



6G BRAINS Deliverable D5.1 E2E Network Slicing Control Enablers

Editor:	Ta Dang Khoa Le, ECOM
Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Consortium Confidential (CO) <i>Until end of August 2022</i>
Contractual delivery date:	31 December 2021
Actual delivery date:	
Suggested readers:	5G and beyond (6G) stakeholders
Version:	1.0
Total number of pages:	84 pages
Keywords:	Network slicing, RAN slicing, Backbone slicing, AI, ML

Abstract

This document reports the up-to-date progress towards achieving smart E2E network slicing. It focuses on the preliminary design and prototyping of enabling technologies for E2E network slicing across network segments. Moreover, it provides initial investigations into the algorithm design of AI-based radio link control and RAN slice scheduler, as well as integration aspects of Management-Orchestration and Cognitive Plane development kit. Together, these enablers form basic building blocks for deeper integrations, technologically and algorithmically, between partners in future implementation of the targeting E2E slicing system.

[End of abstract]

Disclaimer

This document contains material, which is the copyright of certain 6G BRAINS consortium parties, and may not be reproduced or copied without permission.

In case of Public (PU):

All 6G BRAINS consortium parties have agreed to full publication of this document.

In case of Restricted to Programme (PP):

All 6G BRAINS consortium parties have agreed to make this document available on request to other framework programme participants.

In case of Restricted to Group (RE):

All 6G BRAINS consortium parties have agreed to full publication of this document. However, this document is written for being used by <organisation / other project / company etc.> as <a contribution to standardisation / material for consideration in product development etc.>.

In case of Consortium confidential (CO):

The information contained in this document is the proprietary confidential information of the 6G BRAINS consortium and may not be disclosed except in accordance with the consortium agreement.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the 6G BRAINS consortium as a whole, nor a certain part of the 6G BRAINS consortium warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, accepting no liability for loss or damage suffered by any person using this information.

The EC flag in this document is owned by the European Commission and the 5G PPP logo is owned by the 5G PPP initiative. The use of the EC flag and the 5G PPP logo reflects that 6G BRAINS receives funding from the European Commission, integrated in its 5G PPP initiative. Apart from this, the European Commission and the 5G PPP initiative have no responsibility for the content of this document.

The research leading to these results has received funding from the European Union Horizon 2020 Programme under grant agreement number 101017226 – 6G BRAINS – H2020-ICT-2020-2.

Impressum

[Full project title]	Bring Reinforcement-learning Into Radio Light Network for Massive Connections
[Short project title]	6G BRAINS
[Number and title of work-package]	WP5 AI-enabled Directional Slicing RAN Slicing and E2E Network Slicing
[Number and title of task]	T5.1 Novel AI-based Directional Radio Access and Backhaul Network Slicing T5.2 Novel Hybrid Network Slicing for the Backbone Network
[Document title]	D5.1 E2E Network Slicing Control Enablers
[Editor]	Ta Dang Khoa Le (ECOM)
[Work-package leader]	Qi Wang and Jose M. Alcaraz Calero, UWS
[Estimation of PM spent on the Deliverable]	30 PMs

Executive summary

Smart end-to-end (E2E) network slicing is essential to achieve resource-efficient network traffic control and satisfy the varied requirements in quality of service for diverse use cases in beyond 5G and 6G networks. This document presents the up-to-date advances in the EU 5G-PPP 6G BRAINS project in the design and prototyping of the key enablers for smart network slicing over the E2E data path including the radio access network (RAN), edge, transport and core networks. Specifically, the following achievements can be highlighted:

- Enablers for RAN and Core slicing based on the proven open-source 5G and beyond OAI platform including FlexRIC – a flexible and efficient Software Development Kit (SDK) that enables to build service-oriented RAN controllers, and FlexCN – a solution that brings programmability into the Core Network.
- Enablers for network slicing over edge, transport and core networks (wired network segments). Different technical approaches are investigated for high-performance network slicing at the data plane by leveraging data plane programmability. The software-based approach explores the Linux Traffic Control and the Open Virtual Switch (OVS) schemes for low-cost implementation, whilst the hardware-based approach benefits from a programmable FPGA platform for accelerated processing. Experimental evaluation shows that the prototypes achieve data plane network slicing at the speed of close to 20 Gbps in the software-based approach (OVS) and under 0.1ms delay for the prioritized slices in the hardware-based approach.
- Initial algorithmic investigations into the possibilities of using Machine Learning for Radio Link Control and RAN Slice Scheduling, with interesting first insights in the case of Radio Link Control.
- A Reinforcement Learning SDK named TheRLib, which assists the development process of the Cognitive Plane, currently being prototyped with practical first results.

In summary, this deliverable helps establish a more concrete vision of work package 5 in terms of conceptual understandings of the components and what possibilities are there for integrations. Finally, it is noted that these components and investigations are subject to updates and revisions in the due course of the project, in line with the technical evolution in a research project.

List of authors

Company	Author	Contribution
ECOM	Robert Schmidt Ta Dang Khoa Le Hung Nguyen Osama Arouk	RAN slice control enablers (FlexRIC) Joint-slicing in the RAN (section 3.5) Contribute to Section 4 (FlexCN) Contribute to Section 6 (Tlirematics)
UWS	Ruben Ricart Sanchez, Antonio Matencio Escolar Pablo Salva Garcia Enrique Chirivella Perez Ignacio Sanchez Navarro Mohamed Khadmaoui-Bichouna Jose M. Alcaraz Calero Qi Wang	Wired segments' network slice control enablers Lead Sections 4.1, 4.2 and 4.3 Draft Abstract and Introduction Draft Section 2
TSG	Ludovic Carre Alexandre Kazmierowski	Contribute to Section 6 (RLOps) Contribute to Section 6 (RLOps)
UBrunel	John Cosmas	Radio Link Control (section 3.4)

Table of Contents

Executive summary	4
List of authors.....	5
Table of Contents	6
List of figures and tables	8
Abbreviations	10
1 Introduction.....	13
1.1 Objective of this document	13
1.2 Overall methodology	13
1.3 Structure of this document	13
2 Overall E2E Network Slicing Vision	15
3 RAN Slice Control Enablers.....	16
3.1 FlexRIC Overview	16
3.2 FlexRIC SDK Architecture	18
3.2.1 FlexRIC Agent.....	18
3.2.2 FlexRIC Controller.....	20
3.2.3 E2 Protocol Abstraction.....	21
3.2.4 Implementation of the FlexRIC SDK	22
3.3 FlexRIC Performance Evaluation	22
3.3.1 Overhead in the User Plane	22
3.3.2 Impact of E2AP/E2SM Encoding.....	23
3.3.3 Scalability of the Controller.....	24
3.3.4 Comparison to O-RAN RIC.....	25
3.4 Reinforcement Learning Radio Link Control Experimental Considerations	26
3.4.1 Traffic Traces	26
3.4.2 Channel Traces and Block Error Ratios.....	34
3.4.3 Channel Traces from Digital Twin	37
3.5 Reinforcement Learning for Joint Slice Scheduling in the RAN.....	40
3.5.1 The Scenario	40
3.5.2 Problem Modelling.....	40
3.5.3 Proposed Solution	42
3.5.4 Experimental Considerations	45
4 Backbone/Core Network Slice Control Enablers.....	46
4.1 Overview of backbone segments network slicing.....	46
4.2 Hardware-based network slicing.....	47
4.2.1 NetFPGA Platform	47
4.2.2 NetFPGA Network Slicing Module	49
4.2.3 NetFPGA API	51

4.2.4	Empirical Results	54
4.3	Software-based network slicing	57
4.3.1	Traffic Control (TC)	57
4.3.2	Architectural design	58
4.3.3	Queuing schema and filtering expressions	58
4.3.4	Virtual Switch with Network Slicing Support (Based on OVS)	60
4.3.5	Architectural design of the UWS-OVS Software Datapath	61
4.3.6	6G Multi-tenant Traffic Classifier	62
4.3.7	UWS-OVS Virtual Software Datapath API	64
4.3.8	UWS-OVS Virtual Software Datapath Empirical Results	66
4.4	FlexCN-based core network slicing.....	69
5	Initial Integration towards E2E Network Slice Control	71
5.1	Trirematics for the MANO of E2E Network Slicing.....	71
5.1.1	Hydra – Supporting Heterogeneous Deployment.....	71
5.1.2	Infrastructure Resources and Mobile Network Resources	71
5.1.3	Mon-X – The Monitoring Plane	71
5.1.4	AI/ML – The Cognitive Plane	72
5.1.5	Athena – The Brain for Controlling Trirematics	72
5.1.6	GitOps for Continuous Integration.....	72
5.2	The Reinforcement Learning Training Library	73
5.2.1	Training with TheRLib.....	73
5.2.2	Therenv.....	74
5.2.3	Therconnect	75
5.2.4	TheRLib.....	75
5.2.5	Therutils.....	76
5.2.6	Therbench	77
5.2.7	Therdash.....	78
6	Conclusions.....	82
	References.....	83

List of figures and tables

List of figures

Figure 1. 6G BRAINS functional architecture for smart network slicing	15
Figure 2. The FlexRIC SDK consists of an agent and a server library	16
Figure 3. Using the FlexRIC SDK, different specialized controllers can be built	17
Figure 4. The FlexRIC agent architecture and integration with a user plane implementation	18
Figure 5. The agent library handles multiple controllers	19
Figure 6. Normalized CPU usage of FlexRIC and FlexRAN	23
Figure 7. Comparison of E2AP/E2SM encoding schemes.....	23
Figure 8. CPU usage at the controller.....	24
Figure 9. E2AP round-trip times for two hops.....	25
Figure 10. Event list to harmonise the times from different traffic traces and events occurring in network simulator	26
Figure 11. 2-state Markov Model.....	27
Figure 12. 5G/6G OFDM Transmitter and Receiver	34
Figure 13. Measured throughput of a radio channel against SNR for different Modulation and Coding Schemes.....	36
Figure 14. Graph of Block Error Probability against ESNR for different CQI	37
Figure 15. Algorithm for calculating Successful or Errored Block Transmission	37
Figure 16. Pictures of the measured scenario: (a) view from RX1 towards TX1, (b) view from TX2 towards RX2, and (c) RT model.	38
Figure 17. Received and Equalised 4-QAM constellations for SCS = 120 kHz, BW = 400MHz, QAM Order = 4-QAM, fc = 190GHz in conference room	38
Figure 18. Received Frequency Response and BER for RBs of Equalised 4-QAM constellations for for SCS = 120 kHz, BW = 400MHz, QAM Order = 4-QAM, fc = 190GHz in conference room	39
Figure 19. DD-S-UU TDD format in NR	41
Figure 20. Grant-based Transmission in Dynamic UL.....	41
Figure 21. Overall Vision.....	46
Figure 22. NetFPGA data plane and network slicing implementation	47
Figure 23. Basic NetFPGA Network Slicing Module implementation.....	49
Figure 24. Advanced NetFPGA Network Slicing Module implementation	50
Figure 25. Match/Action pseudo-algorithm implementation for the management of the P4-NetFPGA-based queues developed.....	51
Figure 26. Delay achieved when the number of flexible slice instances is increased.....	55
Figure 27. Packet loss achieved when the number of flexible slice instances is increased	56
Figure 28. Jitter achieved when the number of flexible slice instances is increased.....	56
Figure 29. Architecture of Traffic Control with network slicing support	58
Figure 30. Example of a queuing schema in Traffic Control.....	59
Figure 31. Example of a filtering expression in Traffic Control	59
Figure 32. Architecture of the OVS-based Virtual Switch with network slicing support	61
Figure 33. Creation and deletion of a slice in the software data path.....	65
Figure 34. Software slicing architecture.....	65

Figure 35. ITU category for every vertical oriented use case for evaluation of the UWS-OVS Virtual Software Datapath	66
Figure 36. Scalability results for the UWS-OVS component. Average delay when ranging the number of slices (verticals) and the number of IoT devices per slice. It is given the maximum number of IoT per Slice, packets per second (PPS) transmitted and accumulated Tx bandwidth configured for each use case	68
Figure 37. The FlexCN Architecture.....	69
Figure 38. Trirematics Architecture.....	72
Figure 39. Therenv's conceptual role in the RL pipeline	75
Figure 40. The connector is used to link python environment representations to industrial simulations	75
Figure 41. TheRLib holds environment agnostic RL features	76
Figure 42. Therutils logs neural network architectures and experiment parsers	77
Figure 43. Therbench is used to log experiments for non-regression testing	77
Figure 44. The dashboard is used to visualize experiments.....	78
Figure 45. The dashboard is used to monitor training experiments.....	79
Figure 46. We create graphs and log experiment data to follow the training process	80
Figure 47. View of the DRL agent Neural Network architecture.....	80
Figure 48. Episode viewer of trained agents.....	81

List of tables

Table 1. Possible combinations of FlexRIC abstractions	22
Table 2. 2-State Model parameters for Video LAN Client, ffMPEG, Industrial Programmable Logic Controller and Industrial Camera traffic traces.....	28
Table 3. Simulation Parameters	34
Table 4. Modulation Orders, Code Rates and Spectral Efficiencies for different 5G Modulation and Coding Schemes	35
Table 5. Parameters required to add a new rule to a table implemented in hardware	51
Table 6. Values used in fields keymask9, keymask10 and keymask11 to identify different encapsulation protocols.....	52
Table 7. Values required to invoke the different actions provided	53
Table 8. Pcaps modified to test the slicing in the experimental setup proposed	54
Table 9. Maximum value of delay, packet loss and jitter guaranteed in a congested scenario, where 512 users are transmitting traffic.....	57
Table 10. Open Flow fields for traditional 5-tuple based flow key	62
Table 11. New Fields included in the extended 6G flow key	63
Table 12. Open Flow fields for traditional 5-tuple based flow key	64
Table 13 - Vertical oriented use cases for evaluation of the UWS-OVS Virtual Software Datapath: Characteristics, KPIs and traffic profile	67
Table 14 - TheRLib modules in the RL Components.....	73

Abbreviations

5G	Fifth Generation (mobile/cellular networks)
5G PPP	5G Infrastructure Public Private Partnership
6G BRAINS	Internet of Radio Light (project)
AI	Artificial Intelligence
AMF	Access and Mobility Functions
AUSF	Authentication Server Function
BLER	Block Error Ratio
BSR	Buffer Status Reports
CAM	Content-Addressable Memory
CAPEX	Capital Expenditure
CNF	Cloud-native Network Function
CU	Centralised Unit
DDPG	Deep Deterministic Policy Gradient
DMA	Direct Memory Access
DM-RS	Demodulation Reference Symbols
DPI	Deep Packet Inspection
DRL	Deep Reinforcement Learning
DU	Distributed Unit
E2E	End-to-end
eMBB	Enhanced Mobile Broadband
FB	Flatbuffers
HARQ	Hybrid Automatic Repeat Request
iApps	Controller-internal Applications
IoT	Internet of Things
KPI	Key Performance Indicator

MANO	Management and Orchestration
M2M	Machine to Machine
ML	Machine Learning
MLOps	Machine Learning Operations
mMTC	Massive Machine Type Communication
NBI	North Bound Interface
NFV	Network Function Virtualisation
NN	Neural Networks
OPEX	Operational Expenditure
PBT	Population Based Training
PCI	Peripheral Component Interconnect
PNF	Physical Network Function
PRB	Physical Resource Blocks
PUSCH	Physical Uplink Shared Channel
QoE	Quality of Experience
QoS	Quality of Service
R&D	Research and Development
RAN	Radio Access Network
RB	Resource Blocks
REST	Representational State Transfer
RSS	Receive Side Scaling
RTT	Round-Trip Time
SDK	Software Development Kit
SDN	Software Defined Networks
SLA	Service-level Agreement
SMF	Session Management Function

SOTA	State of the Art
SWOT	Strengths, Weaknesses, Opportunities, and Threats analysis
TCAM	Ternary Content-Addressable Memory
TDL	Tapped Delay Line
UDM	Unified Data Management
UPF	User Plane Function
URL	Uniform Resource Locator
URLLC	Ultra-reliable Low Latency
VNF	Virtual Network Function
xApps	Controller-external Applications
XDP	Express Data Path

1 Introduction

Network slicing is a cornerstone technology in 5G and beyond networks to meet the diverse requirements of different vertical/industrial use cases such as those Industry 4.0 and beyond use cases defined in 6G BRAINS [D2.1]. 6G BRAINS envisions that 6G networks would expect to achieve advanced End-to-End (E2E) network slicing capabilities, especially Artificial Intelligence (AI) based Radio Access Network (RAN) slicing, and high-speed and flexible backbone network slicing across the Multi-access/Mobile Edge Computing (MEC), the Transport Network, and the Core Network by exploring advanced data plane programmability to truly guarantee the Service-Level Agreements (SLAs).

This document reports the up-to-date progress towards achieving smart E2E network slicing. The scope of this deliverable mainly focuses on T5.1 and T5.2 in design and prototyping of enabling technologies for E2E network slicing, and additionally it provides initial investigation of the integration aspects considering E2E network slicing between T5.1 and T5.2, and the AI by design between T5.1 and T5.4.

1.1 Objective of this document

The objectives are listed below:

- Advance the functional architecture defined in D2.2 (Section 4.1) with concrete technology selection;
- Describe the technical details and experimental results of the design and prototypes related to E2E network slice control;
- Promote open-source based technical approaches for advanced network slicing;
- Present initial design and experimental results for the AI required to achieve smart network slicing;
- Outline the initial integration consideration to achieve smart network slicing.

1.2 Overall methodology

The overall approach is summarised as follows:

- Network segment-specific control for fine-grained and optimised traffic engineering (D2.2 Section 2.1);
- Advanced automation and Artificial Intelligence (AI) or Machine Learning (ML) support by design (D2.2 Section 2.1);
- Design and prototyping based on proven realistic platforms especially open-sourced platforms such as OpenAirInterface (OAI), Mosaic5G including Flexible RAN intelligent controller (FlexRIC), Flexible Core controller (FlexCN), OpenVSwitch (OVS), NetFPGA etc.

1.3 Structure of this document

The rest of the document is organized as follows:

- Section 2 provides the overall vision for 6G BRAINS E2E network slicing across the various network segments;
- Section 3 presents the RAN control enablers and initial AI design for network slicing over the air;

- Section 4 describes the control enablers for the non-RAN segments;
- Section 5 reports initial Integration towards E2E network slice control;
- Section 6 concludes this document.

2 Overall E2E Network Slicing Vision

Figure 1 illustrates the 6G BRAINS functional architecture for smart network slice control, management and orchestration.

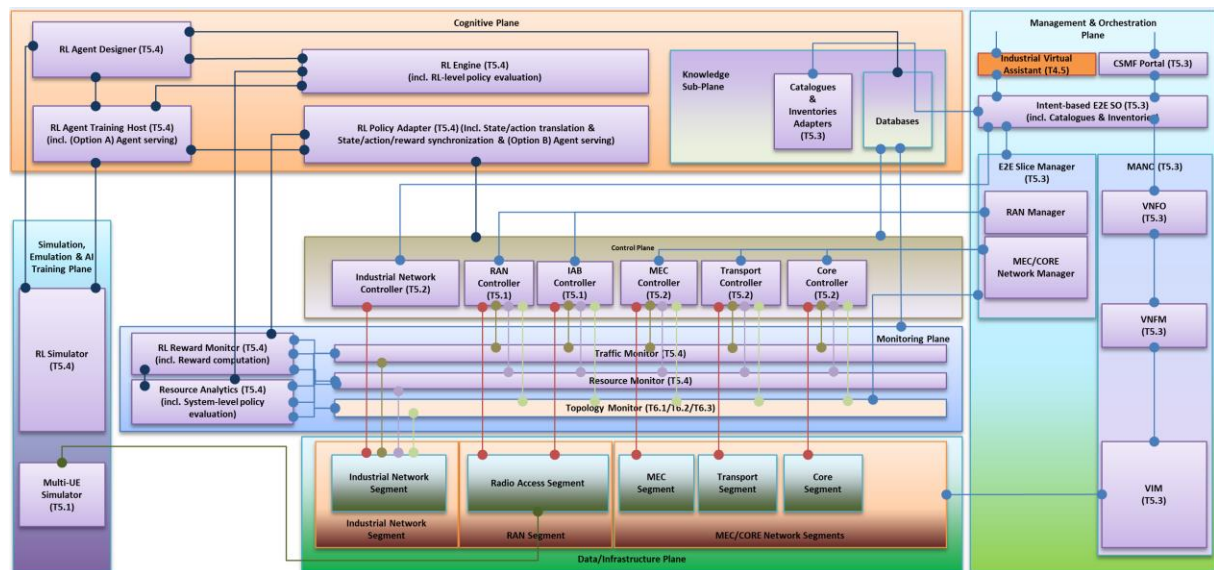


Figure 1. 6G BRAINS functional architecture for smart network slicing

This deliverable focuses on the enabling technologies for the control of E2E network slices across the different network segments especially the essential RAN, MEC, Transport and Core Networks. As shown in Figure 1, these Control Plane enablers refer to the RAN Controller, MEC Controller, Transport Controller and Core Controller, addressing the network slicing in the corresponding network segments in the Data/Infrastructure Plane. Furthermore, it is noted that the monitoring functional blocks (Traffic, Resource, Topology Monitors) in the Monitoring Plane are primarily prototyped as either a part of the controllers or controller applications (Apps).

In addition, this deliverable also reports the initial design of some of the enablers in the Cognitive Plane to provide AI capabilities to the RAN slice control. Specifically, in subsection 3.5, we will algorithmically investigate a possible application of ML in enabling the RAN slicing use case of *relaxed uRLLC*, where the standard strict requirements of uRLLC are relaxed to allow higher multiplexing gains in the public network. Then, in subsection 5.2, we will specify the key components that are used to implement and deploy such ML. Together, they demonstrate a scalable example of the Cognitive Plane for E2E network slicing.

3 RAN Slice Control Enablers

3.1 FlexRIC Overview

FlexRIC is an SDK, consisting of a server library and an agent library with two optional extensions: controller-internal Applications (iApps) and communication interfaces. The objective of the SDK is to facilitate the realization of specialized SD-RAN controllers to target specific use cases, while being simple to use. FlexRIC does not support extra features endogenously following the zero-overhead principle. In its simplest form, the SDK can be used to implement an SD-RAN controller using an E2-compatible protocol, as it is shown in Figure 2.

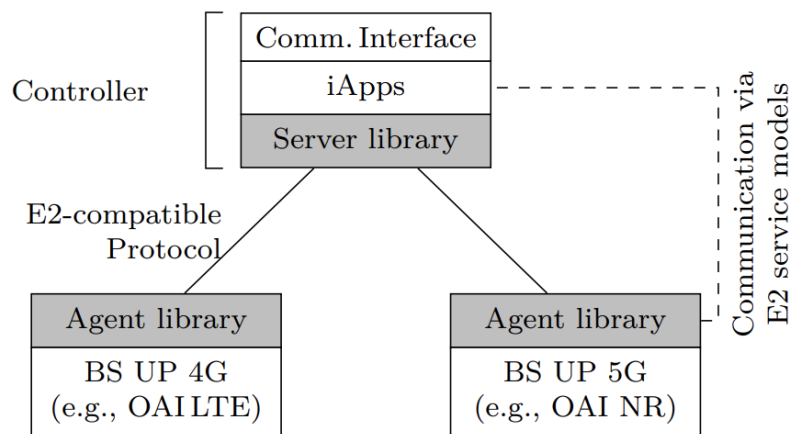


Figure 2. The FlexRIC SDK consists of an agent and a server library

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

A controller is built through the server library, iApps, and optionally a communication interface. The server library manages agent connections, and routes messages between iApps and the agents. Through the iApps, it is possible to modularly build specialized controllers: based on the considered use case, iApps can implement service models themselves, or expose information to external Applications (xApps) via a northbound communication interface. In the latter case, xApps can control the RAN while being functionally isolated from the controller, which is the favored method of the reference O-RAN RIC, but bears a certain overhead. Finally, it is even possible to recursively expose an agent interface at the northbound by reusing the agent library, as shown in Figure 3. Such a virtualization layer delegates control to multiple (per-slice) controllers, each possibly operating on a different set of E2SMs, without controlling the network itself. This recursive property allows not only to compose various specialized controllers into a multi-service SD-RAN infrastructure but also abstract out the heterogeneity of the 4G/5G deployment topology.

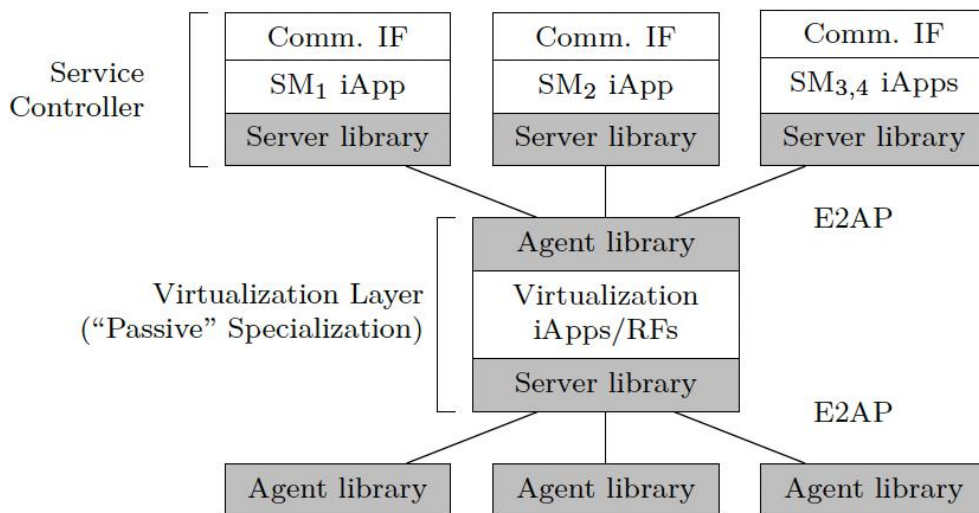


Figure 3. Using the FlexRIC SDK, different specialized controllers can be built

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

3.2 FlexRIC SDK Architecture

3.2.1 FlexRIC Agent

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

Agent Architecture

The architecture of an agent using the FlexRIC SDK is shown in Figure 4. It consists of the agent library, and a user-plane implementation, such as a specific base station or a testing agent.

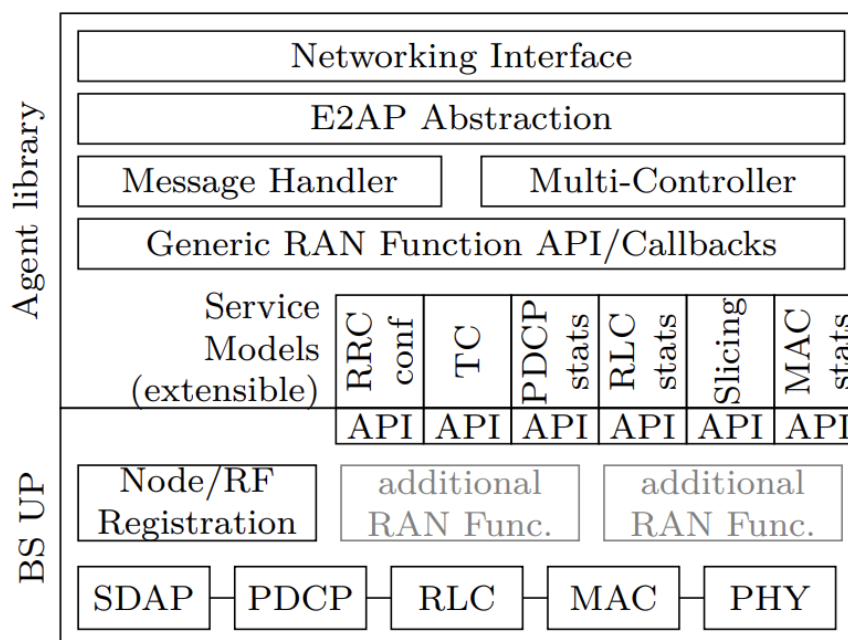


Figure 4. The FlexRIC agent architecture and integration with a user plane implementation

The agent library provides the necessary support to connect to a controller. It consists of a networking interface, the E2AP abstraction, a message handler, and a generic RAN function API. The E2AP abstraction provides an intermediate representation for the E2 protocol, relieving RAN functions from E2 protocol specificities (encoding and decoding). Further, the agent provides the generic RAN function API to implement RAN functions with custom service model-specific logic. This API defines callbacks for E2AP messages, i.e., (i) subscription requests (for new information subscription), (ii) subscription delete request (removal of subscriptions), and (iii) control messages (to trigger service-model-specific actions), which need to be implemented by RAN functions.

Finally, we pre-defined service models for the considered use cases, namely slicing control and traffic control. The service models are tailored towards specific RAN layers (for statistics) or RAN control endpoints (for control) to easily integrate the agent library in disaggregated base stations, and expose a simplified API to the generic RAN function API, facilitating the integration with various base station implementations.

The base station provides basic node information, such as the configured PLMNs and slices, and registers RAN functions according to the underlying node's capability. Unlike what is

shown in Figure 4, not all RAN layers are present in a node for disaggregated base stations with CU and DU, and FlexRIC natively supports such disaggregation through the selection of the appropriate RAN functions. The base station uses the interface of pre-defined RAN functions to expose data and handle control messages, or it defines additional RAN functions using the generic RAN function API.

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

Multi-controller support at the agent library

The agent library provides means to connect to multiple controllers. This becomes useful in a multi-service context, e.g., for “recursive” controllers that virtualize RAN control or base station hypervisors like Orion [1], which expose information to and allow control by multiple controllers while also providing isolation between them.

The FlexRIC agent library enables multiple service controllers, as shown in Figure 5, and assists their handling through (1) management of additional controllers (setup, teardown, providing controller origin to RAN functions for message handling), and (2) a UE-to-controller association.

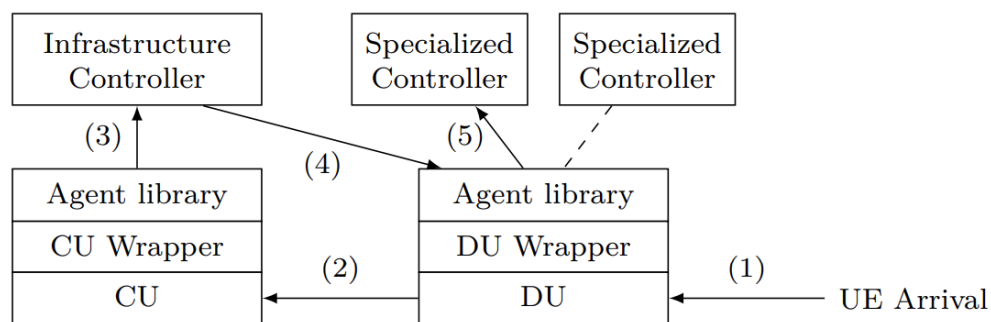


Figure 5. The agent library handles multiple controllers

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

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

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

3.2.2 FlexRIC Controller

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

Controller Specializations

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

The iApps implement specific controller behavior, either directly through E2SMs within the iApps themselves, or by providing platform services that can be leveraged by external applications (xApps). For this purpose, a controller specialization typically exposes a northbound communication interface using a custom protocol, such as a simple REST interface (e.g., FlexRAN [2]), the RMR library (e.g., O-RAN RIC), a message broker (e.g. Redis), or E2AP itself.

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

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

FlexRIC Server Library

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

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

The subscription management’s task is to (i) keep track of existing subscriptions and (ii) route arriving subscription-related messages to the corresponding iApps. When an iApp requests a

new subscription directly or on-behalf of an xApp, it provides a set of callbacks that are called to inform the iApp about the subscription outcome, and to dispatch the corresponding indication messages. When a message arrives, the subscription management simply selects the iApp for which the message is sent and forwards it through the provided callback.

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

3.2.3 E2 Protocol Abstraction

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

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

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

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

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

Finally, we did not create an abstraction around the semantic of E2AP, as this would imply to completely redesign the protocol (e.g., making it stateless instead of the current stateful implementation that is achieved, for example through the setup request procedure).

Table 1 lists the abstraction levels and possible combinations for some example controllers.

Table 1. Possible combinations of FlexRIC abstractions

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

3.2.4 Implementation of the FlexRIC SDK

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

3.3 FlexRIC Performance Evaluation

3.3.1 Overhead in the User Plane

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

We measure the CPU usage over both LTE and NR (non-standalone mode, NSA) base stations. We used an RF setup with Ettus B210 radios for both cells. The LTE cell runs on an Intel Core i7 with 8 cores @ 3.2 GHz, uses a bandwidth of 5 MHz (25 RBs) and serves 3 UEs at MCS 28. The NR cell runs on an Intel Xeon Gold 6208U with 16 cores @ 2.9 GHz, has a bandwidth of 20 MHz (106 RBs) and serves 3 UEs at MCS 20. Figure 6a shows that both FlexRIC and FlexRAN incur a small overhead. The relative overhead decreases when deploying FlexRIC over NR, due to a more demanding PHY.

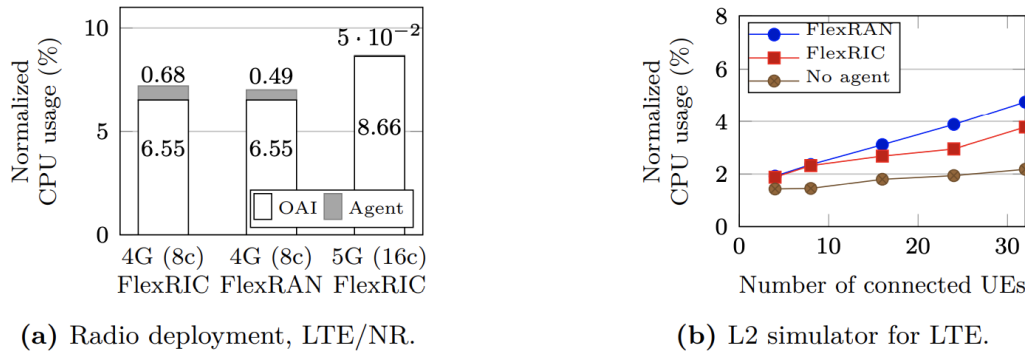


Figure 6. Normalized CPU usage of FlexRIC and FlexRAN

To analyze the overhead for more UEs, we used the “L2 simulator” of OAI, an emulation mode without the physical layer. Figure 6b shows that FlexRIC performs slightly better than FlexRAN especially for more UEs, and thus, FlexRIC is valid for the same scenarios where FlexRAN has been validated (e.g., real-time MAC scheduling).

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

3.3.2 Impact of E2AP/E2SM Encoding

In the following, we evaluate the impact of two encoding schemes, Flatbuffers (FB) and ASN.1, for E2AP and the E2SM. E2 enforces a double encoding of messages: a first encoding pass is done for the “inner” E2SM, and then again for the “outer” E2AP. We modified the “Hello World” service model (HW-E2SM) provided by O-RAN to perform a ping by sending a control message to the RAN function, to which the agent responds with an indication message. To also study the impact of the E2SM encoding, we translated the E2SM 1:1 from ASN.1 to FB.

To measure the Round-Trip Time (RTT), the iApp pings the agent every second for small (100 B) and medium (1500 B) message payloads. The results in Figure 7a indicate that switching from all-ASN.1 to all-Flatbuffers encoding reduces the average RTT by roughly 25 % and 66 % for small and medium payloads, due to the encoding overhead of ASN.1. Using an ASN.1/FB (E2AP/E2SM) encoding even increases the round-trip time, since the larger FB E2SM message (for each FB message, we observe 30-40B overhead) needs to be encoded again by ASN.1 for E2AP. Thus, where ultra-low latency is required, FB might be preferable, especially for larger payloads, but networking delay (our Ethernet-based campus network has RTTs below 1 ms) would make this difference much less pronounced.

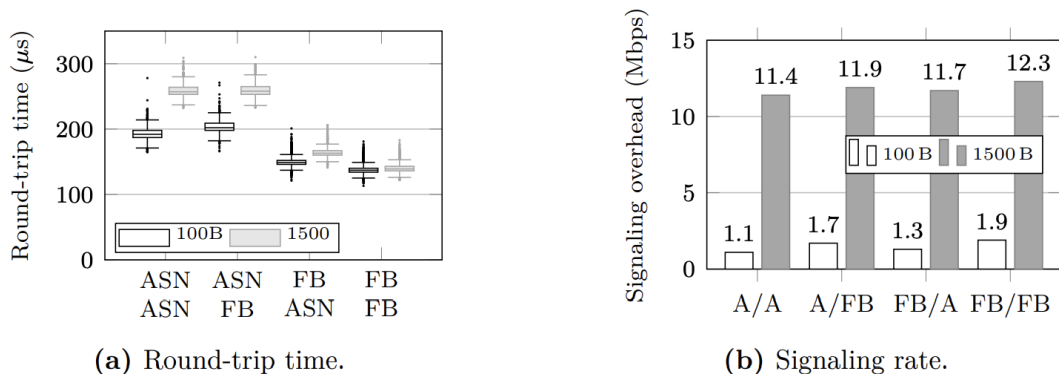


Figure 7. Comparison of E2AP/E2SM encoding schemes

To put the RTT improvements into perspective, we measured the generated message signaling rate for a high ping rate (one packet every 1 ms, which is 4G's transmission time interval). The results in Figure 7b indicate that switching from an ASN.1/ASN.1 to an FB/FB encoding increases the signaling rate by 80% for small payloads due to the overhead of FB; for large payloads, however, the signaling rate overhead is almost negligible. Comparing the "mixed" encodings, we see that the FB/ASN.1 encoding signaling overhead only slightly increases compared to ASN.1/ASN.1, whereas the ASN.1/FB combination does not decrease signaling load, rendering this combination useless.

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

3.3.3 Scalability of the Controller

Next, we evaluate the scalability of the FlexRIC server library in terms of CPU utilization and memory usage and compare it to FlexRAN. We use the FlexRAN controller specialization (in the following called simply "FlexRIC" to avoid confusion with the original FlexRAN controller), which employs a statistics iApp that saves incoming messages to an in-memory data structure like FlexRAN. We only consider the agent-to-controller direction, which typically carries more traffic than in the opposite direction, even for high-traffic scenarios such as remote scheduling. Both controllers are on an Intel Core-i7 machine with 12 cores at 3.2 GHz. As can be seen in Figure 8a, FlexRIC incurs only one tenth of the CPU usage of FlexRAN, due to using Flatbuffers instead of Protobuf. Also, FlexRIC organizes its internal data structure more efficiently, leading to reduced memory consumption.

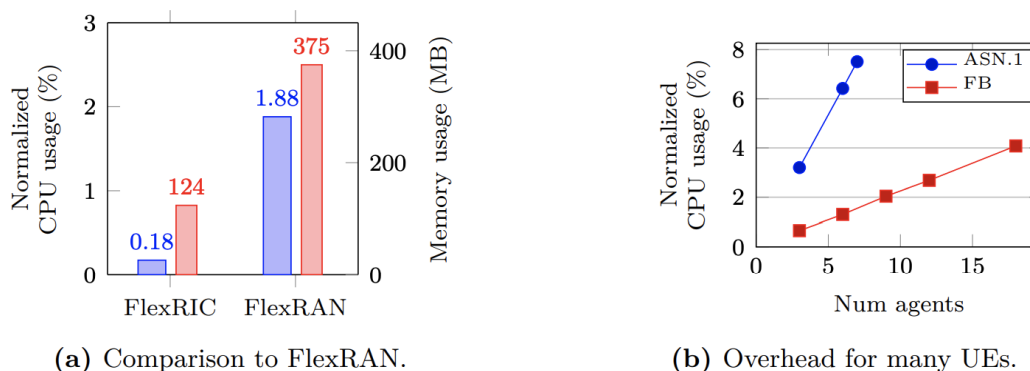


Figure 8. CPU usage at the controller

To further understand the limitations of FlexRIC, we test using dummy test agents (not connected to any base station) that export the same statistics (in FB) as from a real base station, each agent emulating a connection of 32 UEs with a unique default bearer. Figure 8b shows the CPU utilization of FlexRIC over varying number of agents when using a FB or ASN.1 encoding for E2AP. It is apparent that FB has around 4 times lower CPU usage than ASN.1. Since FB's design avoids an explicit decoding step, reading directly from "raw" bytes, the subscription management in the server library can look up the corresponding subscription much faster, resulting in less CPU usage, directly translating to serving many more agents. This suggests that ASN.1 encoding on E2AP can become a limiting factor in terms of CPU, whereas

FB is rather limited by the network, as for 18 agents, the signaling reaches almost 700 Mbps. When reducing the message frequency to 10 ms, the FlexRIC SDK was furthermore able to handle around 100 agents (not depicted in the figures), confirming FlexRIC's scalability.

3.3.4 Comparison to O-RAN RIC

The reference O-RAN RIC architecture relies on micro-services orchestrated by Kubernetes. The O-RAN architecture impacts the latency since it imposes two hops for messages (from xApp via "E2 termination" to agent). In the following experiment, we measured the round trip time with two distinct payloads (i.e., 100 B and 1500 B), between (i) a FlexRIC agent and a FlexRIC controller utilizing the E2SM-HW, and (ii) a FlexRIC agent and O-RAN RIC. In FlexRIC, we use a relaying controller to emulate two hops, which, unlike O-RAN RIC, is not imposed by FlexRIC but added to carry out a fair comparison. The relaying controller serves as a transparent relay for any E2 messages between any northbound controller and southbound agent. As it can be observed from Figure 9, O-RAN RIC increases the round-trip time by at least three times for small and two times for medium load compared to FlexRIC. Note that FlexRIC incurs less jitter for the round-trip time.

In a second experiment, we sent 1500 B payload messages in a 1 ms interval and measured the incurred system load using iostat over 180 s. In essence, FlexRIC incurred 40 % less load (1,5 % vs. 2,5 % total system load). This finding confirms that O-RAN RIC imposes additional overhead by design for communication between xApps and RAN function even if not required, which might become a bottleneck. However, we acknowledge that the O-RAN RIC is designed for commercial deployments, where security is of high concern.

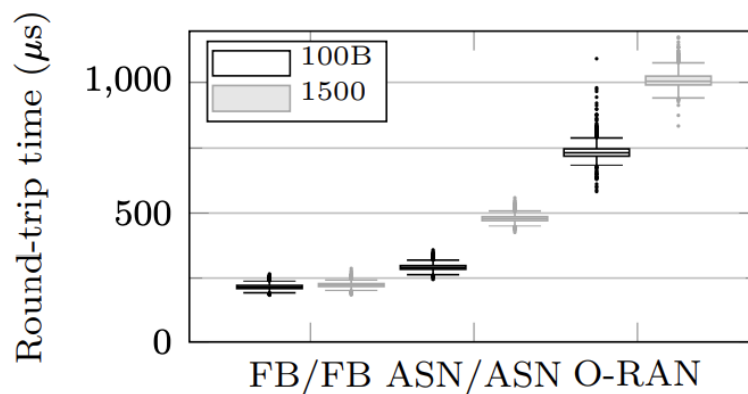


Figure 9. E2AP round-trip times for two hops

3.4 Reinforcement Learning Radio Link Control Experimental Considerations

The most important consideration in all simulations is the level of abstraction of the system being modeled and whether that provides a faithful model to obtain results that emulates the system under study sufficiently accurately enough to provide the required performance results to design and analyse system under study. The models under consideration are: (a) Traffic Traces, (b) Block Error Ratios against Signal to Noise Ratio for different Modulation and Coding Schemes (c) Digital Twin.

3.4.1 Traffic Traces

Wireshark was used to obtain four traffic traces, namely: Video LAN Client, ffMPEG, Industrial Programmable Logic Controller and Industrial Camera. The traces were filtered to obtain the relevant packet data required for network simulations for, namely:

- VLC & ffMPEG video players: Packet Number, Packet departure time, source IP address, destination IP address, Protocol, Packet size, Packet arrival time and player.
- PLC controller
 - Main PLC data: Packet Number, Packet departure time, Server MAC Address, Transmission Mode, Protocol, PacketSize, data in packets
 - Ancillary PLC data: Packet Number, Packet departure time, Server MAC Address, Destination MAC address, Protocol, PacketSize, data in packets
- Industrial Video Camera: Packet Number, Packet departure time, source IP address, destination IP address, Protocol, Packet size

Files of the traffic traces can be accessed and used directly in a network simulator but an event list is needed to harmonize the different departure times of packets and any other processes that may be occurring in the network simulation such as the departure time of slots within 5G Resource Blocks (RBs). A simplified schematic of this event list is shown in Figure 10.



Figure 10. Event list to harmonise the times from different traffic traces and events occurring in network simulator

Multiple instances of traffic can be generated by simply rotating the times of the traffic traces by different time offsets. Alternatively, a 2 State Markov Model, as shown in Figure 11, can be characterised and can be used for generating alternative traffic but it will never reproduce traffic that is as true to the traces from which it was characterised. It only provides traffic that has the same statistical parameters to the original traffic trace. It is characterised by subdividing time into slot intervals and recording the number of packets that have arrived in any interval. Slots within which one or more packets have arrived are 'state 1' burst period slots whereas slots where no packets have arrived are 'state 0' silent interval slots. The probabilities of transitioning between silent intervals and burst periods (P_{01} , P_{10}) or

remaining in silent intervals (P_{00}) or burst periods (P_{11}) can also be measured. Long term probability of being in state 1 (P_{i1}) or state 0 (p_{i0}) can also be calculated and used to calculate mean bit rates. Table 2 presents the 2-State Model parameters for Video LAN Client, ffmpeg, Industrial Programmable Logic Controller and Industrial Camera traffic traces.

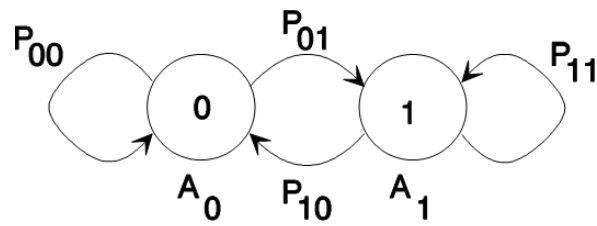


Figure 11. 2-state Markov Model

Table 2. 2-State Model parameters for Video LAN Client, ffMPEG, Industrial Programmable Logic Controller and Industrial Camera traffic traces

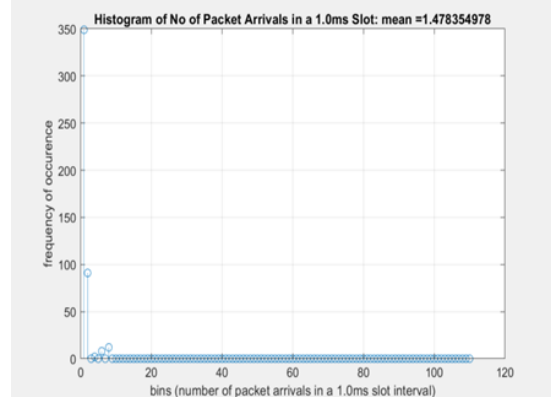
App Mean Mbps Peak Mbps Pkt Size bytes	Slot time (ms)	pi1	pi0	P11	P10	P00	P01	Probability Distribution of Number of Packet Arrivals in a Slot
ffMPEG Mn: 0.824 Pk: 1.65 Mn Pkt Size: 1462 Mn Pkt Size: 456 Min PktSize: 330	1	0.026	0.974	0.747	0.253	0.993	0.007	
VLC Mn: 7.17 Pk:22.8 Pk Pkt Size: 1358 Mn Pkt Size: 1358	1	0.010	0.990	0.483	0.517	0.995	0.005	

App Mean Mbps Peak Mbps Pkt Size bytes	Slot time (ms)	pi1	pi0	P11	P10	P00	P01	Probability Distribution of Number of Packet Arrivals in a Slot
Industrial Camera (uplink) Mn: 414 Pk: 5698 Pk Pkt size: 65226 Mn Pkt size: 11557	1.0	0.467	0.533	0.958	0.042	0.963	0.037	
Industrial Camera (downlink) Mn: 0.445 Pk: 1.989 Pk Pkt size: 67 Mn Pkt size: 66	1.0	0.232	0.768	0.947	0.053	0.984	0.016	

App Mean Mbps Peak Mbps Pkt Size bytes	Slot time (ms)	pi1	pi0	P11	P10	P00	P01	Probability Distribution of Number of Packet Arrivals in a Slot
Industrial Camera (uplink) Mn: 414 Pk: 5698 Pk Pkt size: 65226 Mn Pkt size: 11557	0.5	0.455	0.545	0.973	0.027	0.977	0.023	
Industrial Camera (uplink) Mn: 423 Pk: 5648 Pk Pkt size: 65226 Mn Pkt size: 11557	0.25	0.442	0.558	0.970	0.030	0.976	0.024	

App Mean Mbps Peak Mbps Pkt Size bytes	Slot time (ms)	pi1	pi0	P11	P10	P00	P01	Probability Distribution of Number of Packet Arrivals in a Slot
Industrial Camera (uplink) Mn: 415 Pk: 6073 Pk Pkt size: 65226 Mn Pkt size: 11557	0.125	0.291	0.709	0.479	0.521	0.786	0.214	
Programable Logic Controller (uplink) Mn: 19.27 Pk: 141.28 Pk Pkt size: 1203 Mn Pkt size: 1203	0.25	0.367	0.633	0.318	0.682	0.605	0.395	

App Mean Mbps Peak Mbps Pkt Size bytes	Slot time (ms)	pi1	pi0	P11	P10	P00	P01	Probability Distribution of Number of Packet Arrivals in a Slot
Programable Logic Controller (uplink) Mn: 19.26 Pk: 181.70 Pk Pkt size: 1203 Mn Pkt size: 1203	0.125	0.236	0.764	0.444	0.556	0.828	0.172	<p>Histogram of No of Packet Arrivals in a 0.125ms Slot: mean = 1.060376703</p>
Programable Logic Controller (uplink) Mn: 19.31 Pk: 192.48 Pk Pkt size: 1203 Mn Pkt size: 1203	0.0625	0.125	0.875	0.108	0.892	0.873	0.127	<p>Histogram of No of Packet Arrivals in a 0.0625ms Slot: mean = 1.003494962</p>

App Mean Mbps Peak Mbps Pkt Size bytes	Slot time (ms)	pi1	pi0	P11	P10	P00	P01	Probability Distribution of Number of Packet Arrivals in a Slot
Programable Logic Controller (downlink) Mn: 0.000729 Pk: 0.007799 Pk Pkt size: 344 Mn Pkt size: 174.8	1.0	3.54 e-04	1.000	0.019	0.981	1.000	3.48 e-04	

3.4.2 Channel Traces and Block Error Ratios

A radio transmission model can be used to ascertain the effect of a radio channel on the Transport Block of a typical 5G/6G OFDM system shown in Figure 12. The model was used to measure the physical uplink shared channel (PUSCH) throughput of a 5G New Radio (NR) link over an uplink transport channel (UL-SCH). Matlab was used to simulate the 5G OFDM Transmitter at the User Equipment and Receiver at the gNB.

The transmitter model includes PUSCH demodulation reference symbols (DM-RS tapped delay line (TDL)) propagation channel. At the transmitter a CRC code is generated and the CRC of the previous transmission is checked at the receiver to determine whether a retransmission is required by reporting back to the transmitter the unsuccessful transmission using the HARQ process. If that is not the case the transmitter generates new data for transmission.

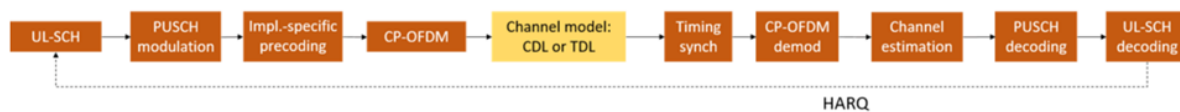


Figure 12. 5G/6G OFDM Transmitter and Receiver

Matlab was used to simulate the 5G OFDM Transmitter and Receiver using a zero-forcing equaliser. The Graphical User Interface of the simulator is presented in Figure 13, and as it can be seen parameters such as Sub-Carrier Spacing, Transmission Mode, Bandwidth, QAM-Order, Radio Channel Type, Radio Environment, K-Factor, Speed of Mobile and SNR can be selected.

Table 3. Simulation Parameters

simParameters	Parameter Value
Number of 10ms frames (NFrames)	8
NRB	52
SubcarrierSpacing	15
CyclicPrefix	'Normal'
NTxAnts	1
NRxAnts	2
PUSCH.SymbolSet	0:13
PUSCH.TransformPrecoding	false
PUSCH.PUSCHMappingType	'A' (slot-wise)
PUSCH.DMRSTypeAPosition	2 (First DM-RS symbol position)
PUSCH.Modulation	'pi/2-BPSK', 'QPSK', '16QAM', '64QAM', '256QAM'
PUSCH.TargetCodeRate	A range of values as defined in Table 4
SNRIn	-20dB to 25dB in 1dB steps
channel.DelaySpread	30e-9 seconds
channel.MaximumDopplerShift	10 Hz

The system parameters are presented in Table 3, and as it can be seen parameters such as Number of Frames, Sub-Carrier Spacing, Number of Resource Blocks, QAM-Order, Demodulation Reference Signals position, Radio Channel Delay Spread and Doppler Shift, SNR-In can be set.

Table 4. Modulation Orders, Code Rates and Spectral Efficiencies for different 5G Modulation and Coding Schemes

MCS Index I_{MCS}	Modulation Order Q_m	Target code Rate x [1024] R	Spectral efficiency
0	2	120	0.2344
1	2	157	0.3066
2	2	193	0.3770
3	2	251	0.4902
4	2	308	0.6016
5	2	379	0.7402
6	2	449	0.8770
7	2	526	1.0273
8	2	602	1.1758
9	2	679	1.3262
10	4	340	1.3281
11	4	378	1.4766
12	4	434	1.6953
13	4	490	1.9141
14	4	553	2.1602
15	4	616	2.4063
16	4	658	2.5703
17	6	438	2.5664
18	6	466	2.7305
19	6	517	3.0293
20	6	567	3.3223
21	6	616	3.6094
22	6	666	3.9023
23	6	719	4.2129
24	6	772	4.5234
25	6	822	4.8164
26	6	873	5.1152
27	6	910	5.3320
28	6	948	5.5547
29	2	reserved	
30	4	reserved	
31	6	reserved	

The output of the radio transmission model is the Percentage Throughput (i.e. Block Error Ratio (BLER)) against Signal to Noise Ratio for different Modulation and Coding Schemes were measured for all the MCSs listed in Table 4, some plots of which is presented in Figure 14 for a simulation run of 8 frames or 80 x 1ms slots. As it can be seen, the percentage throughput varies with SNR. The percentage throughput of all the Modulation Coding Schemes against SNR were saved on file as a Result5G NR-PUSCH.csv file.

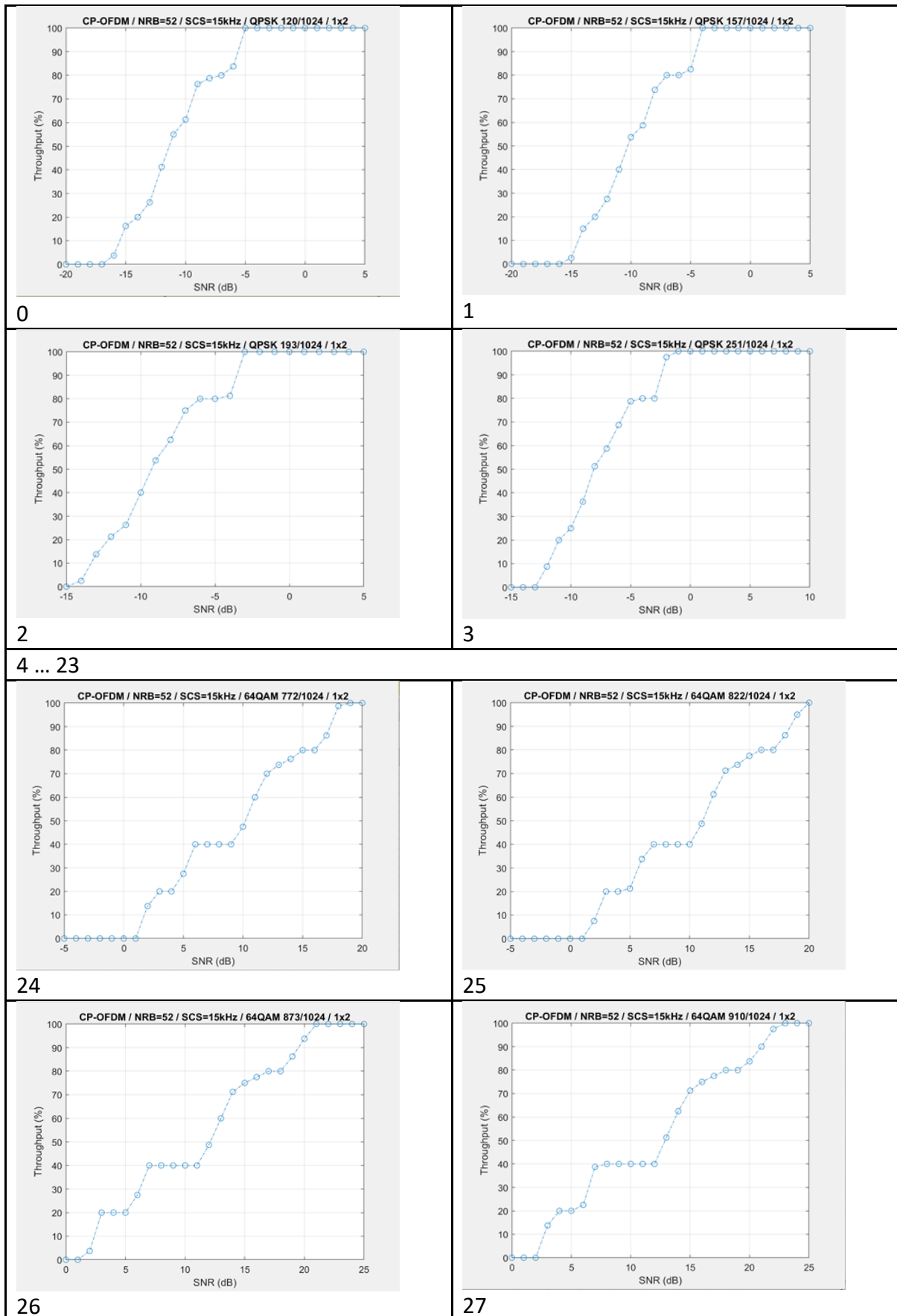


Figure 13. Measured throughput of a radio channel against SNR for different Modulation and Coding Schemes

The performance of all the MCSs is summarised in Figure 14.

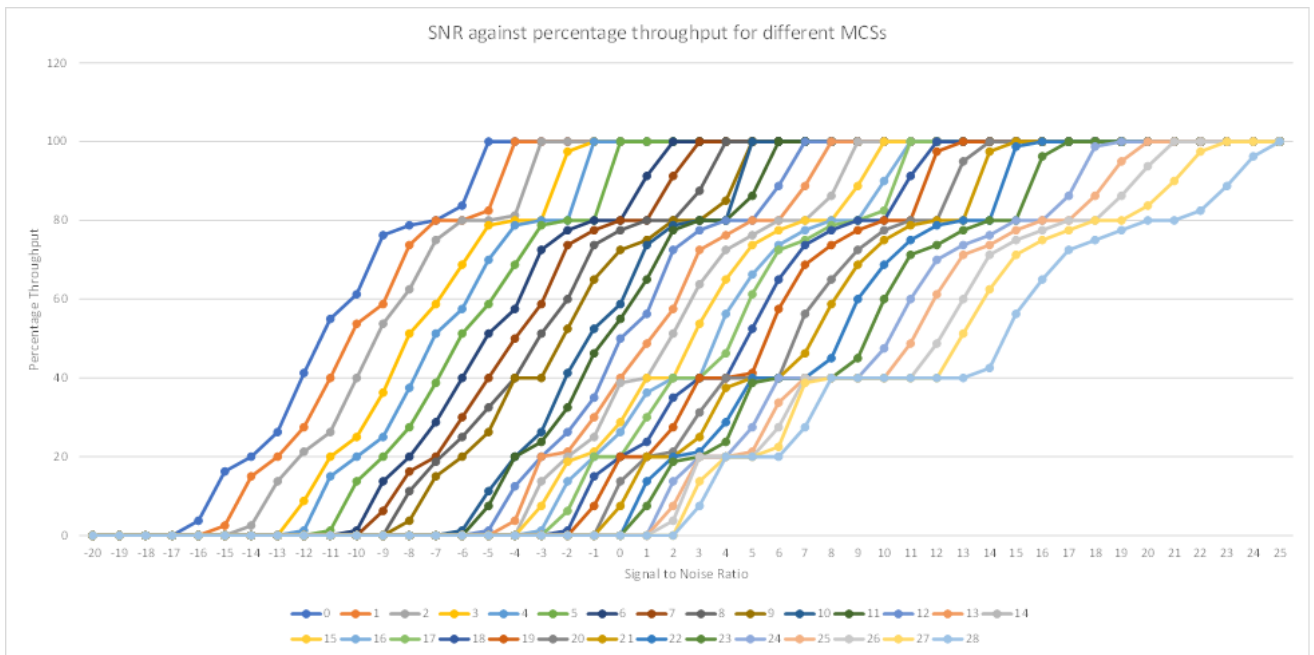


Figure 14. Graph of Block Error Probability against ESNR for different CQI

It can now be simply ascertained whether there has been a successful or an errored transport block during transmission using algorithm shown in Figure 15.

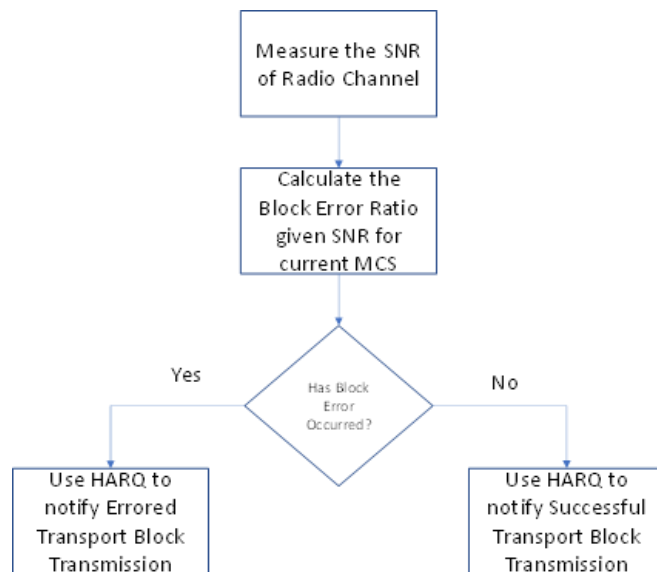


Figure 15. Algorithm for calculating Successful or Errored Block Transmission

3.4.3 Channel Traces from Digital Twin

Ray Tracing simulations of an ultra-wideband 190 GHz double-directional 3D (dual-polarized) transmitter – receiver in a digital twin conference room, whose propagation was characterized by measurements, was used to capture the vertical-vertical, vertical-horizontal, horizontal-vertical and vertical-vertical impulse responses at a grid of receiver positions from the transmitter [3].

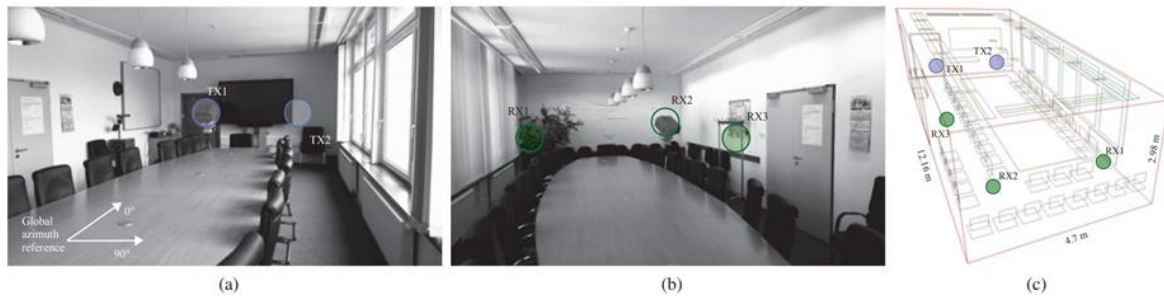


Figure 16. Pictures of the measured scenario: (a) view from RX1 towards TX1, (b) view from TX2 towards RX2, and (c) RT model.

Since the successive impulse response measurements are recorded on the grid, there is no time coherence between successive impulse responses, as required by Resource Block (RB) Channel Estimator, which interpolates between Reference Subcarriers in the vertical frequency direction for OFDM symbols 0, 4, 7, 11 and then interpolates in horizontal time direction for each 4G subframe or 5G slot as shown in Figure 16 to obtain the Frequency Response of the RB. The assumption is that there is high degree of time coherence with OFDM symbols 1, 2, 3, 5, 6, 8, 9, 10, 12, 13.

When the measured impulse response measurement for a single point of the grid is applied to all fourteen OFDM symbols in a RB then the equaliser works as expected as shown in Figure 17 and Figure 18. When the measured impulse response measurement for the next point of the grid is applied to all fourteen OFDM symbols in a RB then there would be no time coherence between RBs.

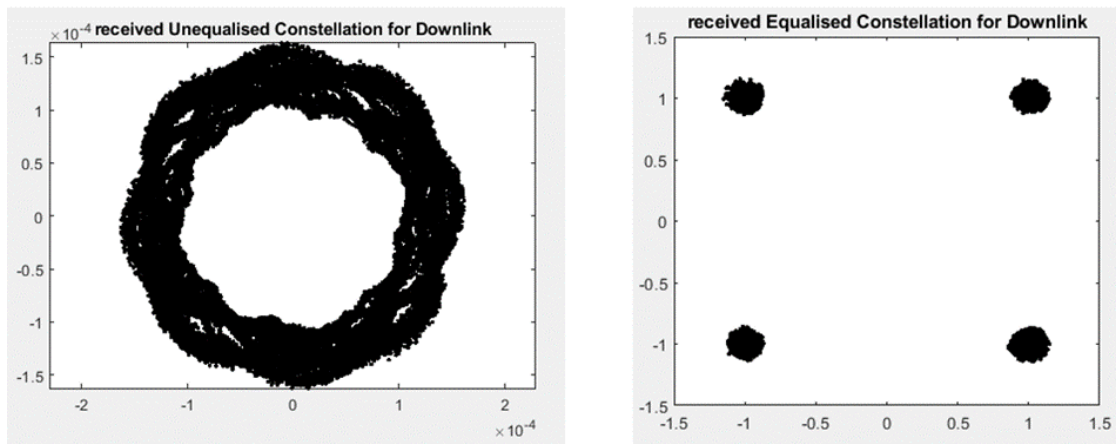


Figure 17. Received and Equalised 4-QAM constellations for SCS = 120 kHz, BW = 400MHz, QAM Order = 4-QAM, fc = 190GHz in conference room

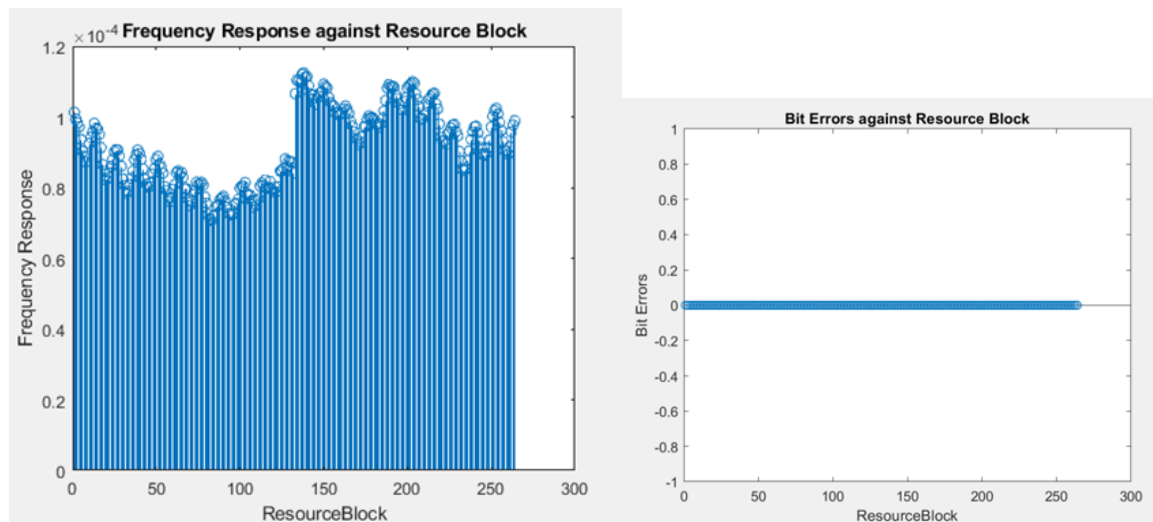


Figure 18. Received Frequency Response and BER for RBs of Equalised 4-QAM constellations for for SCS = 120 kHz, BW = 400MHz, QAM Order = 4-QAM, f_c = 190GHz in conference room

The conclusion is that if digital twin model is to be used in network simulations, then realistic model of user mobility is also required.

3.5 Reinforcement Learning for Joint Slice Scheduling in the RAN

In this section, we will first describe a practical RAN slicing scenario, model the problem, then propose a solution using Deep Reinforcement Learning (DRL). Our approach is to analyze a simple scenario to its fullest, addressing both its theoretical and practical concerns.

3.5.1 The Scenario

We hypothesize a scenario in which the owner of an amusement park, e.g., Disneyland, wants to have 4 RAN slices for 2 types of applications:

- 1 slice for 3 tracking drones within the coverage area
- 3 slices for various maintenance sensors of 3 games within the coverage area

They want to negotiate the Service-level Agreement (SLA), together with a pricing scheme, with an MVNO operating nearby. This is the public RAN extension of the OnSite5G-Outposts joint-service currently provided by Verizon and Amazon [4], in which the MVNO's key concern is whether they should admit these slices. Given that they already have profits with their general eMBB slice for 5G mobile subscribers, a degradation in this service may lead to a net profit decrease due to subscribers leaving the network. This leads to 3 questions:

1. How should the MVNO price the slices?
2. How to schedule these RAN slices to maximize expected profits?
3. How to handle "shifts" in both the underlying slice traffic and the changing slice requirements i.e., being zero-touch?

In this deliverable, we focus on answering Question-2, leaving Question-1 and Question-3 for future deliverables; this question can be (informally) stated as follows.

Problem 1 (Profit-optimal Slice Scheduling) *Schedule the PRBs between the UEs of different slices to maximize the expected profit of the MVNO.*

Firstly, we assume MVNO profits directly come from meeting the negotiated SLAs. Thus, in the modelling subsection-3.5.2, we will design an SLA Template and Profit Function that are commercially motivated.

Secondly, our key contribution in solving this problem does not lie in the solution itself, but in the realistic efficiency of the solution. That is, we will analytically stress-test the solution's theoretical basis in practical settings to improve its efficiency (subsection-3.5.3), then experimentally validate it against State-of-the-Art (SOTA) schedulers (subsection-3.5.4).

Finally, although it's tempting to optimize the profit performance over a time-horizon of 1 year (to capture all possible variations in this amusement park), we propose to only consider an H -hour time horizon during training and rely on a distribution-shift handler to update the schedulers until infinity. This is equivalent to collecting multiple H -hour trajectories (with different initial states) until distribution shift and update.

3.5.2 Problem Modelling

For system modelling, let us consider a RAN system with one Base Station (BS) of n78 carrier (3300-3800Mhz) and one Bandwidth Part. It is configured with 30kHz subcarrier spacing, DD-S-UU TDD format, and dynamic UL scheduling, as illustrated in Figure-19 and Figure-20. This corresponds to 273 PRBs and 0.5ms TTI. The reason for balancing between UL and DL is that

most vertical use cases asked for better UL QoS [5], which is also the case in our hypothetical scenario.

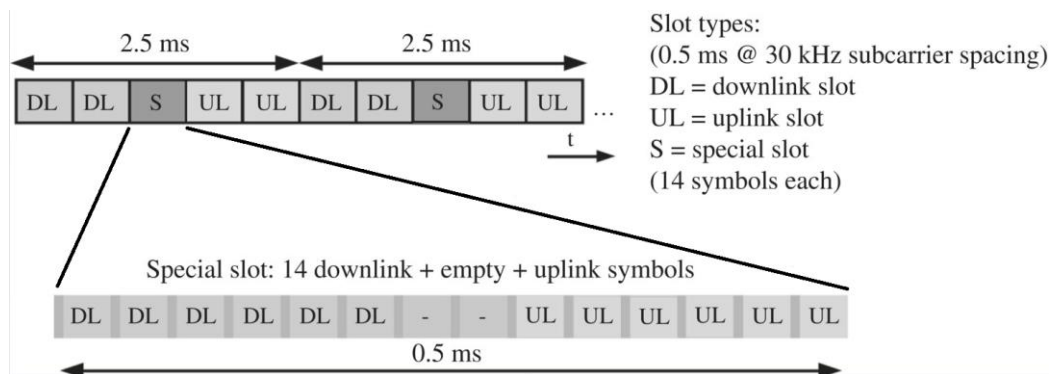


Figure 19. DD-S-UU TDD format in NR

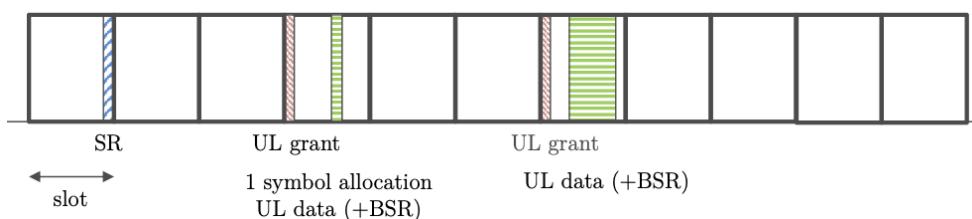


Figure 20. Grant-based Transmission in Dynamic UL

For objective modelling, 3 axioms and 2 properties were taken into account when designing our SLA Template and Profit Function. While the 3 axioms will stay relevant across deliverables, the 2 properties are subject to future changes to produce different reformulations of Problem-1 that are more commercially motivated.

Axiom-1. SLA should be expressed in terms of UE-level QoS because in the end, it's the UEs, not the slice, that experience the quality of the network.

Axiom-2. SLA should be expressed in terms of efficiency and fairness constraints, imposed on the QoS of each slice's UEs.

Axiom-3. SLA should be evaluated at per-frame granularity:

- Smaller than that, vertical customers are recommended to install a private network.
- Larger than that, vertical customers are recommended to use general 5G.

Property-1. Unless the MVNO wants a strict admission policy at the cost of losing multiplexing (monetary) gains, the imposed SLAs should be soft and not hard.

Property-2. Money earned and lost due to satisfactions of the SLAs represent the ultimate incentives of both parties.

Using these 5 pillars, let us consider several definitions that illustrate our SLA Template and Profit Function.

Definition 1 (UE-level Slice SLA) After each evaluation interval, **P%** UEs should have QoS, expressed in terms of throughput, latency, and reliability, being at least **X**.

Following this definition, let us denote the SLA of a slice as (P, X).

Sub-definition 2 (Satisfaction Function of a Slice) Consider a slice with SLA of (P, X) . At each evaluation interval, let $p\%$ be the ratio of UEs that have QoS being at least X , then the satisfaction function of that slice is:

- If $p \geq P\%$, the slice SLA is considered satisfied with SLA-defined monetary gains
- Otherwise, the slice SLA is considered violated and no money is earned.

We remark that in case of SLA violation, following the tradition of favoring the providers in cloud computing [6], we set the monetary gain to zero rather than negative.

Definition 3 (SLA-aware Profit Function). The Profit Function of a RAN system at time t is:

$$\text{profit}_t = -C + \sum_{i=1}^{\#\text{slices}} \text{satisfaction}\left(p_{i,t}, (P, X)_i\right);$$

with $(P, X)_i$ is the SLA of slice- i as defined in definition-1; $p_{i,t}$ is the ratio of UEs meeting QoS of slice- i at time- t as described in sub-definition-2; and **satisfaction(.)** is the satisfaction function defined in sub-definition 2.

3.5.3 Proposed Solution

Why Deep Model-free RL?

Firstly, RAN slice scheduling should be modelled as a sequential decision-making problem. This is because QoS is measured using the status of RLC queues and non-empty queues need to be serviced in the following timesteps. Such property eliminates the use of one-shot decision-making methods such as supervised learning or contextual bandits, when “best” instantaneous actions are provided or need to be explored, respectively.

Secondly, the rich and stochastic nature of the underlying traffic-channel variations discourage the use of rule-based and pure optimization methods. This is because rule-based methods assume optimality in explicitly describable rules, while pure optimization methods require the loss surface to stay the same while iterating on the control variables.

This left us with 3 options:

1. Predict several timesteps ahead, optimize the resource allocations then adapt (model-predictive control)
2. Adapting the resource allocations while learning the stochastic transition of the states (model-based RL)
3. Learn the resource allocations purely through reward signals (model-free RL)

We followed model-free RL simply because it is the most actively researched, allowing the reuse of ideas. Neural Networks (NN) is chosen for function approximation to meet the second challenge stated above. It is necessary to mention that there are other approaches such as game theory and genetic algorithms that are not yet explored due to time constraints.

Finally, our chosen Deep model-free RL approach (DRL) is locally optimal (local with regards to the explored trajectories assuming no performance collapse), conditioned on the approximating State-Action-Reward Markovian, which will be checked in subsection-3.5.3.

Designing State-Action-Reward and Assumptions

For the design of States, there are 2 non-trivial issues:

1. How should we abstract the packet-based workloads to align with the slice-based rewards without losing too much information?
2. How should we deal with the changing number of UEs per slice?

We propose to build states using the following processed statistics of the RLC queues, separately between the DL and the UL (known by the BS via Buffer Status Reports (BSRs)):

- Each state (both UL and DL) is a 2D image with $R \times C$ pixels
- R rows represent the slices ordered by their imposed “slice-level workloads”; C columns represent each slice’s UE-percentiles ordered by current “UE-level workloads”; these “workloads” are computed heuristically
- Each pixel represents the statistics of a UE with 3 channels: *Queue Size* (in bytes), *Maximum RLC Expiry* (in milliseconds), and *Wideband CQI*

Intuitively, on the one hand, we use UE-level abstraction so that the DRL agent can optimally control UE-level QoS that determines slice-level rewards (while avoiding control overhead and learning spurious correlations). On the other hand, we use order statistics to deal with the changing number of UEs (while introducing local semantics that resemble an image).

For the design of Actions, there are 2 issues that prevent us from doing typical joint-slicing (i.e., allocating PRBs directly to the UEs) with DRL:

1. 0.5ms is a very low decision-making time that no model serving can meet (yet). A workaround is to implement the NN in C, but this means saying no to model update since adapting a C-based NN over time is challenging
2. UL dynamic scheduling greatly complicates both the state and the action spaces of DRL-based joint-slicing

However, to satisfy the UE-level Slice SLA defined in definition-1, controlling the number of PRBs allocated to each UE every TTI is recommended. Thus, we propose the use of *Two-level Slicing with Configurable Packet Schedulers*. This way, our control variables are per-frame re-allocation of PRBs and re-configuration of packet schedulers to each slice, as follows:

- We assign PRBs to slices using the first output layer with Softmax activation and size equal to the number of slices.
- We configure the t_c parameter (throughput evaluation interval) of each slice’s proportional-fair scheduler using the second output layer with Sigmoid activation and size equal to the number of slices.

Intuitively, the first action allocates PRBs to the slices in proportion to their corresponding softmax ratios, the second action picks t_c for each slice in proportion (of 10ms) to each slice’s corresponding sigmoid ratio.

For the design of Rewards, it is simply the evaluated Profit Function (definition-3) after each scheduling frame.

With above designs, we can formulate our problem as follows:

Problem 1* (Profit-optimal Slice Scheduling) Finding optimal parameters ϑ of the policy π_{ϑ} under stochastic transition τ as

$$\operatorname{argmax}_{\vartheta} \mathbb{E}_{s_k \sim \tau(\cdot | s_{k-1}, a_{k-1}), a_k \sim \pi_{\vartheta}(\cdot | s_k)} \left[\sum_{t=1}^{\#frames} \text{profit}(s_t, a_t) \right]$$

$$s.t. \#PRB_t \leq \#PRB_{total};$$

with s_t and a_t are the observed state and taken action at frame- t ; **profit(.)** is the profit function defined in definition-3; $\#PRB_t$ and $\#PRB_{total}$ are the number of allocated PRBs at time- t and the total number of PRBs, respectively.

The sample efficiency of DRL in solving this problem depends on two “rules of thumbs”:

1. *Is our (State, Action) pairs closely approximate the modeled system? If so, are they Markovian?*
2. *How varied are the observed rewards for each (State, Action) pair? The less varied, the faster convergence.*

Per the first rule, we condition on the following two assumptions (with the latter being reasonable by [7] and [8]):

Assumption 1. *From “premature” current states, the agent can estimate the true current states (and act optimally) using the true reward signal.*

Assumption 2. *Traffic-Channel variations can be modeled with higher-order Discrete-time Markov Chains at per-frame granularity.*

Per the second rule, following the current status of OAI’s L2-SIM and [9], we condition on these assumptions (with the latter being reasonable in an urban setting).

Assumption 3. *All packets sent, both UL and DL, are observable using the RLC queues. All packet losses, both UL and DL, are due to deadline expiration and not bad channel quality.*

Assumption 4. *The channel coherence time of all UEs exceed 10ms.*

We note that the rationales behind these assumptions are:

- With assumption-1, we want to mitigate the delay difference between the time we sample our network state and the time we execute our scheduling action. With assumption-2, we want to ensure correctness in the policy training process. These can be satisfied with a suitable Neural Network design (discussed in subsection-3.5.3) and an appropriate state design like previously specified.
- With assumption-3, we want to mitigate the effect of unobservable rewards due to the limited capability of OAI’s L2-SIM. With assumption-4, we want to ensure action consistencies between similar configurations of the proportional-fair scheduler.

We further note that assumption-2 and assumption-3 corresponds to modelling errors and inherent noises, respectively. All can be further mitigated with probabilistic training techniques discussed in future deliverables. Therefore, we only have assumption-4 remains.

Designing Neural Network and Algorithm Selection

Following our assumptions, the approximating MDP of Problem-1* is a DT-POMDP that is observable with partial history. This means the policy has to be derived using the approximating “beliefs” of the underlying true states. We propose to do this with local-featured memory-full function approximators i.e., Convolutional-Recurrent Neural Networks. Intuitively, we’re trying to learn the temporal dependencies of the extracted local semantics between the workload-percentiles inter-slice and intra-slice, to decide PRBs assignments in terms of quantity and fairness, respectively.

To deal with the difference between sampled states and before-action states, we add 1 fully-connected layer after the CNN layers to linearly pre-process their kernels’ outputs. To deal with the possibly non-linear relationship between the final spatial-temporal features and the Values and Optimal Actions of the underlying “true” states, we also add 2 more fully-connected layers after the RNN layers.

For training algorithm, we propose the use of Deep Deterministic Policy Gradient (DDPG) due to our deterministic continuous action and DDPG being balanced between simplicity and sample efficiency. For exploration during training, we will use fixed scale independent Gaussian noise.

3.5.4 Experimental Considerations

For our RAN slice scheduler to be industrially ready, it needs to be realistically validated against promising existed RAN slice schedulers. To set up realistic experiments, we propose to build upon the traffic trace described in subsection 3.4.1 with two enhancements:

1. Incorporate realistic distribution shifts and channel variations
2. Replay this new traffic-channel trace on a real testbed, such as OAI’s L2-SIM

For 1), we need to integrate a suitable long-term traffic trace and channel simulator into the original trace with appropriate Time Series techniques. For 2), we need to develop a real-time cognitive plane that talks to the OAI environment. Both are non-trivial tasks that needs further investigation and will be detailed in future deliverables.

For the baseline RAN slice schedulers, we propose two candidates:

- EDF [10] – A simple yet very robust rule-based RAN slice scheduler
- OnSlicing [11] – A state-of-the-art ML-based E2E slicing algorithm that is very efficient yet rather complicated; we propose to use its RAN slicing adapted version

We consider our experiment to be successful if we can develop a cognitive plane that is both simpler than EDF and has the multi-resource slicing capability of OnSlicing, but is more efficient than both EDF and OnSlicing on the proposed real-time test-bed.

4 Backbone/Core Network Slice Control Enablers

Considering the overall E2E network slicing vision in the previous sections, this section further presents the 6G-BRAINS network slice control enablers for backbone network, including edge, core and transport segment. The aim is to investigate different hardware- and software-based enablers, complimentary to hybrid approaches (hardware and software) to provide network slicing in the context of the 6G-BRAINS framework.

4.1 Overview of backbone segments network slicing

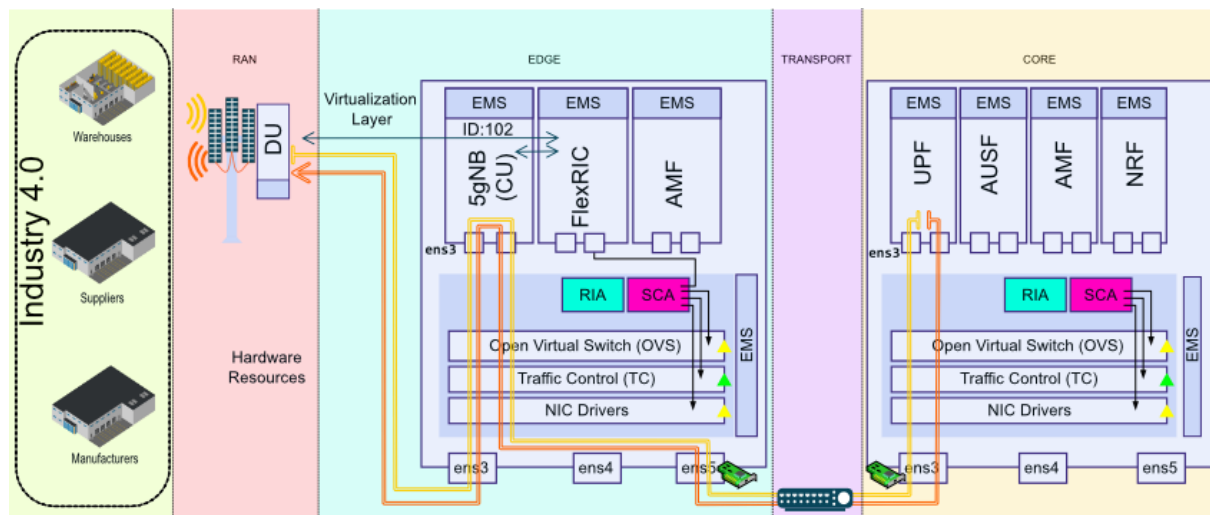


Figure 21. Overall Vision

The technical development towards 6G to update current 5G use cases (eMBB, URLLC and mMTC) are presented in this overall vision of the architecture. It is expected that in the incoming generation of mobile networks, new use cases that require extreme performance arise, as well as new combinations of requirements that are not contemplated in current 5G categories [12]. As can be seen in Figure 21, these 5G RANs consist of different antennas and DUs connected to Edge segment of the network, where CUs are allocated. The CUs bring network services, using NFV technologies, to the last-mile of the network providing better QoE to final users. These CUs also connect the network data plane between the RAN and the Core segment of the network. As depicted, the Edge segment is formed by a physical machine which contains three different NFVs, which act as isolated virtual network operators, sharing the same physical resources.

The Core segment of the network, also shows a physical machine that contains all the different NFVs required by three independent network operators sharing the same physical infrastructure. In this network segment, traffic is received from the different CUs and it is processed in the UPF, which acts as a mobility anchor for the traffic of the 6G users and also relay the data to internet. The AMF/SMF are used to allocate the IP address and to manage the user sessions on the network. Finally, the UDM/AUSF are mainly in charge of the user registration, access authorisation, maintaining the network profiles and also, of authenticating servers and proving encryption keys.

Hybrid software- hardware-based solution allocated in the edge, transport and core segment of the 6G architecture provide network slicing capabilities and complex data path which allows the classification and prioritization of overlay network traffic. In this context, Traffic Control

(TC), and OVS present network slicing capabilities in the kernel segment of the system, while SmartNICs, such as NetFPGA and Netronome, provide both network acceleration and network slicing in the hardware side.

4.2 Hardware-based network slicing

Hardware-based data plane programmability is explored in the vision of 6G-BRAINS project as a network slicing enabler based on Simple Sume Swith model [13] for P4-NetFPGA. These approaches are investigated and technically integrated with the NFV components of the architecture proposed in order to provide E2E, guaranteeing high reliability and low latency.

4.2.1 NetFPGA Platform

Figure 22 depicts an architectural design overview of the different components of the FPGA card and the main contributions of this research to provide a holistic and general idea of the components involved in the framework and the innovations associated. The architectural design presented in Figure 22 is divided into two different planes: Control plane and Data plane. The control plane has been prototyped for the Linux kernel, where a novel UWS Control API has been implemented to allow network administrators to configure the network control rules of the traffic processed by the NetFPGA card and therefore, to dynamically control the network slicing policies. The data plane represents the segment of the architecture implemented directly on the NetFPGA hardware card. The control and data communication channels have been represented in orange and blue respectively. These independent channels allow the exchange of data and control traffic between the card and the operating system of the computer by using a logical division of the PCI slot.

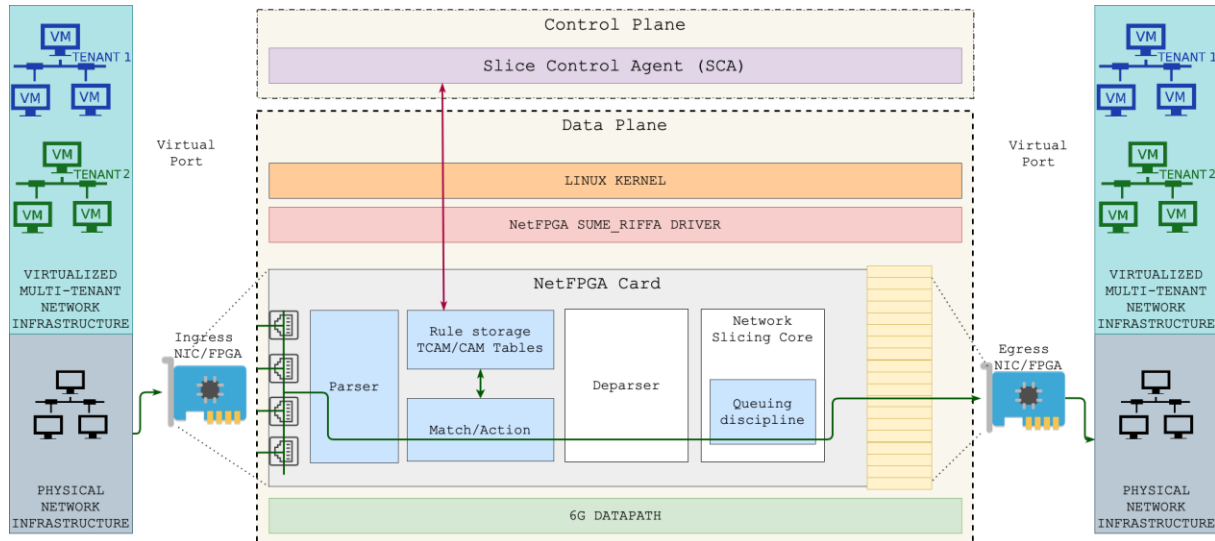


Figure 22. NetFPGA data plane and network slicing implementation

The control plane is based on two different architectural components, UWS Control API and the NetFPGA SUME_RIFFA driver [14]. The first is a NBI that has been designed to provide access to the novel control actions exposed by the NetFPGA. Aside from the control task, this REST API also produces monitoring information messages about traffic statistics, providing a comprehensive message format and simplifying the control of the network rules which are stored inside the hardware in TCAM, CAM and registers. Control plane also utilises the NetFPGA SUME_RIFFA driver, which is the driver implemented with the P4-NetFPGA project, providing support for four logical network ports detected in the linux kernel (*nf0*, *nf1*, *nf2* and

nf3) allowing communication between the NetFPGA card and the kernel of the operating system using the DMA interface.

In the data plane, the NetFPGA card is based on the open source P4-NetFPGA project [13] which provides a fully controllable data plane with P4 language support. This data plane consists of four different physical architectural components, divided into different wired FPGA modules connected and controlled through the operations exposed via NetFPGA Control API. The P4-NetFPGA project provides a pipeline with three different stages (parser, match/action and deparser), which have been significantly extended in this research to provide support for overlay networks. Additionally, a novel FPGA module has been designed, implemented and integrated in this P4 pipeline to provide network slicing support. The P4 language is used to perform the programmability of the data path.

- **Parser:** Refers to the first NetFPGA stage, where network packets are received either from the physical ethernet network interfaces or from the PCI bus into the P4 pipeline. All different headers that need to be parsed require to be defined in this stage, indicating in detail all the different fields of the packet headers. In this stage, all headers of the network packets are classified and their values are extracted to use them in following stages.
- **Match/Action:** Once the parsing is completed and the packet has been classified properly, this goes into the P4 pipeline Match/Action stage. This stage utilises the data extracted in the previous stage to access the entries of the control tables where rules are stored in order to determine if there is a match in the table. If so, the entry in the table has an action associated which is performed over such package. This is the only stage of the P4 pipeline that has access to the TCAM/CAM tables.
- **TCAM/CAM Tables:** Memory structures where network control rules are stored and where the management of rules is undertaken. These memories are based in a key/action model, where the fields specified in the Match/Action stage are considered as keys, and if they match with the values of a rule inserted in the table, then an action is activated. CAM is based on an exact matching of the fields whereas TCAM uses ternary matching values allowing the usage of wildcards, to determine if an action is activated: 0, 1 or irrelevant. The set of actions is dependent on the purpose of the NIC, e.g., a drop action will allow firewalling capabilities, a change IP Source action will allow NAT-ing, and so on.
- **Deparser:** This is the last stage of the P4 pipeline. At this point if the packet has not been dropped, it is reconstructed with the information that may have been modified during the previous stages and is forwarded to the network slicing module.
- **Network Slicing Module:** This stage is located immediately after the P4 deparser stage. This network slicing module can also be referred to as the network slicing core. An internal core or module of the FPGA can be defined as a reusable unit of memory of design logic with defined behaviour. Every non-dropped packet goes through the Network Slicing Module. In this module, an advanced implementation of a double level of queues has been identified as an approach to provide network slicing support to the NetFPGA card. Each queue has a priority associated and the priority of every packet will be decided by the queue where such package is sent. This priority is defined by the values specified in the rules inserted in TCAM tables of the NetFPGA and it is applied to the outbound traffic. Once the packet is removed from the queue, it is sent to the kernel of the operating system through the DMA.

A basic solution of this approach was previously implemented in the H2020 EU project Slicenet, however in 6G-BRAINS an extended version of this solution is presented. This approach presents a larger parser and match/action stages, providing support for the use cases covered in this project. In addition, the NetFPGA API has been extended to provide support for new encapsulations protocols, such as EtherCAT required these novel uses cases. The set actions supported by this network slicing enabler for 6G-BRAINS project also been extended to support the copy of the QoS information between the different encapsulations of the overlay network traffic.

4.2.2 NetFPGA Network Slicing Module

Figure 23 shows a basic NetFPGA approach to achieve network slicing based on the isolation of the network flows for 5G networks. This network slicing implementation has been carried out by extending the basic NetFPGA data path, composed by parser, match/action and deparser stage. As can be seen, this implementation consists of 32 priority queues which are connected to the deparser stage through a demultiplexer. At the same time, the queues are connected to a deficit round robin arbiter, through a multiplexer, which forwards the network traffic to the DMA of the computer where the NetFPGA is located. The priority queue used by a packet is picked in the match/action stage via P4. For this purpose, the default implementation of the NetFPGA P4 module has also been changed in order to allow the user to select the priority queue using the P4 language.

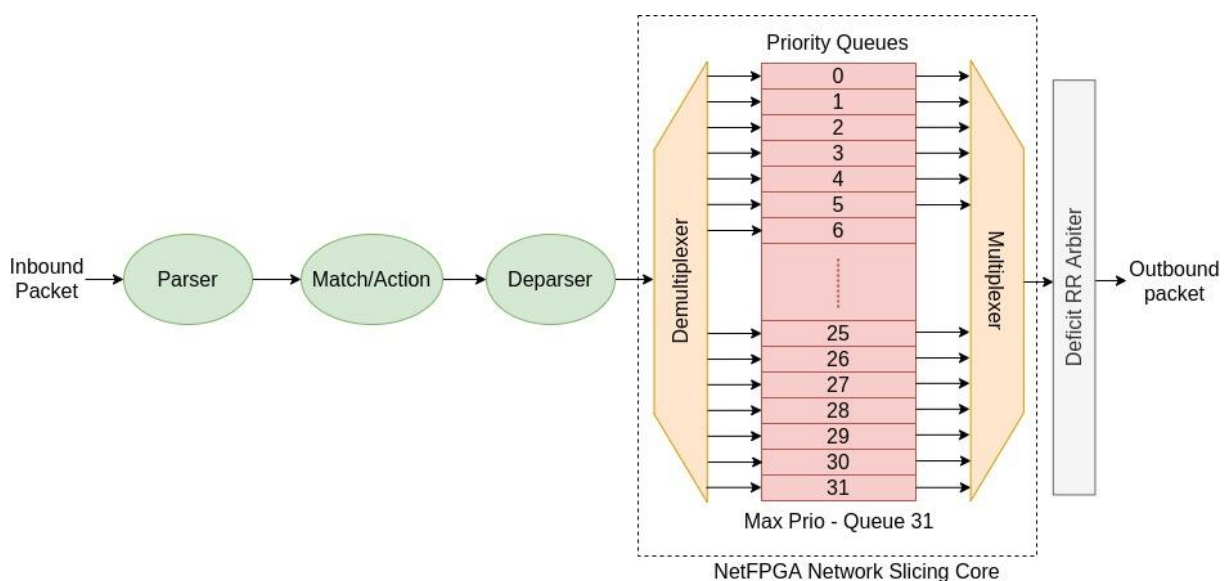


Figure 23. Basic NetFPGA Network Slicing Module implementation

Although the network slicing implementation shown in Figure 23 demonstrates feasibility of the approach by providing real hardware-based isolation, it presents several problems. Firstly, with regard to the data path implementation, the previous solution does not provide multi-tenancy support for virtualised 5G networks. Secondly, the implementation of the NetFPGA slicing module undertaken can cause failure of packet processing in the less priority queues. This is because, while high priority queues are processing packets, the less priority queues are waiting. And the third problem found is that if the same priority queue is assigned to different users and if such queue is suffering congestion, the traffic of the last senders will have to wait until the queue releases the previous packets received from other users.

In order to resolve the problems presented by the first NetFPGA-based network slicing approach, the implementation shown in Figure 24 has been carried out. This prototype consists of two nested levels of queues connected on one side to the deparser stage of the NetFPGA data path and on the other side with a deficit round robin arbiter which forwards the traffic to the DMA. As can be seen, the first level of queues consists of 8 round robin queues, each connected to 4 priority queues, where the queue 0 presents the lowest priority and queue 3 the highest. In summary, based on the implementation shown, this solution presented provides hardware-based data plane which allows network slicing, guaranteeing an E2E isolation of the network flows. This isolated channel can be created by user or tenant and user, among others, depending on the final use case of the application. This implementation is fully functional with 5G/6G networks with overlay networks support.

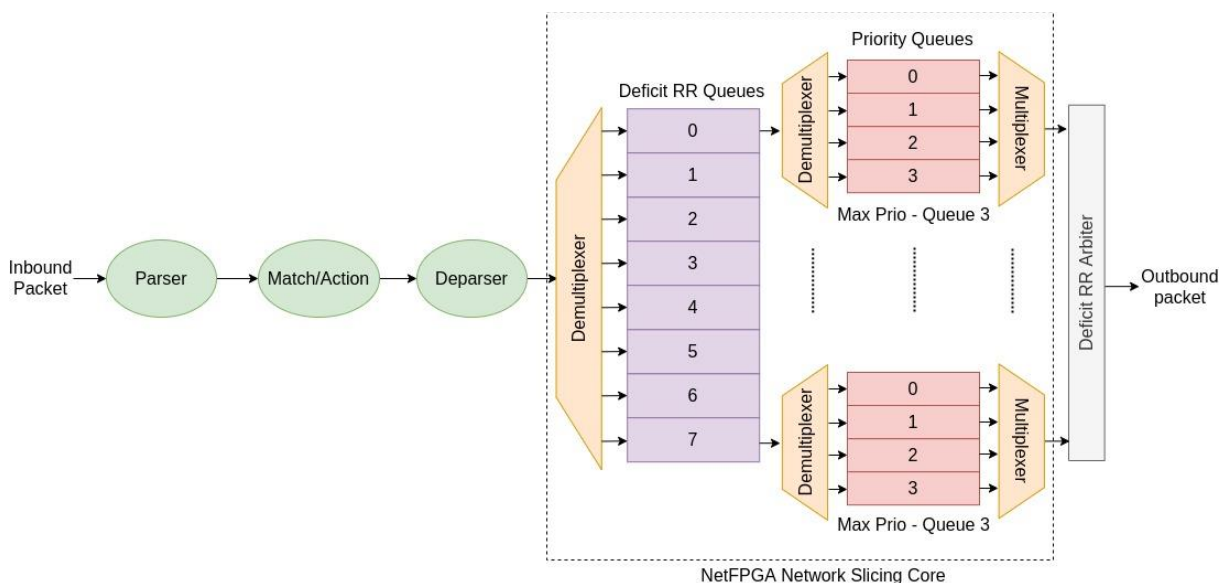


Figure 24. Advanced NetFPGA Network Slicing Module implementation

The pseudo-code shown in Figure 25 presents the algorithm implemented in the advanced network slicing module depicted in Figure 24. In this algorithm, the incoming packets from the physical ports are processed and sent through the DMA channel to the Central Processing Unit (CPU) of the computer, and vice versa. For the packets received from the physical ports, extra treatment is applied. In this case, if there is not a hit the TCAM table, the default action is applied. This default action uses the last three bits of the innermost source IP address to set the value of the output round-robin queues and the two bits 2-1 of the DSCP field of the IP header to set the value of the output priority queues. However, if there is a hit with a rule of the TCAM table, the values of the output priority and round-robin and extracted from the rule matched. At the production stage, this can be managed by the internal Dynamic Host Configuration Protocol (DHCP) management to smartly assign IP addresses based on the subscription type of the user and therefore, automatically prioritise the traffic of the users based on the IP addresses.

```

if FPGA.srcPort == physicalPort then
  if table.hit then
    PRIO=actionRule.prio
    RR=actionRule.rr
  else
    PRIO = packet.innerIPheader.DS(2:1)
    RR = packet.innerIPheader.srcAddr(2:0)
  end if
  FPGA.dstPort = DMA;
else
  if FPGA.srcPort == DMA then
    FPGA.dstPort = physicalPort
  end if
end if

```

Figure 25. Match/Action pseudo-algorithm implementation for the management of the P4-NetFPGA-based queues developed

4.2.3 NetFPGA API

This REST API is used for the communication between upper layers of the architecture and the hardware devices through the network driver. This section provides information about the different operations and actions exposed by this REST API and the structure of the messages exchanged. Table 5 shows the parameters required to add a new rule to a table implemented in hardware. The original 4-tuple (Source IP, Destination IP, Source Port, Destination Port) has been significantly extended to support 5G/6G scenarios and virtualised overlay networks with multi-tenancy. The keymask0 represents the Ethernet Type field and specifies the link layer header, such as 0x88A4 for EtherCAT protocol. The encapsulation headers required by these overlay scenarios, explained in previous sections, have been supported by the operations provided by this API. The implementation of this REST API presents support for several encapsulation headers in order to provide a generic northbound independent of the use case carried out. Table 6 describes the different encapsulation protocols that can be defined within the fields keymask9, keymask10 and keymask11 which must be supported by this northbound interface. Also, the values shown in this table represent the number used by the API to identify the different encapsulation protocols. The northbound interface refers to the Policy Charging Function (PCF) or control plane presented in Figure 1. This northbound interface allows the transmission of control traffic between the administrator of the network and the hardware device.

Table 5. Parameters required to add a new rule to a table implemented in hardware

Field Name	Description
Table_name	Name of the table
Address	Position in the table
keymask0	Ethernet Type

Keymask1	Inner/Overlay source IP
Keymask2	Inner/Overlay destination IP
Keymask3	Inner/Overlay source port
Keymask4	Inner/Overlay destination port
Keymask5	Flow Layer. It indicates the number of encapsulations of a packet
Keymask6	ID of the first encapsulation header
Keymask7	ID of the second encapsulation header
Keymask8	ID of the third encapsulation header
Keymask9	Protocol of the first encapsulation header
Keymask10	Protocol of the second encapsulation header
Keymask11	Protocol of the third encapsulation header
Param1	First action parameter
Param2	Second action parameter
queue	Priority queue
action	Action applied when a packet matches a rule

Table 6. Values used in fields keymask9, keymask10 and keymask11 to identify different encapsulation protocols

Encapsulation protocol	Value
VXLAN	1
GTP	2
GRE	3
RTP	4
MPLS	5
VLAN	6
R-GOOSE	7

EtherCAT

8

Notice that, the proposed design is radically new with regards to traditional rules. It has a significantly enhanced flexibility, allowing the type of encapsulation protocol to be matched and defined as parameters which leads to several rule configuration possibilities. Thus, it has been decided to use ternary tables for this implementation. All the parameters described in Table 5 as keymaskX must contain the value of the field and the mask associated with this value. This allows the provision of a wider range of possibilities when configuring the traffic filtering by the rules.

The field queue is the method used to control the network slicing innovation implemented, allowing the user to configure the hardware network slicing module which permits the prioritisation of the network traffic. Depending on the implementation, this module can be configured with different numbers and levels of queues in order to guarantee E2E network slicing and also a fair traffic distribution between the different queues. This fair distribution of the traffic is achieved by using the network slicing algorithm implemented in the NetFPGA. The algorithm contains a set of round-robin queues which allows an equal distribution of the low priority and high priority traffic between all the output priority queues. The field action is probably one of the most important of the add entry operation. This field allows the selection of the action which is applied over a network packet when it matches a rule inserted in the table. Table 7 shows the different actions exposed by this implementation. As can be seen, each action can require up to three different parameters which indicate: the information of the packet that is going to be overwritten (parameter 1); the outbound interface where the packet will be sent (parameter 2); or the queue where the packet will be sent (queue).

Table 7. Values required to invoke the different actions provided

Action	Value	Parameter 1	Parameter 2	Queue
DROP	1	-	Outer Interface	-
NOPE	2	-	Outer Interface	Queue ID
L2 Outer redirect	3	Destination MAC (outer)	Outer Interface	Queue ID
L3 Outer redirect	4	Destination IP (outer)	Outer Interface	Queue ID
L2 Inner redirect	5	Destination MAC (inner)	Outer Interface	Queue ID
L3 Inner redirect	6	Destination IP (inner)	Outer Interface	Queue ID
Set Outer TOS	7	TOS (outer)	Outer Interface	Queue ID
Set Inner TOS	8	TOS (inner)	Outer Interface	Queue ID
Copy Outer to Layer TOS	9	Layer	Outer Interface	Queue ID
Copy Layer to Outer TOS	10	Layer	Outer Interface	Queue ID

The field action is probably one of the most important of the add entry operation. This field allows the selection of the action which is applied over a network packet when it matches a rule inserted in the table. Table 7 shows the different actions exposed by this implementation. As can be seen, each action can require up to three different parameters which indicate: the information of the packet that is going to be overwritten (parameter 1); the outbound interface where the packet will be sent (parameter 2); or the queue where the packet will be sent (queue). Code below shows an URL example used to invoke add entry:

```
http://192.168.1.15:5353/add?table_name=route_table&address=1&
keymask1=0xc0bc008c/0xffffffff&keymask2=0xc0bc0002/0xffffffff&
keymask3=0x138c/0xffff&keymask4=0x138c/0xffff&keymask5=0b011/0b111&
keymask6=0x0445/0xffff&keymask7=0xd0d4/0xffff&keymask8=0x0060/0xffff&
keymask9=0b001/0b111&keymask10=010/0b111&keymask11=100/0b111&
param1=01010101010101010101010101010101010101010101010101010101&
param2=01&queue=100&action=0010
```

4.2.4 Empirical Results

This section depicts the empirical validation of the NetFPGA-based framework described above. These charts present statistics about delay, jitter, and packet loss statistics when the NetFPGA is applying different QoS configurations to the traffic stemming from up to 512 users. Table 8 shows the settings of the number of traffic senders, the quantity of data sent in each slice, and more information about the traffic. The bars presented in each graph are the average of all the packets that are involved in each of the flows used in the experiment (per service). In fact, the experiment has been carried out five times and the values are the average of those five executions.

Table 8. Pcaps modified to test the slicing in the experimental setup proposed

Slice	IPs (Users)	Flows/Services	#packets	packet size	pcap size (Gb)
1	32	512	477696	1500	5.73
2	64	1024	955392	1500	11.46
4	128	2048	1910784	1500	22.93
8	256	4096	3821568	1500	45.86
16	512	8192	7643136	1500	91.72

Figure 26 shows the delay achieved when network slicing is applied to the traffic transmitted by up to 512 users. The delay specifies how long it takes for a packet to travel across the network from one node to another and to return back. The number of users sending traffic in each slice is always 32. Thus, 32 users are sending traffic when only one flexible slice instance is configured and 512 users when 16 slices are configured (32 users in each slice). The graph

shows an exponential trend in the number of slices. It is clear that, overall, the delays measured in the five scenarios are fairly similar, always around 2 ms, in the queues that have been assigned the lowest priority (queue 0), associated to best-effort traffic. Nevertheless, the delays are reduced dramatically when the traffic is processed by queues that have been assigned higher priorities. With the increase of the priority, delays decrease. The importance is that different scenarios with different slices show exactly the same performance in terms of delay. It can be observed that packets processed by the queues 1, 2, and 3 have a maximum delay of 0.25 ms; in the same manner, the delays reached by queue 3 are always lower than 0.1 ms. The chart represents the standard deviations of the delay in each case between all the packets submitted; the values are not, in any case, higher than 0.1 ms. Overall, these comparable delay results across different cases reveal high scalability of the proposed scheme in terms of the number of users per slice.

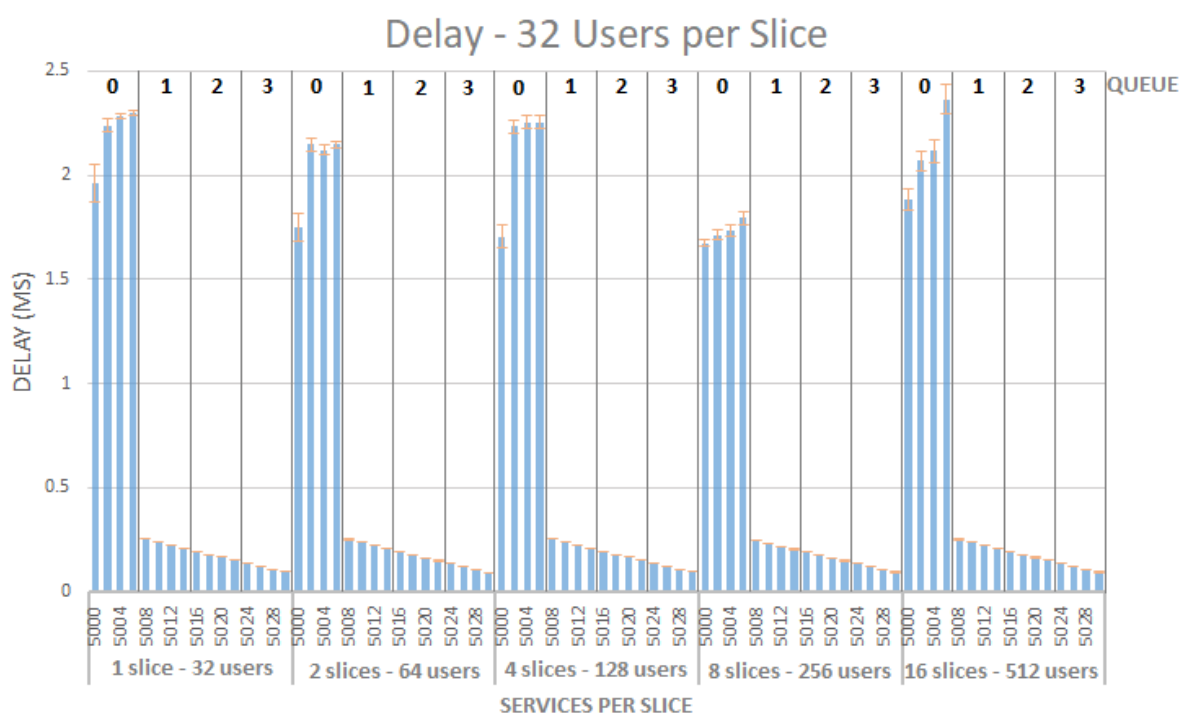


Figure 26. Delay achieved when the number of flexible slice instances is increased

The bar chart shown in Figure 27 presents the number of packet loss in each queue when the NetFPGA is working in a congestion scenario. Packet loss refers to packets that fail to reach their destination. Queue 0 has the worst values concerning delay, and it also has the highest rate of packet loss. These packets are discarded in the NetFPGA RIFFA driver when it is working under a stressful circumstance. Again, as can be observed, the solution proposed scales well, as the packet loss rate when only 32 users are sending traffic is quite similar to that when 512 users are transmitting. It is clear that, in queue 0 (in the service with port 5000), almost 100% of packets are discarded when they are congested in the network, mainly for the performance of the software stack. However, the traffic with a higher QoS, transmitted by the others queues, is arriving with less than 1% of packets loss in queues 1 and 2 and with 0% of packet loss in queue 3, which has the maximum priority.

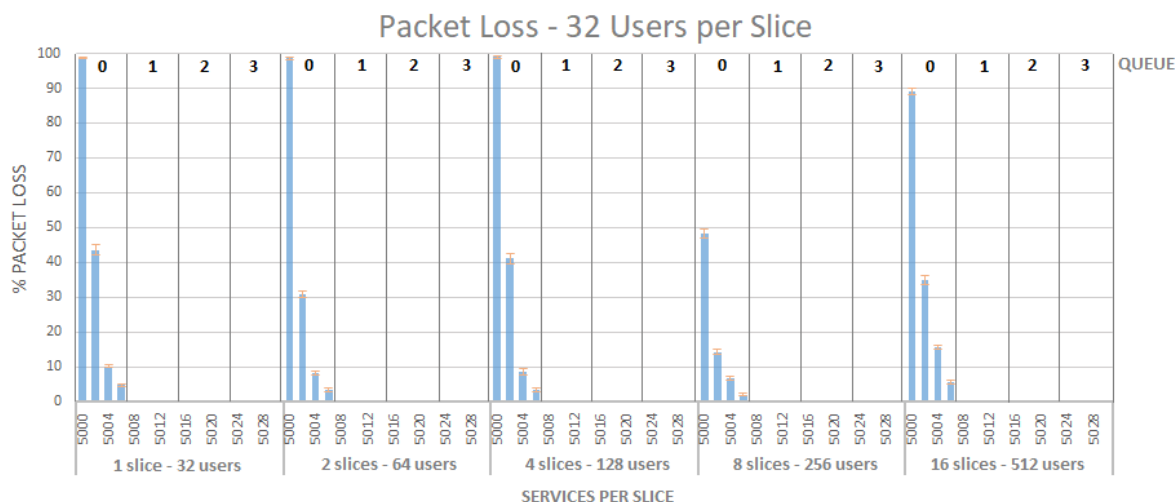


Figure 27. Packet loss achieved when the number of flexible slice instances is increased

Figure 28 represents the jitter results in the communication between the NIC that is sending the traffic and the NetFPGA, which is receiving it. The jitter describes the variation in the delay of received packets. Although the jitter increases when there is congestion in the NetFPGA card, the graph shows high stability in the jitter performance for those services that are assigned to queues different to queue 0, which are those services sensitive to low latency and, thus, also to jitter variations. Again, notice how Figure 28 demonstrates the same performance between different network slices.

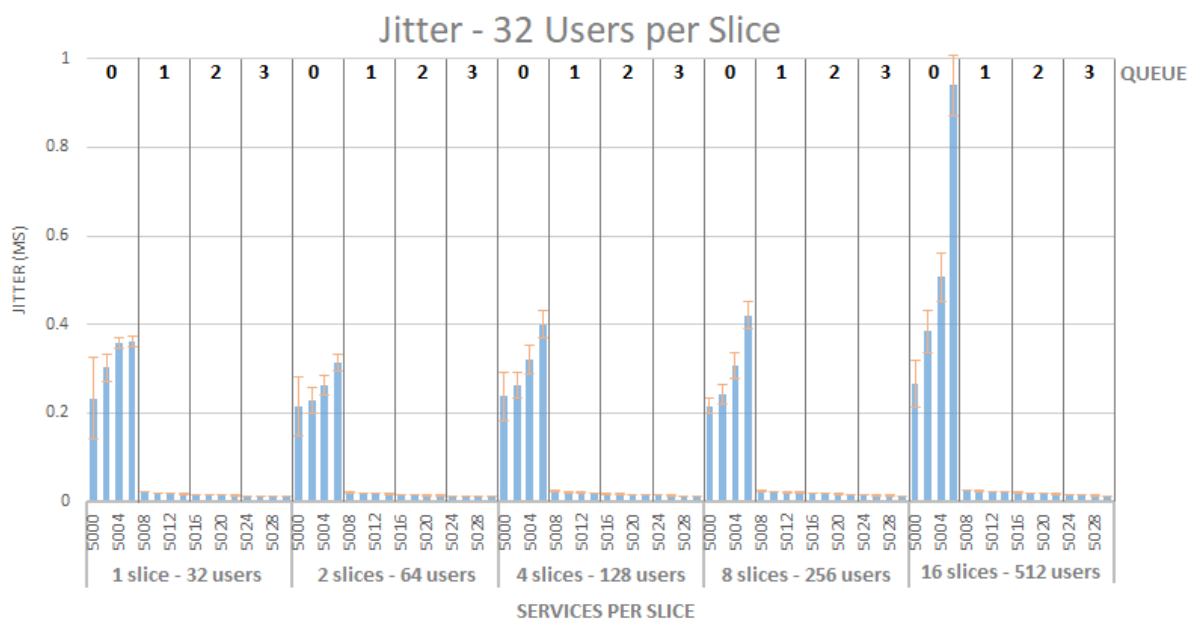


Figure 28. Jitter achieved when the number of flexible slice instances is increased

Table 9 shows the maximum values achieved by such slices which can be used as performance indicators by the solution proposed in terms of delay, packet loss and jitter. Notice that the values presented have been obtained in a scenario under congestion, so this also demonstrates the scalability of the implementation proposed. When modifying the number of users, rules and slices defined in the different testbeds analysed, a congestion has been

detected when the speed transmission exceeds 1.5 Gbps, mainly due to a lack of performance in the Linux driver. Also, the Linux kernel is not able to provide faster transmission than approximately 3 Gbps which is the reason why all tests have been executed with a maximum of 3 Gbps. This throughput refers to the transmission speed achieved in the E2E communication between the two different computers of the testbed, which represent the edge and core segments of the 5G architecture. As can be seen, the guaranteed maximum latency when queue 3 is being used is 0.13ms, with less than 0.2% packet loss and a maximum jitter value of 0.014ms.

Table 9. Maximum value of delay, packet loss and jitter guaranteed in a congested scenario, where 512 users are transmitting traffic

Queue	Delay (ms)	Packet Loss (%)	Jitter (%)
0	2.3624	88.97	0.94003
1	0.25178	0.19632	0.02468
2	0.20891	0.19601	0.01938
3	0.13805	0.19552	0.01487

4.3 Software-based network slicing

4.3.1 Traffic Control (TC)

Empowering the network system with E2E slicing capabilities implies being prepared for the heterogeneity that may exist among the different network devices distributed through a 6G architecture. Section 4.2 presents two different hardware-based approaches to achieve high performance with respect network slicing on the Backbone and Core Network segments. However, it would not be realistic to assume that the whole infrastructure will be commissioned with hardware-based network programmable devices (smartNics), especially in software-based architectures that leverage virtualised resources to perform Virtual Network Functions (VNFs). To cover the wide spectrum of network devices and system requirements that can be found along the E2E network slice data path segments, the 6G-BRAINS framework considers hybrid enablers (hardware and software) to allow flexible network slice definitions. With respect to software approaches, Linux-based systems offer a very rich set of tools for managing and manipulating the transmission of network packets traversing any given network interface. The Linux kernel packet scheduler is part of the Linux Kernel's network stack and manages the transmit and receive ring buffers of all network interfaces. Traffic Control (TC) [15] is the user-space utility program used to configure the Linux kernel packet scheduler. TC represents the sets of queuing systems and mechanisms by which packets are received and transmitted on a network device system. This includes configuring which (and whether) network traffic flows to accept on the input of an interface and determining how to transmit them (order, rate, bandwidth limitations, etc.) on the output of an interface. Traffic shaping smooths the bandwidth requirements of traffic flows by delaying transmission packets when they are queued in bursts. The scheduler decides the timing for the transmitted packets. Quality of Service (QoS) capabilities are used to prioritise traffic based on service class. 6G BRAINS envisions to make use of the Linux network scheduler

to create data-plane slicing when the system resource is not equipped with higher performance programmable data-plane network devices such as those based on hardware acceleration or kernel bypass techniques, or in those scenarios where the link (L2) and network (L3) layers of a resource require to work together. Therefore, the Linux Network scheduler with advanced queuing disciplines and dedicated filtering expressions (covering 6G network traffic) is used to optimize and guarantee performance, improve latency, and increase usable bandwidth for some kinds of services with specific network requirements.

4.3.2 Architectural design

Figure 29 overviews the Linux Network Scheduler architecture for both senses, ingress and egress. The correlation between traffic control elements and Linux components includes shaping elements (Linux classes), scheduling elements (Linux qdisc), classifying elements (Linux filters to perform the classification), and Policing elements (Linux policer that is part of the Linux filter). As depicted in Figure 29, the Slice Control Agent (SCA) uses the user-space application TC to configure these elements in a convenient way (depending on the use case) to create a dedicated queuing schema with specific requirements. After the configuration process, the SCA builds the filter that contains the classification predicate plus the policy and attaches it to the TC system so packets selected (matching) the classification algorithm will follow the same policy.

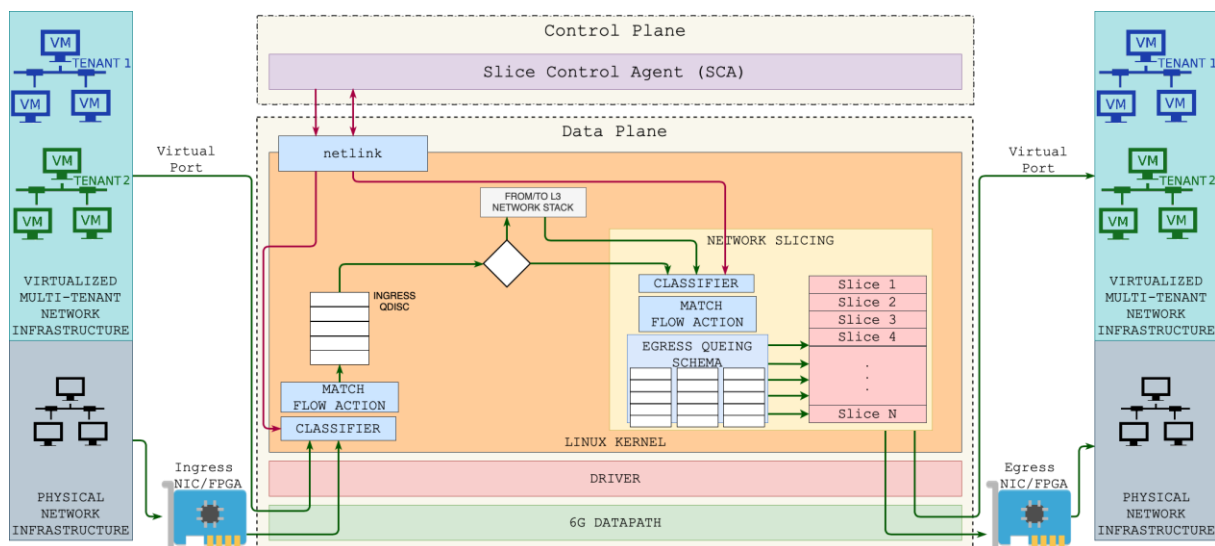


Figure 29. Architecture of Traffic Control with network slicing support

4.3.3 Queuing schema and filtering expressions

The queuing schema, composed by different hierarchical levels of queuing disciplines (qdisc) and class elements may vary drastically depending on the slice definition and requirements on the data plane, while on the other hand, the filter (classifier + policy) provides a convenient mechanism for gluing together several of the key elements of traffic control and is the most complex component in the Linux traffic control system. The packet structure requirements imposed by 6G visualised infrastructures adds an extra level of complexity making it impossible to leverage built-in classifiers to select packets based on their attributes. Nevertheless, the 6G BRAINS framework is equipped with the SCA that uses a data-plane abstraction service to hide the complexity to upper layers of the architecture to create queuing schemes and advanced filtering expressions while using a dedicated data-path agent

in its south-bound interface to handle technology dependent logic. Figure 30 illustrates an example of data-plane network slicing that assigns different Quality of Service (QoS) specifications defined in the queuing schema. To be precise, the scenario presented in Figure 30 configures a hierarchical level of scheduling (qdiscs) and shaping processes (classes) to first add priority to specific service flows and later guarantee and limit the bandwidth usage.

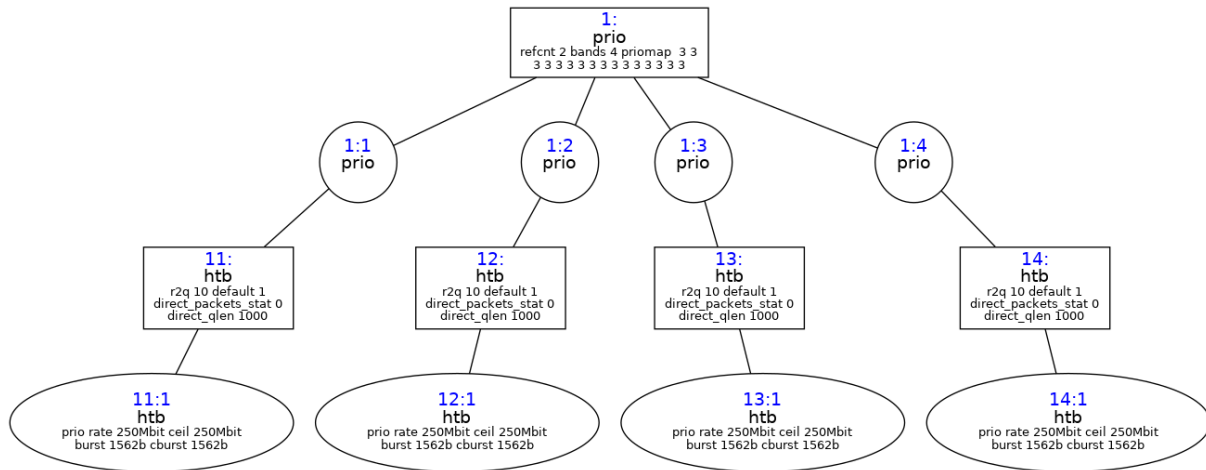


Figure 30. Example of a queuing schema in Traffic Control

Figure 31 depicts an example of a filtering expression that aims to match attributes of the network packets traversing the interface with a specific criterion. Notice that since built-in classifier expressions of the TC tool do not cover the wide spectrum of network protocols (including encapsulation ones) that may be found in network traffic flowing through a 6G architecture, it is then used a customised bit-matching technique based on the u32 classifier [16].

```
tc filter add dev ens2 parent 9000: prio 1 u32 match u8 0x00 0xff at 32 match u8 0x08 0xff
at 33 match u8 0x00 0xff at 34 match u8 0x00 0xff at 82 match u8 0x00 0xff at 83 match u8
0x00 0xff at 84 match u8 0x01 0xff at 85 match u8 0x11 0xff at 95 match u8 0x13 0xff at
106 match u8 0x8c 0xff at 107 match u8 0x13 0xff at 108 match u8 0x8c 0xff at 109 match
u8 0x60 0x7f at 115 flowid 1:1
```

Figure 31. Example of a filtering expression in Traffic Control

ACTIONS:

- Set TX Priority (Latency & Packet Loss Probability)
- Set TX Max bandwidth allowed
- Set TX Min bandwidth warranted

CLASSIFIER:

- 5G/6G Overlay supports (GTP),
- Edge/Core overlay support (VLAN, VXLAN, GRE, GENEVA),
- Nested Encapsulation Supported (MNVO, Infrastructure less)
- Industrial Support (Ethercat)

SCALABILITY:

- 1-2 Million packets per second (Mpps)
- Constant Latency & Packet Loss (SLA warranted)

TECHNOLOGY:

- Linux Kernel packet scheduler

4.3.4 Virtual Switch with Network Slicing Support (Based on OVS)

The 6G BRAINS architectural design (described in deliverable 2.2) follows SDN paradigm, providing a decoupling of control and forwarding network capabilities. SDN technology allows a centralised view of the network status and network topology, enabling centralised control and enhanced dynamic programmability when compared to traditional networks. Furthermore, the adoption of SDN alongside with virtualisation and softwarisation technologies such as NFV or Network Slicing makes it possible to use COTS equipment instead of dedicated hardware and network devices traditionally deployed in mobile infrastructures, COTS equipment is cheaper than dedicated hardware thus leading to a significant reduction of both OPEX and CAPEX costs for infrastructure operators. In this sense, there are many examples of well-known SDN controllers such as Floodlight [17], OpenDayLight [18], Ryu [19], Neutron [20], or ONOS [21], just to mention a few of the most widely known ones. This subsection provides the motivation and description of a novel virtual software switch with network slicing capabilities exposing an OpenFlow-based NBI to the above mentioned SDN controllers or to any other SDN controller with OpenFlow capabilities complying with the SDN paradigm.

The approach presented in the previous subsections provides network slicing capabilities through the Linux traffic control module. However, this approach faces the significant limitations of the bottleneck generated by the networking stack of the Linux kernel. On the one hand, the Linux kernel networking stack provides many features, network capabilities and security mechanisms that make it robust and reliable. On the other hand, these features entail a complexity that results in a bottleneck when processing network traffic in terms of number of packets per second (PPS). Therefore, a different approach is required, shifting to an alternative data path where the stringent requirements demanded by 6G use cases in terms of scalability and QoS are fulfilled.

In the context of the 6G BRAINS project and complementing the network slicing approach and solutions provided for hardware slicing (FPGA in Subsection 4.2) and for slicing in the Linux networking stack (Traffic Control in Subsection 4.3.1), this subsection introduces a novel Virtual Software Switch able to provide network slicing capabilities and enhanced programmability for the software data path. This Virtual Switch component, named UWS-OVS, is focused on providing network slicing capabilities in all the involved network segments that contain softwarised and/or virtualised VNFs, to be precise the software data path in the Edge and the Core network segments of the 6G network, those responsible for inter-connecting VNFs to the physical infrastructure. Such network slicing capabilities address the three different categories of traffic envisioned for 6G networks (eMBB, URLLC, mMTC) and also a concurrent combination of multiple of them running at the same time on the same infrastructure. The proposed solution also meets all the ambitious KPIs envisioned for 6G in terms of latency, bandwidth, density, scalability and reliability and all the challenges associated to achieve that purpose.

Following SDN principles, the virtual switch provides the control and management layers with extended and enhanced network programmability and an abstraction of the underlying network infrastructure. The following subsections provide an overall view of the architectural design, its components (modules, APIs, data structures), their functionality as well as a brief discussion of how these different components interact together to deliver the advanced traffic control features demanded in 6G networks.

4.3.5 Architectural design of the UWS-OVS Software Datapath

Figure 32 depicts the software data path architecture. It consists of six logical components that interact together to provide flexible fine-grained traffic control and network slicing capabilities and thus leading to meet the QoS requirements demanded by verticals and end-users and agreed in SLAs. The implementation has been carried out by significantly extending the code of OpenVSwitch (OVS), a well-known software switch that is considered the de-facto standard in virtualised environments. Regarding the logical system architecture designed in 6G BRAINS project and described in deliverable 2.2, the UWS-OVS component is part of the Data Plane, more precisely it is integrated as part of the wired Data Plane segment, i.e., MEC, transport and Core. Thus, it exposes a NBI to the Control Plane, using the OpenFlow protocol with UWS slicing extension profile to provide enhanced programmability of the Data Plane. The following is a short description of the functionality of each subcomponent of the UWS-OVS:

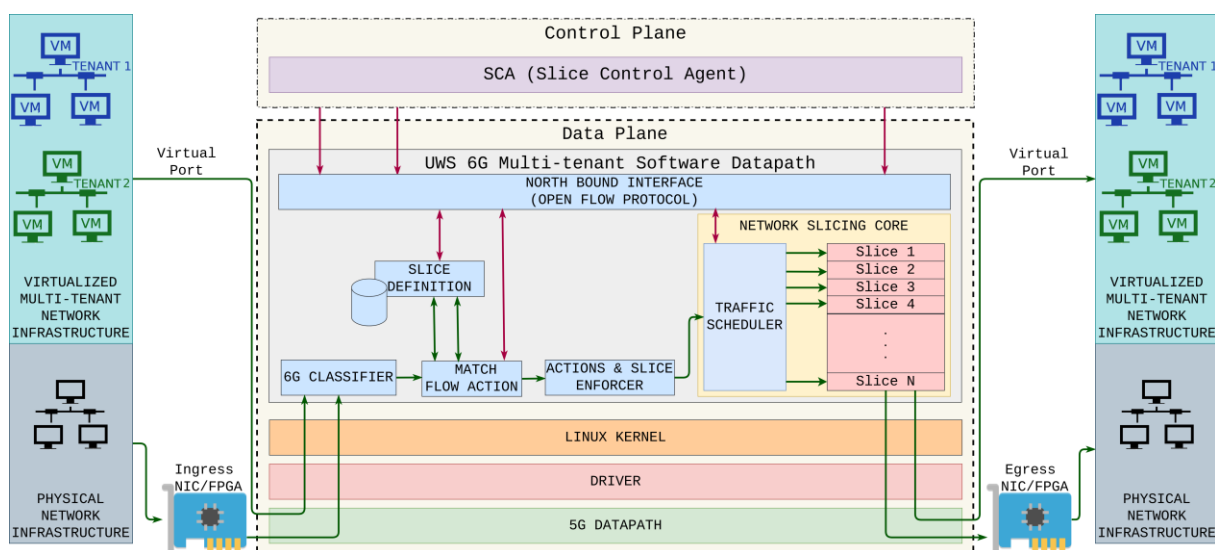


Figure 32. Architecture of the OVS-based Virtual Switch with network slicing support

1. **6G Multi-tenant Classifier:** It is the entry point of the software data path. It receives the network traffic and performs a deep inspection of every packet. The design and development of this parser is a significant enhancement over the current state of the art. It provides support for 5G/6G overlay networks (GTP), Edge/Core overlay support (VLAN, VXLAN, GRE, GENEVE), nested encapsulation and support for industrial environments (EtherCAT protocol). A more detailed explanation about its functionality is provided in subsection 4.3.6. The classifier has been designed with a modular architecture where every module is responsible to inspect a network protocol. There is re-entrance between modules. The re-entrance allows to optimise the parsing time whilst it is the mechanism for parsing overlay traffic with encapsulation and tunnelling protocols, allowing a deep inspection of the outer and inner headers.
2. **Slice Definition Table:** This module keeps a database containing the definition of the network slices in form of OF rules. The left part of the rule is the flow definition matching the slice. The right part includes the ID of the slice assigned to each flow and the actions to be performed on the traffic before sending it to the network slicing core.
3. **Match Flow Action:** This module compares the 6G flow key extracted by the 6G multi-tenant parser with the entries in the slice definition table, seeking a slice matching the packet.

4. **Actions and Slice Enforcer:** This module performs the actions associated with each slice before sending it to the last module of the pipeline. On each network interface or virtual port, a slice is created and set up using three parameters: *Maximum Allowed Bandwidth (MAB)*, *Minimum Guaranteed Bandwidth (MGB)* and *Priority*. The first two parameters define the bandwidth range committed and agreed with verticals and/or final users. The third one, the priority of the slice, is used to set the delay and the likelihood of packet loss (reliability) for each slice.
5. **Network Slicing Core:** This module is in charge of effectively enforcing the network slices in each of the network interfaces connected to the software data path, both physical and virtual.
6. **North Bound Interface:** This interface is responsible for the communication between the Control Plane, i.e., the Slice Controller Agent (SCA), and the software data path. For this purpose, it exposes an Open Flow API. The Open Flow protocol has been significantly extended, adding new fields and actions to enrich the expressiveness of the predicate of the Open Flow rules and thus, allowing a flexible definition of data flows able to encompass all possible data paths in 6G.

Regarding implementation details, the NBI runs as a daemon in user space. This daemon is also responsible for controlling the management of the OF tables. The Open Flow rules are kept in user space as a scalability mechanism. The rest of the architectural components are executed in kernel space as a kernel module integrating all their functionality.

4.3.6 6G Multi-tenant Traffic Classifier

Traditional traffic classifiers are based on 5-tuple including data about the outer L2/L3/L4 headers of the packets (see Table 10). This approach is not suitable to offer a differentiated QoS in 6G networks where the physical infrastructure could be shared by different tenants and where tenants and end users' information is encapsulated in the inner nested headers of the packets. Therefore, to provide enhanced programmability capabilities, the first step is to carry out a deep inspection of the 6G network traffic and extract information about end users and tenants from the inner headers.

Table 10. Open Flow fields for traditional 5-tuple based flow key

Key flow field	Length (Bytes)	Description of the field
ip_src	4	Source IP address from the outer IPv4 header
ip_dst	4	Destination IP address from the outer IPv4 header
udp_src	2	UDP source port from the outer IPv4 header
udp_dst	2	UDP destination port from the outer IPv4 header
ip_proto	1	Match the outer IPv4 protocol type (transport protocol)

For this reason, as a major contribution, the outcome of the 6G multi-tenant classifier is a new extended 6G flow key which represents a significant improvement over the traditional 5-tuple-based flow key provided by the OVS base version. The 6G flow key contains 17 new fields as

listed in Table 11 along with the length in bytes and the field description. The new meta-data extracted contains the required information to provide a flexible and fine-grained definition of network slices in 6G multi-tenant overlay networks. The new fields include information about the innermost overlay network in form of 6-tuple, the innermost DSCP value and information about the overlay networks.

Table 11. New Fields included in the extended 6G flow key

Key flow field	Length (Bytes)	Description of the field
inner_ip_src	4	Source IP address in the inner IP4 header
inner_ip_dst	4	Destination IP address in the inner IPv4 header
inner_tos	1	Type of Service value (TOS) in inner IPv4 header
inner_port_src	2	Source port in the inner UDP header
inner_port_dst	2	Destination port in the inner UDP header
inner_l3_protocol	2	Ethernet type of the inner layer 3 header
inner_l4_protocol	1	IANA protocol number of the inner layer 4 header
tunnel_type_1	1	Tunnel type of encapsulation level 1
tunnel_type_2	1	Tunnel type of encapsulation level 2
tunnel_type_3	1	Tunnel type of encapsulation level 3
tunnel_type_4	1	Tunnel type of encapsulation level 4
tunnel_type_5	1	Tunnel type of encapsulation level 5
tunnel_key_1	4	Tunnel key of encapsulation level 1
tunnel_key_2	4	Tunnel key of encapsulation level 2
tunnel_key_3	4	Tunnel key of encapsulation level 3
tunnel_key_4	4	Tunnel key of encapsulation level 4
tunnel_key_5	4	Tunnel key of encapsulation level 5

The 6G traffic classifier provides support for 8 different encapsulation and tagging protocols typically used in 6G multi-tenant networks as listed in Table 12. For each nested encapsulation level (i.e., for each overlay network), the gathered information consists of the encapsulation protocol (*tunnel_type_X* in Table 11) and the tunnel ID (*tunnel_key_x* in Table 11). The protocol header field matching the tunnel ID depends on the encapsulation protocol used as

indicated in Table 12. Notice that the pair (VNI, TEID) identifies unequivocally a mobile user. The maximum number of nested encapsulations supported is 5. This design decision is a trade-off between the overhead in matching time and flexibility to cope with all possible data paths expected in 6G.

Table 12. Open Flow fields for traditional 5-tuple based flow key

Encapsulation Protocol	Value	Tunnel key description
VXLAN	1	VXLAN Network ID (VNI)
GTP	2	Tunnel endpoint Identifier (TEID)
GRE	3	GRE Key
MPLS	4	MPLS Label
VLAN	5	VLAN Identifier (VNI)
NSH	6	Service Path Identifier (SPI)
VXLAN-GPE	7	VXLAN Network ID (VNI)
GENEVE	8	Virtual Network Identifier (VNI)

4.3.7 UWS-OVS Virtual Software Datapath API

The NBI interface exposed by the UWS-OVS component has been significantly extended over the base functionality supported by OVS, enabling an enhanced programmability of the software data path to the management and control layers. To this end, the Open Flow protocol has been extended with new fields allowing a fine-grained classification of the 6G network traffic. The novelty over the present state of the art is that the new fields include data from the inner encapsulated headers extracted by the 6G Classifier. This new information allows to classify, unambiguously identify and perform actions on data flows from tenants, end-users, or services regardless of the packet structure (number and type of encapsulations).

The first stage in the creation of a slice is the provision of network resources that will enable a guaranteed service delivery to the network traffic matching the slice. To do this, the NBI API provides support through a REST API to manage the complete life cycle of a slice in the software data path: creation (resource provisioning), dynamic modification and deletion (resources releasing). Figure 33 shows an example of creation and deletion of one slice in the port *vp120* with ID 1000. This slice is configured with a Maximum Allowed Bandwidth of 50 Mbps, a Minimum Guaranteed Bandwidth of 25 Mbps and priority 1.


```

root@6g-brains:~# uws-ovs --create-slice --id 1000 --iface vp120 --mba 50Mb --mbg
25Mb --priority 1
Slice ID 1000 created in interface vp120
Maximum Bandwidth Allowed (MAB) set to 50Mb
Minimum Bandwidth Guaranteed (MGB) set to 25Mb
Priority set to 1
root@6g-brains:~# uws-ovs --delete-slice --id 1000 --iface vp120
    
```

Figure 33. Creation and deletion of a slice in the software data path

Once the network resources needed to ensure the QoS for the slice have been provisioned, the next step is to assign traffic to the slice. This is accomplished using Open Flow rules which, as mentioned above, has been extended with enhanced programmability. The NBI API offers the Slice Control Agent two mechanisms for programming the software data plane (i.e. two ways of inserting Open Flow rules:

1. Using a REST API with the command line applications provided by OVS (as depicted in Figure 34). The functionality of the OVS command line applications have also been extended with the new fields and actions.
2. Using a Linux socket for sending Open Flow packets containing the Open Flow rules to the UWS-OVS component.

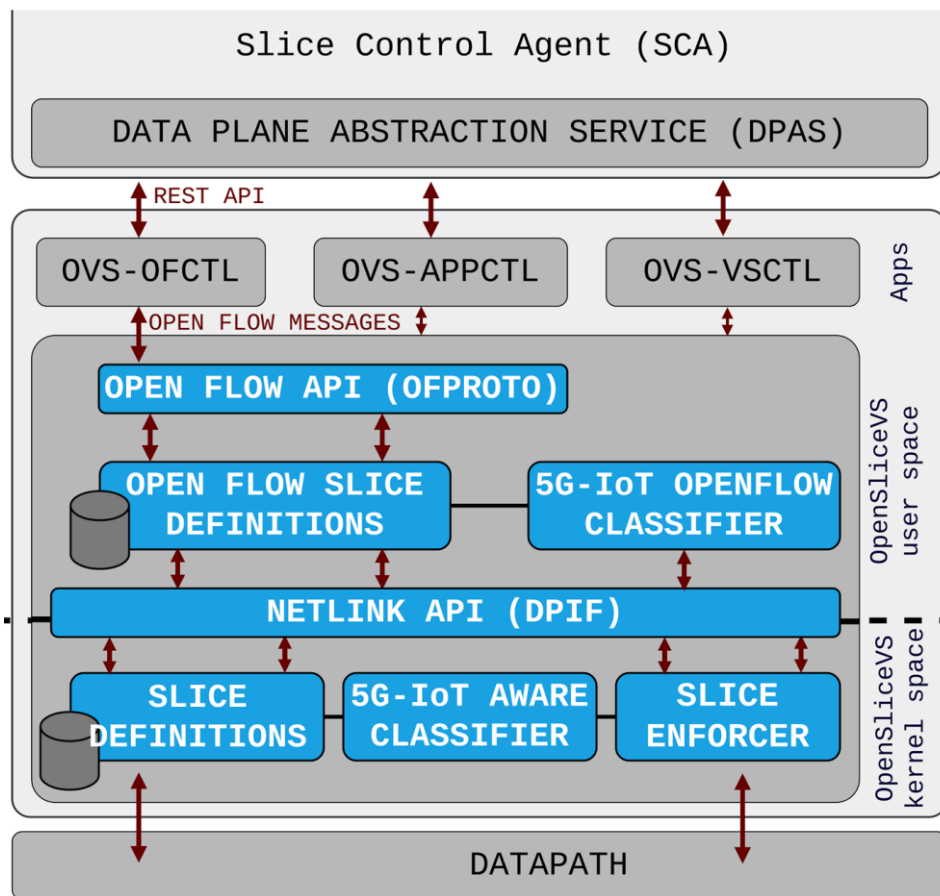


Figure 34. Software slicing architecture

4.3.8 UWS-OVS Virtual Software Datapath Empirical Results

The UWS-OVS component has been validated using 5 realistic IoT use cases that have allowed to evaluate the system performance in scenarios where different verticals pose diverse requirements in terms of latency, data rate, reliability and scalability (number of IoT devices and number of slices). These use cases cope with the three ITU categories or a combination of them as depicted in Figure 35. These use cases are: Ambient monitoring in Smart Agriculture, Robot Control in Industrial Automation, Sound Analysis in Smart City, Video Surveillance in Smart Building and AR-VR Tactile Application in Manufacturing Industry. The characteristics and requirements of each one of the use cases are listed in Table 13. These five use cases are closely aligned with the definition of the 6G-BRAINS enabling use cases for which a detailed technical description can be found in section 4 of deliverable 2.1. The 6G-BRAINS use cases have been carefully selected and defined based on an analysis of the current trends in the adoption of 5G and B5G technology and the available market forecast for the coming years. The results achieved and discussed below are a first approach to the key aspects of the technological innovations expected in 6G BRAINS project as listed in section 2.2 of deliverable 2.1 in terms of delay, reliability and peak data rate for the data plane software switches allocated in the Edge and Core network segments of the infrastructure. This proposed solution also allows a flexible definition of network slices for the software data plane in terms of multi-tenancy and flows with different levels of granularity while providing support for industrial environments and IoT protocols which is another technological innovation to be accomplished in 6GBRAINS project.

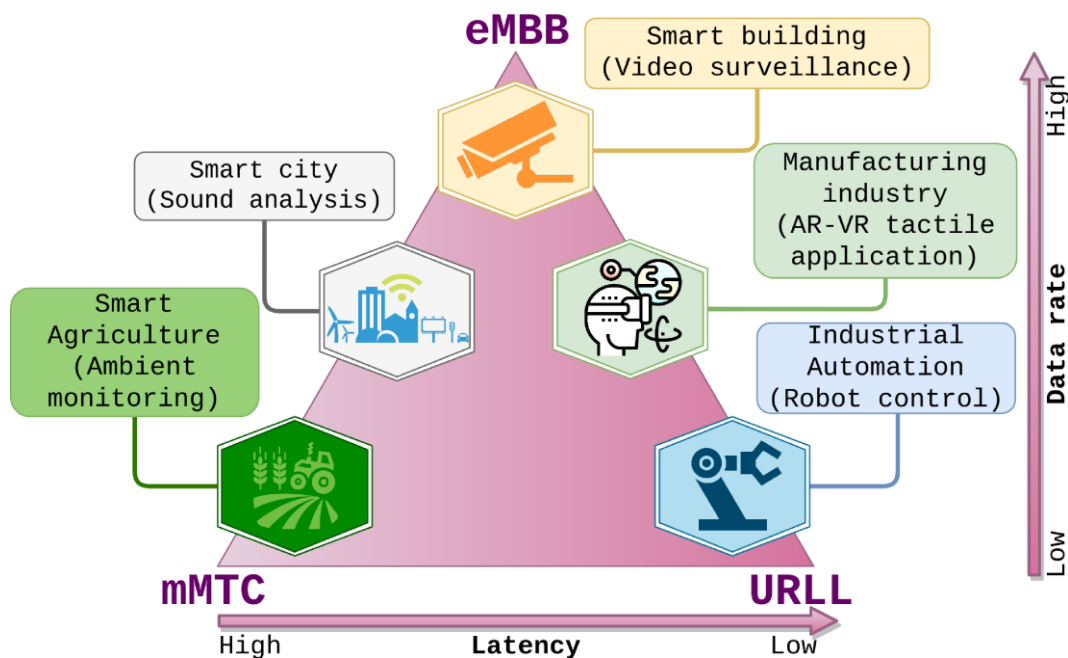


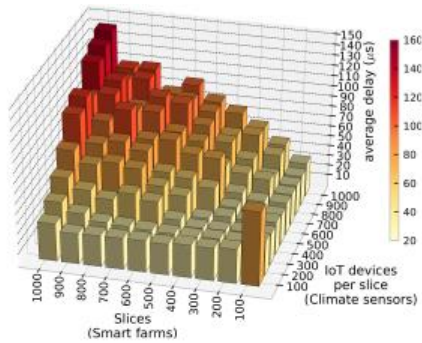
Figure 35. ITU category for every vertical oriented use case for evaluation of the UWS-OVS Virtual Software Datapath

Table 13 - Vertical oriented use cases for evaluation of the UWS-OVS Virtual Software Datapath: Characteristics, KPIs and traffic profile

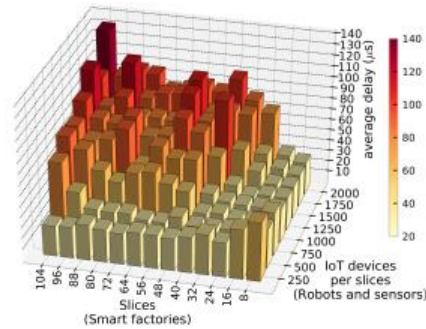
Use Case	Use case 1	Use case 2	Use case 3	Use case 4	Use case 5
	Ambient monitoring in Smart Agriculture	Robot control in Industrial Automation	Sound analysis in Smart Cities	Video surveillance in Smart Buildings	AV-VR tactile application in Manufacturing Industry
Category	mMTC	URLLC	mMTC/eMBB	eMBB	URLLC/eMBB
Latency	High (>1s)	Low (<1 ms)	High (>100 ms)	High (>100 ms)	Low (<1 ms)
Reliability	Low	High	Medium	Medium	High
Priority	Low	High	Medium	Medium	High
Maximum Number of Verticals (Slices)	1000	100	24	40	24
Maximum IoT devices per Vertical	1000	2000	3000	100	20
Maximum total devices	10 ⁶	2x10 ⁵	72000	4000	480
Packet size transmitted by IoT device (Bytes)	128	256	1396	1396	1396
Double encapsulated packet size (Bytes)	232	360	1500	1500	1500
Packets per Second (PPS) per device	1	4	16	400	3300
Device Tx bandwidth	1.86 kbps	11.52 kbps	192 kbps	4.8 Mbps	39.6 Mbps
Maximum accumulated bandwidth	1.86 Gbps	2.40 Gbps	13.82 Gbps	19.20 Gbps	19.00 Gbps

Figure 36 shows the system performance in terms of latency when ranging the number of verticals and the number of IoT devices per vertical for each use case. In the experiments carried out, each vertical has a dedicated slice, i.e., there is as many slices as verticals in every experiment. In the use case 1 (mMTC traffic), the maximum measured delay is 143 μ s when handling with emulated traffic from 1000 smart farms equipped with 1000 IoT sensors each one. This means the system is able to successfully cope with the traffic from 1 million IoT devices simultaneously. Concerning use case 2 (URLLC profile traffic), the outcome of the experiments carried out shows a very low latency (133 μ s for the worst case) and high reliability (0% packet loss). This result fulfils the strict requirements demanded by URLLC where high reliability and low latency are crucial KPIs. For use cases 3 and 4, the results gathered are beyond the requirements imposed by the verticals: maximum average delay of around 1 ms and 0% packet loss even when these use cases are tolerant to packet loss. Finally, use case 5 is the one demanding more severe requirements since its traffic profile is a hybrid URLLC/eMBB category. For this use case, the proposed solution is able to deliver the verticals requirements whilst being able to process a high bandwidth inherent to eMBB traffic: delay lower than 1 ms in the most stressful configuration with 24 verticals (slices) and 20 AR units per vertical, 0% packet loss and a total bandwidth of 19 Gbps.

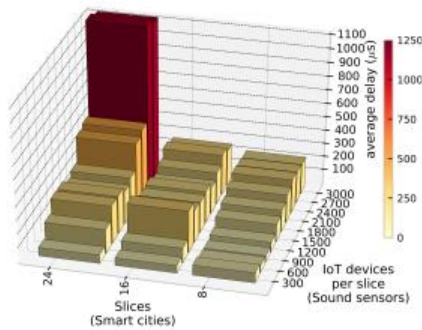
Therefore, to conclude, the gathered results evidence that the proposed UWS-OVS Software Datapath is able to satisfy the diverse QoS requirements imposed by the verticals for all the tested use cases. It is able to provide connectivity for up to 1 million IoT devices in mMTC traffic, achieve over 19 Gbps bandwidth in congested eMBB scenarios or ensure delays in the order of μ s for critical-missions communications (URLLC), providing high reliability in all of the tested scenarios (0% packet loss ratio).



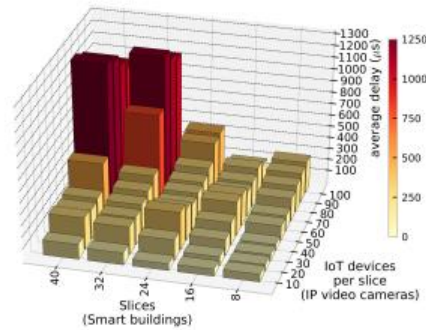
(a) Use case 1: Ambient monitoring in Smart Agriculture. Up to 1000 slices and 1,000,000 IoT devices sending 1,000,000 PPS at 1.86 Gbps.



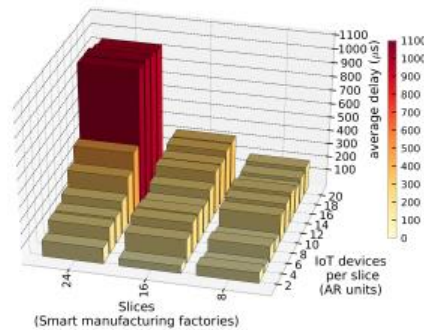
(b) Use case 2: Robot control in Industrial Automation. Up to 104 slices and 208,000 IoT devices sending 832,000 PPS at 2.40 Gbps.



(c) Use case 3: Sound analysis in Smart City. Up to 24 slices and 72,000 IoT devices sending 1,152,000 PPS at 13.82 Gbps.



(d) Use case 4: Video surveillance in Smart Building. Up to 40 slices and 4,000 IoT devices sending 1,600,000 PPS at 19.20 Gbps.



(e) Use case 5: AV-VR tactile app. in Manufacturing Industry. Up to 24 slices and 480 IoT devices sending 1,584,000 PPS at 19.00 Gbps.

Figure 36. Scalability results for the UWS-OVS component. Average delay when ranging the number of slices (verticals) and the number of IoT devices per slice. It is given the maximum number of IoT per Slice, packets per second (PPS) transmitted and accumulated Tx bandwidth configured for each use case

4.4 FlexCN-based core network slicing

5G is intended to deliver several benefits: high bandwidth/throughput, ultra-low latency communication, network slicing, and higher security. It opens up many new business opportunities for third-party partners. FlexCN is a solution that brings programmability to the Core Network. It creates a medium to allow users to manipulate the Core Network to satisfy the user's requirements.

Following its successor's mission, LL-MEC, one of the purposes of FlexCN is enabling mobile network monitoring, control, and programmability while retaining compatibility with 3GPP and ETSI specifications. At the same time, we would like to achieve coordinated resource programmability in end-to-end slicing scenarios by leveraging SDN towards an appropriate allocation of resources, thus drastically improving the performance of slices.

FlexCN could be used in many use cases that demand resource-hungry, content-rich services: low latency and high reliability.

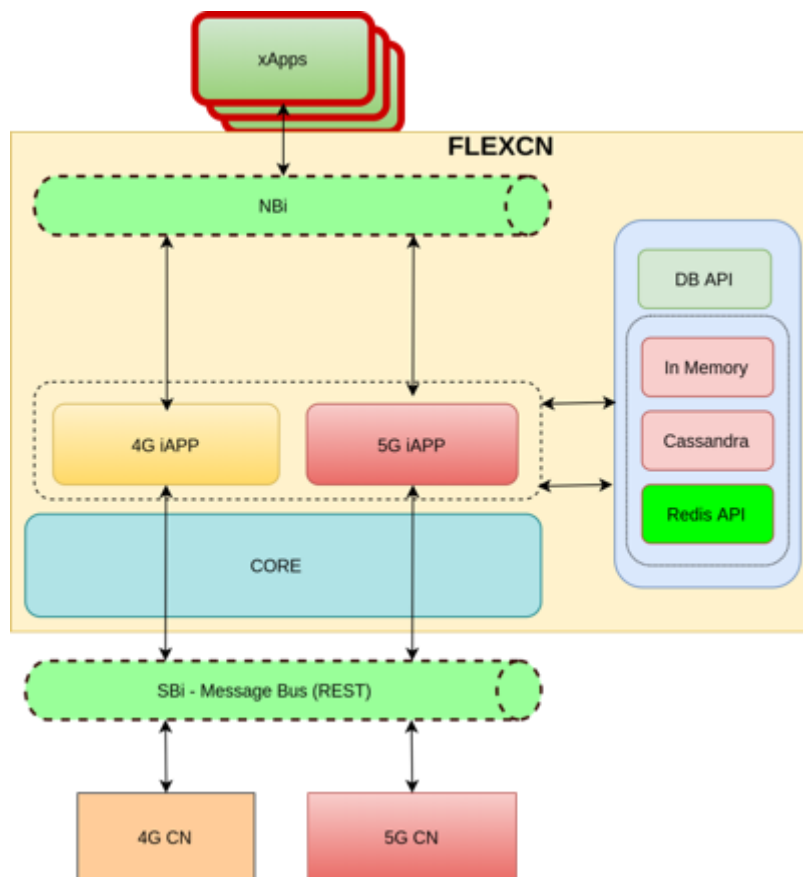


Figure 37. The FlexCN Architecture

FlexCN composes of the following components:

1. Southbound Interface: Is implemented using REST API to be 3GPP compatible. It takes responsibility for communication with the Core Network. For the 5G Core Network, the communication is realized by the exposure services of 5G core network components, where FlexCN plays the role of a client that sends requests to CN.
2. FlexCN's Core: Is the centralized manager of FlexCN to manage different tasks of FlexCN to ensure the communication between the northbound interface and sound bound interface.

3. iApps: iApps means internal apps; each iApp will serve a single functionality of FlexCN. This design decouples FlexCN's functionalities into several independent components. It allows the building of specialized versions of FlexCN, depending on user needs. We should note that iApp could be extended to user-defined components to extend FlexCN functionalities. At the moment of writing, each Core Network component such as SMF or AMF is served by a single iApp.
4. FlexCN's Database/ FlexCN context: All the statistics of the Core Network are stored in this component. These statistics could be served later by the requests of xApps from the North Bound Interface. The communication between the Core and FlexCN context is standardized for each iApp due to the different needs of each iApp.
5. Northbound Interface: Is responsible for the communication between the FlexCN and xApps. This could be realized through an SDK to simplify the communication process.
6. xApps: An external component of FlexCN, the actual brain of the FlexCN ecosystem. It contains the user logic that is described in a higher-level language. xApp can monitor and control the Core Network through FlexCN.

At the moment of writing, the northbound interface is implemented by a REST API as below and focuses on retrieving information from xApps.

API	Method	URLs
Get all the available information at the database	GET	http://FlexCN-URL:PORT/nxapp/v1/stats
Get all the available information of UE has this IMSI at the moment of this request	GET	http://FlexCN-URL:PORT/nxapp/v1/stats/:ueid
Triggering the subscription of FlexCN for a network function	GET	http://FlexCN-URL:PORT/nxapp/v1/trigger_subscribe/:ueid

Note: The exact URL might change in the future to adapt to the requirements of users

FlexCN faces several challenges. By following 3GPP standards, the feasible features are limited by the supported features of 3GPP's Core Network. The last two challenges remain from its predecessor, LL-MEC: scalability with a large number of users and services and, finally, the flexibility of registering low latency applications and services to support time-critical control decisions, priorities, and deadlines.

5 Initial Integration towards E2E Network Slice Control

5.1 Trirematics for the MANO of E2E Network Slicing

Deploying and managing the lifecycle of mobile networks, especially in cloud native environments, is critical to provide the required Service Level Agreement (availability, fast deployment recovery, etc.). Trirematics is an evolution towards an Agile, Zero-touch, and Telco-grade Multi-X platform in cloud native environments. Trirematics can be considered as the lightweight platform for the management and orchestration (MANO) of E2E mobile networks in cloud native environments, while targeting the zero-touch concept as its ultimate goal. Trirematics is composed of the following components (as illustrated in Figure 38):

5.1.1 Hydra – Supporting Heterogeneous Deployment

One of the intrinsic features of Trirematics is that it can support heterogeneous deployment in different environments, like Kubernetes, Openshift, public, private, or even hybrid cloud. Generally speaking, Trirematics can support any type of deployment heterogeneity, from application-level (sandboxed applications like snap or containerized applications like docker and podman) to OS (ubuntu, centos, redhat, etc.); supporting the deployment from multiple vendors (i.e., multi-vendor deployment), or even different versions from the same vendor (i.e., multi-version deployment). Providing such deployment heterogeneity is highly important in order to support, basically, any applications (PNF/VNF/CNF) in different clouds which may require different levels of security and configuration.

5.1.2 Infrastructure Resources and Mobile Network Resources

In order to provide monitoring and optimization for the network as a whole, Trirematics focus on both types of resources:

- Infrastructure resources for the objective of infrastructure optimization (CPU, memory, storage, networking, etc.). This can be done easily by many open-sourced projects like the Python client-library for Kubernetes that can be easily integrated later for machine learning purposes.
- Mobile network resources (e.g., radio resources at RAN part, and data rate per user at CN), for the objective of optimal resource allocation. This can be done easily using RAN and Core Network controllers such as FlexRIC and FlexCN.

Note that Trirematic lays on the top of FlexRIC and FlexCN to provide the E2E slicing, as described later.

5.1.3 Mon-X – The Monitoring Plane

Observability is the core part of checking the health of the whole system (similar in spirit to a doctor). The objective of observability is not only to monitor some metrics of interest (e.g., CPU consumption over time), but also tracing and logging the applications in order to provide meaningful information about the health of the whole system, i.e., the infrastructure, the deployed mobile network, and the applications. This is of paramount importance in very dynamic and heterogeneous environments like deploying Trirematics in Kubernetes.

5.1.4 AI/ML – The Cognitive Plane

Predictions and provisions of important decisions is the task of AI/ML. The decisions taken by AI/ML can be classified into two main classes: i) *short decisions* (i.e, decisions that need to be executed in small timescale) like radio resource scheduling, and ii) *medium/long decisions* (e.g., network slicing). Note that the first class (short decisions) will be executed directly through the interaction between the xApps (i.e., the applications implementing the AI/ML models) and the controllers like FlexRIC and FlexCN, while the second class (medium/long decisions) will be delegated to Auto-X. The delegation of these tasks to Auto-X is required to have a global optimization for the mobile network resources as well as the infrastructure resources, as illustrated in Figure 38. Note that, in many cases, there is a tight coupling between infrastructure and mobile network resource optimizations, e.g, changing the modulation and coding scheme (MCS) in the RAN part will also be translated into changing the CPU and memory consumption. Lastly, the AI/ML adopted in Trirematics follows the Machine Learning Operations (MLOPs) concept, in order to provide fast, efficient and reliable AI/ML models in production environments.

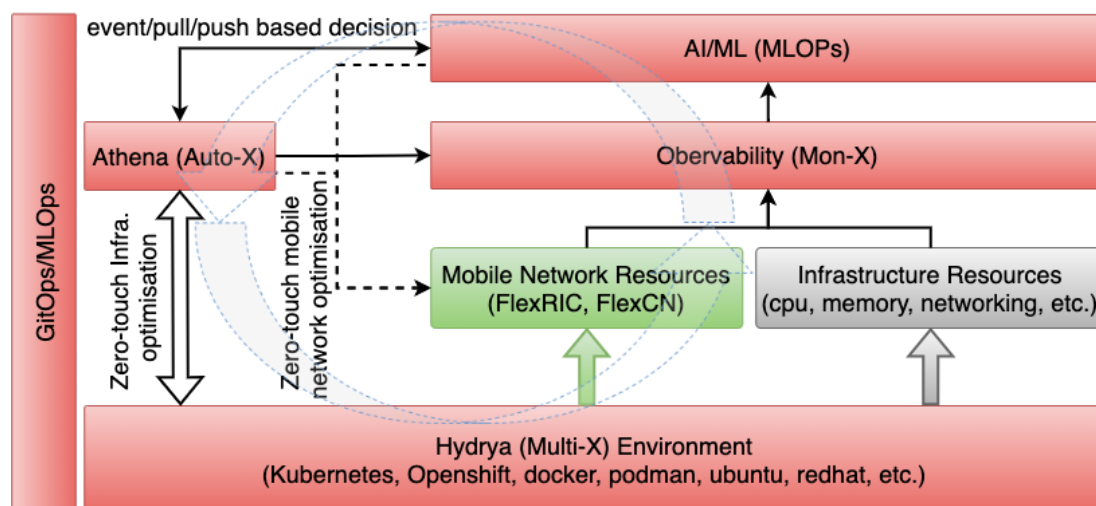


Figure 38. Trirematics Architecture

5.1.5 Athena – The Brain for Controlling Trirematics

It is the brain of Trirematics, where all the decisions are executed on the whole system in order to achieve the ultimate goal of zero-touch. Auto-X interacts with the observability for the objective of detecting any abnormal behaviour in the whole system (infrastructure, services, and applications) and taking actions accordingly, i.e., self-healing and self-(re)configuration. In order to perform proactive actions, not only reactive ones, that are done based on the observability, Auto-X relies on the predictions done by AI/ML for both the infrastructure and mobile networks. Finally, Auto-X interacts with both the infrastructure/application orchestrator (like Kubernetes) and the mobile network controllers (like FlexRIC and FlexCN) to perform the right actions, targeting the zero-touch concept.

5.1.6 GitOps for Continuous Integration

Trirematics relies on Continuous Integration (CI) and Git Operations (GitOps) concepts to provide reliable applications within a short time (i.e., fast time-to-market). This can be done through automating the whole chain of production.

5.2 The Reinforcement Learning Training Library

5.2.1 Training with TheRLib

The Reinforcement Learning Library extends pre-existing work done at Thales for the purpose of training and analysing agents on industrial simulations as well as academic benchmarks. We name that library TheRLib, and it is further divided into multiple conceptual blocks that address all the elements of a Reinforcement Learning experiment. All those conceptual blocks take the form of standalone Python packages:

- **Therenv:** The module responsible for agent-wise representation of environments.
- **Therconnect:** The module responsible for connecting Therenv representations to industrial simulations, when necessary.
- **TheRLib:** This module contains all the reinforcement learning related algorithms. From learning to logging.
- **Therutils:** Therutils makes the connection between the algorithm, the environment and the neural networks.
- **Therbench:** Therbench is responsible for the logging of benchmark data.
- **Therdash:** Therdash is a web interface used to visualize results, training curves, neural network architectures and other useful data.

In the following subsections, we will explain in more details all those modules and what role they play inside the global reinforcement pipeline.

These modules will be used within the different components of the MA-DRL Scheme introduced in the D2.2 and D2.3 deliverables. In Table 14, we explain the contribution of the different TheRLib modules in the RL Components.

Table 14 - TheRLib modules in the RL Components

RL Component	TheRLib module	Contribution
RL Agent Designer	<i>Therutils</i>	Software management of the algorithm's, environments and neural networks' configurations.
	<i>Therbench</i>	Database management of training configurations and trained models.
	<i>Therdash</i>	Visualization tools. For example, neural network architecture.
RL Agent Training Host	<i>Therlib</i>	RL algorithm training and logging. Building blocks for RL algorithms (loss function, replay buffer, etc.).
RL Engine	<i>Therbench</i>	Database management of training configurations and trained models.
	<i>Therdash</i>	General training monitoring and visualization tools.

RL Policy Adapter	<i>Therutils</i>	Software management of the algorithm's, environments and neural networks' configurations.
	<i>Therconnect</i>	Connection to specific deployment ecosystems, e.g., FlexRIC-based.
	<i>Therenv</i>	Generic RL Environment representation.
RL Reward Monitor	<i>Therconnect</i>	Connection to specific deployment ecosystems, e.g., FlexRIC-based.
RL Simulator	<i>Therenv</i>	RL Environment representation, RL Environment data processing and execution wrappers.
Resource Analytics	<i>Therconnect</i>	Connection to specific deployment ecosystems, e.g., FlexRIC-based.

5.2.2 Therenv

This module holds all the environment specific functions. It is mainly responsible to connect an RL agent to the real environment. From the creation of the environment object itself to the pre-processing and post-processing wrappers that surround its interaction with the agent. All the environments represented inside Therenv must oblige to a Gym-like interface. The environment is controlled through the following access points:

1. `reset()`: The reset method prepares all the objects required to the simulation and constructs the initial state of the environment, which is returned by the function.
2. `step(action)`: Step takes an action, applies it on the simulation and returns the state, reward, done and info. Done refers to a Boolean value indicating whether the episode is over not while info contains additional simulation information that might be useful for debugging.
3. `render(mode)`: This method renders the environment and depending on the mode returns an image representation or interacts with a viewer to display the current simulation state.
4. `close()`: Cleans everything to prepare the end of the simulation.

As mentioned above, Therenv also contains a number of wrappers that hold the job of pre-processing and post-processing simulation data. The core idea behind the wrappers is to streamline the pipeline so that the agent simply takes the state as an input and outputs an action without too many considerations on domain representations. Mainly:

- **Multi-processing Handling**: This wrapper allows to structure parallel environments in the form of batch data.
- **Action pre-processing**: It can be useful to change the "normal" action space of the simulation so that it is easier for the agent to handle. For instance, normalizing actions to be in the $[0, 1]$ range instead of the "real" range.
- **State post-processing**: A usual RL trick consists of adding the last action taken by the agent in the state space. It can also be useful to do one-hot encoding or other usual processing techniques.

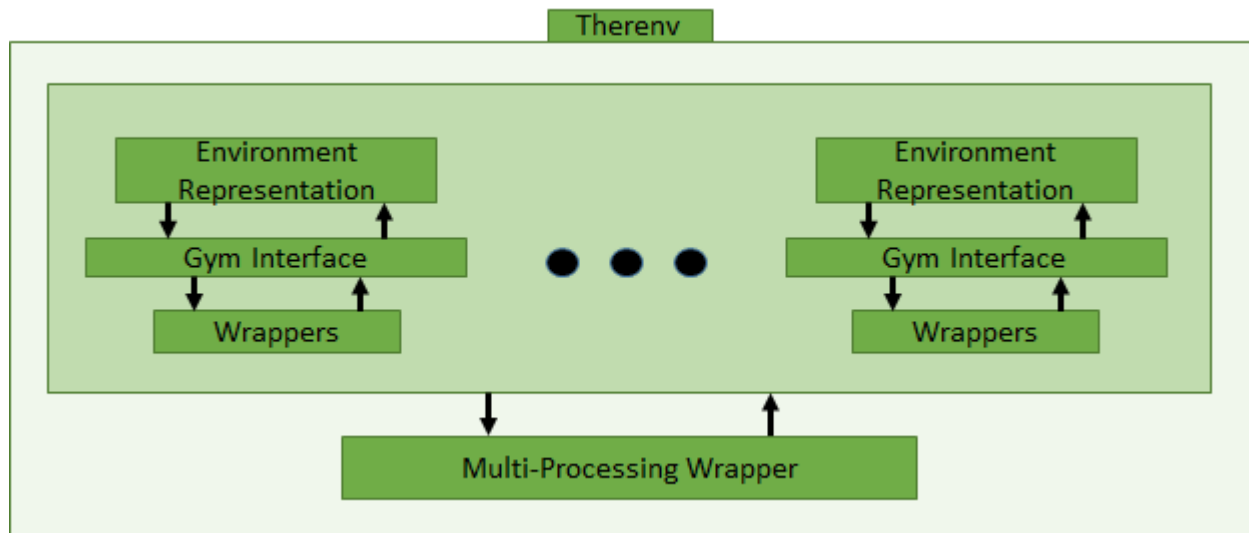


Figure 39. Therenv's conceptual role in the RL pipeline

5.2.3 Therconnect

Therconnect is responsible for connecting Therenv representations to industrial simulations or specific deployment environments, when necessary. In the 6G-BRAINS project, we can consider a set of Control and Monitoring components used to provide actions, retrieve observation and reward data as an Industrial Simulation for Therconnect.

The module manages the interoperability methods and protocols to provide commands to the targeted system and get data from it. If the components we are addressing already provide a Gym-like interface, the adaptations needed in Therconnect are minimal or non-existent.

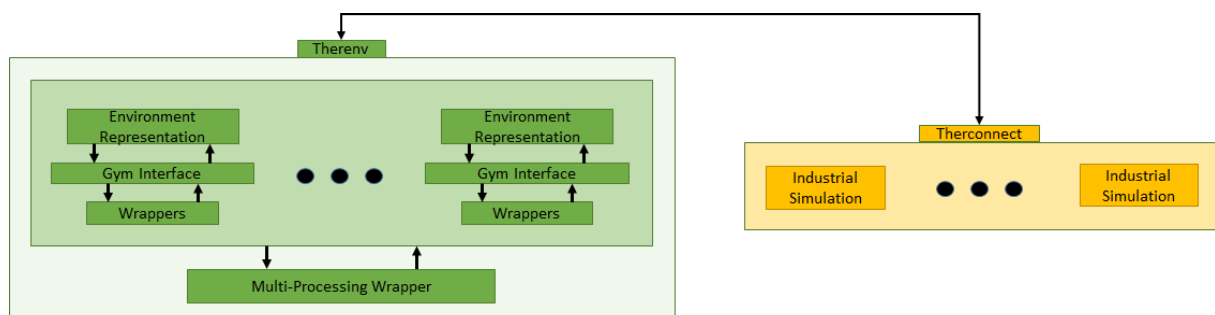


Figure 40. The connector is used to link python environment representations to industrial simulations

5.2.4 TheRLib

The Therlib module contains all the reinforcement learning related algorithms. Its design guidelines are to provide implementation of different state of the art RL algorithms in a modular fashion, while being agnostic to environment representations and agent deep neural network models. We focus on building blocks that are leveraged by different algorithms, such as agent memory management with Replay Buffer for off-policy algorithms, and Trajectory Rollout for on-policy algorithms. We also provide RL loss function implementations for algorithms that are faithful to the research papers' formulations, such as Proximal Policy Optimization (PPO) [22], while adding as option well-known implementation tricks such as gradient clipping. We leverage these modular algorithmic building blocks as well as Therenv

multi-processing management to create training pipelines with several environments and/or agents.

The second function of the Therlib module consists in logging the state of the RL agents throughout their training. The Database logger provides generic utility to:

- Save and load Neural Network representations of the agents.
- Save training related metrics, such as rewards, losses or environment-specific metrics.

Thus, we can exploit those outputs for evaluation and visualization of the trainings, or as intermediate states in distributed training pipelines.

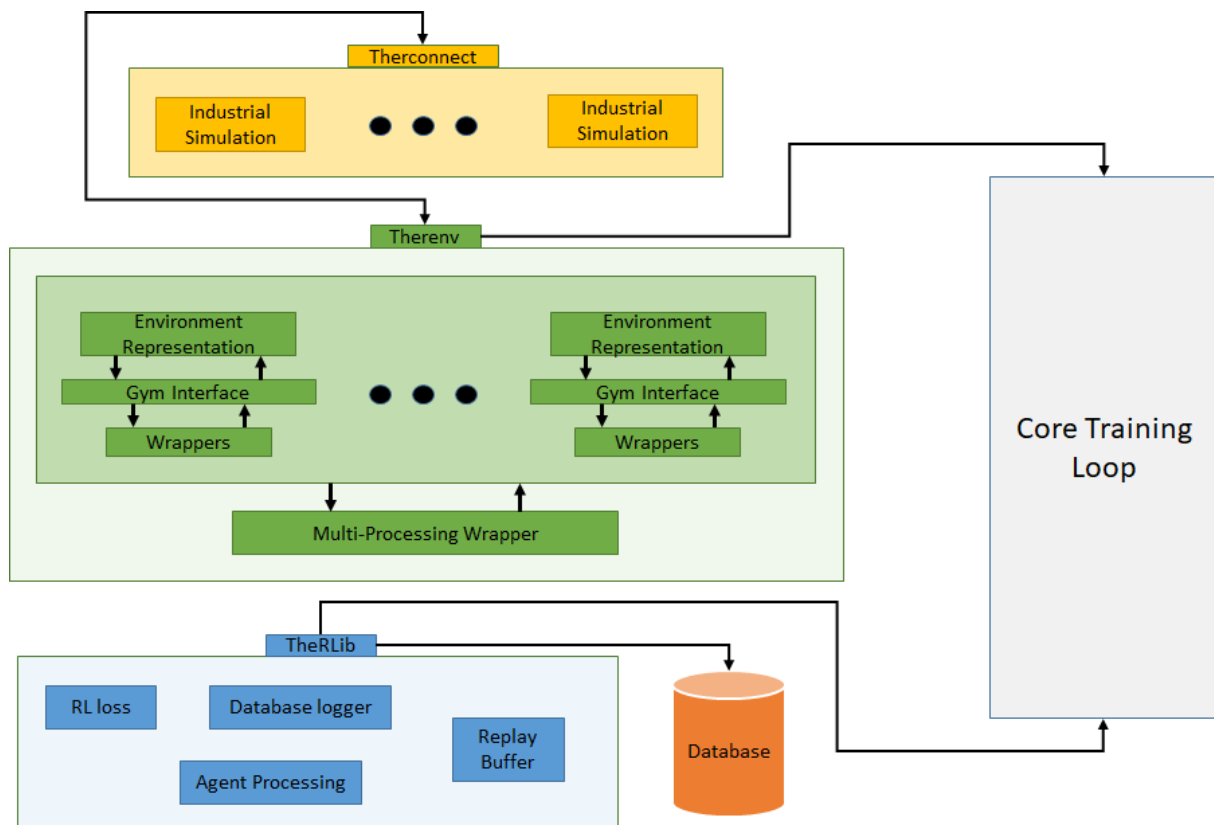


Figure 41. TheRLib holds environment agnostic RL features

5.2.5 Therutils

We use Therutils to bridge the gap between the environments of Therenv and the reinforcement learning algorithms of TheRLib. It contains the neural network architectures dedicated to certain environments and training scheme options used for experiments. This library is a helper module that could be ignored in a minimalistic approach. However, it is very helpful to industrialize trainings.

In the Figure 42 below, a typical training pipeline with TheRLib involves the Core Training Loop block, which uses:

- Therutils sub-components to manage the configuration parameters and the RL agent neural network model.
- A Therenv-based RL Environment.
- A TheRLib-based RL algorithm implementation.
- TheRLib-based model and data logging features.

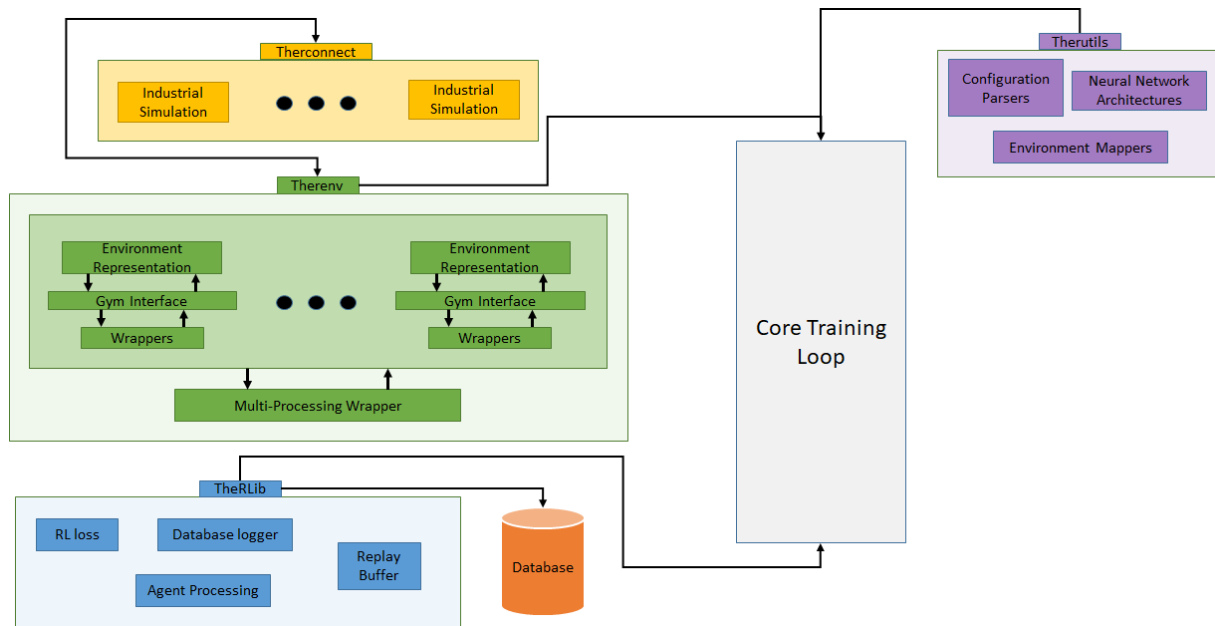


Figure 42. Therutils logs neural network architectures and experiment parsers

5.2.6 Therbench

This module is used for non-regression testing throughout the research and development process.

It manages a workplan on reference experiments with robust algorithm implementations on a set of RL environments from the research literature. These experiments’ agent models and training data are stored in a reference database. We also verify the reproducibility of experiments given that we have specified the relevant random seeds.

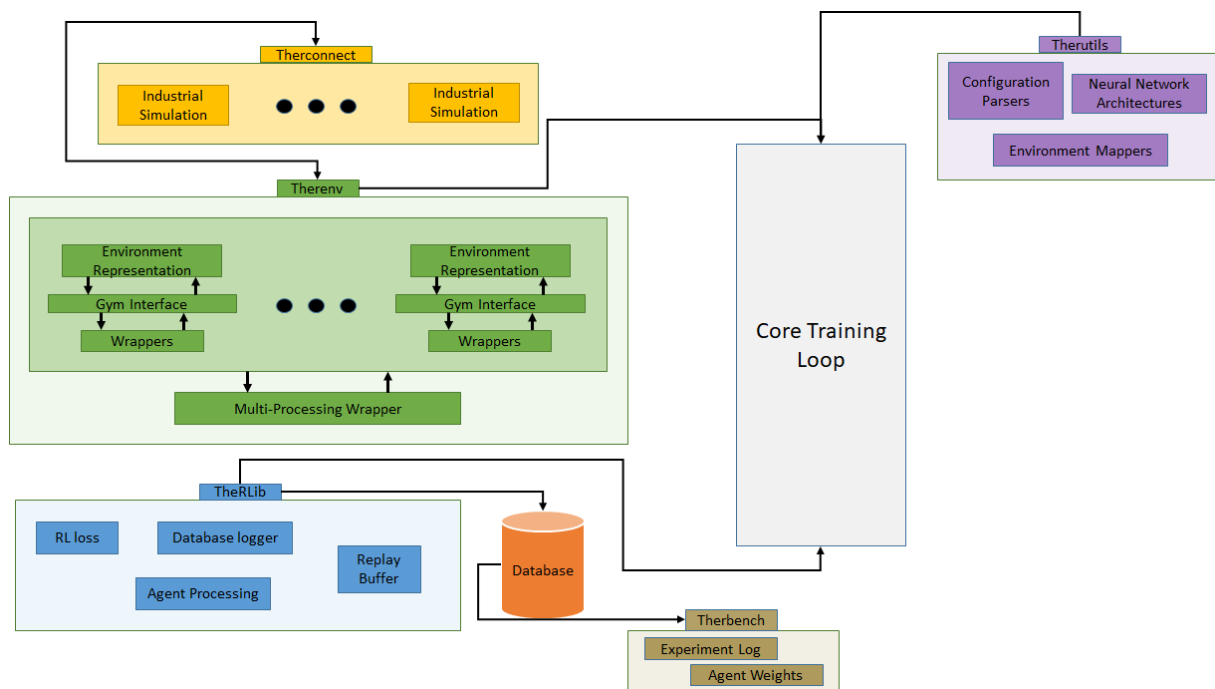


Figure 43. Therbench is used to log experiments for non-regression testing

5.2.7 Therdash

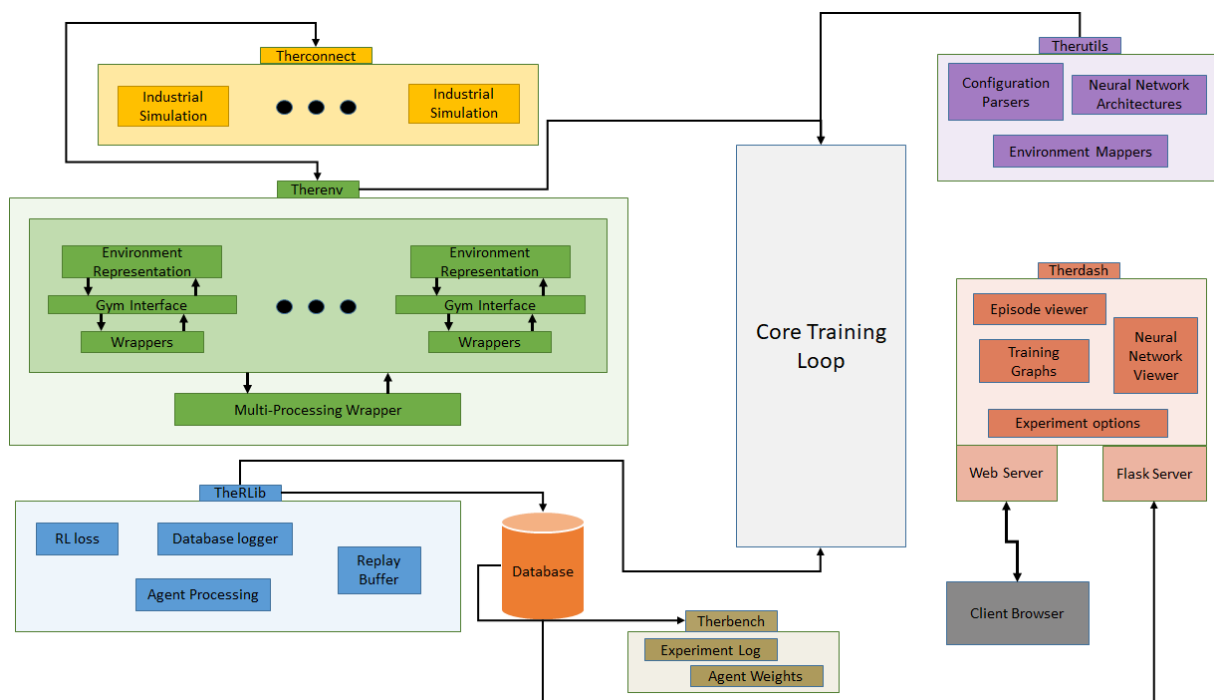


Figure 44. The dashboard is used to visualize experiments

Therdash is a web interface that acts as a front-end to manage RL experiments performed with the TheRLib libraries described above. Similarly, we will use it to manage the RL trainings performed with the MA-DRL scheme during the course of the 6G-BRAINS project.

As usual with web applications, Therdash architecture can be split between two parts: server-side and client-side. The server-side part is now written in Python with the Flask micro-framework. It acts as a glue between the client-side part, the trainings' models and metrics data, and the TheRLib modules. The client-side part is a lightweight web server that uses HTML, CSS and JavaScript to manage the user interactions and the training data visualization. In the setup we describe here, the Therdash client-side and server-side are self-hosted in the local (industrial) network. It is not a Software as a Service we access over a cloud.

We provide two main views in the Therdash web-interface. The first one is the control panel introduced by the Figure 45. It provides an overview of past and present RL experiments and links to the more detailed experiment view. In practice, this view is generated through a request to the Flask Server that queries the different entries of the experiments database. We can see on each row the experiment status, the experiment id that links to the detailed experiment view, a short description and the current and best score. An example naming convention for an experiment is "Experiment with *algorithm* on *environment*", but it is free for the RL researcher to decide. Similarly, one can freely implement the score as the mean episode reward or a success percentage on a set of test episodes.

The second major view is the experiment view that allows us to analyse the outcome of one RL experiment. It is organized with different sub-views represented as blocks within the Therdash part in Figure 44:

- The Experiment Options views lists all hyper-parameters that were used during the training. These can be bound to the simulation environment, the training algorithm or the neural network model.

- The Neural Network viewer, represented in the Figure 47, provides a view of the Neural Network architecture of the RL agent model used in the training.
- The Episode Viewer, represented in the Figure 48, allows us to select a particular state of the agent during the training, perform some test experiments on the RL Environment and visualize the outcome as a video, given that the RL Environment implements the *render* method.
- The Training Graphs, represented in the Figure 46, help us visualize the evolution of metrics throughout the training. We also manage a dedicated view for pipelines such as Population Based Training (PBT) [23] that train multiple agents in parallel on the same problem. This particular algorithm manages a population of RL agents with varying hyper-parameters through an evolutionary strategy. Thus, the desired outcome is the best set of hyper-parameters or the best schedule of hyper-parameters for one particular RL algorithm on one particular RL environment.

All our visualization sub-components are still a rough state, that we intend to refine during the course of 6G-BRAINS. Dedicated views based on Therdash will be especially useful for the RL Agent Designer and RL Engine components.

Control panel

State	Experiment ID	Description	Current Score	Best Score
	1	PPO:16 on CartPole-v0 with 20000 frames for training	198.23	200
	2	PPO:16 on CartPole-v0 with 20000 frames for training	198.23	200
	3	PPO:16 on Pendulum-v0 with 15000 frames for training	-184.5665737288612	-171.2043346835339
	4	PPO:16 on Pendulum-v0 with 15000 frames for training	-184.5665737288612	-171.2043346835339
	9	experiment with IPPO on SMAC	19.73369863013699	20
	10	replication with IPPO on SMAC	19.73369863013699	20
	11	experiment with MAPPO on SMAC	15.882739726027395	18.713972602739727
	12	replication with MAPPO on SMAC	15.882739726027395	18.713972602739727
	13	experiment with QMIX on SMAC	16.470000000000006	16.646438356164378
	14	replication with QMIX on SMAC	16.470000000000006	16.646438356164378
	15	experiment with PBT with PPO on cartpole	200	200
	16	replication with PBT with PPO on cartpole	200	200
	17	experiment with PBT with PPO on pendulum	-155.08289677336757	-148.38615282732903
	18	replication with PBT with PPO on pendulum	-168.87716603489235	-148.38615282732903

Figure 45. The dashboard is used to monitor training experiments

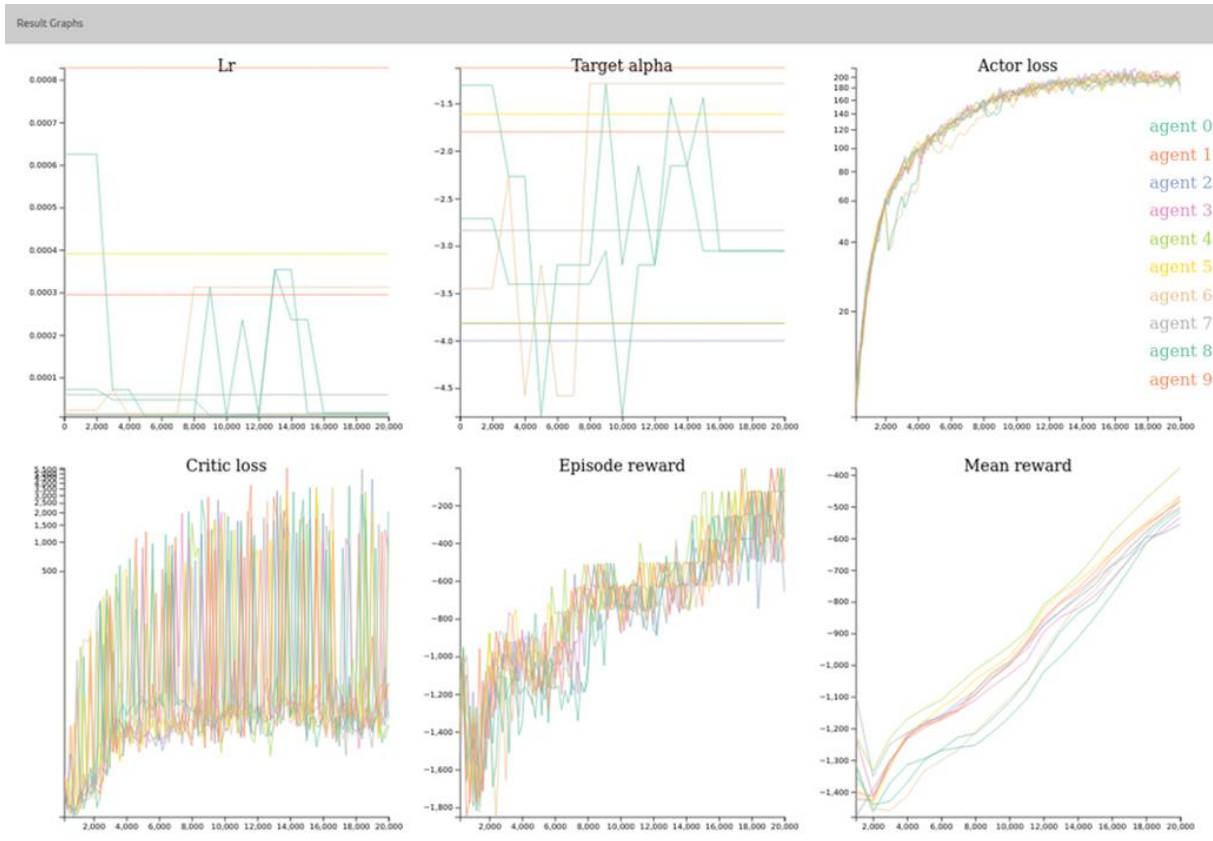


Figure 46. We create graphs and log experiment data to follow the training process

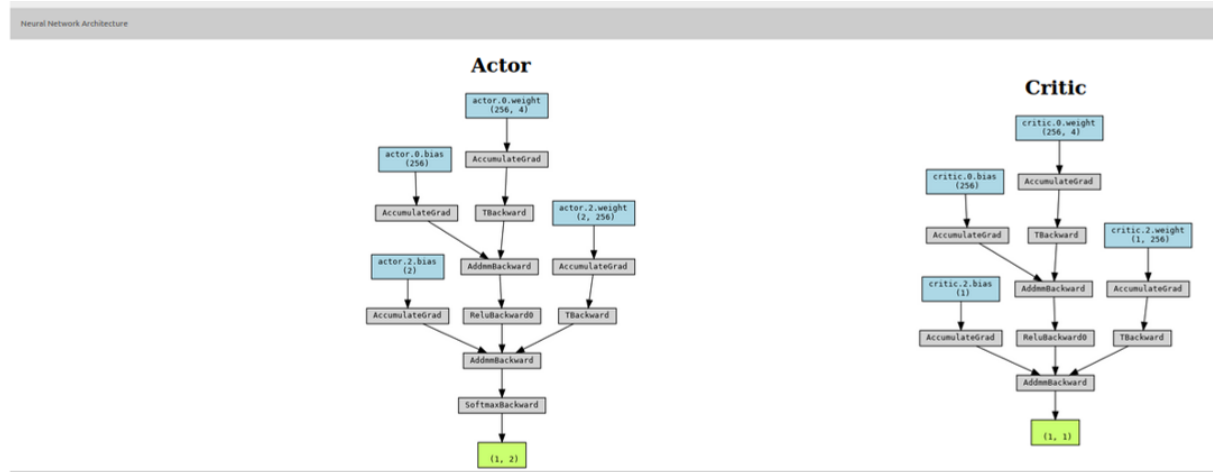


Figure 47. View of the DRL agent Neural Network architecture

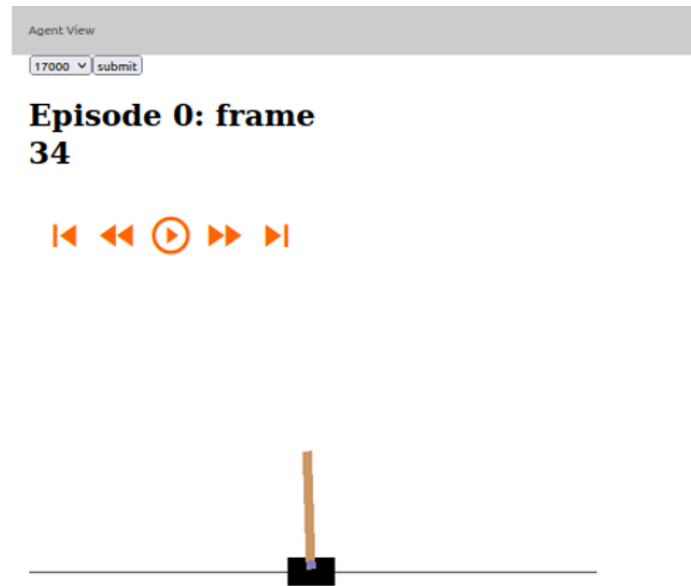


Figure 48. Episode viewer of trained agents

Web-API for training configuration and launching

Currently, Therdash is only used as a visualization module. We want to improve the experiment design process by adding another module dedicated to the configuration and launching of experiments.

Backend connector module

Currently, the link between Therdash and the RL experiments is done through TheRLib. To allow more modularity in the training pipeline, we want to extract this communication module. In that way, it will be possible to use different RL backends (RLlib, stable-baselines, etc.) with a shared visualization module: Therdash. This will prove useful to integrate agents that were trained beforehand by the 6G-BRAINS partners, with their own set of tools.

6 Conclusions

In this deliverable, we have defined and describe the key components that enables an E2E Slicing integration between the partners of WP5.

Technologically, we have defined and described:

- FlexRIC and FlexCN as ways to bring programmability into the RAN and Core Network, respectively
- Both hardware-based and software-based approach for high-performance network slicing over the Edge, Transport, and Core Network
- Trirematics as a toolkit for both the deployment and management-orchestration of an E2E slicing system
- TheRLib as a development kit for implementing a flexible and scalable cognitive plane of an E2E slicing system

Algorithmically, we have explained our first investigations into the possibilities of using ML for Radio Link Control and RAN Slice Scheduling.

With these key components and case studies laid out, we expect to have formed a shared conceptual understanding that facilitates deeper integrations in future deliverables.

References

- [1] M. K. M. K. K. Xenofon Foukas, "Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture," in *International Conference on Mobile Computing and Networking (MobiCom)*, 2017.
- [2] N. N. M. M. K. M. K. M. K. K. Xenofon Foukas, "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks," in *International on Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2016.
- [3] R. M. S. S. M. L. G. D. G. a. R. T. Diego Dupleich, "Characterization of the Propagation Channel in Conference Room Scenario at 190 GHz," in *European Conference on Antennas and Propagation (EuCAP)*, 2020.
- [4] "Verizon 5G Edge with AWS Outposts," Verizon, [Online]. Available: <https://www.verizon.com/business/solutions/5g/edge-computing/aws-outposts-verizon-5g-edge/>. [Accessed 17th December 2021].
- [5] "Network slicing: the top 10 most profitable industries," Ericsson, [Online]. Available: <https://www.ericsson.com/en/blog/2021/6/network-slicing-top-10-industries>. [Accessed 17th December 2021].
- [6] N. B. Ruparelia, *Cloud Computing*, The MIT Press, 2016.
- [7] P. R.-D. S. S. P. A. J. R.-M. J. L.-S. Jorge Navarro-Ortiz, "A Survey on 5G Usage Scenarios and Traffic Models," *IEEE Communications Surveys & Tutorials*, 2020.
- [8] R. A. K. P. B. R. R. S. Parastoo Sadeghi, "Finite-state Markov modeling of fading channels: A survey of principles and applications," *IEEE Signal Processing Magazine*, 2008.
- [9] D. N. C. T. R. L. Pramod Viswanath, "Opportunistic Beamforming Using Dumb Antennas," *IEEE Transactions on Information Theory*, 2002.
- [10] A. S. Tao Guo, "Enabling 5G RAN Slicing With EDF Slice Scheduling," *IEEE Transactions on Vehicular Technology*, 2018.
- [11] N. C. T. H. Qiang Liu, "OnSlicing: Online End-to-End Network Slicing with Reinforcement Learning," in *International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2021.
- [12] N. DOCOMO, "5g evolution and 6g, whitepaper," 2020.
- [13] "Simple Sume Switch Architecture," November 2019. [Online]. Available: <https://github.com/NetFPGA/P4-NetFPGA-public/wiki/Workflow-Overview>.

- [14] M. R. D. H. M. & K. R. Jacobsen, "RIFFA 2.1: A reusable integration framework for FPGA accelerators," *ACM Transactions on Reconfigurable Technology and Systems (TRETs)*, vol. 8, no. 4, pp. 1-23, 2015.
- [15] "Traffic Control tc(8), Linux man page," [Online]. Available: <https://linux.die.net/man/8/tc>.
- [16] "Traffic Control tc-u32(8), Linux man page," [Online]. Available: <https://man7.org/linux/man-pages/man8/tc-u32.8.html>.
- [17] "Floodlight project, Floodlight Controller," [Online]. Available: <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview>.
- [18] "Open daylight, the linux foundation projects," [Online]. Available: <https://www.opendaylight.org/>.
- [19] "Ryu," [Online]. Available: <https://ryu-sdn.org/>.
- [20] "Open stack networking, Neutron," [Online]. Available: <https://wiki.openstack.org/wiki/Neutron>.
- [21] "Open network operating system, Technical steering team (TST)," [Online]. Available: <https://opennetworking.org/onos/>.
- [22] F. W. P. D. A. R. O. K. John Schulman, "Proximal Policy Optimization Algorithms," in *arXiv:1707.06347*, 2017.
- [23] V. D. S. O. W. M. C. J. D. A. R. O. V. T. G. I. D. K. S. C. F. K. K. Max Jaderberg, "Population Based Training of Neural Networks," in *arXiv:1711.09846*, 2017.

[End of document]