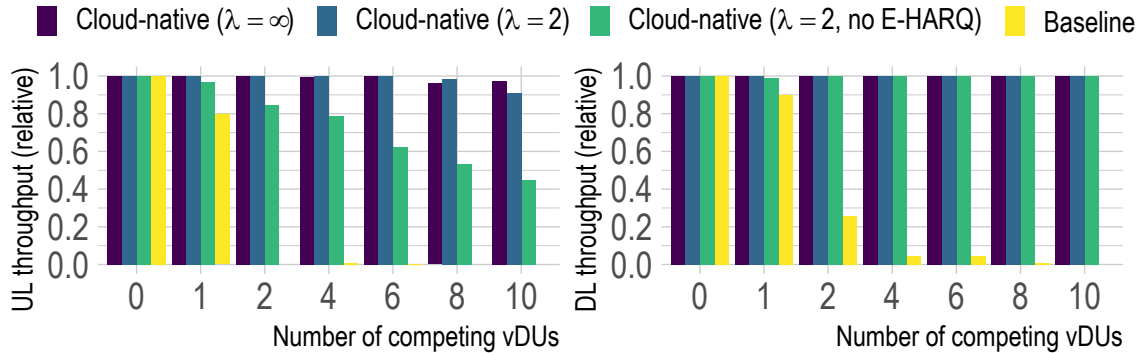


Figure 10.5: Network capacity for a variable number of vDUs contending for computing resources.

10.5. Multiple vDUs

In the previous sections, we have characterized the performance of the proposed cloud-native approach (and the baseline’s) as a function of the processing speed provided by the cloud infrastructure. In the following, we keep the maximum processing speed of our experimental platform (which attains 200 encoded bits/ μ s and ~ 80 decoded bits/ μ s per iteration), and deploy multiple vDU instances sharing the common computing resource (five Intel Xeon cores @ 1.90GHz).

Specifically, we let one vDU under test compete for resources with a variable number of baseline vDUs. Baseline vDUs *do not adapt their workload to the available capacity*; instead, we let them generate DSP jobs with a duration that follows a normal distribution with average 1 ms and standard deviation 0.25 ms. Fig. 10.5 depicts the UL and DL throughput (left and right subplots, respectively) when the vDU under test uses the cloud-native approach with the same settings used before (blueish and green bars), and when it uses the baseline (yellow bar), for different scenarios where the number of competing (baseline) vDUs spans between 0 and 10.

The performance of the proposed cloud-native approach remains high in all scenarios. Only when E-HARQ is not employed, UL throughput drops approximately 5% per competing vDU. When using E-HARQ, UL throughput only drops 10% when $\lambda = 2$ and $\sim 2\%$ when $\lambda = \infty$. Conversely, our cloud-native approach attains the maximum spectral efficiency in the DL channel irrespective of the number of contending vDUs. In marked contrast, the performance of our baseline approach drops significantly with every competing vDU: UL performance collapses when it competes with more than 1 vDU, and DL throughput only reaches 27% with 2 competing vDUs and $< 5\%$ with a higher amount of competitors. This result provides strong evidence of the elasticity and high efficiency of the proposed cloud-native solution in shared cloud environments.

11

Summary Part III

Research has shown that resource contention in cloud infrastructure degrades the performance provided by dedicated platforms. Virtualized RANs are especially sensitive to this problem due to the *real-time timing constraints* of many of its operations. In this section, we have proposed a novel Distributed Unit (DU) design specifically engineered for 4G and 5G vRANs that are virtualized over clouds of shared resources with constrained and/or fluctuating computing capacity.

The proposed cloud-native solution builds around four main techniques: *(i)* high parallelization and decoupling of data processing tasks; *(ii)* tight deadline control of data processing tasks; *(iii)* decodability prediction, which provides additional computing budget to uplink data tasks; *(iv)* a control feedback loop that adapts the generation of workload (data grants to encode/decode) to the availability of computing capacity. In contrast to baseline designs where each DSP job is handled by a single worker, the proposed cloud-native approach enables high intra-job parallelization. Decoupling compute-intensive data processing tasks in this way effectively washes off head-of-line blocking that affects critical control tasks and allows us to increase task parallelization.

A comprehensive evaluation of our solution has been conducted in a proof-of-concept. Our results show that it dramatically improves the robustness of vRANs in cloud environments over state-of-art solutions, e.g., providing as much as $\sim 95\%$ spectrum efficiency when computing resources are reduced by $\sim 80\%$.

12

Conclusions and future work

12.1. Conclusions

Network slicing and virtualization of network functions are key enablers for next generation mobile networks. Along this work, we have investigated and proposed different solutions to properly overcome the challenges arising from them.

The first block describes the design process we have followed to properly characterize the evolution of mobile networks. We have proposed a path towards a cloud-aware mobile protocol stack and a novel architecture envisioned to run functions as atomic pieces independent of the cloud infrastructure.

The second block contains the complete process of design and implementation we have followed to bring network slicing to softwarized mobile networks. More precisely, in Section 6.1 we have presented POSENS, an open source solution for practical end-to-end network slicing based on slice-aware shared RAN. This tool includes all the software components needed to deploy a multi-slice network setup. POSENS enables the slicing of the RAN as well as the core, which are fundamental building blocks for achieving end-to-end network slicing. We have validated POSENS in an experimental deployment, showing how it can obtain per-slice customization without paying a price in terms of performance. Then, we have described in Section 6.2 our hands-on experience gained from the implementation of some distinctive functionality of 5G Networking and *(i)* multi-slice orchestration. All our designs and software implementations (based on open-source software) are publicly available through scientific publications and code repositories. The obtained results and tools that we release will prove very useful to researchers and practitioners working on this area of research. Furthermore, in Section 6.3 we have proposed a new framework for flexible re-orchestration of virtual network functions, which allows on-the-fly orchestration without disrupting ongoing services. We have developed an implementation of a 5G protocol stack that realizes it and have applied it to Virtual Network Functions (VNFs) of different nature. We have evaluated the resulting performance in a realistic network slicing setup, showing the feasibility and

advantages of flexible re-orchestration. The flexible re-orchestration framework envisioned and implemented for this work fits very well the current trends in network softwarization followed by the industry.

Finally, the last block describes the process to properly design a cloud-native RAN. We have designed a novel PHY pipeline architecture specifically thought for cloud environments (Section 9.3) which perfectly fits with the architectural requirements of future network deployments. With this approach, we dramatically improved the robustness and efficiency when suffering from resources fluctuations on shared cloud environments.

The research conducted along this thesis effectively closes the gap between specification and experimentation, proposing at the same time novel ideas to adequately overcome the identified challenges. Moreover, we have covered all the building blocks of next generation mobile networks (RAN, CN, transport and MANO) from design to implementation.

12.2. Future Work

Continuing with the softwarization of networks, the logical step forward is to continue working on the re-design of network functions towards a complete serverless approach. Regarding the network functions placed in the core network, more network functions can be designed using the Context/Execution (c/e) split to evaluate its performance using the proposed framework for flexible orchestration.

At the RAN level, there are still open challenges to address towards a fully serverless design. For example, a complete design to correctly split high/low PHY while being able to accomplish their communication constraints or the usage of task accelerators to perform specific tasks implemented as atomic operations could be interesting challenges to address as future work.

Moreover, initiatives such as Open RAN¹ are promoting architectures to build virtualized RAN on open hardware, enabling AI-based radio control. Within this field, there are multiple interesting challenges related with the design of the proposed components and thinking about how to include and evaluate a RAN implementation similar to the one we propose in this work.

¹<https://www.o-ran.org/>

References

- [1] G. Garcia-Aviles, C. Donato, M. Gramaglia, P. Serrano, and A. Banchs, “Acho: A framework for flexible re-orchestration of virtual network functions,” *Computer Networks*, vol. 180, p. 107382, 2020.
- [2] M. Gramaglia, P. Serrano, A. Banchs, G. Garcia-Aviles, A. Garcia-Saavedra, and R. Perez, “The case for serverless mobile networking,” in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 779–784.
- [3] G. Garcia-Aviles, M. Gramaglia, P. Serrano, F. Gringoli, S. Fuente-Pascual, and I. L. Pavon, “Experimenting with open source tools to deploy a multi-service and multi-slice mobile network,” *Computer Communications*, vol. 150, pp. 1–12, 2020.
- [4] 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.
- [5] G. Garcia-Aviles, M. Gramaglia, P. Serrano, M. Portoles, A. Banchs, and F. Maino, “Semper: a stateless traffic engineering solution for wan based on mp-tcp,” in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.
- [6] S. Henri, G. García, P. Serrano, A. Banchs, and P. Thiran, “Protecting against website fingerprinting with multihoming,” *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 2, pp. 89–110, 2020.
- [7] M. Gramaglia, I. Labrador Pavón, F. Gringoli, G. Garcia-Aviles, and P. Serrano, “Design and validation of a multi-service 5g network with qoe-aware orchestration,” in *Proceedings of the 12th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, 2018, pp. 11–18.
- [8] P. Serrano, M. Gramaglia, D. Bega, D. Gutierrez-Estevez, G. Garcia-Aviles, and A. Banchs, “The path toward a cloud-aware mobile network protocol stack,” *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 5, p. e3312, 2018.

- [9] W. Nakimuli, G. Landi, R. Perez, M. Pergolesi, M. Molla, C. Ntogkas, G. Garcia-Aviles, J. Garcia-Reinoso, M. Femminella, P. Serrano *et al.*, “Automatic deployment, execution and analysis of 5g experiments using the 5g eve platform,” in *2020 IEEE 3rd 5G World Forum (5GWF)*. IEEE, 2020, pp. 372–377.
- [10] 3GPP, “System architecture for the 5G System (5GS),” 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 23.501, 03 2019, version 15.5.0. [Online]. Available: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144>
- [11] “OSM project,” <https://osm.etsi.org/>.
- [12] O-RAN Alliance, “Cloud Architecture and Deployment Scenarios for O-RAN Virtualized RAN v02.01 (O-RAN.WG6.CAD-v02.01),” Technical Report, Jul. 2020.
- [13] P. Serrano, P. Salvador, V. Mancuso, and Y. Grunenberger, “Experimenting With Commodity 802.11 Hardware: Overview and Future Directions,” *IEEE Communications Surveys Tutorials*, vol. 17, no. 2, pp. 671–699, 2015.
- [14] 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, 2014.
- [15] I. Gomez-Miguel, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, “srslte: An open-source platform for lte evolution and experimentation,” in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, 2016, pp. 25–32.
- [16] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega *et al.*, “Network slicing to enable scalability and flexibility in 5g mobile networks,” *IEEE Communications magazine*, vol. 55, no. 5, pp. 72–79, 2017.
- [17] D. Bega, M. Gramaglia, C. J. Bernardos Cano, A. Banchs, and X. Costa-Perez, “Toward the network of the future: From enabling technologies to 5g concepts,” *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 8, p. e3205, 2017.
- [18] 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, 2018.
- [19] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, “From network sharing to multi-tenancy: The 5g network slice broker,” *IEEE Communications Magazine*, vol. 54, no. 7, pp. 32–39, 2016.

- [20] X. Foukas, M. K. Marina, and K. Kontovasilis, "Orion: Ran slicing for a flexible and cost-effective multi-service mobile network architecture," in *Proceedings of the 23rd annual international conference on mobile computing and networking*, 2017, pp. 127–140.
- [21] 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*, vol. 38, no. 2, pp. 69–74, 2008.
- [22] 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, 2014.
- [23] E. P. Report, "1st etsi nfv plugtests," Mar. 2017.
- [24] J. Mendes, X. Jiao, A. Garcia-Saavedra, F. Huici, and I. Moerman, "Cellular access multi-tenancy through small-cell virtualization and common rf front-end sharing," *Computer Communications*, vol. 133, pp. 59–66, 2019.
- [25] C.-Y. Chang, N. Nikaein, and T. Spyropoulos, "Radio access network resource slicing for flexible service execution," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2018, pp. 668–673.
- [26] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "Flexran: A flexible and programmable platform for software-defined radio access networks," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 427–441.
- [27] "Mosaic5G ecosystem," <http://mosaic-5g.io/>.
- [28] X. Foukas, F. Sardis, F. Foster, M. K. Marina, M. A. Lema, and M. Dohler, "Experience building a prototype 5g testbed," in *Proceedings of the Workshop on Experimentation and Measurements in 5G*, 2018, pp. 13–18.
- [29] C.-Y. Huang, C.-Y. Ho, N. Nikaein, and R.-G. Cheng, "Design and prototype of a virtualized 5g infrastructure supporting network slicing," in *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*. IEEE, 2018, pp. 1–5.
- [30] "The POWDER testbed," <https://powderwireless.net/>.
- [31] "The COSMOS lab," <https://www.cosmos-lab.org/>.

- [32] P. Rost, C. J. Bernardos, A. De Domenico, M. Di Girolamo, M. Lalam, A. Maeder, D. Sabella, and D. Wübben, "Cloud technologies for flexible 5g radio access networks," *IEEE Communications Magazine*, vol. 52, no. 5, pp. 68–76, 2014.
- [33] 3GPP, "Feasibility study on new services and markets technology enablers," 2015.
- [34] N. Alliance, "5g white paper," *Next generation mobile networks, white paper*, vol. 1, 2015.
- [35] A. De La Oliva, X. C. Pérez, A. Azcorra, A. Di Giglio, F. Cavaliere, D. Tiegelbekkers, J. Lessmann, T. Haustein, A. Mourad, and P. Iovanna, "Xhaul: toward an integrated fronthaul/backhaul architecture in 5g networks," *IEEE Wireless Communications*, vol. 22, no. 5, pp. 32–40, 2015.
- [36] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.
- [37] E. Dahlman, S. Parkvall, J. Skold, and P. Beming, *3G evolution: HSPA and LTE for mobile broadband*. Academic press, 2010.
- [38] A. L. Amarisoft, "100-software lte base station on pc."
- [39] G. T. 38.801, "Study on new radio access technology: Radio access architecture and interfaces," 2016.
- [40] S. C. Forum, "Small cell virtualization: Functional splits and use cases. *White Paper*, release 6.0, 2016," 2016.
- [41] D. Sabella, P. Rost, A. Banchs, V. Savin, M. Consonni, M. Di Girolamo, M. Lalam, A. Maeder, and I. Berberana, "Benefits and challenges of cloud technologies for 5g architecture," in *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*. IEEE, 2015, pp. 1–5.
- [42] C. Mobile, "C-ran: the road towards green ran," *White paper, ver*, vol. 2, no. 5, pp. 15–16, 2011.
- [43] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann, "Cloud ran for mobile networks-a technology overview," *IEEE Communications surveys & tutorials*, vol. 17, no. 1, pp. 405–426, 2014.
- [44] M. C. Valenti, S. Talarico, and P. Rost, "The role of computational outage in dense cloud-based centralized radio access networks," in *2014 IEEE Global Communications Conference*. IEEE, 2014, pp. 1466–1472.

- [45] E. F. Coutinho, F. R. de Carvalho Sousa, P. A. L. Rego, D. G. Gomes, and J. N. de Souza, “Elasticity in cloud computing: a survey,” *annals of telecommunications-Annales des télécommunications*, vol. 70, no. 7, pp. 289–309, 2015.
- [46] N. R. Herbst, S. Kounev, and R. Reussner, “Elasticity in cloud computing: What it is, and what it is not,” in *10th International Conference on Autonomic Computing (ICAC 13)*, 2013, pp. 23–27.
- [47] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, “How should i slice my network? a multi-service empirical evaluation of resource sharing efficiency,” in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 191–206.
- [48] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices architecture enables devops: Migration to a cloud-native architecture,” *Ieee Software*, vol. 33, no. 3, pp. 42–52, 2016.
- [49] P. Aditya, I. E. Akkus, A. Beck, R. Chen, V. Hilt, I. Rimac, K. Satzke, and M. Stein, “Will serverless computing revolutionize nfv?” *Proceedings of the IEEE*, vol. 107, no. 4, pp. 667–678, 2019.
- [50] M. Condoluci and T. Mahmoodi, “Softwarization and virtualization in 5g mobile networks: Benefits, trends and challenges,” *Computer Networks*, vol. 146, pp. 65–84, 2018.
- [51] 3GPP, “NG-RAN; Architecture description,” TS 38.401, v15.7.0, Mar. 2020.
- [52] —, “System architecture for the 5G System,” TS 28.801, Mar. 2019.
- [53] 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, 2017.
- [54] V. Nagendra, A. Bhattacharya, A. Gandhi, and S. R. Das, “Mmlite: A scalable and resource efficient control plane for next generation cellular packet core,” in *Proceedings of the 2019 ACM Symposium on SDN Research*, 2019, pp. 69–83.
- [55] B. Pfaff et al., “The design and implementation of open vswitch,” in *12th USENIX NSDI 2015*.
- [56] 3GPP, “Management and orchestration; provisioning,” TS 28.531, v15.2.0, Mar. 2019.

- [57] D. Bega, A. Banchs, M. Gramaglia, X. Costa-Pérez, and P. Rost, “Cares: Computation-aware scheduling in virtualized radio access networks,” *IEEE Transactions on Wireless Communications*, vol. 17, no. 12, pp. 7993–8006, 2018.
- [58] J. A. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, A. Banchs, and J. J. Alcaraz, “vrain: A deep learning approach tailoring computing and radio resources in virtualized rans,” in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.
- [59] Cilium, “BPF and XDP Reference Guide,” 2020. [Online]. Available: <https://cilium.readthedocs.io/en/latest/bpf/>
- [60] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, “Performance implications of packet filtering with linux ebpf,” in *2018 30th International Teletraffic Congress (ITC 30)*, vol. 1. IEEE, 2018, pp. 209–217.
- [61] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, “Deepcog: Cognitive network management in sliced 5g networks with deep learning,” in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 280–288.
- [62] Y. Wang, R. Forbes, C. Cavigioli, H. Wang, A. Gamelas, A. Wade, J. Strassner, S. Cai, and S. Liu, “Network management and orchestration using artificial intelligence: Overview of etsi eni,” *IEEE Communications Standards Magazine*, vol. 2, no. 4, pp. 58–65, 2018.
- [63] B. Geró, D. Jocha, R. Szabó, J. Czentye, D. Haja, B. Németh, B. Sonkoly, M. Szalay, L. Toka, C. J. B. Cano *et al.*, “The orchestration in 5g exchange, a multi-provider nfv framework for 5g services,” in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2017, pp. 1–2.
- [64] L. Ma, X. Wen, L. Wang, Z. Lu, and R. Knopp, “An sdn/nfv based framework for management and deployment of service based 5g core network,” *China Communications*, vol. 15, no. 10, pp. 86–98, 2018.
- [65] D. M. Gutierrez-Estevez, M. Gramaglia, A. De Domenico, N. Di Pietro, S. Khatibi, K. Shah, D. Tsolkas, P. Arnold, and P. Serrano, “The path towards resource elasticity for 5g network architecture,” in *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, 2018, pp. 214–219.
- [66] J. Ortín, C. Donato, P. Serrano, and A. Banchs, “Resource-on-demand schemes in 802.11 wlans with non-zero start-up times,” *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3221–3233, 2016.

- [67] “3GPP TS23.501, System Architecture for the 5G System,,” Rel. 15, 2018.
- [68] “3GPP TR28.801, telecommunication management;study on management and orchestration of network slicing for next generation network,,” Rel. 15, 2018.
- [69] O. R. Alliance, “O-ran: towards an open and smart ran,,” *White Paper*, 2018.
- [70] X. Li and C. Qian, “A survey of network function placement,,” in *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2016, pp. 948–953.
- [71] A. Laghrissi and T. Taleb, “A survey on the placement of virtual resources and virtual network functions,,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2018.
- [72] H. Talebian, A. Gani, M. Sookhak, A. A. Abdelatif, A. Yousafzai, A. V. Vasilakos, and F. R. Yu, “Optimizing virtual machine placement in iaas data centers: taxonomy, review and open issues,,” *Cluster Computing*, vol. 23, no. 2, pp. 837–878, 2020.
- [73] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. N. Chang, M. R. Lyu, and R. Buyya, “Cloud service reliability enhancement via virtual machine placement optimization,,” *IEEE Transactions on Services Computing*, vol. 10, no. 6, pp. 902–913, 2016.
- [74] “OSM Release FIVE Technical Overview,,” <https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseFIVE-FINAL.pdf>, Jan. 2019, online; accessed Apr. 2020.
- [75] “ONAP Architecture Overview whitepaper,,” https://www.onap.org/wp-content/uploads/sites/20/2019/07/ONAP_CaseSolution_Architecture_062519.pdf, July 2019, online; accessed Apr. 2020.
- [76] “OSM Information Model,,” https://osm.etsi.org/wikipub/index.php/OSM_Information_Model, 2019, online; accessed Dec. 2019.
- [77] M. E. Elsaid and C. Meinel, “Live migration impact on virtual datacenter performance: Vmware vmotion based study,,” in *2014 International Conference on Future Internet of Things and Cloud*. IEEE, 2014, pp. 216–221.
- [78] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, “A survey on virtual machine migration: Challenges, techniques, and open issues,,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1206–1243, 2018.

- [79] “Openstack Docs live-migrate instances,” <https://docs.openstack.org/nova/pike/admin/live-migration-usage.html>, 2019, accessed: Dec 2019.
- [80] “VMware vSphere vMotion architecture, performance and best practices in vmware vsphere 5: Performance study,” Technical White Paper, 2011, 2019, online; accessed Dec. 2019.
- [81] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, “Voyager: Complete container state migration,” in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2137–2142.
- [82] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, “Unikernels: Library operating systems for the cloud,” *SIGPLAN Not.*, vol. 48, no. 4, pp. 461–472, Mar. 2013. [Online]. Available: <http://doi.acm.org/10.1145/2499368.2451167>
- [83] S. Troia, A. Rodriguez, R. Alvizu, and G. Maier, “Senatus: An experimental sdn/nfv orchestrator,” in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2018, pp. 1–5.
- [84] T. . 3GPP, “Enhancements of dedicated core networks selection mechanism (release 14),” 2016.
- [85] 3GPP, “Nr; overall description; stage-2,” TS 38.300 V15.0.0, Jan. 2018.
- [86] E. G. N.-M. 001, “Network functions virtualisation (nfv); management and orchestration,” v1.1.1, Dec. 2014.
- [87] S.-Q. Lee and J.-u. Kim, “Local breakout of mobile access network traffic by mobile edge computing,” in *2016 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2016, pp. 741–743.
- [88] A. Garcia-Saavedra, X. Costa-Perez, D. J. Leith, and G. Iosifidis, “Fluidran: Optimized vran/mec orchestration,” in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2366–2374.
- [89] Y. Zaki, T. Weerawardane, C. Gorg, and A. Timm-Giel, “Multi-qos-aware fair scheduling for lte,” in *2011 IEEE 73rd Vehicular Technology Conference (VTC Spring)*. IEEE, 2011, pp. 1–5.
- [90] ETSI, “Network Functions Virtualisation (NFV);Use Cases,” Tech. Rep. GS NFV 001, 10 2013, v1.1.1.
- [91] 3GPP, “Management and orchestration; Concepts, use cases and requirements,” 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 28.530, 12

- 2018, version 15.1.0. [Online]. Available: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3273>
- [92] “Snowmix - the swiss army knife of open source live video mixing.” <https://snowmix.sourceforge.io/>.
- [93] “Apache Tomcat project,” <https://tomcat.apache.org/>.
- [94] “FFMPG project,” <https://ffmpeg.org/>.
- [95] X. Feng, J. Tang, X. Luo, and Y. Jin, “A performance study of live vm migration technologies: Vmotion vs xenmotion,” in *2011 Asia Communications and Photonics Conference and Exhibition (ACP)*, Nov 2011, pp. 1–6.
- [96] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, “Opennf: Enabling innovation in network function control,” in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM ’14. New York, NY, USA: ACM, 2014, pp. 163–174. [Online]. Available: <http://doi.acm.org/10.1145/2619239.2626313>
- [97] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, “Split/merge: System support for elastic execution in virtual middleboxes,” in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX, 2013, pp. 227–240. [Online]. Available: <https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/rajagopalan>
- [98] “3GPP TS23.502, Procedures for the 5G System (5GS); stage 2 (release 16),” Rel. 16, 2020.
- [99] “ETSI, network functions virtualisation (nfv) release 3; evolution and ecosystem; report on network slicing support with etsi nfv architecture framework,” 2017.
- [100] “3GPP TS29.510, 5G System; Network function repository services; Stage 3 (Release 15),” Rel. 15, 2020.
- [101] “ONOS Project,” <https://onosproject.org>, 2019, online; accessed Dec. 2019.
- [102] “Cloud-native network functions,” Cisco White Paper, 2018. [Online]. Available: <https://www.cisco.com/c/en/us/solutions/service-provider/industry/cable/cloud-native-network-functions.html>
- [103] “3GPP TS28.541, Management and orchestration; 5G Network Resource Model (NRM); Stage 2 and stage 3,” Rel. 15, 2018.
- [104] “5G-PPP vision and mission,” <https://5g-ppp.eu/about-us/>.

- [105] K.-T. Seo, H.-S. Hwang, I.-Y. Moon, O.-Y. Kwon, and B.-J. Kim, "Performance comparison analysis of linux container and virtual machine for building cloud." Science & Engineering Research Support soCiety, dec 2014. [Online]. Available: <https://doi.org/10.14257/astl.2014.66.25>
- [106] 5G-CORAL, "Refined design of 5G-CORAL orchestration and control system and future directions," D3.2, May 2019.
- [107] Intel, "vRAN: The Next Step in Network Transformation," *White Paper*, 2017.
- [108] Samsung, "Virtualized Radio Access Network: Architecture, Key technologies and Benefits." *Technical Report*, 2019.
- [109] Cisco, Rakuten, Altiostar, "Reimagining the End-to-End Mobile Network in the 5G Era," *White Paper*, 2019.
- [110] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "Nfp: Enabling network function parallelism in nfv," in *Proceedings of ACM SIGCOMM 17*. ACM, 2017, pp. 43–56.
- [111] P. Kumar, N. Dukkupati, N. Lewis, Y. Cui, Y. Wang, C. Li, V. Valancius, J. Adriaens, S. Gribble, N. Foster, and A. Vahdat, "Picnic: Predictable virtualized nic," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM 19. Association for Computing Machinery, 2019, pp. 351–366.
- [112] A. Manousis, R. A. Sharma, V. Sekar, and J. Sherry, "Contention-aware performance prediction for virtualized network functions," in *Proceedings of ACM SIGCOMM 20*. ACM, 2020, pp. 270–282.
- [113] J. Gong, Y. Li, B. Anwer, A. Shaikh, and M. Yu, "Microscope: Queue-based performance diagnosis for network functions," in *Proceedings of ACM SIGCOMM 20*. ACM, 2020, pp. 390–403.
- [114] A. Tootoonchian, A. Panda, C. Lan, M. Walls, K. Argyraki, S. Ratnasamy, and S. Shenker, "Resq: Enabling slos in network function virtualization," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 283–297. [Online]. Available: <https://www.usenix.org/conference/nsdi18/presentation/tootoonchian>
- [115] 3GPP, "5G;NG-RAN; Architecture description ," 3GPP TS 38.401 version 16.2.0 Release 16, Dec. 2020.
- [116] W. T. Han and R. Knopp, "OpenAirInterface: A pipeline structure for 5G," in *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*. IEEE, 2018, pp. 1–4.

- [117] S. Kumar, E. Hamed, D. Katabi, and L. Erran Li, "Lte radio analytics made easy and accessible," in *Proceedings of ACM SIGCOMM 14*, vol. 44, no. 4. ACM, 2014, pp. 211–222.
- [118] E. Hamed, H. Rahul, and B. Partov, "Chorus: Truly distributed distributed-mimo," in *Proceedings of ACM SIGCOMM 18*. ACM, 2018, pp. 461–475.
- [119] Z. A. Qazi, M. Walls, A. Panda, V. Sekar, S. Ratnasamy, and S. Shenker, "A high performance packet core for next generation cellular networks," in *Proceedings of ACM SIGCOMM 17*. ACM, 2017, pp. 348–361.
- [120] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-P  rez, "Resource sharing efficiency in network slicing," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 909–923, 2019.
- [121] Rethink Technology Research, "Special Report: Open Networks." *Technical Report*, 2020.
- [122] J. Ding, R. Doost-Mohammady, A. Kalia, and L. Zhong, "Agora: Real-time massive MIMO baseband processing in software," in *Proceedings of ACM CoNEXT 20*. ACM, 2020.
- [123] E. Dahlman, S. Parkvall, and J. Skold, *5G NR: The next generation wireless access technology*. Academic Press, 2018.
- [124] X. Lin, J. Li, R. Baldemair, J. T. Cheng, S. Parkvall, D. C. Larsson, H. Koorapaty, M. Frenne, S. Falahati, A. Grovlen, and K. Werner, "5G New Radio: Unveiling the Essentials of the Next Generation Wireless Access Technology," *IEEE Communications Standards Magazine*, vol. 3, no. 3, pp. 30–37, 2019.
- [125] F. Kaltenberger, A. P. Silva, A. Gosain, L. Wang, and T.-T. Nguyen, "OpenAirInterface: Democratizing innovation in the 5G Era," *Computer Networks*, p. 107284, 2020.
- [126] F. Civerchia, M. Pelcat, L. Maggiani, K. Kondepu, P. Castoldi, and L. Valcarenghi, "Is opencl driven reconfigurable hardware suitable for virtualising 5g infrastructure?" *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 849–863, 2020.
- [127] L. Hanzo, J. P. Woodard, and P. Robertson, "Turbo decoding and detection for wireless applications," *Proceedings of the IEEE*, vol. 95, no. 6, pp. 1178–1200, 2007.
- [128] P. Salija and B. Yamuna, "An efficient early iteration termination for turbo decoder," *Journal of Telecommunications and Information Technology*, 2016.

-
- [129] J. Li, G. He, H. Hou, Z. Zhang, and J. Ma, “Memory efficient layered decoder design with early termination for ldpc codes,” in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*. IEEE, 2011, pp. 2697–2700.
- [130] 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, 2019.
- [131] Y. Sun and J. R. Cavallaro, “A flexible ldpc/turbo decoder architecture,” *Journal of Signal Processing Systems*, vol. 64, no. 1, pp. 1–16, 2011.
- [132] P. Rost and A. Prasad, “Opportunistic hybrid arq-enabler of centralized-ran over non ideal backhaul,” *IEEE Wireless Communications Letters*, vol. 3, no. 5, pp. 481–484, 2014.