Beyond 5G Multi-Tenant Private Networks Integrating Cellular, Wi-Fi, and LiFi, Powered by Artificial Intelligence and Intent Based Policy

5G-CLARITY Deliverable D4.2

Validation of 5G-CLARITY SDN/NFV Platform, Interface Design with 5G Service Platform, and Initial Evaluation of ML Algorithms

Contractual Date of Delivery:	**June 30, 2021**
Actual Date of Delivery:	**July 30, 2021**
Editor(s):	**Jose Ordonez-Lucena (TID)**
Author(s):	**Oscar Adamuz-Hinojosa, Pablo Ameigeiras, Lorena Chinchilla, Pablo Muñoz, Jorge Navarro-Ortiz, Jonathan Prados-Garzon, Juan José Ramos-Muñoz (UGR), Daniel Camps-Mur, Ferrán Cañellas Cruz (I2CAT), Jordi Pérez-Romero, Oriol Sallent, Irene Vilà (UPC), Meysam Goodarzi, Jesús Gutiérrez, Vladica Sark (IHP), Ardimas Purwita, Anil Yesilkaya (USTRATH), Antonio Garcia, Kiran Chackravaram (ACC), Erik Aumayr, Joseph Mcnamara (LMI), Carlos Colman Meixner, Xueqing Zhou, Amin Emani, Shuangyi Yan (UNIVBRIS), Srinivasan Raju, Rui Bian (PLF), Tezcan Cogalan, Ibrahim Hemadeh (IDCC), Mir Ghoraishi (GIGASYS)**
Work Package:	**WP4**
Target Dissemination Level:	**Public**

This document has been produced in the course of 5G-CLARITY Project. The research leading to these results received funding from the European Commission H2020 Programme under grant agreement No. H2020-871428. All information in this document is provided “as is", there is no guarantee that the information is fit for any particular purpose. The user thereof uses the information at its own risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.

Revision History

Revision	Date	Editor / Commentator	Description of Edits
0.1	29.01.2021	Jose Ordonez-Lucena (TID)	D4.2 ToC and contributors
0.2	12.02.2021	Daniel Camps-Mur (I2CAT) Jose Ordonez-Lucena (TID)	I2CAT: Subsection proposal for Chapter 2 TID: Subsection proposal for Chapter 6
0.3	25.02.2021	Vladica Sark (IHP) Erik Aumayr (LMI) Tezcan Cogalan (IDCC)	IHP: Subsection proposal for Chapter 3 LMI: Subsection proposal for Chapters 4 and 5 IDCC, LMI: Define telemetry use case in 5.3.2 (intent engine)
0.4	12.03.2021	Daniel Camps-Mur (I2CAT) Vladica Sark (IHP) Erik Aumayr (LMI)	I2CAT [editor], ACC, UGR, IDCC, TID, PLF: Initial content on Section 2. IHP [editor], USTRATH, UPC, UNIVBRIS, UGR, IDCC: Initial content on Section 3 LMI [editor]: Initial content on Sections 4 and 5. *For this initial content, results are not available yet. It is a simple draft where contributors describe the considered simulation framework and highlight expected results.*
0.5	26.03.2021	Jose Ordonez-Lucena (TID) Daniel Camps-Mur (I2CAT) Pablo Muñoz, Jonathan Prados-Garzon (UGR)	TID: Review of Section 2 (initial content) I2CAT: Review of Section 3 (initial content) UGR: Review of Section 4 and 5 (initial content)
0.6	09.04.2021	Jose Ordonez-Lucena (TID) Tezcan Cogalan (IDCC)	TID [editor], UGR, UNIVBRIS: Section 6.1 – 1st draft TID [editor], UNIVBRIS: Section 6.2 – 1st draft IDCC [editor], LMI: Section 6.3 – 1st draft
0.7	07.05.2021	Daniel Camps-Mur (I2CAT) Vladica Sark (IHP) Erik Aumayr (LMI) Antonio Garcia (ACC)	I2CAT [editor], ACC, UGR, IDCC, TID, PLF: Section 2 – 1st draft IHP [editor], USTRATH, UPC, UNIVBRIS, UGR, IDCC: Section 3 – 1st draft LMI [editor]: Sections 4 and 5 – 1st draft ACC: Review of Section 6 (1st draft) *The 1st draft of Sections 2, 3, 4 and 5includes the first set of results.*
0.8	14.05.2021	Jose Ordonez-Lucena (TID) Daniel Camps-Mur (I2CAT) Pablo Munoz-Luengo, Jonathan Prados-Garzon (UGR)	TID: Review of Section 2 (1st draft) I2CAT: Review of Section 3 (1st draft) UGR: Review of Section 4 and 5 (1st draft)
0.9	28.05.2021	Jose Ordonez-Lucena (TID) Pablo Munoz-Luengo (UGR)	TID [editor], UGR, UNIVBRIS: Section 6.1 – 2nd draft UGR: Annex A.
0.91	11.06.2021	Daniel Camps-Mur (I2CAT) Joseph McNamara (LMI) Jose Ordonez-Lucena (TID) Tezcan Cogalan (IDCC)	I2CAT [editor], ACC, UGR, IDCC, TID, PLF: Section 2 – 2nd draft USTRATH, UPC, UGR: Sections 3.2, 3.3, 3.7 – 2nd draft LMI [editor]: Section 4 – 2nd draft TID [editor], UNIVBRIS: Section 6.2 – 2nd draft IDCC [editor], LMI: Section 6.3 – 2nd draft *The 2nd draft of Sections 2, 3 and 4 includes the final round of results (depurated results).*
0.92	18.06.2021	Joseph McNamara (LMI)	IHP, UGR: Sections 3.5, 3.6 – 2nd draft LMI [editor]: Section 5 – 2nd draft *The 2nd draft of Section 5 includes the final round of results (depurated results).*

0.93	25.06.2021	Antonio Garcia (ACC) Vladica Sark (IHP) Jose Ordonez-Lucena (TID)	USTRATH, UNIVBRIS, IDCC: Sections 3.1, 3.4, 3.8 – 2nd draft ACC: Review of Section 6 (1st draft) IHP: Consolidate 2nd draft of Section 3. TID: Integrate Chapter 1 (Introduction), Chapter 7 (Conclusions), and Executive Summary
0.94	02.07.2021	Jose Ordonez-Lucena (TID)	Chapters 1-7 and Annex A integrated into the D4.2 master doc The list of acronyms is updated Cross-reference to Figures/Tables are fixed Bibliography references are updated and fixeed Full review comments are provided
0.95	09.07.2021	Mir Ghoraishi (GIGASYS)	External review of D4.2
0.96	16.07.2021	ALL Jose Ordonez-Lucena (TID)	ALL: Reviewers' comments addressed TID: Produce D4.2, final draft
1.0	23.07.2021	Jesús Gutierrez (IHP) Mir Ghoraishi (GIGASYS)	Final proof-reading and D4.2 final version made available.

Table of Contents

List of Acronyms 13
Executive Summary 16
1 Introduction 18
1.1 Scope of this document 18
1.2 Document structure 19
1.3 On the fulfilment of 5G-CLARITY management plane requirements and KPIs 20
2 Validation of 5G-CLARITY Management Stratum Assets 24
2.1 5G-CLARITY service and slice provisioning subsystem 24
2.1.1 5G-CLARITY service and slice provisioning: initial implementation 25
2.1.1.1 RAN slicing: 5GNR, 4G, Wi-Fi and LiFi 26
2.1.1.1.1 4G and 5GNR slicing – Yang modelling 26
2.1.1.1.2 Wi-Fi slicing – Yang modelling 30
2.1.1.1.3 LiFi slicing – Yang modelling 32
2.1.1.2 Compute slicing: the edge cluster 33
2.1.1.3 Transport slicing 34
2.1.1.4 5G-CLARITY service and slice provisioning: interface design 36
2.1.1.4.1 Service interface from multi-WAT non-RT RIC 36
2.1.1.4.2 Service interface exposed by Slice Manager 40
2.1.2 5G-CLARITY service and slice provisioning: quotas and core network support 42
2.1.2.1 Wireless slicing 42
2.1.2.2 Transport slicing 44
2.1.2.3 Compute slicing: virtual core networks serving different slices 47
2.1.3 5G-CLARITY service and slice provisioning subsystem: preliminary evaluation 50
2.2 5G-CLARITY telemetry 51
2.2.1 Telemetry solutions 51
2.2.1.1 Data Lake 51
2.2.1.2 Data Semantics fabric 52
2.2.2 Telemetry data interfaces 54
2.2.2.1 Comparison of existing interfaces 55
2.2.2.2 Data Semantic Fabric Service Interfaces 57
2.2.2.3 Data Lake Service Interfaces 57
2.2.2.4 Near RT-RIC telemetry 58
2.2.2.4.1 Wi-Fi + LiFi xApp 59
2.2.2.4.2 UE Telemetry xApp 61
2.2.2.4.3 End to end MPTCP xApp 64
2.2.2.4.4 Data Lake xApp 67
3 5G-CLARITY ML Algorithms 68
3.1 Predicting SLA violations/success rate 69
3.1.1 Methodology 70
3.1.2 ESN based time series forecasting 71

3.1.2.1 Input pre-processing 72
3.1.2.2 Predicting the DL aggregated traffic values 73
3.1.2.3 ESN-based time series forecasting performance 75
3.1.2.4 SLA violation prediction from ESN-based time series forecasts 77
3.2 RT-RIC: AT3S traffic routing/handover 79
3.2.1 Dataset 80
3.2.1.1 owcsimpy 80
3.2.1.2 CCTV Emulator 82
3.2.2 Methodology 83
3.2.2.1 Methodology for Objective-1 83
3.2.2.2 Methodology for Objective-2 84
3.2.2.3 Methodology for Objective-3 85
3.2.3 ML Algorithm 85
3.2.3.1 Deep Learning Architecture for Objective-1 85
3.2.3.2 Deep Learning Architecture for Objective 2 85
3.2.3.3 Deep Learning Architecture for Objective 3 86
3.2.4 Results and Discussions 87
3.2.4.1 Objective 1 87
3.2.4.2 Objective 2 88
3.2.4.3 Objective 3 89
3.3 RAN slicing in multi-tenant networks 90
3.3.1 Initial implementation 90
3.3.2 Evaluation methodology 93
3.3.3 Evaluation results 94
3.3.3.1 Case study 1 94
3.3.3.2 Case study 2 98
3.3.3.3 Conclusions and future work 101
3.4 Optimal network access problem 102
3.4.1 Problem statement and formulation 102
3.4.1.1 Multi-WAT Infrastructure 103
3.4.1.2 Service Requests 103
3.4.1.3 Objective function 103
3.4.1.4 QoS constraint 103
3.4.1.5 Multi-Connectivity constraints 104
3.4.1.6 WAT channel constraint 104
3.4.1.7 Access network policy assignment constraint 104
3.4.2 Proposed Solution and Initial Implementation 104
3.4.2.1 Architectures and flows 105
3.4.2.2 Deep Q Learning Algorithm Model 106
3.4.3 Evaluation methodology and Initial Results 106
3.4.3.1 Input data and parameters 107
3.4.3.2 Emulation Model in NS3 108

3.4.3.3 Preliminary results 110
3.5 Indoor ranging with NLoS awareness 111
3.5.1 Algorithm design 111
3.5.1.1 Introduction 111
3.5.1.2 DNN for NLoS-aware Ranging 111
3.5.1.2.1 NLoS Identifier 112
3.5.1.2.2 Ranging 112
3.5.2 Evaluation methodology 112
3.5.2.1 SVM evaluation methodology 113
3.5.2.2 DNN evaluation methodology 113
3.5.2.3 Simulation scenario and CSI collection 114
3.5.3 Evaluation results and discussion 115
3.6 Resource partitioning in a multi-technology RAN 116
3.6.1 Initial implementation 116
3.6.1.1 System model 116
3.6.1.2 Proposed method 117
3.6.2 Evaluation methodology 119
3.6.2.1 Description of the scenario 119
3.6.2.2 Evaluation tools 120
3.6.2.3 Description of initial experiments 122
3.6.3 Evaluation results 122
3.6.3.1 Setup 122
3.6.3.2 Preliminary results 123
3.7 Dynamic transport network setup and computing resources provisioning 127
3.7.1 Initial implementation 127
3.7.1.1 System Model 127
3.7.1.2 Problem Statement 128
3.7.1.3 Initial Agent Design 129
3.7.1.4 Practical Issues and Solutions 131
3.7.2 Evaluation methodology 131
3.7.3 Evaluation results 133
3.8 Adaptive AI-based defect-detection in a smart factory 136
3.8.1 Initial implementation 136
3.8.2 Evaluation methodology 137
3.8.3 Evaluation results 138
4 AI Engine 140
4.1 AI Engine technologies 140
4.2 Consolidation of AI engine design – Revision and implementation details 141
4.2.1 Exposed services 141
4.2.2 Containerised ML models 142

4.2.3 ML service registry 143
4.2.4 ML model lifecycle management 144
4.2.4.1 ML model Dockerising 144
4.2.4.2 ML model deployment 144
4.2.4.3 ML model execution 145
4.2.4.4 ML model monitoring 145
4.2.4.5 ML model update 146
4.2.4.6 ML model removal 146
4.3 Functional validation 146
4.3.1 Deployment 147
4.3.2 Update 147
4.3.3 List and model info 147
4.3.4 Execution 148
4.3.5 Removal 149
5 Intent Engine 150
5.1 Consolidation of intent engine design 151
5.1.1 South-bound interfaces 154
5.1.1.1 Slice and Service Provisioning Subsystem 154
5.1.1.2 Data Processing and Management Subsystem 154
5.1.2 North-bound interfaces 154
5.1.3 East-/west-bound interfaces 155
5.1.3.1 Intent Engine → AI engine 155
5.1.3.2 AI Engine → Intent engine 155
5.2 Intent engine – functional validation 155
5.2.1 Intent Dashboard 155
5.2.2 Intent Matching 156
5.3 Target use cases 158
5.3.1 Intent-based slice provisioning 158
5.3.2 Telemetry 159
6 Private-Public Network Integration 161
6.1 Management Capability Exposure – Initial solution design 161
6.1.1 Mediation Function 161
6.1.2.1 UC1: Smart Tourism 166
6.1.2.2 UC2: Industry 4.0 167
6.2.1 Technology solutions 170
6.2.2 Comparative analysis 172
6.2.3 VLAN vs WAN data plane connectivity with 5G-CLARITY framework 173
6.3.1 Splitting AI/ML operation between various network components 175
6.3.2 Distributed telemetry data and AI/ML model 178
6.3.3 Federated Learning 179
7 Conclusions 180
8 Bibliography 181

9 Annex A – MANO federation 185
9.1 MANO federation in ETSI ISG NFV 185
9.2 MANO federation in the research community 186

List of Figures

Figure 2.1. 5G-CLARITY management and orchestration stratum from D4.1 [1] 24
Figure 2.2. Example of multiple 5G-CLARITY slices provisioned over a common infrastructure 25
Figure 2.3. Accelleran dRAX 4G/5G 27
Figure 2.4. 4G eNB YANG model 28
Figure 2.5. 5GNR CU-UP, CU-CP YANG model 29
Figure 2.6. Flexible scenarios based on multi-AMF/multi-UPF configuration 30
Figure 2.7. I2CAT Wi-Fi box model 30
Figure 2.8. UML representation of wireless.yang used to manage 802.11 interfaces in Wi-Fi APs 31
Figure 2.9. UML representation of wired.yang used to manage Ethernet interfaces in Wi-Fi APs 32
Figure 2.10. LiFi YANG model 33
Figure 2.11. 5G-CLARITY compute slicing approach 34
Figure 2.12. Transport slicing using standard Ethernet switches (only one switch is shown for simplicity) 35
Figure 2.13. Architecture of 5G-CLARITY slice and service support system [1] 37
Figure 2.14. Sample call from Slice Manager to non-RT multi-WAT Controller 39
Figure 2.15. Sample radio chunk (left) and compute chunk (right) endpoints available in Slice Manager 41
Figure 2.16. Sample radio service (left) and compute service (right) endpoints available in Slice Manager 41
Figure 2.17. Example of 5G-CLARITY wireless minimum quotas for three 5G-CLARITY wireless NR services in each NR cell, (a) PRB chunk allocation, (b) 5G-CLARITY Wireless Minimum Quotas, (c) UE blocking probability.service (right) endpoints available in Slice Manager 44
Figure 2.18. Scenario considered to illustrate the 5G-CLARITY transport network quotas computation 46
Figure 2.19. Flow rejection probability perceived for each 5G-CLARITY slice given the transport network quotas 46
Figure 2.20. 5G-CLARITY approach to PLMNID-based slicing 48
Figure 2.21. Potential implementation of PLMNID+SNSSAI-based slicing 49
Figure 2.22. Empirical times required to provision the various components of a 5G-CLARITY slice 51
Figure 2.23. Data Lake – initial solution implementation 52
Figure 2.24. Data Semantics fabric – initial solution implementation 53
Figure 2.25. NGSI-LD information model 53
Figure 2.26. Data Flows 54
Figure 2.27. Interfaces between the Data Lake and other 5G-CLARITY architectural components 54
Figure 2.28. S3 end points used by API Gateway 58
Figure 2.29. Telemetry xAPPs executed in dRAX 59
Figure 2.30. Wi-Fi+LiFi telemetry xApp 60
Figure 2.31. Sample of 4G (orange) and Wi-Fi (green) telemetry topics available on the dRAX kafka bus 61
Figure 2.32. UE telemetry sub-system 62
Figure 2.33. Sample JSON output for LTE measurements 63
Figure 2.34. Sample JSON output for 5GNR Stand-Alone mode measurements 63
Figure 2.35. Sample JSON output for 5GNR Non Stand-Alone mode measurements 63
Figure 2.36. Sample JSON output for WI-FI measurements 64
Figure 2.37. Sample JSON output for GPS location 64
Figure 2.38. Example of use of the local MPTCP metrics Python API 66
Figure 2.39. Example of metrics of two subflows pertaining to a MPTCP socket (inode=282035). 66
Figure 2.40. Example scripts for Watchdog file system event handling and file name obtaining. 67
Figure 3.1. SLA Lifecycle and management framework in 5GNR networks. 70
Figure 3.2. The echo state network structure considered for SLA violation/success rate predictions. The solid and dashed lines with arrows indicate the fixed and trainable connections, respectively. 70
Figure 3.3. The architecture of the ESN based SLA violation detection system. 72
Figure 3.4. The aggregated data traffic data in; (a) raw and (b) statistically normalized and smoothed format. 73
Figure 3.5. ESN based time series forecasting methodology; (a) step-by-step, (b) data and resultant prediction vector frames. 74
Figure 3.6. RMSE performance of the designed ESN framework under various parameter choices. 76
Figure 3.7. The RMSE performance of the designed ESN framework for various training set sizes. 77

Figure 3.8. SLA violation detection by ESN based time series predictions; (a) training (black) and validation (red) sets, (b) validation window with both the actual data and the ESN predictions. 78
Figure 3.9. Detected SLA violations; actual (black), predicted by ESN (red). 78
Figure 3.10. A top view of a geometry description, where the black line denotes the path that the user will take based on the RWP model (left). The corresponding frequency responses of the UE that travels following the black line in the left figure (right). 81
Figure 3.11. The dimensions of a human model. 82
Figure 3.12. The flowchart for dataset generation. 82
Figure 3.13. Comparisons between the objects rendered by `owcsimpy` and Unity 3D Game Engine. 82
Figure 3.14. Objectives and methodologies. 83
Figure 3.15. Methodology for Objective 1. 84
Figure 3.16. Ensemble architecture for Objective 1. 85
Figure 3.17. Deep learning architecture for Objective 2. 86
Figure 3.18. Deep learning architecture for Objective-3. 86
Figure 3.19. CDF of MSE of each architecture for Objective 1. 87
Figure 3.20. GRAD-CAM output of VGG16. 88
Figure 3.21. MSE results for Objective 2. 88
Figure 3.22. Prediction results as shown in the red line, and the black line shows the test data (left). The CDF of MSEs for the third objective (right). 89
Figure 3.23. Comparing LiFi RSSI based on the predicted position $\boldsymbol{fpt+1}$ and the end-to-end LiFi RSSI learning $\boldsymbol{fpt+1}$. 89
Figure 3.24. Functional model of the Deep Q Network-based RAN slicing solution 92
Figure 3.25. Offered load of Tenant 1 and Tenant 2 in situation A of case study 1 95
Figure 3.26. Offered load of Tenant 1 and Tenant 2 in situation B of case study 1 96
Figure 3.27. Offered load O(k) vs assigned capacity in Situation A of case study 1 96
Figure 3.28. Offered load O(k) vs assigned capacity in Situation B of case study 1 96
Figure 3.29. CDF of SLA satisfaction of Tenant 1 in Situation B. 97
Figure 3.30. CDF of SLA satisfaction of Tenant 2 in Situation B. 97
Figure 3.31. CDF of system utilisation in Situation B. 97
Figure 3.32. Offered loads of Tenant 1 and 2 during a day considered for the evaluation of case study 2. 99
Figure 3.33. Average reward every 10000 steps during the training for Δ=0.01 and Δt={1,3,5,15} min 99
Figure 3.34. Average reward every 10000 steps during the training for Δ=0.07 and Δt={1,3,5,15} min 100
Figure 3.35. Simulation architecture overview and main flow 105
Figure 3.36. SINR *w* vs RB throughput per MCS 107
Figure 3.37. (a) NS3 Emulation. (b) Monitoring data for further experiment for Wi-Fi (left) and LTE (right). 109
Figure 3.38. Example of results obtained. 110
Figure 3.39. Successful Predicted/Calculated Optimal Network States on between 1 to 100 UEs 110
Figure 3.40. The end-to-end DNN-based model for NLoS-aware ranging. The vector [O_{NLoS}, O_{LoS}] can only take the value [0, 1] or [1, 0] to assure that only one of the arms in the diagram carries out the ranging. 111
Figure 3.41. Architecture of the DNN used for NLoS-aware ranging. 112
Figure 3.42. Layout of the office hall. 114
Figure 3.43. Performance of SVM and DNN -based NLoS identifier. 115
Figure 3.44. Performance comparison of NLoS-aware ranging using DNN-based FP and SVM. 116
Figure 3.45. Industrial scenario layout. 120
Figure 3.46. ML model functionality in 5G-CLARITY architecture. 120
Figure 3.47. Architecture of the critic network. 122
Figure 3.48. eMBB slice agent learning process for a concrete scenario using DQN agent. 123
Figure 3.49. URLLC slice agent learning process for a concrete scenario using DQN agent. 124
Figure 3.50. Training process for different values of the discount factor parameter. 124
Figure 3.51. Training process for different values of the learning rate parameter. 125
Figure 3.52. Agent convergence with and without pretrained model. 125
Figure 3.53. eMBB agent operation. 126
Figure 3.54. URLLC agent operation. 126

Figure 3.55. Asynchronous TSN backhaul network interconnecting two gNBs with the edge cluster. There are two 5G-CLARITY slices whose respective virtualized UPFs are hosted on the edge cluster. There is a segregated VLAN per slice in the backhaul network. 128
Figure 3.56. Computational complexity to find the optimal configuration (e.g., 5G-CLARITY slices prioritization and per-link capacity reservation) versus the number of links in the ATS-based transport network. 130
Figure 3.57. Computational complexity to find the optimal configuration (e.g., 5G-CLARITY slices prioritization and per-link capacity reservation) versus the number of 5G-CLARITY slices in the ATS-based transport network. 131
Figure 3.58. Scenario considered to test the DRL-assisted solution for configuring the ATS-based transport network. 132
Figure 3.59. Critic network design used for the DQN agent used for the configuration of the ATS-based transport network. 132
Figure 3.60. DQN agent learning process for finding prioritization of four 5G-CLARITY slices at a given ATS. 133
Figure 3.61. DQN agent learning process for finding the prioritization of eight 5G-CLARITY slices at a given ATS. 134
Figure 3.62. Worst-case delay experienced and E2E delay budgets for every 5G-CLARITY slice. 136
Figure 3.63. AI-based defect-detection implementation setup 137
Figure 3.64. Cubes with stickers on used to mimic production items on the conveyor belt 137
Figure 3.65. Used DNN architecture. 138
Figure 4.1. Overview of OpenFaaS concepts (source: [68]) 140
Figure 4.2. AI Engine tech stack, with infrastructure, application and machine learning model execution layers 141
Figure 4.3. Output of the `faas-cli` list command 143
Figure 4.4. ML service registry powered by OpenFaaS, showing several deployed ML models whose functionalities can be tested through the OpenFaaS GUI. 143
Figure 4.5. Function deployment through the OpenFaaS GUI 144
Figure 4.6. Monitoring of metrics related to an ML model with the name “mymlmodel”. Visualisation dashboard provided by Grafana. 145
Figure 4.7. Deletion of a deployed function through the OpenFaaS GUI 146
Figure 5.1. Architectural overview of the intelligence stratum 151
Figure 5.2. Intent Engine concept 153
Figure 5.3. Intent language. JSON (left) and YAML (right) 154
Figure 5.4. Example of APEX output information used to deploy ML module 155
Figure 5.5. Intent dashboard 156
Figure 5.6. Intent matching using ML/NLP 157
Figure 5.7. MySliceManager Mockup 158
Figure 5.8: Intent Engine Slice Manager Use Case 159
Figure 5.9: MyDataLake Mockup 160
Figure 5.10: Intent Engine Data Lake Use Case 160
Figure 6.1. 5G-CLARITY mediation function 161
Figure 6.2. 5G-CLARITY Mediation Function - user story 162
Figure 6.3. Access token 163
Figure 6.4. NFVIaaS-related operation through the Mediation Function (MED-F). 165
Figure 6.5. NFVIaaS in UC1. 167
Figure 6.6. WATaaS and SLaaS in UC1. 167
Figure 6.7. NFVIaaS in UC2. 168
Figure 6.8. The composite NS for supporting delay-sensitive services in UC2 169
Figure 6.9. MANO federation using OSM (source: [73]). 170
Figure 6.10. Illustrative Example of a VLAN-WAN point-to-point connectivity of two 5G-CLARITY slices 173
Figure 6.11. AI/ML inference splitting modes over endpoints [75] 178
Figure 6.12. AI/ML model selection and downloading process [75]. 179
Figure 9.1. Example of multiple administrative domains providing a composite NS [77]. 185
Figure 9.2. Architectural framework proposed in [78] for multi-domain orchestration. 187
Figure 9.3. The 5G-TRANSFORMER service orchestrator (SO) software architecture [81]. 187

List of Tables

Table 1-1: Management and Orchestration Stratum - Functional Requirements and KPIs 20
Table 1-2: Intelligence Stratum - Functional Requirements and KPIs 22
Table 2-1. Multi-WAT non-RT RIC Services 37
Table 2-2. Slice Manager Services 40
Table 2-3. Traffic Characteristics for Slices Used to Compute the Transport Network Quotas 46
Table 2-4. Transport network quotas for each 5G-CLARITY5G-CLARITY slice 46
Table 2-5. Maximum Delay Experienced by Any Packet of Each 5G-CLARITY Slice Given the Transport Network 47
Table 2-6. Slice Configuration 48
Table 2-7. Existing Interfaces and Their Properties 55
Table 2-8. 5G-CLARITY Interfaces for the Exchange of Telemetry Data. 56
Table 2-9. Data Semantic Fabric Services 57
Table 2-10. Data Lake Services 57
Table 2-11. Wi-Fi Related Topics Available in dRAX 61
Table 2-12. Main Measurements Obtained by the UE Telemetric Approach (in JSON format) 62
Table 3-1. ML Use Case Execution Times and Deployment Location. 69
Table 3-2. Adopted Generic Simulation Parameters for our ESN Simulations. 75
Table 3-3. MSE of VGG16 and MobileNet-v2 for Objective 1 87
Table 3-4. Statistics comparisons of MSE results for Objectives 1 and 2 88
Table 3-5. Statistics comparison of MSE results for all objectives. 89
Table 3-6 Scenario Parameters for Case Study 1 94
Table 3-7 Training Parameters for Case Study 1 95
Table 3-8. Scenario Parameters for Case Study 2 98
Table 3-9. Training Parameters for Case Study 2 98
Table 3-10. Average Reward for Different Configurations 100
Table 3-11. SLA Satisfaction and System Utilization for Different Configurations 101
Table 3-12. Maximum and Minimum Throughput per SINR level Wi-Fi, LTE/5GNR 108
Table 3-13. Features Extracted to Feed to SVC/SVR 113
Table 3-14. Hyperparameters of DNNs for NLoS-Aware Ranging. 113
Table 3-15. Parameters of CSI measurement campaign. 114
Table 3-16. Input Parameters for Both Types of Controllers (URLLC, eMBB). 118
Table 3-17. Reward function definition. 119
Table 3-18. Configuration of DQN Agent Hyperparameters. 122
Table 3-19. Primary hyperparameters configuration for the DRL agent used to configure the ATS-based transport network. 132
Table 3-20. 5G-CLARITY slice traffic demand characteristics and delay/jitter onstraint 134
Table 3-21. Links utilizations in the scenario depicted in Figure 3.58 134
Table 3-22. 5G-CLARITY slices prioritization, delay budget and worst-case delay for link #1 (see Figure 3.62). 135
Table 3-23. 5G-CLARITY slices prioritization, delay budget and worst-case delay for link #2 (see Figure 3.62). 135
Table 3-24. 5G-CLARITY slices prioritization, delay budget and worst-case delay for link #3 (see Figure 3.62). 135
Table 3-25. 5G-CLARITY slices prioritization, delay budget and worst-case delay for links #4 and #5 (see Figure 3.62). 135
Table 3-26. Latency and energy consumption performance results for object-detection 139
Table 4-1. AI Engine Services. 141
Table 5-1. Intent Engine Services. 150
Table 6-1. Example of 5G-CLARITY Audit Trail 164
Table 6-2. Data Networking Services for Data Plane Connectivity in PNI-NPN Scenarios 172
Table 6-3. Modes for Split AI/ML Operations Between Device and Network [75]. 175

List of Acronyms

3GPP	3rd Generation Partnership Project
5GNR	5G New Radio
AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Access Point
ATS	Asynchronous Traffic Shaper
B5G	Beyond 5G
BSS	Basic Service Set
BSSID	Basic Service Set Identifier
CCTV	Closed Circuit TV
CDF	Cumulative Density Function
CIR	Channel Impulse Response
CNC	Centralized Network Controller
CNN	Convolutional Neural Network
CoS	Class of Service
CPU	Central Processing Unit
CSI	Channel State Information
CU	(gNB) Centralized Unit
CU-CP	Centralized Unit – Control Plane
CU-UP	Centralized Unit – User Plane
CUC	Centralized User Configuration
DNN	Deep Neutral Network
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EPL	Ethernet Private Line
ESN	Echo State Network
EVPN	Ethernet VPN
FaaS	Function as a Service
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FQDN	Fully Qualified Domain Name
FRER	Frame Replication and Elimination for Reliability
GCL	Gate Control List
GPU	Graphical Processing Unit
GST	Generic network Slice Template
IPV	Internal Priority Value
LSTM	Long Short Term memory
LW	Levy-Walk
MARL	Multi Agent Reinforcement Learning
MCS	Modulation and Coding Scheme
MF	Management Function

MILP	Mixed-Integer Lineal Programming (MILP)
ML	Machine Learning
MNO	Mobile Network Operator
MOCN	Multi Operator Core Network
MPTCP	MultiPath Transmission Control Protocol
MSE	Mean Square Error
MTU	Maximum Trasnmission Unit
MVNO	Mobile Virtual Network Operator
NEF	Network Exposure Function
NFV	Network Functions Virtualization
NFVO	NFV Orchestrator
NLoS	Non-Line-of-Sight
NPN	Non-Public Network
NS	Network Service
PLC	Programmable Logic Controller
PLR	Packet Loss Rate
PM	Performance Measurement
PNF	Physical Network Function
PNI-NPN	Public Network Integrated Non-Public Network
PoP	Point of Presence
PRB	Physical Radio Block
QoE	Quality of Experience
QT	Queueing Theory
RBF	Radial Basis Function
RD	Random Direction
RIC	Radio Intelligent Controller
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RSRP	Received Strength Reference Power
RSSI	Received Signal Strenght Indicator
RSSNR	Received Signal Signal to Noise Ratio
RWP	Random Waypoint
S-NSSAI	Single Network Slice Selection Assistance Information
SBC	Single Board Computer
SBMA	Service Based Management Architecture
SD-WAN	Software Defined WAN
SDN	Software Defined Networking
SDO	Standards Development Organization
SLA	Service Level Agreement
SLO	Service Level Objective
SSID	Service Set Identifier
SSO	Single Sign-On

SVC	SVM Classifier
SVDR	SLA Violation Detection Ratio
SVM	Support Vector Machine
SVR	SVM Regressor
TN	Transport Network
TSN	Time Sensitive Network
UDR	Unified Data Repository
VDU	Virtual Deployment Unit
VIM	Virtualized Infrastructure Manager
VLAN	Virtual Local Area Network
VNF	Virtualized Network Function
VPLS	Virtual Private LAN Service
VPN	Virtual Private Network
WWA	Weighted Arithmetic Average
XAI	Explainable Artificial Intelligence
YOLO	You Only Look Once

Executive Summary

This document, 5G-CLARITY D4.2, constitutes the second deliverable of "WP4: Management Plane". It describes an initial implementation and validation of the 5G-CLARITY system artifacts allowing for user-friendly, zero-touch management of services and slices on private network infrastructures. These artifacts are arranged into two 5G-CLARITY system architecture strata: the Management and Orchestration stratum, and the Intelligence stratum.

On the one hand, the **Management and Orchestration stratum** encompasses all the necessary functionality to deploy and operate the different 5G-CLARITY services (and associated resources) throughout their lifetime, from their commissioning to decommissioning. To be compliant with the design principles of OSS solutions in Beyond 5G (B5G) systems - e.g., modularity, scalability, statelessness, extensibility, simplicity and functional abstraction -, this stratum is architected into a set of self-contained management functions that produces/consumes management services via RESTful APIs. The interoperation and communication across these management functions is enabled through a service bus with message routing capabilities, thereby resulting in a Service Based Management Architecture (SBMA). Depending on their intended functionality, the management functions building up the Management and Orchestration stratum are arranged into four subsystems.

- Service and slice provisioning subsystem: VIM, NFVO, Slice Manager, Transport Controller and Multi-WAT non-RT Controller
- Data processing and management subsystem: Data Semantic Fabric and Data Lake
- External access mediation subsystem: Mediation Function
- Cloud native support subsystem: Authentication and Registration Function and Distributed Data Storage

On the other hand, **the Intelligence stratum** is an add-on layer enriching 5G-CLARITY system capabilities with solutions leveraging Artificial Intelligence (AI) and intent based mechanisms. This stratum is composed of two main building blocks.

- AI engine: a containerized execution environment which provides hosting and management of Machine Learning (ML) models, from design phase to run-time phase. The AI engine makes use of data collected from the 5G-CLARITY Management and Orchestration stratum to train and execute ML models.
- Intent engine: it provides a contact to and from the AI engine as well as layer of abstraction towards the consumer of the AI functionalities, typically the private NOP.

The main outcome of 5G-CLARITY D4.1 [1] is a state-of-the-art analysis and initial solution design of the main building blocks in these two strata. In the management and orchestration stratum, the scope has been limited to two subsystems: *i*) *the service and slice management subsystem*, with a functional description of the Slice Manager and the Multi-WAT non-RT Controller and a practical example of their use for the provisioning of a 5G-CLARITY slice across various multiple wireless technologies; *ii) data processing and management subsystem*, where the internal details of Data Semantic Fabric and Data Lake functions have been captured. In the intelligence stratum, both the AI and intent engines have been addressed, not only from architecture perspective, but also from service perspective. Indeed, to illustrate the applicability of these engines into in-scope 5G-CLARITY scenarios, a total of nine ML algorithms and eight intent use cases have been defined.

5G-CLARITY D4.2 provides a validation of the initial solution design provided in 5G-CLARITY D4.1 [1]. The work present in D4.2 captures the progress made in "T4.1: Development of 5G/Wi-Fi/LiFi management platform, including policy language" and "T4.3: AI engine development and learning algorithms, using historical network data for self-learning purpose" in the last nine months, which basically represents the

development activities on the architectural aspects (components and interfaces) and ML algorithms originally designed in D4.1 [1].

In addition, D4.2 also serves to launch "T4.2: Integration with E2E 5G slice framework", which a first solution design of mechanisms enabling the public-private network integration. This integration, based on the interworking and communication between 5G-CLARITY components of external MNO assets, is addressed into this deliverable into throughout three separate workstreams: *i) management capability exposure*, which deals with everything related to the external access mediation subsystem from 5G-CLARITY management and orchestration stratum, including the internal design of mediation function and its use for the enforcement of the service delivery models applicable to the in-project pilots; *ii) public-private network connectivity*, which identifies WAN technology data networking services that can be used for this end, providing a comparative analysis between them in terms of topology, technology, QoS features and cost; *iii) distributed AI,* which assess the implications of partially migrating AI assets from the private premises to the public cloud for the sake of resource efficiency (e.g. data training is resource-demanding, and not always feasible on a 5G-CLARITY site) and performance gains (e.g. enriching AI engine with data collected from both PLMN and private premises).

1 Introduction

5G-CLARITY system provides a rich set of capabilities for private network operation, including *i) on-premise slicing*, for the provisioning of dedicated resource quotas to separate services/tenants on private infrastructures; *ii) multi-WAT real-time telemetry system*, aggregating and processing data for service assurance activities; *iii) ML algorithms*, for supporting autonomous network management; *iv) AI engine*, which deals with the execution and maintenance of ML models; and *v) intent-based networking*, for facilitating customer interaction in private networks. These capabilities are inherent to two 5G-CLARITY architecture strata: **management and orchestration stratum**, dealing with everything about slice provisioning and monitoring; and **intelligence stratum**, which provides necessary AI/ML and intent based artifacts to assist in the slice run-time operation (e.g., assurance).

The present deliverable reports an initial implementation of the above-referred capabilities, building upon the groundwork laid in 5G-CLARITY D4.1 [1], and validates them against in-scope application scenarios. In addition, it provides an initial solution design for public-private network integration feature, which allows a 5G-CLARITY site (private infrastructure) to communicate with the PLMN domain (public infrastructure) for the provisioning of E2E services in Public Network Integrated Non-Public Network (PNI-NPN) scenarios. These scenarios will be a future-proof realization of private 5G networks in the mid and long run, and therefore are also within the scope of 5G-CLARITY project.

1.1 Scope of this document

5G-CLARITY D4.2 is the second deliverable of WP4: Management Plane". It provides implementation details and validation results of the operational capabilities inherent to 5G-CLARITY system, and that were first designed in 5G-CLARITY D4.1 [1]. In addition, 5G-CLARITY D4.2 provides an initial solution design for public-private network integration, with a focus on the applicability of different service delivery models in PNI-NPN scenarios.

5G-CLARITY D4.2 captures outcomes from all WP4 tasks:

- T4.1: Development of 5G/Wi-Fi/LiFi management platform, including policy language. According to the 5G-CLARITY system architecture defined in 5G-CLARITY D2.2 [2], this platform corresponds to two subsystems from the 5G-CLARITY management and orchestration stratum: the *service and slice provisioning subsystem*, and the *data processing and management subsystem*. 5G-CLARITY D4.1 [1] provided a state-of-the-art analysis and initial solution design for these two subsystems. Following this deliverable, activities focused on the development of Slice Manager and multi-WAT near-RT Controller (and their integration with the MANO stack and SDN controller) were launched. 5G-CLARITY D4.2 reports the first implementation and validation results in this regard.
- T4.2: Integration with E2E 5G slice framework. This task aims to address the interworking and communication of 5G-CLARITY components with external MNO assets. To allow for the realization of PNI-NPN scenarios, it is needed to guarantee the availability of solutions for three different levers: *i) management capability exposure*, which deals with everything related to the external access mediation subsystem from 5G-CLARITY management and orchestration stratum, including the internal design of mediation function and its use for the enforcement of the service delivery models applicable to the in-project pilots; *ii) public-private network connectivity*, based on the selection of appropriate WAN technology data networking services enabling network layer connectivity ; and *iii) distributed AI*, which represents the ability to partially migrating AI assets from the private premises to the public cloud, for the sake of resource efficiency and performance gains. 5G-CLARITY D4.2 is the first document that reports on the solution design for these levers.

- T4.3: AI engine development and learning algorithms. According to the 5G-CLARITY system architecture defined in 5G-CLARITY D2.2 [2], this platform corresponds to the 5G-CLARITY intelligence stratum. 5G-CLARITY D4.1 [1]provided a state-of-the-art analysis and initial solution design of the components building up this stratum: AI engine and intent engine. It also illustrates the applicability of these engines into different application scenarios, with the definition of different ML algorithms and intent use cases. Following this deliverable, efforts have been concentrated on the development of AI and intent engines, and the implementation of hosted ML models. 5G-CLARITY D4.2 captures the first results from this work.

The specific objectives of this deliverable are as follows:

- **OBJ-1: Initial implementation of 5G-CLARITY service and slice provisioning system**, with a first release version of Slice Manager and Multi-WAT non-RT Controller (T4.1).
- **OBJ-2: Initial implementation of 5G-CLARITY data processing and management subsystem**, with a first release version of the Data Semantic Fabric and Data Lake (T4.1).
- **OBJ-3: Initial implementation of the AI engine** (T4.1).
- **OBJ-4: Initial implementation of the Intent engine** (T4.1)
- **OBJ-5: Development and evaluation of ML algorithms** (T4.1)
- **OBJ-6: Initial solution design of mechanisms enabling public-private network integration** (T4.2).

1.2 Document structure

The rest of this document is structured as follows:

- Section 2 covers **OBJ-1 and OBJ-2**, with a first release version of following management functions: Slice Manager, Multi-WAT non-RT Controller, Data Semantics Fabric and Data Lake. This section provides implementation details of these management functions (i.e., internal components, built-in models and interfaces across them), together with a functional validation of their capabilities. For this functional validation, a set of application scenarios are specified. This
- Section 3 covers **OBJ-5**, reporting the results of the ML algorithms defined in D4.1. The following aspects are covered for individual algorithms: implementation description (e.g., algorithm architecture, data acquisition & pre-processing, training time, prediction/classification accuracy), evaluation methodology (e.g., methods, scenarios, data sets, KPIs to be assessed) and evaluation results. All these details make this section cover a large part of the deliverable.
- Section 4 covers **OBJ-3**, extending from the initial solution design specified in D4.1. The features developed around ML model hosting and management have been validated and integrated into the first release version of the AI engine.
- Section 5 covers **OBJ-4**, extending from the initial solution design specified in D4.1. The northbound and southbound interfaces have been validated and integrated into the first version of the Intent engine. This section also includes a couple of in-project application scenario to showcase the capabilities and applicability of the Intent Engine.
- Section 6 covers **OBJ-6**, providing an initial solution design for the integration of private network and public network in the 5G-CLARITY ecosystem across all strata, from connectivity layer up to intelligence layer.
- Finally, Section 7 captures the takeaways of this deliverable.

1.3 On the fulfilment of 5G-CLARITY management plane requirements and KPIs

In this section we discuss how the developments presented in this deliverable contribute to the Management and Orchestration stratum requirements and the Intelligence stratum requirements identified in D2.2 [2]. Similarly, we describe how this deliverable contributes to the overall project objective KPIs.

Table 1-1 describes our contribution to the Management and Orchestration stratum requirements and KPIs, and Table 1-2 does the same with the Intelligence stratum requirements and KPIs.

Table 1-1: Management and Orchestration Stratum - Functional Requirements and KPIs

Requirement ID	Requirement Description	Component	Means of Verification [D4.2 section]
CLARITY-MOS-R1	The 5G-CLARITY management and orchestration stratum shall be architected following the Service Based Management Architecture (SBMA) principles, with a set of MFs providing/consuming management services through a service bus.	ALL	Design captured in Figure 2-1 [Section 2]
CLARITY-MOS-R2	The 5G-CLARITY management and orchestration stratum shall allow for the provisioning of 5G-CLARITY resource-facing services (i.e., 5G-CLARITY wireless, compute and transport services).	Service and Slice Provisioning subsystem	Section 2.1 discusses how to configure physical resources to provision 5G-CLARITY slices. Section 2.1.2 presents how to implement resource quotas in wireless, transport and compute.
CLARITY-MOS-R3	The 5G-CLARITY management and orchestration stratum shall keep a resource inventory, with information on the on-premises resources that can be used for the provision of 5G-CLARITY resource-facing services. This includes information on: *i)* the resource capacity of deployed wireless access nodes, including Wi-Fi/LiFi APs and physical gNBs; *ii)* the compute nodes available in the clustered NFVI (RAN cluster and edge cluster), and related computing/storage/networking resources; *iii)* the capacity and topology of deployed transport network.	Slice Manager, multi-WAT non-RT RIC	The Slice Manager and multi-WAT non-RT RIC components described in Section 2.1.1.4 maintain an inventory of the used compute and wireless resources. We leave for future work the definition of mechanisms to track usage of compute resources.
CLARITY-MOS-R4	The 5G-CLARITY management and orchestration stratum shall store a catalog of VxFs/NSDs.	NFVO	Use of OSM RELEASE SEVEN [Section 2.1]
CLARITY-MOS-R5	The 5G-CLARITY management and orchestration stratum shall support to create, retrieve, update and delete VxFDs/NSDs	NFVO	Use of OSM RELEASE SEVEN [Section 2.1]
CLARITY-MOS-R6	The 5G-CLARITY management and orchestration stratum shall allow to create several instances of the same VxF/NFV service.	NFVO	Use of OSM RELEASE SEVEN [Section 2.1]
CLARITY-MOS-R7	The 5G-CLARITY management and orchestration stratum shall allow VxF / NFV service scaling. This scaling includes the scaling-in and scaling-out the resources of deployed VxF / NFV service instances.	NFVO	Use of OSM RELEASE SEVEN [Section 2.1]
CLARITY-MOS-R8	The 5G-CLARITY management and orchestration stratum shall allow for the	Slice Manager	"5G-CLARITY Slice reservation service" and "Service

	provisioning of 5G-CLARITY slices, by defining separate resource quotas when allocating individual 5G-CLARITY resource-facing services.		Instantiation service" available [Section 2.1.1.4.2]. Mechanisms related to wireless, transport and compute quotas described in Section 2.1.2. ML algorithms: on RAN slicing in multi-tenant networks [Section 3.3]; resource partitioning in a multi-technology RAN [Section 3.6]; on dynamic transport network setup and computing resource provisioning [Section 3.7]
CLARITY-MOS-R9	The 5G-CLARITY management and orchestration stratum shall maintain information regarding the mapping between 5G-CLARITY slices, constituent 5G-CLARITY resource-facing services and allocated resources.	Slice Manager	Slice Manager keeps track of compute resources allocated to each 5G-CLARITY slice. Multi-WAT non-RT RIC keeps track of user airtime for WiFi RANs. Additional RATs to be added in the future.
CLARITY-MOS-R10	The 5G-CLARITY management and orchestration stratum shall allow resource elasticity and AI-assisted placement optimization as part of the 5G-CLARITY slice lifecycle management.	Slice Manager	ML algorithms: on optimal network access problem [Section 3.4]; on adaptive AI-based defect-detection in a smart factory [Section 3.8]
CLARITY-MOS-R11	The 5G-CLARITY management and orchestration stratum shall provide means for model-based data aggregation, with the ability to collect and process management data (e.g., performance measurements, fault alarms) from different sources in an automated and scalable manner.	Near-RT RIC	Telemetry data interfaces available [Section 2.2.2.4].
		Data Processing and Management Subsystem	Interfaces for Data Semantic fabric and data lake no available thus far [Section 2.2.2]
CLARITY-MOS-R12	The 5G-CLARITY management and orchestration stratum shall be able to correlate aggregated data with deployed 5G-CLARITY slices and services instances, providing input to the intelligence engine for AI assisted operation of these instances.	Data Processing and Management Subsystem	Interfaces for Data Semantic fabric and data lake no available thus far [Section 2.2.2]
CLARITY-MOS-R13	The 5G-CLARITY management and orchestration stratum shall provide necessary cloud-native capabilities for MF service production/consumption across the entire stratum.	Cloud Native Support Subsystem	Possible in theory but not implemented in our first release since it required re-architecting some of the background assets.
CLARITY-MOS-R14	The 5G-CLARITY management and orchestration stratum shall allow individual 5G-CLARITY customers (e.g. MNOs) to securely access and consume MF services.	Mediation Function	Design of token-based authentication mechanism [Section 6.1]
CLARITY-MOS-R15	The 5G-CLARITY management and orchestration stratum shall provide the means to expose capabilities with appropriate	Mediation Function	Design of API gateway [Section 6.1]

	abstraction levels to individual 5G-CLARITY customers		
CLARITY-MOS-R16	The 5G-CLARITY management and orchestration stratum shall provide isolation among customers' workflows and request	Mediation Function	Design of API gateway [Section 6.1]
KPI	**KPI description**	**Component**	**Means of verification [D4.2 section]**
CLARITY-MOS-KPI1	According to OBJ-TECH-6, the 5G-CLARITY management and orchestration stratum shall provision a service less than 5 minutes, while providing security and isolation to infrastructure and service slices.	Service and Slice Provisioning Subsystem	An initial experiment is performed that delivers times around 1 minute. Figure 2-22 [2.1.3]
CLARITY-MOS-KPI2	According to OBJ-TECH-7, the 5G-CLARITY management and orchestration stratum shall provision an E2E 5G slice integrating compute and transport resources of an MNO in less than 10 minutes	Mediation Function, Slice Manager	Not available thus far. Will be considered in D4.3.

Table 1-2: Intelligence Stratum - Functional Requirements and KPIs

Requirement ID	Description	Component	Means of verification [D4.2 section]
CLARITY-INTS-R1	The 5G-CLARITY intelligence stratum shall leverage machine learning (ML) models to support intelligent management of network functions.	AI Engine	Initial implementation and validation of AI Engine available and demonstrated in Section 4.
CLARITY-INTS-R2	The 5G-CLARITY intelligence stratum shall host ML models and offer them as services that are accessible outside of the intelligence stratum. Consumers of the ML services are either the network operator or other network functions.	AI Engine& Intent Engine	Partial support. In Section 5 we demonstrate how the Intent Engine can make use of services made available by ML model. Similar use could be made by services outside Intelligence stratum.
CLARITY-INTS-R3	The 5G-CLARITY intelligence stratum shall provide a point of access for ML services to consume data from the network and forward recommended configurations to suitable network functions.	AI Engine & Intent Engine	In Section 5.3.2 we define how Intent Engine can access a telemetry flow from the Data Lake and make it available to ML models in the AI Engine
CLARITY-INTS-R4	The 5G-CLARITY intelligence stratum shall provide ML designers a process or interface to manage the lifecycle of ML models, including the deployment as services.	AI Engine	In Section 4.3 we demonstrate how ML models can be provisioned inside the AI Engine making use of OpenFaaS.
CLARITY-INTS-R5	The 5G-CLARITY intelligence stratum shall expose a communication interface towards the end user that simplifies the management of the 5G-CLARITY platform using intents, including intent-based network configuration and intent-based usage of available ML services.	Intent Engine	In Section 5 we demonstrate the Intent Engine. In Section 5.2.2 we describe the Intent Matching component that is used to translate high level intents into actual API calls. In Section 5.3 we describe two example intent use cases.
CLARITY-INTS-R6	The 5G-CLARITY intelligence stratum shall expose an intent management interface	Intent Engine	Section 5.1.2 describes the north bound interface of the

	through which the intent lifecycle can be controlled, including creation and removal.		Intent Engine that enables intent management.
KPI	**KPI description**	**Component**	**Means of Verification [D4.2 section]**
CLARITY-INT-KPI1	According to OBJ-TECH-8, the 5G-CLARITY intelligence stratum shall demonstrate how the AI engine can reduce both manual and semi-automated intervention in at least 2 relevant use cases.	AI Engine and Intent Engine	Partial support. ML algorithms are defined and initially evaluated in Section 3. AI Engine and Intent Engine are demonstrated in Sections 4 and 5. In D4.3 we will focus in demonstrating onboarding of 2 ML models from Section 3 into the AI Engine.

2 Validation of 5G-CLARITY Management Stratum Assets

Figure 2.1 depicts the design of the 5G-CLARITY management and orchestration stratum defined in 5G-CLARITY D2.2 [2] and 5G-CLARITY D4.1 [1]. In this section we present a detailed design and preliminary evaluation of two subsystems of the management and orchestration stratum, namely: *i)* the service and slice provisioning subsystem, discussed in Section 2.1, and *ii)* the data processing subsystem, discussed in Section 2.2.

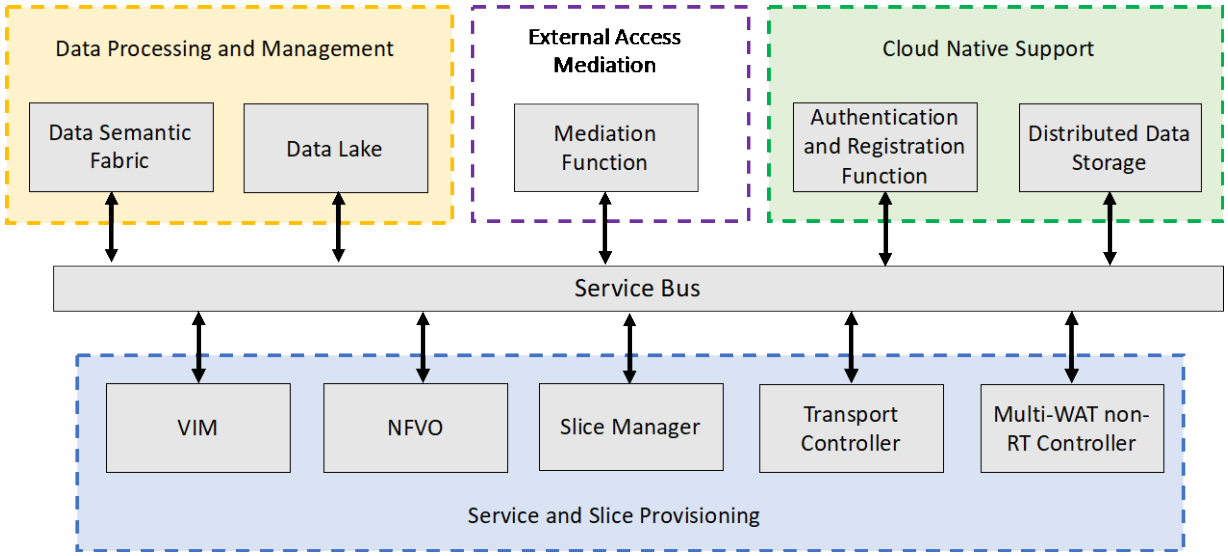


Figure 2.1. 5G-CLARITY management and orchestration stratum from D4.1 [1]

2.1 5G-CLARITY service and slice provisioning subsystem

The 5G-CLARITY service and slice provisioning subsystem, included in the management and orchestration stratum of the 5G-CLARITY architecture (Figure 2.1), allows to provision slices on top of the 5G-CLARITY infrastructure stratum. As introduced in 5G-CLARITY D2.2 [2] and 5G-CLARITY D4.1 [1], 5G-CLARITY slices explicitly consider multi-tenancy to provide infrastructure slicing and deliver isolation among tenants. Notice that this differs from 3GPP slices that take a multi-service approach. Next, we provide an illustrative example that describes the concept of 5G-CLARITY slices. This is built upon the preliminary example provided in 5G-CLARITY D4.1.

Figure 2.2 depicts an example of a network slice deployment over the 5G-CLARITY infrastructure stratum, including: *i)* 5G New Radio (5GNR), Wi-Fi and LiFi access nodes; *ii)* network functions instantiated in the RAN compute cluster, such as the control and user plane functions of the CU; and *iii)* an edge compute cluster with network function virtualization (NFV) functionalities hosting virtual network and application functions. 5G-CLARITY uses the concept of resource chunks to provide isolation among slices. In Figure 2.2, we can see two compute chunks in the edge compute cluster serving two different tenants. Each chunk is composed of specific compute, storage and memory resources. Isolation in the Ethernet transport domain is achieved using virtual local area networks (VLANs), whereas in the wireless domain isolation is achieved through technology specific quotas consisting of physical resource blocks (PRBs) for 5GNR, airtime for Wi-Fi, and a combination of airtime and wavelength for LiFi.

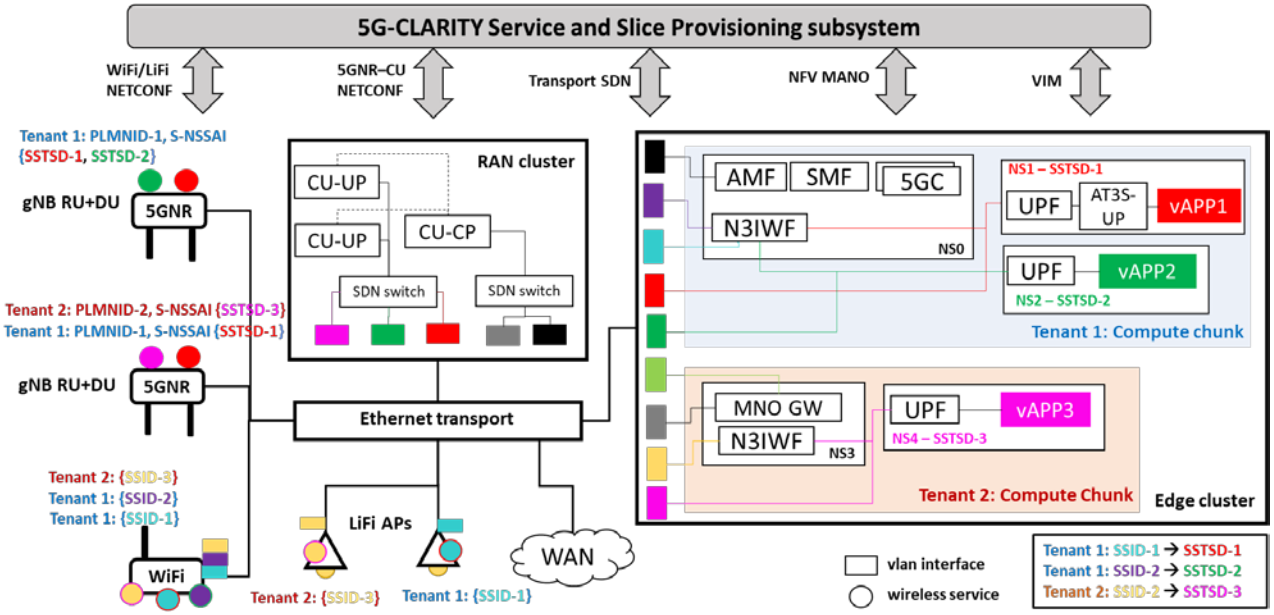


Figure 2.2. Example of multiple 5G-CLARITY slices provisioned over a common infrastructure

Once resource chunks are allocated to a slice, one or more network services can be instantiated providing communication services. In Figure 2.2, we see three network services in the first compute chunk and two in the second one. Network Service 1 (NS1), embedded within the tenant 1 compute chunk, provides access to an application function depicted in red. This network service is advertised using the PLMN identifier (PLMNID) and slice service type – slice differentiator (SSTSD) pair (PLMNID-1, SSTSD-1) in the 5GNR nodes, and using the service set identifier (SSID) SSID1 in the Wi-Fi and LiFi nodes. Note that SSTSDs cannot be advertised by the Wi-Fi/LiFi APs, hence we map each SSTSD into a specific SSID. Traffic connecting to the (PLMNID-1, SSTSD-1) service in the 5GNR nodes is delivered to the red VLAN in the transport domain, which connects to a UPF function deployed in the tenant 1 compute chunk. A separate VLAN (cyan) is used to deliver the Wi-Fi and LiFi traffic to the non-3GPP interworking function (N3IWF) function in the same compute chunk. In addition, an AT3S user plane function sits behind the UPF to allow UEs featuring both Wi-Fi and 5GNR interfaces to benefit from the 5G-CLARITY multi-connectivity framework. Within the same compute chunk a second network service NS2 is advertised using (PLMNID-1, SSTSD-2) in the 5GNR nodes and SSID-2 in the Wi-Fi and LiFi nodes. Notice that both network services are associated to the same 5G core (NS0) and therefore use the same PLMNID. The second compute chunk embeds a basic network service (NS3) including only the user plane functions (N3IWF and UPF), whereas the remaining 5G core functionalities are deployed in the PLMN (not explicitly depicted in Figure 2.2). An additional network service (NS4) sustains the application function depicted in pink, and is advertised using (PLMNID-2, SSTSD-3) and SSID3 in the 5GNR, Wi-Fi and LiFi nodes respectively.

In the next sections the following cases will be described:

i) detailed preliminary implementation of the 5G-CLARITY service and provisioning subsystem,

ii) the way infrastructure isolation can be implemented in different network domains by means of quotas,

iii) a preliminary evaluation of the time required to provision a 5G-CLARITY slice.

2.1.1 5G-CLARITY service and slice provisioning: initial implementation

In this section the initial implementation of the 5G-CLARITY slicing system is introduced which consists of:

- RAN slicing: Where 5GNR, 4G, Wi-Fi and LiFi Physical Network Functions (PNFs) are sliced by advertising a slice-specific service identifier and reserving a quota of RAN resources to that service.

- Compute slicing: Where a set of compute resources are allocated to a 5G-CLARITY slice, defined as CPU, RAM and storage quota. Within these set of resources multiple network services can be instantiated, including a virtualized core network.
- Transport slicing: Consisting in the mapping of a RAN service to a transport service with a defined quality of service. In our initial implementation transport slicing is implemented using Ethernet VLANs and is not further discussed in this section.

In addition to discussing our initial implementations in RAN, compute and transport domains, in the final subsection of this chapter we present design details of the interface between the slice manager and the multi-WAT n-RT RIC management functions (MFs) introduced in 5G-CLARITY D4.1 [1], which are the key MFs that implement the 5G-CLARITY service and slice provisioning subsystem.

2.1.1.1 RAN slicing: 5GNR, 4G, Wi-Fi and LiFi

5G-CLARITY features a heterogeneous set of RAN technologies, including:

- 5GNR and 4G small cells and vRAN components provided by Accelleran,
- Custom Wi-Fi APs provided by I2CAT,
- LiFi APs provided by pureLiFi.

To enable RAN slicing the service and slice orchestration subsystem within the 5G-CLARITY management stratum needs to be able to manage and configure these WATs. To enable this, in 5G-CLARITY we have chosen NETCONF [3] as the common management protocol across RAN technologies. Notice that NETCONF is the protocol chosen by O-RAN to implement the O1 interface for service provisioning. Hence, the 5G-CLARITY approach is aligned with O-RAN standards.

Using NETCONF, a customized YANG data model [4] is required for each RAN technology. The developed YANG models for the aforementioned technologies are described in the next section.

2.1.1.1.1 4G and 5GNR slicing – Yang modelling

Accelleran cloud native dRAX solution shown in Figure 2.3 supports both 4G and 5GNR standalone disaggregated-architectures with Control User Plane Split (CUPS) separation incorporating the control of a cluster of combined 4G DU/RUs such as the ones based on Accelleran E1000 series small cell products, and a cluster of combined or fully disaggregated DU and RU from the O-RAN ecosystem using standardised F1 and Fronthaul 7.2 interfaces.

Within 4G scope, dRAX supports Multi-Operator Core Network (MOCN) and network sharing scenarios with up to 6 different PLMNIDs as per 3GPP specifications. In 5G context, dRAX supports MOCN and network slicing scenarios with up to 12 different PLMNIDs as per 3GPP specifications. While the CU-CP model holds the PLMNIDs supported and sent using SIB1, the CU-UP model holds the Single Network Slice Selection Assistance Information (S-NSSAIs) supported in each PLMNID. Unlike in the 4G case, a key difference between LTE Release 9 and 5G Release 15 is that in 5G each operator sharing a cell can have its own cell identity and tracking area code for that cell, whereas LTE would not be able to support that.

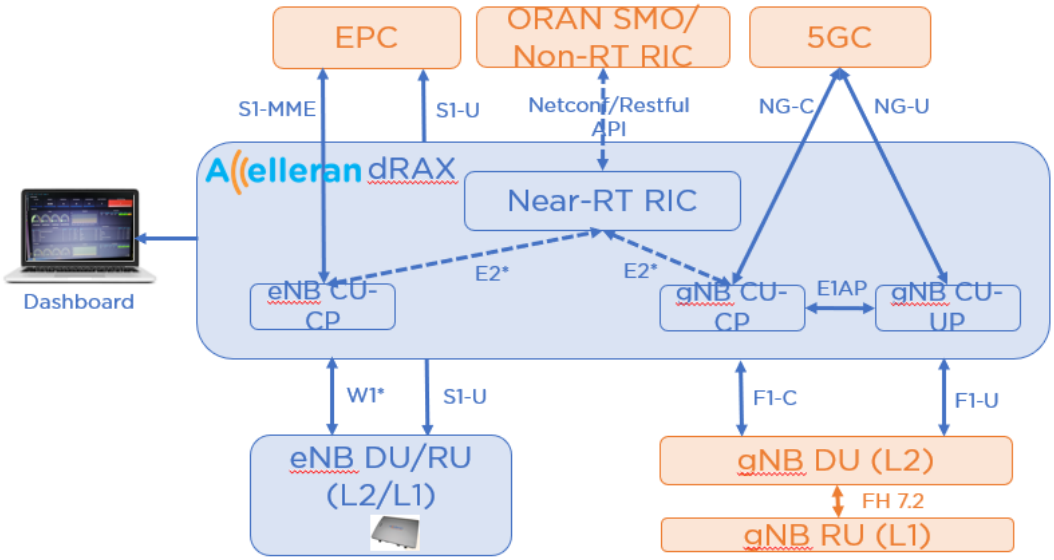


Figure 2.3. Accelleran dRAX 4G/5G

Figure 2.4 shows the 4G YANG model definition for the configuration of the 4G E1000 series small cells managed via dRAX RIC and CU CP function. The primary and secondary PLMN configuration in the model enables the support of a Network Node Selection Function (NNSF) in the dRAX CU CP microservice to enable MOCN and connectivity to different EPC instances over S1-MME based on the PLMNID used by the UE to attach to the network. The configuration of the S1-U tunnels in the model at RAN level towards specific S-GW instances (IP addresses) is not necessary, since this information is provided dynamically by the MME to the dRAX CU-CP using the S1AP interface. This configuration is passed on by dRAX CU-CP to the 4G E1000 series small cells in order to establish direct GTP-U tunnels towards the different involved S-GWs (nominally up to 6 different S-GWs with a maximum of 12 during S1 handovers procedures).

Figure 2.5 shows the preliminary 5G YANG CU model definition for the configuration of the 5G CU CP and CU UP functionality inside dRAX together with its n-RT RIC. The NNSF is done at the dRAX CU CP level enabling MOCN of up to 12 PLMNs. The association between CU-CP and CU-UP instances is flexible to enable scalability and isolation of the resources that can be dedicated to different PLMNs and/or S-NSSAI combinations.

The definition of the preliminary 5GNR YANG data model and the associated functionality enable multi-AMF and multi-UPF connectivity to implement MOCN and slicing approaches and scenarios. A single CU-UP can provide support for multiple operators. For UP resource scalability and slice isolation different CU-UP instances can be spawned and connected to the same CU-CP instance. A particular CU UP instance can be uniquely associated to a particular PLMN/SST pair or to a particular PLMN with all the supported SSTs within. Figure 2.6 represents some possible configuration scenarios that could be flexibly supported.

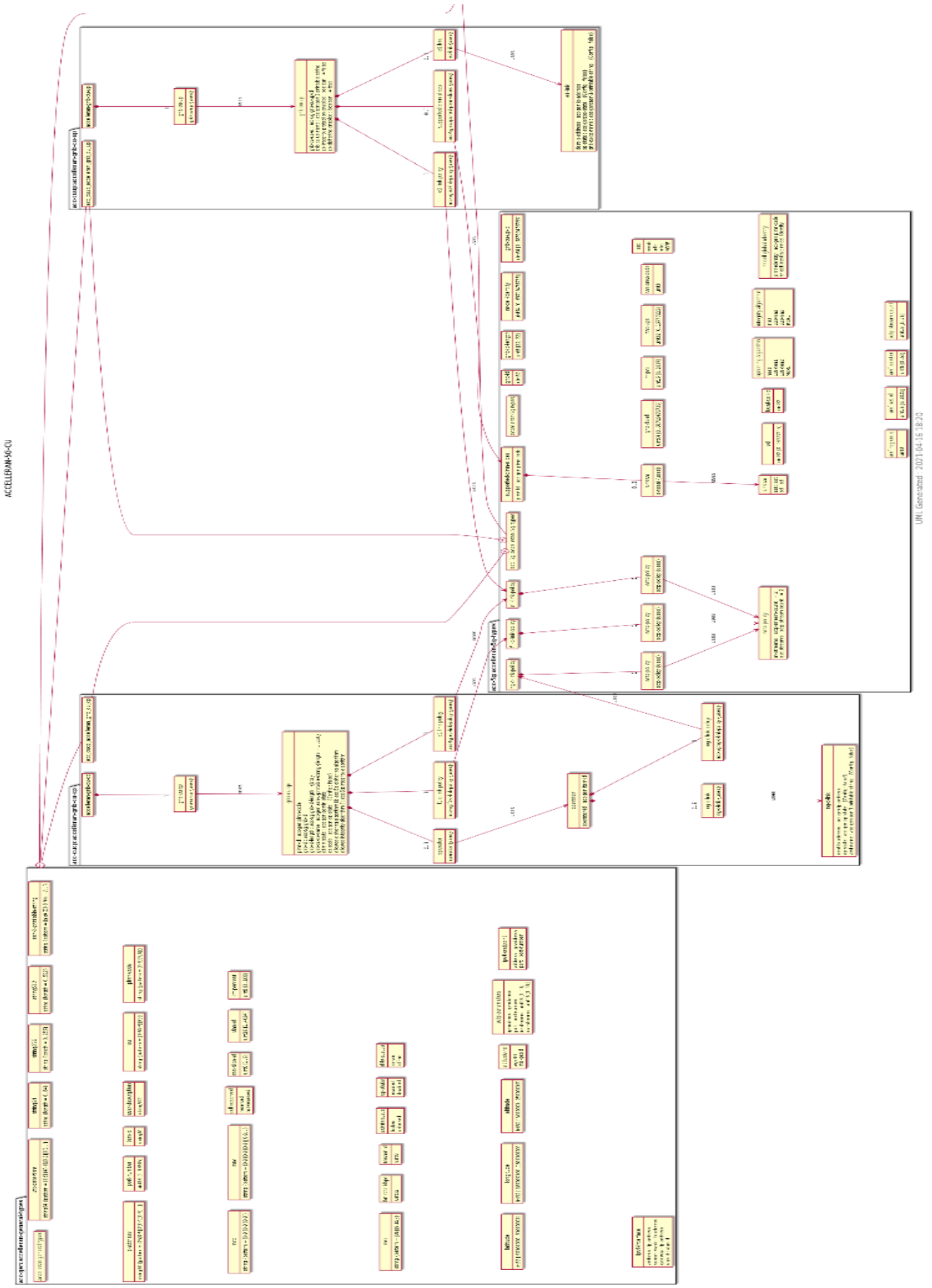


Figure 2.4. 4G eNB YANG model

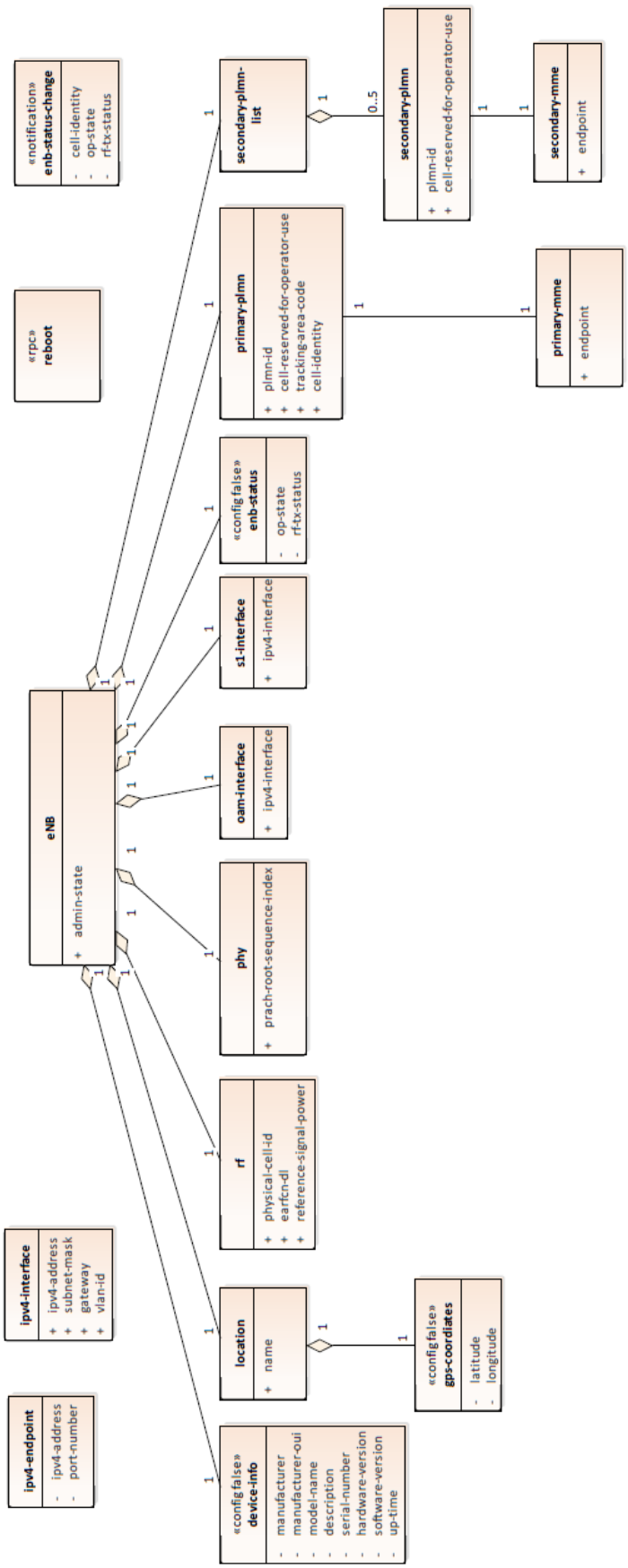


Figure 2.5. 5GNR CU-UP, CU-CP YANG model

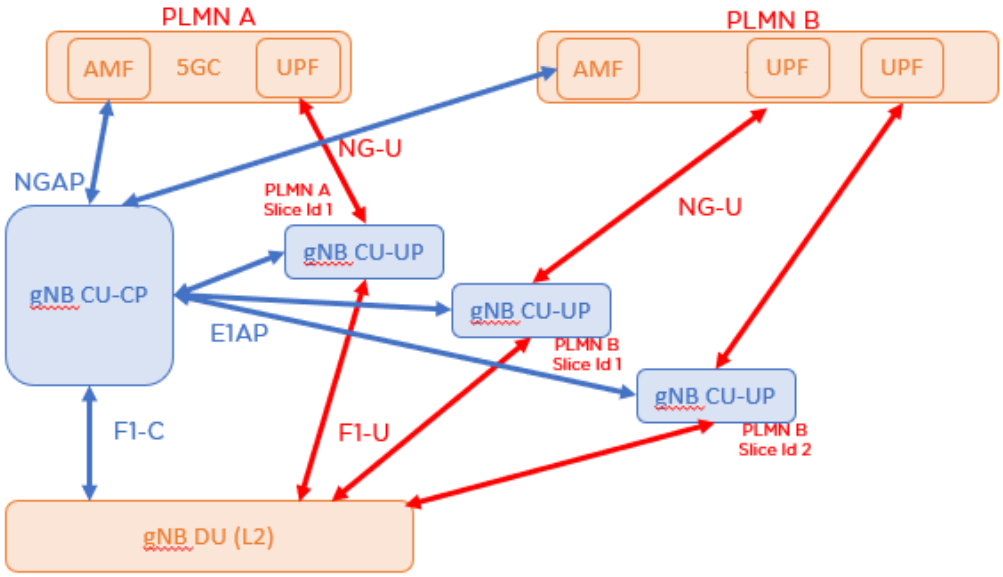


Figure 2.6. Flexible scenarios based on multi-AMF/multi-UPF configuration

2.1.1.1.2 Wi-Fi slicing – Yang modelling

The I2CAT Wi-Fi box was introduced in 5G-CLARITY D3.1 [5], Section 6.2, and is described in Figure 2.7.

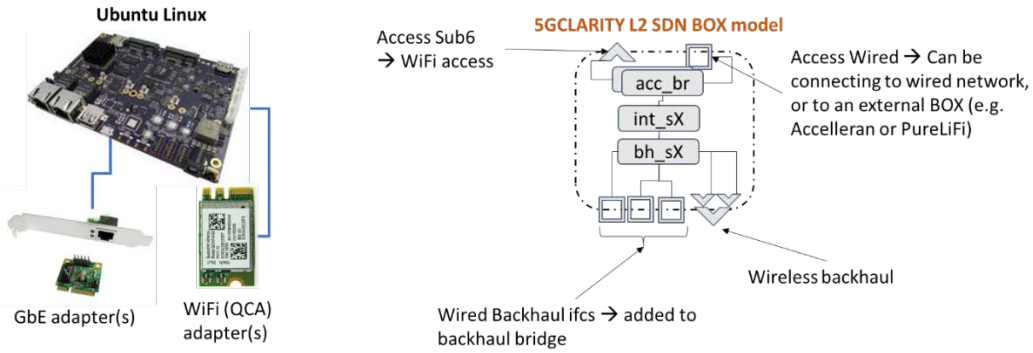


Figure 2.7. I2CAT Wi-Fi box model

The I2CAT Wi-Fi box is composed of a Linux Single Board Computer (SBC) with multiple wireless modems and wired interfaces, connected through a set of dynamically provisioned software bridges. The diagram on the right of Figure 2.7 provides an abstract model of this box. Within the box each interface can be modelled in detail since it has its specific attributes and allowed configurations.

Figure 2.8 and Figure 2.9 describe respectively the high-level structure of two YANG models used by the 5G-CLARITY slice and service provisioning subsystem to manage I2CAT Wi-Fi boxes. The first model, "wireless.yang", is used to manage the multiple wireless (Wi-Fi) interfaces available in an I2CAT Wi-Fi box, whereas the second model, "wired.yang", is used to manage the wired interfaces.

Looking at the "wireless.yang" model in Figure 2.8 we can see that it exposes a list of wireless interfaces, where for each interface we can set configuration parameters, access the current configured state, and importantly create virtual interfaces, where each virtual interface represents a dedicated AP specific SSID, access credentials and QoS parameters. Looking at Figure 2.9 the "wired.yang" model we can see a list of wired (ethernet) interfaces in the I2CAT Wi-Fi box, where for each interface we can set configuration parameters, including the creation of a VLAN, and access the current state.

The models depicted in Figure 2.8 and Figure 2.9 enable the 5G-CLARITY service and slice provisioning subsystem to provision a Wi-Fi service by launching an SSID, creating a VLAN interface, and connecting the SSID to the backhaul VLAN using an *ovsdb* manager that has been released as open source [6].

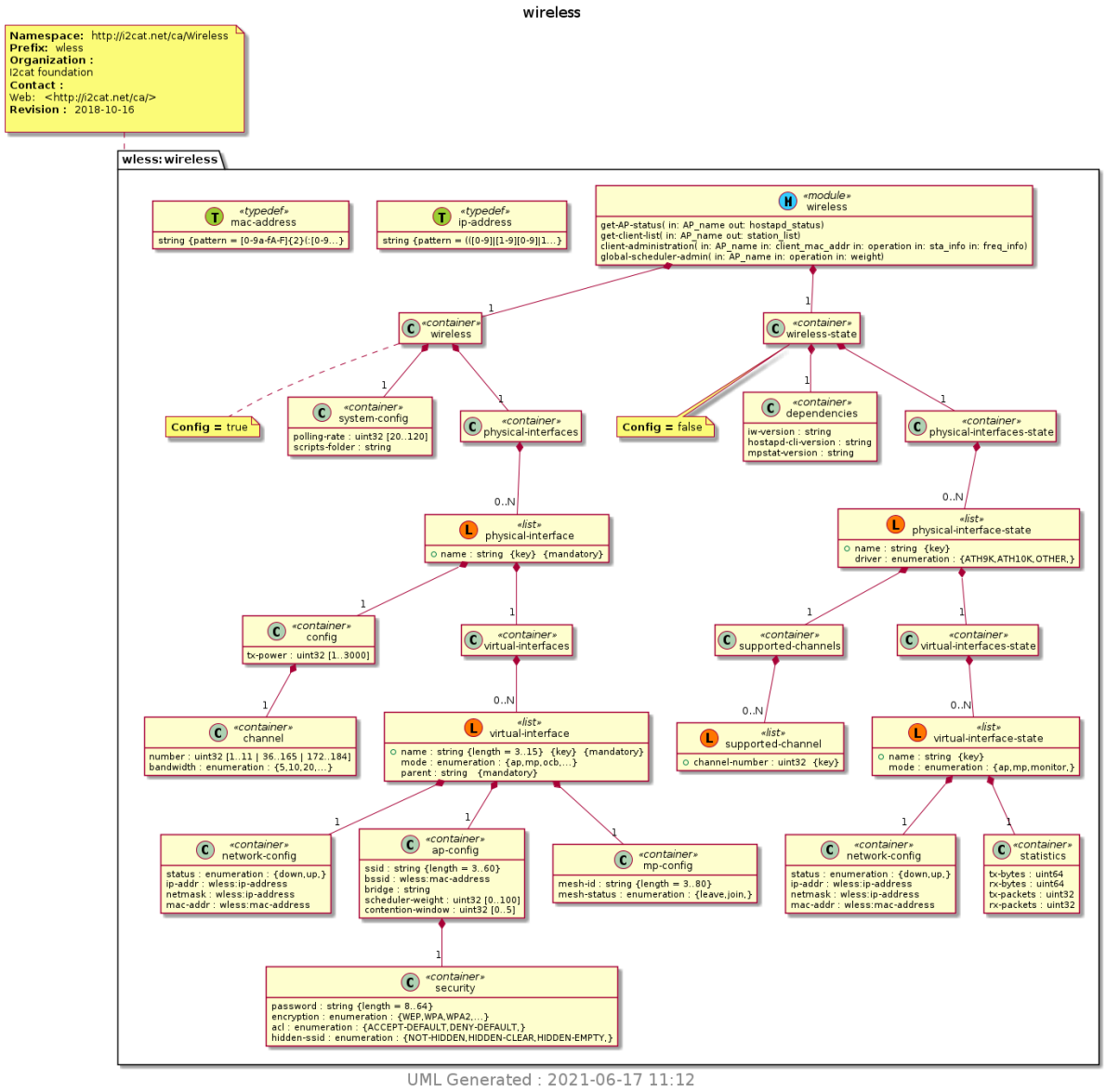


Figure 2.8. UML representation of wireless.yang used to manage 802.11 interfaces in Wi-Fi APs

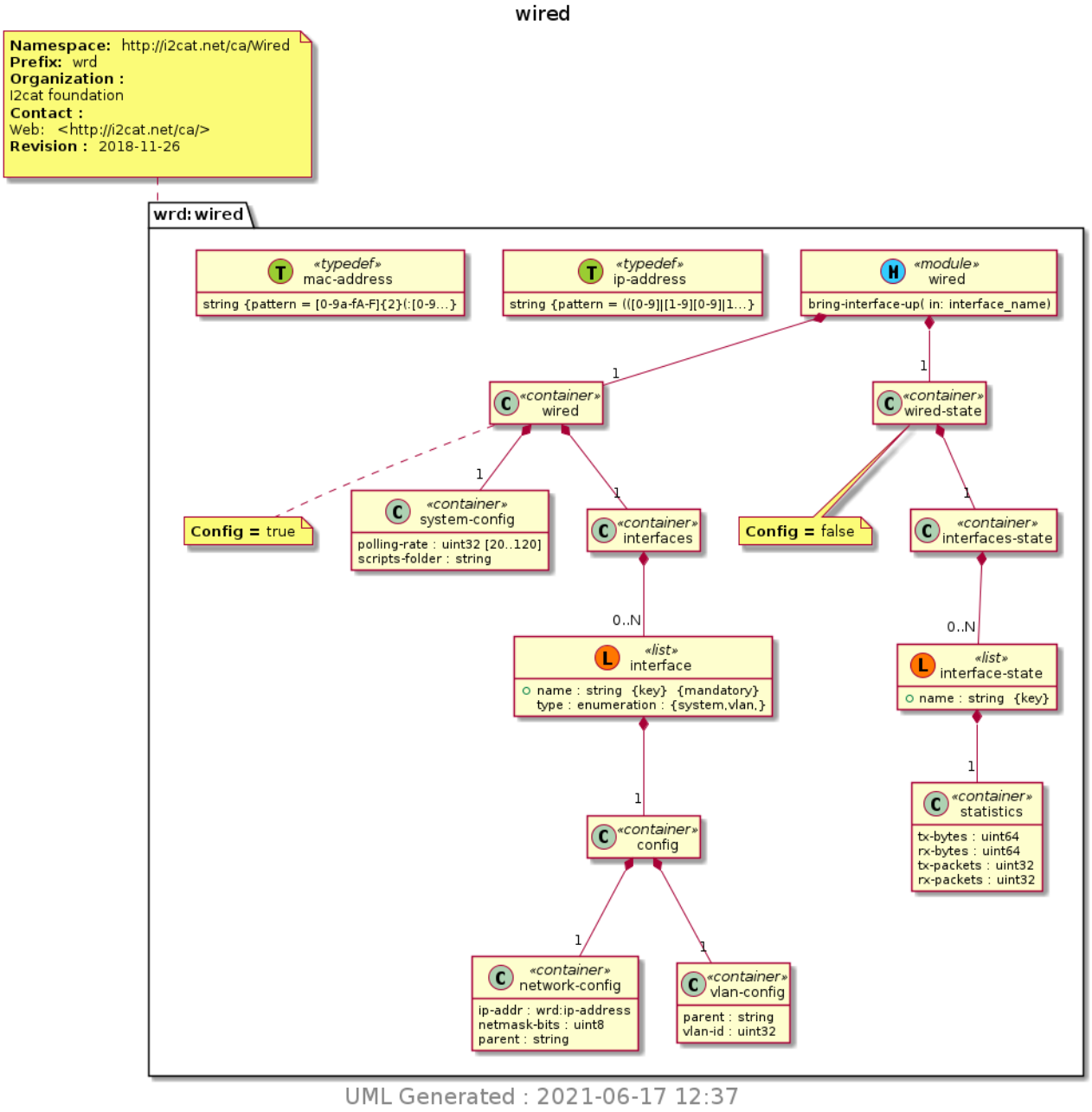


Figure 2.9. UML representation of wired.yang used to manage Ethernet interfaces in Wi-Fi APs

2.1.1.1.3 LiFi slicing – Yang modelling

A LiFi specific YANG model has been developed to allow the 5G-CLARITY service and slice provisioning subsystem to manage pureLiFi APs. Being our choice of LiFi technology based on the same IEEE 802.11 MAC, the LiFi and Wi-Fi models look rather similar. The LiFi yang model is called plf-lifi and is illustrated in Figure 2.9 and Figure 2.10. The model specifies the configuration of LiFi service parameters including SSID, security credentials and VLAN:

- The *lifi* container specifies parameters to configure the LiFi AP including interface parameters such as IP address and network mask.
- The *access-point* container specifies the configurations for a Basic Service Set (BSS) including SSID and encryption type.

The VLAN configuration is also supported in the Yang model to enable LiFi slicing.

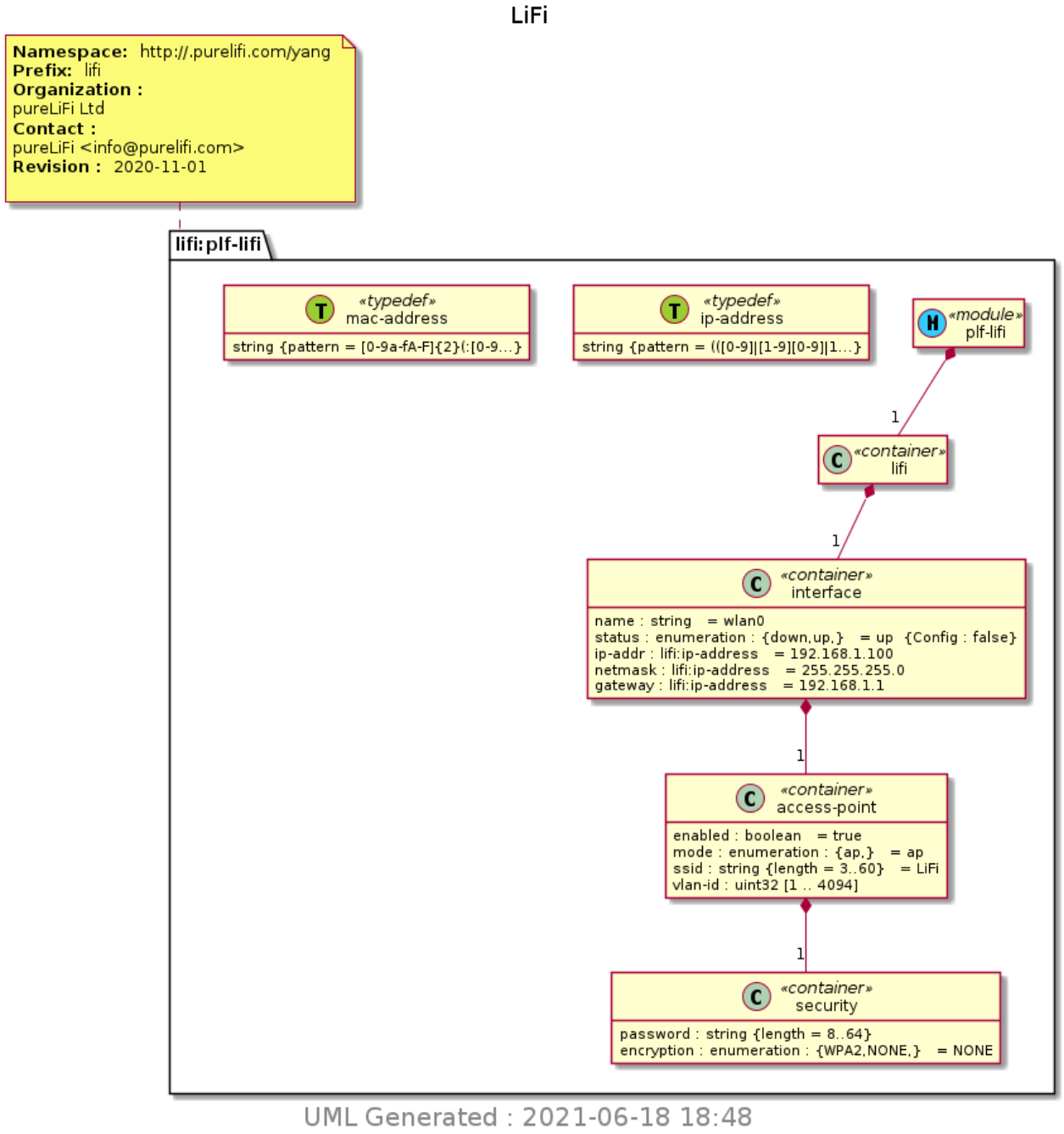


Figure 2.10. LiFi YANG model

2.1.1.2 Compute slicing: the edge cluster

The reference edge compute cluster that is used in 5G-CLARITY is based on OpenStack Ussuri. Cloud native approach based on Kubernetes is also possible within the 5G-CLARITY architecture, but it is decided to choose OpenStack for the following two main reasons. First, preliminary assets developed by 5G-CITY project partners [7] are based on OpenStack. Reusing these assets in 5G-CLARITY simplifies the edge cluster development and allows us to focus development efforts on more innovative areas. Second, OpenStack allows for native L2 networking, whereas Kubernetes default networking plugins are based on building L3 and above networking abstractions [8]. Native L2 networking fits better the 5G-CLARITY slicing approach, where transport slicing is implemented using VLANs. The study on how to integrate a VLAN based transport slicing approach with a cloud native edge compute cluster is left for future works.

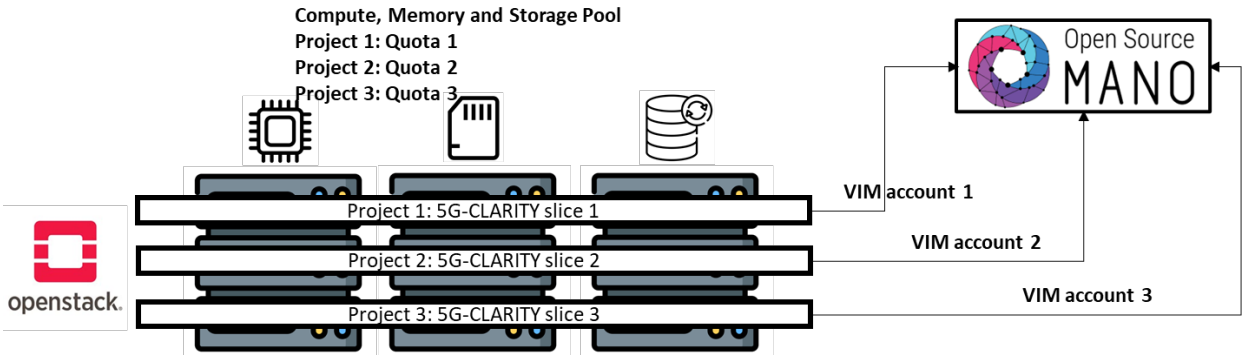


Figure 2.11. 5G-CLARITY compute slicing approach

OpenStack is already a multitenant platform where different tenants can be allocated separate compute (virtual CPUs), storage and memory (RAM) quotas based on the concept of OpenStack projects [9]. Thus, the OpenStack project is the basic OpenStack capability leveraged by the Slice Manager in 5G-CLARITY to allocate separate compute resources to separate slices. Subsequently, multiple OpenStack projects can be attached to an NFV orchestrator that can support different VIM accounts. This is the case of Open Source MANO (OSM) version 7 [10], which is the NFV orchestrator adopted in 5G-CLARITY. Figure 2.11 describes the process used by the 5G-CLARITY slice manager to configure OpenStack projects and attach them to OSM using different VIM accounts.

2.1.1.3 Transport slicing

After configuring wireless and compute devices the service and slice provisioning system configures the transport network to isolate the traffic generated by the RAN and compute slices in the transport network. The underlying transport technology in a 5G-CLARITY system is IEEE 802.1, which can be divided into *i)* standard Ethernet switching; and *(ii)* Ethernet with Time Sensitive Network (TSN) support. Being Ethernet the default transport technology, the natural choice is to use VLAN as the means to isolate 5G-CLARITY slices over the common transport network. How the service and slice provisioning subsystem configures VLANs in the underlaying transport network while considering the default and TSN Ethernet cases is described next.

Figure 2.12 depicts the steps and MFs involved in the provisioning of a 5G-CLARITY slice when using a default Ethernet transport network. The following network management agents are required in the different devices:

- Ethernet switching gear: NETCONF support is required to be able to create VLANs and attach them to a specific interface.
- RAN devices: Assuming these devices run a Linux OS, NETCONF is used that allows to create a VLAN interface and assign an IP address on the configured interface. See for example the “wired.yang” model used in the I2CAT Wi-Fi boxes described in Section 2.1.1.1.2.
- RAN cluster: A Kubernetes based deployment is assumed using the MULTUS CNI [11], which enables to bind a specific interface on the bare metal to a given Kubernetes pod. A NETCONF agent in the bare metal is used to create the VLAN interface that needs to be connected to the pods serving that slice (e.g., a CU-UP pod set up to serve that slice). Despite not being as “VLAN friendly” as OpenStack, Kubernetes is chosen in the RAN cluster since it is aligned to the O-RAN architecture.
- Edge cluster: The OpenStack Neutron agent [12] is used to create a virtual network inside the OpenStack cluster that is connected to an infrastructure VLAN.

Thus, the steps followed by the service and slice provisioning subsystem to configure the transport network upon a slice creation are depicted in Figure 2.12 and described here:

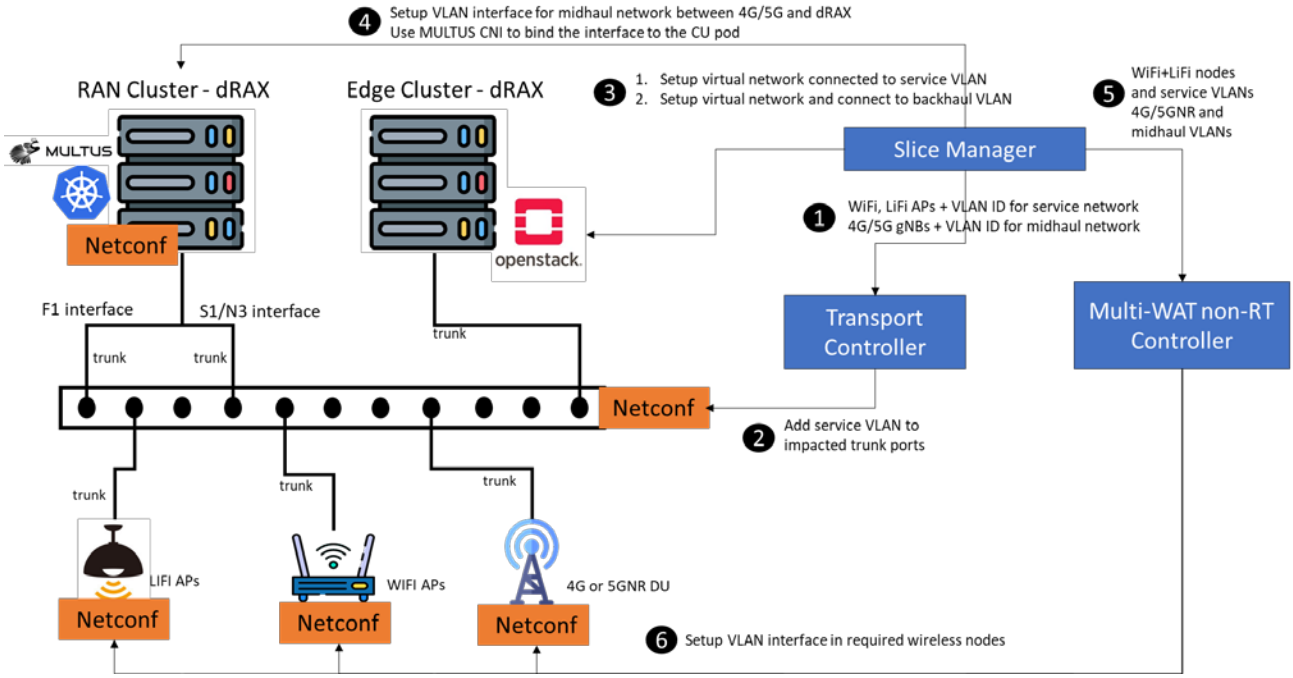


Figure 2.12. Transport slicing using standard Ethernet switches (only one switch is shown for simplicity)

- Steps 1 and 2: The Slice Manager configures the VLANs in the underlaying transport. This is accomplished by having the Slice Manager select the VLANs required for each segment, namely, i) a midhaul VLAN connecting the DU to the RAN cluster node where the CU is provisioned, ii) a backhaul VLAN connecting the CU in the RAN cluster to the UPF in the Edge cluster, and iii) a service VLAN connecting the Wi-Fi and LiFi nodes to the N3IWF in the edge cluster. The Slice Manager communicates these VLANs to the Transport Controller, which implements the NETCONF client to configure the underlaying switches. Notice that the Transport Controller maintains a network topology and understands what devices are connected to each network port, using for example LLDP [13].
- Step 3: The Slice Manager interacts with the edge cluster to create virtual networks corresponding to the service network and the backhaul network for this slice. These virtual networks are connected to the underlying VLANs configured in the previous step.
- Step 4: The Slice Manager interacts with the RAN cluster to create the mid-haul VLAN interfaces required to support this slice, as well as connecting them to the appropriate pod using the MULTUS CNI.
- Steps 5 and 6: As last steps, the Slice Manager interacts with the multi-WAT non-RT controller to deploy the wireless service, providing the backhaul/service VLANs where the wireless services need to be connected to. The multi-WAT non-RT controller includes a NETCONF client used to configure the requested VLAN interfaces in the RAN devices.

When using a TSN transport network, the process of deploying a 5G-CLARITY slice becomes more complex as the slice configurations related to the deterministic QoS provision must be carried out. The steps shown in Figure 2.12 are preserved, but steps 1 and 2 might be further decomposed to reflect these QoS related configurations. Since we consider a fully centralized (SDN-like) architecture for TSN, the transport controller includes the TSN control plane components, namely, Centralized User Configuration (CUC) and Centralized Network Controller (CNC). We also assume that these components are unaware of the 5G-CLARITY slice. Under these considerations, the following additional steps are required to perform the QoS configuration of

the slice:

- The slice manager, playing the roles of listeners/talkers, communicates the streams QoS requirements of the respective 5G-CLARITY slice to the CUC. The CUC interprets these requirements and might combine them with those from other slices.
- The CUC communicates the composed streams requirements to the CNC via a user/network configuration protocol. Then, the CNC performs the computation of the network configuration in order to accommodate the incoming 5G-CLARITY slice, i.e., to allocate the required resources (e.g., per link bandwidth and buffer space at each TSN bridge port), while ensuring its QoS requirements and those of the ongoing 5G-CLARITY slices are met. Note that, depending on the specific algorithm used for computing the configuration, the configuration of the 5G-CLARITY slices previously accommodated might also change. Last, the configuration algorithm might require data analytics to estimate the foreseen traffic matrix and temporal traffic profiles and adapt the configuration accordingly. It also might be assisted by the AI-engine, e.g., the AI-engine executes a ML learning algorithm that assists the master TSN configuration computation algorithm running in the CNC, in order to cope with the configuration complexity of the TSN networks.
- The CNC populates the computed QoS related configuration to the different TSN bridges. More precisely, the output port of each TSN bridge has to be configured. As each 5G-CLARITY slice is uniquely identified by a VLAN ID at a given transport network segment, we consider the CNC provides a configuration per VLAN. For instance, considering a synchronous TSN network based on the Time Aware Shaper (IEEE 802.1Qbv), it is required to configure the Gate Control List (GCL) per output port. This involves splitting the time into windows, each with a specific duration and deciding which VLANs can transmit data at a given time window. Furthermore, the priority level per TSN bridge of each VLAN shall be chosen for configuring the strict priority transmission selection algorithm if applicable. Besides the aforementioned configurations, the per-stream filtering and policing stage includes additional configurations (e.g., traffic regulation and maximum transfer unit).

2.1.1.4 5G-CLARITY service and slice provisioning: interface design

Figure 2.13, first provided in 5G-CLARITY D4.1 [1], depicts the internal architecture of the 5G-CLARITY slice and service support system. In this section, two of the main MFs within this architecture, namely the Slice Manager and the multi-WAT non-RT RIC, are described in more details by looking at the interfaces they offer.

2.1.1.4.1 Service interface from multi-WAT non-RT RIC

The multi-WAT non-RT RIC manages the underlying RAN technologies via NETCONF, as it was described in Section 2.1.1.1. In the north-bound, the multi-WAT non-RT RIC exposes a set of REST end-points that allow other MFs in the slice and service support system to make use of these services.

Table 2-1, introduced in 5G-CLARITY D2.2 [2], describes the services exposed by the multi-WAT non-RT RIC highlighting in green the services that are available at the time of writing this deliverable.

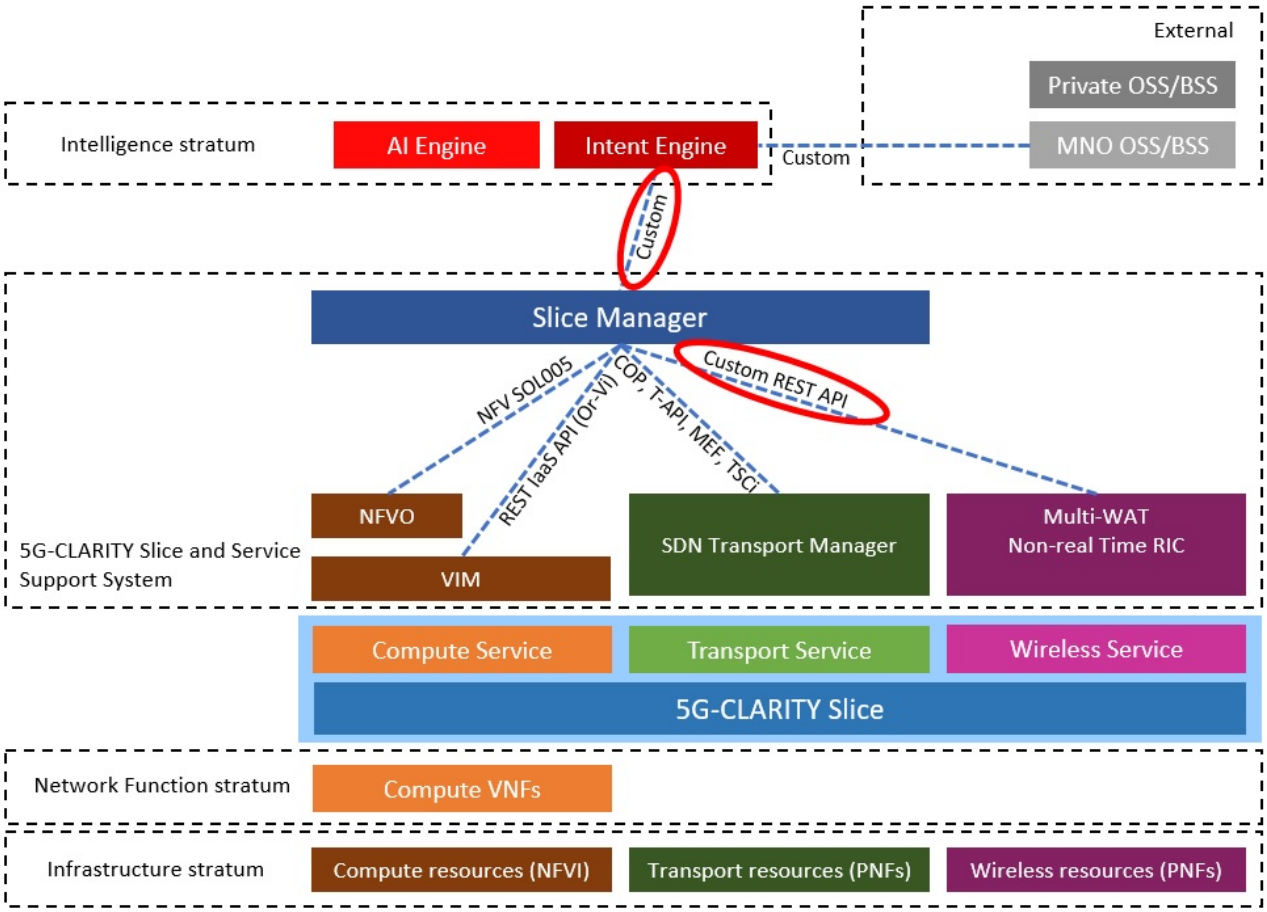


Figure 2.13. Architecture of 5G-CLARITY slice and service support system [1]

Table 2-1. Multi-WAT non-RT RIC Services

MF Service ID	MF Service Name	Description	Reference Specifications
Mwat_Plmn_Lcm	**PLMNID lifecycle management**	This service allows instructing a gNB-DU to radiate a PLMN ID in a set of gNB-DUs under the control of that gNB-CU. In addition, it provisions the IP end-point associated with that PLMN ID	Custom (REST)
Mwat_Snssai_Lcm	**S-NSSAI lifecycle management**	This service allows instructing a gNB-CU to radiate a S-NSSAI in a set of DUs under the control of t gNB-CU	Custom (REST)
Mwat_Wi-Fi_Ssid_Lcm	**Wi-Fi SSID lifecycle management**	This service allows instructing one or more Wi-Fi APs to radiate a SSID with a specific set of security credentials	Custom (REST)
Mwat_Lifi_Ssid_Lcm	**LiFi SSID lifecycle management**	This service allows instructing one or more LiFi APs to radiate a SSID with a specific set of security credentials	Custom (REST)
Mwat_Plmn_Res_Rsrv	**PLMNID resource reservation service**	This service allows allocating a percentage of Physical Resource Blocks (PRBs) in a set of gNB-DU under the control of the target gNB-CU, in order to carry the traffic of a given PLMN ID	Custom (REST)
Mwat_Snssai_R	**S-NSSAI resource**	This service allows allocating a percentage of Physical	Custom (REST)

es_Rsrv	**reservation service**	Resource Blocks (PRBs) in a set of gNB-DUs under the control of the target gNB-CU, in order to carry the traffic of a given S-NSSAI	
Mwat_Wi-Fi_Ssid_Res_Rsrv	**Wi-Fi SSID resource reservation service**	This service allows allocating a percentage of airtime resources in one or more APs to carry the traffic of the provided SSID	Custom (REST)
Mwat_Lifi_Ssid_Res_Rsrv	**LiFi SSID resource reservation service**	This service allows allocating a percentage of airtime resources in one or more APs to carry the traffic of the provided SSID	Custom (REST)
Mwat_5gnr_Cell_Conf	**5GNR Cell configuration service**	This service enables the configuration of a set of 5GNR cells under the control of a gNB-CU including parameters such as carrier frequency, cell identifier, transmission power and neighbour lists	Custom (REST)
Mwat_Wi-Fi_Ap_Conf	**Wi-Fi AP configuration service**	This service enables the configuration of one or more APs including parameters such as the operating channel and bonding mode, the Wi-Fi mode (e.g.VHT, HT, a/b/g) and the transmission power	Custom (REST)
Mwat_Lifi_Ap_Conf	**LiFi AP configuration service**	This service enables the configuration of one or more APs including parameters on device info, WLAN configuration and Lamp configuration	Custom (REST)
Mwat_5gnr_Topo	**5GNR topology service**	This service allows providing 5GNR physical topology including list of cells connected to a given gNB-DU instance, and list of gNB-instances connected to a given gNB-CU instance	Custom (REST)
Mwat_Wi-Fi_Topo	**Wi-Fi topology service**	This service allows providing Wi-Fi topology information including list of physical AP appliances and the capabilities of the physical radios included in each AP	Custom (REST)
Mwat_Lifi_Topo	**LiFi topology service**	This service allows providing a list of the available physical LiFi AP appliances	Custom (REST)
Mwat_Ric_Mgmt	**5GNR rtRIC management service**	This service allows providing a list of rt-RIC instances controlled by the mWAT Non-rt Controller. It enables operations such as policy management, and xAPP management	O-RAN based (REST)
Mwat_Wi-Fi_Ric_Mgmt	**Wi-Fi-LiFi rt controller management service**	This service allows providing a list of Wi-Fi/LiFi real time controllers managed by the mWAT Non-rt Controller. Enables deployment of policies into Wi-Fi/LiFi real time controllers	Custom (REST)
Mwat_Inventory	**Inventory Service**	This service allows returning a data structure containing a list of the currently active wireless services including the service identifier (PLMN ID, S-NSSAI, and SSID), their resource quota and the nodes where the service is active	Custom (REST)

For the sake of space, we will not describe each of the services in detail, but rather focus on a subset of them. Figure 2.14 provides an example of a wireless service definition triggered by the slice manager towards the multi-WAT non-RT RIC. We can see:

- "selectedPhys": populated with information that the Slice Manager obtains through the topology service (not shown here). The field will indicate:
    - List of Wi-Fi radios that are part of this service. Notice that a single Wi-Fi AP can have more than one radio
    - List of LiFi APs that are part of this service
    - List of 4G/5GNR cells that are part of this service
- "wirelessConfig": specifies the configuration of the Wi-Fi and Li-Fi services including SSID and security credentials. The "vlanId" identifies the VLAN that the traffic connecting to this SSID needs to be connected to. In addition, to control resource allocation in Wi-Fi two parameters are supported:
    - The "airtimeWeight" parameter allows to control the percentage of airtime allocated to this service.
    - The "contentionWindowIncrease" field can be used to control the Contention Window size used by devices connecting to this Wi-Fi service. Finally, the "vlanId" identifies the VLAN that the traffic connecting to this SSID needs to be connected to.
- "CellularConfig": specifies the parameters related to the configuration of the cellular service. In Figure 2.14 only PLMN-based slicing is supported, and so the "plmnId" fields indicates the PLMN that needs to be added to the PLMN list of the selected cells. Additional configuration parameters include the "coreAddress" field that indicates the IP address of the virtual core this slice needs to be connected to.


```
{
  "selectedPhys": [
    "0a3dd93a-e527-469d-8c4c-835192d6e6b5", "3f450872-3d22-47c8-a596-e71c45ce9252",
"4930ddd9-985f-41bd-abd9-061d30aac9b7"
  ],
  "vlanId": 1500,
  "selectedRootPhy": "0a3dd93a-e527-469d-8c4c-835192d6e6b5",
  "wirelessConfig": {
    "encryption": "NONE",
    "ssid": "Test",
    "airtimeWeight": 10,
    "contentionWindowIncrease": 1
  },
  "cellularConfig": {
    "plmnId": "00105",
    "amarisoftConfig": {
      "coreIpAddress": "192.168.50.5",
      "corePort": 3333,
      "vlanCore": 1400
    },
    "accelleranConfig": {
      "coreAddress": "192.168.50.6",
      "corePort": 3333
    }
  }
}
```


Figure 2.14. Sample call from Slice Manager to non-RT multi-WAT Controller

Table 2-2. Slice Manager Services

MF Service ID	MF Service Name	Description	Reference Specifications
Sm_Sl_Rsrv	**5G-CLARITY Slice reservation service**	This service receives as input: • Target VIM nodes and compute resource quota • A set of gNB-DUs and PLMN ID to be instantiated • A set of Wi-Fi and LiFi APs and SSIDs to be instantiated Based on this input the Slice Manager configures the resource reservation in the VIM, and instantiates the wireless services in the target WATs	Custom (REST)
Sm_Sl_Actv	**5G-CLARITY Slice activation service**	This service receives as input an NSD identifier, which has been previously on-boarded on to the NFVO, and a 5G-CLARITY slice identifier, and instantiates the target NSD on the VIM resources provisioned for that slice	Custom (REST)
Sm_Srv_Inst	**Service Instantiation**	This service receives as input an NSD identifier, which has been previously on-boarded on to the NFVO, and a 5G-CLARITY slice identifier, and instantiates the target NSD on the VIM resources provisioned for that slice	
Sm_Sl_Inv	**5G-CLARITY Slice inventory**	This service returns a data structure containing the list of slices currently deployed in the system along with the corresponding slice descriptors	Custom (REST)

2.1.1.4.2 Service interface exposed by Slice Manager

The services exposed by the Slice Manager function are examined in this section, where Table 2-2 indicates the Slice Manager services identified in 5G-CLARITY D2.2 [2], highlighting in green the services discussed in this section. Note that service “Sm_Srv_Inst” has been added with respect to D2.2.

The slice reservation service is used to reserve a set of compute and radio resources to be part of that slice. This is made available in the slice manager through the concept of radio and compute “chunks”. An example of these is represented in Figure 2.15, where the following components are observed:

- Radio chunk: Contains a set of physical ineterfaces, which can be of type cellular, Wi-Fi or LiFi.
- Compute chunk: includes a “compute_id” indicating the physical compute node that will be part of the chunk, as well as the resource requirements in terms of CPUs, RAM and storage.

```
{
  "name": "RadioChunk",
  "user_id": "5b63089158f568073093f70d",
  "chunk_topology": {
    "physicalInterfaceList": [
       {
        "id": "f9af122a-c641-4084-ad61-
2cdd9353fbc0",
        "type": "SUB6_ACCESS",
        "name": "phy0",
        "config": {
          "channelNumber": 36,
          "txPower": 200,
          "channelBandwidth": 20
        }
      }
    ]
  }
}
```

```
{
  "name": "ComputeChunk",
  "user_id": "60a392bbd68f2863dd75307e",
  "username": "test_fixed_ip",
  "password": "test_fixed_ip",
  "description": "test_fixed_ip",
  "compute_id": "60a393fdd68f2863dd753084",
  "requirements": {
    "ram": {
      "required": 8192,
      "units": "MB"
    },
    "cpus": {
      "required": 4
    },
    "storage": {
      "required": 10,
      "units": "GB"
    }
  }
}
```


Figure 2.15. Sample radio chunk (left) and compute chunk (right) endpoints available in Slice Manager


```
{
  "name": "SS_02",
  "user_id": "5b63089158f568073093f70d",
  "slic3_id": "5d63089158f568073093f71e",
  "Wi-Fi_config": {
    "ssid": "MYSSID",
    "encryption": "WPA",
    "password": "minimumlengthpassword",
    "airtimeWeight": 100,
    "contentionWindowIncrease": 5
  },
  "selected_root_phy": "f9af122a-c641-4084-
ad61-2cdd9353fbc0",
  "cellular_config": {
    "plmn_id": "00103",
    "enable_day2": true,
    "apn": "apn_test",
    "imsi_list": [
      "001030056734816"
    ],
    "user_profile": {
      "opc":
"000102030405060708090A0B0C0D0E0F",
      "k":
"00112233445566778899AABBCCDDEEFF",
      "amf": "8000"
}
```

```
{
  "name": "nova",
  "user_id": "5b63089158f568073093f70d",
  "slic3_id": "5b63089158f568073093f70d",
  "network_service_id": "3b015075-7aac-4510-
b2d8-f7fb416e6dd9",
  "description": "Instance example",
  "ports": [
    80
  ],
}
```


Figure 2.16. Sample radio service (left) and compute service (right) endpoints available in Slice Manager

Figure 2.16 describes the Slice Manager APIs exposed for the slice activation service (left) and the service instantiation service (right). For the slice activation the following fields are required. First, the “plmnId” configured in the virtual core that will be instantiated as part of this slice. The “slice3_id” field linking to the previously created slice, i.e., concatenation of the compute and radio chunks. Additional service parameters related to the Wi-Fi and LiFi networks.

For the service instantiation service, Figure 2.16 (right) describes the required fields, where it is worth highlighting the “network_service_id” field that contains the identifier of a network service that has been previously onboarded on to the OSM instance, and the “slice3_id” that is used to select the VIM account that will be used to host this service (c.f. Section 2.1.1.2).

2.1.2 5G-CLARITY service and slice provisioning: quotas and core network support

After discussing how the 5G-CLARITY service and slice provisioning subsystem configures the physical infrastructure to serve different slices, this section discussed how slice isolation could be achieved through various network domains.

2.1.2.1 Wireless slicing

The instantiation of a 5G-CLARITY slice requires the allocation of resource chunks for individual resource-facing services (i.e., 5G-CLARITY wireless, transport and compute services). A resource chunk is defined as a set of resources of the same type which allows the performance requirements of a resource-facing service are met. To meet these requirements, the 5G-CLARITY slice provider needs to compute the specific amount of resources for each chunk. This amount of resources is referred to as resource quota. The definition of resource quota for every 5G-CLARITY slice must be detailed for each constituent 5G-CLARITY resource-facing service.

For slicing at the access, 5G-CLARITY wireless services need to be provisioned with needed resource quotas. A 5G-CLARITY wireless quota is the set of wireless resources which are allocated in each AP which serves a specific 5G-CLARITY wireless service. This quota could be dedicated, minimum or maximum.

- 5G-CLARITY wireless dedicated quota: defines the wireless resources which are dedicated for a 5G-CLARITY wireless service. These wireless resources cannot be shared even if the associated 5G-CLARITY wireless service does not use them throughout its lifetime.
- 5G-CLARITY wireless minimum quota: defines the minimum wireless resources, including the dedicated and prioritized wireless resources for each 5G-CLARITY wireless service. Prioritized wireless resources are those that are preferentially used by the associated 5G-CLARITY wireless service. When prioritized resources are not used, other 5G-CLARITY wireless services could use them. The goal of this quota is to guarantee a minimum capacity for each 5G-CLARITY wireless service in a situation of resource scarcity.
- 5G-CLARITY wireless maximum quota: defines the maximum wireless resources, including dedicated, prioritized and shared wireless resources for each 5G-CLARITY wireless service. Shared wireless resources are those which are shared among all the existing 5G-CLARITY wireless services. This means the shared wireless radio resources may not be guaranteed for the slices throughout their lifetime. The availability of these resources will depend on the system load conditions at a given time. The goal of this quota is to ensure each 5G-CLARITY wireless service cannot consume more capacity than an upper bound defined by the 5G-CLARITY slice provider.

When the WAT supported by the 5G-CLARITY wireless service consists of LTE/5GNR nodes, the wireless resources correspond to PRBs. If the WAT supported by the wireless service consists of Wi-Fi APs, the wireless resources correspond to the percentage of airtime consumption. Finally, if the WAT supported by the wireless service consists of LiFi APs, the wireless resources are specific wavelengths and/or airtime on a wavelength.

The 5G-CLARITY slice provider computes the 5G-CLARITY wireless quotas before deploying the 5G-CLARITY wireless services. Furthermore, it can also tune them dynamically throughout the lifetime of each wireless service. To that end, the 5G-CLARITY slice provider can use different mathematical tools such as game theory, heuristics or machine learning algorithms (we propose some algorithms in Section 3). Below, we describe some key steps to derive the 5G-CLARITY wireless quotas, and provide an illustrative example using game theory.

First, the 5G-CLARITY slice provider needs to determine the time period when the wireless infrastructure is most congested (e.g., maximum inter-cell interference levels in the 5GNR/LTE gNB/eNB. This is a challenging

task because the 5G-CLARITY slice provider needs to, *i)* consider the lifetimes of the 5G-CLARITY wireless services, which could be partially overlapped over time, and *ii)* estimate the traffic intensity experienced by each 5G-CLARITY wireless service throughout its lifetime.

Once the congested period is identified, the 5G-CLARITY slice provider translates the performance requirements of each 5G-CLARITY wireless service into the specific amount of wireless resources required in each access node. For example, knowing the inter-cell interference levels in the 5GNR-gNB/LTE-eNB for the worst-case scenario, the 5G-CLARITY slice provider can estimate the amount of PRBs required by each 5G-CLARITY 5GNR/LTE node to meet a guaranteed aggregated throughput and a maximum aggregated throughput. This means computing the 5G-CLARITY wireless minimum and maximum quotas, respectively.

To perform this translation, the 5G-CLARITY slice provider must optimize the allocation of wireless resources for individual 5G-CLARITY wireless services in each AP, with the aim of accommodating all these wireless services in an efficient way. For instance, focusing on 5GNR/LTE nodes, the 5G-CLARITY slice provider can minimize the inter-cell interference levels to allocate the minimum number of PRBs per 5G-CLARITY wireless service and thus, allowing more 5G-CLARITY wireless services to be accommodated into the private wireless infrastructure. When the 5G-CLARITY slice provider deploys a specific 5G-CLARITY wireless service, the wireless resource management algorithms (i.e., those defined in deliverable D3.2 [14]) dynamically allocate wireless resources according to the computed 5G-CLARITY wireless quota for each access node. This means these algorithms use these quotas as input constrains. For instance, the 5G-CLARITY wireless maximum quota avoids a wireless resource management algorithm allocates for a 5G-CLARITY wireless service more than a specific amount of wireless resources throughout its lifetime.

For the sake of clarity, we provide an illustrative example of how the 5G-CLARITY slice provider could proceed to compute the quotas, specifically the 5G-CLARITY wireless minimum quotas. Assuming the 5G-CLARITY slice provider plans the deployment a set $\mathcal{M}$ of 5G-CLARITY wireless services with stringent requirements in terms of Guaranteed Bit Rate (GBR), we can formulate the radio resource allocation problem as Eq. (1) shows. The operator $|\cdot|$ denotes the cardinality of a set (the number of elements in this set). The parameter S_i is the PRB allocation for the $|\mathcal{M}|$ 5G-CLARITY wireless services in the 5G cell $i \in \mathcal{I}$ which minimize the $\max(\cdot)$ function. The parameter $\mathcal{S}^i$ denotes the set of all the possible combinations of PRB allocation for each 5G-CLARITY wireless service. Finally, $\bar{B}_m$ $B_{i,m}$ and B^{th} are the average UE blocking probability for 5G-CLARITY wireless service m, the UE blocking probability for 5G-CLARITY wireless service m in the 5G cell i, and the upper bound for the UE blocking probability, respectively. Note that $\bar{B}_m = \sum_{i\in\mathcal{I}} \omega_{i,m} B_{i,m}$, where $\omega_{i,m}$ is the probability that a GBR session for the for 5G-CLARITY wireless service m is established in the 5G cell i.

$$\min_{S_i \in \mathcal{S}^i \, \forall i \in \mathcal{I}} \max\left(\bar{B}_1, \bar{B}_2, \dots, \bar{B}_m, \dots, \bar{B}_{|\mathcal{M}|}\right)$$
$$\text{s.t.} \quad B_{i,m} \leq B^{\text{th}} \; \forall m \in \mathcal{M}, \forall i \in \mathcal{I} \tag{1}$$

This equation aims to minimize the average UE blocking probability $\bar{B}_{m'}$ (i.e., probability that a UE cannot establish a GBR session in the considered access network) of the 5G-CLARITY wireless service m' which has the highest value for this parameter. Furthermore, the provided constraint enforces the UE blocking probability $B_{i,m}$ for each 5G-CLARITY wireless service m in each 5G cell i is below an upper bound B^{th}. This upper bound is way of guaranteeing an accessibility level for each 5G-CLARITY wireless service.

Solving the formulated problem can be seen as a combinatorial optimization, based on allocating specific PRBs for all the 5G-CLARITY wireless services in each 5G cell while Eq. (1) is minimized. On the one hand, since performing an exhaustive search to find the optimal solution is not computationally tractable, as an alternative, searching a local optimum in an iterative way is a better option. For instance, by using game theory to model the formulated problem, we can find a local optimum by determining a Nash Equilibrium solution. On the other hand, the specific PRB allocation and the interference levels computed in each

iteration must be translated into the UE blocking probability for each 5G-CLARITY wireless service. To that end, valuable models as the one proposed in [15] can be used.

Figure 2.17 illustrates the results obtained with a potential game to compute the 5G-CLARITY wireless minimum quotas for three 5G-CLARITY wireless services. Specifically, Figure 2.17(a) shows the specific allocation of PRB chunks (i.e., a set of consecutive PRBs) which minimize the inter-cell interference levels. An algorithm to minimize these levels can be found in [16]. In Figure 2.17(b), we illustrate the 5G-CLARITY wireless minimum quotas in each 5G cell, i.e., the percentage of PRB chunks allocated for each 5G-CLARITY wireless service. Finally, Figure 2.17(c) shows how the computed quotas satisfy Eq. (1) when the 5G-CLARITY slice provider sets $B^{\text{th}} = 0.01$.

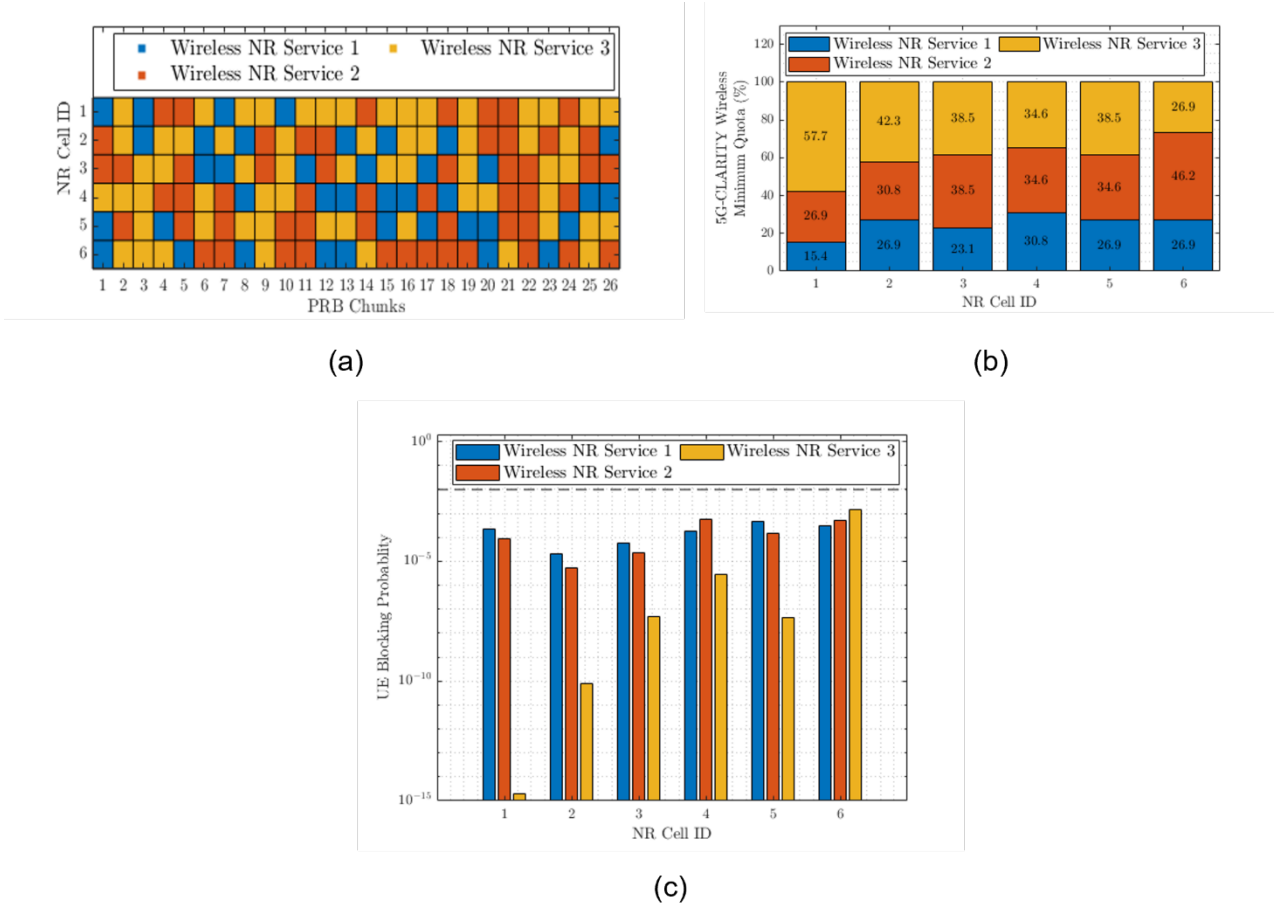


Figure 2.17. Example of 5G-CLARITY wireless minimum quotas for three 5G-CLARITY wireless NR services in each NR cell, (a) PRB chunk allocation, (b) 5G-CLARITY Wireless Minimum Quotas, (c) UE blocking probability.service (right) endpoints available in Slice Manager

2.1.2.2 Transport slicing

There are several challenges to realize the transport slicing concept in 5G-CLARITY that features a transport network (TN) that might combine multiple technologies, including standard Ethernet and TSN. In this regard, an SDN controller (SDNC) able to integrate these technologies and hide the TN configuration complexity from the service and slice orchestration subsystem is needed. The management services exposed by the SDNC through the northbound interface were defined in 5G-CLARITY D2.2 [2] to enable the provision of both virtual networks and connectivity services, as well as the cross-coordination between the TN and the rest of the 5G-CLARITY domains. The latter is crucial to ensure the cohesion and feasibility of the configurations of the

different domains.

This section delves deeper into the details of the 5G-CLARITY slices isolation assurance and QoS provision in the transport network. More precisely, it covers the mechanisms to provide isolation and QoS provision for 5G-CLARITY slices when TSN is used as the L2 transport network technology. Furthermore, technical requirements to enable the integration of the TSN-based transport network with the 5G system are identified.

One of the key features of the 5G-CLARITY slices is their high degree of isolation, which is especially suitable for multitenancy. The key ingredients in the 5G-CLARITY system to provide such a degree of isolation in the TN are, i) the use of dedicated VLANs for separate 5G-CLARITY slices, and ii) the resource quota concept for the transport service, based on the reservation of some TN resources (e.g., links capacities and buffer space at the TN devices egress ports) for a specific slice. The resource quota along with other TN configurations (e.g., 5G streams-to-priority levels mapping) must guarantee the 5G-CLARITY slice QoS requirements, which might be expressed in terms of upper bounds for the latency, jitter delay and frame loss ratio and a lower bound for the reliability. A certain level of reliability is ensured through frame replication and forwarding through disjoint paths (path redundancy).

The computation of the transport resource quotas allocated to each 5G-CLARITY slice is a computationally complex task, especially for TSN networks, that might need to be assisted by ML algorithms deployed at the 5G-CLARITY intelligence stratum. Furthermore, the transport resource quotas for a given 5G-CLARITY slice might change over time because of either traffic demand fluctuation or UEs mobility. To deal with this issue, proactive algorithms for the TN configuration and resource allocation that harness historical data might be used.

To illustrate the transport network quota concept, we consider here the scenario depicted in Figure 2.18. In the scenario, there is an asynchronous TSN network acting as backhaul. There are three 5G-CLARITY slices to be allocated whose primary traffic characteristics and QoS requisites are summarized in Table 2-3. The definitions of these characteristics and performance requisites are included below:

- Average flow rate: Average sustainable data rate generated by a flow of the respective 5G-CLARITY slice.
- Flow burstiness (burst size): Maximum amount of data generated at a given time instant by any flow of the 5G-CLARITY slice.
- Maximum packet size: The largest packet size generated by any flow of the 5G-CLARITY slice.
- TN delay budget
- TN jitter budget: The maximum allowed variation in time delay for conveying a packet of a flow from its source to its destination in the TN.
- Reliability requisite: The required probability of success for the TN to seamlessly transport the traffic of a flow while fulfilling its throughput, delay budget and jitter budget requirements.
- Average flow duration: The mean flow lifetime in the network.

For this setup, the backhaul network resource quotas specification comprises the capacity allocated and the buffer space reserved for each slice at each link and asynchronous traffic shaper (ATS), which is the building block of the asynchronous TSN networks used to handle the transmission of the frames at a given link, respectively.

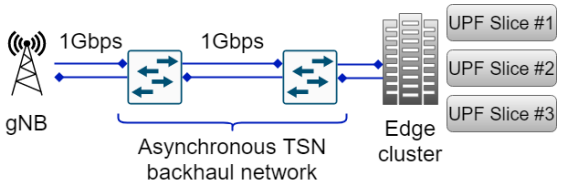


Figure 2.18. Scenario considered to illustrate the 5G-CLARITY transport network quotas computation

Table 2-3. Traffic Characteristics for Slices Used to Compute the Transport Network Quotas

5G-CLARITY Slice	Avg. Flow Rate	Flow Burstiness	Max. Packet Size	TN Delay Budget	TN Jitter Budget	Reliability Requisite	Avg. Flow Duration
Slice #1	1.55 Mbps	324 bytes	324 bytes	1 ms	1 ms	0.99	28800 s
Slice #2	20 Mbps	25000 bytes	1500 bytes	10 ms	10 ms	0.95	3600 s
Slice #3	0.05 Mbps	186 bytes	186 bytes	20 ms	20 ms	0.95	28800 s

The optimization is aimed to drive the backhaul resource quota computation to minimize the overall flow rejection probability of the network while ensuring the requisites in delay, jitter, and reliability for the different slices. There are further technological constraints related to the flows-to-shaped buffers assignment restrictions (refer to [17] and [18]), the size of the buffers, and link capacities. Table 2-4 includes the resource quotas allocated to each 5G-CLARITY slice for the downlink. Please note that the same configuration is shared among all the ATSs/links as the considered backhaul network has a daisy chain topology. Given the resource quotas in Table 2-4, Figure 2.19 and Table 2-5 show the flow rejection probability and maximum delay experienced for each 5G-CLARITY slice, respectively. Observe that the backhaul network delay budgets are met for all the flows. Last, although we have considered here a lenient reliability requisite for every slice and one path is enough to ensure it, for a different setup, transmitting the packets through several disjoint paths using the Frame Replication and Elimination for Reliability (FRER) TSN capability might be required.

Table 2-4. Transport network quotas for each 5G-CLARITY5G-CLARITY slice

5G-CLARITY Slice	Allocated Link Capacity at Each Link (ATS) [Mbps]	Buffer Size at Each ATS [kB]	Priority Level
Slice #1	283.65	59.3	1
Slice #2	340	425	2
Slice #3	119.2	443.5	3

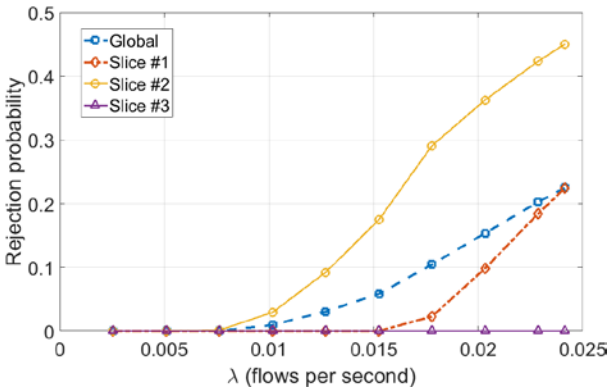


Figure 2.19. Flow rejection probability perceived for each 5G-CLARITY slice given the transport network quotas

Table 2-5. Maximum Delay Experienced by Any Packet of Each 5G-CLARITY Slice Given the Transport Network

5G-CLARITY Slice	Maximum Packet Delay [ms]
Slice #1	0.921
Slice #2	18.07
Slice #3	9.986

The application of the right policies and QoS enforcement for the different 5G-CLARITY slices requires traffic differentiation at the edge nodes, i.e., those TSN bridges that are directly connected with a 5G entity (e.g., gNB and UPF) instance. For instance, stream identification at the boundaries of a TSN network has to be used. This way, each TN device can classify incoming frames to associate them with the corresponding 5G-CLARITY slices and apply the proper forwarding policies and QoS mechanisms. Then, for instance, the 5G-CLARITY slice each frame belongs to might be encoded in an immutable field of the IEEE 802.1Q header (e.g., VLAN ID). The flow metering in TSN standards uses stream IDs which includes the VLAN ID, and priority level to apply the predefined bandwidth profiles for each stream. Please note that the traffic classification might also be required for a bare IEEE 802.1Q (without TSN extensions), for instance, to apply traffic prioritization per traffic class at every network bridge.

2.1.2.3 Compute slicing: virtual core networks serving different slices

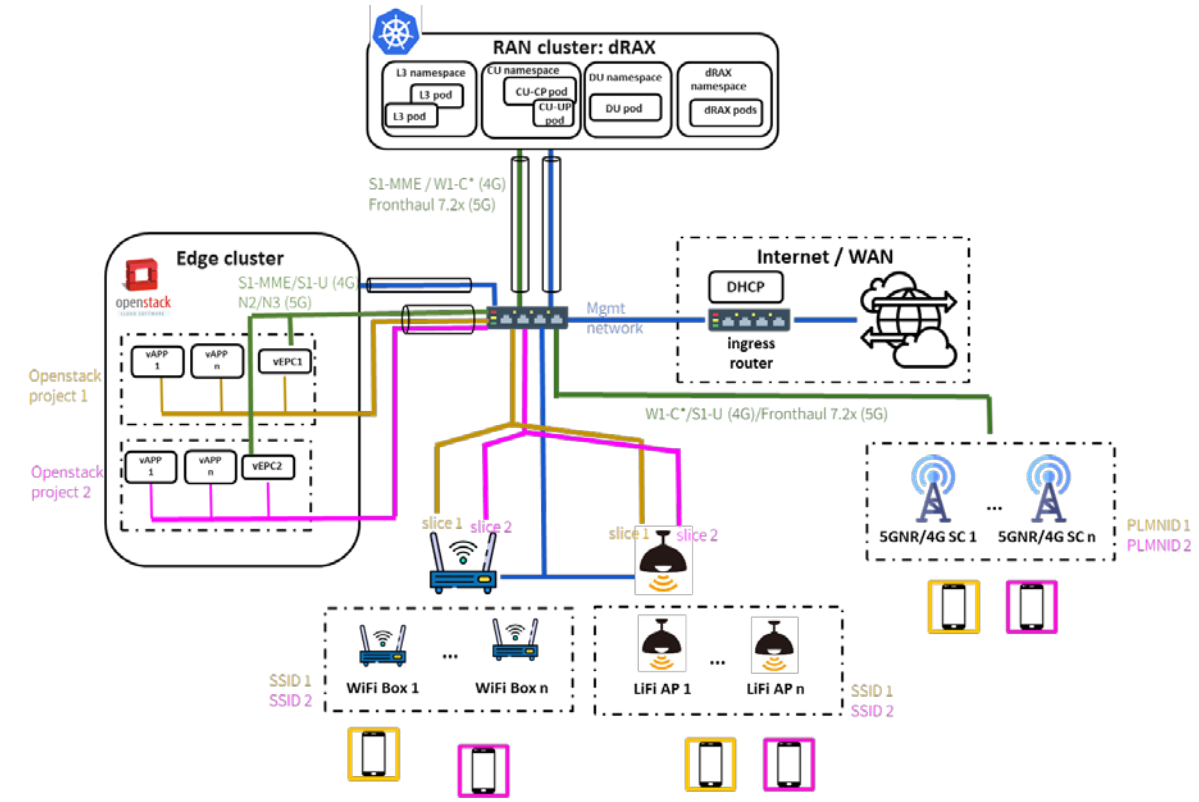
Isolation in the 5G-CLARITY edge cluster is implemented using the concept of OpenStack projects described in Section 2.1.1.2. In this section the 3GPP core network, deployed within a 5G-CLARITY compute chunk, and the way it is used to support 5G-CLARITY slicing is introduced. To this end two complementary approaches are discussed:

- PLMNID-based slicing: used in multitenant scenarios, where each tenant is represented by a separate core with a specific PLMNID instantiated within a compute slice. This approach to slicing can be implemented either with a 4G EPC or with a 5GC.
- PLMNID+SNSSAI-based slicing: used where slicing is required within a tenant. In this case a single PLMNID can be sliced into multiple S-NSSAIs. This approach to slicing requires a 5GC, since EPC does not support SNSSAI.

Notice that both slicing models can be applied for standalone NPNs, and for PNI-NPNs. In this section though, the focus is on standalone NPNs. Figure 2.20 depicts the initial design for PLMNID-based slicing, where the following elements can be identified:

- The Wi-Fi and LiFi nodes, which have a dedicated VLAN for each deployed slice. In Figure 2.20 two slices can be observed, i.e., slice 1 (yellow) and slice 2 (pink).
- 4G/5GNR Accelleran small cells, which have a dedicated VLAN for the F1 interface connecting them to the CU function located in the RAN cluster.
- The RAN cluster, which features a Kubernetes over bare metal implementation hosting the dRAX RIC from Accelleran. The RIC has separate namespaces for the L3 pods that support 4G small cells, the CU pods supporting 5GNR small cells, and a separate namespace hosting dRAX services. The RAN cluster has two physical Ethernet interfaces. One is used for management services (blue), and the other is used to carry all the S1 and F1 traffic for the different slices (green). In future implementations it will be tried to map the S1 traffic from different 5G-CLARITY slices to different transport VLANs to achieve slice separation also in the transport network. Isolating the F1 interface in the transport is more complex as the L3/CU pods are shared across slices.
- The Edge cluster features an OpenStack deployment hosting a virtual core (vEPC in the figure) per slice. Each per-slice core is provisioned within an OpenStack project, as described in Section 2.1.1.2. All virtual core have two network interfaces. First, a network interface connected to the network

used to carry the S1 traffic (green network). Second, shown in yellow and pink, a network interface connected to the service network for that slice providing local breakout. We can see that the Wi-Fi and LiFi nodes have each slice directly connecting to the service network in the Edge cluster. The service network also hosts virtual applications (vAPPs) connected to each slice.



Note: W1-C* is currently a special High Layer Split for the control plane of the eNB similar to what 3GPP defines for ng-eNB as W1

Figure 2.20. 5G-CLARITY approach to PLMNID-based slicing

A prototype of the "PLMNID-based slicing" model has been developed and a preliminary evaluation is reported in Section 2.1.3.

A preliminary design of the approach to "PMNID+SNSSAI-based slicing" is presented in this section, which will allow a single tenant to support multiple slices under the same PLMNID, each slice identified with an S-NSSAI.

Figure 2.21 shows the initial approach to PMNID+SNSSAI-based slicing, describing how multiple 3GPP S-NSSAI slices can be deployed within a single PLMN instantiate over the 5G-CLARITY infrastructure stratum, comprising the DUs, the RAN cluster and the Edge cluster.

In Figure 2.21 one PLMN is instantiated that offers two slices, where each slice served multiple Data Network Names. The configuration used herein is captured in Table 2-6. In addition, each slice is served by a separated SMf and a dedicated CU-UP.

Table 2-6. Slice Configuration

5G-CLARITY Slice	Identifier	Serving Data Network Name
Slice #1	S-NSSAI#1: {SST=1, SD=n/a}	DNN1 through UPF1 DNN2 through UPF2
Slice #2	S-NSSAI#2: {SST=1, SD=2}	DNN12 through UPF12

Based on the previous specification, various slice components could be mapped to the compute chunks defined in the 5G-CLARITY slicing model as:

- Edge cluster
  - Compute chunk 1 (green): used to instantiate the PLMN control plane (AMF and UDM), as well as the SMF for slice 1 and the UPF serving DNN1. This can be considered the basic system required to support the PLMN.
  - Compute chunk 2 (orange): used to support the UPF and application services for DNN2. This compute chunk would have a separate CPU/RAM/storage quota than compute chunk 1, thus providing service level isolation between DNN1 and DNN2.
  - Compute chunk 3 (blue): supports the SMF deployment for slice 2, as well as the UPF and application services for DNN12.
- RAN cluster
  - Compute chunk 1 (green): used to support the control plane, CU-CP, as well as the default user plane serving slice 1.
  - Compute chunk 2 (blue): used to instantiate the user plane for slice 2, thus achieving user plane isolation between slice 1 and slice 2.

Alternative mappings between the 3GPP network functions and the 5G-CLARITY compute chunks defined in the Edge and RAN cluster are possible. The PLMNID-based slicing model will be pursued and an experimental demonstration will be provided in 5G-CLARITY D4.3.

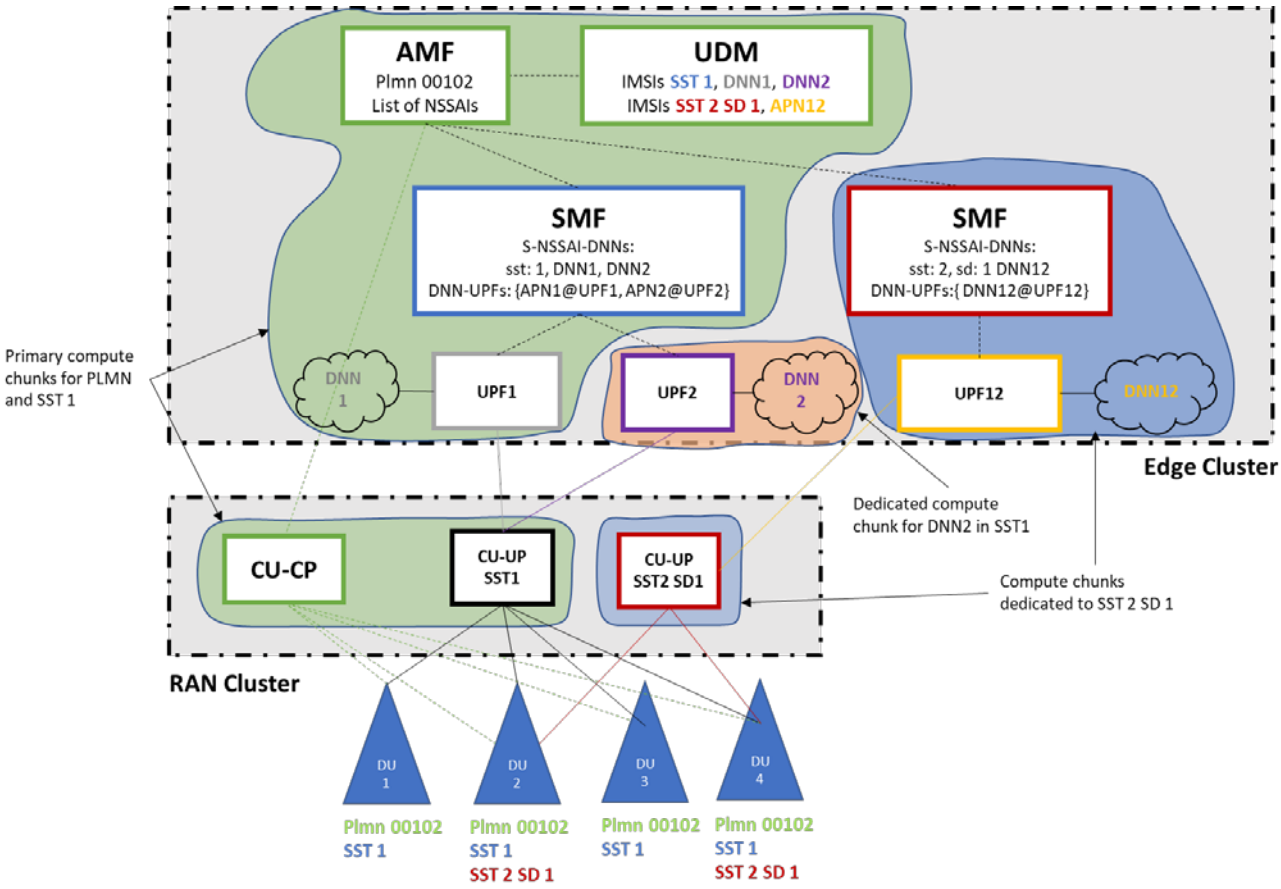


Figure 2.21. Potential implementation of PLMNID+SNSSAI-based slicing

2.1.3 5G-CLARITY service and slice provisioning subsystem: preliminary evaluation

This section presents a preliminary evaluation of the time required to provision 5G-CLARITY infrastructure slice following the PLMNID-based slicing model described in Section 2.1.2.3. The evaluation of PMNID+SNSSAI-based slicing model will be performed and report in the next deliverable.

To evaluate the time required to provision 5G-CLARITY slices a testbed featuring NETCONF enabled wireless nodes is deployed including, *i)* a custom Wi-Fi AP provided by i2CAT, ii) a LiFi AP provided by pureLiFi, and *iii)* a 4G small cell provided by Accelleran. The 5GNR small cells were not available in the testbed but no significant differences are expected regarding the provisioning times. The testbed also includes an OpenStack based edge cluster and the components of the service and slice provisioning subsystem including an NFV orchestrator based on OSM.

Given our focus on the PLMNID-based slicing model, the slice provisioning time is characterised by looking separately at the time to advertise new PLMNID and SSID services in the wireless technologies, and the time to provision the virtual network functions required to support the slices. In particular, the following parameters are defined:

- LiFi/Wi-Fi-Prov: the time required to configure an SSID and a VLAN in the LiFi/Wi-Fi nodes using NETCONF,
- 4G-Prov: the time required to add a PLMNID to the 4G cell using NETCONF,
- vEPC-Prov: the time required to instantiate a virtual evolved packet core (vEPC) based on open5gs [19] to support the traffic of the slice. The vEPC is provisioned directly on top of the OpenStack,
- AT3S-Prov: the time required to provision a virtual AT3S proxy used to support the multi-connectivity framework defined in 5G-CLARITY D3.2 [14]). This virtual function is deployed as a VNF using the NFV MANO orchestrator.

Figure 2.22 depicts the Cumulative Density Function (CDF) of the provisioning times obtained from 50 experiments. It can be observed that configuring a new SSID service in Wi-Fi and LiFi using NETCONF can be performed in less than 3 seconds, while configuring a PLMNID in the 4G cell requires almost 40 seconds as it involves a full system reboot. Deploying the vEPC and AT3S functions in the edge cluster involves booting virtual machines which requires between 10 and 20 seconds in our OpenStack based edge cluster deployment. The overall slice provisioning time can be extracted from the previous measurements in the following way. First, the vEPC needs to be provisioned, which takes around 20 seconds. Then, the AT3S function and all wireless technologies can be configured in parallel, dominated by the 4G-Prov time of around 40 seconds. Thus, we can conclude that an end-to-end 5G-CLARITY slice including wireless, transport and compute services can be deployed in around 60 seconds.

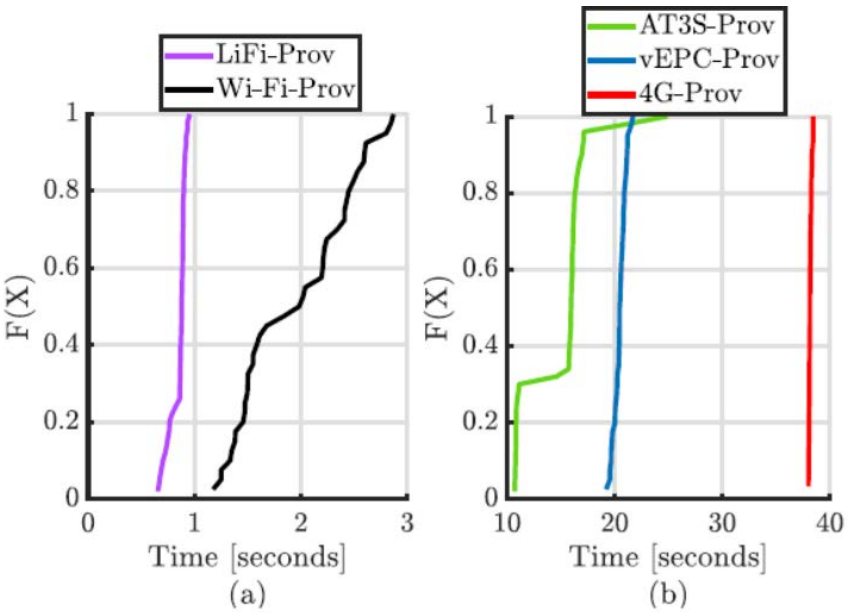


Figure 2.22. Empirical times required to provision the various components of a 5G-CLARITY slice

2.2 5G-CLARITY telemetry

2.2.1 Telemetry solutions

2.2.1.1 Data Lake

In 5G-CLARITY D2.2 [2] and D4.1 [1], Data Lake is considered as one of the 5G-CLARITY solutions on data management and processing subsystem. The focus of the Data Lake is to centralize the data management and allow virtual unlimited data storage allowing a cost-effective management of data and its access. Analysing large data comes with a number of challenges, which include infrastructure, cost, storage and security. One solution to these challenges relies on cloud computing, which migrate the in-house infrastructure requirement to an external platform. In this section, we propose an AWS cloud-based approach. AWS is a cloud computing platform provided by Amazon. It comprises a multitude of services, which includes computing, networking, storage, database, analytics and IoT. The bulk of AWS services lie in the background and are not exposed to the consumer. These services are only utilized through API calls. This solution extends the telemetry handling to the cloud side via an edge premises. Data Lakes allow *i)* data-based access controls which enables multiple roles within the same organization or external organizations to gain access to a specific data for a specific time; *ii)* running analytics without the need to move data to a separate analytics system; and *iii)* large data storages to be tiered based on access frequency.

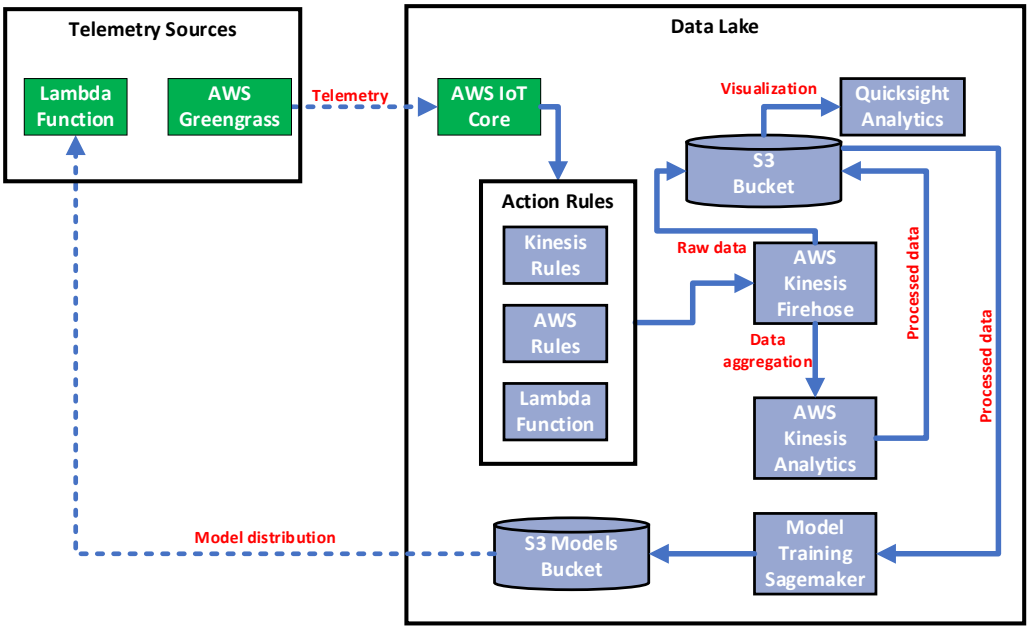


Figure 2.23. Data Lake – initial solution implementation

Figure 2.23 illustrates the overall framework of the AWS cloud-based data lake solution targeted in 5G-CLARITY. The specific AWS services/components within the Data Lake and telemetry sources can be summarized as follows:

- **AWS IoT Greengrass** seamlessly extends AWS to edge devices so they can act locally on the data they generate. Can run Docker containers, execute predictions based on ML models and communicate with other devices securely – even when not connected to the Internet.
- **AWS IoT Core** is the entry point to the cloud for devices that run IoT Greengrass.
- **AWS Kinesis Firehose** is used to capture, transform, and load streaming data continuously into AWS from data sources.
- **AWS Kinesis Analytics** service allows you to process streaming data coming from IoT devices in real time with standard SQL.
- **AWS Lambda** is a serverless compute service that lets you run code without provisioning or managing servers, creating workload-aware cluster scaling logic, maintaining event integrations, or managing runtimes.
- **AWS Simple Storage Service (S3)** is an object storage service that offers scalability, data availability, security, and performance.
- **AWS Sagemaker** offers an IDE to build, train and deploy ML models.

2.2.1.2 Data Semantics fabric

The concept of Data Semantic Fabric allows consolidating data from a wide variety of *sources* and turn them into useful information *consumers*, by applying necessary processing on the collected data before their storage and/or transmission. As was captured in D2.2 and D4.1, this Data Fabric includes a set of logical nodes, each with a well-defined functional functionality:

- The **collector,** responsible for data harvesting,
- The **aggregator,** in charge of manipulating and combining individual data collected together, making them available for their consumption. This processing is done according to some rules (e.g., arithmetic operations, filtering, thresholding),

- The **dispatcher,** which sends aggregated data out to the target destination. This destination is where the data is stored for their consumption.

Figure 2.24 shows an initial solution implementation for the Data Semantics Fabric. As seen, this solution leverages on the NGSI-LD framework (see details in deliverable D4.1), articulated into two main artifacts: *i)* the NSGI-LD Information Model entity, kept at NGSI-LD Context Broker, and which allows to deal with the fabric lifecycle management; and *ii)* the NGSI-LD API, which is the API that allows translating orders from the orchestrator to requests to the Data Semantics Fabric.

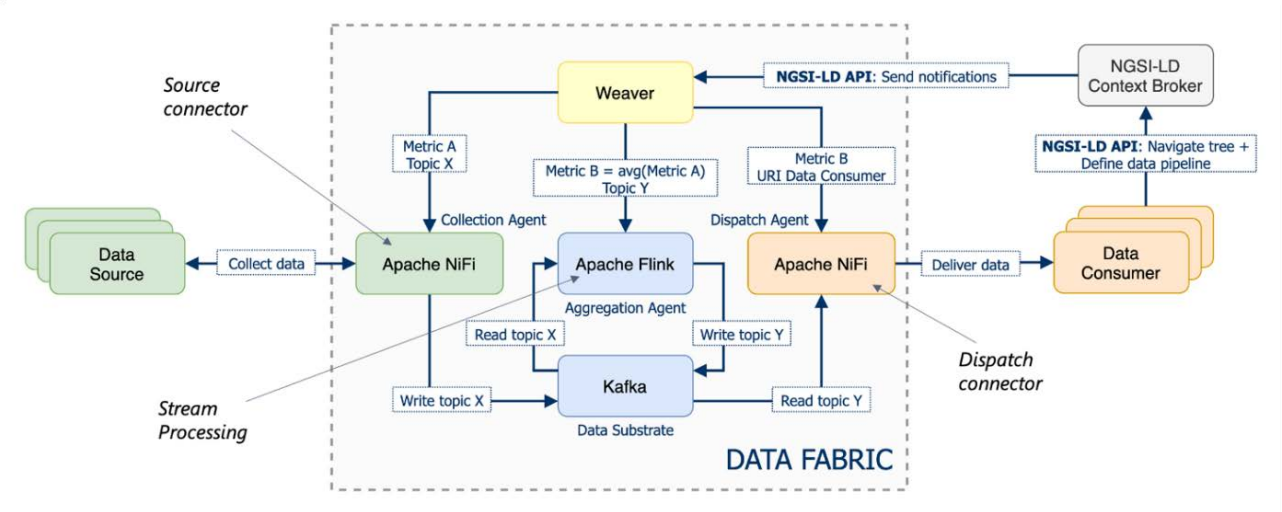


Figure 2.24. Data Semantics fabric – initial solution implementation

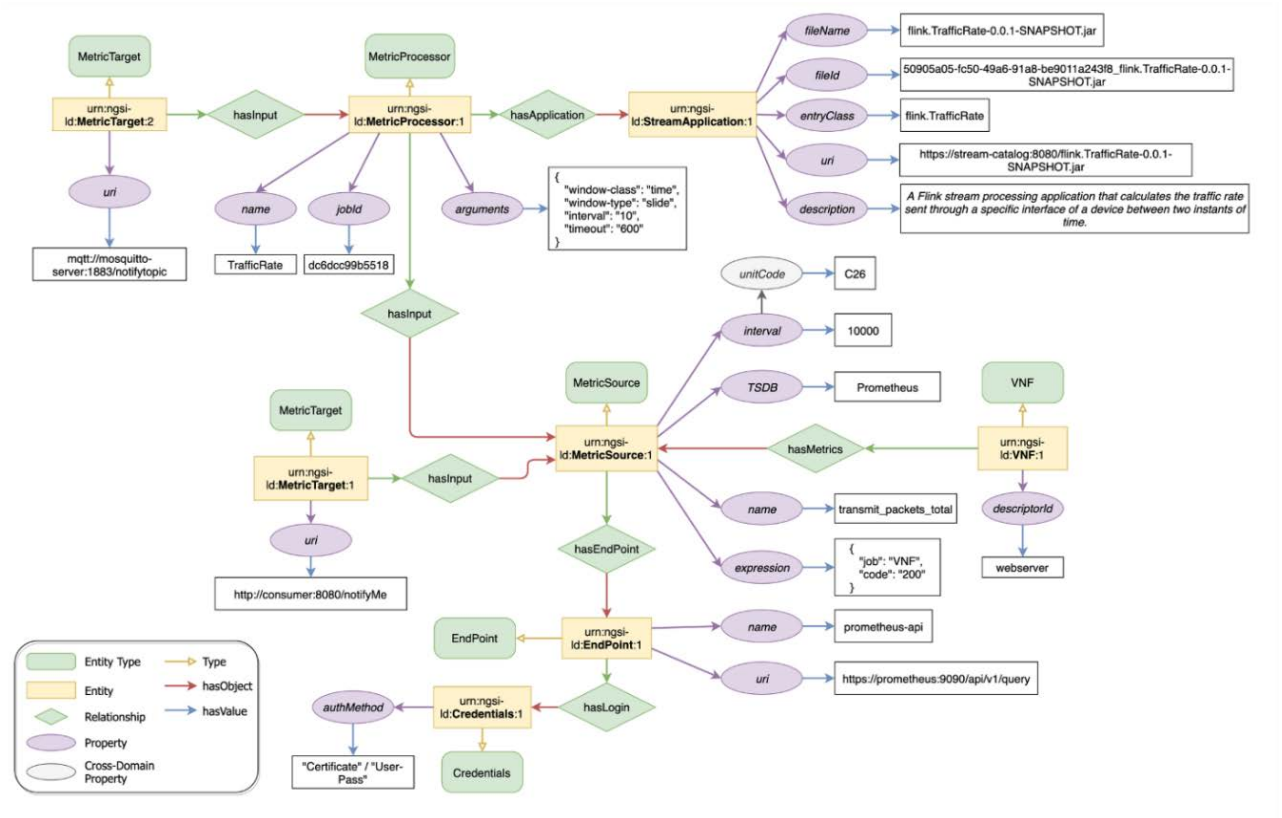


Figure 2.25. NGSI-LD information model

On receiving the notifications from the NGSI-LD Context Broker, the weaver manages (instantiate, deploy,

upgrade, stop, delete) required agents on the collector and dispatcher, and configures the stream processing platform according to the content captured in the NGSI-LD information model. An example of this model is captured in Figure 2.25. On the one hand, the collector and dispatcher are both based on *Apache NiFi* [20], which is reliable and secure data distribution system enabling a graph-oriented programming of data flows. Given that NiFi's job is to bring data from wherever it is (data source), to wherever it needs to be (data consumer), it makes sense to bring data to and from Kafka. On the other hand, the stream processing platform is implemented with to *Apache Flink* [21], which is a framework and distributed engine for stateful computations over unbounded and bounded data streams. This processing can be based on stream processing (i.e., events are processed as they come) or micro-batch processing (i.e., all data is ingested before performing any computations).

The typical scenario is illustrated in Figure 2.26. As seen, NiFi ingests data tom Kafka, which makes it available to Flink with the results being written back to a different Kafka topic where the NiFi is consuming from, and the result being delivered to intended data consumer.

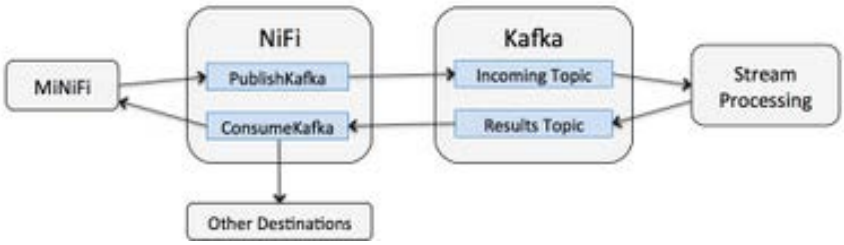


Figure 2.26. Data Flows

2.2.2 Telemetry data interfaces

A set of candidate interfaces (northbound and southbound) that can be used between the telemetry solutions, namely the data lake and data semantic fabric, and other 5G-CLARITY architectural components such as AI engine, Intent engine, RAN/dRAX and so on are discussed and compared. Figure 2.27 shows these interfaces. The section-interface mapping shown in the figure indicates the interfaces described in this deliverable. Details of the remaining interfaces will be provided in the next deliverable.

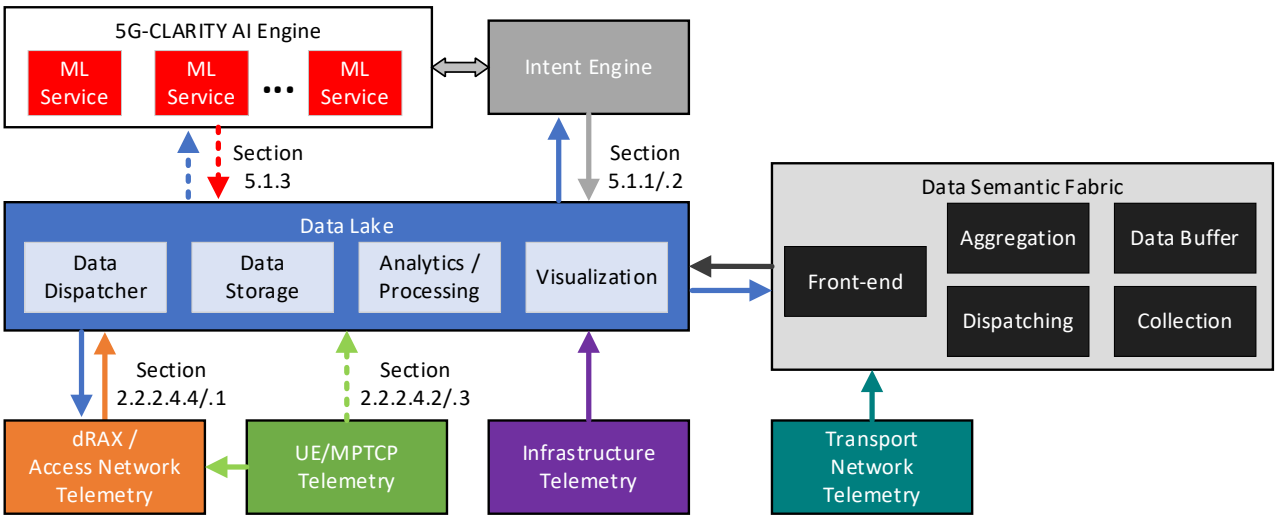


Figure 2.27. Interfaces between the Data Lake and other 5G-CLARITY architectural components

The list of these interfaces is captured below:

- **From/to AI engine to/from Intent engine**: will be covered in Section 5 – Intent Engine

- **From Intent Engine to Data Lake**: This is the interface that the Intent Engine requests/pulls data from the data lake based on any request from *i)* AI Engine regarding the ML models; or *ii)* private network operator to visualize network performance/analytics.
- **From Data Lake to Intent Engine**: This is the interface that the Data Lake provides requested data/analytics information to the Intent Engine.
- **From AI Engine to Data Lake**: The physical interface between the Data Lake and Intent Engine can be used to provide the request to the Data Lake. A logical interface (shown with a red dashed line) can be described between the AI Engine and Data Lake.
- **From Data Lake to AI Engine**: The physical interface between the Data Lake and Intent Engine can be used to provide requested data to the AI engine. A logical interface (shown with a blue dashed line) can be described between the Data Lake and AI Engine.
- **From dRAX/Access Network to Data Lake**: This interface provides network telemetry data (may also include UE telemetry) from dRAX/RAN to the Data Lake.
- **From Data Lake to dRAX**: This is the interface to update an ML model based xApp residing in the dRAX. Training of the ML model can be carried out in the Data Lake or AI Engine and ML model updates can be pushed to dRAX RIC xApp via this interface. xApp onboarding can be done via REST API interface.
- **From UE/MPTCP to dRAX**: This is the interface to provide UE telemetry and MPTCP telemetry data to dRAX.
- **From UE/MPTCP to Data Lak**e: 3GPP does not support direct collection from UE (see key issue #8 from [22])– the physical interface between the Data Lake and dRAX can be used to provide UE telemetry as well as MPTCP telemetry data to the Data Lake. A logical interface (shown as a green dashed line) can be described between the Data Lake and UE.
- **From Infrastructure to Data Lake**: This interface is to provide infrastructure telemetry (e.g., VNF consumption) to the Data Lake.
- **Interface between Data Lake components**: This is the interface inside the Data Lake to dispatch, store and/or direct data to appropriate component.
- **From Transport Network to Data Semantic Fabric**: This interface provides transport network telemetry to the Data Semantic Fabric.
- **From Data Semantic Fabric to Data Lake**: This interface retrieves the processed data from the Data Lake.
- **From Data Lake to Data Semantic Fabric**: This interface injects the processed data to the Data Semantic Fabric.

2.2.2.1 Comparison of existing interfaces

Table 2-7 provides a summary on the existing interfaces and their triggering condition, implementation aspects, and pros and cons.

Table 2-7. Existing Interfaces and Their Properties

Interface	Triggering Condition	Pros	Cons
REST	Data receiver actively triggers the data retrieval process.	• Good for short lived processes • Simple to implement/use by asking for data, getting data if available	• NA
KAFKA	Data receiver is passive, it is triggered by incoming	• Real-time streaming data pipelines.	• Hardware requirements may be too high for

	data.	• Enables applications to react to the streams of data named as topic. • Many consumer groups can subscribe to the same topic. • Open-source, free to use.	lightweight devices. • Implementation and maintenance complexity.
AWS Kinesis	The data producer emits the data records as they are generated and the data consumer retrieving data stream as it is generated.	• Good for processing big data in real-time. • Simplifies the development process of certain applications such as operational decision making with streaming data. • Writes each message synchronously to different machines. • Easy to implement and maintain.	• AWS product, pay and use. • Limitations compared to open-source products.

The noted interfaces in the table above are considered in 5G-CLARITY, depending on their triggering condition. For example, the northbound interfaces shown in Figure 2.27, there is no need for real-time streaming. Therefore, REST interface is considered for the interfaces between *i)* the data lake and AI engine and *ii)* the data lake and Intent engine. On the other hand, the southbound interfaces shown in Figure 2.27, some telemetry data such as dRAX and UE telemetry is needed in real or near-real time, whereas some telemetry data can be retrieved in non-real time. Therefore, depending on the telemetry data retrieving frequency and implementation consideration, Kafka and AWS kinesis are used for the southbound interfaces. In addition, as the AWS cloud-based approach is used for one of the 5G-CLARITY telemetry solutions, the capabilities in AWS Kinesis such as data streaming, Kinesis firehouse and data analytics are used within components inside the data lake.

Based on the above rationale, Table 2-8 provides the selected solutions for the interfaces allowing for the exchange telemetry data among involved MFs.

Table 2-8. 5G-CLARITY Interfaces for the Exchange of Telemetry Data.

5G-CLARITY Interface	Interface Implementation	Triggering Condition
From/to AI engine to/from Intent engine	REST	ML algorithm residing in the AI engine requires data or takes an action.
From Intent Engine to Data Lake	REST	A specific data set available in the data lake is needed.
From Data Lake to Intent Engine	REST	There is a request from the Intent Engine/AI Engine.
From AI Engine to Data Lake	REST	ML algorithm residing in the AI engine requires data.
From Data Lake to AI Engine	REST	There is a request from the AI Engine.
From dRAX/Access Network to Data Lake	KAFKA/AWS Kinesis	Continuous access network/UE telemetry data streaming – near real-time.
From UE to dRAX	REST	Continuous UE telemetry streaming – near real-time.
From MPTCP to dRAX	REST	Continuous MPTCP telemetry streaming – near real-time.
From dRAX to Wi-Fi-LiFi	REST	Using Prometheus HTTP API to retrieve Wi-Fi-LiFi metrics stored in the Prometheus server. API available here: https://prometheus.io/docs/prometheus/latest/querying/a

		pi/
From UE/MPTCP to Data Lake	Logical	Continuous UE telemetry streaming – near real-time.
From Infrastructure to Data Lake	REST	Continuous infrastructure telemetry streaming – non-real-time.
Interface between Data Lake components	AWS Kinesis	New data file/set is available.
From Transport Network to Data Semantic Fabric	gNMI/NiFi	Continuous transport network telemetry streaming – non-real-time.
From/to Data Semantic Fabric to/from Data Lake	KAFKA/AWS Kinesis	There is a need for a specific data available in the data lake or there is a request from the data semantic fabric.

2.2.2.2 Data Semantic Fabric Service Interfaces

Table 2-9, introduced in D2.2 [2], describes the management services exposed by the data semantics fabric. The APIs implementing these services are still in roadmap, so demonstrations and further detail implementations will be provided in the upcoming 5G-CLARITY D4.3.

Table 2-9. Data Semantic Fabric Services

MF Service ID	MF Service Name	Description	Reference Specifications
DSF_Src_Mgmt	**Source management**	This service allows manipulating (create, update, read, delete) the entries in the source registry. Each entry contains the class corresponding to a different data source. The start (termination) of a subscription - with a data source deployed on the 5G-CLARITY system - translates into the insertion (removal) of a new (existing) subscription into (from) the source registry.	Custom (REST)
DFS_Cns_Mgmt	**Consumer management**	This service allows manipulating (create, update, read, delete) the entries in the consumer registry. Each entry contains the information corresponding to a different data consumer.	Custom (REST)
DFS_Pipe_Prov	**Data pipeline provisioning**	This service allows issuing individual service requests for the creation of data pipelines and retrieving information about their state throughout their lifecycle.	Custom (REST)
DFS_Cap_Ex	**Capability exposure**	This service allows providing file version system capabilities. Examples of these capabilities include in-built rules for data aggregation. It is worth noting that different (more recent) versions may bring different (more advanced) set of capabilities.	Custom (REST)

2.2.2.3 Data Lake Service Interfaces

Table 2-10, introduced in 5G-CLARITY D2.2 [2] describes the management services exposed by the data lake.

Table 2-10. Data Lake Services

MF Service ID	MF Service Name	Description	Reference

			Specifications
DataLake_Ingress_Service	**Data Lake Ingress Service**	This service allows ingestion of structured and unstructured data to data lake.	Custom
DataLake_Exposure_Service	**Data Lake Exposure Service**	This service exposes data lake storage to authenticated users.	Custom
DataLake_DataSecurity_Service	**Data Lake Data Security Service**	This service maintains data access policies of the data lake.	Custom
DataLake_Discovery_Service	**Data Discovery Service**	This service allows data discovery queries to the data lake metadata.	Custom
DataLake_Exploration_Service	**Data Exploration Service**	Data Exploration Service Allows User to gain access to specified data in data lake in order to explore it.	Web service (e.g. Jupyter Notebooks)

In order to clarify the noted data lake services, some AWS data lake related details are provided as follows. The entry way to the AWS Cloud is an API Gateway that allows to fetch any telemetry data produced by various components shown in Figure 2.27 and redirect them to storage as well as analysis endpoints within the AWS cloud. Figure 2.28 shows various paths that can be used by the API to upload and send data to different buckets.

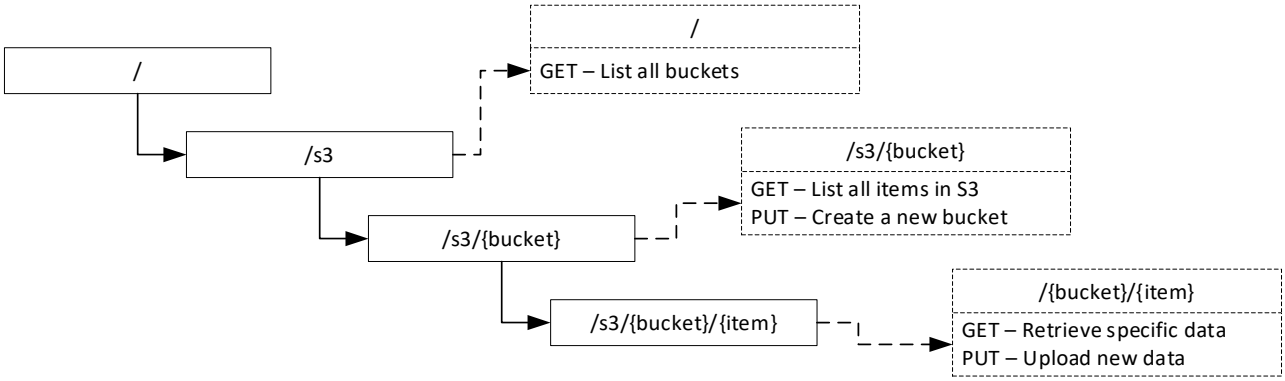


Figure 2.28. S3 end points used by API Gateway

Regarding the Data Lake Ingress Service, a Lambda function can be used to trigger ingestion of structured or unstructured data to the data lake. Another Lambda function can also be used to react on any new object/data events on S3 (PUT data to a specific bucket and/or item inside a bucket), meaning that every new object stored in the bucket will trigger the Lambda function, as depicted in Figure 2.23. The same approach can also be used for Data Lake Exposure Service and Data Lake Discovery Service where one Lambda function exposes the s3 buckets and/or items list (GET to list all buckets or all items in the bucket), and another Lambda function discovers and retrieves a specific data (GET inside an item to retrieve the specific data).

2.2.2.4 Near RT-RIC telemetry

In the 5G-CLARITY architecture the near RT RIC plays a pivotal role in integrating different telemetry streams towards the data-lake, so that data becomes available to the machine learning models hosted in the AI Engine.

The basic technology supporting this data integration is the xApp framework available in the dRAX near RT RIC developed by ACC in 5G-CLARITY. The xApp framework allows us to develop customized python-based applications that will fetch data from different sources, to then publish the aggregate data-stream into a kafka-based data bus available inside dRAX. Thus, a generic telemetry collector xApp will subscribe to the available topics in the kafka bus and push them towards the data-lake.

Figure 2.29 illustrates the xApps that will be developed in 5G-CLARITY in charge of pulling telemetry from different data sources into the dRAX's kafka-bus. As we can see the following xApps have been identified:

- Wi-Fi+LiFi xApp. This xAPP will be in charge of pulling telemetry from the Wi-Fi and LiFi APs and publishing this telemetry into the dRAX data bus. An initial version of this xApp has been developed in this deliverable and is reported later in this section.
- UE telemetry xApp. This xApp will be in charge of pulling metrics captured from the UE perspective and publishing the metrics into the dRAX databus. The UE telemetry xApp will make use of a UE telemetry agent that is described in Section 2.2.2.4.2.
- MPTCP xApp. This xApp will pull end-to-end metrics obtained from the AT3S user plane function, i.e. the MPTCP socket. There are two endpoints that offer this telemetry, namely the MPTCP socket in the 5G-CLARITY CPE and the MPTCP socket in the MPTCP proxy sitting behind the UPF in the 5G-CLARITY edge cluster. In Section 2.2.2.4.3 we describe the data that will be made available to the MPTCP xApp.
- Data-Lake xApp. This xApp will be in charge of subscribing to the relevant topics in the kafka bus and exporting the telemetry towards the Data Lake, from where the data will be made available to the models in the AI Engine.

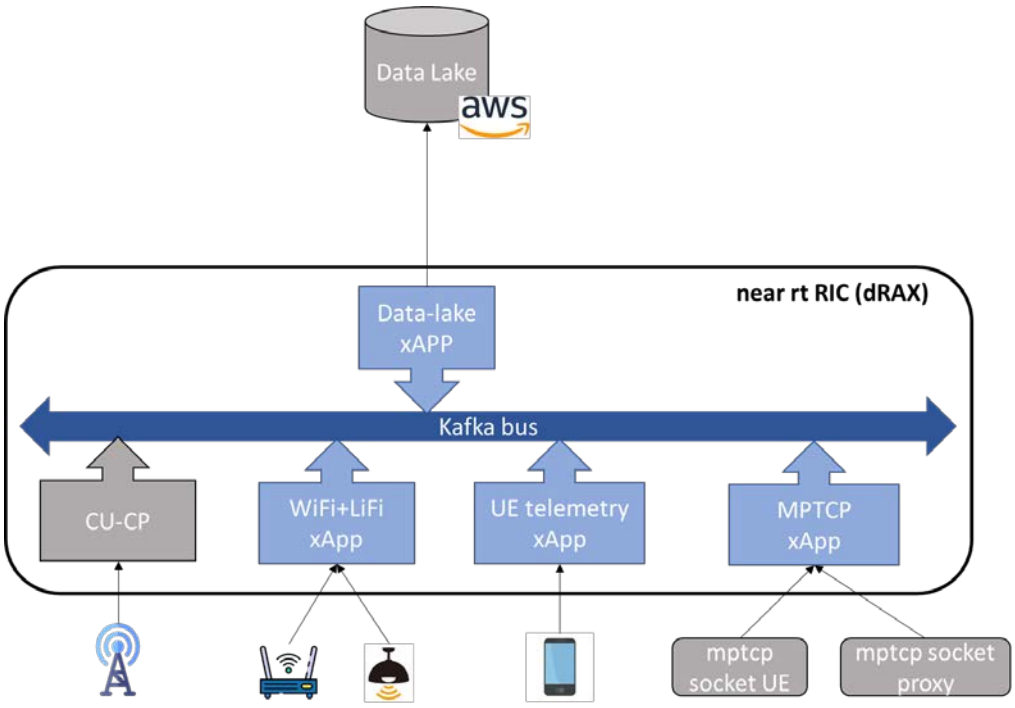


Figure 2.29. Telemetry xAPPs executed in dRAX

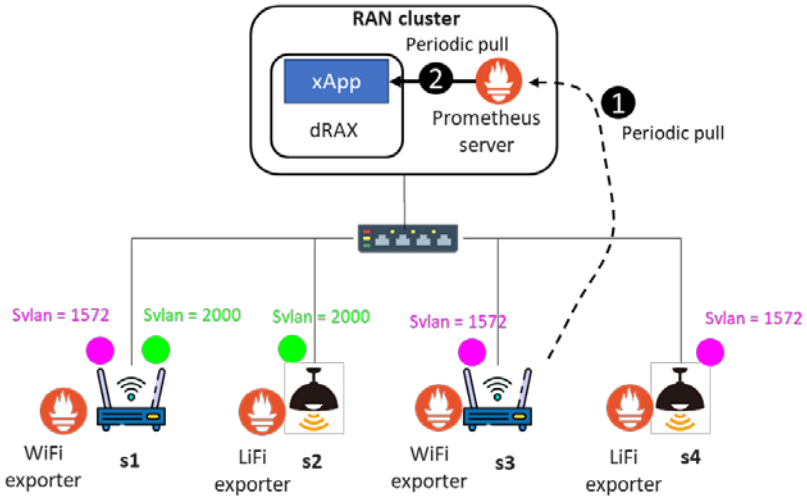
2.2.2.4.1 Wi-Fi + LiFi xApp

Next, we discuss in detail the implementation of "Wi-Fi+LiFi" xApp that is illustrated in Figure 2.30. The "Wi-Fi+LiFi" xApp can collect telemetry from various physical Wi-Fi and LiFi APs. In addition, following the 5G-CLARITY slicing model we know that each physical Wi-Fi and LiFi AP can instantiate multiple virtual Aps, each with a defined SSID and connected to a separate transport VLAN. In Figure 2.30 we see physical APs identified with a "boxname", e.g. s1, and we represent with a pink and light green circle two virtual APs connected to their respective VLANs. Physical APs are then connected via Ethernet to the RAN cluster, where the near RT RIC (dRAX) and a Prometheus server are deployed. The flow of data in this architecture is the following:

- The Prometheus server in the RAN cluster periodically pulls the various counters from the Wi-Fi and LiFi nodes through their respective hostapd exporters. For Wi-Fi nodes the following hostapd

exporter developed by I2CAT is used [23]. For the case of LiFi a custom exporter developed by PLF is used.

- The Wi-Fi+LiFi xApp uses a standard Prometheus API to periodically pull data from the Prometheus server and publishes the retrieved data into the dRAX databus, where data becomes available to any other xApp.



[INFO] 2021-06-15 09:21:56,510 (PROCESSOR) {'metadata': {'Svlan': '1572', 'id': 's1_phy1_1572', 'instance': '192.168.11.119:9551', 'job': 's1'}, 'topic': 'I2cat.i2cat_team.wifi.external.raw.hostapd_ap_num_stations', 'type': 'WIFI_AP_STATS', 'timestamp': 1623748916.497, 'value': '0'}

[INFO] 2021-06-15 09:53:24,389 (PROCESSOR) {'metadata': {'Svlan': '1572', 'id': 's1_phy1_1572', 'instance': '192.168.11.119:9551', 'job': 's1', 'mac_sta': '00:23:15:ff:82:82'}, 'topic': 'I2cat.i2cat_team.wifi.external.raw.hostapd_sta_signal_dBm', 'type': 'WIFI_STA_STATS', 'timestamp': 1623750804.387, 'value': '-51'}

[INFO] 2021-06-15 09:53:24,389 (PROCESSOR) {'metadata': {'Svlan': '1572', 'id': 's1_phy1_1572', 'instance': '192.168.11.119:9551', 'job': 's1', 'mac_sta': '00:23:15:ff:82:82'}, 'topic': 'I2cat.i2cat_team.wifi.external.raw.hostapd_sta_total_airtime', 'type': 'WIFI_STA_STATS', 'timestamp': 1623750804.387, 'value': '44344'}

Figure 2.30. Wi-Fi+LiFi telemetry xApp

Figure 2.30 also describes the structure of the topics published by the Wi-Fi+LiFi xApp in the databus, focusing on the case of Wi-Fi. The integration of the LiFi exporters was still under development at the time of writing this deliverable. The following fields are available:

- Topic name and value (marked in red): Defines the type of metric being published and the respective value. The available metrics are described in Table 2-11.
- Topic type (marked in yellow): "WI-FI_AP_STATS", if this is a topic that refers to an AP level metric, and "WI-FI_STA_STATS", if the topic refers to a station specific counter.
- Timestamp (marked blue): Defines the moment the metric was pulled from the AP by the Prometheus server.
- Service VLAN (SVlan – marked pink): Defines the transport VLAN that this AP or station is connected to. This field can be used as a slice identifier within the 5G-CLARITY architecture. Thus, an ML model in the AI Engine could subscribe to all metrics for a given slice using this field.
- AP Identifier (id – marked purple): Identifies the physical AP (e.g., s1) and the radio module within that AP (e.g. "phy1"). This field can be used to aggregate counters on a per-AP basis.

Finally, Figure 2.31 provides a snapshot of the dRAX bus where one can find both Wi-Fi related telemetry topics (marked in green) and 4G related telemetry topics (marked in orange).

Table 2-11. Wi-Fi Related Topics Available in dRAX

Topic Name	Type
I2cat.i2cat_team.Wi-Fi.external.raw.hostapd_sta_tx_rate_bps	WI-FI_STA_STATS
I2cat.i2cat_team.Wi-Fi.external.raw.hostapd_sta_backlog_bytes_total	WI-FI_STA_STATS
I2cat.i2cat_team.Wi-Fi.external.raw.hostapd_sta_connected_time_total	WI-FI_STA_STATS
I2cat.i2cat_team.Wi-Fi.external.raw.hostapd_sta_rx_bytes_total	WI-FI_STA_STATS
I2cat.i2cat_team.Wi-Fi.external.raw.hostapd_sta_rx_rate_bps	WI-FI_STA_STATS
I2cat.i2cat_team.Wi-Fi.external.raw.hostapd_sta_signal_dBm	WI-FI_STA_STATS
I2cat.i2cat_team.Wi-Fi.external.raw.hostapd_sta_total_airtime	WI-FI_STA_STATS
I2cat.i2cat_team.Wi-Fi.external.raw.hostapd_sta_tx_bytes_total	WI-FI_STA_STATS
I2cat.i2cat_team.Wi-Fi.external.raw.hostapd_ap_freq_Hz	WI-FI_AP_STATS
I2cat.i2cat_team.Wi-Fi.external.raw.hostapd_ap_max_txpower_dBm	WI-FI_AP_STATS
I2cat.i2cat_team.Wi-Fi.external.raw.hostapd_ap_num_stations	WI-FI_AP_STATS
I2cat.i2cat_team.Wi-Fi.external.raw.hostapd_ap_channel	WI-FI_AP_STATS

Figure 2.31. Sample of 4G (orange) and Wi-Fi (green) telemetry topics available on the dRAX kafka bus

2.2.2.4.2 UE Telemetry xApp

The overall UE telemetric sub system is described in Figure 2.32, wherein an application running in the UE sends through REST-API in JSON format the measurements of set of radio parameters to the near RT RIC by the "UE telemetry" xApp, as illustrated in Figure 2.29. The collected UE telemetry then can be sent to a data lake hosted in a close-range edge computing (i.e., 5G-CLARITY Edge cluster and AI engine) or in a far cloud computing environment.

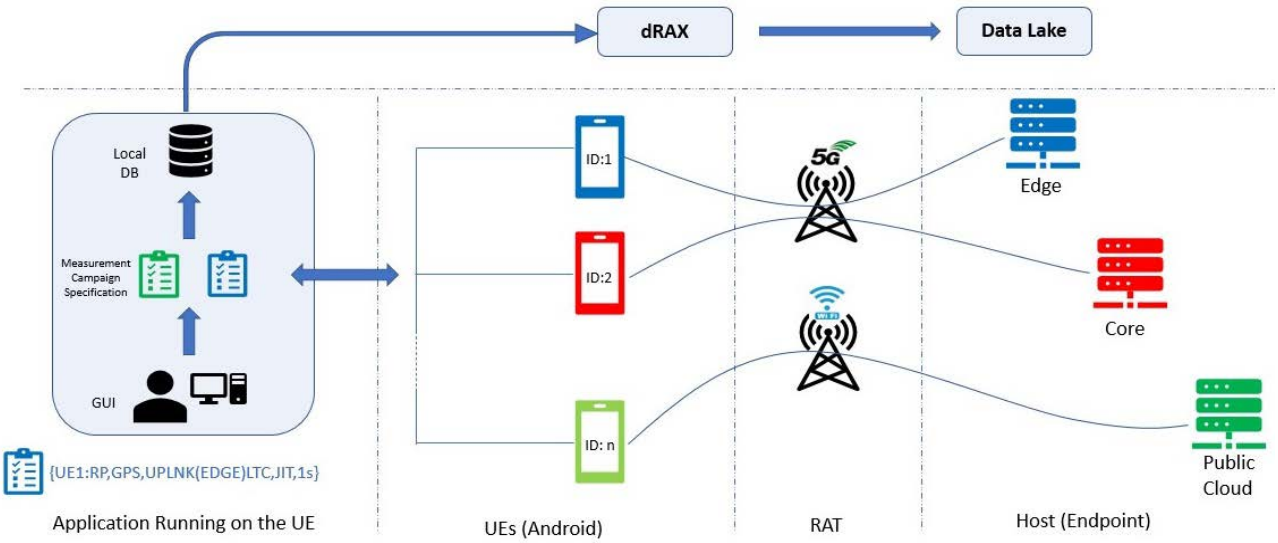


Figure 2.32. UE telemetry sub-system

Table 2-12. Main Measurements Obtained by the UE Telemetric Approach (in JSON format)

WAT/RAT	Source	REST API ID	Description
5G (SA and NSA) / LTE	UE/CPE	UE_Cellular_KPIs	Cellular parameters retrieved from UE radio such as Received Strength Reference Power (RSRP), Received Signal Signal to Noise Ratio (RSSNR), CellID, etc.
Wi-Fi	UE/CPE	UE_Wi-Fi_KPIs	Wi-Fi parameters such as Received Signal Strenght Indicator (RSSI), Channel, Link spped, Basic Service Set Identifier (BSSID), etc.
LiFi	UE/CPE	UE_LiFi_KPIs	LiFI parameters such as RSSI, Link speed, SSID, etc.
GNNS	UE/CPE	UE_GPS_coordinate	UE/CPE geographical location

The following figures represent the output of the measurement from UE telemetry application in JSON format for different wireless access technologies as described in Table 2-12. These are the measured parameters send to data lake for processing in the AI Engine. Each measurement has a timestamp that indicates the accurate time of measurement and the device's unique id. The “measurement_id” parameter will be used to make each measurement unique.

Figure 2.33 represents the measured parameters when the UE camped on an LTE cell; this includes the MNO information and Serving Cell (SC) information.

Figure 2.34 and Figure 2.35 represent the measured parameters for the 5G network in both stand-alone and non-stand-alone modes. When the UE register with 5G-NSA (EN-DC), both Master Cell (MC), which is an LTE cell and New Radio (NR) cell information captures.

```
"Device": {
        "TimeStamp": "2021-02-26 17:32:11",              //Time Stamp
        "Device_Id": "A8:DB:03:E1:E3:63",                //Unique Device ID
        "Measurement_Id": "514e841h533b584109d144356t", //Unique Measurement ID
        "SDK_version": "30",                             //SDK version of Android
        "Connection_Type": "Cellular",                   //Radio Access technology
        "Network_Type": "LTE (4G)",                      //Cellular Network Mode
        "Network_Operator": "00105",                     //Network Operator Name/PLMNID
        "SC_Cell_MCC": 001,                              //Mobile Country Code
        "SC_Cell_MNC": 05,                               //Mobile Network Code
        "SC_Cell_RSRP": -82,                             //Serving Cell RSRP
        "SC_Cell_RSRQ": -9,                              //Serving Cell RSRQ
        "SC_Cell_RSSNR": 0,                              //Serving Cell RSSNR
        "SC_Cell_ASU": 58,                               //Serving Cell ASU
        "SC_Cell_PCI": 5,                                //Serving Cell PCI
    }
```

Figure 2.33. Sample JSON output for LTE measurements

```
"Device": {
        "TimeStamp": "2021-02-26 17:32:11",              //Time Stamp
        "Device_Id": "A8:DB:03:E1:E3:63",                //Unique Device ID
        "Measurement_Id": "523e833h123b582209d142334t", //Unique Measurement ID
        "SDK_version": "30",                             //SDK version of Android
        "Connection_Type": "Cellular",                   //Radio Access technology
        "Network_Type": "NR (5G-SA)",                    //Cellular Network Mode
        "Network_Operator": "00105",                     //Network Operator Name/PLMNID
        "SC_Cell_MCC": 001,                              //Mobile Country Code
        "SC_Cell_MNC": 05,                                   //Mobile Network Code
        "SC_Cell_RSRP": -82,                             //Serving Cell RSRP
        "SC_Cell_RSRQ": -9,                              //Serving Cell RSRQ
        "SC_Cell_RSSNR": 0,                              //Serving Cell RSSNR
        "SC_Cell_PCI": 6,                                //Serving Cell PCI
    }
```

Figure 2.34. Sample JSON output for 5GNR Stand-Alone mode measurements

```
"Device": {
        "TimeStamp": "2021-02-26 17:32:11",              //Time Stamp
        "Device_Id": "A8:DB:03:E1:E3:63",                //Unique Device ID
        "Measurement_Id": "523e833h123b582209d142334t", //Unique Measurement ID
        "SDK_version": "30",                             //SDK version of Android
        "Connection_Type": "Cellular",                   //Radio Access technology
        "Network_Type": "NR (5G-NSA)",                   //Cellular Network Mode
        "Network_Operator": "00105",                     //Network Operator Name/PLMNID
        "MC_Cell_MCC": 001,                              //Mobile Country Code
        "MC_Cell_MNC": 05,                               //Mobile Network Code
        "MC_Cell_RSRP": -82,                             //Master Cell RSRP
        "MC_Cell_RSRQ": -9,                              //Master Cell RSRQ
        "MC_Cell_RSSNR": 0,                              //Master Cell RSSNR
        "MC_Cell_ASU": 48,                               //Master Cell ASU
        "MC_Cell_PCI": 6,                                //Master Cell PCI
        "NR_Cell_RSRP": -102,                            //NR Cell RSRP
        "NR_Cell_RSRQ": -7,                              //NR Cell RSRQ
        "NR_Cell_RSSNR": 0,                              //NR Cell RSSNR
        "NR_Cell_PCI": 6,                                //NR Cell PCI
    }
```

Figure 2.35. Sample JSON output for 5GNR Non Stand-Alone mode measurements

```
"Device": {
        "TimeStamp": "2020-08-26 11:30:56",              //Time Stamp
        "Device_Id": "6C:C7:EC:84:87:3A",                //Unique Device ID
        "Measurement_Id": "113e243h123b542209d936734t", //Unique Measurement ID
        "SDK_version": "29",                             //SDK version of Android
        "Connection_Type": "WIFI",                       //Radio Access technology
        "Network_Frequency": 5.54,                       //WIFI Frequency
        "WIFI_SSID": "5GUK",                             //WIFI SSID
        "WIFI_RSSI": -59,                                //WIFI RSSI
        "WIFI_Channel": 108,                             //WIFI Channel
        "WIFI_Linkspeed": 270,                           //WIFI Calculated Link Speed
        "WIFI_NetworkID": 2,                             //WIFI Network ID
        "WIFI_BSSID": "0c:f4:d5:26:ff:bc"                //WIFI Access Point BSSID
    }
```

Figure 2.36. Sample JSON output for WI-FI measurements

```
"Location": {                                    //UE location by GPS module
    "lat": "51.4498991",                         //Latitude
    "lon": "-2.5998304"                          //Longtitude
}
```

Figure 2.37. Sample JSON output for GPS location

If the UE attach to a Wi-Fi network, the following parameters will be captured as illustrated in Figure 2.36.

In most cases, the UE telemetry application requires access to the telephony manager of the UE to retrieve Radio parameters, nevertheless according to the GDPR, retrieving information related to the SIM card, including SIM number, IMSI, or other unique identifiers are not possible in commercial UEs like smartphones in recent days. So, the “Device_Id” parameter is a unique random identifier for the UE that does not include any personal information.

The UE telemetric system currently is under testing and redevelopment to support a broader measurement required by 5G-CLARITY use cases. The near RT RIC dRAX connection through “UE telemetry” xApp will be implemented in 5G-CLARITY D4.3.

2.2.2.4.3 End to end MPTCP xApp

The end-to-end metrics are obtained from the AT3S user plane function, namely the MPTCP socket. More specifically, the MPTCP socket in the 5G-CLARITY CPE and the MPTCP socket in the MPTCP proxy sitting behind the UPF in the 5G-CLARITY edge cluster provide the end-to-end MPTCP metrics. The telemetry data from these two endpoints will be made available to the “MPTCP” xApp residing in the near RT RIC, as illustrated in Figure 2.29.

The MPTCP module includes two APIs to configure and manage the MPTCP schedulers available. One of the APIs is composed of a set of Python functions which are executed locally. The other allows access to this functionality remotely.

Part of this API is devoted to gather metrics of the subflows owned by the MPTCP sockets. Each subflow is a TCP connection which uses the same linux kernel data structure than regular TCP connections. This information can be exposed by different means to a monitoring process.

There exist several ways to access the TCP and MPTCP related metrics locally. The set of parameters that can be accessed depends on the procedure and source used. The procedures identified to gather MPTCP telemetry data are the following:

- By modifying the MPTCP schedulers at kernel level. The schedulers implementation has access to the internal data structures which subflows use. Those parameters can be exposed as /proc/net/mptcp/<metric> files. The simplest way of including data in the <metric> file is to define

a structure with the proper kernel data type (for instance, as an array of integers). This data, such as the socket structure of each subflow, can be read and write from the schedulers code, each time they send a segment. However, although this method allows to expose any customised data, it requires changing every scheduler source code.

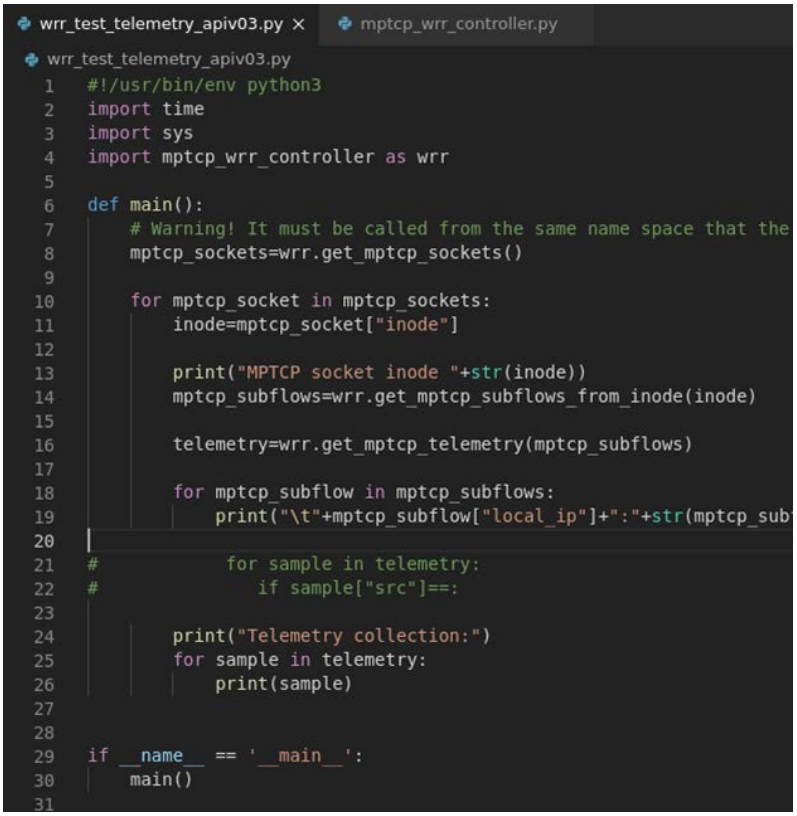
- By customising the /proc/net/tcp and /proc/net/mptcp files [24]. The kernel exposes some parameters of the active TCP flows (and therefore, MPTCP subflows) in the /proc/net/tcp special file. This file is generated at file ipv4_tcp.c of the linux kernel, and shows the inode to which the socket is bound, and parameters such as the congestion window and the retransmission timeout value. Additionally, file /proc/net/mptcp lists the active MPTCP sockets, indicating the inode to which are bound. Each subflow has the same inode that the MPTCP socket which created them. Although this is an easy method to get up to date metrics, it only provides two metrics. This drawback can be overcome by modifying the kernel source code to include additional information from the subflow socket structure. This may lead to unexpected parsing errors of legacy network monitoring applications, if they use these files.
- By using the "ss" tool of the IPRoute2 [25]. The "ss" tool can provide extended information of TCP sockets, including their inode, congestion window, segments sent, estimated egress throughput, estimated mean round trip time, its deviation, and other socket level information. This tool also exposes information about the configuration of the socket, such as the congestion control algorithm or the explicit congestion notification option status. The "ss" tool accesses to this kernel information by using netlink sock_diag messages [26], which is a more suitable way of communication between the kernel and user space processes. The only issue of using this method is that it adds overhead to the monitoring process, and may not be called at high rates.

The method selected to gather the MPTCP metrics is based on integrating with the Python API a customised "ss" version. This way, there is no need to change the expected format of /proc/net files, gathering rich information at the same time.

The Python API correlates /proc/net/mptcp sockets list with the per flow information offered by the "ss" tool.

As an example of use of the API, is shown in Figure 2.38. This API uses a JSON based representation of the telemetry data, so it is easier to integrate this into a REST based server. An example of output telemetry obtained from a scenario with a MPTCP socket using two interfaces, can be shown in Figure 2.39.

The near RT RIC dRAX connection through "MPTCP" xApp will be implemented in 5G-CLARITY D4.3.



wrr_test_telemetry_apiv03.py

```python
#!/usr/bin/env python3
import time
import sys
import mptcp_wrr_controller as wrr

def main():
    # Warning! It must be called from the same name space that the
    mptcp_sockets=wrr.get_mptcp_sockets()

    for mptcp_socket in mptcp_sockets:
        inode=mptcp_socket["inode"]

        print("MPTCP socket inode "+str(inode))
        mptcp_subflows=wrr.get_mptcp_subflows_from_inode(inode)

        telemetry=wrr.get_mptcp_telemetry(mptcp_subflows)

        for mptcp_subflow in mptcp_subflows:
            print("\t"+mptcp_subflow["local_ip"]+":"+str(mptcp_sub

#            for sample in telemetry:
#               if sample["src"]==:

        print("Telemetry collection:")
        for sample in telemetry:
            print(sample)


if __name__ == '__main__':
    main()
```

Figure 2.38. Example of use of the local MPTCP metrics Python API


```
{
 'src': '10.0.0.20:48696',
 'dst': '10.0.0.100:5001',
 'inode': 282035,
 'sk': '1af',
 'con_alg': 'cubic',
 'wscale': '7,7',
 'rto': '979',
 'rtt': 33.218,
 'rtt_var': 34.495,
 'mss': '1428',
 'pmtu': '1500',
 'rcvmss': '536',
 'advmss': '1428',
 'cwnd': '33',
 'ssthresh': '31',
 'bytes_sent': '109652',
 'bytes_acked': '109653',
 'segs_out': '80',
 'segs_in': '36',
 'data_segs_out': '77',
 'send': '11349028bps',
 'lastsnd': '28752',
 'lastrcv': '29386',
 'lastack': '28738',
 'delivered': '78',
 'busy': '647ms',
 'rcv_space': '14280',
 'rcv_ssthresh': '64108',
 'minrtt': '12.064'
}
```

```
{
 'src': '10.0.0.21%eth1:55787',
 'dst': '10.0.0.100:5001',
 'inode': 282035,
 'sk': '1b0',
 'con_alg': 'cubic',
 'wscale': '7,7',
 'rto': '522',
 'rtt': 79.046,
 'rtt_var': 110.539,
 'mss': '1428',
 'pmtu': '1500',
 'rcvmss': '536',
 'advmss': '1428',
 'cwnd': '12',
 'bytes_sent': '21420',
 'bytes_acked': '21421',
 'segs_out': '17',
 'segs_in': '17',
 'data_segs_out': '15',
 'send': '1734281bps',
 'lastsnd': '28738',
 'lastrcv': '29371',
 'lastack': '28513',
 'delivered': '16',
 'busy': '256ms',
 'rcv_space': '14280',
 'rcv_ssthresh': '64108',
 'minrtt': '2.135'
}
```


Figure 2.39. Example of metrics of two subflows pertaining to a MPTCP socket (inode=282035).

2.2.2.4.4 Data Lake xApp

As the access network telemetry or UE telemetry data will be continuously streamed with a specific frequency, the process of detecting new telemetry data and triggering data upload to the data lake should be supported. Such a process may be implemented by some file system monitoring APIs such as Watchdog Python API that observes changes in a directory and conducts any given process. When a new telemetry data file is added to a specific directory that is being monitored, a function that obtains the name of the recently added file may be used to trigger another function to upload the telemetry data to the relevant s3 buckets/items in the data lake.

```
class Handler ( FileSystemEventHandler ):
    @staticmethod
    def on_any_event ( event ):
        if event . is_directory :
            return None
        elif event . event_type == 'created ':
            print (" Received created event - %s.")
            time . sleep (5)
            fileAdded ()
        elif event . event_type == 'modified ':
            print (" Received modified event - %s.")
```

```
def getLatestFileName ():
    list_of_files = glob . glob ('C:/ Users /.../*. db ')
    latest_file = max ( list_of_files , key=os. path . getctime )
    return latest_file
```

Figure 2.40. Example scripts for Watchdog file system event handling and file name obtaining.

The near-RT RIC dRAX connection to data lake through “Data Lake” xApp will be implemented in the next deliverable D4.3.

3 5G-CLARITY ML Algorithms

In this section an initial implementation of the ML algorithms proposed in 5G-CLARITY D4.1 [1] are described. The 5G-CLARITY leverages different ML algorithms to support automatic network management, as well as to support different network functionalities. The initial implementation of the following algorithms is presented in this section:

- **Predicting SLA violations/success rate (Section 3.1).** The proposed algorithm uses echo state networks (ESNs) to forecast real-time traffic value, which will be occurring in the next time step. Since the network traffic shows a chaotic time-series properties, ESNs with very sparse connection among the reservoir neurons are shown to be very effective in predicting the future values. After the initial input normalization, the internal parameter optimization for our ESN framework is presented. Lastly, the time series forecasted values used to detect a potential SLA violation in the next time step.
- **RT RIC: AT3S traffic routing/handover (Section 3.2).** This focuses on developing the model-based part of the hybrid model that we proposed in the previous deliverable. The model-based part process input data that are available from indoors, e.g., Wi-Fi and LiFi RSSIs as well as image data from Closed Circuit Television (CCTVs). The output of the model is a prediction of a metric that is a function of locations of UEs at the next time step. We apply an ensemble deep learning architecture consisting of Convolutional Neural Networks (CNNs) and fully connected neural networks and investigate the reliability of the model-based part.
- **RAN slicing in multi-tenant networks (Section 3.3).** The proposed algorithm focuses on how to distribute the available capacity in a multi-cell NG-RAN infrastructure among different slices where each slice provides service to a different tenant. The algorithm targets to fulfil the SLA requirements of each tenant and to achieve an efficient utilisation of the radio resources.
- **Optimal Access Networks (Section 3.4).** This focuses on the optimal multi-WAT access network problem by describing the initial modelling and designing of an AI based architecture. The proposed approach will predict access network states to recommend a set of optimal multi-WAT access network policy that maximize the QoS and mobility. Section 3.4 introduces the problem statement and formulation as linear programming, the initial design of our model followed by some initial results and evaluations.
- **Indoor ranging with LoS awareness (Section 3.6).** The indoor NLoS-aware ranging is an algorithm relying mainly on the Deep Neural Network (DNN) approach as well as the channel impulse responses collected in an office environment. In particular, it first detects the link condition, i.e. LoS or NLoS, and then depending on that employs a specific DNN model to estimate the distance of the user.
- **Resource provisioning in a multi-technology RAN (Section 3.7).** This algorithm attempts to solve the radio resource provisioning problem in an industrial network scenario with a multi-technology RAN in which URLLC and eMBB services are supported. Unlike the algorithm proposed in Section 3.3, where the network is shared among different tenants, in this use case an industrial network is considered as a standalone NPN managed by a unique private operator.
- **Transport network setup (Section 3.8).** This algorithm finds a satisfiable configuration for an asynchronous TSN transport network to accommodate the traffic of the different 5G-CLARITY slices while ensuring its end-to-end delay and jitter budgets for all of them.
- **Adaptive AI-based defect-detection in a smart factory (Section 3.9).** This algorithm aims to have a zero-defect manufacturing system. It focuses on a production line in a smart factory where defective pieces on the production line have been detected by an AI-based defect-detection algorithm. Once

Table 3-1. ML Use Case Execution Times and Deployment Location.

Use Case	ML Model Type	Planned Execution Time	Ideal Location of Deployment
Predicting SLA violations/success rate	ESN	1-200 ms	PCF/UPF
RT RIC: AT3S traffic routing/handover	RL	100 ms	UPF
RAN slicing in multi-tenant networks	RL	Minutes	MANO layer
Optimal network access problem	RL	20-300 ms	UE
Indoor ranging with nLOS awareness	SVN, DNN	200-500 ms	Localization server / Edge cluster
Resource provisioning in a multi-technology RAN	RL	Minutes	Non-RT RIC
Transport network setup	RL	Minutes/days	Slice Manager, NFVO, VIM, SDN Controller
Adaptive AI-based defect-detection in a smart factory	DNN (YOLO v3)	Seconds	Edge cluster

a defective item is detected, an automatic intervention in order to stop the line and take the defective pieces out of the line is triggered.

- The details of all of the initial implementation of these ML algorithms, as well as the initial evaluation is given in the corresponding subsections. In

Table 3-1, a summary of the ML model type, execution, time as well as, ideal location of deployment is summarized. The use cases employ state of the art ML algorithms, including Echo State Networks (ESN), Reinforcement Learning (RL), Support Vector Machine (SVM) and Deep Neural Networks (DNN).

3.1 Predicting SLA violations/success rate

The service-oriented architecture (SOA) provides modularity and granularity to the 5GNR network framework, where the control plane functionalities and data repositories at the core network are comprised of a set of interconnected network functions (NF). One of the key features of 5GNR is the guaranteed high-level quality of service (QoS) for diverse user requirements by using the same conflicting infrastructure. This challenge has been overcome by the concept of network slicing, which suggests the partitioning of legacy network resources into virtual units to enhance the overall system efficiency and flexibility. Often, the QoS levels could be translated into service level agreements (SLAs), which specify the agreements between customers and service providers. In other words, an SLA emphasizes the responsibilities of each party along with the underlining performance standards. It is important to note that the legal consequences e.g., penalties for any breaches could have a detrimental effect on the mobile virtual network operators (MVNOs). On the contrary, there might be bonuses for exceeding the agreed level of service performance.

The SLA lifecycle is an important part of the provision of 5GNR related services. The broad adoption of the software defined network (SDN) in 5G has also affected the evolution of the SLA lifecycle. In Figure 3.1 the phases of an SLA's lifecycle are depicted in the context of an SLA management framework. As can be seen from the figure, dynamic SLA management consists of four phases: *(i)* SLA template generation, *(ii)* slice instantiation, *(iii)* agreement creation and (iv) SLA monitoring. In the first phase, selection of four key components is required for successful SLA template generation: *(i)* network services, *(ii)* SLA name, (iii) valid future expiration date and *(iv)* at least one service level objective (SLO). The SLA template, in the form of a descriptor, is generated and the parameters are defined by commercial officer on behalf of the network operator, where the descriptor is stored in a database. In the next phase, a negotiation between the customer and commercial officer takes place. Accordingly, a NS is requested by the customer from the commercial officer and the available network services are presented along with the SLA templates.

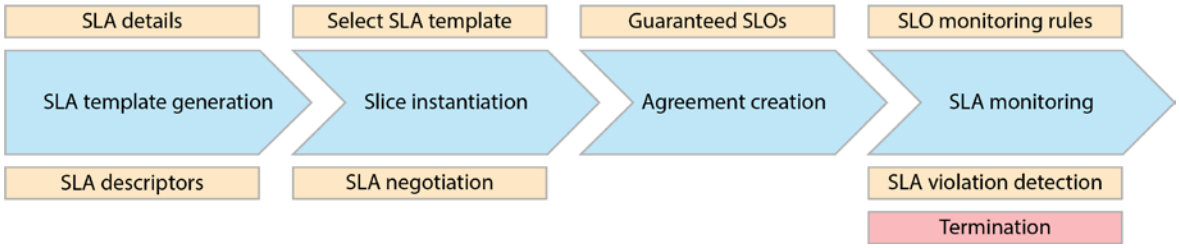


Figure 3.1. SLA Lifecycle and management framework in 5GNR networks.

Then, a network service instantiation within a specific network slice occurs in the MVNOs infrastructure. In phase three, the official association between the SLA and the instantiated slice takes place. Lastly, in phase four, monitoring of the SLA is initiated. Moreover, the monitoring rules such as assessment on settlement, termination commitments, customer care and data related processes are also taken care of by the monitoring manager. Once a violation of the SLA is determined by the monitoring manager; a violation alert is generated and stored in the database. Furthermore, the termination of the entitlement between MVNOs and the network service customer, which includes the legal basis are also taken care of within this phase.

3.1.1 Methodology

To satisfy the customer demand as well as ensuring sufficient system performance, estimation and/or prediction of the required resources is of vital importance. As the whole operation dynamics of the 5GNR network could be thought of as a living organism, both the detection and reaction to potential SLA violations must be prompt. Moreover, the estimation of the upcoming potential traffic along with the possible violations will help MVNOs to generate a significant financial benefit. Consequently, echo state networks (ESNs) are adopted in this work to predict the SLA violations that might possibly happen in the lifecycle of SLAs. The reason behind the selection of ESNs for this task is two-folds. Firstly, highly unpredictable, and potentially non-linear input output relationships inside the 5GNR functional blocks, which yields a mathematically intractable complex network structure. Since the development of an accurate mathematical model for a system of this scale is not an easy task, the data-driven structure of ESNs, as opposed to model-driven ones, play a significant role in black-box optimization for the given application. Secondly, the sparse interconnection requirement among neurons in ESN, which practically yields significantly reduced computational overheads compared to other conventional systems. For instance, artificial neural networks (ANNs) with multiple and large number (known as deep learning) of hidden layers.

In Figure 3.2, the adopted ESN structure to detect/estimate SLA violation/success is depicted. Similar to conventional ANN structures, ESN consist of three layers: input, intermediate and output, which consist of K, N_x and L neurons, respectively.

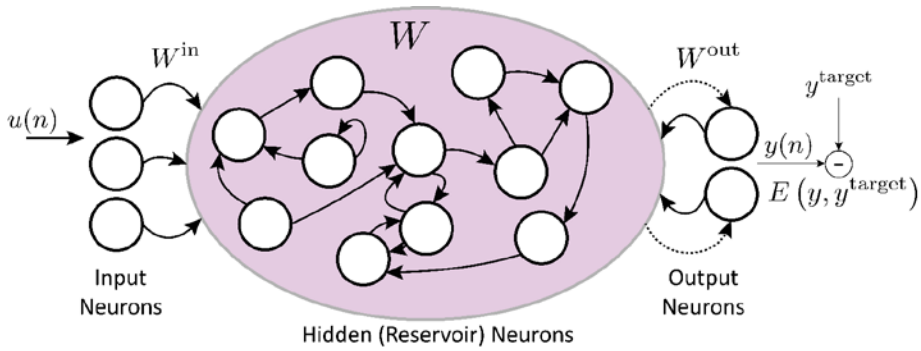


Figure 3.2. The echo state network structure considered for SLA violation/success rate predictions. The solid and dashed lines with arrows indicate the fixed and trainable connections, respectively.

In the first layer, the input vector whose activation in time step $n = 1,2,\cdots,T$ is denoted by $\boldsymbol{u}(n) = [u_1(n) \quad u_2(n) \cdots \quad u_K(n)]^{\mathrm{T}}$ is fed to the network. The parameter T denotes the number of data points in the training dataset. Here, the inputs will be both low- and high-level descriptions/requirements, and the policies. The low-level descriptors are the infrastructure related parameters, whereas the high level involves virtual network functions (VNFs) and NS network load and slice traffic parameters. In the intermediate stage, N_x hidden or in other words, reservoir activation vector $\boldsymbol{x}(n) = [x_1(n) \quad x_2(n) \cdots \quad x_{N_x}(n)]^{\mathrm{T}}$ is employed. It is important to note that the biggest advantage of ESNs come from this stage, where unlike fully connected ANNs, a sparsely/loosely connected neural network is employed in the reservoir. It should also be noted that the internal connecting weights in the reservoir are initialised randomly, where the asymptotic state convergence occurs as time passes after the effect of initial conditions have vanished. For a large ESN network, it could take a few hundred steps to get rid of the initialisation contamination. Lastly, at the output stage L readout neurons $\boldsymbol{y}(n) = [y_1(n) \quad y_2(n) \cdots \quad y_L(n)]^{\mathrm{T}}$ are employed. The $N_x \times K$ connection weights matrix between the input and reservoir neurons, which is denoted by $\boldsymbol{W}^{\mathrm{in}}$, is also depicted in Figure 3.2 .Similarly, the $N_x \times N_x$ reservoir weights matrix is given by $\boldsymbol{W}$. Finally, the $L \times (K + N_x)$ connection weights matrix between the input and reservoir neurons to output neurons is given by $\boldsymbol{W}^{\mathrm{out}}$. The main goal of the ESN is to learn a model, $\boldsymbol{y}^{\mathrm{target}}(n) \in \mathbb{R}^L$, where the predicted output, $\boldsymbol{y}(n)$, matches the target output as accurately as possible by minimizing the error, $E(\boldsymbol{y}, \boldsymbol{y}^{\mathrm{target}})$. Please note that our expectation from the ESN is to gain the ability to generalize the model and predict and accurate results even for completely new data. Typically, root-mean-squared-error (RMSE) is used as the error expression, $E(\boldsymbol{y}, \boldsymbol{y}^{\mathrm{target}})$,

$$E(\boldsymbol{y}, \boldsymbol{y}^{\mathrm{target}}) = \frac{1}{L}\sum_{i=1}^{L}\sqrt{\frac{1}{T}\sum_{n=1}^{T}\left(y_i(n) - y_i^{\mathrm{target}}(n)\right)^2}$$

where the above expression is also averaged over L dimensions of the output signal. Moreover, the typical update equations for ESN could be given as follows:

$$\boldsymbol{x}(n) = (1-\alpha)\boldsymbol{x}(n-1) + \alpha\tilde{\boldsymbol{x}}(n)$$

where $\tilde{\boldsymbol{x}}(n) \in \mathbb{R}^{N_x}$ represents the reservoir activation update vector at a time step n. Furthermore, the parameter $\alpha \in (0,1]$ is the leaking rate that determines the speed of the reservoir update dynamics. To obtain $\tilde{\boldsymbol{x}}(n)$, the element-wise activation function of $\tanh(\cdot)$ could be applied by

$$\tilde{\boldsymbol{x}}(n) = \tanh\left(\boldsymbol{W}^{\mathrm{in}}[1; \boldsymbol{u}(n)] + \boldsymbol{W}\boldsymbol{x}(n-1)\right)$$

where the operation $[\cdot\,;\cdot]$ denotes the vertical vector/matrix concatenation. Note that the $\tanh(\cdot)$ is the most popular activation function used in the literature, however, other sigmoid activation functions could also be used. After the ESN reservoir activation update, the output vector can be expressed by $\boldsymbol{y}(\boldsymbol{n}) = \boldsymbol{W}^{\mathrm{out}}\boldsymbol{X}$, where $\boldsymbol{X} = [\boldsymbol{1}; \boldsymbol{u}(n); \boldsymbol{x}(n)]$. The optimal output weights can also be obtained as follows:

$$\boldsymbol{W}^{\mathrm{out}} = \boldsymbol{y}^{\mathrm{target}}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}} + \beta\boldsymbol{I}\right)^{-\boldsymbol{1}}$$

where the parameter β is a regularization coefficient that is utilized to mitigate the over-fitting as well as to maintain the feed-back stability. The matrix $\boldsymbol{I}$ denotes the $(K + N_x + 1) \times (K + N_x + 1)$ identity matrix. An additional non-linearity can be applied to $\boldsymbol{y}(n)$ by using feedback connections $\boldsymbol{W}^{\mathrm{feedback}} \in \mathbb{R}^{N_x \times L}$ as follows:

$$\tilde{\boldsymbol{x}}^{\mathrm{feedback}}(n) = \tanh\left(\boldsymbol{W}^{\mathrm{in}}[1; \boldsymbol{u}(n)] + \boldsymbol{W}\boldsymbol{x}(n-1) + \boldsymbol{W}^{\mathrm{feedback}}\boldsymbol{y}(n-1)\right)$$

3.1.2 ESN based time series forecasting

In this subsection, the details of the dataset that will be used in the ESN based time series prediction and SLA violation detection will be explained.

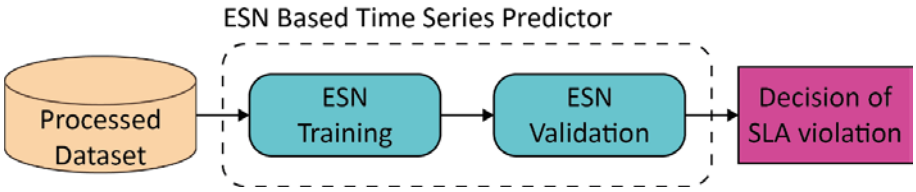


Figure 3.3. The architecture of the ESN based SLA violation detection system.

The block diagram of the utilized architecture for time series forecasting and SLA violation detection is given in Figure 3.3. As depicted in the figure, the potential SLA violation decision is made by using the ESN based time series forecasting results. The main reason behind this is the flexibility and adaptability requirements of the current 5G networks. Accordingly, the AI/ML aided SLA anomaly detection would require repetition of both the training and validation phases with the change in the SLA descriptor and template of MVNOs every single time. Therefore, our approach decouples the prediction phase and SLA descriptors, where only the customer key performance indicators (KPIs) are predicted with the aid of ML.

The dataset dimension/feature extraction is the first step to obtain the "reduced dataset". In this work, 5G radio access network (RAN) traffic statistics dataset generated by COSMOTE [30] are used as our input data. In the adopted dataset, the measurement-based data is collected for both the UMTS and LTE setup with 15 and 11 base stations (BSs) / micro-BSs, respectively. Since the main goal is to predict potential SLA violations in a given slice to avoid penalties that could affect the infrastructure owner/operator, we utilized the LTE measurement dataset without loss of generality.

In the LTE dataset, there are 26 parameters, where both the parameters and their descriptions are provided by the files given in [30]. For the sake of simplicity of our analysis, two of these parameters are employed:

(i) Time stamp, which gives the absolute time of the measurement sample, and
(ii) Downlink data traffic, which measures the downlink (DL) data traffic, in Mbps, for 15 minutes of intervals,

In Figure 3.4, the aggregated DL data traffic in GBytes, which is obtained by the combination of the 9 BS/micro-BS measurements, is depicted. Please note that the reason behind this is the various sizes of the measurement data that are available for each BSs. Thus, to capture the maximum number of BS measurements as well as the time samples, the BSs (BS1, BS4, BS5, BS6, BS7, BS8, BS9, BS10_reconf and BS12), which yield at least 4211 time-samples are chosen. Then, the aggregated DL traffic input, which will be used as our stationary time series at the input stage of ESN is obtained.

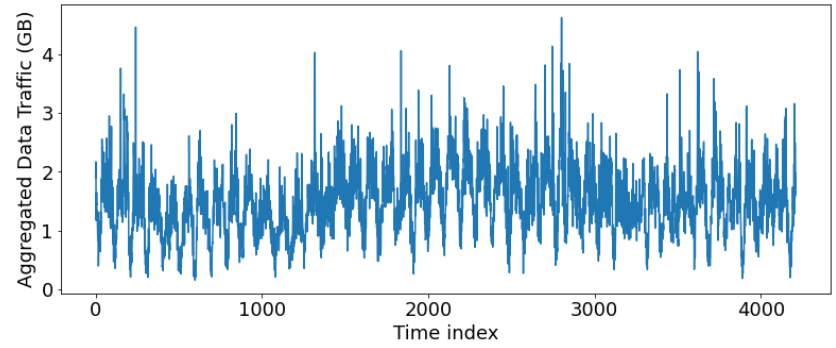
3.1.2.1 Input pre-processing

It is important to note from Figure 3.4(a) that the DL aggregated data traffic samples follow an approximate Gaussian distribution with the mean and variance values of 1.49 and 0.58, respectively. Since the input data normalization plays an important role for a faster and more accurate convergence of the gradient descent algorithm, the input data has been normalized as depicted in Figure 3.4(b).

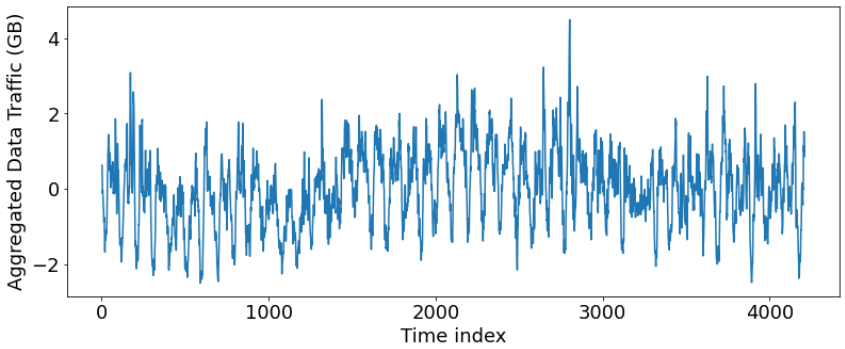
It is important to note that there are two major ways of data normalization:

- Linear range transformation of the data within the interval $[l \quad u]$, where the minimum and maximum values of the dataset are mapped into the values l and u, respectively, as follows:

$$\boldsymbol{d}_{\text{normalized}} = \frac{\boldsymbol{d} - \boldsymbol{d}_{\min}}{\boldsymbol{d}_{\max} - \boldsymbol{d}_{\min}} \times (u - l) + l$$



(a)



(b)

Figure 3.4. The aggregated data traffic data in; (a) raw and (b) statistically normalized and smoothed format.

where the (4211×1) measurement vector is represented by $\boldsymbol{d} = [d_0 \quad d_1 \quad \cdots \quad d_{L_t+L_v-1}]$. The parameters L_t and L_v denote the length of the training and validation sets, respectively. Moreover, the minimum and maximum values of the dataset could be found by $\boldsymbol{d}_{\min} = \min\{\boldsymbol{d}\}$ and $\boldsymbol{d}_{\max} = \max\{\boldsymbol{d}\}$, respectively.

- Another important input normalization is the statistical normalization to obtain the mean and standard deviation values of zero and one, respectively, which could be represented by,

$$\boldsymbol{d}_{\text{normalized}} = \frac{\boldsymbol{d} - \mu}{\sigma}$$

where the mean and the standard deviation of the adopted dataset vector are denoted by μ and σ, respectively. In our simulations, the input normalization method (ii) is adopted since it yields a better normalized root-mean-squared-error performance (NRMSE) performance. Moreover, the data denoising is another key data pre-processing procedure that plays a significant role in the system performance. As can be seen from Figure 3.4(a), the aggregated DL traffic data contains instantaneous fluctuations, which could confuse our ML model and yield a performance degradation for the predictions. To overcome this problem, a rolling window based denoising filter is employed, where the data after denoising with the window size of $w = 4$ is depicted in Figure 3.4(b).

3.1.2.2 Predicting the DL aggregated traffic values

Forecasting the DL aggregated traffic value is of vital importance for the SLA violation detection, where we can flag and take an action accordingly even before an anomaly in traffic causes a violation. To execute our time series prediction simulations, we have created a framework in Python 3.9.4.

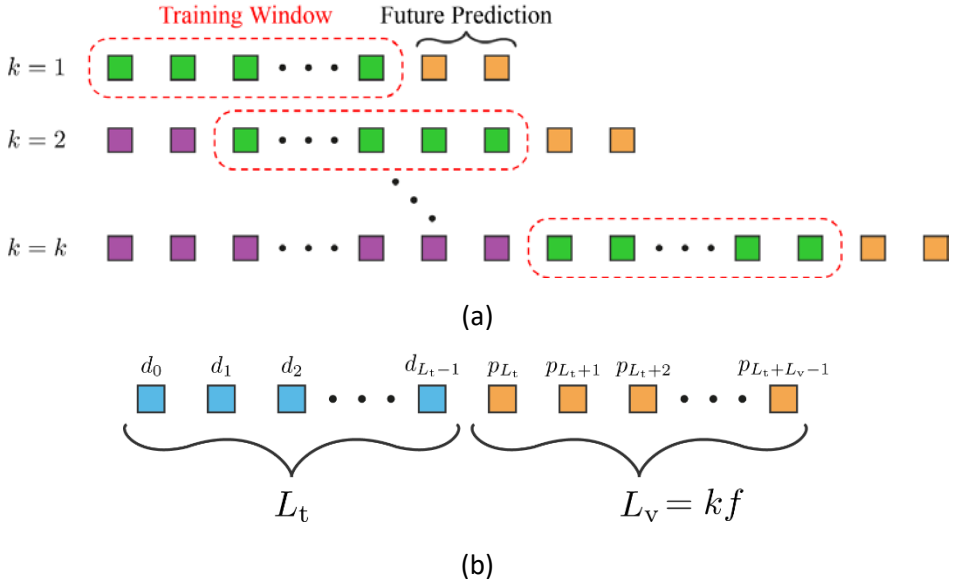


Figure 3.5. ESN based time series forecasting methodology; (a) step-by-step, (b) data and resultant prediction vector frames.

We have also used an ESN package obtained from the open source pyESN library [31] to utilize our reservoir computing (RC) based solution. Specifically, we aim to forecast f time samples ahead by using the $L_t \times 1$ training vector. After k steps, we will compare the obtained forecast vector with the $L_v \times 1$ validation vector, where $L_v = kf$. The step-by-step forecasting procedure is given in Figure 3.5.

After the forecasting procedure, the $L_v \times 1$ prediction vector $\boldsymbol{p} = [p_{L_t} \quad p_{L_t+1} \quad \cdots \quad p_{L_t+L_v-1}]$ is compared with the validation vector, $\boldsymbol{v} = [d_{L_t} \quad d_{L_t+1} \quad \cdots \quad d_{L_t+L_v-1}]$, to obtain a RMSE performance as follows:

$$\mathrm{RMSE} = \sqrt{\mathrm{E}\{(\boldsymbol{p}-\boldsymbol{v})^2\}} = \sqrt{\mathrm{E}\{(\boldsymbol{e})^2\}} = \sqrt{\frac{1}{L_v}\sum_{i=1}^{L_v} e_i^2}$$

The length of the training (L_t) and validation (L_v) as well as the ESN specific parameters, which play a significant role in RMSE performance of time series forecasting, are given as follows:

- n_{input} : Number of inputs / input dimensions
- n_{output} : Number of outputs / output dimensions
- N_x : Number of neurons in the reservoir
- r : Random seed for the pseudo-random number generator
- s : Sparsity of the reservoir neuron connections. In other words, the proportion of the recurrent weights which are set to zero
- ρ : Spectral radius of the recurrent weights' matrix
- λ : Noise magnitude, which is added to each neuron for the regularization

In Table 3-2, the simulation parameters. which are adopted in our ESN based time series forecasting and SLA violation detection simulations are summarized.

Table 3-2. Adopted Generic Simulation Parameters for our ESN Simulations.

Parameter	Value(s)
$\boldsymbol{n}_{\text{input}}$	1
$\boldsymbol{n}_{\text{output}}$	1
$\boldsymbol{N}_{\text{x}}$	300
$\boldsymbol{S}$	0.3
$\boldsymbol{\rho}$	1.2
$\boldsymbol{\lambda}$	0.0005
$\boldsymbol{L}_{\text{t}}$	3788 ($\approx \mathbf{90}\%$)
$\boldsymbol{L}_{\text{v}}$	420 ($\approx \mathbf{10}\%$)
$\boldsymbol{f}$	1
$\boldsymbol{k}$	$\boldsymbol{L}_{\text{v}}$
$\boldsymbol{w}$	4

It is important to note from the above table that the prediction step length, f, is taken to only predict the next step, which corresponds to the next 15 minutes of the DL data traffic. The main reason behind this is due to the knowledge that after successful training the model will be capable of predicting future values with high accuracy. However, as f increases, the prediction accuracy drops as the ESN is not using the fresh data to estimate the trend of the curve. Furthermore, the one time-step forecast uses all the historic data for the prediction, which naturally means a better accuracy for the given system. Therefore, the $f = 1$ case is investigated in this work. The scenarios with wider prediction steps and its effect on the system performance will be addressed in work as part of a future deliverable.

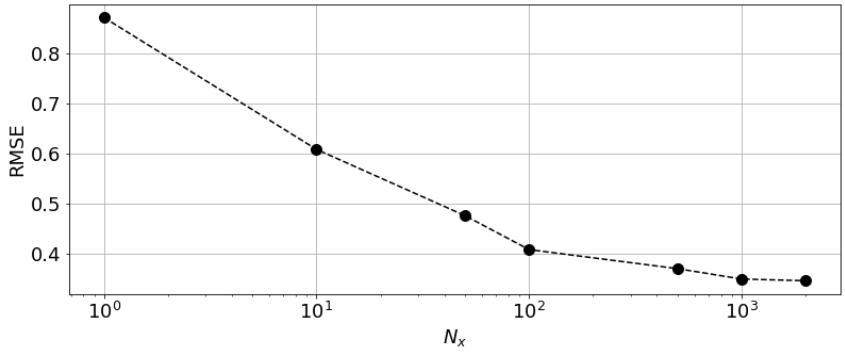
3.1.2.3 ESN-based time series forecasting performance

In this subsection, we will investigate the relationship between the values of the parameters N_{x}, S, $\boldsymbol{\rho}$, λ and the RMSE performance of the ESN based predictions, which are also depicted in Figure 3.6.

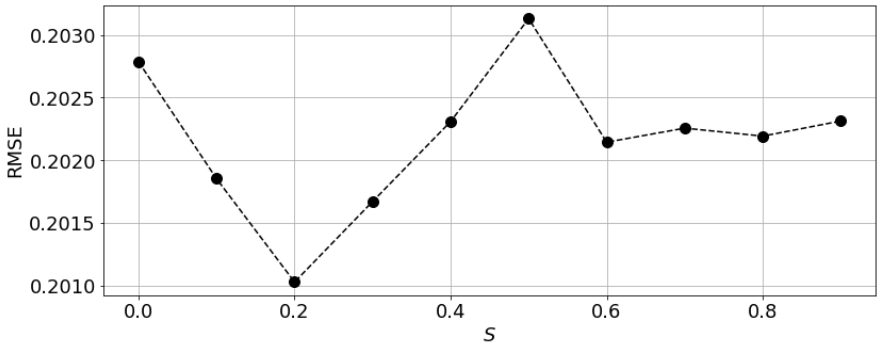
In Figure 3.6(a), the RMSE performance of ESN is depicted with respect to the reservoir size (N_{x}). Accordingly, N_{x} is chosen to be $[1 \quad 10 \quad 50 \quad 100 \quad 500 \quad 1000 \quad 2000]$. Practically, reservoir sizes of 20, 50, 100, 1000, and even 10 000 are covered by research outlined in the literature. As a rule of thumb formula, the value of the reservoir size could be determined by using L_{t} as follows [32]:

$$N_x \leq \frac{L_{\text{t}}}{2}$$

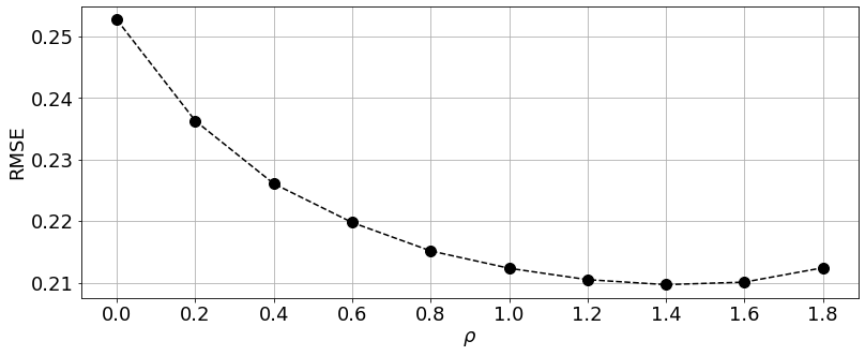
From the figure, the RMSE performance of the ESN network increases with the increase in the reservoir size, which is also the case reported by many research items in the literature. The minimum RMSE value is achieved when $N_{\text{x}} = 2000$ in our simulations. However, note that the performance gain obtained with the increase in reservoir size could only be achieved as long as the appropriate measures for stability and over-fitting are taken into consideration. The term over-fitting explains situations where the ML model fails to generalize the details and noise in the training dataset. Furthermore, the computational complexity in the training phase of the ESN compared with RNNs is significantly low, which yields up to the applications with the reservoir size of $N_{\text{x}} = 10^4$ without increasing the required computational expenses significantly. However, the marginal performance gain with the increase of N_x is not significant after $N_{\text{x}} = 100$ in our empirical results. This is due to the higher under-training of the reservoir neurons in our ESN framework according to the expression above.



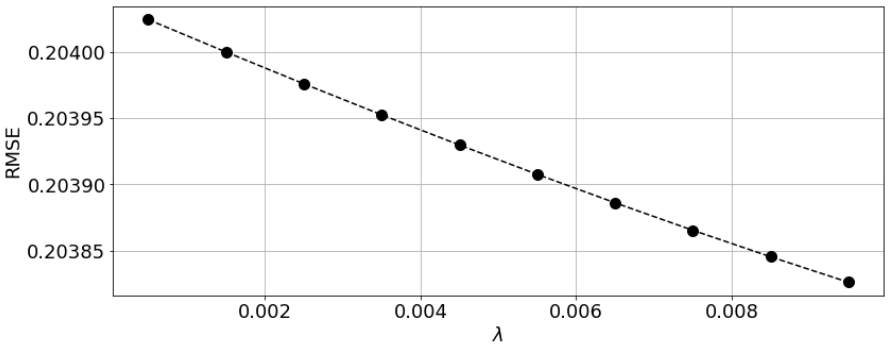
(a) $S = 0.2$, $\rho = 1.4$, $\boldsymbol{\lambda} = 0.001$ and 90 % training - 10% validation configuration.



(b) $N_x = 100$, $\rho = 1.2$, $\boldsymbol{\lambda} = 0.0005$ and 90 % training - 10% validation configuration.



(c) $N_x = 100$, $S = 0.2$, $\boldsymbol{\lambda} = 0.001$ and 90 % training - 10% validation configuration.



(d) $N_x = 100$, $S = 0.2$, $\rho = 1.4$, and 90 % training - 10% validation configuration.

Figure 3.6. RMSE performance of the designed ESN framework under various parameter choices.

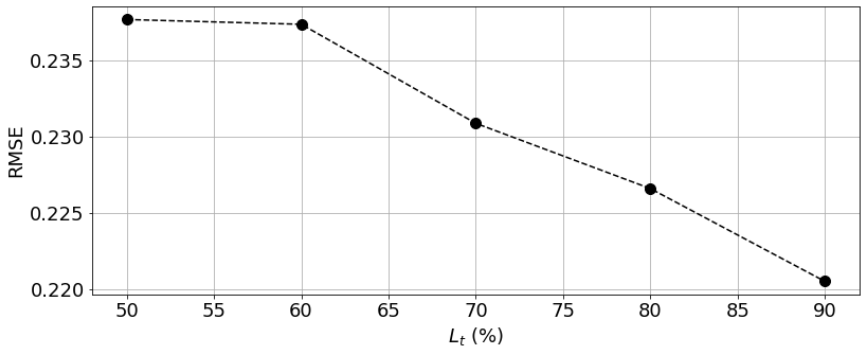


Figure 3.7. The RMSE performance of the designed ESN framework for various training set sizes.

In Figure 3.6(b), the RMSE performance versus the reservoir sparsity (S) of our ESN framework is depicted. It is important to note that irrespective of the reservoir size, the sparsity factor in the weight matrix is generally set to be low (sparse configuration) on average to exploit random sparse echo state connections and to reduce the computational complexity. The empirical results obtained by our simulations for $S \in \{0.0,\ 0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.6,\ 0.7,\ 0.8,\ 0.9\}$. As shown in figure that the RMSE performance shows a couple of local peaks and troughs for various sparsity values. More importantly, the minimum RMSE value is obtained when $S = 0.2$ in our simulations.

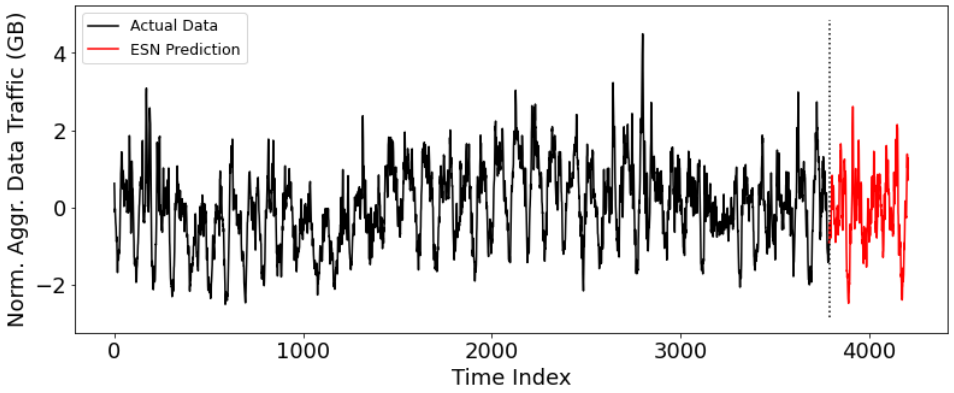
The spectral radius (ρ), is the parameter which defines the maximum value of absolute eigenvalues of the reservoir matrix $\boldsymbol{W}$. In other words, the width of the non-zero entry distribution of matrix $\boldsymbol{W}$ is defined by ρ. In the literature, it has been stated that in most cases, $\rho < 1$ ensures the echo state property [33]. However, note that the converse is also correct in some cases. It is also important to emphasize that the larger radius yields a better performance in tasks, which requires significant input history utilization. The result of our simulation is depicted in Figure 3.6(c) for the spectral radius values of $\rho \in \{0.0\ \ 0.2\ \ 0.4\ \ 0.6\ \ 0.8\ \ 1.0\ \ 1.2\ \ 1.4\ \ 1.6\ \ 1.8\}$. In our simulations, the minimum RMSE is achieved when the spectral radius value of $\rho = 1.4$ is adopted.

Another crucial parameter for regression-based regularization for stability purposes is the noise magnitude ($\boldsymbol{\lambda}$). Accordingly, the RMSE performance with respect to the noise magnitude is simulated and the result is depicted in Figure 3.6(d). As depicted in the figure, increase in the noise figure enhances the system performance due to the higher system stability.

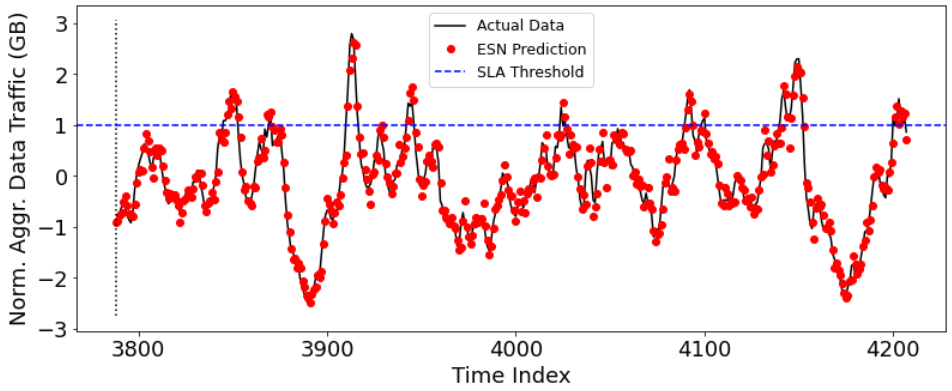
Lastly, in Figure 3.7, the RMSE performance of the ESN framework is compared against the size of the training set. Accordingly, the training set used in the simulations are proportioned into $\boldsymbol{x}\%$ training set and $(\boldsymbol{100 - x})\%$ validatin set, where $\boldsymbol{x}$ denotes the $\boldsymbol{x}$-axis values in the figure. As expected, the increase in the training size increases the system performance indefinitely since more information lets the ESN generalize the input dataset model to a greater extend. However, this is not the case for both $\boldsymbol{L_t = 50\%}$ and $\boldsymbol{L_t = 60\%}$. The main reason behind this is the lack of sufficient information about the dataset, which prevents the ESN to learn about the dataset and present any learning capacity.

3.1.2.4 SLA violation prediction from ESN-based time series forecasts

In this subsection, we will detail the utilization of the previously mentioned ESN time-series forecasting simulator for potential SLA violation detection, which will prevent any penalties onto the MVNOs. Accordingly, the SLA violation detection decision block, please refer to Figure 3.3, asks the following question: "*Will there be an event where the rule is satisfied in future f steps?*". Here, the rule will be defined as the SLA violations and f number of steps that will be predicted in the future. For the sake of simplicity, the SLA violation rule/threshold is defined by, $V_{\text{th}} = \mu + \gamma\sigma$, where the parameter γ represents the arbitrary parameter to define the range of the SLA violation.



(a)



(b)

Figure 3.8. SLA violation detection by ESN based time series predictions; (a) training (black) and validation (red) sets, (b) validation window with both the actual data and the ESN predictions.

In Figure 3.8, the ESN time-series forecasting, and SLA violation detection procedures are depicted. Both the training and validation sets (black curve) as well as the future ESN predictions (red curve) are given in Figure 3.8(a). As can be seen from the figure, the ESN prediction values are able to follow the trend of the actual data very closely. This shows us that the ESN parameters are configured well enough to learn the general model of the aggregated traffic data. Similarly, Figure 3.8(b) focuses only on the validation set portion of Figure 3.8(a). As can be seen from Figure 3.8(b), the SLA threshold is also depicted by a dashed blue curve, where the data points above this threshold will be labelled as an "*SLA violation*".

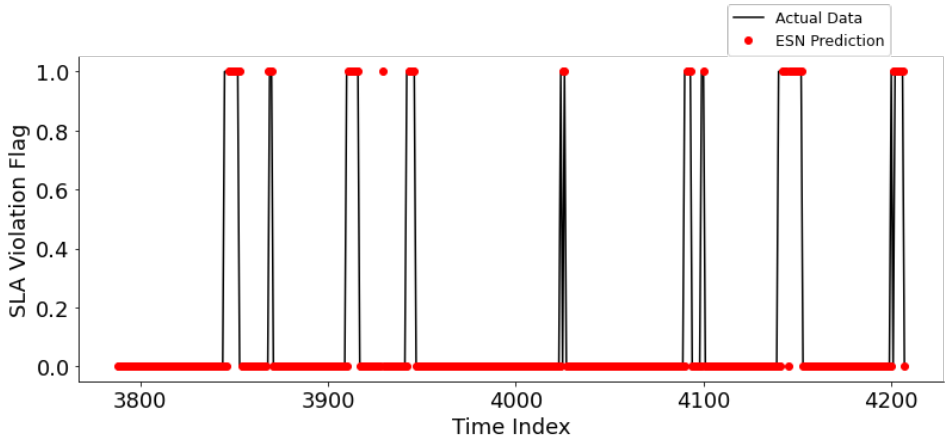


Figure 3.9. Detected SLA violations; actual (black), predicted by ESN (red).

It is also important to note that future predictions are matching the actual data very well in capturing the fast fluctuations as well as the peak values. The accuracy of the time domain predictions is measured by the RMSE metric. Thus, for the prediction depicted in Figure 3.8, the RMSE between the actual data and the ESN predictions becomes 0.217.

The SLA violation detection performance of the ESN is given in Figure 3.9. As can be seen from the figure, the time indexes which yield an SLA violation are labelled as "1" where the others are labelled as "0". It can be clearly seen from the figure that the actual and ESN predicted SLA violations match very closely. To assess the performance of the ESN in terms of the SLA violation detections, another metric, namely SLA violation detection ratio (SVDR) is used. Accordingly, SVDR can be calculated by

$$\text{SVDR} = \frac{\#\text{ of predicted SLA violations}}{\#\text{ of actual SLA violations}}$$

where the number of predicted and actual SLA violation values are obtained from the ESN predictions and actual data, respectively. Therefore, for the results depicted in Figure 3.9, the SVDR of 0.878 is achieved. This achieved SVDR value practically means that our ESN framework is able to predict the SLA violations for the next time step with more than 87% accuracy. Further optimizations and performance enhancements on the ESN are carried to deliverable D4.3 for the sake of readability.

3.2 RT-RIC: AT3S traffic routing/handover

First, we would like to recall the original plan for the RT RIC that aims to implement a hybrid of model-free and model-based Deep Reinforcement Learning (DRL) algorithms, which is already explained in D4.1 [1]. In this deliverable, we only focus on the model-based predictor as depicted in Figure 3-. By using the terminology from the RL algorithm, the model-based predictor receives a state at t from the environment and predicts an output of a function of position at t+1, which is denoted as $f(\mathbf{p}_{t+1})$. By being able to predict what will happen at the next time step, then in the next Deliverable 4.3 we will focus on developing an RL agent that can utilize the prediction. For example, if the model-based predictor predicts that the RSSI of LiFi interface at the next time step is below a certain threshold, then the RL agent could anticipate accordingly, e.g., steer the traffic to other wireless access interfaces. In the following subsections, we will discuss our platforms to generate datasets, our methodology, our ML algorithm, and results.

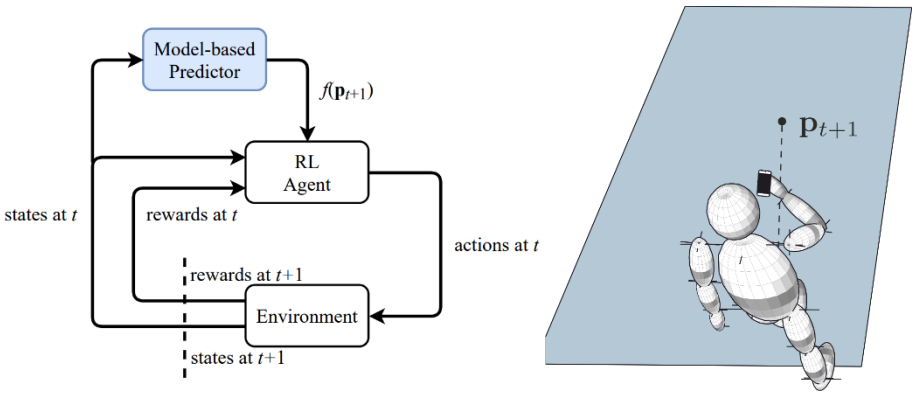


Figure 3-10. A model-based predictor that aims to predict an output of a function of position at *t*+1 given a state at *t*. Note that a similar figure is also shown in [12].

3.2.1 Dataset

The model-based predictor receives states from the environment as illustrated in Figure 3-10. States can be various things, such as RSSI or image snapshots from CCTVs. And, the output can be predictions, like the position of a user at the next time step, or a derived metric of it, e.g., RSSI at the next time step that can be estimated from the predicted future position. In the following discussions, we will first discuss our platforms to generate dataset, which is a collection of states from the environment.

3.2.1.1 owcsimpy

For this contribution, we develop a library called `owcsimpy`[1] whose current main usage is to calculate LiFi or optical wireless channel impulse response (CIR) for a given geometry description. An example of a geometry description is shown in Figure 3 where modelling of a small, indoor room consisted of a desk, a chair, a human, a LiFi-enabled UE, and a LiFi-enabled AP. The desk is modelled by a 3D plane; the chair is modelled by a cube, the human is modelled by a stack of cubes as can be seen also from Minecraft objects. The directions of the UE, the AP, and the human are represented by vectors as illustrated with arrows.

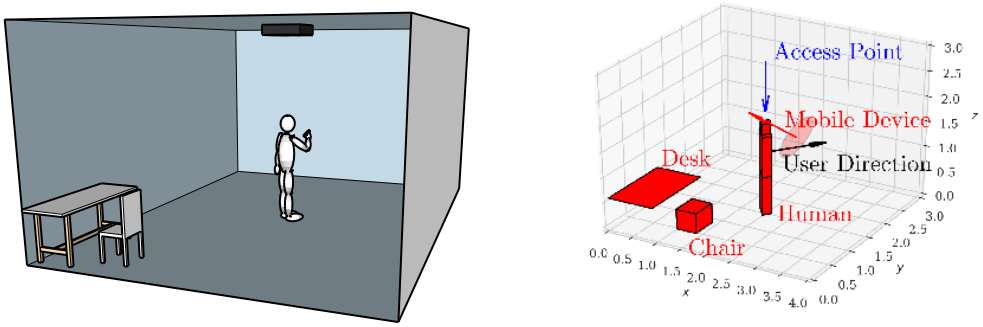
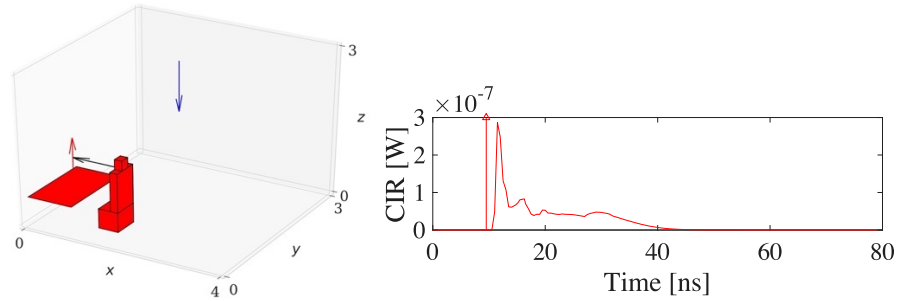
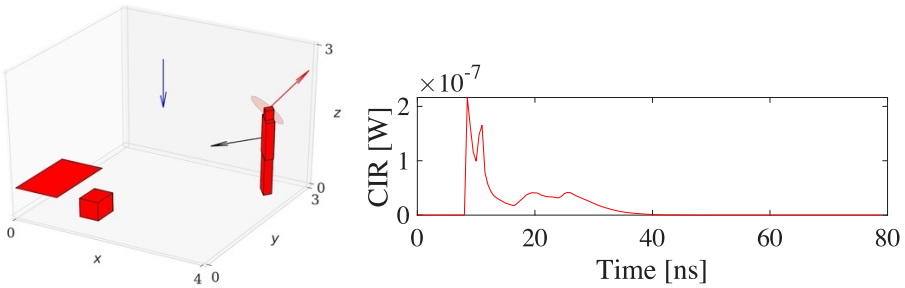


Figure 3-11.A geometry description from owcsimpy.



(a)



(b)

Figure 3-12. Simple use cases of owcsimpy and their corresponding CIRs.

[1] https://github.com/ardimasp/owcsimpy

By using `owcsimpy`, we can define a simple use case and obtain the optical wireless CIR for the defined use case. For example, Figure 3-(a) shows the optical wireless CIR for a scenario where a LoS link exists between the LiFi AP and a sitting user as depicted on the left side of the figure. Moreover, another simple scenario with NLOS link is also considered as depicted in left side of Figure 3-(b). The significant difference between the two wireless CIRs is the existence of a dirac impulse at the beginning of the curve. In addition, a lower received power shown in the bottom curve is mainly due to the fact the distance between the UE and the AP.

The main benefit of `owcsimpy` is that it is lightweight compared to other similar softwares such as Zemax®. The primary reason for this is that `owcsimpy` implements deterministic approaches in calculating CIRs, e.g., the iterative-method [34] or the frequency-domain approach [35], as opposed to the ray-tracing or stochastic methods. Therefore, we use `owcsimpy` to generate our dataset. The dataset that we generated is a collection of CIRs and the geometry descriptions. Specifically, our dataset consists of:

- 3D location and orientation of UE and all objects,
- location of objects (a human, a desk, and a chair),
- CIRs and the corresponding frequency responses of LiFi channels,
- pathloss of Wi-Fi channel.

It is worth noting here that the locations of the UE are generated based three different random mobility models, i.e., random waypoint (RWP) [36], random direction (RD) [37], and the truncated Levy-Walk (LW) model [38]. Moreover, the random orientation of the UE is modelled based on [39].

Figure 3.10 shows a sample of our dataset that is generated by using `owcsimpy`. Specifically, the left figure shows a snapshot of a geometry configuration of the user, UE, desk, chair, and a realization of the RWP model as shown by the black line. That is, the black line indicates the path that the user will follow. While moving, we also calculate, for example, the corresponding frequency responses of LiFi channel over time as shown in the right figure. In our dataset, we also collect the pathloss of Wi-Fi channel. The pathloss model (in dB) of Wi-Fi channel is taken from [40], which is defined as

$$P_{\mathrm{L}}(d, f_{\mathrm{c}}) = 40.5 + 20\ \log_{10}\frac{f_{\mathrm{c}}}{2.4} + 20\log_{10}\min(d, 5) + 1(d > 5)35\log_{10}\frac{d}{5} + 18.3\,F^{\frac{F+2}{F+1}-0.46} + 5W + \frac{5Hf_{\mathrm{c}}}{2.4}$$

where, d is the distance between the UE and the Wi-Fi AP, f_{c} is the center frequency of Wi-Fi signal in GHz, F is the number of floors traversed, and H is the number of humans traversed.

$$P_{\mathrm{L}}(d, f_{\mathrm{c}}) = 40.5 + 20\ \log_{10}\frac{f_{\mathrm{c}}}{2.4} + 20\log_{10}\min(d, 5) + 1(d > 5)35\log_{10}\frac{d}{5} + 18.3\,F^{\frac{F+2}{F+1}-0.46} + 5W + \frac{5Hf_{\mathrm{c}}}{2.4}$$

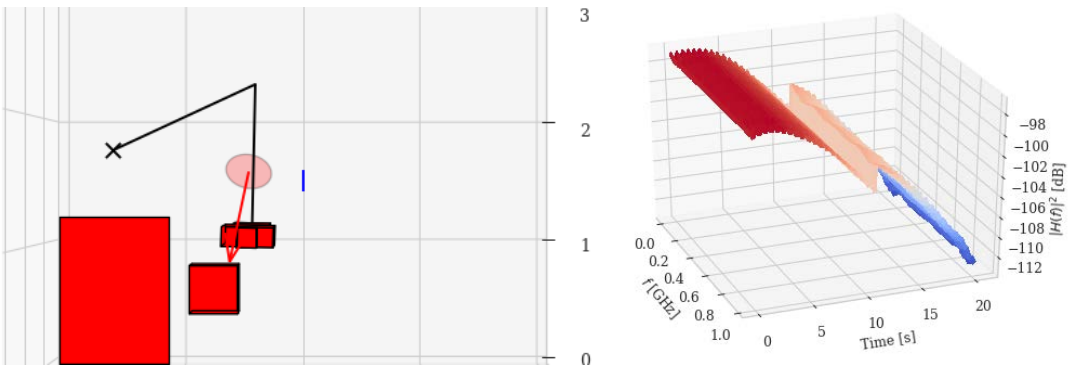


Figure 3.10. A top view of a geometry description, where the black line denotes the path that the user will take based on the RWP model (left). The corresponding frequency responses of the UE that travels following the black line in the left figure (right).

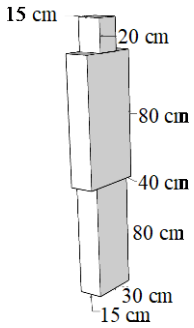


Figure 3.11. The dimensions of a human model.

Geometry details of the objects that are used to generate our dataset are as follows. The dimension of the room is 4m x 3m x 3m. The locations of both LiFi and Wi-Fi AP are at the center of the ceiling. The dimensions of the chair and the table are 0.4m x 0.4m x 0.4m and 1.2m x 0.9m, respectively. Figure 3.11 shows the dimensions of a human model.

Based on the descriptions above, our dataset is generated based on the flowchart below. Note that *N* is the cardinality of the dataset, where we set N as 12,000. The random mobility model is uniformly picked between RWP, RD, and LW with the minimum speed of 0.1 m/s, the maximum speed of 1 m/s, and the maximum waiting time of 0.5 s.

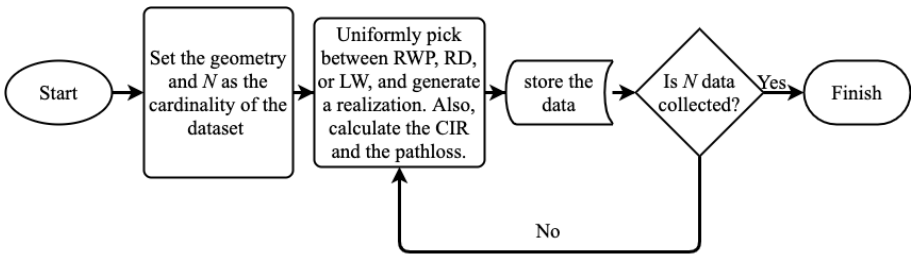


Figure 3.12. The flowchart for dataset generation.

3.2.1.2 CCTV Emulator

In the previous discussion, we mention that we collected both CIRs from LiFi channel and pathloss from Wi-Fi channel. This indicates that a mixed of information source is used, i.e., LiFi and Wi-Fi. In this deliverable, we are also interested to mix another source of information that most likely will be available in indoors, which is images from CCTV. It is obvious from Figure 3 to Figure 3.10 that the objects are oversimplified. That is, 3D objects are modelled by combining 2D planes and cubes. Not even, for example, lighting and shadowing are considered.

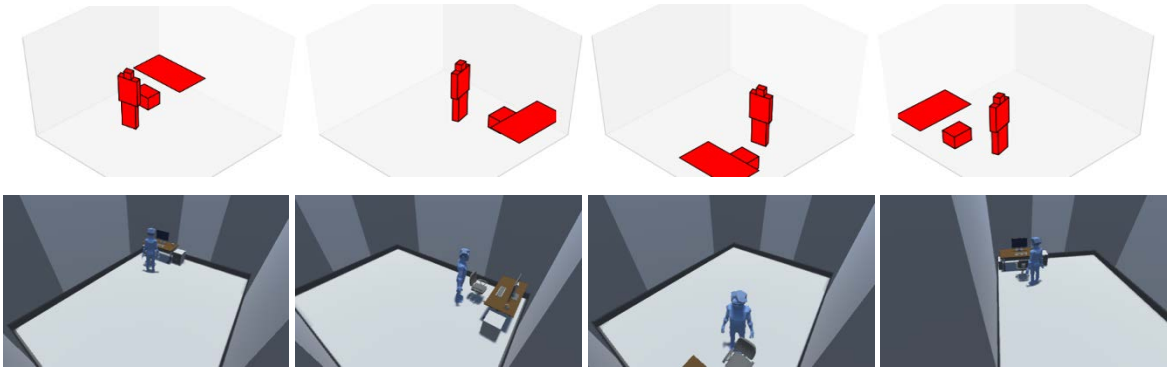


Figure 3.13. Comparisons between the objects rendered by `owcsimpy` and Unity 3D Game Engine.

Therefore, we need a sanity check mechanism to measure the reliability of modelling methodology of `owcsimpy`. In this deliverable, we intend to use a game engine that can support close-to-reality object renderings. Specifically, we use Unity 3D Game Engine[2] as a benchmark platform.

We assume that there are 4 CCTVs that are located at each top corner of the room. Point-to-point comparisons from each CCTV are depicted in Figure 3.13Figure 3.13. Similarly, we collect 12,000 realizations generated from the random mobility models that are considered in the previous discussion.

3.2.2 Methodology

In this subsection, our objectives and methodologies are detailed, where Figure 3.14 summarizes them. The goal in this deliverable is to predict the time-series position at t+1 or a derived metric, i.e., $f(\mathbf{p}_{t+1})$ such as RSSI at t+1. However, before that, there are prior objectives that need to be investigated. As we will rely on `owcsimpy` to generate various forms of information, such as images, CIRs, or pathloss, especially for images, we need a mechanism to justify if the platform is sufficiently good to generate dataset of images. For other forms of data, i.e., CIR and pathloss, the reliability depends on the referenced model, i.e., [34] and [35] for LiFi CIR and [40] for Wi-Fi pathloss model. Particularly, image-based indoor localization will be used. Then, the next objective is to investigate if combining various information sources will make the performance better. Similarly, we will use indoor localization task results as our justification. Lastly, the time-series prediction evaluation of $\mathbf{p}_{t+1}$ and $f(\mathbf{p}_{t+1})$ will be conducted. The main reason to consider both of them is to prepare for thorough investigations when we combine the model-based predictor to the model-free RL model. Next, we will explain our methodologies individually.

3.2.2.1 Methodology for Objective-1

Figure 3.15 illustrates our methodology for the first objective. First, the datasets that contain images from cameras will be split into train/val set and test set. Then, pre-processing, such as image rescaling, is conducted before feeding images to CNN architectures. Next, the training phase that uses train/val set from `owcsimpy` and the Unity is performed individually. As we want to predict the position of UE, we will use Mean Square Error (MSE) as the loss function. The MSEs for training and validation will be used to stop the training early in order to avoid overfitting. There are two pretrained models that will be generated, i.e., CNN[3]-O that uses the dataset generated by `owcsimpy` and CNN-U that uses the dataset generated by the Unity engine.

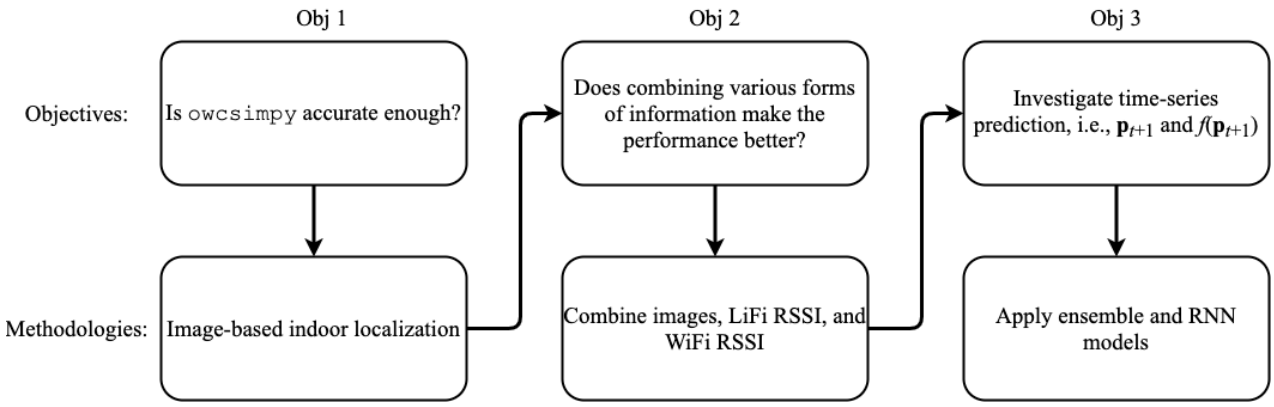


Figure 3.14. Objectives and methodologies.

[2] https://unity.com/
[3] Here, CNN stands for Convolutional Neural Network

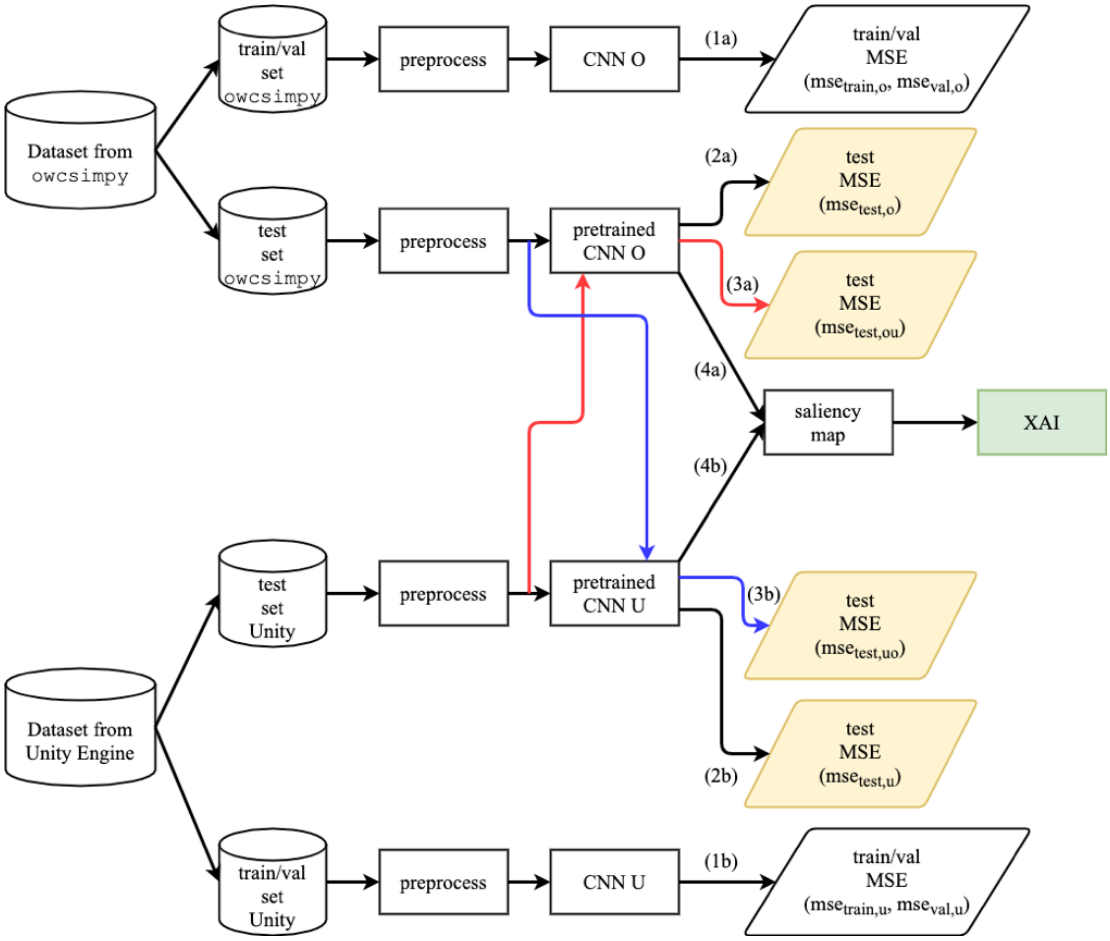


Figure 3.15. Methodology for Objective 1.

After obtaining the pretrained CNN models, there are four MSE metrics that will be measured to test generalization capability of the models, i.e.:

- $mse_{test,o}$: MSE for the CNN-O that uses test set from `owcsimpy`,
- $mse_{test,ou}$: MSE for the CNN-U that uses test set from the Unity game engine,
- $mse_{test,u}$: MSE for the CNN-U that uses test set from the Unity, and
- $mse_{test,uo}$: MSE for the CNN-U that uses test set from `owcsimpy`.

Moreover, the pretrained model will be equipped with a saliency map in order to have the Explainable AI (XAI) [44], which can tell us what the model looks at while making a prediction. The output of XAI is an image that is overlaid with a heatmap.

3.2.2.2 Methodology for Objective-2

The methodology for the second objective is quite straightforward. The main goal of this objective is to investigate the performance gain of the predictor when we combine all sources of information, i.e., images, LiFi RSSI, and Wi-Fi RSSI. We will combine them and measure the MSE. The obtained MSE is, then, compared to observe the performance gain with respect to the result from the first objective.

It is worth noting that Wi-Fi RSSI is calculated by using the assumption that the transmit power of the UE is 15 dBm per antenna and 20 dBm per antenna for the AP based on [40]. The RSSI of LiFi is calculated based on the DC channel gain of the CIRs and the total optical transmit power assumption of 10 W based on [42].

3.2.2.3 Methodology for Objective-3

There are two tasks that are targeted in this objective. First, we will predict the position of the user at the next time step, i.e., $\mathbf{p}_{t+1}$. Then, we also investigate the prediction of $f(\mathbf{p}_{t+1})$. Specifically, the aim is to predict the LiFi RSSI at the next time step.

3.2.3 ML Algorithm

In this subsection, we will discuss the deep learning architectures that are used in each objective.

3.2.3.1 Deep Learning Architecture for Objective-1

Two different CNN architectures are used for this objective, namely VGG16 and MobileNet-v2. These two architectures are chosen to compare their performances and complexities. That is, VGG16 has more computational complexities and has a better performance on the ImageNet dataset. A pretrained model from each architecture that is trained to the ImageNet dataset is used. It is worth noting that the size of input image of VGG16 should be rescaled to 240x240x3 px, and the size of input image of MobileNet-v2 should be rescaled to 160x160x3 px. As we have 4 input images for each position, we ensemble the CNN architecture and calculate the average as depicted in Figure 3.16 and defined in following equation

$$\hat{\mathbf{p}}_t = 0.25 \sum_{i=1}^{4} f_{\mathbf{W}_i}(\mathbf{X}_i),$$

where $\mathbf{W}$ is the weights of the CNN model ($f_{\mathbf{W}i}$ denotes a parameterized CN) and $\mathbf{X}_i$ is the image that is captured from the i^{th} camera as depicted in Table 3-16.

3.2.3.2 Deep Learning Architecture for Objective 2

Figure 3.17 shows the deep learning architecture for objective 2. The pretrained feature learning layers are obtained from the first objective. The outputs of the feature learning are fed into the fully-connected layer, which consists of 1 input layer having 16386 neurons[4], one hidden layer having 4096 neurons, and an output layer. As in the first objective, MSE is used as the loss function.

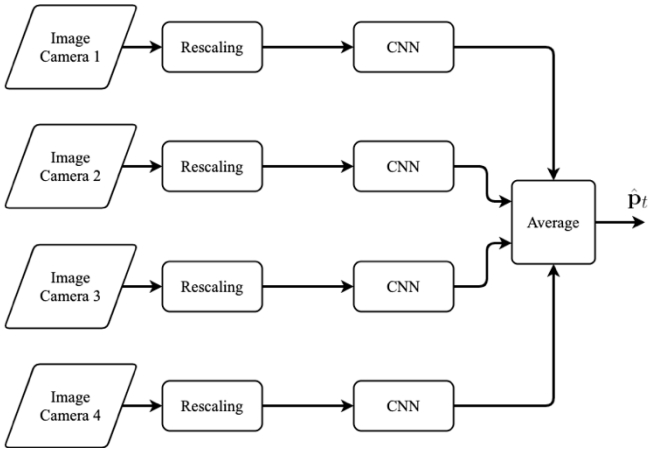


Figure 3.16. Ensemble architecture for Objective 1.

[4] The last layer of the feature learning of VGG16 gives 4096 channels. Since we have 4 feature learnings and two additional RSSI information, then the input layer of the fully-connected has 16386 neurons.

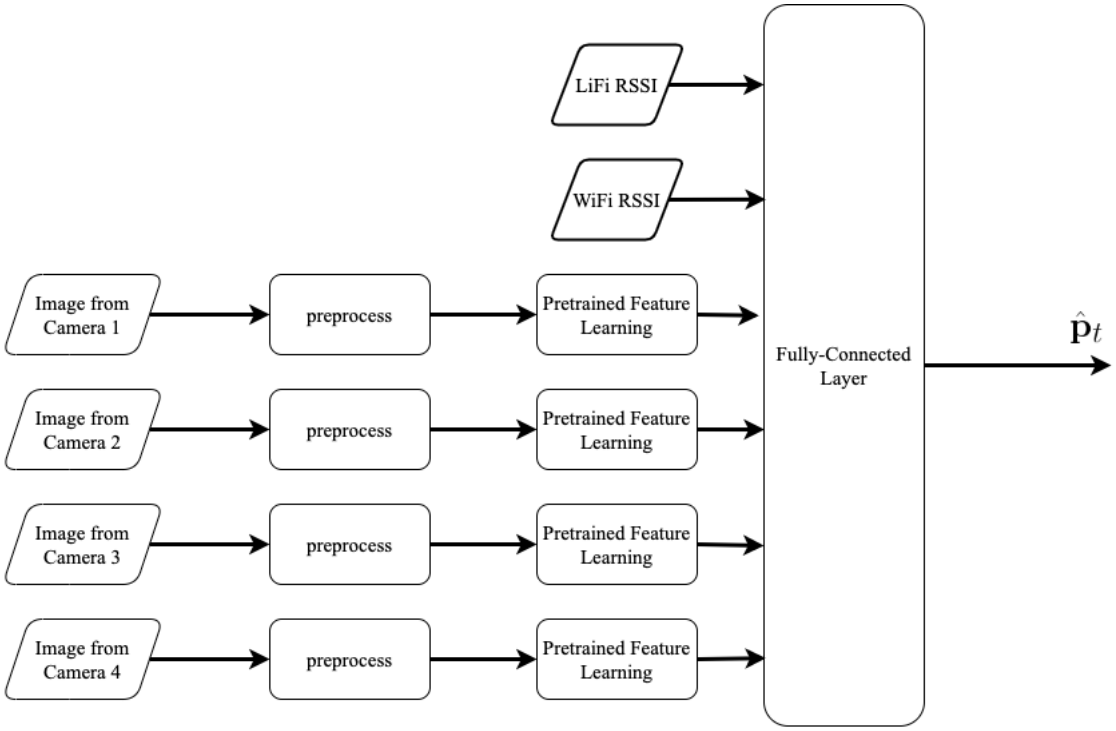


Figure 3.17. Deep learning architecture for Objective 2.

3.2.3.3 Deep Learning Architecture for Objective 3

For the third objective, we implement a class of recurrent neural network (RNN), namely long- short-term memory (LSTM). Figure 3.18 shows the integration of CNN-based model and LSTM. The CNN-based model is the pretrained model that is obtained from the second objective. The model receives a combination of images and RSSIs at different time instances. The variable *T* shows the memory length of our model. The output states of all LSTM cells are then fed to a fully connected layer that has a hidden layer. In this deliverable, we implement *T*=8 and the number of neurons in the hidden layer is 4.

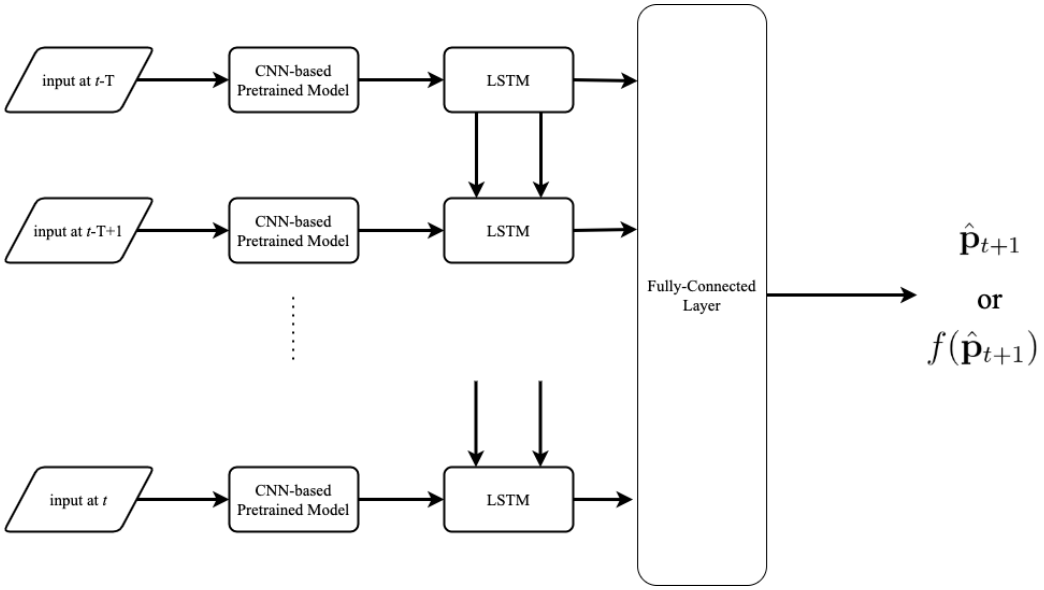


Figure 3.18. Deep learning architecture for Objective-3.

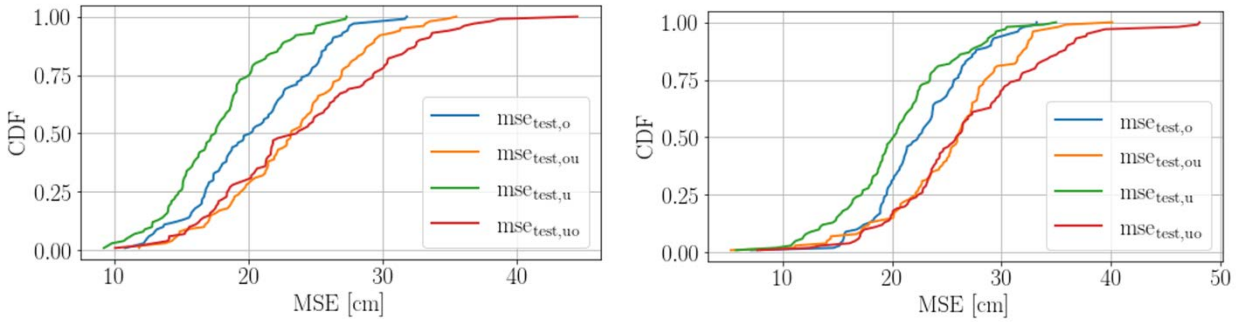
3.2.4 Results and Discussions

Our results and discussions are explained in this subsection. It is worth noting that the train/val and test sets are split with the ratio of 70% and 30%. During the training process, the Adam optimizer [43] is used with the learning rate of 0.001.

3.2.4.1 Objective 1

First, we measure the MSEs according to Figure 3.15. Figure 3.19 shows the obtained MSE results.

It is obvious from the figure that for each metric, VGG16 gives a better accuracy than that of MobileNet-v2. The use of dataset also affects the accuracy. That is, the use of Unity dataset gives a better accuracy compared to that of `owcsimpy`. However, the generalization of CNN with the `owcsimpy` dataset is better to that of CNN with the Unity dataset. This fact can be seen by comparing $mse_{test,ou}$ and $mse_{test,uo}$. Values of the MSEs are detailed in Table 3-3.



(a) VGG16 (b) MobileNet-v2

Figure 3.19. CDF of MSE of each architecture for Objective 1.

Table 3-3. MSE of VGG16 and MobileNet-v2 for Objective 1

Metrics	Criterion	VGG16 (in cm)	MobileNet-v2 (in cm)
$mse_{test,o}$	5% CI	12.41	15.49
	mean	20.35	22.38
	Std. dev.	4.63	4.93
	95% CI	27.42	30.72
$mse_{test,ou}$	5% CI	14.45	14.06
	mean	22.69	27.43
	Std. dev.	5.16	6.84
	95% CI	31.31	32.82
$mse_{test,u}$	5% CI	10.90	11.19
	mean	18.69	20.25
	Std. dev.	4.35	5.16
	95% CI	25.00	29.43
$mse_{test,uo}$	5% CI	14.07	16.62
	mean	25.19	27.59

	Std. dev.	6.53	6.92
	95% CI	35.75	37.79

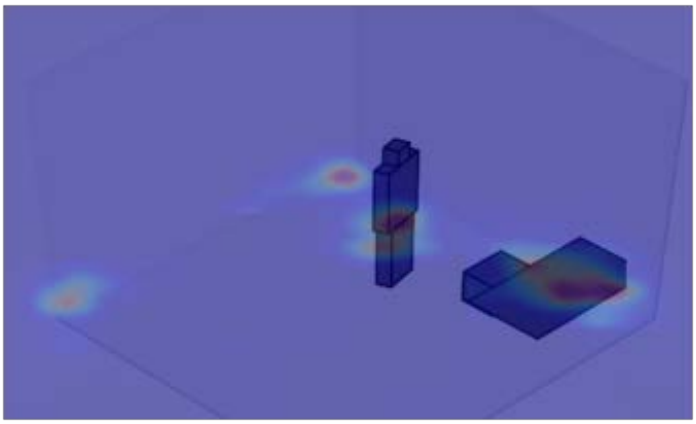


Figure 3.20. GRAD-CAM output of VGG16.

In this subsection, we also add an XAI in order to ensure that our ML models are looking at the correct objects when they make a prediction by means of a saliency map method. Specifically, we implement the GRAD-CAM [41], and an output sample of GRAD-CAM to one of test images is shown in Figure 3.20. The output of the GRAD-CAM is a heatmap image, where the heatmap shows the features that are significant. Our interpretation to the GRAD-CAM result is that our model uses static objects as references to estimate the position of the dynamic object, i.e., the user. Static objects include corners of the room, the desk, and the chair.

Based on the fact that the performance difference of the ML model with the dataset that is generated by using `owcsimpy` and Unity game engine is minor, we will use `owcsimpy` for the second and third objectives.

3.2.4.2 Objective 2

Figure 3.21 shows the MSE comparisons that are obtained for the second objective. The statistics are detailed in Table 3-4. 1 cm to 2 cm gain is achieved by adding the RSSI information.

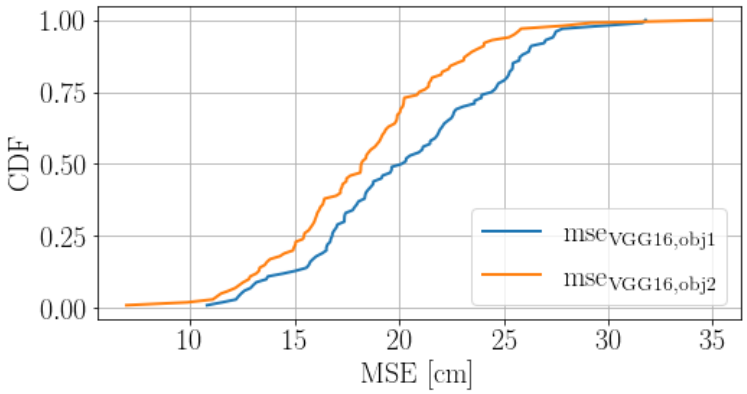


Figure 3.21. MSE results for Objective 2.

Table 3-4. Statistics comparisons of MSE results for Objectives 1 and 2

Objectives	5% CI	Mean	Std. Dev.	95% CI

1	12.41	20.35	4.63	27.42
2	11.44	18.35	4.56	25.50

3.2.4.3 Objective 3

Figure 3.22 depicts one of prediction results of our ML model as shown by comparing the red line and the black line. The CDF is shown on the right figure of Figure 3.22, and the statistics are listed on Table 3-5. It is obvious that by considering the past-history of the estimated position, the accuracy of the prediction increases.

Here, we also measure the LiFi RSSI based on the predicted position (which is denoted by $f(\hat{\mathbf{p}}_{t+1})$) and comparing it with an end-to-end training for the LiFi RSSI (which is denoted by $\hat{f}(\mathbf{p}_{t+1})$).

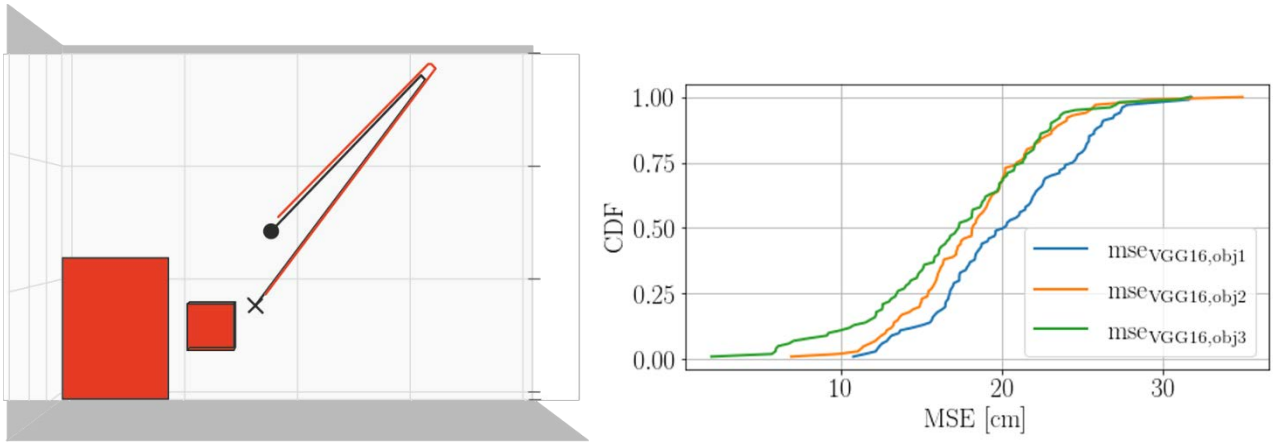


Figure 3.22. Prediction results as shown in the red line, and the black line shows the test data (left). The CDF of MSEs for the third objective (right).

Table 3-5. Statistics comparison of MSE results for all objectives.

Objectives	5% CI	Mean	Std. dev.	95% CI
1	12.41	20.35	4.63	27.42
2	11.44	18.35	4.56	25.50
3	6.11	16.66	5.23	24.58

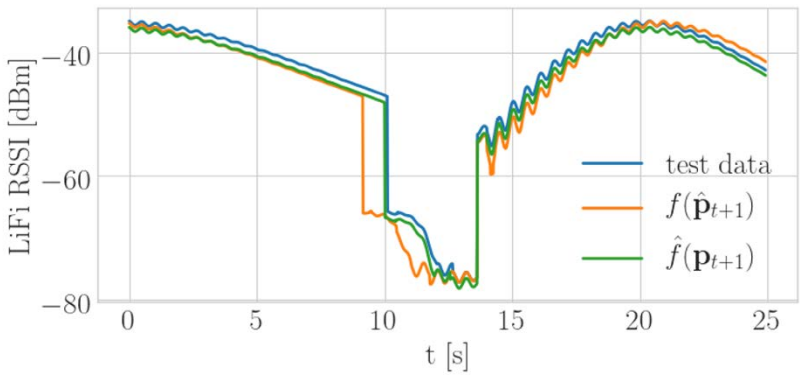


Figure 3.23. Comparing LiFi RSSI based on the predicted position $f(\hat{\mathbf{p}}_{t+1})$ and the end-to-end LiFi RSSI learning

$\hat{f}(\mathbf{p}_{t+1})$.

One of the results are show in Figure 3.23. The MSE of $f(\hat{\mathbf{p}}_{t+1})$ in the figure below is 4.39 dBm, while the MSE for the other one is 1.69 dBm. Therefore, whenever it is possible, it is better to perform an end-to-end learning. However, predicting the next position might be useful for our RL agent later.

3.3 RAN slicing in multi-tenant networks

3.3.1 Initial implementation

This section considers a private venue network owner of a NG-RAN infrastructure composed of a number of cells with diverse deployment characteristics (i.e., access technology, cell radius, transmission power, frequency of operation). Each cell has a different amount of physical resources that provide a certain cell capacity. These physical resources and their amount depend on the specific access technology used by the cell, e.g., Physical Resource Blocks (PRBs) for the case of 5G NR or LTE, airtime in case of Wi-Fi, etc. Following the WAT as a Service (WATaaS) service delivery model explained in deliverable D2.2 [2], the network is shared among different tenants, each of them provided with a RAN Slice Instance (RSI). Then, the considered problem consists in determining how the available capacity should be distributed among the different RAN slices in the different cells while fulfilling the SLA requirements of each tenant and at the same time achieving an efficient utilisation of the available resources.

Section 4.4.3 of deliverable D4.1 [1] presented the initial design of a Multi Agent Reinforcement Learning (MARL) algorithmic solution based on Deep Q-Network (DQN) to deal with this capacity sharing problem. Starting from this design, the functional model and components of the solution in the context of the 5G-CLARITY architecture are illustrated in Figure 3.24. The MARL solution resides at the AI engine and consists of two main components, namely the ML inference host and the ML training host. These components receive inputs and/or provide outputs from/to the other elements of the 5G-CLARITY architecture through the intent engine as it was explained in Section 6.3.4 of 5G-CLARITY D4.1 [1].

The solution is designed to operate with *N* cells and *K* RAN slices and to keep track of the traffic variations of the different tenants in periods (time steps) of Δt minutes. This is achieved through the dynamic configuration of the resource quota of each cell on a per RAN slice basis, assuming the 5G-CLARITY wireless dedicated quota model explained in Section 2.1. The value of the resource quota for the *k*-th slice in the *n*-th cell at time step *t* is denoted as $\alpha_t(k,n)$. As shown in Figure 3.24 the resource quota is determined by the ML inference host and is provided through the intent engine to the slice manager that will configure it in the RAN nodes. This configuration will depend on the specific technology of each cell. For example, in the case of 5G NR cells, the resource quota is mapped to the *rRMPolicyDedicatedRatio* attribute defined in the 3GPP 5G Network Resource Model [45].

The SLA specification is done in terms of two parameters that constitute the RAN NSI service profile defined by the private network operator and reflect the requirements to be fulfilled:

- Scenario Aggregated Guaranteed Bit Rate *SAGBR*(*k*): This specifies the aggregate bit rate to be provided to tenant *k*, if it has enough demand, across all the cells in the network during a time period of Δt minutes. It is worth noting that this parameter would correspond to the *dlThptPerSlice* attribute included in the `ServiceProfile` <<datatype>> of [45] and directly inherited from the GSMA Generic network Slice Template (GST) [45], which provides a standardized list of attributes (e.g. performance related, function related, etc.) that can be used to characterize different types of network slices.
- Maximum Cell Bit Rate *MCBR*(*k,n*): This specifies the maximum bit rate that can be provided to tenant *k* in cell *n*. This limit is defined in order to avoid that all the capacity of a cell is assigned to a single tenant under highly extreme heterogeneous spatial load distributions with tenants demanding

excessive capacity in certain cells. Then, it is assumed that, when the demand of the tenant in one cell exceeds *MCBR*(*k,n*), the SLA only requires to provide *MCBR*(*k,n*) in this cell, even if this could mean that the aggregate *SAGBR*(*k*) is not provided across all cells. The *MCBR*(*k,n*) parameter would be related with the *dlThptPerUe* and the *termDensity* attributes defined in [45] that are inherited from the GSMA GST. They specify, respectively, the average data rate delivered by the network slice per UE and the maximum user density over the coverage area of the network slice. Then, *MCBR*(*k,n*) is the product of *dlThptPerUe* for slice *k*, *termDensity* for slice *k* and the service area of cell *n*.

The ML inference host is in charge of determining the resource quota for each slice and cell using the learnt model provided by the ML training host. For this purpose, the ML inference host includes *K* per-slice action selection policies whose outputs are injected to the resource quota computation function.

The action selection policy $\pi(k)$ of slice *k* gets the network state $\boldsymbol{s}_t(k)$ observed for this slice at the time *t* when the policy is executed and determines the action $\mathbf{a}_t(k)$ to be applied for this slice. The action $\mathbf{a}_t(k)$ is composed of *N* per-cell actions that take one out of three possible values corresponding to: increase the resource quota $\alpha_t(k,n)$ for slice *k* in cell *n* in an amount of Δ for the next time step, maintain the same resource quota or decrease it in an amount of Δ.

In turn, the state $\boldsymbol{s}_t(k)$ includes *N* different per-cell components, each one given by the triple $<\rho_t(k,n), \alpha_t(k,n), \alpha_{ava,t}(n)>$ where $\rho_t(k,n)$ is the fraction of physical resources occupied by the slice *k* in cell *n* at time *t* and $\alpha_{ava,t}(n)$ is the total amount of resource quota in the cell not allocated to any slice. While the values of $\alpha_t(k,n)$ and $\alpha_{ava,t}(n)$ are directly available at the ML inference host, the value of $\rho_t(k,n)$ is obtained from the Performance Measurements (PM) collected from the telemetry system at the different cells.

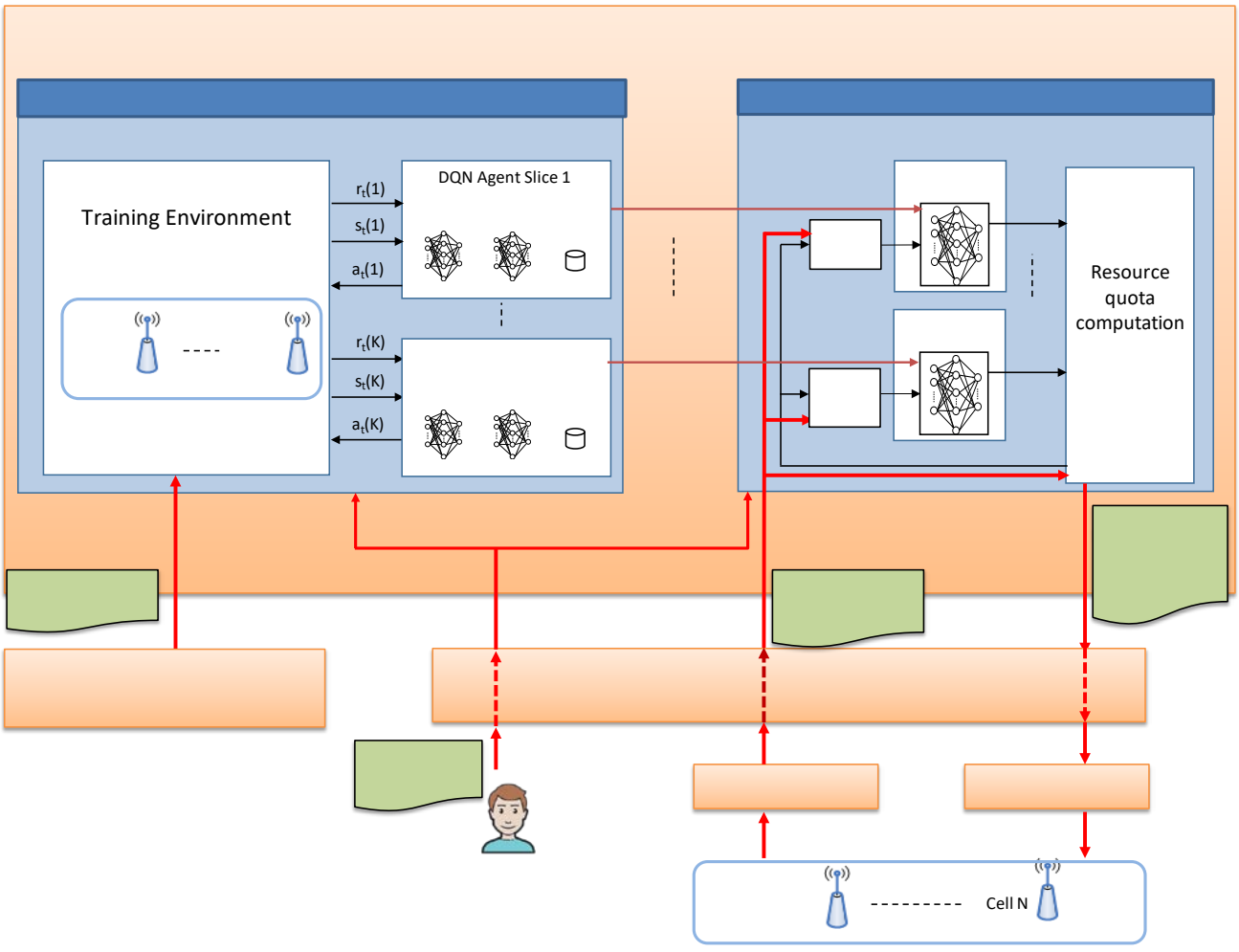


Figure 3.24. Functional model of the Deep Q Network-based RAN slicing solution

For example, for the case of 5G NR and based on [47], this metric would be obtained as the average over the interval (t-Δt,t) of the ratio between the "DL PRB used for data traffic", which measures the number of PRBs used in average for data traffic in a given slice and cell, and the "DL total available PRB", which measures the number of available PRBs in the cell.

Following the DQN approach, the action selection policy $\pi(k)$ of the k-th slice for a given state $\boldsymbol{s}_t(k)$ is defined as $\text{argmax}_{\mathbf{a}_t(k)} Q_k(\boldsymbol{s}_t(k), \mathbf{a}_t(k), \theta_k)$ where $Q_k(\boldsymbol{s}_t(k), \mathbf{a}_t(k), \theta_k)$ is the output of a DNN for the input state $\boldsymbol{s}_t(k)$ and the output action $\mathbf{a}_t(k)$. The internal structure of the DNN is specified by the vector of parameters θ_k that contains the weights of the different neuron connections. The optimum values of θ_k that determine the policies to be followed by the different slices in order to maximize the cumulative reward are learnt offline by the ML training host who provides them to the ML inference host.

The resource quota computation function determines the value of the resource quota $\alpha_t(k,n)$ to be assigned to each slice and cell for the next time step by applying the increase/maintain/decrease actions provided by the action selection policies of all the slices. When applying the actions, this function ensures that the $MCBR(k,n)$ values are not exceeded. Moreover, since the action selection policies for the different slices operate independently, this function also checks that the aggregated resource quota for all the slices in a cell after applying the actions does not exceed 1 in order not to exceed the cell capacity. If this happens, it applies first the actions of the slices involving a reduction or maintenance of the resource quota and the remaining capacity is distributed among the slices that have increase actions. This distribution is proportional to their $SAGBR(k)$ values, as long as their current throughput is not already higher than $SAGBR(k)$. For doing this adjustment, the measured throughput per slice across all the cells in the last time step is needed, which is assumed to be obtained from the telemetry system.

The ML training host is in charge of learning the DNN parameters θ_k that determine the per-slice action selection policies. This is done through a multi-agent DQN approach in which each DQN agent learns the optimum policy of a different RAN slice by continuously interacting with a training environment and updating the DNN parameters as a result of these interactions. The training environment considered here is a network simulator that mimics the behavior of the real network when varying the offered load of the different slices in the different cells and when modifying the resource quota allocated to each slice as a result of the actions made by the DQN agents. In this respect, the simulator is fed by training data consisting of multiple time patterns of the required capacity (i.e., offered load) of the slices in the different cells. This data can be either built synthetically or extracted from real network measurements through the 5G-CLARITY data management framework.

For carrying out the training process, each DQN agent is composed by three different elements: *(i)* The evaluation DNN, which corresponds to the function $Q_k(\boldsymbol{s}_t(k),\mathbf{a}_t(k),\theta_k)$ being learnt that will eventually determine the policy to be applied at the ML inference host; *(ii)* The target DNN, which is another neural network with the same structure as the evaluation DNN but with weights $\bar{\theta}_k$. It is used for obtaining the so-called Time Difference (TD) target required for updating the evaluation DNN; *(iii)* The experience data set (ED), which stores the experiences of the agent resulting from the interactions with the training environment.

The interactions between the DQN agent and the training environment occur in time steps of (simulated time) duration Δt. In each time step t, the DQN agent of the k-th slice observes the state $\boldsymbol{s}_t(k)$ in the training environment and selects an action $\mathbf{a}_t(k)$. Action selection is based on an ε-Greedy policy that, with probability 1-ε, chooses the action that maximizes the output of the evaluation DNN, and, with probability ε, chooses a random action. As a result of applying the selected action, the training environment generates a reward value $r_t(k)$ that assesses how good the action was from the perspective of the desired behavior. The reward function includes three components with corresponding weights φ_1, φ_2, φ_3 that capture, respectively, the SLA satisfaction ratio of the slice k, the aggregated SLA satisfaction ratio for the rest of slices $k' \neq k$ and the

capacity utilization. The SLA satisfaction ratio is a metric that measures the obtained throughput by the slice in relation to the SLA requirement, and the capacity utilization measures the throughput of the slice in relation to the capacity corresponding to the assigned resource quota. Detailed formulation of these terms was given in Section 4.4.3 of 5G-CLARITY D4.1 [1].

As a result of the interactions between the training environment and the DQN agent, each experience of the ED is represented by a tuple that includes: *i)* the state observed at the beginning of a given time step; *ii)* the selected action; *iii)* the obtained reward as a result of this action; and *iv)* the new state observed at the end of the time step duration. The experiences stored in the ED are used by the DQN agent to progressively update the values of the weights of the evaluation and target DNNs. The reader is referred to section 4.4.3 of deliverable D4.1 for details on the mathematical formulation of this process. The training process stops after a sufficient number of time steps that ensures the convergence of the process. At this point, the ML training host is ready to provide the evaluation DNN parameters θ_k so that the model can be applied on the real network using the ML inference host. Notice that it could also be possible to continue with the training process in parallel to further update the ML model. In this case the DQN agents would obtain the states and rewards from the real network instead of from the training environment.

3.3.2 Evaluation methodology

The proposed approach is evaluated by means of system-level simulations. The simulation model has been developed in Python by using the library *TF-Agents [48]*, which provides tools for the development of deep reinforcement learning models, including DQN. The simulation environment considers a network with a number of cells and takes as inputs the offered load patterns of the different tenants in each of the cells. The simulator operates in two stages as explained in the following:

- Training stage: This corresponds to the operation of the ML training host. Training has been conducted by using a dataset composed of multiple synthetically generated offered load patterns of the tenants in the different cells. Each pattern includes the traffic in b/s required by the tenant during one day, measured in periods of 15min. The different offered load patterns are injected one after the other in the simulator that operates in time steps of duration Δt. In every time step, the DQN agents select the actions that determine the resource quota assigned to each slice in each cell. Then, the number of physical resources (i.e., PRBs) that are utilized by the slice is the minimum between the assigned PRBs in accordance with the resource quota and the required PRBs, which are determined by the offered load and the spectral efficiency. Then, the throughput achieved by each slice is obtained using the number of utilized PRBs and the spectral efficiency. This is used to compute the reward that, together with the selected action and the actual and previous states, is stored in the ED and is used to update the weights of the evaluation and target DNNs. This process is repeated until reaching the maximum number of training time steps. At the end, the resulting weights of the evaluation DNN determine the trained policy to be used by the ML inference host.
- Evaluation stage: Once the training stage has been completed, the ML inference host assesses the obtained policy using the same system level network simulator of the training, considering the actual offered load patterns of the different tenants in each cell for one day duration. This pattern corresponds to one of the patterns of the training dataset adding on top of it a random fluctuation of 5%. The trained policy is executed every time step to obtain the resource quota values that are applied in the difference cells. Based on this, the relevant Key Performance Indicators (KPIs) are determined.

The main KPIs to assess the performance of the model are the following ones:

- Assigned capacity to tenant *k* at time step *t* ($A_t(k)$): It is measured in b/s and is obtained from the resource quota $\alpha_t(k,n)$ and the capacity of each cell c_n as:

$$A_t(k) = \sum_{n=1}^{N} c_n \cdot \alpha_t(k,n)$$

- Reward of tenant k (R_k): It is computed as the average of the reward $r_t(k)$ obtained by the tenant over a duration of G time steps (i.e a total time of $G\cdot,\Delta t$ minutes).

$$R_k = \frac{1}{G}\sum_{t=0}^{G-1} r_t(k)$$

- SLA satisfaction of tenant k (SS_k): It measures the ratio between the throughput $T_t(k)$ provided to tenant k and the minimum between the aggregated offered load of the tenant $O_t(k)$ and its *SAGBR(k)* value. It is measured in each time step and it can be averaged over a duration of G time steps, that is:

$$SS_k = \frac{1}{G}\sum_{t=0}^{G-1} \min\left(\frac{T_t(k)}{\min(O_t(k), SAGBR(k))}, 1\right)$$

  It ranges $0 \leq SS_k \leq 1$, taking $SS_k=0$ value when the SLA is not satisfied and $SS_k=1$ when it is fully satisfied. Note that the definition of SS_k considers that when $O_t(k)$ is lower than the *SAGBR(k)*, $O_t(k)$ needs to be provided, whereas in the case that $O_t(k)$ is greater than *SAGBR(k)*, at least *SAGBR(k)* needs to be provided.

- System utilization (U): It is computed as the average for all cells of the ratio between the number of physical resources used by all the tenants in a cell during a time step and the total number of physical resources in the cell. It is measured in each time step and can be averaged over a duration of G time steps, that is:

$$U = \frac{1}{G}\sum_{t=0}^{G-1}\frac{1}{N}\sum_{n=1}^{N}\frac{1}{N_T(n)}\sum_{k=1}^{K} u_t(k,n)$$

  where $u_t(k,n)$ is the number of physical resources used by tenant k in cell n averaged during time step t and $N_T(n)$ is the total number of physical resources available in cell n.

3.3.3 Evaluation results

The performance evaluation of the proposed approach is carried by means of two different case studies. Case study 1 consists in a first assessment of the algorithm in a simple scenario composed of a single cell. In turn, case study 2 presents a more detailed analysis of the algorithm in a multi-cell scenario including the sensitivity to the configuration parameters affecting the determination of the resource quota.

3.3.3.1 Case study 1

This case study assumes a single cell scenario that provides service to two tenants, denoted as Tenant 1 and Tenant 2. The parameters of the scenario are presented in Table 3-6. The training of the DQN model has been performed using the parameters of Table 3-7. This includes the hyperparameters of the DQN as well as other parameters of the model, such as the time step duration Δt, the action step Δ that specifies the increase/decrease in resource quota associated to an action or the weights of the reward function (see details in section 4.4.3 of D4.1 [1]). To obtain the values of these parameters, a prior analysis of the model behavior with different combinations of parameters has been conducted.

Table 3-6 Scenario Parameters for Case Study 1

Parameter	Value
Number of cells	1

Number of tenants		2
PRB Bandwidth		360 kHz
Number of available PRBs		51 PRBs
Average spectral efficiency		5 b/s/Hz
Cell capacity (c_n)		91.8 Mb/s
Total capacity (C)		91.8 Mb/s
SAGBR(k)	Tenant 1	55 Mb/s (60% of total capacity)
	Tenant 2	36 Mb/s (40% of total capacity)
MCBR(k)	Tenant 1	73 Mb/s (80% of total capacity)
	Tenant 2	

Table 3-7 Training Parameters for Case Study 1

Parameter	Value
Initial collect steps	2000
Maximum number of training time steps	160000
Experience Replay buffer maximum length	10^5
Mini-batch size	100
Discount factor	0.9
Learning rate	0.001
ε value (ε-Greedy)	0.1
Neural network architecture	1 layer x 100 nodes
Step time (*Δt)*	3 min
Action step (Δ)	0.3
Reward weights (j_1, j_2, j_3)	(0.3, 0.2, 0.5)

The evaluation of the trained model has been performed by considering two different offered load patterns, denoted as Situation A and Situation B. They are characterised by the temporal evolutions of Figure 3.25 and Figure 3.26, respectively, which plot the aggregated offered load $O_t(k)$ by the two tenants in the cell during a day. Situation A corresponds to an offered load pattern where Tenant 1 requires more capacity than Tenant 2 during the morning while the contrary case is given during the afternoon. This complementarity among tenants allows better illustrating the flexibility of the algorithm to adapt the resource allocations to each tenant. Situation B presents the contrary case, where the roles of Tenant 1 and Tenant 2 are reversed. Notice that, in Situation A, the aggregated load by both tenants does not exceed the total capacity (C_T=91.8Mb/s) at any time while in Situation B the total capacity is exceeded for a long period of time.

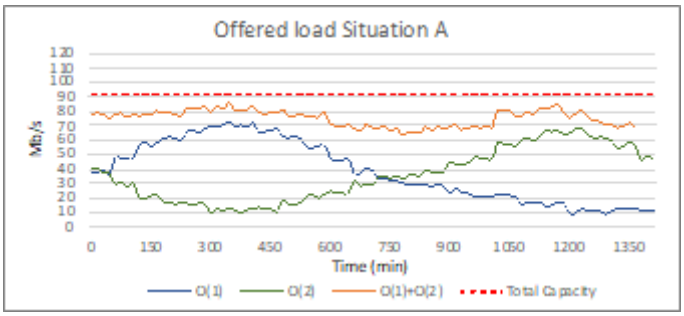


Figure 3.25. Offered load of Tenant 1 and Tenant 2 in situation A of case study 1

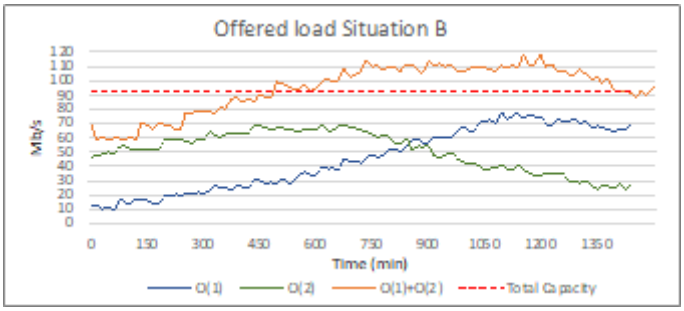


Figure 3.26. Offered load of Tenant 1 and Tenant 2 in situation B of case study 1

Figure 3.27 and Figure 3.28 compare the assigned capacity $A_t(k)$ with the offered load and the *SAGBR*(*k*) for *Tenant* 1 and *Tenant* 2 in *Situation A* and *Situation B*, respectively. In *Situation A*, the offered load of both tenants is generally served as there is enough capacity to fulfil the requirements for both of them. The capacity sharing mechanism provides the demanded capacity to both tenants, including those cases when the offered loads of *Tenant* 1 and *Tenant* 2 exceed *SAGBR*(1) and *SAGBR*(2), respectively, making efficient use of the available capacity and exhibiting the capability to exploit the complementarities in the traffic profiles between both tenants. In *Situation B,* a similar behaviour is observed but the assigned capacity is lower than the offered load during the periods where more capacity than available is requested. In those periods, the required capacity is given to the tenants whose offered load is lower than *SAGBR*(*k*), which shows that the capacity sharing function assures the *SAGBR*(*k*) established in the SLA.

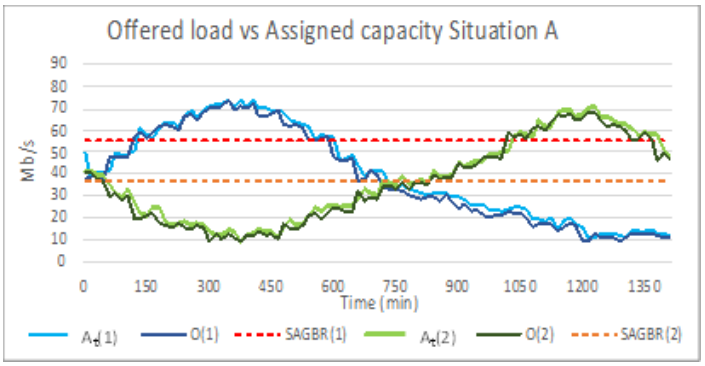


Figure 3.27. Offered load O(k) vs assigned capacity in Situation A of case study 1

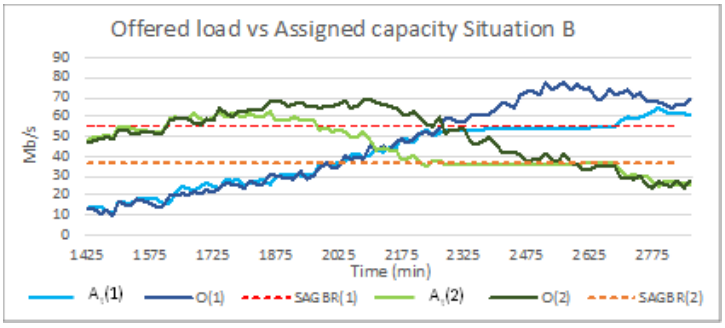


Figure 3.28. Offered load O(k) vs assigned capacity in Situation B of case study 1

For benchmarking purposes, the performance obtained in *Situation B* by the proposed approach has been compared against two reference capacity sharing solutions: *Reference* 1, which considers *SAGBR*(*k*) as the capacity assigned to tenant *k* in any case, and *Reference* 2, which considers that both tenants share the overall capacity independently on *SAGBR*(*k*). In *Reference* 2, whenever more capacity than available is requested, the capacity is distributed among tenants according to their offered load. Figure 3.29 and Figure

3.30 show the CDF of the SLA satisfaction ratio for Tenant 1 and Tenant 2, respectively. While Reference 1 always fulfils the SLA, Reference 2 presents much lower SLA satisfaction, given that both tenants are provided with lower throughput than required when the offered load exceeds the total capacity. The proposed MARL solution is able to improve the SLA satisfaction of Reference 2 by serving all the offered load as long as it is lower than SAGBR(k). Moreover, Figure 3.31 compares the CDF of the system utilisation in percentage of the three approaches. Reference 1 presents the lowest system utilisation, as no more than SAGBR(k) is provided even though there is enough capacity unused in the system to satisfy the offered load. However, the proposed MARL approach substantially improves the utilisation, with a performance very close to Reference 2. These results show how the presented approach allows satisfying the SLA and making efficient use of the resources, achieving a good trade-off between the two benchmarking schemes. This overall better behaviour of the proposed approach is also reflected in the obtained reward values for the different strategies. Specifically, the obtained average reward of the two tenants (i.e. the average of R1 and R2) is 0.972 for the MARL approach, while for References 1 and 2 it is, respectively, 0.938 and 0.953.

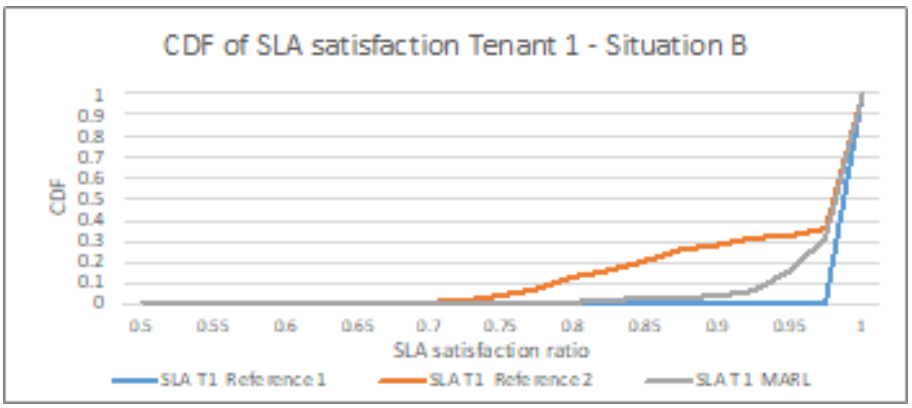


Figure 3.29. CDF of SLA satisfaction of Tenant 1 in Situation B

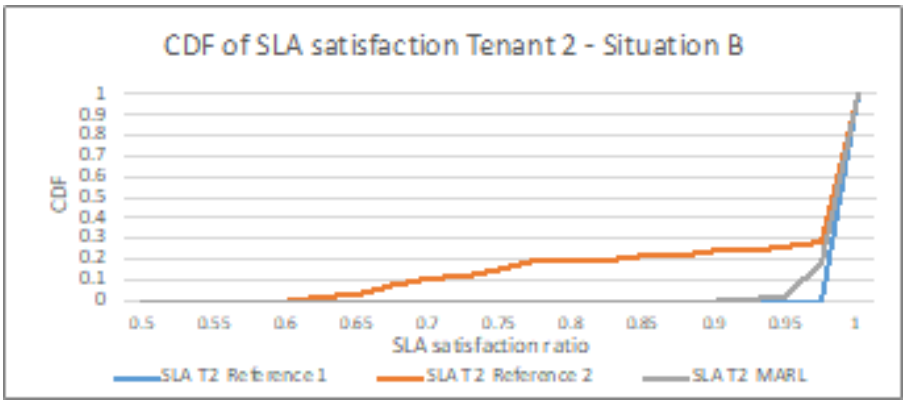


Figure 3.30. CDF of SLA satisfaction of Tenant 2 in Situation B

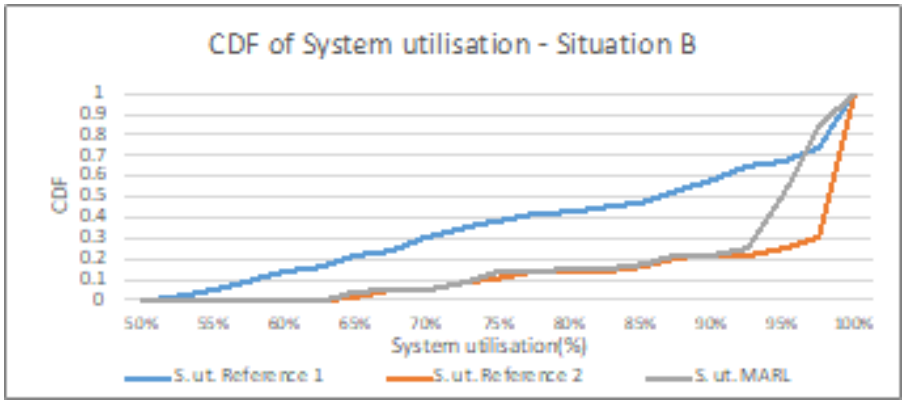


Figure 3.31. CDF of system utilisation in Situation B

3.3.3.2 Case study 2

This case study intends to analyse the sensitivity of the proposed MARL algorithm to the two operational parameters that determine how and when the resource quota allocated to each tenant is modified, namely the action step Δ and the time step Δt. This study is performed in a multi-cell scenario with two tenants. The scenario parameters are presented in Table 3-8. The training of the DQN model has been performed using the parameters of Table 3-9, which have been selected from a preliminary analysis of the model behavior with different combinations of these parameters. As observed, the study considers different configurations of the action step Δ that determines the increase/decrease in resource quota for each tenant, ranging between 0.001 and 0.09, and the time step values Δt that determine the times at which the resource quota is modified, ranging between 1 and 15 min.

Figure 3.32 presents the aggregate offered load of each tenant $O_t(1)$, $O_t(2)$ across all the cells of the scenario that is considered for the evaluation of the learnt model during one day. The figure also includes the aggregate load of both tenants (i.e. $O_t(1)+O_t(2)$), their *SAGBR*(k) values and the total system capacity. The considered loads of the two tenants have a complementary behaviour: $O_t(1)$ has higher values during the beginning of the day and at night whereas $O_t(2)$ reaches higher values at the middle of the day.

Table 3-8. Scenario Parameters for Case Study 2

Parameter		Value
Number of cells		5
Number of tenants		2
PRB Bandwidth		360 kHz
Number of available PRBs		65 PRBs
Average spectral efficiency		5 b/s/Hz
Cell capacity (c_n)		117 Mb/s
Total capacity (C)		585 Mb/s
SAGBR(k)	Tenant 1	351 Mb/s (60% ot total capacity)
	Tenant 2	234 Mb/s (60% ot total capacity)
MCBR(k)	Tenant 1	93.6 Mb/s (80% of cell capacity)
	Tenant 2	

Table 3-9. Training Parameters for Case Study 2

Parameter	Value
Initial collect steps	5000
Maximum number of training time steps	100000
Experience Replay buffer maximum length	10^7
Mini-batch size	256
Discount factor	0.9
Learning rate	0.0001
ε value (ε-Greedy)	0.1
Neural network architecture	2 layers x 100 nodes
Step time (Δt)	{1, 3, 5, 15} min
Action step (Δ)	{0.01, 0.03, 0.05, 0.07, 0.09}
Reward weights (j_1, j_2, j_3)	(0.3, 0.2, 0.5)

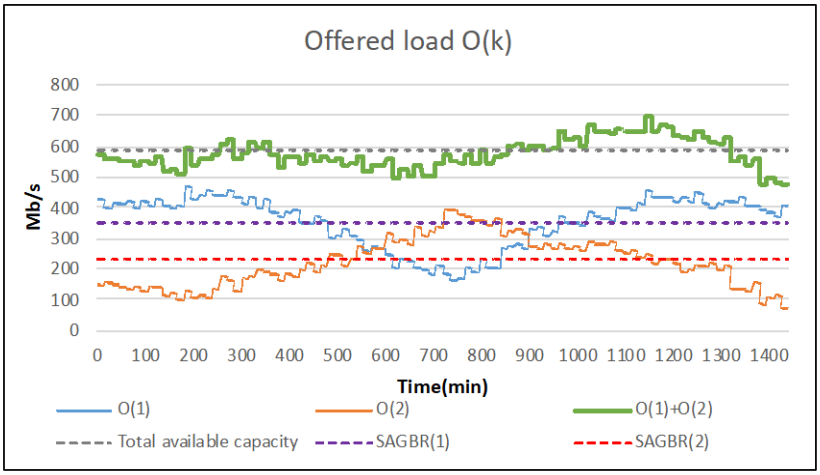


Figure 3.32. Offered loads of Tenant 1 and 2 during a day considered for the evaluation of case study 2

To capture different extreme cases, the pattern includes situations in which the offered loads of the tenants exceed their *SAGBR*(*k*) for some times during the day (e.g. between 0 and 500 min or after 1000 min for Tenant 1 or between 500 and 1100 min for Tenant 2) and situations in which the total offered load $O_t(1)+O_t(2)$ is higher than the available system capacity (e.g. during the time period from 900 min to 1300 min). A uniform distribution of the load among the different cells has been considered.

In order to analyse the impact of the values of time step duration Δt and action step Δ on the evolution of the training, the policies learnt for *Tenant* 1 and *Tenant* 2 (i.e. π(1) and π(2)) every 10000 time steps of the training stage have been evaluated using the offered loads of Figure 3.32. This allows capturing the evolution of the training process when increasing the number of training steps. Then, Figure 3.33 and Figure 3.34 show the evolution of the aggregate reward of the two tenants, i.e. R_1+R_2, averaged over the whole day, for Δ=0.01 and Δ=0.07, respectively, when considering all the values of Δt={1, 3, 5, 15} min. The selected values Δ=0.01 and Δ=0.07 are, respectively, representative of small and large action step values. Some similarities are obtained for the training evolutions for Δ=0.01 and Δ=0.07. For both cases, higher average reward is achieved for lower Δt values. The reason is that lower values of Δt provide a better adaptability to the offered loads since the policy is triggered more frequently, so the policy can easily react to changes. However, when comparing the results obtained for Δ=0.01 and Δ=0.07 given a certain Δt, different average rewards are obtained. In the case of Δt=1min higher average reward values are achieved for Δ=0.01, whereas for Δt=15min higher average rewards are obtained for Δ=0.07.

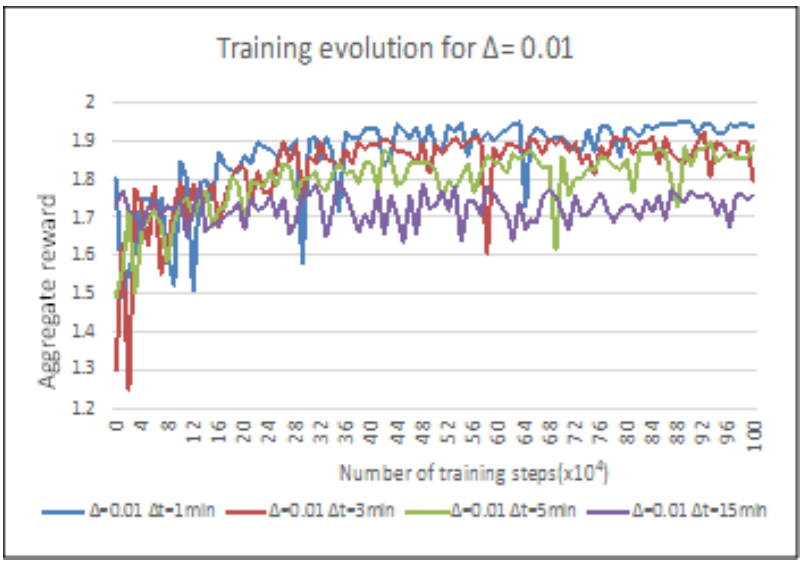


Figure 3.33. Average reward every 10000 steps during the training for Δ=0.01 and Δt={1,3,5,15} min

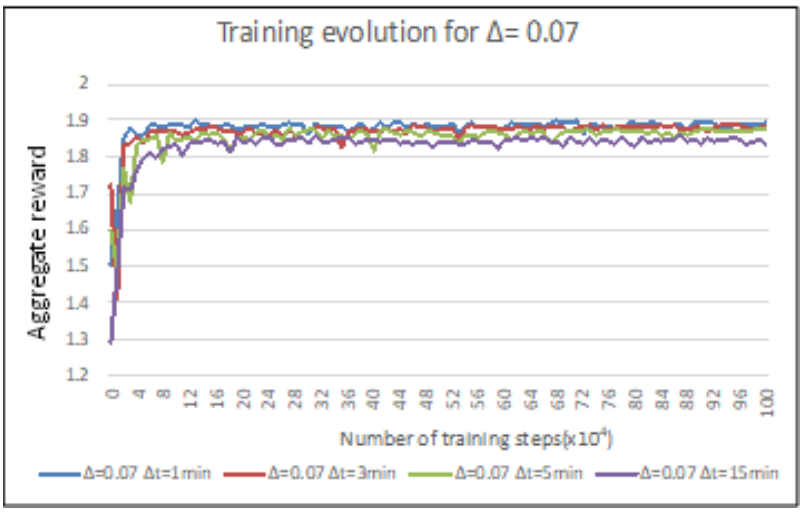


Figure 3.34. Average reward every 10000 steps during the training for Δ=0.07 and Δt={1,3,5,15} min

These results suggest that some combinations of Δ and Δt are more suitable to respond to the offered loads of Figure 3.33.

More differences are observed between the training evolution when using Δ=0.01 and Δ=0.07. For Δ=0.07, the average reward presents an initial period where it increases and presents fluctuations. This initial period has different durations depending on the value of Δt but, it does not take longer than $20 \cdot 10^4$ training steps in any of the studied cases. After this initial period, the value of average reward stabilises and the fluctuations decrease drastically, presenting a smooth average reward during training. Instead, when using Δ=0.01, the duration of the initial period until the average reward stabilises is much longer, and the fluctuations remain high, presenting some peaks. These differences between the training for Δ=0.01 and Δ=0.07 can be explained by the fact that in the case of Δ=0.01, the updates of the resource quota $\alpha_t(k,n)$ are performed in small steps, which make more difficult the process of learning, since the actions performed at each time step have a low impact on the next state and the obtained reward. Consequently, the agents need a larger number of time steps to learn how to behave in the different states and, in some cases, the learning does not stabilise (e.g. for Δ=0.01 and Δt=15min, where the average reward does not increase and the fluctuations remain high because only increasing Δ=0.01 every Δt=15min does not allow adapting to the offered loads and leads to poor training performance).

Table 3-10 depicts the aggregate reward of the two tenants corresponding to the average of the results obtained from the evaluations done between $50 \cdot 10^4$ and $100 \cdot 10^4$ training steps. It is observed that, in general, better reward is obtained for lower values of Δt. Then, in relation to the action step size, the highest average reward is obtained approximately for Δ=0.03 for all the time step durations. Results also show that, for values of Δ higher than 0.03, the average reward tends to decrease when increasing Δ. Looking jointly at the results of Figure 3.33, Figure 3.34 and Table 3-10 it is observed that a trade-off exists when selecting the value of Δ: a higher reward is generally achieved for low values of Δ but at the cost of higher fluctuations during the training process and longer training duration. Based on the obtained results, it is concluded that the selection of the Δ and Δt values has a clear impact on the training evolution of the policies and an adequate selection of these values is fundamental for ensuring an accurate learning process.

Table 3-10. Average Reward for Different Configurations

Action Step (Δ)	Aggregate Reward (R_1+R_2)			
	Δ*t*=1min	Δ*t*=3min	Δ*t*=5min	Δ*t*=15min
0.01	1.91	1.88	1.83	1.73

0.03	1.91	1.90	1.90	1.85
0.05	1.90	1.90	1.88	1.85
0.07	1.88	1.88	1.86	1.84
0.09	1.86	1.85	1.85	1.83

Table 3-11. SLA Satisfaction and System Utilization for Different Configurations

Action step (Δ)	SLA Satisfaction								System Utilization (*U*)			
	Tenant 1 (SS_1)				Tenant 2 (SS_2)							
	Δ*t*=1 min	Δ*t*=3 min	Δ*t*=5 min	Δ*t*=15 min	Δ*t*=1 min	Δ*t*=3 min	Δ*t*=5 min	Δ*t*=15 min	Δ*t*=1 min	Δ*t*=3 min	Δ*t*=5 min	Δ*t*=15 min
0.01	0.98	0.93	0.93	0.91	0.97	0.95	0.97	0.92	0.88	0.83	0.84	0.76
0.03	0.98	0.94	0.93	0.93	0.95	0.94	0.96	0.96	0.86	0.84	0.83	0.82
0.05	0.97	0.97	0.95	0.93	0.95	0.94	0.94	0.95	0.87	0.86	0.85	0.83
0.07	0.96	0.95	0.96	0.93	0.92	0.92	0.91	0.92	0.85	0.84	0.85	0.82
0.09	0.95	0.95	0.94	0.94	0.91	0.91	0.89	0.91	0.84	0.83	0.81	0.81

Table 3-11 presents the performance of the proposed approach in terms of SLA satisfaction of Tenant 1 and Tenant 2 (SS_1, SS_2) and system utilization (U) after evaluating the learnt models with all the considered combinations of Δ and Δt values using the offered load patterns of Figure 3.32. For both tenants, the proposed approach achieves high SLA satisfaction above 0.9 for almost all the combinations of Δ and Δt. It is also observed that the SLA satisfaction tends to decrease when increasing Δ. The reason is that large values of Δ reduce the adaptability to the offered load variation. It is also noticed that the maximum SLA satisfaction is achieved for different values of Δ depending on the value of Δt and these maximums are different for each of the tenants. For instance, for Tenant 1 the best performance in general is achieved for Δt=1 min and the maximum reward for this value is reached for Δ=0.01, whereas for Δt=3 min and Δt=5 min the maximum is obtained for Δ=0.05 and Δ=0.07, respectively. Instead, the best performance for Tenant 2 is achieved for Δ=0.01 for all values of Δt except for Δt=15 min. The differences between the behaviour observed for Tenant 1 and Tenant 2 suggest that the values of Δ and Δt should be selected according to the traffic behaviour in order to obtain the best possible SLA satisfaction.

Concerning the system utilization, as seen in Table 3-11, the highest value is achieved for Δt=1 min, and the lowest one for Δt=15 min, given that in this case resource quotas are updated at lower frequency, which leads to a lower adaptability to the dynamics of the traffic demands of the system. Regarding the impact of the value of Δ on the average system utilisation, it is once again observed that depending on the value of Δt, a different value of Δ maximises the system utilisation. However, the differences in utilisation as a function of Δ are in general small and a reduction trend is only observed for values of Δ beyond 0.05. This reveals that the selection of Δ and Δt has a greater impact on some performance metrics such as the SLA satisfaction than on others such as the average system utilization.

3.3.3.3 Conclusions and future work

The initial evaluation results of the multi-agent reinforcement learning solution based on DQN for RAN slicing in multi-tenant scenarios have led to the following conclusions:

- In the analysed case studies, the proposed approach allows properly adapting the assigned capacity to each tenant to their traffic requirements while achieving high service level agreement satisfaction (i.e. SLA satisfaction ratios higher than 0.95) and efficiently using the available capacity in the system (i.e. system utilisation values higher than 0.85).
- The proposed approach has been compared against two reference approaches, namely:
    - Reference 1 that considers a fixed capacity allocation in accordance with the bit rate

requirements in the SLA, and thus it provides a strict fulfilment of the SLA at the expense of a poor resource utilization.

- Reference 2 that assumes that the total capacity can be utilised by both tenants, and thus it provides a very high resource utilisation but at the expense of a poor SLA fulfilment.

The comparison has revealed that the proposed approach provides the best trade-off between resource utilisation and SLA satisfaction among all the strategies.

- A sensitivity analysis has been conducted to assess the impact of two important parameters of the algorithm, namely the action step Δ, which determines the increase/decrease in the resource quota allocation, and the time step duration Δt, which reflects the periodicity at which the resource quota is modified by algorithm. Results indicate that a trade-off exists when selecting the value of the action step Δ: in general, higher reward is achieved for low values of Δ but at the cost of higher fluctuations during the training process. Then, it has been obtained that Δ=0.03 appears to be an adequate choice as it provides the best aggregate reward. Concerning the time step Δt, results show that low values in the order of 1min provide the best performance in terms of both SLA satisfaction and resource utilisation thanks to a better reaction capability in front of offered load changes.

Overall, the results presented here reflect a promising behaviour of the proposed DQN-based RAN slicing approach. In this respect, some directions have been identified for further assessing the algorithm as work for future deliverable D4.3. First, the capability of generalizing the learnt policies will be studied, trying to explore to what extent the policy that has been learnt for a given tenant can also be successfully applied for another tenant. This would be relevant from a practical perspective as it would simplify the training process. Second, the behaviour of the algorithm when adding new tenants in the scenario will be analysed, trying to see if the previously learnt policies need to be updated. Last but not least, the operation of the algorithm in front of heterogeneous traffic distributions at multi-cell level will also be studied.

3.4 Optimal network access problem

As defined in 5G-CLARITY D4.1 [1], optimal network access is a combination of optimal communication resources matching and allocation to satisfy diverse requirements from users and services. Requirements in terms of QoS (maximum latency and minimal throughput) and network resources optimization. The problem combines two NP problems, many-to-many matching and bin-packing that make hard to deal in dynamic scenario. With the expansion of 5G and B5G networks, multi-WAT will be integrated to support complex and heterogenous network and services scenarios, with high user mobility and very dynamic traffic loads and changes in radio conditions. To deal with this issue, UEs will need to predict in advance consequent network states with changes in the radio conditions and choose the optimal and/or the intelligent access network policies to match and optimize resources while preserving the predicted QoS requirements. In this section we extend the introduction presented in 5G-CLARITY D4.1 [1] by formulating the optimal network access problem first as linear formulation followed by the design of our Deep Reinforcement Learning algorithm and initial experiments design and benchmarking. The complete and final model, results and conclusion will be included in 5G-CLARITY D4.3.

3.4.1 Problem statement and formulation

The Optimal Access Network Problem input are the Multi WAT infrastructure and QoS of service requests provided by UE telemetry and RAN telemetry (Section 2.2). In this sub section we present the problem formulation starting with the input parameters and decision variables and finishing with the linear formulation of the objective functions and key constraints.

3.4.1.1 Multi-WAT Infrastructure

- $G = (U, \Gamma, \Lambda)$: graph representing the multi-RAT/WAT access infrastructure where U Is the set of active User Equipment (UE) u with multi-RAT/WAT capability, Γ is the set of access nodes γ Wi-Fi, Li-Fi, 4G, and 5G. And Λ is the set of all wireless or radio channels/resources λ available grouped by access nodes γ i.e., $\Lambda = \{\Lambda \cup \Lambda_\gamma\} \forall \gamma \in \Gamma$.
- W is the set of levels of signal-to-interference-plus-noise ratio (SINR) w dB recorded by the UE telemetry from different network states.
- C is the of b_γ^λ maximum number λ channel/frequency/PRB available in an access node $\gamma \in \Gamma$
- Δ is the set of d_γ^λ minimum latency achieved by each λ channel/frequency/PRB resource in ms, associated to an access node $\gamma \in \Gamma$.
- M is the set of m_u multi-RAT/WAT capability in number of ports by each UE $u \in U$.
- N is the set of coefficients for spectrum efficiency n_w^γ of access node γ for each SINR level w recorded by the telemetry. 0 indicates non transmission capacity, and 1 corresponds to 100% of theoretical throughput. This coefficient is related to spectrum efficiency of each WAT technology as well as the modulation coding scheme.

3.4.1.2 Service Requests

- S is set of services s accessed by UEs.
- B is the set of β_s minimum throughput in bits per second required by the service s.
- L is the set δ_s of maximum latency in ns tolerated by the service s.
- T is the set of t time slots or network states measured by the UE telemetry.

3.4.1.3 Objective function

For a private network, like a factory, the owner of the factory should aim to max the throughput while the QoS of specific applications are satisfied, which should be the objective of our algorithm. (QoS limitation on bandwidth, packet average time delay, packet loss rate). In our model, a set of policy based on the minimization of the total QoS violation is defined with the equation

$$\min \sum_{u \in U} \sum_{\lambda \in \Lambda} \sum_{\gamma \in \Gamma} \sum_{s \in S} \sum_{\tau \in T} \sum_{w \in W} P_{u,\lambda,\gamma,s}^{w,\tau}$$

When: $P_{u,\lambda,\gamma,s}^{w,\tau}$ = 1, the policy determines that UE u must connect to service s by using the wireless or radio resource λ of the access node γ, during slot time or predicted network state t in case of level of SINR w, to enforce the best QoS possible. 0 otherwise.

After the model is executed the UE u, will receive a $\hat{P}_u$ set of $P_{u,\lambda,\gamma,s}^{w,t}$ = 1 obtained, considering one or many services s with different QoS requests, in all states t in all SINR levels scenarios reported w by the UE telemetric as historical data.

3.4.1.4 QoS constraint

The QoS constraints determines de minimum bandwidth and latency supported by the service scheduled. Two constraints are used to enforce the QoS the minimal bandwidth constraint and the maximum latency tolerated. The maximum bandwidth constraint is

$$\sum_{\lambda\in\Lambda}\sum_{\gamma\in\Gamma}\sum_{w\in W} n_w^{\gamma}\, b_{\gamma}^{\lambda} P_{u,\lambda,\gamma,s}^{w,\tau} \geq \beta_s^{\tau} \forall u \in U, s \in S, \tau \in T, \{b_{\gamma}^{\lambda}, \beta_s^{\tau} \in B\}$$

This equation determines the minimum throughput in bits per second β_s^{τ} required by the service *s*, active or measured at time slot τ. In which, b_{γ}^{λ} is the maximum bandwidth supported by the channel and access node multiply by a coefficient of signal degradation n_w^{γ} based on the wireless technology, MCS, and frequency used on each level of signal quality level SINR in *dB.*

The maximum latency tolerated is

$$\sum_{\lambda\in\Lambda}\sum_{\gamma\in\Gamma}\sum_{w\in W} \epsilon_w^{\gamma}\, d_{\gamma}^{\lambda} P_{u,\lambda,\gamma,s}^{w,\tau} \leq \delta_s^{\tau} \forall u \in U, s \in S, \tau\ \in T, \{d_{\gamma}^{\lambda}, \delta_s^{\tau} \in \Delta\}$$

This equation determines the maximum delay δ_s^{τ} tolerated by service s in the given slot time τ. In which the d_{γ}^{λ} is the end-to-end latency obtained by a reference point used in a specific channel and access node in the reach of the UE. Similar previous equation the ϵ_w^{γ} is a coefficient of delay added by the signal quality or SINR which indicates the delay added by packet drop or retransmissions.

3.4.1.5 Multi-Connectivity constraints

Multi-connectivity constraint implements the multi-WAT aggregation in the UE (e.g., MPTCP) the following equation determines the maximum aggregation or WAT ports or channels n_u supported the UE *u.*

$$\sum_{\lambda\in\Lambda}\sum_{\gamma\in\Gamma}\sum_{w\in W} P_{u,\lambda,\gamma,s}^{w,\tau} \leq n_u \ \forall\ u \in U, w \in W, s \in S, \tau \in T$$

The following equation enables de multi-connectivity traffic distribution bounded by the maximum aggregation supported by the UE.

$$\sum_{\lambda\in\Lambda}\sum_{\gamma\in\Gamma}\sum_{w\in W} \{b_{\Lambda}^{\lambda} P_{u,\lambda,\gamma,s}^{w,\tau} + 0.5 b_{\gamma}^{\lambda} P_{u,\lambda,\gamma,s}^{w,\tau} + \{b_{\gamma}^{\lambda}/n_u\} P_{u,\lambda,\gamma,s}^{w,\tau}\} \geq \beta_s^{\tau}\ \ \forall\ u \in U, s \in S, \tau \in T$$

3.4.1.6 WAT channel constraint

This equation enables de WAT channel capacity of each access node considering the capacity for channel aggregation and sharing if allowed (e.g., Wi-Fi).

$$\sum_{s\in S} \{b_{\gamma}^{\lambda} P_{u,\lambda,\gamma,s}^{w,\tau} + 0.5 b_{\gamma}^{\lambda} P_{u,\lambda,\gamma,s}^{w,\tau} + \{b_{\gamma}^{\lambda}/n_u\} P_{u,\lambda,\gamma,s}^{w,\tau}\} \leq c_{\lambda}^{\gamma}\ \ \forall\ u \in U, \lambda \in \Lambda, \gamma \in \Gamma, w \in W, \tau \in T$$

3.4.1.7 Access network policy assignment constraint

Finally, the following two equations ensure the assignment of policy of each services request on each UE *u* based on different levels of SINR *w* in all measured states and channels.

$$\sum_{\lambda\in\Lambda}\sum_{\gamma\in\Gamma}\sum_{w\in W} P_{u,\lambda,\gamma,s}^{w,\tau} \geq 0\ \forall\ u \in U, s \in S, w \in W, \tau \in T$$

$$\sum_{\lambda\in\Lambda}\sum_{\gamma\in\Gamma}\sum_{w\in W} P_{u,\lambda,\gamma,s}^{w,\tau} \leq 1 \forall\ u \in U, s \in S, w \in W, \tau \in T$$

3.4.2 Proposed Solution and Initial Implementation

After defining 5G-CLARITY D4.1 and formulate Optimal Access Network Problem in Section 3.4.1, in this subsection we introduce the architecture and flows of our proposed solution of our initial implementation

followed by preliminary tests and input parameters obtained.

3.4.2.1 Architectures and flows

The model described above can be solved as Mixed-Interfere Linear Programming (MILP) only for static and small scenarios as a result we design an artificial intelligence (AI) based algorithm and architecture to solve the optimal access network problem in dynamic scenario. Our model is designed for three types of architectures:

a) **User centric**: User Equipment (UE) monitors the APs state and takes its connection decisions based on threshold performance parameters, and other WAT/RATs characteristics. The mayor challenge for heuristics or AI based models is the lack of knowledge of the whole network and limited capacity of the UE to host and process large volume of data and power.

b) **RAN-Assisted:** An information exchange is done between the access node (WAT/RAT) and the UE to decide what access node to connect. It provides broader feedbacks that cannot be measure locally the UE. However, still it requires large storage and computing capability in the UE as the user centric architecture.

c) **RAN-Controlled:** Architecture adopted by 3GPP for addressing dual-connectivity issues with capacity for centralized or distributed decision normally taken by one or multiple Radio Access Network (RAN) controller/s that oversees various WAT/RAT networks. In this architecture UE does not store or process large information, only report periodically performance measurements (e.g., SNR). The controller combines the feedbacks from Multi-WAT/RAT networks through their UEs to build an overall view of the network.

However, at this stage our first model is based on the RAN-controlled architecture with the capability to be extended to the other two architectures to be more adaptable. In addition, the RAN-controlled architecture match better with the 5G-CLARITY Infrastructure stratum and Intelligent stratum. The proposed AI tool is designed to be deployed on the 5G-CLARITY AI Engine as introduced in the Figure 3.38.

Figure 3.35 (left side) summarizes the architecture of the solution as well-as its initial implementation in simulated environment. In this example, four access nodes (ANs) are connected to a node emulating 5G-CLARITY RAN cluster, then the AI engine or solution is deployed with the data lake and the telemetric control centre in a node emulating 5G-CLARITY edge node. The service accessed in this example is hosted also in the node emulating the edge cluster.

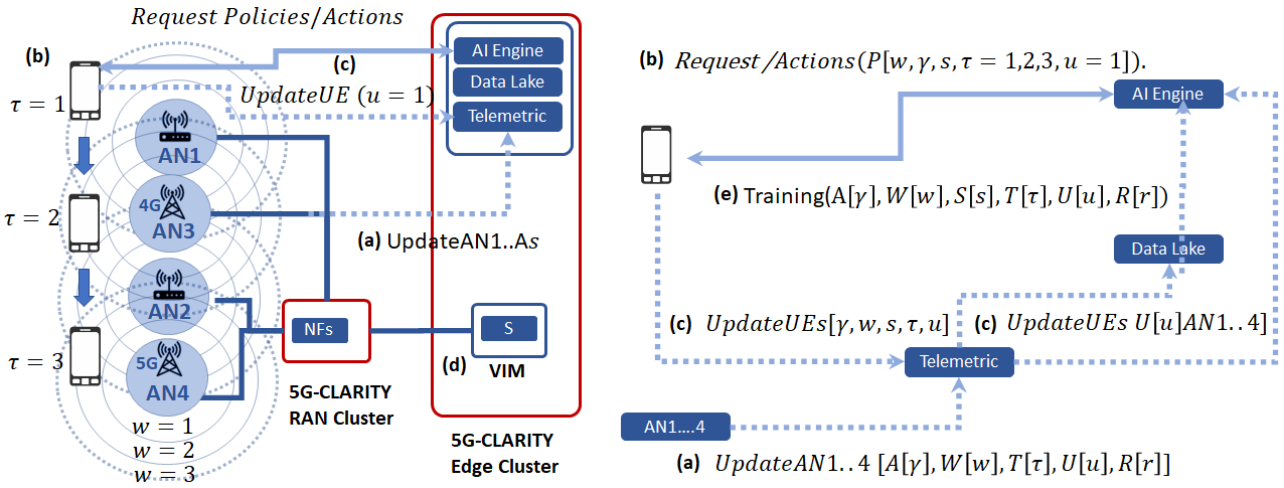


Figure 3.35. Simulation architecture overview and main flow

Figure 3.38 (right side) describes the main flows of our framework, connected to the right side by references (a)-(c). Figure 3.38(a) presents the first step in which the access nodes send regular updates of network status $UpdateAN1..4\ [A[\gamma], W[w],\ T[\tau], U[u],\ R[r]]$ the data sent the set A of active ANs (own and neighbors), set W of different levels of SINR associated to neighbors ANs and UEs, set of network states T and set of U UEs connected in each state τ and finally, the set of resource R used and available per state of the sender AN (e.g., MCS, number of PRBs(5G/4G), and channels from Wi-Fi available). Then the UE send the request for access network policy assuming the AI/optimization framework already finished a training process with large data set of network states (Figure 3.38(b)). Finally, the AI engine and optimization model sent a set of predicted network states and optimal access network policies covering each state and each UE assuming 2x Multi-Connectivity Wi-Fi and LTE or 5GNR.

3.4.2.2 Deep Q Learning Algorithm Model

To design our approach, we define three main elements based on the problem formulation:

a) **State space** defined as (resources available for service) $\hat{S} = \{f_{\lambda,\gamma}^{\tau,w}, ..\}\ \forall\ \lambda \in \Lambda, \gamma \in \Gamma, w \in W, \tau \in T$ where:

$$\sum_{s\in S}\sum_{u\in U}\{b_\gamma^\lambda P_{u,\lambda,\gamma,s}^{w,\tau} + 0.5 b_\gamma^\lambda P_{u,\lambda,\gamma,s}^{w,\tau} + \{b_\gamma^\lambda / n_u\} P_{u,\lambda,\gamma,s}^{w,\tau}\} - c_\lambda^\gamma = f_{\lambda,\gamma}^{\tau,w}\ \ \forall\ \lambda \in \Lambda, \gamma \in \Gamma, w \in W, \tau \in T$$

b) **Action space** is defined by $\hat{A} = \{P_{u,\lambda,\gamma,s..}^{w,\tau}\}\ \ \forall\ u \in U, s \in S, w \in W, \tau \in T$

c) **Reward function**: for each access node $\gamma \in \Gamma$ and time slot $\tau \in T$ is:

$$R_\gamma^\tau = \sum_{\lambda\in\Lambda}\sum_{u\in U}\sum_{w\in W}\sum_{s\in S}\left\{P_{u,\lambda,\gamma,s..}^{w,\tau}\left\{\frac{|c_\lambda^\gamma - f_{\lambda,\gamma}^{\tau,w}| + \beta_s^\tau}{|c_\lambda^\gamma - f_{\lambda,\gamma}^{\tau,w}|}\right\}\right\}\ \forall\ \gamma \in \Gamma,\ \tau \in T$$

The given reward function will provide incentive to the algorithm to allocate all their resources. By using the dynamic policy, the proposed AI will solve the dynamic policy for optimal access network in a multi-WAT network. Since the state space is large and dynamic, we design a Deep Q Learning based solution. The Pseudo-Code of the algorithm is summarized in Algorithm 1.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

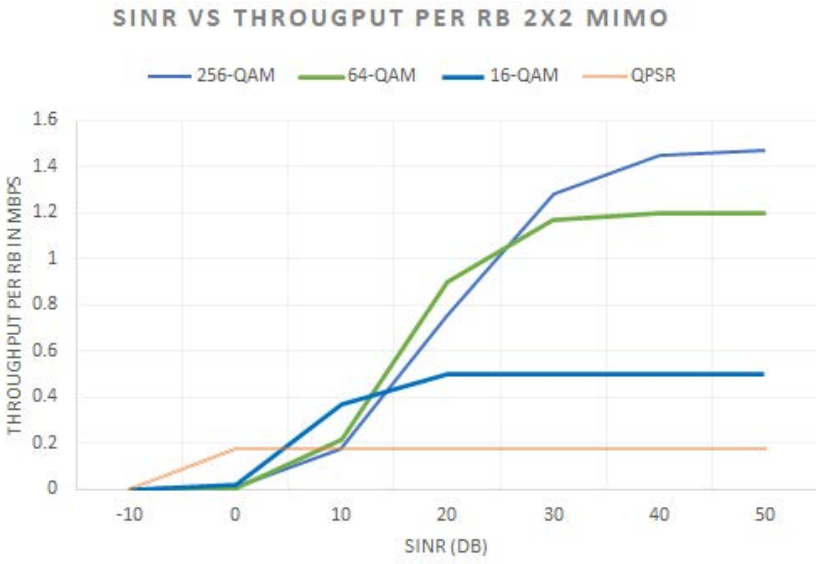
3.4.3 Evaluation methodology and Initial Results

During this first stage of modelling and designing of our solution we began: *(i)* a round of test to determine appropriated inputs and parameters for our model; *(ii)* an implementation and test of an experimental

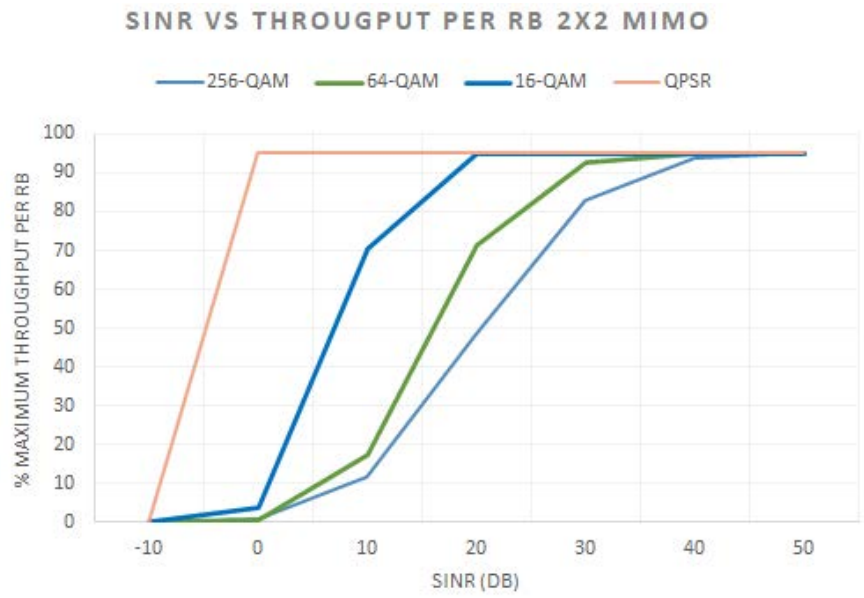
environment using NS3 to setup large scale simulations; *(iii)* solved as MILP as benchmark to compare and tune parts of our proposed DRL solution.

3.4.3.1 Input data and parameters

We performed an initial evaluation considering LTE/5GNR radio and three Wi-Fi standards. The LTE/5GNR considered uses 50 MHz of bandwidth and 275 RBs (aka PRBs) per channel. We use the average PRB throughput in bits per second coding scheme or Modulation Coding Scheme (MCS) measured per SINR (Figure 3.36). The given parameters measured by UE telemetry obtained an average between 0.2 and 1.45 Mbps per RB (or between 55 Mbps to a maximum of 400 Mbps). Figure 3.36 (b) also presents a pattern of spectrum efficiency and sensibility to SINR levels based on the MCS.



(a)



(b)

Figure 3.36. SINR *w* vs RB throughput per MCS

Table 3-12. Maximum and Minimum Throughput per SINR level Wi-Fi, LTE/5GNR

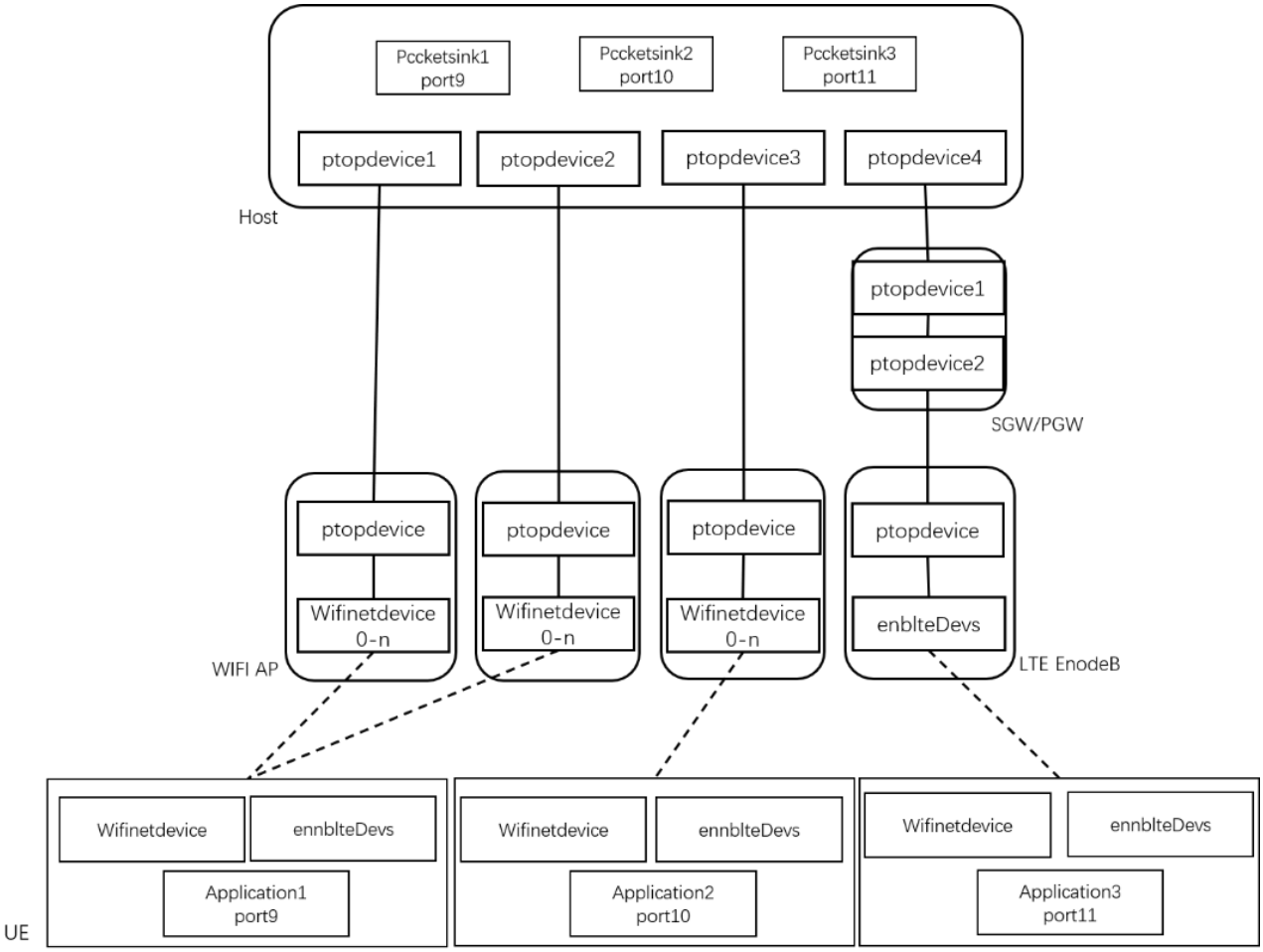
Level	Protocol/Code	Wi-Fi 802.11x	LTE/5G QPSR	LTE/5G 16-QAM	LTE/5G 64-QAM	LTE/5G 256-QAM
1	**Minimum SINR (dB)**	**> 25**	**> 0**	**> 20**	**> 30**	**> 45**
	Minimum average throughput in Mbps per channel or resource block (RB) = 90% <	10 < 50 < 140 <	0.180 <	0.650 <	1.18 <	1.45 <
2	**Max SINR (dB)**	**< 25**	**< 0**	**< 20**	**< 30**	**< 45**
	Maximum average throughput in Mbps per channel or resource block (RB) = < 90%	< 10 < 50 <140	> 0.180	> 0.650	> 1.18	> 1.40
	Min SINR (dB)	**10 <**	**- 5 <**	**8 <**	**15 <**	**20 <**
	Minimum average throughput in Mbps per channel or resource block (RB) = 50% <	6 < 20 < 70 <	0.090 <	0.300 <	0.600 <	0.800 <
3	**Max SINR (dB)**	**< 10**	**< -5**	**< 8**	**< 15**	**< 20**
	Maximum average throughput in Mbps per channel or resource block (RB) = < 50%	< 6 < 20 < 70	< 0.090	< 0.300	< 0.600	< 0.800
	Min SINR (dB)	**2 <**	**-8 <**	**2 <**	**5 <**	**8 <**
	Maximum average throughput in Mbps per channel or resource block (RB) = 10% <	1 < 4 < 10 <	0.020 <	0.080 <	0.10 <	0.15 <
4	**Max SINR (dB)**	**< 2**	**< -8**	**< 2**	**< 5**	**< 8**
	Maximum average throughput in Mbps per channel or resource block (RB) = < 10%	< 1 < 4 < 10	< 0.020	< 0.080	< 0.10	< 0.15

We assumed the impact of SINR to the end-to-end latency from the UE to the application considering different types of services or QoS requirements we confirm the latency proportionality to the SINR level given the increase of number of packet retransmission[5]. Table 3-12 present the parameters used during the first simulations.

3.4.3.2 Emulation Model in NS3

The emulation of real multi-WAT technology is in process to be implemented in NS3 network simulator. The network structure of our initial test is shown on Figure 3.37(a). Then Figure 3.40(b) presents an illustrative example of simulation with our model emulating three UEs, three Wi-Fi ANs with 3 channels each and one AN LTE/5GNR. In this example we have four types of services, voice, video, web, and background traffic generated on each UE and AN. The Figure 3.37(b) presents, the data including delay, jitter, throughput, drop rate of channel in Wi-Fi and LTE, collected in the emulation.

[5] Calculation were performed using MATLAB and real signal proceessing data collected by the UE telemetry



(a)

```
Flow : 1 (192.168.1.4 -> 1.1.2.2)
 Tx bytes : 256688
 Rx bytes : 254584
 First Tx Pkt time : 4.00819
 Last Rx Pkt time : 9.99297
 delay : 2.58687
 Jitter 0.0366704
 Wifi Throughput : 0.324544 Mbps

Flow : 2 (192.168.1.5 -> 1.1.2.2)
 Tx bytes : 256688
 Rx bytes : 254584
 First Tx Pkt time : 4.00819
 Last Rx Pkt time : 9.99265
 delay : 2.5019
 Jitter 0.00812
 Wifi Throughput : 0.324561 Mbps

Flow : 3 (192.168.1.6 -> 1.1.2.2)
 Tx bytes : 256688
 Rx bytes : 254584
 First Tx Pkt time : 4.00819
 Last Rx Pkt time : 9.99297
 delay : 2.5788
 Jitter 0.0360198
 Wifi Throughput : 0.324544 Mbps
```

```
Flow : 1 (7.0.0.2 -> 1.1.1.2)
 Tx bytes : 256688
 Rx bytes : 253532
 First Tx Pkt time : 4.00819
 First Tx Pkt time : 4.03193
 Last Rx Pkt time : 9.99793
 delay : 5.17712
 Jitter 0.685088
 LTE Throughput : 0.322935 Mbps

Flow : 2 (7.0.0.3 -> 1.1.1.2)
 Tx bytes : 256688
 Rx bytes : 253532
 First Tx Pkt time : 4.00819
 Last Rx Pkt time : 9.99793
 delay : 5.17733
 Jitter 0.685088
 LTE Throughput : 0.322935 Mbps

Flow : 3 (7.0.0.4 -> 1.1.1.2)
 Tx bytes : 256688
 Rx bytes : 253532
 First Tx Pkt time : 4.00819
 Last Rx Pkt time : 9.99793
 delay : 5.17754
 Jitter 0.685088
 LTE Throughput : 0.322935 Mbps
```

(b)

Figure 3.37. (a) NS3 Emulation. (b) Monitoring data for further experiment for Wi-Fi (left) and LTE (right).

3.4.3.3 Preliminary results

To complete the final design of our AI model and architecture we create an initial benchmarking by solving the optimal access network problem with MILP solver and with an early version of the Q-learning non-gradient training function. After the initial optimization, a data set is generated using random generated values emulating the UE telemetry parameters. Figure 3.38 (a) presents a snapshot of the initial results presenting a prediction of next two network states *t=2*, and *t=3* describing a movement from AN 1 to AN 4. The model also predicted the optimal access network policy for each states considering other UEs and the traffics and usage of the infrastructure (Figure 3.38(b)(c)). The scalability and the practical implementation of solutions in networks on production is the expected benefits of the DRL model in design.

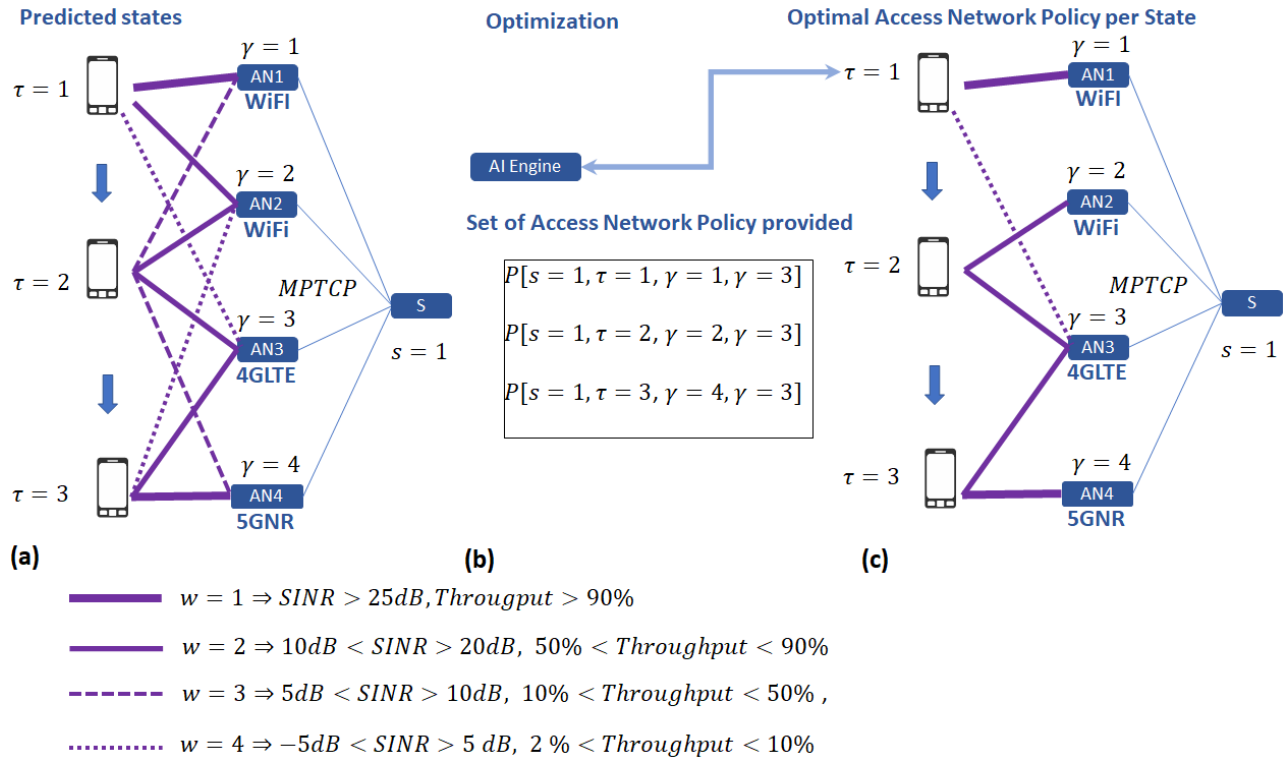


Figure 3.38. Example of results obtained.

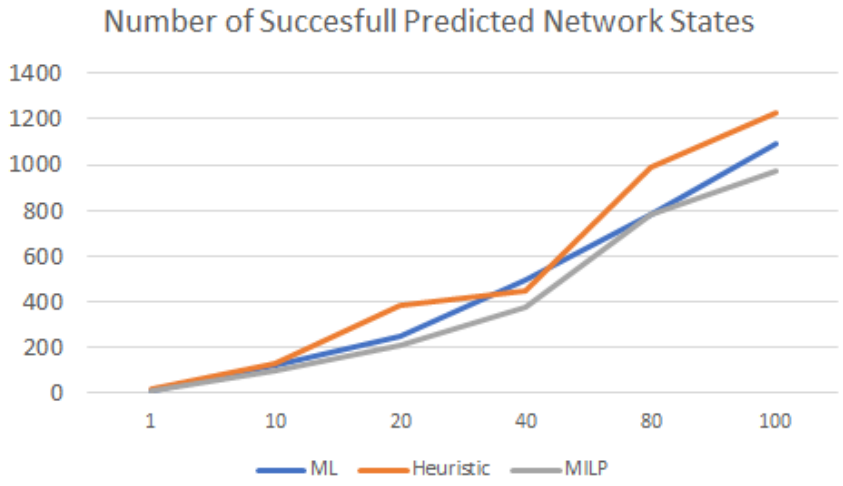


Figure 3.39. Successful Predicted/Calculated Optimal Network States on between 1 to 100 UEs

The success of prediction where measure based on the obtained results from the MILP and an early implementation of our DRL model using the reward and actions functions introduced. The MILP obtained the accurate number of predictions, and a brute force heuristic solution miss predicts near 20% and by deploying the proposed DRL functions it tends to reduce to 5%.

3.5 Indoor ranging with NLoS awareness

3.5.1 Algorithm design

3.5.1.1 Introduction

In 5G-CLARITY D4.1 [1] we described the details of ML-based Non-Line-of-Sight (NLoS) identification and NLoS-aware ranging. In particular, the Support Vector Machine (SVM) classification/regression method features were presented, its advantages and disadvantages were discussed, and its potential for NLoS-aware ranging was revealed.

In this deliverable we introduce a new approach based on DNNs, which tackles the disadvantages of SVM, i.e. increase of computational complexity with the growth of training set. In other words, SVM acts as a baseline to DNN-based NLoS identification and ranging. In what follows, we firstly describe the characteristics of DNN-based NLoS identification and ranging. Subsequently, the performance of SVM and DNN approaches are compared by means of an extensive simulation and measurement campaign.

3.5.1.2 DNN for NLoS-aware Ranging

DNNs have shown superb performance in a wide variety of classification problems, such as image classification, handwriting classification, etc. In the context of NLoS identification, their potential performance has already been revealed in [49]. Furthermore, recent works in Fingerprinting (FP) localization, such as [50], suggest that DNNs show great potential to increase the precision of localization/ranging, especially in indoor environments.

Given that, we leverage three DNN blocks, namely NLoS identifier, NLoS ranging, and LoS ranging, to design an NLoS-aware ranging algorithm for high precision indoor ranging (shown in Figure 3.40). It receives the Channel State Information (CSI) measurements as input, applies Fast Fourier Transform (FFT) to transform it into frequency domain, and feeds the output of the FFT block to the DNNs. Depending on the outcome of the NLoS-identifier block, one of the arms is activated to perform ranging. All of the DNN blocks are trained offline and able to immediately return a prediction/estimation.

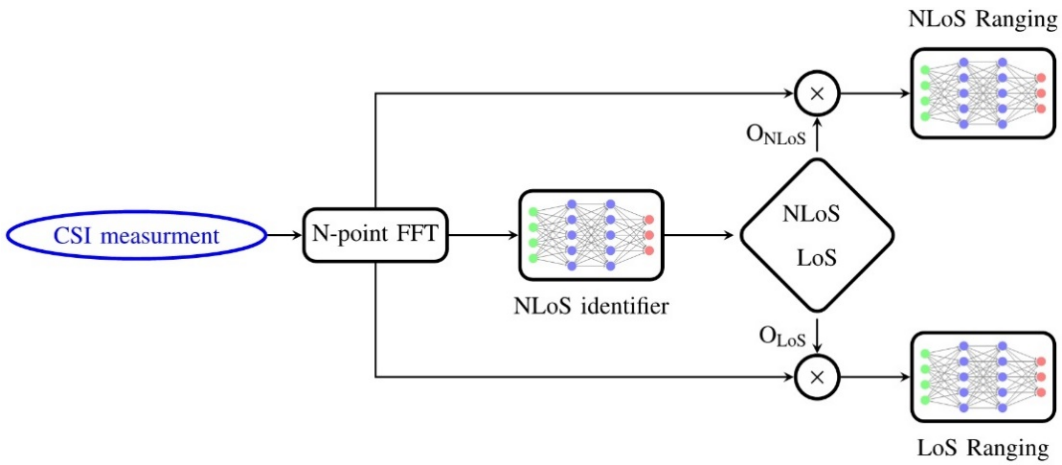


Figure 3.40. The end-to-end DNN-based model for NLoS-aware ranging. The vector [O_{NLoS}, O_{LoS}] can only take the value [0, 1] or [1, 0] to assure that only one of the arms in the diagram carries out the ranging.

In the sequel, we discuss the characteristics of each block as well as their relation among each other.

3.5.1.2.1 NLoS Identifier

We employ the DNN in Figure 3.41 to perform the NLoS identification, whose purpose is to distinguish between LoS and NLoS links. The network comprises an input layer, two hidden layers, and an output layer. Choosing the number of hidden layers and their respective neurons is always a challenging task and requires a great deal of intuition. Nevertheless, as a rule of thumb, for a classification problem, it's suggested that the number of neurons is selected in the interval between [N_input, N_class] [51]. The selection of the number of hidden layers is carried out by trial-and-error, where we did not observe any significant improvement for more than two layers. Moreover, the network's input for training is the normalized magnitude of the Channel State Information (CSI) vectors collected under two scenarios, i.e., LoS and NLoS, in multiple positions.

3.5.1.2.2 Ranging

To have a robust high precision ranging algorithm, we draw on the fingerprinting approach [50]. In particular, ranging is treated as a classification problem where we aim to train a DNN-based fingerprint classifier that is then utilized to classify a collected CSI into one of the fingerprints, thereby estimating its corresponding distance from the AP.

To this end, we train two structurally equal models, one for the LoS scenario and one for the NLoS, following the model shown in. The former is trained using only the LoS CSI data while the latter is trained by means of NLoS CSIs. Both models have similar architectures, as shown in Figure 3.41. Furthermore, a Weighted Arithmetic Average (WAA) unit is clipped to the output of each ranging model to compute the final ranging estimation.

We note that the output of the network is a vector indicating the probability of each class into which the input CSI may be classified. The NLoS identifier and ranging DNN models can then be combined in the manner depicted in Figure 3.40, where we first feed a collected CSI to the NLoS identifier network and, subsequently, based on its outcome, i.e., LoS or NLoS, one of the two ranging models is picked to conduct the ranging.

3.5.2 Evaluation methodology

In this section we present the principles of evaluation for our baseline model, i.e., the model based on SVM described in D4.1 [1], and those considered for the DNN-based model proposed in this deliverable.

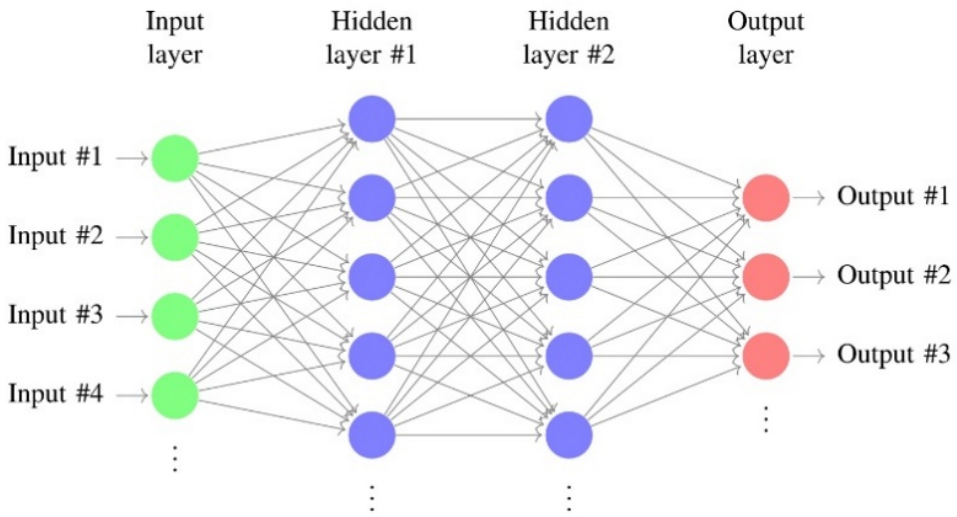


Figure 3.41. Architecture of the DNN used for NLoS-aware ranging.

Table 3-13. Features Extracted to Feed to SVC/SVR

Total energy	$\varepsilon = \int \lvert r(n)\rvert^2 dt$	Maximum amplitude	$maxAmp = \max_t \lvert r(t)\rvert$
Mean excess delay	$\tau_m = \int t \frac{\lvert r(t)\rvert^2}{\varepsilon} dt$	RMS delay	$\tau_{rms} = \int (t-\tau_m)^2 \frac{\lvert r(t)\rvert^2}{\varepsilon} dt$
Mean	$\mu = \frac{1}{T}\int \lvert r(t)\rvert dt$	Variance	$\sigma^2 = \frac{1}{T}\int (\lvert r(t)\rvert - \mu)^2 dt$
Skewness	$\kappa = \frac{1}{\sigma^3 T}\int (\lvert r(t)\rvert - \mu)^3 dt$	Kurtosis	$\kappa = \frac{1}{\sigma^4 T}\int (\lvert r(t)\rvert - \mu)^4 dt$

3.5.2.1 SVM evaluation methodology

The first step towards running the SVM-Classifier (SVC) is training. To this end, the CSI data is pre-processed to, for example, extract the relevant features. These relevant features, which have proved to be essential according to [52], are as shown in Table 3-13.

Once the desired features are extracted, we need to obtain the weights of SVM, the kernel type, and its parameters. A kernel is a non-linear transformation applied to the features to map them into high dimensional spaces and, consequently, rendering them more separable. For the purpose of this work, we choose Radial Basis Function (RBF) with parameters *C* and γ. The *C* parameter trades off correct classification of training examples against maximization of the decision function's margin. The γ parameter defines how far the influence of a single training example reaches. An optimization technique such as grid search can then be employed to find the optimal values for the aforementioned parameters. All these can be readily done using *sklearn* python package without much human intervention.

For the ranging problem, we draw on the SVM Regressor (SVR), as initially referred to in D4.1 [1]. The same steps taken to pre-process the data and to select the suitable kernel are repeated for the SVRs as well. In particular, we train two models, one only trained by LoS CSI data while the other is trained only using NLoS CSI data. Furthermore, based on the output of SVC, one of the SVRs is selected to perform ranging. The general structure resembles that of Figure 3.41. However, the DNNs are replaced by SVC and SVRs. Moreover, instead of the FFT of CSI, we use the time-domain CSI samples. This is, in particular, due to the fact that manually computing the features relevant for NLoS-identification and FP, i.e., those mentioned in Table 3-13, is not possible as they are time-domain features.

3.5.2.2 DNN evaluation methodology

The DNN-based NLoS-aware ranging employs two DNN models, one for NLoS identification and one for ranging. Note that, in the latter, we train the model twice and save two sets of weights, one of which is then chosen depending on the channel condition, i.e., LoS or NLoS. We use the Keras framework of Python to implement the neural networks. Table 3-14 describes the detailed description of hyperparameters for each network.

Table 3-14. Hyperparameters of DNNs for NLoS-Aware Ranging.

Network	# of inputs	# of hidden layers (# of neurons)	# of outputs (function type)	Optimizer (parameters)	# of epochs	batch size	Cost function

NLoS-identifier	50	2 (34, 18)	2 (softmax)	Adam (lr=0.001, beta_1=0.9, beta_2=0.999)	10	16	Binary cross-entropy
Ranging	50	2 (62, 74)	86 (softmax)	Adam (lr=0.001, beta_1=0.9, beta_2=0.999)	25	16	Categorical cross-entropy

3.5.2.3 Simulation scenario and CSI collection

To evaluate the performance of the proposed ML algorithm, we consider the scenario in Figure 3.41. The CSIs are collected under LoS and NLoS conditions for the 86 red fingerprints in a big office hall of 198 m^2. An SDR N310 is programmed to periodically propagate m-sequences in the environment. At the receiver side, another SDR mounted on a trolley records the sequences at each fingerprint. By pre-processing the received signal at the receiver, we can extract the desired CSIs. Further parameters of the measurement campaign can be found in Table 3-15.

Given this setup, we collect 100 CSIs at each fingerprint or, alternatively, 8600 channel realizations for each scenario, i.e. LoS and NLoS. The NLoS scenario is created by putting a whiteboard in the middle of the transmitter and the receiver to block the direct path. Another crucial remark to make is that, while we can use the absolute value of the FFT of the CSIs as input to the DNNs, we need to extract the feature matrix ***X*** from the CSIs to be able to feed them to the SVM/SVR. A detailed description of these features has already been provided in the previous subsections.

Table 3-15. Parameters of CSI measurement campaign.

Frequency	Bandwidth	# of TX/RX antennas	# measurments per fingerprint	Resolution of fingerprinting	Height of TX/RX
5.2 GHz	20 MHz	1	100	1 m	2.5 m/1.0 m

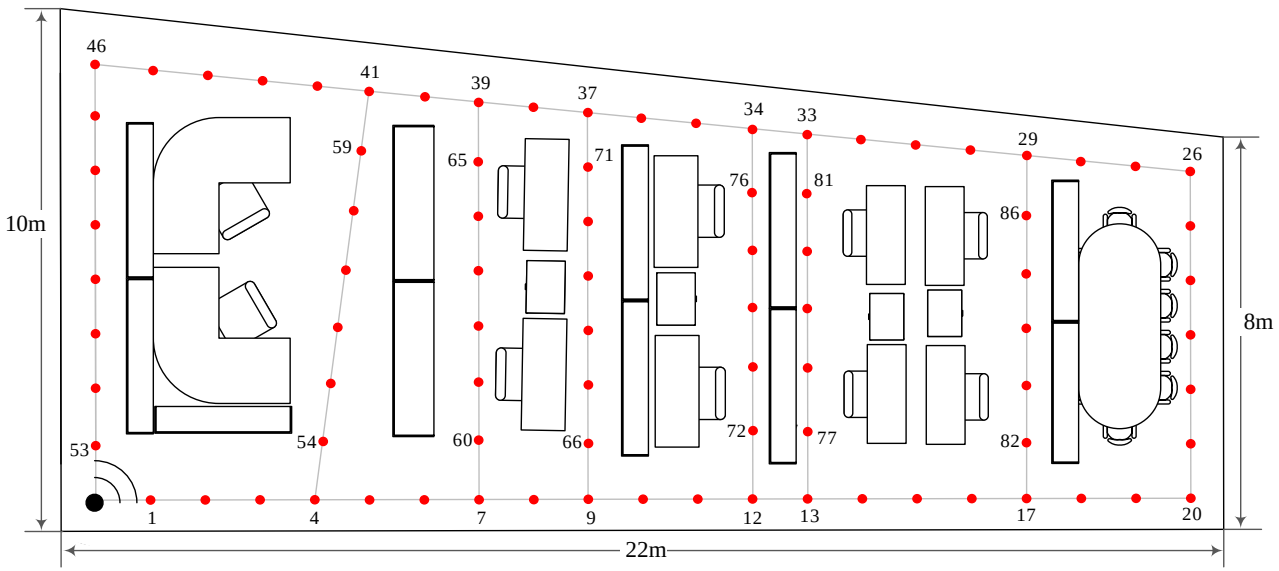


Figure 3.42. Layout of the office hall.

Finally, the collected CSI set is split into two sets, a training set comprising 80% of the data and a test set consisting of 20% of the CSIs. For the SVM model, both sets are pre-processed to extract the features, while for the DNN model, we only perform FFT on the sets before we feed them to the network.

It must be noted that a more reliable test-set would be obtained by conducting measurements at random positions in the environment, meaning that the results acquired by this dataset are considered to be optimistic. More realistic results will be provided in the next WP4 release by conducting an extensive measurement campaign to include random positions in the dataset. Nevertheless, the current dataset suffices to show to the superior performance of the DNN-based algorithm and the importance of NLoS-awareness when performing ranging.

3.5.3 Evaluation results and discussion

We evaluate the performance of the proposed algorithms in terms of LoS/NLoS prediction accuracy and Cumulative Density Function (CDF) of the ranging error. To evaluate the performance of SVM-based algorithm, once the models are trained, the feature vector corresponding to each CSI in the test set is fed into the SVC, where the channel condition (LoS and NLoS) is predicted. Subsequently, based on the predicted value, one of the SVRs is chosen to perform ranging. The same procedure is repeated for the DNN-based algorithm, with the inputs being the magnitude of the FFT of the CSIs.

Figure 3.43 shows the LoS/NLoS false alarm probabilities and the prediction accuracy of the proposed algorithms. The LoS/NLoS false alarm probability is calculated by

$$Pf_{LoS} = \frac{Nf_{LoS}}{Nt_{LoS}+Nt_{NLoS}+Nf_{LoS}+Nf_{NLoS}}, \qquad Pf_{NLoS} = \frac{Nf_{NLoS}}{Nt_{LoS}+Nt_{NLoS}+Nf_{LoS}+Nf_{NLoS}},$$

where Nt and Nf stand for "number of true detected" and "number of false detected", respectively. The accuracy can be then computed by $P_{acc} = Pf_{LoS} + Pf_{NLoS}$.

As can be seen from Figure 3.43, the DNN-based algorithm outperforms the SVM-based by a significant margin. The reasons of that are twofold. On the one hand, DNN can efficiently extract the relevant features without human intervention, which can often be flawed due to the lack of profound understanding of the phenomenon in question. In other words, the DNNs perform a more systematic feature engineering, thereby extracting the most relevant features and enhancing the accuracy of classification. On the other hand, DNNs possess a remarkable capability to implement the non-linear functions, which, from the classification point of view, enables the classifier to define almost any decision boundary to separate the classes.

Figure 3.44 presents the CDF of the ranging error for SVM- and DNN-based NLoS aware ranging algorithms. As can be seen, the DNN-based algorithm outperforms the SVM-based algorithm by a large margin, mainly due to the following reasons.

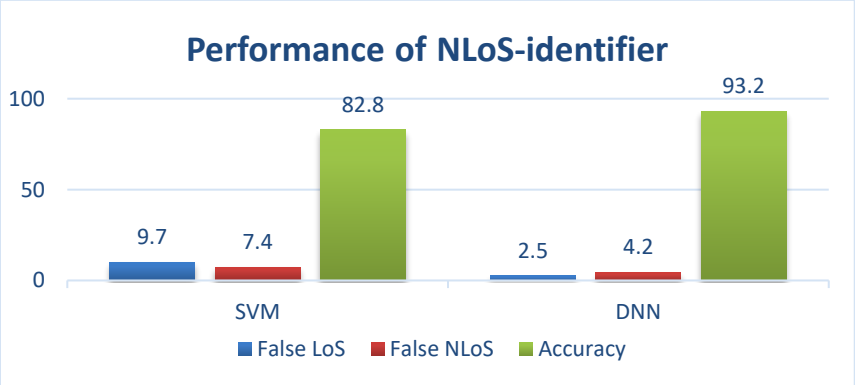


Figure 3.43. Performance of SVM and DNN -based NLoS identifier.

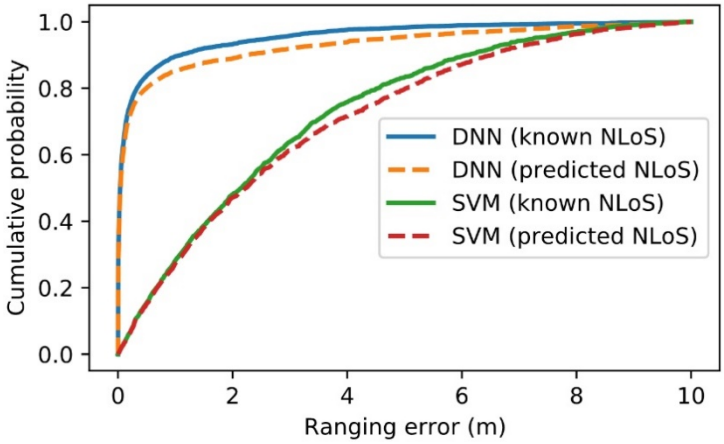


Figure 3.44. Performance comparison of NLoS-aware ranging using DNN-based FP and SVM.

On the one hand, NLoS identifier predicts the channel condition with a higher accuracy, whereby preventing a LoS/NLoS CSI to be fed to the SVR (or DNN) corresponding to the NLoS/LoS. On the other hand, the precision of ranging conducted by DNN is higher than that of SVR as it has intrinsically more capabilities to extract the particularly suitable features for ranging. On the contrary, the features for the SVM model need to be handpicked, which often leads to the deterioration in performance when compared to DNN models.

3.6 Resource partitioning in a multi-technology RAN

3.6.1 Initial implementation

This ML algorithm shares some similarities with the multi-tenancy use case exposed in Section 3.3. In both cases, the ML algorithm uses a DQN approach to allocate radio resources to each slice. However, there are also significant differences causing the resource allocation problem to be addressed in a different way. In particular, the proposed algorithm in this section focuses on the resource provisioning problem of an industrial network scenario, where the following distinctions can be drawn:

- Overall, in industrial scenarios only one operator prevails. The industrial network in this problem can be considered as a standalone NPN managed by a unique private operator.
- The radio access network of the industrial scenario in this problem is composed of different radio access technologies (i.e., 5GNR and Wi-Fi), which will be leveraged to satisfy the variety of traffic demands of different nature (e.g., wide variety of services with different QoS requirements).
- As pointed out previously, the services offered in an industrial network are very different in terms of requisites. In this problem we face with two of the most representative types of 5G services: URLLC services are considered for the industrial controllers deployed along the production lines of the factory and eMBB services are requested by the factory workers for applications such as video streaming and augmented reality.

3.6.1.1 System model

Considering a two-tier Heterogeneous Network consisting of a set $B = B_G \cup B_W$ of cells, where B_G is the set of 5GNR cells and B_W is the set of Wi-Fi cells, all managed by a given network operator. On top of this infrastructure, multiple services can be provided on separate 5G-CLARITY slices. Each service is provided on a separate 5G-CLARITY slice to guarantee its specific requirements. Let C be the set of 5G-CLARITY slices that are deployed in the network. There are two kind of services that can be delivered: eMBB and URLLC. For example, there can be one URLLC service and two eMBB services that are provided on three different 5G-CLARITY slices. Due to the technology limitations, URLLC services can only be served on 5GNR cells. On the

other hand, eMBB services can be served on 5GNR, Wi-Fi, or simultaneously through 5GNR and Wi-Fi cells, leveraging the AT3S functionality that is supported by the 5G-CLARITY system. The eMBB traffic offloading to Wi-Fi can be maximized to achieve better system performance. To that end, we assume ideal response of the AT3S function for updating the routing factor (please, for more information about the AT3S functionality in 5G-CLARITY D3.2 [14]).

The service demand of each 5G-CLARITY slice is non-uniformly distributed over the considered area, where a set U of user equipment (UEs), both static and mobile, exist in the scenario. Let us represent the subset of UEs belonging to slice c using the variable U_c, $c \in C$. Each UE of an eMBB service demands a capacity D_{eMBB}, while each UE of a URLLC service demands a capacity D_{URLLC} and a delay in the access network less than τ_{max}.

The UEs of a slice should be provided with enough resources to satisfy their service requirements (throughput, latency). To that end, we use the concept of 5G-CLARITY quota, which enables isolation of radio resources between the different 5G-CLARITY slices, to guarantee service performance. Let $\xi_{c,b}$ be the quota for slice c in cell b. In 5GNR cells, the 5G-CLARITY quota corresponds to the 5G-CLARITY wireless maximum quota, which defines the maximum resource usage quota for a given slice. PRBs can be exclusively allocated to one slice (i.e., no other slice can use them) or shared between slices. We adopt the shared resources approach as it allows to maximize resource usage in the system. For shared resources, PRBs are allocated on demand, i.e., when the slice needs to use them. In addition, a fraction of these resources can be guaranteed to one slice in case of resource scarcity (prioritized resources). This quota is referred to as the 5G-CLARITY wireless minimum quota. More details on 5G-CLARITY quotas can be found in Section 2.1.1.1.

The 5G-CLARITY wireless maximum quota can be adapted to handle the varying environmental conditions and user traffic dynamics. The lower bound for this quota is given by the quota for prioritized resources, or 5G-CLARITY wireless minimum quota. In this case, the slice would not use resources other than the guaranteed fraction of PRBs. On the other side, the upper bound is the system bandwidth, so that the slice could use as many free PRBs as possible.

In Wi-Fi cells, the quota is given by the airtime (i.e., the wireless transmission time usage). Since Wi-Fi does not provide QoS guarantees, a resource control technique is required to enforce airtime fraction for competing virtual networks, or slices (see section 3 of D3.2 [14] where an example of such technique is provided). The Wi-Fi APs are then utilized to offload traffic from 5GNR cells, especially when there are services such as URLLC that can only be served through 5GNR.

3.6.1.2 Proposed method

The allocation of the optimal amount of resources to 5G-CLARITY slices in an industrial network is essential to ensure effective network performance and efficient use of radio resources. The proposed ML algorithm addresses the radio resource allocation problem at high-time scales (i.e., non-real time, specifically in the order of minutes) as part of the network planning activities. At this level, the allocated resources are defined in terms of quotas and the main network dynamics are due to user behaviour and mobility patterns, which result in spatio-temporal variations of the traffic demand. To cope with this traffic dynamics, the ML algorithm redistributes the available radio resources by adapting the radio resource quotas and leveraging the multi-WAT RAN feature to offload traffic depending on the current network state. For that purpose, the algorithm will follow a closed loop operation process, which comprises the monitoring, measurement and evaluation of the network traffic and then the optimization of the radio resource provisioning.

From the design viewpoint, the closed loop automated slice provisioning can be seen as a controller (agent) that distributes the resources as a function of the current traffic demands and targeted KPIs. The proposed solution is based on a multi-agent distributed approach, where one agent is deployed per 5GNR cell and per slice. Specifically, the controller's design depends on the characteristics of the 5G-CLARITY slice that is deployed in the industrial scenario: eMBB or URLLC.

Table 3-16. Input Parameters for Both Types of Controllers (URLLC, eMBB).

Input parameter	Controller
Cell resource utilization (L_b)	URLLC, eMBB
Resource quota ($\xi_{c,b}$)	URLLC, eMBB
Packet Loss Ratio ($PLR_{c,b}$)	URLLC
Normalized user throughput ($\mu'_{c,b} = \mu_{c,b} / D_{eMBB}$)	eMBB

Table 3-16 shows the controller's inputs for each kind of controller (URLLC or eMBB). The cell resource utilization L_b is defined as the relation between the used PRBs and the total number of PRBs in a cell. The Packet Loss Ratio (PLR) is the ratio of the number of lost packets and the number of arriving packets considering that a packet is discarded if the packet is not delivered in a time less than τ_{max}. Let PLR_{max} be the maximum *PLR* that can be allowed for the URLLC service.

The controller's output is given by the increase or decrease in the allocated number of PRBs. Let Δ_q be the magnitude of the change in the slice resource quota.

Regarding the operation, each controller adapts the wireless resource quota of every slice periodically. Let T_{alloc} be the time interval over which the calculated quota is valid for resource allocation. The operation of the controllers is limited by the following considerations:

- On the one hand, as stated in the system model, the controller for URLLC can only be applied to 5GNR cells.
- On the other hand, the operation of the controller for eMBB in 5GNR cells is influenced by the UEs that are also connected to Wi-Fi cells (using the AT3S function) and vice versa.

The resource quota calculated by the controller should satisfy D_{eMBB} for eMBB traffic or PLR_{max} for URLLC traffic.

There can be some cases where different controllers in the same cell can take the action of increasing the quota to the slices so that the total amount of required resources exceeds the system bandwidth. In order to solve this conflict, a *super agent* that knows the status of the cell will take the responsibility to determine which controllers are allowed to modify the quota.

Each slice controller in every 5G cell stands for a DQN agent. Each agent derives its policy according to a DQN based on RL theory. The DQN is able to combine RL with a class of artificial neural network known as DNN. DQN algorithm follows Bellman´s equation, as indicated below:

$$\mathrm{Q(s,a)} = \mathrm{E}\left[r + \gamma \max_{\mathrm{a'}} \mathrm{Q(s', a')|s, a}\right]$$

Specifically, in DQNs, a neural network is used to approximate the *Q*-value function.

At every time step of the operation process, the agent observes a state *s* from the environment and takes an action *a*. After taking the action, the agent will receive a reward *r* whose value will depend on whether the action is appropriate or not for the observed state. The agent learns to maximize the long-term reward as follows:

$$R_t = \sum_{t=0}^{T} \gamma^t r_t$$

where γ represents a constant that discounts future rewards ($0 \leq \gamma \leq 1$) and *T* is the total time steps of the training process.

Table 3-17. Reward function definition.

Reward (r_t)		Controller
r_t =	1 if distance to the required target throughput is getting lower -1 otherwise	URLLC and eMBB

RL (and also DRL) learns what is the best action for a state through exploration and exploitation. That is, when the agent explores, it can improve its current knowledge as it gets more information about the environment. This results in better rewards in the long run. However, in the exploitation phase, the agent chooses an action based on its previous knowledge of the environment, resulting in a higher immediate reward. In order to have sufficient initial exploration and higher reward values a balance or trade-off between exploration and exploitation would be desired.

In the following, the state, action and reward of the proposed approach are detailed:

a) State

At each time step *t*, the state s_t is obtained from the environment. It can be expressed as $s_t = \{s_t(c,b)\}$, where each element $s_t(c,b)$ represents the state of the slice *c* in cell *b*. The state is defined as the inputs of the controllers. In particular, for an eMMB agent, the state is defined as $s_t(c,b) = \{L_b, \xi_{c,b}, \mu'_{c,b}\}$ and for a URLLC agent, the state is defined as $s_t(c,b) = \{L_b, \xi_{c,b}, PLR_{c,b}\}$.

b) Action

Once the agent has observed the state, it triggers an action in order to adapt the allocated radio resources to the new environment conditions. The action taken at time step *t* for the slice *c* in cell *b* is denoted as $a_t(c,b)$ and it will update the associated resource quota $\xi_{c,b}$ by increasing, decreasing or maintaining the value determined in the previous step *t-1*. The action will modify the quota progressively in steps of a given size Δ. Thus, three different values are possible to be taken by the action, $a_t(s,b) \in \{\Delta_q , -\Delta_q , 0\}$, resulting in the modification of the slice quota, as indicated in the following expression: $\xi_{c,b}(t) = \xi_{c,b}(t-1) + a_t(c,b)$.

c) Reward

It is necessary to evaluate the goodness of the action a_t that is taken by the agent in step *t* from state s_t. A possible reward function r_t that captures the benefit/loss of the taken action is defined in Table 3-17.

3.6.2 Evaluation methodology

3.6.2.1 Description of the scenario

For the performance evaluation of our proposal, we consider an industrial scenario (e.g., a factory in the context of the Industry 4.0), which attempts to represent the BOSCH factory (UC2.1 [53]).

The physical dimensions of the industrial site service area considered are 100 m x 100 m. A set of wireless URLLC and eMBB users (UEs) are deployed in the scenario.

The UEs considered as URLLC stand for Programmable Logic Controllers (PLCs) in charge of monitor and control the industrial processes. We contemplate a total of 224 URLLC UEs distributed in 4 production lines. On the other hand, several eMBBs UEs are also located in the scenario. These eMBB users are randomly located along the factory premises (e.g., they represent factory workers with smart phones or tablets).

Moreover, 4 gNBs (transmitting at 3.5 GHz) and 5 Wi-Fi APs (transmitting at 2.4 GHz) are considered to be deployed in the scenario. Figure 3.45 shows the described industrial scenario layout considered.

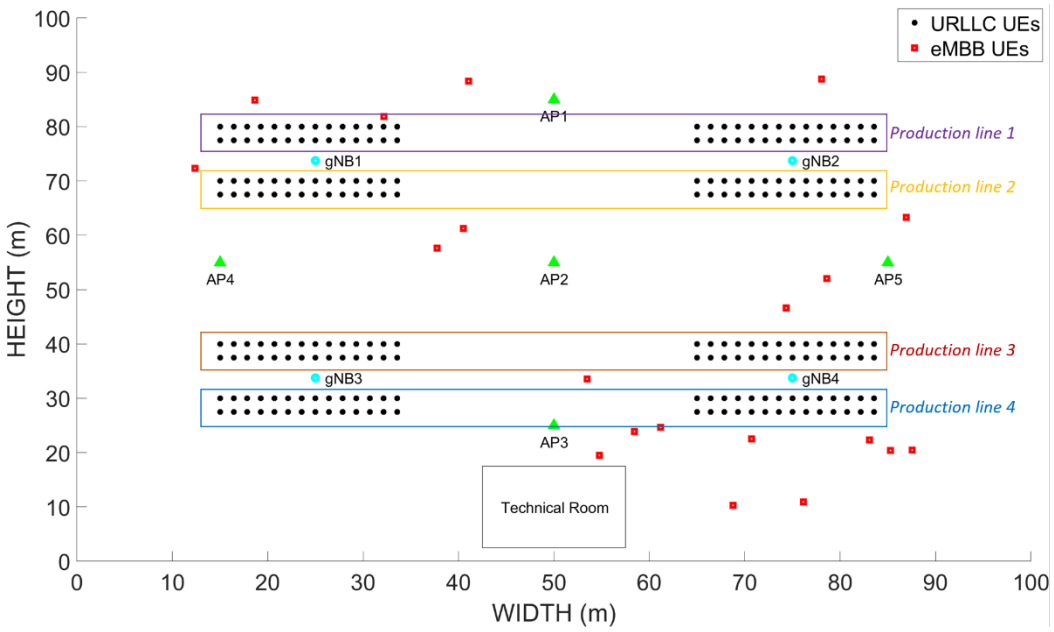


Figure 3.45. Industrial scenario layout.

3.6.2.2 Evaluation tools

The ML model proposed in this section can be seen represented in the framework of 5G-CLARITY in Figure 3.46. As illustrated in the figure, the solution based on multiple DQN agents resides in the AI Engine, where the ML training host and the ML inference host are the two components required for the operation of the ML model.

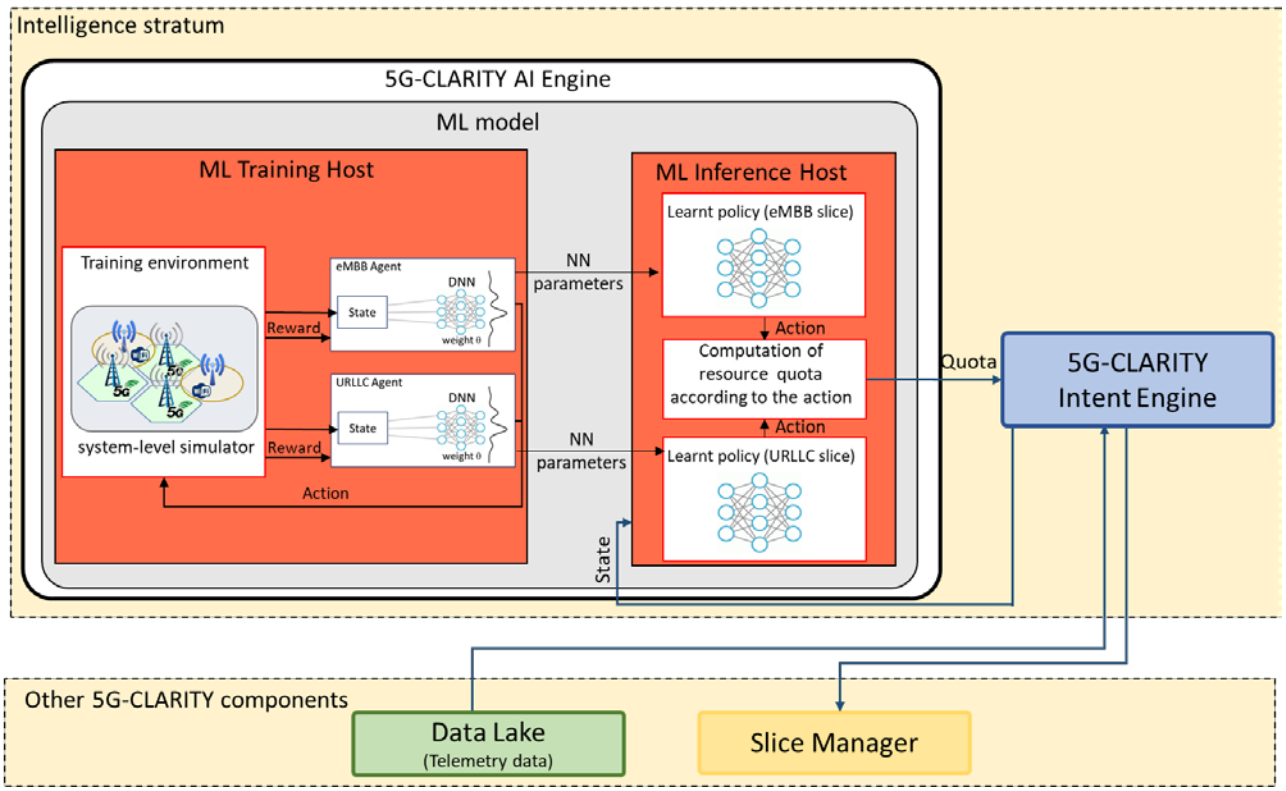


Figure 3.46. ML model functionality in 5G-CLARITY architecture.

As illustrated in the figure, the solution based on multiple DQN agents resides in the AI Engine, where the ML training host and the ML inference host are the two components required for the operation of the ML model. Particularly, in the ML training host, the training of this model takes place making use of the telemetry data that is available in the AI Engine. During the training process, the ML training host learns the DNN parameters (e.g., weights) in order to obtain a good Q-value function approximation that determine the action selection policies.
For the evaluation of the performance of our ML algorithm proposal, a RAN simulator developed in MATLAB and analytical performance models will be used as the training environment.

The simulator follows a snapshot-based model to capture the different realizations of the demand distribution in a scenario that resembles a typical industrial floor. More precisely, each snapshot represents a random realization of the demand distribution. The different realizations of the traffic demands and positions of the mobile eMBB UEs ensure reliable statistical significance analysis.

On the one hand, the computation of radio features related with 5G technology in the simulator, such as users´ SINR, spectral efficiency and so on, is based on the approach followed in the work in [55]. On the other hand, the estimation of Wi-Fi-related radio features such as the interferences among APs and the users´ reachable throughput is based on the works in [57] and [56], respectively.

One of the key issues of DRL can be its long training time. In order to ameliorate it we adopt analytical performance models to estimate the radio interface performance metrics for URLLC services in an agile an accurate way. Particularly, we use a Queueing Theory (QT)-based model to ensure that a given resource quota assignment for an URLLC slice meets the service requirements [54]. Given a radio resource quota allocation expressed in terms of number of PRBs for an URLLC slice, the analytical performance evaluation comprises the following steps:

1. A single-server queue with finite buffer models that packets are served on time. The length of the buffer L_{max} is calculated as follows:

$$L_{max} = W \cdot \left\lfloor \frac{\tau_{max}}{\tau} - 1 \right\rfloor$$

   where W represents the number of PRBs, τ_{max} stands for the radio interface maximum delay constraint, i.e., the maximum time spent for packet delivery, and τ represents the transmission time slot duration.

2. The required W to satisfy a given PLR_{max}, i.e., $PLR < PLR_{max}$, is then calculated.

Once the training has reached a considerable number of training steps, the ML inference host will be the responsible for providing the variation of the resource quota assigned to each slice, using the policy learnt in the training process. In order to determine the action for the slice *c* in cell *b* $a_t(c,b)$ based on the policy learnt, the state $s_t(c,b)$ is observed for this slice in the corresponding cell at the time *t*. In the inference mode, and putting the ML model in the context of the 5G-CLARITY architecture, the parameters that compose the state of the environment are metrics taken from the 5G-CLARITY data management framework (ML model input, see Section 7.2 of D2.2 [2]). Then, the ML model derives the outputs according to the ML logic. In this case, the ML model output will be the resource quota assigned to a given slice c in a determine cell b $\xi_{c,b}$ modified accordingly to the action taken. The ML model operation in a 5G-CLARITY system requires the request of telemetry data in order to use it as input for the model and the communication of the slice resource quotas configuration to the Slice Manager, which will be carried out through the Intent Engine (see Section 8.2 of 5G-CLARITY D2.2 [2]).

3.6.2.3 Description of initial experiments

The training process in DRL comprises an agent that interacts with the environment. To accelerate the training process and facilitate the required data collection, the intended ML-model will be trained making use of the RAN simulator and performance analytical models, which were introduced in the previous subsection.

First, in order to obtain preliminary results, several experiments that do not depend on the agent execution will be addressed. To that end, some metrics that define a specific state will be considered to evaluate the obtained reward. The purpose of these experiments is to get familiarized with some parameters related with the neural network configuration, in order to see how they influence on the learning and convergence process of the agent. Thus, these results will show the curve of the agent convergence process when varying the parameters under study (e.g., discount factor, learning rate).

Then, some experiments will be performed to analyse the impact of some RL parameters such as the discount rate that determines the relative importance of future rewards and the learning rate. In addition, different behaviours of the algorithm with a particular trade-off between exploration and exploitation will be evaluated. Some variants of the reward function could also be evaluated in order to select the most appropriate, as well as the possibility to include other factors in the formula to consider high-level operator's objectives (e.g., energy savings).

3.6.3 Evaluation results

3.6.3.1 Setup

The design of the DQN agent is based on a critic representation. This critic takes the observations and the actions as input and returns the corresponding expectation of the long-term reward. Figure 3.47 depicts the architecture of the neural network that conforms the critic. In the figure, we can see that the critic network is composed of multiple paths. Specifically, one of the paths is created for the observations and the other for the actions. Then, these paths are combined with a combination layer, whose output is fed to another neural network. The settings of some of the parameters related with the configuration of the neural network and the DQN agent hyperparameters are shown in Table 3-18.

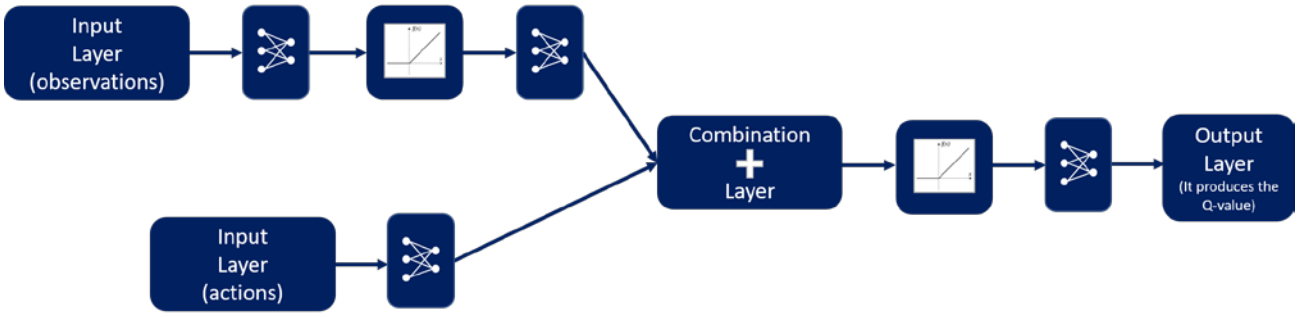


Figure 3.47. Architecture of the critic network.

Table 3-18. Configuration of DQN Agent Hyperparameters.

DQN Agent Hyperparameters	Configuration
Reinforcement learning method	DQN with critic network (value based)
Learning rate	0.001
Gradient Threshold	1
Mini-batch size	32

Discount factor	0.9
Experience buffer length	2000
Target update frequency	4
Target update method	Periodic
ε-greedy exploration	
Epsilon	0.9
EpsilonMin	0.01
EpsilonDecay	0.005

3.6.3.2 Preliminary results

Firstly, we include some results related to the agents training process. Figure 3.48 summarizes the training process of the eMBB slice agent obtained with the RL Matlab framework for a specific scenario. More precisely, it includes the instantaneous and average rewards, and the episode Q0. The average reward is computed over 3 samples of the instantaneous reward. The Episode Q0 indicates the estimate of the discounted long-term reward at the start of each episode, given the initial observation of the environment. As training progresses, if the critic is well designed, Episode Q0 approaches the true discounted long-term reward as the training progresses (see Figure 3.48).

As observed, the agent needs around 300 episodes to get the convergence and learn the optimal amount of PRBs that has to allocate to the eMBB slice in order to meet the aggregated mean throughput in a specific scenario realization.

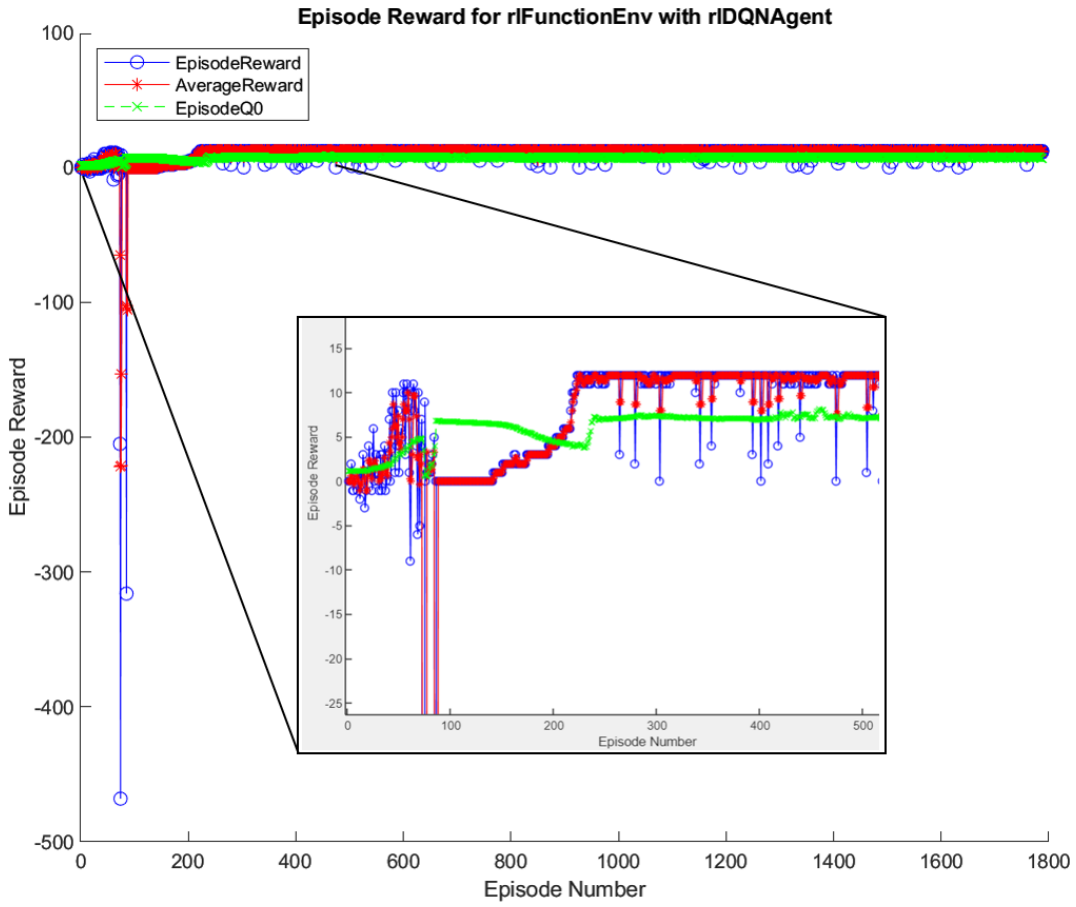


Figure 3.48. eMBB slice agent learning process for a concrete scenario using DQN agent.

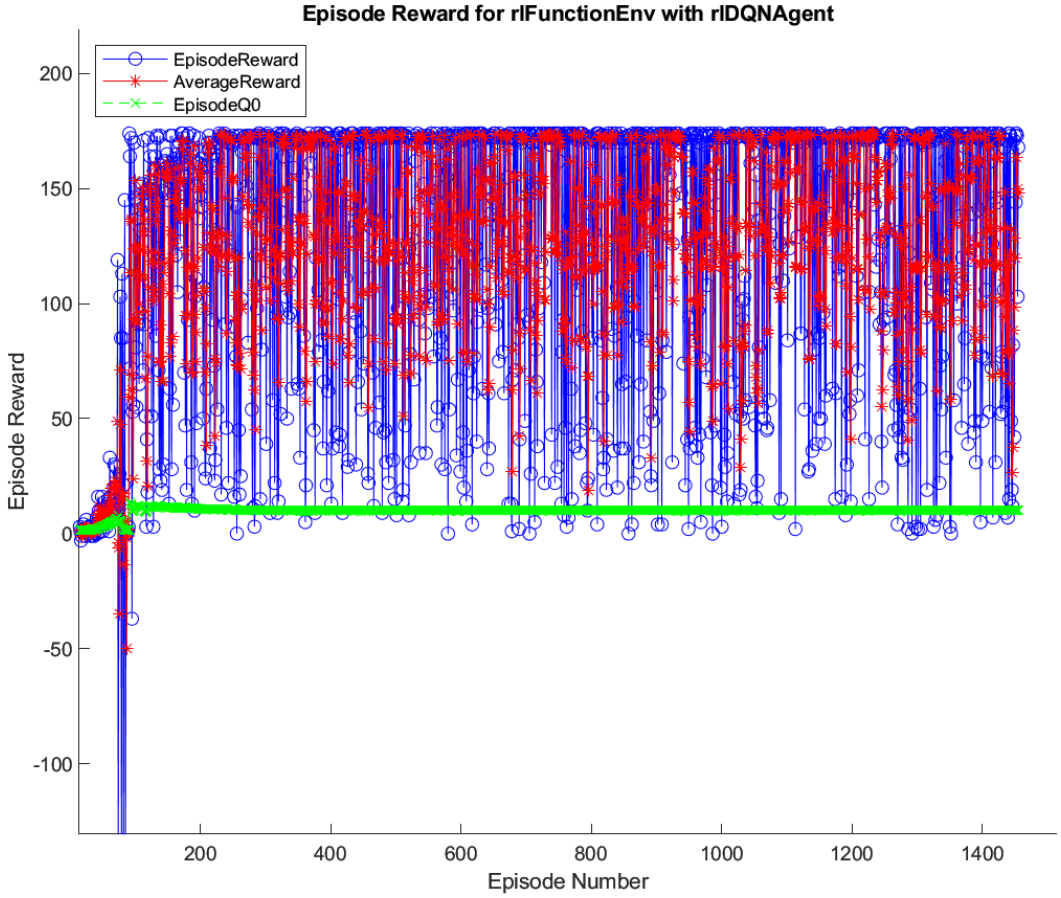


Figure 3.49. URLLC slice agent learning process for a concrete scenario using DQN agent.

Similarly, Figure 3.49 depicts the training process of the URLLC slice agent obtained with the same framework.

Preliminary results suggest that the agent requires approximately around 200 episodes to learn the optimal amount of PRBs that has to allocate to the URLLC slice in order to meet a specified packet loss ratio and a mean aggregated throughput for a specific scenario realization.

On the other hand, several experiments have been conducted for the sake of analyzing how the configuration of the hyperparameters impact on the convergence of the training process. Specifically, these experiments have been conducted using Python tool. Figure 3.50 shows the convergence of the training process of the eMBB agent for several values of the discount factor. As we can observe in the figure, the value of the discount factor implies a variation of the number of training steps to get the convergence of the agent. The curves in the figure demonstrates that the larger the value of this parameter, the smaller the number of steps needed to get the convergence in the agent training process.

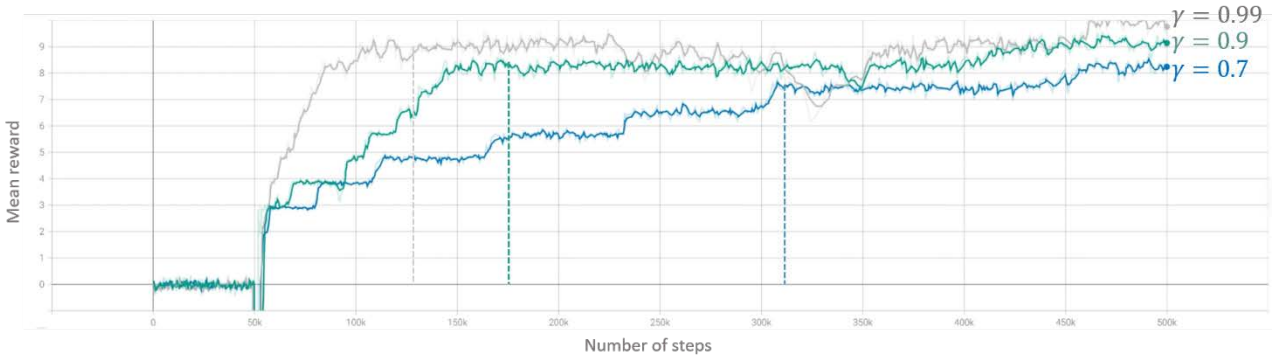


Figure 3.50. Training process for different values of the discount factor parameter.

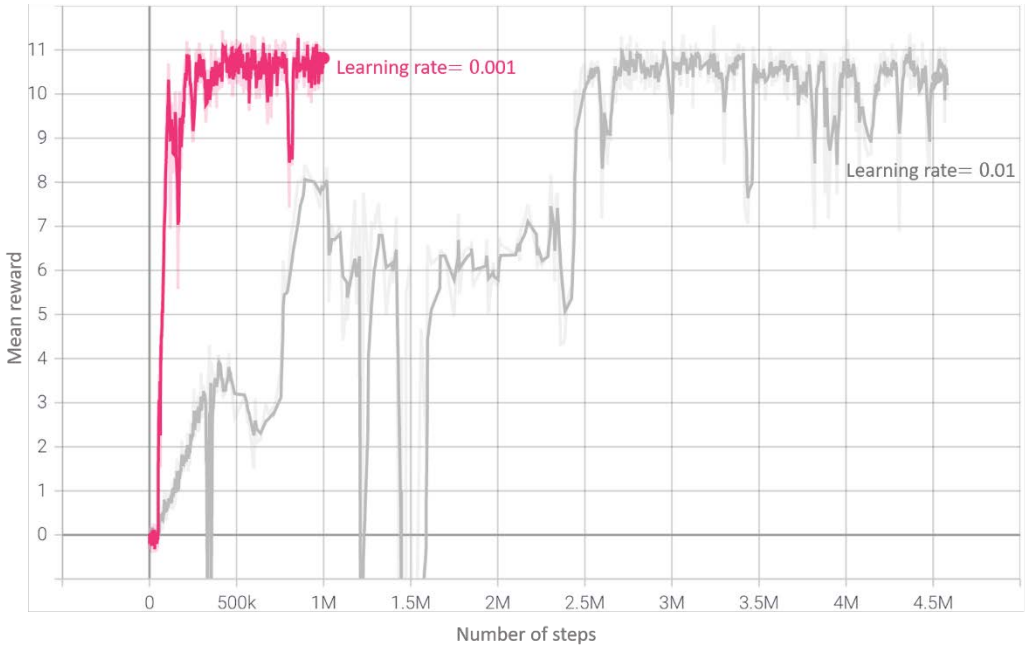


Figure 3.51. Training process for different values of the learning rate parameter.

This means that if the agent is aware of the rewards in the distant future that are relative to the ones obtained in the immediate future, the convergence will be reached in a shorter training period. The learning rate is other hyperparameter whose configuration is crucial for the training process. The learning rate measures how fast the model is able to get adapted to the problem. Small learning rates require more training epochs due to the small changes made to the neural network weights in each update. In contrast, large learning rate values result in rapid changes and require fewer training epochs, but could conduct the model to converge too quickly to a suboptimal solution. In line with Figure 3.50, Figure 3.51 depicts the curves of the convergence of the eMBB agent training process. As observed in the figure and as previously discussed, the training process duration is also dependent of this parameter. We can see that when the learning rate is higher, a larger number of training steps are needed to get the optimal solution of the mean reward.

Figure 3.51 compares the training process between starting using already trained model (line in orange) or not (line in blue). As observed, using an already trained model speeds up the training process for new unknown scenarios. This saving time becomes more important when the variability of the temporal traffic load increases as it is expected the agent must be trained more often to adapt faster to the changing traffic conditions. Finally, proper operation of the agent is validated using a single setup.

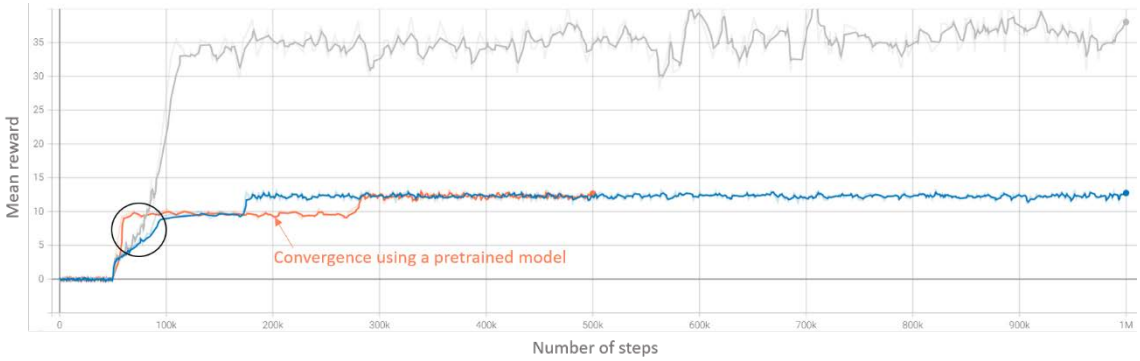


Figure 3.52. Agent convergence with and without pretrained model.

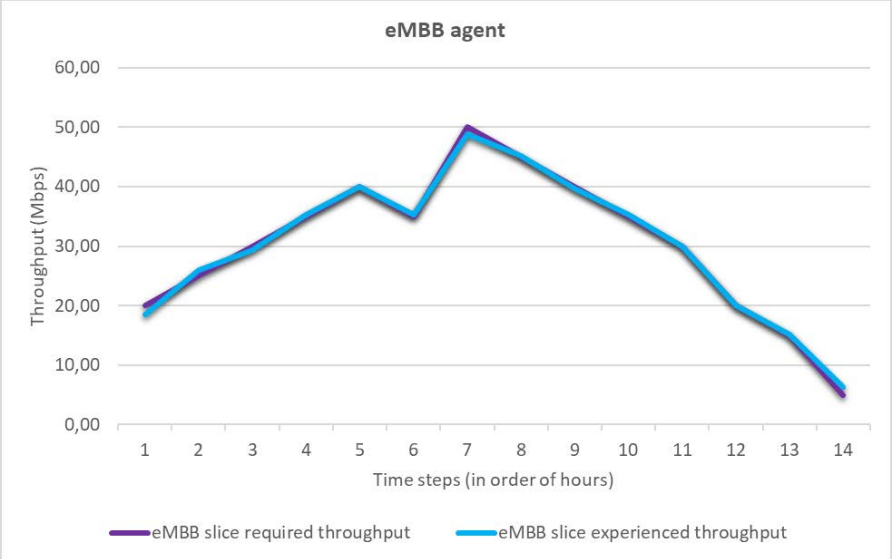


Figure 3.53. eMBB agent operation.

On the one hand, we have trained the eMBB agent on a simplified version of the simulator based on Shannon´s capacity. Additionally, for the time being we are considering only the users mean SINR as an input parameter for the agent in order to make the decision of the amount of resources to be allocated. Figure 3.53 demonstrates the effectiveness of the eMBB agent operation for the dynamic resource provisioning. We can see that the agent learns to allocate to the eMBB slice the radio resources that are necessary to meet the slice traffic demand, represented with the line in purple.

On the other hand, the URLLC agent performance can be found in Figure 3.54. The URLLC agent tries to satisfy the radio resources needs of services that demand stringent latency requisites. Particularly, in our scenario (depicted in Figure 3.45) the URLLC agent is responsible for allocating radio resources to the URLLC slice composed of the control devices deployed along the production lines.

To that end, the URLLC agent performance is based on an analytical model that provides the amount of radio resources (in terms of number of PRBs) required to reach a certain capacity to deal with the traffic demands ensuring a delay requirement and guaranteeing a packet loss ratio of 1 ms and 10^{-4}, respectively (refer to Section 3.2 of 5G-CLARITY D2.3 [58] for further model details).

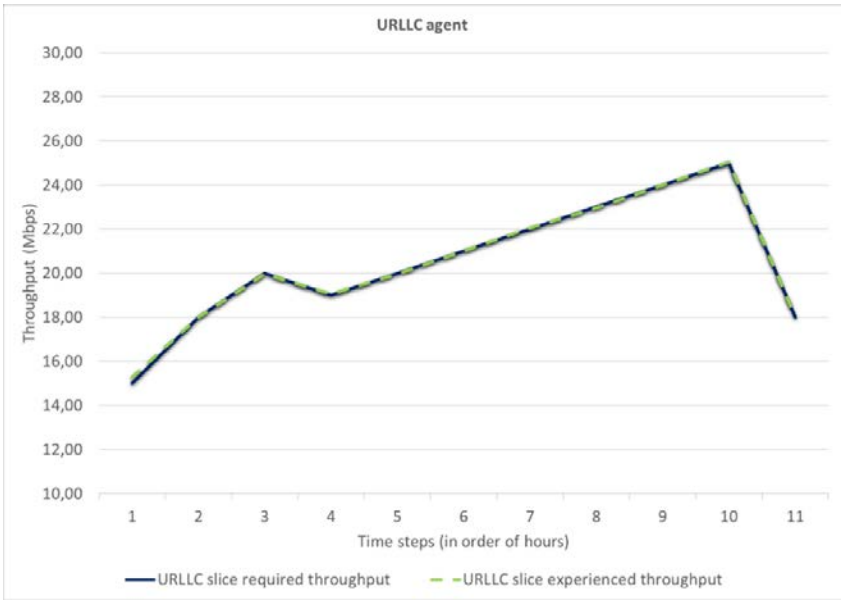


Figure 3.54. URLLC agent operation.

An initial description of the agent operation is as follows: every time there is a variation on the traffic demand, the agent adapts the amount of resources assigned to the URLLC slice it manages. This amount of resources is given by the analytical model above mentioned. This analytical model is computationally complex and takes large amount of computational time to run the simulations, what will lead to long-lasting trainings. For that reason, we have run some simulations of the model for several values of the URLLC slice bandwidth (represented by *W*) for the sake of obtaining a representative data set that makes us able to interpolate the rest of values inside the contemplated range of *W*.

In a similar way to the eMBB agent, in Figure 3.54 we show the performance of the URLLC agent. As observed, the agent learns to allocate the amount of PRBs that are necessary to adapt to the traffic variations.

3.7 Dynamic transport network setup and computing resources provisioning

In 5G-CLARITY D4.1 an initial high-level design of a deep reinforcement learning (DRL)-based multi-agent solution was proposed to address the transport network setup and computing resources provisioning in a coordinated way. That initial design comprises four DRL agents: the agent in charge of distributing the delay budget among the different network domains, another agent to optimize the setup and allocate the transport resources quotas for each slice, and a couple of agents responsible for the computing resources provisioning and the virtual network functions (VNFs) embedding. In this deliverable, we focus on the DRL solution for configuring and allocating resources in an asynchronous TSN-based transport network. Unlike in D4.1, the goal here is to delve deeper into the implementation details, develop the proposal and provide some preliminary results showing its performance.

3.7.1 Initial implementation

3.7.1.1 System Model

Figure 3.55 illustrates the system model considered. We assume the transport network is an asynchronous Time-Sensitive Networking (TSN) network, i.e., the constituent forwarding plane elements (e.g., TSN bridges) do not need a common and precise time reference to be synchronized. In these networks, each TSN bridge's egress port includes an Asynchronous Traffic Shaper (ATS), whose queuing stages are depicted in Figure 3.55. The first queuing stage corresponds to the interleaved shaping to regulate the streams and consists of S shaped queues. The second stage comprises P first-come first-served queues arbitrated by a strict priority transmission selection scheme, i.e., the traffic at a given queue has to wait for transmission as long as there are packets at any buffer with higher priority. The number of shaped queues S might limit the implementable number of priority levels due to the queue allocation rules (QARs) of the interleaved shaping: a given shaped queue is assigned to an input port (QAR 1), an internal priority level (QAR 2), and a priority level in the previous hop (QAR 3). We assume there are a set of 5G-CLARITY slices sharing TSN-based TN. Each 5G-CLARITY slice is mapped onto a VLAN (please refer to D4.1) and a traffic class (TC) that are encoded in the VLAN ID and the Priority Code Point fields of the IEEE 802.1Q header, respectively. There might be multiple 5G-CLARITY slices with the same assigned TC. Two 5G-CLARITY slices might have different priority levels at a given ATS even though they belong to the same TC. To that end, the Internal Priority Value (IPV) at the corresponding ATS can be used [59]. We assume enough shaped buffers at each ATS allow for eight implementable priority levels as considered in TSN standards by default. We consider the aggregated data rate and aggregated burstiness demanded by each slice is known in advance and used as input for optimizing the transport network configuration. Last, we suppose that the paths of the transport network are predefined.

Although TSN standards allow for a finer granularity of the configuration at each ATS, our goal here is to find a configuration shared among all the flows of the same 5G-CLARITY slice.

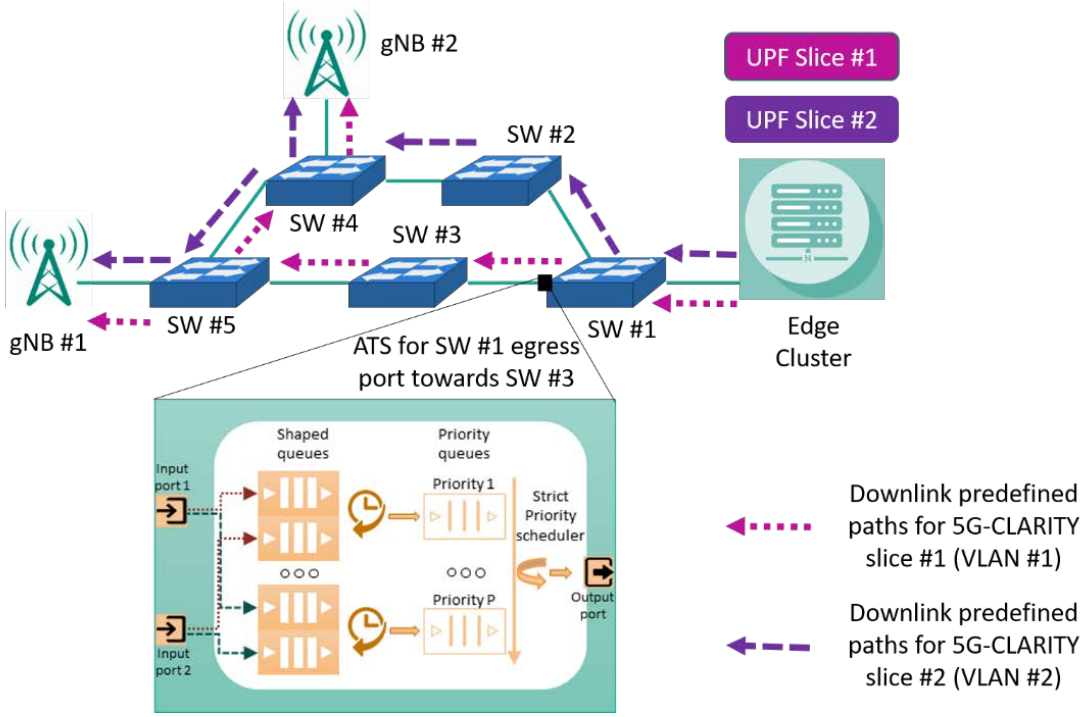


Figure 3.55. Asynchronous TSN backhaul network interconnecting two gNBs with the edge cluster. There are two 5G-CLARITY slices whose respective virtualized UPFs are hosted on the edge cluster. There is a segregated VLAN per slice in the backhaul network.

Under this consideration the primary parameters to be configured at each ATS are the following:

- Priority level assigned to the 5G-CLARITY slice.
- Shaped buffers assigned to the 5G-CLARITY slice following the aforementioned QARs, though, for simplicity, this decision is not addressed here.

The worst-case delay and jitter experienced by any flow of a given 5G-CLARITY slice $\tau \in T$ depend on its priority level p_τ. Specifically, the worst-case delay D_τ and jitter J_τ of the TC is given by:

$$D_\tau = \frac{\sum_{k=1}^{p_\tau} \hat{b}_k + \hat{l}^L_{p_\tau}}{C - \sum_{k=1}^{p_\tau - 1} \hat{r}_k} + \frac{l_\tau}{C}$$

$$J_\tau = \frac{\sum_{k=1}^{p_\tau} \hat{b}_k + \hat{l}^L_{p_\tau}}{C - \sum_{k=1}^{p_\tau - 1} \hat{r}_k}$$

Where $\hat{r}_k$ and $\hat{b}_k$ are the aggregated committed rate and aggregated burstiness or burst size at the priority level k, respectively. $\hat{l}^L_{p_\tau}$ is the maximum frame size or Maximum Transmission Unit (MTU) allowed in the priority levels lower than the priority level assigned to the 5G-CLARITY slice. C denotes the ATS link capacity. And l_τ stands for the maximum packet size generated by the 5G-CLARITY slice.

3.7.1.2 Problem Statement

Under the system model considered above, the problem covered here consists of finding a feasible configuration of the asynchronous TSN transport network subject to the following constraints:

- The E2E transport network jitter/delay budgets of every 5G-CLARITY slice is met.
- The aggregated capacity allocated to a given ATS must not exceed the physical capacity of the respective link.

The problem aims to find a satisfiable 5G-CLARITY slice-to-priority level assignment at each ATS of the

network.

3.7.1.3 Initial Agent Design

To solve the problem stated above, we opt for developing a deep reinforcement learning (DRL)-based agent to find a feasible 5G-CLARITY slices-to-priority levels assignment for an ATS/link. We use this agent to find the configuration of every ATS in the transport network. In order to provide coherence among the configurations of the different ATSs, e.g., per-hop delay/jitter budgets distribution for every path and 5G-CLARITY slice and estimation of the per slice traffic demand at each ATS, we rely on a master algorithm in charge of configuring the inputs of the DRL agents based on some criteria. At each ATS, there is a set of 5G-CLARITY slices T, each with a traffic demand characterized by an aggregated sustainable rate, an aggregated burst size, and a maximum transfer unit. The DRL agent is in charge of determining the priorities for each 5G-CLARITY slice at each ATS.

To model the priority assignment problem for the asynchronous traffic shaping using the RL framework, the following components or their operation (e.g., agent) need to be formally defined for the specific problem: *i)* agent*, ii)* environment, *iii)* observations or state, *iv)* the actions and v) the reward.

- **Agent**: The agent's goal is to find a satisfiable 5G-CLARITY slice-to-priority assignment while fulfilling the constraints listed in Subsection 3.7.1.2. The agent's operation is as follows. At the beginning all the 5G-CLARITY slices are assigned to the priority level one (the highest priority level). At every episode step, the agent chooses the 5G-CLARITY slice for which decreasing its priority level. The episode will finish either when a maximum number of steps are run or when the agent finds a valid configuration for all the 5G-CLARITY slices, i.e., a configuration that meets all the problem constraints. The maximum number of steps per episode is set to the number of 5G-CLARITY slices times the number of available priority levels.
- **Environment**: ATS with eight priority levels. There is a set of 5G-CLARITY slices, each with a traffic demand characterized by an aggregated sustainable rate, an aggregated burst size, and a maximum transfer unit at the ATS. Each 5G-CLARITY slice has also delay and jitter budgets to be met at the ATS. The agent is in charge of mapping the 5G-CLARITY slices onto the priorities so that the traffic demand of the slice, and the hop's delay and jitter budgets are met.
- **Actions**: Let $A = \{\downarrow_1, \dots, \downarrow_\tau, \dots, \downarrow_{TC}\}$ denote the set of agent's actions, where $\downarrow_\tau$ stands for the agent decreases the priority level of the 5G-CLARITY slice $\tau \in [1, TC]$ at the ATS.
- **Observations**: The agent's observations are:
  - The aggregated data rate $\hat{r}_\tau$ and burstiness $\hat{b}_\tau$ to be allocated for each 5G-CLARITY slice $\in [1, TC]$.
  - The maximum packet size of each 5G-CLARITY slice.
  - The E2E transport network delay/jitter budget $\Psi_\tau = \min\{D_\tau^{qos}, J_\tau^{qos}\}$ for each 5G-CLARITY slice $\in [1, TC]$, where D_τ^{qos} and J_τ^{qos} are the E2E transport network delay and jitter budgets, respectively.
  - The number of implementable priority levels P at the ATS.
  - The physical link capacity C handled by the ATS.
- **Reward**: For every action taken by the agent, it is compensated or penalized according to the following reward function:

$R \leftarrow 0;$
$N \leftarrow 0;$
For each 5G-CLARITY slice $\tau \in [1, TC]$
    $p_\tau \leftarrow$ get priority currently assigned to slice τ
    If $p_\tau \geq 8$

```
                    If the delay/jitter constraints for the slice are met
                        If the delay/jitter experienced by the slice has changed
                            R ← R + 1;
                            N ← N + 1;
                        Endif
                    Else
                        If the delay/jitter experienced by the slice has changed
                            R ← R − 1;
                            N ← N + 1;
                        Endif
                    Endif
                    Else
                    R ← −10;
            Endif
    Endfor
    R <- R/N;
```

The reward defined above encourages the agent to decrease the priority level of the 5G-CLARITY slices with lenient delay/jitter constraints, thus decreasing the delay experienced by the 5G-CLARITY slices with the most stringent delay requirements. More precisely, given an action, the 5G-CLARITY slice that has changed its priority level contributes positively to the reward as long as its delay and jitter requirements are still fulfilled. The rest of the 5G-CLARITY slices, i.e., those whose priority levels are not affected by the action, will contribute positively to the reward if only if their experienced worst-case delay and jitter is decreased as a consequence of the action. The rationale behind this reward is that decreasing the priority level of a given 5G-CLARITY slice will increase or keep the same delay/jitter of the slice and reduce or keep the same delay of the rest of 5G-CLARITY slices. On the other hand, when the agent decreases the priority level of a 5G-CLARITY slice and their delay/jitter constraints are not fulfilled anymore as a consequence of the action, then, the agent is negatively rewarded. Last, observe the agent is penalized with –10 when it tries to assign an out-of-range priority level to a given 5G-CLARITY slice (e.g., priority 9 to any slice in an ATS with 8 priority levels as considered).

Regarding the RL agent operation phase (inference), the agent is triggered at high time scales, i.e., time frames from several minutes to hours. Specifically, the agent will be triggered when new 5G-CLARITY slices are created or destroyed or when the committed traffic demand of one of the ongoing 5G-CLARITY slices changes. Finally, it shall be noted that the use of ML for assisting the asynchronous TSN network configuration problem is justified by the computational complexity exhibited by analytical optimization methods, as shown in Figure 3.56 and Figure 3.57. Figure 3.56 shows the computational complexity for configuring an ATS-based transport network with four 5G-CLARITY slices as a function of the number of links. On the other hand, Figure 3.57 depicts the computational complexity for configuring an ATS-based transport network with eight links as a function of the number of 5G-CLARITY slices.

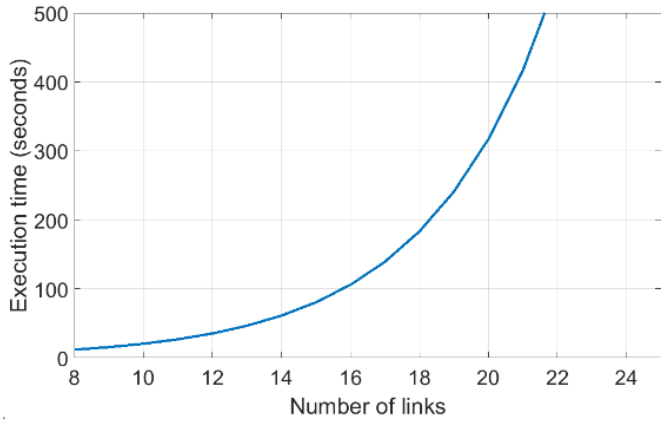


Figure 3.56. Computational complexity to find the optimal configuration (e.g., 5G-CLARITY slices prioritization and

per-link capacity reservation) versus the number of links in the ATS-based transport network.

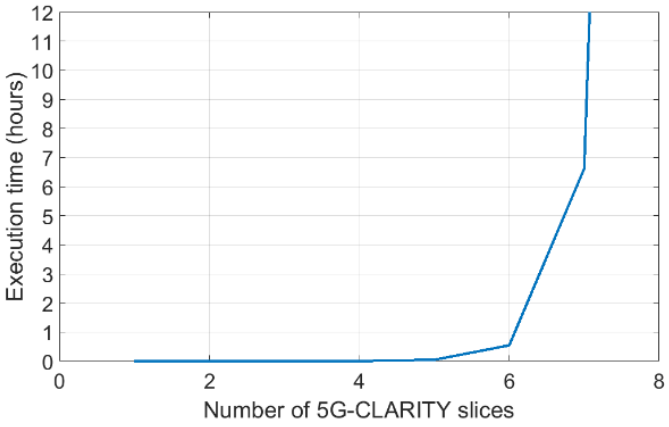


Figure 3.57. Computational complexity to find the optimal configuration (e.g., 5G-CLARITY slices prioritization and per-link capacity reservation) versus the number of 5G-CLARITY slices in the ATS-based transport network.

The ATS-based configuration problem was formulated as a convex mixed-integer nonlinear optimization program that was solved using Mosek solver [61]. As observed, using exact methods, finding the optimal configuration becomes an intractable problem for a network with more than 22 links and four 5G-CLARITY slices (see Figure 3.56) or a network with eight links conveying more than seven 5G-CLARITY slices (see Figure 3.57).

3.7.1.4 Practical Issues and Solutions

The proposed solution overcomes some of the DRL framework's practical issues as described below:

- Slow training process. DRL exhibits an inefficient learning process, then, speeding up the simulation of the environment is crucial. In this regard, our solution relies on the analytical performance models of the ATS which enable to estimate the performance of the asynchronous network with great agility. On the other hand, our solution intentionally sets the per hop delay/jitter budget in order to allow for the compositional analysis of the network, thus further reducing the time required to estimate the network performance.
- Lack of reliability. The actions issued by the agent are uncertain and might be unfeasible, i.e., the corresponding configuration might not fulfil all the problem constraints. To deal with this issue, again, we reckon on the analytical performance models of the asynchronous TSN networks to validate the agent's actions. If the actions are not feasible, they are not applied, and the agent is penalized with a negative reward and has to issue new actions for configuring the network. Thus, the QoS requisites of the transport network data plane are always met. Please refer to [59] and [60] for further details.

On the other hand, it is difficult or even impossible either to estimate by simulation or directly measure in a real network the worst-case jitter/delay experienced by a given stream. In this way, the use of analytical performance models overcomes this problem.

3.7.2 Evaluation methodology

The proposed RL agent is evaluated through simulation. The scenario considered is depicted in Figure 3.58. An asynchronous TSN network is used as the 5G-CLARITY system backhaul network. There are two physical machines or servers connected to the BN. Each server hosts four virtualized UPF instances, each belongs to a different 5G-CLARITY slice. Then, there are eight 5G-CLARITY slices whose priorities must be configured for every ATS in the network. We considered the physical network interfaces of the servers also include an ATS for handling the frames at layer 2. The traffic of each 5G-CLARITY slice is distributed equally among the two gNBs.

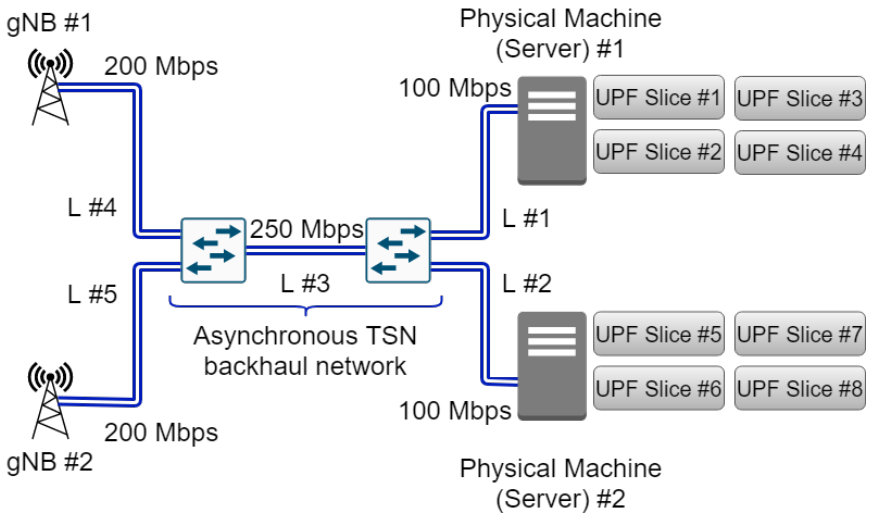


Figure 3.58. Scenario considered to test the DRL-assisted solution for configuring the ATS-based transport network.

The agent has been implemented in MATLAB using the RL toolbox. Specifically, we use a DQN agent with a critic network. The configuration of the main hyperparameters is included in Table 3-19. Figure 3.59 depicts the design used for the critic network.

Table 3-19. Primary hyperparameters configuration for the DRL agent used to configure the ATS-based transport network.

DQN agent hyperparameters	Configuration
Reinforcement learning method	DQN with critic network (value based)
Learning rate	0.0001
Gradient Threshold	1
Mini-batch size	32
Discount factor	0.9
Experience buffer length	10000
Target update frequency	4
Target update method	Periodic
ε-greedy exploration	
Epsilon	1
EpsilonMin	0.1
EpsilonDecay	0.0001

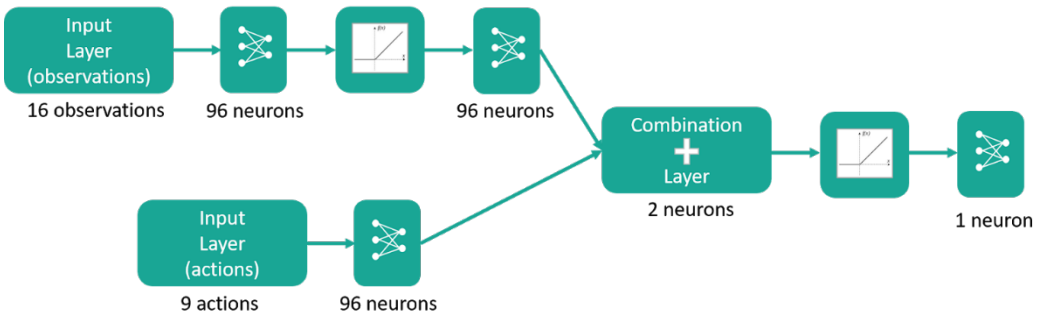


Figure 3.59. Critic network design used for the DQN agent used for the configuration of the ATS-based transport network.

3.7.3 Evaluation results

Figure 3.60 and Figure 3.61 illustrate the training process of the DQN agent in the RL framework of MATLAB for two specific scenarios, namely, the priority configuration of four and eight 5G-CLARITY slices, respectively. Specifically, the figure includes the instantaneous and average rewards, and the episode Q0. The average reward is computed over three samples of the instantaneous reward. The episode Q0 is "*the estimate of the discounted long-term reward at the start of each episode, given the initial observation of the environment. As training progresses, if the critic is well designed episode Q0 approaches the discounted long-term reward*". As observed, the agent needed around 2500 episodes to converge, i.e., for finding the way to prioritize the four 5G-CLARITY slices that maximizes the long-term reward while meeting the requirements of all the slices.

In contrast, the agent required around 5500 episodes to converge in the case of prioritizing eight 5G-CLARITY slices. The results of this experiment suggest is more complex to find the configuration as the number of 5G-CLARITY slices increases.

In fact, Figure 3.61 shows the DQN agent only found a valid 5G-CLARITY slices prioritization during the first 2500 episodes in the case of eight slices, whereas the agent finds valid prioritization for the slices from the very beginning in the case of four slices (see Figure 3.60). Also, although not illustrated in these figures, we observe that the complexity of finding a valid ATS configuration increases with the utilization of the link.

One of the key benefits of using RL to find the optimal configuration of the ATS-based transport network is the computational complexity reduction. Solving the problem of assigning priorities to each 5G-CLARITY slice given an optimization criterion by exhaustive search requires TC^P iterations, where TC is the number of slices to be prioritized and P is the number of priority levels in the ATS to be configured. For instance, for 8 priority levels and 8 traffic classes, roughly 17 million of iterations are required. We measured the execution time to compute the optimal priority assignment to eight slices considering eight priority levels in order to maximize the time slack of the slices. The exhaustive search approach required 2 hours and 48 minutes to find the optimal solution, whereas the DRL agent found the solution in approximately 0.7 seconds.

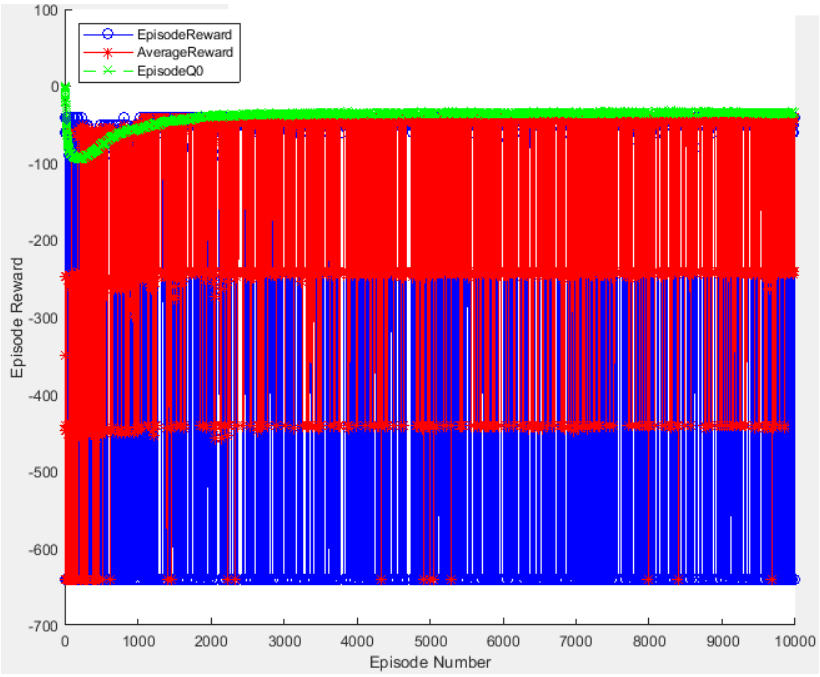


Figure 3.60. DQN agent learning process for finding prioritization of four 5G-CLARITY slices at a given ATS.

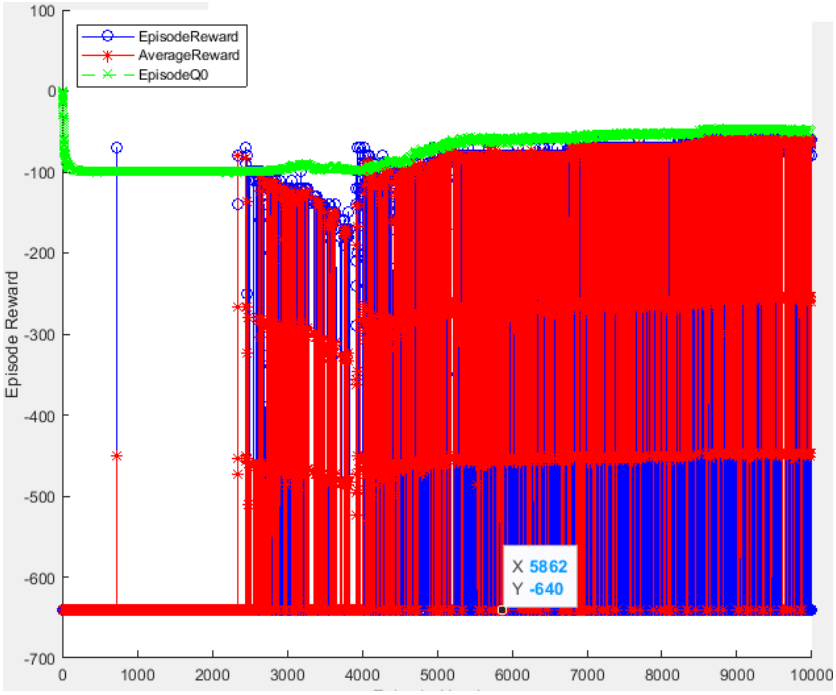


Figure 3.61. DQN agent learning process for finding the prioritization of eight 5G-CLARITY slices at a given ATS.

Last, we validated the proper operation of our solution by finding the prioritization of the eight 5G-CLARITY slices in Figure 3.58 for every ATS. The traffic demand for every slice together with the end-to-end delay/jitter budget is included in Table 3-20. Table 3-21 includes the utilizations of the different links in the network given our setup. Table 3-22, Table 3-23, Table 3-24, and Table 3-25 include the traffic prioritization of the different 5G-CLARITY slices along with their experienced worst-case delay for links #1, #2, #3, and #4 and #5, respectively. As shown in Figure 3.62, the end-to-end delay/jitter requirements for all the slices are met, thus validating the operation of the DRL-assisted solution described here.

Table 3-20. 5G-CLARITY slice traffic demand characteristics and delay/jitter onstraint

Slice	Agg. Rate	Agg. Burstiness	MTU	End-to-end delay/jitter budget
Slice #1	5 Mbps	12000 bytes	1500 bytes	1.25 ms
Slice #2	5 Mbps	12000 bytes	1500 bytes	1.5 ms
Slice #3	5 Mbps	12000 bytes	1500 bytes	1.75 ms
Slice #4	5 Mbps	12000 bytes	1500 bytes	2 ms
Slice #5	5 Mbps	12000 bytes	1500 bytes	1.375 ms
Slice #6	5 Mbps	12000 bytes	1500 bytes	1.5 ms
Slice #7	5 Mbps	12000 bytes	1500 bytes	1.875 ms
Slice #8	5 Mbps	12000 bytes	1500 bytes	2.125 ms

Table 3-21. Links utilizations in the scenario depicted in Figure 3.58

Link	Link #1	Link #2	Link #3	Link #4	Link #5
Utilization	20%	20%	16%	10%	10%

Table 3-22. 5G-CLARITY slices prioritization, delay budget and worst-case delay for link #1 (see Figure 3.62).

Slice	Priority	Delay Budget	WC Delay
Slice #1	1	500 µs	480 µs
Slice #2	1	500 µs	480 µs
Slice #3	2	700 µs	653.3 µs
Slice #4	2	700 µs	653.3 µs

Table 3-23. 5G-CLARITY slices prioritization, delay budget and worst-case delay for link #2 (see Figure 3.62).

Slice	Priority	Delay Budget	WC Delay
Slice #5	1	500 µs	480 µs
Slice #6	1	500 µs	480 µs
Slice #7	2	700 µs	653.3 µs
Slice #8	2	700 µs	653.3 µs

Table 3-24. 5G-CLARITY slices prioritization, delay budget and worst-case delay for link #3 (see Figure 3.62).

Slice	Priority	Delay Budget	WC Delay
Slice #1	1	250 µs	240 µs
Slice #2	2	500 µs	456.5 µs
Slice #3	1	350 µs	240 µs
Slice #4	2	600 µs	456.5 µs
Slice #5	1	375 µs	240 µs
Slice #6	2	500µs	456.5 µs
Slice #7	2	475µs	456.5 µs
Slice #8	3	725µs	494.5 µs

Table 3-25. 5G-CLARITY slices prioritization, delay budget and worst-case delay for links #4 and #5 (see Figure 3.62).

Slice	Priority	Delay budget	WC Delay
Slice #1	1	500 µs	480 µs
Slice #2	1	500 µs	480 µs
Slice #3	1	700 µs	480 µs
Slice #4	1	700 µs	480 µs
Slice #5	1	500 µs	480 µs
Slice #6	1	500 µs	480 µs
Slice #7	2	700 µs	578.9 µs
Slice #8	2	700 µs	578.9 µs

Figure 3.62. Worst-case delay experienced and E2E delay budgets for every 5G-CLARITY slice.

3.8 Adaptive AI-based defect-detection in a smart factory

In 5G-CLARITY D4.1 [1], the problem statement and initial high-level design of the adaptive AI-based defect-detection in a smart factory have been provided. According to the given problem statement, this use case considers a production line in a smart factory where defective pieces on the production line have been detected by an AI-based defect-detection algorithm and an automatic intervention in order to stop the line and take the defective pieces out of the line is triggered. As such an immediate intervention requires real-time processing and visualization of geometric features for manufactured parts. This can be done either at an edge device within the factory or at a remote worker location outside the factory. Therefore, an adaptive low-latency and energy efficient AI-powered defect detection solution is needed. In this deliverable, the initial implementation details and preliminary results on latency and energy efficiency for different implementation options are provided.

3.8.1 Initial implementation

For the initial implementation, a smart factory testbed is considered. As depicted in Figure 3.63, the smart factory testbed is divided into two parts as *(i)* factory level that consists of the robotic arm, conveyor belt, camera and a fog device that is located close to the robotic arm and the camera in order to them e.g., send comment to the robotic arm to remove the defective item, change the zoom or tilt of the camera; and *(ii)* edge side that consists of Field Programmable Gate Array (FPGA), Central Processing Unit (CPU) or Graphics Processing Unit (GPU) based platforms. The camera provides continuous video streaming to the edge server. The edge server performs an AI-based object-detection algorithm to detect defective pieces on the streamed video. If the piece on the conveyor belt is detected as a defective item, the edge server sends a control signal to intervene the production and remove the defective item from the conveyor belt.

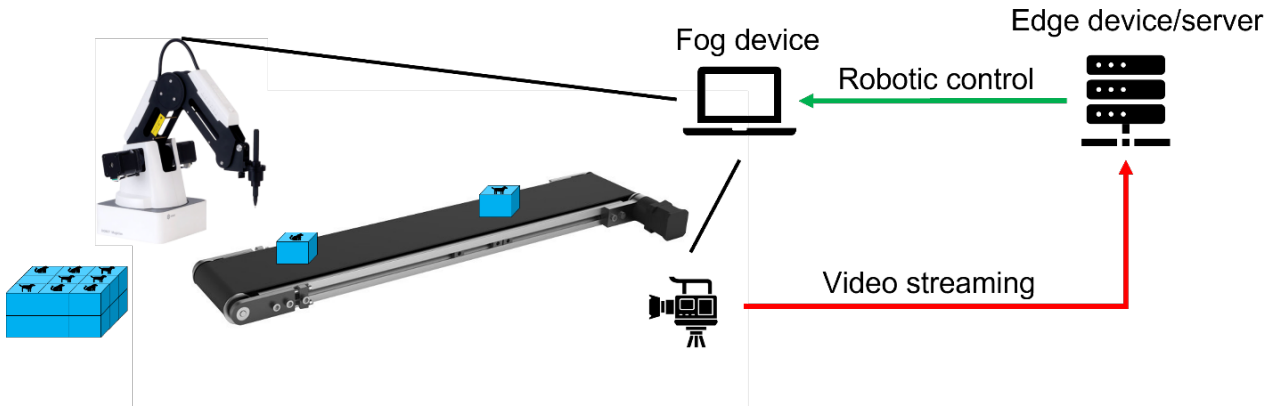


Figure 3.63. AI-based defect-detection implementation setup

Figure 3.64. Cubes with stickers on used to mimic production items on the conveyor belt

In the setup, cubes are used as the products on the production line. As shown in Figure 3.64, the cubes are used with several stickers on them to represent whether the item is defective or not. Accordingly, if a specific sticker or set of stickers is detected by the AI-based object-detection algorithm running on the edge device/server, the cube (product/item) on the conveyor belt is considered as a defective item.

3.8.2 Evaluation methodology

The algorithm used for defect detection typically follows a framework for designing and training DNN. The choice of the detection algorithm and its AI training model depends on the characteristics and capabilities of the resources deployed for both the training and inference environments. These characteristics may vary dynamically and therefore an adaptation to the algorithm and/or its training model, as well as where it is instantiated, may be required to achieve low latency and energy efficient deployment. As noted in the initial high-level design of this use case in D4.1, the YOLO "you only look once" algorithm is used for object-detection/defect-detection. The YOLO algorithm targets real-time processing where the detection procedure is taken as a regression task in order to increase the detection speed and pass the image only one time through the network. In this implementation, the considered version of the YOLO algorithm is version 3. The neural network used for YOLOv3 improves the accuracy of the previous versions. The YOLO algorithm is built on Darknet [62] which is a GPU accelerated framework to design and train DNNs. The version Darknet-53 which acts as a backbone for the YOLOv3 object-detection approach is considered in this implementation. The Darknet-53 is trained with Imagenet [63] classification data set. The DNN in Darknet-53 uses multi-scale predictions, which means that detection phase is applied to different scales of the feature map.

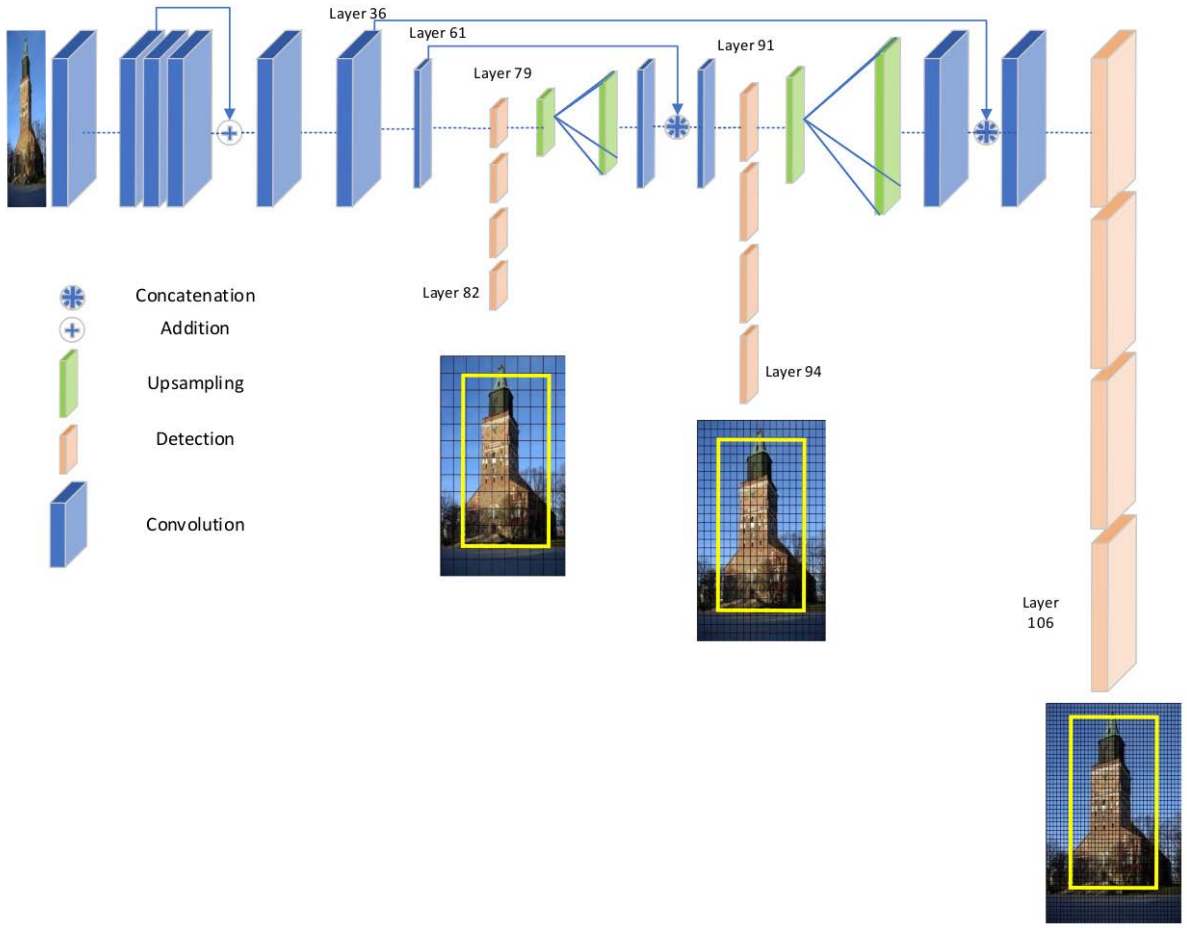


Figure 3.65. Used DNN architecture

The input image is down sampled three times by factors of 32,18, and 8. As shown in Figure 3.65, the first detection is made by layer 82, where up to there, the image is down sampled by the convolutional layers such that 81st layer has a stride of 32. If the input image is 416x416 then the resultant feature map is 13x13. One detection is made using 1x1 detection kernel, giving a detection feature map of 13x13x255. Then, from layer 79, the feature map in sent through convolutional layers before being up sampled by a factor of 2. Afterwards, the feature map is concatenated with previous features from layer 61. The second detection is made at layer 94 yielding a feature map of 26x26x255. The final detection is done at layer 106, which produces a feature map of size 52x52x255. Detection at different layers helps in increasing the accuracy in detecting small objects. The up samples concatenated with previous layers help in preserving the fine grain detection done before.

As the YOLO algorithm is built on Darknet, which is a GPU accelerated framework for designing and training DNNs, the YOLO algorithm suffers from challenges pertaining to execution latency and energy consumption. To mitigate these challenges, instead of an GPU-based solution, an FPGA-based solution has been studied. The FPGA can provide lower latency than the traditionally used GPU platforms for this type of application. The FPGA platform may provide a better energy efficiency solution compared to CPU and GPU based solutions. Taking advantage of the data processing units, higher throughput in terms of number of processed frames per second can also be achieved by FPGA-based solution.

3.8.3 Evaluation results

Comparison between FPGA-based and GPU-based object-detection by means of latency and energy efficiency is provided in Table 3-26. For both platforms, three different object-detection algorithms, namely

YOLOv3, Mobilenetv2_SSD [64][65] and VGG16_SSD [66] are deployed to investigate end-to-end (E2E) latency, power consumption and frame-per-second (FPS) per Watt consumption. For the FPGA-based platform, Xilinx ZCU102 that has a quad-core Arm® Cortex®-A53, dual-core Cortex-R5F real-time processors, and a Mali™-400 MP2 GPU is used. The MaliTM-400 MP2 GPU in Xilinx ZCU102 has a clock rate up to 667 MHz. Whereas, for the GPU-based platform, GeForce RTX 2080 Ti which has a clock rate up to 1545 MHz is used.

Regarding the E2E latency performance, the deployed object-detection algorithms can be listed from minimum to maximum achieved latency as YOLOv3, VGG16_SSD and Mobilenetv2_SSD for the GPU-based platform and YOLOv3, Mobilenetv2_SSD and VGG16_SSD for FPGA-based platform. For both platforms, YOLOv3 achieves significantly lower E2E latency compared to other two algorithms. The lowest E2E latency performance between the considered algorithms and platforms is achieved when the FPGA-based platform is used with YOLOv3 algorithm. In addition to that YOLOv3 consumes 120 Watts, which is the same with Mobilenetv2_SSD algorithm and half of the power consumption of VGG16_SSD algorithm that is 240 Watts, when the GPU-based platform is used. When the FPGA-based platform is used with YOLOv3 algorithm, the power consumption is 4 Watts higher than what Mobilenetv2_SSD achieves, and 5.5 Watts lower than what VGG16_SSD algorithm achieves. Therefore, it can be said that YOLOv3 achieves a better latency-energy consumption trade-off compared to Mobilenetv2_SSD and VGG16_SSD algorithms. For the considered platforms, the FPGA-based platform consumes significantly lower power and provides better E2E latency performance than the GPU-based platform.

Table 3-26. Latency and energy consumption performance results for object-detection

Platform	Object-Detection algorithm	E2E latency (FPS multithreaded)	Power [W]	FPS per Watt
GPU: GeForce RTX 2080 Ti	YOLOv3	5.72 ms (175)	120	1.46
	Mobilenetv2_SSD	49.75 ms (20)	120	0.17
	VGG16_SSD	22.1 ms (45)	240	0.19
FPGA: Xilinx ZCU102	YOLOv3	4.52 ms (220)	17	12.94
	Mobilenetv2_SSD	16.37 ms (61)	13	4.69
	VGG16_SSD	22 ms (45)	22.5	2

4 AI Engine

The initial design of the 5G-CLARITY AI Engine was described in 5G-CLARITY D4.1 [1][1]. The goal of this section is to revise that initial design and to discuss the used technologies and implementation details of the first AI engine prototype.

4.1 AI Engine technologies

The open-source Function-as-a-Service (FaaS) platform OpenFaaS [33] has been selected as the basis for implementing the AI engine. OpenFaaS is a flexible and lightweight toolkit that advertises to be able to run anywhere, with any code and at any scale:

- Anywhere: Avoid lock-in through the use of Docker. Run on any public or private cloud.
- Any code: Build both microservices & functions in any language. Legacy code and binaries.
- Any scale: Auto-scale for demand or to zero when idle.

The conceptual diagram shown in Figure 4.1 gives a high-level overview of OpenFaaS concepts, with endpoints for the user, monitoring, Kubernetes as the FaaS provider and two example functions (e.g. executable ML models in the 5G-CLARITY case). Towards the user, OpenFaaS exposes REST endpoints that can be accessed via a CLI client, a graphical user interface or REST requests directly. The central FaaS provider (recently Kubernetes) provides the infrastructure for deployed functions. Each function contains a function watchdog that listens for incoming requests and the process that fulfils the request. In the 5G-CLARITY AI engine, this process represents an ML model that can be called and acts upon the given input. For example, a pre-trained indoor ranging ML model (Section 3.5) might await a trigger and/or data to predict the range between two locations such as the UE and the base station, which will then produce a range figure as output.

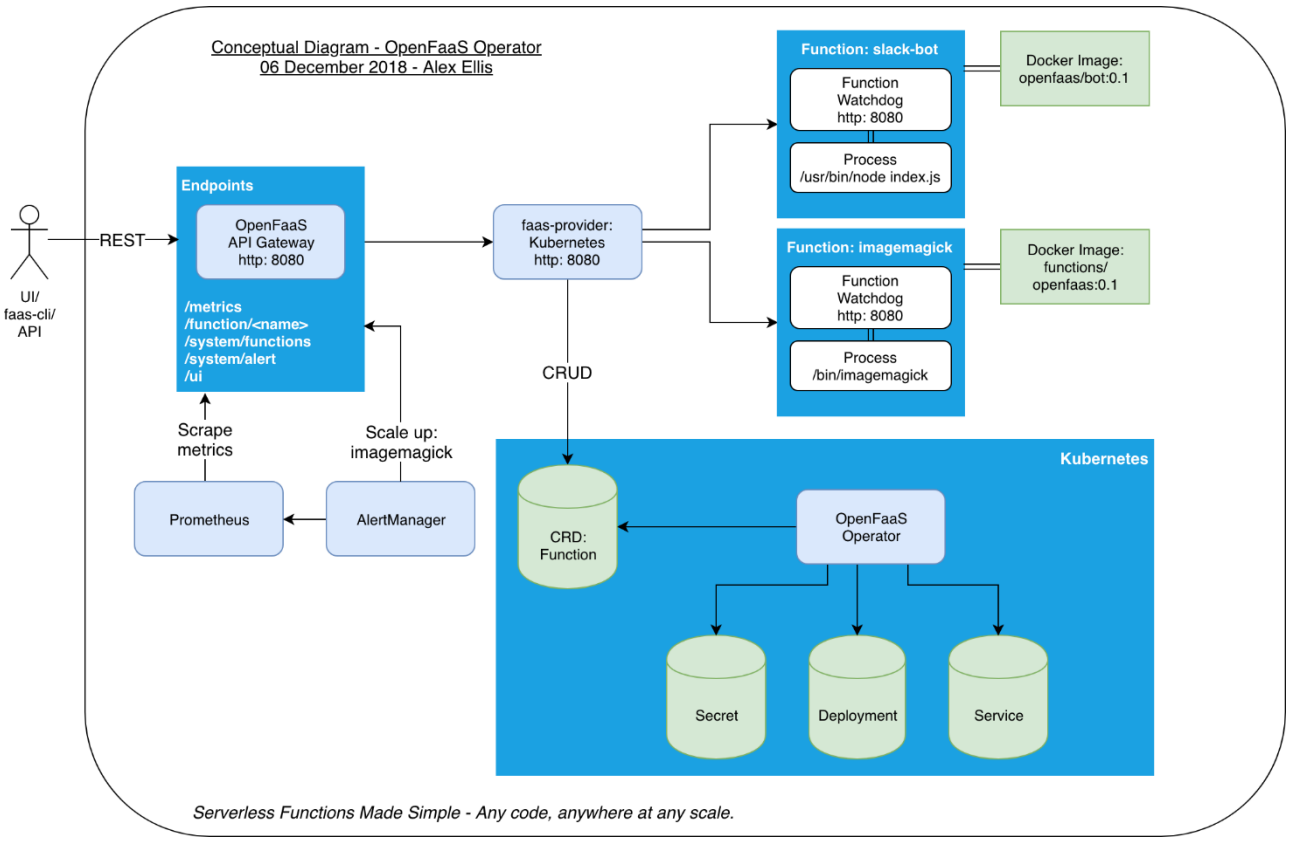


Figure 4.1. Overview of OpenFaaS concepts (source: [68])

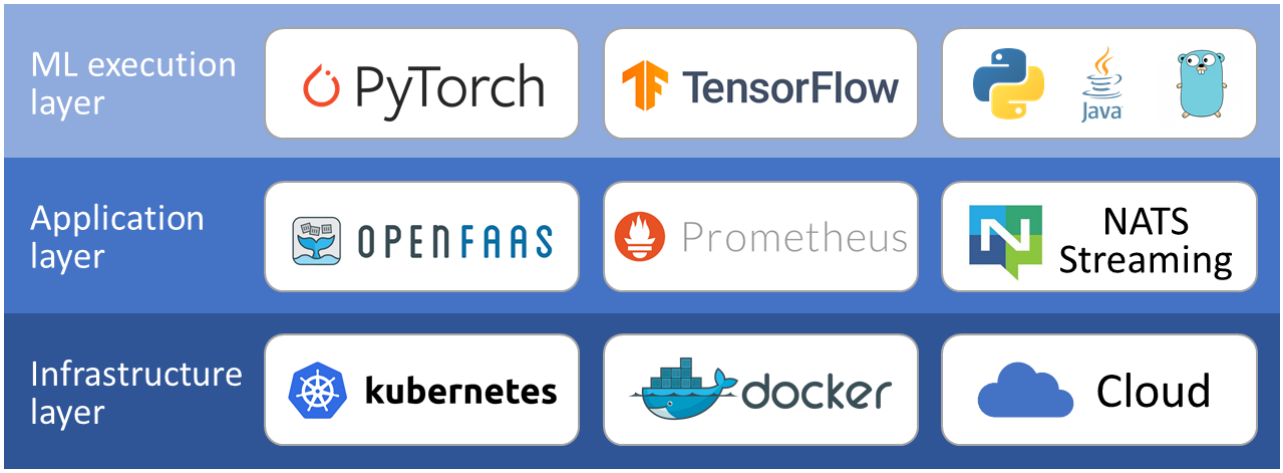


Figure 4.2. AI Engine tech stack, with infrastructure, application and machine learning model execution layers

Figure 4.2 presents the tech stack of the AI engine, with OpenFaaS as the supporting framework. This shows that the AI engine uses a containerised approach and can be deployed in any Docker-able infrastructure, such as Kubernetes clusters on-premises or in the cloud.

In the application layer is the OpenFaaS core, which uses Prometheus to monitor the demand of services (which are automatically scaled by OpenFaaS), and NATS Streaming to enable asynchronous function calls for long-running services such as ML model training. On top of the service demand monitoring, the 5G-CLARITY AI engine also employs Prometheus (in conjunction with a visualisation tool like Grafana) to monitor the health and performance of the running ML models.

The ML execution layer contains the actual ML models that are deployed in the AI engine, with direct support for several programming languages as well as generic binary files. That means that any ML algorithm written in, for example, Python, Java or Go, or using ML libraries like PyTorch or TensorFlow, can be wrapped into an OpenFaaS function and deployed into the AI engine. In this way, the deployed ML models can be provided language and framework independent.

4.2 Consolidation of AI engine design – Revision and implementation details

Sections 4.2.1 to 4.2.4 discuss implementation details for various aspects of the AI engine, including updates to the initial design from 5G-CLARITY D4.1 1.

4.2.1 Exposed services

There are some changes to the initial AI engine services that were described in 5G-CLARITY D4.1. Table 4-1 lists the services together with revisions for 5G-CLARITY D4.2. The main changes concern the split of the ML Model Management service into ML Model Deployment and ML Model Removal services. Another change is the removal of the ML Model Registry service, as that functionality is being provided partially by the deployment service (ML models are automatically registered in the deployment process) and partially by the ML Model List service (which lists the registered ML models).

Table 4-1. AI Engine Services.

MF Service ID	MF Service Name	Description	Reference Specifications
AIEngine_Ml_Mod_Deploy NOTE: This is a	ML Model Deployment service	Deployment of a new ML model. If the model already exists, it can be updated instead. A new model will automatically be registered in the ML Model Registry during the	Custom (REST)

combination of the AIEngine_Ml_Mod_Mgmt and AIEngine_Ml_Mod_Register services defined in D4.1		deployment process.	
AIEngine_Ml_Mod_Remove NOTE: This is replacing the AIEngine_Ml_Mod_Mgmt service defined in D4.1	ML Model Removal service	Removal and de-registration of an existing ML model.	Custom (REST)
AIEngine_Ml_Mod_List	ML Model List service	Lists the ML models that are currently deployed and ready for execution.	Custom (REST)
AIEngine_Ml_Mod_Run	ML Model Execution service	Runs a deployed ML model that is available in the ML Service Registry. Once executed, the deployed ML model may run as a once-off or recurrently.	Custom (REST)
AIEngine_Push_XApp	xApp Pushing service	Certain (real-time ready) ML models can be pushed down into the near-RT-RIC as xApps.	Custom (REST)

4.2.2 Containerised ML models

As was described in D4.1, the 5G-CLARITY AI engine hosts ML models that are wrapped in Docker containers for flexible, language independent and scalable execution. The containerisation of the ML models is an automated part of the deployment process, which means that the ML model designer requires little or no knowledge about Docker or containerisation. The ML model designer merely needs to provide the ML algorithm and include it into an OpenFaaS function template, while adhering to the templated input and output formats. The OpenFaaS platform takes care of the rest. OpenFaaS provides function templates for many languages, as well as a generic template for binary files.

OpenFaaS comes with a command line interface tool (i.e., `faas-cli`) to control the various aspects of a function lifecycle, including creating a new function from a template. For example, executing the following command will create a new Python 3 function for the AI engine:

```
faas-cli new --lang python3-aiengine <name>
```

The newly created OpenFaaS function will contain a number of files, where the core part is the Python function `handle()`, inside a file called handler.py:

```
def handle(request):
    output = apply_ml_model_to(request)
    expose_metrics(output)
    return output
```

The code inside this function will automatically be wrapped in a container and executed at every call to the AI engine, where the imported ML model code inside the `handle()` function will be applied to the input data (e.g. input feature values). The additional monitoring boilerplate code will expose ML model metrics to

the Prometheus monitoring system. With the `faas-cli` and the provided templates, the ML model designer is equipped to deploy their models into the AI engine.

4.2.3 ML service registry

Deployed ML models are available in the ML service registry, which is provided by OpenFaaS' function list. One option to access the ML service registry is through command line, where the following command lists all deployed models:

```
faas-cli list
```

The output of this command may look like depicted in Figure 4.3, listing all deployed models, how many replicas they have and how many invocations.

Function	Replicas	Invocations
Model A	1	324
Model B	2	2091
Model C	1	12

Figure 4.3. Output of the `faas-cli` list command

Another way to view the deployed ML models is through the graphical user interface (GUI) that is provided by OpenFaaS. Figure 4.4 shows this GUI with some deployed ML models. The GUI also allows to directly test the functionality of the ML model by creating input values, calling the ML model and inspecting the output. Both the `faas-cli` tool as well as the GUI will call the model through its REST API under the hood.

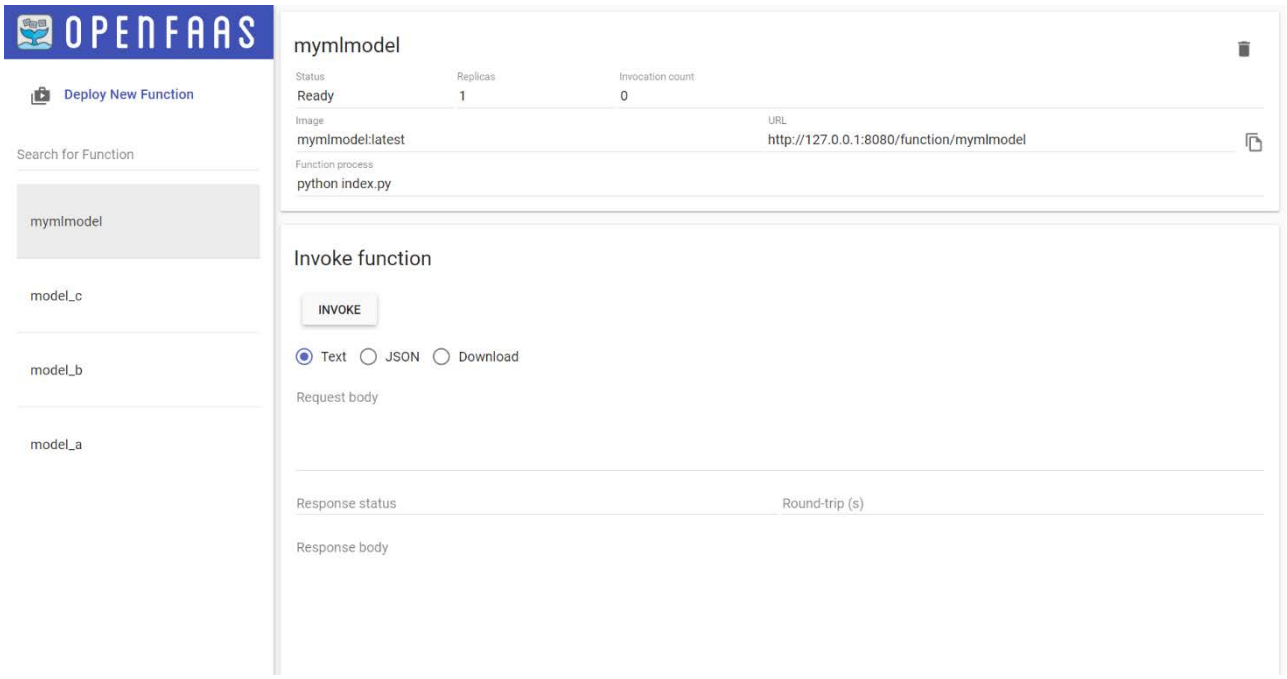


Figure 4.4. ML service registry powered by OpenFaaS, showing several deployed ML models whose functionalities can be tested through the OpenFaaS GUI.

4.2.4 ML model lifecycle management

The lifecycle of an ML model in relation to the AI engine consists of the following processes: *i)* dockerising; *ii)* deployment; *iii)* execution; *iv)* monitoring; *v)* update and *vi)* removal. These processes will be discussed in the following subsections.

4.2.4.1 ML model Dockerising

An ML model has to be Dockerised before it can be deployed in the AI engine. OpenFaaS provides a simple Dockerisation process via the `faas-cli` tool. It enables ML model designers to wrap their models into deployable Docker images simply by inserting the ML model code into a provided template. The following command will build a Docker image from the stack file (YAML) together with the raw code or binary file that were inserted into the template:

```
faas-cli build -f <filename>
```

The output is a Docker image, which can also be pushed to a Docker registry for future use. This step is optional and can be achieved by calling faas-cli push on the YAML stack file:

```
faas-cli push -f <filename>
```

A Docker registry allows for easy re-deployment as well as deployment in other systems without the need to build the image from scratch every time.

4.2.4.2 ML model deployment

After the ML model has been Dockerised, it can be deployed in the AI engine. The following command deploys an ML model that has been Dockerised beforehand:

```
faas-cli deploy -f <filename>
```

Alternatively, the graphical user interface (GUI) provided by OpenFaaS can be used for model deployment (and other functionalities). Figure **4.5** exemplifies a deployment using the GUI, with the Docker image of the ML model and its name as input values.

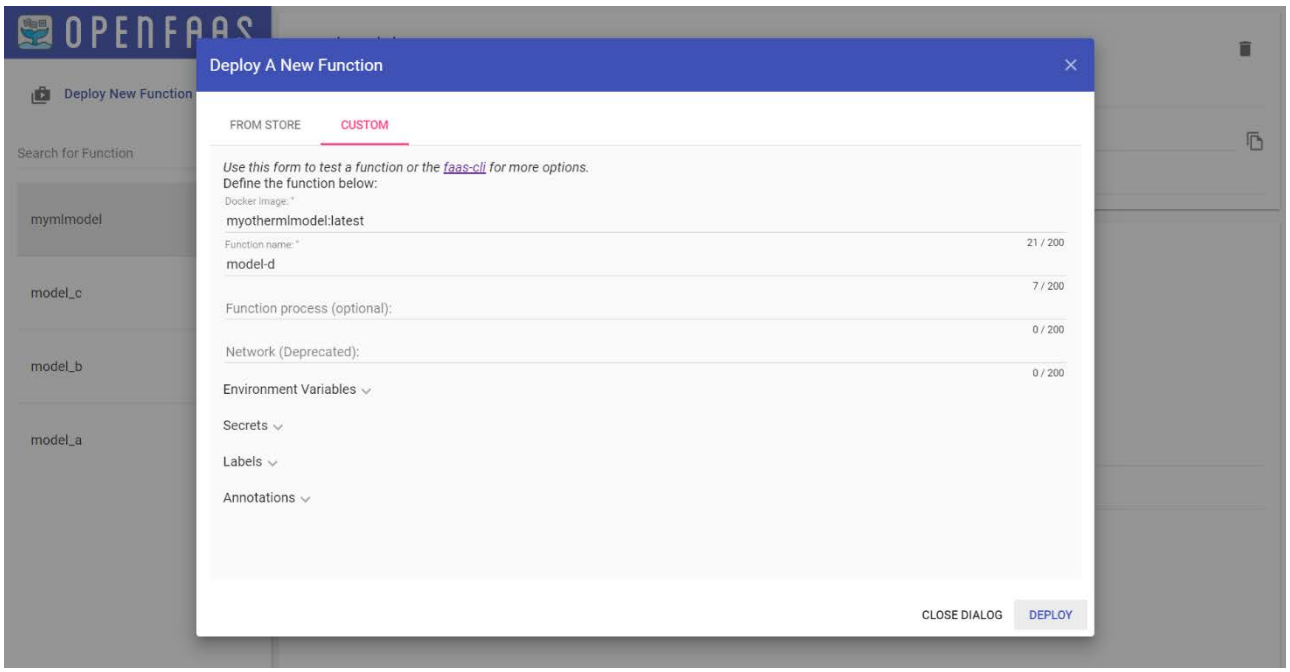


Figure 4.5. Function deployment through the OpenFaaS GUI

4.2.4.3 ML model execution

An ML model that has been successfully deployed can now be executed, or triggered, to deliver predictions to the user. There are two ways of executing a function: synchronously and asynchronously. Synchronous functions fulfil the role of short-lived functions that return an output in a short time and can time out if the processing is not finished within the desired time frame. Asynchronous functions can be long-lived without a set time out, but their output must be polled for separately (i.e., continuously asked whether the result is ready). Synchronous functions are suited for execution of trained ML models, whereas asynchronous functions are suited for ML model training. OpenFaaS uses NATS streaming to enable asynchronous functions.

The `faas-cli` call below will execute a deployed ML model, where the optional -a flag specifies whether the model shall be executed asynchronously.

```
faas-cli invoke <modelname> [-a]
```

ML models can also be executed from within the GUI as shown in Figure 4.4, although programmatic execution is more practical for real world use cases (see Section 4.3).

4.2.4.4 ML model monitoring

During the lifetime of an ML model, its performance can be monitored. For this purpose, the Prometheus instance that comes with OpenFaaS is configured to also monitor ML model metrics in addition to the default Infrastructure metrics, and the provided AI engine function template includes boilerplate code to expose the desired metrics to Prometheus.

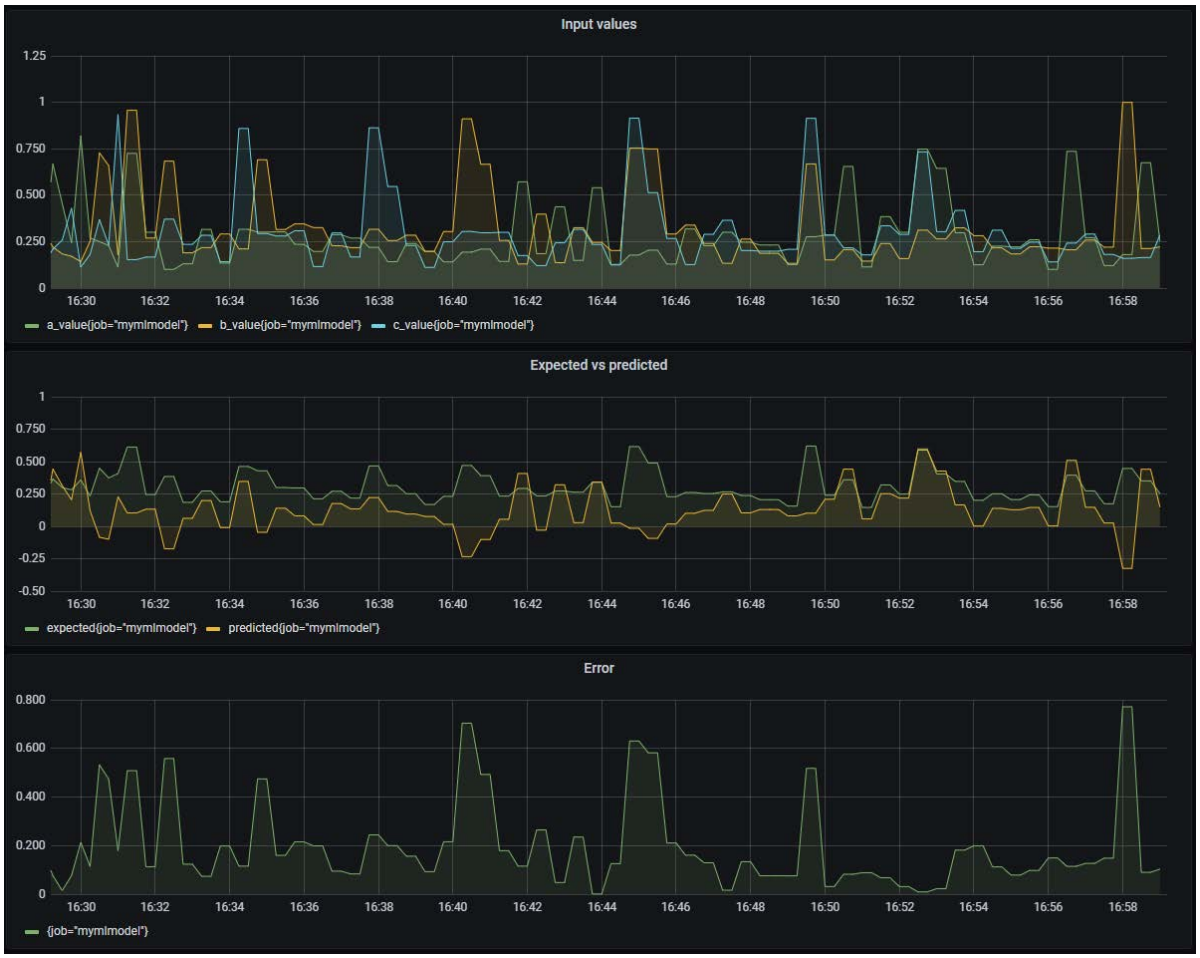


Figure 4.6. Monitoring of metrics related to an ML model with the name "mymlmodel". Visualisation dashboard provided by Grafana.

Grafana can be used to visualise the exposed ML model metrics, and a recent version of Grafana is included in the AI engine installation. The graphs shown in Figure 4.6 exemplify the visualisation of metrics that are exposed by an ML model with the name “mymlmodel”, including the input values, prediction value, and prediction error. A possible use case for visualising ML model metrics is to be alerted when the ML model behaviour changes drastically and requires human intervention. Both Prometheus and Grafana can be used to set up alarms to notify the human users about certain changes.

4.2.4.5 ML model update

The user can update the deployed ML model in the AI engine when a new version of an ML model has been developed or when the model has been trained on new data – for example after detecting a decline in prediction accuracy from the monitoring. The update process comprises the same steps as the deployment process (including Dockerisation of the new ML model) described in Sections 4.2.4.1 and 4.2.4.2, as the model deployment will overwrite an existing model with the same name.

4.2.4.6 ML model removal

Finally, an ML model may be removed from the AI engine. The faas-cli command for removal is as follows:

```
faas-cli delete <name>
```

Alternatively, the models may also be removed directly from the GUI as shown in Figure 4.7.

4.3 Functional validation

The sections above introduced the faas-cli tool and the graphical user interface (GUI) as ways to control the ML model lifecycle in the AI engine in a user-friendly way. The programmatic way to interact with the AI engine is through REST, which will be the main interface between the intent engine and AI engine as well. This section describes the REST API endpoints that allow for programmatic life-cycle management of ML models.

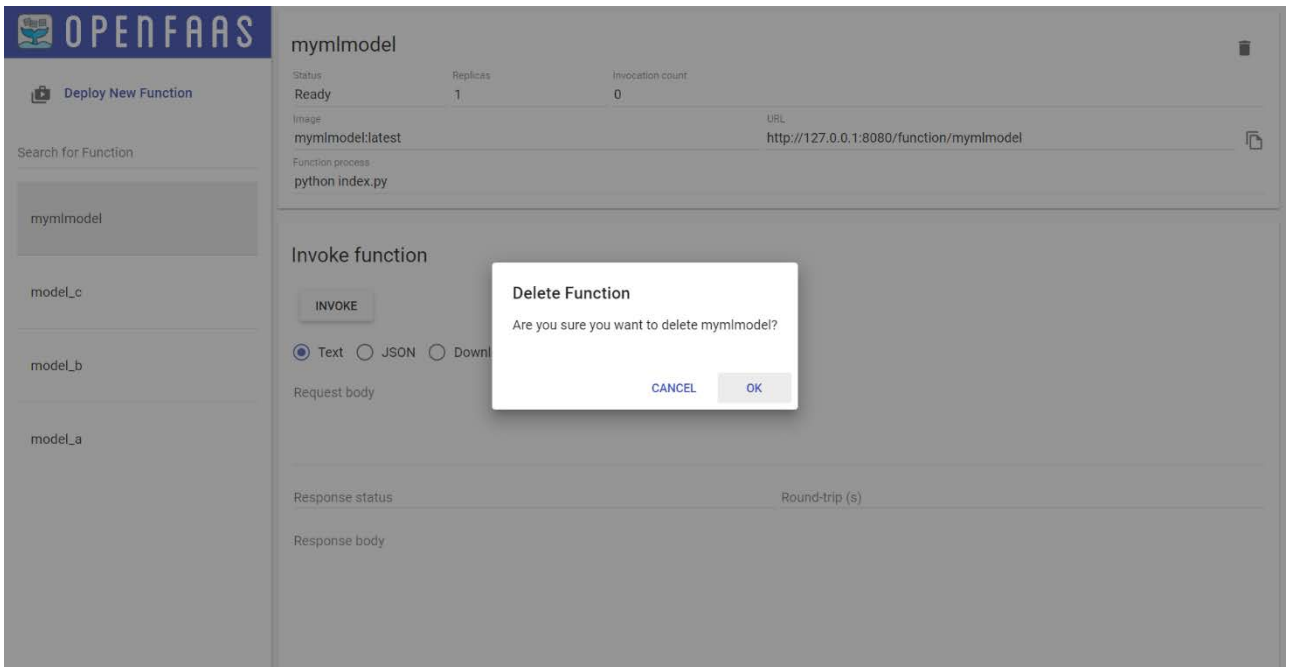


Figure 4.7. Deletion of a deployed function through the OpenFaaS GUI

4.3.1 Deployment

`/system/functions` (POST):

Deploy an ML model, where the model is specified in the request body as exemplified below.

```
{
  "service": "mymlmodel",
  "network": "aimodel-net",
  "image": "mymlmodel:latest",
  "envProcess": "python index.py",
  "labels": {
    "name": "MyMLmodel"
  },
  "annotations": {
    "topics": "awesome-kafka-topic",
    "desc": "This model is doing awesome ML stuff"
  },
  "limits": {
    "memory": "128M",
    "cpu": "0.01"
  },
  "requests": {
    "memory": "128M",
    "cpu": "0.01"
  }
}
```

The response is 202 for a successfully deployed ML model and 400/500 otherwise.

4.3.2 Update

`/system/functions` (POST):

Update an ML model, with similar request body and responses to the deployment through POST.

`/system/scale-function/<name>` (POST):

Manually scale the ML model with a request body as outlined below.

```
{"service": "mymlmodel", "replicas": 10}
```

With the response codes 200/202 for successful scaling, 404 for model not found and 500 for other scaling errors.

4.3.3 List and model info

`/system/functions` (GET):

Get a list of all deployed ML models. This query has no input parameters and returns the following output (exemplified).

```
[
  {
    "name": "mymlmodel",
    "image": "mymlmodel:latest",
    "invocationCount": 1337,
    "replicas": 2,
    "availableReplicas": 2,
    "envProcess": "python index.py",
```

```
    "labels": {
      "foo": "bar"
    },
    "annotations": {
      "topics": "awesome-kafka-topic",
      "foo": "bar"
    }
  }
]
```

`/system/function/<name>` (GET):

Get a summary of the deployed model <name>, with the following successful (exemplified) response.

```
{
  "name": "mymlmodel",
  "image": "mymlmodel:latest",
  "invocationCount": 1337,
  "replicas": 2,
  "availableReplicas": 2,
  "envProcess": "python index.py",
  "labels": {
    "foo": "bar"
  },
  "annotations": {
    "topics": "awesome-kafka-topic",
    "foo": "bar"
  }
}
```

Response code 200 is successful info retrieval, code 404 means ML model not found and code 500 indicates an internal server error.

`/system/logs?name=<name>` (GET):

Get a stream of the logs for the model <name>, with the response as follows.

```
{
  "name": "mymlmodel",
  "namespace": "",
  "instance": "vb70z722qxblac3u9vu437cfu",
  "timestamp": "2021-03-09T20:41:16.2652549Z",
  "text": "2021/03/09 20:41:16 POST / - 200 OK - ContentLength: 52"
}
```

Response code 200 is successful log retrieval, code 404 means ML model not found and code 500 indicates an internal server error.

4.3.4 Execution

`/function/<name>` (POST):

Invoke an ML model in synchronous mode (get immediate response). The example below shows the input and output of a prediction model.

```
{
  "input_a": 10,
  "input_b": 13,
  "input_c": 7
```

```
}
```

With the return response reporting the output of the model:

```
{
  "expected": 11,
  "predicted": 10.2
}
```

Response code 200 is successful model invocation, code 404 means ML model not found and code 500 indicates an internal server error.

`/async-function/<name>` (POST):

Invoke an ML model asynchronously, which will queue the model execution. The input format is identical to the synchronous ML model execution above, but the only immediate response are the REST codes (200, 404 and 500 as above). There is no immediate model output.

4.3.5 Removal

`/system/functions` (DELETE):

Remove a model, where the model name is specified in the request body as follows:

```
{
  "functionName": "mymlmodel"
}
```

The response codes are 200 for successful removal, 400 for a bad request, 404 for model not found, and 500 for internal server errors.

5 Intent Engine

The initial design of the 5G-CLARITY Intent Engine was described in D4.1 [1].The goal of this section is to revise that initial design and to discuss the used technologies and implementation details of the first intent engine prototype.

Table 5-1. Intent Engine Services.

MF Service ID	MF Service name	Description	Reference Specifications
IntentEngine_NtntProv_Mgmt	**Intent Provider Management service**	For an intent to be fulfilled, it needs to be matched to an intent provider. This service allows the addition, updating and removal of intent providers for the intent engine to match intent descriptions against. Update intent providers for when intent providers change. Many AI models are intent providers, which may be updated and replaced from time to time. Intent providers may be removed when they are outdated or no longer desired.	Custom (REST)
IntentEngine_ NtntProv _List	**Intent Provider List service**	Lists currently registered and available providers. The list can be simple (names) or extended (provider names and supported intents).	Custom (REST)
IntentEngine_ NtntProv _Describe	**Intent Provider Description service**	Returns a description of an intent provider. This description can be simple (overview of the provider's functionality), enhanced (description with overview of supported intents), or detailed (description of a particular intent and how the provider supports it).	Custom (REST)
IntentEngine_ NtntProv _Run	**Intent Execution service**	Runs a received intent on a provider. The provider can be explicitly named. When no provider is named, the intent engine will try to match the provided intent description with an existing intent provider, who executes the intent description. It may then return a result or run recurrently.	Custom (REST)
IntentEngine_Ntnt_Terminate	**Intent Terminations service**	Terminates an intent that is currently executed on the engine.	Custom (REST)
IntentEngine_Ntnt_Status	**Intent Status service**	Returns the status of a running intent.	Custom (REST)
IntentEngine_Ntnt_Get_Telemetry	**Telemetry Data Retrieval service**	Get telemetry data from the Telemetry Collector. This service allows ML models to retrieve network data.	Custom (REST)
IntentEngine_Ntnt_Push_Config	**Network Configuration Forwarding service**	Forward network configurations to various network elements (e.g. Slice Manager). This service allows ML models to push network configurations into the network.	Custom (REST)
IntentEngine_Ntnt_Mgmt	**Intent Management service**	Allows for the deployment of intents on the engine that are ready to be executed (run) and to remove intents from the engine. If the intent is currently executed, it will be terminated before removal.	Custom (REST)

IntentEngine_Ntnt_List	**Intent List service**	Lists currently deployed intents. The list can be simple (intent names) or extended (intent names and matching providers).	Custom (REST)
IntentEngine_Ntnt_Describe	**Intent Description service**	Describes an intent. The description can be simple (overview of the intent) or detailed (explaining all details of the intent)	Custom (REST)

5.1 Consolidation of intent engine design

This approach takes the architectural overview of the 5G-CLARITY Intelligence Stratum with modifications to the representation of components from the perspective of a practical usage scenario. The intelligence stratum is functionally similar to an intent-driven, AI supported controller with monitoring capabilities. For the sake of simplicity, let it be 'intelligent controller' in the rest of this section.

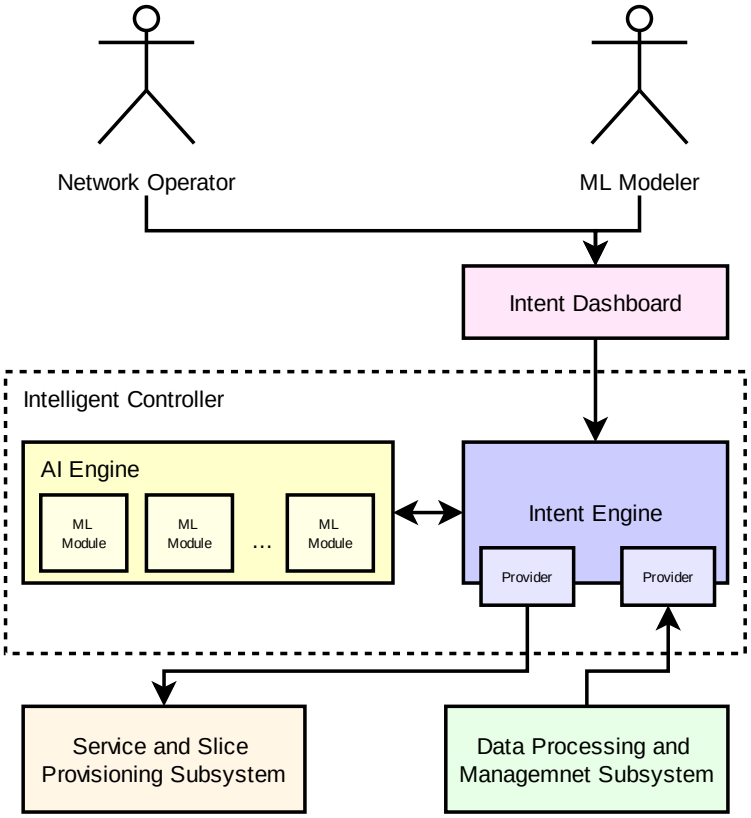


Figure 5.1. Architectural overview of the intelligence stratum

Intents are specified by the network operator. The supporting AI models (and algorithms) are designed by an ML modeler. Intents are taken by the "Intent Engine", which serves as the central coordination component. The AI modules provide the machine learning intelligence to deal with intent presented by the network operator. The Intent Engine can then use the 5G-CLARITY Service and Slice provisioning subsystem to realize (often also called enforce) configuration decisions. This subsystem in Figure 5.1 represents any target MF responsible for realising the request detailed in the intent message. For example, an intent requesting the reconfiguration of a slice (or network service or transport network) means the target MF is the Slice Manager (or the NFVO or the SDN controller). Once an intent has resulted in a configuration, the underlying managed system can be monitored. Information about the network comes from the 5G-CLARITY Data Processing and Management subsystem, either from Data Semantics Fabric or Data Lake. This result in

a 4-step process.

The first step is the injection of an intent into the "Intent Engine". This communication is typically between the "Network Operator" and the "Intent Engine"; however the source of the intent message can also be an internal component within the 5G-CLARITY system through the "Intent Engine" REST interface. This should be facilitated by an interface that drastically simplifies intent expressions, an example is shown in Section 5.2. Close to natural language would be preferable, to minimize any required training of the network operator's personal. An intent here could be for instance "create a new RAN slice with 4 RAN nodes, 2 Wi-Fi nodes, and 1 Li-Fi node; add a core network for all users of factory X". This intent can be encoded using the language and the encoding examples shown in Section 5.2. The Intent Engine receives the intent, stores it in a local (runtime) store, and validates if an appropriate ML module is available to deal with the intent. Optionally, this can be supported by a translation of the original intent into an intermediate format, which is designed to capture all ML modules. Once the intent is translated and an appropriate ML module is selected, the Intent Engine can send this new intent to the ML module.

The second step is the communication between the Intent Engine and the ML modules. As stated above, a new intent is created and sent to a selected ML module. This module runs its algorithm. For a pure runtime system, this will require a fully trained ML model. The outcome of an ML module should be detailed instructions for the "Intent Engine" on what it must do next. These instructions can be again described as an intent, or a configuration template, or otherwise. If a ML module is not selected the "Intent Engine" will search for internal information specific to the intent request. If the "Intent Engine" cannot identify the appropriate next steps a message is sent to the issuer of the intent informing them of the problem. In a scenario where a ML module requires information from Telemetry in order to generate an action the "Intent Engine" needs to be provided the location of the telemetry data so it can be retrieved. The information is then forwarded to the ML module.

The third step is the communication between the Intent Engine and the 5G-CLARITY. Once the Intent Engine receives its detailed instructions, it may take them directly (if appropriate) or translate them for a selected MF. Translation should be the preferred method here to decouple the ML modules from orchestration details. The MF can be for instance a slice manager, a use case for which is described in Section 5.3.1. Once the instructions are sent to the MF, the Intent Engine will also need to register data that should be monitored until the original intent is either fulfilled (automatically terminated) or until the original intent is undeployed (manually removed by the network operator). The utilisation of continuous loops to provide assurance in the scenario of a prolonged intent are described in D4.1 [1], Section 7.2.2.

Finally, the fourth step is the communication between the Data Management and Processing subystem and the Intent Engine. Independent of the second step where the Intent Engine retrieves information to feed a ML module, in this scenario monitoring data is requested by the Intent Engine. The monitor sends data to the Intent Engine. The engine now can check if any of the data is associated with an intent that had resulted in a configuration. If so, it can negotiate with the original ML module to jointly decide if the new data violates the original intent, and if thus a new configuration or a configuration update needs to be created and issued to the MF.

The rationale above shows a simple way to build the Intent Engine. The 4-step process requires three state machines inside the intent engine. We can call the state machines S1, S2, and S3. The state machine S1 will take an operator's intent as input, translate it into an intermediate intent, select a deployed ML module that is able to handle the intermediate intent, log all this information in the local store (to create traceable history and to store state information about processed intents), and then send the new intent to the selected ML module. This state machine can only be triggered by deploying a new intent, and it terminates once a new intent is sent to an ML module. However, if an ML module is not selected to handle the intent the system moves to the next state with just the original intent information.

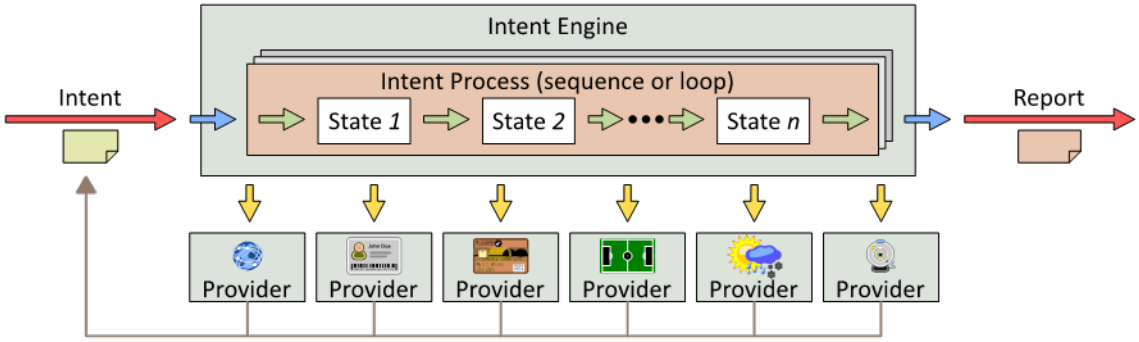


Figure 5.2. Intent Engine concept

The state machine S2 takes as input an intent (or detailed instructions) from an ML module, relates this to an original intent (from the local store), selects an MF, optionally translates the instruction into what the MF understands, sends the translated instructions to the MF, associates monitoring events to the intent to allow for effective assurance during the monitoring phase, and stores all process related information in the local store under the original intent. This state machine can only be triggered by an ML module, and it terminates once instructions are sent to the MF.

The state machine S3 receives monitoring data from the 5G-CLARITY data processing and management subsystem. It then determines if the data is associated to an original intent. If it is, the local information of the original intent will point to the ML module that has created the original detailed instructions. The state machine can now, jointly with this ML module, decide if the new monitoring data represent an intent violation and if any actions are required. If actions are required, the second state machine S2 can be triggered to generate a new configuration or a configuration update.

Figure 5.2 shows the intent engine as a container executing any number of state machines. As described above, we can realize the whole intent coordination with three independent state machines, each realized as a sequential machine (or a tree in case of more complex local states). No loop in the state machines is required, since they are all part of a closed control loop system already. Input to the engine are intents and monitoring data (not shown in the figure). The “Intent Providers” represent the ML modules and the MF. The state machines will also create reports stored locally, which can be send to any interested component outside the controller. This might be interesting in case of intent fulfilment, but also to provide monitoring information about the Intent Engine and its state machines to higher-level processing systems, e.g., an OSS or a BSS.

The Intent Engine can be implemented using any system or environment that can implement state machines. One candidate is the policy engine APEX [27], which in fact is a generic state machine executor. Here, each of the described state machines S1-3 could be realized as an APEX policy, and all of them in combination be embedded into an APEX Policy Model. In a simpler way, each state machine could also be implemented in form of imperative logic, coordinated objects, or functional.

The Intelligent Controller can be extended by adding more ML modules, other MFs, or more sophisticated Telemetry components. In a simple implementation, this will require to add logic to the intent state machines, to deal with the new artifacts. However, the whole system can also be designed in a model driven way. Here, an ML module that is added to the system would provide all information, as a model, to the Intent Engine: supported intermediate intents, supported original intents, monitoring data associated with the functionality, MFs and translations to them that support the functionality, and even details for the GUI to present the new functionality to the network operator. In this case, the state machines in the Intent Engine can be implemented generically, following the same process using the modelled information.

5.1.1 South-bound interfaces

This section provides details on the southbound interactions of the intent engine.

5.1.1.1 Slice and Service Provisioning Subsystem

Communication between the Intent Engine and this subsystem is handled over REST. Instructions derived from the intent request or produced by the AI Engine are translated into requests executable over the Slice Manager REST API. Acknowledgement of these requests are validated to ensure the successful execution of the Intent Engine. Functions described in the documentation of the Slice Manager API are built in the latter parts of the Intent Engine execution cycle. Parameters provided in the intent message are used to populate required fields for generated request calls.

5.1.1.2 Data Processing and Management Subsystem

Communication between the Intent Engine and Telemetry is yet to be finalised. The most likely interfaces are over REST or Kafka. More information may be found in Section 2.2. Two approaches have been considered for the interaction between Telemetry and the Intent Engine. The first treats the Telemetry resource as an intent provider in which case a data request is packaged by the Intent Engine and on receipt of the data, returns the response to the intent issuer. Alternatively, there is a mechanism available to ML Modules to interact with Telemetry directly. In this scenario the Intent Engine would provide the location of the requested data to the ML Modules allowing them to pull the data directly.

5.1.2 North-bound interfaces

The northbound interactions of the intent engine are limited to exchange of request-response messages to the intent issuer dashboard. Intents will be issued through a dashboard using a Kafka messaging system to communicate with the Intent Engine. The dashboard provides an intent message template containing Intent Engine specific information alongside editable fields expecting Intent Language compliant syntax in the YAML or JSON format. Figure 5.3 illustrates an example of this YAML/JSON encoded intent language.

The above-referred approach constitutes a straightforward starting point for intent representation in the system. As the Intent Engine matures different mechanism such as Natural Language Processing can be incorporated to allow for flexible representations of the intent message.

```
{
    "Intent" : {
        "Header" : {
            "Creator" : "user/vdmeer",
            "ID" : "session/zv3J::1",
            "Timestamp" : "Wed 26 Feb 2020 15:14:10 GMT",
            "Template" : "count/lines+of+code/0.0.1"
        },
        "Primary" : {
            "Subject" : "user/vdmeer",
            "Verb" : "count",
            "Object" : "lines+of+code"
        },
        "Secondary" : [
            "How details yes"
        ]
    }
}
```

```
---
Intent
    Header
        Creator:user/vdmeer
        ID:session/zv3J::1
        Timestamp:Wed 26 Feb 2020 15:14:10 GMT
        Template:count/lines+of+code/0.0.1
    Primary
        Subject:user/vdmeer
        Verb:count
        Object:lines+of+code
    Secondary:
        - How details yes
...
```

Figure 5.3. Intent language. JSON (left) and YAML (right)

Figure 5.4. Example of APEX output information used to deploy ML module

5.1.3 East-/west-bound interfaces

This section captures the east/west bound interactions of the intent engine. These interactions correspond to the exchange of information and request-response messages with the other functional component within the 5G-CLARITY intelligence stratum: the AI engine.

5.1.3.1 Intent Engine → AI engine

The Intent Engine will communicate with the AI Engine through the REST API defined in Section 5.1.1.1. This is predominantly utilised when a ML module is selected as a result of processing a received intent. However, is also used in the scenario where an ML module is the "Intent Provider", that is when the intent requests information specifically generated by an ML module.

When an intent is received, the Intent Engine queries internal context for information related to the functions of ML modules currently available in the AI Engine. If a ML module is associated with the parameters of the intent message, a REST request is sent to invoke the ML module. The response of this request is then used to inform the parameters of the generated action to fulfil the intent request. An example of this output is shown in Figure 5.4.

5.1.3.2 AI Engine → Intent engine

The northbound interface of the Intent Engine is accessible to ML Modules running inside the AI Engine. This allows ML Modules to generate a number of requests informed through interactions with Telemetry to produce a context aware change in the system. Requests generated by ML Modules follow the same process as any request received on the northbound interface.

It is possible to implement a modelled machine-to-machine interface for ML modules to communicate with and the Intent Engine. However, this would contrast the dynamic and flexible building of interactions between the Intent Engine and Intent Providers.

5.2 Intent engine – functional validation

5.2.1 Intent Dashboard

Intent Dashboard will communicate with an Intent Engine. The dashboard will provide fields with default values specific for the Intent Engine. The message field expects Intent Language compliant text detailing the intent. A radio button is provided to specify the syntax (YAML or JSON) used in the message text area. An example of the Intent Dashboard is shown in Figure 5.5.

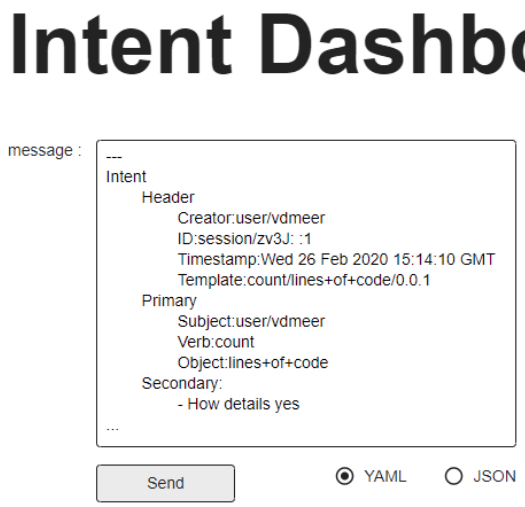


Figure 5.5. Intent dashboard

The Intent Dashboard uses Kafka to send intents to the Intent Engine. When the Send button is triggered a Kafka producers sends the message to the Kafka cluster where it is then forwarded to the Intent Engine. However, as the Intent Engine supports both Kafka and REST interfaces on the consumer side users are free to develop their own mechanisms for sending intents using these interfaces.

5.2.2 Intent Matching

One big challenge in intent-based networking is translating the user generated intent (natural language) into a concrete executable network function. The approach used here is to employ natural language processing (NLP) to match the given intent with available network functions that are registered in the Intent engine. The NLP model determines the semantic similarity between the user input and the functional description of each network function. The functional description can come from the documentation that describes the various available network functions (e.g., through an API description like Swagger[6]) or have been provided manually during the registration of the function in the intent engine.

FastText provides the capability to compare the semantic similarity between the intent message and the description of functions. FastText is an open-source, free, lightweight library that allows users to learn text representations and text classifiers [28]. It is executable on a wide variety of platforms and models can be reduced to fit on small devices.

Opensource pre-trained models are available in a wide variety of languages, the model used in this implementation is trained on English Wikipedia data using fastText. These vectors in dimension 300 were obtained using the skip-gram model described in [29] with default parameters. Word vectors come in both the binary and text default formats of fastText. In the text format, each line contains a word followed by its vector. Each value is space separated. Words are ordered by their frequency in a descending order.

The main principle behind FastText is that the morphological structure of a word carries important information about the meaning of the word. Such structure is not taken into account by traditional word embeddings like Word2Vec, which train a unique word embedding for every individual word. This is especially significant for morphologically rich languages (German, Turkish) in which a single word can have a large number of morphological forms, each of which might occur rarely, thus making it hard to train good word embeddings.

[6] https://swagger.io/

FastText attempts to solve this by treating each word as the aggregation of its subwords. For the sake of simplicity and language-independence, subwords are taken to be the character *ngrams* of the word. The vector for a word is simply taken to be the sum of all vectors of its component *char-ngrams*. FastText can obtain vectors even for out-of-vocabulary (OOV) words, by summing up vectors for its component char-*ngrams*, provided at least one of the *char-ngrams* was present in the training data.

According to a Gensim documentation, fastText does significantly better on syntactic tasks as compared to the original Word2Vec, especially when the size of the training corpus is small. Word2Vec slightly outperforms fastText on semantic tasks though. The differences grow smaller as the size of the training corpus increases.

The primary benefit of fastText in this application is a robust similarity evaluation mechanism that provides consistent results when processing acceptable ambiguity in intent requests

The fastText library modules used in the implementation allow training word embeddings from a training corpus with the additional ability to obtain word vectors for out-of-vocabulary words. This module contains a fast native C implementation of fastText with Python interfaces. It is not only a wrapper around Facebook's implementation. This module supports loading models trained with Facebook's fastText implementation. It also supports continuing training from such models.

Once you have a model, you can access its keyed vectors via the model.wv attributes. The keyed vectors instance is quite powerful: it can perform a wide range of NLP tasks. Intent requests that are forwarded to the system parse the content through a pre-processing function to prepare the data. This data is then compared using n_similarity function of the fastText wordvector.

The output of this system lists a similarity score for each description in comparison to the provided intent. Based on parameters set by the Intent Engine the output can be capped to the top n scores to reduce the size and time need to process the requests.

Figure 5.6 exemplifies the mapping between user-generated intent and the available network functions that are known to the Intent engine. In this (simplified) example, the user wants to create a new network slice. The intent engine has n network functions registered, along with their description and service location (URL). The task of the intent matching module is to find the closest match of function to the user's intent. In this example, the intent "I want to create a slice" is matched to the function/service that is located at http://slice-manager/create.

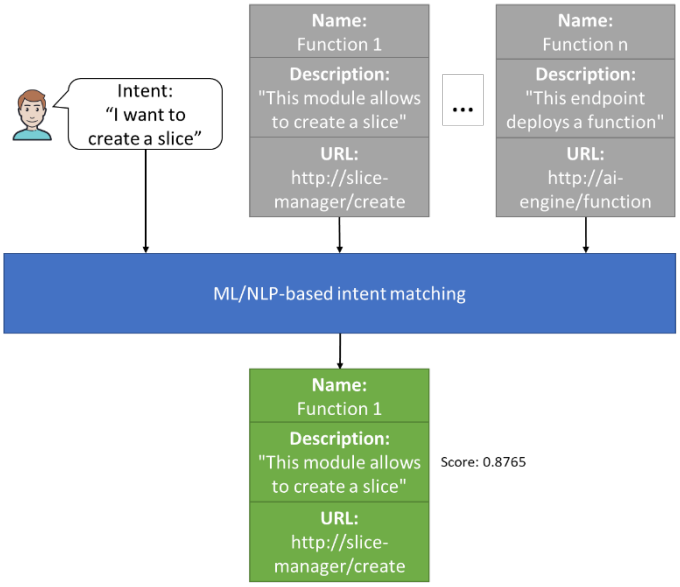


Figure 5.6. Intent matching using ML/NLP

5.3 Target use cases

This section provides information on the targeted use cases of intent-based slice provisioning and telemetry. The slice provisioning and telemetry use cases were chosen for implementation due to the required capabilities aligning with early implementation goals of the intent engine. The slice provisioning use case requires both intent matching and action translation capabilities provided by Natural Language Processing and the functional descriptions of "Intent Providers". The telemetry use case aims to provide more information to the practical interaction between the Intent Engine and Data Lake.

5.3.1 Intent-based slice provisioning

During operation the Intent Engine may need to engage with the Slice Manager as an "Intent Provider". This can be triggered through an intent message requesting functionality of the Slice Manager. On receipt of an intent message the Intent Engine first consults internal functionality templates for the descriptions of functions. When an "Intent Provider" is registered with the Intent Engine it will provide its functionality template. The functionality template allows an action to be associated with the Intent Message through the descriptions of functions. Once the function is identified an internal action builder is instantiated to build the Intent Engine output compliant with the interface of the "Intent Provider".

In this scenario an intent message would request "a new compute node chunk". This will trigger the intent matching stage of the Intent Engine where descriptions are matched with the intent message producing a correlation score. This correlation score is used to identify the Slice Manager function associated with the intent request. The functionality template of the Slice Manager is then used to build the identified function generating an action, in this case Intent Engine would build a REST request of the Slice Manager API. This action is then sent to the Slice Manager.

For testing purposes, we have developed a mock slice manager as an ML module with straightforward endpoints inspired by Slice Manager API documentation. These endpoints are shown in Figure 5.7. Based on this we have developed a use case for intent driven creation of Openstack projects as compute node chunks. An Intent is sent from the Intent Dashboard to the Intent Engine. On receipt of the intent message the Intent Engine triggers a request of all the Functionality Templates stored on the engine. Each description field is extracted and sent with the intent message to Intent Matching. Here text distancing is performed on the collected descriptions returning a score from 0 to 1 based on the similarity of function descriptions to the intent message.


```
{
  "faasresponse": {
    "delete_openstack_project": "This deletes an Openstack compute node chunk",
    "delete_openstack_vlan": "This deletes an Openstack VLAN chunk",
    "endpoints": "This",
    "get_openstack_project": "This retrieves an Openstack compute node chunk",
    "get_openstack_vlan": "This retrieves an Openstack VLAN chunk",
    "post_openstack_project": "This creates an Openstack compute node chunk",
    "post_openstack_vlan": "This creates an Openstack VLAN chunk"
  },
  "name": "module_response",
  "nameSpace": "module.response.events",
  "source": "openfaas",
  "target": "apex",
  "version": "0.0.1"
}
```


Figure 5.7. MySliceManager Mockup

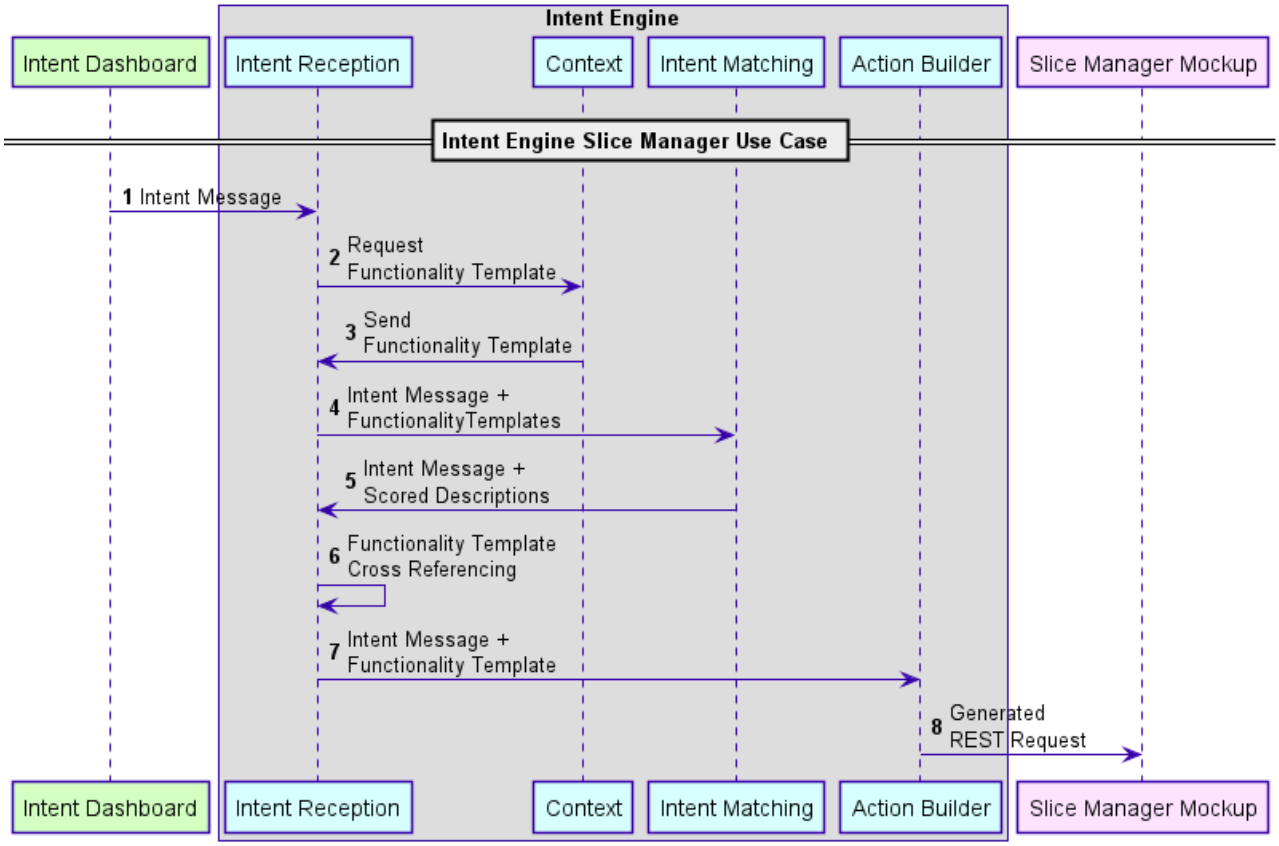


Figure 5.8: Intent Engine Slice Manager Use Case

The highest scoring description is then cross referenced against to Functionality Templates identifying the template and path to the correlated function. This information is then forwarded to the Action Builder which identifies the appropriate interface for the Slice Manager and the mechanism for building the Intent Engine output. The full use case is shown in Figure 5.8.

5.3.2 Telemetry

During operation an instance may occur that requires the Intent Engine to retrieve data from the data lake. This can be triggered in two ways, either through a ML Module requesting data or through an Intent request for specific data.

A ML Module inside the AI Engine may require external data for its execution. In this scenario the ML Module states the information required for its execution to the Intent Engine. At this point there are two mechanisms available to retrieving information from the data lake. The Intent Engine can request the data and pass it directly to the ML module or the Intent Engine can request the location of the required data and pass that to the ML Module which would then be responsible for retrieving the data. The interfaces used in this scenario are described in Section 2.

In a scenario where a data request is triggered through intent explicitly, the Intent Engine will behave similar to if a ML Module had made the request. This results in a request being forwarded to the data lake and the result would be packaged into the response body of the Intent message.

We have developed a temporary data lake mock-up as an ML Module called “MyDataLake” to demonstrate the interaction. “MyDataLake” accepts a key value pair and returns the values requested in the response body. The endpoints of the mock-up are shown in Figure 5.9.

Figure 5.9: MyDataLake Mockup

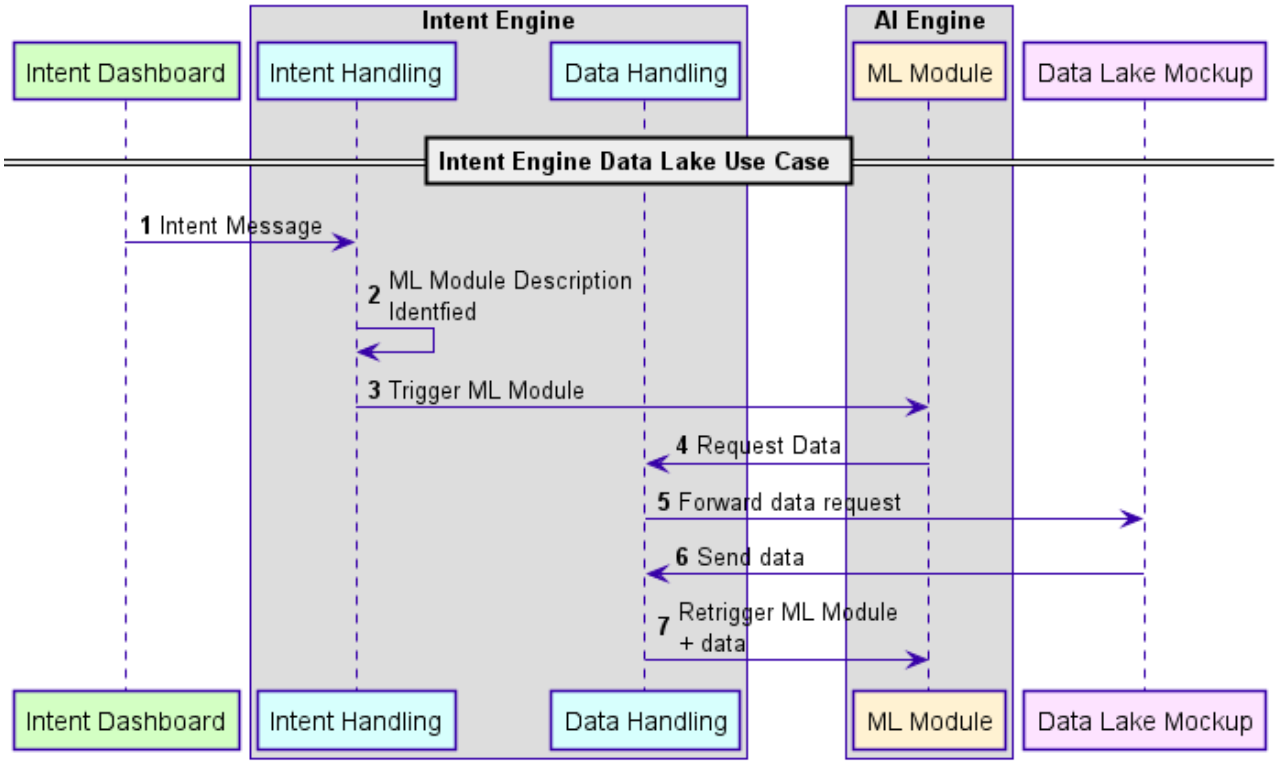


Figure 5.10: Intent Engine Data Lake Use Case

An intent is sent from the Intent Dashboard to the Intent Engine. This triggers the same behaviour described in Section 5.3.1 leading to the identification of the ML Module as the Intent Provider for this scenario. After triggering the ML Module, the Intent Engine is alerted to a request for data from the ML Module. This triggers a pass-through mechanism in the Intent Engine which forwards the data request to the Data Lake Mock-up. When the data is received the Intent Engine retriggers the ML Module with the requested data attached. The full use case is shown in Figure 5.10.

6 Private-Public Network Integration

Private-public network integration is one of the main distinguished features of the 5G-CLARITY system. It represents the ability of the 5G-CLARITY assets (i.e., on-premises resources) to communicate and interwork with the MNO's managed PLMN domain, thereby allowing a seamless operation of PNI-NPNs. The enablers of this feature were first discussed in deliverable D2.2 [2], Section 9, where the project's consortium gave an insight on the integration of public and private networks in 5G-CLARITY system, throughout different deployment scenarios and an analysis on their interaction at the management and orchestration stratum. This integration was also accompanied with security mechanisms, so that privacy and trustworthiness between the corresponding administrative domains can be ensured.

According to the T4.2 objectives committed in the DoW, D4.2 shall capture a first solution design of the enablers allowing for this public-private network integration. For the activities with regards to these enablers, three workstreams have been defined: *i)* management capability exposure; *ii)* public-private network connectivity, and *iii)* Intelligent stratum integration. The initial outcomes from these workstreams are captured in the different subsections defined in this chapter.

6.1 Management Capability Exposure – Initial solution design

Management capability exposure can be defined as the ability of a NOP to securely expose capabilities from their managed functions towards one or more authorized tenants. This mechanism provides means to establish a clear demarcation point between a private NOP (i.e., 5G-CLARITY system operator) and public NOP (e.g., MNO), defining their individual management scope in the operation of PNI-NPN. For a fine-grained control of this exposure, 5G-CLARITY system leverages the use of Mediation Function (Section 6.1.1). The token-based authentication and auditability features provided by the 5G-CLARITY mediation function allows for the definition of different service delivery models, each adapted to the specificities of each use case (Section 6.1.2).

6.1.1 Mediation Function

As described in deliverable D2.2 [2], the Mediation Function provides a single-entry point to the 5G-CLARITY Management and Orchestration stratum for external consumers. These consumers include *i)* MFs from the 3GPP management system of a public NOP; and *ii)* services from the intelligence stratum, when hosted off-premises (e.g., 3rd party cloud node). From a conceptual viewpoint, the mediation function acts as a combination of Unified Data Repository (UDR) and Network Exposure Function (NEF) functionalities as defined in the 5GC, policing what external consumers are allowed to access in each MF service defined in the 5G-CLARITY Management and Orchestration stratum, and how much information and capabilities they can get from this MF service, using its exposed APIs.

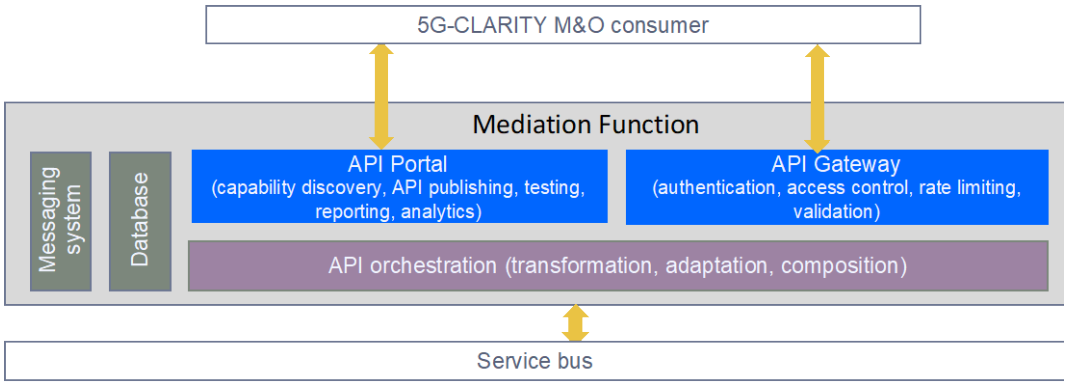


Figure 6.1. 5G-CLARITY mediation function

Figure 6.1 illustrates the initial design of the 5G-CLARITY mediation function. In this first design, the following internal modules are considered:

- **API Gateway (mandatory)**, which is the front-facing service for 5G-CLARITY management and orchestration stratum, enforcing policies and access control between MFs and external consumers. As the entrance of the mediation function, all requests shall go through the API gateway to the specific MF service.
- **API Portal (mandatory),** which has an informative role for external consumers. The portal describes what APIs are available for usage, listing them all and providing a description for their consumption: API endpoint (e.g., IP address, Fully Qualified Domain Name [FQDN]), API lifecycle information, eligibility to be the consumer of the API, API health insights (e.g., real-time monitoring), etc. The documentation on the portal should also provide the authentication and authorization mechanism, use cases that describe the business context and live real implementations.
- **API orchestration (optional)**, which is a MF service responsible for consuming service bus exposed APIs (i.e., APIs offered by the different 5G-CLARITY management and orchestrated MFs) and apply transformation operations on them, if needed (e.g., for making them more user-friendly for some tenants, for adapting them to the intelligence stratum). Notice this transformation is optional – its application is dependent on use case. Examples of this transformation include stage 3 translation (e.g. XML↔JSON, SOAP↔REST) and protocol transformation (e.g. HTTP→HTTPs). Once transformed, these APIs can be securely exposed through the API gateway.
- **Supporting services,** which are MF services that support the operation of API gateways and API portal. On the one hand, there is a database, in charge of keeping a registry with available and published APIs together with their endpoints. On the other hand, there is the message system, which allows the exchange of internal messages on the 5G-CLARITY mediation function.

Figure 6.2 illustrates the user story of the MF logic, showing an end-to-end consumer-to-API flow example passing through the API gateway, where the main policies and the access control are captured.

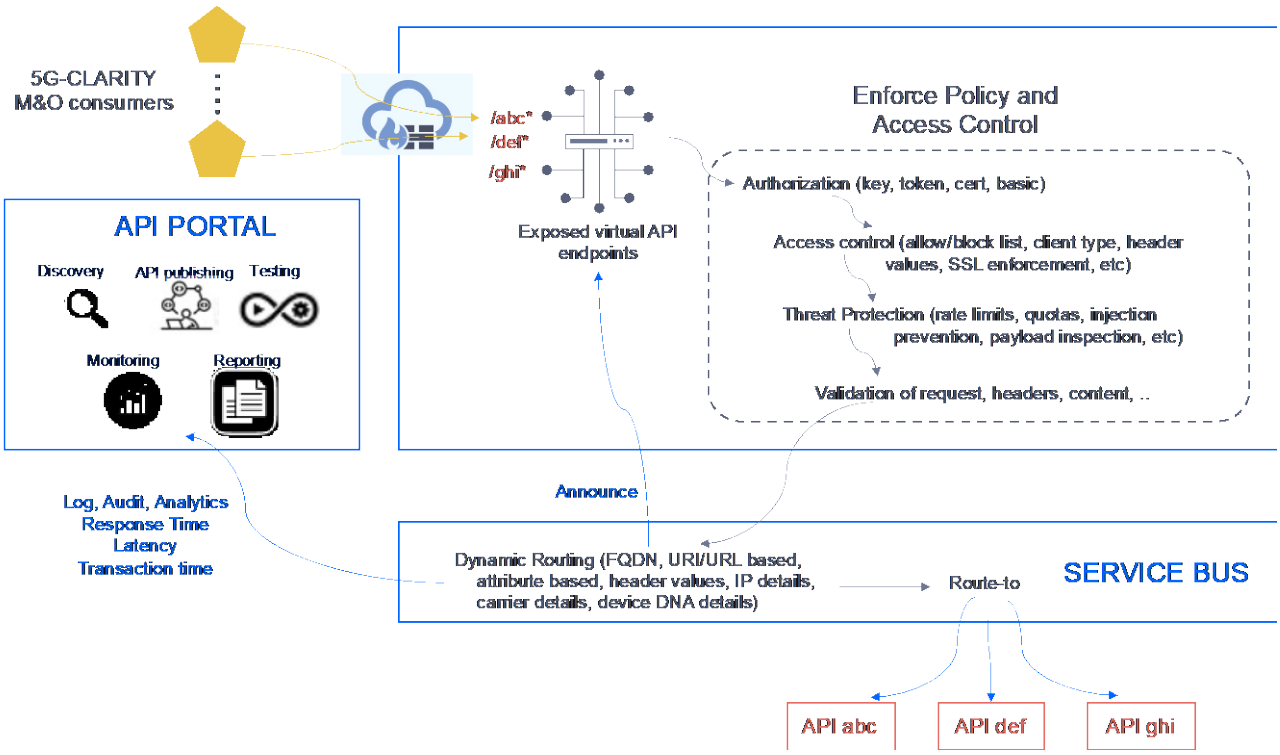


Figure 6.2. 5G-CLARITY Mediation Function - user story

```
{
    ...
    " tok_type " : "AT",
    "user_id" : "testuser",
    "user_tenantname" : "<value-of-identity-tenant>"
    "tenant" : "<value-of-resource-tenant>"
    "client_id" : "testclient",
    "client_tenantname": "<value-of-client-tenant>"
  ...
}
```

Figure 6.3. Access token

The mediation function is responsible for enabling multi-tenancy support, which is an inherent feature of the 5G-CLARITY system architecture. This feature is based on the enforcement of two functionalities: capability exposure (5G-CLARITY D2.2 [2],Section 9.3.1) and auditability (5G-CLARITY D2.2 [2], Section 9.3.2).

On the one hand, capability exposure allows the mediation function to define the degree of control a 5G-CLARITY customer (e.g., a public NOP) can take over a 5G-CLARITY managed instance, such as a PNI-NPN. For a fine-grained control of this capability exposure, a token-based authentication framework is used. This framework is based on the definition of two token types, each used for codifying a different tenancy type: **URL token**, which identifies the tenancy of the application that requests a service; and **access token**, which codifies the tenancy that is the target of such access (e.g., application tenancy) as well as the user tenancy that is given access. An access token includes at least a claim/statement indicating the resource tenant name and the time the request for the access token was made (e.g., the customer), a claim/statement indicating the user tenant name, a claim/statement indicating the name of the OAuth client making the request, and a claim/statement indicating the client tenant name. Figure 6.3 captures an example of a how access token can be implemented following JSON functionality.

Based on the above rationale, when the 5G-CLARITY tenant registers into the private NOP admin domain for the first time using single sign-on (SSO) functionality, the tenant is granted with a unique access token. This token, provided by the API gateway, specifies the set of APIs that the tenant can consume at operation time. With this token in hand, the workflow is as follows: every time the tenant invokes an API from a MF to consume a given MF service, the MF checks the permissions imbued in the access token. If these permissions include the requested management service, then the MF authorizes the API consumption.

On the other hand, auditing in a multi-tenant environment presents a number of challenges that broadly relate to providing individual tenants with appropriate visibility to audit information. One problem is that audit events frequently are not easily traceable back to individual tenants. Another problem is that audit logs are not easily disseminated to individual tenants. Moreover, the typical manner in which audit logs are generated and stored does not support the ability to prove that tenant information is compartmentalized. One approach to address these problems involves augmenting audit APIs in a cloud operational environment (either virtualized or cloud-native) so that logs are annotated with an identifier for each tenant. This approach, while technically feasible and useful, requires changes to software components in the cloud environment to enable them to take advantage of these audit services. The high development cost and change management impact may make this approach less competitive in terms of time-to-market, which has a direct impact on the short-/mid- term validity of 5G-CLARITY solution.

Tthere remains a need to provide a multi-tenant audit solution that enables 5G-CLARITY mediation function to provide audit trails with a single tenant audit view and that sufficient proof that audit information from the tenant is not being leaked between or across tenants. This solution shall allow the generation of clear, original (unchanged), verifiable and accurate audit trails for the interplay of private NOP with the public NOP, ensuring traceable and secure interactions across them, at both orchestration and control levels. The messages exchanged between these two administrative domains, including request-response and subscribe-notify messages, need to be logged with accurate timestamps and support non-repudiation.

Table 6-1. Example of 5G-CLARITY Audit Trail

Audit Trail Field	Description
action	The type of operation performed (e.g., request-response, notify-subscribe-unsubscribe)
callSize	The size of the call
callerId	A unique identifier of the caller
callerIpAddress	The IP address of the caller
beginTime	Data time of when the operation started (ISO format)
endTime	Date time of when the operation ended (ISO format)
httpMethod	The HTTP method: [GET/DELETE/PUT/PATCH] for request-response type action, and POST/DELETE for notify-subscribe-unsubscribe type action
httpStatus	The HTTP status
httpUrl	The HTTP URL called
resourceId	A unique identifier of the resource accessed
resourceClass	The class of the resource accessed (e.g., if resource accessed corresponds to a 5G-CLARITY compute service, then resourceClass points to the corresponding NSD).

In 5G-CLARITY system, we propose a solution based on storing audit trails on a centralized, SQL database on every administrative domain involved. Table 6-1 captures an example on the schema of a potential audit trail. These audit trails may be fetched and checked by corresponding administrators, which might be the NOPs themselves (e.g., use audit trails to monitor and gather data about specific database activities, detect security breaches), or even a 3rd party (e.g., uses audit trails to address auditing-related requirements for compliance, such as the European Union Directive on Privacy and Electronic Communications). To illustrate the use of the Mediation Function, we next provide, as an example, an operation that is executed in a multi-domain scenario where the NFVIaaS model is used. Specifically, the two administrative domains correspond to an MNO and a private operator relying on the 5G-CLARITY architecture. Further details of this operation can be found in [69], as well as in Section 6.1.2 of this document. This operation supports the orchestration of 5G-CLARITY compute services by an external organization. To achieve this, the ETSI NFV MANO stack is deployed in both domains and interconnected. The interaction between the two entities is supported by the Mediation Function, which acts as a proxy to manage the operations.

Figure 6.4 shows the process of a composite NS instantiation across the two administrative domains. As explained in Section 6.1.2, a composite NS is composed of multiple nested NSs, which can be deployed in different administrative domains. First, the 5G-CLARITY's NFVO registers their services in the Mediation Function. To consume such services, the MNO's NFVO interacts with the Mediation Function which carries out the authentication and authorization process and generates the access token that is delivered to the MNO. Suppose that the MNO wants to create an instance of a given composite NS. The MNO's OSS sends the creation request to its NFVO, where it is processed. Then, the MNO's OSS sends the instantiation request to its NFVO. The NFVO finds the corresponding NFVO (i.e., the 5G-CLARITY's NFVO) from the mapping <NSD, NFVO> and sends the request for creating a nested NS to the Mediation Function in the domain of the private operator. The authorized request is transferred to the NFVO in this domain and logged in the audit database. Note that the audit database may belong to any of the participant domains. A similar process is applicable to the instantiation request for the nested NS. In case of successful nested NS instantiation, the 5G-CLARITY's NFVO sends a notification to the Mediation Function, which transfers it to the MNO's NFVO and logs it in the audit database. If the composite NS comprises VNFs as part of it, the NFVO performs the VNF instantiation procedure and connects the constituent VNF instances and nested NS instances. Finally, in case of successful instantiation for the composite NS, a notification is delivered to the MNO's OSS.

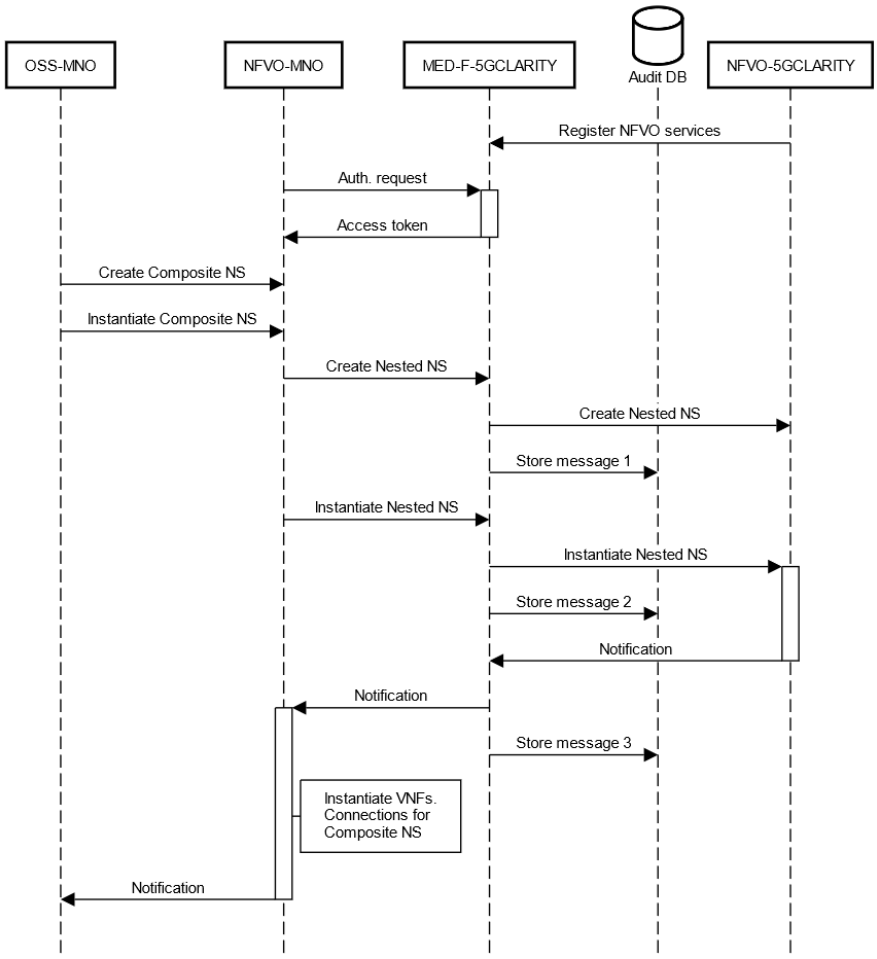


Figure 6.4. NFVIaaS-related operation through the Mediation Function (MED-F).

6.1.2 Service delivery models

The aim of this section is to discuss how the service delivery models introduced in 5G-CLARITY D2.2 and the associated management models (MMs) described in 5G-CLARITY D4.1 can be applied to the use cases, UC1 and UC2.1, presented in 5G-CLARITY D5.1 [70]. The MMs represent different ways of managing the exposed services according to the service delivery models, WATaaS, NFVIaaS and SlaaS. The logic behind these MMs is that higher levels of management provide the customer with more advanced management capabilities, including those capabilities offered by lower levels. The interested reader is referred to 5G-CLARITY D4.1 for a better understanding of the definition and notation used for each defined 5G-CLARITY MM.

The MMs can be applied in a different way depending on the deployment scenario and the potential interactions between public and private operators. In 5G-CLARITY D5.1 [70], an introduction is provided to the Smart Tourism and Industry 4.0 use cases, where the services can be deployed over standalone NPN or PNI-NPN. In the latter approach, the role of the public operator is key to support these services. By following this approach, the Smart Tourism pilot (UC1) would benefit from the multi-tenancy support, enabling on-demand services such as public safety systems and third-party special events (e.g., conferences, seminars, etc.). In the case of the industry 4.0 pilot (UC2), the PNI-NPN approach would improve the in-factory connectivity relying on network infrastructure (e.g., the 5G core) provided by the public operator.

Next, we specifically analyse the benefits and limitations of applying the different service delivered models and MMs to each use case. In addition, some guidelines are provided for future experiments in the pilots.

6.1.2.1 UC1: Smart Tourism

5G-CLARITY UC1 aims at enabling enhanced human-robot interaction as-well-as providing infrastructure slicing for third party services such as public safety and private events production. The use case will demonstrate the benefits of 5G-CLARITY Framework which combines E2E slicing, multi-WAT, and the PNI-NPN support. Three narratives are envisioned to be deployed, a multi-WAT on a standalone NPN deployed to support human-robot interaction and two PNI-NPN scenarios supported by E2E slicing. The UC1 deployment might support NFVIaaS, WATaaS, and SLaaS delivery models with 5G-CLARITY Infrastructure stratum.

NVFIaaS service delivery model: In the UC1, demos for narratives 1 (standalone NPN), 2 and 3 (PNI-NPN) may allow three MMs to create, instantiate, and onboard VNFs for nested NS between an MNO and the private NOP. Figure 6.5 introduces the overview of the NFVIaaS delivery model for a nested NS considering UC1 narratives 2 and 3. The MM applicable here are the following:

- NFVI.MM1, with limited access to telemetry for FCAPS only allowing the MNO or tenants to upload the software image and VNFD/NSD for the UPF to a repository in the 5G-CLARITY edge cluster
- NFVI.MM2, which allows the capability to create VNF/NS by requiring a connection between private NOP's NFVO (M-Shed Museum) and MNO NFVO (5GUK Test Network). In this example the VNFD/NSD can be used to host UPF
- NFVI.MM3, which includes secure connection between the private NOP's VIM through the MNO's NFVO to manage the VNF/NS lifecycle and control the 5G-CLARITY compute quotas to allocate additional functions in case the performance of the real time video streaming requires to maintain the 360-degree camera or guide robot APIs (e.g., UC1 Narrative 2). Finally, the NFVO on the private NOP will create the VNF to onboard UPF and the VIM will allow the NS cycle to be controlled by the MNO NFVO.

WATaaS service delivery model: As NFIaaS, WATaaS can be allocated to setup the MNO wireless coverage for visitors. To do thiss, the private NOP and MNO will setup a connecting through a gateway to a dedicated VLAN or network slice (defined by a set of S-NSSAIs) associated to WAT (Wi-Fi and/or Li-Fi) and/or 5G-RAN-functions of the 5G-CLARITY RAN cluster. This service delivery requires WAT.MM1 over an infrastructure slice. Figure 6.6(a) introduces the setup planned for the UC1 in which a standalone 5GC hosted by the private NOP is used to provide or extend WAT services of an MNO. In this example the WAT.MM1 (OM-SL: MNO manager with private NOP slice manager) will generate a request from SL-No RT-RIC to allocate S-NSSAIs (VLAN-ID, PLMN ID, and SSIDs) to the WAT infrastructure of the Museum. This example can be replicated in standalone NPN or PNI-NPN deployed in Airports, Malls, and other private premises to enhance WAT coverage of MNOs.

SLaaS service delivery model: In the UC1 narratives 2 and 3 the management SL.MM1, 2, and 3 are provided to the surveillance services and special event services in coordination with the MNO, both requiring low latency and high throughput video and content delivery. Figure Figure 6.6(b) presents the SL.MM1 (WAT.MM1 exposure (S-NSSAI) and NFVI.MM1 exposure) that must be enabled in the UC1 narratives 2 and 3 deployments to create or onboard VNFs (NSSAI LCM + VNFD/NSD). In the case, UC1 narrative 2 for surveillance systems requires the control over the NS lifecycle, as a result SL.MM2 and SL.MM3 exposure must be ensured. For the case of the special event provider of narrative 3 the SL.MM1 might be enough. As a results SLaaS presented will be the next level WATaaS in which MNOs and Service providers can slice larger capacity of the NPN infrastructure through infrastructure slices.

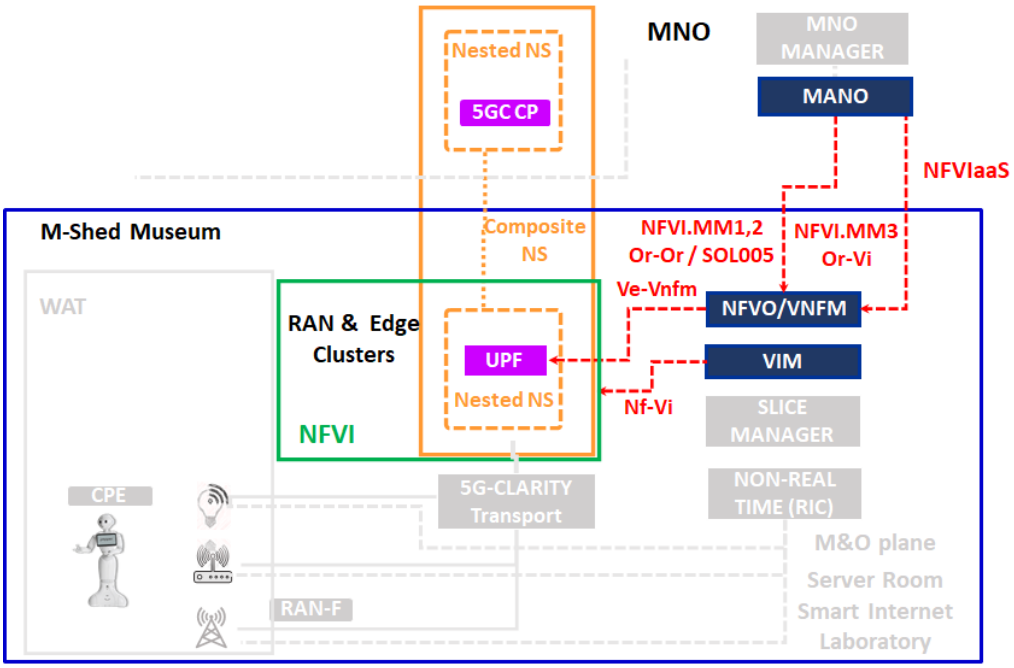


Figure 6.5. NFVIaaS in UC1.

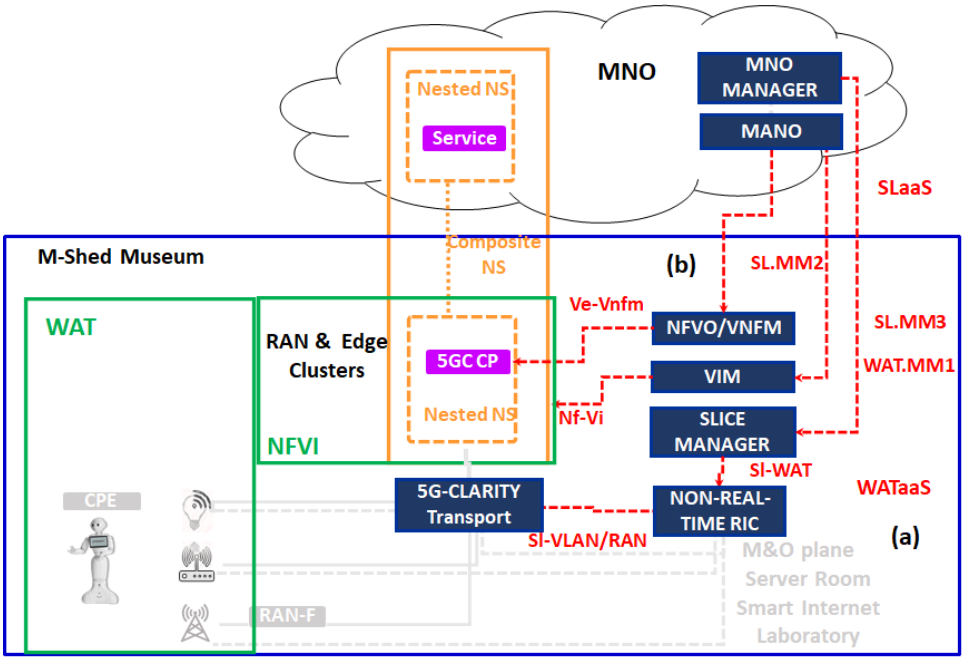


Figure 6.6. WATaaS and SLaaS in UC1.

6.1.2.2 UC2: Industry 4.0

5G-CLARITY UC2 aims at improving the in-factory connectivity, replacing current Ethernet wired connections by an innovative multi-WAT environment. One of the envisioned deployment scenarios consists of a PNI-NPN based on a PLMN-provided 5GC and public SIMs for 5G-CLARITY CPEs. This scenario may reduce the entry barrier for some verticals interested on deploying NPNs. Depending on the distribution of NFs between the public and private administrative domains, some service delivery models are more appropriate than others for the provision of private services. Next, we analyse the implementation of the 5GC in this scenario where a specific service delivery model and associated MMs can be applied. One of the main challenges in

5G-CLARITY UC2 is the need to cope with the stringent requirements of URLLC services. In this case, the UPF is deployed inside the private venue to process delay-sensitive traffic, while the 5GC-CP resides in the MNO infrastructure footprint. The NFV orchestrator on the MNO side has to instantiate the VNF that executes the UPF. To do this, the NFVIaaS model is the appropriate service delivery model whereby a virtual infrastructure is made available in the form of dedicated compute resource quotas for VNF hosting.

As shown in Figure 6.7, the UPF is deployed on the 5G-CLARITY edge cluster located in the technical room. This network function aims at processing the traffic coming from each production line. To limit the resources available to each UPF instance (there can be several production lines) and to provide the required isolation, 5G-CLARITY compute quotas are assigned to the Virtual Deployment Units (VDUs) hosting the UPF instances. The UPF is kept separated from the 5GC CP, which remains on the MNO side.

While the role of the private NOP is to configure the access nodes to support the 5G-CLARITY services and establish wireless quotas, the MNO is responsible for instantiating the UPF(s) on the edge cluster following the NFVIaaS model. There are three different MM levels, referred to as NFVI.MM1/2/3, which can be applied in this scenario depending on the desired capability exposure. The NFVI.MM1 includes limited access to telemetry for FCAPS and it allows the MNO to upload the software image and VNFD/NSD for the UPF to a repository in the edge cluster. The NFVI.MM2 extends this capability exposure by enabling the management of the lifecycle of VNF/NS. This MM level requires a connection to the private NOP's NFVO so that, for example, the VNFD/NSD for the UPF can be on boarded. Lastly, the NFVI.MM3 includes a connection to the private NOP's VIM through the MNO's NFVO to have increased capability to manage the VNF/NS lifecycle and control the 5G-CLARITY compute quotas assigned to the different production lines.

Assuming that the private NOP is interested in outsourcing the deployment, management and maintenance of the 5GC, the main advantage of the NFVIaaS model is that the UPF can be deployed on premise to support time-critical services. The implementation of the interface between public and private domains would be relatively simple since there are standardized solutions as discussed in Annex A. Specifically, based on the existing alternatives for federated MANO-based orchestration, we next provide an analysis of the 5G-CLARITY UC2 to support the deployment of private services across different administrative domains. Since the management entity that handles the 5G-CLARITY compute services is the ETSI NFV MANO, we can assume the same component on the MNO side to establish the peering connection. Figure 6.8 illustrates the deployment scheme of UC2, where the UPF is deployed inside the private venue to process the traffic of delay-sensitive services and the 5GC CP is kept in the MNO infrastructure footprint.

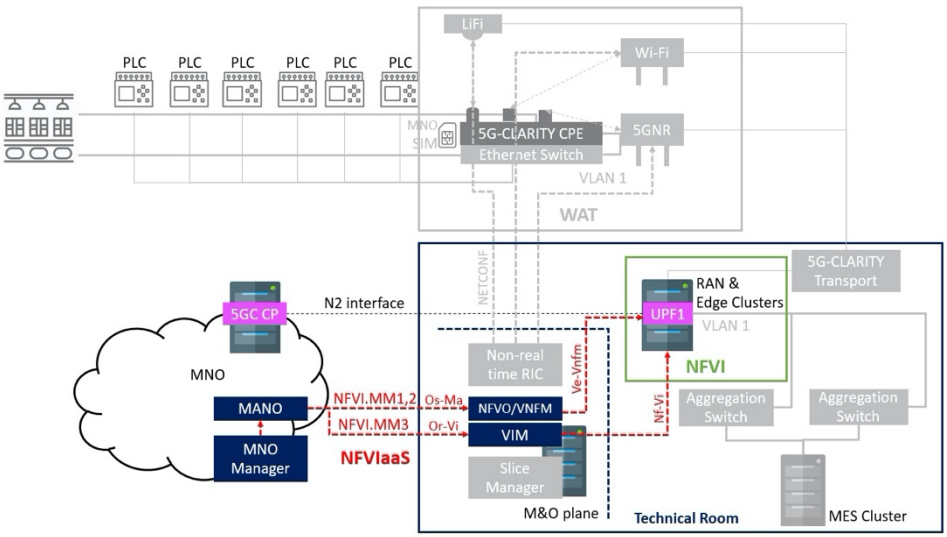


Figure 6.7. NFVIaaS in UC2.

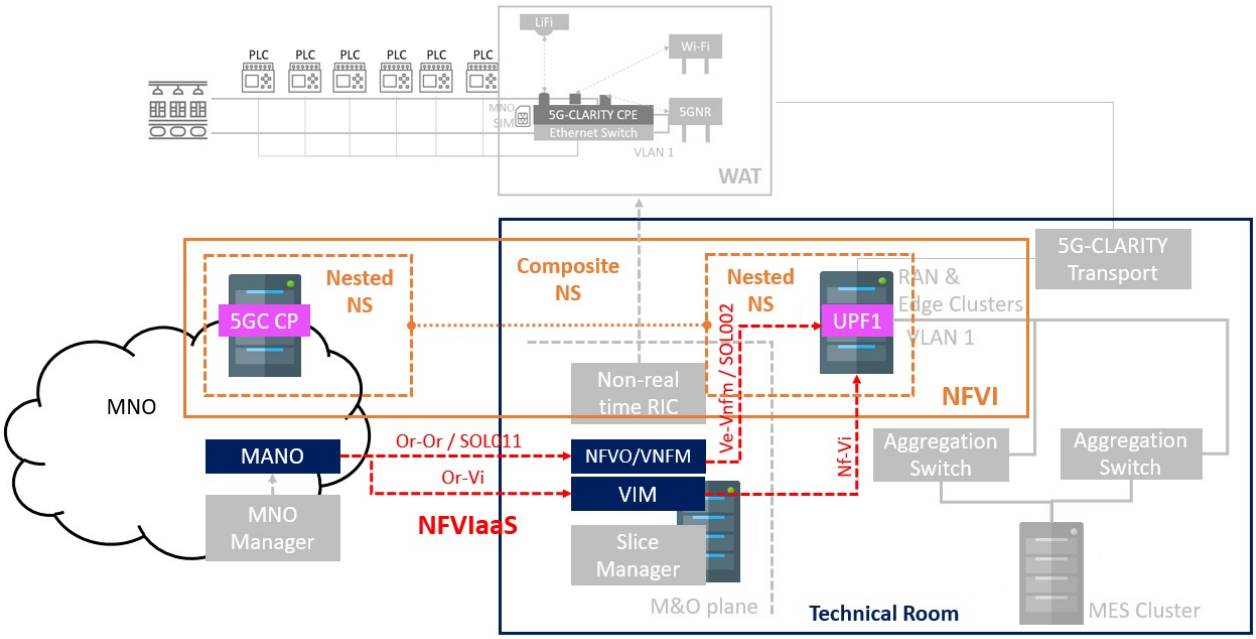


Figure 6.8. The composite NS for supporting delay-sensitive services in UC2

The proposed solution relies on the ETSI ISG NFV framework and consists of a composite NS involving multiple administrative domains. Specifically, the composite NS representing the 5GC is composed of two nested NS, namely the 5GC-CP and the UPF. The MNO may request the deployment of the nested NS in the 5G-CLARITY domain (i.e., the UPF) as part of the composite NS when delay-sensitive services are required. In addition, it is necessary to create the virtual networks to enable the virtual links between the nested NS. To support these operations using the NFVI.MM2/3 of the 5G-CLARITY NFVIaaS model, the NFVOs at the two administrative domains can be interconnected through the Or-Or interface. In case of NFVI.MM3 is required, for simplicity, the connection to the VIM at the private premises can rely on a single logical point of contact.

The proposed scheme is appropriate when the MNO infrastructure footprint and the 5G-CLARITY premises are deployed in remote NFVI points-of-presences (PoPs) and interconnected through a WAN infrastructure. In this case, the management of an NS that is deployed over multiple PoPs may negatively impact its performance. For example, delays in the WAN segment may degrade the monitoring of metrics/KPIs if they cannot be delivered on time to the NFVO. Consequently, lifecycle management decisions would not be properly taken, and related actions would also be delayed. For this reason, the adopted solution distributes the functions of the management plane in such a way that each administrative domain handles its own ETSI NFV MANO stack and the coordination between the NFVOs takes places via the Or-Or reference point [71], with SOL011 [72] as protocol interface solution. Nevertheless, the set of federation-related operations that can be executed may be limited by the latency requirements of the scenario.

Solutions for NFVO-NFVO interaction is a problem that has been discussed over the past years. Annex A provides a summary of the takeaways of these discussions, in both standards and research projects. In 5G-CLARITY system, where we have OSM as NFVO solution, this scenario can be achieved using the federation capabilities available from OSM Release SEVEN, as captured in Figure 6.9 [73]. We can leverage this solution to implement MANO federation for the in-project use cases, as follows:

- Higher layer OSM is part of the public NOP's MANO stack. This NFVO instance is deployed in MNO's footprint, typically in a central cloud node.
- Lower layer OSM is part of the private NOP's MANO stack. This NFVO instance is deployed in 5G-CLARITY on-premises infrastructure. As seen, 5G RAN and 5G MEC nodes represented in Figure 6.9 correspond to RAN cluster and edge cluster nodes in the 5G-CLARITY system.

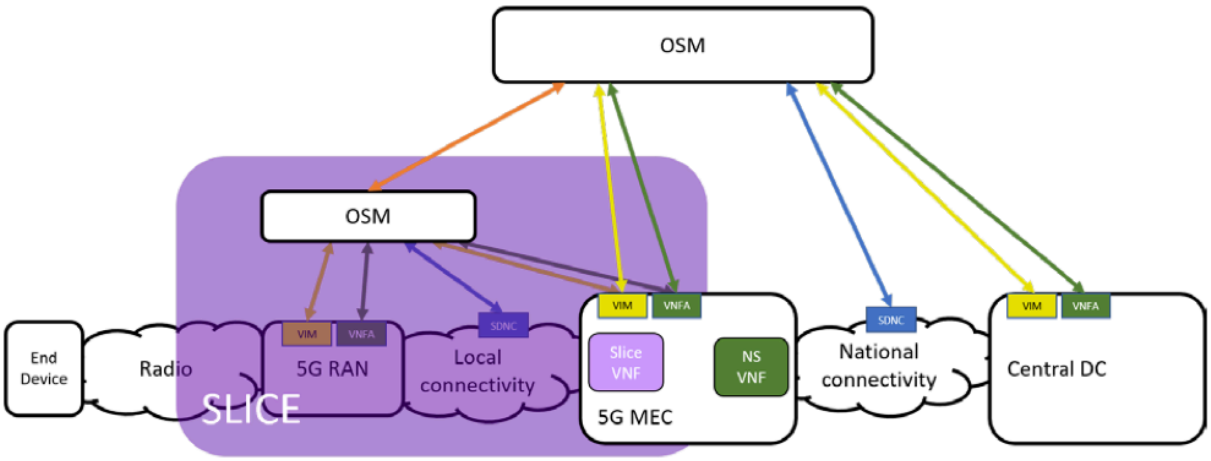


Figure 6.9. MANO federation using OSM (source: [73]).

6.2 Public-private network connectivity – initial solution design

Archetypal PNI-NPNs deployments are based on the distribution of non-public VxFs across two (or more) administrative domains, where at least one of these administrative domains is the PLMN. PNI-NPN scenarios in 5G-CLARITY are compliant with this layout, with some VxFs executed on the 5G-CLARITY node (private premises), and the rest hosted by an PLMN node, either central office (telco edge node) or data center (telco regional node) or both. To allow conveying the data, control and management traffic between the 5G-CLARITY and PLMN nodes hosting the non-public VxFs, data networking services based on WAN technologies need to be set up. The provider of these connectivity services is typically the public NOP (e.g., MNO), with private NOP being the customer.

This section provides an overview of the different WAN connectivity services that are valid to 5G-CLARITY, specifying their main features as well as fields of applicability (Section 6.2.1), and providing a comparative analysis among them (Section 6.2.2). This overview is complemented with a real-world example that illustrates the problem of realizing E2E connectivity when public and private administrative domains are involved (Section 6.2.3), which in 5G-CLARITY consists in translating on-premises VLAN solution to WAN connectivity solutions.

6.2.1 Technology solutions

IPSec VPN

IPsec Virtual Private Network (VPN) securely connects a number of a sites the same private network using Internet connectivity as the data communications network. This type of VPN is deployed between a security appliance or firewall at each location, ensuring a secure IPsec tunnel between sites. The LAN sits behind these security devices and software isn't required on laptops, desktops, or servers to enable VPN connectivity between locations. VPN network topologies are available in a hub and spoke or meshed configuration.

The main benefits of these types of data networking services are cost, the ability to use existing Internet connectivity for data transport, and easy integration of remote users with VPN software. IPsec VPN connectivity does have its flaws in that the QoS is not consistent due to Internet network congestion or poor performance. Also, there is increased potential for network downtime if only using one Internet connection with no failover connectivity. IPsec VPN networks are a good choice for customers with limited IT budgets, many remote users, or basic applications and uptime requirements.

Software-Defined WAN (SD-WAN)

SD-WAN is an emerging type of WAN technology that leverages the SDN principles of control-user plane separation to automatically determines the best routers to and from locations over Internet connections and private data networks. SD-WAN creates tunnels that are transport-agnostic, so public NOP can use Internet connections like DSL, cable, wireless, shared fiber, or dedicated connectivity. Private NOP can also keep existing private data network services (MPLS, EPL, EVPL, VPLS, etc) in addition to regular Internet connectivity. This helps to improve SD-WAN network performance and reliability.

SD-WAN uses multiple tunnels to increase and optimize WAN bandwidth between different types of WAN technologies, a big advantage over traditional IPsec VPNs. This ensures applications have the highest QoS, increased WAN speeds, as well as additional network redundancy and failover. SD-WAN centralizes network control and traffic management over these links through a centralized controller or orchestrator. For more information on SD-WAN, see [74].

VPN security is layered on top, while SDN software enables IT staff to remotely manage network edge devices and applications more easily. SD-WAN is a good option for businesses of all sizes and needs. Enabling data networking over low-cost Internet connectivity, as well as more expensive dedicated WAN links, is a big plus. Increased reliability, performance, network agility are all key features of SD-WAN service, along with a competitive price point.

Metro Ethernet

Metro Ethernet is a point-to-point Ethernet data networking service connecting locations within a metropolitan area (MAN). Ethernet over Synchronous Optical Network (SONET) technology is used for secure point to point WAN connectivity. Circuit speeds typically range from 10 Mbps to 10 Gbps, with 100Gbps available in some metropolitan areas.

Provider networks are Layer 2, so you have control over addressing and routing. Metro Ethernet service is ideal for businesses with two or more locations in a metro area that need high bandwidth connectivity with QoS requirements. In most cases, average costs for Metro Ethernet service tend to be low due to minimal distances and limited network infrastructure used to provide service.

Ethernet Private Line (EPL)

EPL provides dedicated point-to-point Ethernet network connectivity between two or more locations. Like Metro Ethernet, Ethernet over Synchronous Optical Network (SONET) is the type of WAN technology used. EPL circuits provide a reliable data networking service for customers with high bandwidth and low latency needs. A key component of EPL service is network resiliency and performance through SONET protection (network reroute).

Being a Layer 2 network, addressing and routing is customer controlled. Ethernet Private Line is available from 10 Mbps to 10Gbps, with 100Gbps available in some locations. EPL is one of the more expensive types of WAN technologies due to distance-sensitive pricing and dedicated network infrastructure used.

MPLS VPN

MPLS VPN is a virtual private network built on top of a provider's Multiprotocol Label Switching Network to provide Layer 2 or Layer 3 VPN data networking services. The topology configurations available include site to site, multipoint, and meshed networks. Customer data is partitioned from each other, keeping it private across the provider's infrastructure. Data partitioning is created using MPLS tags rather than encryption.

MPLS is different from other VPN data networking services due to the fact that you can prioritize traffic types over the MPLS provider network. This allows control over application performance (low to high QoS). MPLS

circuit speeds typically range from 10 Mbps to 10Gbps, with costs similar to dedicated Internet connectivity. MPLS networks are the current industry standard for a private data networking service, due to its superior performance, reliability, flexibility, and competitive pricing.

Ethernet Virtual Private Line (EVPL)

Also referred to as E-Line, EVPL provides point-to-multipoint connectivity over a provider's MPLS network. EVPL uses Ethernet Virtual Connections (EVCs) to connect multiple locations together, as well as multiple services on a single User-to-Network Interface (UNI) at the host or hub site.

EVPL is a Layer 2 data networking service utilizing MPLS tags and supporting multiple classes of service (CoS) for low to high QoS applications. Ethernet Virtual Private Lines are available from 10 Mbps to 10 Gbps. EVPL is ideal for customers looking for a reliable type of data communications network for a hub site to multiple remote locations. Due to shared network infrastructure and limited distance costs, EVPL pricing is not as expensive as EPL.

Virtual Private LAN Service (VPLS)

Also referred to as E-LAN, a VPLS is a data network service for multiple sites in a single bridged domain over a provider managed MPLS network. All sites on a VPLS network will appear to be on the same LAN, regardless of the location. Like EVPL, it is a Layer 2 type of data communications network that utilizes MPLS tags. VPLS also supports multiple classes of service (CoS) for low to high QoS application needs. Multiple types of WAN technologies (MPLS VPN, Internet, EVPL) are supported on a single port and circuit.

Routing and management of the VPLS network can be done by the customer or provider. VPLS networks offer the ability of a meshed network config (any to any), so all sites can communicate with each other, increasing network continuity. VPLS is ideal for customers looking for reliable connectivity for a hub site to many remote locations. VPLS speeds are usually 10Mbps to 10Gbps with pricing comparable to EVPL and MPLS data networking services.

Wavelengths

Wavelength Service is an optical data networking solution for customers requiring very large, dedicated point-to-point data connections. This is ideal for business continuity, data center replication, backup solutions, streaming media, or very large data transfers. Applications that require low latency and high-speed connectivity are ideal for this type of WAN technology.

Speeds typically available are 2.5Gbps, 10Gbps, 40Gbps, 100Gbps delivered as an optical handoff. Wavelength service is provisioned over a Dense Wave Division Multiplexing (DWDM) network, providing full Layer 1 transparency and management. Unprotected and protected network reroute is available to ensure the resiliency of data network connectivity. Wavelength service has the lowest per Gbps cost of all data networking services, but an overall higher price point due to large bandwidth sizes.

6.2.2 Comparative analysis

As seen above, the selection of one or another solution depends on the targeted goal (e.g., network performance, reliability, security, price, application use case), which can be a bit overwhelming. To clarify their pros and cons, and therefore make the decision easier for public NOP, we provide a comparative analysis of these solutions in Table 6-2.

Table 6-2. Data Networking Services for Data Plane Connectivity in PNI-NPN Scenarios

Solution	Topology	OSI	Technology	Underlay	QoS	Cost (per BW unit)

IPSec	PtP[7], MP[8], Mesh	Layer 3	IP	Shared	Low	Low
SD-WAN	PtP, MP, Mesh	Layer 3-7	SDN	Shared	Low-Mid	Low-Mid
Metro Ethernet	PtP	Layer 2	SONET	Dedicated	High	Low-Mid
EPL	PtP	Layer 2	SONET	Dedicated	High	High
MPLS VPN	PtP, MP, Mesh	Layer 2-3	MPLS	Shared	Low-Mid	Mid-High
EVPL	PtP, MP	Layer 2	MPLS	Shared	Low-High	Mid-High
VPLS	PtP, MP, Mesh	Layer 2	MPLS	Shared	Low-High	Mid-High
Wavelength	PtP	Layer 1	DWDM	Dedicated	High	Low

6.2.3 VLAN vs WAN data plane connectivity with 5G-CLARITY framework

Both the 5G-CLARITY Infrastructure stratum as well Management and Orchestration stratum will enable slicing and orchestration through different administrative domains. Virtual local area network (VLANs) protocols e.g., IEEE 802.1q play an important role in the slicing of NPN and MNO transport network. A 5G-CLARITY slice integrates VLANs to isolate traffic and apply the appropriated QoS rules. Enabling PNI-NPN scenarios a private or public gateway will be used to interconnect NPN with an MNO through a public NOO. Following the comparative technologies introduced on Table 6-2, the capacity and logical topology required to meet the services requirements either point-to-point or multi-point each technology might offers advantages and disadvantage for this interconnection. In this section we describe with the illustrative example on how 5G-CLARITY slices will use the integration between the VLAN domains of NPN and MNO will be integrated through a MAN/WAN transport domain managed by a public NOP to interchange control and data plane of each slice. The example Figure 6.10, is a generalization for a point-to-point connectivity MPLS-VPN.

In this example two L3-MPLS-VPNs between the NPN and MNO are provisioned through customer edge (CE) nodes serving also as on-premises gateway equipment (GW) (see D2.2 [2],Section 5) and router (R) (i.e., NPN-CE, MNO-CE) Figure 6.10(a). Each CE node is connected to the closest provider edge (PE) router enabling MPLS protocol NPN-CE to PE 1 and MNO-CE to PE 2. The transport network domain of this example is formed by PE 1 and PE 2 and providers routers P 1, P 2, and P 3.

The two infrastructure slices used in this illustrative example represent data plane interconnection between private and public networks to support service delivery models specified in 5G-CLARITY framework.

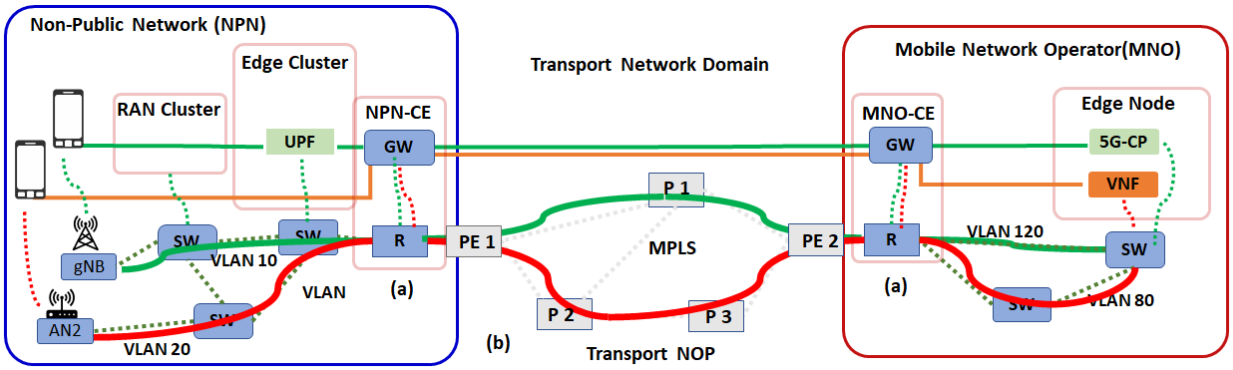


Figure 6.10. Illustrative Example of a VLAN-WAN point-to-point connectivity of two 5G-CLARITY slices

[7] Point-to-Point
[8] Multi-Point

In particular, 5G-CLARITY slice 1 (green) provides WATaaS and SLaaS, whereas 5G-CLARITY slice 2 (red) provides NFVIaaS. For these two slices, the following assumptions are taken: *i)* 5G-CLARITY slice 1 user plane uses the IP private network 192.168.1.0/24; *ii)* 5G-CLARITY slice 2 user plane uses the IP private network 192.168.2.0/24; *iii)* control plane of both 5G-CLARITY slices uses the IP private network 192.168.0.0/24; *iv)* private NOP and Public NOP encapsulate control plane traffic from slice 1 on VLAN 1 and from slice 2 on VLAN 2, respectively.

For the sake of understandability, let's focus on details about the 5G-CLARITY slice 1:

- **VLAN domain NPN:** Layer 2 control encapsulated into VLAN1, and IP data traffic encapsulated on VLAN 10, are switched from the L2 switch of the RAN cluster connecting the 5G DU/CU to an on-premises gateway (GW) virtual ports 1 (VLAN 1) and 10 (VLAN 10) of the NPN-CE nodes.
- **VLAN domain MNO**: Layer 2 control traffic is recovered from the MNO-CE/GW virtual port 1 to be encapsulated as VLAN 1 and the IP data plane traffic recovered from the MNO-CE/GW virtual port 120 is encapsulated in the VLAN 120. Both L2 VLANs traffic are switched from the MNO-CE to the Edge Node hosting the 5G-CP.
- **MAN/WAN transport domain**: a L3-VPN is provisioned, based on the QoS (latency and throughput) required by slice 1 and network resources available in the transport network domain (Figure 6.10(b)). In this example a shortest path with the lowest latency is obtained by a routing protocol or by adding a static route between NPN-CE to the MNO-CE (Green line). The MPLS protocol will allocate label following the shortest path, to switch L2 traffic from PE 1 (received from the NPN-CE) to PE-2 through the provider router P1 (Figure 6.10(b) thick green line). Finally, the PE 2 will route the traffic to the MNO-CE router.
- **VLAN domain and WAN/MAN transport domain:** NPN-CE might translate and encapsulate control and data plane traffic of the IP network (192.168.x.x) from transport domain IP address 10.x.x.x using NAT following a policy from the GW setup for each virtual port (TCP port and NAT reserved ports). In the other side the MNO-CE will do similar procedure to recover the traffic of slice 1 into the VLAN domain.

6.3 Intelligent stratum integration – initial solution design

AI/ML-based applications require high computation and memory capabilities and have a high-power consumption profile. The performance of an AI/ML-based applications also highly rely on the available data set (amount/diversity of the data, data refreshing frequency), how fast the existing model is re-trained with the new data set and how fast the re-trained model is deployed to take actions. On the one hand, not all components in a network architecture such as mobile end-user devices, light-weight edge servers have high-level computation and storage capabilities as in data centres. On the other hand, providing data or data sets to a more central location like data centres for AI/ML processing would increase network traffic burden and may not provide QoS requirements for delay-sensitive services such as URLLC. Hence, where to deploy an AI/ML-based application has a significant impact on achieving the service requirements.

In addition to performance-related objectives, data security and privacy protection are other aspects that should be taken into account while discovering options on AI engine deployment. For example, different network deployment options such as a standalone or public network integrated private network deployment as considered in 5G-CLARITY may have privacy concerns on uploading required data sets to outside the private premises. Therefore, privacy protection should also be considered when investigating the trade-off between the location of an AI engine that hosts AI/ML algorithms/applications and the performance gain/achievable QoS KPIs.

Accordingly, splitting AI/ML operation between various network components, distributing telemetry data

and AI/ML model, and gathering fragmented model training data for Federated Learning are discussed in detail in this deliverable to investigate AI engine location options.

This section provides an overview of the different deployment options for AI engines to perform ML model training as well as inference. The AI engine location aspects are discussed for the options on splitting AI/ML operations (training and inference) between various network components (Section 6.3.1), distributing the available telemetry data and trained ML model among the involved components (Section 6.3.2) and aggregating/combining different AI/ML models that are partially trained with a limited data set in either end user devices or edge devices (Section 6.3.3).

6.3.1 Splitting AI/ML operation between various network components

Splitting AI/ML operation between various network components can be considered in two aspects, namely, AI/ML training and AI/ML inference. Based on the device/edge server capabilities and service requirements, different options on offloading AI/ML inference or AI/ML training from one network component to another can be considered. The two inherent options are performing training and inference either at the end user device or at the edge/cloud server. However, any kind of splitting option for the AI/ML inference into multiple parts according to the current task and environment can also be considered to adapt load/congestion on both ends, the end user device and edge/cloud server. This kind of adaptive splitting options will alleviate the pressure of computation, memory/storage, power and required data rate. Hence, these options may obtain a better model inference performance on latency, accuracy and privacy protection.

Several modes for splitting AI/ML operations between different network components are discussed in [75]. In general, the considered modes are applicable for AI/ML training as well as inference, and they focus on data rate requirements, latency, splitting point and privacy. In other words, different options are considered *i)* to offload the computation-intensive, power-intensive parts to high-end network components such as edge and cloud servers; and *ii)* to perform the privacy-sensitive and delay- sensitive parts at the end user devices. Based on the considered splitting mode, either the device or edge server executes the operation/model up to a specific part/layer and send the intermediate data to other network components (device to edge server, device to cloud server) or other devices (device to device). The remaining parts/layers are then executed at these other network components and inference results are fed back to the device. A summary of the AI/ML operation splitting modes along with their advantages and disadvantages is provided in Table 6-3 where an illustration of the considered modes is also provided in Figure 6.11.

Table 6-3. Modes for Split AI/ML Operations Between Device and Network [75].

Mode	Mode Name	Description	Advantages	Disadvantages
(a)	**Cloud/edge-based inference**	• Training and inference are only carried out in a cloud or edge server. • The device only reports the sensing/perception data to the server.	• Limiting the device complexity as the device does not need to support AI/ML training and inference operations.	• The inference performance depends on communications data rate and latency between the device and the server. • Disclosure of the privacy-sensitive data to the network.
(b)	**Device-based inference**	• Inference is performed locally at the mobile device. • Training can be performed either at the device or server where pre-trained models are downloaded by the	• During the inference process, the device does not need to communicate with the cloud/edge server. • Preserves the privacy at the data source.	• Imposing an excessive computation/memory/storage resource to the device. • The device always keeps all the potentially needed AI/ML models onboard.

		device.		
(c)	**Device-cloud/edge split inference**	• Model is split into two parts between the device and the cloud/edge server. • The device will execute the AI/ML inference up to a specific part or layer and send the intermediate data to the cloud/edge server. • The server will execute the remaining part/layers and sends the inference results to the device. • Training can be either (i) partly executed in the server and device; or (ii) fully executed in the server, and the model is then downloaded by the device.	• Compared to Mode (a), more flexible and more robust to the varying computation resource and communications condition.	• The inference performance depends on communications data rate and latency between the device and the server. • Disclosure of the privacy-sensitive data to the network. • Need to properly select the optimum split point between the device and network side.
(d)	**Edge-cloud split inference**	• An extension of Mode (a). • The difference is that the DNN model (training and inference) is executed through edge-cloud synergy, rather than executed only on either cloud or edge server.	• The latency-sensitive part of an AI/ML inference operation or layers of an AI/ML model can be performed at the edge server. • The computation-intensive parts/layers that the edge server cannot perform can be offloaded to cloud server. • The device does not need to support AI/ML training/inference operations.	• The inference performance depends on communications data rate and latency between the edge and the cloud servers. • Disclosure of the privacy-sensitive data to the network. • A proper split point needs to be selected for an efficient cooperation between the edge and cloud servers.
(e)	**Device-edge-cloud split inference**	• A combination of Modes (c) and (d). • An AI/ML inference operation or an AI/ML model is split over the mobile device, the edge and the cloud servers. • The device sends the intermediate data outcome from its computation to the edge	• The computation-intensive parts/layers of an AI/ML operation/model can be distributed among the cloud and/or edge server. • The latency-sensitive parts/layers can be performed on the device or the edge	• The inference performance depends on communications data rate and latency between the device, the edge and the cloud servers. • Two split points need to be selected for an efficient cooperation between the device, the edge and the cloud servers.

		server. • The edge server sends the intermediate data outcome from its computation to the cloud server. • The training can be executed either fully at the cloud server and then distributed to the edge and device; or federated learning can be employed between the cloud server, edge server and device.	server. • The privacy-sensitive data can be left at the device	
(f)	**Device-device split inference**	• Provides a de-centralized split inference. • An AI/ML inference operation or model can be split over different mobile devices. • Devices exchange intermediate data between each other. • Training is executed in the server and then the pre-trained model(s) is downloaded by the devices.	• The computation load can be distributed over devices. • Each device preserves its private information locally.	• The inference performance depends on communications data rate and latency between the devices. • A proper split point needs to be selected for an efficient cooperation between the devices.
(g)	**Device-device-cloud/edge split inference**	• A combination of Modes (c) and (e). • An AI/ML inference operation or model is split into the device part and network part. • The device part can be executed in a de-centralized manner (further split over different mobile devices). • Training is executed in the server and then the pre-trained model(s) is downloaded by the devices.	• The intermediate data can be sent from one device to the cloud/edge server. • Multiple devices can send intermediate data to the cloud/edge server.	• The inference performance depends on communications data rate and latency between the devices. • A proper split point needs to be selected for an efficient cooperation between the device and other devices/the edge server/the cloud server.

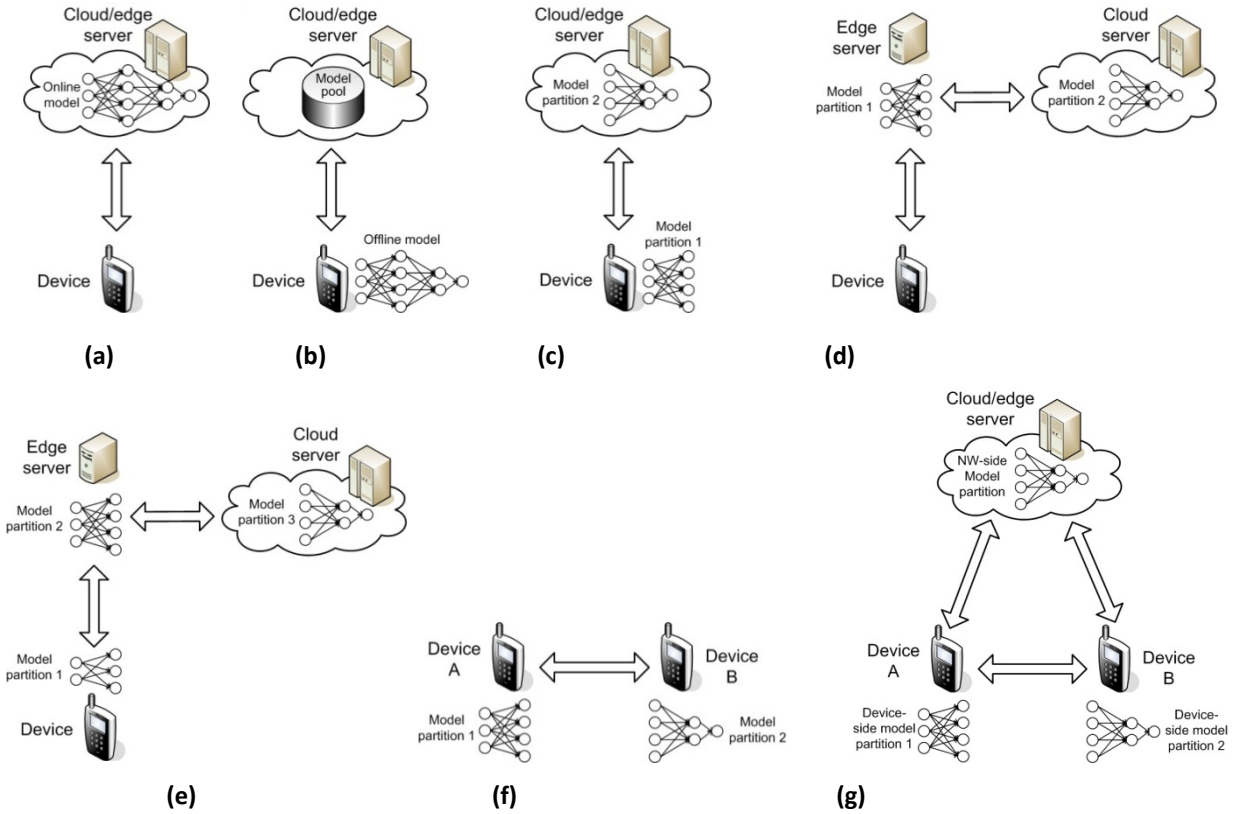


Figure 6.11. AI/ML inference splitting modes over endpoints [75].

6.3.2 Distributed telemetry data and AI/ML model

As noted, the end user devices have limited computation and storage capabilities compared to the edge and cloud servers. Therefore, AI/ML model training and inference performance including required data set size, accuracy, required time for training/inference and power consumption may not provide the expected gains from the AI/ML applications. Especially, the adapting the AI/ML model to various tasks and changing environments is the main challenge perform AI/ML training and inference on the end user devices as *i)* a large set of data is needed to improve AI/ML model/inference accuracy; and *ii)* the end user devices are computation-limited, storage-limited and battery powered, hence, training an AI/ML model may take quite some time. In addition, it may not be possible to have or store all the required telemetry data at the end user device. One of the options to compensate inference accuracy and performance KPIs is on adaptively selecting the AI/ML model from a set of already trained models and downloading the model, as noted some of the considered modes in Table 6-3. Such an option enables the end user device to respond different task requirements and environment variations without exhaustively utilizing the end user computation, storage and power resources.

Figure 6.12 shows an exemplary process of selecting and downloading an AI/ML model. First of all, this process depends on the end user device capabilities. In case running an AI/ML model is beyond the computation capabilities of the device, selecting and downloading an AI/ML model cannot improve the performance. Instead, other mode of splitting options listed in Table 6-3 such as device to edge or cloud server should be adopted. In case the device has the capacity and capability to run the AI/ML model, the selection of the model can be done based on the AI/ML task, specific environment, available telemetry data set and a list of the models available at the edge or cloud server. The AI/ML model selection can be controlled by either the end user device or network itself.

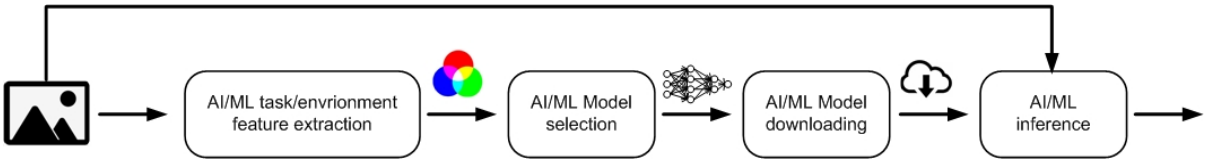


Figure 6.12. AI/ML model selection and downloading process [75].

As different network components may run the same AI/ML model with different data sets, there may be cases where the end user device first needs to provide environment-specific telemetry data to the edge or cloud server, and then the edge or cloud server takes the AI/ML model selection decision to accommodate AI/ML task requirements.

6.3.3 Federated Learning

The end user device data on network performance, link status, sensors, camera, etc. is essential to utilize huge gains promised by AI/ML applications. Thus, in conventional AI/ML applications, the end user data is collected at the edge or cloud servers for centralized AI/ML model training. However, as noted previously, user privacy/data security and network traffic overhead that may happen while uploading the end user device data to edge/cloud servers challenge the adoption as well as accuracy of AI/ML applications. To overcome this issue, an approach termed Federated Learning has been thoroughly discussed in [76]. In Federated Learning, the collected data is not the raw end user data, instead it is a partially trained AI/ML model data that is used to train a global AI/ML model. As the end user device (or the edge devices deployed in private premises) shares the AI/ML model training data, the user privacy is protected. There can be issues regarding the data security while transmitting the trained model data, the data security is enhanced from the end user perspective.

In Federated Learning, the main challenge lies on the aggregating/combining different AI/ML models that are partially trained with a limited data set on the end user devices or edge servers. As the accuracy level of the trained model depends on the available data set, computation and storage capabilities of the end user device or edge server, it can be expected that the confidence level of gathered interim training results varies between various devices. In order to achieve a desirable accuracy level in Federated Learning, several approaches such as iterative model averaging, frequency of aggregation for the global AI/ML model and so on are proposed [76].

7 Conclusions

5G-CLARITY system provides a number of capabilities for private network operators to easily deploy, configure and monitor on-premises infrastructure slices. These capabilities are provided by management functions distributed across two 5G-CLARITY architecture strata: **management and orchestration stratum**, focusing on slice provisioning and monitoring; and **intelligence stratum**, which provides necessary AI/ML and intent based artifacts to assist in the slice run-time operation (e.g., assurance). This deliverable has provided a first release version of these two strata, together with an initial evaluation of selected ML algorithms. Additional to these on-premises operational capabilities, this deliverable has also provided a first solution of the levers allowing for public-private network integration.

Section 2 reported the initial results on the development of the 5G-CLARITY management and orchestration stratum, providing implementation details on relevant management functions (Slice Manager, Multi-WAT non-RT Controller, Data Semantics Fabric and Data Lake), with some validations on relevant application scenarios. The next three sections reported initial results on the development of the 5G-CLARITY intelligence stratum. In particular, **Section 3** provided the initial results on selected ML algorithms, while **Sections 4 and 5** captured the first release version of AI and intent engines, respectively. Finally, **Section 6** provided an initial solution design for the interoperation with PLMNs, so the integrity of PNI-NPN can be ensure across all layers, from connectivity layer up to the intelligence layer.

The next deliverable of WP3, 5G-CLARITY D4.3, will report on the final release of the two in-scope 5G-CLARITY architecture strata, and on the final evaluation of selected ML algorithms. It also will provide a refined solution design for the integration with public networks, with a special focus on the mediation function and distributed AI.

8 Bibliography

[1] 5G-CLARITY Deliverable D4.1, "Initial Design of the SDN/NFV Platform and Identification of Target 5G-CLARITY ML Algorithms", October 2020.

[2] 5G-CLARITY Deliverable D2.2, "Primary System Architecture", October 2020.

[3] IETF RFC 6421, "Network Configuration Protocol (NETCONF)"

[4] IETF RFC 6020, "YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF)"

[5] 5G-CLARITY Deliverable D3.1, "State-of-the-Art Review and Initial Design of the Integrated 5GNR/Wi-Fi/LiFi Network Frameworks on Coexistence, Multi-Connectivity, Resource Management and Positioning", August 2020.

[6] https://github.com/Fundacio-i2CAT/ovsdb-manager

[7] H2020 5G-CITY project, https://www.5gcity.eu/

[8] Gary Duan, "How Kubernetes networking works – the basics" [Online]. Available : https://blog.neuvector.com/article/kubernetes-networking [Access in March 2021]

[9] Openstack project, "Openstack quotas" [Online]. Available: https://wiki.openstack.org/wiki/Quotas [Access in April 2021]

[10] ETSI Open Source MANO (OSM), "OSM Release SEVEN webinar" [Online]. Available: https://www.youtube.com/watch?v=KjwlgqmQ1-Y [Access in March 2021]

[11] Kubernetes, "Kubernetes Multus CNI" [Online]. Available: https://github.com/k8snetworkplumbingwg/multus-cni [Access in March 2021]

[12] OpenStack project, "Openstack Neutron service". Available at: https://wiki.openstack.org/wiki/Neutron [Access in March 2021]

[13] Wikipedia, "Link Layer Discovery Protocol (LLDP)" [Online]. Available: https://en.wikipedia.org/wiki/Link_Layer_Discovery_Protocol [Access in April 2021]

[14] 5G-CLARITY Deliverable D3.2, "Design Refinements and Initial Evaluation of the Coexistence, Multi-connectivity, Resource Management and Positioning Frameworks", May 2021.

[15] O. Adamuz-Hinojosa, P. Ameigeiras, P. Munoz, and J. M. Lopez- Soler, "Analytical Model for the UE Blocking Probability in an OFDMA Cell providing GBR Slices," in *IEEE Wireless Communications and Networking Conference (WCNC)*, Nanjing, China, March 2021.

[16] P. Muñoz, O. Sallent and J. Pérez-Romero, "Self-Dimensioning and Planning of Small Cell Capacity in Multitenant 5G Networks," in *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4552-4564, May 2018.

[17] J. Prados-Garzon and T. Taleb, "Asynchronous Time-Sensitive Networking for 5G Backhauling," in *IEEE Network*, vol. 35, no. 2, pp. 144-151, March/April 2021.

[18] J. Prados-Garzon, L. Chinchilla-Romero, P. Ameigeiras, P. Muñoz and J. M. Lopez-Soler, "Asynchronous Time-Sensitive Networking for Industrial Networks", in *IEEE European Conference on Networks and Communications (EuCNC),* Virtual Conference (Porto, Portugal), June 2021.

[19] Open5GS, "Open source project of 5GC and EPC (Release 16)" [Online]. Available: https://open5gs.org [Access in March 2021]

[20] Apache NiFi, "NiFi" [Online]. Available: https://nifi.apache.org [Access in May 2021].

[21] Apache Flink, "Flink" [Online]. Available: https://flink.apache.org/flink-architecture.html [Access in May 2021].

[22] 3GPP TR 23.700-91, "Study on enablers for network automation for the 5G System (5GS); Phase 2"

[23] I2CAT – hostpad Prometheus exporter. Available: https://github.com/Fundacio-i2CAT/hostapd_prometheus_exporter [Access in April 2021].

[24] The Linux Kernel documentation — "The Linux Kernel documentation" [Online]. Available : https://www.kernel.org/doc/html/latest/ [Access in May 2021].

[25] Linux manual page, "ss(8) — Linux manual page" [Online]. Available : https://man7.org/linux/man-pages/man8/ss.8.html [Access in May 2021].

[26] Linux manual page, "sock_diag(7) — Linux manual page" [Online]. Available: https://man7.org/linux/man-pages/man7/sock_diag.7.html [Access in May 2021].

[27] L. Fallon, S. van der Meer and J. Keeney, "Apex: An engine for dynamic adaptive policy execution," *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 2016, pp. 699-702.

[28] Facebook Inc, "What is FastText?" [Online]. Available: https://fasttext.cc/ [Access in April 2021]

[29] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, "Enriching Word Vectors with Subword Information", CoRR, 2016.

[30] COSMOTE – "Traffic Statistics Dataset".

[31] pyESN – "Echo State Networks in Python", https://github.com/cknd/pyESN

[32] Herbert Jaeger. Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach, volume 5. GMD-Forschungszentrum Informationstechnik, 2002.

[33] A Practical Guide to Applying Echo State Networks" - Mantas Lukosevicius, Jacobs University Bremen, Bremen, Germany.

[34] J. B. Carruthers and P. Kannan, "Iterative site-based modeling for wireless infrared channels," in *IEEE Transactions on Antennas and Propagation*, vol. 50, no. 5, pp. 759-765, May 2002.

[35] H. Schulze, "Frequency-Domain Simulation of the Indoor Wireless Optical Communication Channel," in *IEEE Transactions on Communications*, vol. 64, no. 6, pp. 2551-2562, June 2016.

[36] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Network," in *Mobile Computing*. Imielinski and Korth, Eds. Kluwer Academic Publishers, 1996, vol. 353.

[37] E. M. Royer, P. M. Melliar-Smith, and L. E. Moser, "An analysis of the optimum node density for ad hoc mobile networks," in *IEEE International Conference on Communications*. Conference Record (Cat. No.01CH37240), vol. 3, Helsinki, Finland, June 2001, pp. 857–861 vol.3.

[38] S. Hong, I. Rhee, S. J. Kim, K. Lee, and S. Chong, "Routing Performance Analysis of Human-Driven Delay Tolerant Networks Using the Truncated Levy Walk Model," in *Proceedings of the 1st ACM SIGMOBILE Workshop on Mobility Models, ser*. MobilityModels '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 25–32.

[39] A. A. Purwita, M. D. Soltani, M. Safari, and H. Haas, "Terminal Orientation in OFDM-Based LiFi Systems," in IEEE Trans. Wireless Commun., vol. 18, no. 8, pp. 4003–4016, Aug 2019.

[40] S. Merlin, G. Barriac, H. Sampath, et al. "TGax Simulation Scenarios". 2015 [Online]. Available : https://mentor.ieee.org/802.11/dcn/14/11-14-0980-16-00ax-simulation-scenarios.docx.

[41] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017

[42] D. Tsonev, N. Serafimovski, M. Uysal, et al. "Light Communications (LC) for 802.11: Link Margin Calculations". 2017 [Online]. Available: https://mentor.ieee.org/802.11/dcn/14/11-14-0980-16-00ax-simulation-scenarios.docx.

[43] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6980, 20.

[44] A. Adadi and M. Berrada, "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)," in *IEEE Access*, vol. 6, pp. 52138-52160, 2018.

[45] 3GPP TS 28.541, “Management and orchestration; 5G Network Resource Model (NRM); Stage 2 and stage 3 (Release 17)”

[46] GSMA NG.116 - Generic Network Slice Template Version 2.0, October 2019.

[47] 3GPP TS 28.552, “Management and Orchestration; 5G performance measurements (Release 17)”, December, 2020.

[48] S. Guadarrama, et. al (2018). TF-Agents: A library for Reinforcement learning in TensorFlow [Online]. Available: https://github.com/tensorflow/agents

[49] F. Xiao, Z. Guo, H. Zhu, X. Xie and R. Wang*, “AmpN: Real-time LOS/NLOS Identification with Wi-Fi,” *in IEEE ICC 2017 Ad-Hoc and Sensor Networking Symposium*, 2017.

[50] X. Wang, L. Gao, S. Mao and S. Pandey, “CSI-based fingerprinting for indoor localization: A deep learning approach,” *IEEE Transactions on Vehicular Technology*.

[51] J. Heaton, Introduction to Neural Networks for Java, Heaton Research, Inc, 2008.

[52] S. Marano, W. M. Gifford, H. Wymeersch and M. Z. Win, “NLOS identification and mitigation for localization based on UWB experimental data,” *IEEE Journal on selected areas in communications*, 2010.

[53] 5G-CLARITY Deliverable D2.1, “Use-Cas Specifications and Requirements”, February 2020.

[54] A. Karamyshev, E. Khorov, A. Krasilov, I.F. Akyildiz, “Fast and accurate analytical tools to estimate network capacity for URLLC in 5G systems”, *Computer Networks*, vol. 178, 2020.

[55] P. Muñoz, O. Adamuz-Hinojosa, J. Navarro-Ortiz, O. Sallent and J. Pérez-Romero, "Radio Access Network Slicing Strategies at Spectrum Planning Level in 5G and Beyond," in *IEEE Access*, vol. 8, pp. 79604-79618, 2020.

[56] S. Navaratnarajah, M. Dianati and M. A. Imran, "A Novel Load-Balancing Scheme for Cellular-WLAN Heterogeneous Systems With a Cell-Breathing Technique," in *IEEE Systems Journal*, vol. 12, no. 3, pp. 2094-2105, Sept. 2018.

[57] L. Simić, M. Petrova and P. Mähönen, "Wi-Fi, but not on Steroids: Performance analysis of a Wi-Fi-like Network operating in TVWS under realistic conditions," *2012 IEEE International Conference on Communications (ICC),* 2012.

[58] 5G-CLARITY Deliverable D2.3, “Primary System Architecture”, July 2021.

[59] J. Prados-Garzon and T. Taleb, "Asynchronous Time-Sensitive Networking for 5G Backhauling," in *IEEE Networ*k, vol. 35, no. 2, pp. 144-151, March/April 2021.

[60] J. Prados-Garzon, T. Taleb and M. Bagaa, "LEARNET: Reinforcement Learning Based Flow Scheduling for Asynchronous Deterministic Networks," *ICC 2020 - 2020 IEEE International Conference on Communications (ICC*), 2020, pp. 1-6.

[61] https://www.mosek.com/

[62] https://pjreddie.com/darknet/

[63] https://www.image-net.org/

[64] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510-4520.

[65] https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet

[66] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in International Conference on Learning Representations (ICLR), 2015

[67] OpenFaaS, “OpenFaaS platform” [Online]. Available: https://www.openfaas.com/ [Access in March 2021].

[68] OpenFaaS, “OpenFaaS API Gateway / Portal” [Online]. Available: https://docs.openfaas.com/architecture/gateway/ [Access in March 2021].

[69] Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Os-Ma-Nfvo reference point - Interface and Information Model Specification”

[70] 5G-CLARITY Deliverable D5.1, “Specification of Use Cases and Demonstration Plan”, January 2021.

[71] ETSI GS NFV-IFA 030, “Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Multiple Administrative Domain Aspect Interfaces Specification”.

[72] ETSI GS NFV-SOL 011, "Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; RESTful protocols specification for the Or-Or Reference Point"

[73] OSM-Federation: OSM White Paper, “OSM: Deployment and Integration”, Issue 1, Feb 2020. [Online]. Available: https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_Deployment_and_Integration.pdf [Access in April 2021].

[74] BSimplify, “SD-WAN Tutorial – The technology and Benefits” [Online]. Available: https://www.bsimplify.com/sd-wan-technology-tutorial/ [Access in April 2021]

[75] 3GPP TR 22.874, “5G System (5GS); Study on traffic characteristics and performance requirements for AI/ML model transfer”.

[76] W. Y. B. Lim *et al*., “Federated Learning in Mobile Edge Networks: A Comprehensive Survey”, in *IEEE Communications Surveys & Tutorials,* vol. 22, no. 3, pp. 2031-2063, Q3 2020.

[77] ETSI GS NFV-IFA 028, “Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Report on architecture options to support multiple administrative domains”.

[78] V. Sciancalepore et al., "A Future-Proof Architecture for Management and Orchestration of Multi-Domain NextGen Networks," in *IEEE Access*, vol. 7, pp. 79216-79232, 2019.

[79] SliceNet Deliverable D2.2, “Overall Architecture and Interfaces Definition”, January 2018.

[80] 5G-TRANSFORMER Deliverable D1.3, “5G-TRANSFORMER Refined Architecture”, May 2019.

[81] J. B. Hortiguela et al., "Realizing the Network Service Federation Vision: Enabling Automated Multi-Domain Orchestration of Network Services", IEEE Vehicular Technology Magazine, vol. 15, no. 2, pp. 48-57, June 2020.

[82] 5Growth, “Initial Design of 5G End-to-End Service Platform”, Deliverable D2.1, December 2019.

9 Annex A – MANO federation

9.1 MANO federation in ETSI ISG NFV

The multi-domain problem in virtualized environments has been addressed in ETSI ISG NFV [77], where some architecture options to support the multi-MANO inter-working service as well as simplified operational flows are analysed. One interesting use case in [77] is referred to as network services (NS) provided using multiple administrative domains. It can be mapped to the NFVI.MM2/3 of the 5G-CLARITY NFVIaaS model, where the MNO establishes a connection to the 5G-CLARITY NFVO. In this case, each administrative domain deploys the functional blocks of the ETSI NFV MANO architectural framework, including an NFVO to manage the specific NSs that are hosted and offered by the organization. This solution relies on the concept of composite NS and nested NSs specified in [69]. Specifically, an administrative domain can build up a composite NS from several constituent nested NSs, which would be offered by a different administrative domain. Nested NS can also be shared by another composite NS that may belong to other administrative domains. In such a hierarchical layout, the on-top administrative domain(s) manage(s) the composite NS, while the other administrative domains manage the nested NS. Figure 9.1 shows a multi-domain scenario, where each administrative domain holds its own ETSI NFV MANO stack. The administrative domain managing the composite NS (C) is interconnected to the administrative domains that are responsible of the nested NS (A and B).

Some issues arise when using composite NS and nested NS among different administrative domains. One is that an SLA is required between organizations. Consequently, for SLA supervision, the resource usage of the constituent nested NSs needs to be monitored. To ensure proper operation of the system, some fault and performance monitoring is also required. In addition, specific protocols may be defined to provide auto-discovery of the ETSI NFV MANO functional blocks from other administrative domains. The required interoperability to overcome these issues is provided by a new reference point between NFVOs, named Or-Or, as shown in Figure 9.1.

In case an operator wants to deploy a composite NS spanning multiple administrative domains, the Or-Or interface should allow the operator's NFVO to query the NSDs of the nested NS in order to identify and select them.

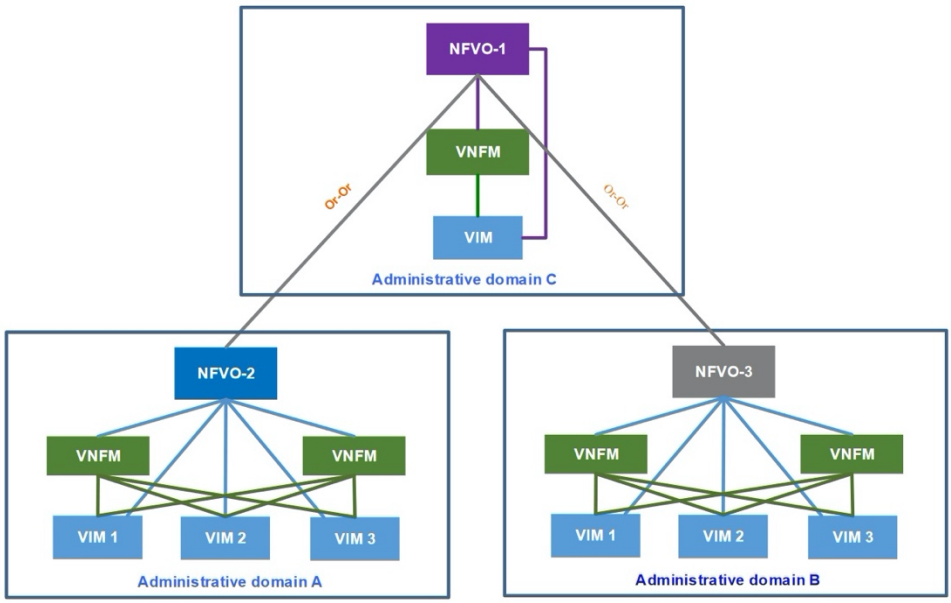


Figure 9.1. Example of multiple administrative domains providing a composite NS [77].

Additionally, some operations for the lifecycle management of the composite NS should be supported, such as instantiation, scaling and termination. For example, in Figure 9.1, the NFVO-1 can trigger the instantiation of the nested NSs to the NFVO-2 and NFVO-3. The functional requirements, interfaces and operations through the Or-Or reference point is covered in NFV-IFA 030 [71],while the specification of a set of RESTful protocol and data models for the interfaces over the Or-Or reference point is captured in NFV-SOL 011 [72].

Another representative use case described in [69] is the NFVIaaS, which can be mapped to the NFVI.MM3 of the 5G-CLARITY NFVIaaS model. In this case, the NFVIaaS consumer (e.g., the MNO) establishes a connection to the VIM of the NFVIaaS provider (e.g., the 5G-CLARITY system). The solution may have one or multiple logical points of contact between the administrative domains. In addition, the consumer's VNFM may invoke the operations directly on the provider's VIM or indirectly through the consumer's NFVO. The number of logical points of contact influences the way in which 5G-CLARITY computes quotas would be managed. In particular, a solution based on multiple connections allows the consumer's NFVO to break down the quota to the respective VIMs and maintains the quota information across the VIMs. On the contrary, with a single connection, the responsibility for coordinating the quota across the different VIMs falls on the side of the NFVIaaS provider. As in the previous use case, it is expected that an SLA is established between the NFVIaaS provider and consumer, as well as monitoring of resource usage is carried out for SLA supervision. The NFVIaaS provider should also give an overall view of the NFVI resources, including some fault and performance monitoring information. The NFVIaaS consumer may be responsible for establishing quotas and other constraints. However, it would only receive information on resources related to its NFVI service.

9.2 MANO federation in the research community

In addition to the efforts of Standards Development Organizations (SDOs) to face the multi-domain orchestration, there are also some works in the research community that deserve special attention. For example, the authors of [78] propose an architecture that extends the standard ETSI NFV MANO system, offering a versatile solution. While the solution presented in Figure 9.1 is based on a peer-to-peer communication between the respective NFVO instances, the coordination between the different ETSI NFV MANO systems in this new solution is performed by an over-arching entity, as shown in Figure 9.2. In particular, the solution relies on an inter-slice resource broker which enables orchestration per administrative domain by managing a set of complete ETSI NFV MANO stacks. The role of this coordination entity is to give a general view of the whole infrastructure that can be offered within a single administrative domain and to provide monitoring of the resource usage for each NFVI consumer. As resource broker, this entity also controls the dimensioning of resources that are assigned to each NFVI consumer. However, the proposed architectural framework is limited by the fact that only one NFVIaaS provider, i.e., the network operator, is assumed in the scenario. Note that the solid rectangles in Figure 9.2 represent the management functions of the network operator. In this way, the resources from different administrative domains are integrated into a single NFVI block that is orchestrated from a single administrative domain.

There are also some relevant research projects addressing the multi-domain orchestration. Specifically, the H2020 SliceNet project [79] defines a cognition-based approach where the orchestration system interacts with the cognition system to provide the required actions that guarantee the Quality of Experience (QoE) of the multiple NS deployed across different administrative domains. The cognition system enables, for example, effective fault management by identifying the administrative domain that requires modifications to enforce the corresponding recovery actions.

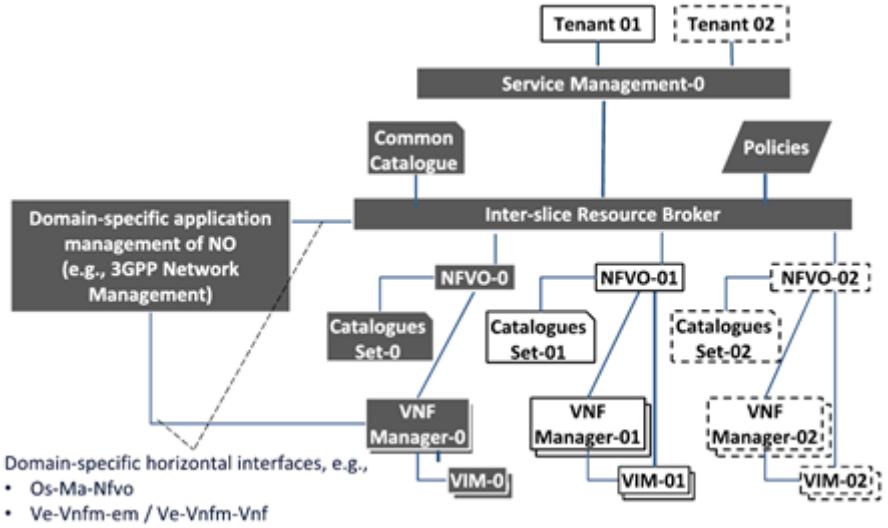


Figure 9.2. Architectural framework proposed in [78] for multi-domain orchestration.

The H2020 5G-TRANSFORMER project [80] proposes a platform architecture that is decomposed into three building blocks: the vertical slicer, supporting the creation and management of slices for verticals; the service orchestrator (shown in Figure 9.3), for end-to-end service orchestration and federation of resources and services from multiple domains; and the mobile transport and computing platform, acting as the underlying front-haul and back-haul transport network infrastructure. To enable a multi-domain capable service orchestrator, the architecture includes two additional blocks (highlighted in Figure 9.3): a hierarchical (parent-child) service orchestration engine (SOE) and a composite resource orchestrator engine (CROOE). The SOE parent coordinates with other SOE parents for multi-domain service orchestration and interacts with the corresponding CROOE, which manages the interconnections between the nested NSs. Given this, the service orchestration part would require minimum modifications compared to NFV-IFA013/030 for supporting federation [81].

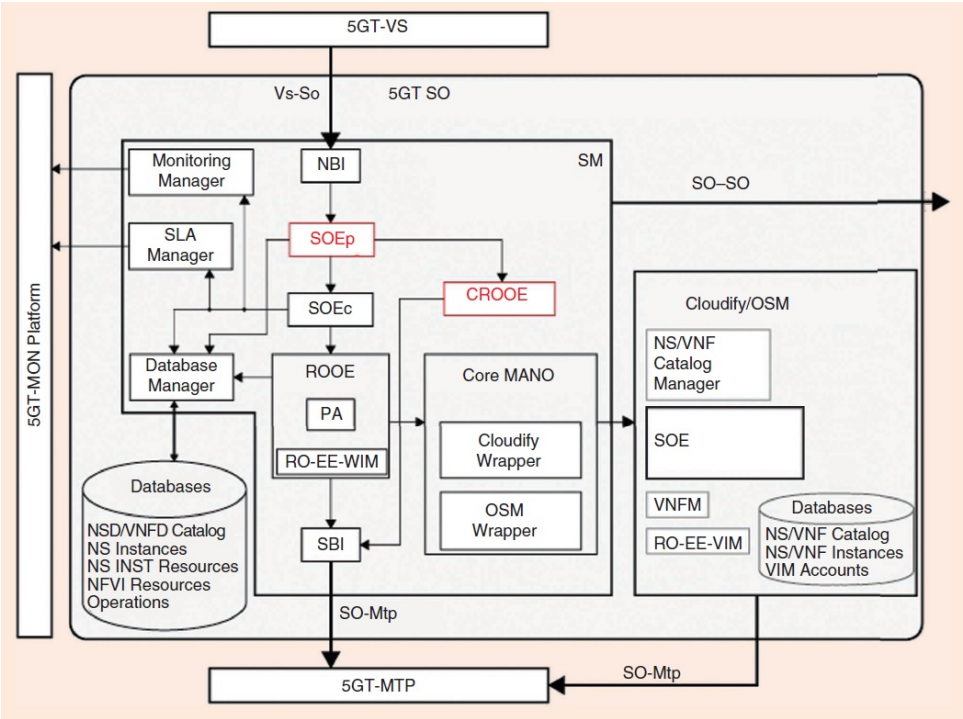


Figure 9.3. The 5G-TRANSFORMER service orchestrator (SO) software architecture [81].

The H2020 5Growth project [82] is envisioned to enhance the 5G-TRANSFORMER platform. In particular, the vertical slicer component provides additional support towards the multi-domain at the service level. It defines a new functional block, the communication service federation function, which decides how to split the communication service into subservices that can be assigned to different provider domains. Federation can also occur at the network level, where the service orchestrator acts as an NFVO, comprising service and resource orchestration functionalities. In this case, the NS request is handled as a composite NS and the service orchestrator will interact with the service orchestrators of other domains to deploy the nested NS.