xDevSM: Streamlining xApp Development With a Flexible Framework for O-RAN E2 Service Models

Angelo Feraudo[†], Stefano Maxenti[*], Andrea Lacava[*‡],
Paolo Bellavista[†], Michele Polese[*], Tommaso Melodia[*]
[†]Department of Computer Science and Engineering, University of Bologna, Italy
[*]Institute for the Wireless Internet of Things, Northeastern University, Boston, MA, U.S.A.
[‡]Sapienza University of Rome, Italy

Abstract

RAN Intelligent Controllers (RICs) are programmable platforms that enable data-driven closed-loop control in the O-RAN architecture. They collect telemetry and data from the RAN, process it in custom applications, and enforce control or new configurations on the RAN. Such custom applications in the Near-Real-Time (RT) RIC are called xApps, and enable a variety of use cases related to radio resource management. Despite numerous open-source and commercial projects focused on the Near-RT RIC, developing and testing xApps that are interoperable across multiple RAN implementations is a time-consuming and technically challenging process. This is primarily caused by the complexity of the protocol of the E2 interface, which enables communication between the RIC and the RAN while providing a high degree of flexibility, with multiple *Service Models (SMs)* providing plug-and-play functionalities such as data reporting and RAN control. In this paper, we propose xDevSM, an open-source flexible framework for O-RAN service models, aimed at simplifying xApp development for the O-RAN Software Community (OSC) Near-RT RIC. xDevSM reduces the complexity of the xApp development process, allowing developers to focus on the control logic of their xApps and moving the logic of the E2 service models behind simple Application Programming Interfaces (APIs). We demonstrate the effectiveness of this framework by deploying and testing xApps across various RAN software platforms, including OpenAirInterface and srsRAN. This framework significantly facilitates the development and validation of solutions and algorithms on O-RAN networks, including the testing of data-driven solutions across multiple RAN implementations.

CCS Concepts

• **Networks** → **Network experimentation**; **Programmable networks**; **Network measurement**.

Keywords

O-RAN, xApp, Near-RT RIC, RIC, Service Model, Open RAN, E2AP, E2SM

ACM Reference Format:
A. Feraudo, S. Maxenti, A. Lacava, P. Bellavista, T. Melodia, M. Polese. 2024. xDevSM: Streamlining xApp Development With a Flexible Framework for O-RAN E2 Service Models. In *The 30th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '24), November 18–22, 2024, Washington D.C., DC, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3636534.3697325

1 Introduction

The Open Radio Access Network (RAN) vision, being defined into a network architecture by the O-RAN ALLIANCE, brings a fundamental paradigm shift in how networks are deployed and optimized, through closed-loop control of the RAN exercised by the so-called RAN Intelligent Controllers (RICs). The current O-RAN specifications include two instances of the RIC, operating at different time scales and on different parameters. The Non-Real-Time (RT) RIC, hosted in the network Service Management and Orchestration (SMO), supports control loops with a time scale higher than 1 s, and enforces policies or high-level configurations on the system. The Near-RT RIC performs radio resource management by directly tuning the configuration parameters of the RAN at a time scale between 10 ms and 1 s. These components host custom applications, called rApps and xApps, respectively, which implement the closed-loop control logic [24].

In this paper, we focus on the Near-RT RIC. This component directly exercises radio resource management and thus holds the potential to significantly influence the performance of the RAN. Prior work has shown how dynamically configuring the RAN stack through closed-loop control exercised through xApps can lead to significant performance improvements in spectrum utilization, throughput, and user satisfaction [31, 26, 34, 5]. Data-driven agents controlling and

ACM ISBN 979-8-4007-0489-5/24/11
https://doi.org/10.1145/3636534.3697325

optimizing the radio resource allocation yield performance gains in metrics that are orthogonal (e.g., throughput and latency) and hence often challenging to jointly optimize [23]. The role of the RICs has been explored in the context of network slicing [31], load balancing and handover [7, 4], traffic shaping [5], and spectrum sharing [29], among others [24].

Such flexibility and capabilities have led to several open-source projects focused on the Near-RT RIC, as well as implementations of the RAN termination for the interface connecting the Near-RT RIC to the disaggregated Next Generation Node Bases (gNBs), i.e., the E2 interface [27]. Among these, the O-RAN Software Community (OSC) provides an open-source implementation of a Near-RT RIC [11], an xApp framework [12], and an E2 termination [15]. Popular open-source 5G RAN implementations also provide E2 terminations on their base station code base, e.g., for OpenAirInterface (OAI) [9] and srsRAN [30].

While the availability of multiple projects on E2 and RIC has fostered experimental research on data-driven Open RAN solutions [2, 6, 32], it remains challenging to develop an xApp that works across components developed by different projects or vendors. The E2 interface itself is split into two protocols: an application protocol, or E2AP, which manages the connectivity between the RAN and the RIC, and the E2 Service Models (SMs), which builds functionalities on top of E2AP, e.g., to expose Key Performance Measurements (KPMs) or control the RAN parameters. While the E2AP implementations across the RAN projects discussed above interoperate with the OSC Near-RT RIC, the SMs implementations often refer to different versions of the O-RAN specifications, with non-overlapping sets of features and capabilities. This leaves researchers and developers in the Open RAN space with limited options for developing and testing xApps. An additional challenge is represented by the complexity of the SM implementation, with the serialization and deserialization of the data needing to abide by specifications for ASN.1 payloads.

To address this challenge, in this paper, we design, develop, and test xDevSM, a flexible and open-source framework to develop xApps for the OSC Near-RT RIC capable of interoperating with different E2SM implementations.[1] xDevSM provides xApp developers with a Software Development Kit (SDK) with Application Programming Interfaces (APIs) that implement the steps defined in various E2SM protocols, part of the interaction between the xApp, the RIC, and, eventually, the E2 termination on the RAN. This removes the challenges associated with developing code to configure the E2SM messages and leaves the developer with the task of defining the application logic. xDevSM leverages the OSC RIC components and extends the xApp logic with a wrapper that can interface with different shared libraries implementing E2SM KPM. We tested the framework with multiple open-source RAN implementations, including srsRAN, OAI, and NVIDIA Aerial RAN CoLab (ARC), validating the functionality as well as the flexibility of the proposed approach.

The remainder of the paper is organized as follows. Section 2 provides details and background on the O-RAN architecture, xApp frameworks and development, and summarizes the key challenges. Section 3 introduces xDevSM, with a discussion on its architecture and implementation. Section 4 reviews the RAN implementations which have been successfully interoperated with xDevSM. Finally, we discuss lessons learned, conclude the paper, and suggest future work in Section 5.

2 xApp Development Background

To ensure vendor interoperability across RAN and RIC implementations, the O-RAN ALLIANCE Working Group (WG) 3 has developed specifications for multiple protocols that define E2 interface [17]. The first specification is the E2 Application Protocol (E2AP) [22], which is used for general management operations, such as the connection and disconnection of a Distributed Unit (DU) or Central Unit (CU) with the RIC, as well as providing a list of RAN functions supported by the RAN nodes. The E2 Service Model (E2SM) specifications [20] define general elements for the SMs, which express the semantics of the interactions between the xApps and the RAN nodes. Each service model is featured in a document extending the general specification of the E2SM with parameters related to the use case of interest. Examples of these extensions are the E2SM KPM [18], for data collection, and the E2SM RAN Control (RC) [21], which defines the protocol to perform Radio Resource Management (RRM) actions such as traffic steering through handover management.

The O-RAN ALLIANCE, through the OSC, provides open-source implementations of the components of the O-RAN architecture, contributed by telecom and xApp vendors, researchers, and academia. The OSC follows a release cycle that translates the O-RAN technical specifications into different versions of the OSC codebase, together with bug fixes and overall improvements of the infrastructure [1].

In this context, the OSC has released a reference implementation for the Near-RT RIC, leveraging a micro-service architecture. This platform comprises various components, including internal message routing based on the RIC Message Router (RMR), E2 termination, a subscription manager, a network information base database, and a shared data layer API among the others. Each of these components operates within a Kubernetes cluster, ensuring scalability and efficient

[1]https://github.com/wineslab/xDevSM
https://openrangym.com/tutorials/xdevsm-tutorial

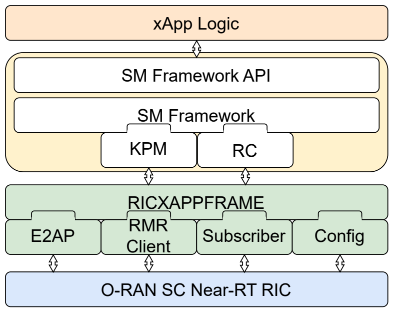

Figure 1: xDevSM and RIC components. The xDevSM components are in yellow, while the OSC RIC platform is in green and blue. The xApp developer only takes care of the xApp logic, in orange.

resource management. Besides the Near-RT RIC implementation, the OSC project also offers an xApp framework [12] that uses the RIC RMR to manage the message exchange with the remaining components of the RIC platform.

Nevertheless, the rapid cycle with which the O-RAN ALLIANCE publishes new specifications often conflicts with the time required to develop new OSC releases, complicating the maintenance and update of the E2 specifications. This has led to a flurry of scattered initiatives around E2 interface implementations. An example of this is represented by FlexApp paper [3], whose authors implement an E2* interface that is built on the E2 procedures but provides an additional layer of abstraction. Other approaches have introduced simplified versions of the SMs based on, for example, Protobuf buffers rather than ASN.1 data structures [8, 33]. While this approach simplifies the development and testing of new SMs, it is not fully O-RAN compliant and requires both xApp and RAN to support Protobuf. Additionally, encoding in Protobuf incurs higher overhead than using directly the binary field provided by the ASN.1 encoding [25].

The RIC platform itself has been subject to several development efforts, primarily for scaled-down versions of the Near-RT RIC, such as the srsRAN implementation [30] and the OAI FlexRIC. The srsRAN implementation, which is integrated with the srsRAN RAN components, provides the key functionalities of the original RIC at the I release version, but requires the deployment of the same RIC E2 libraries also on the RAN side, thus breaking the foundational concept of vendor interoperability. Compared to this approach, we provide a comprehensive framework that leverages a modular approach on the xApp side, facilitating scalability to different RAN implementations. The OAI FlexRIC is another SDK for software-defined-RAN controllers based on an E2-compatible protocol. This SDK includes an iApp component, which implements E2SMs and exposes information to the xApp, and a server library, which multiplexes connections between xApps and dispatches E2AP messages to E2 Nodes. The FlexRIC approach relies on a monolithic architecture, where all Near-RT RIC operations are included in a single component, decreasing the flexibility of the RIC platform. Moreover, OAI FlexRIC E2 Agent and OSC interoperability has not been tested properly, leading to incompatibility between OAI RAN and the OSC Near-RT RIC.

This paper relies on the O-RAN components provided by the OSC, as these serve as a reference point that closely aligns with the architecture and specifications developed by the O-RAN ALLIANCE.

3 xDevSM: A Flexible SM Framework for xApp Development

In this section, we first review the architecture of xDevSM (Section 3.1), and then discuss its design and implementation in the context of the OSC RIC framework (Section 3.2).

3.1 xDevSM Architecture

As mentioned in previous sections, the OSC provides a comprehensive suite of libraries and functions aligned with their Near-RT platform version to support xApp development. This suite includes a message-level API for the RMR, featuring callback registration, E2AP encoding and decoding, subscription APIs, and the Shared Data Layer (SDL) interface. In our scenario, we utilized the Python-based framework [16], which introduces two primary types of xApps: Reactive xApps (`RMRXApp`) and general xApps (`XApp`). Reactive xApps respond exclusively to incoming RMR messages by invoking the appropriate callback based on the message type, with the main xApp capable of registering multiple callbacks. On the other hand, general xApps require the main xApp to actively fetch messages from the RMR buffer.

The frameworks provided by OSC focus on the communication aspect within RIC components. However, once the xApp correctly subscribes or connects to the E2 node, there is a lack of easy-to-use functions to interact with the E2SM (and the RAN functions) exposed by the E2 nodes. Hence, the burden of handling the serialization and deserialization procedures related to the SM exposed by E2 nodes falls on the xApp developers. This requires them to know the specific E2SM versions supported by the E2 node to implement a fully working xApp.

To address this challenge, we designed xDevSM, a SM framework that enables encoding and decoding of E2SM data in a plug-and-play fashion and through simple and well-defined APIs. As illustrated in Figure 1, the framework is built on top of the Python xApp framework provided by the OSC [16, 10] and comprises two components: the SM Framework and the SM Framework API.

The SM Framework offers Python objects and functions that enable accurate decoding and encoding of messages related to supported E2SMs. These functions are implemented

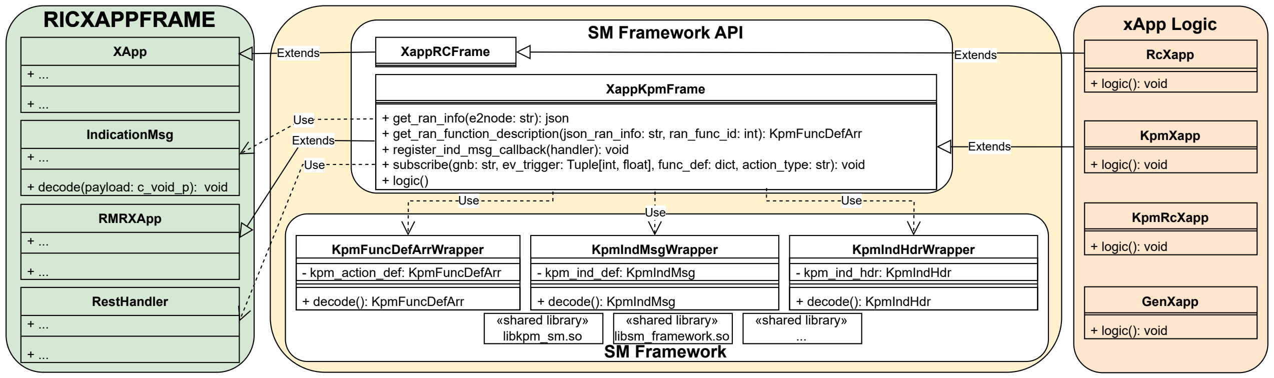

Figure 2: Class diagram for xDevSM.

in shared libraries written in C, which are dynamically loaded by the framework as needed. In this initial version, the shared libraries are generated using E2SM functions defined in the FlexRIC project [28], along with custom functions that simplify some xApp operations requiring data decoding. The Python objects within the framework handle the memory deallocation of the corresponding C structures by leveraging Python's garbage collector. To include new SMs, developers can extend the shared libraries with the necessary encoding/decoding functions, and then add the corresponding Python objects.

The API component of the SM framework provides the necessary interfaces to interact with the Near-RT RIC and E2 Nodes and implement the xApp logic. It includes an abstract method that the xApp developer must define. This method is called after the xApp setup and expresses the application-specific logic. For example, it can request the decoded RAN function description or store Key Performance Indicators (KPIs) in a time-series database. Additionally, the xApp containing the logic can receive decoded KPM messages and can directly send RC messages in an encoded form. These messages are managed using the SM framework functionalities described above and sent to the RIC components using the OSC framework.

Compared to other solutions [30], xDevSM relies on shared libraries to manage ASN.1 serialization and deserialization procedures, leveraging the C language, which is the most widely adopted in this context [28, 14, 13]. This approach enhances the robustness of xDevSM, as it simplifies the process of finding and building shared libraries to support new service models. Additionally, the extensive use of C in this environment ensures better performance and compatibility, making xDevSM a more reliable and adaptable framework for xApp development.

3.2 Design and Implementation of xDevSM

The new abstraction layer introduced in our framework facilitates data translation and optimizes memory management by handling the allocation and deallocation of external structures dynamically. As shown in Figure 2, the API component provides a class named `XappKpmFrame`, which extends the `RMRXApp` class provided by the OSC xApp framework[10]. It does this by introducing methods to retrieve E2 node information, such as the node connection status to the Near-RT RIC and its available RAN functions, as well as methods related to the subscription procedure. Methods for RAN information retrieval use a Python object named `KpmFuncDefArrWrapper`. This wrapper references a custom data structure defined within the `libsm_framework.so` shared library, which is essential for decoding and building the *RAN Function Definitions*. Decoding these definitions is necessary as the Near-RT RIC subscription manager stores RAN function descriptors in hexadecimal and E2SM-encoded formats. Furthermore, the wrapper handles memory deallocation related to the custom structure as soon as the Python object is no longer needed.

The `subscribe` method of the `XappKpmFrame` class allows developers to specify the E2 node, the RAN functionalities to be monitored, and the reporting period for the xApp subscription. Before sending this information to the subscription manager, the `subscribe` method encodes the RAN functionalities and the reporting period using functions defined in the `libsm_framework.so` shared library. Upon receiving this encoded information, the subscription manager performs an additional level of encoding using the E2AP libraries and then forwards the subscription to the E2 nodes. It should be noted that the encoding functions provided by `libsm_framework.so` utilize procedures defined in `libkpm_sm.so`, generated using FlexRIC [28], as these are closely related to the E2SM.

The xApp developers can use this framework by either extending the `XappKpmFrame` class or by encapsulating an instance of this class. Extending the `XappKpmFrame` offers more flexibility, allowing developers to customize behaviors of internal methods such as post-initialization actions, logger level adjustment, and modified decoding functions. On the other hand, encapsulating an instance of the `XappKpmFrame` class allows developers to utilize the class procedures without altering its internal behavior, simplifying integration and minimizing potential errors.

In this paper, we focus on the first option as illustrated in Figure 2 (orange block). The `KpmXapp` class inherits from the `XappKpmFrame` and registers a callback method to manage *Indication Messages* and defines the `logic` method. This ensures that all operations related to the dispatching of indication messages and decoding of the information are handled within the framework. Specifically, the framework automatically decodes the E2AP part using the `RICXAPPFRAME IndicationMsg` class (green block Figure 2) and the E2SM data using the `KpmIndMsgWrapper` and `KpmIndHdrWrapper` classes (yellow block Figure 2). The E2SM-related classes expose a `decode` API, which invokes a function defined in the `libkpm_sm.so`. This function decodes the E2SM-related information and returns a Python Object corresponding to the KPM related data. This modular approach allows for easy updates to the E2SM by simply changing the shared library version, provided that function signatures remain consistent across different E2SM versions. Once the received message has been decoded, the registered function containing the behavior defined by the xApp developer is executed.

The `logic` method is an abstract method provided by the `XappKpmFrame` that allows the developers to define the xApp behavior. Within this method, developers can specify which E2 node to select based on a particular logic, identify RAN functionalities of interest, set reporting period, and more. We provide an example[2] of how an xApp can be defined using this framework, to demonstrate how it simplifies the process for developers when designing and implementing an xApp using existing frameworks, allowing them to focus exclusively on the application logic.

4 Testing xDevSM with Open-Source RANs

In this section, we showcase the flexibility and robustness of xDevSM by deploying an xApp based on our framework in combination with various real-world and simulated environments. We first consider OAI, in Section 4.1, and then we test srsRAN in Section 4.2.

4.1 OpenAirInterface

This section explores the effectiveness of our solution using software provided by OAI for the RAN, core, and User Equipment (UE). We analyze a simulated scenario, using only OAI-based software, and a real-world scenario where the OAI RAN is paired with the Open5GS core and Commercial Off-the-Shelf (COTS) UEs.

[2]https://github.com/wineslab/xDevSM-xapps-examples

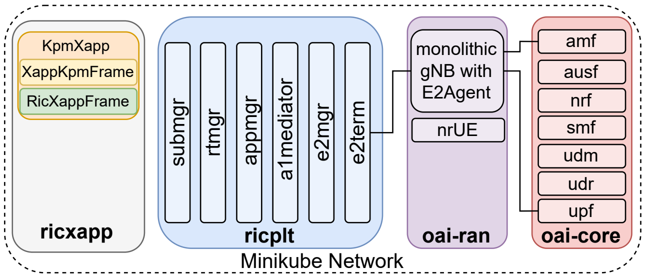

Figure 3: OpenAirInterface deployment in Minikube

```
"inventoryName": "gnb_001_001_00000e05",
"globalNbId": {
"plmnId": "00F110",
"nbId": "00000000000000000000111000000101"
},
"connectionStatus": "CONNECTED"
```

Listing 1: Registration of the gNB on the Near-RT RIC

4.1.1 *OpenAirInterface in a Box.*

As illustrated in Figure 3, this deployment involves setting up a 5G network using Minikube, leveraging OAI for both RAN and core. To support the E2 interface and facilitate connectivity with the Near-RT RIC, the OAI RAN has been built with its default E2 agent, provided by FlexRIC project [28]. A pod in the `oai-ran` namespace implements an NR UE and is connected to the base station using the OAI RFSim. This deployment creates an O-RAN-compliant environment in a single, isolated setup, enabling efficient testing and prototyping of xApps within a controlled environment. We utilize the `KpmXapp` described in the previous section to test its compatibility with OAI. In this configuration, the SM framework utilizes the shared library (`libkpm_sm.so`) produced during the building phase of the E2Agent from the FlexRIC SDK. We conducted several tests to confirm that OAI and the latest OSC Near-RT RIC Releases I and J can communicate effectively, while xApps perform their designated tasks. Listing 1 shows a successful connection of a base station to the Near-RT RIC, whereas Listing 2 shows an example of the RAN function IDs and capabilities retrieved from the Base Station by the xApp and decoded by xDevSM. These capabilities include the amount of PDCP traffic exchanged in the measurement interval (`DRB.PdcpSduVolumeDL`, `DRB.PdcpSduVolumeUL`), the average throughput (`DRB.UEThpDl`, `DRB.UEThpUl`), and the number of Physical Resource Blocks (PRBs) allocated (`RRU.PrbTotDl`, `RRU.PrbTotUl`).

This deployment served as a base for developing, testing, and prototyping the framework proposed in this paper. All shared libraries created in this environment are reused in other deployments.

4.1.2 *Over-the-Air OpenAirInterface Deployment with NVIDIA ARC.*

Building on the simulated environment, we have extended our deployment to a real-world scenario that involves

```
{[...], "id": "ricxappframe.xapp_frame", "mdc": {},
    "msg": "Available functions: {0: [], 1: [], 2:
    [], 3: ['DRB.PdcpSduVolumeDL',
    'DRB.PdcpSduVolumeUL', 'DRB.RlcSduDelayDl', 'DR
B.UEThpDl', 'DRB.UEThpUl', 'RRU.PrbTotDl',
    'RRU.PrbTotUl'], 4: []}"}
{[...], "id": "ricxappframe.xapp_frame", "mdc": {},
    "msg": "Selected functions: {3:
    ['DRB.PdcpSduVolumeDL', 'DRB.PdcpSduVolumeUL',
    'DRB.UEThpDl', 'DRB.UEThpUl', 'RRU.PrbTotDl',
'RRU.PrbTotUl']}"}
{[...], "id": "ricxappframe.xapp_frame", "mdc": {},
    "msg": "Preparing subscription for gnb:
    gnb_001_001_00000e05"}
{[...], "id": "ricxappframe.xapp_frame", "mdc": {},
    "msg": "event trigger encoded: (8, 3, 231)"}
```

Listing 2: Registration of the gNB on the xApp after decoding the ASN

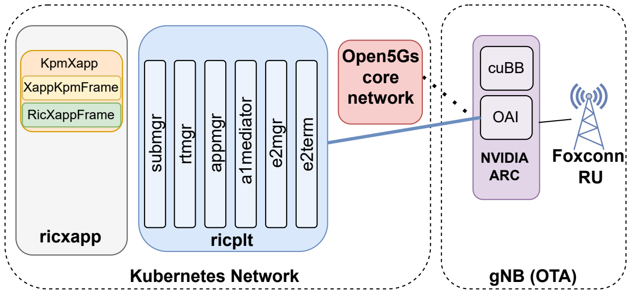

Figure 4: OpenAirInterface over the air

an over-the-air setup. In particular, we leverage the X5G testbed [33], with OAI as the CU and the DU-high, while the DU-low uses the NVIDIA ARC inline acceleration on Graphics Processing Unit (GPU). This is connected to a Foxconn RU operating in the 3.7-3.8 GHz band, with a bandwidth of 100 MHz, and connected to Samsung S23 and OnePlus AC2003 Nord COTS smartphones. In the same way as the previous setup, we test the default E2 agent for the OAI stack. Differently from the previous setup, though, we use Open5GS as a core network to validate functionalities with different Access and Mobility Management Function (AMF) and User Plane Function (UPF) implementations, which may encode UE information in slightly different ways. The final setup can be seen in Figure 4.

We tested up to five connected UEs, and the xApp correctly reports metrics for all of them. We also tested UEs disconnections, which resulted in the correct removal from the list of available UEs sent to the Near-RT RIC from the gNB. The RAN stack exposes the functions already shown in Listing 2.

In Figure 5 we show that the values reported by the xApp are indeed tracking the traffic generated by the UE using iperf3 over TCP, both in downlink and uplink. There is a slight offset of around 2%, related to both a misalignment in reporting of the KPMs by the xApp and iPerf3 logs, and the additional overhead introduced by headers in the TCP/IP stack and at PDCP, which is captured by the xApp and not by iPerf3. The latter indeed reports the application layer throughput, while the xApp measures throughput within the RAN.

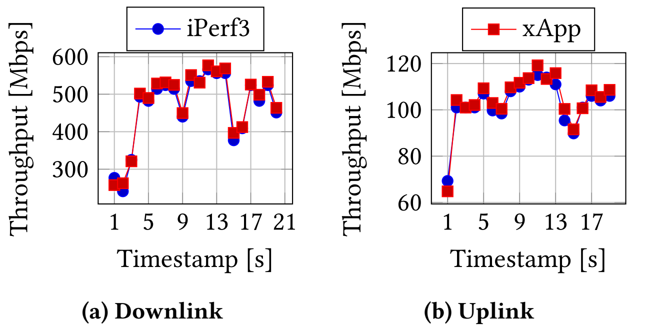

Figure 5: Comparison of iPerf3 (blue dots) and xApp (red squares) reported values with OAI

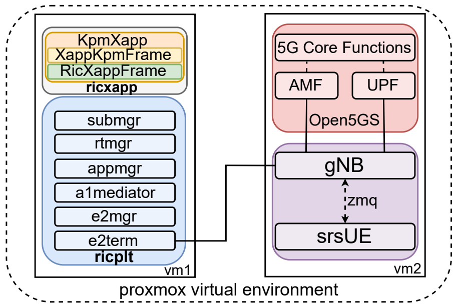

Figure 6: srsRAN simulated deployment

4.2 srsRAN

This section analyzes the integration of xDevSM in a deployment based on srsRAN, to test the effectiveness of our framework with a different RAN software. Like the previous scenario, we first built a simulated environment using two virtual machines hosting the Near-RT RIC and the 5G components. The RIC virtual machine was also reused to test the xDevSM framework in a real-world scenario with over-the-air communications.

4.2.1 srsRAN in a Box. This deployment involves the simulated scenario provided in the srsRAN tutorial[3], where the srsUE and the gNB communicate through a ZeroMQ-based RF driver and the deployed core network is Open5GS.

As illustrated in Figure 6, our setup differs from the one provided in the srsRAN tutorial in one way. We use two virtual machines within the same network: one that features all the 5G-related components, that is, 5G RAN, 5G UE, and the 5G core, and the other hosts the Near-RT RIC Release J. The virtual machine running the Near-RT RIC is configured

[3]https://docs.srsran.com/projects/project/en/latest/tutorials/source/near-rt-ric/source/index.html

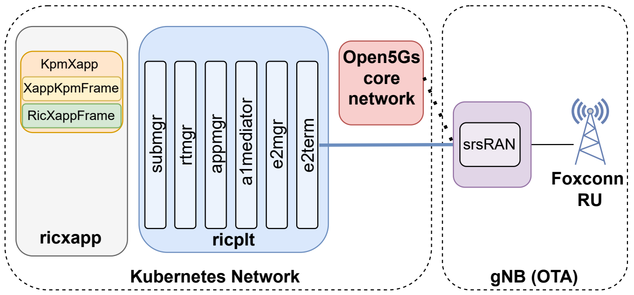

Figure 7: srsRAN over the air

as a single-node Kubernetes cluster, where the E2Term SCTP service of the Near-RT RIC has been exposed to interact with the E2 node provided in srsRAN. For this test, we built the shared library for the service model KPM v3.00 using the FlexRIC SDK, as srsRAN supports KPM v3.00 [19]. We then deployed the xApp using our framework, exploiting the SM designed for this environment. We notice that, by default, srsRAN exposes more KPMs compared to OpenAirInterface.

4.2.2 Over-the-Air srsRAN with 7.2 Split. This deployment builds upon the setup from the previous scenario. We use srsRAN with the same Foxconn RU as shown in Figure 7. We transmit over a 40 MHz bandwidth and run the same iPerf3 tests as in the previous case. As expected, the xApp receives, decodes, and prints metrics from the gNB. Figure 8 compares the traffic generated by iPerf3 using TCP and the traffic calculated by the xApp. We notice a 1% difference in downlink and around 10% difference in uplink. This last difference is due to the fact that the throughput in uplink is limited to 10 Mbps and the xApp measures the performance at the PDCP level, therefore the overhead due the TCP/IP and PDCP headers is higher compared to other experiments. Finally, in some instances, the xApp fails to decode information associated to the update of the list of UEs, when UEs disconnect from the srsRAN base station. This does not happen with the OAI RAN implementation, thus further analysis on the srsRAN side is required.

5 Conclusions, Lessons Learned, and Future Extensions

In this paper, we proposed xDevSM, a flexible and open-source framework to develop xApps for the OSC Near-RT RIC. xDevSM introduces an abstraction layer to handle E2SM encoding/decoding procedures, enabling xApp developers to focus on the control logic of their xApps and moving the service model logic behind simple API. We proved the robustness and effectiveness of xDevSM by deploying xApps based on this framework in various environments with different RAN software. Our results demonstrate that xDevSM reduces the complexity of xApp development and facilitates the validation of solutions and algorithms in O-RAN networks.

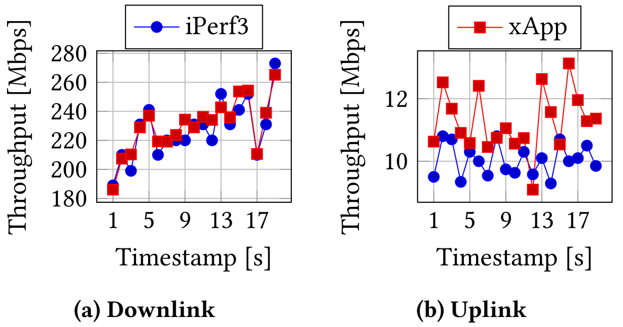

Figure 8: Comparison of iPerf3 (blue dots) and xApp (red square) reported values with srsRAN

The promising results shown in Section 4 position the open-source framework as a platform with a significant potential for extensions and further development. At the time of writing, xDevSM only supports the E2SM KPM service model, with extensions planned for RC. Additional extensions will be implemented for other service models as the O-RAN ALLIANCE WG3 defines them.

To do this, and to change existing E2SMs, xDevSM needs the design of new Python objects that match the corresponding newer and modified underlying structures (currently based on C). To address this for different E2SM versions, the framework supports a set of custom structures that map the E2SM notation structures to the Python objects supported by the framework. Nevertheless, this procedure is performed manually. The underlying C code used to match the SM notation to the custom structure is manually modified, and then the shared library is rebuilt to reflect these changes. As part of our future work, we will consider automated approaches to achieve the same goal.

Additionally, the E2AP encoding/decoding depends on the xApp framework provided by the OSC, and can get out of sync with respect to RAN E2AP implementations. Therefore, as part of our future work, we will track development in OSC and implement continuous testing solutions for xDevSM, the Near-RT RIC, and the RAN implementations considered in this paper.

Acknowledgments

This article is based upon work partially supported by the O-RAN ALLIANCE, the U.S. National Science Foundation under grants CNS-2112471 and CNS-21200447, by the National Telecommunications and Information Administration (NTIA)'s Public Wireless Supply Chain Innovation Fund (PWSCIF) under Award No. 25-60-IF054, and by the National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR) RI SoBigData - Prot. IR0000013.

References

[1] Fransiscus Asisi Bimo et al. 2022. OSC Community Lab: The Integration Test Bed for O-RAN Software Community. In *2022 IEEE Future Networks World Forum (FNWF)*, 513–518. DOI: 10.1109/FNWF55208.2022.00096.

[2] Leonardo Bonati, Michele Polese, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. 2023. OpenRAN Gym: AI/ML development, data collection, and testing for O-RAN on PAWR platforms. *Computer Networks*, 220, 109502.

[3] Chieh-Chun Chen, Mikel Irazabal, Chia-Yu Chang, Alireza Mohammadi, and Navid Nikaein. 2023. FlexApp: Flexible and Low-Latency xApp Framework for RAN Intelligent Controller. In *ICC 2023 - IEEE International Conference on Communications*, 5450–5456. DOI: 10.1109/ICC45041.2023.10278600.

[4] Estefanía Coronado, Shuaib Siddiqui, and Roberto Riggio. 2022. Roadrunner: O-RAN-based cell selection in beyond 5G networks. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 1–7.

[5] Mikel Irazabal and Navid Nikaein. 2024. TC-RAN: A Programmable Traffic Control Service Model for 5G/6G SD-RAN. *IEEE Journal on Selected Areas in Communications*, 42, 2, (Feb. 2024), 406–419.

[6] David Johnson, Dustin Maas, and Jacobus Van Der Merwe. 2021. NexRAN: Closed-loop RAN slicing in POWDER -A top-to-bottom open-source open-RAN use case. In *Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental Evaluation & CHaracterization* (WiNTECH '21). Association for Computing Machinery, New Orleans, LA, USA, 17–23. ISBN: 9781450387033.

[7] Andrea Lacava et al. 2024. Programmable and Customized Intelligence for Traffic Steering in 5G Networks Using Open RAN Architectures. *IEEE Transactions on Mobile Computing*, 23, 4, (Apr. 2024), 2882–2897.

[8] Eugenio Moro, Michele Polese, Antonio Capone, and Tommaso Melodia. 2023. An Open RAN Framework for the Dynamic Control of 5G Service Level Agreements. In *2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 141–146. DOI: 10.1109/NFV-SDN59219.2023.10329597.

[9] Navid Nikaein et al. 2014. OpenAirInterface: A Flexible Platform for 5G Research. *SIGCOMM Comput. Commun. Rev.*, 44, 5, (Oct. 2014), 33–38. DOI: 10.1145/2677046.2677053.

[10] O-RAN Software Community. 2024. https://wiki.o-ran-sc.org/display/ORANSDK/Overview. (2024).

[11] O-RAN Software Community. [n. d.] Near Realtime RAN Intelligent Controller (RIC). Accessed July 2024.

[12] O-RAN Software Community. [n. d.] RIC Applications (RICAPP). Accessed July 2024.

[13] O-RAN Software Community. 2024. ric-app-bouncer repository. Accessed July 2024.

[14] O-RAN Software Community. 2024. ric-app-kpimon-go repository. Accessed July 2024.

[15] O-RAN Software Community. 2022. sim-e2-interface repository. Accessed July 2024.

[16] O-RAN Software Community. 2024. Xapp-frame-py. https://docs.o-ran-sc.org/projects/o-ran-sc-ric-plt-xapp-frame-py/en/latest/overview.html. (2024).

[17] O-RAN Working Group 3. 2021. O-RAN Near-Real-time RAN Intelligent Controller Architecture & E2 General Aspects and Principles 2.00. O-RAN.WG3.E2GAP-v02.01 Technical Specification. (July 2021).

[18] O-RAN Working Group 3. 2021. O-RAN near-real-time RAN intelligent controller E2 service model (E2SM) KPM 2.0. ORAN-WG3.E2SM-KPM-v02.00 Technical Specification. (July 2021).

[19] O-RAN Working Group 3. 2023. O-RAN near-real-time RAN intelligent controller E2 service model (E2SM) KPM 3.0. O-RAN.WG3.E2SM-KPM-R003-v03.00 Technical Specification. (Mar. 2023).

[20] O-RAN Working Group 3. 2021. O-RAN near-real-time RAN intelligent controller E2 service model 2.00. ORAN-WG3.E2SM-v02.00 Technical Specification. (July 2021).

[21] O-RAN Working Group 3. 2021. O-RAN near-real-time RAN intelligent controller E2 service model, ran control 1.0. ORAN-WG3.E2SM-RC-v01.00 Technical Specification. (July 2021).

[22] O-RAN Working Group 3. 2020. O-RAN near-real-time RAN intelligent controller, E2 application protocol 2.00. O-RAN.WG3.E2AP-v02.00 Technical Specification. (July 2020).

[23] Imtiaz Parvez, Ali Rahmati, Ismail Guvenc, Arif I Sarwat, and Huaiyu Dai. 2018. A survey on low latency towards 5G: RAN, core network and caching solutions. *IEEE Communications Surveys & Tutorials*, 20, 4, 3098–3130.

[24] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. 2023. Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges. *IEEE Communications Surveys & Tutorials*, 25, 2, 1376–1411. DOI: 10.1109/COMST.2023.3239220.

[25] Srđan Popić, Dražen Pezer, Bojan Mrazovac, and Nikola Teslić. 2016. Performance evaluation of using Protocol Buffers in the Internet of Things communication. In *2016 International Conference on Smart Systems and Technologies (SST)*, 261–265. DOI: 10.1109/SST.2016.7765670.

[26] Corrado Puligheddu, Jonathan Ashdown, Carla Fabiana Chiasserini, and Francesco Restuccia. 2023. SEM-O-RAN: Semantic and Flexible O-RAN Slicing for NextG Edge-Assisted Mobile Systems. In *Proceedings of IEEE INFOCOM 2023*. New York Area, NJ, USA, (May 2023).

[27] Joao F Santos et al. 2024. Managing O-RAN Networks: xApp Development from Zero to Hero. *arXiv preprint arXiv:2407.09619*.

[28] Robert Schmidt, Mikel Irazabal, and Navid Nikaein. 2021. FlexRIC: an SDK for next-generation SD-RANs. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies* (CoNEXT '21). Association for Computing Machinery, Virtual Event, Germany, 411–425. ISBN: 9781450390989. DOI: 10.1145/3485983.3494870.

[29] Rob Smith, Connor Freeberg, Travis Machacek, and Venkatesh Ramaswamy. 2021. An O-RAN approach to spectrum sharing between commercial 5G and government satellite systems. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*. IEEE, 739–744.

[30] srsRAN. 2023. Open RAN Near-RT RIC and xApp framework. Accessed: 2024-07-10. (2023). https://github.com/srsran/oran-sc-ric.

[31] Maria Tsampazi et al. 2024. PandORA: Automated Design and Comprehensive Evaluation of Deep Reinforcement Learning Agents for Open RAN. *arXiv preprint arXiv:2407.11747*.

[32] Pratheek S Upadhyaya, Nishith Tripathi, Joseph Gaeddert, and Jeffrey H Reed. 2023. Open AI Cellular (OAIC): An Open Source 5G O-RAN Testbed for Design and Testing of AI-Based RAN Management Algorithms. *IEEE Network*.

[33] Davide Villa et al. 2024. X5G: An Open, Programmable, Multi-vendor, End-to-end, Private 5G O-RAN Testbed with NVIDIA ARC and OpenAirInterface. (2024). https://arxiv.org/abs/2406.15935 arXiv: 2406.15935 [cs.NI].

[34] Mohammad Zangooei, Morteza Golkarifard, Mohamed Rouili, Niloy Saha, and Raouf Boutaba. 2024. Flexible RAN Slicing in Open RAN With Constrained Multi-Agent Reinforcement Learning. *IEEE Journal on Selected Areas in Communications*, 42, 2, (Feb. 2024), 280–294.