Frequency Scanning xApps: O-RAN RIC and GNU Radio with RTL-SDR Use Case

Iago Rego, Lucas Medeiros, Vitor Lopes, Vinícius Filho, Vicente Sousa, Eduardo Aranha, Allan Martins, Marcelo Fernandes, Ramon Fontes, Augusto Neto and Guilherme Pupio

Abstract— **The O-RAN Alliance conceived an open architecture model that aims at driving new levels of openness in the radio access network (RAN). Allied to the trend of softwarized networks, the open-access network covers most of the telecommunications system when combined with the Software Defined Radio (SDR) paradigm. O-RAN defines the RAN Intelligent Controller (RIC) as a platform based on microservices for implementing RAN monitoring and control techniques, called xApps. This work uses the O-RAN open-source platform to exemplify its accessibility and versatility to implement xApps with control strategies of a simplified RAN based on GNU Radio. A frequency scanning algorithm is proposed as a use case to demonstrate how SDR platforms allied to the O-RAN paradigm can be easily used in teaching and research initiatives.**

Keywords— **xApps, O-RAN, RIC, GNU Radio, SDR.**

I. INTRODUCTION

Aiming at breaking the proprietary network paradigm, the Open RAN Alliance (O-RAN Alliance) [1], a global organization formed by several institutions that work in the scope of RANs [2], proposes an architecture based on virtualized network elements, white-box, and standardized interfaces. O-RAN is pursuing a vision of openness and intelligence for the next-generation wireless networks, yielding an open-source communication network with lower implementation costs, higher implementation flexibility, and support for native Machine Learning (ML) techniques [2], [3].

The O-RAN proposed architecture introduces a new software-defined RAN model through virtualization. By virtualizing and allocating radio, computer, and storage resources to virtual access elements, the network controller can dynamically create and optimize virtual access elements based on service requirements. This is important for modern communication systems, which require spectrum efficiency, traffic capacity, increased flow, low end-to-end latency, reliability, and an increased number of connected devices. While researchers, companies, and professionals are getting familiar with the O-RAN vision, many challenges of implementing virtualized, open, programmable, and data-driven networks still need to be addressed. Openness, programmability, and disaggregation are just the primary enablers of data-driven applications. However, these are only the first steps toward seamless integration of Artificial Intelligence (AI) techniques and control loops into the communication chain.

Traditionally, RAN improvements are associated with new transmission and reception schemes of transceivers and more efficient radio resource management (RRM) strategies [4]. The time granularity of RRM strategies is the transmission time interval (TTI), acting on a millisecond time scale, with control algorithms running entirely on the base station. In accordance with definitions from O-RAN Working Group 2 [5], a RAN Intelligent Controller (RIC) is defined on top of a traditional RAN to extend the RRM actions using AI-based techniques. Depending on the RRM paradigm, RAN control strategies can be performed in non-real time (RT), operating at a scale greater than 1000 ms, and in near-real time (performing operations between 10 ms and 1000 ms). O-RAN also defines communication interfaces between the architecture entities. The E2 interface, between the RIC and RAN, implements the E2AP protocol, which materializes the bidirectional communication between RIC and RAN through the E2SM service model [6].

Trends of softwarized networks and open protocols cover the entire telecommunications system when allied to the Software-Defined Radio (SDR) paradigm [7]. With the idea of scanning the signal as close to the antenna as possible, these software-based radio units are responsible for processing the system's physical layer signal (whatever technology, e.g., Wi-Fi, 4G LTE, or 5G NR), but under a development platform completely governed by high-level software. These new hardware and software components have been radically changing how the scientific community and the telecommunications industry design, plan, build, deploy, and interact with telecommunications systems, including radio infrastructure [7]. In this way, developing, prototyping, and testing new RRM algorithms and protocols experience new challenges, but with an unprecedented time to market, as the prototype has never been closer to the product than now.

This article aims to present the implementation of four xApps that run on the near-real time RIC O-RAN platform. The O-RAN platform exchanges messages with an FM receiver implemented in GNU Radio, acting as a RAN. By implementing a frequency scan algorithm, we intend to show the accessibility and versatility of implementing measurement and control strategies of a RAN on the RIC platform. Finally, the goal is to contribute to the dissemination and use of SDR platforms, such as GNU Radio, and open platforms, such as the O-RAN RIC. The set of xApps and GNU Radio Python scripts presented herein can be used as an active-learning

Guilherme Pupio is with Lenovo Brazil (e-mail: gpupio@lenovo.com). The rest of authors are with Federal University of Rio Grande do Norte, e-mails: {iago.diogenes.072, lucas.medeiros.114, vitor.lopes.096, vjmtbf, vicente.sousa, eduardo.aranha}@ufrn.edu.br, {allan, mfernandes}@dca.ufrn.br, ramon.fontes@imd.ufrn.br, augusto@dimap.ufrn.br.

strategy, bringing practical experience and fulfilling new requirements of engineering education [8].

A. *Related Works*

Regarding O-RAN based systems, a few works discuss xApps implementation aspects and operation with different Radio Access Technologies (RATs) [9]. A second aspect is the implementation of the Traffic Steering Use Case [9], [10]. Other contributions explore the usage of Machine Learning algorithms to manage connections [11] and to cope with threats in O-RAN [12].

There is an abundance of works focusing on prototyping communication systems via *Software Defined Radio* (SDR). Full stack-system implementation of communication systems are explored in solutions for 3G, 4G and 5G (OpenAirInterface [13]–[15] and srsLTE [16], [17] and Radisys [18]). As RF front-end, Remote Radio Units (RRUs) are implemented on top of commodity hardware. There are different RF front-end hardware like Universal Software Radio Peripheral (USRP) [19], Iris [20], limeSDR [21], and bladeRF [22]. Several recent works have been produced using testbeds of research projects like POWDER-RENEW [23], COSMOS [24], AERPAW [25], Colosseum [26], 5TONIC [27], FED4FIRE+ [28], CORNET [29], FIT [30], Drexel Grid [31].

This paper aims to contribute to O-RAN and GNU Radio communities by providing a simple example of a frequency scanning *xApp* using the O-RAN RIC platform and GNU Radio, implemented on two separate computers. To the best of the authors' knowledge, a set of xApps to provide the closed-loop communication between RIC and GNU Radio has not been presented in literature before.

II. System Model

A. *O-RAN RIC*

The architecture of O-RAN-based systems can be split into three main components [5], [6]. The first component is the non-real time RIC. It is responsible for the non-real time control and optimization of RAN elements and resources. It is also accountable for AI/ML workflow, including model training and updates, and policy-based guidance of applications/features in near-RT RIC. The second component is the near-real time RIC. It is responsible for near-real time control and optimization of O-RAN elements and resources via fine-grained data collection and actions over E2 interface. Finally, the third component is the E2 Node which encompasses all the elements in the front end of the RAN, such as base stations and control nodes.

Although the O-RAN architecture is well defined and established, the implementation, and development of its components are still being heavily updated. The official O-RAN repository [32] is constantly being changed and the RIC architecture is provided as the main development branch. Different versions of the main branch are named according to words whose first letters follow the alphabetical order. Up until now, there have been 5 (five) releases namely Amber, Bronze, Cherry, Dawn, and E. Given that E has been released very recently, for this work all tests and studies were conducted using the Dawn release.

Fig. 1 shows the overall architecture of O-RAN's RIC. The three main components communicate with each other via specific links divided into three main categories: A1, O1, and E2 links. The A1 links are responsible for sending policies from the non-RT RIC present in the Service Management and Orchestration (SMO) to the near-RT RIC. The O1 link is used to transmit large volumes of data such as files and ML models. Finally, the E2 link is used to transmit data from the E2 nodes, such as KPIs (Key Performance Indicators) and devices' status. Each link is managed by a terminator module responsible for abstracting outward communications between RIC components.

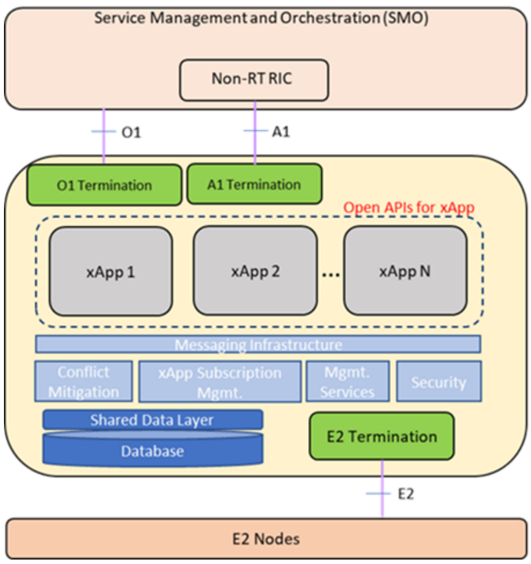

Fig. 1: O-RAN RIC Components. Source: https://www.o-ran.org/.

1) Near-RT RIC: As mentioned before, the near-RT RIC is the RIC module that operates in near-real time (timeframe between 10ms and 1s) and is responsible for control and optimization of the RAN. It consists of several modules that provide infrastructure for the xApps that can perform monitoring and control of the UE via cell-specific metrics. The infrastructure present in the near-RT RIC has two main general-purpose components: A database system and a communications system. The database serves as a standard key-value general-purpose storing facility where xApps and other modules can store temporary or permanent key-value pairs of data. The communications channel aims to provide a subscription/broadcast-type communication between xApps and other modules via named channels.

2) Non-RT RIC: As also mentioned before, the non-RT RIC is the part of the RIC system that runs in non-real time with time frames larger than 1 second. It is situated as part of the SMO and it also has an infrastructure that aims to provide functionalities to the applications that can be installed on its cluster (known as rApps). One of the application types that can be instantiated as modules (rApps) in the non-RT RIC are the ML training apps. Since ML training is a time-consuming task and, in general, takes longer to execute, its services can run in the non-RT RIC.

B. SDR and GNU Radio

Experimental tools to evaluate new designs are key to support academic and industrial research. Hence, network simulators, emulators, and new testbeds have received increasing attention. For O-RAN RIC developing purposes, one can have emulators and simulators for each type of RIC interfaces, such as A1, E2, O1 [6]. Physical SDR platforms, such as the USRP can also be considered. They provide a design solution to rapidly prototyping wireless communications systems as well as a complete wireless communication solution by using open-source software suites such as GNU Radio and OpenAirInterface (OAI). Furthermore, different platforms can work together and thus increase the gains for the experiments. For example, SDR platforms and emulators can bring more realism while providing reasonable scalability.

GNU Radio is a free and open-source software development toolkit that provides signal processing blocks to implement software radios [33]. As an SDR toolkit, the idea is to perform the required signal processing in software instead of using dedicated integrated circuits in hardware. Thus, the same hardware can be used to create many kinds of radios for several different communications standards by using personal computers or servers. With GNU Radio, one can rapidly test RIC communication interfaces and have a fast way to provide functional over-the-air demos. Thus, a GNU Radio instance running at a USRP, or even cheaper RTL-SDR dongles, could be an alternative external emulator for research.

III. Use Case Description, Installation and Deployment

The main idea of this use case is to provide a framework that harnesses the GNU Radio to settle a connection capable of generating signal-strength measurement to near-RT RIC O-RAN's xApp. Then, the near-RT RIC controls the tuned GNU Radio's frequency to get better signal strength.

We use two laptops to implement the use case, the first one with a bare-metal installation of Ubuntu 18.04 LTS and GNU Radio, and the second with a Virtual Box VM of the Ubuntu 18.04 LTS on top of Microsoft Windows 10 for O-RAN RIC. We use Python 3.8.10 and GNU Radio 3.8 on the laptop 1, and Kubernetes v1.16.0, Helm 3, Lens 4.3.5 for the O-RAN RIC Dawn deployment and visualization on machine 2. Fig. 2 shows a block diagram of our deployment. We use a RTL-SDR dongle as our RF front-end platform.

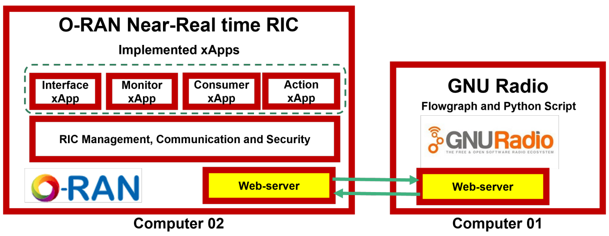

Fig. 2: Block Diagram describing the Use Case.

A. O-RAN RIC xApps

Our xApps aim at scanning the FM frequency range, select the radio station with higher received power, and finally tune the receiver at that frequency. Inspired by O-RAN Traffic Steering Use Case, we implemented four xApps (see Fig.3):

- **Interface xApp** with the following goals:
  - Read KPI from GNU Radio through a web-server;
  - Send read KPI to Monitoring xApp by means of the RIC Message Router (RMR) interface;
  - Send the frequency adjustment to GNU Radio by means of a web-server whenever the Action xApp sends a message through RMR interface.
- **Monitoring xApp** with the following goals:
  - Receive KPI from Interface xApp;
  - Feed SDL (O-RAN Shared Data Layer) with new KPI.
- **Consumer xApp** with the following goals:
  - Read KPI from SDL using wrappers;
  - Test whether the current station is the strongest so far.
  - Send the frequency adjustment to the Action xApp by means of RMR. If it did not scan the entire range yet, it sends a 400kHz shift. If it already scanned the entire range, it sends the frequency of the strongest radio station.
- **Action xApp** with the following goals:
  - Receive the frequency adjustment from Consumer xApp;
  - Close the loop by sending the frequency adjustment to the Interface xApp.

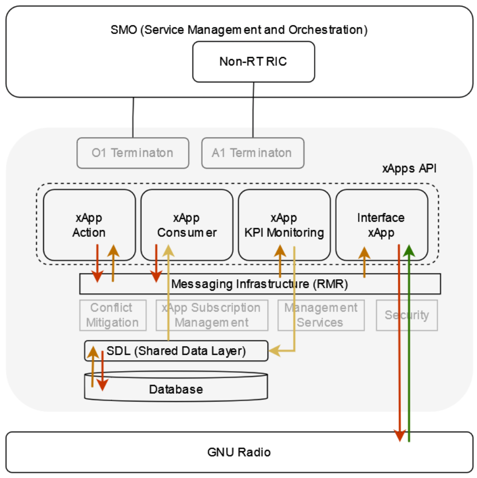

Fig. 3: Proposed architecture and signal flow for the Use Case.

B. GNU Radio Flowgraph and Python Scripts

Our approach on GNU radio is twofold. In the first step (see Fig. 4), we created a flowgraph of a standard wideband FM stereo receiver (found in *gr-analog* examples at `https://github.com/gnuradio/gnuradio/blob/main/gr-analog/examples`) with the inclusion of the *Crtlport Probe* block to export the vector of received signal to flowgraph's Python script (manually edited in the second step). In the second step, we added the python server code to (i) process the vector of the received signal (compute received power); (ii) add a thread using the Python's threading library to host the web server; and (iii) implement the web-based socket server using Python's Flask library. Fig. 5 shows those codes.

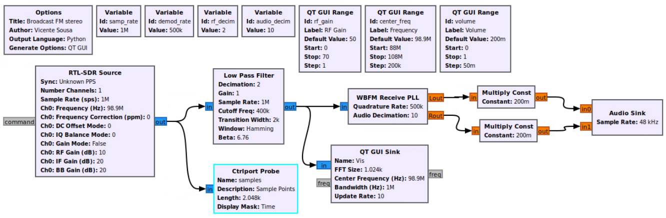

Fig. 4: Reference GNU Radio flowgraph.

IV. Results

As shown in Fig. 3, the use case presented in this paper consists basically in of the same Traffic Steering process, but here, the E2 nodes are replaced by the GNU Radio as the source of KPIs and the entity to be controlled.

Thus, as our main result, Fig. 6 shows the O-RAN/GNU Radio interface working.

The process can be described as follows:

1) The measurement of the tuned FM radio station (named here as KPIs) enters the RIC platform via the Interface xApp coming from GNU Radio; The Interface xApp periodically requests the power and center frequency of the connected station. These KPIs are delivered as notifications on the RMR bus;
2) That notification triggers the Monitoring xApp, which receives these KPIs; Sequentially, it sends the KPI data to be stored in the database via SDL; Finally, the SDL stores the KPI in the database, which is available for pooling and reading.
3) The Consumer xApp periodically performs the pooling for KPIs via SDL. By comparing the KPIs, it stores the frequency of the radio station with higher received power. If the frequency range has not been entirely scanned yet, the Consumer xApp sends a 400kHz shift to any subscribed xApp via RMR. Otherwise, it sends the frequency of the station with higher received power.
4) The Action xApp, which is subscribed to receive the values from the Consumer xApp, sends the corresponding actions to any subscribed xApps via RMR. Each type of action is sent through a different port.
5) The Interface xApp, which is subscribed to the messages from the Action xApp, triggers the GNU Radio server with a new frequency. This new frequency can be a shift of 400kHz from the previously tuned station or the frequency of the radio station with higher received power.
6) The GNU Radio server receives the new frequency and tunes its radio reception. That closes the loop, and the frequency scanning restarts if the whole FM spectrum has not been scanned yet. Otherwise, the best radio is finally tuned.

```
class T(gr.sync_block):
    def __init__(self):
        gr.sync_block.__init__(
            self,
            name='Embedded Python Block',
            in_sig=[np.complex64],
            out_sig=[np.complex64]
        )
        self.history = 0

    def work(self, input_items, output_items):
        output_items[0][:] = 0.0
        dataI = input_items[0].real
        dataQ = input_items[0].imag
        dataI = np.convolve(dataI, np.ones(20)/20.0)
        dataQ = np.convolve(dataQ, np.ones(20)/20.0)
        measurementsRef[0] = np.mean(np.sqrt(dataI**2 +
         dataQ**2))
        return len(input_items[0])
```

(a) Code to process the vector of received signal (calculate received power).

```
def thread_function(tb):
    app = Flask(__name__)
    app.debug = False
    app.use_reloader = False
    @app.route('/measure')
    def measure():
        return '%.4f'%(measurementsRef[0])
    @app.route('/tune')
    def tune():
        dFreq = float(request.args.get('df'))
        newFreq = tb.get_center_freq() + dFreq
        tb.set_center_freq(newFreq)
        return '%.2f'%(newFreq)
    @app.route('/tuneradio')
    def tuneradio():
        dFreq = float(request.args.get('df'))
        #newFreq = tb.get_center_freq() + dFreq
        newFreq = dFreq
        tb.set_center_freq(newFreq)
        strOut='\n ###### \n New tuned FM station: {} MHz
         \n ###### \n'.format(newFreq/1e6)
        print(strOut)
        return '%.2f'%(newFreq)
    @app.route('/getfreq')
    def getfreq():
        central_freq = tb.get_center_freq()
        return '%.2f'%(central_freq)
    app.run(port=6660, host='0.0.0.0')
def startServer(tb):
    x = threading.Thread(target=thread_function, args=(tb,))
    x.start()
    return x
```

(b) Code to add a thread process using the Python's threading library, and to implement a web-based application (socket connection) using Python's Flask library.

Fig. 5: Additional codes of flowgraph's Python script.

Fig. 7 illustrates the GNU Radio side of the experiment, where one can see a log of messages from and to RIC as well as the receiver frequency re-tuning.

V. Conclusions

This contribution presents four xApps to compose a use case of O-RAN RIC and GNU Radio. We show an effective way to connect those open-source platforms deployed in two different machines by means of Python's Flask library. We demonstrate the accessibility and versatility of using O-RAN open and GNU Radio open source codes to exercise modern communication paradigms like SDR, openness, and virtualized network using standardized interfaces. Such methodology can be used to teaching and research initiatives by prototyping real systems at very low cost.

Our further implementations include the usage of O-RAN E2 interface as well as prototyping of more complex radio access on GNU Radio.

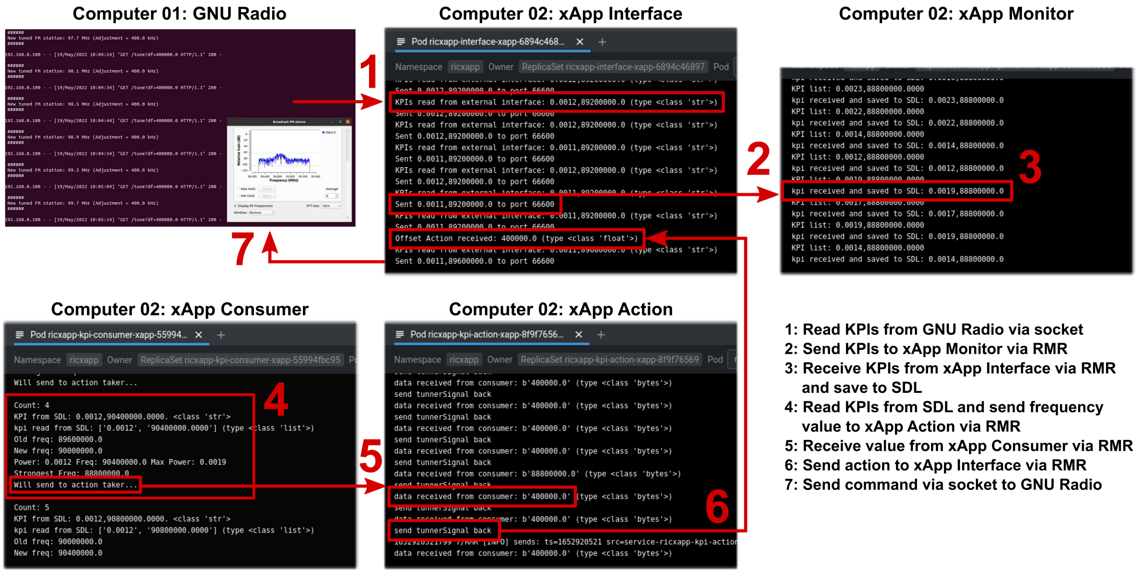

Fig. 6: Use case signaling flow execution.

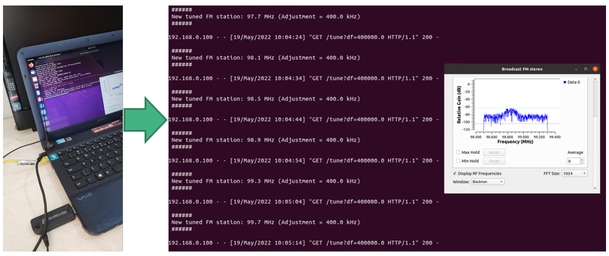

Fig. 7: GNU Radio side of the experiment.

ACKNOWLEDGEMENTS

This research was partially funded by Lenovo, as part of its R&D investment under Brazilian Informatics Law, and by the Coordenacao de Aperfeicoamento de Pessoal de Nivel Superior - Brasil (CAPES) - Finance Code 001.

REFERENCES

[1] "O-RAN Alliance portal." [Online]. Available: https://www.o-ran.org/
[2] O-RAN Alliance, "O-RAN: Towards an Open and Smart RAN," 2018.
[3] X. Liu, *Optical Communications in the 5G Era*. Elsevier Science, 2021.
[4] H. Holma, A. Toskala, and T. Nakamura, *5G Technology: 3GPP New Radio*. Wiley, December 2019.
[5] O-RAN WG2, "O-RAN A1 interface: General Aspects and Principles 2.03," 2021.
[6] O-RAN WG3, "O-RAN Near-Real-time RAN Intelligent Controller Architecture & E2 General Aspects and Principles 2.01," 2022.
[7] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead," *Computer Networks*, 2020.
[8] L. R. L. C. (Presidente), A. de Araujo Freitas Júnior (Relator), A. C. Netto, F. C. de Sá Barreto, and P. M. V. B. B. (Membros), *Diretrizes Curriculares Nacionais do Curso de Graduação em Engenharia*, Ministério da Educação Conselho Nacional de Educação, apr 2019.
[9] M. Dryjanski, L. Kulacz, and A. Kliks, "Toward Modular and Flexible Open RAN Implementations in 6G Networks: Traffic Steering Use Case and O-RAN xApps," *Sensors*, 2021.
[10] R. Brandon, Rahul, Wasi, and Pankaj, "Building the World's First O-RAN-Compliant, AI-Powered, Closed-Loop Near-RT RIC," 2022. [Online]. Available: https://www.mavenir.com/blog/
[11] O. Orhan, V. N. Swamy, T. Tetzlaff, M. Nassar, H. Nikopour, and S. Talwar, "Connection Management xAPP for O-RAN RIC: A Graph Neural Network and Reinforcement Learning Approach," in *ICMLA*, 2021.
[12] R. Bitton, D. Avraham, E. Klevansky, D. Mimran, O. Brodt, H. Lehmann, Y. Elovici, and A. Shabtai, "Adversarial Machine Learning Threat Analysis in Open Radio Access Networks," 2022.
[13] "Open Air Interface portal." [Online]. Available: https://openairinterface.org/
[14] F. Kaltenberger, A. Silva, A. Gosain, L. Wang, and T.-T. Nguyen, "OpenAirInterface: Democratizing Innovation in the 5G Era," *Computer Networks*, 2020.
[15] L. Bonati, S. D'Oro, L. Bertizzolo, E. Demirors, Z. Guan, S. Basagni, and T. Melodia, "Cellos: Zero-touch softwarized open cellular networks," *Computer Networks*, 2020.
[16] I. Gomez-Miguelez, A. Garcia-Saavedra, P. Sutton, P. Serrano, C. Cano, and D. Leith, "srsLTE: an open-source platform for LTE evolution and experimentation," 2016.
[17] A. Puschmann, P. D. Sutton, and I. Gómez, "Implementing NB-IoT in Software - Experiences Using the srsLTE Library," *CoRR*, 2017.
[18] "Radisys." [Online]. Available: https://radisys.com
[19] "Universal Software Radio Peripheral." [Online]. Available: https://www.ettus.com/products/
[20] "Skylark Wireless." [Online]. Available: https://www.skylarkwireless.com
[21] "LimeSDR portal." [Online]. Available: https://wiki.myriadrf.org/LimeSDR
[22] "Nuand bladeRF." [Online]. Available: https://www.nuand.com
[23] "Platform for Open Wireless Data-driven Experimental Research." [Online]. Available: https://www.powderwireless.net
[24] "Cloud Enhanced Open Software Defined Mobile Wireless Testbed for City-Scale Deployment (COSMOS)." [Online]. Available: http://www.cosmos-lab.org
[25] "Aerial Experimentation and Research Platform for Advanced Wireless (AERPAW)." [Online]. Available: https://www.aerpaw.org
[26] "Colosseum." [Online]. Available: https://www.colosseum.net
[27] "5TONIC." [Online]. Available: https://www.5tonic.org
[28] "Federation 4 Future Internet Research and Experimentation Plus (FED4FIRE+)." [Online]. Available: https://www.fed4fire.eu
[29] "COgnitive Radio NETwork (CORNET)." [Online]. Available: https://cornet.wireless.vt.edu
[30] "Future Internet of Things (FIT)." [Online]. Available: https://fit-equipex.fr
[31] "Drexel Grid SDR testbed." [Online]. Available: https://research.coe.drexel.edu/ece/dwsl/research/drexel-grid-sdr-testbed
[32] "O-RAN Software Community Repository." [Online]. Available: https://wiki.o-ran-sc.org/display/REL/Releases
[33] "GNU Radio portal." [Online]. Available: https://www.gnuradio.org/