RIC-O: Efficient Placement of a Disaggregated and Distributed RAN Intelligent Controller With Dynamic Clustering of Radio Nodes

Gabriel Matheus Almeida, Gustavo Zanatta Bruno, *Graduate Student Member, IEEE*, Alexandre Huff, Matti Hiltunen, *Member, IEEE*, Elias Procopio Duarte Jr., *Senior Member, IEEE*, Cristiano Bonato Both, and Kleber Vieira Cardoso

Abstract— The Radio Access Network (RAN) is the segment of cellular networks that provides wireless connectivity to end-users. The O-RAN Alliance has been transforming the RAN industry by proposing open RAN specifications and the programmable Non-Real-Time and Near-Real-Time RAN Intelligent Controllers (Non-RT RIC and Near-RT RIC). Both RICs provide platforms for running applications called rApps and xApps, respectively, to optimize the RAN behavior. We investigate the disaggregation of the Near-RT RIC into components that meet stringent latency requirements while presenting a cost-effective solution. For example, the O-RAN Signalling Storm Protection requires the Near-RT RIC to support end-to-end control loop latencies as low as 10 ms. We propose the novel RIC Orchestrator (RIC-O) that optimizes the deployment of the Near-RT RIC components across the cloud-edge continuum. Edge computing nodes often present limited resources and are expensive compared to cloud computing. Performance-critical components of Near-RT RIC and certain xApps should run at the edge while other components can run on the cloud. Furthermore, RIC-O employs an efficient strategy to react to sudden changes and re-deploy components dynamically. The proposal is evaluated both analytically and through real-world experiments in an extended Kubernetes deployment implementing RIC-O and the disaggregated Near-RT RIC.

Index Terms— RAN intelligent controller, O-RAN, near-RT RIC, placement, disaggregation.

Gabriel Matheus Almeida and Kleber Vieira Cardoso are with the Institute of Informatics (INF), Universidade Federal de Goiás (UFG), Goiás 74690-900, Brazil (e-mail: gabrielmatheus@inf.ufg.br; kleber@inf.ufg.br).

Gustavo Zanatta Bruno and Cristiano Bonato Both are with the Applied Computing Graduate Program (PPGCA), University of Vale do Rio dos Sinos (UNISINOS), São Leopoldo 93022-750, Brazil (e-mail: zanattabruno@edu.unisinos.br; cbboth@unisinos.br).

Alexandre Huff is with the Federal Technological University of Paraná (UTFPR), Toledo 85902-490, Brazil (e-mail: alexandrehuff@utfpr.edu.br).

Matti Hiltunen is with AT&T Shannon Labs, Bedminster, NJ 07921 USA (e-mail: hiltunen@research.att.com).

Elias Procopio Duarte Jr. is with the Department of Informatics (DInf), Federal University of Parana (UFPR), Curitiba 81531-980, Brazil (e-mail: elias@inf.ufpr.br).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSAC.2023.3336159.

Digital Object Identifier 10.1109/JSAC.2023.3336159

I. INTRODUCTION

THE Radio Access Network (RAN) is considered the most critical segment of a cellular network [1]. A RAN comprises network nodes that provide direct wireless connection to end-users. It requires continuous innovations to improve the overall mobile network performance and user experience. Novel RAN technologies have included innovations related to wireless communication technologies (e.g., mmWave, massive MIMO, intelligent reflective surfaces, and THz communications). Other innovations are related to current networking technologies, particularly network softwarization, virtualization, programmability, and management (e.g., NFV, SDN, Network Slicing, and Zero touch network & Service Management). Traditionally, the RAN accounts for most of the CAPEX and OPEX costs of a wireless mobile network. Network operators also have to face several hurdles, such as interoperability issues, resulting from the diversity of hardware from different vendors and the burden of continuously supporting new services and applications. In this context, the O-RAN Alliance [2] has been transforming the RAN industry by proposing open, virtualized, fully interoperable, and intelligent mobile networks [3].

The O-RAN Alliance defines a series of specifications describing open interfaces to ensure multi-vendor interoperability among the main components that form the RAN. That "openness" has been carefully introduced to allow interoperability without hurting the intellectual property of the participating companies. Given the significant investments necessary to promote innovation of RAN technologies, this approach is essential to keep the interest of those companies in contributing to this vital area. Yet another advantage of an open RAN is that it decreases the barriers for newcomers that can contribute to solving specific problems and move the technology further, particularly in academia. On a side note, it is important to highlight that O-RAN specifications [4] are aligned with 3GPP standards [5]. Therefore, it is reasonable

to assume that as the 3GPP introduces enhancements, they are promptly incorporated by the O-RAN Alliance.

The new RAN architecture [4] stands out as a key contribution among the O-RAN specifications. The architecture combines concepts from SDN and NFV and also takes into account cloud-native technologies while largely adopting Artificial Intelligence (AI) and Machine Learning (ML) technologies [6]. It adopts SDN concepts, such as control and data planes separation and the possibility of having a remote RAN controller. The design of a RAN controller presents formidable challenges, as it must support the execution of tens to hundreds of RAN functions while featuring a large protocol stack. The O-RAN architecture splits the controller into two main building blocks: the Near-Real-Time RAN Intelligent Controller (Near-RT RIC) for time-sensitive operations and the Non-Real-Time RAN Intelligent Controller (Non-RT RIC) for operations that present fewer time restrictions. Moreover, the O-RAN specifications standardize open interfaces among the architecture components, which run as virtual (network) functions (or services) on cloud-native infrastructures.

The RAN controllers run AI/ML-based applications that establish control loops with the RAN nodes under their management. The Non-RT RIC runs applications called rApps that can present control loop latencies above 1 s. The Near-RT RIC runs applications called xApps that establish control loops constrained to time intervals between 10 ms and 1 s. The time constraints of a given control loop depend on the RAN function under the management of the corresponding xApp. For example, an xApp related to medium access management may need to complete the control loop under the 10 ms threshold, while an xApp related to user session management may tolerate longer delays of up to 1 s. In a large RAN, the Near-RT RIC (or some of its components) and latency-sensitive xApps must be replicated and assigned to manage a limited set of RAN nodes, i.e., a cluster of RAN nodes. Determining the minimum number of Near-RT RIC instances and where they must run is a non-trivial resource allocation problem. The problem becomes even more challenging considering the dynamics of a mobile wireless network. While this problem has been previously identified [7], some works rely on multiple instances of the Near-RT RIC [8]. To the best of our knowledge, there is no comprehensive solution to the problem as we describe in the present work.

In this work, we propose a disaggregation strategy of the Near-RT RIC so that individual components can be distributed and placed across the cloud-edge continuum. We also propose a RIC Orchestrator (RIC-O) to deploy (or place) and monitor the Near-RT RIC components so that they can meet the stringent latency requirements. The RIC-O employs optimization to deploy the Near-RT RIC components across the cloud-edge continuum while keeping the overall cost as low as possible. Edge computing nodes often present limited resources and are expensive compared to cloud computing nodes. While performance-critical components of the Near-RT RIC platform and certain xApps should run at the edge, other components can run on cloud nodes. Furthermore, RIC-O employs a fast and efficient strategy to react dynamically to sudden changes and redeploy components. We also explore the flexibility of the O-RAN architecture to introduce a proposal that replicates some specific components of the Near-RT RIC. We evaluate the proposal through analytical modeling and real-world experiments in an extended Kubernetes deployment that runs the RIC-O and the disaggregated Near-RT RIC.

The main contributions of this work can be summarized as follows; this work:

- Proposes a disaggregation strategy to place the Near-RT RIC components across the cloud-edge continuum;
- Formalizes the problem of minimizing the overall cost of the placement of Near-RT RIC components while ensuring the latency-sensitive control loop;
- Proposes a flexible approach that combines heuristic and optimal strategies to quickly provide cost-efficient solutions for placing Near-RT RIC components;
- Provides performance evaluation results to illustrate the advantage of the proposed approach;
- Furthermore, all implementations are publicly available, including the source code[1], thus allowing reproducibility.

The rest of the paper is organized as follows. Section II presents an overview of the O-RAN architecture, particularly of the Near-RT RIC. Section III describes the related work. Section IV provides the system model, problem formulation, and the optimal and heuristic strategies for the placement of the Near-RT RIC components. Section V introduces the RIC orchestrator, i.e., the proposed architecture that allows efficient placement of Near-RT RIC components and clustering of RAN nodes across the cloud-edge continuum. The implementation of the RIC orchestrator is described in Section VI. Real-world experiments and an analytical performance evaluation are presented in Section VII. Finally, Section VIII concludes the article and discusses future work.

II. BACKGROUND

The O-RAN architecture defines two RICs responsible for controlling and managing RAN nodes on different time scales. While the Non-RT RIC is in charge of tasks that can present a latency above 1 s, the Near-RT RIC is responsible for tasks that demand a latency between 10 ms and 1 s. Figure 1 illustrates the O-RAN architecture highlighting the Near-RT RIC internal components. Several of those components are not latency-sensitive, such as the `Management Services`, the `xApp Manager`, the `Subscription Manager`, and A1 and O1 terminations. However, `E2 Nodes`, E2 termination (E2T), `xApps`, and `Shared Layers` can be latency-sensitive and require proper management. The `xApp Manager` provides a flexible way for deploying and managing near real-time applications (xApps). The xApps are applications that run services responsible for improving the performance of the RAN. Each `xApp` runs a well-defined function to access, control, and monitor `E2 Nodes`. While some xApps may implement latency-sensitive functions, other xApps can also tolerate latencies up to 1 s depending on the tasks they execute. Moreover, xApps can provide services to other xApps. The `Subscription Manager` is responsible for providing E2 subscriptions from xApps to E2 nodes. The `Management`

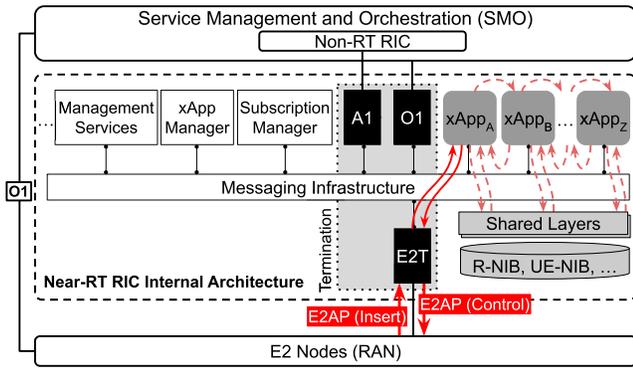[1]https://github.com/LABORA-INF-UFG/paper-GGAMECK-2023

Fig. 1.　O-RAN architecture.

`Services` includes several services provided by the Near-RT RIC to the xApps and E2 nodes, such as routing management, alarm notification service, logging, and E2 node management.

The `A1` termination also shown in Fig. 1 provides an interface between `Non-RT RIC` and `Near-RT RIC` to exchange information about policies and machine learning models that are executed by xApps. The `O1` termination represents the interface between the `Service Management and Orchestration` (`SMO`) framework with network functions and E2 nodes. This termination enables the `SMO` to execute Fault, Configuration, Accounting, Policy, and Security (FCAPS) operations on those network functions and E2 nodes. All O-RAN components are expected to support the `O1` termination when exchanging messages with `SMO`. The `SMO` is responsible for managing and orchestrating the entire RAN, relying on the Non-RT RIC for RAN optimization, and the `O1` termination for RAN instrumentation. Moreover, the `SMO` is in charge of managing and orchestrating any cloud infrastructure in use [4]. The `E2T` is responsible for connecting the `Near-RT RIC` with one or more `E2 Nodes` using the E2 Application Protocol (E2AP) and E2 Service Model (E2SM) which are described ahead. The `Messaging Infrastructure` element corresponds to the underlying messaging subsystem employed by all the Near-RT RIC components to exchange information [9]. The `Shared Layers` correspond to the Shared Data Layer (SDL) and Shared Time-Series Layer (STSL), which are high-speed interfaces for accessing data storage shared by several stateless Near-RT RIC components and xApps. `R-NIB` and `UE-NIB` are examples of databases that store network information about the RAN and User Equipment (UE), respectively.

According to the O-RAN specifications [4], RAN nodes (i.e., DUs, CUs, or O-RAN-compliant LTE eNBs) are `E2 Nodes` controlled through the E2 interface. This interface allows the Near-RT RIC to control procedures and functionalities of those nodes and is logically organized in two parts: E2AP and E2SM. E2AP enables the communication between the Near-RT RIC and E2 nodes, which provide four services to the Near-RT RIC: Report, Insert, Control, and Policy [9]. The Report service allows xApps to subscribe to E2 nodes to receive information about specific RAN events. The Insert service also provides RAN information to xApps, but also allows xApps to configure E2 nodes through control messages.

The Control service allows the Near-RT RIC and xApps to send control messages to E2 nodes, which can initiate new procedures or resume a previously suspended procedure associated with that E2 node. Such a procedure should specify exactly and completely the functional behavior of a given E2 node. Finally, the Near-RT RIC uses the Policy service to establish policy-driven monitoring and control the behavior of the corresponding E2 node [10]. These services can be combined in different ways to implement E2SMs. An E2SM [9] can be described as a contract between xApps and the RAN functions on E2 nodes. Each RAN function exposed by a given E2SM allows the Near-RT RIC to monitor, suspend, stop, override or even control the behavior of the RAN on that E2 node. Therefore, xApps and their corresponding E2 nodes must implement the same E2SM definitions to communicate and control the RAN.

The E2AP Insert service is the most latency-sensitive loop of the Near-RT RIC. This loop starts when a given E2 node sends a message to the `E2T` using the E2AP (Insert) service. The `E2T`, in its turn, delivers that message to the corresponding `xApp`. This `xApp` then processes the message and sends back the corresponding reply using the E2AP Control service. Moreover, the control loop can involve access to `Shared Layers` (i.e., SDL or STSL), it may also have to access one or more databases (e.g., `R-NIB` and `UE-NIB`) and even other `xApps`, as illustrated in Fig. 1. We note that in some use cases, such as the O-RAN Signaling Storm Protection [11], it is essential to deliver the RIC control message back to the E2 node within the 10 ms threshold. In this example use case, the E2 node suspends the current procedure execution and waits for the corresponding reply from an E2AP (Control) service to resume its operation. Although an E2 node can apply a default action when the 10 ms threshold is exceeded, this usually implies on performance penalties for the RAN, as well as the application of further optimization procedures. In this work, we propose to disaggregate and distribute the Near-RT RIC components to guarantee the latency restrictions of the time-sensitive xApps, which execute under a control loop limit of 10 ms.

The O-RAN Software Community (OSC) provides an open-source reference implementation [12] of the Near-RT RIC. This implementation also functions as a baseline that allows the community to explore alternative deployments, including, for instance, replication, among other innovations. The O-RAN Alliance chose Kubernetes as the tool for orchestrating the OSC components, which is presently becoming common practice for mobile network systems that are designed to run on cloud-native infrastructures [13]. The SMO framework must be particularly aware of any alternative configuration so that it can dynamically (1) detect any latency violation (e.g., latency above the limit in the critical loop); (2) redeploy components to solve the violations; (3) re-orchestrate the Near-RT RIC components; and, (4) reconfigure all involved elements. As illustrated in Fig. 1, the `SMO` communicates with all RAN elements, including the `E2 Nodes`, through the `O1` interface that offers the necessary means to reconfigure and reorganize the cluster of E2 nodes that each `E2T` and `xApps` are responsible for. Current SMOs do not fully implement all

tasks in a coordinated manner. We claim one of the contributions of the current work is to present an SMO extension that can provide all necessary capabilities to accomplish those tasks coherently. Another contribution is related to task (2), which comprises the resource allocation problem, for which we formulate and propose optimal and heuristic strategies.

III. RELATED WORK

This section presents an overview of related work that also has a focus on the RIC architecture. In a small RAN with a few dozen or less E2 nodes, it may be possible to deploy a single Near-RT RIC that satisfies the latency-sensitive control loop of the corresponding xApps. However, larger RANs with hundreds or even thousands of E2 nodes demand a different approach. Dryjanski and Kliks [7] present two options for implementing the Near-RT RIC: centralized and distributed. In the centralized option, every E2 node (i.e., the whole gNB or eNB) is handled by the same and only Near-RT RIC, which can take unified decisions for an individual E2 node and globally optimize operations. In the distributed option, each E2 node type (i.e., O-CU, O-DU, or O-eNB) is handled by a specialized instance of the Near-RT RIC that allows optimizing these individual types of E2 nodes. The authors discuss the impact of these two implementation options in the design of the E2 interface. However, they do not tackle the distribution and replication of the Near-RT RIC components.

Singh and Nguyen [14] propose a framework called O-RANFed to deploy and optimize a set of Federated Learning (FL) tasks that provide 5G slicing services in the context of the O-RAN specifications. To be precise: the authors introduce a theoretical model of the RIC architecture with support for FL. Moreover, the authors present an optimization model for the problem of local learning selection and resource allocation. The performance of FL improves with modeling and training done in every iteration. However, some details of the proposed FL-supporting RIC architecture are missing.

Huff et al. [15] discuss and evaluate techniques to make the RIC fault-tolerant while preserving high scalability. The fundamental assumption of this work is that traditional replication techniques cannot sustain high throughput and low latency as required by RAN elements. The authors propose techniques that use state partitioning, partial replication, and fast re-route with role awareness to decrease the overhead. Moreover, the authors implemented the fault tolerance techniques as a library called RIC Fault Tolerance (RFT) considering a distributed RIC, but do not deal with the problem of the placement of RIC components in a disaggregated virtual infrastructure.

D'Oro et al. [8] introduce a novel orchestration framework called OrchestRAN for the Non-RT RIC. OrchestRAN allows network operators to specify high-level control and inference objectives. The orchestrator defines the optimal set of data-driven algorithms and their execution locations to achieve intents specified by the network operators. The work assumes that a complete instance of the Near-RT RIC is deployed to control each cluster of E2 nodes. Moreover, the authors show that the intelligence orchestration problem in O-RAN is NP-hard and design low-complexity approaches to support real-world applications. A prototype of OrchestRAN was

TABLE I
SUMMARY OF RELATED WORK

Works	Placement type*	Problem formulation	Real-world experiments	Approach
[7]	C/D_1	✗	✗	Conceptual
[13]	D_1	✓	✗	Federated Learning
[14]	C/D_1	✗	✓	Fault tolerance
[8]	C/D_1	✓	✓	Orchestration framework
[15]	C	✗	✓	Architectural
[16]	C/D_1/D_2	✗	✗	Architectural
RIC-O	C/D_1/D_2	✓	✓	Optimal flexible placement

*Placement type: C - Centralized D_1 - Distributed D_2 - Disaggregated

implemented and tested at scale on Colosseum, i.e., the world's largest wireless network emulator with hardware in the loop.

Schimidt et al. [16] present FlexRIC, a software development kit to build specialized service-oriented controllers. FlexRIC has a centralized modular architecture with a minimal footprint and is designed for extensibility. The authors validate FlexRIC building implementations for two use cases: (i) a recursive RAN controller to allow multiple tenants to concurrently control and operate their services in a shared infrastructure over heterogeneous 5G networks, and (ii) a Software Defined RAN (SD-RAN) controller supplying programmability for RAN slicing with multi-radio technology, and flow-based traffic control aiming at low-latency communications.

Balasubramanian et al. [6] disaggregate the traditional monolithic control plane of the RAN architecture. The authors introduce a Near-RT RIC platform that decouples the control and data planes of the RAN. The motivation of the project is to work towards intelligent and continuously evolving radio networks by fostering network openness and empowering network intelligence with AI-enabled applications. The authors provide a functional and software architecture of the Near-RT RIC and discuss related design challenges. Moreover, they elaborate on how the Near-RT RIC can enable network optimization in 5G for the dual-connectivity use case using machine learning control loops. In this context, the Near-RT RIC architecture design is generic, providing several options for its implementation and deployment.

Other works focus on developing xApps and rApps. For example, Cao et al. [17], [18] propose an intelligent user access control scheme with Deep Reinforcement Learning (DRL). A federated DRL-based method is proposed, with a global model server running on the RIC that updates the distributed deep Q-networks (DQNs) parameters to optimize the performance of DQNs trained by UE. Johnson et al. [19] introduce NexRAN, a use case of the Powder mobile and wireless research platform, allowing closed-loop control of an approach for O-RAN slicing. O-RAN slicing, in this case, is implemented in the srsRAN open-source mobility stack and is exposed through a custom service model to xApps, executing on a Near-RT RIC.

Table I summarizes relevant works discussed in the literature and our RIC-O proposal. We classify the works considering four key indicators. The first indicator is related to the place-
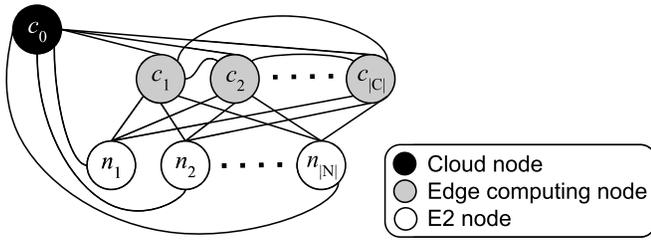
Fig. 2.   Overlay communication network among E2 nodes and CNs.

Fig. 3.   Summarized version of the Near-RT RIC architecture.

ment type, i.e., centralized, distributed, and disaggregated. The second indicator considers the availability of formalization of the placement problem. The third indicator shows which works present real-world experiments evaluation. The last indicator lists the focus of each work.

IV. SYSTEM MODEL, PROBLEM FORMULATION, AND THE PROPOSED STRATEGY

In the following, we describe the system model, formulate the problem, and describe the optimal and heuristic strategies proposed to solve the problem.

A. System Model

We assume a Radio Access Network that follows the O-RAN specifications and is composed of a set $N = \{n_1, n_2, \ldots, n_{|N|}\}$ of E2 nodes and a set $C = \{c_1, c_2, \ldots, c_{|C|}\}$ of edge computing nodes (CNs), where each CN is a host consisting of general purpose hardware characterized by its processing capability (c_m^{Proc}), memory (c_m^{Mem}), and storage (c_m^{Sto}) resources. Moreover, we consider a node c_0 representing a cloud computing infrastructure (e.g., a large data center) with unlimited processing, memory, and storage resources but positioned far from the E2 nodes. We propose disaggregating the Near-RT RIC so that different components can run on various nodes. The node c_0 and each CN $c_m \in C$ can run any disaggregated RIC function as long as it does not exceed its resources. To represent the overlay communication network among all nodes, we define a graph $G = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = \mathcal{V}_C \cup N$ representing the nodes (where $\mathcal{V}_C = \{c_0\} \cup C$), and $\mathcal{E} = \{e_{ij}; v_i, v_j \in \mathcal{V}\}$ representing the set of links, where each link is characterized by its latency $e_{i,j}^{Lat}$. If $v_i = v_j$, then $e_{i,j}^{Lat} = 0$, which is relevant for different components running on the same computing node. As illustrated in Fig. 2, c_0 and each CN $c_m \in C$ can communicate with each other, while each E2 node $n_i \in N$ can communicate with c_0 or any CN $c_m \in C$. Therefore, the paths starting at E2 nodes and ending at computing nodes (that run RIC components) are defined over this graph G.

We assume that a Near-RT RIC can be disaggregated and distributed into a certain number of component groups. Moreover, the specific number depends on implementation aspects, such as the resource demand of each component. For example, in this work, we considered the Near-RT RIC implementation from OSC, and we chose to disaggregate it into five main groups of components: Near-RT RIC Management (RIC_Man), E2T, SDL/STSL, NIBs, and xApps, as illustrated in Fig. 3. It is relevant to highlight the model is flexible, demanding only simple adjustments to attend to
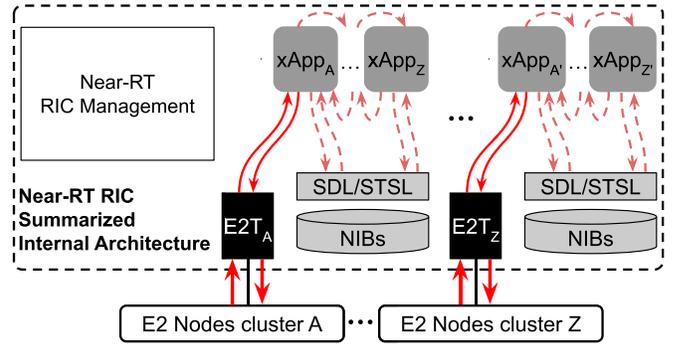
a different number of groups. Each E2 node $n_i \in N$ is connected to an E2T component, which serves a set of xApps $A = \{xApp_1, xApp_2, \ldots, xApp_{|A|}\}$. While the control loop of a latency-sensitive xApp must be monitored per E2 node $(n_i \in N)$, we are also interested in creating clusters of E2 nodes to minimize the replication of Near-RT RIC components, as illustrated in Fig. 3. The red lines with arrows in this figure represent the control loop that each latency-sensitive xApp establishes with each E2 node. Given the computing nodes where the Near-RT RIC components are running and the graph from Fig. 2, it is possible to compute the round-trip latency starting from an E2 node $n_i \in N$, going through an E2T up to an xApp (and possibly other RIC components), and back through the same path.

For each E2 node $n_i \in N$, there are four atomic (or indivisible) groups: RIC_Man, E2T, SDL/STSL, and NIBs, the placement of which is represented by a tuple $p = (r, t, s, d)$. We also employ the following auxiliary variables: $\mathcal{R}_{c_m} = \{0, 1\}$ that indicates if CN $c_m \in \mathcal{V}_C$ runs at least an instance of RIC_Man, $\mathcal{T}_{c_m} = \{0, 1\}$ that indicates if CN $c_m \in \mathcal{V}_C$ runs at least an instance of E2T, $\mathcal{S}_{c_m} = \{0, 1\}$ that indicates if CN $c_m \in \mathcal{V}_C$ runs at least an instance of SDL/STSL, and $\mathcal{D}_{c_m} = \{0, 1\}$ that indicates if CN $c_m \in \mathcal{V}_C$ hosts at least one NIB instance. For example, the tuple $p = (c_0, c_1, c_1, c_1)$ represents RIC_Man running on c_0 and the other atomic groups running on c_1, i.e., $\mathcal{R}_{c_0} = 1$, $\mathcal{T}_{c_1} = 1$, $\mathcal{S}_{c_1} = 1$, and $\mathcal{D}_{c_1} = 1$. Finally, we define a set of configurations $P = \{p_1, p_2, \ldots, p_{|P|}\}$ that lists all combinations of atomic groups running on the available computing nodes. However, the group of xApps can be further disaggregated and distributed into multiple computing nodes and, therefore, is not part of this configuration.

B. Problem Formulation

The objective function minimizes the total cost to run a disaggregated and distributed Near-RT RIC. The cost has two parts, one fixed and another variable. The fixed cost corresponds to activating or leasing a CN, which is paid independently of the number of consumed resources. While the variable cost depends on the number of Near-RT RIC components running on a CN. Each computing node may be assigned a fixed cost, a variable cost, or both.

Let the decision variable $x_i^p \in \{0, 1\}$ represent the choice of configuration $p \in P$ to serve E2 node $n_i \in N$. Let $U_{c_m}^p = \{0, 1\}$ indicate if CN $c_m \in \mathcal{V}_C$ is used in

configuration $p \in P$. The activation (or leasing) of CN $c_m \in \mathcal{V}_C$ is given by:

$$\mathcal{U}_{c_m} = \begin{cases} 1, & \text{if } \sum_{n_i \in N} \sum_{p \in P} (x_i^p \, U_{c_m}^p) > 0 \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

The fixed cost is defined as:

$$\Phi_{Fix} = \sum_{c_m \in \mathcal{V}_C} (\mathcal{U}_{c_m} F(c_m)), \quad (2)$$

where $F(c_m) \in \mathbb{R}$ represents the cost to activate CN $c_m \in \mathcal{V}_C$. While the fixed cost is less common in public data centers, it was adopted in the model to increase flexibility, allowing the representation of private infrastructures as well as configurations such as bare-metal services.

Let the decision variable $y_{i,m}^a \in \{0,1\}$ represent the choice of xApp $a \in A$, running on $c_m \in \mathcal{V}_C$, to control the E2 node $n_i \in N$. The utilization of CN $c_m \in \mathcal{V}_C$ to run xApp $a \in A$ is given by:

$$\mathcal{A}_{c_m}^a = \begin{cases} 1, & \text{if } \sum_{n_i \in N} y_{i,m}^a > 0 \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

The variable cost is defined as:

$$\Phi_{Var} = \sum_{c_m \in \mathcal{V}_C} \Big(\mathcal{R}_{c_m} \mathcal{C}_R(c_m) + \mathcal{T}_{c_m} \mathcal{C}_E(c_m) + \mathcal{S}_{c_m} \mathcal{C}_S(c_m) \\ + \mathcal{D}_{c_m} \mathcal{C}_D(c_m) \Big) + \sum_{a \in A} \mathcal{A}_{c_m}^a \mathcal{C}_{A_a}(c_m), \quad (4)$$

where $\mathcal{C}_R(c_m)$, $\mathcal{C}_E(c_m)$, $\mathcal{C}_S(c_m)$, $\mathcal{C}_D(c_m)$, and $\mathcal{C}_{A_a}(c_m)$ represent the cost of running RIC_Man, E2T, SDL/STSL, NIBs, and xApp $a \in A$, respectively, on a given CN $c_m \in \mathcal{V}_C$.

The objective function for minimizing the total cost is finally defined as:

$$minimize \quad \Phi_{Fix} + \Phi_{Var}. \quad (5)$$

Next, we present the constraints of the problem. For each E2 node $n_i \in N$, exactly a single configuration $p \in P$ must be selected:

$$\sum_{p \in P} x_i^p = 1, \qquad \forall n_i \in N. \quad (6)$$

The latency-sensitive control loop has two mandatory segments. The latency of the first one is measured between the E2 node $n_i \in N$ and CN $c_m \in \mathcal{V}_C$, which runs the corresponding E2T:

$$\mathcal{L}_i^{c_\mathcal{T}} = \sum_{c_m \in \mathcal{V}_C} (\delta(x_i^p) \mathcal{T}_{c_m} e_{n_i, c_m}^{Lat}),$$

where $\delta(x_i^p) = \{0,1\}$ indicates if CN $c_m \in \mathcal{V}_C$ runs the E2T that controls the E2 node $n_i \in N$.

Concerning the second segment of the latency-sensitive control loop, its latency is measured between CN $c_m \in \mathcal{V}_C$ that runs E2T (associated with the E2 node $n_i \in N$) and CN $c_{m'} \in \mathcal{V}_C$ that runs xApp $a \in A$ (responsible for controlling the same E2 node $n_i \in N$):

$$\mathcal{L}_i^{c_\mathcal{T}, c_\mathcal{A}} = \sum_{c_m, c_{m'} \in \mathcal{V}_C} (\delta(x_i^p, y_{i,m'}^a) \mathcal{A}_{c_{m'}}^a e_{c_m, c_{m'}}^{Lat}),$$

where $\delta(x_i^p, y_{i,m'}^a) = \{0,1\}$ indicates if CNs $c_m, c_{m'} \in \mathcal{V}_C$ run, respectively, E2T and xApp $a \in A$ responsible for controlling the E2 node $n_i \in N$.

In a latency-sensitive control loop, however, an xApp $a \in A$ (running on CN $c_m \in \mathcal{V}_C$) may need to access an SDL/STSL (running on CN $c_{m'} \in \mathcal{V}_C$) to reach a NIB (running in CN $c_{m''} \in \mathcal{V}_C$). This communication has the following latency:

$$\mathcal{L}_i^{c_\mathcal{A}, c_\mathcal{S}, c_\mathcal{D}} = \psi(a) \sum_{c_m, c_{m'}, c_{m''} \in \mathcal{V}_C} \big(\delta(y_{i,m}^a, x_i^p) \mathcal{S}_{c_{m'}} e_{c_m, c_{m'}}^{Lat} \\ + \delta'(y_{i,m}^a, x_i^p) \mathcal{D}_{c_{m''}} e_{c_{m'}, c_{m''}}^{Lat} \big),$$

where $\psi(a) = \{0,1\}$ indicates if xApp $a \in A$ needs to access SDL/STSL and NIB; $\delta(y_{i,m}^a, x_i^p) = \{0,1\}$ indicates if CNs $c_m, c_{m'} \in \mathcal{V}_C$ run, respectively, xApp $a \in A$ and SDL/STSL; and $\delta'(y_{i,m}^a, x_i^p) = \{0,1\}$ indicates if CNs $c_m, c_{m''} \in \mathcal{V}_C$ run, respectively, xApp $a \in A$ and NIB. These components are related to E2 node $n_i \in N$.

Additionally, in a latency-sensitive control loop, an xApp $a \in A$ (running on CN $c_m \in \mathcal{V}_C$) may need to interact with other xApp $a' \in A$ (running on CN $c_{m'} \in \mathcal{V}_C$), which generates the following latency:

$$\mathcal{L}_i^{c_\mathcal{A}, c_{\mathcal{A}'}} = \psi'(a) \sum_{c_m, c_{m'} \in \mathcal{V}_C} (\delta(y_{i,m}^a, y_{i,m'}^{a'}) \mathcal{A}_{c_{m'}}^{a'} e_{c_m, c_{m'}}^{Lat}),$$

where $\psi'(a) = \{0,1\}$ indicates if xApp $a \in A$ needs to communicate with another xApp, and $\delta(y_{i,m}^a, y_{i,m'}^a) = \{0,1\}$ indicates if CNs $c_m, c_{m'} \in \mathcal{V}_C$ run, respectively, xApps $a, a' \in A$, which control the E2 node $n_i \in N$.

Considering the potential chain of xApps $A_{ch} \subseteq A$ involved in the control loop of a certain xApp $a \in A$, its total latency is the sum of the four previous equations and cannot exceed the threshold ρ_a:

$$\mathcal{L}_i^{c_\mathcal{T}} + \mathcal{L}_i^{c_\mathcal{T}, c_\mathcal{A}} + \mathcal{L}_i^{c_\mathcal{A}, c_\mathcal{S}, c_\mathcal{D}} \\ + \sum_{a, a' \in A_{ch}} \big(\mathcal{L}_i^{c_\mathcal{A}, c_{\mathcal{A}'}} + \mathcal{L}_i^{c_{\mathcal{A}'}, c_\mathcal{S}, c_\mathcal{D}} \big) \le \rho_a, \forall n_i \in N, \forall a \in A. \quad (7)$$

The Near-RT RIC components running on a CN $c_m \in \mathcal{V}_C$ must not exceed its processing capability (c_m^{Proc}), memory (c_m^{Mem}), and storage (c_m^{Sto}) resources:

$$\mathcal{R}_{c_m} \mathcal{R}^{Proc} + \mathcal{T}_{c_m} \mathcal{T}^{Proc} + \mathcal{S}_{c_m} \mathcal{S}^{Proc} + \mathcal{D}_{c_m} \mathcal{D}^{Proc} \\ + \sum_{a \in A_{c_m} \subseteq A} (\mathcal{A}_{c_m}^a \mathcal{A}^{a, Proc}) \le c_m^{Proc}, \qquad \forall c_m \in \mathcal{V}_C, \quad (8)$$

$$\mathcal{R}_{c_m} \mathcal{R}^{Mem} + \mathcal{T}_{c_m} \mathcal{T}^{Mem} + \mathcal{S}_{c_m} \mathcal{S}^{Mem} + \mathcal{D}_{c_m} \mathcal{D}^{Mem} \\ + \sum_{a \in A_{c_m} \subseteq A} (\mathcal{A}_{c_m}^a \mathcal{A}^{a, Mem}) \le c_m^{Mem}, \qquad \forall c_m \in \mathcal{V}_C, \quad (9)$$

$$\mathcal{R}_{c_m} \mathcal{R}^{Sto} + \mathcal{T}_{c_m} \mathcal{T}^{Sto} + \mathcal{S}_{c_m} \mathcal{S}^{Sto} + \mathcal{D}_{c_m} \mathcal{D}^{Sto} \\ + \sum_{a \in A_{c_m} \subseteq A} (\mathcal{A}_{c_m}^a \mathcal{A}^{a, Sto}) \le c_m^{Sto}, \qquad \forall c_m \in \mathcal{V}_C, \quad (10)$$

where $\mathcal{R}^{\bullet}, \mathcal{T}^{\bullet}, \mathcal{S}^{\bullet}, \mathcal{D}^{\bullet}$, and $\mathcal{A}^{a,\bullet}$ represent processing, memory, and storage demands for RIC_Man, E2T, SDL/STSL, NIBs, and xApp $a \in A_{c_m} \subseteq A$, respectively.

As discussed in [20] and [21], the problem represented by Equations (5 - 10) is an NP-hard since it corresponds to a mixed integer quadratically constrained problem (MIQCP). Moreover, MIQCP is a subclass of mixed-integer nonlinear programming (MINLP) problems, well-known as an NP-hard. The proof shows that since mixed-integer linear programming (MILP) is a MINLP subclass and MILP is NP-hard, MINLP is NP-hard by a polynomial reduction from the Boolean Satisfiability Problem (SAT).

C. Heuristic Strategy

A MIQCP such as the one represent by Equations (5 - 10) can be optimally solved using traditional solvers, e.g., IBM ILOG CPLEX. However, this problem has a large number of decision variables that can be estimated as $(|N| \cdot |\mathcal{V}_C|)^{comp} + (|N| \cdot |\mathcal{V}_C|)^{|A|}$, where $comp$ represents the number of Near-RT RIC components. For example, a configuration with $|N| = 100$ E2 nodes, $comp = 4$, $|\mathcal{V}_C| = 5$ computing nodes, and $|A| = 5$ xApps involves more than $3 \cdot 10^{13}$ optimization variables. Therefore, due to the computational time and the amount of resources required, which are extremely high, solving the problem optimally is impractical, even for small instances, as we present in Section VII.

To deal with large instances of the problem and quickly provide solutions for RIC-O reorchestration requests, the efficient heuristic strategy described as Algorithm 1 is proposed. The heuristic starts with a feasible solution and then seeks to improve cost efficiency through a greedy strategy. The heuristic takes as input the same parameters used in the optimal strategy and produces as output the placement of the Near-RT RIC components across CNs, including xApps. The output is represented by $Places[n_i, \text{'Near-RT RIC component'}]$, where $n_i \in N$ identifies the E2 node that is associated with a 'Near-RT RIC component', which can be RIC_Man ($\mathcal{R}_i$), E2T ($\mathcal{T}_i$), SDL/STSL ($\mathcal{S}_i$), NIBs ($\mathcal{D}_i$), or xApp $a_i \in A_i$. Moreover, let $(Z_i, \preceq) = \mathcal{T}_i \cup A_i \cup \mathcal{S}_i \cup \mathcal{D}_i$ be the ordered set of components involved in the latency-sensitive control loop, as described in Section IV-A and illustrated in Fig. 3. Next, we describe the algorithm.

Initially, for each $n_i \in N$, the heuristic strategy seeks to place every Near-RT RIC component $\zeta_i \in (Z_i, \preceq)$ at the closest CN $c_m \in \mathcal{V}_C$ with enough available resources, as shown in Algorithm 1 (lines 1–3). This procedure corresponds to the best solution regarding latency, but it ignores the cost of the CNs and causes unnecessary replication of Near-RT RIC components, e.g., E2T and xApps, which also increases cost. Next (lines 4, 5), in $CNcost[\hat{\zeta}_i]$, the algorithm sorts in decreasing order of cost the list of CNs that are available for running each Near-RT RIC component $\hat{\zeta}_i \in \mathcal{R}_i \cup (Z_i, \preceq)$. RIC_Man is included because it is a Near-RT RIC component, despite not being involved in the latency-sensitive control loop. The last part of the algorithm (lines 6–9) focuses on improving cost efficiency. First (line 7), the RIC_Man component is placed on the cheapest CN $n_i \in N$ that has enough available resources.

Algorithm 1 Heuristic Strategy

Input : The same input of the optimal strategy
Output: Placement of the Near-RT RIC components

1   **forall** $n_i \in N$ **do**
2   **forall** $\zeta_i \in (Z_i, \preceq)$ **do**
3       $Places[n_i, \zeta_i] \leftarrow \text{closestCN}(n_i, \zeta_i)$

4   **forall** $\hat{\zeta}_i \in \mathcal{R}_i \cup (Z_i, \preceq)$ **do**
5     $CNcost[\hat{\zeta}_i] \leftarrow \text{sortByDecreasingCost}(\hat{\zeta}_i, \mathcal{V}_C)$

6   **forall** $n_i \in N$ **do**
7     $Places[n_i, \mathcal{R}_i] \leftarrow \text{rePlace}(\mathcal{R}_i, CNcost[\mathcal{R}_i])$
8   **forall** $\zeta_i \in (Z_i, \preceq)$ **do**
9       $Places[n_i, \zeta_i] \leftarrow \text{rePlace}(\zeta_i, CNcost[\zeta_i])$

10   **return** $Places[\bullet, \bullet]$

RIC_Man tends to be placed on c_0 since this tends to be the cheapest CN, and, generally, function `rePlace` determines that node has enough available resources. Moreover, a single RIC_Man is probably able to serve all E2 nodes. Last (lines 8, 9), the algorithm seeks to find the cheapest CN $n_i \in N$ for each Near-RT RIC component $\zeta_i \in (Z_i, \preceq)$. Therefore, in addition to checking CN resource availability, function `rePlace` also checks compliance with the latency-sensitive control loop. These final steps of the algorithm also tend to reduce the number of instances of Near-RT RIC components because a single component may be able to ensure the latency is within the required threshold for multiple E2 nodes.

The proposed heuristic strategy solves the problem in polynomial time and produces satisfactory results regarding the placement of the Near-RT RIC components, as we present in Section VII. To be precise, we can represent the asymptotic complexity of Algorithm 1 in the worst case as $\mathcal{O}((comp + |A|)^2 \cdot |N| \cdot |\mathcal{V}_C|)$. In the worst case, all CNs can host any component of the Near-RT RIC for all E2 nodes at the same cost. In this scenario, every run of Algorithm 1 must check the possibility of placing every component on every CN. Regarding the convergence of Algorithm 1, we cannot guarantee convergence to an optimal solution since the heuristic is based on systematic progression. However, as our heuristic starts from a feasible solution and iteratively makes changes only to achieve another feasible solution, we can ensure that Algorithm 1 always returns a good quality viable solution, as shown in Section VII.

V. RIC Orchestrator

This section describes the architecture of the RIC-O which is responsible for the dynamic and efficient placement of the disaggregated Near-RT RIC components. The architecture of the RIC-O is shown in Fig. 4, along with some of the RIC components required to describe the orchestrator. RIC-O was designed considering components that run in both Non-RT RIC and Near-RT RIC. The `RIC-O Optimizer` and `RIC-O Deployer` run on the Non-RT RIC, while the `RIC-O Triggers` execute on the `Metrics Server` of the
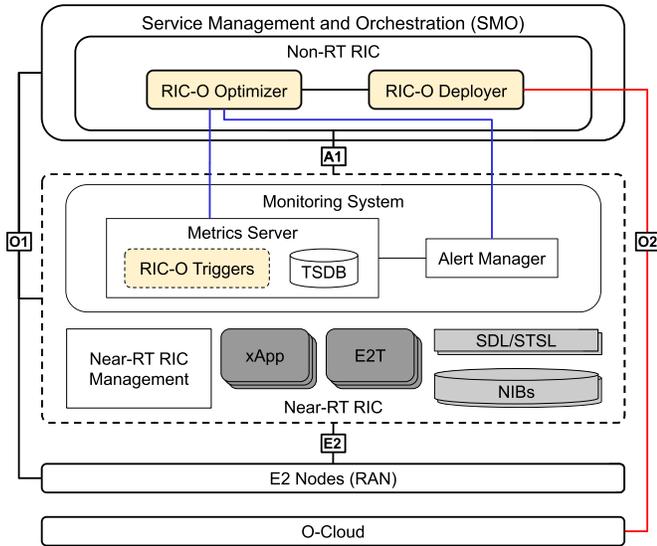
Fig. 4. RIC-O architecture.

Near-RT RIC. The `Monitoring System` is in charge of monitoring resource usage of the `Near-RT RIC` components, `E2 Nodes`, and `O-Cloud` infrastructure, in addition to throwing alerts whenever it detects threshold violations in the latency-sensitive control loop.

The `Metrics Server` is a monitoring module that collects metrics information from all `xApp` and `E2T` instances running on the `Near-RT RIC` platform. Example metrics include CPU utilization, memory usage, and the communication latency up to the monitored instances. The latencies to access SDL/STSL shared layers and databases are also collected by the `Metrics Server`. All metrics are stored in a time-series database (`TSDB`) for further processing and reporting. The `RIC-O Triggers` component consists of predefined alert rules (e.g., the latency is above a given threshold) upon the occurrence of which an alert is generated. As the `Metrics Server` detects some metric violation that is associated with some trigger, it notifies the `Alert Manager` that a metric has been violated. The `Alert Manager`, in its turn, sends a message to the `RIC-O Optimizer` requesting a new round of optimization to reconfigure the system so that it can satisfy the metric that has been violated.

When a request is delivered to the `RIC-O Optimizer`, it fetches the current metric measurements from the `Metrics Server`, executing the optimization strategies described in Section IV. The heuristic and optimal optimization strategies run in parallel. The heuristic strategy should find a new solution quickly. However, the optimal strategy execution continues even if the heuristic completes earlier. The solution generated by the optimal strategy is only applied if it is superior to the heuristic and if it becomes available before a new notification of metric violation. The `RIC-O Optimizer` sends the configuration of the computed solution to the `RIC-O Deployer`, which redeploys the `Near-RT RIC` components accordingly. The redeployment is only executed if the `RIC-O Optimizer` finds a solution that differs from the one in use. Lastly, the `RIC-O Deployer` uses the standardized O2

interface defined by O-RAN to exchange messages with the `O-Cloud` element to effectively apply the new placement.

VI. IMPLEMENTATION

We implemented RIC-O as a proof-of-concept prototype to validate and evaluate our proposal. The prototype is described in this section, along with the testbed used for its evaluation. Both the `RIC-O Optimizer` and `RIC-O Deployer` were implemented using the Python language. The `Monitoring System` component comprises the monitoring subsystem of the Near-RT RIC platform [22] that runs on Kubernetes (K8S). The Near-RT RIC uses Prometheus to collect metrics, and the `Metrics Server` corresponds to the main Prometheus server. Metrics are collected at the pod and cluster levels and stored as time series data in the internal Prometheus database (`TSDB`). Prometheus was a natural choice to collect metrics, since it is the native K8S solution for data collection and is part of the OSC Near-RT RIC platform.

We also employed the `Alert Manager` component from Prometheus to handle alerts issued by different client applications (e.g., `Metrics Server`). Alert handling may include deduplicating, grouping, silencing, inhibiting, and routing alerts to other endpoints. The `Alert Manager` notifies the `RIC-O Optimizer` of the need for starting a new optimization round when a given RIC-O trigger detects that a latency threshold has been violated. The `RIC-O Triggers` element was implemented using the Prometheus Query Language (PromQL). In this context, our prototype monitors CPU, memory, and network latency from K8S system pods, Near-RT RIC pods, and K8S nodes. One of the monitored metrics is the latency-sensitive control loop of the Near-RT RIC, which cannot surpass a given threshold (i.e., above 10 ms for certain control loops). Upon reaching this threshold, a trigger launches the `RIC-O Optimizer` to compute a new placement solution for the Near-RT RIC components. After the optimization is computed, a notification is delivered to `RIC-O Deployer`, which redeploys the Near-RT RIC components based on the outcome of the `RIC-O Optimizer`.

Figure 5 shows the messages exchanged between the RIC-O components, Monitoring System, Near-RT RIC components, and the O-Cloud element while orchestrating the optimization and deployment tasks. Initially, the deployment is executed as a response to the original request coming from the network operator. Subsequently, a re-deploy is executed only after the `Alert Manager` sends a notification to the `RIC-O Optimizer` due to a latency violation in the Near-RT RIC control loop or due to the unavailability of required computing resources in the `O-Cloud` element.

The `Monitoring System` shown in Fig. 5 collects metrics regularly from the `Near-RT RIC Components` and the `O-Cloud` element. These metrics include CPU and memory usage of Near-RT RIC pods, K8S nodes, and their communication latency (messages 1, 2, 3, and 4). The `Monitoring Loop` is executed regularly at every second. Once `Metrics Server` notifies `Alert Manager` that a given RIC-O trigger has been violated, message 5 is delivered to `RIC-O Optimizer`.
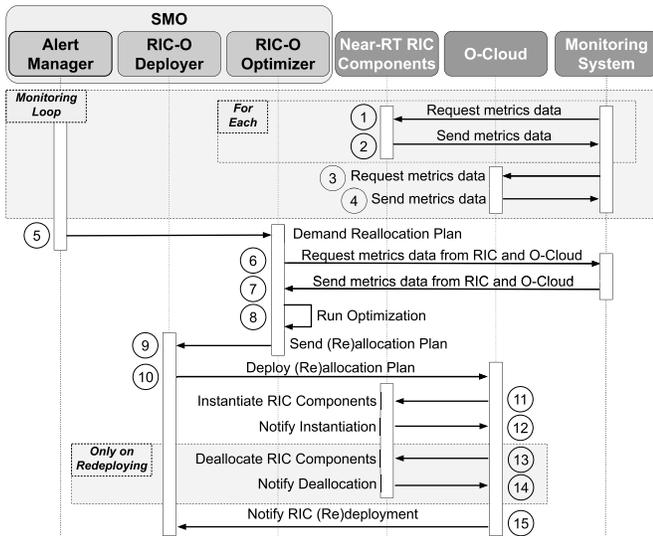
Fig. 5.   Near-RT RIC optimization: messages exchanged.

Fig. 6.   xApp deployment and execution: messages exchanged.

Afterward, the `RIC-O Optimizer` starts a reallocation plan to compute the placement of the `Near-RT RIC Components` that run on the `O-Cloud` element. The `RIC-O Optimizer` then requests metrics information that includes the available computing resources of K8S pods, CNs, and the communication latency between them (messages 6 and 7), which is the information required as input for the optimization strategies. Next, the heuristic and optimal strategies are started and run in parallel (message 8). Once an outcome from the optimization strategies becomes available, it is sent to the `RIC-O Deployer` in message 9. Eventually, if there is enough time for computing the optimal solution, two outcomes are generated. The `RIC-O Deployer`, in turn, starts deploying the new (re)allocation plan in `O-Cloud` through the O2 interface defined by O-RAN (message 10). Near-RT RIC components are instantiated by `O-Cloud`, which is notified when all the components are deployed (messages 11 and 12). In the particular case in which the Near-RT RIC components were previously deployed, `O-Cloud` releases the resources of the Near-RT RIC components that are no longer required (messages 13 and 14). Finally, `O-Cloud` notifies the `RIC-O Deployer` that the deployment has completed (message 15).

We also implemented an extended xApp and an E2 simulator (E2Sim), both coded in C++ language. Our implementation is based on the open-source versions of the Bouncer xApp[2] and the E2Sim[3] from the OSC. OSC's E2 simulator (E2 interface) offers only basic functions to simulate E2 nodes connecting to the E2 termination, enabling simulated end-to-end E2 flows. Therefore, several functions of a RAN stack are not available. Originally, the Bouncer and E2Sim employed the Key Performance Measurement service model (E2SM-KPM) to expose performance measurements of logical functions running on E2 Nodes [23]. However, E2SM-KPM does not support RAN control through the E2AP Control service. Therefore, we implemented the E2SM-RC (RAN Control)
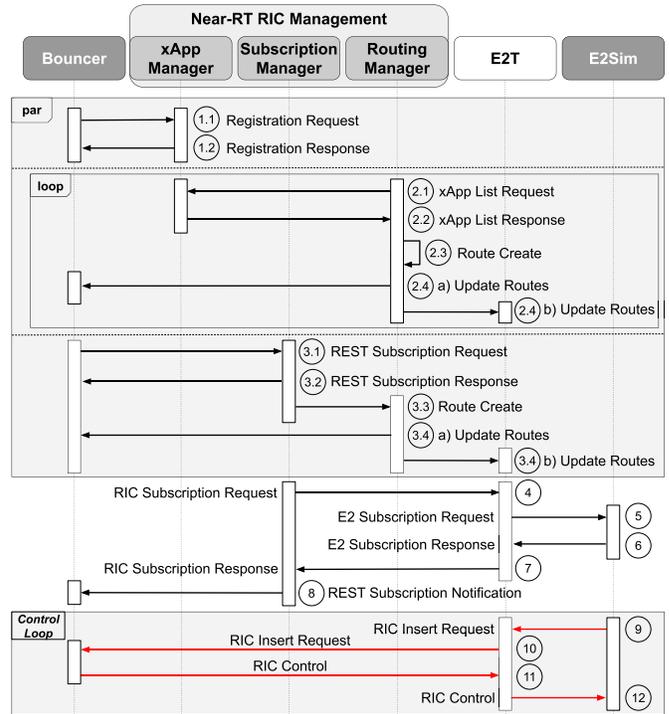
[24] service model, which provides RAN control. Figure 6 shows the messages exchanged between the `Bouncer` xApp, `E2Sim`, and Near-RT RIC components. The goal is to monitor the latency-sensitive control loop between the E2 nodes and the corresponding xApp. We assume that all the Near-RT RIC components and `E2Sim` are already set up and running when `Bouncer` xApp is deployed.

When the `Bouncer` xApp initializes, it registers itself with the Near-RT RIC platform by sending a registration message to `xApp Manager`, which sends back a reply indicating the outcome (messages 1.1 and 1.2). In the meantime, the `Routing Manager` periodically retrieves the list of xApps registered in `xApp Manager` to set up the corresponding routing rules in the Near-RT RIC platform (messages 2.1 and 2.2). Moreover, the `Routing Manager` builds a routing table for all registered xApps (message 2.3) which is then distributed to the `Bouncer` and `E2T` (messages 2.4a and 2.4b). These routes are required to allow transparent and dynamic message delivery to the endpoints concerned in the communication. During the registration, the `Bouncer` xApp also subscribes to the `Subscription Manager` to receive messages from the E2AP Insert service from the RAN (messages 3.1 and 3.2). Upon receiving the subscription request, the `Subscription Manager` notifies the `Routing Manager` to create the corresponding routing rules to deliver those messages to `Bouncer` xApp (message 3.3). Next, the `Routing Manager` distributes the generated routing rules to the respective endpoints (messages 3.4a and 3.4b).

Additionally, the `Subscription Manager` issues a new subscription request (message 4) to the `E2T`, which implements the E2 interface to communicate with E2 nodes.

---

[2]https://gerrit.o-ran-sc.org/r/admin/repos/ric-app/bouncer
[3]https://gerrit.o-ran-sc.org/r/admin/repos/sim/e2-interface

The `E2T` then deals with the subscription communication with the `E2Sim` (messages 5 and 6) according to the subscription that the `Bouncer` xApp has generated in message 3.1. The communication between `E2T` and `E2Sim` is established with the Stream Control Transmission Protocol (SCTP). Messages 7 and 8 notify the `Subscription Manager` and the `Bouncer` xApp about the subscription outcome. At this point, the `Bouncer` xApp is fully deployed on the Near-RT RIC platform, and the communication with `E2Sim` can take place through the `E2T` component.

The `Control Loop` block in Fig. 6 illustrates the latency-sensitive control loop of 10 ms. When a given E2 node issues a RIC Insert Request message to the Near-RT RIC, the E2 node suspends its execution and waits for the corresponding reply message (i.e., RIC Control) to resume its operation. Messages 9-12 illustrate this control loop that needs to complete within the 10 ms threshold. If no RIC Control message is delivered within this threshold time, a default action is executed by the E2 node, which usually implies penalties in the optimization and performance of the RAN. Moreover, we implemented the capability of computing the control loop latency between `E2Sim` and `Bouncer` xApp. In this case, `E2Sim` collects the timestamps both as message 9 is sent and message 12 is received. The latency-sensitive control loop is exported to the Prometheus metrics collector.

VII. EVALUATION

This section presents a performance evaluation of RIC-O both from a theoretical perspective, using analytical modeling (in Subsection VII-A), and from a practical perspective, employing real-world experiments (in Subsection VII-B). In Subsection VII-A, we evaluate the scalability and quality of the solutions found by each approach, considering the processing time, cost, and differences between the optimal and heuristic solutions. In Subsection VII-B, we present the results related to the effective operation of RIC-O (described in Section VI) in a practical environment using our placement solutions integrated with a real-world Near-RT RIC (OSC) running over a traditional orchestration system (Kubernetes).

A. Analytical Modeling

In this part of the evaluation, we assume as RAN topology a next-generation hierarchical network with 512 E2 nodes organized in three main tiers [25]. The top one (Tier-1) is closest to the core and so to the cloud node (i.e., c_0). The one-way latency (in ms) between the tiers is a uniform random choice from the set $\{1, 2, 2, 3, 3\}$, and the one-way latency from Tier-1 to c_0 is 4 ms. We vary the number of edge computing nodes (i.e., $c_m \in C$) along the network tiers and, for each number of CNs, extract an overlay topology equivalent to the one shown in Fig. 2. There is always a cloud node, and all others are edge computing nodes distributed from bottom to top along the hierarchical RAN network as the number of CNs increases. The cloud node has no fixed cost and presents the following variable cost for Near-RT RIC components: two (RIC_Man, E2T), one (SDL/STSL, NIBs), and one (xApp). There are two xApps, and both of them

TABLE II
PARAMETERS OF THE UNDERLAY RAN TOPOLOGY AND REQUIREMENTS OF THE NEAR-RT RIC COMPONENTS

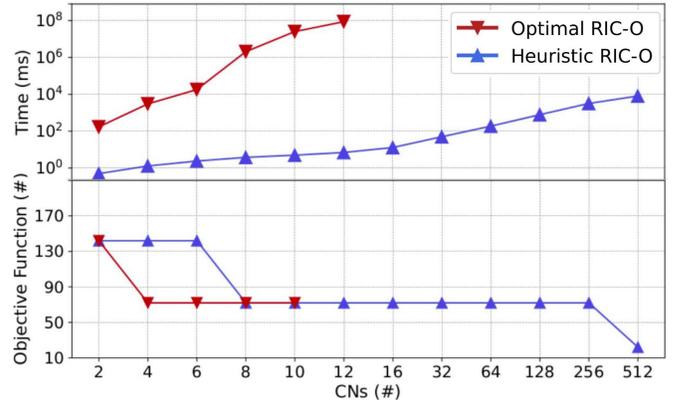| Parameters	Tier-1	Tier-2	Tier-3
Number of E2 nodes	5	20	487
CN fixed cost	10	20	30
CN variable cost: (RIC_Man, E2T), (SDL/STSL, NIBs), (xApps)	4, 2, 1	8, 4, 2	16, 8, 4
c_m^{Proc} (cores)	32	16	8
c_m^{Mem} (GB)	64	32	16
c_m^{Sto} (GB)	256	256	256
Near-RT RIC requirements	$Proc$	Mem	Sto
RIC_Man	4	8	4
E2T	2	4	2
SDL/STSL	2	4	1
NIBs	1	2	50
xApps	1	2	1

Fig. 7. Comparison between the optimal and heuristic strategies in terms of computing time and total cost (objective function).

access the database, i.e., SDL/STSL and NIBs. Additional information about the underlay RAN topology, and Near-RT RIC is presented in Table II.

We first investigate to which extent the complexity of the exact optimization model (Subsection IV-B) limits its scalability. Although our heuristic strategy (Subsection IV-C) is very efficient, it is necessary to investigate the quality of its solutions, i.e., how close to optimal those solutions tend to be. Therefore, we evaluate and compare the optimal and heuristic strategies regarding computing time and objective function as presented in Fig. 7. The computing time is related to the scalability of each strategy, and the objective function, i.e., the total cost, represents the quality of the solution. Moreover, the metrics vary as a function of the number of CNs.

We can observe in Fig. 7 that the optimal strategy scales poorly, being unable to find a solution for a RAN with 12 CNs even after running for an entire day. Therefore, the optimal strategy is rarely helpful for large real-world instances. Nevertheless, the heuristic strategy scales very well and finds a solution for 512 CNs in less than 10 seconds. Concerning the quality of the solutions, as expected, the heuristic strategy cannot always find the optimal solution. However, a decreasing trend can be discerned in the objective function that suggests the heuristic strategy is in the right direction, including in scenarios with a large number of CNs, e.g., 512 CNs. In those
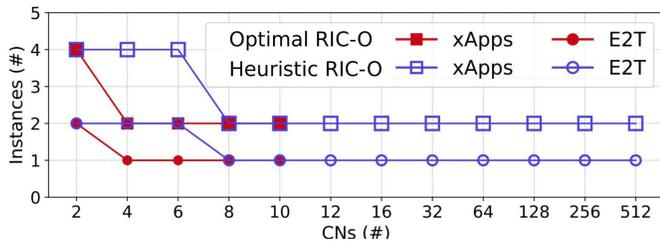
Fig. 8. Comparison between the optimal and heuristic strategies regarding the number of xApps and E2T instances.

Fig. 9. Reaction to a sudden violation of the latency-sensitive control loop requirements.

cases, the cheapest edge computing nodes (i.e., those in Tier-1) are readily available, and the solution improves noticeably.

We further investigate the optimization strategies by evaluating the number of instances of the main Near-RT RIC components involved in the latency-sensitive control loop, i.e., xApps and E2T. As shown in Fig. 8, our heuristic strategy finds the best (i.e., the smallest) number of E2T instances, matching the result obtained by the optimal strategy when this is available and keeping it at the minimum (i.e., one instance) as the number of CNs increases. Moreover, the heuristic strategy finds some non-optimal solutions for the number of xApps instances, e.g., with four and six CNs. This behavior is expected as the heuristic strategy prioritizes Near-RT RIC components involved in the latency-sensitive control loop in the following order: E2T, xApps, SDL/STSL, and NIBs. Since latency optimization and cost optimization are performed in this order and without revisiting previous decisions for improvement, the heuristic strategy potentially finds non-optimal solutions involving xApps, SDL/STSL, or NIBs.

B. Real-World Experiments

Next, we describe experiments run in a scenario with five CNs, which are virtual machines (VMs) with the following configuration: 4 vCPUs, 8 GB RAM, and 50 GB of the virtual disk. One CN represents the cloud node (i.e., c_0), and the others represent the edge computing nodes (i.e., $c_m \in C$). These CNs are worker nodes in a K8S cluster managed by a primary node running a sixth VM with the following configuration: eight vCPUs, 16 GB RAM, and 100 GB of the virtual disk. All VMs are hosted on a DELL PowerEdge M610 server with four Intel Xeon X5660 processors and 192 GB RAM, which runs VMware ESXi 6.7 as the hypervisor. Additional details about the software tools employed in the experiments are available in the public repository of this article. We also employed an E2Sim to represent four E2 nodes that the Near-RT RIC must serve. Overlay and underlay topologies are the same presented in Fig. 2.

To illustrate the orchestration capabilities of RIC-O, we designed two scenarios in which the latency-sensitive control loop is disrupted and show how our proposal acts to bring the Near-RT RIC back to normal operation. In the first scenario, RIC-O must deal with a sudden and high increase in the latency of the path used to serve a certain E2 node. The second scenario is more challenging because RIC-O needs to deal with a CN that becomes unavailable, i.e., any latency-
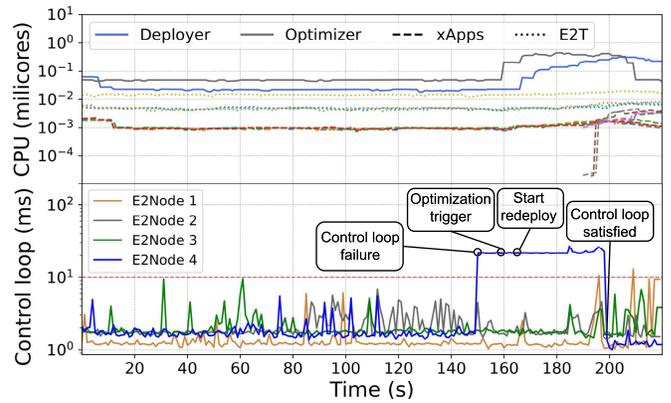
sensitive control loop involving this CN disappears, since the Near-RT RIC components running in it suddenly become inaccessible.

In a real-world RAN, the latency between a pair of nodes may change, for example due to a (re)route decision in the underlying network. Since our underlay network matches the overlay one, we emulate the sudden increase in the latency between the E2 node and its corresponding E2T by reconfiguring the latency in the virtual link connecting these nodes. Figure 9 illustrates the main events occurring along the time in this scenario. This figure shows the status of the control loop between each E2 node and its corresponding xApp. In addition, the figure presents the CPU utilization of some essential software components (i.e., `RIC-O Deployer`, `RIC-O Optimizer`, xApps, and `E2T`), which helps keep track of the actions performed by RIC-O.

As illustrated in Fig. 9, the first scenario is initially in a fully operational state, and the latency-sensitive control loop of each E2 node is satisfied by the Near-RT RIC thanks to the initial orchestration defined by RIC-O. Then, at time instant 150 s, the latency of the control loop from E2 node 4 suddenly increases and remains persistently above 10 ms, as indicated by event *Control loop failure* in the figure. After 10 seconds, the `Monitoring System` considers that the event is a consistent control loop violation and notifies the `RIC-O Optimizer` to compute a new solution, at time instant 160 s, as indicated by the event *Optimization trigger*. The heuristic strategy of the `RIC-O Optimizer` quickly finds a solution and requests `RIC-O Deployer` to apply this new placement nearly 5 seconds later, as indicated by the event *Start redeploy*. The `RIC-O Optimizer` keeps running the optimal strategy thread. Finally, the redeploy of the Near-RT RIC components and reconfiguration of E2 nodes completes at time instant 200 s, as indicated by event *Control loop satisfied*, when the latency-sensitive control loop is again limited to 10 ms.

Figure 9, and also Fig. 10 show a few measurements of the control loop that go above 10 ms. This behavior is related to the underlying operating system and virtualization platform (i.e., hypervisor). A tight threshold of 10 ms is on such a
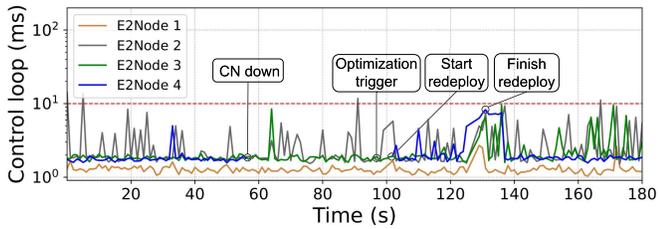
Fig. 10.   Reaction to sudden unavailability of the CN under use.

Fig. 11.   Total cost comparison between the disaggregated RIC-O (optimal and heuristic) and the monolithic Optimal M-RIC.

sensitive scale that even a traditional process scheduler may sometimes cause a small variation. Since fine-tuning those systems is out of the scope of this work, we configured the `Monitoring System` to report only persistent violations of the 10 ms threshold in latency-sensitive control loops.

The second scenario, in which a CN suddenly crashes, represents either a software or hardware failure, or network outage. It is emulated by abruptly forcing a shutdown of the VM running the CN. Figure 10 illustrates the main events occurring along the time in this scenario. Similar to Fig. 9, Fig. 10 also shows the status of the control loop between each E2 node and its corresponding xApp. However, we have not identified relevant information that justified presenting measurements related to Near-RT RIC nor RIC-O components.

As illustrated in Fig. 10, the second scenario also starts from a fully operational state, where the latency-sensitive control loops of all E2 nodes are satisfied by the Near-RT RIC, thanks to the initial orchestration defined by RIC-O. The CN running Near-RT RIC components responsible for serving the E2 node 4 suddenly becomes unavailable, as indicated by the event *CN down* in the figure. In this case, the latency-sensitive control loop of this E2 node is totally disrupted, i.e., there are no more measurements of the control loop. After 50 seconds, the `Monitoring System` detects the problem and notifies the `RIC-O Optimizer` to compute a new solution, as indicated by the event *Optimization trigger*. This time interval for reporting the problem may seem long, but it is the default K8S policy for detecting worker node unavailability. Again, after nearly 5 seconds, the heuristic strategy of `RIC-O Optimizer` finds a solution and requests `RIC-O Deployer` to apply the new placement, as indicated by the event *Start redeploy*. Finally, the redeployment of the Near-RT RIC components finishes, and the latency-sensitive control loop of E2 node 4 is reestablished at time instant 130 s, as indicated by the event *Finish redeploy*.

C. Disaggregated Near-RT RIC Benefits

We present two benchmark analyzes in this section. First, we compare the disaggregated Near-RT RIC with the conventional monolithic OSC version of the Near-RT RIC. Second, we show the advantages of disaggregating two open-source implementations of Near-RT RIC, namely OSC and SD-RAN [26], compared with their monolithic counterparts. We added a new evaluation approach (Optimal M-RIC) to compute the optimal deployment of the monolithic Near-RT RIC and perform its automatic deployment. Actually, this
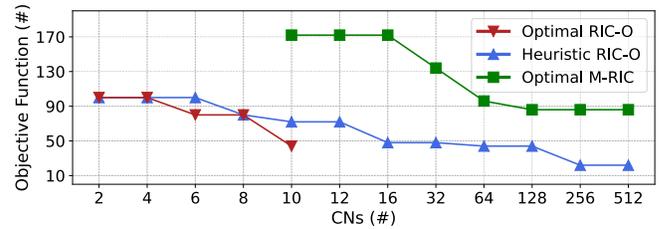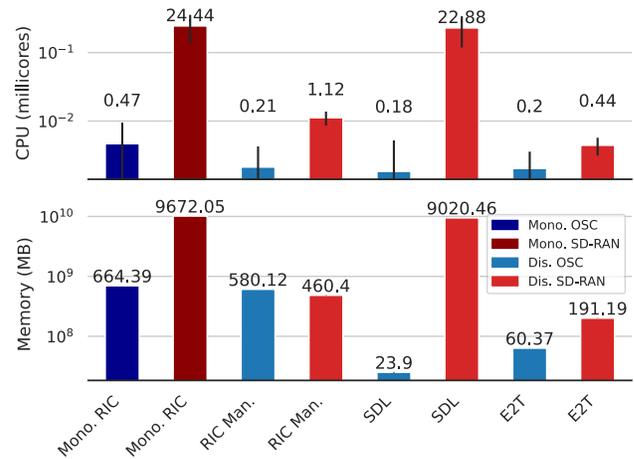

Fig. 12.   Comparison of resource usage between Near-RT RIC from OSC and SD-RAN.

approach is a variant of our proposal in which the Near-RT RIC cannot be disaggregated. Several parts of the optimal orchestration process would need human intervention without the Optimal M-RIC. Moreover, this comparison illustrates the flexibility of our model regarding the number of components from the Near-RT RIC.

Figure 11 compares the disaggregated RIC-O (optimal and heuristic) and the monolithic Optimal M-RIC regarding total cost, i.e., the objective function value. We employed the same configuration described at the beginning of Section VII-A. However, we assume that the CN resources (i.e., processing, memory, and storage) are only partially available for the Near-RT RIC. This scenario can be common in shared CNs that are used to run other workloads, such as components of a disaggregated RAN stack and Multi-access Edge Computing (MEC) applications. Given this limitation of the available resources, the Optimal M-RIC becomes feasible only from 10 CNs, while the Optimal RIC-O is able to operate from 2 CNs thanks to the minimal component replication. As expected, deploying a monolithic Near-RT RIC has a total cost noticeably higher than disaggregated Near-RT RIC.

In our second analysis, we present the average consumption of processing and memory resources by monolithic and disaggregated versions of OSC and SD-RAN, as illustrated in Fig. 12. Similar to OSC, SD-RAN is also available for deployment on Kubernetes, but the SD-RAN is not suitable for disaggregation into the corresponding five components. However, we identified the possibility of disaggregation into

four components by grouping SDL/STSL with NIBs, which can be easily dealt with by our model. Thus, we changed the number of components in which the OSC is disaggregated to four as well as suppressed the set of xApp components since OSC and SD-RAN employ different application development and deployment approaches. Figure 12 shows that the disaggregated version of both OSC and SD-RAN Near-RT RIC implementations may offer better solutions than their corresponding monolithic versions, since each set of components can be flexibly distributed while ensuring the latency-sensitive control loop. In fact, the disaggregated strategy can be even more beneficial to the SD-RAN due to the approach it adopts for implementing set SDL (i.e., SDL/STSL and NIBs) which implies on a high demand for processing and memory.

VIII. CONCLUSION

In this work, we claim that disaggregating and distributing the components of the Near-RT RIC is the most efficient approach to ensure that it can support a large-scale real-world RAN. However, it is necessary to orchestrate the components properly across the edge-cloud continuum, so that the Near-RT RIC can meet the stringent latency requirements of xApp control loops. We propose the RIC orchestrator (RIC-O) that employs a hybrid strategy combining optimal and heuristic strategies for the efficient placement of the Near-RT RIC components. We evaluate RIC-O through analytical modeling and real-world experiments to illustrate its properties and benefits. Future work includes the investigation of machine learning-based time series forecasting for improving the orchestration of Near-RT RIC components. Moreover, the adoption of the O-RAN architecture by large-scale production networks will provide additional data on its usage, thus allowing extensive investigation of novel optimization techniques. Finally, we envision the potential of the RIC-O being adopted as part of the Non-RT RIC, running as an rApp, which will demand further experiments and software integration.

REFERENCES

[1] V. S. Pana, O. P. Babalola, and V. Balyan, "5G radio access networks: A survey," *Array*, vol. 14, Jul. 2022, Art. no. 100170. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590005622000315

[2] (2022). *O-RAN Alliance*. [Online]. Available: https://www.o-ran.org/

[3] A. Garcia-Saavedra and X. Costa-Pérez, "O-RAN: Disrupting the virtualized RAN ecosystem," *IEEE Commun. Standards Mag.*, vol. 5, no. 4, pp. 96–103, Dec. 2021.

[4] *O-RAN Architecture Description*, O-RAN Alliance, Alfter, Germany, O-RAN.WG1.O-RAN-Architecture-Description-v07.00, 2022.

[5] *NG-RAN; Architecture Description*, document TS 38.401, V17.1.1, 3GPP, 2022.

[6] B. Balasubramanian et al., "RIC: A RAN intelligent controller platform for AI-enabled cellular networks," *IEEE Internet Comput.*, vol. 25, no. 2, pp. 7–17, Mar. 2021.

[7] M. Dryjanski and A. Kliks, "The O-RAN whitepaper 2022 RAN intelligent controller, xApps and rApps," White Paper, 2022. Accessed: Nov. 24, 2023. [Online]. Available: https://rimedolabs.com/blog/the-oran-whitepaper-2022-ran-intelligent-controller

[8] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "OrchestRAN: Network automation through orchestrated intelligence in the open RAN," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2022, pp. 270–279.

[9] *Near-Real-time RAN Intelligent Controller Architecture & E2 General Aspects and Principles*, O-RAN Alliance, Alfter, Germany, O-RAN.WG3.E2GAP-v02.02, 2022.

[10] *Near-Real-time RAN Intelligent Controller, E2 Application Protocol (E2AP)*, O-RAN Alliance, Alfter, Germany, O-RAN.WG3.E2AP-v02.03, 2022.

[11] *O-RAN.WG1.Use-Cases-Detailed-Specification-v09.00*, O-RAN Alliance, Alfter, Germany, 2022.

[12] Software Commu. *O-RAN Software Community (SC)*. Accessed: Nov. 24, 2023. [Online]. Available: https://oran-osc.github.io

[13] O-RAN Alliance. *O-RAN Software Community (SC) Installation Guides*. Accessed: Nov. 24, 2023. [Online]. Available: https://docs.o-ran-sc.org/projects/o-ran-sc-it-dep/en/latest/installation-guides.html

[14] A. K. Singh and K. Khoa Nguyen, "Joint selection of local trainers and resource allocation for federated learning in open RAN intelligent controllers," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2022, pp. 1874–1879.

[15] A. Huff, M. Hiltunen, and E. P. Duarte, "RFT: Scalable and fault-tolerant microservices for the O-RAN control plane," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, May 2021, pp. 402–409.

[16] R. Schmidt, M. Irazabal, and N. Nikaein, "FlexRIC: An SDK for next-generation SD-RANs," in *Proc. 17th Int. Conf. Emerg. Netw. EXperiments Technol.*, Dec. 2021, pp. 411–425.

[17] Y. Cao, S.-Y. Lien, Y.-C. Liang, and K.-C. Chen, "Federated deep reinforcement learning for user access control in open radio access networks," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2021, pp. 1–6.

[18] Y. Cao, S.-Y. Lien, Y.-C. Liang, K.-C. Chen, and X. Shen, "User access control in open radio access networks: A federated deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 21, no. 6, pp. 3721–3736, Jun. 2022.

[19] D. Johnson, D. Maas, and J. V. D. Merwe, "NexRAN: Closed-loop RAN slicing in POWDER—A top-to-bottom open-source open-RAN use case," in *Proc. 15th Workshop Wireless Net. Testbeds, Experim. Eval.*, 2021, pp. 17–23.

[20] S. Burer and A. Saxena, "Old wine in a new bottle: The MILP road to MIQCP," *Optimization Online*, vol. 154, pp. 373–406, 2009.

[21] L. Liberti, "Undecidability and hardness in mixed-integer nonlinear programming," *RAIRO-Oper. Res.*, vol. 53, no. 1, pp. 81–109, Jan. 2019.

[22] O-RAN SC. (2020). *RIC Alarm System*. [Online]. Available: https://wiki.o-ran-sc.org/display/RICP/RIC+Alarm+System

[23] *Near-Real-Time RAN Intelligent Controller E2 Service Model (E2SM) KPM*, O-RAN Alliance, Alfter, Germany, O-RAN.WG3.E2SM-KPM-v02.02, 2022.

[24] *Near-Real-Time RAN Intelligent Controller E2 Service Model (E2SM), RAN Control*, O-RAN Alliance, Alfter, Germany, ORAN.WG3.E2SM-RC-v01.02, 2022.

[25] F. Z. Morais et al., "PlaceRAN: Optimal placement of virtualized network functions in beyond 5G radio access networks," *IEEE Trans. Mobile Comput.*, vol. 22, no. 9, pp. 5434–5448 Sep. 2023.

[26] Open Networking Foundation. (2021). *SD-RAN*. [Online]. Available: https://opennetworking.org/sd-ran

Gabriel Matheus Almeida received the bachelor's and M.Sc. degrees in computer science from Universidade Federal de Goiás (UFG) in 2022 and 2023, respectively, where he is currently pursuing the Ph.D. degree in computer science. He has been a member of the Laboratory Computer Networks and Distributed Systems (LABORA), since 2018. His research spans wireless networks, virtualization, resource allocation, and performance evaluation.

Gustavo Zanatta Bruno (Graduate Student Member, IEEE) received the M.Sc. degree from Universidade Federal Fluminense. He is currently pursuing the Ph.D. degree in computer science with UNISINOS, Brazil. He is also a Student Intern with CCI, Virginia Tech; and an IT Infrastructure Analyst with MTI, Brazil, a public IT company. His research focuses on infrastructure automation, 5G, 6G, O-RAN, and cloud computing.

Alexandre Huff received the degree in computer technology from Universidade Paranaense in 2004, the master's degree in computer science from the State University of Maringá in 2010, and the Ph.D. degree in informatics from the Federal University of Paraná in 2021. He is an Adjunct Professor with Universidade Tecnológica Federal do Paraná, Toledo, Brazil, where he has been working since 2013. His research topics include distributed systems, network function virtualization, O-RAN, and cloud computing.

Matti Hiltunen (Member, IEEE) is currently a Lead Inventive Scientist with AT&T Labs Research, Florham Park, NJ, USA. He was a PTL of the RICAPP Project at O-RAN Software Community, from 2019 to 2021. His research interests include open RAN, dependable distributed systems and networks, and adaptive systems. He is a member of the IEEE Computer Society and IFIP Working Group 10.4 on Dependable Computing and Fault-Tolerance.

Elias Procopio Duarte Jr. (Senior Member, IEEE) is a Full Professor with Universidade Federal do Paraná, Curitiba, Brazil, where he is a Leader of the Computer Networks and Distributed Systems Laboratory (LaRSis). He has published more than 250 peer-reviewed papers and has supervised more than 130 students both on the graduate and undergraduate levels. His research interests include computer networks, distributed systems, and their dependability, management, and algorithms.

Cristiano Bonato Both is a Professor of the Applied Computing Graduate Program, University of Vale do Rio dos Sinos (UNISINOS), Brazil. He coordinates research projects funded by H2020 EU-Brazil, CNPq, FAPERGS, and RNP. His research focuses on wireless networks, next-generation networks, softwarization, and virtualization technologies for telecommunication networks.

Kleber Vieira Cardoso received the degree in computer science from Universidade Federal de Goiás (UFG) in 1997 and the M.Sc. and Ph.D. degrees in electrical engineering from COPPE, Universidade Federal do Rio de Janeiro, in 2002 and 2009, respectively. He has been an Associate Professor with the Institute of Informatics, UFG, since 2009. He spent his sabbaticals with Virginia Tech, USA, in 2015; and the Inria Saclay Research Center, France, in 2020. His research spans wireless networks, virtualization, resource allocation, and performance evaluation.