POLITECNICO DI TORINO
Repository ISTITUZIONALE

CORMO-RAN: Energy Efficiency at the Near-RT RIC via Lossless Migration of O-RAN xApps

Original
CORMO-RAN: Energy Efficiency at the Near-RT RIC via Lossless Migration of O-RAN xApps / Calagna, A., Maxenti, S., Bonati, L., D'Oro, S., Melodia, T., Chiasserini, C.F.. - In: IEEE TRANSACTIONS ON MOBILE COMPUTING. - ISSN 1536-1233. - (In corso di stampa).

Availability:
This version is available at: 11583/3013275 since: 2026-07-19T16:42:27Z

Publisher:
IEEE

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©9999 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

14 October 2026

CORMO-RAN: Energy Efficiency at the Near-RT RIC via Lossless Migration of O-RAN xApps

Antonio Calagna, *Member, IEEE*, Stefano Maxenti, *Graduate Student Member, IEEE*, Leonardo Bonati, *Member, IEEE*, Salvatore D'Oro, *Member, IEEE*, Tommaso Melodia, *Fellow, IEEE*, Carla Fabiana Chiasserini, *Fellow, IEEE*

***Abstract*—Open Radio Access Network (RAN) is a key paradigm to attain unprecedented flexibility of the RAN via disaggregation and Artificial Intelligence (AI)-based applications called xApps. In dense areas with many active RAN nodes, compute resources are engineered to support potentially hundreds of xApps monitoring and controlling the RAN to achieve operator's intents. However, such resources might become underutilized during low-traffic periods, where most cells are sleeping and, given the reduced RAN complexity, only a few xApps are needed for its control. In this paper, we propose CORMO-RAN, a data-driven orchestrator that dynamically activates compute nodes based on xApp load to save energy, and performs lossless migration of xApps from nodes to be turned off to active ones while ensuring xApp availability during migration. CORMO-RAN tackles the trade-off among service availability, scalability, and energy consumption while (i) preserving xApps' internal state to prevent RAN performance degradation during migration; (ii) accounting for xApp diversity in state size and timing constraints; and (iii) implementing several migration strategies and providing guidelines on best strategies to use based on resource availability and requirements. We prototype CORMO-RAN as an rApp, and experimentally evaluate it on an O-RAN private 5G testbed hosted on a Red Hat OpenShift cluster with commercial radio units. Results demonstrate that CORMO-RAN is effective in minimizing energy consumption of the RAN Intelligent Controller (RIC) cluster, yielding up to 64% energy saving when compared to existing approaches.**

***Index Terms*—Open RAN, xApp, Stateful Migration, Shared Data Layer**

I. Introduction

The Open Radio Access Network (RAN) paradigm—and its embodiment proposed by the O-RAN ALLIANCE [1]—has been heralded as the vehicle to bring unprecedented flexibility to 5G-and-beyond RAN architectures. O-RAN promotes enhanced *flexibility* via RAN disaggregation and virtualization, as well as *adaptability* through data-driven control loops that optimize the RAN performance [2]. Cornerstones of the O-RAN architecture, depicted in Fig. 1a, are the RAN Intelligent Controllers (RICs), which oversee the operations of the RAN through closed control-loops at different time scales: near-real-time (or near-RT, between 10 ms and 1 s), and non-real-time (or non-RT, above 1 s). RAN control is actuated via intelligent applications hosted as microservices on the RICs—namely, xApps on the near-RT and rApps on the non-RT RIC—that leverage Key Performance Measurements (KPMs) coming from the RAN to perform inference/forecasting or compute policies to optimize network performance.

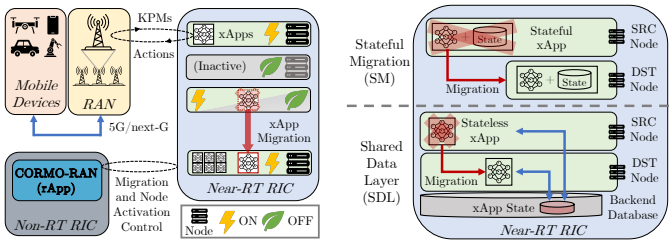


Figure 1: (a) Concept representation of CORMO-RAN, and (b) comparison of SM and SDL xApp migration approaches.

Existing Issues. xApps enable for the first time self-optimizing and zero-touch cellular networks. However, their contribution to the RIC's cluster energy consumption is non-negligible, especially in large deployments with hundreds of xApps [3]. Additionally, the number of xApps needed to control the RAN might vary significantly from peak hours, when traffic demand is high, to nighttime, when most cells might be in energy-saving mode or even turned off. Therefore, even though only a few xApps may be actively controlling RAN elements, many compute nodes would still be active and underutilized, resulting in unnecessary energy consumption.

In this context, microservice migration—i.e., transferring a microservice from a source to a destination node—is a powerful tool to reallocate xApps across different nodes of the same near-RT RIC cluster to dynamically minimize the number of active nodes and turn off the inactive ones, depending on the network load. While *stateless* xApp migration relies on well-established and low-latency techniques that simply deactivate xApps on the source node and recreate them on the destination node, migration of *stateful* xApps is not as trivial. Indeed, stateful xApps (e.g., used in forecasting, beam tracking and mobility management) need to maintain a history of context-

Antonio Calagna and Carla Fabiana Chiasserini are with the Department of Electronics and Telecommunications, Politecnico di Torino, 10129, Turin, Italy (email: antonio.calagna@polito.it; carla.chiasserini@polito.it). Carla Fabiana Chiasserini is also with CNIT, 43124, Parma, Italy and Chalmers University, SE412-96, Göteborg, Sweden. Stefano Maxenti, Leonardo Bonati, Salvatore D'Oro, and Tommaso Melodia are with the Institute for the Wireless Internet of Things, Northeastern University, Boston, MA 02115, USA (e-mail: maxenti.s@northeastern.edu; l.bonati@northeastern.edu; s.doro@northeastern.edu; t.melodia@northeastern.edu). This work was partially supported by the U.S. National Telecommunications and Information Administration (NTIA)'s Public Wireless Supply Chain Innovation Fund (PWSCIF) under Award No. 25-60-IF002, by the U.S. National Science Foundation under grant CNS-2112471, by the EC through Grant No. 101139266 (6G-INTENSE project), and by the Qatar Research Development and Innovation Council ARG01-0525-230339. The content is solely the responsibility of the authors and does not necessarily represent the official views of Qatar Research Development and Innovation Council.

based data to accomplish their tasks. This data is stored in an internal *state* that must be preserved upon migration to retain control effectiveness and avoid performance degradation.

In this work, we focus on the lossless migration of stateful xApps and consider two approaches (Fig.1b): Stateful Migration (SM) [4], and the O-RAN Shared Data Layer (SDL) [2]. SM migrates xApps together with their state, causing a service disruption referred to as *downtime*, with two variants: SM-MR, minimizing resource usage; and SM-MD minimizing downtime. Instead, SDL decouples xApps from their state by storing it in a backend database, making xApps virtually stateless from a migration viewpoint. However, this distributed database must guarantee strong consistency, potentially limiting SDL's scalability and feasibility.

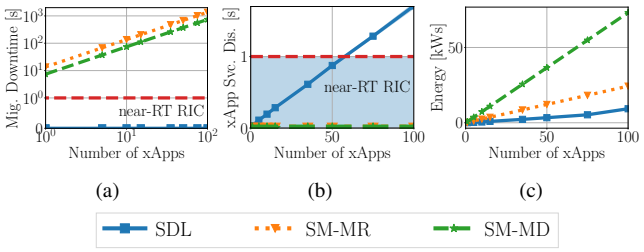


Figure 2: Comparing SDL, SM-MR, and SM-MD: (a) migration downtime, (b) xApp service disruption, and (c) energy usage.

To highlight their differences, in Fig. 2 we experimentally compare these strategies for several copies of a same exemplary Deep Reinforcement Learning (DRL)-based xApp that receives KPMs from the RAN, stores them as state, and computes a control action. The red dashed line indicates the near-RT RIC control loop deadline, which we here set to 1 s as an example. We notice that: (a) while SDL enables zero-downtime migration, SM has a downtime that linearly increases with the number of xApps, always violating any near-RT RIC deadline (Fig. 2a); (b) contrarily to SM, SDL has scalability issues, yielding a periodic xApp service disruption that can lead to near-RT RIC deadline violations (Fig. 2b); and (c) SDL yields up to 87% reduction in energy consumption compared to SM (Fig. 2c). These results show that migrating stateful xApps in O-RAN involves a trade-off among service availability, scalability, and energy consumption.

Novelty. To tackle the above challenges, in this paper we propose CORMO-RAN, a data-driven orchestrator that jointly optimizes compute node activation and xApp migration strategies. Differently from existing works, CORMO-RAN:

- *(i)* accounts for stateful xApps whose state must be preserved, and whose control tasks need to be executed within a strict temporal deadline;
- *(ii)* encompasses both SM and SDL migration techniques as well as a diverse xApp catalog that captures varying workload use cases;
- *(iii)* is evaluated on a real near-RT RIC deployment using a pre-trained, publicly available AI-driven xApp;
- *(iv)* identifies the feasibility regions of each migration strategy based on traffic load and available resources;
- *(v)* dynamically computes the optimal allocation of xApps across near-RT RIC nodes that minimizes the overall system energy consumption.

Importantly, our work is the first to (i) experimentally characterize performance and trade-offs of state-of-the-art migration techniques in the O-RAN context; and (ii) to leverage such techniques to minimize the near-RT RIC energy footprint. We prototype CORMO-RAN as a non-RT RIC rApp, and leverage it to jointly orchestrate the node activation and migration of xApps on a Red Hat OpenShift cluster. For this, we consider real-world xApps that (i) reflect varying levels of RAN workload complexity; (ii) are deployed on a cluster of nodes, together with an open-source near-RT RIC; and (iii) are used to re-configure a private 5G testbed with commercial Radio Units (RUs) and User Equipments (UEs). Our results demonstrate that CORMO-RAN effectively addresses the aforementioned trade-off and enables up to 64% reduction of the system energy consumption.

Paper Structure. The rest of the paper is organized as follows. We first review relevant related works and emphasize the uniqueness of our study in Sec. II. Then, we describe the SM and SDL xApp migration approaches and present the experimental testbed we developed to evaluate them in Sec. III and Sec. IV, respectively. We use this testbed to analyze the migration process across diverse classes of xApps in Sec. V, and leverage the resulting experimental evidences to formulate our optimization problem and proposed solution, CORMO-RAN, in Sec. VI. Finally, we conduct a comprehensive performance evaluation of CORMO-RAN in Sec. VII and draw our conclusions in Sec. VIII.

II. Related Work

Ongoing efforts and challenges related to sustainable mobile networking are analyzed in [5], [6], [7], [8]. Work in [9] finds the RAN segment to be the one impacting energy consumption the most (up to 73%), and Artificial Intelligence (AI) has been identified as a promising solution to minimize such energy consumption [10], [11], [12] and improve quality of experience [13], [14], [15]. For instance, [16], [17] provide AI-driven solutions to enhance O-RAN energy efficiency through, respectively, traffic steering and cell on/off control. [18] proposes an O-RAN orchestrator that, by semantically sharing xApps across RAN services, aims to maximize the number of the services concurrently deployed while minimizing their overall energy consumption. Nonetheless, the proliferation of AI-based xApps inevitably contributes to the energy footprint, posing an additional challenge toward network sustainability. Indeed, [3] profiles various types of xApps in terms of their energy consumption and demonstrates that scaling up the number of concurrently running xApps leads to a proportional increase in the overall energy usage of the near-RT RIC cluster. Also, it is shown that xApps are a dominant contributor to the overall system energy footprint, thus highlighting the importance of energy-aware orchestration and placement strategies.

As per service migration, [19], [20] give an overview of current stateful migration techniques along with their Key Performance Indicators (KPIs) and discuss the potential of such techniques in addressing critical mobile scenarios in the general context of edge computing, where service continuity is of utmost importance. [21], [22] propose practical solutions

for seamless service migration at the network edge, focusing, respectively, on video analytics and real-time rendering applications. To address the lack of a migration model, [4] analyzes and captures the practical aspects of stateful migration. [23] introduces MOSE, a novel framework that efficiently implements stateful migration and effectively orchestrates the migration process by fulfilling both network and application KPI targets. Leveraging migration techniques, [24], [25], [26], [27], [28] propose solutions to attain an optimal service placement while prioritizing mobile end user requirements.

Regarding xApp state decoupling, although SDL is defined as part of the O-RAN specifications [2], its implementation details—including the choice of backend database—remain open and are yet to be standardized. In this context, a growing concern regarding the need to rethink traditional database architectures is raised in [29], [30]. Specifically, it is observed that the inherently decentralized data management of microservice architectures poses significant challenges for coordination, as state dependencies and consistency issues are often overlooked, with a non-negligible amount of applications requiring strong consistency guarantees over the shared information they access. [31] proposes varying architectures and implementations of a holistic data access platform at the edge—sharing the same design principles as SDL—and thoroughly characterizes their performance and trade-offs across a spectrum of scenarios, ranging from loosely controlled loops to latency-critical and compute-demanding use cases.

Distinctive Methodology and Contribution. To the best of our knowledge, our work is the first to experimentally evaluate cutting-edge migration approaches in the O-RAN context to jointly optimize compute node activation and xApp placement while minimizing energy consumption and guaranteeing xApp service availability. Specifically, we: (i) characterize migration techniques to assess their benefits, drawbacks, and performance; (ii) derive a model that captures the fundamental trade-off between downtime, energy consumption, and availability; and (iii) develop algorithms to identify the feasibility regions of each technique and optimize service migration and placement to minimize energy consumption under strict availability constraints. Importantly, although node activation and workload optimization have been investigated in the broader context of edge computing [32], [33], our work delivers a fundamentally different perspective that, unlike prior art, captures the unique challenges of the O-RAN context, i.e., by explicitly accounting for its stringent timing requirements and the practical, real-world challenges of xApp migration, e.g., the need to preserve their internal state as a way to guarantee service continuity.

III. Overview of xApp Migration

This section describes the two main technologies for the lifecycle management of stateful xApps. First, we introduce the concept of stateful migration, along with its KPIs (Sec. III-A).[1] Then, we provide an overview of the shared data layer approach used in O-RAN [2] to support data access and sharing among multiple xApps (Sec. III-B).

[1]In this work, KPIs refer to migration strategies, while KPMs refer to data produced by the RAN and collected by the near-RT RIC and xApps.

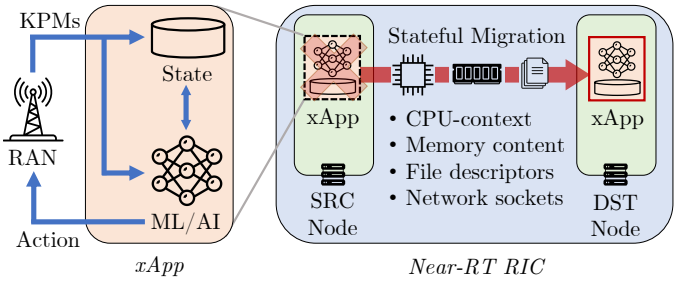


Figure 3: Stateful migration of xApps in O-RAN.

A. Stateful Migration (SM)

As shown in Fig. 3, this approach considers the case where the state of the xApps (e.g., context-related metrics) is embedded in the xApp, which runs as a microservice container. To preserve the state and ensure service continuity, SM relocates the entire container (which includes the state) from the source node to the destination node. As shown in Fig. 3, besides the container image, SM requires the following pieces of information at the destination node: (i) CPU-context state, e.g., registers, processes tree structure, and namespaces; (ii) memory content, i.e., the pages allocated in the main memory; (iii) network sockets; and (iv) open file descriptors.

SM has two variants: SM-MR and SM-MD. The former prioritizes resource minimization during the migration process; the latter focuses on minimizing the migration downtime. For both variants, we consider to migrate multiple xApps in a sequential way, which reflects the practical limitations of the off-the-shelf, application-agnostic solution we leveraged.

SM-MR uses *Cold Migration*, consisting of the following steps: (i) collection of the state checkpoint at the source node; (ii) transfer of such state from source to destination node; and (iii) restoration of the container state at the destination. To prevent state inconsistency, the container is stopped at the source node throughout these steps, thus causing a service disruption period, i.e., the *migration downtime*.

SM-MD implements the *Iterative PreCopy* algorithm and draws on the *dirty-page rate* concept, i.e., the number of memory pages per second a container modifies. This strategy consists of: (i) the iterative transfer of dirty pages to the destination node while the container is still running at the source node; and (ii) stopping the container and transferring the remaining dirty pages to the destination node. By minimizing the amount of data to be transferred, SM-MD trades a longer *total migration duration* for a shorter downtime.

B. Shared Data Layer (SDL)

To regulate data production and consumption between xApps, O-RAN introduces a data access platform, called SDL, which acts as an abstraction layer between the applications and a backend database where data is stored and shared. As in Fig. 4, SDL can be used to decouple the xApp from its state, which can be instead stored in the backend database. From a migration viewpoint, SDL effectively transforms stateful xApps into stateless as the state is still present but stored externally in the SDL. Therefore, this approach (i) conforms with the requirements of 5G-and-beyond networks [34] and

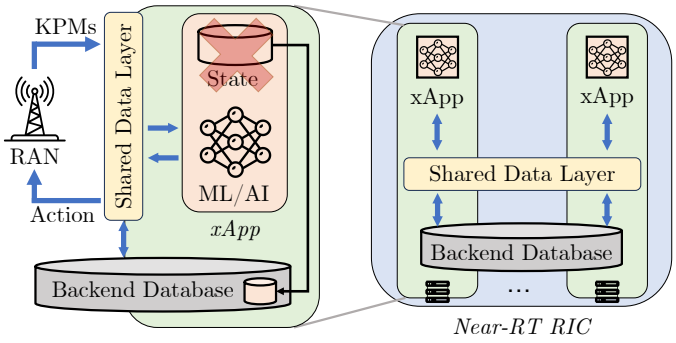


Figure 4: O-RAN shared data layer architecture to decouple xApps from their internal state.

recent microservice-oriented architecture design patterns [35], both requiring microservices to be stateless to maximize efficiency and scalability; and (ii) enables migration strategies that are zero-downtime by design, i.e., yielding no service disruption to the final users.

Nevertheless, since the state is now outsourced to the backend database, accessing the state incurs in additional delay that might impact xApp performance and timeliness of control policies. Therefore, data access must happen with the lowest possible latency. Also, to avoid the creation of a single point of failure, the backend database must be tolerant to faults and network partitions, e.g., by distributing multiple replicas of its content across the near-RT RIC nodes. In CORMO-RAN we consider a migration strategy where we proactively duplicate the xApp at the destination node and, upon success, we remove it from the source node. Contrarily to SM, which needs the xApp instance at the source node to be stopped before resuming execution at the destination host, under SDL, the two xApp instances can keep on serving incoming requests from the RAN and updating their internal state while the migration process takes place, yielding no service disruption. Importantly, since the xApp state is shared by the two instances during the migration process, the backend database needs to effectively support concurrent data accesses while guaranteeing strong data consistency to prevent race conditions.

In summary, SDL requires a backend database with: (i) high availability; (ii) high reliability and fault-tolerance; and (iii) strong data consistency. We address such technical challenges by choosing `etcd` [36] as the near-RT RIC backend database. Etcd is a distributed reliable key-value store for the most critical data of a distributed system that, by leveraging the Raft [37] consensus algorithm, enforces strong data consistency, and tolerance to network partitions and machine failure at the cost of a reduced availability [38]. We remark that, while other popular state-of-the-art databases such as Redis [39] prioritize high availability by favoring eventual consistency guarantees, our work focuses on the more challenging scenario in which strong data consistency must be ensured. This requirement is critical to maintain correctness and coherence of the information shared among multiple xApps, particularly during the coordination of latency-sensitive near-RT RIC control loops. As shown in [31], a comparison between etcd- and Redis-based implementations reveals fundamental trade-offs in terms of scalability, availability, data consistency, and resource usage, with etcd demonstrating superior performance when resilience and strong data consistency are of utmost importance. Compared to other strongly consistent databases such as Zookeeper and Consul, etcd offers a well-established balance of stability, reliability, scalability, and performance—even when operating at multi-gigabyte scales—while avoiding the architectural complexity and latency overhead associated with NewSQL systems [40], [41], [42]. These characteristics, together with its role as the official Kubernetes' core data store, have made etcd a widely adopted and frequently referenced solution for state-of-the-art distributed systems.

To guarantee high reliability, etcd stores data in a multi-version persistent key-value store, preserving the previous version of a key-value pair when its value is updated. As a result, etcd keeps an exact history of its keyspace, which should be periodically compacted to avoid performance degradation and eventual storage space exhaustion. Since compacting old revisions internally fragments etcd by leaving gaps in the backend database, it is also necessary to release this storage space back to the file system through a defragmentation process. Importantly, during defragmentation, the etcd member rebuilds its states and is thus blocked from reading and writing data, yielding service disruption for the xApps. In the following, we refer to the combination of the compaction and defragmentation processes as a *maintenance* operation whose periodicity can be controlled to prevent resource exhaustion and etcd performance degradation. Our analysis accounts for the service disruption duration, denoted as *defrag downtime*, yielded by each maintenance operation and assesses if, and to what extent, such downtime is compatible with the near-RT RIC control loop deadlines.

IV. Experimental O-RAN Testbed

In this section, we describe the testbed that we developed to evaluate the two approaches above, i.e., SM and SDL, identify their feasibility region, and determine which approach is best suited to certain operational conditions and compute loads.

Computing cluster. We deploy an end-to-end O-RAN system, comprising an open-source near-RT RIC [43] from the O-RAN Software Community (OSC) on a Red Hat OpenShift cluster [44]. OpenShift [45] is a commercial container orchestration platform that extends Kubernetes with production-grade security, reliability, resilience, and fault tolerance functionalities, among others. From an infrastructure standpoint, our cluster comprises four Dell R760 compute nodes equipped with 128 Intel Xeon 8462Y+ CPUs, 512 GB of RAM (16×32 GB DDR5 blocks working at 4.8 GHz frequency) and 960 GB of storage (NVMe disk operating at a maximum link speed of 16 gigatransfer/s). Nodes are connected via 10 Gbps interfaces to a Dell S5248F-ON Software Defined Networking (SDN) switch that enables fast and reliable communication among them. Besides local storage, cluster nodes are connected to a Network-attached Storage (NAS) that provides persistent storage for the containers and the internal image registry.

To gather accurate and comprehensive metrics for our experimental analysis, our testbed integrates Prometheus [46] and

Kepler [47]. Prometheus is a widely adopted Kubernetes monitoring system that facilitates effective cluster-wide metrics aggregation. Kepler, on the other hand, is a renown framework that uses advanced power models to estimate real-time energy consumption at the pod level (i.e., at the Kubernetes fundamental unit). Given the importance of accurately estimating a system carbon footprint [48], Kepler accounts not only for the active computations but also for idle power, i.e., the static node power. As thoroughly discussed in [49], [50], [51], such idle contribution mainly consists of the power related to hardware components, such as motherboard, fans, network interface cards, and other peripherals, as well as the power consumed by the Kubernetes elements that are necessary for the system to be functional, e.g., the Kubelet and the control plane.

xApp. To run our experiments, we consider two publicly available xApps that differ in terms of computational complexity, namely, the DRL-xApp [52], [53] and the KPM-xApp [54]. The former is representative of AI-driven control tasks and implements a pre-trained DRL agent that, by leveraging RAN KPMs, computes the optimal scheduling policy for the network slices implemented at the base station. The latter is a monitoring xApp that exemplifies less demanding tasks, as it collects and aggregates KPMs to derive RAN performance statistics. Both xApps operate according to an event-driven approach, i.e., executing their logic whenever a KPM message of arbitrary size is produced by the RAN. To allow for an accurate scalability analysis of the aforementioned migration strategies when hundreds of xApps are deployed on the RIC, we build an E2 agent emulator capable of synthetically generating traffic with varying loads (see Sec. V-A). Importantly, the insights and results presented in the following remain valid even when actual RAN nodes are connected to the RIC, and are independent of the specific xApp we use, thus remaining broadly applicable to any kind of control task, regardless of its complexity. Since SM and SDL rely on different xApp architectures, we have modified the above xApps to consider two programmable variants that differ in how the internal state is stored and accessed. We thus recall that the xApp internal state is defined as the set of all relevant information, e.g., history of context-based data, that the xApp requires to accomplish its tasks and that must be preserved upon migration to ensure control effectiveness and service continuity. The SM variant retains the internal state in a queue of tunable size and stored in the main memory, along with the other components necessary for xApp execution (e.g., AI model weights). Instead, the SDL variant leverages etcd APIs to delocalize such state queue onto the database and performs the following steps: (i) interrupt-based watch of the KPM key, which is updated every time the RAN produces a new KPM message; (ii) push such message into the state queue; (iii) pop the least recent message from the queue; (iv) produce the control message jointly leveraging the newest message and the queue, and put it on the database so that it can be consumed by the RAN. Furthermore, we remark that while SM technology relies on application-agnostic tools and migrates the whole xApp memory content, SDL permits to selectively decouple just the xApp internal state onto the backend database, thus excluding the extra memory content that is independent of the specific xApp-related context.

SM. To implement SM we leverage the Migration Orchestration framework for microServices at the Edge (MOSE) [23], which consists of two fundamental off-the-shelf tools, namely, CRIU [55] and Podman [56]. The former is widely considered the key tool to achieve SM at a process level, and the latter extends CRIU functionalities to a container level (e.g., xApp containers). Furthermore, MOSE implements the Processing-aware Migration (PAM) model [4] to accurately characterize the fundamental migration KPIs as a function of the xApp memory usage and dirty-page rate (see Sec. III-A). Leveraging CRIU, Podman, and PAM model, MOSE configures and orchestrates the migration process to fulfill both the target migration KPIs and the vertical's objective, i.e., to minimize either the migration downtime (SM-MD) or the resource consumption in terms of required network bandwidth and CPU usage (SM-MR). Also, depending on such objective, we configure the maximum bandwidth used by MOSE as follows: 1 Gbps (i.e., underutilizing our bandwidth resources) for SM-MR and 5 Gbps for SM-MD (i.e., saturating our bandwidth resources).

SDL. As mentioned in Sec. III-B, to attain migration based on SDL, we use etcd to create a distributed backend database. For fault-tolerance purposes, we fix the size of the etcd cluster to three, i.e., the number of the control-plane nodes in our OpenShift cluster. It is worth mentioning that the number of etcd instances is not meant to vary in real-time, as it depends only on the cluster architecture design. To control the etcd maintenance operations, we consider two parameters: (i) the "snapshot count", i.e., the number of key-value pairs revisions to retain before compaction; and (ii) the "maintenance period", i.e., how often an etcd instance performs compaction and defragmentation. While the latter can be configured in real-time, the snapshot count can be configured only upon etcd cluster bootstrap. Therefore, we set such value to 100 as (i) previous revisions become obsolete, i.e., we only retain the key-value pairs that are needed but we are not interested in their history of changes; and (ii) we observed that this value is the smallest that prevents etcd overload with negligible impact on the overall performance in our testbed.

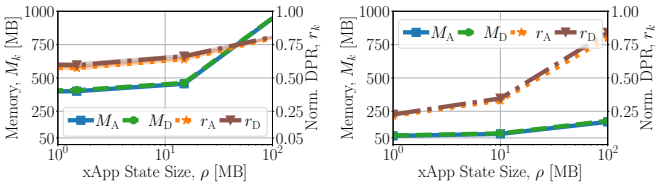
To conclude, our testbed includes: (i) a real end-to-end O-RAN system; (ii) a full-fledged computing architecture; (iii) a representative, programmable, and AI-driven xApp; and (iv) a migration framework based on off-the-shelf tools. This setup enables accurate emulation of real-world O-RAN scenarios and thorough evaluation of migration performance and trade-offs under various strategies and traffic conditions.

V. Experimental Analysis

We use our testbed to experimentally characterize the xApp migration process under SM and SDL. We focus on a diverse set of xApp models that capture different use cases (Sec. V-A). Then, we thoroughly analyze performance and limitations of both approaches, focusing on temporal KPIs (Sec V-B) and resource usage (Sec V-C). All presented results are averaged over 50 repetitions and have a 95% confidence interval.

Table I: Classes of xApp. Each class is also evaluated against different values of state size $\rho \in \{1\,\text{MB}, 10\,\text{MB}, 100\,\text{MB}\}$

Type/Features	A	B	C	D
Message Size, $\omega_{\text{s},k}$	100 B	100 B	100 kB	100 kB
Message Period, $\omega_{\text{p},k}$	1 s	100 ms	1 s	100 ms
Reference use case	mMTC	IoT	Analytics	UAVs



(a) DRL-xApp (b) KPM-xApp

Figure 5: xApp memory usage M_k and normalized dirty-page rate r_k across DRL- and KPM- xApps and varying classes k.

A. xApp Reference Scenarios

Although our approach is general, for the sake of illustration, we consider a set $\mathcal{K}$ of four representative classes of xApp, i.e., $\mathcal{K}=\{A, B, C, D\}$. Each class represents a realistic RAN workload scenario and differs in the (i) number of KPMs requested from the RAN, which reflects the size $\omega_{\text{s},k}$ of the E2 RIC Indication (report) messages; (ii) the periodicity $\omega_{\text{p},k}$ of such messages; and (iii) the xApp state size ρ.

As shown in Table I, each xApp class $k \in \mathcal{K}$ is defined by the 2-tuple $(\omega_{\text{s},k}, \omega_{\text{p},k})$. Class A addresses scenarios with loose control loops and few KPMs (i.e., small message size), typical of control for Massive Machine-Type Communications (mMTC) applications. Class B also features few KPMs but with tight control loops, aligning with Internet of Things (IoT) telemetry requirements. Class C involves large messages and loose control loops, common in surveillance and analytics applications. Eventually, Class D targets scenarios where control is frequent (e.g., every 100 ms) and many KPMs are processed at the same time (i.e., large message size), which models time-critical applications, e.g., self-driving Unmanned Aerial Vehicles (UAVs), requiring low latency control. To consider a wide range of use cases and applications, for each xApp class k we also consider multiple values of state size, i.e., $\rho \in \{1\,\text{MB}, 10\,\text{MB}, 100\,\text{MB}\}$.

B. Temporal KPIs Analysis

SM. We recall that the temporal KPIs of the stateful migration process are the migration downtime and the total migration duration, respectively denoted as T_D^SM and T_M^SM. These can be characterized via the PAM model [4] which describes them as functions of the xApp memory usage M_k, and the dirty-page rate. To analyze the dirty-page rate in a way that is independent of the state size, we use its normalized version r_k with respect to the minimum and maximum dirty-page rate values a microservice can achieve. The former is 1 page/s and the latter is total number of pages allocated in memory per second. By focusing on the least and most demanding classes, i.e., A and D, we now characterize M_k and r_k and the corresponding values for the KPIs.

Fig. 5 shows M_k and r_k for varying state size ρ and xApp classes. We notice that the values of M_k of the DRL-xApp are

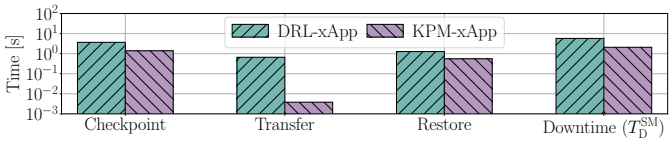


Figure 6: Migration downtime components across DRL- and KPM-xApps under the SM-MD strategy and for ρ=1 MB.

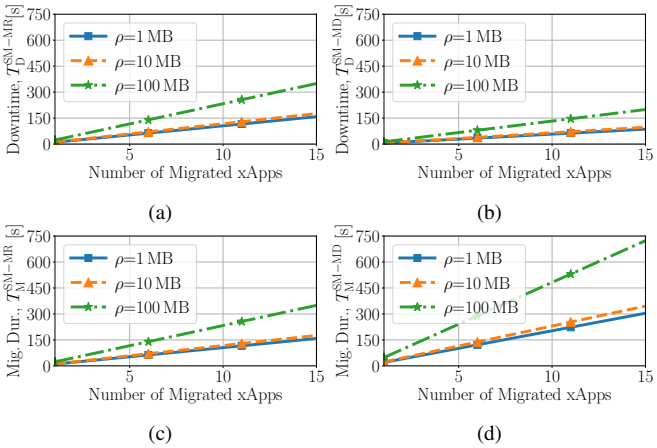


Figure 7: Stateful migration KPIs for varying xApp state size ρ.

significantly higher than ρ and they are independent of k, as M_k is affected by neither the message size nor the message frequency. Also, r_k takes large values, which indicates that most of the xApp memory content changes every second and these variations are independent of the xApp class. This behavior is because (i) the xApps's AI model consumes more memory than that used to store the KPMs received over E2; and (ii) the execution of AI models requires frequent allocation/release of memory pages to handle tensors [57]. Despite these results have been obtained by using the DRL-based xApp architecture from [53], they can be extended to general AI-based xApps, whose models and workload characteristics may vary, but still be dominant in terms of memory consumption. Looking at the results for the KPM-xApp (Fig. 5b), we observe that the values of M_k are considerably lower than those of the DRL-xApp. Also, r_k is now strongly correlated with ρ, thus ranging from small to large values. This behavior is due to the absence of the AI model, which makes the xApp state the dominant contribution to the overall memory usage. Nevertheless, despite such difference in behavior, both M_k and r_k remain independent of the traffic class k as in the DRL-xApp case, yielding that frequency and size of the KPM messages have no impact on the way in which dynamic memory is managed by the operating system.

Finding 1 (Relevant xApp features)**.** *Regardless of the xApp nature, its memory usage M_k and dirty-page rate r_k depend primarily on the state size ρ and not the traffic class k.*

Before evaluating the migration KPIs under varying load conditions, we first analyze the migration downtime into its fundamental components and assess whether SM is compatible with near-RT RIC control loop deadlines. As discussed in Sec. III-A, we recall that such downtime primarily consists of three main stages, namely, checkpoint, transfer, and restore

of the xApp state. We characterize each stage separately by migrating a single DRL-/KPM-xApp under ρ=1 MB and SM-MD, i.e., the strategy that minimizes downtime. Fig. 6 shows that migrating a KPM-xApp yields an approximately 3× lower value of $T_{\mathrm{D}}^{\mathrm{SM}}$ compared to the DRL-xApp, with highest gap on the network transfer contribution. This is due to the fact that, as shown in Fig. 5, the KPM-xApp features much lower values of M_k and r_k, yielding lower processing complexity and a lighter data transfer from source to destination node. However, consistently with the evidences in [4], even when migrating a lightweight KPM-xApp under SM-MD, the latency overhead introduced by checkpoint and restore operations dominates the overall migration downtime, which is in the order of 2 s and thus incompatible with any near-RT RIC control loop deadline. Despite this, SM remains a fundamental technique to consider, particularly in scenarios where xApp state decoupling via SDL is impractical or infeasible, as discussed in the following. Since SM is a one-time event, it can be flexibly scheduled at those times where load is low and/or temporary service disruption has minimal impact on network performance and is tolerable. To account for this and for potential future developments that reduce the processing complexity, we keep our model of the migration KPIs general and allow tunable values for the maximum acceptable migration downtime.

Finding 2 (Stateful migration feasibility)**.** *Regardless of the nature and features of the xApp, current technical limitations of the available migration tools render the SM strategy incompatible with any near-RT RIC control loop deadline.*

From now on, we focus our evaluation on the DRL-xApp, which realistically represents most demanding and AI-driven use cases. All considerations are in fact independent of the nature of the xApp and thus generalize to the case of simpler xApps. Fig. 7 depicts the cumulative $T_{\mathrm{D}}^{\mathrm{SM}}$ and $T_{\mathrm{M}}^{\mathrm{SM}}$ as functions of the number of xApps being sequentially migrated and the value of xApp state size ρ, respectively. Results demonstrate that SM-MR yields $T_{\mathrm{D}}^{\mathrm{SM}}$=$T_{\mathrm{M}}^{\mathrm{SM}}$ while SM-MD achieves a lower $T_{\mathrm{D}}^{\mathrm{SM}}$ at the cost of a higher value of $T_{\mathrm{M}}^{\mathrm{SM}}$. Also, it can be observed that (i) both KPIs depend on ρ; (ii) regardless of the SM strategy, the dependency of the KPIs on the number of xApps can be described by a linear function.

Finding 3 (Stateful migration KPIs)**.** *Although the migration downtime and the migration duration depend on the stateful migration strategy and value of state size, both linearly increase with the number of migrated xApps.*

SDL. We now investigate the performance and resource usage of the xApp migration process with SDL. Specifically, we (i) assess the impact of SDL on the migration KPIs and the xApp resource usage; (ii) characterize the service disruption due to etcd maintenance; and (iii) analyze the etcd resource usage in terms of power consumption and CPU, memory, and disk usage for varying system configurations. We recall that the SDL strategy decouples the stateful component of each xApp from the xApp itself as the state is stored in the backend database. Therefore, under SDL, stateful xApps are treated as stateless from the migration viewpoint, enabling a zero-downtime migration process (see Sec. III-B).

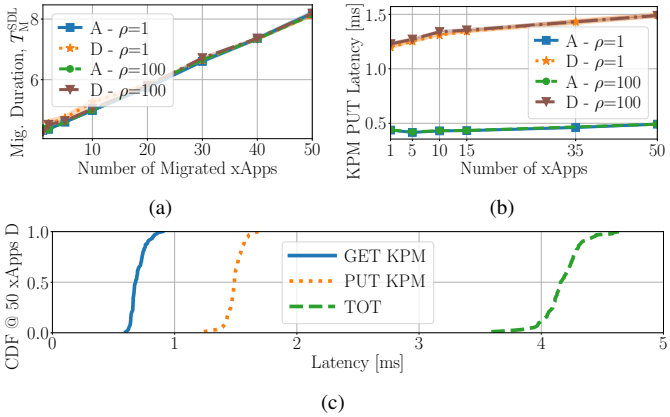


Figure 8: Stateless migration performance analysis: (a) migration duration, (b) average etcd KPM PUT latency, (c) etcd latency CDFs.

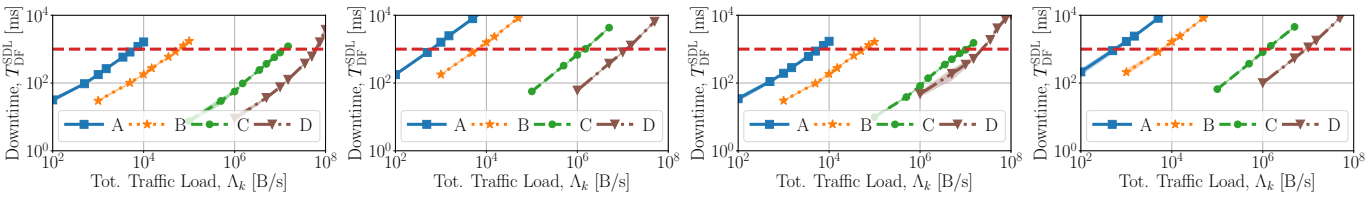
Fig. 8a shows the migration duration $T_{\mathrm{M}}^{\mathrm{SDL}}$ as a function of the number of migrated xApps. Results highlight that $T_{\mathrm{M}}^{\mathrm{SDL}}$ is independent of both k and ρ. Indeed, xApps are virtually stateless under SDL and $T_{\mathrm{M}}^{\mathrm{SDL}}$ corresponds to the time needed to instantiate new xApps, which mostly depends on the amount of memory to be allocated, that is now independent of the xApp class and state size. For the same reason, $T_{\mathrm{M}}^{\mathrm{SDL}}$ is up to two orders of magnitude lower than $T_{\mathrm{M}}^{\mathrm{SM}}$ (Fig. 7).

Finding 4 (SDL migration KPIs)**.** *Under SDL, xApps are virtually stateless migration-wise. The migration duration (i) is independent of both the xApp class and the state size; and (ii) grows with the number of xApps being migrated linearly.*

As discussed in Sec. III-B, etcd is a reliable and robust backend database solution to allow SDL in effectively decoupling the xApp from its internal state. Now, we also demonstrate experimentally that etcd meets the strict timing requirements of the near-RT RIC. To this end, we characterize the latency overhead that an xApp experiences when reading and writing key-value pairs from/to etcd as part of its control loop. First, we measure the average latency of a write operation (commonly referred to as *PUT* latency) for a single KPM key-value pair. Then, we delve into the entire xApp control loop, from KPM generation to control message production, and provide a breakdown of the total latency overhead.

Fig. 8b reports the KPM PUT latency as a function of the number of xApps for different xApp classes and state size ρ values. Results show that PUT latency is (i) increasing with the number of xApps due the larger number of requests to access the database; (ii) independent of the xApp state size that is stored on etcd; and (iii) dependent on the xApp class. We recall that class A is characterized by small and infrequent messages, while class D puts a higher pressure on etcd by generating large and frequent messages. Importantly, the PUT latency never exceeds two milliseconds even in extreme scenarios with many xApps of class D.

To assess etcd feasibility under any near-RT RIC control loop deadline ranging from 10 ms to 1 s, we now analyze the latency of the entire xApp control loop. As described in Sec. IV, for each KPM message prodcued by the RAN, the xApp performs the following steps: (1) read the KPM message (GET); (2-3) update the state queue (KPM PUT of



(a) $\rho = 1$ MB, $\nu = 1$ s (b) $\rho = 10$ MB, $\nu = 1$ s (c) $\rho = 1$ MB, $\nu = 120$ s (d) $\rho = 10$ MB, $\nu = 120$ s

Figure 9: Etcd defrag downtime for varying classes of xApp, values of xApp state size ρ and etcd maintenance period ν.

Table II: Breakdown of the etcd communication latency overhead within an xApp control loop under high traffic load, i.e., 50 xApps of type D

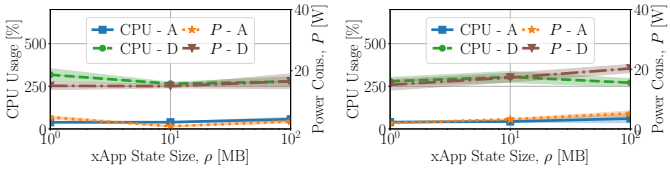
Steps / Duration [ms]	95% C.I.	p95	p99
1) Watch (GET) KPM	0.699 ± 0.012	0.824	0.865
2–3) Push/Pop (PUT) KPM	1.491 ± 0.014	1.600	1.640
4) Push (PUT) Ctrl Msg	0.492 ± 0.005	0.527	0.539
Total	4.137 ± 0.033	4.396	4.551

size $\omega_{s,k}$); (4) produce and push the control message (CTRL PUT of size fixed to 100 B). We focus on a high traffic load scenario featuring 50 concurrently running xApps of class D. Fig. 8c reports the latency CDFs for the GET KPM and PUT KPM operations as well as the cumulative control loop latency overhead. Similarly, Table II analyzes the latency of each step, including the 95 % confidence interval and 95th- and 99th-percentile values. Results demonstrate that (i) the KPM PUT operations are slower than KPM GET due to the strong consistency guarantees, (ii) the KPM PUT operations are also slower than CTRL PUT ones because of their larger payloads, and (iii) the latency distributions—both for individual steps and for the full control loop—exhibit tight tails. These findings reveal than even in such high demand scenario, the total latency overhead never exceeds five milliseconds, which is compatible even with the tightest near-RT RIC control loop deadline of 10 ms.

Finding 5 (Etcd feasibility)**.** *Regardless of the xApp class and its state size, the communication latency introduced by etcd is negligible with respect to the near-RT control loop deadlines, making etcd a suitable solution for SDL's backend database.*

As discussed in Sec. III-B, etcd needs periodic maintenance operations, i.e., compaction and defragmentation of stale key-value pairs. Let ν be the maintenance period. The defragmentation of an etcd instance makes that instance unavailable every ν seconds. Therefore, to assess the impact of etcd maintenance on performance and resource usage, we now consider ν as a parameter for our analysis.

We start by investigating the defrag downtime T_{DF}^{SDL} as a function of the total traffic load Λ_k directed towards the etcd database. We define $\Lambda_k = N_k \cdot \omega_{s,k} / \omega_{p,k}$, where N_k is the number of concurrently active xApps of class k. We found Λ_k to be the best auxiliary metric to compactly capture—and jointly encompass—both the number of xApps and the class-specific xApp features that influence the etcd workload, thus providing the clearest visualization of our results. Since each xApp class k yields a fixed configuration of $\omega_{s,k}$ and $\omega_{p,k}$, analyzing T_{DF}^{SDL} as a function of Λ_k is equivalent to observing its relation with respect to the total number of xApps N_k.



(a) xApp non-SDL (b) xApp SDL

Figure 10: xApp resource usage, for both (a) non-SDL, and (b) SDL options and varying classes of xApp.

Fig. 9 reports T_{DF}^{SDL} as a function of Λ_k for varying classes of xApp, state size ρ, and maintenance period ν. The red dashed line in all plots underlines the exemplary 1 s control loop deadline. Some relevant findings on T_{DF}^{SDL} can be highlighted: (i) regardless of the xApp class, its trend with respect to Λ_k can be well approximated by a linear relation; (ii) it is strongly influenced by ρ, denoting a positive correlation; (iii) the dependency on ν is negligible, with the only exception of xApps class D, for which T_{DF}^{SDL} increases with ν due to the significant load etcd is subject to; and (iv) T_{DF}^{SDL} may be incompatible with the arbitrary near-RT RIC control loop deadline, which hints at scalability issues for SDL.

Finding 6 (Defrag downtime)**.** *For all classes of xApp, the defrag downtime increases linearly with the total number of xApps. It substantially increases with the xApp state size while the dependency on the maintenance period is not as strong. Also, given the near-RT RIC threshold on such downtime, scalability limits of the SDL approach emerge.*

C. Resource Usage Analysis

xApp. We now analyze SDL's impact on xApp resource utilization in terms of CPU and power consumption. Fig. 10a and 10b compare the case where the xApp allocates its state in memory (xApp non-SDL), with that where the xApp uses SDL to put its state on the backend database (xApp SDL). Both figures report CPU and power consumption as functions of the xApp state size ρ for varying xApp classes. We notice that the values of CPU and power consumption in both cases are comparable, suggesting that the way the xApp retains its state has no significant impact on its resource consumption. Results also underline that CPU and power consumption are independent of the xApp state size but strongly depend on the xApp class. In fact, the AI algorithm of an xApp of class D produces an inference on the input metrics every 100 ms, yielding a higher resource consumption than an xApp of class A, which, instead, does that every 1 s.

Finding 7 (xApp resource consumption)**.** *Regardless of the xApp state size, the use of SDL has negligible impact on the xApp resource consumption. Furthermore, the resource consumption strongly depends on the xApp class and the frequency with which its AI algorithm is executed.*

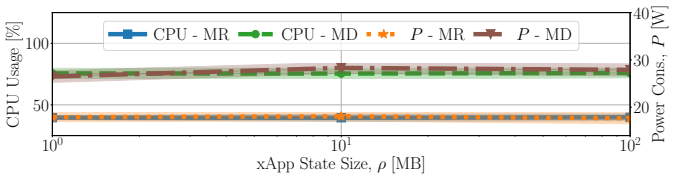


Figure 11: Resource usage for both SM-MR and SM-MD.

SM. Fig. 11 shows the instantaneous CPU usage and power consumption as functions of the xApp state size for both SM-MR and SM-MD. As discussed in Sec. III-A, the additional complexity introduced by SM-MD to attain a lower downtime with respect to SM-MR yields higher CPU and power consumption. Remarkably, regardless of the SM strategy, the instantaneous CPU usage and power consumption remain constant as the state size grows. Indeed, the dependence on state size emerges in the SM KPIs (Finding 3), yielding that, as the state size increases, this same instantaneous resource usage must be sustained for a longer duration.

Finding 8 (Resource usage)**.** *Stateful migration instantaneous CPU and power usage are independent of the xApp state size and they are functions of the selected SM strategy.*

SDL. Finally, we examine the impact of the maintenance period ν on etcd's resource consumption. Fig. 12 depicts CPU usage and power consumption of etcd as a function of the total traffic load Λ_k for different xApp classes, values of state size ρ, and ν. As expected, lower values of ν imply more frequent etcd maintenance operations, yielding an increase on CPU usage and power consumption that is up to two orders of magnitude for low values of Λ_k (e.g., comparing Fig. 12a and Fig. 12c). On the contrary, no significant impact on resource consumption is observed when ρ increases, as the amount of state size being retained does not affect CPU or power consumption. Moreover, when ν=1 s, both CPU usage and power consumption exhibit a slightly decreasing trend with respect to Λ_k. This is because etcd saturates due to: (i) the frequent maintenance operations that make etcd instances unavailable; and (ii) the increasingly high number of key-value pairs being stored/accessed by the xApps. On the other hand, when ν=120 s (i.e., when etcd is not saturating), CPU usage and power consumption grow with Λ_k. Remarkably, regardless of the values of ν and ρ, the dependency of CPU usage and power consumption upon Λ_k can be well approximated by a linear relation.

Finding 9 (Etcd CPU and power usage)**.** *Etcd instantaneous CPU and power consumption substantially decreases with the maintenance period but is practically independent of the state size. For all xApp classes, both CPU and power consumption exhibit a linear relationship with the total number of xApps.*

Similarly, Fig. 13 depicts the etcd memory and disk usage versus the total traffic load Λ_k and for varying xApp classes, values of state size ρ, and maintenance period ν. First, we notice that the impact of both ρ and ν on the results is not negligible and depends on the type of xApp. Indeed, despite increasing values of ρ and ν yield a general increase on memory and disk usage, two exceptions can be observed: (i) when xApps of class D are considered, the value of ρ has negligible impact on the memory and disk usage; and (ii) when the xApps are of class A and they feature a state size ρ=1 MB, varying the value of ν makes no significant difference in memory and disk usage. Secondly, focusing on the configurations that do not violate the near-RT RIC deadline (see Fig. 9), results show that the dependency of both memory and disk usage on the total traffic load can be well approximated by a linear relation regardless of ρ and ν.

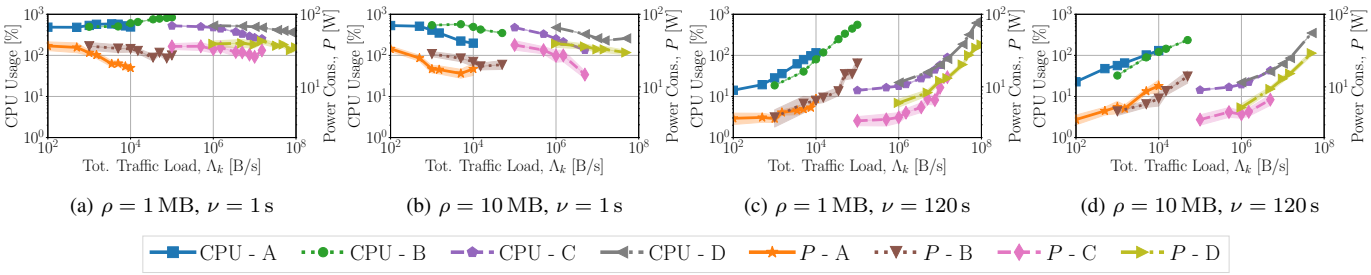
Finding 10 (Etcd memory and disk usage)**.** *Etcd memory and disk utilization depend on the xApp classes, their state size, and the value of the maintenance period. In general, both memory and disk usage exhibit a linear relation with respect to the total traffic load, i.e., the number of xApps.*

VI. Problem Formulation

Our findings show that achieving lossless migration of stateful xApps is non-trivial due to a variety of trade-offs involving resource utilization, scalability, and service availability. Cloud-native technologies allow to dynamically activate compute nodes but do not consider the strict requirements of O-RAN systems described above. For this reason, we propose CORMO-RAN, an energy-aware framework that jointly optimizes compute nodes activation and lossless xApp migration while guaranteeing uninterrupted xApp control. To integrate CORMO-RAN within the O-RAN architecture, we prototyped CORMO-RAN as an rApp running on the non-RT RIC, which is a component of the Service Management and Orchestration (SMO) framework and it is in charge of handling all orchestration, management and automation procedures to monitor and control RAN components.

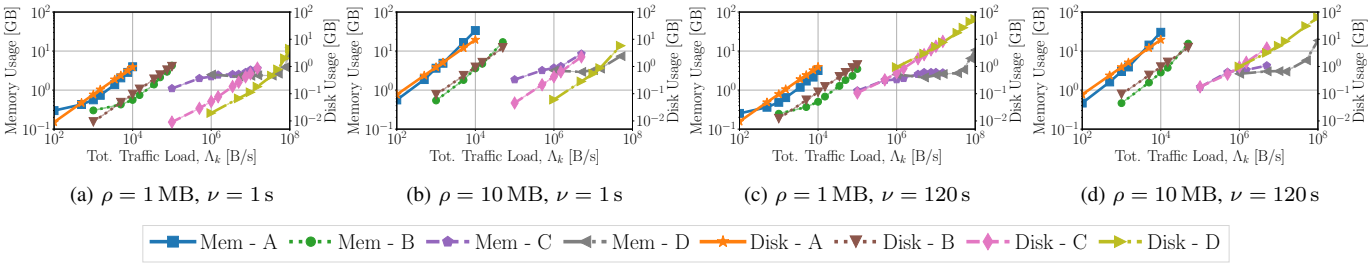
A. System Model

We consider a compute cluster of nodes, each consisting of a server. Let $\mathcal{S}$ be the set of servers. The cluster hosts the near-RT RIC along with a total number N_k of xApps for each class k. Consistently with our testbed (see Sec. IV), we consider resource-constrained and identical servers with respect to CPU, memory and disk availability. However, we remark that the notation can be easily extended to heterogeneous deployments, making CORMO-RAN independent of the specific cluster architecture and resource capabilities. For each server $s\in\mathcal{S}$, we define a binary indicator α_s that identify servers that can be turned off to save energy (i.e., α_s=1), and those that must be on always (i.e., α_s=0) such as master servers, or servers hosting the near-RT RIC and other fundamental services. We introduce a binary variable μ_s to identify which server is active (i.e., μ_s=1) or turned off (i.e., μ_s=0). We let $\boldsymbol{\mu}=(\mu_s)_{s\in\mathcal{S}}$ denote the server activation policy, and let μ_s=0 only if α_s=1.



(a) $\rho = 1\,\text{MB}$, $\nu = 1\,\text{s}$ (b) $\rho = 10\,\text{MB}$, $\nu = 1\,\text{s}$ (c) $\rho = 1\,\text{MB}$, $\nu = 120\,\text{s}$ (d) $\rho = 10\,\text{MB}$, $\nu = 120\,\text{s}$

Figure 12: Etcd CPU and power consumption for varying xApp classes, values of xApp state size ρ, and maintenance period ν.



(a) $\rho = 1\,\text{MB}$, $\nu = 1\,\text{s}$ (b) $\rho = 10\,\text{MB}$, $\nu = 1\,\text{s}$ (c) $\rho = 1\,\text{MB}$, $\nu = 120\,\text{s}$ (d) $\rho = 10\,\text{MB}$, $\nu = 120\,\text{s}$

Figure 13: Etcd memory and disk usage for varying xApp classes, values of xApp state size ρ, and maintenance period ν.

We consider a timeslot-based optimization problem where the joint server activation and xApp migration problem is solved periodically at discrete time intervals of ΔT hours. Let the superscript 0 denote the system state at the beginning of the current timeslot. For each xApp class $k \in \mathcal{K}$, $n^0_{k,s} \in \mathbb{N}^+_0$ is a non-negative integer parameter to indicate how many xApps of class k are running on server s at the beginning of the timeslot. We also consider $\boldsymbol{\mu}^0_s = (\mu^0_s)_{s\in\mathcal{S}}$ where $\mu^0_s \in \{0,1\}$ indicates whether server s is active ($\mu^0_s = 1$) at the beginning of the timeslot, or not.

For a given cluster status (e.g., defined by the number $n^0_{k,s}$ of xApps already deployed on each server s and its activation status μ^0_s), the goal of CORMO-RAN is to determine both the server activation policy $\boldsymbol{\mu}$ and the xApp migration policy $\mathbf{x}$. The latter is defined as $\mathbf{x} = (x_{k,s,s'})_{k\in\mathcal{K},(s,s')\in\mathcal{S}^2}$ where $x_{k,s,s'} \in \mathbb{N}^+_0$ is used to indicate how many xApps of class k are being reallocated from s to s'. If $s \neq s'$, $x_{k,s,s'}$ represents the number of xApps that are being migrated; if $s = s'$, $x_{k,s,s}$ represents the number of xApps that remain on s.

In addition to migration, we consider both deployment of new xApps as well as undeployment. Let n^-_k and n^+_k be the number of xApps of class k to be undeployed and deployed, respectively. Without loss of generality, we introduce a virtual server $\tilde{s} \in \mathcal{S}$ hosting all xApps to be deployed. Thus, we set $n^0_{k,\tilde{s}} = n^+_k$ for all $k \in \mathcal{K}$. Also, $\tilde{s}$ has infinite computational resources and zero energy consumption, as this server does not contribute to any utility or cost, but it is only used to simplify the notation while retaining generality. Since xApps to be undeployed become irrelevant to RAN operations, at the beginning of each slot we remove a total of n^-_k from all servers in $\mathcal{S}\backslash\{\tilde{s}\}$. In this way, $\sum_{s'\in\mathcal{S}\backslash\{s\}} x_{k,s,s'}$ represents the total number of xApps of class k to be migrated from s.

Temporal KPIs. Finding 1 suggests that memory usage and dirty-page rate are dominated by AI execution and depend on the xApp state size ρ. Finding 2 indicates that due to technical limitations, the SM strategy is incompatible with the near-RT RIC control loop deadline but still worth to consider. Since Findings 3 and 4 suggest a linear relationship, the migration downtime and the total migration duration are:

From Experimental Findings 1, 2, 3, 4

$$T^{\tau}_{\mathrm{D}_{k,s}} = \delta^{\tau}_{\mathrm{D}} \cdot \sum_{s'\in\mathcal{S}\backslash\{s\}} x_{k,s,s'} + b^{\tau}_{\mathrm{D}} \qquad (1)$$

$$T^{\tau}_{\mathrm{M}_{k,s}} = \delta^{\tau}_{\mathrm{M}} \cdot \sum_{s'\in\mathcal{S}\backslash\{s\}} x_{k,s,s'} + b^{\tau}_{\mathrm{M}}, \qquad (2)$$

where $\tau \in \{\mathrm{SDL}, \mathrm{SM{-}MR}, \mathrm{SM{-}MD}\}$ and $\delta^{\tau}_{\mathrm{D}}$, $\delta^{\tau}_{\mathrm{M}}$ are the slopes of the linear approximation we have experimentally measured from Fig. 7 for SM, and Fig. 8a for SDL, while b^{τ}_{D} and b^{τ}_{M} are the intercept for the two KPIs. The values of all parameters are summarized in Tables III, IV and V. It is worth mentioning that Finding 4 provides experimental evidence that xApps behave as stateless under SDL, which results in zero-downtime migration, i.e., $T^{\mathrm{SDL}}_{\mathrm{D}_{k,s}} = 0\ \forall k, s$. Moreover, Fig. 7 shows that $b^{\mathrm{SM-MD}}_{\mathrm{M}} = b^{\mathrm{SM-MR}}_{\mathrm{M}} = 0$ and $\delta^{\mathrm{SM-MR}}_{\mathrm{D}} = \delta^{\mathrm{SM-MR}}_{\mathrm{M}}$. Further, although our model is derived under sequential migrations of multiple xApps, the linear formulation would remain valid in the case of parallel strategies—albeit with slopes and intercepts acquiring different physical interpretations. This makes our model (i) adaptive to future SM developments that support parallel migrations, (ii) compatible with any configuration of the Kubernetes scheduler to handle batches of parallel deployments, and (iii) broadly applicable, as capturing the worst-case sequential behavior ensures that any faster parallel approach naturally falls within the bounds demonstrated in our following evaluation.

Note that the time necessary to instantiate new xApps does not depend on the specific migration strategy as the state is always empty upon instantiation. Therefore, the time to instantiate new xApps can be computed by using Fig. 8a (i.e., which corresponds to the time needed to migrate a virtually stateless xApp in SDL) and is defined as:

Table III: Experimental parameter settings for SDL, under ρ=1, ν=1 (upper), and xApp resource consumption (lower)

	$\delta^{\text{SDL}}_{E,k}$ [W]	$\delta^{\text{SDL}}_{\text{CPU},k}$	$\delta^{\text{SDL}}_{\text{MEM},k}$ [GB]	$\delta^{\text{SDL}}_{\text{DISK},k}$ [GB]	$b^{\text{SDL}}_{E,k}$ [W]	$b^{\text{SDL}}_{\text{CPU},k}$	$b^{\text{SDL}}_{\text{MEM},k}$ [GB]	$b^{\text{SDL}}_{\text{DISK},k}$ [GB]	σ_x [ms]
A	-0.18	-0.00	0.04	0.01	32.35	5.32	0.20	0.00	16.62
B	-0.09	0.03	0.04	0.01	33.60	5.57	0.17	0.00	17.07
C	-0.10	-0.03	0.02	0.00	35.48	4.97	1.82	0.00	7.71
D	-0.06	-0.01	0.08	0.03	40.20	5.00	1.04	0.00	11.62

	$p_{E,k}$ [W]	$p_{\text{CPU},k}$	$p_{\text{MEM},k}$ [GB]
A	3.43	0.47	0.52
B	16.48	2.86	0.52
C	3.43	0.47	0.52
D	16.48	2.86	0.52

Table IV: Experimental parameter settings for idle near-RT RIC consumption and SM resource usage $\forall k, \rho, \nu$

δ^{SDL}_{M} [s]	b^{SDL}_{M} [s]	$b^{\text{SM-MR}}_{\text{CPU}}$ [s]	$b^{\text{SM-MD}}_{\text{CPU}}$ [s]	$b^{\text{SM-MR}}_{E}$ [W]	$b^{\text{SM-MD}}_{E}$ [W]	q_{E_s} [W]	q_{CPU_s}	q_{MEM_s} [GB]	q_{DISK_s} [GB]
0.08	4.27	0.40	0.76	17.87	27.56	120	0.1	5.7	3.2

Table V: Experimental parameter settings for SM KPIs $\forall k, \nu$

	$\delta^{\text{SM-MR}}_{D}$ [s]	$\delta^{\text{SM-MD}}_{D}$ [s]	$\delta^{\text{SM-MD}}_{M}$ [s]
$\rho = 1$ MB	10.55	5.74	20.28
$\rho = 10$ MB	11.73	6.49	23.02
$\rho = 100$ MB	23.3	13.3	48.2

From Experimental Finding 4

$$\tilde{T}_{k,s} = \delta^{\text{SDL}}_{\text{M}} \cdot x_{k,\tilde{s},s} + b^{\text{SDL}}_{\text{M}}. \quad (3)$$

SDL feasibility. As we pointed out in Finding 5 and 6, etcd is indeed a valid solution for the SDL backend database but it is subject to scalability limits as the defrag downtime may exceed the near-RT RIC control loop deadline. To capture this aspect, we model the defrag downtime as a linear function of the total number N_k of xApps, with σ_k being the slope we experimentally measure from Fig. 9, i.e.,

From Experimental Findings 5, 6

$$T^{\text{SDL}}_{\text{DF}} = \sum_{k \in \mathcal{K}} \sigma_k N_k \,. \quad (4)$$

Moreover, we denote T_{active} as the time an xApp is active within the maintenance period ν. For etcd to be a feasible lossless xApp migration strategy in O-RAN, it must always avoid permanent service disruption, i.e., $T_{\text{active}}{=}\nu{-}T^{\text{SDL}}_{\text{DF}}{>}0$.

Resource Consumption. To model the resource consumption associated to a server s we consider three contributions: (i) the idle consumption; (ii) the load-based resource consumption, which scales linearly with the number of xApps hosted by s [58]; and (iii) the resource consumption required to execute the specific migration strategy.

The general resource consumption model for any server $s{\in}\mathcal{S}\backslash\{\tilde{s}\}$ with respect to migration strategy τ is:

From Experimental Finding 7

$$R^{\tau}_{\chi_s} = \mu_s q_{\chi_s} + \sum_{k \in \mathcal{K}} \sum_{s' \in \mathcal{S}} p_{\chi_k} x_{k,s,s'} + \tilde{R}^{\tau}_{\chi_s} \quad (5)$$

where $\chi{\in}\{\text{CPU}, \text{MEM}, \text{DISK}\}$ is the type of resource, used to indicate CPU, memory and disk resources, respectively.

The first term in (5) represents the idle consumption q_{χ_s} when the server is active (i.e., $\mu_s{=}1$). The second term considers the load-based consumption observed in Finding 7, where p_{χ_k} is the slope of the linear approximation evaluated experimentally. Disk resources leveraged by our xApps (Sec. IV) are negligible, yielding $p_{\text{DISK},k}{=}0$. The other values for q_{χ_s} and p_{χ_k} are summarized in Tables IV and III. The third element captures the intrinsic resource consumption of both SM and SDL on each server s defined as:

From Experimental Findings 8, 9, 10

$$\tilde{R}^{\text{SDL}}_{\chi_s} = \frac{1}{|\mathcal{S}|} \cdot \sum_{k \in \mathcal{K}} \left(\delta^{\text{SDL}}_{\chi_k} N_k + b^{\text{SDL}}_{\chi_k} \right) \quad (6)$$

$$\tilde{R}^{\text{SM-MR}}_{\chi_s} = b^{\text{SM-MR}}_{\chi} \quad (7)$$

$$\tilde{R}^{\text{SM-MD}}_{\chi_s} = b^{\text{SM-MD}}_{\chi} \quad (8)$$

Accordingly, the SDL resource consumption, modeled in (6), is equally distributed across all servers and linearly depend on the total number N_k of xApps, with slope $\delta^{\text{SDL}}_{\chi_k}$ and intercept $b^{\text{SDL}}_{\chi_k}$. Also, the CPU consumption for SM is practically constant regardless of the value of state size, and only depends on the specific SM strategy being employed. Moreover, the consumption of memory and disk resources are negligible, i.e., $b^{\tau}_{\text{MEM}}{=}b^{\tau}_{\text{DISK}}{=}0$ for $\tau{\in}\{\text{SM-MR}, \text{SM-MD}\}$.

Energy Consumption. To evaluate energy consumption of each migration strategy, we need to consider the energy consumed by resource utilization due to xApp execution, as well as the energy caused by the migration process itself. From Finding 8, the energy consumption caused by SM is:

From Experimental Finding 8

$$E^{\tau}_{s} = b^{\tau}_{E} \sum_{k \in \mathcal{K}} T^{\tau}_{\text{M}_{k,s}}, \quad \tau{\in}\{\text{SM-MR}, \text{SM-MD}\} \quad (9)$$

where, $T^{\tau}_{\text{M}_{k,s}}$ is defined in (2), and b^{τ}_{E} represents the measured constant power consumption as reported in Tables III and IV.

With respect to SDL, Findings 9 and 10 show that the energy associated to SDL linearly depends on the total number N_k of xApps. Similarly to (6), this energy cost is distributed across the $|\mathcal{S}|$ servers, and the SDL energy cost per server s is:

From Experimental Findings 9, 10

$$E_s^{\mathrm{SDL}} = \frac{\Delta T}{|\mathcal{S}|} \cdot \sum_{k \in \mathcal{K}} \left(\delta_{\mathrm{E}_k}^{\mathrm{SDL}} N_k + b_{\mathrm{E}_k}^{\mathrm{SDL}} \right) \tag{10}$$

where $\delta_{\chi_k}^{\mathrm{SDL}}$ and $b_{\chi_k}^{\mathrm{SDL}}$ are reported in Table III.

In (10), the cost to maintain SDL is continuous over the entire optimization interval ΔT as the states of the xApps need to be continuously updated in the backend database. This substantially differs from SM where the cost of maintaining the state is incurred only for the duration of the migration process. However, we also notice that the migration process prevents servers from being turned off before the migrated xApps are activated on the destination server, yielding an extra active time that is in the order of a few seconds for SDL, but reaches several hundreds of seconds for SM. Hence, the total energy consumption of the system is

$$E_s = E_s^{\tau} + \sum_{k \in \mathcal{K}} \left(T_{\mathrm{M}_{k,s}}^{\tau} + \tilde{T}_{k,s} \right) \cdot \left(q_{\mathrm{E}_s} + \sum_{k \in \mathcal{K}} p_{\mathrm{E}_k} n_{k,s}^0 \right) + \\ + \left[\Delta T - \sum_{k \in \mathcal{K}} \left(T_{\mathrm{M}_{k,s}}^{\tau} + \tilde{T}_{k,s} \right) \right] \cdot \left(\mu_s q_{\mathrm{E}_s} + \sum_{k \in \mathcal{K}} \sum_{s' \in \mathcal{S}} p_{\mathrm{E}_k} x_{k,s',s} \right), \tag{11}$$

where E_s^{τ} is defined in (9) or (10) based on the migration strategy $\tau \in \{\mathrm{SDL}, \mathrm{SM{-}MR}, \mathrm{SM{-}MD}\}$ being selected. The second term in (11) accounts for the energy consumed during the migration process, and the third term accounts for the energy consumed by the server to execute the xApps it hosts.

B. Formulating the Problem

We can now formulate the joint Server Activation and Lossless stateful xApp migration (SAL) problem:

$$\min_{\mathbf{x}, \boldsymbol{\mu}} \quad \sum_{s \in \mathcal{S}} E_s \tag{SAL}$$

$$\text{s.t.} : \sum_{s' \in \mathcal{S}} x_{k,s,s'} = n_{k,s}^0 \quad \forall (k,s) \in \mathcal{K} \times \mathcal{S} \tag{12}$$

$$\sum_{k \in \mathcal{K}} \sum_{s \in \mathcal{S}} x_{k,s,\tilde{s}} = 0 \tag{13}$$

$$\sum_{s \in \mathcal{S} \setminus \{\tilde{s}\}} x_{k,\tilde{s},s} = n_{a,\tilde{s}}^0 \quad \forall k \in \mathcal{K} \tag{14}$$

$$\sum_{k \in \mathcal{K}} \sum_{s' \in \mathcal{S} \setminus \{s\}} x_{k,s,s'} \leq M \mu_s^0 \quad \forall s \in \mathcal{S} \tag{15}$$

$$\sum_{k \in \mathcal{K}} \sum_{s' \in \mathcal{S}} x_{k,s',s} \leq M \mu_s \quad \forall s \in \mathcal{S} \tag{16}$$

$$\mu_s \leq \sum_{k \in \mathcal{K}} \sum_{s' \in \mathcal{S}} x_{k,s',s} \quad \forall s \in \mathcal{S} \tag{17}$$

$$R_{\chi_s} \leq R_{\chi_s}^{\mathrm{MAX}} \mu_s \quad \forall s \in \mathcal{S} \tag{18}$$

$$\mu_s \geq 1 - \alpha_s \quad \forall s \in \mathcal{S} \tag{19}$$

$$\sum_{k \in \mathcal{K}} T_{D_{k,s}}^{\tau} \leq T_{D_s}^{\max} \quad \forall s \in \mathcal{S} \tag{20}$$

$$T_{\mathrm{DF}}^{\mathrm{SDL}} < T_{\mathrm{DF}}^{\max} \quad \forall s \in \mathcal{S} \tag{21}$$

$$T_{\mathrm{active}} > 0 \quad \forall s \in \mathcal{S} \tag{22}$$

where $E_s(\cdot)$ is defined in (11), $\chi \in \{\mathrm{CPU}, \mathrm{MEM}, \mathrm{DISK}\}$, and $\tau \in \{\mathrm{SDL}, \mathrm{SM{-}MR}, \mathrm{SM{-}MD}\}$. Constraint (12) ensures that we migrate only active xApps, and that we allocate all required xApps (those in the virtual server and those already deployed). Constraints (13) and (14) ensure that no xApps remain on the virtual server. Constraint (15) imposes that xApps are instantiated on active servers only. Constraints (16) and (17) ensure that we migrate xApps only from active servers and we shut down inactive servers, where M is any large number such that $M > \sum_{k \in \mathcal{K}} \sum_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} x_{k,s,s'}$. Constraint (18) enforces resource constraints on each server. Constraint (19) makes sure that we shut down only servers that can be turned off (i.e., with $\alpha_s = 1$). Constraint (20) imposes that the downtime due to xApps being migrated to s for any migration strategy τ is below a tolerable threshold $T_{D_k}^{\max}$. Finally, Constraints (21) and (22) enforce SDL feasibility with respect to an arbitrary near-RT RIC control loop deadline $T_{\mathrm{DF}}^{\max}$, chosen, for instance, to ensure service continuity of the xApp with the tightest timing constraints.

Theorem 1. *Problem* (SAL) *is NP-hard.*

Proof. The (SAL) problem is a mixed integer quadratic programming (MIQP) problem as it involves both binary ($\boldsymbol{\mu}$) and integer ($\mathbf{x}$) variables. It is well-known that the general decision version of MIQPs is NP-complete [59]. Being Problem (SAL) a MIQP, we can build a polynomial-time reduction to the general formulation of MIQP in [59], which proves that Problem (SAL) is NP-hard by reduction. □

C. Solving the SAL Problem

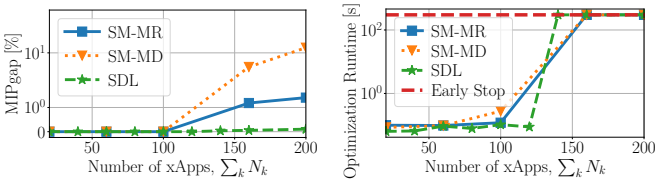
Although SAL problem is NP-hard, it can be solved optimally via branch-and-bound (B&B) where the original problem is transformed into its linear-programming relaxation and is iteratively solved by exploring the branches and assessing the integrality (and binary) constraints of variables. This process can also be made more efficient using cutting planes that exclude inefficient branches. It has been shown [60] that polynomial-time ϵ-approximation algorithms for MIQP exist. How to build such polynomial approximation for the SAL problem is out of the scope of this paper, but, as shown in Sec. VII, the SAL problem can still be optimally solved within 1 second even in the case of 100 xApps to be migrated.

VII. CORMO-RAN Evaluation

To evaluate CORMO-RAN and compute an optimal solution to the SAL Problem, we use MATLAB and Gurobi on a server with Intel Xeon E5-2680 with 28 cores and 16 GB of RAM.

We consider a cluster of four nodes, hosting the near-RT RIC components as well as a varying number of xApps, and, for each value, we consider 75% of them to be of class k (which corresponds to the dominant class) and the remaining 25% to be evenly distributed among the other classes. To be consistent with our testbed in Sec. IV, we set $R_{\mathrm{CPU}_s}^{\max} = 128$ (virtual) CPU cores, $R_{\mathrm{MEM}_s}^{\max} = 125$ GB, and $R_{\mathrm{DISK}_s}^{\max} = 250$ GB and consider realistic values for the temporal parameters: $\Delta T = 1$ h, i.e., running CORMO-RAN optimization cycles on an hourly basis, $T_{D_k}^{\max} = 300$ s, i.e., the arbitrary

maximum stateful migration downtime that can be tolerated, and $T_{\mathrm{DF}}^{\max}$=1 s, i.e., the near-RT deadline that must not be exceeded while performing periodic SDL maintenance. It is worth mentioning that the choice of ΔT depends on how rapidly traffic fluctuates across the cells controlled by the near-RT RIC. While we consider CORMO-RAN optimization cycles to run on an hourly basis and as an rApp within the non-RT RIC, the ultimate decision of when and whether to trigger CORMO-RAN is driven by traffic dynamics and is thus left to the network operator. Indeed, our formulation is compatible not only with periodic executions but also with event-based triggering, e.g., upon detecting workload increases or any other relevant condition identified through continuous monitoring operations.



(a) (b)

Figure 14: CORMO-RAN performance: (a) MIPgap and (b) runtime for k=A, ρ=1 MB, and ν=1 s.

Fig. 14 shows the optimization performance for varying number of xApps and for the following exemplary configuration: dominant xApp class k=A, state size ρ=1 MB, and maintenance period ν=1 s. Results demonstrate that up to about 120 xApps SAL can be solved optimally and within 1 second, regardless of the migration strategy being used. As the complexity of the scenario increases, i.e., the number of xApps grows above 120, the optimization runtime reaches the early stop deadline, i.e., 300 s, but still yielding a reasonably small MIPgap (up to 10% in the case of SM-MD and 200 xApps). We thus conclude that, despite being NP-hard, SAL can be solved optimally without algorithmic approximations.

Fig. 15 shows the energy gain and servers activation ratio as functions of the migration strategy and for varying configurations of dominant xApp class k, xApp state size ρ and maintenance period ν. We compute the energy gain with respect to a baseline in which all compute servers remain permanently active and xApps are placed according to the resource-based load balancing scheme that is natively implemented in OpenShift—a widely adopted approach in practice and frequently considered in the literature [61], [62], [63]. We recall that, to the best of our knowledge, no prior work addresses xApp migration in the O-RAN context; therefore, the chosen baseline both reflects the state of the art and provides a well-grounded reference for quantifying the benefits introduced by our framework. It can be observed that, by turning off compute servers that are not required during low traffic periods, CORMO-RAN attains a significant reduction in energy consumption. As the number of xApps grows, a higher number of active servers is needed, yielding an increased activation ratio and a reduced energy gain. Such gain approaches 0% when the activation ratio is 1, i.e., same energy consumption as the baseline. Notably, both energy gain and activation ratio strongly depend on the configuration that is set: (i) as the dominant xApp class changes from low to high demanding, e.g., from A to B (Fig. 15a vs Fig. 15g) or from C to D (Fig. 15i vs Fig. 15k), the energy gain decreases with higher pace and a fewer number of xApps can be hosted due to constraint (20), i.e., the one on the resource usage; (ii) looking at, e.g., Fig. 15a and Fig. 15e, the larger ρ, the smaller the number of xApps that can be migrated compatibly with the maximum downtime (see constraint (19)); (iii) comparing, e.g., Fig. 15a and Fig. 15c, when the value of ν increases from 1 s to 120 s, the cost due to SDL maintenance is reduced, yielding a higher energy gain and lower activation ratio; and (iv) in general, comparing to SDL, SM strategies achieve higher values of energy gain (up to 64%) as they do not require the additional cost to host and maintain the SDL backend database.

Figures 16 and 17 show the feasibility region of, respectively, SDL and SM migration strategies for varying configurations of dominant xApp class k, xApp state size ρ and maintenance period ν under $T_{D_k}^{\max}$=300 s and $T_{\mathrm{DF}}^{\max}$=1 s. To compute such regions we enforce (20) for SM, and, (21) and (22) for SDL. Fig. 16 demonstrates that, due to scalability limits, the feasibility of SDL-based migration strongly depends on the values of ρ and ν: (i) when ρ increases, the maximum number of xApps that SDL can host (compatibly with the near-RT RIC strict timing requirements) decreases, up to ρ=100 MB for which no configuration is actually feasible, regardless of the number of xApps and the dominant class; and (ii) when ν increases, despite the higher energy gain observed in Fig. 15, the maximum number of xApps significantly decreases, due to higher values of the defrag downtime that lead to near-RT RIC deadline violation. On the other hand, Fig. 17 shows that SM is way more feasible, allowing also for the extreme scenario of ρ=100 MB, and SM-MD attains higher feasibility values thanks to migration downtime minimization. We recall that, despite its limited feasibility, SDL is the only strategy that enables zero-downtime migration process. SM, instead, implies a migration downtime that is way above the near-RT RIC deadline and needs to be accounted for (see Fig. 2a).

Thus, we conclude that CORMO-RAN effectively addresses the trade-off among service availability, scalability, and energy consumption. In the case of large deployments all servers need to be active and CORMO-RAN has no significant impact on the energy consumption. On the other hand, when the traffic load is low and the number of xApps is small, e.g., at nighttime, CORMO-RAN allows to identify, for varying system configurations, which migration strategy is feasible and its effectiveness in reducing the overall energy consumption, yielding a cost reduction that is up to 64%.

VIII. Conclusions

In this paper, we proposed CORMO-RAN, a data-driven orchestrator that jointly optimizes the activation of near-RT RIC compute nodes and the migration of stateful xApps to minimize the overall system energy consumption, while ensuring uninterrupted xApp control. We first introduced the two key technologies for preserving the xApp internal state

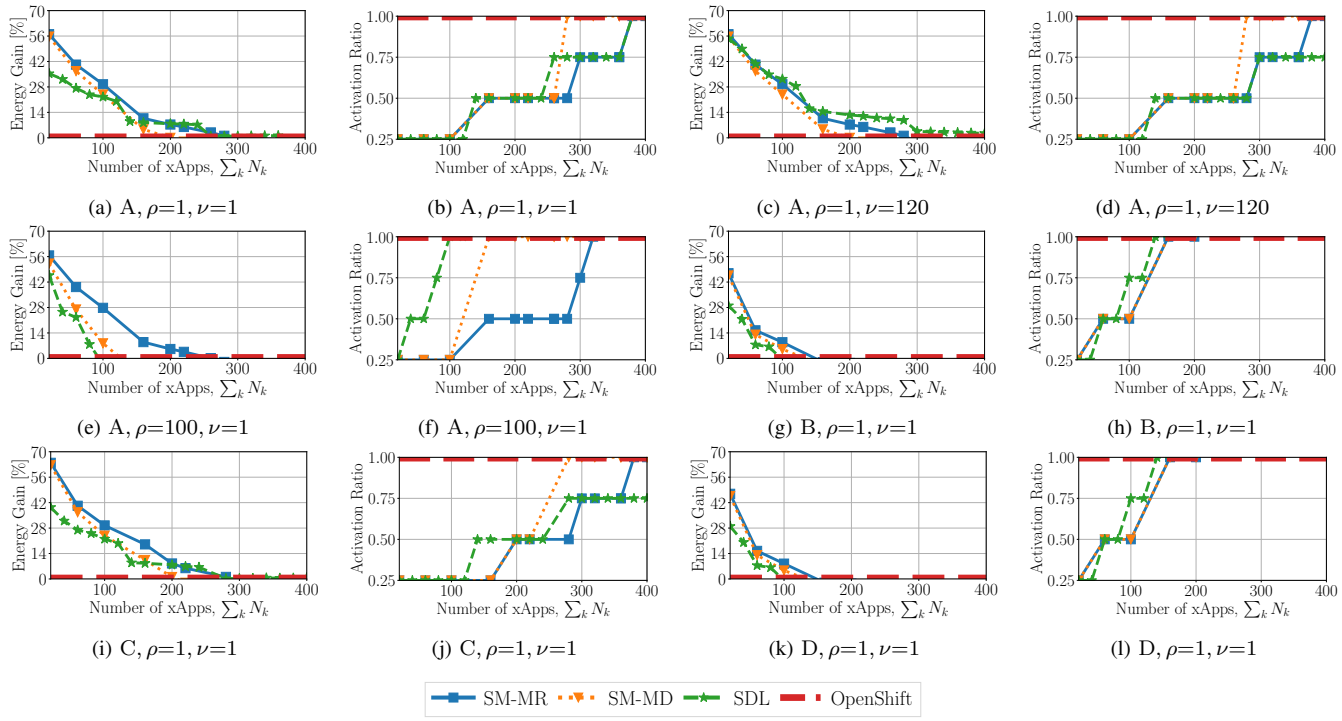


Figure 15: CORMO-RAN energy consumption reduction with respect to the OpenShift default scheduler and servers activation ratio for varying: (i) dominant xApp class (75% distribution); (ii) xApp state size ρ; (iii) maintenance period ν; (iv) migration strategy.

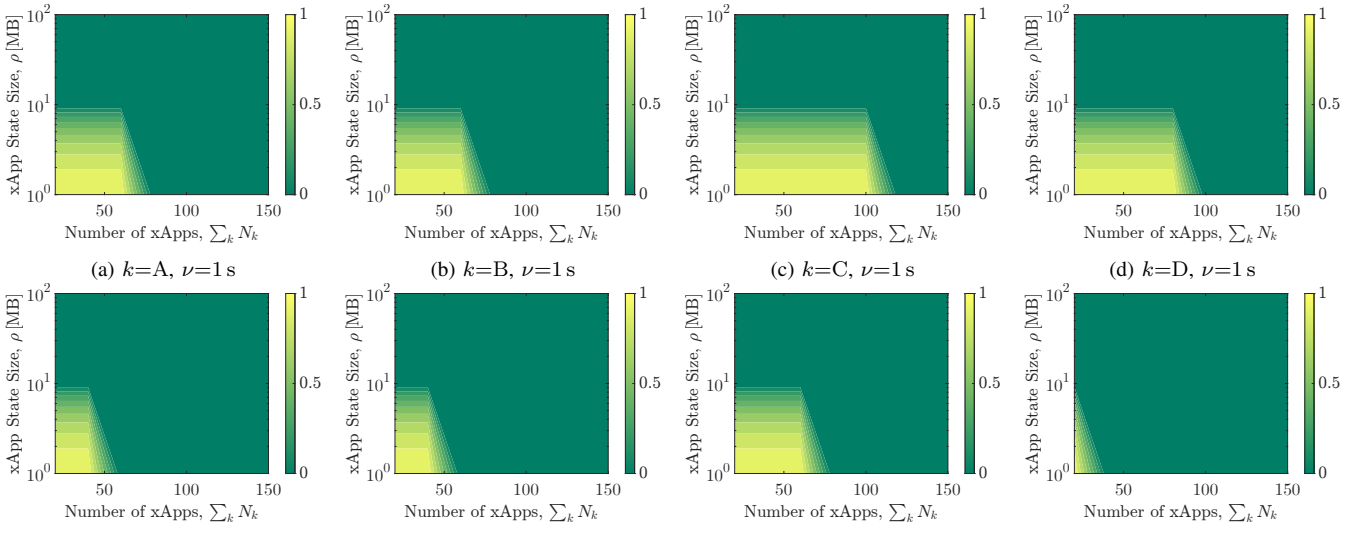


Figure 16: CORMO-RAN feasibility analysis for SDL-based migration and varying class k, state size ρ and maintenance period ν.

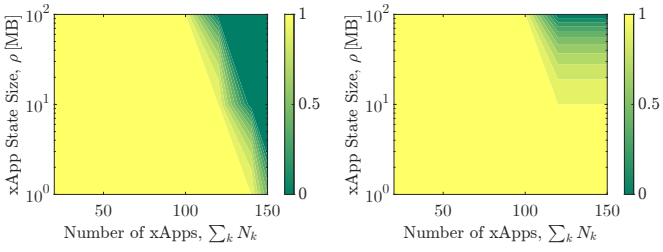


Figure 17: CORMO-RAN feasibility analysis under (a) SM-MR and (b) SM-MD for varying xApp state size ρ.

upon migration, i.e., SM and SDL, while accounting for the O-RAN context and time constraints. Then, we leveraged our experimental testbed based on Red Hat OpenShift to perform a thorough temporal KPIs and resource usage analysis under both migration strategies and varying use case scenarios, revealing pivotal trade-offs involving resource usage, scalability, and service availability. Our results demonstrate that CORMO-RAN accurately identifies feasibility and effectiveness of each migration strategy and computes the optimal xApp allocations across the available compute nodes, yielding up to 64% reduction of the system energy consumption.

REFERENCES

[1] O-RAN Alliance, "O-RAN WhitePaper - Building the Next Generation RAN," https://www.o-ran.org/resources, October 2018.

[2] 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.

[3] S. Maxenti, S. D'Oro, L. Bonati, M. Polese, A. Capone, and T. Melodia, "ScalO-RAN: Energy-aware Network Intelligence Scaling in Open RAN," in *Proceedings of International Conference on Computer Communications (INFOCOM)*. IEEE, 2024.

[4] A. Calagna, Y. Yu, P. Giaccone, and C. F. Chiasserini, "Design, Modeling, and Implementation of Robust Migration of Stateful Edge Microservices," *IEEE Transactions on Network and Service Management*, vol. 21, no. 2, pp. 1877–1893, 2024.

[5] X. Wang, A. V. Vasilakos, M. Chen, Y. Liu, and T. T. Kwon, "A survey of green mobile networks: Opportunities and challenges," *Mobile Networks and Applications*, vol. 17, pp. 4–20, 2012.

[6] M. Masoudi, M. G. Khafagy, A. Conte, A. El-Amine, B. Françoise, C. Nadjahi, F. E. Salem, W. Labidi, A. Süral, A. Gati, D. Bodéré, E. Arikan, F. Aklamanu, H. Louahlia-Gualous, J. Lallet, K. Pareek, L. Nuaymi, L. Meunier, P. Silva, N. T. Almeida, T. Chahed, T. Sjölund, and C. Cavdar, "Green Mobile Networks for 5G and Beyond," *IEEE Access*, vol. 7, pp. 107 270–107 299, 2019.

[7] D. López-Pérez, A. De Domenico, N. Piovesan, G. Xinli, H. Bao, S. Qitao, and M. Debbah, "A survey on 5G radio access network energy efficiency: Massive MIMO, lean carrier design, sleep modes, and machine learning," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 1, pp. 653–697, 2022.

[8] G. Baldini, R. Bolla, R. Bruschi, A. Carrega, F. Davoli, C. Lombardo, and R. Rabbani, "Toward Sustainable O-RAN Deployment: An In-Depth Analysis of Power Consumption," *IEEE Transactions on Green Communications and Networking*, pp. 1–1, 2024.

[9] L. M. P. Larsen, H. L. Christiansen, S. Ruepp, and M. S. Berger, "Toward Greener 5G and Beyond Radio Access Networks — A Survey," *IEEE Open Journal of the Communications Society*, vol. 4, pp. 768–797, 2023.

[10] X. Liang, Q. Wang, A. Al-Tahmeesschi, S. B. Chetty, D. Grace, and H. Ahmadi, "Energy Consumption of Machine Learning Enhanced Open RAN: A Comprehensive Review," *IEEE Access*, vol. 12, pp. 81 889–81 910, 2024.

[11] L. M. Larsen, H. L. Christiansen, S. Ruepp, and M. S. Berger, "The evolution of mobile network operations: A comprehensive analysis of open RAN adoption," *Computer Networks*, vol. 243, p. 110292, 2024.

[12] L. Kundu, X. Lin, and R. Gadiyar, "Toward energy efficient RAN: From industry standards to trending practice," *IEEE Wireless Communications*, vol. 32, no. 1, pp. 36–43, 2025.

[13] K. Ramezanpour and J. Jagannath, "Intelligent zero trust architecture for 5G/6G networks: Principles, challenges, and the role of machine learning in the context of O-RAN," *Computer Networks*, vol. 217, p. 109358, 2022.

[14] M. Tsampazi *et al.*, "PandORA: Automated Design and Comprehensive Evaluation of Deep Reinforcement Learning Agents for Open RAN," *IEEE Transactions on Mobile Computing*, vol. 24, no. 4, 2025.

[15] J. Dai, L. Li, R. Safavinejad, S. Mahboob, H. Chen, V. V. Ratnam, H. Wang, J. Zhang, and L. Liu, "O-RAN-Enabled Intelligent Network Slicing to Meet Service-Level Agreement (SLA)," *IEEE Transactions on Mobile Computing*, vol. 24, no. 2, pp. 890–906, 2025.

[16] M. Dryjański, Ł. Kułacz, and A. Kliks, "Toward modular and flexible open ran implementations in 6g networks: Traffic steering use case and o-ran xapps," *Sensors*, vol. 21, no. 24, 2021.

[17] M. Catalan-Cid, J. Pueyo, J. Sanchez-Gonzalez, J. Gutierrez, and M. Ghoraishi, "BeGREEN Intelligent Plane for AI-driven Energy Efficient O-RAN management," in *Proceedings of EuCNC & 6G Summit*. IEEE, 2024, pp. 1–6.

[18] F. Mungari, C. Puligheddu, A. Garcia-Saavedra, and C. F. Chiasserini, "O-RAN Intelligence Orchestration Framework for Quality-Driven xApp Deployment and Sharing," *IEEE Transactions on Mobile Computing*, vol. 24, no. 6, pp. 4811–4828, 2025.

[19] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A Survey on Service Migration in Mobile Edge Computing," *IEEE Access*, vol. 6, 2018.

[20] M. Terneborg, J. K. Rönnberg, and O. Schelén, "Application Agnostic Container Migration and Failover," in *Proceedings of Conference on Local Computer Networks (LCN)*. IEEE, 2021, pp. 565–572.

[21] C. Rong, J. H. Wang, J. Wang, Y. Zhou, and J. Zhang, "Live Migration of Video Analytics Applications in Edge Computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 3, pp. 2078–2092, 2024.

[22] Y. Li, S. Wang, Y. Li, A. Zhou, M. Xu, X. Ma, and Y. Liu, "Seamless Cross-Edge Service Migration for Real-Time Rendering Applications," *IEEE Transactions on Mobile Computing*, vol. 23, no. 6, 2024.

[23] A. Calagna, Y. Yu, P. Giaccone, and C. F. Chiasserini, "MOSE: A Novel Orchestration Framework for Stateful Microservice Migration at the Edge," *IEEE Trans. on Network and Service Management*, 2025.

[24] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic Service Migration in Mobile Edge Computing Based on Markov Decision Process," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1272–1288, 2019.

[25] A. Mukhopadhyay, G. Iosifidis, and M. Ruffini, "Migration-Aware Network Services With Edge Computing," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1458–1471, 2022.

[26] G. Panek, P. Matysiak, N. E.-h. Nouar, I. Fajjari, and H. Tarasiuk, "5G-Edge Relocator: A Framework for Application Relocation in Edge-enabled 5G System," in *Proceedings of International Conference on Communications*. IEEE, 2023, pp. 4885–4891.

[27] K. Afachao, A. M. Abu-Mahfouz, and G. P. Hanke, "Efficient Microservice Deployment in the Edge-Cloud Networks With Policy-Gradient Reinforcement Learning," *IEEE Access*, vol. 12, 2024.

[28] M. Adeppady, Y. Yu, A. Rahmanian, C. F. Chiasserini *et al.*, "Efficient management of composite edge applications," in *Proceedings of Global Communications Conference (GLOBECOM)*. IEEE, 2025.

[29] R. Laigner, Y. Zhou, and M. A. V. Salles, "A distributed database system for event-based microservices," in *Proceedings of the International Conference on Distributed and Event-Based Systems*. ACM, 2021.

[30] R. Laigner, Y. Zhou, M. A. V. Salles, Y. Liu, and M. Kalinowski, "Data management in microservices: state of the practice, challenges, and research directions," *Proceedings of the VLDB Endowment*, 2021.

[31] A. Calagna, S. Ravera, and C. F. Chiasserini, "Enabling efficient collection and usage of network performance metrics at the edge," *Computer Networks*, vol. 262, p. 111158, 2025.

[32] B. Gómez, S. Bayhan, E. Coronado, J. Villalón, and A. Garrido, "ODESA: Load-Dependent Edge Server Activation for Lower Energy Footprint," in *Proceedings of Wireless Communications and Networking Conference (WCNC)*. IEEE, 2024, pp. 1–6.

[33] M. Avgeris, D. Spatharakis, D. Dechouniotis, A. Leivadeas, V. Karyotis, and S. Papavassiliou, "ENERDGE: Distributed energy-aware resource allocation at the edge," *Sensors*, vol. 22, no. 2, 2022.

[34] U. Kulkarni, A. Sheoran, and S. Fahmy, "The cost of stateless network functions in 5G," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems*. ACM, 2022, p. 73–79.

[35] T. Erl, *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, 2nd ed. Prentice Hall Press, 2016.

[36] etcd team, "A distributed, reliable key-value store for the most critical data of a distributed system," https://etcd.io, 2013-2024.

[37] D. Ongaro and J. Ousterhout, "Raft: In search of an understandable consensus algorithm," in *Proceedings of the USENIX Annual Technical Conference*, 2014, p. 305–320.

[38] S. Gilbert and N. A. Lynch, "Perspectives on the cap theorem," *Computer*, vol. 45, no. 02, pp. 30–36, 2012.

[39] Redis team, "An in-memory database that persists on disk," https://redis.io and https://github.com/redis/redis, 2020-2024.

[40] etcd team, "etcd versus other key-value stores," https://etcd.io/docs/v3.6/learning/why/, 2025.

[41] J. C. Corbett *et al.*, "Spanner: Google's globally distributed database," *Transactions on Computer Systems*, vol. 31, no. 3, Aug. 2013.

[42] J. Zhou *et al.*, "FoundationDB: A distributed unbundled transactional key value store," in *Proceedings of the International Conference on Management of Data*. ACM, 2021.

[43] O-RAN Software Community, "RIC Platform GitHub Repository," https://github.com/o-ran-sc/ric-plt-ric-dep, 2024.

[44] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "NeutRAN: An Open RAN Neutral Host Architecture for Zero-Touch RAN and Spectrum Sharing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 1–13, August 2023.

[45] Red Hat, "Red Hat OpenShift Platform Plus," https://www.redhat.com/en/resources/openshift-platform-plus-datasheet and https://github.com/openshift, 2011-2025.

[46] Prometheus, "Open-source systems monitoring and alerting toolkit," https://prometheus.io and https://github.com/prometheus/prometheus, 2015-2025.

[47] Kepler, "Kubernetes-based Efficient Power Level Exporter," https://sustainable-computing.io/ and https://github.com/sustainable-computing-io/kepler, 2015-2025.

[48] Cloud Native Computing Foundation (CNCF): Environmental Sustainability, "Idle Power Matters: Kepler Metrics for Public Cloud Energy Efficiency," https://tag-env-sustainability.cncf.io/blog/2024-06-idle-power-matters-kepler-metrics-for-public-cloud-energy-efficiency/, 2024.
[49] M. Amaral, H. Chen, T. Chiba, R. Nakazawa, S. Choochotkaew, E. K. Lee, and T. Eilam, "Kepler: A Framework to Calculate the Energy Consumption of Containerized Applications," in *Proceedings of International Conference on Cloud Computing (CLOUD)*, 2023, pp. 69–71.
[50] C. Centofanti, J. Santos, V. Gudepu, and K. Kondepu, "Impact of power consumption in containerized clouds: A comprehensive analysis of open-source power measurement tools," *Computer Networks*, 2024.
[51] M. Akbari, R. Bolla, R. Bruschi, F. Davoli, C. Lombardo, and B. Siccardi, "A Monitoring, Observability and Analytics Framework to Improve the Sustainability of B5G Technologies," in *Proceedings of ICC Workshops*, 2024.
[52] WiNES Lab, "Colosseum O-RAN COMMAG Dataset GitHub Repository," https://github.com/wineslab/colosseum-oran-commag-dataset, 2021.
[53] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia, "Intelligence and learning in O-RAN for data-driven NextG cellular networks," *IEEE Communications Magazine*, vol. 59, no. 10, pp. 21–27, 2021.
[54] WiNES Lab, "xDevSM-xapps-examples," https://github.com/wineslab/xDevSM-xapps-examples, 2024.
[55] CRIU, "Checkpoint/restore," https://criu.org/Checkpoint/Restore and https://github.com/checkpoint-restore/criu, 2017.
[56] The Containers Organization, "Podman," https://github.com/containers/podman/ and https://podman.io/, 2022.
[57] Y. Gao, Y. Liu, H. Zhang, Z. Li, Y. Zhu, H. Lin, and M. Yang, "Estimating GPU memory consumption of deep learning models," in *Proceedings of ESEC/FSE*. ACM, 2020, p. 1342–1352.
[58] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, p. 13–23, 2007.
[59] A. D. Pia, S. S. Dey, and M. Molinaro, "Mixed-integer quadratic programming is in NP," *Mathematical Programming*, vol. 162, 2017.
[60] A. D. Pia, "An approximation algorithm for indefinite mixed integer quadratic programming," *Mathematical Programming*, vol. 201, 2023.
[61] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 1, p. 45–52, Jan. 2011.
[62] A. Bauer, V. Lesch, L. Versluis, A. Ilyushkin, N. Herbst, and S. Kounev, "Chamulteon: Coordinated auto-scaling of micro-services," in *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 2015–2025.
[63] A. Gulati, G. Shanmuganathan, A. Holler, and I. Ahmad, "Cloud scale resource management: Challenges and techniques," in *Proceedings of USENIX Workshop on Hot Topics in Cloud Computing*, 2011.

Antonio Calagna is a Postdoctoral Researcher with Politecnico di Torino, Italy. He received the B.Sc. degree in Electronics Engineering in 2019, the M.Sc. degree in Communication and Computer Networks Engineering in 2021, and the Ph.D. degree (cum laude) in Electrical, Electronics and Communications Engineering in 2025, all from Politecnico di Torino. His research interests include the orchestration and management of AI-driven edge services for next-generation mobile networks, with an emphasis on time-sensitive resource allocation and service optimization. He received the Marie Skłodowska-Curie Actions (MSCA) Seal of Excellence in 2025.

Stefano Maxenti is a Ph.D. Candidate at the Institute for the Wireless Internet of Things at Northeastern University, under Prof. Tommaso Melodia. He received a Bachelor's degree in Engineering of Computing Systems in 2020 and a Master of Science degree in Telecommunication Engineering from Politecnico di Milano, Italy. He is interested in AI applications for wireless communications and orchestration, integration, and automation of O-RAN networks.

Leonardo Bonati is an Associate Research Scientist at the Institute for the Wireless Internet of Things, Northeastern University, Boston, MA, USA. He received a Ph.D. degree in Computer Engineering from Northeastern University in 2022. His main research focuses on softwarized approaches for the Open Radio Access Network (RAN) of the next generation of cellular networks, on O-RAN-managed networks, and on network automation, orchestration, and virtualization. He was awarded the 2024 Mario Gerla Award for Research in Computer Science. Leonardo was TPC co-chair for the IEEE DTwin 2025 workshop, a co-chair for the track on Testbeds, Experimentation and Datasets for Communications and Networking of IEEE CCNC 2025, and a guest editor of the special issue of Elsevier Computer Networks on Advances in Experimental Wireless Platforms and Systems.

Salvatore D'Oro is a Research Associate Professor at Northeastern University, CTO and co-founder of zTouch Networks. He received his Ph.D. degree from the University of Catania and is an area editor of IEEE Vehicular Technology Magazine and Elsevier Computer Communications. He serves on the TPC of IEEE INFOCOM, IEEE CCNC & ICC and IFIP Networking. He is one of the contributors to OpenRAN Gym, the first open-source research platform for AI/ML applications in the Open RAN. His research interests include optimization, AI & network slicing for NextG Open RANs.

Tommaso Melodia is the William Lincoln Smith Chair Professor with the Department of Electrical and Computer Engineering at Northeastern University in Boston. He is also the Founding Director of the Institute for the Wireless Internet of Things and the Director of Research for the PAWR Project Office. He received his Ph.D. in Electrical and Computer Engineering from the Georgia Institute of Technology in 2007. He is a recipient of the National Science Foundation CAREER award. Prof. Melodia has served as Associate Editor of IEEE Transactions on Wireless Communications, IEEE Transactions on Mobile Computing, Elsevier Computer Networks, among others. He has served as Technical Program Committee Chair for IEEE INFOCOM 2018, General Chair for IEEE SECON 2019, ACM Nanocom 2019, and ACM WUWnet 2014. Prof. Melodia is the Director of Research for the Platforms for Advanced Wireless Research (PAWR) Project Office, a $100M public-private partnership to establish four city-scale platforms for wireless research to advance the US wireless ecosystem in years to come. Prof. Melodia's research on modeling, optimization, and experimental evaluation of Internet-of-Things and wireless networked systems has been funded by the National Science Foundation, the Air Force Research Laboratory the Office of Naval Research, DARPA, and the Army Research Laboratory. Prof. Melodia is a Fellow of the IEEE and a Distinguished Member of the ACM.

Carla Fabiana Chiasserini is currently a Full Professor with the Department of Electronics and Telecommunications Engineering at Politecnico di Torino, Italy, a WASP Guest Professor at Chalmers University of Technology, Sweden, and a Research Associate with the Italian National Research Council (CNR) and CNIT. She was a visiting researcher with UCSD, a visiting professor with Monash University, Technische Berlin University, and HPI at Potsdam University. Her research interests include 5G-and-beyond networks, NFV, mobile edge computing, connected vehicles, and distributed machine learning at the network edge.