OREO: O-RAN intElligence Orchestration of xApp-based network services

OREO: O-RAN intElligence Orchestration of xApp-based network services

F. Mungari[1,2], C. Puligheddu[1], A. Garcia-Saavedra[3], C. F. Chiasserini[1,2,4]
1: Politecnico di Torino, Italy – 2: CNIT, Italy – 3: NEC Labs Europe, Germany – 4: CNR-IEIIT, Italy

***Abstract*—The Open Radio Access Network (O-RAN) architecture aims to support a plethora of network services, such as beam management and network slicing, through the use of third-party applications called xApps. To efficiently provide network services at the radio interface, it is thus essential that the deployment of the xApps is carefully orchestrated. In this paper, we introduce OREO, an O-RAN xApp orchestrator, designed to maximize the number of offered services. OREO's key idea is that services can share xApps whenever they correspond to semantically equivalent functions, and the xApp output is of sufficient quality to fulfill the service requirements. By leveraging a multi-layer graph model that captures all the system components, from services to xApps, OREO implements an algorithmic solution that selects the best service configuration, maximizes the number of shared xApps, and efficiently and dynamically allocates resources to them. Numerical results as well as experimental tests performed using our proof-of-concept implementation, demonstrate that OREO closely matches the optimum, obtained by solving an NP-hard problem. Further, it outperforms the state of the art, deploying up to 35% more services with an average of 28% fewer xApps and a similar consequent reduction in the resource consumption.**

I. Introduction

As mobile networks continue to gain momentum, it has become essential to accommodate a growing number of traffic classes and demanding services [1]. The O-RAN architecture [2], promoted by the O-RAN Alliance, addresses such a need by building upon virtualization and openness principles [3], [4]. A key feature of the O-RAN architecture is network automation, enabled by *xApps* and *rApps*, third-party applications running in Radio Intelligent Controllers (RICs) and operating, respectively, below and above the 1-second timescale. In this context, the integration of Machine Learning (ML) plays a crucial role, since ML-based xApps can serve as the foundation of the RAN intelligence for closed-loop and agile network control [5]. However, as the use of ML takes a substantial toll on the system resources, it is imperative to devise novel orchestration policies that effectively minimize the xApps computational footprint within the O-RAN architecture, especially when resource availability is limited.

Existing research gap. The design of an efficient policy for the orchestration of RAN intelligence in O-RAN platforms is still an open challenge. The state of the art, discussed in details in Sec. VI, either fails to acknowledge the complexity of the RAN intelligence management in O-RAN, or only considers monolithic services and xApps with oversized resource allocations, resulting in sub-optimal decisions. To fill this gap, we propose a novel O-RAN intElligence Orchestration (OREO) framework that, given the Mobile Network Operators' (MNO) demand for network services, identifies *the set of xApps to deploy as well as their operational configuration*, while meeting the specific service requirements and keeping resource expenditure at the minimum.

OREO distinctive features. Our study is the first that integrates the Network Function Virtualization (NFV) paradigm into the O-RAN architecture, recognizing that RAN services, such as beam allocation and handover prediction, can be built by interconnecting elementary RAN functions, like load forecaster and traffic classificator. Importantly, our approach allows for sharing common functions among different services and for efficiently tuning the resources allocated to the xApps that contribute to such services. As detailed in Sec. II-A, we also account for the fact that functions with the same semantic can be deployed as xApps at distinct complexity levels, leading to different trade-offs among the function output quality, resource consumption, and execution speed.

Technical challenges. Jointly coping with the orchestration of RAN services and their associated resources gives the opportunity to envision an enhanced orchestration policy. However, to grasp such an opportunity, an O-RAN service orchestrator has to address multiple challenges: (*i*) it has to identify the most convenient service configuration, i.e., the xApps, from those available in the catalog of the near-real-time (near-RT) RIC, that have the right semantic to implement a service matching the desired functional requirements; (*ii*) the output quality, hence the level of complexity, of each xApp must adhere to the service quality requirements; (*iii*) the computational (e.g., CPU, GPU) and memory (e.g., RAM, disk) resources allocated to the xApps must be sufficient for fulfilling the service latency requirements, while not exceeding the available resource budget.

Summary of novel contributions.

• By exploiting the NFV paradigm, we look at O-RAN services not as monolithic, rigid entities, but rather as sets of interconnected elementary functions that can be implemented as O-RAN xApps with distinct levels of complexity (i.e., yielding different levels of output quality). Importantly, this approach dramatically increases the flexibility in instantiating xApps for RAN services.

• By developing a model that captures all relevant aspects, we formulate the (NP-hard) xApp Deployment and Sharing (xDeSh) problem that maximizes the number of offered services while accounting for the service and system constraints.

• We design an intelligent orchestrator – named OREO – and integrate it into the O-RAN architecture. In view of the complexity of the xDeSh problem, the OREO engine implements a heuristic solution to xDeSh that: (i) selects the xApps providing the functions that are semantically necessary for the service deployment, (ii) at the level of complexity that best trades off output quality with resource expenditure. To our knowledge, OREO is the first to leverage the NFV paradigm for an efficient and flexible deployment of O-RAN xApps.

• We evaluate OREO through an extensive numerical analysis and experimental results obtained with our proof-of-concept testbed, integrated in a O-RAN platform and running real-world RAN services. OREO can support a number of services close to the optimum, and, compared to the state of the art, it enables the co-existence of more services (16.2% more on average and up to 35%), while reducing resource expenditure (by 25.6% less on average and up to 31%).

II. The OREO Framework

This section presents the OREO framework, first outlining its purpose in an O-RAN system and describing the distinctive features of its engine (Sec. II-A), and then providing the rationale behind its design and its integration within the O-RAN architecture (Sec. II-B).

A. OREO driving purpose and distinctive features

The success of next-generation mobile networks greatly hinges upon the quality of RAN intelligence and, hence, upon the performance of its orchestration framework, which is responsible for deploying RAN management services [6]. Our orchestrator, OREO, acts upon a set of service requests by the MNOs in an O-RAN platform. Given such requests, OREO selects the xApp(s) required to deploy the services in the near-RT RIC, and the specific xApps configuration that lets a service meet its performance requirements while matching the resource availability in the platform. OREO's orchestration decisions are made by its core component, the OREO engine, which greatly differs from state-of-the-art O-RAN orchestrators such as the pioneering work in [7].

A key differentiating principle that drives the design of the OREO engine consists in conceiving O-RAN services as sets of interconnected functions rather than monolithic entities. This approach capitalizes on the known benefits of the NFV paradigm, such as enhanced flexibility, scalability, and cost-effectiveness [8]. By recognizing that O-RAN services can be built by interconnecting elementary RAN management functions, OREO leverages xApps as fundamental building blocks to efficiently offer such services. More specifically,

- *Service composition:* Each network service request fed by an MNO to the OREO engine is associated with a minimum service quality and a maximum latency requirement. Depending on such performance targets, services can be deployed by using different configurations: each configuration corresponds to a different set of functions, with each function implementing a certain task. As an example, network slicing can be enabled through a single function implementing a reinforcement learning (RL)-based policy, as in [9], or combining such function with a traffic predictor that feeds its output to the RL model for improved system response to changes in the traffic conditions.
- *Implementing service functions through xApps:* Additionally, each function can be implemented through multiple xApps, each executing semantically the same function, but instantiated at a different operating point, hereinafter also referred to as *complexity factor*. Importantly, complexity factors provide different trade-offs between the output quality and processing latency of the function offered by the xApp and the computational resources necessary to run that xApp.

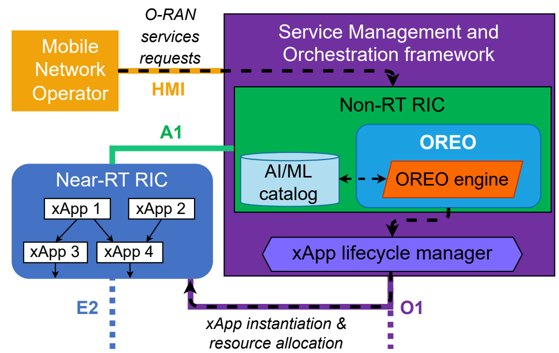

Fig. 1. OREO design and integration in the O-RAN architecture. The workflow (dashed black line) is as follows: (i) the MNO submits service requests via the Human-Machine interface (HMI); (ii) OREO processes such requests and, with the support of the xApp lifecycle manager, instructs the near-RT RIC via the O1 interface about which xApps to deploy.

Thus, based on the above concepts, in OREO the quality and latency incurred by a service depend upon:

- The specific configuration (set of functions) that is selected to enable the service;
- The specific xApps (hence levels of complexity) that are chosen to implement the functions in the selected configuration.

The OREO engine identifies the service configuration *and* the corresponding xApps in such a way that it can best suit the service requirements. Furthermore,

- Whenever multiple services require the same function, *OREO allows such services to share the xApp that implements the semantics of that function*, if the xApp complexity level meets the requirements of the services;
- As the service latency targets can be fulfilled by properly setting the resources allocated to the shared xApps, *OREO scales the resources assigned to an xApp according to the overall load imposed by the corresponding services*, as well as the available resource budget. Importantly, in so doing, OREO avoids resource over-provisioning, as opposed to relying upon a fixed amount of resources allocated to xApps as in state-of-the-art solutions [7].

B. OREO system architecture

The OREO framework, illustrated in Fig. 1, is designed to be integrated into the O-RAN Service Management and

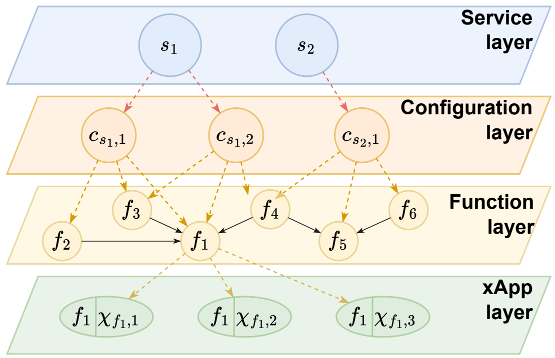

Fig. 2. Graph-based representation of the system under study and relation between its main components. For clarity, the xApp layer only includes the xApps implementing function f_1.

Orchestration (SMO), which is responsible for managing and orchestrating all control and monitoring procedures of the RAN components via the O1 interface. In particular, OREO operates within the non-RT RIC, which supports the execution of third-party applications known as rApps and, through the A1 interface, enables closed-loop control of the RAN.

By accessing the Human-Machine interface (HMI), OREO receives management intents from an MNO, which submits requests for a set of services, specifying the maximum tolerable delay and minimum quality for each service. As mentioned in the previous section, the OREO engine calculates the deployment of xApps satisfying the service requests. xApps are indeed third-party applications that implement customized logic to drive the RAN efficiently and run within the near-RT RIC, i.e., the central control and optimization unit of the RAN operating on a sub-second time scale. Through the E2 interface and open APIs, the near-RT RIC interacts with the RAN centralized and distributed units (O-CUs and O-DUs, respectively), collecting RAN performance metrics and providing control actions.

As detailed in Sec. V, we have implemented all OREO components and integrated them in the O-RAN architecture, leveraging an O-Cloud platform [10] that hosts the SMO, the non-RT RIC, and the near-RT RIC. So doing, we have developed a proof-of-concept testbed used to measure the performance of the proposed solution in real-world settings.

III. xDeSh: xApp Deployment and Sharing

We now introduce a model that captures all relevant system aspects (Sec. III-A) and formulate the xApp Deployment and Sharing (xDeSh) problem (Sec. III-B).

A. System model

Fig. 2 depicts the main system components, which we further detail in the following; the notation used in this section is summarized in Tab. I.

• **Services and service configurations.** We focus on decision-making services managed by the near-RT RIC. Each service s is characterized by: (i) a target latency T_s specifying the maximum acceptable delay to output a decision since a service request arrives; and (ii) a target decision quality Q_s,

TABLE I
Notations

Parameters	
Symbol	**Description**
$s\in\mathcal{S}$	RAN service
T_s (Q_s)	Target latency (quality) of service s under configuration c_s
p_s	Priority of service s
$c_s\in\mathcal{C}_s$	Service s configuration
$\mathcal{V}_{c_s}$	Set of nodes of the service configuration function graph c_s
$f\in\mathcal{F}$	RAN function
$\chi_f\in\mathcal{X}_f$	Complexity factor of function f
f_χ	xApp implementing function f with complexity χ
$f_\chi^{(j)}$	j-th instance of xApp f_χ
$\lambda_{\mathcal{P}(f_\chi^{(j)})}$	Input data rate of $f_\chi^{(j)}$ when shared among $s\in\mathcal{P}(f_\chi^{(j)})$
θ_{f_χ}	Amount of input data processed by f_χ in a CPU cycle
$\mathcal{K}$	Set of resource types
$\mathbf{B}$	Vector of resource budgets of the different types
q_{c_s,f_χ}	Quality of xApp f_χ
$l_{f_\chi^{(j)}}$	Processing latency of the j-th instance of xApp f_χ
τ_{c_s}	Latency of service s

e.g., the minimum required accuracy for a traffic classification task. Further, a service is assigned a priority level p_s, which can be used to determine which services should be dropped in case of insufficient resource availability. A service can be provided using different configurations, $c_s \in \mathcal{C}_s$, i.e., sets of interconnected elementary functions. Each configuration is associated with a level of quality and resource demand, determined by the set of functions appearing in the configuration; thus, properly selecting c_s makes it possible to trade off the performance of a service with its deployment and running cost.

• **Functions and xApps.** A function $f\in\mathcal{F}$ represents a low-level operation and serves as the fundamental building block of one or more services. Examples of functions include traffic forecasting and traffic classification. Functions may process (i) metrics collected by the RAN elements (O-DU, O-CU, etc.) and shared with the near-RT RIC via the E2 interface; and/or (ii) information provided by other functions. A service configuration can then be modeled as a directed graph whose vertices ($\mathcal{V}_{c_s}$) and edges represent, respectively, the functions composing the configuration and the dependency relations between them. Specifically, an edge exists from function f' to function f whenever the execution of f requires the output of f'. Each function can be implemented with a different *complexity factor*, $\chi_f\in\mathcal{X}_f$. For instance, a traffic classification function can be provided by different ML models, each offering a different accuracy-resource demand trade-off.

A specific function with a given complexity factor defines an xApp, which is thus indicated as $f_\chi = (f, \chi_f)$. Let $\mathcal{P}(f_\chi)$ be the set of service configurations that include xApp f_χ, and $\lambda_{\mathcal{P}(f_\chi)}$ be the rate at which data is fed to, and needs to be processed by, the xApp per unit of time. We remark that, multiple instances (i.e., replicas) of a given xApp can be implemented and, hence, coexist in the system; we then denote the j-th instance of f_χ with $f_\chi^{(j)}$.

• **Near-RT RIC resources.** The O-Cloud can provide the near-RT RIC with computing and storage resources (e.g., CPU, GPU, memory) to run xApps. We denote with

$\mathcal{K}$=$\{1, \ldots, K\}$ the set of available resource types, and with $\boldsymbol{B} = [B_1, \ldots, B_K]$ the vector collecting, for each type, the available resource budget. For simplicity and without loss of generality, in the following we focus on CPU, memory, and disk storage. Thus, we let $\boldsymbol{\rho}_{f_\chi^{(j)}}$=$[\rho_{f_\chi^{(j)},1}, \ldots, \rho_{f_\chi^{(j)},K}]$, with $\rho_{f_\chi^{(j)},k}$≤$\boldsymbol{B_k}$ denote the amount of resource of type k required by the xApp instance $f_\chi^{(j)}$.

• **xApp and service quality.** Let q_{c_s,f_χ} be the quality score obtained by the xApp f_χ associated with service configuration c_s. Common quality metrics include prediction and classification accuracy, regression error, and expected reward. The quality score q_{c_s,f_χ} depends on the quality of the input data, which, in turn, depends on the complexity level associated with the function f' preceding f in the configuration graph. Accordingly, the quality metric for a service s implemented under configuration c_s, is equal to the quality of the last xApp's output in the configuration graph.

• **xApp and service latency.** Given the considered resource types, as long as a minimum memory and disk allocation is guaranteed, only the CPU allocation has an impact on the xApps processing latency, denoted by $l_{f_\chi^{(j)}}(\rho_{f_\chi^{(j)},\text{cpu}})$ for the j-th instance of xApp f_χ. Drawing on the existing works [11]–[13], we can model a function, $f_\chi^{(j)}$, that is shared among the service configurations in $\mathcal{P}(f_\chi^{(j)})$, as an M/M/1 queue. We can then write the corresponding average processing latency as:

$$l_{f_\chi^{(j)}}(\rho_{f_\chi^{(j)},\text{cpu}}) = (\rho_{f_\chi^{(j)},\text{cpu}}\theta_{f_\chi^{(j)}} - \lambda_{\mathcal{P}(f_\chi^{(j)})})^{-1}$$

where $\rho_{f_\chi^{(j)},\text{cpu}}$ is expressed as CPU cycles per second, and $\theta_{f_\chi^{(j)}}$ represents the xApp complexity and expresses the amount of input data processed by the xApp in a CPU cycle. Also, let τ_{c_s} be the latency of service s when implemented with configuration c_s. Defining a path π_{c_s} on the graph of configuration c_s as a set of edges connecting an input function with an output function in c_s, τ_s is the latency associated with the most time-consuming path in the graph. The latency of a path clearly depends on the complexity factor and resource allocation of each of the functions composing the path, i.e.,

$$\tau_{c_s}(\{\boldsymbol{\rho}_{f_\chi}, \chi_f\}_{f_\chi \in c_s}) = \underset{\{\pi_{c_s}\}}{\arg\max} \sum_{f \in \pi_{c_s}} l_{f_{j,\chi_{f_j}}}(\boldsymbol{\rho}_{f_j})\,. \tag{1}$$

B. xDeSh problem formulation

Given the above model, we now introduce the xDeSh optimization problem, along with some additional system variables and parameters defining the current state of the system. Further, we prove that the xDeSh problem is NP-hard.

• **Service configuration selection.** Let $\mathcal{S}$ be the set including both the existing, and still to be kept, services and the new services to be deployed. For each service s∈$\mathcal{S}$, the OREO orchestrator identifies the most suitable configuration c_s to be used. We denote with z_{c_s} the binary decision variable taking 1 if configuration c_s is selected for service s. Notice that: (i) it may happen that none of the possible configurations of a service s can be deployed, due to insufficient resources to guarantee the minimum required service performance; (ii) at most one configuration per service can be selected. That is, the following constraint must hold:

$$\sum_{c_s \in \mathcal{C}_s} z_{c_s} \le 1,\, \forall s \in \mathcal{S}\,. \tag{2}$$

• **Selection of xApps to implement and share.** Whenever an xApp f_χ is required by more than one service, the orchestrator has to determine whether to let such services share the same instance $f_\chi^{(j)}$, or to implement multiple instances thereof. We thus introduce the binary decision variable $v_{c_s,f_\chi^{(j)}}$, to indicate whether $f_\chi^{(j)}$ is used by service configuration c_s or not. Clearly, all the functions required by a selected service configuration must be implemented. Moreover, neglecting the possibility of scaling out xApps, a service configuration cannot use more than one instance of a given xApp. The above requirements translate in the following constraint:

$$\sum_{\chi \in \mathcal{X}_f} \sum_j v_{c_s,f_\chi^{(j)}} = z_{c_s},\, \forall s{\in}\mathcal{S}, \forall c_s{\in}\mathcal{C}_s, \forall f{\in}\mathcal{V}_{c_s}\,. \tag{3}$$

Similarly, an xApp implementing function f cannot be associated with a service configuration that does not include f:

$$\sum_{\chi \in \mathcal{X}_f} \sum_j v_{c_s,f_\chi^{(j)}} = 0,\, \forall s{\in}\mathcal{S}, c_s{\in}\mathcal{C}_s, f{\notin}\mathcal{V}_{c_s}\,. \tag{4}$$

• **Meeting service requirements.** OREO has to select a service configuration (i.e., the functions that implement the service) and the corresponding resources in such a way that the quality and latency targets are satisfied. That is, for any s∈$\mathcal{S}$ and c_s∈$\mathcal{C}_s$,

$$q_{c_s}(\{\chi_f \mid \sum_j v_{c_s,f_\chi^{(j)}} = 1\}_{f \in c_s}) \ge Q_s \cdot z_{c_s} \tag{5}$$

$$\tau_{c_s}(\{\boldsymbol{\rho}_{f_\chi^{(j)}}, \chi_f \mid \sum_j v_{c_s,f_{\chi_f}^{(j)}} = 1\}_{f \in c_s}) \cdot z_{c_s} {\le} T_s\,. \tag{6}$$

• **Complying with the resource budget.** We also need conventional capacity constraints, i.e., the near-RT RIC resource budget $\boldsymbol{B}$ must not be exceeded:

$$\sum_{f \in \mathcal{F}} \sum_{\chi \in \mathcal{X}} \sum_j \rho_{f_\chi^{(j)},k} \le B_k,\, \forall k{\in}\mathcal{K}\,. \tag{7}$$

• **Near-RT RIC's settings.** Let $\hat{\mathcal{S}} = \{\hat{s}\}$ denote the set of services that are already deployed, and $c_{\hat{s}}$ capture their service configuration. Accordingly, the binary parameter $\hat{z}_{c_{\hat{s}}}$ takes 1 if configuration $c_{\hat{s}}$ of service $\hat{s}$ is implemented, and 0 otherwise. Now, given an xApp instance $f_\chi^{(j)}$, we let $\hat{v}_{c_{\hat{s}},f_\chi^{(j)}}$ indicate whether service configuration $c_{\hat{s}}$ is using $f_\chi^{(j)}$, and $\hat{\boldsymbol{\rho}}_{f_\chi^{(j)}}$ denote the vector indicating its current resource allocation.

• **No service disruption.** It is critical to account for the cost incurred by the system whenever OREO determines a new configuration for an existing service s. Indeed, due to the need to ensure continuity for a service s∈$\hat{\mathcal{S}}$∩$\mathcal{S}$, both the xApps required by $\{c_{\hat{s}} \mid \hat{z}_{c_{\hat{s}}}\}_{\hat{s} \in \hat{\mathcal{S}} \cap \mathcal{S}}$ and by $\{c_s \mid z_{c_s}\}_{s \in \hat{\mathcal{S}} \cap \mathcal{S}}$ have to coexist before (i) turning off the relative currently implemented, but no longer required, xApps, and (ii) instantiating the remaining functions required by the residual services in $\mathcal{S}$. We

TABLE II
NOTATION USED FOR DECISION VARIABLES

Symbol	Description
z_{c_s}	Binary variable for service configuration c_s selection
$v_{c_s, f_\chi^{(j)}}$	Binary variable indicating if $f_\chi^{(j)}$ is used in configuration c_s
$\boldsymbol{\rho}_{f_\chi^{(j)}}$	Resource allocation for the j-th instance of xApp f_χ

then define $\mathcal{F}_1$ as the set of xApps required by the existing services that should not be deactivated:

$$\mathcal{F}_1 = \{f_\chi^{(j)} \mid \sum_{c_{\hat{s}} \in \mathcal{C}_{\hat{s}}} \hat{v}_{c_{\hat{s}}, f_\chi^{(j)}} = 1\}_{f\in\mathcal{F},\, \chi_f\in\mathcal{X}_f,\, j,\, \hat{s}\in\hat{\mathcal{S}}\cap\mathcal{S}} \,.$$

Similarly, we define $\mathcal{F}_2$ as the set of xApps required in the last defined near-RT RIC's setting for the services whose operation must not be disrupted: Then, to avoid service disruptions, we must have:

$$\sum_{f_\chi^{(j)} \in \mathcal{F}_1 \setminus \mathcal{F}_2} \hat{\rho}_{f_\chi^{(j)},k} + \sum_{f_\chi^{(j)} \in \mathcal{F}_2} \rho_{f_\chi^{(j)},k} \leq B_k,\ \forall k \in \mathcal{K} \qquad (8)$$

where $\mathcal{F}_1 \setminus \mathcal{F}_2 = \{f_\chi^{(j)} \mid f_\chi^{(j)} \in \mathcal{F}_1 \wedge f_\chi^{(j)} \notin \mathcal{F}_2\}$.

• **Objective function.** The xDeSh problem aims to define an xApp selection and resource allocation policy that (i) maximizes the number of offered services, and (ii) minimizes the near-RT RIC resource consumption. Accordingly, we describe the problem objective function as:

$$\Psi(\boldsymbol{z},\boldsymbol{v},\boldsymbol{\rho}) = \sum_{s\in\mathcal{S}} \sum_{c_s\in\mathcal{C}_s} z_{c_s} p_s - \frac{1}{K} \sum_{f\in\mathcal{F}} \sum_{\chi\in\mathcal{X}_f} \sum_{j} \sum_{k\leq K} \frac{\rho_{f_\chi^{(j)},k}}{B_k}$$

where the decision variables (see Tab. II) have been vectorized. The xDeSh problem can then be formulated as:

xApp Deployment and Sharing (xDeSh) Problem

$$\max_{\boldsymbol{z},\boldsymbol{v},\boldsymbol{\rho}} \ \Psi(\boldsymbol{z},\boldsymbol{v},\boldsymbol{\rho})$$

$$\text{s.t.} \ (2), (3), (4), (5), (6), (7), (8)$$

$$z_{c_s} \in \{0,1\} \ \forall s \in \mathcal{S}, c_s \in \mathcal{C}_s$$

$$v_{c_s, f_{\chi_f}^{(j)}} \in \{0,1\} \ \forall c_s \in \mathcal{C}_s, f \in \mathcal{F}, \chi_f \in \mathcal{X}_f, j$$

$$\rho_{f_\chi^{(j)},k} \in [0, B_k] \ \forall k \in \mathcal{K}, f \in \mathcal{F}, \chi_f \in \mathcal{X}_f, j$$

Proposition 1. *The xDeSh problem is NP-hard.*

Proof. We show that any instance of the well-known NP-hard multi-commodity facility location problem (FLP) can be reduced to an instance of the xDeSh problem. To this end, we first recall that FLP aims to determine (i) the optimal location for deploying facilities, and (ii) the commodities that each facility offers to fulfill the consumers' requests while minimizing construction costs [14]. We then focus on a simplified version of the xDeSh problem where each service can have multiple configurations, but each configuration consists of a single function ($|\{f\}_{f\in\mathcal{C}_s}|=1$, $\forall s\in\mathcal{S}, c_s\in\mathcal{C}_s$). We disregard the ability to activate functions with different complexity factors ($|\mathcal{X}_f|=1$, $\forall f\in\mathcal{F}$) and assume that the minimum resource allocation for functions to meet the target service performance is known. Also, we neglect reconfiguration costs.

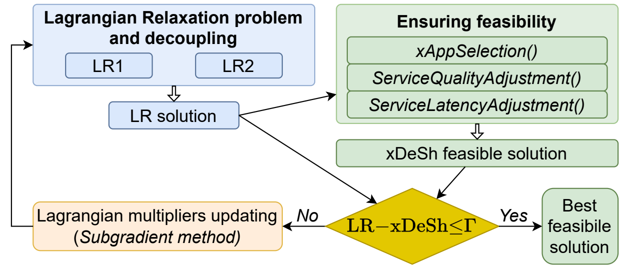

Fig. 3. The xDeSh problem is solved with an iterative algorithm that alternates the Lagrangian relaxation and the subgradient method until the set stopping criterion is met.

The following mapping can then be defined between the entities in the multi-commodity FLP and the reduced xDeSh problem: (i) facility deployment locations correspond to functions; (ii) commodities available at facilities correspond to the services that functions can provide; (iii) customers are the MNOs; (iv) the construction cost corresponds to the function deployment cost. By further observing that the above reduction can be obtained in polynomial time, the thesis follows. ■

IV. Solving the xDeSh Problem

Motivated by Proposition 1 above, we propose an efficient heuristic solution, which is outlined in Sec. IV-A. Then, we describe in detail each of its building blocks, namely, a Lagrangian relaxation and a decoupling method (Sec. IV-B), a feasibility testing algorithm (Sec. IV-C), and, finally, a subgradient method (Sec. IV-D). We remark that our solution algorithm resides in the OREO engine, introduced in Sec. II.

A. Overview of the algorithmic solution

In the proof of Prop. 1, we underlined the similarity between the xDeSh problem and the FLP. Then, inspired by existing efficient FLP solvers [15]–[17], we design our algorithmic solution adopting an iterative, two-stage approach. As illustrated in Fig. 3, our solution framework first leverages the Lagrangian Relaxation (LR) method, a relaxation technique that incorporates the effect of the constraints that entail the problem's complexity into the objective function. To enforce these constraints, the method introduces penalty terms, known as Lagrange multipliers. However, this approach may provide a solution to the xDeSh problem that is not feasible.

To solve this issue, we combine the LR method with an algorithm capable of identifying the violated constraints and making adjustments to the relaxed solution. Importantly, the feasible and infeasible solutions that we get represent the lower and upper bounds on the optimal solution, respectively. To obtain increasingly tighter bounds, we leverage the subgradient method – a robust technique that provides a policy for progressively updating the Lagrangian multipliers.

The above solution process is repeated until one of the three stopping criteria is met. The first criterion terminates the process when the LR and the obtained solutions differ by less than a given threshold, Δ. Subsequently, the iterative process is stopped if the step size, determining the size of updates to the Lagrangian multipliers through the subgradient method, drops below a designated threshold Γ. Indeed, the step size is initially set to large values to facilitate rapid updates and then halved when the iterative process fails to improve the solution for N iterations, aiming to refine and stabilize the overall process. The third stopping criterion finally sets a predefined maximum number of overall iterations, Λ.

B. Problem relaxation and decoupling

To apply the LR to the xDeSh problem, we note that constraints (3), (5), and (6) entangle the service configuration and the xApp selection subproblems. However, since the LR deals with inequalities, we split constraint (3) into:

$$\sum_{\chi\in\mathcal{X}_f}\sum_{j} v_{c_s,f_\chi^{(j)}} \geq z_{c_s}, \qquad \forall s\in\mathcal{S}, c_s\in\mathcal{C}_s, f\in\mathcal{V}_{c_s} \tag{9}$$

$$\sum_{\chi\in\mathcal{X}_f}\sum_{j} v_{c_s,f_\chi^{(j)}} \leq 1, \qquad \forall s\in\mathcal{S}, c_s\in\mathcal{C}_s, f\in\mathcal{V}_{c_s}\,. \tag{10}$$

The two inequalities above indeed provide, respectively, a lower and an upper bound on the number of xApps implementing the same function for a given service configuration c_s, and they collapse into (3) for the selected configuration. Moreover, we linearize (6), which links the configuration selection with the relative expected latency by adopting the big-M linearization for each service s implemented according to configuration c_s:

$$\tau_{c_s}(\{\boldsymbol{\rho}_{f_\chi^{(j)}} \mid \sum_{j} v_{c_s,f_{\chi_f}^{(j)}}=1\}_{f\in c_s}) - T_s \leq M(1-z_{c_s})\,. \tag{11}$$

We relax constraints (5), (9), and (11) by introducing, respectively, the non-negative Lagrangian penalty terms $\boldsymbol{\beta}=\{\beta_{c_s,f}\}_{s,c_s,f}$, $\boldsymbol{\gamma}=\{\gamma_{c_s}\}_{s,c_s}$, and $\boldsymbol{\delta}=\{\delta_{c_s}\}_{s,c_s}$, which leads to the below LR formulation.

xDeSh Lagrangian Relaxation problem (LR):

$$\max_{\boldsymbol{z},\boldsymbol{v},\boldsymbol{\rho}} \ \Psi_L(\boldsymbol{z},\boldsymbol{v},\boldsymbol{\rho},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta})$$
$$\text{s.t.} \ (2),(4),(7),(8),(10)$$

with the Lagrangian function Ψ_L defined as:

$$\begin{cases} \Psi_L(\boldsymbol{z},\boldsymbol{v},\boldsymbol{\rho},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) = \Psi_{L,1} + \Psi_{L,2} \\ \Psi_{L,1}(\boldsymbol{z},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) = \sum_{c_s} z_{c_s}(p_s - \gamma_{c_s}Q_s - M\delta_{c_s} - \\ \qquad \sum_{f\in c_s}\beta_{c_s,f}) + \sum_{c_s}\delta_{c_s}(M+T_s) \\ \Psi_{L,2}(\boldsymbol{v},\boldsymbol{\rho},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) = \sum_{s,c_s}[\gamma_{c_s}q_{c_s} - \delta_{c_s}\tau_{c_s} + \\ \qquad \sum_{f\in c_s,\chi,j}\beta_{c_s,f}v_{c_s,f_\chi^{(j)}}] - \frac{1}{K}\sum_{f,\chi,j,k}\frac{\rho_{f_\chi^{(j)},k}}{B_k}\,. \end{cases}$$

Conveniently, the xDeSh LR problem defined above can be easily decomposed by applying primal decomposition. LR is indeed separable into two independent subproblems, dealing, respectively, with (i) the service configuration selection (**LR1**) and (ii) the xApp instantiation and resource reservation (**LR2**):

$$\textbf{LR1 problem:} \quad \max_{\boldsymbol{z}} \quad \Psi_{L,1}(\boldsymbol{z},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) \qquad \text{s.t.} \quad (2)$$

$$\textbf{LR2 problem:} \quad \max_{\boldsymbol{v},\boldsymbol{\rho}} \quad \Psi_{L,2}(\boldsymbol{v},\boldsymbol{\rho},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) \qquad \text{s.t.} \quad (4),(7),(8),(10)\,.$$

Algorithm 1 Ensuring feasibility

Input: $\{\bar{\boldsymbol{z}},\bar{\boldsymbol{v}},\bar{\boldsymbol{\rho}}\}$ ▷ *Relaxed solution.*
Output: $\{\hat{\boldsymbol{z}},\hat{\boldsymbol{v}},\hat{\boldsymbol{\rho}}\}$ ▷ *Feasible solution.*
$\hat{\boldsymbol{z}} \leftarrow \bar{\boldsymbol{z}}$ ▷ *Accept relaxed service configuration choice.*
if Eq. (9) is **not** respected for any service $s\in\mathcal{S}$ **then**
  $\hat{v}_{c_s,f_\chi^{(j)}} \leftarrow$ `xAppSelection` algorithm ▷ *Fix the relaxed xApp selection.*
if Eq. (5) is **not** respected for any service $s\in\mathcal{S}$ **then**
  $\hat{v}_{c_s,f_\chi^{(j)}} \leftarrow$ `ServiceQualityAdjustment` ▷ *Increase the service quality by adjusting functions complexity.*
if Eq. (11) is **not** respected for any service $s\in\mathcal{S}$ **then**
  $\hat{\rho}_{f_\chi^{(j)},\text{cpu}} \leftarrow$ `ServiceLatencyAdjustment` ▷ *Reduce the service latency by adjusting xApp CPU allocation.*
while Eq. (7) or (8) are **not** respected **do**
  $\tilde{s} \leftarrow$ the lowest-priority implemented service with the highest deployment cost
  $\hat{z}_{c_{\tilde{s}}} \leftarrow 0$ ▷ *Deactivate service $\tilde{s}$*

LR1 and LR2 are convex and therefore can be solved efficiently using standard solvers. Moreover, as LR1 and LR2 are completely independent and separable, they can be solved concurrently. The LR solution, also referred to as the relaxed solution, is obtained by combining the solutions of LR1 and LR2. We can then prove the following proposition:

Proposition 2. *The solution of the LR1 and LR2 problems provides a solution to the xDeSh Lagrangian Relaxation problem with an approximation ratio of* 3.

Proof. LR1 is a knapsack problem with the constraint that each service can have only one active configuration at a time. LR2, instead, is a variant of the single-layer multi-commodity FLP, as xApps correspond to facilities, service configurations to customers, and services to commodities, with the additional objective of allocating resources to xApps to maximize the quality and minimize the services latency. Heuristics with known approximation ratio exist for the knapsack ($(1-\epsilon)$ if the items size is within ϵ of the knapsack capacity [18]) and the uncapacitated FLP (3 using Primal-Dual methods [19]). Consequently, considering that (i) the relaxed solution is obtained by combining the LR1 and LR2 solutions, (ii) there is only 1 possible configuration per service (i.e., $\epsilon = 1$ in the knapsack), and (iii) overlooking the xApp resource allocation problem, the thesis holds. ■

C. Ensuring feasibility

As mentioned, the solution of the LR1 and LR2 problems provide a solution to the xDeSh Lagrangian Relaxation problem, which however may be unfeasible. We thus propose a multi-stage algorithm to derive a feasible solution at a later step. As depicted in Fig. 3, this algorithm receives the relaxed solution, and then identifies and properly rectifies any violation of the relaxed constraints (5), (9), and (11). The stages of this approach are reported in Alg. 1 and detailed below.

1) Ensuring a compliant xApp selection. The first stage assesses the compliance with constraint (5) for each chosen service configuration. This involves verifying if the xApps selected by the relaxed solution can meet the configuration functional requirements. If this condition is not met, the missing xApps need to be added to the initial solution. Then it is checked which xApps included in the solution are already deployed, and, hence, could be shared. Sharing is applied only if the increase in resource consumption due to the consequent additional load would be smaller than the amount of resources needed to create a new instance of the xApp.

2) Meeting service quality requirements. Once the xApps offering the functions of each selected service configuration have been identified, we have to ensure that the service quality target is satisfied. If any service fails to meet this criterion, our strategy adopts the service configuration selection offered by the relaxed solution while enhancing the complexity (output quality) of the deployed functions.

The method *ServiceQualityAdjustment* (pseudocode omitted for brevity) accomplishes this task by identifying the xApp that, when selected with increased complexity, contributes the most to the provided service quality at the minimum resource cost, i.e., it has the highest quality efficiency. Furthermore, after ensuring sufficient quality for all services, it is evaluated whether it is possible to reduce the complexity of any function to decrease resource demand without violating the quality constraints of any service. If in so doing, the quality constraints of all services are still met, then the complexity reduction is considered appropriate.

3) Meeting service latency targets. A similar approach is undertaken for service latency. Specifically, a method *Service-LatencyAdjustment* (pseudocode omitted for brevity) increases the CPU allocation for each xApp contributing to a service that fails to meet its latency target. CPU allocation is increased first for the xApps for which the smallest CPU increase makes their latency equal to the target value.

4) Meeting the resource budget. The previous stages identify the xApps needed for the deployment of the requested services and adjust the compute resource allocation accordingly. However, the constraints (7)–(8) have to be fulfilled as well, i.e., it is imperative to verify the feasibility of the current solution and drop service requests as needed. The services to be discarded (if any) are those with the lowest priority and the highest deployment cost. This iterative procedure is repeated until the available budget is met by all resource types.

D. The subgradient method

To penalize the violations of the relaxed constraints, the values of the Lagrangian multipliers can be determined so that the extent of such violations is minimized. The subgradient method is a viable and computationally efficient approach to solving this [20]. The subgradient method is an iterative optimization algorithm that generalizes the gradient descent algorithm for non-differentiable functions. It involves iteratively updating the Lagrange multipliers in the direction of the subgradients of the LR problem objective function with respect to the Lagrange multipliers. Importantly, the subgradient method is effective with non-smooth and non-convex functions [20], as is the case of the xDeSh problem.

V. Performance Evaluation

We first evaluate OREO through extensive simulations (Sec. V-A), then we run experimental tests using a proof-of-concept implementation of the OREO framework supporting real-world RAN services and xApps (Sec. V-B).

TABLE III
Testing Scenarios

Scenario	N_s	$\lvert\mathcal{C}_s\rvert$	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{V}_{c_s}\rvert$	$\mathcal{X}_f$
Tiny (**T**)	8	2	8	4	3
Small (**S**)	8	3	8	4	2
Medium (**M**)	8	3	8	3	3
Large (**L**)	8	3	8	4	3
Extra Large (**XL**)	10	3	8	4	3
Huge (**H**)	12	3	10	4	3

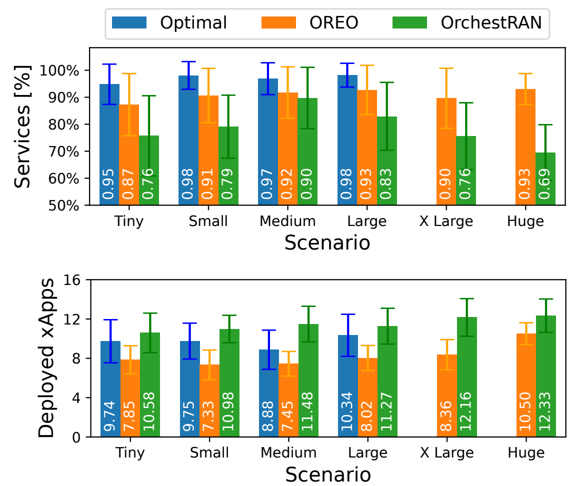

Fig. 4. Numerical results: Percentage of services (top) and xApps (bottom) deployed by Optimal, OREO, and OrchestRAN.

A. Numerical analysis

To test the effectiveness of OREO at scale, we developed a Python simulator. In the considered scenario, an MNO generates requests for a set of N_s services. The MNO requests are forwarded to the non-RT RIC, where the OREO selects the most amenable configuration among $|\mathcal{C}_s|$ possible ones. Each configuration involves at most $|\mathcal{V}_{c_s}|$ RAN functions among the $|\mathcal{F}|$ available, with each function featuring $\mathcal{X}_f$ different levels of complexity. We consider 6 scenarios of different scales as illustrated in Tab. III and present average results over 100 runs.

Benchmarks. We compare OREO against two alternative policies: the "Optimal" policy, which uses Gurobi to optimally solve the xDeSh problem, and OrchestRAN [7], a state-of-the-art O-RAN orchestrator. For a fair comparison, since OrchestRAN does not consider services as compositions of interconnected xApps, we let OrchestRAN handle each configuration of a service requested by the MNO as an xApp.

Approximation ratio estimate and run times. Tab. IV reports the numerical estimates of the approximation ratio α (i.e., the ratio of the heuristic performance to the optimum) and the execution times for the implemented orchestration policies. Specifically, the table presents the 90% confidence interval lower-bound (0.9α) and the average value $\bar{\alpha}$, for both OREO and OrchestRAN. Note that, for the two largest scenarios (i.e., XL and H), where the number of optimization variables is over

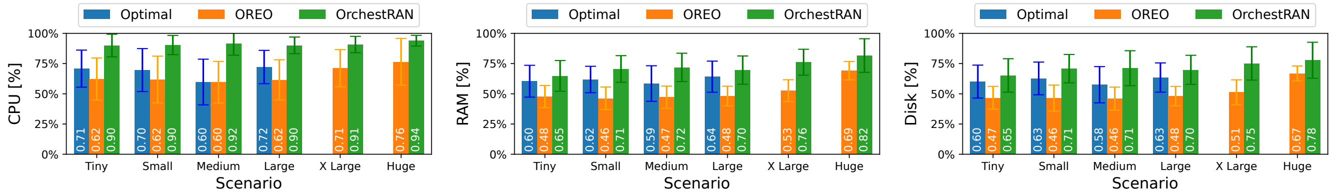

Fig. 5. Numerical results: CPU (left), RAM (center), and Disk (right) resources used by Optimal, OREO, and OrchestRAN.

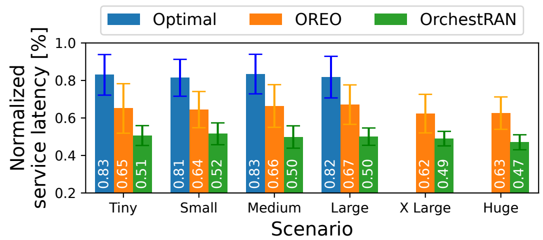

Fig. 6. Normalized latency performance of RAN services offered by Optimal, OREO, and OrchestRAN.

10^4, we were unable to compute such metrics, as computing the optimum becomes impractical.

OREO consistently provides solutions that are at least within 0.70 from the optimum across all tested scenarios. With 90% confidence, the estimated ratio increases to 0.83; also, the relatively narrow confidence interval suggests that OREO's performance consistently remains close to the average values. Conversely, the gap between OrchestRAN's performance and the optimum progressively widens as the size of the scenario grows, and even in the simplest scenario (T), it achieves an approximation ratio that is 38% worse than that of OREO.

As for the execution times, OREO is highly efficient, exhibiting average values that are lower than 10 s, even when handling large-scale scenarios, and it proves to be from 3 to 40 times faster than the optimum.

Deployed services and xApps. We now look at the percentage of services and xApps provided by OREO and its counterparts. According to Fig. 4(top), the Optimal provides approximately 95% deployed services, as the scenario varies from T to L, and OREO achieves comparable (within 8%) performance. On the contrary, OrchestRAN results in a significant decrease in the service implementation rate, as it yields 15% fewer services compared to the optimum. The underlying reasons can be found in Fig. 4 (bottom), which illustrates the average number of xApps deployed by OREO and its alternatives in the considered scenarios. Although OREO deploys more services than OrchestRAN, it implements fewer xApps, thanks to its superior ability to share xApps and adjust the resources allocated to them. In particular, OREO instantiates 28.2% fewer xApps than OrchestRAN and 20.7% fewer than the optimum, which however can better use the xApps it deploys to ultimately offer more services.

TABLE IV
EMPIRICAL APPROXIMATION RATIO AND RUN TIMES

	OREO					OrchestRAN			Opt.
	α	0.9α	$\bar{\alpha}$	t	S	α	0.9α	$\bar{\alpha}$	t
T	0.71	0.83	0.85	8.36	6.46	0.44	0.69	0.73	54.00
S	0.77	0.86	0.88	3.44	15.75	0.23	0.35	0.38	54.18
M	0.71	0.88	0.92	6.47	3.12	0.24	0.39	0.41	20.19
L	0.81	0.86	0.89	4.51	41.33	0.16	0.34	0.37	186.40
XL	–	–	–	6.33	–	–	–	–	–
H	–	–	–	10.31	–	–	–	–	–

Near-RT RIC resource consumption. The higher ability to share xApps exhibited by OREO in comparison to its benchmark is confirmed by its lower resource utilization at the near-RT RIC. While OrchestRAN employs almost the whole CPU budget under all scenarios (Fig. 5 (left)) and takes a significant toll also on RAM (Fig. 5 (center)) and Disk (Fig. 5 (right)), OREO yields substantial resource savings (25.6% of CPU). Notice that also Optimal uses a significant amount of resources, as minimizing resource consumption is not the only objective of the xDeSh problem. Accordingly, relatively to OREO, it makes up for the larger resource consumption with a higher number of deployed services.

Meeting service requirements. OREO properly scales the resources allocated to the xApps according to the aggregated load of the services sharing the xApp, so that it can successfully fulfill the service requirements. Focussing on service latency for brevity, Fig. 6 shows the normalized service latency (i.e., the ratio of actual to target service latency) over the test scenarios. Both OREO and its counterparts can meet the target service latency in all cases (notice that the average is below 1 as the schemes must ensure the target latency for all possible service requests). However, looking at Fig. 6 and Fig. 5 together, it is evident that OREO, similarly to the Optimal policy, fulfills service requirements while saving substantial resources with respect to OrchestRAN.

B. *Testbed setup and results*

Our testbed integrates the OREO architecture (Fig. 1) with an emulated softwarized cellular base station. Our experimental platform includes an O-Cloud environment where xApps can be instantiated as docker containers.

In our experiments, an MNO module generates a service request every 100 s. Each request involves from 1 to 3 services, chosen with equal probability among those listed below, and with a randomly-selected target latency-quality pair. The possible latency targets for all services are 0.1, 0.2, and 0.5 s; the service quality targets, instead, are service-specific.

• *Traffic forecasting:* it predicts future user traffic loads, as required by many proactive RAN controllers. It consists of a single configuration with a traffic forecaster xApp f_1 that uses a Long Short-Term Memory (LSTM) model to predict the traffic load. The xApp supports 3 complexity levels, corresponding to different numbers of input samples (6 to 30) and LSTM layers (1 to 4), and different sizes of the LSTM

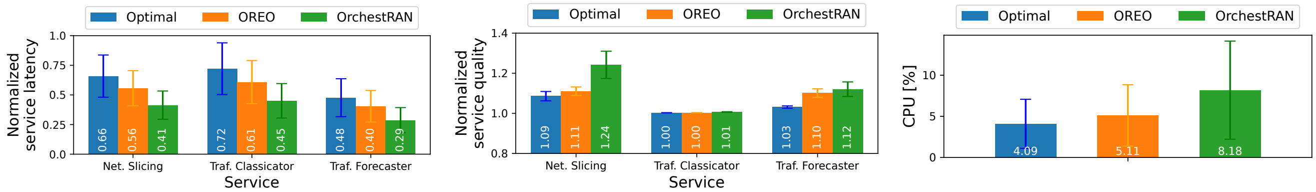

Fig. 7. Testbed results: normalized service latency (left) and quality (center), and CPU consumption (right) of OREO, Optimal and OrchestRAN.

hidden layers (1 to 5). The possible service target quality figures are 0.9, 0.925, or 0.95.

• *Traffic classification:* it identifies the applications generating monitored traffic flows, e.g., so that application-dedicated policies can be applied. The service includes a single configuration with a traffic classificator xApp f_2 that uses a Random Forest classifier to label traffic samples. The xApp supports 3 complexity levels, depending on the number of applications that can generate traffic flows (3 to 20). The service can be requested with target quality values equal to 0.7, 0.8, or 0.9.

• *Network slicing:* it optimizes the allocation of radio resources to network slices according to their type (eMBB, uRLLC, or mMTC). The service can be implemented with 4 different configurations: $f_1{+}f_2{\rightarrow}f_3$; $f_1{\rightarrow}f_3$; $f_2{\rightarrow}f_3$; f_3, where f_3 is the RL-based slicing policy introduced in [9]. The service can be requested with a target quality value of 0.6, 0.8, or 0.9.

Once received, the non-RT RIC processes the service requests and uses OREO (or one of the benchmarks) to decide the set of xApps and their resource allocation that best provide the service. Then, the xApp lifecycle manager in the non-RT RIC instructs the instantiation of the xApps in the near-RT RIC accordingly. Each service has a 100-s lifetime and the traffic scenario is determined using the dataset in [9], which refers to a base station serving 6 users, evenly split into eMBB, URLLC, and mMTC-like traffic patterns.

Results. Fig. 7 presents the average value and the 90% confidence interval of the normalized (i.e., the ratio of actual to target) service latency (left), service quality (center), and CPU consumption (right). Different than some of the scenarios presented in the previous subsection, the reduced scale of the scenario considered here allows both OREO and its benchmarks to meet all the service requirements. However, OREO selects better service configurations, similar to the Optimal policy, which renders a reduced utilization of CPU resources when compared to OrchestRAN. In fact, OrchestRAN struggles to find the right balance between meeting service requirements and resource utilization. More specifically, although OrchestRAN provides lower service latency and higher service quality, it uses 60% and 100% higher CPU consumption than OREO and Optimal, respectively. The savings attained by OREO, only consuming 24.9% more CPU resources than Optimal, are achieved by pushing the latency (23%) and quality (5%) requirements closer to their target than OrchestRAN.

VI. Related Work

Recently, considerable attention has been paid to the design of near-RT RIC xApps [21], aiming at optimally controlling O-RAN networks. Relevant examples are xApps for network traffic classification [22], [23] and network load forecasting [24], [25]. Further, an xApp can lay down policies to define RAN slices, as done in [9], [26], [27], or to manage the RAN radio resources [28], [29].

In this context, proficiently managing and orchestrating the RAN becomes even more crucial, necessitating the best exploitation of the multitude of multi-vendor solutions [30]. Despite the large body of work existing on RAN orchestration [31], [32], few studies have tackled network intelligence management in O-RAN. Among these, [33] proposes a computationally efficient orchestrator for energy consumption optimization in virtual RANs. The study in [34] introduces, instead, a distributed dynamic policy for instantiating inference models, providing performance guarantees.

The closest work to ours is OrchestRAN [7], a pioneering O-RAN orchestrator that identifies the optimal set of xApps to deploy and their location to offer the services requested by network operators while meeting the performance targets. It assumes the RAN services to be monolithic, i.e., solely provided by an xApp, thus limiting the possibility of sharing low-level operations between services. Thus, [7], unlike our work, disregards the issue of RAN resources consumption – a major contribution to MNOs' OPEX [33].

VII. Conclusions

We proposed OREO, an O-RAN orchestrator for xApp-based network services. Unlike previous works, OREO considers that services can be deployed through different sets of shareable elementary functions, with each function possibly yielding an output of different quality and implemented as an xApp. To limit resource consumption while fulfilling the service requirements, OREO properly tunes xApps at different complexity factors, which correspond to different quality-latency-resource demand tradeoffs. Numerical results show that OREO performs close to the optimum, and, compared to the state of the art, it allocates more (16.2% on average and 35% in the largest scenario) services and consumes less CPU (25.6% on average and over 31% in small-medium scenarios), while meeting the service requirements. Experimental results obtained through our proof-of-concept testbed confirm the good performance of OREO and its ability to efficiently deploy xApps, yielding a 37.5% reduction on CPU consumption with respect to the state of the art.

Acknowledgments

This work was supported by the European Commission through Grant No. 101095890 (PREDICT-6G project) and Grant No. 101096379 (CENTRIC project), and by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE0000001 - program "RESTART").

References

[1] Z. Zhang, Y. Xiao, Z. Ma, M. Xiao, Z. Ding, X. Lei, G. K. Karagiannidis, and P. Fan, "6G Wireless Networks: Vision, Requirements, Architecture, and Key Technologies," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 28–41, 2019.

[2] O-RAN Working Group 1, "O-RAN Architecture Description 9.00," June 2023.

[3] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges," *IEEE Communications Surveys Tutorials*, vol. 25, no. 2, pp. 1376–1411, 2023.

[4] A. Garcia-Saavedra and X. Costa-Pérez, "O-RAN: Disrupting the Virtualized RAN Ecosystem," *IEEE Communications Standards Magazine*, vol. 5, no. 4, pp. 96–103, 2021.

[5] O-RAN Working Group 1, "Use Cases Analysis Report 11.0," June 2023.

[6] M. Camelo, L. Cominardi, M. Gramaglia, M. Fiore, A. Garcia-Saavedra, L. Fuentes, D. De Vleeschauwer, P. Soto-Arenas, N. Slamnik-Krijestorac, J. Ballesteros, C.-Y. Chang, G. Baldoni, J. M. Marquez-Barja, P. Hellinckx, and S. Latré, "Requirements and Specifications for the Orchestration of Network Intelligence in 6G," in *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, 2022, pp. 1–9.

[7] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "OrchestRAN: Network Automation through Orchestrated Intelligence in the Open RAN," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 270–279.

[8] J. Zhang, Z. Wang, N. Ma, T. Huang, and Y. Liu, "Enabling efficient service function chaining by integrating NFV and SDN: Architecture, challenges and opportunities," *IEEE Network*, vol. 32, no. 6, pp. 152–159, 2018.

[9] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "ColO-RAN: Developing Machine Learning-based xApps for Open RAN Closed-loop Control on Programmable Experimental Platforms," *IEEE Transactions on Mobile Computing*, pp. 1–14, 2022.

[10] O-RAN Working Group 6, "Cloud Platform Reference Designs 2.0," February 2021.

[11] I. Cohen, C. F. Chiasserini, P. Giaccone, and G. Scalosub, "Dynamic Service Provisioning in the Edge-cloud Continuum with Provable Guarantees," *arXiv preprint arXiv:2202.08903*, 2022.

[12] F. Ben Jemaa, G. Pujolle, and M. Pariente, "QoS-Aware VNF Placement Optimization in Edge-Central Carrier Cloud Architecture," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–7.

[13] R. Gouareb, V. Friderikos, and A.-H. Aghvami, "Virtual Network Functions Routing and Placement for Edge Cloud Latency Minimization," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2346–2357, 2018.

[14] J. Karkazis and T. Boffey, "The Multi-Commodity Facilities Location Problem," *Journal of the Operational Research Society*, vol. 32, no. 9, pp. 803–814, 1981.

[15] A. Klose, "A Lagrangean relax-and-cut approach for the two-stage capacitated facility location problem," *European journal of operational research*, vol. 126, no. 2, pp. 408–421, 2000.

[16] ——, "An LP-Based Heuristic for Two-Stage Capacitated Facility Location Problems," *Journal of the Operational Research Society*, vol. 50, no. 2, pp. 157–166, 1999.

[17] B. Gendron, P.-V. Khuong, and F. Semet, "A Lagrangian-Based Branch-and-Bound Algorithm for the Two-Level Uncapacitated Facility Location Problem with Single-Assignment Constraints," *Transportation Science*, vol. 50, no. 4, pp. 1286–1299, 2016.

[18] A. M. Frieze, M. R. Clarke *et al.*, "Approximation Algorithms for the m-Dimensional 0-1 Knapsack Problem: Worst-Case and Probabilistic Analyses," *European Journal of Operational Research*, vol. 15, no. 1, pp. 100–109, 1984.

[19] K. Jain and V. V. Vazirani, "Approximation Algorithms for Metric Facility Location and k-Median Problems Using the Primal-Dual Schema and Lagrangian relaxation," *Journal of the ACM (JACM)*, vol. 48, no. 2, pp. 274–296, 2001.

[20] M. Held, P. Wolfe, and H. P. Crowder, "Validation of Subgradient Optimization," *Mathematical programming*, vol. 6, pp. 62–88, 1974.

[21] O-RAN Working Group 3, "O-RAN Near-RT RAN Intelligent Controller Near-RT RIC Architecture 2.01," March 2022.

[22] J. Shi, M. Liu, C. Hou, M. Jiang, and G. Xiong, "Online Encrypted Mobile Application Traffic Classification at the Early Stage: Challenges, Evaluation Criteria, Comparison Methods," in *2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS)*, 2021, pp. 1128–1135.

[23] Y. Li, B. Liang, and A. Tizghadam, "Robust Online Learning against Malicious Manipulation and Feedback Delay With Application to Network Flow Classification," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2648–2663, 2021.

[24] A. M. Nagib, H. Abou-Zeid, H. S. Hassanein, A. Bin Sediq, and G. Boudreau, "Deep Learning-Based Forecasting of Cellular Network Utilization at Millisecond Resolutions," in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–6.

[25] N. Salhab, R. Langar, R. Rahim, S. Cherrier, and A. Outtagarts, "Autonomous Network Slicing Prototype Using Machine-Learning-Based Forecasting for Radio Resources," *IEEE Communications Magazine*, vol. 59, no. 6, pp. 73–79, 2021.

[26] S. Bakri, P. A. Frangoudis, A. Ksentini, and M. Bouaziz, "Data-driven ran slicing mechanisms for 5g and beyond," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4654–4668, 2021.

[27] J. Mei, X. Wang, K. Zheng, G. Boudreau, A. B. Sediq, and H. Abou-Zeid, "Intelligent Radio Access Network Slicing for Service Provisioning in 6G: A Hierarchical Deep Reinforcement Learning Approach," *IEEE Transactions on Communications*, vol. 69, no. 9, pp. 6063–6078, 2021.

[28] S. Tripathi, C. Puligheddu, C. F. Chiasserini, and F. Mungari, "A Context-Aware Radio Resource Management in Heterogeneous Virtual RANs," *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, no. 1, pp. 321–334, 2022.

[29] S. Shen, T. Zhang, S. Mao, and G.-K. Chang, "DRL-Based Channel and Latency Aware Radio Resource Allocation for 5G Service-Oriented RoF-MmWave RAN," *Journal of Lightwave Technology*, vol. 39, no. 18, pp. 5706–5714, 2021.

[30] A. Arnaz, J. Lipman, M. Abolhasan, and M. Hiltunen, "Towards integrating intelligence and programmability in open radio access networks: A comprehensive survey," *Ieee Access*, 2022.

[31] J. A. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Pérez, A. Banchs, and J. J. Alcaraz, "vrain: Deep learning based orchestration for computing and radio resources in vrans," *IEEE Transactions on Mobile Computing*, vol. 21, no. 7, pp. 2652–2670, 2022.

[32] X. Li, A. Garcia-Saavedra, X. Costa-Perez, C. J. Bernardos, C. Guimarães, K. Antevski, J. Mangues-Bafalluy, J. Baranda, E. Zeydan, D. Corujo, P. Iovanna, G. Landi, J. Alonso, P. Paixão, H. Martins, M. Lorenzo, J. Ordonez-Lucena, and D. R. López, "5Growth: An End-to-End Service Platform for Automated Deployment and Management of Vertical Services over 5G Networks," *IEEE Communications Magazine*, vol. 59, no. 3, pp. 84–90, 2021.

[33] R. Singh, C. Hasan, X. Foukas, M. Fiore, M. K. Marina, and Y. Wang, "Energy-Efficient Orchestration of Metro-Scale 5G Radio Access Networks," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[34] T. Si Salem, G. Castellano, G. Neglia, F. Pianese, and A. Araldo, "Towards Inference Delivery Networks: Distributing Machine Learning with Optimality Guarantees," in *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, 2021, pp. 1–8.