An Experimental O-RAN Environment for Evaluating AI-Driven RAN Control

Lucas Rodrigues, Elen Gomes, Diego Bezerra, Djamel F. H. Sadok and Glauco Gonçalves

Abstract— **Open RAN (O-RAN) aims to reduce operational costs and improve interoperability by promoting disaggregated and virtualized Radio Access Network architecture. In this context, development of new xApps is limited by the capabilities of underlying experimental platforms. This paper presents an experimental environment integrating srsRAN and OSC-RIC, two widely used open-source O-RAN software components, and extends srsRAN to expose additional performance metrics via the E2 interface. Such extension enables the periodic reporting of detailed KPIs from the PHY and MAC layers, enhancing RAN observability and offering a cornerstone for evaluating new xApps for closed-loop control and adaptive optimization in 5G networks.**

Keywords— **O-RAN, 5G Beyond, Near-rt-RIC, Network Management.**

I. Introduction

Significant advances and the growing demand for enhanced connectivity in wireless communications have resulted in new cost and management challenges for network operators, increasing demands for flexibility, scalability, and intelligent automation [1]. In this context, the Open Radio Access Network (O-RAN) has emerged as an alternative to overcome current RAN limitations.

Based on disaggregation and virtualization, the O-RAN paradigm enables multi-vendor interoperability, reducing maintenance and operational costs [2]. It also opens the door to data-driven and closed-loop control through the deployment of xApps, microservices that can operate in near-real-time [3], hosted by the RAN Intelligent Controller (RIC).

However, practical development and testing of xApps remain limited by the lack of accessible and flexible experimental environments [1]. Bridging this gap is essential to accelerate research, validate optimizations, and demonstrate the feasibility of intelligent RAN control in real-world scenarios.

This paper addresses this challenge by presenting a distributed and fully virtualized experimental environment that integrates the srsRAN Project (for gNB and UE simulation), the O-RAN Software Community RIC (OSC-RIC), and the Open5GS 5G Core. Furthermore, we propose and implement custom extensions to the srsRAN Project stack to support the collection of additional RAN metrics via the E2 interface, which are essential for enabling RAN monitoring and xApp control loops.

By showcasing the deployment of an xApp focused on energy consumption analysis, this work presents a reusable experimental environment and demonstrates the potential of O-RAN for intelligent resource management.

This work is organized as follows: Section II describes the technologies used in this work. Section III describes the limitations of srsRAN and the proposed extensions in this work. Section IV describes the experimental environment setup. Section V presents the results. Finally, Section VI showcases the main findings and future research directions.

II. srsRAN & OSC-RIC

The O-RAN paradigm introduced open interfaces, flexibility, and intelligence into traditionally closed and monolithic RAN architectures [1]. With this in mind, open-source projects such as srsRAN and OSC-RIC have emerged as essential tools for researchers and developers, enabling the prototyping, testing, and deployment of disaggregated RAN components in a virtualized environment.

srsRAN is a fully open-source software suite compliant with 3GPP and O-RAN Alliance specifications[1]. It provides a collection of 4G and 5G RAN components, including support for User Equipment (UE), core network, eNB, and gNB. Therefore, it allows the deployment of an end-to-end, robust network environment. In alignment with the specifications of the O-RAN Alliance, srsRAN also supports the commonly used 5G functional 7.2x split, which decomposes the gNB into O-RU (Radio Unit), O-DU (Distributed Unit), and O-CU (Centralized Unit). By dividing the physical layer (PHY) into a high-PHY and a low-PHY, this split is designed to achieve a balance between flexibility, cost, and performance. To manage software complexity, srsRAN offers containerized solutions, including an Open5GS bundle[2] that enables the seamless integration of core network functions with radio components.

OSC-RIC plays an important role in enabling intelligent control and management of network resources. It introduces a standardized framework and interfaces for deploying near-real-time RIC (Near-RT-RIC) unit, in order to achieve a data-driven closed-loop control over radio resources. It is an essential element for monitoring and managing RAN elements. The Near-RT RIC can host multiple microservices known as xApps, performing services based on data extracted from E2 nodes. An xApp can operate control-loops functions within 10 *ms* and 1 *s* intervals allowing faster control over network functions [1]. Enforcing RRC policies over the RAN components, xApps work by collecting exposed data from E2 nodes.

Lucas Rodrigues, Elen Gomes and Glauco Gonçalves are part of LASSE, UFPA, Belém-PA. Diego Bezerra and Djamel Sadok are part of GPRT, UFPE, Recife-PE. E-mails {lucas.lima.rodrigues, elen.cristina}@itec.ufpa.br, glaucogoncalves@ufpa.br, {diego.bezerra, jamel} @gprt.ufpe.br.

[1]Available at: `https://docs.srsran.com/en/latest/`. Accesed on May 05, 2025.

[2]Available at: `https://github.com/srsran/srsRAN_Project/tree/main/docker/open5gs`. Accessed on May 5, 2025.

The OSC-RIC I-release minimal version simplifies the development and deployment of xApps by providing a Docker container with the xApp Python Framework, which houses the commonly required features for xApps. These include built-in and essential support for communication with the RIC Message Router (RMR) and the Shared Data Layer (SDL).

III. srsRAN E2 Interface: Limitations and Extensions

srsRAN is a widely adopted and actively maintained open-source tool for 5G research and prototyping. Despite its popularity, the current implementation remains limited in the diversity and granularity of metrics exposed via the E2 interface. The E2SM-KPM service model supports the periodic reporting of a limited set of standardized performance indicators [4]. Among the supported metrics are downlink and uplink throughput per user (`DRB.UEThpDl`, `DRB.UEThpUl`), RLC packet drop rate in the downlink (`DRB.RlcPacketDropRateDl`), PDCP packet success rate (`DRB.PacketSuccessRateUlgNBUu`), and transmitted SDU volume (`DRB.RlcSduTransmittedVolumeDL`, `DRB.RlcSduTransmittedVolumeUL`). Furthermore, metrics such as channel quality indicators are currently included as placeholders and, according to the documentation, are expected to be deprecated in future releases[3].

Besides, E2 interface does not expose indicators such as per-layer latency, HARQ retransmission statistics, modulation and coding scheme (MCS) usage, handover attempts, signal-to-interference-plus-noise ratio (SINR), or UE velocity. Furthermore, the measurement reporting component (`e2sm_kpm_du_meas_provider.cc`) does not include provisions for incorporating more detailed internal metrics.

For this work, we provided a set of custom measurement retrieval functions to expose new metrics for collection via the E2 interface[4]. These additions allowed periodic reporting of a set of Key Performance Indicators (KPIs) beyond the default metrics. The implemented functions include: `get_pci` (Physical Cell ID, unitless), `get_rnti` (UE identifier, unitless), `get_mcs_dl` and `get_mcs_ul` (modulation and coding scheme for downlink and uplink, unitless), `get_brate_dl` and `get_brate_ul` (bitrate per direction in kbps), `get_nof_ok_dl`, `get_nof_nok_dl`, `get_nof_ok_ul`, and `get_nof_nok_ul` (number of successful and failed transport blocks, unitless), `get_bsr` (Buffer Status Report in bytes), `get_dl_bs` (downlink buffer size in bytes), `get_ta` (Timing Advance in nanoseconds), `get_phr` (Power Headroom Report in dB), `get_pusch_snr` (uplink SNR on PUSCH in dB), and `get_ri` (Rank Indicator, unitless).

New metrics can be added to E2SM-KPM reports by extending the measurement provider and registering new MeasurementRecord entries with custom MeasTypeName identifiers. To incorporate these metrics, each getter function was properly linked to the internal state of the srsRAN gNB, collecting runtime information from the PHY and MAC layers. These were registered as custom measurement types and included in the E2SM-KPM message assembly process. This extension enhances the observability of radio link conditions and user-specific performance, enabling the development of xApps capable of proactive RAN control, fine-grained failure detection, and adaptive, context-aware optimization.

IV. Implementation

The experimental environment **was deployed across two bare-metal servers** both running Ubuntu 22.04. Server-1, equipped with an Intel Core i7-12700 CPU @ 2.10GHz and 16GB of RAM, hosted the srsRAN gNB, acting as the base station in the network setup. Server-2, powered by an Intel Core i7-14700 CPU @ 2.10GHz and 16GB of RAM, runs the emulated srsRAN UE as well as containerized instances of Open5GS and OSC-RIC i-release minimal version.

The official srsRAN Project repository on Github[5] and [5] provide a comprehensive and detailed tutorial, including building tools and software dependencies. It is essential to follow these installation instructions carefully to ensure proper configuration and compatibility with both hardware and software.

To ensure a direct connection between the two servers, we assign static IP addresses to each server's network interfaces, which are configured in the same subnet. Furthermore, firewall rules can be defined to ensure proper and secure connection between the various services.

After building the srsRAN projects, the next step is to set up communication between the components. This needs modifying the gNB and UE YAML configuration files. In the gNB configuration, the IP address responsible for binding traffic to the AMF must be defined by setting the `bind_addr` parameter within the `cu_cp` section to Server-2's IP address. Additionally, to establish communication between the gNB and the UE, the `device_args` field must be correctly configured in both YAML files.

To integrate with the Near-RT RIC, specific modifications must be made in the `e2` section of the gNB YAML file. The `addr` field should be set to Server-1's IP address, while the `bind_addr` should correspond to Server-2's IP. Additionally, in the OSC-RIC deployment, in the docker-compose file, the `ports` section under the `e2term` service must be uncommented to ensure that the necessary E2 interface ports are properly exposed.

Finally, a static route must be added to establish communication between the gNB and the 5G Core. On Server-2, this can be achieved by setting a route to the 5G core network segment via the IP address of Server-1 [5]. Upon successful configuration, the resulting architecture of the environment should resemble that shown in Figure 1.

V. Results

Taking advantage of the new RAN performance metrics exposed in this work, xApps can extract additional KPIs through

[3]Available at: https://docs.srsran.com/projects/project/en/latest/tutorials/source/near-rt-ric/source/index.html. Accessed on May 5, 2025.

[4]Available at: https://github.com/gt-oiran/srsran Accessed on May 5, 2025.

[5]Available at: https://github.com/srsran/srsRAN_Project. Accessed on May 7, 2025.

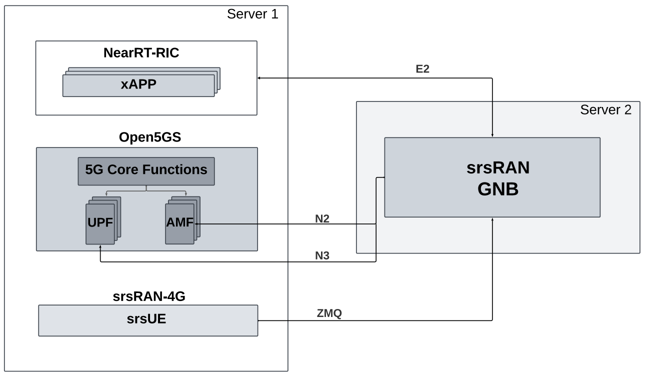

Fig. 1. Environment architecture.

the E2 interface, enabling more precise control decisions, and improved adaptability to dynamic network conditions.

The deployed xApp is available in the project's Github repository[6]. This xApp leverages different Machine Learning (ML) models to predict power consumption at the gNB. Figure 2 illustrates the xApp workflow, where KPIs such as `McsUl`, SNR, `RRU.PrbAvailUl` and `RRU.PrbTotUl` are recorded into a CSV file and stored on the server's local drive. A sliding buffer window is applied to smooth the KPI data before it is passed to the ML model for prediction.

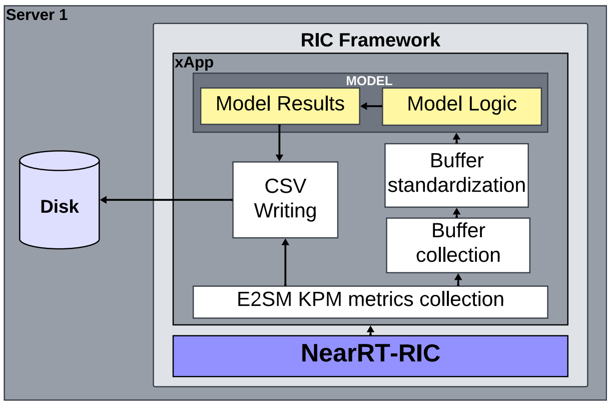

Fig. 2. Power Consumption xApp workflow.

With the environment at hand, the user may next capture metrics via E2 nodes using the xApp, following the custom E2SM-KPM. We utilize the example xApp presented on Github to store relevant KPIs in a CSV (Comma Separated Values) format file.

Figure 3 illustrates the collection of selected KPIs before the treatment with sliding buffer window was applied. During the experiment, traffic was generated using *iperf3* to simulate an uplink data stream from the UE to the 5G Core. A custom script initiated the scenario with a 60-second warm-up period, after which the test alternated between 120 seconds of active traffic and 60 seconds of idle time, over a total duration of 540 seconds.

[6]Available at: `https://github.com/gt-oiran/power-consumption-module`. Accessed on May 8, 2025.

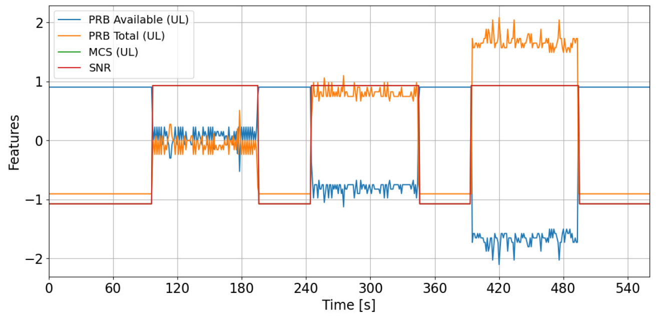

Fig. 3. KPIs collected before sliding window treatment.

To emulate realistic and fluctuating network conditions, the traffic bandwidth was dynamically adjusted after each idle interval. This allowed the observation of KPI variations under different load patterns.

VI. Conclusion

This paper presented a robust test environment for developing and deploying gNB-focused xApps. Leveraging srsRAN implementations and open-source components, the proposed scenario can be easily replicated. This setup not only supports current experimentation with 5G RAN architectures but also serves as a valuable foundation for future studies, including the development and evaluation of machine learning-based xApps and other custom logic for intelligent RAN control.

Aknowledgements

The authors would like to thank the National Council for Scientific and Technological Development (CNPq) (process: 408326/2023-9), the Rede Nacional de Ensino e Pesquisa (RNP), the Centro de Pesquisa e Desenvolvimento em Telecomunicações (CPQD), and the OpenRAN@Brasil program (process MCTI No: A01245.014203/2021-14).

References

[1] 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. DOI: `10.1109/COMST.2023.3239220`.

[2] J. Groen, S. D'Oro, U. Demir, *et al.*, "Implementing and Evaluating Security in O-RAN: Interfaces, Intelligence, and Platforms," *IEEE Network*, pp. 1–1, 2024. DOI: `10.1109/MNET.2024.3434419`.

[3] A. Lacava, M. Bordin, M. Polese, *et al.*, "ns-O-RAN: Simulating O-RAN 5G Systems in ns-3," in *Proceedings of the 2023 Workshop on Ns-3*, ser. WNS3 '23, Arlington, VA, USA: Association for Computing Machinery, 2023, pp. 35–44, ISBN: 9798400707476. DOI: `10.1145/3592149.3592161`. [Online]. Available: `https://doi.org/10.1145/3592149.3592161`.

[4] O-RAN ALLIANCE, "ORAN.WG3.E2SM-RC-R003-v04.00," O-RAN ALLIANCE e.V., Technical Specification ORAN.WG3.E2SM-RC-R003-v04.00, 2023, Clause 9.2.1.15. [Online]. Available: `https://www.o-ran.org/specifications`.

[5] P. Liu, K. Lee, F. Cintron, *et al.*, *Blueprint for Deploying 5G O-RAN Testbeds: A Guide to Using Diverse O-RAN Software Stacks*, en, Oct. 2024. DOI: `https://doi.org/10.6028/NIST.TN.2311`. [Online]. Available: `https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=958753`.