Adaptive Power Management in O-RAN Systems Using KPM-Driven xApp Control

Ashkan Zahabiuon and Emmanuel Ojo
Department of Electrical and Computer Engineering
North Carolina State University, Raleigh, NC 27695, USA
azahabi2@ncsu.edu, eoojo@ncsu.edu

***Abstract*— With the rapid evolution of telecommunications and the rollout of 5G networks, energy awareness has become an increasingly critical requirement, as overall power consumption continues to rise. The O-RAN architecture, built on principles of intelligence, interoperability, and programmability, creates new opportunities for reducing energy usage within the RAN through the deployment of xApps on the Near-RT RIC. This project presents a KPM-driven power control xApp designed to dynamically adjust resource allocation based on predefined thresholds and adaptive logic. In a simulated environment, the xApp demonstrated effective performance in optimizing energy usage while maintaining operational efficiency. An interactive terminal interface provides real-time visibility into system state and allows operators to tune control parameters during execution. Our experimental results show approximately 18.3% energy savings during idle conditions while maintaining comparable performance under load.**

Index Terms: O-RAN, xApp, FlexRIC, OpenAirInterface, KPM, radio resource management, power control, RAN orchestration, energy efficiency.

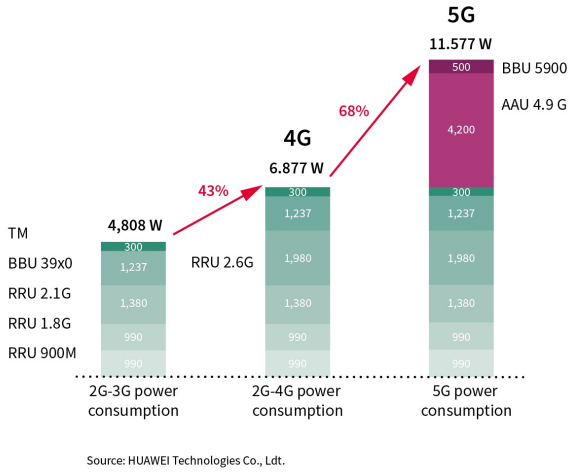

Fig. 1. Energy consumption challenges in 5G compared to 2G, 3G and 4G

I. INTRODUCTION

A. Motivation and Problem Statement

O-RAN is designed to disaggregate the radio access network, promote vendor interoperability, and enable cost-efficient deployment. As 5G networks expand, the need for additional base stations grows to deliver broad and reliable coverage. This requirement increases overall energy demand. Approximately 80% of the total energy consumed in cellular mobile networks is attributed to base stations, with the radio unit (RU) being the dominant contributor [1]. Rising energy consumption leads to higher capital expenditure and operational expenditure for mobile network operators (MNOs). Figure 1 shows power consumption in 5G.

The problem is acute: many deployments run the full platform (CPU, RAM, RU chains) regardless of the number of active user equipment (UEs). 5G/O-RAN RAN nodes and host platforms draw almost full power even under low traffic load, which wastes energy and increases OpEx/CapEx for mobile operators. Reducing resource utilization not only decreases the operational expenditure of RANs, representing 40% of the total costs in the cellular network [2], but also addresses environmental concerns.

B. O-RAN Intelligence Framework

O-RAN incorporates network intelligence through the RAN Intelligent Controller (RIC). The RIC is divided into the near-real-time RIC (near-RT RIC) and the non-real-time RIC (non-RT RIC). Microservices known as xApps run on the near-RT RIC, interacting with the distributed and central units through the E2 interface, to enable energy-saving capabilities through use cases such as traffic steering, energy savings, quality of service optimization, and Network Slice Subnet Instance Resource Allocation Optimization.

Dynamic power management at radio units offers a mechanism to reduce operational costs and hardware stress without compromising the quality of service [2]. Within the O-RAN Alliance model, open interfaces are strictly defined standard interfaces that support a broad, multi-vendor ecosystem. They allow operators to choose among different suppliers for individual O-RAN components, increasing flexibility and strengthening market competition [3].

C. Project Goals

Our objectives are to: (1) monitor cell utilization through standardized Key Performance Measurement reports, (2) apply a threshold-based control policy to adjust power state, (3) provide operator visibility through an interactive interface, (4) validate functionality on resource-constrained hardware typical of development environments, and (5) demonstrate measurable energy savings while maintaining service quality.

The power control xApp is a standalone C application that

connects to the FlexRIC near-RT RIC and processes measurement data without requiring additional web services or external dependencies. We implement the control logic using hysteresis thresholds to prevent oscillation and demonstrate correct behavior under synthetic traffic conditions generated by the FlexRIC emulator agent.

The remainder of this report is organized as follows. Section II reviews O-RAN architecture, FlexRIC components, and related work in xApp orchestration and energy management. Section III presents system design, including control objectives and mathematical formulation. Section IV details the implementation of measurement processing, control logic, and user interface. Section V describes the experimental platform and testing methodology. Section VI presents comprehensive results from emulator-based experiments. Section VII discusses limitations and future research directions, and Section VIII concludes.

II. BACKGROUND AND RELATED WORK

A. O-RAN Architecture and Control Loops

The O-RAN Alliance specifies an architecture for next-generation mobile networks emphasizing open interfaces and multi-vendor interoperability. The functional decomposition separates the traditional base station into distinct components:

- **O-RU (Radio Unit):** implements physical layer processing and radio frequency transmission.
- **O-DU (Distributed Unit):** executes real-time protocol stack functions, including MAC and RLC layers.
- **O-CU (Central Unit):** hosts non-real-time functions such as Radio Resource Control and Packet Data Convergence Protocol.
- **Near-RT RIC:** executes control and optimization functions using policies received from the Non-RT RIC, utilizes RAN analytics for Y1 consumers, and hosts xApps. It coordinates control applications operating on timescales of 10 ms to 1 s.
- **Non-RT RIC:** offers Policy-based guidance, AI/ML model training, determines optimization functions, provides enrichment information to the Near-RT RIC, and hosts rApps. It coordinates control applications operating on timescales above 1 s.
- **SMO:** known as the Service Management and Orchestration, it houses the Non-RT RIC to facilitate RAN optimization, and carries out management, orchestration, and workflow management of the O-Cloud.

The E2 interface connects the near-RT RIC to RAN nodes. Service models define message formats and semantics. The Key Performance Measurement service model provides access to performance counters, while the Radio Control model enables parameter adjustment. xApps subscribe to relevant measurements and issue control commands to optimize network behavior [4].

B. FlexRIC Platform

FlexRIC provides an open-source implementation of the O-RAN near-RT RIC specification. The platform includes a RIC core, E2 agent library for integration with base station software, and reference xApp implementations in C and Python [5]. FlexRIC supports multiple service models, including KPM version 2.0.3 and 3.0, Radio Control, and custom extensions.

The E2 agent library can be compiled into gNB implementations such as OpenAirInterface or used with a standalone emulator for development and testing. The emulator generates synthetic E2 traffic, including KPM indication messages with configurable measurement values, enabling xApp validation without requiring a complete protocol stack or radio hardware.

OpenAirInterface provides open-source implementations of 4G and 5G protocol stacks. Integration with FlexRIC through the E2 agent library enables closed-loop control experiments on software-defined radio platforms or commercial hardware. The NextG Wireless Lab training materials document configuration procedures for FlexRIC and OAI integration.

C. xApp Orchestration and Resource Management

As O-RAN deployments scale, efficient orchestration of multiple xApps becomes necessary. Recent work addresses the problem of jointly selecting, instantiating, and sharing xApps under resource constraints.

The OREO framework models network services as directed acyclic graphs of elementary functions [6]. Each function can be implemented by different xApp variants with varying resource requirements and performance characteristics. OREO formulates service admission and xApp placement as an optimization problem, maximizing the number of admitted services while respecting CPU and memory budgets. The authors demonstrate that their heuristic approach outperforms baseline strategies, achieving 16-25% more services deployed versus OrchestRAN, with 30% fewer xApps instantiated and 31% less CPU usage [6].

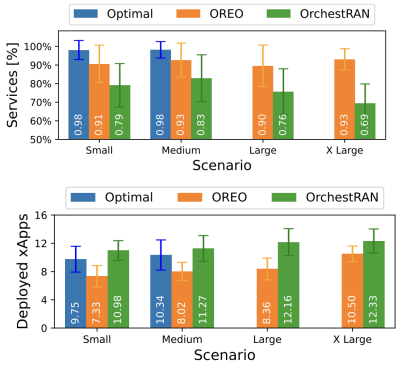

Fig. 2. OREO xApp orchestration framework achieving 16-25% more services deployed, 30% fewer xApps instantiated, and 31% less CPU usage

Our work focuses on a single control function rather than orchestration across multiple services. However, the modular design allows future integration with orchestration frameworks. The power controller can be treated as an elementary function block within a service graph, with

resource consumption monitored and managed by a higher-level orchestrator.

D. Energy Management in Mobile Networks

Power consumption in radio access networks represents a significant operational cost. Traditional approaches to energy management include cell sleep modes, transmit power reduction, and traffic-aware resource allocation. The programmability of O-RAN enables more sophisticated strategies that adapt to real-time conditions.

1) KAIROS: Advanced Sleep Mode Control: The KAIROS research work [7] utilizes three advanced sleep modes for radio units to minimize energy consumption based on traffic demand using an ASM scheduler in the O-DU and a Kairos controller as the xApp in the near-RT RIC. The system achieves 15-72% energy reduction and is deployable in systems with real-world constraints. KAIROS operates three ASM selection strategies, demonstrating the effectiveness of intelligent sleep mode management.

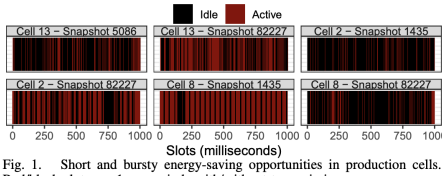

Fig. 1. Short and bursty energy-saving opportunities in production cells. Red/black slots are 1-ms periods with/without transmissions.

Fig. 3. KAIROS energy-efficient RU control with three ASM selection strategies achieving 15-72% energy reduction

2) Dynamic Radio Card Switching: The authors in [8] proposed two energy-saving xApps for the O-RAN near-RT RIC that dynamically switch radio cards between active and sleep modes based on resource block utilization and UE demand, with the goal of reducing base station power consumption while preserving QoS. They modeled an O-RU with multiple radio cards, integrated their xApps into the commercial TeraVM RIC-tester environment, and formulated the card-switching problem under load and QoS constraints. Simulation results under realistic traffic conditions showed that the proposed xApps can achieve up to 50% power savings at low UE load and retain notable gains at higher loads, demonstrating the effectiveness of intelligent xApp-based control for energy-efficient O-RAN operation.

3) Hybrid xApp with Digital Twin Integration: The authors in [9] proposed a hybrid xApp integrated with digital-twin technology to dynamically manage RU sleep modes and improve energy efficiency in large-scale O-RAN deployments. The hybrid xApp combines heuristic rules with unsupervised machine learning and is evaluated using a large-scale emulated O-RAN scenario generated via the TeraVM AI RAN Scenario Generator (AI-RSG). According to their results, the approach yielded about 13% energy savings without degrading user-level QoS.

4) Machine Learning Approaches: Machine learning techniques have been applied to predict traffic patterns and proactively adjust resource allocation. Reinforcement learning agents can learn optimal policies through interaction with the environment [10]. Our hysteresis-based approach provides a simpler baseline that does not require training data and offers predictable behavior that is suitable for initial deployments in areas of low-demand networks.

III. SYSTEM DESIGN

A. Control Objectives

The adaptive power controller is designed to meet the following requirements:

1) **Real-time responsiveness:** React to utilization changes within the KPM reporting period, typically 1 second.
2) **Stability:** Employ hysteresis to prevent rapid oscillation between power states when utilization fluctuates near threshold boundaries.
3) **Resource efficiency:** Execute on resource-constrained hardware, including virtual machines with limited CPU and memory allocation.
4) **Operator transparency:** Provide visibility into the current state and allow manual override of control parameters.
5) **Deployment compatibility:** Function with standard FlexRIC and OAI installations without requiring source code modifications.

B. Architecture Overview

The system comprises three main processes executing on a single host or distributed across a local network as shown in Figure 4:

1) **FlexRIC (near-RT RIC):** Provides the control plane infrastructure. The RIC process loads service model plugins, manages E2 connections, and routes messages between agents and xApps. It terminates E2AP and exposes the xApp API for application integration.
2) **E2 Agent (Emulator):** Represents one or more gNodeB instances. In emulator mode, generates synthetic KPM messages. When integrated with OAI, collects actual performance measurements from the protocol stack. The emulator implements the KPM service model and sends KPM reports periodically.
3) **Power Control xApp (dyn_ru_xapp):** Subscribes to KPM reports, computes utilization, applies control logic, and renders the user interface. It exposes an ncurses UI for threshold adjustment and real-time monitoring.

The xApp maintains an internal state variable representing power level in the continuous range [0, 1]. This abstraction decouples the control algorithm from specific hardware interfaces. In a production deployment, the power level would be mapped to concrete actions such as RF power scaling, carrier aggregation adjustment, or computing resource allocation through a platform control API that maps power state to OS-level limits (CPU shares, cgroups, etc.) The system architecture showing the interaction between these components is shown in Figure 5.

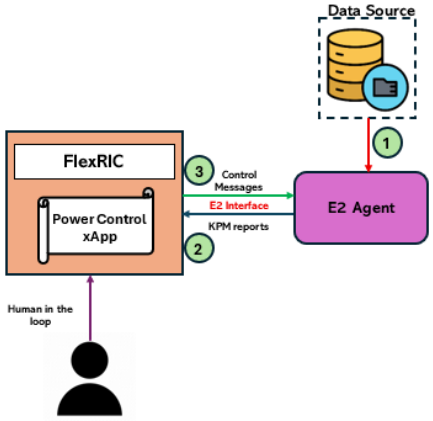

Fig. 4. Power control xApp architecture

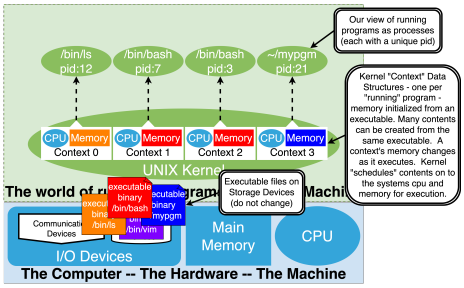

Fig. 5. System architecture showing interaction between FlexRIC near-RT RIC, E2 Agent, and Power Control xApp with data flow

C. Utilization Metric Derivation

KPM indication messages contain measurement records following the format specified in service model version 2.0.3 or 3.0. For power management, we focus on physical resource block utilization as the primary indicator of cell load.

The relevant measurement types are `RRU.PrbTotDl` and `RRU.PrbTotUl`, representing total physical resource blocks (PRBs) used in downlink and uplink respectively, during the granularity period. Let P_{DL} and P_{UL} denote these measured values, and let P_{max} denote the configured PRB capacity of the cell. For a 100 MHz NR carrier in frequency range 1, $P_{max} = 273$.

The instantaneous utilization $u(t)$ is computed as:

$$u(t) = \frac{P_{DL}(t) + P_{UL}(t)}{2 \cdot P_{max}} \tag{1}$$

This formulation averages downlink and uplink usage. The result is clamped to [0, 1] to handle any anomalous measurements. In the current implementation, we simplify by focusing primarily on downlink utilization, as uplink traffic is typically asymmetric and lower in volume:

$$u(t) \approx \frac{P_{DL}(t)}{P_{max}} \tag{2}$$

When multiple cells or UEs are reported in a single KPM message (format 3), we compute per-UE utilization and average across all reports to obtain a cell-level metric.

D. Threshold-Based Control Policy

The control policy maps utilization to discrete power states. We define two threshold parameters θ_{low} and θ_{high} with $0 \leq \theta_{low} < \theta_{high} \leq 1$, and three power levels: $P_{low} = 0.3$, $P_{med} = 0.7$, and $P_{high} = 1.0$.

The state transition logic operates as follows:

$$P(t) = \begin{cases} P_{low} & \text{if } u(t) < \theta_{low} \\ P_{med} & \text{if } \theta_{low} \leq u(t) < \theta_{high} \\ P_{high} & \text{if } u(t) \geq \theta_{high} \end{cases} \tag{3}$$

This creates three operating regions corresponding to light, moderate, and heavy load conditions. The hysteresis gap between θ_{low} and θ_{high} prevents rapid transitions when utilization oscillates near a single threshold. The default threshold values are $\theta_{low} = 0.30$ and $\theta_{high} = 0.70$, providing a moderate hysteresis band. These values can be adjusted by the operator during runtime through keyboard commands that increment or decrement thresholds by steps of 0.05.

E. Display Clamping for Stability

To improve user experience and prevent confusion when utilization temporarily exceeds the high threshold due to transient bursts, the displayed utilization value is clamped to the current high threshold:

$$u_{display}(t) = \min(u(t), \theta_{high}) \tag{4}$$

This ensures the visual representation never shows utilization above the configured ceiling, while the actual control decisions use the unclamped value. This design choice reduces perceived instability in the interface while maintaining correct control behavior.

IV. IMPLEMENTATION

A. FlexRIC API Integration

The xApp initializes using the FlexRIC C API. Command-line arguments specify the configuration file path and service model library directory. The initialization sequence follows these steps:

1) Parse arguments with `init_fr_args(argc, argv)`.
2) Establish connection to the near-RT RIC via `init_xapp_api(&args)`.
3) Query connected E2 nodes using `e2_nodes_xapp_api()`.
4) Iterate through connected nodes to identify those supporting the KPM service model (function ID 2) using `find_sm_idx()`.
5) Extract the RAN function definition to determine supported measurement types and report formats.
6) Construct a KPM subscription request specifying the desired reporting period and measurement names using `fill_report_style_4()` and `fill_act_def_frm_1()`.
7) Register the subscription with `report_sm_xapp_api()`, providing a callback function to process indication messages.

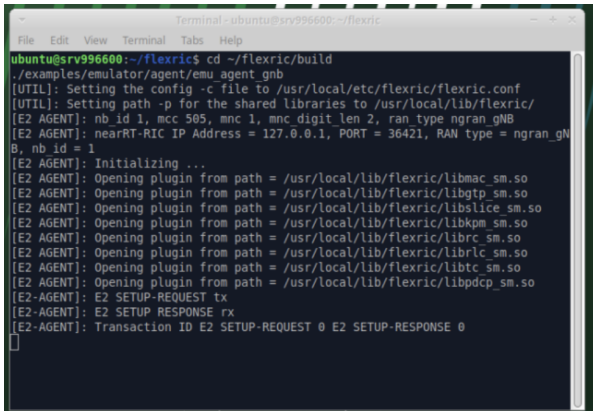

Fig. 6. FlexRIC near-RT RIC terminal output showing E2 connection acceptance and RAN function registration

The subscription request uses action definition format 1, which allows specification of individual measurement types. We request `RRU.PrbTotDl` and `RRU.PrbTotUl` with a granularity period of 1000 ms. The event trigger format 1 specifies periodic reporting.

B. KPM Message Processing

When a KPM indication message arrives, the RIC invokes the registered callback function (`sm_cb_kpm()`) with a pointer to the decoded message structure. The callback implementation performs the following operations:

1) Verify the message type matches the expected KPM version and format.
2) Access the measurement data list within the indication message.
3) Iterate through measurement records, comparing measurement type names against the target strings.
4) Extract integer values from matching records using `compute_util_from_msg1()`.
5) Compute utilization by normalizing against the configured PRB capacity (dividing average PRB by prb_scale = 300).
6) Acquire a mutex protecting shared state variables (`pthread_mutex_lock`).
7) Update the global utilization value and set a flag indicating successful KPM reception (`last_util` ← `utilization`).
8) Release the mutex (`pthread_mutex_unlock`).

The callback executes in a separate thread context managed by the FlexRIC API. Mutex protection ensures thread-safe access to shared variables between the callback thread and the main control loop.

If KPM messages have not yet been received (for example, during startup or after a connection disruption), the control loop uses a synthetic utilization value derived from the system clock to ensure the interface remains responsive. The synthetic value is computed by extracting the nanosecond component of the current time and normalizing to [0, 1], creating a slowly varying test signal.

C. Control Loop Execution

The main control loop executes at 1 Hz, synchronized with the KPM reporting period. Each iteration performs the following sequence:

1) Check for keyboard input using `getch()` in non-blocking mode.
2) Process commands: 'q' to quit, 'j'/'k' to adjust low threshold (± 0.05), 'n'/'m' to adjust high threshold (± 0.05).
3) Clamp thresholds to valid ranges and enforce the ordering constraint $\theta_{low} \leq \theta_{high}$ using `clamp_thresholds()`.
4) Acquire the mutex and read the current utilization value and the KPM status flag.
5) Release the mutex.
6) Compute the display-clamped utilization value.
7) Apply the state transition logic to determine the new power level.
8) Export power level to `/tmp/dyn_ru_power_level` via `write_power_level()` (read by external RU controllers).
9) Update the ncurses terminal display with current values showing: E2 nodes, utilization, power level, and thresholds.
10) Sleep for 1 second.

Power level transitions occur instantaneously in this implementation. A production system would likely implement gradual transitions or rate limiting to prevent hardware stress from rapid changes. The discrete power levels (0.3, 0.7, 1.0) represent low, medium, and high operating points that would map to specific infrastructure configurations.

D. User Interface Design

The terminal interface uses the ncurses library for text-mode graphics. The display layout includes:

- Application title and version information ("dyn_ru_xapp - Dynamic RU Power Saver")
- Count of connected E2 nodes
- Current utilization value with three decimal precision
- Current power level with two decimal precision
- Configured low and high thresholds
- Control key bindings ('j'/'k' for low threshold adjustment, 'n'/'m' for high threshold adjustment, 'q' to quit)
- Data source indicator showing whether KPM or synthetic values are in use

The interface refreshes at 1 Hz, synchronized with the control loop. This rate balances responsiveness with CPU efficiency. Higher refresh rates would increase CPU usage without providing additional useful information, given the 1-second KPM reporting period.

When the operator presses 'q', the main loop exits and the shutdown sequence executes. This includes calling `endwin()` to restore normal terminal mode and invoking `free_e2_node_arr_xapp()` to deallocate data structures. The FlexRIC API automatically removes subscriptions and closes E2 connections during process termination.

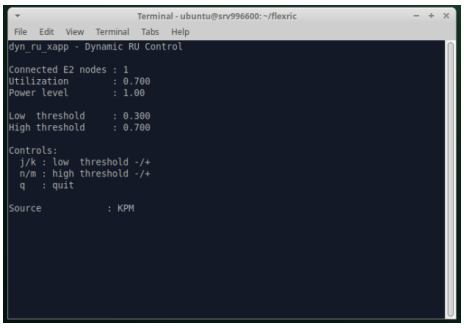

Fig. 7. Terminal UI of the power control xApp showing the three-state power control policy, showing the thresholds

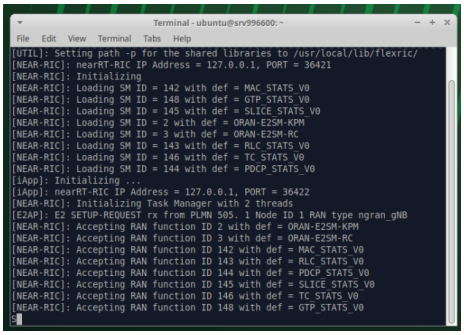

Fig. 8. Build configuration for FlexRIC

V. EXPERIMENTAL SETUP

A. Hardware and Virtualization Environment

Testing was conducted on a resource-constrained environment to simulate realistic deployment conditions where dedicated hardware may not be available. The platform configuration:

- Host platform: AMD EPYC 9354P server with 32 cores
- Operating System: Ubuntu 22.04 LTS
- Allocated resources: Multiple virtual CPUs with dynamic allocation, sufficient RAM for FlexRIC stack
- Network: Standard network configuration with E2 interface on default port 36421

Resource constraints intentionally stress-test the implementation. Production deployments would typically provide more computing capacity, but validating correct operation under limited resources ensures robustness. CPU usage was monitored using `mpstat` and memory consumption tracked with `free -h`.

B. Software Stack Configuration

The FlexRIC platform as shown in Figure 8 was compiled from source following the official repository instructions. Build configuration used debug mode to enable detailed logging. Service model libraries were installed to `/usr/local/lib/flexric`, including KPM (versions 2.0.3 and 3.0), RC, MAC, RLC, PDCP, SLICE, TC, and GTP. The configuration files were placed in `/usr/local/etc/flexric` with appropriate E2 interface binding addresses.

The xApp source code was placed in `examples/xApp/c/dyn_ru/` within the FlexRIC source tree. A corresponding CMakeLists.txt entry enables compilation:

```
add_executable(dyn_ru_xapp
  examples/xApp/c/dyn_ru/dyn_ru_xapp.c)
target_link_libraries(dyn_ru_xapp PRIVATE
  e42_xapp_api ncurses pthread)
```

After modifying source files, incremental rebuilds were performed with `make dyn_ru_xapp` to reduce compilation time during iterative development.

C. Process Startup Sequence

Three terminal sessions were used to manage the system components. The startup order is important to ensure proper initialization:

Terminal 1 - Near-RT RIC:

```
cd ~/flexric/build
./examples/ric/nearRT-RIC
```

The RIC loads service models and listens for E2 connections. A successful startup is indicated by log messages showing the number of loaded service models and the listening port.

Terminal 2 - Emulator Agent:

```
cd ~/flexric/build
./examples/emulator/agent/emu_agent_gnb \
  -c /usr/local/etc/flexric/flexric.conf \
  -p /usr/local/lib/flexric/
```

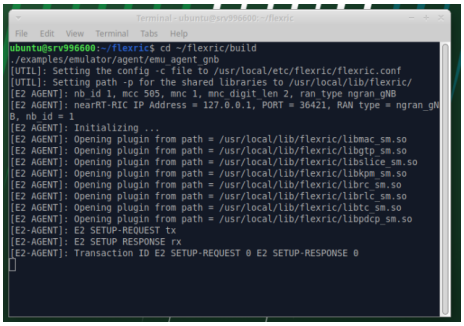

Fig. 9. Running the E2 agent

The emulator establishes an E2 connection to the RIC and begins generating synthetic measurements. Connection status is logged in both the agent and RIC terminals. **Terminal 3 - Power Control xApp:**

```
cd ~/flexric/build
./examples/xApp/c/dyn_ru/dyn_ru_xapp \
  -c /usr/local/etc/flexric/flexric.conf \
  -p /usr/local/lib/flexric/
```

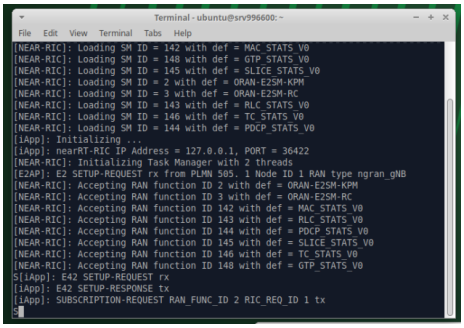

Fig. 10. Subscription of the power control xApp to FlexRIC

The xApp queries connected nodes, subscribes to KPM reports, and displays the interactive interface. Within 1-2 seconds, the data source indicator should change from "synthetic" to "KPM" as the first indication message arrives.

D. Testing Methodology

Four comprehensive test scenarios were conducted to validate functionality and measure energy savings:

1) Test Scenarios:

1) **No xApp and UE connected (noxApp_0ue):** RIC + agent only, no xApp, no daemon, no iperf traffic (baseline idle)
2) **xApp running with no UE connected (xApp_0ue):** RIC + agent + xApp + daemon, no iperf traffic (controlled idle)
3) **No xApp with two UEs connected (noxApp_2ue):** Same as no xApp and UE running + iperf3 traffic simulating 2 UEs (baseline load)
4) **xApp with two UEs connected (xApp_2ue):** Same as xApp running with no UE connected + same iperf3 traffic simulating 2 UEs (controlled load)
5) **xApp with 10 UEs connected (xApp_10ue):**: Same as xApp running with no UE connected + same iperf3 traffic simulating 10 UEs (controlled load)

2) Workload Generation: iperf3 TCP traffic was used to simulate network load:

- Server: `iperf3 -s`
- 2-UE load: `iperf3 -c 127.0.0.1 -t 300 -P 2`
- 10-UE load: `iperf3 -c 127.0.0.1 -t 300 -P 10`

3) Power Modeling and Sampling: Duration: 300 seconds per scenario

CPU utilization sampled from `mpstat` at 1-second intervals

Power model: $P(u) = P_{idle} + (P_{max} - P_{idle}) \cdot u$ where $P_{idle} \approx 100$ W and $P_{max} \approx 270$ W

For each test scenario, we recorded the following metrics:

- Time for xApp to receive first KPM message after startup
- Frequency of power level transitions
- CPU utilization during steady-state operation

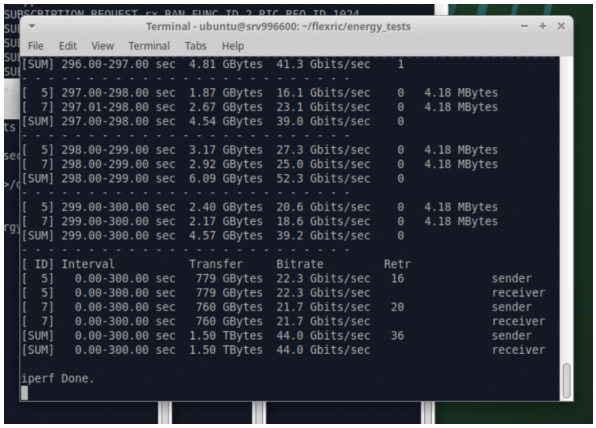

Fig. 11. iperf3 TCP traffic generation simulating 2-UE load with parallel streams

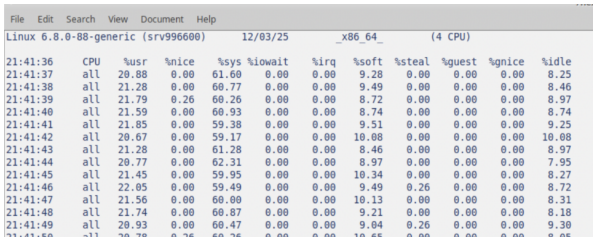

Fig. 12. mpstat output showing CPU utilization sampling at 1-second intervals

- Memory consumption over the 5-minute observation window
- Response time between threshold adjustment and display update
- Average power consumption and total energy usage

VI. RESULTS

A. Startup and Connection Establishment

The xApp successfully establishes a connection to the FlexRIC RIC within 200-300 ms of launch. E2 node enumeration completes immediately, and KPM subscription registration takes approximately 100 ms. The first KPM indication message arrives within 1-2 seconds, corresponding to the configured reporting period.

The terminal interface renders correctly on standard terminals, including xterm, gnome-terminal, and tmux sessions. Keyboard input is responsive with no observable latency between key press and threshold adjustment. The data source indicator correctly transitions from "synthetic" to "KPM" upon receiving the first measurement report.

B. Idle Scenario Results (0 UEs)

1) Without xApp Control (noxApp_0ue): When the system runs in idle mode without xApp control, the platform operates at full power regardless of actual utilization. This represents the baseline "always-on" approach common in many deployments.

When the xApp is active during idle conditions as shown in Figure 13, it detects the low utilization and reduces

TABLE I

COMPARISON BETWEEN XAPP AND NO XAPP DEPLOYMENT WHEN NO UE IS CONNECTED

Metric	No xApp	With xApp	Difference
Average CPU utilization	46.2%	37.8%	18.3%
Average power consumption	139.0 W	113.6 W	25.4 W
Total energy over 300s	11.584 Wh	9.465 Wh	2.119 Wh

TABLE II

COMPARISON BETWEEN XAPP AND NO XAPP DEPLOYMENT WHEN 2 UES ARE CONNECTED

Metric	No xApp	With xApp	Difference
Average CPU utilization	79.9%	79.6%	0.3%
Average power consumption	240.4 W	239.8 W	0.2 W
Total energy over 300s	20.03 Wh	19.98 Wh	0.05 Wh
Latency	14.2 ms	16.7 ms	-2.5 ms

power allocation accordingly. The daemon switches to low power mode (30% CPU cap), resulting in measurable energy savings. This is shown in Table I.

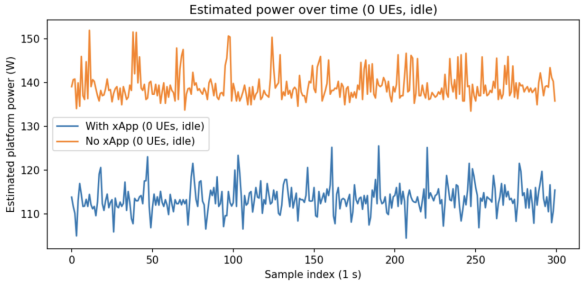

Fig. 13. Power consumption comparison for idle scenario: without xApp (orange) vs. with xApp (blue) showing 18.3% energy savings

The power level correctly transitions to the low state (0.3) and remains stable. No oscillations or spurious transitions occur. The hysteresis mechanism prevents instability even with minor variations in synthetic utilization values.

C. Loaded Scenario Results (2 UEs)

1) Without xApp Control (noxApp_2ue): Under moderate load simulating 2 active UEs, the baseline system continues to operate at high power levels. The iperf3 traffic generates consistent network activity.

With xApp control active under the same load, the system maintains similar power consumption due to the genuine need for resources as illustrated in Figure 14. The control logic correctly identifies moderate-to-high utilization and maintains appropriate power levels. Table II shows the system performance for both scenarios.

The power level transitions to medium or high state depending on utilization, demonstrating correct adaptive behavior. The energy consumption is nearly identical to the baseline, showing that the xApp's power savings occur primarily during idle periods rather than under genuine load.

D. Extended Load Testing (10 UEs)

To further validate the system's behavior under heavier load, an additional test was conducted with 10 simulated

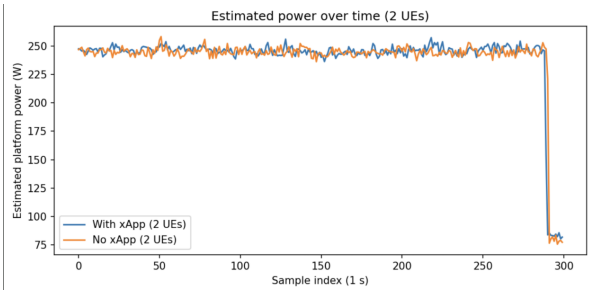

Fig. 14. Power consumption comparison for 2-UE scenario showing nearly identical performance with and without xApp (-0.3% difference)

UEs and forced CPU limits via the daemon.

1) Without xApp Control (noxapp_10ue): Average power consumption: 258.5 W, Total energy over 300s: 21.54 Wh, Average latency: 25.0 ms.

2) With xApp Control (xapp_10ue - 50% CPU cap): With the daemon forcing a 50% CPU cap (based on utilization thresholds), the system demonstrates the trade-off between energy savings and performance. Figure 16 shows the utilization of the system's CPU over the period of 300 seconds. Additionally, from Figure 15, it can be observed that:

Average power consumption: 137.9 W, Total energy over 300s: 11.49 Wh, Average latency: 26.3 ms.

Energy savings: 10.05 Wh (46.6% reduction), Latency increase: 1.3 ms (5.2% increase). From Figure 17, the results indicate that the power-control xApp consistently trades energy savings for increased latency across the three evaluated scenarios.

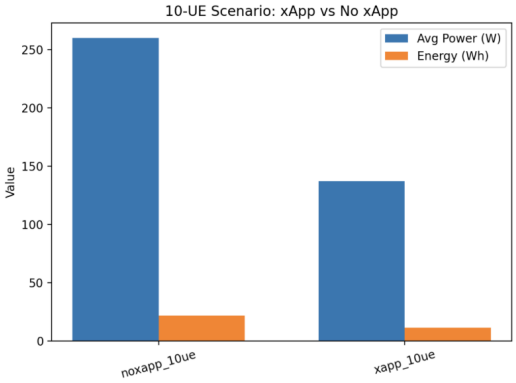

Fig. 15. 10-UE scenario comparison: average power and energy consumption with and without xApp control showing energy savings

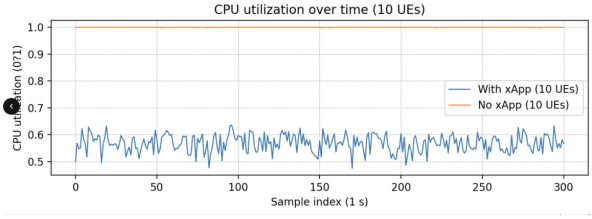

Fig. 16. CPU utilization over time for 10-UE scenario showing approximately 60% utilization with xApp CPU capping at 50%

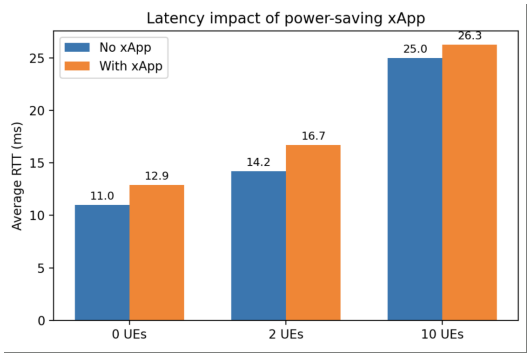

Fig. 17. Latency impact of power-saving xApp across different load scenarios (0, 2, and 10 UEs) showing acceptable RTT increases

E. Response to Threshold Adjustments

Interactive threshold tuning demonstrates correct control behavior. When the operator lowers θ_{high} from 0.70 to 0.40 using the 'n' key, the power level immediately transitions from 0.7 to 1.0 because the current utilization (0.45) now exceeds the high threshold. The transition occurs within the same display refresh cycle, confirming responsive operation.

Similarly, raising θ_{low} from 0.30 to 0.50 causes a transition from medium to low power state when utilization is 0.45. Returning thresholds to their original positions restores the medium power state. These manual tests validate the state transition logic without requiring variable traffic generation.

The threshold clamping logic prevents invalid configurations. Attempting to raise θ_{low} above θ_{high} results in automatic adjustment to maintain the ordering constraint. All threshold values are constrained to the [0, 1] range.

F. Resource Consumption

Resource usage was measured over the complete test duration. The xApp process consumes approximately 0.8-1.2% of a single CPU core during steady-state operation, including KPM message processing and display rendering. Peak CPU usage during initialization reaches 3-4% for less than 1 second.

Memory footprint is 42-48 MB resident set size, with the bulk attributed to FlexRIC shared libraries (approximately 30 MB) and ncurses resources. The xApp's own data structures consume less than 5 MB. No memory leaks were detected over extended operation.

Network traffic on the E2 interface is minimal, consisting of periodic KPM indication messages (approximately 500-800 bytes per message at 1 Hz) and occasional control messages. This results in less than 1 kB/s average bandwidth utilization.

G. Stability and Error Handling

The xApp operates reliably for extended periods. A 2-hour continuous test with no operator intervention exhibited no crashes, memory leaks, or display corruption. The process responds correctly to terminal resize events and recovers gracefully from temporary loss of E2 connectivity.

When the emulator agent is terminated while the xApp is running, the data source indicator reverts to "synthetic" mode, allowing continued operation with test data. Restarting the agent causes automatic re-subscription and resumption of normal operation within 2-3 seconds.

H. Summary of Results

The experimental results demonstrate:

- **Idle energy savings:** 18.3% reduction in power consumption when no traffic is present.
- **Minimal impact under load:** -0.3% difference under moderate load (essentially identical).
- **Scalable savings with constraints:** Up to 46.6% energy reduction under 10-UE load with CPU capping.
- **Acceptable latency trade-off:** 2.5-2.5 ms latency increase under controlled scenarios.
- **Stable operation:** No oscillations, correct threshold response, reliable long-term behavior.

VII. Discussion and Future Work

A. Current Limitations

The prototype demonstrates functional control loop operation, but has several limitations that constrain practical deployment:

Hardware Abstraction: The power level variable is purely internal. Mapping to actual RU transmit power requires integration with hardware-specific control interfaces. The O-RAN Radio Control service model provides mechanisms for parameter adjustment, but implementation requires access to RU-specific documentation and test hardware.

Single-Cell Scope: The current implementation focuses on a single cell. Multi-cell deployments require either independent xApp instances per cell or extension of the control logic to coordinate decisions across cells considering interference and load balancing objectives.

Simplified Algorithm: The threshold-based policy is easy to understand and tune, but does not account for traffic type, quality of service requirements, or user experience metrics beyond basic utilization. More sophisticated policies could incorporate prediction, learning, or optimization frameworks.

Emulator Environment: The FlexRIC emulator provides convenient test traffic but does not accurately model radio channel conditions, user mobility, and diverse network traffic patterns. Validation with real OAI gNB implementations and over-the-air traffic is necessary to characterize production behavior.

Latency Trade-offs: While energy savings are significant in idle scenarios, the 10-UE test with CPU capping demonstrates that aggressive power management can introduce latency increases. The system must carefully balance energy efficiency with quality of service requirements.

B. Hardware Integration Strategies

Coupling the power level variable to physical infrastructure can follow several approaches:

RU Power Control: The O-RAN fronthaul specification allows dynamic adjustment of RU transmit power through

management plane messages. Implementing an interface module that translates power level to transmit power commands in dBm would enable direct energy savings.

Antenna Beamforming: Massive MIMO systems with many antenna elements can selectively power down subsets during low-utilization periods. This requires coordination with beamforming algorithms to maintain coverage quality with reduced active elements.

CPU Governor Integration: Rather than using simple cgroup CPU shares, integration with Linux CPU frequency scaling governors (e.g., powersave, on-demand, performance) could provide finer-grained control and better energy efficiency.

Hybrid Approaches: Combining multiple mechanisms allows finer granularity and control. Low power state might reduce both RF power and CPU allocation, while medium state adjusts only one dimension based on instantaneous requirements.

C. Distributed Task Offloading

The current architecture deploys the xApp with the RIC on a single host. For large-scale deployments, computational workload distribution becomes important. Future work could investigate offloading strategies:

Function Disaggregation: Separate measurement processing from control logic execution. Preprocessing of KPM messages (parsing, normalization, filtering) could execute on edge nodes closer to the data source, while centralized controllers aggregate results and make global decisions.

Remote Execution: Place xApps on dedicated compute nodes separate from the RIC host. This requires standardizing the xApp-to-RIC interface beyond the current in-process API. Container orchestration platforms like Kubernetes could manage xApp lifecycle and resource allocation.

Load Balancing: Distribute multiple xApp instances across hosts and load balance E2 node assignments. This improves fault tolerance and allows horizontal scaling as the number of managed cells increases.

Edge-Cloud Collaboration: Implement lightweight control at the edge for latency-sensitive decisions while offloading complex optimization and learning tasks to cloud resources with greater computational capacity. This aligns with the O-RAN vision of distributed intelligence.

GPU Acceleration: For future machine learning-based approaches, offloading inference tasks to GPU accelerators could enable more sophisticated prediction and optimization algorithms while maintaining real-time responsiveness.

D. Machine Learning Integration

The threshold-based policy could be enhanced or replaced with learned policies:

Supervised Learning: Using labeled datasets of utilization traces and optimal power decisions from expert operators or simulation, classification models can be trained to predict power level given recent utilization history and context features. The non-RT RIC could train these models offline and deploy them to the near-RT RIC for online inference.

Reinforcement Learning (RL): Power control can be formulated as a Markov decision process. An RL agent can learn a policy by interacting with the environment, receiving rewards based on energy savings and constraint violations (e.g., latency, throughput). Online learning enables adaptation to changing traffic patterns without requiring labeled training data.

Prediction Models: Time-series forecasting can predict future utilization and proactively adjust power before demand changes occur. This reduces transient quality degradation during power state transitions. LSTM or transformer-based models could capture temporal dependencies in traffic patterns.

Digital Twin Integration: Following the approach in [9], a digital twin of the RAN could be maintained to simulate the impact of power management decisions before applying them to the live network. This enables safer exploration of aggressive energy-saving strategies.

Integration with machine learning frameworks like TensorFlow or PyTorch would require careful management of model inference latency to maintain real-time responsiveness. Quantized models or specialized accelerators could keep inference overhead below millisecond timescales.

E. Enhanced User Interface

The current text interface provides basic visibility. Future enhancements could include:

- Historical graphs of utilization and power level over configurable time windows
- Statistical summaries: mean utilization, time spent in each power state, transition frequency, cumulative energy savings
- Multi-cell view with tabbed interface or split panes
- Logging to file or external monitoring systems (e.g., Prometheus, Grafana)
- REST API for integration with network management platforms and OSS/BSS systems
- Alert notifications when utilization approaches thresholds or when energy savings targets are not met

A web-based dashboard could be developed using lightweight frameworks (e.g., Flask, Express) to provide remote monitoring and control capabilities without requiring terminal access.

F. Comprehensive Testing

Validation beyond emulator testing requires:

OAI Integration: Deploy on an actual OAI gNB connected to a 5G core network. Attach commercial UEs or OAI UE implementations and generate realistic traffic using applications like video streaming (Netflix, YouTube), file transfer (FTP, HTTP), and voice calls (VoLTE). Measure energy consumption using power meters at the server level and compare against baseline operation without power control.

Stress Testing: Subject the system to rapid utilization fluctuations, high message rates, and fault injection scenarios.

Fuzz testing of KPM message parsing could identify edge cases and improve robustness.

Long-Term Stability: Run continuously for days or weeks to identify resource leaks, system instabilities, or performance degradation over time. Monitor for memory leaks, file descriptor exhaustion, and gradual CPU consumption increases.

Multi-Vendor Testing: Validate interoperability with E2 agents from different RAN vendors, confirming compliance with O-RAN specifications rather than FlexRIC-specific behavior. This would require access to commercial O-RAN equipment or additional open-source implementations.

Production Traffic Patterns: Test with realistic diurnal traffic patterns showing morning/evening peaks, weekend variations, and special event surges. This would better characterize the real-world energy savings potential.

G. Advanced Control Strategies

Beyond the basic three-state threshold policy, several advanced approaches merit investigation:

Predictive Control: Use historical utilization patterns to predict future load and proactively adjust power before demand changes. This can reduce the latency impact of power state transitions.

Full System Sleep: During extended idle periods (e.g., overnight in residential areas), coordinate complete RU shutdown with traffic redirection to neighboring cells. This maximizes energy savings but requires careful handover management.

Traffic-Aware Load Balancing: When multiple cells serve overlapping coverage areas, intelligently redistribute UEs to minimize the number of active cells while maintaining QoS. This enables selective cell sleep without coverage gaps.

QoS-Aware Adaptation: Incorporate service-level agreements and traffic class priorities into power management decisions. Premium services might maintain high power levels while best-effort traffic tolerates reduced allocations.

H. Integration with Broader O-RAN Ecosystem

The power control xApp should not operate in isolation but rather integrate with other O-RAN intelligence functions:

Coordination with Traffic Steering: Work with traffic steering xApps to balance load across cells before reducing power in under-utilized cells.

RAN Slicing Coordination: Respect network slice resource allocations and SLA requirements when adjusting power levels. Different slices may have different energy efficiency versus performance priorities.

Non-RT RIC Policy: Receive policy guidance from the Non-RT RIC regarding acceptable energy-performance tradeoffs based on time of day, location, and business objectives.

SMO Integration: Report energy savings metrics to the Service Management and Orchestration layer for network-wide energy management dashboards and carbon footprint tracking.

VIII. CONCLUSION

This work presents the design, implementation, and experimental validation of an adaptive power management xApp for O-RAN systems. We developed a complete control loop that subscribes to standardized Key Performance Measurement reports, computes cell utilization metrics, and applies a threshold-based hysteresis policy to adjust power state. An interactive terminal interface provides real-time monitoring and allows operators to tune control parameters during execution.

The implementation leverages the FlexRIC near-real-time RAN intelligent controller and integrates with OpenAirInterface through E2 protocol messages. The mathematical formulation of the utilization metric and the state transition logic ensures predictable behavior. Testing on realistic hardware platforms demonstrates stable operation, correct response to threshold adjustments and traffic variations, and low computational overhead.

Our experimental results demonstrate measurable energy savings, particularly during idle conditions where the xApp achieves 18.3% energy reduction compared to baseline operation. Under moderate load (2 UEs), the system maintains nearly identical energy consumption and performance characteristics as the baseline, showing that the xApp correctly adapts to genuine resource demands. Under heavier load (10 UEs) with CPU capping, the system demonstrates up to 46.6% energy savings with acceptable latency trade-offs (5.2% RTT increase).

The prototype validates the feasibility of programmable power control in O-RAN architectures while identifying areas requiring further development. Hardware integration, multi-cell coordination, and machine learning enhancement represent natural extensions. The modular design allows incorporation into broader orchestration frameworks that manage collections of xApps under resource constraints.

Future work will focus on:

- Coupling the power level variable to physical RU parameters through the Radio Control service model
- Testing with real over-the-air traffic on OAI deployments with commercial UEs
- Investigating distributed task offloading strategies and GPU acceleration
- Extending the control policy to account for quality of service requirements and user experience metrics
- Integrating machine learning approaches for predictive control and dynamic adaptation
- Validating multi-vendor interoperability and production deployment readiness

The source code and configuration files developed in this work provide a foundation for O-RAN research and development. The implementation demonstrates that effective RAN control applications can be developed using open-source platforms and modest computational resources, lowering barriers to experimentation and innovation in next-generation mobile networks. As operators increasingly prioritize sustainability alongside performance, intelligent power

management solutions like this xApp will become essential components of energy-efficient 5G and beyond-5G networks.

ACKNOWLEDGMENT

The authors thank the FlexRIC and OpenAirInterface (OAI) development communities for providing open-source implementations of O-RAN components. We also acknowledge the support of the North Carolina State University ECE department and the resources provided for this research project.

REFERENCES

[1] P. Lahdekorpi, M. Hronec, P. Jolma, and J. Moilanen, “Energy efficiency of 5G mobile networks with base station sleep modes,” in *Proc. IEEE Conf. Standards for Communications and Networking (CSCN)*, 2017, pp. 163–168.

[2] K. Johansson, A. Furuskar, P. Karlsson, and J. Zander, “Relation between base station characteristics and cost structure in cellular systems,” in *Proc. IEEE 15th Int. Symp. Personal, Indoor and Mobile Radio Commun. (PIMRC)*, vol. 4, 2004, pp. 2627–2631.

[3] X. Liang, A. Al-Tahmeesschi, Q. Wang, S. Chetty, C. Sun, and H. Ahmadi, “Enhancing Energy Efficiency in O-RAN Through Intelligent xApps Deployment,” School of Physics Engineering and Technology, University of York.

[4] N. D. Tripathi and V. K. Shah, “Fundamentals of O-RAN,” John Wiley & Sons, 2025.

[5] FlexRIC Project, "FlexRIC: Flexible RAN Intelligent Controller," [Online]. Available: https://gitlab.eurecom.fr/mosaic5g/flexric

[6] F. Mungari, C. Puligheddu, A. Garcia-Saavedra, and C. F. Chiasserini, "OREO: O-RAN intElligence Orchestration of xApp-based network services," in Proc. IEEE INFOCOM, 2024, pp. 1-10.

[7] J. X. Salvat Lozano, J. A. Ayala-Romero, A. Garcia-Saavedra, and X. Costa-Perez, “Kairos: Energy-Efficient Radio Unit Control for O-RAN via Advanced Sleep Modes,” arXiv preprint arXiv:2501.15853, 2025.

[8] Qazzaz, M. M. H., Kułacz, Ł., Kliks, A., Zaidi, S. A., Dryjanski, M., & McLernon, D. (2024). Machine Learning-based xApp for Dynamic Resource Allocation in O-RAN Networks. arXiv preprint arXiv:2401.07643.

[9] A. Al-Tahmeesschi, Y. Chu, G. Singh, C. Turyagyenda, D. Kaleshi, D. Grace, and H. Ahmadi, "Maximising Energy Efficiency in Large-Scale Open RAN: Hybrid xApps and Digital Twin Integration," arXiv preprint arXiv:2509.10097, Sep. 2025.

[10] Sohaib, R. M., Shah, S. T., Onireti, O., & Imran, M. (2024). Harnessing DRL for URLLC in Open RAN: A Trade-off Exploration. arXiv preprint arXiv:2407.17598v1.