871780 — MonB5G — ICT-20-2019-2020

Deliverable D4.2
Final Report on AI-driven Techniques for the MonB5G Decision Engine

Document Summary Information

Grant Agreement No	871780	**Acronym**	MonB5G
Full Title	Distributed Management of Network Slices in beyond 5G		
Start Date	01/11/2019	**Duration**	42 months
Project URL	https://www.monb5g.eu/		
Deliverable	D4.2 – Final report on AI-driven techniques for the MonB5G Decision Engine		
Work Package	WP4		
Contractual due date	M31	**Actual submission date**	30.06.2022
Nature	Report	**Dissemination Level**	Public
Lead Beneficiary	ORA-FR		
Responsible Author	Fabrice Guillemin (ORA-FR)		
Contributions from	Luis Blanco (CTTC), Farhad Rezazadeh (CTTC), Engin Zeydan (CTTC), Hatim Chergui (CTTC), Josep Mangues (CTTC), Sarang Kahvazadeh (CTTC), Fabrice Guillemin (ORA-FR), Sławomir Kukliński (ORA- PL), Robert Kołakowski (ORA- PL), Rafał Tępiński (ORA-PL), Michał Rosiński (ORA-PL), Jerzy Jegier (ORA-PL), Thrasyvoulos Spyropoulos (EUR), Pavlos Doanis (EUR), Adlen Ksentini (EUR),		

	Lanfranco Zanzi (NEC), Francesco Devoti (NEC), Mohamed Rahali (b<>com), Anestis Dalgkitsis (IQU), Anne Marie Bosneag (LMI), Ashima Chawla (LMI)

Revision history

Version	Issue Date	Complete (%)	Changes	Contributor(s)
V0	June 13, 22	60%	Compilation of the contribution	ORA-FR
V1	June 15, 22	80%	Integration of the different contributions	ORA FR, CTTC, ORA-PL, EUR, IQU, LMI, NEC, b<>com
V2	June 20, 22	90%	Stable version ready for reviewing	ORA FR, CTTC, ORA-PL, EUR, IQU, LMI, NEC
V3	June 27, 22	99%	Version for final polishing	ORA FR, CTTC, ORA-PL, EUR, IQU, LMI, NEC
V4	June 29, 22	100%	Final version	ORA FR, CTTC, ORA-PL, EUR, IQU, LMI, NEC

Disclaimer

The content of the publication herein is the sole responsibility of the publishers, and it does not necessarily represent the views expressed by the European Commission or its services.

While the information contained in the documents is believed to be accurate, the authors(s) or any other participant in the MonB5G consortium make no warranty of any kind regarding this material including, but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Neither the MonB5G Consortium nor any of its members, their officers, employees, or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

Without derogating from the generality of the foregoing neither the MonB5G Consortium nor any of its members, their officers, employees, or agents shall be liable for any direct or indirect or consequential loss or damage caused by or arising from any information advice or inaccuracy or omission herein.

Copyright message

TABLE OF CONTENTS

List of Figures

List of Tables

List of Acronyms

Acronym	*Description*
3GPP	*Third Generation Partnership Project*
AE	Analytic Engine
AE-F	Analytic Engine Function
AE-S	Analytic Engine Sublayer
AI	Artificial Intelligence
CLA	Closed-loop Automation
CNF	Cloud Native function
DE	Decision Engine
DE-F	Decision Engine Function
DE-S	Decision Engine Sublayer
EEM	Embedded Element Manager
eMBB	Enhanced Mobile Broadband
eTOM	Enhanced Telecom Operations Map
ETSI	European Telecommunications Standards Institute
ECA	Event Condition Action
ENI	Experiential Networked Intelligence
FCAPS	Fault, Configuration, Accounting, Performance, Security
ISM	In-Slice Management
ITU	International Telecommunication Union
KPI	Key Performance Indicator
LCM	Lifecycle Management
ML	Machine Learning
MANO	Management and Orchestration
MaaS	Management as a Service
MAN-F	Management Function
mMTC	Massive Machine Type Communications
MEO	MEC Orchestrator

MNO	Mobile Network Operator
MLaaS	MonB5G Layer as a Service
MS	Monitoring System
MS-F	Monitoring System Function
MS-S	Monitoring System Sublayer
MEC	Multi-access Edge Computing
NFVO	Network Function Virtualization Orchestrator
NSD	Network Service Descriptor
NSO	Network Service Orchestrator
NSP	Network Service Provider
NSI	Network Slice Instance
NSMF	Network Slice Management Function
NSSMF	Network Slice Subnetwork Management Function
NST	Network Slice Template
NSSI	Network sub-Slice Instance
NGMN	Next Generation Mobile Networks
NFVI	NFV Infrastructure
OAI	Open Air Interface
ONAP	Open Network Automation Platform
OSM	Open-Source MANO
OSS	Operation System Support
PaaS	Platform as a Service
PoC	Proof of Concept
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
SON	Self-Organizing Network
SLA	Service Level Agreement
SFL	Slice Functional Layer
SML	Slice Management Layer

SM	Slice Manager
uRLLC	Ultra-Reliable Low-Latency Communication
VIM	Virtual Infrastructure Manager
VNF	Virtual network Function
VNFM	Virtual network Function Manager
ZSM	Zero-touch network and Service Management

1 Executive summary

Decision Engine (DE) is one of the key elements of the closed-control loop devised in the MonB5G project aiming at achieving zero-touch management of a massive number of network slices in Beyond 5G networks. DE oversees the decision-making process of the closed-control loop, mainly using AI and relying on the Analytical Engine (AE) inputs during the Lifecycle (LC) of network slices. WP4 is dedicated to studying and devising distributed and scalable AI-Driven algorithms for DE to manage a massive number of network slices. This deliverable contains a final version of the devised DE algorithms that either complements and improves the initial versions introduced in D4.1 or introduces new algorithms. It is worth noting that we improved D4.1 solutions by putting efforts toward covering not only a single domain but multiple technological and administrative domains and improving the scalability of the different solutions. We also added a new algorithm that particularly focuses on DE conflict resolution.

This deliverable comprises five key parts that summarize the achievement of tasks T4.1, T4.2, and T4.3. The integration and implementation results will be presented in a separate document D4.3. However, initial performance evaluation results of the different schemes and algorithms are also included in this deliverable.

The first part of this deliverable details and recalls the role of the DE in the MonB5G architecture, particularly focusing on the specifications, functions, interfaces, and cross-domain operations. It is important to understand the interaction, particularly how DE communicates with the AE, Monitoring System (MS), and other DE to achieve decentralized management. Indeed, one of the key achievements in this deliverable is the distributed AI-driven solutions, which in addition to the algorithmic part, need communication interfaces across domains to ensure the end-to-end DE.

The second part is dedicated to admission control algorithms that can support massive numbers of slice arrival events. Admission control is a vital function of the LC Management (LCM) of network slices as it allows managing the resources efficiently while guaranteeing SLA. Knowing the diversity of slice types, the complexity of VNF/PNF chain configuration and placement options, and the inherent uncertainty in future slice requests, we explored in MonB5G solutions that are model-free and based on both regular and deep re-enforcement learning. We aim at maximizing the number of co-existing slices while minimizing SLA violations. In D4.1, we have devised algorithms for admission control based on Deep Q-reinforcement Learning (DQL) and Regret Machine (RM) solution for RAN and Cloud domains, while we identified several potential algorithms for slice component placement. In D4.2, we introduce two other slice admission approaches. The first one is based on Deep Reinforcement Learning (DRL) approach, which has been combined with a heuristic to reduce the learning period of the DRL. The second solution is based on the Time-of-Day (ToD) traffic curve that reflects the human activity, which has been used to predict eMBB and use a DQL for admission control.

The third part is devoted to intra-slice orchestration consisting of intra-slice reconfiguration and resource allocation. In D4.1, we devised solutions by organizing them into (i) domain-specific, such as VNF configuration, reconfiguration, and migration; (ii) cross-domain, such as e2e VNF and slice placement using multi-agents Reinforcement Learning (RL). Unlike D4.1, where we focused mainly on domain-specific solutions, in this deliverable, we envision particularly cross-domain solutions. To this end, we devised a solution for service chain management in a multi-technological domain using a Distributed RL algorithm.

The fourth part concerns the inter-slice orchestration that lies at a level above its intra-slice counterpart and requires a wider picture of the network and its performance. In D4.1, we presented the first algorithmic approaches, focusing mainly on domain-specific inter-slice orchestration. The proposed solutions included a RL algorithm for VNF placement and reconfiguration, a Multi-Armed Bandit algorithm for RAN resource allocation, and a probing scheme for VNF bottleneck localization. In D4.2, we build on to either provide extensions to the solutions of D4.1 (aiming mainly at increasing their scalability and at supporting many slices) or to apply them in more practical use cases. More specifically, we extended from D4.1: (i) the VNF placement and migration algorithm; (ii) the RAN resource allocation algorithm; (iii) the probing scheme for VNF bottleneck localization.

The final part of the document is dedicated to a new challenge that was not integrated in D4.1, which is related to the control loops coordination. This issue arises when different functions (or sometimes DE) are trying to optimise a single goal, which can lead to suboptimal results or chaotic system behaviour. To this end, we devise a novel coordination framework compliant with the MonB5G architecture that aims at predicting reconfiguration's impact and hence generating recommendations to minimise conflicts related to network or slice configuration parameters.

2 Introduction and Problem Scope

2.1 Problem Scope

MonB5G proposes *a fully data-driven decision engine and algorithm stack. R*ecent works and related EU projects addressing the introduction of AI solutions into the 5G/B5G architecture, either focus primarily on data analytics or target specific components of the orchestration architecture in a piecemeal fashion. In contrast, MonB5G explores data-driven algorithms as a key element of all slice lifecycle management operations, from admission control, to scaling and migration operations both for a specific slice ("*intra-slice* operation") involving one or more technological (e.g., RAN, MEC, core, cloud) and/or administrative domains, to end-to-end orchestration operations involving a large number of such slices ("*inter-slice* operation"). The Decision Engine (DE), the main architectural component targeted by WP4, has as its main objective to react to sophisticated KPI predictions or anomalies reported by the AE (and sometimes directly connected to the Monitoring System, e.g., for training), to both improve the *network resource usage* (across all slices) and (and perhaps more importantly) to meet the *diverse Service Level Agreements (SLAs)* per slice.

The MonB5G architecture (and its WP4 solutions) brings several important innovations:

First, the DE architecture is designed to be flexibly *distributable across different administrative domains, technological domains*, and even fine granularities such as one DE per tenant, slice, or even VNF (virtual network function). Flexible distribution means that the proposed DE can operate in different configurations as needed. In the solutions provided in this project, we will encounter algorithms that fit a *hierarchical DE* implementation, where a "central" DE (e.g., in the cloud or core domain) has control over different "edge" DEs that are (partially) responsible for different RAN domains, edge clouds (e.g., MEC), etc., or even slice-specific DEs, as mentioned earlier, all working toward optimal lifecycle management of slices that span all of these domains. We will also see solutions consisting of multi-agents facilitated by even more decentralized (or federated) distribution of DEs across domains and/or components.

Second, another important innovation compared to most existing approaches is that the solutions proposed in MonB5G WP4 can orchestrate slices consisting of multiple VNFs across different technological (and even administrative) domains. Equally important, the solutions described can handle diverse service level agreements with sophisticated end-to-end KPIs (e.g., end-to-end delay constraints across an entire, complex VNF graph).

Third, while several recent works have addressed data-driven solutions for 5G analytics and sometimes slice orchestration, the solutions described in this deliverable go beyond the state-of-the-art both in terms of (i) improving the performance of previously proposed data-driven algorithms for such tasks and (ii) distributing the components of the data-driven algorithms (e.g., multi-agents) or even the components of the neural network architecture (Distributed DNNs). If DE algorithms are located as close as possible to the network components for which the decisions are made, decision latency can be significantly improved, especially for tasks that operate at a much smaller time granularity (e.g., RAN) than traditional data-driven systems (e.g., application-level image classification). At the same time, the amount of information that needs to be collected and transported over the network to enable the intensive use of modern data-driven algorithms is orders of magnitude larger. Finally, if the trained model used is very large (e.g., modern deep neural network architectures used for sophisticated ML tasks, may have millions of trained weights), distributing the actual model across local DE components is a necessity for B5G architectures.

2.2 Deliverable Scope

This deliverable D4.2 describes progress on Tasks 4.1, 4.2, and 4.3 related to distributed slice lifecycle management, beyond the preliminary work documented in the deliverable D4.1 and presented during the midterm review. As planned, our results for Task 4.4 will be documented in a separate, future deliverable D4.3. In this second and final part of the MonB5G effort to develop algorithms for the Decision Engine, we have focused our efforts on several key dimensions, considering the suggestions of midterm reviewers:

From single domain to multi-domain solutions: A large portion of the solutions presented in D4.1 worked with slices that involved a single administrative or even technological domain. We have endeavoured to extend all the solutions presented in this new deliverable to either support multiple domains or to be integrable/interoperable with other algorithms covering other domains.

Improved and measurable scalability: We have already documented in DoW why Decision Engine distribution (and associated algorithms) *improves* the scalability of baseline/existing orchestration solutions. We have also provided preliminary validations of our algorithms in D4.1 that support these claims to some degree. Nevertheless, in this second phase of the project, we have made significant effort to both *significantly improve* the scalability of our solutions and *clearly validate* the scalability of each solution using simulation scenarios designed to demonstrate this scalability against well-defined baseline solutions.

Integration/Interoperability of different solutions: Although the actual integration is part of Task 4.4, we have taken initial steps to integrate and/or complement a subset of the proposed algorithms (while also fostering collaboration between partners). Examples include the SafeRL-based extension of the Schema algorithm presented in Chapter 5, the multi-agent algorithm from Chapter 6, and the “umbrella” decision conflict resolver from Chapter 7.

Algorithmic improvements: In addition to the above dimensions, we have improved several the algorithms proposed in D4.1 in terms of scalability (as mentioned), SLA performance, convergence, etc., but also introduced some new algorithms.

2.3 Outline and focus

The deliverable begins with an elaboration of the proposed (distributed) Decision Engine (Chapter 3), focusing on intra-DE interfaces and the interfaces with the Monitoring System (MS) and AE. It continues with three chapters (Chapters 4-6), each presenting an example of complete algorithms for Tasks T4.1, T4.2 and T4.3.

Specifically, Chapter 4 presents two schemes for slice admission control, the subject of Task 4.1: (i) the evolution of a multi-domain data-driven scheme, originally documented in D4.1 that combines both modern Reinforcement Learning (RL) methods and integer program solving techniques to address the problem; (ii) a new admission control scheme that also introduces a type of “calendaring” and attempts to exploit the periodicity of human activity (and associated traffic).

We then move to Chapter 5, which deals with intra-slice management (the topic of Task 4.2), e.g., how to scale up or down the resources of a slice, or migrate the VNFs of a slice chain, in a distributed manner to satisfy different types of SLAs, using data-driven mechanisms. These solutions not only consider the current slice and network status (e.g., via MS and AE), but also optimally consider future evolution. We first describe the evolution of our previous solution, SCHEMA, focusing on the various improvements that have been made. We then propose an important extension of this algorithm, SafeSCHEMA, which gracefully integrates the important SafeRL framework into the scheme to operate in scenarios where the natural tendency of RL algorithms to explore "any" possible (slice) configuration might be forbidden (or prohibitively expensive) and therefore the exploration/exploitation components need to be protected.

In Chapter 6, we go a step further and present some algorithms that deal with related reconfigurations and scaling events when multiple slices, possibly belonging to different tenants, overlap (partially or completely) across domains (the topic of Task 4.3). Specifically, we first document how our inter-slice VNF placement and migration algorithm, originally demonstrated for one domain in D4.1 has been extended as follows: (i) it now supports multiple VNFs per slice, i.e., an arbitrary, probabilistic VNF graph (which also allows for loops); (ii) various realistic end-to-end performance metrics for the average flow served by such a VNF graph can now be supported; (iii) a multi-agent solution where a DE (agent) is located at each slice (or even at each VNF of a slice) radically improving the scalability of the scheme as the number of slices (or VNF host sites) increases, while still maintaining near-optimal performance.

Finally, Chapter 7 deals with the very important topic of DE conflict resolution. This is a subject of the overarching algorithms (and DE components) of all Tasks 4.1, 4.2, and 4.3, and proposes a solution to resolve potential disagreements between the various (local) DEs that might otherwise destabilize the system and lead to oscillations or crushing.

In Table 1, we summarize the topic of each chapter and how it relates to the WP4 tasks.

Table 1. Summary of each chapter's content in this deliverable.

Chapter	Description	Task(s)	Starting Month
2	Problem and deliverable scope and outline	T4.1 – T4.3	M7
3	Decision engine architecture evolution	T4.1 – T4.3	M7
4	Describe progress in admission control algorithms, both evolution of already presented ones, as well as new ones	T4.1	M7
5	Describe progress in intra-slice orchestration algorithms	T4.2	M7
6	Describe progress in inter-slice orchestration algorithms	T4.3	M7
7	New work on DE conflict resolution	T4.1-T4.3	M7

Finally, we would like to point out that we have chosen to present each algorithm in each of these chapters in the following format to better focus the reader's attention on the key messages.

Problem Setup: Each algorithm presented first introduces the detailed problem it is working with. This subsection helps to clarify: (i) the assumed slice model, (ii) the targeted KPIs or SLAs, (iii) the control variables of the problem, i.e., what configurable system "knobs" the algorithm is trying to choose; examples of control variables are resources allocated to specific VNFs (e.g., RAN resource blocks, CPU cores), locations where slice components (e.g., VNFs) could be placed, etc., and (iv) what kind of additional constraints (in the state or action space) the problem might impose. Here we also discuss some recent related work per solution.

Solution Methodology: For each algorithm, we then present the solution methodology. In other words, we describe what kind of tool(s) the algorithm uses to solve the above problem. The tools presented in the context of the different algorithms include modeling tools (e.g., Markov Chains) as well as optimization tools such as (Deep) Reinforcement Learning, Online Convex Optimization (e.g., multi-armed bandits), discrete optimization solvers, etc. The goal is to address both the "why" (this algorithm was chosen for the problem) and the "how" (algorithm variation, parameter tuning, etc.)

Scalability Validation: In this second phase of the project, we have made significant efforts to both improve the scalability of our solutions and, perhaps more importantly, to validate and demonstrate that scalability against well-motivated baselines.

Positioning in the architecture of Decision Engine: In this subsection, we attempt to give an initial idea of the position of each of the algorithms discussed within the architecture of decision engine. We briefly address whether the DE implementation implied by the algorithm is hierarchical or more decentralized, what domains the algorithm covers, and what DE granularity is required (e.g., per BS, per slice, per VNF, etc.).

MonB5G checklist**)** Finally, we provide a summary of the important MonB5G goals that the specific algorithms help to achieve, be it distributed operation, latency reduction, convergence speed improvement, types of data-driven algorithms promised, etc.

3 Positioning of the DE in the MonB5G architecture

The goal of this section is to provide an overview on the MonB5G architectural elements described in [2] related to the decision engine (DE) and the data-driven decision schemes. This section discusses the **Decision Engine specifications, functions, interfaces, and cross-domain operation. It starts by positioning DE in the MonB5G architecture and presenting its relation to the other elements of the Monitor-Analyze-Plan-Execute (MAPE) framework**, i.e., the Monitoring System (MS), the Analytic Engine (AE), as well as the Actuators (ACT). **The proposed framework is in line with the recently proposed ETSI ZSM closed loop automation architecture [1].** The section continues with a thorough description of the different DE interfaces, either within the slice or with other external elements of the MonB5G distributed architecture, providing a description of the blocks of the DE in a generic way, which can be instantiated differently depending on the target scope of analysis (domain, slice, or network function). The section then moves to show how the distributed architecture of MonB5G can be harnessed for decentralized data-driven decision making, either intra-domain or cross-domain, involving the end-to-end Decision Engine or the local DEs. This will be illustrated further through the practical DRL algorithms described in Sections 4-6 to reveal the nature of cooperation between the different DEs.

3.1 Generic Description

3.1.1 ZSM CLOSED LOOP CONTROL

ETSI has recently proposed some insights on the Zero-touch Service Management (ZSM) framework for Closed Loop (CL) management automation [1]. CLs may exist in each of the management domains of the ZSM architecture. Figure 1 presents the CL functional scheme, in which the 'Decision' stage plays a key role. The aim of this section is to briefly present the ZSM framework proposed by ETSI and link it with MonB5G DE.

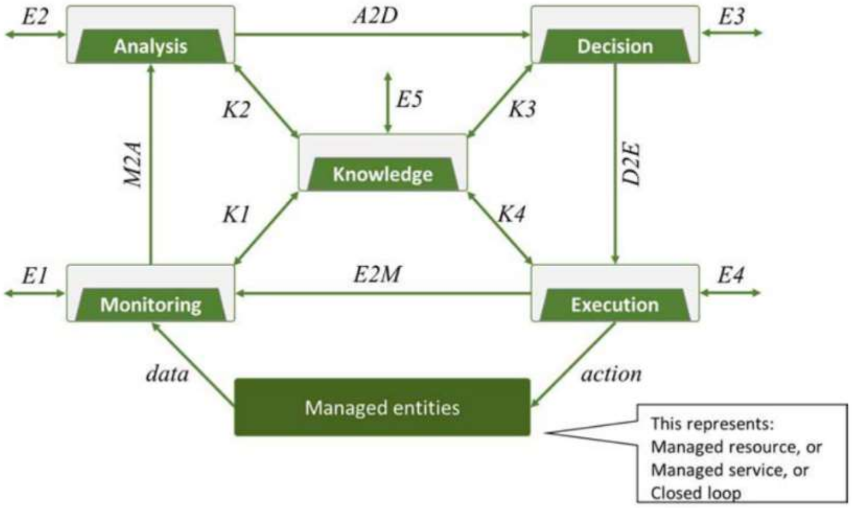


Figure 1: Block diagram of a closed loop in the ZSM framework.

This Figure is composed by four stages besides the 'Knowledge' functional block. The 'monitoring/collection' stage is the responsible for gathering and pre-processing the raw data from the managed entities or external resources (in this context, a managed entity is either a service, a managed resource, or another closed loop). Since raw data can have different heterogeneous formats, coming from different sources, can be transformed in a way that it allows to analyze it in conjunction with the data coming from other sources. After this, comes the 'Analysis' stage, which provides insights from the available data obtained from the Monitoring stage. Then, the **'Decision' stage**, which governs managed entity, decides the action that must be taken based on the issues detected by the analysis block. These actions can be reactive, proactive, or predictive. It should be remarked that the decision stage is only responsible for deciding which actions are necessary, but not for their execution. This is the competence of the 'Execution' stage, which translates the decided actions into commands. The Execution stage oversees the execution of the necessary workflows in the managed entity to implement the actions determined by the decision stage. This execution could also involve other management domains. When this happens, interactions with other CLs are needed. Thus, multiple distributed CLs are required for the automation of E2E service management. The 'Knowledge' block in the functional diagram presented above is not technically a stage of the CL, it refers to the storage and retrieval of historical, configuration and operational data that are shared between the stages of a closed loop as well as between different CLs in the network.

The 'Decision' stage involves different primary flows or interfaces:

- **A2D interface**. It connects the Analysis function with the Decision stage. It provides insights on historical and/or real-time information provided by the collection/monitoring stage. It can also provide information for tuning the analytic models and starting/terminating the analytics processes.
- **D2E interface**. It is used by the Decision stage to provide action plans in form of workflows (e.g., configuration changes, onboarding services and resources). It can also be used to provide information to tune the decision models and start/stop the decision processes.
- **E3 external interface** represents the data and control inputs and outputs from/to other closed loops or external entities. It can be used to: i) start/stop the decision processes; ii) change the settings of the Decisions stage and attributes of the models; iii) retrieve the historical or real-time data of the function, such as logs, outcomes of the Decision function; iv) provide the resulting data of the Decision stage to other closed loops or authorized entities outside the ZSM framework, e.g., external management systems.
- **K3 interface** is a knowledge-enable flow, which is represented by a double-headed arrow, and is used for data-related inputs and outputs from the Decision stage. The primary interfaces exposed above can be augmented by data stored and retrieved from the 'Knowledge' functional block. The data can be historical workflows (generated over the time or coming from external resources) or real-time workflows (continuously generated by the operations of the Decision stage).

3.1.2 DECISION ENGINE IN THE MONB5G ARCHITECTURE

As depicted in Figure 2, the considered architecture provides several degrees of freedom to deploy advanced decentralized AI algorithms for scalable and sustainable massive network slicing. In MonB5G, the **DE is part of slice runtime management and is embedded in the slice** (and is therefore part of the slice template). Next

Figure shows the generic structure of the MonB5G slice. Two separate layers can be distinguished: the Slice MonB5G Layer (SML) and the Slice Functional Layer (SFL).

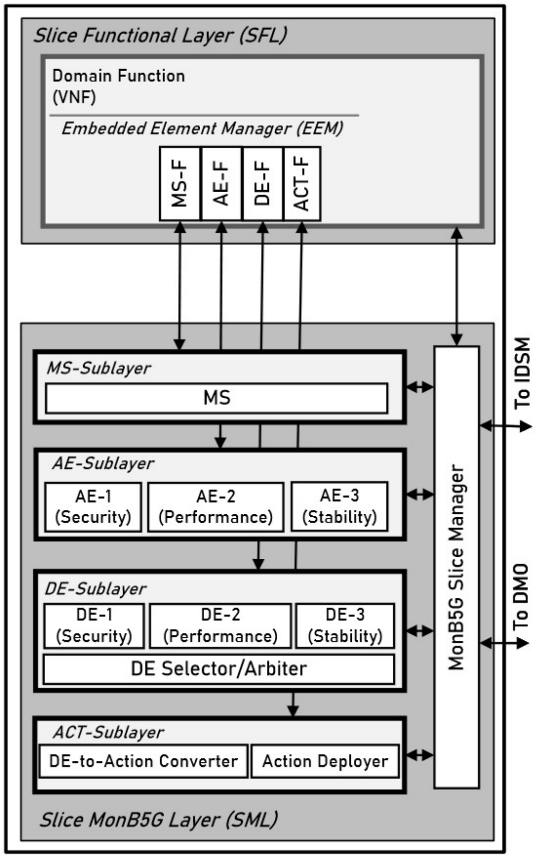


Figure 2. Generic structure of MonB5G single domain slice (composed of SFL and SML). It shows the MonB5G DE Positioning with MS, AE and ACT.

The SFL part, built of virtual functions, provides the communications service-related functionality, whereas the SML part, also built using virtual functions, provides the management. **The SML is further divided into different sublayers, which are responsible for Monitoring System (MS), Analytic Engines (AEs) and Decision Engines (DE), as well as the Actuators (ACTs).** The **MS** oversees collecting raw performance and configuration data measurements from the managed resources and contains an internal memory that stores the measurements as well as the decision story and the AI performance metrics, serving as the ‘Knowledge block’ in the ZSM in framework [1]. The **AE** then performs time-series predictions as well as feature space regressions, clustering, and classification to extract insights from the measurements collected by the MS, while the **DE** decides on the lifecycle management (LCM) actions that need to be applied to face the issues detected by the AE. **DEs implement control algorithms, such as Deep Reinforcement Learning (DRL) methods, to take online decisions regarding slice LCM actions** that need to be applied to face the issues detected by the AE and may propose network reconfiguration to solve the detected problems. It may also control the MS measurement granularity or the AE prediction parameters. Typically, in the management systems, multiple goals must be optimized, therefore multiple AEs and DEs can be part of SML. The DEs decisions are transferred to the **ACT**, which converts the DE decision into a set of elementary actions. Furthermore, actuators monitor the execution of this set of actions and provide feedback to the DE.

The DE sublayer in MonB5G is envisioned as a composition of multiple DE entities, which are responsible for the reconfiguration of the SFL and/or SML. Each of them may pursue a local goal, for instance, resource allocation optimization, local fault healing, security decisions or energy-aware reconfigurations. The DE sublayer interacts with the MS and AE sublayers to acquire appropriate information from the ongoing slice deployments and infrastructure utilization.

The control loop-based management system may generally have multiple goals what justifies, multiple AEs and DEs as a part of SML. This inevitably leads to conflicts between DEs' outputs. To solve the problem, the DE Selector/Arbiter component is present, which can be AI-driven. It is involved in the stability of the system observation and counteracts the ping-pong effect or chaotic system behavior (too many, random-like reconfigurations). The DEs have not only management, but also orchestration capabilities. They may change configuration parameters and send new function VNF orchestration requests. In the contrast to the reactive resource scaling, it may also proactively request an update of resource allocation or scaling, based on number of slice users' QoE, slice usage trends or spatial distribution changes. The **MonB5G DE and its serving administrative elements (MS, AE, and ACT) are in line with the ETSI ZSM CL framework [1].**

These three management components (MS/AE/DE) present different levels of instantiation that can be used according to the target scope of analysis and decision, where the aim is to minimize the raw data exchange and allow a fast local analysis and decision:

- Virtual network function (VNF)/physical network function (PNF) level, where MS, AE and DE are placed close to the monitored resources of a given slice.
- Slice-level, where each slice template includes dedicated AE and DE that may respectively collect compressed metrics/prediction models or policy weights from AEs and DEs of their VNFs/PNFs to perform slice-wide analytics and decision via decentralized AI strategies such as federated learning and multi-agent/federated deep reinforcement learning.
- Domain-level (i.e., RAN, Edge, Cloud), where a dedicated MS, AE and DE triplet can perform domain-wide AI-driven management by relying on the inputs of the VNFs/PNFs decentralized elements.
- Inter-Domain Manager and Orchestrator (IDMO)-level, which similarly to 3GPP Network Slice Management Function (NSMF), it manages the lifecycle of end-to-end network slices by carrying out global end-to-end analysis or decisions at both cross-slice and cross-domain levels. To that end, it includes high-level AE and DE that leverage local AE/DE inferences and actions to solve wide-scope issues.

3.2 Interfaces

Figure 3 illustrates the communication between the DE, the MS and the AE. It also includes the actuators (ACT), which are the entities that translate the DE LCM decision into API calls that involve the different slice components, i.e., VNFs, PNFs, links, etc., in each of the technological domains (RAN, Edge and/or Cloud) that the slice is supposed to cross.

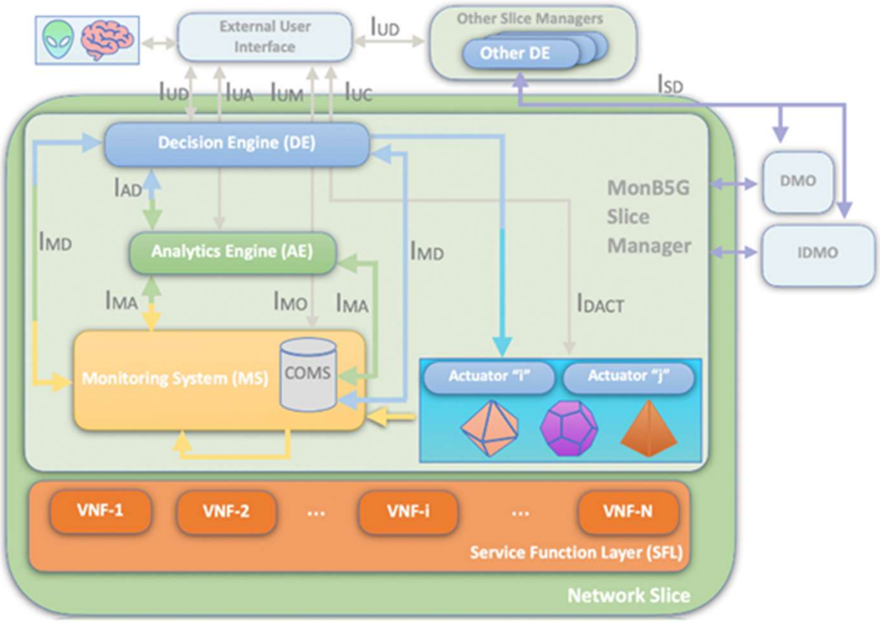


Figure 3. Interfaces between the different triplet MS-AE-DE elements and the actuators.

The different interfaces between the MonB5G DE and the rest of control blocks are presented in Table 3.1.

The MS in Figure 4 gathers different metrics from the managed system that the DE controls. This information can be passed to the AE and DE blocks directly, but it is also stored in a Common Online Memory Store (COMS), which is illustrated in the figure as a grey cylinder. The COMS entity is considered to avoid hard synchronization constraints between the MS-AE-DE functional blocks. Proceeding this way provides more flexibility to the AE and DE in terms of their length of processing, without compromising the granularity at which the MS can sample the monitoring data form the controlled system. Therefore, it is the MS that defines how fast the data is sampled. **It should be noticed that this COMS block is in line with the 'knowledge' functional block in the ETSI ZSM scheme** presented in Section 3.1.1.

The monitoring information is read by the AE from the COMS and processes it to provide insights to the DE (i.e., it performs predictions, classifications, etc.). The AE might also read the information directly from the MS, but this is expected to be in more punctual cases, where some synchronization is needed. The prediction interval needs to be set by from the Externa User Interface (EUI).

Like the Analytics Engine, the DE is expected to read its input from the COMS, even though, it is also possible to receive this information form the AE or the MS. Based on this information, the DE will take the corresponding decisions, which will be also stored in the COMS. In addition, it will issue them to the corresponding actuators, which are the entities that translate the decisions into API calls that implement the corresponding LCM decision in the managed entity.

Table 2. DE interfaces and the associated roles

Interface	Type	Role
I_{AD}	Tensors/Database query	DE Reads the predicted KPI from AE (either online or from COMS)
I_{MD}	Tensors/Database query	DE reads raw MS measurements (either online or from COMS)/Store AI metrics and DE decisions in COMS
I_{MA}	Tensors/Database query	AE reads raw MS measurements/Store AI metrics, predictions in COMS
I_{UD}	Database Query	EUI reads/Changes DE configuration (e.g., discount factor of a DRL algorithm)
I_{UA}	Database Query	EUI reads/Changes AE configuration (e.g., prediction interval, learning rate)
I_{UM}	Database Query	EUI reads/changes MS configuration (e.g., granularity)
I_{UC}	Database Query	EUI reads/changes actuation configuration (e.g., API primitives' parameters)
I_{DACT}	REST API Call	DE sends decisions to Actuators

Cross-domain operations between local DEs (i.e., DEs of each technological domains) or with end-to-end DE passes through the IDMO (Inter-Domain Manager and Orchestrator), while operation between inter-slice DEs is ensured by the DMO.

3.3 Cross-Domain Operations

A MS/AE/DE triplet is present at each technological domain (e.g., RAN, edge and cloud). Furthermore, a system might have more instances of the triplet, depending on the control granularity needed for the management system. Namely, instances of the MS/AE/DE triplet could be per slice, tenant, domain, infrastructure provider, etc. Given the fact that there might be multiple instances of the Decision Engine, the design of the DE and its corresponding interfaces need to be agnostic to the technological domain or system layer in which the DE is deployed. Furthermore, the DEs must cooperate to achieve the corresponding end-to-end KPIs the system optimization targets.

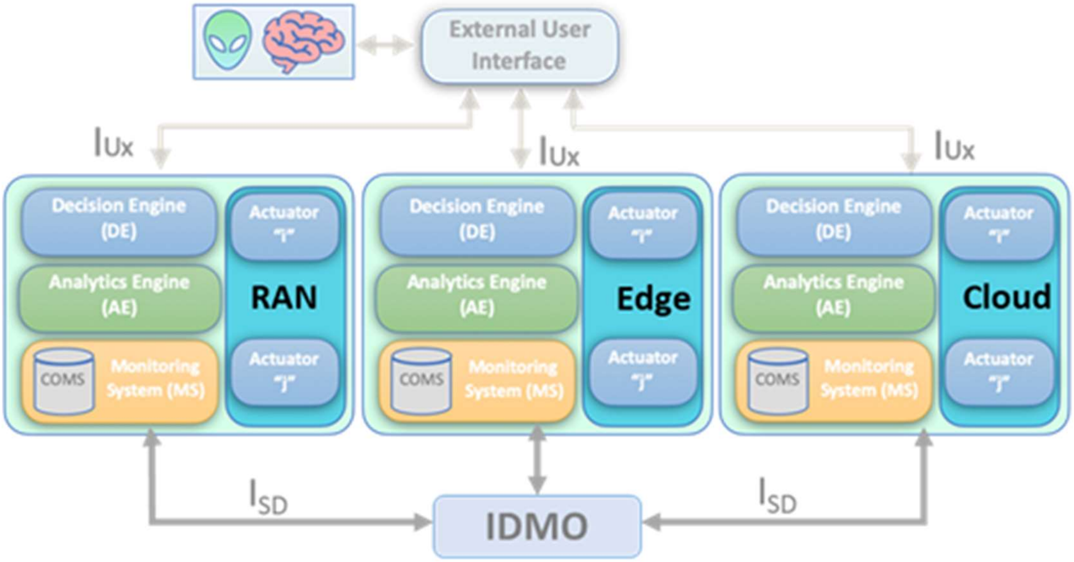


Figure 4. MS/AE/DE triplet instances at different technological domains

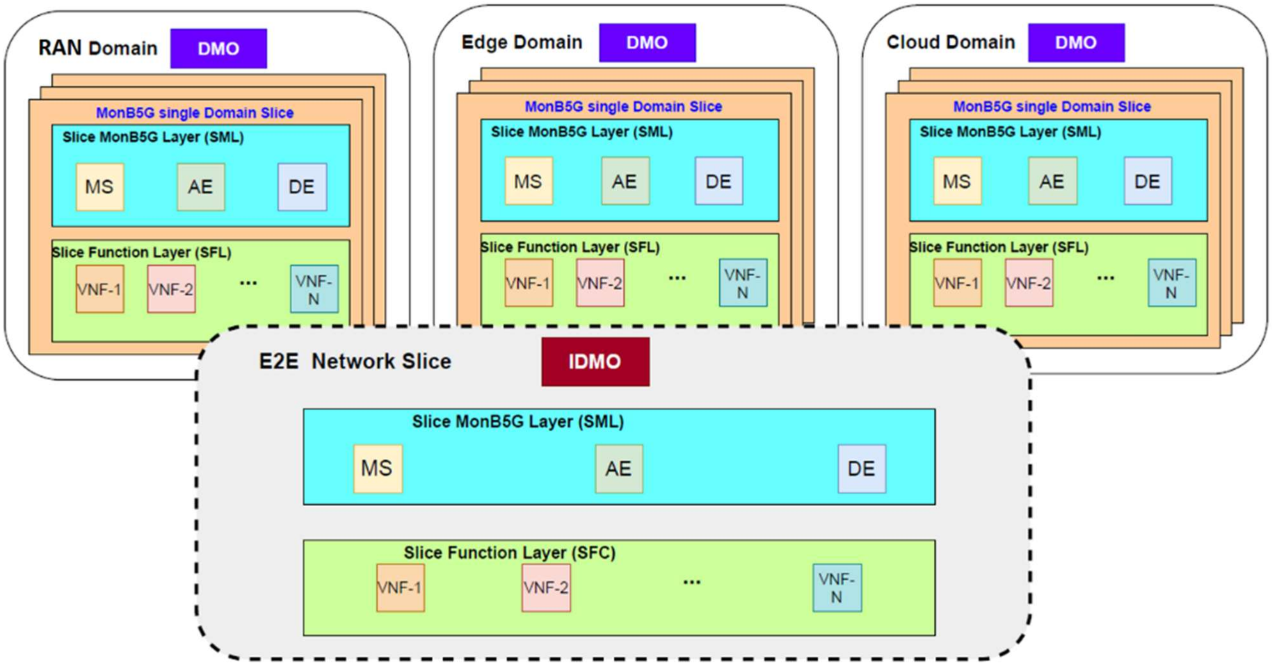


Figure 5. Distributed sliced MonB5G Architecture with multiple domains and slices.

The distributed architecture that MonB5G envisions (see Figure 5**) is specially well suited for different types of decentralized AI-native approaches**. To exemplify this, hereby, two of these AI/ML approaches are exposed:

- **Federated Deep Reinforcement Learning.** In this scheme, the local DEs are built upon either value-based DRL (e.g., DQN) or policy-based DRL (e.g., A2C, DDPG, SAC). The key idea is that, as long as the action/state spaces of the different DEs are homogeneous, one can dramatically improve the decision accuracy, and leverage the experiences learned in e.g., other domains or slices, by letting the local DEs exchange their weights of the embedded decision actor/critic or Q networks, with the E2E DE, which performs a custom averaging to generate a more accurate model. Next, the E2E broadcasts it to the DEs, updating their local networks and improving their decision. This approach avoids exchanging raw data between local and the E2E domain.
- **Multi-Agent Deep Reinforcement Learning.** Local DEs play the role of Actors and can learn policies, using only local information (i.e., their own observations in local domain, slice, etc) at execution. On the other hand, the Critic is in the E2E domain. This allows the policies to use extra information to ease training, so long as this information is not used at test time. Within this context, actor-critic policy gradient methods, where the critic is augmented with extra information about the policies of other agents, are suitable to implement a decentralized decision leveraging MonB5G architecture.

4 Slice Admission Control

4.1 Introduction

In D4.1, we identified several potential algorithms for slice placement, some of which are based on heuristics and others on ML. In this deliverable, we report on two algorithms for the placement of slices in a network.

Specifically, we present two slice admission approaches in this chapter. The first is based on Deep Reinforcement Learning and a heuristic. The Power of Two choice (P2C) heuristic was introduced in D4.1. This heuristic proves to be very efficient in placing slices but requires extensive signalling (due to the availability of polling resources in data centres). Therefore, a DRL approach was introduced, namely the A3C DRL algorithm. This algorithm is centralized and can place slices when the rewards are based on the ratio of placed slices. However, the learning period can be very long, and performance can be limited when the arrival rates of slice changes. In contrast, P2C is very resilient. To combine the efficiency of heuristics and the sobriety of DRL in terms of signalling, we developed the Heuristically Assisted DRL (HA-DRL) approach, which achieves very good performance when the arrival rate of slices varies. The second approach, TASAC, is based on the Time-of-Day (ToD) traffic curve. This curve, which is typically related to human activity, was used to estimate the daily eMBB traffic, and based on this, the resource consumption of the eMBB slices was predicted. The second category of slices analysed in this case is mMTC. Unlike eMBB, the traffic generated by mMTC slices does not depend on the ToD. The approach is using ToD predictions based on actual observations. The DQN algorithm was used for admission control.

4.2 Heuristically assisted DRL for slice placement

4.2.1 *PROBLEM DESCRIPTION*

We consider a network consisting of a hierarchical cloud infrastructure, as is the case in many operational networks. The lower level consists of edge data centers connected to fog (also referred to as core data centers), which provide an intermediate-level cloud infrastructure; the fog data centers themselves are connected to a central cloud infrastructure. This global cloud infrastructure together with the transport network (consisting of transmission links as well as the switching/routing capacities to interconnect the data centers), form the substrate network, referred to below as the Physical Substrate Network (PSN). An illustration of the PSN considered in the following is given in Figure 6. Each data center consists of servers with RAM and CPU capacities, which are the scarcest resources in clouds, interconnected by switches and routers, connected to the transport network. Importantly users making slice requests are connected to the data edge data centers. In other words, when a slice request is issued by a user, the slice is rooted in the sense that the user of the slice is connected to an edge data center.

The cloud infrastructure is used to host slices. A slice request is called a Network Slice Placement Request (NSPR), which can be described by a graph with vertices (the compute nodes needed to execute the service offered by the slice and the links for exchanging messages between these compute nodes). In the following, a compute node will be referred to as VNF, which should be understood in a broad sense (a Virtualized Network Function can be hosted in a container, not necessarily in a virtual machine). We also assume that a slice is a linear chain of services (VNFs) that exchange messages.

The problem of network slice placement can be formulated as follows: Given an NSPR $\mathrm{G_v} = (V, E)$, arriving at time $t_a \in [0, T]$ and exiting the network in exit date $t_e \in [t_a, T]$ and a PSN graph $G_s = (N, L)$, find a mapping $h: G_v \rightarrow \bar{G}_s = (\bar{N}, \bar{L})$, $\bar{N} \subset N$, $\bar{L} \subset L$, where *T* is the length of the lifetime of the system. The mapping must take into account the following constraints:

- the CPU and RAM capacities (cap_s^{cpu} and cap_s^{ram}), where *s* is in the set of servers of the PSN and requirements (req_v^{cpu} and req_v^{ram}), where *v* is in the set of compute nodes of the NPSR,
- the bandwidth capacities ($cap_{(a,b)}^{bw}$), where *(a,b)* is the link between servers *a* and *b* in the NSP, and requirements ($req_{(a,b)}^{bw}$), where *(a,b)* is the link between compute nodes *a* and *b* in the NPSR,
- the link latency capacities ($\delta_{(a,b)}$) and requirements ($req_{(\bar{a},\bar{b})}^{lat}$),
- the access latency capacities (α_s) and requirements (α_{max}^c),
- the E2E latency requirements (δ_c).

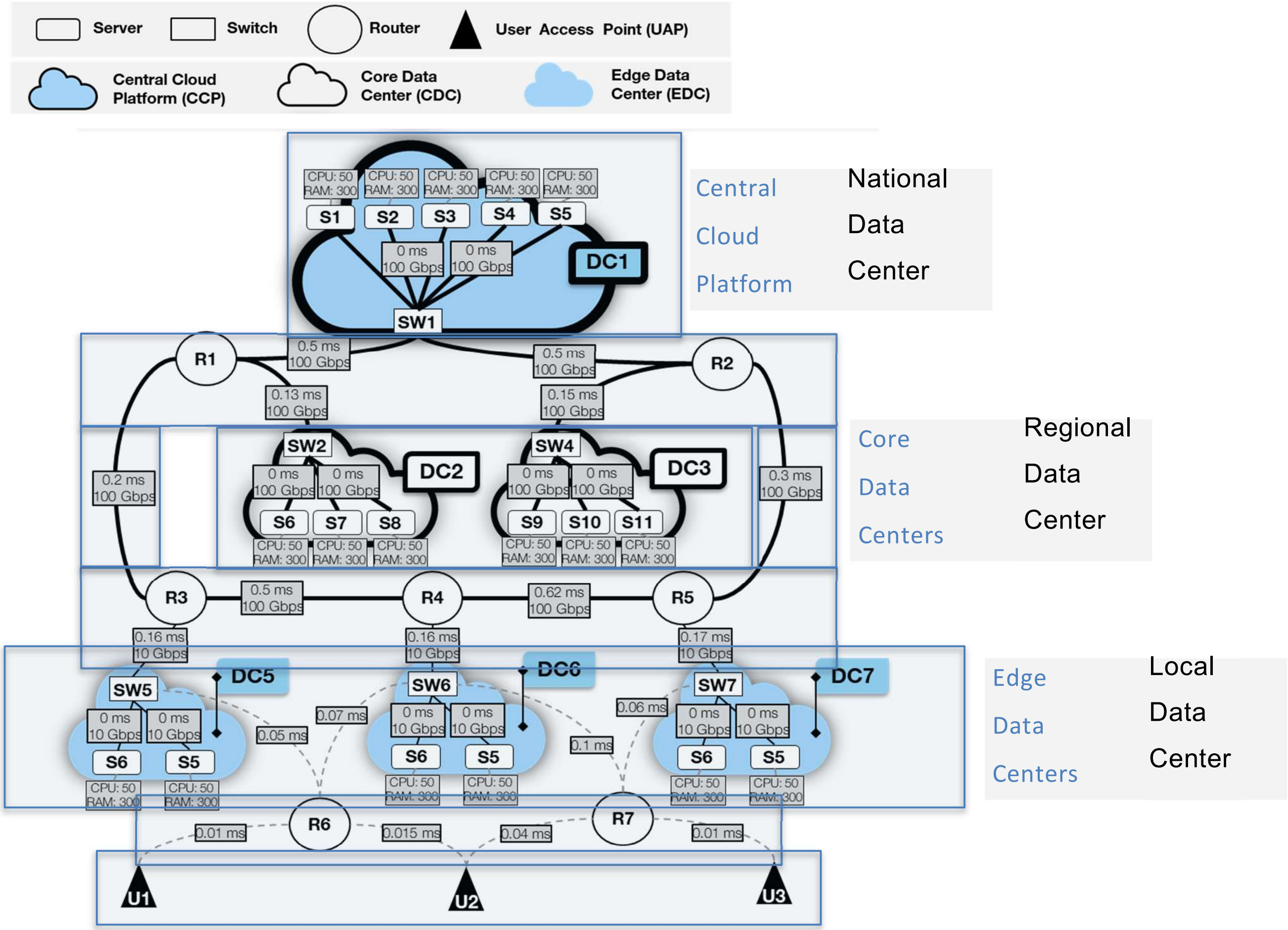


Figure 6. Physical Substrate Network (PSN)

Several optimization criteria can be envisaged for the mapping *h*:

- maximization of slice acceptance ratio,
- minimization of total resource utilization,
- maximization of node load balancing.

Each problem can be formulated by mean of an Integer Linear Programming problem (ILP). More precisely,

- The placement of VNF (compute node) is controlled by the decision variables $x_s^n \in \{0,1\}, \forall r \in R, \forall n \in N^r, \forall s \in S$,
- The VNF chaining: $y_{(a,b)}^{(\bar{a},\bar{b})} \in \{0,1\}, \forall r \in R, \forall(\bar{a},\bar{b}) \in E^r, \forall(a,b)$,
- NSPR acceptance: $z^r \in \{0,1\}, \forall r \in R$

where R *is the set of* requests, N^r is the set of VNFs of each request, E^r is the set of VLs of each request.

To solve the placement of slices must account of several constraints (see [3] for details):

- The VNF placement and resource capacity constraints read

$$\forall r \in R, \quad v \in V^r, \quad \sum_{s \in S} x_s^v = 1$$

$$\forall t \in T_a, \forall s \in S, \quad \sum_{r \in R} \sum_{v \in V^r} req_v^{cpu} m_t^r x_s^v \leqslant cap_s^{cpu}$$

$$\forall t \in T_a, \forall s \in S, \quad \sum_{r \in R} \sum_{v \in V^r} req_n^{ram} m_t^r x_s^v \leqslant cap_s^{ram}$$

$$\forall t \in T_a, \forall(a,b) \in L, \quad \sum_{r \in R} \sum_{(\bar{a},\bar{b}) \in E^r} req_{(\bar{a},\bar{b})}^{bw} m_t^r y_{(a,b)}^{(\bar{a},\bar{b})} \leqslant cap_{(a,b)}^{bw}.$$

- The network slice latency requirement constraints are

$$\forall r \in R, \forall(\bar{a},\bar{b}) \in E^r, \quad \sum_{(a,b) \in L} \delta_{(a,b)} y_{(a,b)}^{(\bar{a},b)} \leqslant req_{(\bar{a},\bar{b})}^{lat},$$

$$\forall r \in R, \forall c \in C^r, \sum_{s \in S} \alpha_s^r x_s^{v_{root}^{r,c}} \leqslant \alpha_{max}^{r,c},$$

$$\forall r \in R, \forall c \in C^r, \sum_{s \in S} \alpha_s^r x_s^{v_{root}^{r,c}} + \sum_{(a,b) \in L} \sum_{(\bar{a},\bar{b}) \in c} \delta_{(a,b)} y_{(a,b)}^{(\bar{a},\bar{b})} \leqslant \delta_c^r.$$

- The eligible physical path calculation is

$$\forall a \in S, \; \forall r \in R, \; \forall(\bar{a},\bar{b}) \in E^r \qquad \sum_{\substack{b \in N: \\ (a,b) \in L}} y_{(a,b)}^{(\bar{a},\bar{b})} - \sum_{\substack{b \in N: \\ (b,a) \in L}} y_{(b,a)}^{(\bar{a},\bar{b})} = x_a^{\bar{b}} - x_a^{\bar{a}},$$

$$\forall a \in N \setminus S, \forall r \in R, \forall(\bar{a},\bar{b}) \in E^r, \quad \sum_{\substack{b \in N: \\ (a,b) \in L}} y_{(a,b)}^{(\bar{a},\bar{b})} - \sum_{\substack{b \in N \\ (b,a) \in L}} y_{(b,a)}^{(\bar{a},\bar{b})} = 0,$$

$$\forall(a,b) \in L, \forall r \in R, \forall(\bar{a},\bar{b}) \in E^r, \; y_{(a,b)}^{(\bar{a},\bar{b})} + y_{(b,a)}^{(\bar{a},\bar{b})} \leqslant 1.$$

The minimization of total resource utilization corresponds to the problem

$$\min_{x,y} \sum_{r\in R} \sum_{(\bar{a},\bar{b})\in E^r} \sum_{(a,b)\in L} req^{bw}_{(\bar{a},\bar{b})} y^{(\bar{a},\bar{b})}_{(a,b)}$$

and the maximization of slice acceptance ratio to

$$\max_{x,y,z} \sum_{r\in R} \frac{z_r}{|R|} \qquad \text{with} \qquad \begin{aligned} &\forall r \in R, \forall v \in V^r, && z_r \leqslant \sum_{s\in S} x^v_s, \\ &\forall r \in R, && z_r \geqslant \sum_{s\in S} \sum_{v\in V^r} x^v_s - |V^r - 1|. \end{aligned}$$

The above optimization problems can be solved by using CPLEX. Therefore, a heuristic has been proposed in [4], which is based on the Power of the 2 choices (P2C). This heuristic runs as follows: For each VNF *v* of the NSPR,

- Calculate set S' of eligible servers
- If S' not empty, select 2 candidate servers S1 and S2 to place VNF *v,*
  - Calculate substrate network paths P1 and P2 to interconnect VNF *v,*
  - If P1 and P2 are feasible, choose the shortest one,
  - If only P1 is feasible, map virtual link to P1,
  - If only P2 is feasible, map virtual link to P2,
  - If no path is feasible, reject NSPR,
  - If S' is empty, reject NSPR,
- Accept the NSPR

To select servers, we consider two policies: full random (policy 1) and rand with priority to cloud and core servers (policy 2).

To evaluate the efficiency of the above heuristic, we conducted two online optimization experiments on the Grid 5000 infrastructure for two objectives: minimal bandwidth consumption (referred to as ILP1 for the ILP solved with CPLEX) and maximum acceptance ratio (denoted as ILP2 for the ILP solved with CPLEX). Two server selection policies were also implemented for the heuristic, denoted as P2C1 for policy 1 and P2C2 for policy 2. We considered three slice categories: Best Effort (BE), uRLLC and eMBB, with the following mix of traffic: 67% BE, 22% eMBB and 11% uRLLC. The baseline PSN includes 126 nodes and is expandable to 16128 nodes. Each NSPR is a sequence of 5 VNFs chained by virtual links, both with specific requirements. Figure 7 shows that the execution time for the ILPs increases rapidly with the number of servers, while the increase for the P2C heuristics is limited. Figure 8a) shows that P2C has the best final blocking ratio in the uRLLC scenario, while this policy concentrates the load on cloud and core data centers to offload the edge data centers. Figure 8b) displays the resource consumption. We note that the edge and core data centers are highly occupied while the cloud one is almost not used. This is since VBNFs are placed close to end users to limit latency.

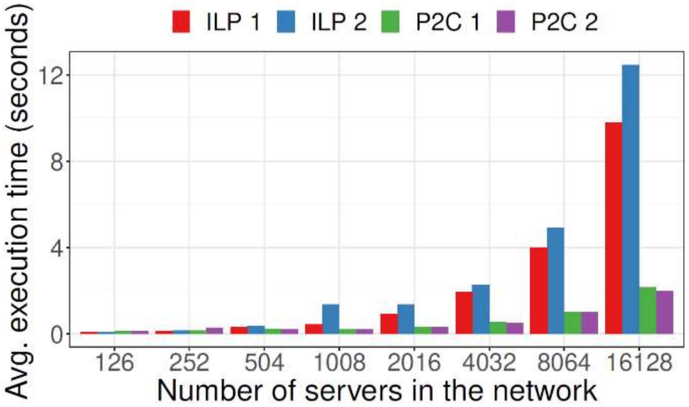


Figure 7. Execution times of the algorithms.

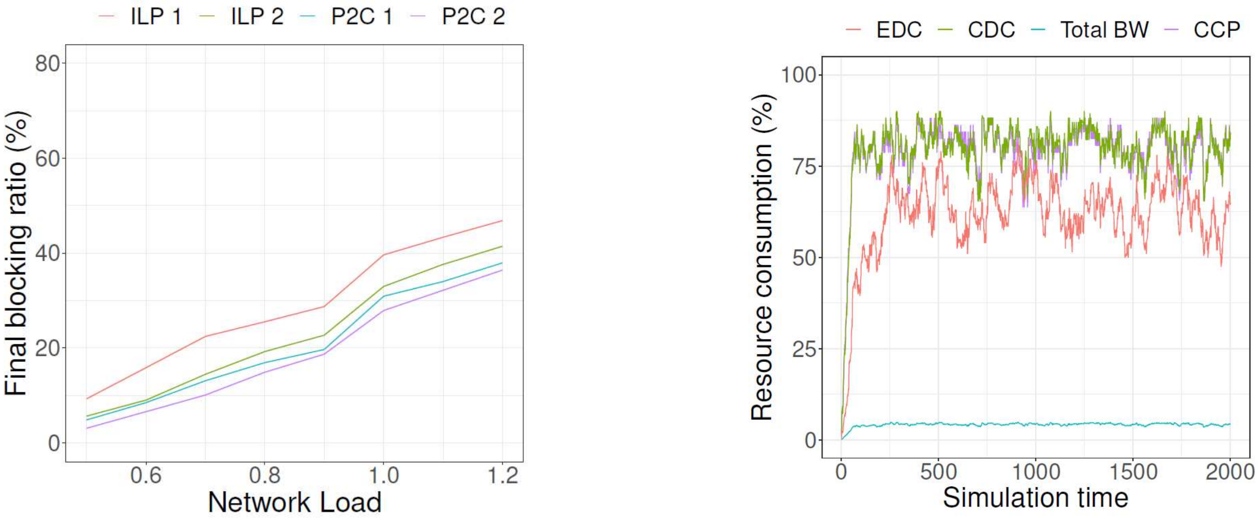


Figure 8. Blocking rates and data center resource consumption.

The P2C proves to be very efficient to place slice requests and is important in a distributed environment. However, we need to select two servers for each request and check the availability of resources on each server. To circumvent this process, which can be very costly in terms of signaling messages, we have developed an alternative method based on DRL that partially relies on the P2C heuristic.

4.2.2 *SOLUTION TOOL*

Some recent approaches to network slice placement are based on Deep Reinforcement Learning (DRL), see [5] for a state-of-the-art review of DRL applied to slice placement in 5G networks. However, these algorithms are very tedious to train, typically several days, which this training period is not compatible with highly varying network conditions. Therefore, we developed an alternative approach that combines DRL (namely the A3C algorithm [6]) and the P2C heuristic presented in the previous section. The architecture of the algorithm is shown in Figure 9. To account for non-stationary conditions, we introduced an additional module to the classical DRL algorithm that describes the state (namely, the load of the network). The resulting algorithm is referred to as eDRL.

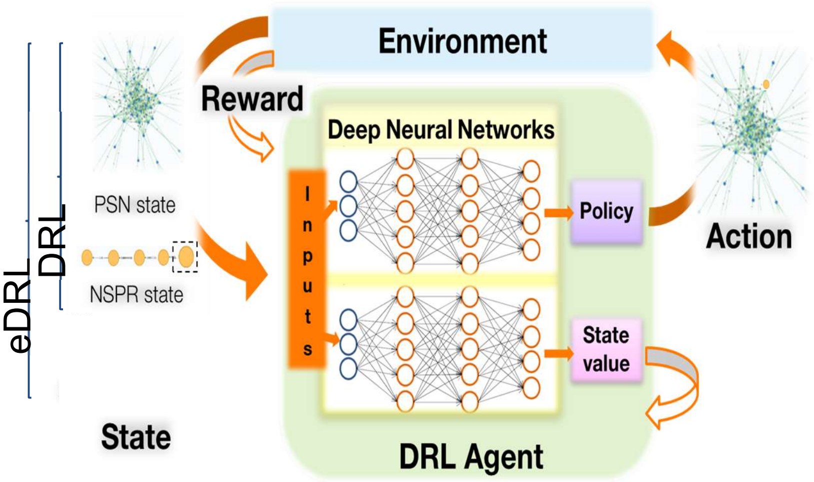


Figure 9. Architecture of the algorithm (initial DRL and enhanced DRL accounting of the network state).

To develop our approach, we make the following assumptions. NSPR are undirected path graphs (namely, service chains). A training step is the placement of one VNF of an NSPR and the mapping of its associated Virtual Links (VLs). At each training step *t*, we select the server to place the VNF based on the policy and map VLs, record the reward, and update the PSN. At the end of a training episode, we compute the loss function based on the cumulated rewards and state values. Finally, we update the weights of the network using the gradients of the loss functions (see [5] for more details). The architecture of the algorithm is shown in Figure 10, the various components are assembled from open-source software to build the algorithm. The PSN is abstracted in the form of a Graph Convolutional Network (GCN). The function Z_Θ maps each state and action to a real value; the details are given in [5]. This function is provided by the Actor Critic algorithm, as illustrated in Figure 10.

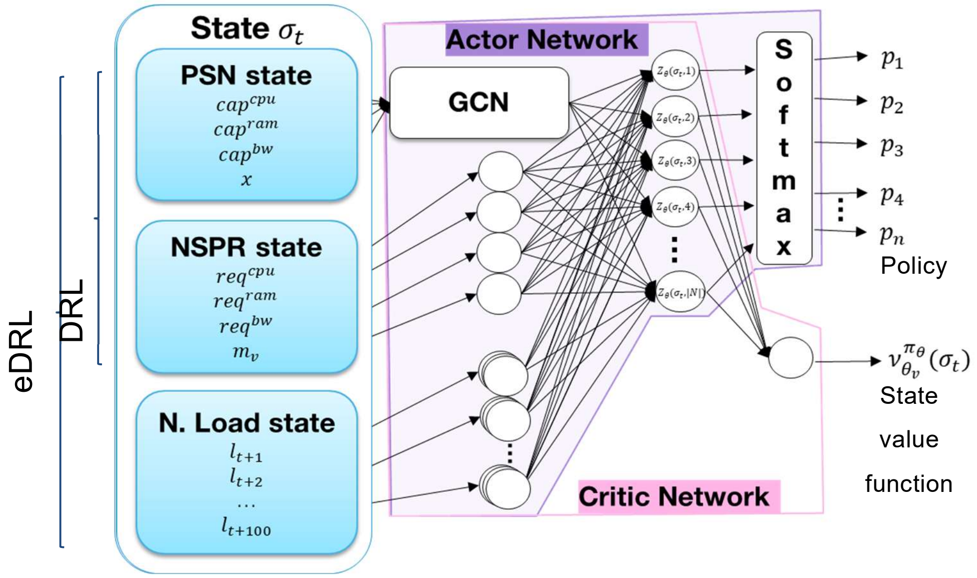


Figure 10. DRL (based on A3C algorithm) and eDRL applied to slice placement.

The preliminary evaluation of the algorithm has shown that the above algorithm is inadequate for nonstationary network loads. Therefore, we introduce a modification in the computation of the *Z*-function by using the P2C heuristic (the formula is given by Equation (21) in [5], where a coefficient β is used to modulate the weight of the heuristic in the modification of the *Z* function). The principle of the so-called Heuristically Assisted DRL (HA-DRL) is shown in Figure 11.

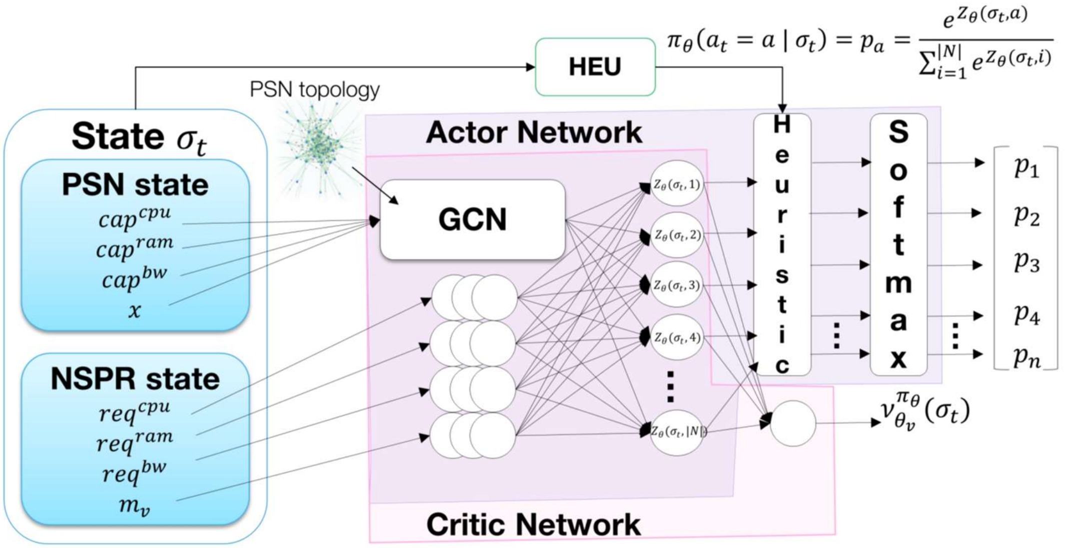


Figure 11. Heuristically Assisted DRL algorithm.

An example of the HA-DRL is illustrated in Figure 12. We have the following steps explains how the heuristic modifies the decisions taken by the DRL algorithm (see [5] for details):

Initial Z function is $Z(s_t,.) = [1.0,0.6,0.9,1.2]$

DRL Policy gives $\pi^{\theta}(.|s_t) = [0.26,0.18,0.24,0.32]$

Action calculated by HEU (the heuristic function) $HEU(s_t) = 2$

Heuristic Function $H(s_t,.) = [0.0,0.6,0.0,0.0]$

New Z function $Z(s_t,.) = [1.0,1.3,0.9,1.2]$ for $\beta = 1$, $\boldsymbol{\xi} = 1$, $\boldsymbol{\eta} = 0.1$

DRL+HEU Policy $\pi^{\theta}(.|s_t) = [0.22,0.30,0.20,0.27]$

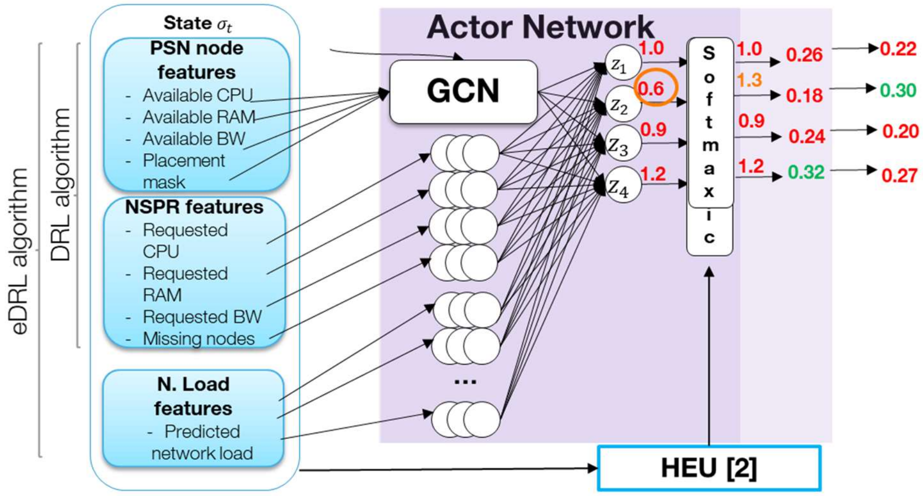


Figure 12. HA-DRL (the Z function is modified by the heuristic).

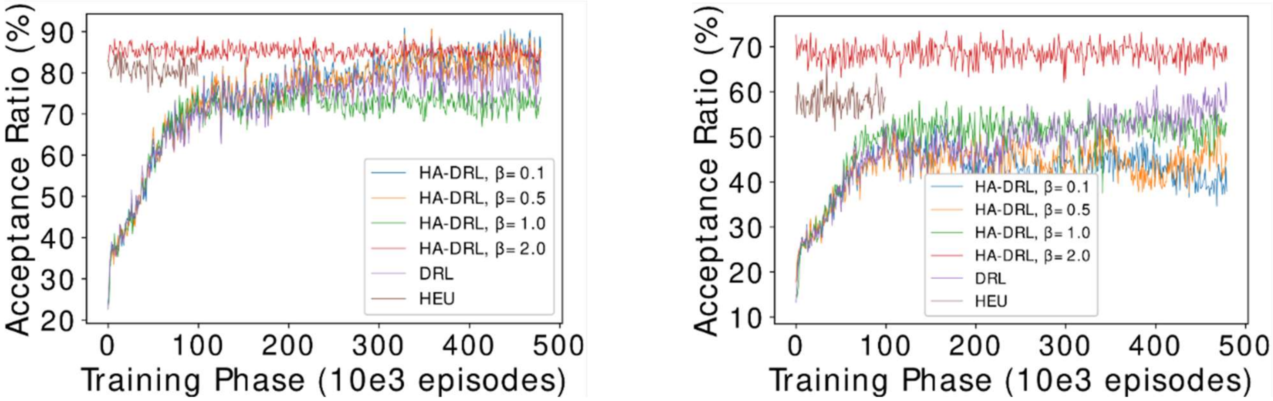
4.2.3 FIT INTO THE DE DISTRIBUTED ARCHITECTURE

The algorithm presented in the previous section is centralized but the P2C heuristic is compatible with a distributed architecture. Upon receiving an NSPR, the ingress node (access domain) can run the P2C algorithm to decide about the placement of VNFs. If placement in the ingress domain is not possible, the remaining NSPR is forwarded to the fog/core domain and finally to the cloud. Then each domain can make local decisions. These decisions can then be forwarded to the central HA-DRL algorithm for better slice placement. A fully distributed algorithm is planned for further studies.

4.2.4 VALIDATION RESULTS

To test the performance of the algorithm presented in the previous section, we performed simulations. The global framework of the simulation is shown in Figure 12. The NSPRs arrive following a Poisson process. We considered three scenarios. The first is when the arrival rate is constant. In the second scenario, the arrival rate is sinusoidal to reflect the daily fluctuations and in the third scenario, there is a sudden termination of the arrival process. This is to check the capability of the algorithm to adapt to non-stationary conditions. The exact simulation settings can be found in [5], Section VII.A. NSPR are chains of VNFs to be placed on the PNS with bandwidth requirements between the VNFs.

Examples of simulation results are given in Figure 13, showing that all DRL algorithms can learn the best placement for slices and that they all converge to an acceptable acceptance ratio after some time. The HA-DRL with β = 2 appears to converge the fastest and provides the same performance as the P2C heuristic (HEU). However, further results (see [5]) show that HA-DRL with β = 2 strongly depends on HEU and performs worse after the training phase for traffic changes. This suggests that HA-DRL with β = 2 should always be run in combination with HEU. The conclusions are very similar in the case of periodic variations in network load.

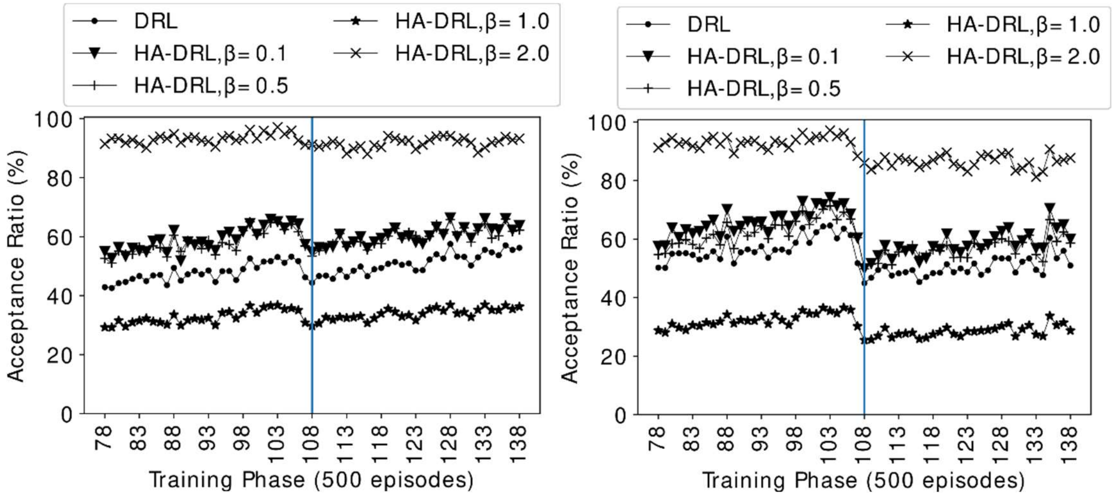


Network load = 80%

Network load = 100%

Figure 13. Slice acceptance ration for HA-DRL, DRL and the heuristic.

To test the robustness of the algorithms to sudden changes in traffic load, we ran simulations in which the arrival rate of NSPR was increased by a certain percentage. The results are shown in Figure 14. Further results can be found in [7].



10% increase

30% increase

Figure 14. Acceptance ratio of the algorithm in case of load increase at training episode 108.

4.3 Time-of-day aware slice admission control

Network slicing in 5G networks allows multiple tenants to allocate and share the underlying network resources of an Infrastructure Provider (INP), that owns the NPS presented in the previous section. Each tenant generates slice requests specifying the required resources, and the INP collects revenue from the tenants for hosting the admitted slices. Slices are typically classified as elastic and inelastic, with flexible and rigid service guarantee requirements. The INP decides whether to accept or reject the incoming slice requests by considering certain criteria. This decision helps the INP to efficiently manage its underlying resources, achieve fair resource allocation among different slice types, and increase its revenue while minimising SLA violations of admitted slices [8]. Typically, Slice Admission Control (SAC) decisions are based on the maximum resource requirements of the given slice (fixed number of resources). There are numerous research activities that propose various SAC algorithms, admission policies, resource allocation, resource usage prediction, and slice modelling [8]. However, defining an efficient resource management strategy is problematic considering that real network data related to slice orchestration processes does not exist yet.

Several approaches using RL techniques have already been proposed. A framework for slice brokering called RL-NSB is presented in [10]. The authors formulate the admission control as a NP-hard, geometric two-dimensional Knapsack problem and propose two heuristic algorithms to deal with the regular and irregular arrival of network slice requests. In [15], Q-Learning (QL) and DQL algorithms are proposed to solve the SAC and Resource Allocation (RA) problems. DQL achieves over 3.6% more gain than QL and the baseline methods (7.9% for the Node Ranking algorithm and 11.7% for the Always Admit Request algorithm). The advantage of using RL in deriving SAC policies was demonstrated in [9], where QL, DQL and the online algorithm called Regret Matching (RM) are used to maximise the operator's revenue from deployed slices. Each of the algorithms provided satisfactory results, with the highest financial gain obtained with RM, presumably due to the regret formula, which allows faster policy adaptation in case of negative rewards. Some efforts that use Markov Chain Process to model the SAC problem can also be found in the literature [14]. In [13], an analysis of a system modelled as a Semi-Markov Decision Process and the optimisation of the infrastructure provider's revenue, and the design of Deep Reinforcement Learning that maximises revenue and QoS control was presented. The authors of [18] present a slice admission strategy based on RL in the presence of services with different priorities. The considered use case is a flexible 5G RAN, where slices from different mobile service providers are virtualised over the same RAN infrastructure. The proposed policy learns which services have the potential to provide high profit (i.e., high revenue with low degradation) and should therefore be accepted.

However, most approaches to slice admission do not explicitly consider the dependencies between the time-of-day (ToD) of slice operation and the number of consumed resources but focus only on the frequency of arriving slice requests. This relationship between resources, demands and time has been observed and modelled in previous generations of fixed and mobile networks, and can also be observed on the Internet. An example of such a curve is shown in Figure 15. The behaviour is mainly driven by human-related traffic and depends on time-of-day or day-of-week. The approach has been widely used in the past for planning the network in case of non-elastic traffic (voice calls, etc.). In [11], the authors propose the stochastic Markov traffic model derived from the traces obtained from the LTE scheduling information acquired in four locations of one of the European metropolitan cities.

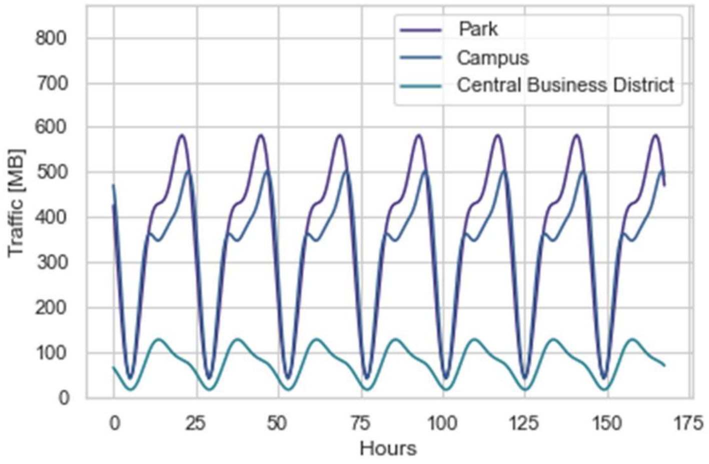


Figure 15. Time-of-day traffic curves for several areas (example)

The other approach is proposed in [12], where the sinusoidal superposition model is introduced. The model is evaluated using the real data from a dense urban area (park, university campus and business district) in a city in China and has demonstrated good accuracy. The ToD resource consumption can be efficiently exploited in SAC:

- For long-lived slices (duration > 24h), by predicting the ToD peak, it is possible to reduce the probability of congestion during peak hours.
- For short-lived slices (duration < 24h), it is possible to allocate more resources depending on the ToD; the SAC can be more aggressive with descending slope of ToD and less aggressive with the ascending one.

- For short-lived slices, it is possible to use the slice-calendaring mechanism, i.e., proposing the users shift their requests (if possible) to off-peak times. Such a mechanism needs an incentive, which in the past, in the case of the fixed network voice calls, was fiscal.

The above mechanisms lead to a much more efficient and realistic SAC than approaches that ignore the ToD resource consumption pattern. In this section, we describe and demonstrate the behaviour of a SAC approach for long-lived network slices that takes ToD into account.

4.3.1 TASAC CONCEPT DESCRIPTION

To propose a new SAC mechanism based on ToD, it is necessary to make some assumptions about slice-related traffic. The consumption of resources of each network slice depends on two main factors:

- Resources needed for the deployment of the slice (i.e., slice footprint) and its primary operations in an "idle" state.
- Resources needed to support the activities of the terminals connected to the slice, some of which may be human-operated.

The human-related network traffic is ToD dependent - at night, networks are typically lightly loaded only. Unlike human-operated terminals, IoT devices or Industry 4.0 terminals and security cameras can transfer a similar amount of data in a 24/7 cycle. Therefore, their activity is independent of ToD and time-of-week. In our analysis, we consider both types of traffic and refer to the first as eMBB traffic and the second as mMTC traffic. Each request must be specified in terms of the above slice types and additional parameters.

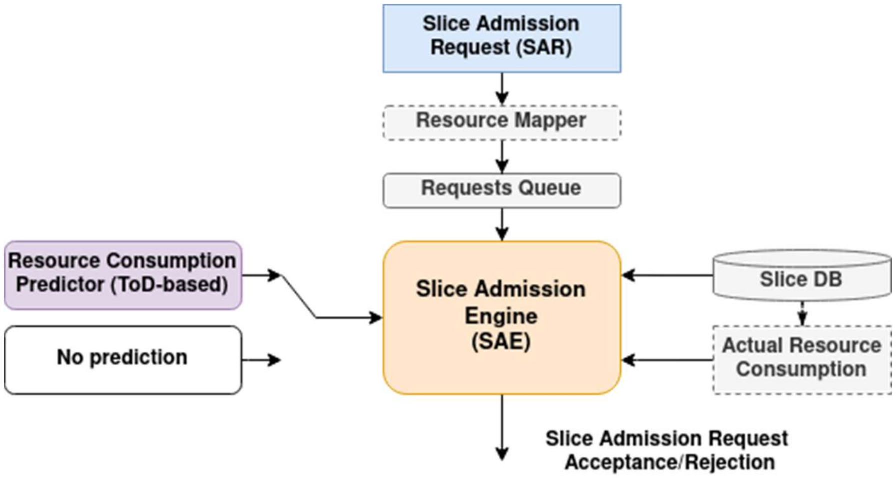

Slice Admission Request (SAR)
Resource Mapper
Requests Queue
Resource Consumption Predictor (ToD-based)
No prediction
Slice Admission Engine (SAE)
Slice DB
Actual Resource Consumption
Slice Admission Request Acceptance/Rejection

Figure 16. Concept of ToD aware Slice Admission (TASAC).

The ToD Aware SAC (TASAC) concept is depicted in Figure 16. It is compliant with the MonB5G architecture. Slice tenants send Slice Admission Requests (SARs) via the MonB5G Portal that contain the high-level information of the desired slice (i.e., duration, deployment time, capacity). The high-level intent is transferred to the Resource Mapper entity, whose task is to map and scale the generic requirements into the slice resource requirements, priorities, etc. (a similar approach to GST/NEST [20] according to the capacity preferences and slice type. Then, the mapped slice request is placed into Requests Queue (priority queue) to be analysed by the Slice Admission Engine (SAE). The decision regarding slice admission/rejection is taken based on the following factors:

- Actual resource consumption - aggregate resource utilisation data on system capacity provided by the MS layer.
- Current system status - information about currently deployed slices in the network, their types, peak resource requirements etc. This information is stored in the Slice Database and is updated when the system status changes (SAR acceptance, slice termination).
- Prediction of resource consumption during the period of interest - it is assumed that different prediction techniques can be used to produce accurate forecasts under different circumstances and system states.

The slice admission/rejection policy can vary depending on the implemented SAE behaviour. In TASAC, the DQN [16] approach is used to derive the optimal slice admission policy.

The TASAC concept includes the component responsible for mapping high-level tenant requests, but this is a different problem from SAC and is omitted in this section. The operations of TASAC require ToD prediction. For this purpose, the Holt-Winters model-based resource prediction [17] can be used. The model predicts a typical value (average), a slope (trend), and a cyclical repeating pattern (seasonality). Another practical approach is a set of so-called ETS (Exponential Time Smoothing) models, i.e., a bundle of time series models with an underlying state-space model consisting of a level component, a trend component (T), a seasonal component (S), and an error term (E) [21].

4.3.2 TASAC ALGORITHM

The resources need for slice deployment are typically split into computing, storage, and connectivity resources. Without the loss of generalization, we will focus on a single resource type only: connectivity, i.e., bandwidth. The total resource pool in the system, denoted as R, describes the available bandwidth in the network (we take the value as a scalar). Each SAR k ∈ K is described by a tuple ⟨bkr, qk , tkd , tkt⟩, which defines the maximum requested bandwidth by the slice bkr, the slice type qk, slice deployment tk0 and termination tkt time. Each SAR arrives with a λ rate, and when accepted, its metadata is stored in slice database i.e., k ∈ K. The optimization goal is to maximise bandwidth consumption, for eMBB-like (qk=0) slice of which resource consumption is dependent on ToD, and mMTC-like (qk = 1) resource consumption that is constant during the whole mMTC slice lifetime

The generic concept of DQN is presented in Figure 17.

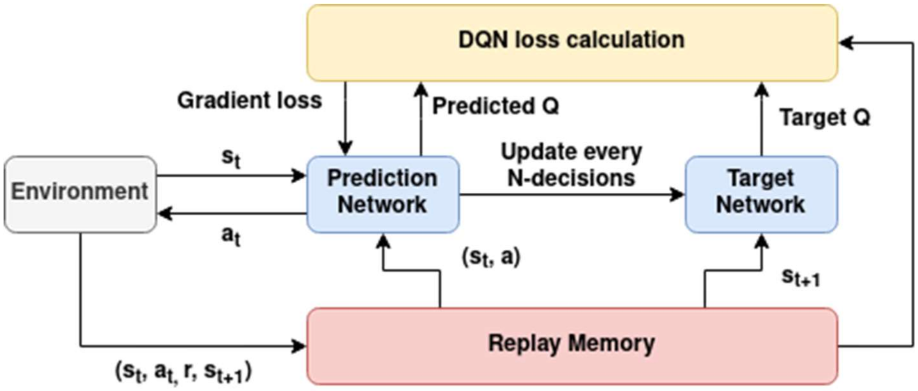


Figure 17. High-level concept of DQN

The DQN approach is based on the Q-learning method. The main idea behind the Q-learning is that if we consider a function *Q*: State x Action −→ R,* which provides the information regarding the return based on the current state and the adopted action. Knowing such a function enables easy construction of the policy that could maximise the rewards of the actions *(π*(s): argmax Q∗(s, a).* For complex environments, the *Q** function cannot be derived analytically. The DQN approach solves this issue by leveraging Deep Neural Networks to approximate the *Q** function. The DQN agent consists of 4 blocks: the Prediction Network, the Target Network, the DQN loss calculation block and the Replay Memory. Each time the agent needs to decide, it makes it based on the information about the network state s_t obtained from the environment. The Prediction Network calculates the *Q* values for the given states and suggests undertaking action in the environment a^t. After the action is taken the reward *r* is calculated, the network moves to the next state $s_{t+1,}$ and the information about the state and the transition is stored in the replay memory. The role of the Target Network is to stabilise the learning approach and improve the long-term memory of the agent (the network updates are done every N-steps of the algorithm with the decay effect applied). The information stored in the Replay Memory is used during training of the Prediction Network. The training update rule follows the Bellman equation:

$$Q^{\pi}(s,a) = r + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1})),$$

where γ is a decay coefficient and π is a policy function.

The role of the DQN loss calculation module is to compute the temporal difference error:

$$\delta = Q(s,a) - \ r + \gamma \ maxQ(s_{t+1}, a)$$

To minimise the error, several functions can be used, however, the Hueber loss is commonly used. To derive the policy regarding slice admission actions, the DQN agent takes decisions based on an observed network state s_t, which consists of:

- Aggregated Requested Resources

$$ARR = \sum_{k=1}^{K} \frac{b_r^k(t)}{R}$$

- Aggregated Resources Consumption

$$ARC = \sum_{k=1}^{K} \frac{b_c^k(t)}{R}$$

- Predicted Aggregated Resources Consumption

$$PARC = \sum_{k=1}^{K} \frac{b_p^k(t)}{R}$$

- Ratio of currently accepted mMTC slices

$$R_{mMTC} = \sum_{k=1}^{K} \frac{1}{|K|} \quad iff \quad q^k = 1$$

- Ratio of currently accepted eMBB slices

$$R_{eMBB} = \sum_{k=1}^{K} \frac{1}{|K|} \quad iff \quad q^k = 0$$

- Relative time of day

$$\mathrm{ToD} = (\mathrm{t} \bmod \mathrm{T})/\mathrm{T}$$

Only two actions *$a \in \{0, 1\}$* are permitted for the agent to take, SAR acceptance *($a = 1$)* and SAR rejection *($a = 0$)*. To evaluate agent's actions, two different reward functions have been used: Equation (1) if no resource prediction is taken into account *($b_p(t) = 0$)* and Equation (2) when resource prediction is used. The used resource prediction is based on the ToD in the case of eMBB, while in the case of mMTC, where ToD effect is not present, the prediction is a constant value (i.e., the maximum bandwidth requested in SAR). The number of bandwidth samples considered in period $\langle t_d, t_t \rangle$ is denoted as $|T|$.

$$(t_d, t_t, a) = \begin{cases} 0 & iff \quad a = 0 \\ \sum_{t=t_d}^{t_t} \sum_{k=1}^{\mathcal{K}} \frac{b_r^k(t)}{\mathcal{R}|T|} & iff \quad a = 1 \; and \; \wedge_{t \in [t_d;\, t_t]} \sum_{k=1}^{\mathcal{K}} \frac{b_r^k(t)}{\mathcal{R}} \leq 1 \\ -\sum_{t=t_d}^{t_t} \sum_{k=1}^{\mathcal{K}} \frac{b_r^k(t)}{\mathcal{R}|T|} & otherwise \end{cases} \tag{1}$$

$$r(t_d, t_t, a) = \begin{cases} 0 & iff \quad a = 0 \\ \sum_{t=t_d}^{t_t} \sum_{k=1}^{\mathcal{K}} \frac{b_p^k(t)}{\mathcal{R}|T|} & iff \quad a = 1 \; and \; \wedge_{t \in [t_d;\, t_t]} \sum_{k=1}^{\mathcal{K}} \frac{b_p^k(t)}{\mathcal{R}} \leq 1 \\ -\sum_{t=t_d}^{t_t} \sum_{k=1}^{\mathcal{K}} \frac{b_p^k(t)}{\mathcal{R}|T|} & otherwise \end{cases} \tag{2}$$

It is expected that the agent should perform actions that lead the system to the desired final state, i.e., the state in which the requested (or predicted) network resource consumption in the time frame of concern is equal to *R* during the whole period (i.e., the resources are used completely). The various parameters used by TASAC are given in Table 3.

Table 3. Parameters for TASAC.

Parameter	Description
K	Slice Requests
b_r	maximum requested slice bandwidth [bu]
$b_c(t)$	consumed slice bandwidth in moment *t* [bu]
$b_p(t)$	predicted consumed bandwidth by the slice in moment *t* [bu]
t_d	slice deployment time [tu]
t_t	slice termination time [tu]
q	slice type (SST)
T	length of the day [tu]
R	resource pool
a	action
s	State

4.3.3 TASAC SIMLUATION RESULTS

The presented concept (cf. Figure 16) has been evaluated using an event-driven simulator implemented in Python using SimPy [19] package. The adopted time unit t_u is 1 s. We follow the most popular approach and model the influx of admission requests as the Poisson process [15] [9] [14]with $\lambda = 720\ t_u$ (which corresponds to 5 slice requests per hour). During the tests, we validate the algorithms under two different resource constraints: $R_s = 2000$ ru and $R_l = 1000\ ru$. The requested slice types and demanded capacity follow a discrete uniform distribution, *U{0, 1}* and *U{1, 10},* respectively. It is assumed that a single eMBB slice consumes ten times more resources than a single mMTC slice, meaning that the maximum requested bandwidth is in the range $\mathrm{b}_{krmaxmMTC} \in \langle 1; 10 \rangle$ for mMTC and $b_{k\ rmaxmMTC} \in \langle 10; 100 \rangle$ for eMBB. In the evaluation, we consider both short-lived and long-lived slices – the slice duration follows the uniform distribution *U{60, 86.4 ∗ 103}* (1 min to 1 day). The normalized resource consumption of the eMBB slice is modelled by the curve presented in Figure 18a, while for predictions, the curve shown in Figure 18b is used. Both curves have been derived from the real-life data acquired from the operator's monitoring system.

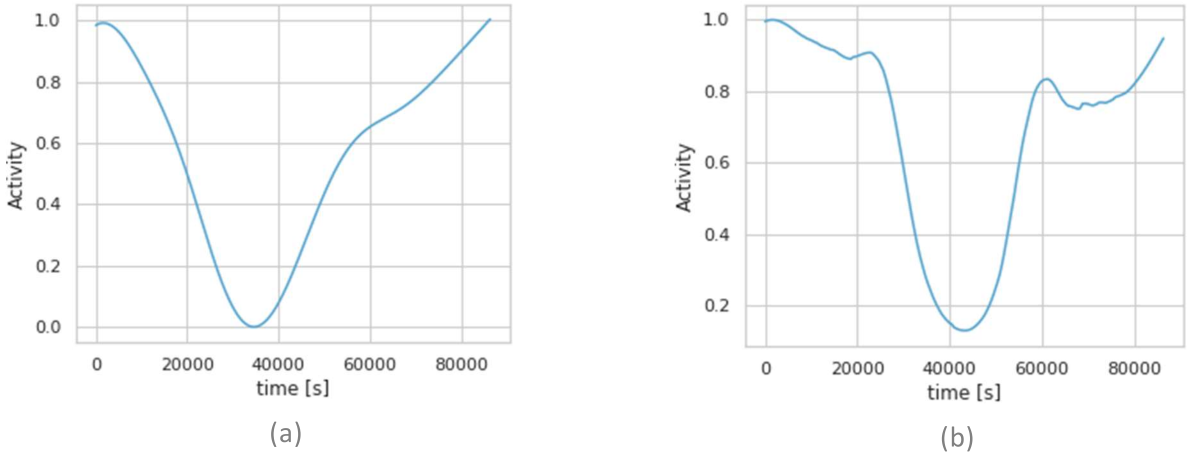


Figure 18. Real (a) and predicted ToD curve (b) of the eMBB traffic

The gains obtained from using the TASAC approach were analysed for two weeks observation time. The exemplary traces obtained for DQN and TASAC approaches are presented in Figure 19 and Figure 20. It can be observed that under given conditions, the periods in which resource violations occur are shorter for the TASAC approach.

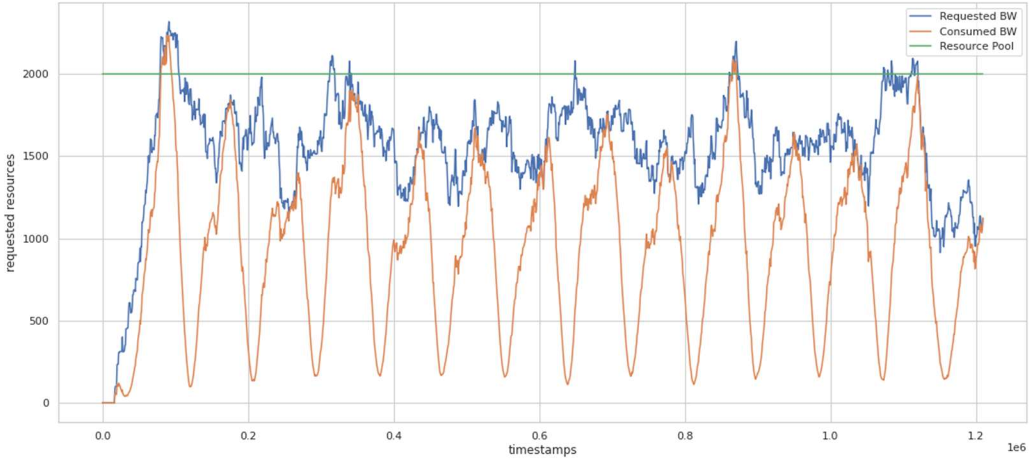


Figure 19. DQN-based slice admission (not aware of ToD)

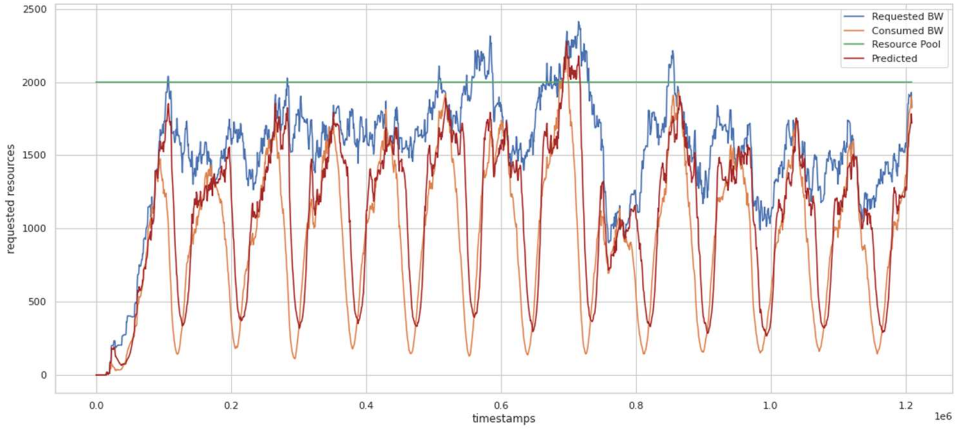


Figure 20. TASAC-based slice admission (ToD aware)

The benefits of the TASAC method can be observed in several admissions in Figure 21 and resource utilisation in Figure 22. Under severe resource constraints, i.e., R_l, the TASAC approach enables a much higher admission rate and effective utilisation of resources (Figure 21 and Figure 22). For a larger resource pool R_s both the admission rate as well as the cumulative resources consumption is similar. As the resource margin is relatively high and the rejection rate is very low for both cases, similar performance can be noted for both DQN-based and TASAC approaches.

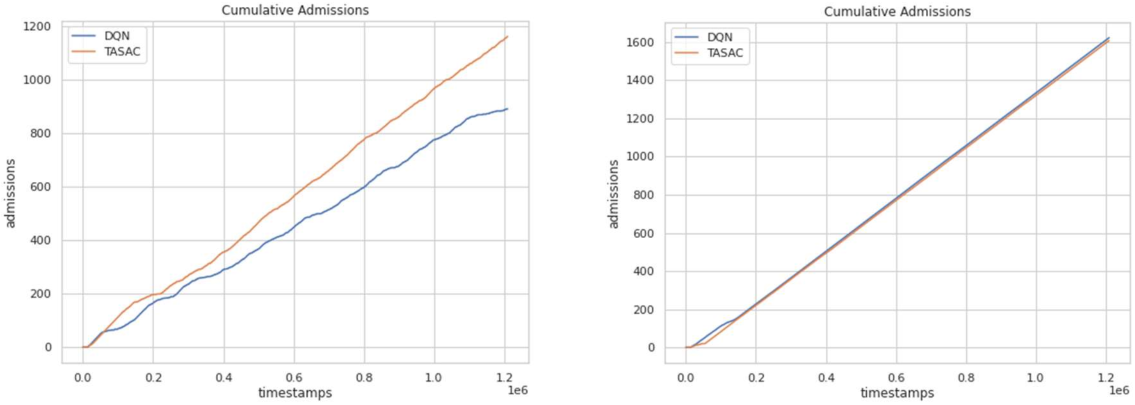


Figure 21. Comparison of a number of slice admissions during the period of two weeks: low resource pool R_l (left), large resource pool R_s (right)

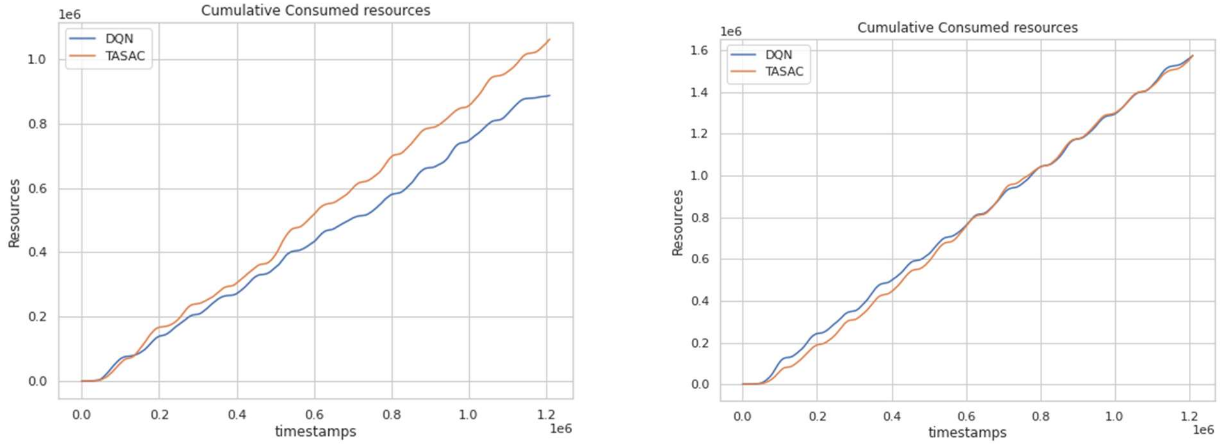


Figure 22. Comparison of resources utilisation during the period of two weeks: low resource pool R_l (left), large resource pool R_s (right)

The slice admission policies learned by the agents can lead to wrong decisions that can cause over-utilisation of the provided maximum resource pool. The penalties (measured as a cumulative exceeded bandwidth) for both evaluated cases are presented in Figure 23.

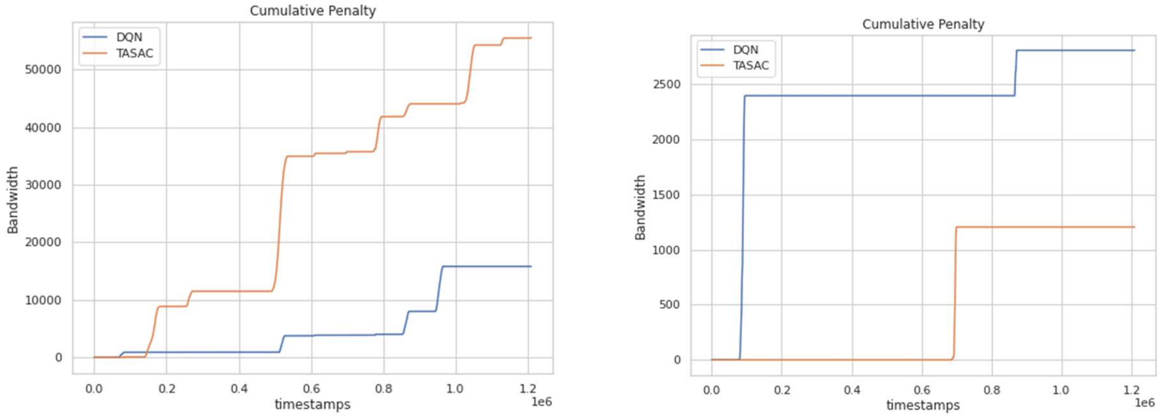


Figure 23. Comparison of exceeded bandwidth during the period of two weeks: low resource pool R_l (left), large resource pool R_s (right)

Conclusion

The TASAC approach offers high advantages in the case of R_S. Despite the similar performance in terms of admission rate and resources, the penalty is significantly lower than the DQN-based approach. In the case of low available resources (Figure 19), it must be noted, however, that the method used for the prediction of resources consumption is quite inaccurate (cf. Figure 18b). The use of more accurate predictors can contribute significantly to aggregate resource utilization.

The obtained results are satisfactory since the goal of increasing the admission rate and resource utilisation using resource prediction has been achieved. Nevertheless, it should be noted that the method used for the prediction of resources consumption is quite inaccurate (cf. Figure 20). The selection of more accurate predictors can contribute to better resource utilisation. Moreover, the analysed cases were performed for the balanced number of mMTC and eMBB slices. It must be emphasised that the number of requested slice types can vary. The presented TASAC approach aims to maximize the bandwidth utilisation, i.e., the slices with a high amplitude of activity are not preferred. Therefore, in some cases it might be essential to define additional policies that consider the resource aspect and the individual priority of the slice to be deployed.

5 Intra-slice Orchestration

5.1 Introduction

The goal of this section is to present an extensive overview of the intra-slice orchestration Decision Engine (DE) elements. This section introduces SCHEMA, a Service Chain Elastic Management framework, based on a Distributed Reinforcement Learning algorithm, its extension SafeSCHEMA and the way they integrate into the MonB5G proposed architecture. First, the problem statement will be discussed, then the proposed solution and its system model will be presented. Second, the experimental setup and results will be presented in detail.

In continuation of the previous section, the section continues with a description of the various decentralized DE modules used to tackle the intra-slice orchestration problem. Finally, the section demonstrates through experimental results analysis how the proposed architecture can be used efficiently for decentralized decision making in the MonB5G architecture.

The contribution of this section is threefold:

1. Introducing a modular and flexible architecture that can be deployed in multi-domain networks with multiple slices and services.
2. Providing a solution to minimize the unsafe actions of the RL-agents, while still allowing enough autonomy to explore the space.
3. Demonstrating a collaborative agent KPI optimization. Specifically, latency minimization for Ultra-Reliable Low Latency Communication (URLLC) services will be used as our use-case scenario.

5.2 SCHEMA: Service Chain Elastic Management with Distributed Reinforcement Learning

5.2.1 *PROBLEM DESCRIPTION*

The network consists of multiple SDN-NFV-enabled technological domains that accommodate slice VNFs. The domains are connected through a WAN that we model as a link, the users connect to the edge domains and request access to the network services that are operated by slices. Each slice is a Service Function Chain (SFC) with flows between the VNFs. Some VNFs are shared among the slices and may need vertical scaling depending on the number of requests. Due to the limited resources of the domain hardware and the number of hops between the servers, the mapping of the slice VNFs affects the average slice latency. If there are available resources on the network, the network accepts the incoming slice requests.

5.2.2 *SOLUTION TOOL*

Overview

The proposed solution responds dynamically to the traffic load fluctuation by initiating re-configurations in predetermined time-steps only when required to avoid additional costs or load to the network. Unlike similar frameworks where there is a centralized entity responsible for the slice VNF orchestration even for multiple domains, we employ a distributed system of agents that can operate *per-se*. Instead of utilizing a global network algorithm responsible for the orchestration of the slice VNFs, we employ a local VNF placement and orchestration algorithm for each domain to share the enormous problem state space of the VNF management and orchestration problem. The domains are performing VNF orchestration and VNF clustering internally, without affecting the rest of the slice in the other domains. When there is a need for VNF migration between the domains, an Auction takes place in which each local agent bids with its Confidence metric to receive and place locally a VNF from another local domain. The agents are bidding in the VNF auction through a confidence metric and only the highest bidder receives the VNF to place and orchestrate it locally.

Deep Reinforcement Learning-based solution

We formalize the intra-domain VNF placement and orchestration problem as a Markov Decision Process (MDP) by representing the problem environment through *Actions*, *States* and *Rewards*. In this specific problem we define them as follows:

- As **State** we define the intra-domain computational resources of the servers that are able to host the slice VNFs (the available CPU cores, GBs of RAM and storage), the available bandwidths and latencies of the intra-domain links.
- The **Action** of the local domain agents is the *Confidence metric* which is a float number indicating the willingness to receive the VNF of the slice that is auctioned through the *Auction Mechanism*.
- As **Reward** we define the optimization function of the RL algorithm. The Reward is common between all local domain agents to enable cooperation as stated in [22]. The goal of SCHEMA is the dynamic orchestration of the slice with a distributed algorithm while keeping the service latency low and it is defined as follows:

$$Reward = \sum_{i=0}^{n} \frac{T_i}{w_i \cdot L_i},$$

where n is the total number of VNFs, T_i is the minimum service throughput and L_i is the maximum service latency and w_i is the weight of L_i in the reward.

Local Deep Q-Network RL agents

For the domain RL agents, we utilize a DQN agent as defined in [23]. The agents interact with the network through the *Observations*, *Actions* and *Rewards* of the network. The goal of the agents is to select placements or *Actions* that maximize the *Reward* and thus, minimizing the service latency. We use a Deep Neural Network (DNN) to approximate the optimal action-value function, also known as Q-value function defined as:

$$Q^*(s,a) = \max_{\pi} \quad E[r_t + \gamma_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = a, \pi]$$

The Q-value function can be defined as the maximum sum of all rewards r_t, discounted by the parameter γ at each time-step t . The maximum sum of all rewards r_t is achieved by a behavioral policy $\pi = P(a|s)$, after an *Observation* s and taking an *Action* a .

To avoid instability during the training of the agents, we employ the *Experience Replay* technique, which randomizes the *Observations* and removes the correlation between them during the early training phase to force the agent to embrace exploration. We store the experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ of the agent at each time-step in the dataset $D_t = e_t, \dots, e_t$, that we later use to retrieve them. We apply Q-learning value updates on mini-batches of experience (s, a, r, s'), $U(D)$, drawn uniformly at random from the dataset D_t of stored experiences to perform learning of the agent. The Q-learning update during iteration utilizes the following loss function:

$$L_i(\theta_i) = \epsilon_{(s,a,r,s')}\left(r + \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i^-)\right)^2,$$

where γ denotes the discount factor that determines the agent's horizon, θ the parameters of the Q-network during iteration i and θ_i^- are the network parameters used to compute the target value at iteration i .

Every local domain is served by a VM instance of the RL agent and is responsible for the internal re-configuration, by selecting the host that the VNF will be migrated to. In addition to the placement *Action*, a *Confidence metric* can be extracted as a percentage of how confident the agent is for the chosen placement decision. The *Confidence metric* is obtained before applying the *arguments of the maxima*, also known as *argmax* function, in the output vector of the DNN, which is described with the following equation:

$$\text{argmax}_{x \in D} f(x) = \{x | f(x) \geq f(y)\, \forall y \in D|\},$$

where $f(x)$ is the set of inputs x from the DNN output D that achieves the highest function value. The *Confidence metric* is extracted from the set D as $\max(D)$, whereas $f(x)$ denotes the placement.

Inter-domain communication: The Auction Mechanism

We propose a multi-domain, distributed and non-cooperative scheme to enable scalability in the SFC placement problem while keeping the benefits of using RL. See Figure 24.

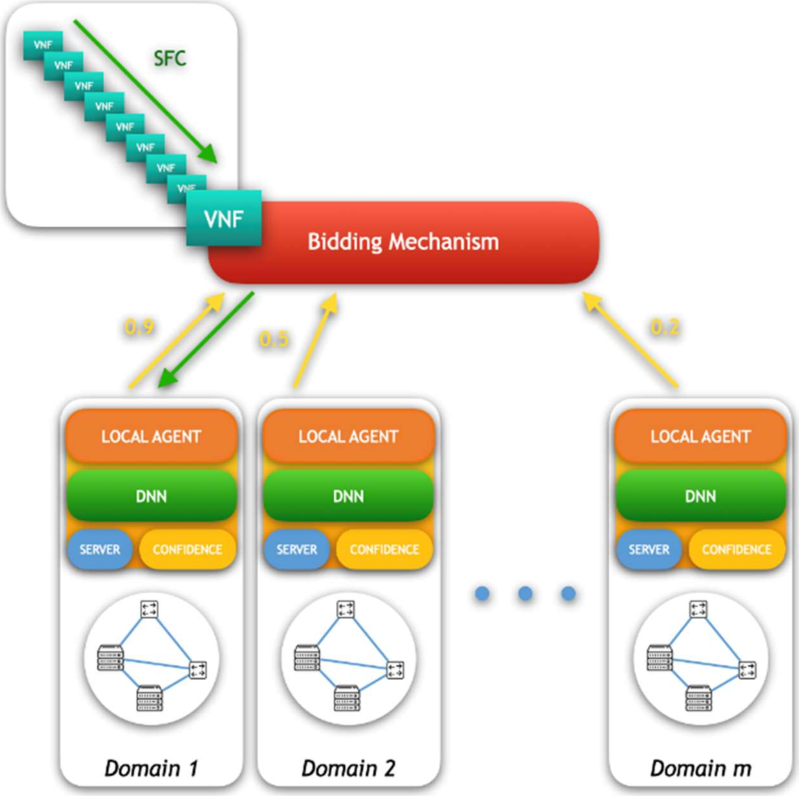


Figure 24. The Auction Mechanism architecture

We introduce the *Auction Mechanism*, an entity that performs an *Auction* of all slice VNFs at every time-step in a serialized manner. Each local domain agent places a bid with its placement *Confidence metric* to receive a VNF of the given slice and only the highest bidder can receive it to perform an internal placement, as the output of the agent denotes. Domains have only knowledge of their own resources making them autonomous to the intra- domain placement procedure. Local domain agents do not communicate with each other to determine the optimal solution, but rather use the global *Reward* to introduce cooperation between the agents, as proposed in [22].

5.2.3 FIT INTO THE DE DISTRIBUTED ARCHITECTURE

The proposed framework SCHEMA utilizes a set of distributed agents instantiated in every domain, making it compliant with the distributed architecture of MonB5G. SCHEMA agents perform local placement in each domain and share the enormous problem state space of the placement problem. The domains perform VNF orchestration and clustering internally in a closed loop, without affecting the rest of the slice. When VNF migration between the domains is necessary, an Auction takes place in which each local agent bids with its Confidence metric to receive and place locally a VNF from another local domain. The agents are bidding in the VNF auction through a confidence metric and only the highest bidder receives the VNF to place and orchestrates it locally.

5.2.4 VALIDATION RESULTS

We have performed multiple experimental scenarios with groups of $U = \{100,500,1000,1500,2000\}$ users uniformly scaled per slice, and $D = \{3,5,7,9,11\}$ domains. The inter-domain links were assigned a random latency following a normal distribution $l_S^{lat} \in [2,3]$ ms. The intra-domain network is composed of servers with 32 Cores, RAM 128 GB, 1TB storage and the intra-domain links are assigned a random latency value between $l_D^{lat} \in [1,2]$ ms. Users are connected through a 5G mmWave wireless link to a base station connected to a server of the inter-domain network with a 2% to 10% loss. The slice VNFs have 1 to 2 CPU cores, 2 to 4 GB RAM and 1 to 80 GB storage as computational resource requirement for placement. In our experiments, we examine the performance of the proposed algorithm SCHEMA by comparing both the average user service latency and the average number of rejections.

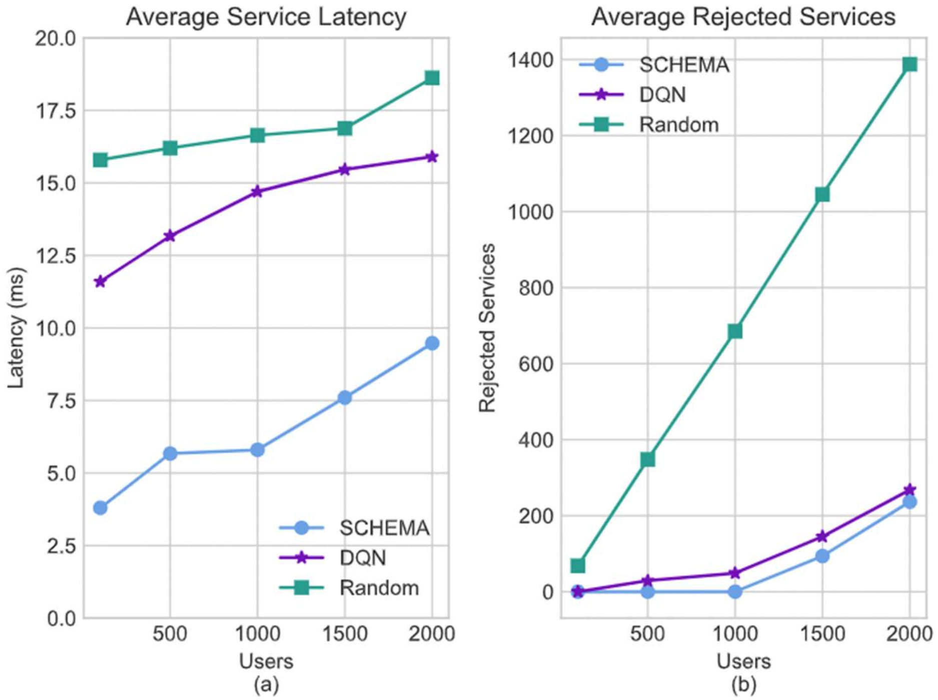


Figure 25. Performance of SCHEMA (service latency and rejection ratio).

(a) Average service latency of accepted services for 3 domains. (b) Rejected services by the number of users for 3 domains.

As we can observe in Figure 25, increasing the number of users per slice in the network also increases the average service latency due to insufficient computing resources in servers within the local domains. Specifically, in the case of 3 domains, SCHEMA was able to outperform both baseline solutions by offering considerably lower service latency by almost 60.54% in the case of 1000 users.

Accordingly, SCHEMA rejected less users compared to the DQN by 54.25% in the case of 1500 users, demonstrating better SFC placements for the same infrastructure. As the number of users increases, we can see reaching the limit of insufficient resources, near the 1600 users.

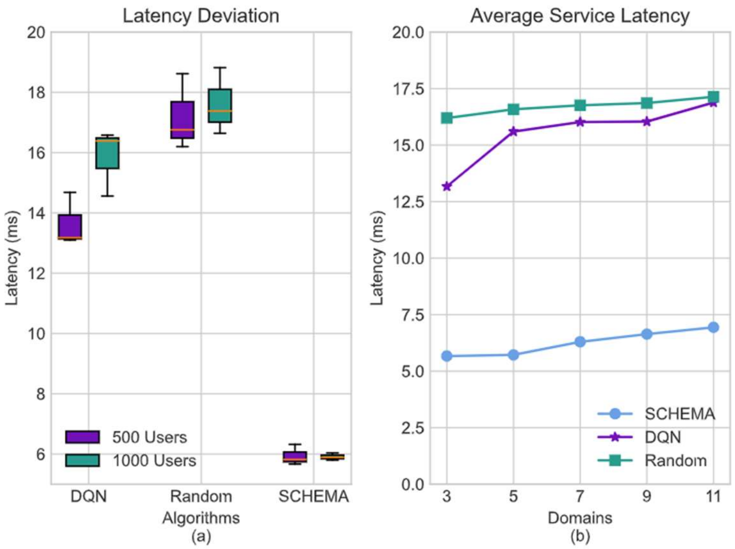


Figure 26. SCHEMA compared with other algorithms (latency).

(a) Latency variation in 500 and 1000 user scenarios for 3 domains. (b) Average service latency of 500 users for 3 domains.

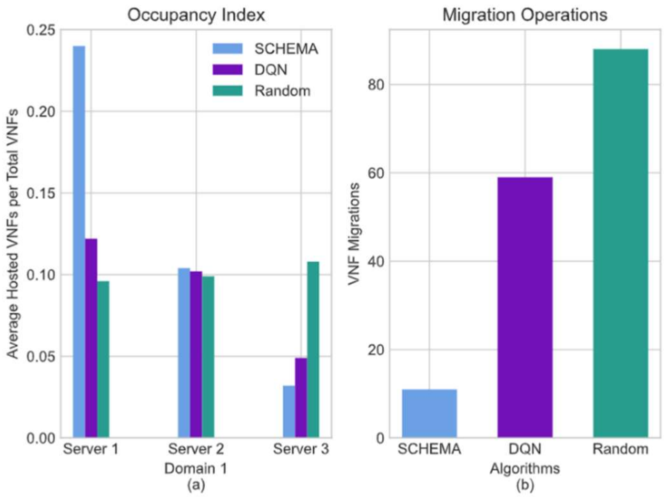


Figure 27. SCHEMA performance (occupancy and migration).

(a) VNF Occupancy index or the average hosted VNFs per total VNFs of 100 simulation iterations for 500 users. (b) Migration operations of simulation iterations and 50 SFCs with 125 VNFs.

Moreover, Figure 26a) outlines the service latency variation for the cases of 500 and 1000 users. After training, SCHEMA was able to conceive a better placement than the DQN, leading to consistently lower service latency by almost 63.33% in the case of 5 domains. It is evident that the DQN is heavily affected by the number of users comparing the height difference in the boxes. With the help of Figure 26a), we can conclude that SCHEMA gravitated towards consolidating the slice VNFs in the same server to further reduce the number of hops to the end user. In Figure 27b) we have further evidence that due to the introduction of the Auction Mechanism, the local agent of the depicted Domain in Figure 27a) was keeping the same placement to avoid inter-domain slice VNF re-configurations.

5.2.5 *MONB5G RELATED CHECKBOXES/KPIS*

✓Intra-slice re-configuration and resource allocation.

✓Reconfigure VNF location and chaining dynamically.

✓Utilize key inputs (capacity, load of links, congestion level of local and alternative computing resources, history thereof, and KPI predictors from the distributed AEs.

✓Propose reinforcement learning algorithms running locally.

✓Receive KPI predictions from the AE (periodically or event-based) and attempt to identify feasible local reconfigurations (i.e., affecting only a part of the chain) that could reconcile the slice SLA or apply a specific intent policy, without the need for global reconfiguration or migrating the entire chain.

✓Quantify the confidence in the ability of the locally optimized/reconfigured slice to resolve the potential problem.

5.3 SafeSCHEMA: Multi-domain Orchestration of Slices based on SafeRL for B5G Networks

5.3.1 *PROBLEM DESCRIPTION*

Safe interaction of the AI agents with the network is one of the predominant challenges, especially when Reinforcement Learning (RL) is used in critical environments. Slice management is one of the major features that operators want to automate, to offer future services at large scales with manageable complexity to the operator. However, during the exploration phase, RL agents can cause significant performance degradation during operation and possibly introduce irreversible damage to the service being offered. To address this major challenge, we propose a multi-agent, modular, SafeRL architecture for distributed slice orchestration, called Safe RL-based Service Chain Elastic MAnagement or SafeSCHEMA for short.

5.3.2 *SOLUTION TOOL*

Overview

We propose an extension of the previously proposed algorithm SCHEMA. A modular architecture for safe and automated slice management in multi-domain networks. Similar to SCHEMA, the proposed framework consists of multiple distributed agents that co-operate to orchestrate the slice elements. The RL-enabled agents are wrapped with a Safety Shield, which prevents the execution of unsafe placement actions that can be proven dangerous for the operation of the End-to-End (E2E) slice performance.

Intra-Domain Distributed Orchestration with Safe Distributed Reinforcement Learning

In this work, we deploy Safe RL agents per each domain to orchestrate the slice VNFs inside the borders of a domain n. Additionally, a module called the *Auction Mechanism* that enables inter-domain migration is developed. During the operation, the *Auction Mechanism* circulates through the slice VNFs and the domain

Safe RL agents bid to win the auction and receive the VNF in auction. The goal is to use multiple distributed Safe RL agents, one per domain to orchestrate slice VNFs with respect to a specific KPI, e.g., E2E average slice latency in this work, in a scalable way.

The domains are responsible to orchestrate the hosted VNFs locally and asynchronously, to avoid unnecessary migrations. The local agents consist of multiple modules that cooperate to perform the task of safe inter-domain, intra-slice VNF placement. Our approach is a SafeRL-enabled agent based on the work of Aumayr et al. [24] that uses a modular shield architecture and a safe baseline. The action proposed by the RL agent is benchmarked against a safe baseline through a *Safety Shield* that chooses the safest action to be performed on the environment.

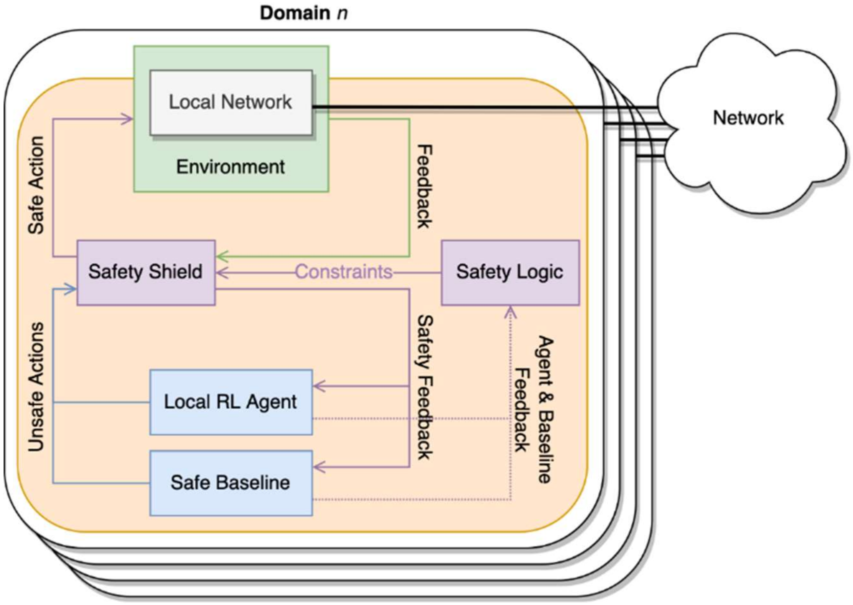


Figure 28. SafeRL-based local domain architecture.

Briefly, the Local Agent architecture and components, as shown in Figure 28, are:

1. *Environment:* This component represents the environment that the agent is interacting with, modeled as a standard RL problem. It provides interaction with the local domain n part of the network infrastructure which was described in the previous section.
2. *RL Agent:* The *RL agent* is continuously trained during the interaction with the *Environment* to improve their future recommended actions for VNF placement. The agent indirectly interacts with the *Environment* by proposing actions to the *Safety Shield* and getting feedback on the actual action performed.
3. *Safe Baseline:* The *Safe Baseline* provides a default action, usually sub-optimal in terms of performance, yet safe with regards to what safety indicates for the system at hand. In our scenario, we define safety as an action that does not lead to breaking any service SLAs. *Safe Baseline* is a rule-based algorithm that recommends actions satisfying safety rules, usually designed by domain experts. Baselines also receive the current state of the *Environment* as feedback, to suggest future actions.

4. *Safety Shield:* This component is the middleman between the *RL agent* and the *Environment*. In the non-Safe RL architecture, the RL agent directly interacts with the environment and receives feedback. On the contrary, in the SafeRL architecture the *Safety Shield* is a broker between the *RL Agent* and the *Environment* to protect the environment from unsafe actions. The *Safety Shield* collects the actions proposed by the *RL agent* and the *Baseline* and selects a final safe action to be performed in the *Environment*. The environment feedback includes the action that is performed by the shield on the environment and is fed back to *RL agent* and the *Baseline*, so that they are aware of the new state of the *Environment* before the next iteration.
1. *Safety Logic:* The logic provides constraints to the *Safety Shield*. It includes the logic used to determine what is the safest action between the one proposed by *RL agent* and the *Safe Baseline* at any time. Due to the modularity of this architecture, it supports various implementations.

Local Deep Q-Network RL agents

We model the intra-domain VNF placement and orchestration problem as a Markov Decision Process (MDP) which consists of a *State*, an *Action Space* and a *Reward Function*. The MDP problem space can be defined as follows:

1. *State Space:* Consists of the intra-domain computational resources of the domain nodes u_n. The local domain state space S_n is defined as a set that contains the VNF hosting computing requirements, the slice SLA and the domain computing and networking utilization ratio for each network element.
2. *Action Space:* The *Confidence Vector* A_n, a vector that contains an offer ∈ $[-1,1]$ for each domain server u_n to receive the VNF currently to be placed. The maximum of this local action indicates an intra-domain VNF migration.
3. *Reward:* The objective of all domains n is to maximize the sum of the reward function in every iteration and thus, converging in a common state-action that enables cooperation among the domains:

$$R^{sha} \quad = \frac{1}{w_D \cdot D} + P,$$

where w_D is the weight of the total slice delay D .

In addition, we define a penalty function that is applied when the SLA of the slice s is violated:

$$P = \begin{cases} -w_p & if\ \varphi_s^{delay} \\ 0 & otherwise, \end{cases}$$

where w_P expresses the penalty weight.

Inter-Domain VNF Orchestration & Auction Mechanism

The *Auction Mechanism* (see Figure 29) is a system that enables inter-domain VNF migration. It acts as an auctioneer between the domains who bid with their actions to receive a slice VNF to their domain.

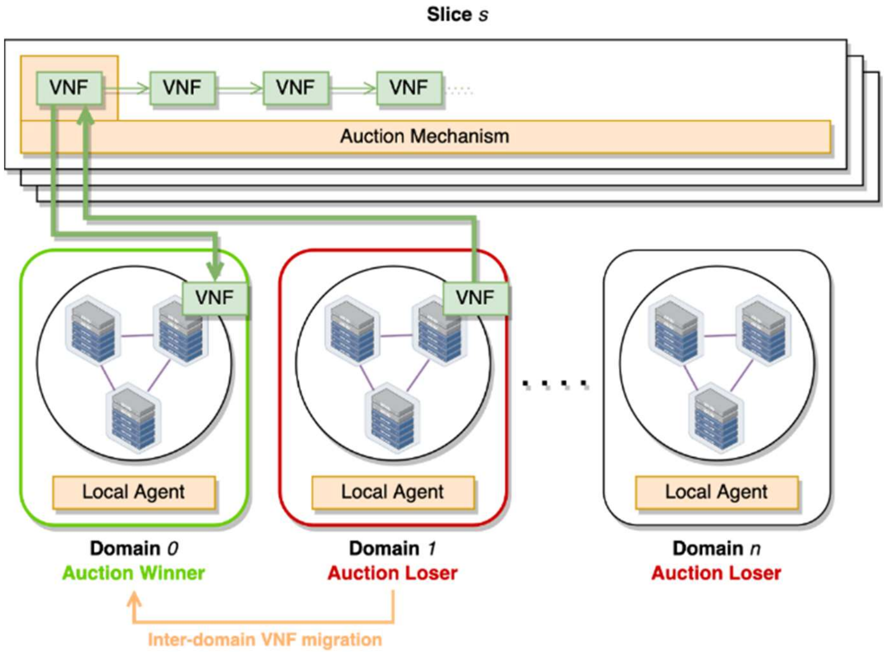


Figure 29. Overview of the multi-domain and distributed Auction Mechanism.

All local domains can orchestrate VNFs in parallel and exchange VNFs only during an auction.

The *Auction Mechanism* repeats the following steps in every iteration:

1. *Selection:* The *Auction Mechanism* circulates through the slice VNFs during an interval, a predefined time frame. As an auctioneer, it selects the next VNF in round robin fashion for the auction process.
2. *Participation:* The per domain n Local Agents act as bidders that generate their bid, an action A_n produced by the *Local RL Agent* and secured by the *Safety Shield* as previously described.
3. *Auction:* The mechanism receives all domain bids and selects the highest bidder (the domain with the highest bid). The candidate domain is notified to initiate the migration.
4. *Orchestration:* If the candidate domain is not the current domain, the inter-domain migration is initiated. Otherwise, the domain agent performs an intra-domain migration based on the safeRL action to the node with the highest bid.

The *Auction Mechanism* is only responsible for the inter-domain communication. It is non-essential for the local domain orchestration. The *Auction Mechanism* can be deployed quickly as a stateless container at any node of the network, eliminating the single point of failure.

5.3.3 FIT INTO THE DE DISTRIBUTED ARCHITECTURE

In this work, we deploy Safe RL agents per each domain to orchestrate the slice VNFs inside the borders of a domain. Additionally, a module called the Auction Mechanism that enables inter-domain migration is developed. During the operation, the Auction Mechanism circulates through the slice VNFs and the domain Safe RL agents bid to win the auction and receive the VNF in auction. The goal is to use multiple distributed Safe RL agents one per domain to orchestrate slice VNFs with respect to a specific KPI, e.g., E2E average slice latency in this work, in a scalable way.

5.3.4 VALIDATION RESULTS

Configuration

As *Safe Baseline* we use an algorithm called *Compu* that migrates the VNF to the local server with the most resources available as a default action. This action is deemed safe, as the inter-domain servers are in proximity and introduce less latency than servers located in different domains, leaving computational resources as the major source of additional delay.

Overloaded servers introduce software delays; thus, resource utilization should be a decisive factor and be prioritized compared to the server distance when it comes to designing a baseline action which is applied in case of an unsafe action proposed by the *RL Agent*. We use it as a demonstration of custom rules.

Baselines

The performance of SafeSCHEMA is compared with three main baselines:

1. **SCHEMA** is a non-safeRL version of SafeSCHEMA. Comparing it with a flat distributed solution.
2. **DQN** is a DRL-based orchestration algorithm located in a central location. It is a common type of baseline approach in related research literature.
3. **Static** is a placement method adopted by many works as the default baseline.

Performance Evaluation

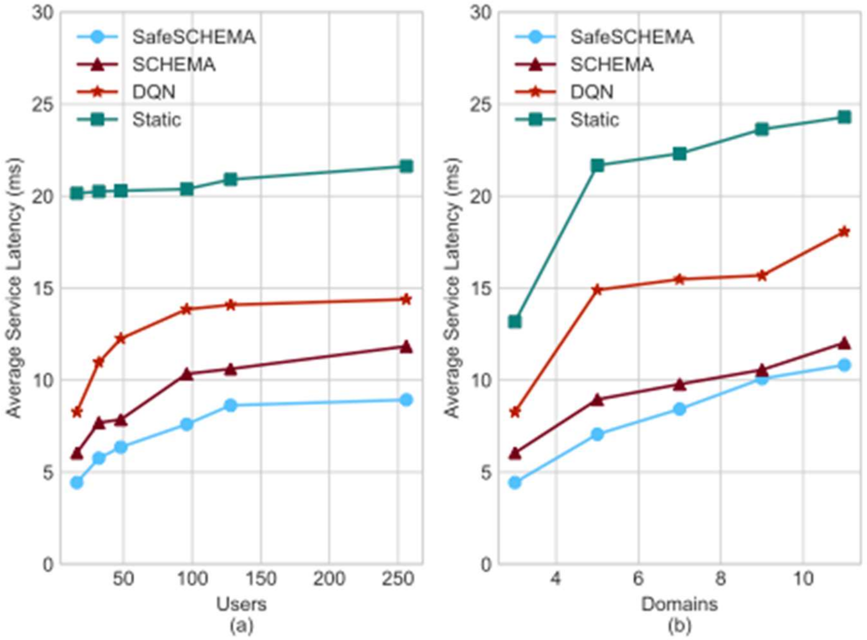


Figure 30. Performance of SafeSCHEMA (service latency).

a) Average slice latency per number user. (b) Average slice latency per number of domains. Lower is better for both figures.

In Figure 30, subfigure a) depicts the average E2E slice latency in milliseconds (ms) for a varying number of users, uniformly scaled per slice. We observe that the average slice latency increases with the number of users as they occupy more network resources and limit the possible VNF placement options that satisfy all users with low latency, no matter of their location in the network. It is apparent that the guaranteed SLA, defined as 10ms for the slice is respected by SafeSCHEMA. Additionally, it can outperform the distributed baseline of SCHEMA as it filters any unwanted proposed actions from the agents that will violate the SLA, thus lowering the average slice latency.

Subfigure b) demonstrates the scalability of SCHEMA and the superiority of SafeSCHEMA as the *Safety Shield* module and the *Baseline* protect the network from unsafe RL agent actions. Introducing more domains in the network exponentially increases the complexity of the orchestration, but as we can see in the same figure SCHEMA and SafeSCHEMA were able to resist the degradation of performance as the state space grew with the introduction of new domains. This behavior can be attributed to the distributed architecture that orchestrates the slice VNFs locally and exchanges them only when needed by both algorithms. The performance superiority of SafeSCHEMA can be attributed to the use of the *Safety Shield* that prevents any unsafe actions from being executed in the network and replaces them with non-optimal but safe actions that can keep the average slice latency low.

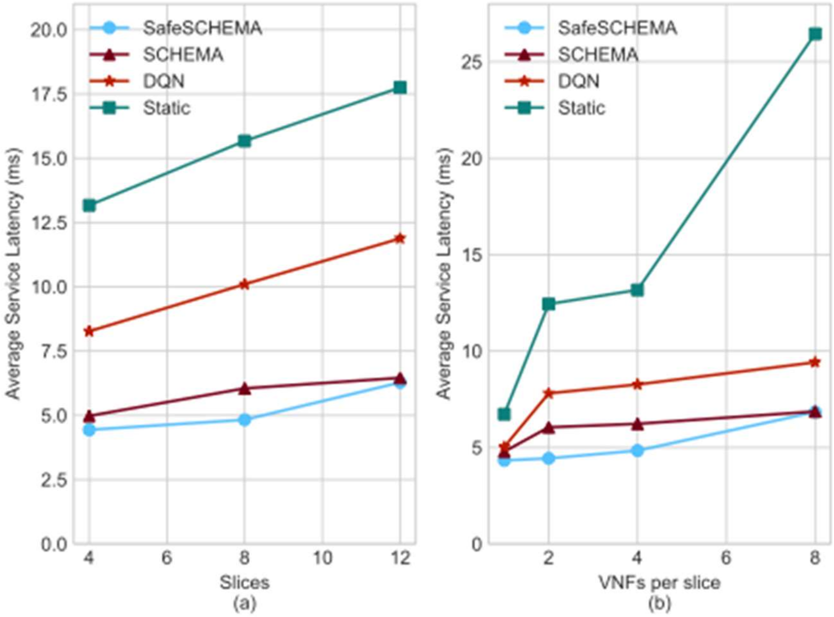


Figure 31. SafeSCHEMA (slice latency).

(a) Average slice latency per number of slices. (b) Average slice latency per number of chained slice VNFs. Lower is better for both figures.

Figure 31a) presents the performance of the compared algorithms during the operation of multiple slices, hence their ability to maintain performance, demonstrating the ability to scale horizontally. We observe again that both distributed solutions were able to outperform the rest of the baselines. SafeSCHEMA takes the edge in this scenario by having a 15.38% lower latency than SCHEMA. The reason for this is the *divide-and-conquer* approach that drastically reduces the orchestration complexity and tackles it with minimal overhead.

In Figure 31 b) the average slice latency is plotted against the number of chained VNFs on each of 4 slices. It is clear, especially in the case of 2 and 4 chained slice VNFs that SafeSCHEMA was able to maintain a lower slice latency by 20.18% from SCHEMA and 126.62% from Static in the case of 8 VNFs.

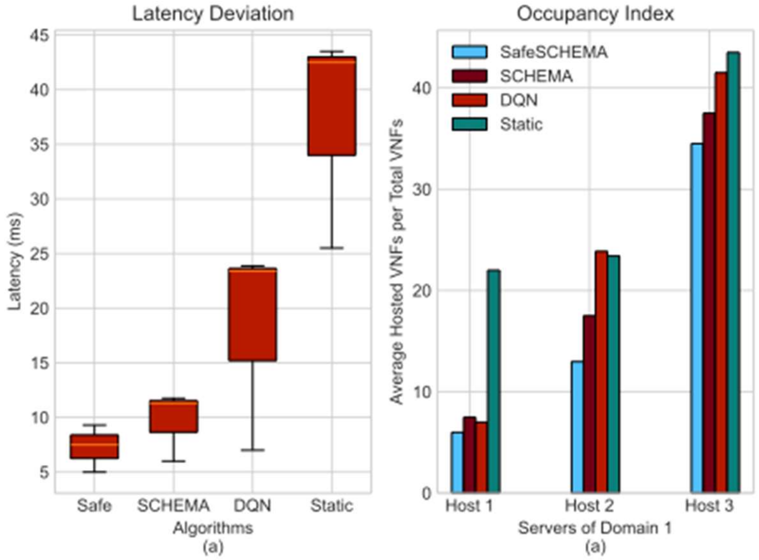


Figure 32. Comparison of SafeSCHEMA with other algorithms.

(a) Latency deviation per algorithm. (b) VNF Occupancy index represents the average number of hosted VNFs per total number of VNFs for 100 iterations of simulated traffic.

Finally, in Figure 32 a) we show the latency deviation of all competing solutions of our work for 4 slices. It is apparent that from the two distributed solutions, only SafeSCHEMA was able to avoid violating the slice SLA of 10ms as the *Safety Shield* protects the network from unsafe actions.

Figure 32 b) presents the behavior of SafeSCHEMA that gravitated towards consolidating the chained slice VNFs to reduce the number of hops to the end-user and thus, reduce the average slice latency.

5.3.5 MONB5G RELATED CHECKBOXES/KPIS

✓Intra-slice re-configuration and resource allocation.

✓Reconfigure VNF location and chaining dynamically.

✓Utilize key inputs (capacity, load of links, congestion level of local and alternative computing resources, history thereof, and KPI predictors from the distributed AEs.

✓Propose reinforcement learning algorithms running locally.

✓Receive KPI predictions from the AE (periodically or event-based) and attempt to identify feasible local reconfigurations (i.e., affecting only a part of the chain) that could reconcile the slice SLA or apply a specific intent policy, without the need for global reconfiguration or migrating the entire chain.

✓Quantify the confidence in the ability of the locally optimized/reconfigured slice to resolve the potential problem.

6 Inter-slice Orchestration

6.1 Introduction

This Chapter focuses on the problem of inter-slice orchestration, which lies a level above its intra-slice counterpart and requires a wider picture of the network and its performance. Some of the key characteristics of this problem, as discussed in D4.1, are that a DE at this level needs to take special care of reconfiguration costs, diverse SLA agreements, efficient use of network resources, and to be able to handle a large number of slices. However, this is very challenging to achieve, because multiple slices (possibly spanning across multiple technological domains) give rise to very high complexity problems that are difficult to solve even for a very small number of slices.

In the corresponding Chapter of the previous deliverable (D4.1) we presented some first algorithmic approaches, focusing mainly on domain specific inter-slice orchestration. The proposed solutions included a Reinforcement Learning algorithm for VNF placement and reconfiguration, a Multi-Armed Bandit algorithm for RAN resource allocation, and a probing scheme for VNF bottleneck localization. We also previewed how the system model could be extended to possibly incorporate more than one domain.

This deliverable builds on to either provide extensions to the solutions of D4.1 (aiming mainly to increase their scalability and to support many slices) or to apply them in more practical use cases. More specifically, each of the following subsections will be dedicated to a different solution related to inter-slice orchestration.

In Section 6.2, the VNF placement and migration algorithm, originally demonstrated for one domain in D4.1 has been extended as follows: (i) it now supports multiple VNFs per slice, namely an arbitrary, probabilistic VNF graph (allowing also for loops); (ii) various realistic end-to-end performance metrics for the average flow served by such a VNF graph can now be supported; (iii) a multi-agent solution, where a DE (agent) resides with each slice (or even with each VNF of a slice) that radically improves the scalability of the scheme as the number of slices (or VNF host locations) increase while still maintaining close to optimal performance.

In Section 6.3, the RAN resource allocation algorithm has been extended so that: (i) it enables resource allocation at the edge of the network, thus accounting for more timely and accurate information; (ii) it dramatically decreases the amount of control information that needs to cross the network to reach the central controller, thus reducing overhead towards the core network and avoiding bottlenecks; (iii) it enables the provisioning of federated learning schemes to further enrich the capabilities of the decision agents.

In Section 6.4, the probing scheme for VNF bottleneck localization is now applied in a more practical use case relying on a docker based Service Chaining Function.

6.2 Independent DQN agents for end-to-end slice reconfiguration

In this Section we propose a multi-agent DRL algorithm for dynamic and efficient end-to-end (e2e) slice embedding and reconfiguration. We first describe the problem at hand and the system model in 6.2.1. Then, in 6.2.2, we describe the solution tools proposed by first formulating the problem as a Markov Decision Process (MDP) and then presenting the candidate RL algorithms. In 6.2.3 we explain how the proposed solution fits into the MonB5G distributed DE architecture and in 6.2.4 we validate it by simulations, verifying the scalability of the proposed algorithm and its superior performance compared to the baselines. Finally, in 6.2.5 we outline which MonB5G goals are achieved by the proposed algorithm.

6.2.1 *PROBLEM DESCRIPTION*

The problem at hand is to dynamically decide the configuration (embedding) of slices on top of a multi-domain Physical Network (PN) for unknown/stochastic user generated traffic, in a way that minimizes e2e SLA violations, operating costs and reconfigurations in the long run. In what follows we will present the components of the system model; to clearly define the various components, we will often be referring throughout this Section to Figure 33, which depicts a simple example of embedding 2 slices on top of a multi-domain PN.

6.2.1.1 MAIN COMPONENTS OF THE SYSTEM MODEL

The system model comprises two main components, the Physical Network, and the Network Slices, which are commonly represented by graphs in the literature [25], [26]; a similar modeling viewpoint is used here as well.

Physical Network. It is represented by a weighted undirected graph $G = (\mathcal{V}, \mathcal{E})$, and possibly comprises multiple (technological or administrative) domains. Each (physical) node $v \in \mathcal{V}$ (V in total) has a corresponding capacity b_v to host VNFs of its domain (it can be resource blocks, CPU cores, containers, etc., depending on the domain). Some of the nodes may be just routers, so they have $b_v = 0$. Accordingly, each (physical) link (or path) $e \in \mathcal{E}$ (E in total) has some fixed capacity b_e (e.g., bandwidth). In the example of Figure 33 there are 3 domains (CRAN, MEC, CN) with (3, 2, 3) nodes respectively, where VNFs could be executed (servers), and associated capacities b_v. These servers are connected either through physical links (of capacity b_e), or paths comprising multiple links and nodes of the network[1].

Network Slices. We consider a set of slices $\mathcal{K}$ (K in total), which must be embedded on top of the PN. A slice k is represented by a directed graph $H_k = (\mathcal{N}_k, \mathcal{L}_k)$, where the nodes correspond to Virtual Network Functions (VNFs) and the edges correspond to Virtual Links (VLs). The VNFs process the flows that belong to this slice and can be viewed as tasks running on host servers, while the VLs indicate the order of how these tasks must be executed. In Figure 33, Slice 1 is an example of a simple "VNF chain", where all flows must first get processed by VNF1 and then by VNF2, to receive the offered e2e service. Our model is generic, allowing for both loops (e.g., flows passing by the same VNF multiple times), as well as probabilistic routing of flows (e.g., to capture scenario where not all flows of a slice require all VNFs in the same order). An example of such a chain is depicted in Figure 33, where for Slice 2, a percentage of flows from VNF1 proceed to VNF2 directly, while the rest must pass through VNF 3, possibly going back to VNF 1 as well.

[1] w.l.o.g., and to keep the model simple, we assume that the routing path between any two nodes is given (predetermined).

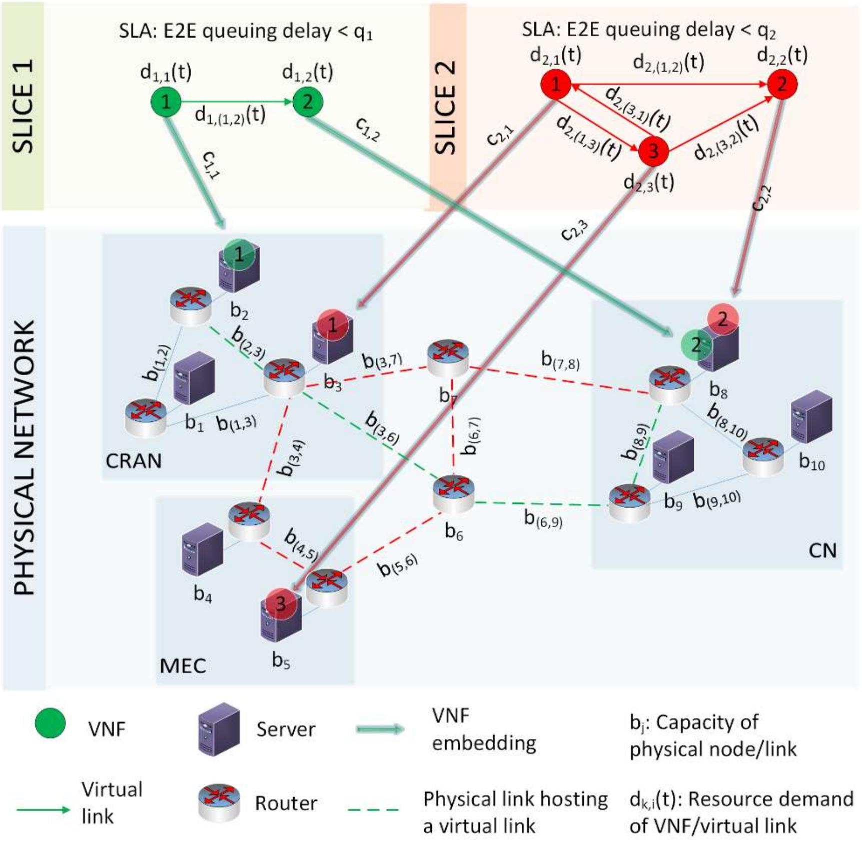


Figure 33. Graphical Illustration of the system model

VNF demands $d_{k,n}$ *:* Each VNF n of slice k is associated with a resource demand $d_{k,n}(t)$ at each time t, that will be imposed on the physical node where the VNF is executed.

Virtual link (VL) demands $d_{k,l}$: Similarly, each (virtual) link l of slice k has resource demand $d_{k,l}(t)$ [2].

Remark: These demands are often unknown, stochastic, nonstationary (correlation between VNFs of the same slice is also common); the main reason why we require a learning-based optimization algorithm to tackle this problem.

Demand vector D_t: Let vector D_t denote the VNF and VL demands of all slices at time t.

Service Level Agreements q_k : Each slice k comes with some slice-specific requirement q_k, which defines the maximum (or minimum) acceptable value for an end-to-end KPI metric (e.g., in Figure 33 each of the slices has a service level agreement for some maximum e2e queuing delay).

Control variables $x_{k,i,j}$ *:* equal to 1 when VNF/VL i of slice k is hosted by node/link j, and 0 otherwise.

[2] Note that this load will be added to all physical links along the PN path between the execution nodes hosting the two VNFs connected by virtual link l.

Configuration C_t *:* Let vector C_t denote the configuration (embedding) of all VNFs to physical nodes at time t. As an example, in Figure 33, VNF 1 of Slice 1 is hosted by Node 2, hence $c_{1,1} = 2$, and the configuration is $C_t = [c_{1,1}, c_{1,2}, c_{2,1}, c_{2,2}, c_{2,3}]$.

6.2.1.2 E2E KEY PERFORMANCE INDICATORS AND SLAS

In what follows we will first introduce the functions that determine the "local" performance of a VNF or a VL (Local key performance indicator functions), which is related to the aggregate demand of the VNFs or VLs assigned to the host physical node or link and its corresponding resources. Then, we build on that to define the slice-specific e2e key performance indicators and the associated SLAs.

Local Key Performance Indicator functions. We define a slice-specific local key performance indicator function $f_j(C_t, D_t)$, which can be applied on any physical node/link j traversed by the slice and gives a local performance metric as a function of the aggregate traffic demand on the specific node/link and its total capacity. These functions might even differ among technological domains and could possibly capture any performance indicator metric. We will give two simple examples by introducing the queuing delay and the underprovision local performance indicator functions (however more types of KPIs can be also incorporated in the framework).

- *Queuing delay*: We assume that the packets for each VNF $n \in \mathcal{N}_k$ and VL $l \in \mathcal{L}_k$ of slice k arrive according to a Poisson process with rate $d_{k,n}$ and $d_{k,l}$ respectively. Consequently, at each server or physical link $j \in \mathcal{V} \cup \mathcal{E}$ the packet arrivals are Poisson with rate equal to the sum of rates over the VNFs/VLs assigned on the server/link. Moreover, we model each server and link as an $M/G/1/PS$ queue with Processor Sharing scheduling policy and a generic distribution of service times with mean value $1/b_j$. We calculate the mean response at each server (queuing delay) by the standard closed form formula for a stable system [33] [3] :
$$f_j(C_t, D_t) = 1/(b_j - z_j(C_t, D_t))$$
$$\text{where } z_j(C_t, D_t) = \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{N}_k \cup \mathcal{L}_k} d_{k,i} \cdot x_{k,i,j}$$

- *Underprovision*: We assume that the resource demand is the capacity required on a server or a physical link to serve the user-generated traffic. When the aggregated demand of the VNFs or VLs assigned on a server or physical link exceeds its nominal capacity, the QoS perceived by the end-users of the associated slices is negatively affected [34]. The corresponding performance function can be defined by:
$$f_j(C_t, D_t) = \begin{cases} z_j(C_t, D_t) - b_j & , if \;\; z_j(C_t, D_t) > b_j \\ \quad 0 & , otherwise \end{cases}$$

E2E Slice Key Performance Indicators. An e2e key performance indicator $f_k(C_t, D_t)$ for a slice k can be calculated based on the local key performance indicator functions $f_j(C_t, D_t)$, and in the simplest case scenario it is just the sum of their output values when they are applied to all physical nodes and links traversed by the

[3] We assume that two VNFs of the same slice cannot be executed in the same node.

slice. As an example, in Figure 33, the e2e queuing delay experienced by slice 1 is the sum of the delays in node 2, link (2,3), link (3,6), link (6,9), link (8,9), and node 8. In the case of a more complex slice, a Jackson network type of analysis could be applied to calculate the delay. In the remainder we focus on simple chain slices without loops.

6.2.1.3 OPERATIONAL COSTS

Given the (usually unknown) demands D_t and the configuration C_t at time t, we assume that the system suffers an instantaneous cost related to both the network performance (i.e., direct cost to the operator) and slice performance (e.g., indirect cost related to SLA violations). We choose to consider the following three cost quantities in this work (other components can be straightforwardly added to the framework):

Type 1 cost: Node utilization. This is the cost paid by the network operator for operating expenses. It is equal to the number of servers that are active (host at least one VNF) and relates to the energy consumption of the network. Minimizing this cost also facilitates the admission of new slices by maximizing the free space of resources [29].

$$cost_1(C_t) = \textstyle\sum_{v \in \mathcal{V}} \beta_v \,,$$

$$\text{where } \beta_v = 1 \;\; if \;\; \textstyle\sum_{k \in \mathcal{K}} \sum_{n \in \mathcal{N}_k} x_{k,n,v} \geq 1$$

Type 2 cost: Reconfiguration. This is the cost for migrating VNFs from their host servers to other servers in the network. It relates to the overhead generated for reassigning all VNFs and the delays incurred by this action [30].

$$cost_2(C_t, C_{t+1}) = 1/2 \cdot \sum_{k \in \mathcal{K}} \sum_{n \in \mathcal{N}_k} \sum_{v \in \mathcal{V}} |x_{k,n,v}(t+1) - x_{k,n,v}(t)|$$

Type 3 cost: SLA violation. When the maximum value q_k defined by the SLA is exceeded, a penalty is paid by the network operator to the slice tenant. This penalty can take any form that is appropriate to model the violation of the corresponding e2e KPI, e.g., linear or quadratic. We give as an example the linear form:

$$cost_3(C_t, D_t) = \sum_{k \in \mathcal{K}} (f_k(C_t, D_t) - q_k)$$

6.2.2 *SOLUTION TOOL*

Our goal in the dynamic slice embedding problem is to decide the configuration C_t at every time t, (i) towards optimizing the total system cost (consisting of the various cost components) over a discounted infinite time horizon, while (ii) not knowing a priori how demands D_t evolve over time. This is an online learning and control problem, for which Reinforcement Learning (RL) schemes are a natural candidate. Towards this direction, we will first formulate the problem as an infinite horizon discounted MDP, assuming that traffic demand is discretized and transitions between different levels follow a known Markov Chain. Then we will present the exact algorithms that can solve this problem optimally (Policy Iteration, Q-learning), and explain why their application in large scale scenarios is precluded. Finally, we propose two DRL algorithms (DQN and IDQN) that use approximation in state and action spaces to reduce problem complexity/convergence time and advance scalability.

6.2.2.1 MDP PROBLEM FORMULATION

To formulate the problem as an MDP we must define the state and action spaces, the transition probabilities between states (the system's dynamics), the reward function and the objective function.

States. The state of the system at time t is defined by the tuple $S_t = (C_t, D_t)$, where C_t is the system's configuration and D_t the demand vector.

Actions. If at time t the system is at state S_t, then the action A_t, taken at t, deterministically defines the configuration (C_{t+1}) to be applied at t + 1 (without knowing the demand vector D_{t+1} at the next time step).

Transition Probabilities. To facilitate the formulation of the problem as a finite Markov Decision Process, we will consider the case of discretized VNF/VL demands that take values from a set of demand levels $\mathcal{B}$ (with cardinality B). The transition to state $S_{t+1} = (C_{t+1}, D_{t+1})$ at t+1 depends on the action A_t, and on the probability distribution of the demand vector in the next time interval. This is assumed to be known and can possibly capture traffic correlation among VNFs of the same slice but also among slices as well. Since the only stochastic variable of the system is the demand, the probability of a transition from S_t to S_{t+1} is given by:

$$P(S_{t+1}|S_t, A_t) = P(D_{t+1}|D_t)$$

Reward. It is a function of the state S_t and the next state of the system S_{t+1}, and involves the three cost terms introduced in 6.2.1.3 through the following weighted sum:

$$r_{t+1} = -(w_1 \cdot cost_1(S_{t+1}) + w_2 \cdot cost_2(S_t, A_t) + w_3 \cdot cost_3(S_{t+1}))$$

The three weights w_1, w_2, w_3 determine the importance of each term and sum to one. Note that the immediate cost is equal to the negative reward.

Objective. The objective is to find the optimal policy $\pi^*(S)$, meaning the mapping from states to actions that maximizes the expected accumulated reward over a discounted infinite horizon:

$$\pi^*(S) = arg \max_{\pi(S)} \mathbb{E}_{\pi(S)}(\sum_{k=0}^{\infty} \gamma^k r_{t+k})$$

Note that the expectation is over the transition probabilities $P(S_{t+1}|S_t, A_t)$ while following policy $\pi(S)$. Moreover, $\gamma \in [0,1)$ is the discount factor, where for larger values of γ future costs become more important.

6.2.2.2 POLICY ITERATION

Policy Iteration (PI) is a standard Dynamic Programming algorithm; details about its structure can be found in most DP/RL textbooks (e.g., [31]). PI computes offline the optimal policy using an iterative procedure of consecutive policy evaluation and policy improvement stages.

It is applicable in the slice embedding problem when the traffic demand values are discrete, and their statistics (transition probabilities) are known. The advantage of this algorithm is that it is guaranteed to converge to the optimal policy. However, traffic dynamics are usually not known beforehand in a realistic slice embedding scenario. Moreover, PI is plagued by the state and action complexity, which rapidly increases its computational and memory requirements, while at the same time a key characteristic of the slice embedding problem is its immense number of states and actions. As an example, let's consider just a single domain network with slices that comprise only one VNF each (no virtual links), and all VNFs have the same number of traffic demand levels B. Then, the state space consists of $|\boldsymbol{S}| = V^K \cdot B^K$ states, since each VNF/slice (K in total) can be placed on 1 out of V servers and the corresponding demand of each slice takes 1 out of B values. Even in this toy scenario and for a moderate number of servers ($V = 10$), slices ($K = 10$) and demand levels ($B = 10$), the cardinality of the state space soars to 10^{20} states while there are 10^{10} actions per state. Consequently, PI is practically applicable only in small sized toy scenarios of the slice embedding problem, as it suffers from the curse of dimensionality.

According to the above-mentioned characteristics, we utilize PI in the validation section as the optimal baseline in small sized MDP scenarios, where the traffic demand is discrete. However, to tackle realistic scenarios we need to employ an algorithm with the ability to work under unknown traffic dynamics and large state and action spaces.

6.2.2.3 Q-LEARNING

Q-learning (QL) is a standard tabular RL algorithm; details can be found in any RL textbook, e.g., [32]. It is an off-policy temporal difference method that learns the optimal action value function $Q^*(S, A)$ through interaction with the system (training). The optimal action value function is represented by a table, called the Q-table, with as many entries as the number of possible state-action pairs.

QL is a better fit for the slice embedding problem compared to PI, since it does not require any knowledge of traffic statistics beforehand. It is also guaranteed to converge to the optimal policy under minimal conditions. However, QL still suffers from the curse of dimensionality, which renders it not applicable in realistically sized scenarios (or scenarios where the traffic demand is continuous). Recalling the toy example of 6.2.2.2, the QL algorithm would have to learn the correct values of $10^{20} \cdot 10^{10} = 10^{30}$ table entries, which would require a vast amount of memory and computation time. Note that QL updates only the value of the state-action pair encountered each time, so many visits at each state-action pair are required for the algorithm to converge.

We utilize QL as another baseline in small sized MDP toy scenarios in the validation section. However, it is obvious that the tabular representation of the Q function is an obstacle to be overcome in order to tackle larger scale problems.

6.2.2.4 DQN

A way to reduce the state complexity of the problem at hand is to use a parametric function to approximate the action value function instead of the tabular representation in QL. Towards this direction, we apply a DQN, introduced in [33], which employs a deep neural network for the approximation of the Q function, and mechanisms like the experience replay memory and the target network that counteract to any instabilities caused during the learning process due to the use of a neural network approximator.

Architecture. We have retained the main architectural elements and features of the DQN as they were introduced in [33]. The only major difference is that the DNN we use is just a simple Multi-Layer Perceptron (MLP) comprising three hidden layers. This choice is towards having an architecture that is as simple as possible for the specific problem but without compromising the algorithm's performance. The input to the MLP is the state of the system, represented by the vector S as defined in subsection 6.2.2.1. The number of neurons at the output layer is equal to the number of possible actions (system configurations), and each output neuron corresponds to the Q-value of an action. As for the hidden layers, each of them is equipped with 60 neurons while its output is activated by a Rectifying Linear Unit (ReLU) function.

Application scenarios. DQN can be applied in the slice embedding problem to reduce state complexity when the traffic demands are either discrete or continuous and the number of possible system configurations is not very large. This algorithm is not suitable for large action spaces because the number of output neurons of the DNN is equal to the number of possible actions. Since, in realistic scenarios, the problem at hand demonstrates a high action complexity, a slightly different approach is required to deal with the large action space and enable the use of DQN.

6.2.2.5 IDQN

One way to reduce the action complexity in Q learning is to use multiple QL agents instead of a single centralized one [34]. In that way, the central action can be decomposed into several components, each taken by a different agent. Towards reducing both the state and the action complexity, the most straightforward approach is to simply combine the multi-agent concept with the DQN algorithm [35]. This results to a multi-agent scheme with Independent DQN agents (IDQN), which uses approximation both in state and action spaces, and is the proposed algorithm for better scalability.

Action decomposition. For the slice embedding problem, the decomposition of the action into multiple components can be realized in different levels, e.g., each agent may be responsible for the configuration of a slice or even the configuration of a single VNF. In the remainder we consider one agent per VNF to completely avoid the combinatorial aspect of the action and maximize the algorithm's scalability. Using this type of decomposition, the action for each agent is to decide the node where the associated VNF will be placed in the next timeslot. Consequently, the number of possible actions per agent is equal to the number of nodes that can host the VNF.

Inputs and Outputs. Here we consider Independent DQN (IDQN) agents that work cooperatively towards a common goal. So, all agents have access to the global state of the system and the global cost of their actions, but they are independent since they have no information regarding the behavior of the other agents. Consequently, the input and the reward for all agents is the same as in the centralized DQN version (the input is the global state and the reward is the negative cost incurred by the collective action of all agents), but the output is different for each agent, since it corresponds to the Q-values of all possible agent-specific actions.

Architecture. The IDQN architecture for each agent is similar to the single centralized DQN described in 6.2.2.4. So, each agent maintains a DNN and has its own set of parameters, its policy and target networks, and replay buffer. The important difference is that the number of output neurons is significantly lower now due to the decomposition of the centralized action (it is equal to the number of nodes where the associated VNF can be placed).

6.2.3 FIT INTO THE DE DISTRIBUTED ARCHITECTURE

The proposed IDQN algorithm can support multiple slices/VNFs across different technological domains considering e2e KPIs. Consequently, it could be positioned in the IDMO functional block of the MonB5G architecture, presented in Chapter 3. As discussed in Subsection 6.2.2.5, the action can be decomposed so that each IDQN agent is responsible for the placement (embedding) of a slice (e2e) or even for the placement of just a VNF. Hence, since an IDQN agent can be perceived as a DE, the distribution of the DEs is flexible (either 1 DE per slice or 1 DE per VNF). In the validation section we consider 1 DE per VNF in order to have the minimum number of output neurons per agent. In Figure 34 there is an example demonstrating the distribution of the DEs. In this example we consider a physical network with two technological domains (Edge and Core), each of them comprising two servers. There is also one slice with two VNFs to be embedded (one per domain). A DE is attached to each VNF and is responsible for selecting its host server. Each DE receives at its input the state of the system and the reward of the collective action by the MS (and possibly the AE). Note that the agents are conceptually distributed, in order to tackle the high complexity of the problem. Moreover, the computations can be distributed by assigning each agent to a different CPU core. However, the agents still require global information to make a decision. So, this is not a fully decentralized algorithm as the MS and the AE must have a wide view of the system.

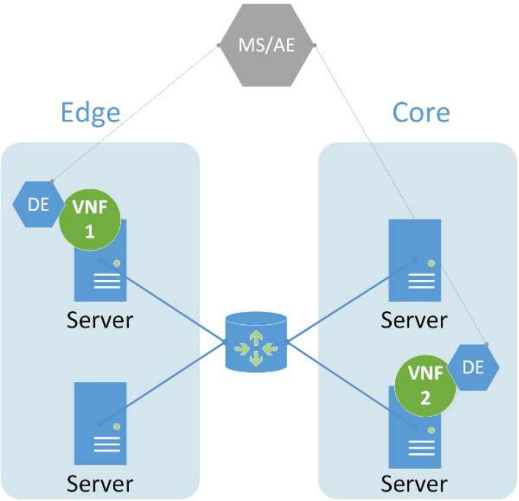


Figure 34. Link to MonB5G architecture.

6.2.4 VALIDATION RESULTS

The validation section is divided into two parts. In the first part we examine the scalability of the algorithms presented in 6.2.2 by comparing both their convergence speed and the optimality of the obtained policies. After verifying the better scalability of IDQN compared to the centralized solutions, we will apply it in a realistic large-scale scenario and compare its performance with static baseline policies.

Algorithmic parameters. In the scenarios that follow, all algorithms use the same γ value ($\gamma = 0.9$) and their exploration rate is set constant to $\varepsilon = 0.1$ during training (to facilitate the comparison of their convergence speed). Regarding the hyperparameters of the DRL algorithms, we use the following values both for DQN and IDQN, minibatch size: 32, target network update period: 500 timeslots, experience replay memory size: 5000 timeslots.

6.2.4.1 SCALABILITY OF ALGORITHMS

Setup parameters. We consider a PN that consists of a single technological domain (e.g., the Edge). The number of servers and the number of slices are both tunable, and we set them according to the size of the problem we want to examine. Towards keeping the scenario as simple as possible and without loss of generality, the slices to be embedded comprise just one VNF each. The traffic demand of each VNF/slice transitions between two levels of demand (it is either 0 or 1) according to the following Markovian transition probability matrix:

$$P_k = \begin{bmatrix} 0.98 & 0.02 \\ 0.02 & 0.98 \end{bmatrix}$$

This captures a very simple scenario where each slice has bursty traffic periods followed by long silence periods, not necessarily coinciding, to better illustrate the optimality of the chosen actions, as well as the performance of static heuristics. We also assume that traffic demands of different slices are independent (which can be a realistic scenario since each slice is devoted to a different application). Note that the use of Markovian traffic in this first part of the validation section facilitates the use of PI algorithm that is guaranteed to converge to the optimal policy and can provide a benchmark. Using the above dynamics we generate two different datasets, one for the training of the algorithms ($2 \cdot 10^6$ timeslots) and one for their testing (80000 timeslots). As for the SLAs, we consider the underprovision SLA, where $q_k = 0$ for all slices and a quadratic penalty is incurred when SLA violations occur.

Tested Scenarios. We will examine two different scenarios towards validating the scalability of IDQN. First, a small-scale scenario, where we can apply all four algorithms of section 6.2.2 (PI, QL, DQN, IDQN). This will allow us to compare both the convergence speed of the (D)RL algorithms and the quality of the obtained policies with respect to the optimal one given by PI. After verifying that both DQN algorithms can obtain near-optimal policies faster than tabular QL, we will further increase the size of the problem and examine how the performance of DQN and IDQN is affected.

Scenario 1 – Small size problem. In this scenario we consider a single domain PN with 2 servers and 4 slices. Each of the algorithms is trained and tested over the respective datasets 10 individual times, with a different random seed each time. The results are given in Figure 35 and Figure 36. The plot of Figure 35 depicts the cost incurred by each algorithm (averaged over the 10 runs) as a function of the timeslot during training. This plot demonstrates the faster convergence of both DQN algorithms compared to QL, despite the small size of the problem. However, IDQN does not seem to convergence faster than DQN. This is because in this scenario the number of actions at each state is 16, which is still very small to have any impact on the learning process of the DQN.

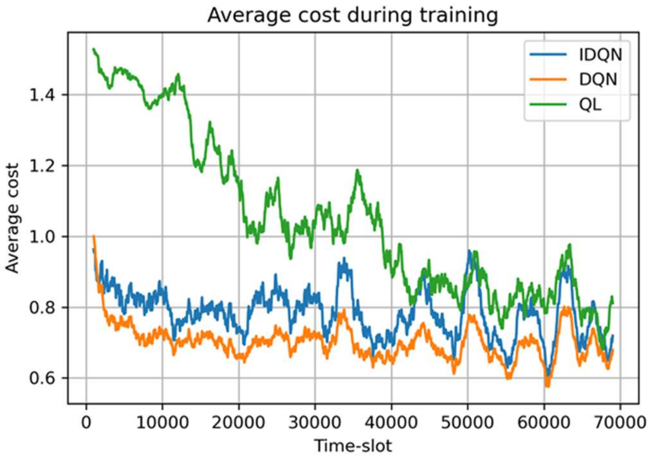


Figure 35. Convergence speed comparison for IDQN, DQN and QL algorithms (scenario 1 - small scale problem).

In Figure 36, there is a box plot comparing the average cost per timeslot in the testing stage by each of the algorithms for the 10 runs. This plot includes the minimum cost given by the optimal policy (PI), as well as the cost of two static baseline policies (split all and group all) and the cost of taking random actions. Note that split all policy minimizes SLA violations by splitting the VNFs to all available nodes, while group all policy minimizes energy consumption by placing all VNFs on the node with the largest capacity. It can be deduced that despite using Q function approximation, the derived policies by DQN and IDQN are close to the true optimal. Of course, the cost of IDQN demonstrates larger variance, due to the additional approximation in the action space. However, IDQN still performs much better than the two static baseline policies used as benchmarks (split all and group all).

Consequently, this first scenario verifies that DQN and IDQN can convergence to near optimal policies faster than QL, and those sequential decision-making algorithms can provide dynamic policies that are more efficient compared to some heuristic baselines (static policies).

Scenario 2 – Moderate size problem. In this scenario we consider a single domain PN with 3 servers and 7 slices. This time we apply only DQN and IDQN, since the high computation and memory requirements of PI and QL are prohibitive (e.g., QL requires a Q-table with 6×10^8 table entries). We follow the same procedure as in Scenario 1, and the results are given in Figure 37. *IDQN converges roughly* $10 \times$ *faster than DQN, to a policy of similar quality.* This is due to the size of the action space which is starting to grow large and makes the training of a centralized agent slower. Note that for this scenario the MLP of a centralized agent requires 2187 output neurons while the MLP of a distributed IDQN agent requires only 3.

All in all, in this section we have verified the scalability of IDQN compared to the centralized solutions (QL and DQN).

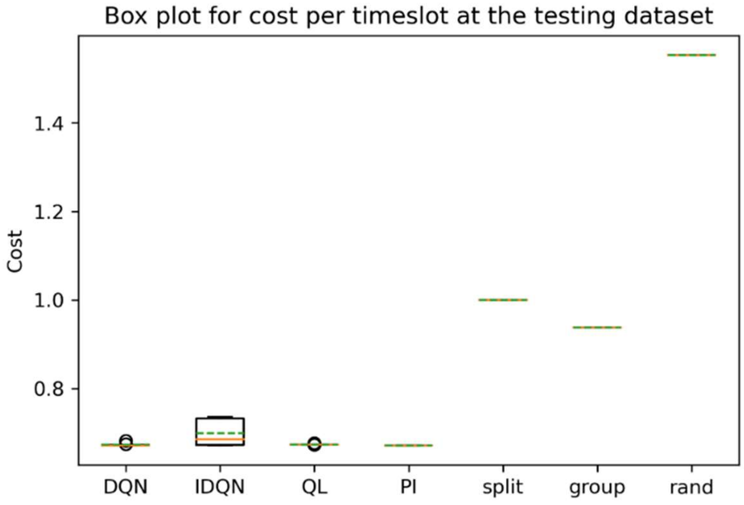


Figure 36. Cost comparison of (D)RL algorithms with optimal and baseline policies (scenario 1 - small scale problem)

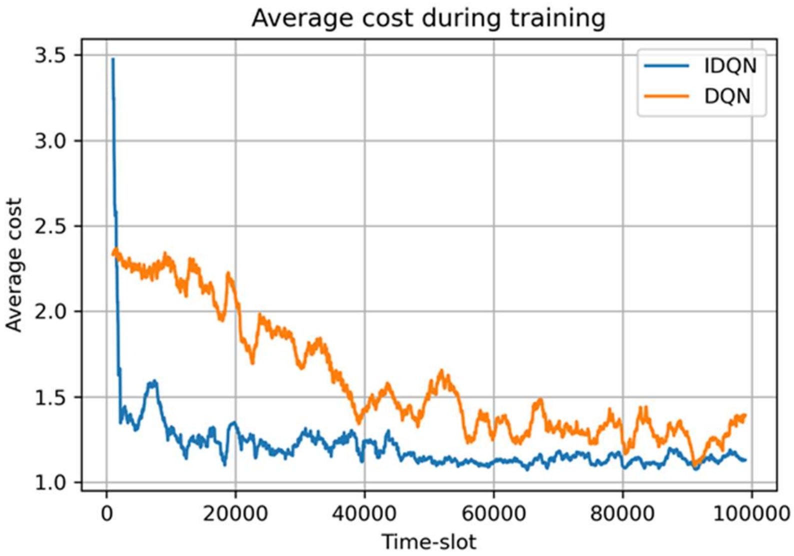


Figure 37. Convergence speed comparison for IDQN and DQN algorithms (scenario 2 - larger scale problem).

6.2.4.2 APPLICATION OF IDQN IN LARGE SCALE SCENARIO

Setup parameters. We consider a PN consisting of two technological domains, the Edge, and the Core. In this second part of the validation section, we consider SLAs related to the e2e queuing delay (linear penalty for SLA violations), so each server is characterized by its mean service rate. The Edge comprises 9 servers while the Core consists of 3 servers. We made this choice since it is more realistic to have a higher number of servers with lower mean service rates distributed on the Edge and a lower number of servers with higher mean service rates in the Core network (large datacenters). Moreover, there are 10 slices with different SLAs to be embedded on top of the network. Each slice comprises two VNFs, one for the Edge and one for the Core domain (for simplicity we assume again that there are no VLs involved). The traffic demand of the VNFs is continuous with unknown time dependence, as we import it from the Milano dataset. More specifically, we map the timeseries of a different base station to each VNF and choose correlated timeseries for VNFs of the same slice (to make the setup as realistic as possible).

Results. The results are given in Figure 38, which demonstrates the average cost per timeslot in the testing dataset (averaged over 10 runs) for each of the policies. Since it is not possible to apply a centralized algorithm in such a large-scale scenario (a centralized DQN would require 2×10^{14} output neurons), we use only the static baseline policies as benchmarks. *According to the results IDQN demonstrates 43% lower cost compared to the baseline policies.*

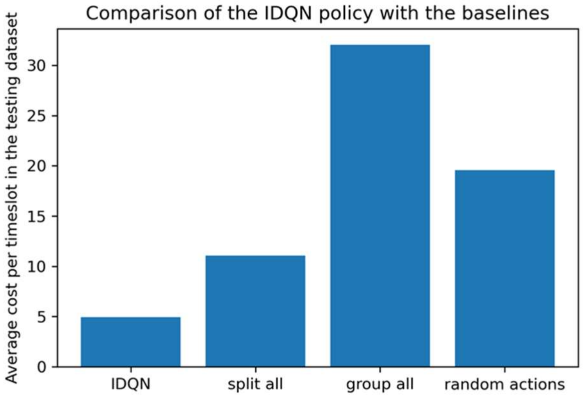


Figure 38. Cost comparison of IDQN with baseline policies (large scale scenario).

6.2.5 MONB5G RELATED CHECKBOXES/KPIS

The proposed IDQN achieves one of the goals of the MonB5G architecture, which is the use of multi-agent solutions. More specifically IDQN is employed for the task of inter-slice orchestration and minimizes the combined cost of energy consumption, SLA violations and reconfigurations in the long run. The most important attribute of this algorithm is its scalability compared to the centralized solutions, which was verified in the validation section (roughly $10\times$ improvement in convergence speed compared to the centralized DQN). The MonB5G related checkboxes are outlined below:

- ✓ Multi-agent algorithm
- ✓ Convergence speed improvement (10x improvement according to the validation section)
- ✓ Flexible DE distribution (1 DE per slice, or even 1 DE per VNF)
- ✓ Handles multiple VNFs across different technological domains
- ✓ Supports diverse KPIs and service level agreements

6.3 Specialization of FDRL Agents for Scalable RAN Slicing Orchestration

6.3.1 PROBLEM DESCRIPTION

End-user mobility and temporal variations of the traffic demand deeply complicate resource planning and allocation tasks, especially in the radio access network (RAN) domain where resource allocation decisions, e.g., in terms of bandwidth, must cope with the additional variability inherent of the wireless channel.

Traditional RAN slicing solutions envision a centralized controller with a holistic and real-time view of the network, especially about resource utilization, availability, and real-time wireless channel statistics. However, similar approaches suffer from scalability issues in real deployments, where the amount of monitoring information to be exchanged, together with the massive number of deployed base stations (BSs), make it practically impossible to devise optimal resource allocation schemes in a timely and resource-efficient manner.

We take on this challenge and propose a distributed architecture for network slice resource orchestration. Given the variable spatio-temporal distribution of mobile traffic demands [30], we envision the dynamic setup of a network of local decision agents (DAs) as virtual software instances able to access local RAN monitoring information and extract local knowledge without the need of a centralized entity performing decisions on aggregated information.

Our framework leverages a dynamic agent selection mechanism based on local traffic conditions similarity, which enables more efficient information exchange and collaboration among groups of local DAs while specializing their decision policy. The benefit coming from our approach are several:

- it enables resource allocation at the edge of the network, thus accounting for more timely and accurate information
- the amount of control information that needs to cross the network to reach the central controller dramatically decreases, thus reducing overhead towards the core network and avoiding bottlenecks
- by allowing information exchange among local DAs, we enable the provisioning of federated learning schemes to further enrich the capabilities of the decision agents.

Let us introduce a mobile network infrastructure composed of a set $\mathcal{B}$ of base stations (BSs), wherein a set of slices $\mathcal{I}$ is deployed. Each BS $b \in \mathcal{B}$ is characterized by a capacity $\mathrm{C_b}$, expressed in terms of a discrete number of physical resource blocks (PRBs) of a fixed bandwidth.

This resource availability must be divided into subsets of PRBs, and dynamically assigned to each network slice according to their real-time traffic demand and SLA requirements. As part of the SLA between the network operator and the slice owner, we assume each network slice to come with predefined latency and throughput requirements defined by the variables Λ_i and λ_i, respectively. Let us consider a time-slotted system where time is divided into *decision intervals* $t \in T = \{1, 2, \dots, T\}$. The PRB allocation decisions can be taken only at the beginning of each decision interval, whose duration ϵ may be decided according to the infrastructure provider policies, ranging from few seconds up to several minutes.

We assume the presence of a preliminary admission and control mechanism, e.g., the one presented in [36], to verify the admissibility of the current network slice setup within the available networking capacity, and focus our effort on meeting the resource allocation for the downlink traffic.

We envision the allocation of radio resources towards the end-users as a two-step process [37]. Initially, once network slices are admitted into the system, the infrastructure provider schedules the assignment of slots of radio resources for each of the tenants. Then, based on the slice resource availability, each tenant may decide to enforce proprietary scheduling solutions towards its end-users, depending on use-case or business requirements [38].

We focus on the correct and fair dimensioning of the inter-slice PRB allocation. To this aim, we denote with the variable $\mathrm{a}_{i,b}^{(\mathrm{t})}$ the PRB allocation decision for the i-th slice under the b-th BS taken at t-th decision time interval, and with $\sigma_{i,b}^{(\mathrm{t})}$ the signal-to-noise ratio (SNR) value expressing the wireless channel quality, averaged over the duration of a decision time interval ϵ, and over the end-users of the i-th slice attached to the b-th BS. Similarly, we introduce $\phi_{i,b}^{(\mathrm{t})}$ as the aggregated downlink traffic demand generated by the users of the i-th slice under the coverage area of the b-th BS within the t-th time interval.

All together, we can formalize our problem as:

Problem RAN Resource Allocation:

$$\min \lim_{T\to\infty} \sum_{t=1}^{T} \mathbb{E}\left[\sum\nolimits_{i\in\mathcal{J}} \mathrm{d}_{i,b}^{(\mathrm{t})}\right]$$

s. t.:

$$\mathrm{E}_{i,b}^{(\mathrm{t})} \le \Lambda_i\,, \qquad \forall\, t \in \mathcal{T}, i \in \mathcal{J}, b \in \mathcal{B},$$

$$\sum\nolimits_{i\in\mathcal{J}} \mathrm{a}_{i,b}^{(\mathrm{t})} \le C_b\,, \qquad \forall\, t \in \mathcal{T}, b \in \mathcal{B},$$

$$\mathrm{a}_{i,b}^{(\mathrm{t})} \in \mathbb{Z}^+\,, \mathrm{d}_{i,b}^{(\mathrm{t})} \in \mathbb{R}^+, \qquad \forall t \in \mathcal{T}, i \in \mathcal{J}, b \in \mathcal{B},$$

where $\mathrm{E}_{i,b}^{(\mathrm{t})} = \mathbb{E}\left[\frac{\phi_{i,b}^{(\mathrm{t})}}{\Gamma\left(\mathrm{a}_{i,b}^{(\mathrm{t})}, \sigma_{i,b}^{(\mathrm{t})}\right) + \mathrm{d}_{i,b}^{(\mathrm{t})}}\right]$ defines the expected transmission latency, and $\Gamma(a,\sigma)$ is a function that translates the PRB allocation a in the equivalent transmission capacity, given the experienced channel quality σ.

The traffic demand generated within a decision interval might not be fully satisfied due to erroneous PRB allocation estimations, incurring in additional transmission delay due to traffic queuing at the base station. Therefore, we introduce the variable $\mathrm{d}_{i,b}^{(\mathrm{t})}$ as a deficit value indicating the volume of traffic not served within the agreed slice latency tolerance Λ_i , and that is therefore dropped.

6.3.2 A MULTI-AGENT ARCHITECTURE FOR RAN RESOURCE ALLOCATION

Due to fast traffic variations, slice resource allocation decisions at the RAN domain should be taken in a dynamic, proactive, and flexible way to avoid service and performance degradation.

While advanced admission and control mechanisms could select the set of slices to be admitted to the system and provide static resource allocation boundaries to satisfy the available capacity, the dynamic nature of the slice's traffic load and wireless channel statistics may lead to suboptimal performances. Additionally, the optimization problem underlying RAN resource allocation is a non-linear integer programming problem, which has been proven to be NP-hard [36], and suffers scalability issues when addressed in a centralized manner.

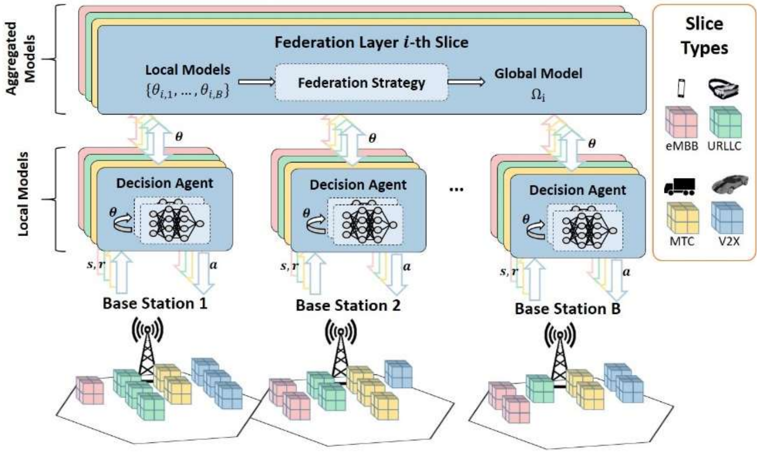


Figure 39 Federated DRL architecture for RAN slicing.

For these reasons, we advocate for the adoption of an FDRL-based architecture to address the RAN slicing scenario. We rely on local DAs running as software instances within the premises of each BS, as shown in Figure 39. 1. Each agent oversees performing slice PRB allocation decisions based on local monitoring information coming from the underlying network monitoring system, or BS *context*.

To concurrently address the above issues, we introduce an FL layer that allows inter-agent information exchange and expedites the learning procedure local knowledge sharing.

6.3.3 LOCAL RAN SLICING VIA DDQN AGENTS

DQN is a popular reinforcement learning [39] algorithm that evolves from the well-known concepts of Q-learning and neural network function approximation.

DQN represents a model-free approach. It stores the trajectory of experiences for each interaction with the environment in a replay buffer, as to update the network parameters without prior knowledge of the underlying environment statistics.

In the following, we will use the i index interchangeably while referring to slices and DAs, assuming a one-to-one mapping of each DA with the corresponding network slice.

With focus on a single BS and a single decision interval the design choices of our DQN model are as follows:

State Space $\mathcal{S}$ we define the state of the i-th agent associated to the b-th BS as a tuple of local monitoring information $s_i^{(t)} = \left\{\left(\sigma_i^{(t)}, \lambda_i^{(t)}, \nu_i^{(t)}\right), \forall i \in \mathcal{I}\right\}$, where $\sigma_i^{(t)}$ is the SNR value, averaged over the duration of a decision time interval experienced by the users of the i-th slice, $\lambda_i^{(t)}$ is the aggregated traffic volume generated by the i^{th} slice over the time decision duration ϵ, and $\nu_i^{(t)}$ is the amount of available capacity left by the previous decisions of other agents.

Action Space $\mathcal{A}$ without loss of generality, we define ι as the minimum PRB allocation step, or *chunk size*, and assume that the PRB allocation decision of the i-th agent can only take values that are an integer multiple of ι, hence $= \left\{\iota \cdot k \mid k = \left\{0,1,\dots,\left\lfloor \frac{C_b}{\iota} \right\rfloor\right\}\right\}$.

Such discrete action space allows controlling the dimensionality of the action space and positively influences the learning process [40].

Reward $\mathcal{R}$ we adopt an iterative reward-penalty approach to guide the agent learning procedure, which translates into maximizing a reward function. An accurate PRB allocation should concurrently guarantee the satisfaction of transmission latency Λ_i and the traffic requirements $\lambda_i^{(t)}$, while avoiding both under-provisioning and over-provisioning of resources. Given the instantaneous slice traffic volume $\lambda_i^{(t)}$, and the corresponding allocation decision $a_i^{(t)} \in \mathcal{A}$, we can identify an *allocation gap* $\alpha_i^{(t)} = \Gamma\left(a_i^{(t)}, \sigma_i^{(t)}\right) - \lambda_i^{(t)}$. To measure the goodness of the action, we therefore introduce two variables, namely $\rho_{UP}^{(t)}$ and $\rho_{LOWER}^{(t)}$, which characterize the upper and lower boundaries of the allocation gap as $\rho_{UP}^{(t)} = 2\Gamma\left(\iota, \sigma_i^{(t)}\right)$, and $\rho_{LOWER}^{(t)} = -\Gamma\left(\iota, \sigma_i^{(t)}\right)$. Accordingly, we define the instantaneous reward $r_i^{(t)} \in \mathbb{R}$ of the i-th agent as:

$$r_i^{(t)} = \begin{cases} \alpha_i^{(t)} - 4\rho_{LOWER}^{(t)}, & \text{if } \alpha_i^{(t)} \leq \rho_{LOWER}^{(t)} \\ \left(1 - \frac{\alpha_i^{(t)}}{\rho_{UP}^{(t)}}\right)\frac{\alpha_i^{(t)}}{\rho_{UP}^{(t)}}, & \text{if } \rho_{LOWER}^{(t)} < \alpha_i^{(t)} \leq \rho_{UP}^{(t)} \\ -\left(\alpha_i^{(t)} - \rho_{UP}^{(t)}\right), & \text{if } \alpha_i^{(t)} > \rho_{UP}^{(t)} \end{cases}$$

Notably, the first case linearly penalizes the occurrence of under provisioning decisions, while the third case acts in a similar way on the over-provisioning cases. The middle case is the target scenario, which assumes correct PRB allocation decisions in response to the instantaneous slice traffic request.

We envision the multi-agent RAN slicing problem as a sequential procedure, where at the beginning of each decision interval t, the different agents perform local decisions according to a priority valueμ_i. Nevertheless, multiple and independent agents may perform inaccurate decisions and leave the subsequent agents with no spare resources, especially in the initial training phase. Therefore, at the end of each training period, we calculate a penalty $\mathrm{P}_i^{(t)} = \eta_i \mathbf{1}\left(\mathrm{a}_i^{(t)} - \mathrm{v}_i^{(t)}\right)$, where η_i is the penalty coefficient of the i-th slice, and $\mathbf{1}(\cdot)$ denotes the logical operator. This penalty overrides the instantaneous agent reward r_i if the decision $\mathrm{a}_i^{(t)}$is greater than the number of spare resources left by the previous decisions of the other agents that in turn prevents the agents to exceed the available resources at the base station.

Training of agents the training of the local agent implies the characterization of the action-value function : $\mathcal{S} \rightarrow \mathcal{A}$. Let us define the policy π as a probabilistic function mapping states to actions. The agent makes decisions and selects the corresponding actions based on π, determining the best action for each state. Under a given policy π, the action-value function can be defined as, $Q_\pi(s^{(t)}, a^{(t)})$ = $E_\pi\left[\gamma^n r^{(t+n+1)} | s^{(t)}, a^{(t)}\right]$, where $\gamma \in [0,1]$ is a discount factor that weights the short-sighted and far-sighted reward, and n is the temporal index. According to Bellman's equation [31], the optimal state-action value function can be expressed as $Q^*(s^{(t)}, a^{(t)}) = E_\pi\left[r^{(t)} + \gamma \max_{a^{t+1}} Q^*\left(s^{(t+1)}, a^{(t+1)}\right) | s^{(t)}, a^{(t)}\right]$, and thereby the Q-learning update rule based on temporal difference (TD) [31] is given by,

$$Q(s^{(t)}, a^{(t)}) \leftarrow Q\left(s^{(t)}, a^{(t)}\right) + \xi\left[r^{(t)} - \gamma \max_{a^{(t+1)}}\left(Q\left(s^{(t+1)}, a^{(t+1)}\right) - Q\left(s^{(t+1)}, a^{(t+1)}\right)\right)\right]$$

where ξ is the learning rate. DQN adopts deep neural network (DNN) to approximate the state-action value and surmount the curse of dimensionality concerning inordinate large state spaces. To limit the catastrophic interference problem [41], which is the tendency of a neural network to forget about previously learned information upon learning new ones, we adopt an experience replay strategy. Let us introduce β_i as the experience buffer.

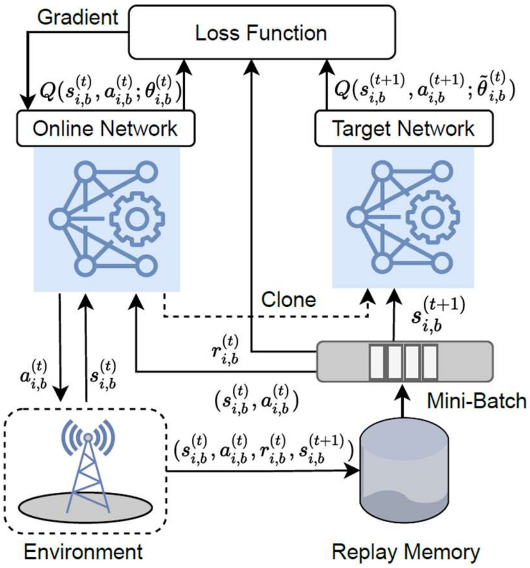

Gradient
Loss Function
$Q(s_{i,b}^{(t)}, a_{i,b}^{(t)}; \theta_{i,b}^{(t)})$
$Q(s_{i,b}^{(t+1)}, a_{i,b}^{(t+1)}; \tilde{\theta}_{i,b}^{(t)})$
Online Network
Target Network
Clone
$s_{i,b}^{(t+1)}$
$r_{i,b}^{(t)}$
$a_{i,b}^{(t)}$
$s_{i,b}^{(t)}$
$(s_{i,b}^{(t)}, a_{i,b}^{(t)})$
Mini-Batch
$(s_{i,b}^{(t)}, a_{i,b}^{(t)}, r_{i,b}^{(t)}, s_{i,b}^{(t+1)})$
Environment
Replay Memory

Figure 40 Illustration of DDQN workflow.

As depicted in Figure 40, in every training interval, we store the tuple $(s_i^{(t)}, a_i^{(t)}, r_i^{(t)}, s_i^{(t+1)})$ describing the instantaneous experience generated by the agent while interacting with the environment, and sample from β_ia random batch of past experiences to regularize the training. Additionally, DQNs are well known to provide an overoptimistic value estimation. We alleviate this problem by leveraging an additional DQN network, in the form of DDQN [42]-[43]. With a slight abuse of notation, let us introduce $Q(s^{(t)}, a^{(t)}; \theta_i^{(t)})$ and $Q(s^{(t)}, a^{(t)}; \tilde{\theta}_i^{(t)})$ as the online network and target network respectively, where $\theta_i^{(t)}$ and $\tilde{\theta}_i^{(t)}$ denote the model parameters. To optimize the parameter set $\theta_i^{(t)}$ and approximate the optimal action-value function $Q^*(s^{(t)}, a^{(t)})$, we use the following loss function,

$$L\left(\theta_i^{(t)}\right) = E\left[y_i^{(t)} - \ Q\left(s^{(t)}, a^{(t)}; \theta_i^{(t)}\right)\right]^2$$

where $y_i^{(t)} = = r_i^{(t)} + \gamma \max_{a^{t+1}} \ Q\left(s^{(t+1)}, a^{(t+1)}; \tilde{\theta}_i^{(t)}\right)$ and $\tilde{\theta}_i^{(t)}$ is copied from $\theta_i^{(t)}$ at the end of each episode.

Finally, the objective function of the DDQN model can be written as,

$$y_i^{(t)} = = r_i^{(t)} + \gamma\, Q\left(s_i^{(t+1)}, \underset{a^{(t+1)}}{\operatorname{argmax}} Q\left(s^{(t+1)}, a^{(t+1)}; \theta_i^{(t)}\right); \tilde{\theta}_i^{(t)}\right)$$

where $\theta_i^{(t)}$ is a local training model used for selecting actions, and $\tilde{\theta}_i^{(t)}$ is used to evaluate their values according to a different policy, thus mitigating over-estimations issues and improving the decision agents' performances [42]. The loss function estimates the difference between true action-value and target action-value. As the overall training procedure aims at minimizing this loss function, we adopt stochastic gradient descent (SGD) approach [44] to pursue this goal.

6.3.4 *FEDERATED DRL FOR RAN SLICING*

FL allows training machine learning models across multiple decentralized entities which have access to a limited set of the overall data available. Conversely to multi-agent reinforcement learning, which defines a set of autonomous agents that observe a global state (or partial state) of the system, select individual actions and receive individual rewards, FL allows to collaboratively learn a shared prediction model by iteratively aggregating multiple model updates, thus decoupling the learning procedure from the need of centralized data sources. A refined version of the original models, combination of multiple local models according to specific federation strategies, is then shared to the agents allowing to significantly improve the learning rate, ensure privacy [45] and provide better generalization [46].

As depicted in Figure 39, within the context of our FDRL-based framework each agent trains a local DDQN model $\theta_{i,b}^{(t)}$ and shares its experience, under the form of model hyperparameters, to those entities belonging to the corresponding federation layer. This iterative training approach enables each federation layer to aggregate the collected knowledge of single agents into a global updated model $\Omega_i^{(t+1)}$, usually stored into a cloud platform or a nearby edge platform to allow faster feedbacks.

In order to enhance efficiency and avoid communication overhead, we allow the federation layer to collect the local models (and share the updated ones) only every $\hat{\mathrm{T}}$ decision intervals, defining this time period as *federation episode*.

Different strategies can be adopted to derive the global federated model, each one implementing a predefined federation strategy function $f_{strategy}(\cdot)$.

In *Average* federation strategy, dubbed as *FDRL* in the following of this work, the collective federation model for the next time interval $\Omega_i^{(t+1)}$ is derived as the simple average of the incoming model weights belonging to all the agents, as $\Omega_i^{(t+1)} = \frac{1}{B}\sum_{b\in\ \mathcal{B}}\theta_{i,b}^{(t)}$.

Aggregated mobile traffic demands follow repetitive spatio-temporal trends due to human activities [47]. In this context, it is expected that a good characterization of such processes would allow more accurate forecasting of the network utilization and, in turn, enable an efficient and even proactive planning of the resource allocation.

However, as highlighted in [48], it is not enough to leverage the geographical locations and related spatial proximity of the BS to obtain a comprehensive view of traffic demands, as the land usage of the slice resources may differ even within base stations belonging to the same geographical areas. This introduces an additional issue in our framework, as not all the federated agents should exchange knowledge with each other, nor this should be restricted to only nearby entities. To address this fundamental issue, in the following we propose a clustering algorithm to guide DA subsets definition, based on network monitoring traces and their similarity.

6.3.5 *DYNAMIC TRAFFIC-AWARE AGENT SELECTION*

Given the rapid spatio-temporal variation of the traffic demand due to end-user mobility, we advocate for the setup of a clustering algorithm to derive the subset of slice agents that should exchange their local knowledge, while considering both mobility and traffic demand variations.

Let us introduce $\tau_{i,b}$ as the time series describing the downlink traffic demand of the i-th slice instantiated over base station b. Then, for each pair $j, z \in \mathcal{B}$, we can compute the similarity of the recorded monitoring information as $DTW^{(j,z)} = f_{DTW}(\tau_{i,j}, \tau_{i,z})$, where $f_{DTW}(\cdot)$ is the Dynamic Time Warping distance [49], a state-of-the-art distance metric for time series analysis.

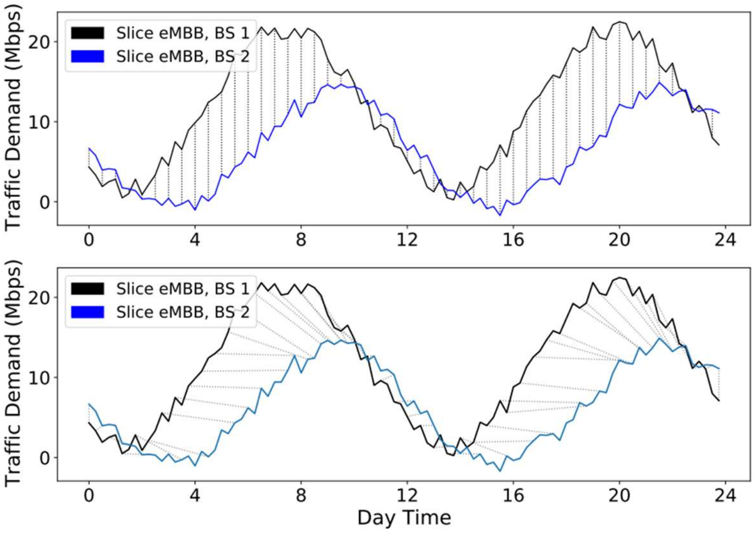


Figure 41 Comparison between Euclidean distance and Dynamic Time Warping distance over traffic demand time series.

DTW is particularly suitable in our scenario as it allows, conversely to standard distance metrics, e.g., Euclidean distance, to calculate accurate similarity value even in presence of differently sized sequences, and independently of their time shift. An example of DTW distance calculation is depicted in Figure 41, where it can be noticed how maximum and minimum values of the traces are correctly mapped to each other. The pairwise distances are then collected into the distance matrix $\boldsymbol{D} = DTW^{(j,z)} \in \mathbb{R}^{|\mathcal{B}| \times |\mathcal{B}|}$, and provided as input of our clustering algorithm.

DTW has linear space complexity, but quadratic time complexity. To reduce the latter, a number of options are available. In our case, we limit the maximal shift by setting a fixed time a window of few hours, thus reducing the complexity even in case of long sequences. Nevertheless, recent work from [50] proposed a novel efficient implementation which breaks the quadratic time complexity to $O(n^2 \, log\{n\})$, where n is the length of the sequence.

To perform the final classification, we rely on an extended version of the Density-based spatial clustering of applications with noise (DBSCAN) algorithm, introduced in [51]. DBSCAN is a non-parametric density-based clustering algorithm that allows finding the most representative points within a dataset (also known as *core samples*) based on their density in a multi-dimensional space and expands clusters from them. It expects two inputs: ϵ_{d}, representing the maximum distance between two samples for one to be considered as in the neighborhood of the other, and n_{min}, which defines the minimum number of samples in a neighborhood of a point to be considered as a core sample.

Given the above, at the end of each federation episode, we can derive in a dynamic way (and based on updated mobile monitoring information) the clusters $\Psi_k \ \in \ \Psi$, $k = \{1, \dots , |\mathcal{J}|\}$, where Ψ is the cluster set. Each cluster includes the set of base station $b \ \in \ \Psi_k$ that should be involved in the following model update procedure. Therefore, the framework spawns multiple federation models Ω_k, one for each detected cluster k, which evolve in parallel till the next federation episode.

It follows that the updated federation model, combination of the information coming from the elements of the cluster Ψ_k, can be derived following the Full-Cluster strategy, namely $f_{FC}(\cdot)$, as:

$$\Omega_{i,k}^{(t+1)} = \frac{1}{|\psi_\mathrm{k}|}\sum\nolimits_{b\in\psi_\mathrm{k}} \omega_{i,b}^{(t)}\ \theta_{i,b}^{(t)}\ ,\forall\ \psi_\mathrm{k} \in \psi$$

where $|\psi_k|$ is the cardinality of ψ_k , and $\omega_{i,b}^{(\mathrm{t})} = \frac{\hat{\mathrm{r}}_{i,b}^{(\mathrm{t})}}{\sum_{b\in\psi_\mathrm{k}} \hat{\mathrm{r}}_{i,b}^{(\mathrm{t})}}$ is a weight parameter.

It should be noted that within these settings, the federation step will occur among models with high degree of similarity, thus favoring the *specialization* of the agents towards the most common traffic statistics.

6.3.6 *VALIDATION RESULTS*

To assess the performance of the proposed architecture, we consider 3 different network slices, i.e., URLLC, eMBB, and mMTC, each one characterized by the SLA latency values of $\Lambda_i = [10{,}40{,}20]$ ms, and deployed over a wide area network comprising 50 BSs. The distribution of the BS follows the RAN deployment of the city of Milan, Italy, collected from publicly available sources[4]. We simulate user mobility patterns as per [52]. The throughput requirement of each slice depends on its user mobility pattern. Therefore, we set $\lambda_{i,b} = \varphi_{i,b}^{(t)}$ to let agents adapt to the instantaneous traffic volume. The SNR variability follows a Rayleigh distribution with average value set to 25 dB. The minimum PRB allocation step is set to $\iota \ = \ 10$ PRBs.

Per each DA we consider a DNN architecture comprised by two fully connected hidden layers with 24 neurons with ReLU activation function for both the Online and Target networks. Each learning episode consists of 5 decision intervals of duration of 60 seconds, within which local monitoring information is collected to build the local agent state. For the training, we consider Adam optimizer, a discount factor $\gamma = 0.99$, and learning rate $\xi = 0.001$. The replay buffer size of each agent $\beta_{i,b}$ is set to 20000 samples, out of which a batch of 32 samples is extracted for each episode. We set $\eta_i = 100$ as penalty value for all the slices.

6.3.6.1 LOCAL AGENT PERFORMANCE ASSESSMENT

First, we compare the performances of different RL algorithms when dealing with radio resource allocation, without involving federated learning.

[4] see https://opencellid.org

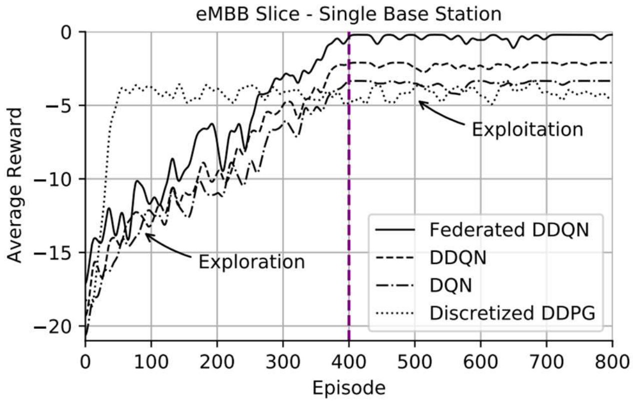


Figure 42. The convergence performance of different local decision algorithms and an FDRL approach for a single decision agent.

Figure 42 depicts the training procedure for the eMBB slice, comparing different local decision algorithms. In particular, we consider the single DQN approach, which implements standard Q-Learning procedures, discretized Deep Deterministic Policy Gradient (d-DDPG) a popular reinforcement learning algorithm, and our DDQN scheme. We gradually limit the exploration capabilities of the agents up to a minimum of 2 % in favor of the adoption of the learned policies.

The variability of the network slicing environment leads to experience learning curves with high fluctuations. As expected, the DQN approach hardly copes with the definition of suitable PRB allocation policies, providing lower performances in terms of cumulative reward and convergence time. Similarly, d-DDPG suffers the temporal periodicity of the traffic demand, resulting in a steep learning curve that saturates to suboptimal performances. Conversely, the DDQN approach can allocate in a more consistent way correct amount of PRBs to each slice according to the corresponding real-time traffic and latency demands. It is worth highlighting that in terms of convergence time, in general, FDRL schemes do not necessarily provide better performances when compared against standard DRL approaches. In fact, one of the main features of FL is that it allows local DRL agents to indirectly gain knowledge on a wider state space, extending their local experience with that coming from other DAs deployed within the same environment. This enables the DAs to provide better performances when deployed in realistic environments. Nevertheless, Figure 42 provides an overview of the local model training procedure, with and without the adoption of FL schemes. It can be noticed how DRL curves present slower convergence time and higher fluctuations when compared against Federated DDQN approach. Additionally, DRL curves present lower cumulative reward after 400 episodes, suggesting a lower capability of the DAs to adapt their decisions at the current traffic conditions.

6.3.6.2 COMPARISON OF DIFFERENT FEDERATION STRATEGIES

We consider a dynamic agent selection method based on the time similarity of traffic demands, dubbed as Dynamic Clustering (DC). In particular, we consider three DC aided approaches, namely Full-Cluster (FC), Best-Representative (BR) and Random-Representative (RR) and compare their performances against a standard federation strategy which derives a new federated model by averaging all available local models (i.e., without DC), dubbed as FDRL.

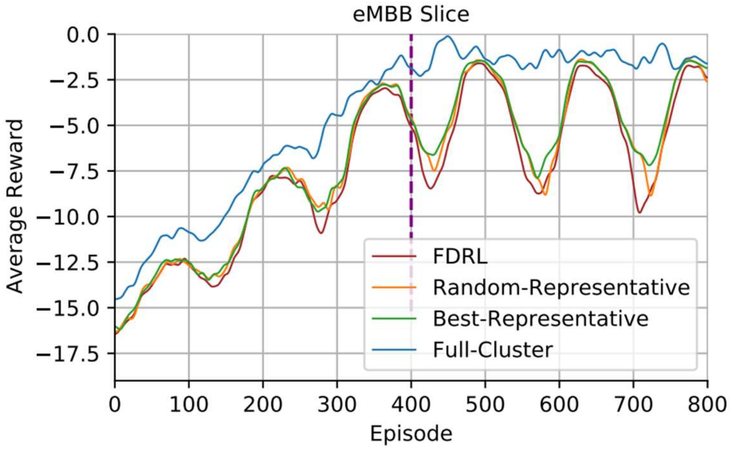


Figure 43. Comparison of global performances for different dynamic and non-dynamic federation approaches.

Figure 43 provides a comparison of learning performances for different federation strategies in terms of average reward. Interestingly, from our experiments it turns out that aggregation of widely heterogeneous local models limits the capability of the global federated model to converge to a one-fits-all unified model, motivating our dynamic agent selection approach which favors the specialization of federated agents working under similar RAN and mobility contexts. From the figure, we can observe how Full-Cluster approach achieves better generalization of the learning policies, resulting in stable performances. Conversely, Best-Representative, Random-Representative and FDRL suffer the dynamic behavior of the underlying traffic conditions, presenting inconsistent reward traces.

6.3.6.3 OVERHEAD COMPARISON FOR DIFFERENT FEDERATION STRATEGIES

Federated Learning aims at building global knowledge from the exchange of multiple locally trained models towards a centralized entity. Such a frequent model exchange however introduces significant communication overhead and synchronization issues, especially in wide scenarios as those considered in our work.

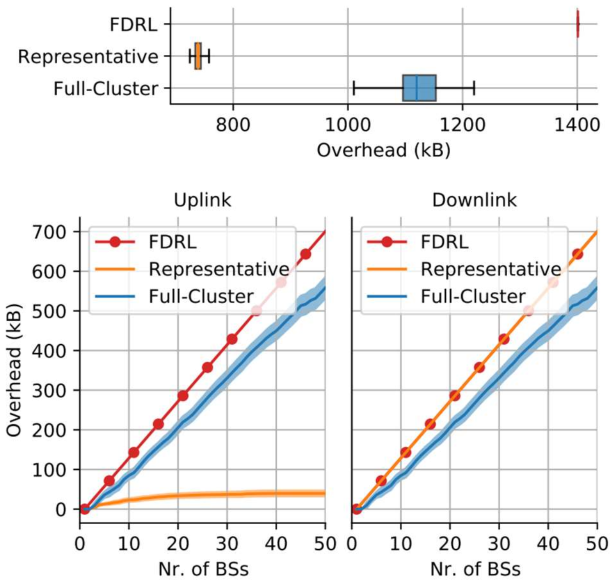


Figure 44. Communication overhead per federation episode for different federation strategies (top-part) and for different number of BSs deployed (bottom-part). RR and BR federation strategies are referred as Representative.

Figure 44 shows the model exchange overhead per federation episode for a different number of BSs. The benchmark FDRL approach assumes the exchange of all the locally trained model weights to derive the federated ones, which implies the highest communication overhead. The BR and RR approaches (referred to as Representative) allow reducing the uplink information exchange by selecting a single representative of each cluster, regardless of the dimensions of the group itself, thus minimizing the communication overhead in each federation episode. Finally, the Full-Cluster approach is characterized by an intermediate average value but higher variance due to the variable size of the DAs clusters, which follows the real-time traffic variations, saving communication resources from those base stations that presented peculiar traffic traces and remain unclustered.

On the lower part of the picture, we differentiate between uplink and downlink model exchange overhead. The FDRL approach presents a symmetric behavior matching the model exchange of all the running DAs, in both directions. Conversely, the RR/BR approaches show an asymmetric behavior that favors the upload communication with respect to the downlink one, as only a single DA per cluster shares its local model during the federation process, resulting in a logarithmic trend (with respect to the number of BSs) characterizing the overhead in uplink. This would guarantee better scalability, at the expense of suboptimal performances, as shown in our evaluation. Finally, the FC approach shows a sublinear trend, with a slower growth rate than the benchmark FDRL, but with significant better performances thanks to the specialization of the DAs. It is safe to assert that the proposed dynamic clustering approach enhances the efficiency of the federated learning scheme, limiting the overall communication overhead with respect to traditional approaches, while providing better performances.

6.4Statistical methods for VNF bottleneck localization

In Deliverable D4.1, we introduced in chapter 7.1.2, the concept of Service Function Chaining (SFC), see Figure 45. Then, we presented the theoretical modelling of the SFC monitoring use case and validated the concept with simulation.

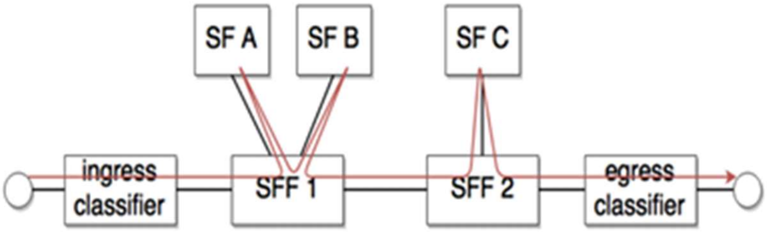


Figure 45 Service Function Chaining architecture

This section explains an application of the described solution in a more practical use case relying of on a docker based Service Chaining Function.

6.4.1 PROBLEM DESCRIPTION

In 5G networks, the core network functions are deployed with cloud native technologies as containers to guarantee the scalability and resilience. Even the evolution of the random-access network is going on this trend with the development of the cloud ran and edge cloud concepts. One of the key enablers for this evolution are the support of the heterogeneous cloud environment and the ability to orchestrate resources among multiple technical domains like RAN, edge cloud, central/core cloud, private and public cloud. All the capability coming from the hardware acceleration should be exposed to the services to enable more flexible function placement to address high performance.

The adoption of a cloud native [53] approach for the implementation and deployment of virtual network functions helps to achieve the required flexibility and autonomy. The use of containers enables to reduce the overhead and enhances the availability of the network services [54]. Future networks must be able to support multiple applications with different quality of service requirements. Eventually, the VNFs that compose these services will share the same infrastructure and will enter in competition for resources if the demand is too high. This situation may be more frequent in environments with limited resources such as the Edge [55].

The Kubernetes technology represents an essential tool for the management and orchestration of cloud infrastructures. This technology has shown great industrial maturity and is becoming the key player in cloud-native environments. It offers several features for automatic management, scalability, and services availability. Meanwhile, the price of this flexibility is losing control of the physical layer. For example, when a problem occurs on the underlay layer, all the virtual services sharing this resource may have performance degradation. The cause of such incidents may be a networking problem, a worker node failure, or a cloud/network resource shortage. Tracking the root cause of the incident is tough due to the adopted abstraction mechanisms like services and load-balancers. The proposed solution aims to adapt the tomography approach for node failure localization for anomaly root causes analysis in virtual network services deployed with Docker and Kubernetes.

6.4.2 *SOLUTION TOOL*

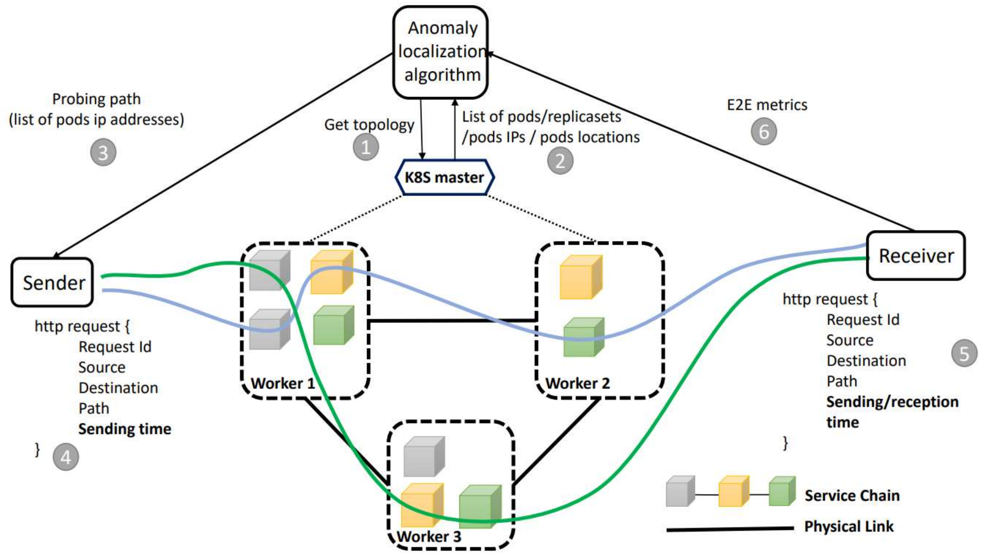


Figure 46 : Solution deployment and call flow.

This section describes an application of the proposed DE algorithm to detect the failed nodes in a network service deployed with Kubernetes.

Figure 46 explains the main steps of our solution:

- The DE sends a request to the K8S master to get the topology of the deployed service. The topology, in this case, means the list of the deployed VNFs, the number of replicas for each VNF, and the replicas pods' IP addresses. Note that each VNF is deployed as a K8S replica set.
- The K8S master sends the requested information in JSON format.
- The probing DE generates all the possible probing paths and selects the best one according to the strategy described in algorithm "Probing path selection".
- The probing path information is sent to the actuator called "sender" in *Figure 46*. This information includes the Pods' IP addresses forming the paths and other metadata.
- The Sender creates a Post HTTP request including all the path information in the body in a JSON format. This request is sent to the first pod in the list. Its IP address is removed from the path.
- Each pod repeats the same process until reaching the destination.
- The pod destination adds the sending and the reception time in the body response call back to be sent to the sender to the DE.
- The DE computes the E2E metric and computes the state of the pods denoted by α following the procedure described in Algorithm "Probing path selection".
- If the algorithm converges the anomaly detection process is finished unless the DE selects a new path and the whole process is repeated.

6.4.3 KRS: KUBERNETES RESOURCE SCHEDULER FOR RESILIENT NFV NETWORKS

6.4.3.1 KUBERNETES ARCHITECTURE

Cloud native and container technologies have significantly evolved, offering great opportunities for network operators to enhance their infrastructure management. The adoption of the cloud native approach in the implementation of VNFs enables building highly resilient and autonomous networks. Indeed, moving to Cloud native Network Functions (CNFs) for NFV networks might make it easier to overcome multiple limitations of VNFs by transforming most network functions into scalable containers. In this context, the Kubernetes ecosystem becomes an essential tool for the deployment and management of NFV infrastructures. The interest of this technology is mainly explained by its ability to manage dynamic, complex, and distributed infrastructures with great flexibility and agility.

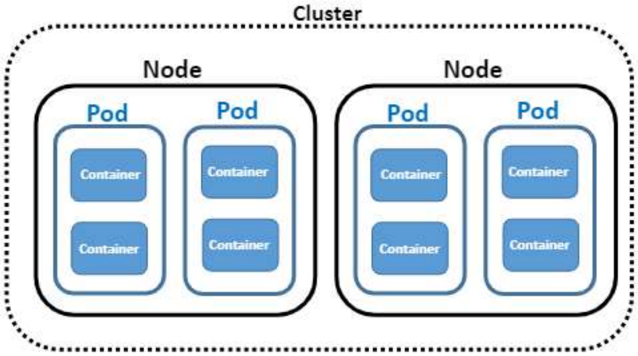


Figure 47. Kubernetes cluster components

Figure 47 illustrates the main components of the Kubernetes architecture which is based on multiple abstraction levels. The first one is the cluster which gathers multiple physical or virtual machines representing the available resources in terms of memory and CPU. Each cluster has a master node responsible for the management and scheduling of these resources. The cluster is composed of multiple nodes that can be either physical or virtual. These nodes host the pods, the most basic entity that can be created and controlled by Kubernetes. Each pod runs a single service instance (a VNF instance in our case), and it can be composed of one or several containers.

6.4.3.2 RESOURCE ALLOCATION AND SCHEDULING IN KUBERNETES

Kubernetes offers multiple tools for resource management shared by the deployed CNFs. The network infrastructure can host heterogeneous services, allowing to take full advantage of the available resources and reduce consumption costs. On the other hand, sharing the infrastructure between heterogeneous services can reveal resource allocation problems, especially in case of a resource shortage. The allocation of cloud resources must be highly efficient. They must be neither under-used, as this means a loss of revenue, nor over-used, as this can lead to service failures or SLA degradation causing penalties.

Kubernetes offers some features for pods/containers resources scheduling which are mainly requests and limits [56]. For each pod, we can configure the requested memory or CPU which correspond to the guaranteed resources that will be reserved for it. Thus, the sum of the requested resources for the containers inside a node must be less than the total node capacity. Limits specify the maximum limit of resources that can be used by a container. These parameters are very important for the CNF lifecycle management as they are taken into consideration in the priority and location of pods deployment by the scheduler and also in the allocation of resources inside a node, especially when it is overloaded.

Let's take an example of a node that hosts pods and their memory consumption reaches its limit. The Kubernetes orchestrator has to select a pod on the host to stop it. The pods that do not have a fixed requests value will be the first to be stopped. Then, the orchestrator selects those that consume the most compared to their requests value.

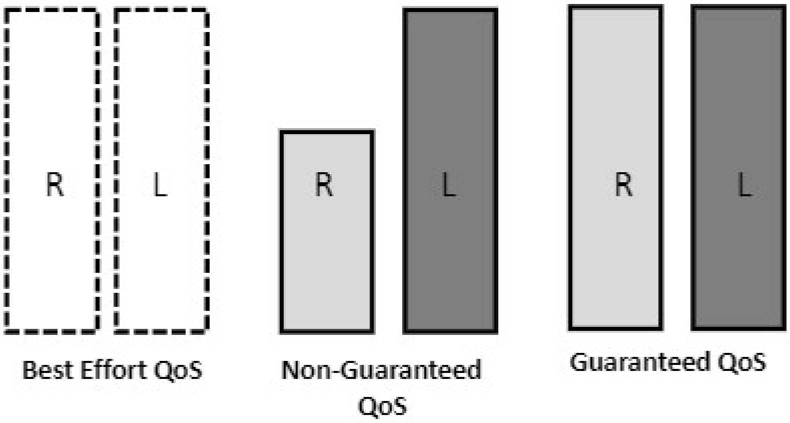
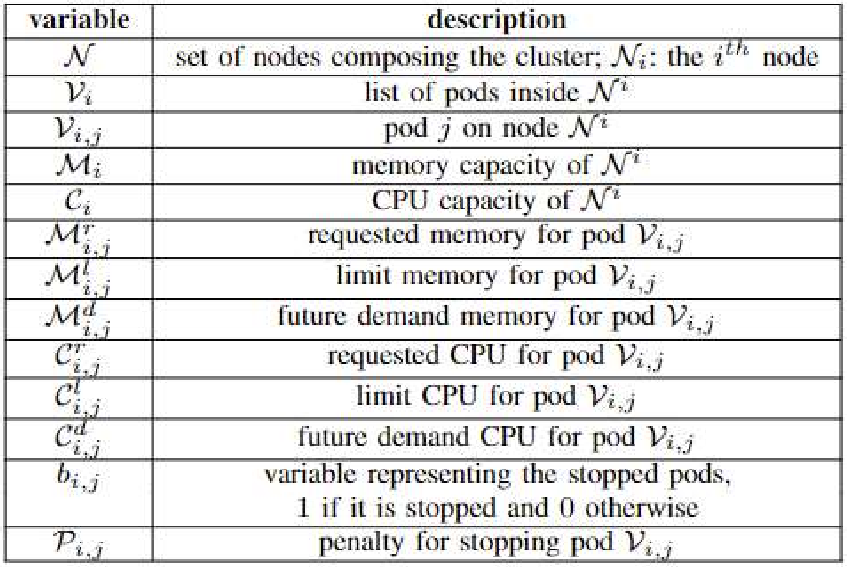


Figure 48: Different CNF deployment configurations

For instance, in Figure 48 we represent three pods deployed on a single node. The first one is deployed with "best-effort" QoS, the requests and limits are not specified. The second is deployed with a "non-guaranteed" QoS, where the requests value is less than the limits. The third one has a "guaranteed" QoS since the requests value is equal to the limits. In this situation, if there is a lack of resources, the pods will be stopped in the given order, the first one, then the second one, and finally the third one, to liberate the required resources.

The objective of this work is to give a method to schedule the resources for the pods hosting the VNF instances, to limit the failures and the caused damage in the case of lack of resources. Our strategy aims to avoid the resource shortage in the first step. However, if it is unavoidable, the goal is to protect the most critical functions.

Table 4. List of main used variables



variable	description
$\mathcal{N}$	set of nodes composing the cluster; $\mathcal{N}_i$: the i^{th} node
$\mathcal{V}_i$	list of pods inside $\mathcal{N}^i$
$\mathcal{V}_{i,j}$	pod j on node $\mathcal{N}^i$
$\mathcal{M}_i$	memory capacity of $\mathcal{N}^i$
$\mathcal{C}_i$	CPU capacity of $\mathcal{N}^i$
$\mathcal{M}^r_{i,j}$	requested memory for pod $\mathcal{V}_{i,j}$
$\mathcal{M}^l_{i,j}$	limit memory for pod $\mathcal{V}_{i,j}$
$\mathcal{M}^d_{i,j}$	future demand memory for pod $\mathcal{V}_{i,j}$
$\mathcal{C}^r_{i,j}$	requested CPU for pod $\mathcal{V}_{i,j}$
$\mathcal{C}^l_{i,j}$	limit CPU for pod $\mathcal{V}_{i,j}$
$\mathcal{C}^d_{i,j}$	future demand CPU for pod $\mathcal{V}_{i,j}$
$b_{i,j}$	variable representing the stopped pods, 1 if it is stopped and 0 otherwise
$\mathcal{P}_{i,j}$	penalty for stopping pod $\mathcal{V}_{i,j}$

In this section, we give the problem formulation for the CPU and memory scheduling. The objective is to find the optimal parameters of limits and requests that minimizes the risk of resource shortage. Table 4 summarizes the notation adopted in this section.

We consider a Kubernetes cluster hosting multiple VNF instances. N denotes the set of nodes composing it and N_i refers to the ith node. Each node has a fixed memory and CPU capacities denoted by M_i and Ci respectively. The set of pods deployed on node Ni is denoted by V_i . For each pod j on a node N_i , the values of requests and limits are denoted by $M^r_{i,j}$ and $M^l_{i,j}$respectively for the memory, $C^r_{i,j}$and $C^l_{i,j}$ for the CPU. The resources that will be consumed by each pod vary over time and depend on the VNF that hosts them. Therefore, the future resource demand for each pod is considered as a random variable denoted by $M^d_{i,j}$ and $C^d_{i,j}$for the memory and CPU respectively. The VNF instances must support the traffic load, otherwise they will be stopped, and this can introduce penalties according to their importance. We denote by $P_{i,j}$the penalty introduced when stopping CNF $V_{i,j}$.

The objective of this work is to propose an algorithm that identifies the appropriate parameters of the requests and limits for each CNF, to reduce the expected penalty in a given period, where the profile of the future resource demand is known. This information is presented in the form of the probabilistic distribution of random variables M^d and C^d which can be obtained from the monitoring system. When the demanded resources exceed the capacity of the node, the orchestrator selects the pod to stop. Variable $b_{i,j}$ represents the state of the CNF. The selected ones should respect the condition described by:

$$\mathcal{M}^d_{i,j} - \mathcal{M}^r_{i,j} \leq \mathcal{M}^d_{i,j'} - \mathcal{M}^r_{i,j'}.$$

for i = 1, 2, . . . , N and for all j, j' such that $b_{i,j}$= 1 and $b_{i,j'}$ $= 0$. The objective is to find the appropriate parameters M^r , M^l , C^r and C^l that minimize the expected penalties:

$$
\begin{aligned}
&\text{minimize } \mathbb{E}\Big(\sum_{i=1}^{V}\sum_{j=1}^{V_i} b_{i,j}\mathcal{P}_{i,j}\Big)\\
&\text{subject to:}\\
&\sum_{j=1}^{V_i}\max(\mathcal{M}^r_{i,j},\mathcal{M}^d_{i,j})b_{i,j} \le \mathcal{M}_i, \quad i = 1,\dots,N,\\
&\sum_{j=1}^{V_i}\max(\mathcal{C}^r_{i,j},\mathcal{C}^d_{i,j})b_{i,j} \le \mathcal{C}_i, \qquad i = 1,\dots,N.
\end{aligned}
$$

Following a statistical formulation allows to deal with the randomness of the input information. The obtained optimization problem is non-linear due to the pod kubernetes scheduler behavior. Thus, we cannot rely for the resolution on the well-known existing algorithms for linear problems. Thus, we try to design a specific solution based on genetic algorithms detailed in paper named KRS (Kubernetes Resource Scheduler) [56].

6.4.4 *VALIDATION RESULTS*

In this section we evaluate the performances of our KRS module compared to the Kubernetes best effort mode where the resource scheduling parameters are not configured on the CNFs. For this evaluation, we use simulation as follows. At the beginning, for a given cluster with a known number of nodes, we generate randomly the memory and CPU capacity values for these nodes. Then, on each node, we vary the number of deployed CNFs between 10 and 20. For each CNF, the future demand of resource is considered as a random variable (M^d,C^d) following a Poisson distribution. The parameters of this distribution are randomly generated. The penalty for stopping the CNF is randomly generated and takes normalized values between 0 and 1. Then, the KRS algorithm is performed to find the correct CNFs configuration.

The outcome of this algorithm will be the α distribution, representing the unknown variables, which often converges to an exact value. Finally, the penalty expectation for the obtained configuration is calculated and compared to the one given by the Kubernetes best effort mode.

Figure 49 illustrates the performance of the two approaches, namely the KRS module and the best effort mode. The number of nodes in the cluster varies between 2 and 10. For each configuration we make 50 tests with different input data and then make the average of the results (the penalty and the computing time). The tests show that our solution always performs better than best effort mode. For example, for tests done with 6 nodes in the cluster, the penalty expectation for the best effort mode is more than 6 while it is less than 3 with the KRS module.

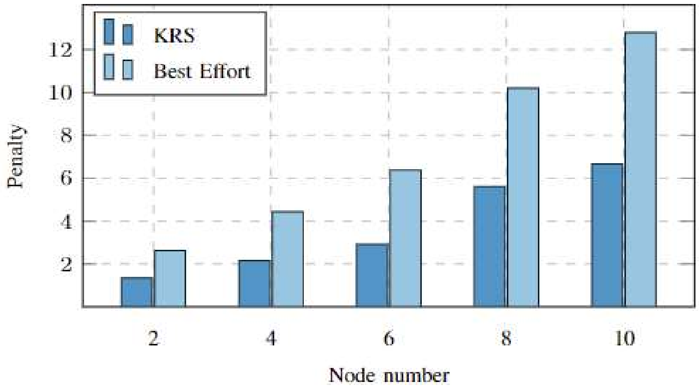


Figure 49: Comparing the accumulated penalty for KRS and the best effort Kubernetes mode for different node number in the cluster

This result is expected since the best effort mode does not differentiate between the deployed functions, and they are treated equally in the resource allocation. Concerning the computing complexity, Figure 50 shows the required time to compute these parameters. We can see that it evolves linearly with the number of nodes. This is because the algorithm processes each node independently of the others, and it can lead to miss the optimal decisions globally, but it ensures the horizontal scalability of the solution. Meanwhile, regarding the vertical scalability of the algorithm, i.e., when we have many pods on the same node, the computation time increases significantly. This remains a point to improve in our approach.

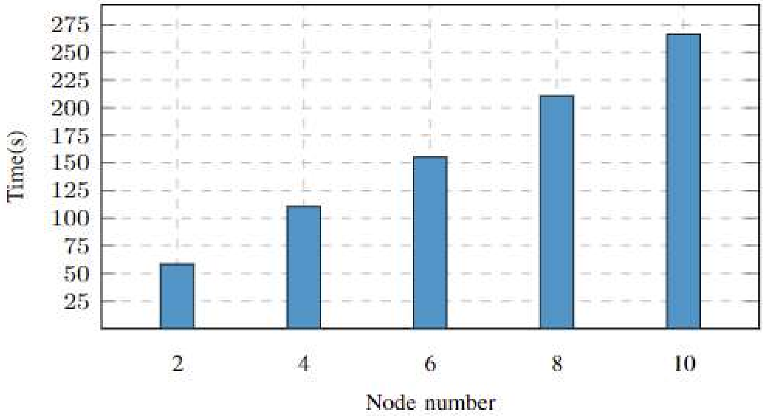


Figure 50: Comparing the computing time for KRS and the best effort Kubernetes mode for different node number in the cluster

6.4.5 MONB5G RELATED CHECKBOXES/KPIS

✓ Utilize key inputs (capacity, load of links, congestion level of local and alternative computing resources, history thereof, and KPI predictors from the distributed AEs.

✓ Quantify the confidence in the ability of the locally optimized/reconfigured slice to resolve the potential problem.

7 Control loops coordination

7.1 Introduction to control loops

The control loop-based optimisation is a popular accepted solution to automate the network, slice or service management. The management is typically very complex and must deal with multiple goals. It has been widely accepted that management systems implement FCAPS services; however, each of the services can be composed of numerous mechanisms that must be used to achieve the management service goal. Due to complexity and lack of programmability (modification/update of management services), the use of a single, multi-objective optimisation function has been rejected. The use of AI in such a case is also not feasible because the state-action space is becoming too big to converge (learn) in an acceptable time.

In practice, therefore, a single objective optimisation is in use, but this leads to implementing numerous Control Loop (CL)-based management functions. Various functions, each trying to optimise a single goal, can lead to suboptimal results or even chaotic system behaviour. Such conflicts can be of different natures. An exhaustive list of conflict types is provided [57]. Among them, there are mentioned the following conflicts:

- conflicts related to the modification of the same system parameter(s) by two or more CL-based functions (see conflict concerning parameter P2 in Figure 51a);
- an indirect impact of a change of a parameter, which is not used by a specific function, on the function output (KPI). In Figure 51a), Function 3 is the only controller of parameters P4 and P5, but their value change impacts the environment and indirectly impacts KPIs controlled by other functions. This is a so-called logical dependency conflict.

The management community has been trying to solve the problem of cooperation/coordination of multiple functions for about a decade. It has been researched, mostly in the case of SON [58], [59], [60], [61], [62]; however, the problem is generic. An exhaustive overview of the proposed solutions can be found in [57]. So far, no satisfactory solution has been found. One practical, simple approach is making time-scale separation of functions operations [68], defining priorities or defining some policy rules. There are also approaches based on machine learning [63], [64], [65], [66], [67] and game-theoretical approach [71]. Most of them are not agnostic to coordinated functions [69]. It must be noted that the coordination problem can be solved by hierarchical reinforcement learning techniques that have been extensively reviewed in [72] and [73].

The MonB5G project assumes the existence of multiple CL-based functions, and their coordination is a must. The MonB5G architecture includes components related to coordinating several AI-driven functions (see Deliverable D2.4 of the MonB5G project). In contrast to the existing approaches, MonB5G assumes that CL-based functions can be dynamically added or removed. Such an approach calls for flexible and CL-function agnostic coordination.

In this section, we will first describe a Coordination Framework compliant with the MonB5G architecture. Later, we will show how it can be used in an implementation where the coordinator treats the CL-based management functions as black boxes. The approach performs the role of a recommender that uses predictions to avoid or minimise conflicts related to network or slice configuration parameters.

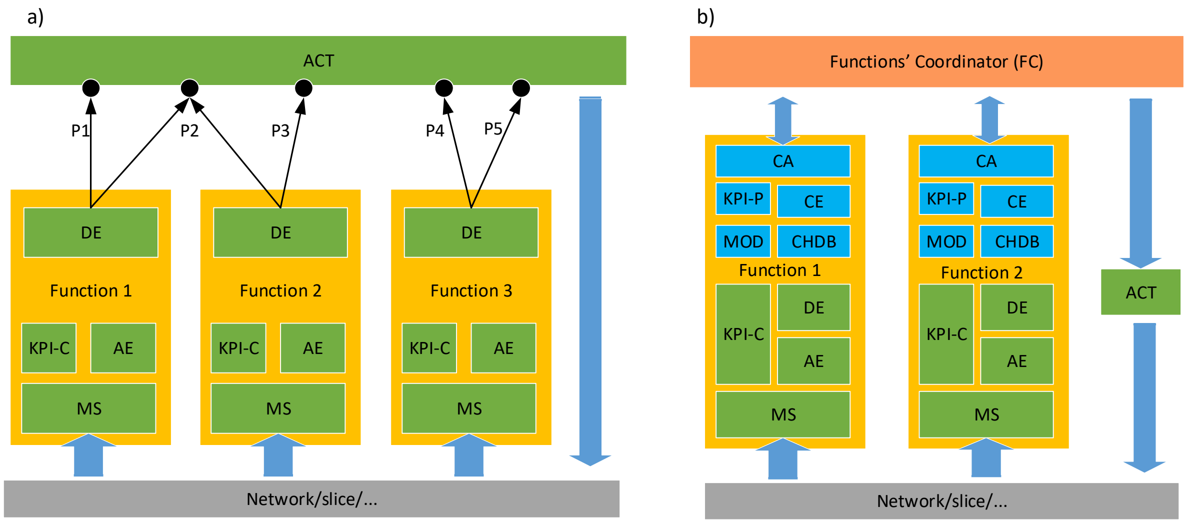


Figure 51. Example of conflicts of DE-driven functions (a) and the proposed approach to Functions coordination (b)

In Figure 51b), the proposed Coordination Framework (CF) is depicted. The framework is partly distributed - it consists of coordination-related components added to each function, which is typically composed of MS/AE/DE and KPI-C (KPI Calculator). The MS/AE and DE components were described in deliverable D2.4 of the MonB5G project. The KPI-C calculates the KPI of every Function. The goal of each Function is to keep the KPI above the level (threshold) defined by the operator. For example, fault management doesn't have such an entity, but in terms of coordination, we are focused on the network (slice) optimisation problems. The roles of Function components (except the already mentioned ones) are the following:

- *Configuration History Database (CHDB)* keeps records of historical system parameters and Function-specific KPI. CHDB can be used to interpolate KPI values or restore the system to a stable configuration. CHDB can also be used to evaluate the impact of each parameter on KPI.
- *Coordinator Agent (CA)* of each function, responsible for the dialogue with Functions' Coordinator (FC).
- *Configuration Engine (CE)* is the entity that manages all Function's entities involved in the coordination procedure. It is used for setup operator's preferences, thresholds, etc.
- *Model (MOD)* is a model of the environment that can be used to estimate the system's output for a new set of parameters proposed by FC or CE.
- *KPI Predictor (KPI-P)* is a predictor of KPI for a set of parameters that can be generated by the CE. KPI-P may use CHDB to estimate KPI; it may also use MS/AE/DE stack with artificial input generated by MOD or FC. In case of minor changes in the existing reconfiguration parameters, the CHDB can be used to estimate the KPI change.
- *Functions Coordinator (FC)* is an entity that approves, modifies, or rejects reconfiguration requests. It interacts with all Functions and collects information about their present KPIs. It may send requests

to all Functions asking them for estimation of KPI for a specific parameter set. Finally, based on the obtained information, it sends a compromised set of parameters to ACT for execution.

A generic procedure of messages exchange of CF is shown in Figure 52. The configuration change Requester is a management Function that proposes a new parameter set. In response to this request, the FC asks all Functions to estimate their KPIs for the new configuration. After collecting all responses, the FC checks if there are KPIs that are below an acceptable level or not. Based on the responses, the FC may accept the reconfiguration, reject it or propose a new configuration set, which is sent to ACT for execution. All Functions record the new values within their CHDB.

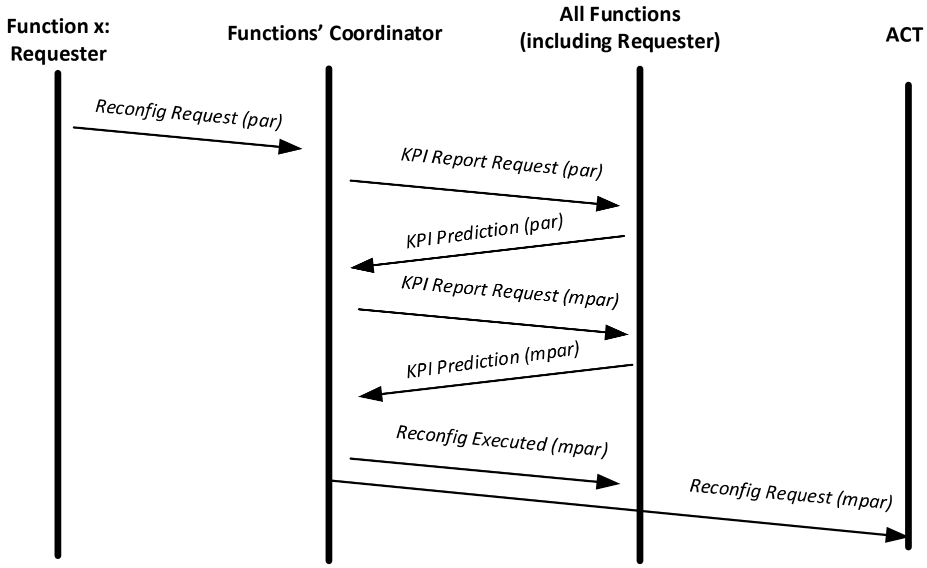


Figure 52. A generic message exchange chart between FC and CAs

The proposed solution allows for direct parameter conflict avoidance; however, due to overall complexity, the future behaviour of the network (or slice) can be different than assumed. As the solution is generally agnostic to how CL-based operations are performed, coordination-related entities can be implemented differently, especially KPI-P, MOD and CE. The presented concept intentionally does not specify the FC algorithm that governs the coordination. It is assumed that many different algorithms for that purpose can be used. A usage example of the proposed framework is described in the following subsection.

7.2 Example instantiation of the Coordination Framework

In this section, an example of the instantiation of the Coordination Framework will be described. In the proposed heuristic approach, the impact of a new reconfiguration proposed by one Function on KPIs of other Functions is estimated. If the overall impact on other KPIs is negative (according to the evaluation of the algorithm), the proposed reconfiguration is rejected; in the opposite case, it is accepted. In future work, the usage of the approach for the modification of the change will be analysed. The concept, called HFMM (Heuristic Fisher Market Model) is an instantiation of the presented in the previous subsection framework.

The core operations of HFMM are based on the Fisher Marked Model [74]. The model is used to estimate the impact of the change of any system parameter on all Functions' KPI. Before we present the overall approach, a short description of the Fisher Market Model (FMM) will be provided in the following subsection. The idea has been already used for management conflict resolution [70], but in the mentioned case, only one system configuration parameter was considered.

7.3 Fisher Market Model

In FMM [74], there are buyers and products. Each of the buyers has its budget. The game's goal is to find the price for all the products that will lead to market clearance while providing the highest possible satisfaction to buyers. The satisfaction for each buyer depends on the amount of the product it buys and its (predefined) value to the buyer. As a result, the number of products purchased by each buyer and the price of all products is known.

A Fisher market consists of a set *N = 1, ..., n* of buyers (agents) and a set *M = 1, ..., m* of divisible products. Each product *j* has price $c_j > 0$. Every buyer *i* has:

- specified budget $B_i > 0$, which can be used by the buyer to acquire products;
- utility function $u_{i,}$ which defines the value of a bundle of products, represented by a vector $x = x_1, ...,$ x_m, where is x_j is the quantity of the product *j*, purchased by the buyer. The value of product j for buyer *i* is defined by the w_{ij} parameter of the utility function.

In the model, it is often assumed that the supply of each product is one unit, and the total budget of all buyers is normalised to one. The utility functions typically used by FMM belong to the *Constant Elasticity of Substitution (CES)* utility functions family that takes the generic form of

$$u_i(x_i) = \left(\sum_{j=1}^{m} w_{ij}\, x_{ij}^{\rho} \right)^{1/\rho}$$

where ρ parametrises the functions' family, and $-\infty < \rho \leq 1,\ \rho \neq 0$.
The three CES functions are typically used in FMM:

- *Leontief* (for ρ approaching $-\infty$)

$$u_i(x_i) = min_{j \in [m]} \left\{ \frac{x_{ij}}{w_{ij}} \right\}$$

- *Cobb-Douglas* (for ρ approaching 0)

$$u_i(x_i) = \prod_{j=1}^{m} x_{ij}^{w_{ij}}$$

- *Linear* (for ρ =1)

$$u_i(x_{i)} = \sum_{j=1}^{m} w_{ij}\, x_{ij}$$

The Leontief function captures the utility of complementary products; the Cobb-Douglas function expresses a balance between complements and substitutes. Finally, the Linear function captures the utility of products that are substitutes.

The FMM market equilibrium maximises the utility of each buyer, considering its budget constraints, and clears the market (all items are entirely sold but not oversold). Formally, FMM market equilibrium is achieved if and only if:

- for all $i \in N$, x_i maximizes buyer i utility for given prices $P$and budget B_i;
- each product j either is completely sold or has a price 0, i.e.

$$\left(\sum_{i=1}^{n} x_{ij} - 1\right) c_j = 0, \qquad \forall j \in [m]$$

- all budgets get spent, i.e.

$$\sum_{j=1}^{m} x_{ij}\, c_{ij} = B_i, \qquad \forall i \in [n]$$

Market equilibrium is guaranteed to exist if at least one buyer desires each item and each buyer desires at least one item [75]. For buyers with utility functions from the same class of the CES family (i.e., for the same ρ), the equilibrium allocation can be found by the Eisenberg-Gale formula, and it maximizes

$$max\left(\prod_{i=1}^{n} (u_i)^{B_i}\right)^{1/B}$$

where B is the overall budget. This maximization is equivalent to the maximization of

$$\sum_{i=1}^{n} B_i\, log(u_i)$$

It has been proved that for the Linear utility function if for each product j some buyer i has $w_{ij} \geq 0$, then the prices clear the market - a budget of each buyer is spent, and each item is completely distributed among buyers. There are several algorithms proposed to solve the FMM problem [76], [79], [78], including online ones [80], [81], [82].

7.4 The HFMM algorithm

The HFMM approach uses FMM to evaluate the impact of the proposed reconfiguration on the KPI of Functions; more precisely, it estimates the impact of the change of each configuration parameter on the KPI change. The HFMM algorithm uses the following data (please note some notations change in comparison to the previous subsection):

- historical vectors of the network (slice, etc.) parameters nc^t (at time t) with corresponding KPI of each Function F_i, stored in CHDB;

- the actual configuration vector denoted as $c^{a,}$ and the proposed configuration vector pointed as c^p;
- each Function F_i based on the analysis of data stored in CHDB, finds the configuration vector c^B that has provided the highest KPI of the function;
- allocated by the operator to each Function budget b_i that indicates importance (priority) for the operator of the KPI elaborated by the Function.

The steps of the HFMM algorithm are the following:

1. The absolute difference between actual (nc^a) and the proposed configuration (nc^p) is calculated, and a vector p, that represents the FMM amount of goods available on the market is obtained

$$p = |nc^p - nc^a|$$

   As the goods on the market must have a non-negative value, the absolute value of the difference is taken. At this step, we added one additional item to the market. This product has no impact on the system's behavior. Its purpose is to allow each buyer (Function) to spend resources in case the other items already present on the market have no value to a Function, i.e., none of these parameters has an impact on KPIs or p_j . A product on the Fisher market can't have a zero price, so a small positive amount is added in such a case in the simulations.

2. The nc^p, nc^a values, and other values of nc^t, where *t* is the time mark, are used to estimate the sensitivity of the KPI of every Function (F_i) on the change of parameter nc_j . Such estimation can *use ΔKPI /Δ*nc_j. The obtained for Function F_i value, for parameter p_j *is* weight w_{ij} of the utility function of Function F_i.
3. The input data to FMM are prepared: budgets, b_i, products p_j and weights w_{ij} of the utility function, so we may try to find the market equilibrium, i.e., prices of each item and their allocation to Functions. This can be done using the primal-dual algorithm (equivalent to the Eisenberg-Gale prime) approach.
4. The interpretation of the values obtained in the previous step. Such analysis can use sophisticated (learned) algorithms; however, some direct interpretations of the results are possible:
   a. A relatively high value of the price of the last 'product', i.e., additionally added parameter, is an indicator of the limited impact of the reconfiguration on other Functions.
   b. A price of an input parameter shows its overall impact on reconfiguration. Please note that the budget of each Function doesn't have to be equal (reflects operator interest in the result of the optimization made by a specific Function), which has an impact on the price of parameters.
   c. The value of the utility function of each Function, calculated without the last, artificially added parameter, is directly related to the reconfiguration's impact on this Function. If, for any Function, this value is higher than in the case when the Function that has triggered reconfiguration, it means that probably such Function's KPI will be more affected than the KPI of the Function that has triggered the reconfigurationand such operation should be avoided.
   d. The FMM-based analysis of the previous step has ignored the sign of the change in configuration parameters, showing only the impact of the absolute difference of the parameter. However, the change (even high) of a parameter can be positive if the proposed reconfiguration is making the nc^p closer to nc^b that was nc^0, where nc^b is a vector for which function F_i has provided the highest KPI.

7.5 HFMM BEHAVIOUR EXAMPLE

It is difficult to demonstrate the proposed concept's behaviour in a simulated environment, as it requires the implementation of several Functions and scenarios in which the coordination actions should be happening. Nevertheless, it is possible to perform HFMM demonstration using a specially prepared dataset, which is the approach we have taken. In our simulations, we have assumed that the system has four functions *F_i*, and the configuration parameter set has four elements plus one that has no real impact on the system configuration, but it is used for spending of Function's budget in the case when the reconfiguration has limited impact on the function. We have demonstrated the behavior of the HFMM, defining two scenarios.

Scenario 1

The input parameters to the HFMM procedure in this scenario are the following:

- budget of each Function reflecting the importance (priority) of the Function to the operator (element of the management policy)

$$B^{S1} = [200 \quad 100 \quad 50 \quad 300]$$

- input vector that reflects the number of products in the market. This vector consists of the absolute values of changes of each parameter of the configuration set regarding the previous configuration. In Scenario 1, this vector is defined in the following way

$$p^{S1} = [10 \quad 1 \quad 0.1 \quad 0.1 \quad 5]$$

- matrix, each row represents the Linear utility function of the market products. The matrix has been defined in the following way

$$W^{S1} = \begin{bmatrix} 5 & 1 & 1 & 4 & 1 \\ 1 & 4 & 0 & 3 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 1 & 0 & 5 & 5 & 5 \end{bmatrix}$$

The FMM solver has provided the following product allocation matrix *A* and product prices *C* for the above-described parameters

$$A^{S1} = \begin{bmatrix} 10 & 0 & 0 & 0.005 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0.1 & 0.095 & 0 \end{bmatrix}$$

$$C^{S1} = [192.308 \quad 100 \quad 153.846 \quad 153.846 \quad 50]$$

The utility functions values for each Function are the following

$$U^{S1} = [50.02 \quad 4 \quad 0 \quad 0.975]$$

The matrix *A* shows an interesting feature – for each Function, important is a change of a single parameter only; moreover, this is a different parameter for each function. The parameter is a_{11}, dominant in the matrix and the most important for Function F_1. Function F_2 has noticed an impact of p_2. The reconfiguration has no real impact on F_3 as it decided to 'buy' only p_5, a parameter that has no physical meaning. For F_4, the only vital parameters are p_3 and p_4; however, their overall value is limited. The prices of each parameter (vector C^{S1}) express the overall impact of each parameter on all Functions. The values are dependent on the budget allocated to functions, and it is difficult to interpret them directly. There is no significant difference between the obtained values except the value for C^{S1}. The last parameter is a measure of the lack of impact of reconfiguration on other functions (in comparison to other values of the vector). In Scenario 1, function F_3 spent all its budget on the parameter p_5. The analysis of the utility values of Functions shows that the most impacted is F_1 (what is expected, as it has triggered the reconfiguration), slightly impacted are functions F_2 and F_4, and F_3 is not impacted at all. Please note the form of Function budget. In conclusion, the reconfiguration brings no significant change in KPIs controlled by other Functions.

Scenario 2

In the case of Scenario 2, the input parameters have been changed compared to Scenario 1 to demonstrate the negative impact of reconfiguration on other functions. The function F_1, like in the previous scenario, triggers the reconfiguration. The new input values are the following:

- the budget of each Function is now equal to

$$B^{S2} = [100 \quad 300 \quad 50 \quad 200]$$

  please note that the budget of F_2 is much higher than the budget of F_1;

- in Scenario 2, this vector, reflecting the change of configuration parameters, has the following values

$$p^{S2} = [10 \quad 1 \quad 0.4 \quad 0.4 \quad 5]$$

- the matrix *W* of which row *j* represents the value of the product on the market for F_i has been defined in the following way

$$W^{S2} = \begin{bmatrix} 5 & 0 & 1 & 3 & 1 \\ 5 & 4 & 2 & 3 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 1 & 0 & 5 & 5 & 1 \end{bmatrix}$$

In this Scenario, the FMM solver has generated the following products allocation matrix A^{S2} and products prices C^{S2}

$$A^{S2} = \begin{bmatrix} 4.033 & 0 & 0 & 0.064 & 0 \\ 5.967 & 1 & 0 & 0.069 & 0 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0.4 & 0.267 & 0 \end{bmatrix}$$

$$C^{S2} = [200 \quad 160 \quad 120 \quad 120 \quad 50]$$

The utility functions values for each Function in this scenario are the following

$$U^{S2} = [20.3592 \quad 34.048 \quad 0 \quad 3.3333]$$

The most significant difference between the scenarios is that in Scenario 1, the highest value of the utility function has Function F_1 and in Scenario 2 the highest value has Function F_2. This means that the impact of the reconfiguration on Function F_2 will be much higher than the impact on F_1 - a function that has triggered the reconfiguration. As the F_1 goal is different, such reconfiguration should be avoided. Similarly, to the previous Scenario, the reconfiguration has no impact on F_3, p_1 impacts strongly F_1 and F_2, and p_3 impacts all functions except F_3.

7.6 Final remarks concerning the Coordination Framework

In this section, we have presented the architectural concept of the Coordination Framework, which is a necessary element of autonomic or cognitive (i.e., AI-driven) management architecture. In a simple case, we have demonstrated the usage of the framework. The proposed instantiation is a recommender that tries to predict the reconfiguration's impact on all the CL-driven management functions according to their value to the operator. The proposed approach is agnostic to CL-functions and allows their addition or removal. However, it has some limitations because it assumes that configuration changes are linked with an optimization that uses small reconfiguration steps. Such an assumption made it possible to use the linear dependency of the configuration parameters on KPI, i.e., to define the *W* matrix. The matrix can be found in a much more accurate way with realistic, advanced simulators and machine learning techniques. Moreover, we have provided a straightforward interpretation of the obtained results, but also, in this case, the use of machine learning could help in finding all necessary dependencies in a much more accurate way. Unfortunately, when writing this section, there was no available mobile network simulator that implemented several CL-based management functions.

8 Conclusions

In this deliverable, we have presented the final version of the distributed AI-driven DE algorithms devised in WP4. We have covered three key management functions of the life-cycle process of a network slice: admission control, intra and inter-slice orchestration. We also added a new topic related to distributed decisions: conflict resolution of DE decisions. This deliverable complements D4.1 by either improving the initial version of the introduced solutions or adding new solutions. We have improved the solutions of D4.1 mainly by extending them to cover not only one single domain but also cross-domain and to enhance their scalability.

We started the deliverable with an elaboration of the proposed (distributed) Decision Engine (Chapter 3), focusing on intra-DE interfaces and the interfaces with the Monitoring System (MS) and AE as well as other DEs. Chapters 4, 5, and 6 presented an example of complete algorithms for Tasks T4.1, T4.2, and T4.3, respectively.

In chapter 4, we presented two schemes for slice admission control: (i) the evolution of a multi-domain data-driven scheme, originally introduced in D4.1; (ii) a new admission control scheme that also introduces a type of "calendaring" and attempts to exploit the periodicity of human activity (and associated traffic).

In chapter 5, we introduced two solutions for intra-slice orchestration. We first described the evolution of our previous solution, SCHEMA, focusing on the various improvements that have been made. We then introduced an important extension of this algorithm, SafeSCHEMA, which integrated the SafeRL framework to operate in scenarios where the natural tendency of RL algorithms to explore "any" possible (slice) configuration might be forbidden (or prohibitively expensive).

In chapter 6, we focused on inter-slice orchestration, where we introduced an improvement to schemes initially presented in D4.1. Indeed, we extended from D4.1: (i) the VNF placement and migration algorithm; (ii) the RAN resource allocation algorithm; (iii) the probing scheme for VNF bottleneck localization.

Finally, in chapter 7, we introduced a new solution that addresses the critical challenge of decision conflicts that happen when parallel decisions need to be enforced. Therefore, we devised a novel coordination framework that predicted reconfiguration's impact and generated recommendations to minimise conflicts related to network or slice configuration parameters.

Even though this deliverable included preliminary results from the different mechanisms, more results, particularly obtained through Proof of Concept (PoC) will be included in D4.3. Further, we use WP6 use-cases ("Zero-Touch multi-domain service management with end-to-end SLAs" and "Elastic end-to-end slice management") to demonstrate a set of the mechanisms devised in WP4 and presented in D4.1 and D4.2 using the CTTC testbed.

References

[1] ETSI GS ZSM 009-1, "Zero-Touch Network and Service Management (ZSM); Closed-loop automation Enablers", March 2021.

[2] S. Kuklinski et al., "Deliverable D2.4: Final release of the MonB5G architecture (including security)", Oct. 2021.

[3] J. J. A. Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Location-based Data Model for Optimized Network Slice Placement", 2020 6th IEEE Conference on Network Softwarization (NetSoft), 2020, pp. 404-412, doi: 10.1109/NetSoft48620.2020.9165427.

[4] J. J. Alves Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Heuristic for Edge-enabled Network Slicing Optimization using the 'Power of Two Choices'", 2020 16th International Conference on Network and Service Management (CNSM), 2020, pp. 1-9, doi: 10.23919/CNSM50824.2020.9269099.

[5] J. J. A. Esteves, A. Boubendir, F. Guillemin, and P. Sens, "A Heuristically Assisted Deep Reinforcement Learning Approach for Network Slice Placement", in IEEE Transactions on Network and Service Management, doi: 10.1109/TNSM.2021.3132103.

[6] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning", International conference on machine learning. PMLR, 2016.

[7] J. J. A. Esteves, A. Boubendir, F. Guillemin, and P. Sens. "On the Robustness of Controlled Deep Reinforcement Learning for Slice Placement", J. Netw. Syst. Manage. 30, 43 (2022). https://doi.org/10.1007/s10922-022-09654-8.

[8] M. O. Ojijo and O. E. Falowo, "A Survey on Slice Admission Control Strategies and Optimisation Schemes in 5G Network," IEEE Access, vol. 8, pp. 14977–14990, 2020, doi: 10.1109/ACCESS.2020.2967626.

[9] Bakri, S., Brik, B., & Ksentini, A. On using reinforcement learning for network slice admission control in 5g: Offline vs. online. International Journal of Communication Systems, 34(7), e4757. 20,21Retrieved from https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.4757(e4757dac.4757) doi: https://doi.org/10.1002/dac.4757

[10] V. Sciancalepore, X. Costa-Perez, and A. Banchs, "RL-NSB: Reinforcement Learning-Based 5G Network Slice Broker," IEEE/ACM Trans. Networking, vol. 27, no. 4, pp. 1543–1557, Aug. 2019, doi: 10.1109/TNET.2019.2924471

[11] H. D. Trinh, N. Bui, J. Widmer, L. Giupponi, and P. Dini, "Analysis and modeling of mobile traffic using real traces," in 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Montreal, QC, Oct. 2017, pp. 1–6. doi: 10.1109/PIMRC.2017.8292200.

[12] S. Wang, X. Zhang, J. Zhang, J. Feng, W. Wang, and K. Xin, "An Approach for Spatial-temporal Traffic Modeling in Mobile Cellular Networks," 2015 27th International Teletraffic Congress, pp. 203–209, Sep. 2015, doi: 10.1109/ITC.2015.31.

[13] Bega, D., Gramaglia, M., Banchs, A., Sciancalepore, V., & Costa-Pérez, X. (2020). A machine learning approach to 5g infrastructure market optimisation. IEEE Transactions on Mobile Computing, 19(3), 498-512. doi: 10.1109/TMC.2019.2896950

[14] Han, B., Feng, D., & Schotten, H. D. (2019). A markov model of slice admission control, IEEE Networking Letters, 1(1), 2-5. doi: 10.1109/LNET.2018.2873978

[15] Jácome, W., Caicedo Rendon, O., & Fonseca, N. Admission control for 5g network slicing based on (deep) reinforcement learning., 2021, doi: 10.36227/techrxiv.14498190

[16] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv. Retrieved from https://arxiv.org/abs/1312.5602 doi: 10.48550/ARXIV.1312.5602

[17] C. Chatfield, The Holt-Winters Forecasting Procedure, Journal of the Royal Statistical Society. Series C (Applied Statistics) Vol. 27, No. 3 (1978), pp. 264-279 (16 pages), Wiley, 1978

[18] Raza, M. R., Natalino, C., Öhlen, P., Wosinska, L., & Monti, P. (2019). Reinforcement learning for slicing in a 5g flexible ran. Journal of Lightwave Technology, 37 (20), 5161-5169. doi: 10.1109/JLT.2019.2924345

[19] Simpy: Discrete event simulation for Python. (n.d.). [Website]. Retrieved from https://simpy.readthedocs.io/en/latest/ (Accessed: Jun. 10, 2022)

[20] GSMA, NG.116 Generic Network Slice Template Version 6.0, 25 November 2021, https://www.gsma.com/newsroom/wp-content/uploads//NG.116-v6.0.pdf

[21] Hyndman, Rob J., and Athanasopoulos, George. Forecasting: principles and practice, 3rd edition, OTexts, 2021. https://otexts.com/fpp3/expsmooth.html

[22] .H. Mao, Z. Gong, and Z. Xiao, "Reward design in cooperative multi-agent reinforcement learning for packet routing," arXiv preprint arXiv:2003.03433, 2020.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "Human-level control through deep reinforcement learning," nature, vol. 518, no. 7540, pp. 529–533, 2015.

[24] S. Feghhi, E. Aumayr, F. Vannella, E. Hakim, and G. Iakovidis, "Safe reinforcement learning for antenna tilt optimisation using shielding and multiple baselines," in 32nd IEEE Annual International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2021, 9 2021, pp. 1148–1153.

[25] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, "The algorithmic aspects of network slicing," IEEE Comms. Magazine, 2017.

[26] F. Schardong, I. Nunes, and A. Schaeffer-Filho, "Nfv resource allocation: a systematic review and taxonomy of vnf forwarding graph embedding," Computer Networks, vol. 185, p. 107726, 2021.

[27] M. Harchol-Balter, Performance Modeling and Design of Computer Systems: Queueing Theory in Action, 1st ed. USA: Cambridge University Press, 2013.

[28] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Deepcog: Cognitive network management in sliced 5g networks with deep learning," in IEEE INFOCOM, 2019.

[29] M. Shojafar, N. Cordeschi, and E. Baccarelli, “Energy-efficient adaptive resource management for real-time vehicular cloud services,” IEEE Trans. on Cloud Computing, 2019.

[30] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, “Aztec: Anticipatory capacity allocation for zero-touch network slicing,” in IEEE INFOCOM, 2020.

[31] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. The MIT Press, 2018.

[32] D. Bertsekas, Reinforcement Learning and Optimal Control. Athena Scientific, 2019.

[33] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[34] Tan, Ming. "Multi-agent reinforcement learning: Independent vs. cooperative agents." Proceedings of the tenth international conference on machine learning. 1993.

[35] Tampuu, Ardi, et al. "Multiagent cooperation and competition with deep reinforcement learning." PloS one 12.4 (2017): e0172395.

[36] Sciancalepore, V., Costa-Perez, X., & Banchs, A. (2019). RL-NSB: Reinforcement learning-based 5G network slice broker. IEEE/ACM Transactions on Networking, 27(4), 1543-1557.

[37] Foukas, X., Marina, M. K., & Kontovasilis, K. (2017, October). Orion: RAN slicing for a flexible and cost-effective multi-service mobile network architecture. In Proceedings of the 23rd annual international conference on mobile computing and networking (pp. 127-140).

[38] Tun, Y. K., Tran, N. H., Ngo, D. T., Pandey, S. R., Han, Z., & Hong, C. S. (2019). Wireless network slicing: Generalized kelly mechanism-based resource allocation. IEEE Journal on Selected Areas in Communications, 37(8), 1794-1807.

[39] Azimi, Y., Yousefi, S., Kalbkhani, H., & Kunz, T. (2021). Energy-Efficient Deep Reinforcement Learning Assisted Resource Allocation for 5G-RAN Slicing. IEEE Transactions on Vehicular Technology.

[40] Zanzi, L., Sciancalepore, V., Garcia-Saavedra, A., Schotten, H. D., & Costa-Pérez, X. (2020). LACO: A latency-driven network slicing orchestration in beyond-5G networks. IEEE Transactions on Wireless Communications, 20(1), 667-682.

[41] Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., & Bengio, Y. (2013). An empirical investigation of catastrophic forgetting in gradient-based neural networks. arXiv preprint arXiv:1312.6211.

[42] Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence (Vol. 30, No. 1).

[43] Pei, J., Hong, P., Pan, M., Liu, J., & Zhou, J. (2019). Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks. IEEE Journal on Selected Areas in Communications, 38(2), 263-278.

[44] Lyu, X., Ren, C., Ni, W., Tian, H., Liu, R. P., & Dutkiewicz, E. (2019). Optimal online data partitioning for geo-distributed machine learning in edge of wireless networks. IEEE Journal on Selected Areas in Communications, 37(10), 2393-2406.

[45] Wang, J., Zhang, J., Bao, W., Zhu, X., Cao, B., & Yu, P. S. (2018, July). Not just privacy: Improving performance of private deep learning in mobile cloud. In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 2407-2416).

[46] Aledhari, M., Razzak, R., Parizi, R. M., & Saeed, F. (2020). Federated learning: A survey on enabling technologies, protocols, and applications. IEEE Access, 8, 140699-140725.

[47] Zhang, C., & Patras, P. (2018, June). Long-term mobile traffic forecasting using deep spatio-temporal neural networks. In Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing (pp. 231-240).

[48] Bega, D., Gramaglia, M., Fiore, M., Banchs, A., & Costa-Perez, X. (2019, April). DeepCog: Cognitive network management in sliced 5G networks with deep learning. In IEEE INFOCOM 2019-IEEE Conference on Computer Communications (pp. 280-288). IEEE.

[49] Wang, W., Lyu, G., Shi, Y., & Liang, X. (2018, November). Time series clustering based on dynamic time warping. In 2018 IEEE 9th international conference on software engineering and service science (ICSESS) (pp. 487-490). IEEE.

[50] Gold, O., & Sharir, M. (2018). Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. ACM Transactions on Algorithms (TALG), 14(4), 1-17.

[51] Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In kdd (Vol. 96, No. 34, pp. 226-231).

[52] Pappalardo, L., & Simini, F. (2018). Data-driven generation of spatio-temporal routines in human mobility. Data Mining and Knowledge Discovery, 32(3), 787-829.

[53] S. Sharma, R. Miller, and A. Francini, "A cloud-native approach to 5g network slicing," IEEE Communications Magazine, vol. 55, no. 8, pp. 120–127, 2017.

[54] T. Taleb, A. Ksentini, and B. Sericola, "On service resilience in cloud-native 5g mobile systems," IEEE Journal on Selected Areas in Communications, vol. 34, no. 3, pp. 483–496, 2016.

[55] P. L. Vo, M. N. H. Nguyen, T. A. Le, and N. H. Tran, "Slicing the edge: Resource allocation for ran network slicing," IEEE Wireless Communications Letters, vol. 7, no. 6, pp. 970–973, 2018.

[56] M. Rahali, C. -T. Phan and G. Rubino, "KRS: Kubernetes Resource Scheduler for resilient NFV networks," 2021 IEEE Global Communications Conference (GLOBECOM), 2021, pp. 1-6, doi: 10.1109/GLOBECOM46510.2021.9685328.

[57] A. Bayazeed, K. Khorzom, M. Aljnidi, A survey of self-coordination in self-organising network, Computer Networks, Vol. 196, 2021, 108222, ISSN 1389-1286.

[58] K. Tsagkaris, N. Koutsouris, P. Demestichas, R. Combes and Z. Altman, "SON Coordination in a Unified Management Framework," 2013 IEEE 77th Vehicular Technology Conference (VTC Spring), 2013, pp. 1-5, doi: 10.1109/VTCSpring.2013.6692759

[59] T. Bandh, R. Romeikat, H. Sanneck and Haitao Tang, "Policy-based coordination and management of SON functions," 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops, 2011, pp. 827-840, doi: 10.1109/INM.2011.5990492.

[60] L. C. Schmelz, M. Amirijoo, A. Eisenblaetter, R. Litjens, M. Neuland and J. Turk, "A coordination framework for self-organisation in LTE networks," 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops, 2011, pp. 193-200, doi: 10.1109/INM.2011.5990691.

[61] R. Combes, Z. Altman and E. Altman, "Coordination of autonomic functionalities in communications networks," 2013 11th International Symposium and Workshops on Modeling and Optimisation in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2013, pp. 364-371.

[62] A. Zakrzewska, L. Ho, H. Gacanin and H. Claussen, "Coordination of SON Functions in Multi-Vendor Femtocell Networks," in IEEE Communications Magazine, vol. 55, no. 7, pp. 165-171, July 2017, doi: 10.1109/MCOM.2017.1600530

[63] O. Iacoboaiea, B. Sayrac, S. Ben Jemaa and P. Bianchi, "SON Coordination for parameter conflict resolution: A reinforcement learning framework," 2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), 2014, pp. 196-201, doi: 10.1109/WCNCW.2014.6934885

[64] O. Iacoboaiea, B. Sayrac, S. Ben Jemaa and P. Bianchi, "Low complexity SON coordination using reinforcement learning," 2014 IEEE Global Communications Conference, 2014, pp. 4406-4411, doi: 10.1109/GLOCOM.2014.7037501.

[65] O. Iacoboaiea, B. Sayrac, S. Ben Jemaa and P. Bianchi, "SON Coordination in Heterogeneous Networks: A Reinforcement Learning Framework," in IEEE Transactions on Wireless Communications, vol. 15, no. 9, pp. 5835-5847, Sept. 2016, doi: 10.1109/TWC.2016.2571695.

[66] J. Moysen, M. Garcia-Lozano, L. Giupponi and S. Ruiz, "Conflict Resolution in Mobile Networks: A Self-Coordination Framework Based on Non-Dominated Solutions and Machine Learning for Data Analytics [Application Notes]," in IEEE Computational Intelligence Magazine, vol. 13, no. 2, pp. 52-64, May 2018, doi: 10.1109/MCI.2018.2807038.

[67] D. Preciado, M. Kasparick, R. L. G. Cavalcante and S. Stanczak, "SON Function Coordination in Campus Networks Using Machine Learning," 2022 IEEE Wireless Communications and Networking Conference (WCNC), 2022, pp. 2130-2135, doi: 10.1109/WCNC51071.2022.9771586

[68] M. Qin et al., "Learning-Aided Multiple Time-Scale SON Function Coordination in Ultra-Dense Small-Cell Networks," in IEEE Transactions on Wireless Communications, vol. 18, no. 4, pp. 2080-2092, April 2019, doi: 10.1109/TWC.2019.2898002

[69] J. Moysen and L. Giupponi, "Self Coordination among SON Functions in LTE Heterogeneous Networks," 2015 IEEE 81st Vehicular Technology Conference (VTC Spring), 2015, pp. 1-6, doi: 10.1109/VTCSpring.2015.7146076.

[70] A. Banerjee, S. S. Mwanje and G. Carle, "Optimal configuration determination in Cognitive Autonomous Networks," 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2021, pp. 494-500.

[71] A. Banerjee, S. S. Mwanje and G. Carle, "Game theoretic Conflict Resolution Mechanism for Cognitive Autonomous Networks," 2020 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2020, pp. 1-8.

[72] Hutsebaut-Buysse M, Mets K, Latré S. Hierarchical Reinforcement Learning: A Survey and Open Research Challenges. Machine Learning and Knowledge Extraction. 2022; 4(1):172-221. https://doi.org/10.3390/make4010009

[73] Shubham Pateria, Budhitama Subagdja, Ahhwee Tan, and Chai Quek. 2021. Hierarchical Reinforcement Learning: A Comprehensive Survey. ACM Comput. Surv. 54, 5, Article 109 (June 2022), 35 pages. https://doi.org/10.1145/3453160

[74] S. Brânzei, C. Yiling, X. Deng, A. Filos-Ratsikas, K. S. Frederiksen and J. Zhang. "The Fisher Market Game: Equilibrium and Welfare." AAAI (2014).

[75] R. R. Maxfield, General equilibrium and the theory of directed graphs, Journal of Mathematical Economics, Volume 27, Issue 1, 1997, Pages 23-51, ISSN 0304-4068.

[76] J. B. Orlin. Improved algorithms for computing Fisher's market clearing prices: computing fisher's market clearing prices. In Proceedings of the 42 ACM symposium on Theory of computing (STOC '10). ACM, New York, NY, USA, 2010.

[77] B. Birnbaum, N. R Devanur, and L. Xiao. Distributed algorithms via gradient descent for fisher markets. In Proceedings of the 12th ACM conference on Electronic commerce, pages 127-136. ACM, 2011.

[78] C. Kroer, Alexander Peysakhovich, Eric Sodomka, and Nicolas E Stier-Moses. Computing large market equilibria using abstractions. In Proceedings of the 2019 ACM Conference on Economics and Computation, pages 745-746, 2019.

[79] A. S. Prasad, M. Arumaithurai, D. Koll, Y. Jiang and X. Fu, "OFM: An Online Fisher Market for Cloud Computing," IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 2575-2583, doi: 10.1109/INFOCOM.2019.8737641.

[80] S. Angelopoulos, A. D. Sarma, A. Magen, and A. Viglas, "On-Line Algorithms for Market Equilibria," in 11th Annual International Conference Computing and Combinatorics (COCOON). Springer Berlin Heidelberg, Aug. 2005, pp. 596–607.

[81] A. Blum, T. Sandholm, and M. Zinkevich, "Online Algorithms for Market Clearing," Journal of the ACM, vol. 53, no. 5, pp. 845–879, Sep. 2006.

[82] Y. Azar, N. Buchbinder, and K. Jain, "How to Allocate Goods in an Online Market?" Algorithmica, vol. 74, no. 2, pp. 589–601, Feb. 2016.