UNIVERSIDAD DE GRANADA

TRABAJO FIN DE MÁSTER

INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Desarrollo de una xAPP dentro del Near-RT RIC de una red 5G Open RAN

Autora

Alejandra Oliver Boada

Directores

Jorge Navarro Ortiz, Félix Delgado Ferro

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Junio de 2025

Desarrollo de una xAPP dentro del Near-RT RIC de una red 5G Open RAN

Autora
Alejandra Oliver Boada

Directores
Jorge Navarro Ortiz, Félix Delgado Ferro

Desarrollo de una xAPP dentro del Near-RT RIC de una red 5G Open RAN

Alejandra Oliver Boada

Palabras clave: 4G, 5G, RAN, O-RAN, RIC, xApp, interfaz E2, redes móviles, monitorización radio, control radio, gestión de recursos.

Resumen

Actualmente, se tiene una situación tecnológica global caracterizada por el crecimiento exponencial en el uso de dispositivos móviles y el aumento constante de los servicios digitales. Para poder hacer frente a esta demanda, no solo en términos de capacidad, sino también de flexibilidad y eficiencia en el uso de recursos, las redes han tenido que evolucionar hacia arquitecturas más abiertas, inteligentes y adaptables.

En esta línea, surge *Open Radio Access Network* (O-RAN), que nace como una evolución del modelo tradicional de *Radio Access Network* (RAN), este último caracterizado por ser cerrado y dependiente de soluciones propietarias. Esta iniciativa tiene como objetivo promover la interoperabilidad entre equipos de diferentes fabricantes y permitir una mayor capacidad de gestión dinámica y programable de los recursos en la red de acceso. Está impulsada por la *O-RAN Alliance*, un consorcio formado por operadores de red, fabricantes y otros actores clave del sector de las telecomunicaciones, que colaboran en el desarrollo de especificaciones abiertas y en la definición de una arquitectura más flexible, inteligente y centrada en el software.

Una de las piezas clave de esta nueva arquitectura es el *Radio Access Network Intelligent Controller* (RIC), un componente lógico que forma parte del plano de control de la red de acceso y que introduce una capa adicional de inteligencia en el sistema. Este se encarga de tomar decisiones sobre la red, gracias a la ejecución de aplicaciones conocidas como *xApps / rApps* (dependiendo del tipo de RIC).

En este proyecto se analiza la transición de las redes RAN tradicionales hacia las arquitecturas O-RAN en el contexto de las redes móviles 5G, centrando la atención en el papel del RIC dentro de este marco. En concreto, se despliegan dos escenarios con dos plataformas distintas, en los que se realizan diversas configuraciones y experimentaciones con distintas implementaciones del RIC, de forma que se puedan demostrar los beneficios que trae consigo para las redes 5G/6G modernas.

Development of an xAPP within the Near-RT RIC of a network 5G Open RAN

Alejandra Oliver Boada

Keywords: 4G, 5G, RAN, O-RAN, RIC, xApp, E2 interface, mobile networks, radio monitoring, radio control, resource management.

Abstract

Currently, the global technological landscape is marked by the exponential growth in the use of mobile devices and the constant increase in digital services. To meet this growing demand, not only in terms of capacity but also in flexibility and resource efficiency, networks have had to evolve toward more open, intelligent, and adaptable architectures.

In this context, O-RAN emerges as an evolution of the traditional RAN model, the latter being characterized by closed architectures and proprietary solutions. The main goal of this initiative is to promote interoperability between equipment from different vendors and enable greater dynamic and programmable management of access network resources. It is driven by the *O-RAN Alliance*, a consortium formed by network operators, vendors, and other key players in the telecommunications sector, who collaborate in the development of open specifications and the definition of a more flexible, software-centric, and intelligent architecture.

One of the key components of this new architecture is the RIC, a logical element that belongs to the control plane of the access network and introduces an additional layer of intelligence into the system. It is responsible for making decisions about the network by running applications known as *xApps / rApps* (depends on the type of RIC).

This project analyzes the transition from traditional RAN networks to O-RAN architectures in the context of 5G mobile networks, focusing on the role of the RIC within this framework. Specifically, two scenarios are deployed with two different platforms, in which various configurations and experiments are carried out with different implementations of the RIC, so as to demonstrate the benefits it brings to modern 5G/6G networks.

Yo, **Alejandra Oliver Boada**, alumna del Máster en Ingeniería de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77552695F, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Alejandra Oliver Boada

Granada a 30 de junio de 2025.

D. **Jorge Navarro Ortiz**, Profesor Titular de Universidad del Área de Ingeniería Telemática del Departamento Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada y

D. **Félix Delgado Ferro**, estudiante de doctorado en el Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada

Informan:

Que el presente trabajo, titulado ***Desarrollo de una xAPP dentro del Near-RT RIC de una red 5G Open RAN***, ha sido realizado bajo su supervisión por **Alejandra Oliver Boada**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 30 de junio de 2025.

Los directores:

Jorge Navarro Ortiz **Félix Delgado Ferro**

Agradecimientos

Estos dos años de Máster han sido duros al mismo tiempo que satisfactorios. He vivido muy buenas experiencias con mis compañeros y he aprendido a disfrutar cada momento del aprendizaje. Por ello, quiero agradecer y dedicar este proyecto:

A mis tutores, Jorge y Félix, por todo el apoyo recibido a lo largo del camino, y por darme consejos y ayudarme en todo lo posible. Habéis hecho que el TFG y el TFM sean un proceso de aprendizaje.

A mi familia, por apoyarme en mis decisiones siempre, aguantar los malos momentos y celebrar mis logros. Y a mi perro, por acompañarme en cada desvelo.

A mis amigas y a Alejandro, por hacer de estos últimos seis años una etapa inolvidable, llena de apoyo, consejos y compañía.

Y, en especial, a mi abuelo, porque aunque este año no le dejó verme llegar hasta aquí, siempre estuvo orgulloso y creyó en mi.

Gracias.

Índice general

1. Introducción **23**
1.1. Contexto y Motivación 23
1.2. Objetivos y Alcance del Proyecto 26
1.2.1. Objetivos 26
1.2.2. Alcance del Proyecto 26
1.3. Metodología 27
1.4. Estructura de la Memoria 28

2. Estado del Arte **33**
2.1. Proyectos y Aplicaciones Reales 33
2.2. Implementaciones Disponibles 38

3. Fundamentos Teóricos **43**
3.1. Fundamentos de las redes 5G 43
3.1.1. Escenarios 43
3.1.2. Requisitos técnicos 44
3.2. Arquitectura 5G 45
3.2.1. *Core* 5G 46
3.2.2. Radio 5G 48
3.2.3. Pila de protocolos 50
3.3. Evolución de las redes 5G 55
3.3.1. O-RAN 56
3.3.2. Inteligencia en O-RAN 5G: RIC 59
3.3.3. Interfaz E2: Enlace entre RIC y O-RAN 64

4. Planificación y Costes **71**
4.1. Planificación Temporal 71
4.2. Recursos y Coste 75
4.2.1. Recursos Humanos 75
4.2.2. Recursos Hardware 76
4.2.3. Recursos Software 77
4.2.4. Estimación de Costes 79
4.2.5. Modificaciones en la Planificación Inicial 79

5. Diseño e Implementación **81**
5.1. Contexto Práctico 81
5.2. Despliegue de la arquitectura funcional de *Open Air Interface Cellular* (OAIC) 82
5.2.1. Instalación de la *Virtual Machine* (VM) y dependencias 82
5.2.2. Implementación de la red 84
5.3. Implementación del escenario *Software Radio Systems Radio Access Network* (srsRAN) 93
5.3.1. Instalación de la VM y dependencias 93
5.3.2. Despliegue de la red 95

6. Pruebas y Experimentación **107**
6.1. Entorno de Pruebas 107
6.1.1. Observaciones técnicas 108
6.2. OAIC. Prueba de Funcionalidad 109
6.3. srsRAN. Monitorización y Control 113
6.3.1. Consideraciones 114
6.3.2. Monitorización de Métricas 115
6.3.3. Control de Recursos Radioeléctricos 123
6.3.4. Comparativa 135
6.4. Escenario real 136
6.5. Estudio del procedimiento E2 en Wireshark 140
6.6. Complicaciones y desafíos encontrados 145

7. Conclusiones y Líneas Futuras **147**
7.1. Conclusiones 147
7.2. Líneas Futuras 148

A. Fichero de configuración de la *Next Generation Node B* (gNB) en OAIC **159**

B. Opciones de configuración de la gNB en OAIC **163**
B.1. Procesamiento, configuración *User Equipment* (UE)/gNB, estimación de canal 163
B.2. Opciones de *logging* y servidores 165
B.3. Modo PHYTest 165
B.4. Parámetros Adicionales 166

C. Opciones de configuración del UE en OAIC **167**
C.1. Procesamiento, configuración UE/gNB, ganancias, estimación de canal 167
C.2. Modo de ejecución, simulación, radiofrecuencia, interfaces, *Non-Standalone* (NSA) 169
C.3. *Logging* y servidores embebidos 170

C.4. Trazado y debug visual 170

D. Fichero de configuración de la gNB en srsRAN 171

E. Fichero de configuración del UE en srsRAN 175

F. *xApps* para monitorización de métricas 179
F.1. *Flexible Radio Access Network Intelligent Controller* (FlexRIC) en OAIC 179
F.2. *Open Radio Access Network Software Community Radio Access Network Intelligent Controllerr* (ORAN-SC-RIC) en srsRAN 187
F.3. FlexRIC en srsRAN 190

G. *xApps* para control de *Physical Resource Block* (PRB)s 193

Índice de figuras

1.1. Tres pilares del 5G [4]. 24

2.1. Arquitectura de OpenCare5G [3]. 34
2.2. Arquitectura *Advanced Wireless Research Platform for Digital Agriculture & Rural Environments* (ARA) O-RAN [15]. . 36
2.3. *Base Station* (BS) ARA O-RAN [15]. 36
2.4. UE ARA O-RAN [15]. 36
2.5. *Objective Modular Network Testbed in C++* (OMNeT++) [16]. 37
2.6. Solución para el problema del *handover* [16]. 37
2.7. SD-CORE [20]. 39
2.8. SD-RAN [20]. 39
2.9. Arquitectura ns-O-RAN [25]. 39
2.10. *Open Air Interface* (OAI) CN [26]. 40
2.11. Red radio OAI [28]. 41
2.12. *Open 5 Generation System* (Open5GS) *core* [36] 41
2.13. srsRAN radio [8] 42

3.1. Escenarios de uso del 5G [38]. 44
3.2. Requisistos técnicos del 5G [38]. 45
3.3. Arquitectura general 5G [40]. 45
3.4. Arquitectura 5G NSA [41]. 46
3.5. *Core* 5G *Standalone* (SA) [41]. 47
3.6. *Radio* 5G SA [41]. 49
3.7. Pila de protocolos en el plano de control 5G [41]. 50
3.8. Pila de protocolos en el plano de usuario 5G [41]. 50
3.9. Estructura de trama en 5G [42]. 53
3.10. Arquitectura RAN vs. O-RAN [43]. 56
3.11. Arquitectura lógica de O-RAN. 56
3.12. Arquitectura del *Non Real Time Radio Access Network Intelligent Controller* (Non-RT RIC) [7]. 60
3.13. Arquitectura del *Near Real Time Radio Access Network Intelligent Controller* (Near-RT RIC) [6]. 62
3.14. Pila de protocolos de la interfaz E2 [46]. 64

3.15. Establecimiento de sesion E2 [46]. 65
3.16. Suscripción de una *xApp* con E2 [6]. 67
3.17. Monitorización de métricas con E2 [46]. 69
3.18. Control de recursos con E2 [46]. 69

4.1. Resumen de la temporización de las fases y tareas del proyecto. 73
4.2. Diagrama de Gantt. 74

5.1. Escenario general de implementación. 81
5.2. Instalación de librerías generales. 83
5.3. Instalación de SWIG desde código fuente. 83
5.4. Instalación del compilador GCC. 83
5.5. Instalación de Docker y configuración desde su repositorio oficial. 84
5.6. Instalación del OAI 5G *Core Network* (CN). 85
5.7. Descarga y preparación de la red de acceso OAI 5G. 86
5.8. Descarga de FlexRIC en la rama *aymanfatich* [59]. 89
5.9. Compilación de FlexRIC. 89
5.10. Fichero de configuración del FlexRIC en la rama *aymanfatich* [59]. 90
5.11. *Ping* del UE al *User Plane Function* (UPF) del *core* y viceversa. 92
5.12. Arquitectura funcional con OAIC. 92
5.13. Instalación de las librerías básicas. 93
5.14. Compilación de srsRAN 5G. 94
5.15. Compilación de srsRAN 4G. 95
5.16. Descarga de FlexRIC en la rama 'br-flexric'. 96
5.17. Compilación de FlexRIC. 96
5.18. Fichero de configuración del FlexRIC en la rama 'br-flexric'. . 97
5.19. Clonación del repositorio de ORAN-SC-RIC. 97
5.20. Despliegue de ORAN-SC-RIC mediante Docker. 97
5.21. *Script ip_config.sh* para configurar el enrutamiento entre el UE y el *core*. 103
5.22. *Ping* del UE a la *Internet Protocol* (IP) virtual del *core* y viceversa. 104
5.23. Arquitectura funcional con srsRAN. 105

6.1. Relación PRBs-*Subcarrier Spacing* (SCS)-Ancho de banda [68].108
6.2. Experimentación OAIC con configuración inicial. 111
6.3. Experimentación OAIC con ancho de banda reducido. 111
6.4. Experimentación OAIC a mayor ganancia y atenuación. . . . 112
6.5. Experimentación OAIC en una banda superior (SCS mayor) y ancho de banda reducido. 113
6.6. Diferencia temporal entre el canal físico real y el canal simulado con *Zero Message Queue* (ZeroMQ). 115

6.7. Experimentación srsRAN - ORAN-SC-RIC de referencia. . . 117
6.8. Experimentación srsRAN - ORAN-SC-RIC de referencia (Grafana). 118
6.9. Experimentación srsRAN - ORAN-SC-RIC con ancho de banda reducido. 118
6.10. Experimentación srsRAN - ORAN-SC-RIC con ancho de banda reducido (Grafana). 119
6.11. Experimentación srsRAN - ORAN-SC-RIC con ancho de banda reducido y aumento de la ganancia del UE. 120
6.12. Experimentación srsRAN - ORAN-SC-RIC con ancho de banda reducido y aumento de la ganancia del UE (Grafana). . . . 120
6.13. Experimentación srsRAN - ORAN-SC-RIC en banda n2 con aumento del ancho de banda. 121
6.14. Experimentación srsRAN - ORAN-SC-RIC en banda n2 con aumento del ancho de banda (Grafana). 121
6.15. Experimentación srsRAN - FlexRIC de referencia. 122
6.16. Experimentación srsRAN - FlexRIC con BW reducido. 122
6.17. Experimentación srsRAN - FlexRIC con ancho de banda reducido y aumento de la ganancia del UE. 122
6.18. Experimentación srsRAN - FlexRIC en banda n2 con aumento del ancho de banda. 122
6.19. Esquema .grc para canal ascendente y descendente en GNU Radio. 125
6.20. Funcionamiento de *simple_xapp.py* (I). 127
6.21. Funcionamiento de *simple_xapp.py* (II). 128
6.22. Funcionamiento de *simple_xapp.py* con *pathloss* (I). 129
6.23. Funcionamiento de *simple_xapp.py* con *pathloss* (II). 129
6.24. Diagrama de flujo sobre el funcionamiento de la *xApp* implementada. 133
6.25. Condiciones iniciales en la *xApp*. 134
6.26. Funcionamiento de la *xApp*. 135
6.27. Montaje del escenario real. 137
6.28. Prueba inicial de conectividad en la red. 138
6.29. UE + RAN + CN + ORAN-SC-RIC. 138
6.30. Ejecución de *simple_xapp.py* en el escenario real. 139
6.31. Intercambio de mensajes de control entre la *xApp* y la gNB. . 140
6.32. Procedimiento E2 en Wireshark. 141
6.33. *E2 Setup Request* (I). 141
6.34. *E2 Setup Request* (II). 142
6.35. *E2 Setup Request* (III). 142
6.36. *E2 Suscription Request*. 143
6.37. *RIC Indication*. 144
6.38. *RIC Control Request*. 145
6.39. *RIC Control Acknowledge*. 145

A.1. Configuración del gNB y parámetros de red (I). 159
A.2. Configuración del gNB y parámetros de red (II). 160
A.3. Configuración del gNB y parámetros de red (III). 161
A.4. Configuración del gNB y parámetros de red (IV). 162

D.1. Fichero de configuración de la gNB en srsRAN (I). 171
D.2. Fichero de confguración de la gNB en srsRAN (II). 172
D.3. Fichero de confguración de la gNB en srsRAN (III). 173

E.1. Fichero de configuración del UE en srsRAN (I). 175
E.2. Fichero de configuración del UE en srsRAN (II). 176
E.3. Fichero de configuración del UE en srsRAN (III). 177

F.1. *xapp_kpm_moni.c* (I). 179
F.2. *xapp_kpm_moni.c* (II). 180
F.3. *xapp_kpm_moni.c* (III). 181
F.4. *xapp_kpm_moni.c* (IV). 182
F.5. *xapp_kpm_moni.c* (V). 183
F.6. *xapp_kpm_moni.c* (VI). 184
F.7. *xapp_kpm_moni.c* (VII). 185
F.8. *xapp_kpm_moni.c* (VIII). 186
F.9. *kpm_mon_xapp.py* (I). 187
F.10. *kpm_mon_xapp.py* (II). 188
F.11. *kpm_mon_xapp.py* (III). 189
F.12. *kpm_mon_xapp.py* (IV). 190
F.13. *xapp_mon_e2sm_kpm.conf* (I). 190
F.14. *xapp_mon_e2sm_kpm.conf* (II). 191

G.1. *simple_xapp.py xApp* (I). 193
G.2. *simple_xapp.py xApp* (II). 194
G.3. *simple_xapp.py xApp* (III). 195
G.4. *xApp* que adapta la asignación de PRBs en función del estado del canal (I). 196
G.5. *xApp* que adapta la asignación de PRBs en función del estado del canal (II). 197
G.6. *xApp* que adapta la asignación de PRBs en función del estado del canal (III). 198
G.7. *xApp* que adapta la asignación de PRBs en función del estado del canal (IV). 199

Índice de Tablas

3.1. Rango de frecuencias en 5G *New Radio* (NR). 52
3.2. Relación entre el espaciado de subportadoras (SCS) y el número de ranuras por subtrama en 5G NR. 53
3.3. Resumen de las características de los canales físicos en 5G. . 54
3.4. Interfaces clave en la arquitectura O-RAN 58

4.1. Tabla resumen de horas dedicadas a las diferentes tareas. . . 76
4.2. Coste total de recursos humanos. 76
4.3. Características del portátil utilizado. 77
4.4. Coste total de recursos hardware. 77
4.5. Coste Total de recursos software. 79
4.6. Coste total del proyecto. 79
4.7. Coste Total de los Equipos del Escenario Real. 80

5.1. Indicadores clave de rendimiento (KPI) en O-RAN 91
5.2. Tabla de rutas IP del *host* tras ejecutar `ip_config.sh`. 104

6.1. Resumen de comandos para la puesta en marcha del entorno con FlexRIC y OAIC. 109
6.2. Configuraciones empleadas en la validación funcional con OAIC. 110
6.3. Configuraciones empleadas en la validación de la monitorización de ORAN-SC-RIC con srsRAN. 116
6.4. Resumen de comandos para la puesta en marcha del entorno de monitorización con srsRAN. 116

Capítulo 1

Introducción

1.1. Contexto y Motivación

La quinta generación de redes móviles (5G) ha supuesto una revolución tecnológica muy importante, con el objetivo de ofrecer una gran variedad de servicios y aplicaciones, la mayoría de ellos con exigencias de latencia, velocidad, fiabilidad, seguridad y capacidad de adaptación. En el contexto de la redes 5G, se busca ofrecer mayor ancho de banda y cobertura, pero sobre todo, introducir un ecosistema flexible y modular pensado para casos de uso muy diversos, como la conducción autónoma [1], la automatización industrial [2], la medicina remota [3] o el mundo *Internet of Things* (IoT) [2].

Para responder a estas exigencias, las redes 5G se apoyan en tres ejes tecnológicos: *Enhanced Mobile Broadband* (eMBB) permite ofrecer velocidades de conexión significativamente superiores, orientadas al consumo intensivo de datos por parte de los usuarios. Por otro lado, *Ultra-Reliable & Low Latency Communications* (URLLC) está diseñado para garantizar una latencia mínima y una alta fiabilidad, características esenciales en aplicaciones críticas como la automatización industrial o los vehículos autónomos. Finalmente, *Massive Machine-Type Communications* (mMTC) proporciona la capacidad de conectar simultáneamente una enorme cantidad de dispositivos, lo que resulta clave en entornos IoT densamente poblados. La figura 1.1 muestra los principales beneficios de cada una de estas áreas.

Sin embargo, estas capacidades han requerido de una transformación de la arquitectura de red tradicional conocida. El modelo *Radio Access Network* (RAN), basado en estaciones base cerradas y rígidas, ha demostrado no ser suficiente en esta nueva era. Es por esto que surge el enfoque *Open Radio Access Network* (O-RAN), promovido por la *O-RAN Alliance*, que busca la desagregación funcional de los componentes de red, la apertura de interfaces

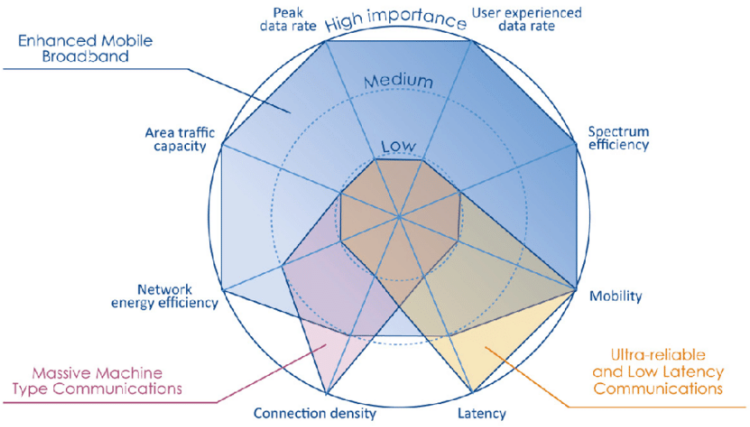


Figura 1.1: Tres pilares del 5G [4].

y la introducción de software interoperable entre múltiples fabricantes [5]. En términos generales, se pasa de una RAN tradicional donde el hardware y el software provienen de un único proveedor, a una arquitectura O-RAN en la que los bloques funcionales tradicionales (como la *Radio Unit* (RU) o la *BaseBand Unit* (BBU)) pueden separarse, virtualizarse y comunicarse a través de interfaces *open-source*. Esto permite a los operadores una mayor flexibilidad, evitando el bloqueo por parte de proveedores específicos (*vendor lock-in*) y abriendo la puerta a mejores innovaciones en la capa de control y gestión de la red.

A medida que las redes 5G se vuelven más *open-source* y se desagregan con el surgimiento del enfoque O-RAN, surgen nuevas necesidades en el ámbito de la monitorización y control de red, ya que las funciones que antes estaban embebidas en equipos propietarios ahora deben gestionarse mediante entidades externas. Estas permiten no solo configurar y operar la red, sino también adaptarse de forma dinámica a los cambios del entorno y a las necesidades del servicio.
Esta nueva condición se materializa en la figura del *Radio Access Network Intelligent Controller* (RIC), el cual puede ser de dos tipos ([6] y [7]) en función del marco temporal de operación:

- ***Non Real Time Radio Access Network Intelligent Controller* (Non-RT RIC)**: actúa sobre escalas de tiempo mayores a 1 segundo y se encarga de la planificación estratégica, la optimización a largo plazo y la coordinación de políticas.
- ***Near Real Time Radio Access Network Intelligent Controller***

(Near-RT RIC): gestiona decisiones operativas en tiempo casi real (entre 10 ms y 1 segundo), como la asignación dinámica de recursos radioeléctricos, la movilidad de usuarios, la gestión de interferencias y la adaptación de la calidad de servicio. Este tipo de RIC es el que será investigado a lo largo de este proyecto.

Estos controladores se apoyan en xApps (para el Near-RT RIC) y rApps (para el Non-RT RIC), aplicaciones que pueden ser desarrolladas por terceros y desplegadas sobre los Near-RT RICs para dotar de un control inteligente y adaptable al sistema, además de permitir a los operadores modular la red sin modificar el hardware subyacente.

Por otro lado, el núcleo de red o *core* también requiere mecanismos avanzados de control. En este ámbito, destacan herramientas como:

- ***Network Exposure Function* (NEF)**: expone las funciones del core 5G a servicios externos y facilita la integración con aplicaciones de terceros.
- ***Service Communication Proxy* (SCP)**: actúa como intermediario inteligente en las comunicaciones entre funciones del núcleo.
- ***Network Repository Function* (NRF)**: se encarga de mantener el catálogo de funciones disponibles y facilitar su descubrimiento y coordinación.

Como se ha podido intuir en el transcurso de esta sección, la propuesta 5G O-RAN - RIC trae consigo muchos beneficios, entre los que destacan:

- **Interoperabilidad**: posibilidad de integrar componentes de distintos fabricantes gracias a interfaces abiertas y estandarizadas.
- **Reducción del *vendor lock-in***: se elimina la dependencia de un único proveedor, facilitando la competencia y la diversidad de soluciones.
- **Virtualización de funciones**: permite desplegar funciones de red como software en plataformas genéricas, reduciendo costes de hardware y aumentando la flexibilidad.
- **Despliegue ágil de servicios**: facilita la incorporación rápida de nuevas funcionalidades de control de recursos y monitorización mediante las xApps o rApps del Near-RT RIC sin necesidad de modificar el hardware, aportando eficiencia en el uso de recursos y mejorando la experiencia del usuario.
- **Innovación abierta**: fomenta el desarrollo de soluciones por parte de terceros, lo que acelera la evolución tecnológica y la personalización de la red.

Para finalizar esta sección, se destaca que la investigación en este campo está en total desarrollo, con proyectos como *Software Radio Systems Radio Access Network* (srsRAN) [8] o *Open Air Interface Cellular* (OAIC) [9], que siguen avanzando en la implementación y optimización de este ecosistema. Además, existen líneas paralelas de investigación donde se busca introducir la *Artificial Intelligence* (AI) y el *Machine Learning* (ML) en el Near-RT RIC [10], con el objetivo de mejorar la toma de decisiones en tiempo real y la optimización dinámica de la red.

1.2. Objetivos y Alcance del Proyecto

1.2.1. Objetivos

Los principales objetivos a perseguir con la realización de este proyecto incluyen:

1. Investigación del cambio de las redes RAN convencionales hacia las arquitecturas O-RAN en el marco de la evolución de las redes móviles 5G.

2. Análisis del papel que juega el RIC en la arquitectura O-RAN, con un enfoque en los dos tipos de RIC y cómo interactúan con el nodo *Next Generation Node B* (gNB) a través de la interfaz E2.

3. Evaluación de las soluciones disponibles en el mercado para la implementación de la arquitectura O-RAN, así como las plataformas que facilitan el desarrollo del RIC, en concreto del Near-RT RIC, junto con diversas *xApps*.

4. Despliegue de una red O-RAN junto con un controlador de tipo Near-RT RIC, realizando experimentos con varias *xApps* sobre un *User Equipment* (UE) para validar su rendimiento y efectividad en entornos simulados.

5. Diseño de una *xApp* propia que permita explorar en la gestión y optimización de los recursos radioeléctricos dentro del sistema simulado.

1.2.2. Alcance del Proyecto

El alcance de este proyecto abarca las siguientes labores:

- Estudio de las redes RAN y O-RAN en el contexto de las comunicaciones 5G, con especial atención a sus componentes principales, como el RIC, y a su integración en la arquitectura de red a través de la interfaz E2. Asimismo, se realiza un análisis en profundidad del Near-RT RIC y de las distintas implementaciones disponibles actualmente.

- Revisión, evaluación y presentación de casos de uso y proyectos disponibles en el mercado que implementan soluciones O-RAN + RIC, destacando su carácter abierto, eficiente y autogestionado.

- Diseño e implementación de dos entornos simulados para el despliegue de una arquitectura O-RAN junto con un controlador de tipo Near-RT RIC (todas sus implementaciones), incluyendo pruebas con diversas *xApps* sobre un UE. Con ambos entornos se pretenden evaluar los dos proyectos más influyentes de los que han sido investigados.

- Verificación del correcto funcionamiento del sistema mediante el desarrollo adicional de una *xApp* en una de las implementaciones del Near-RT RIC, dentro de uno de los entornos simulados utilizados en el proyecto. Esta aplicación se orienta a tareas de optimización y gestión de los recursos radioeléctricos de la red.

- Despliegue de la red en un escenario real donde se comprueba la correcta monitorización y control de los recursos de red por medio del Near-RT RIC.

1.3. Metodología

En esta sección se describen las distintas actividades y procedimientos llevados a cabo a lo largo del desarrollo del proyecto, organizados de forma cronológica y orientados al cumplimiento de los objetivos planteados.

La metodología adoptada en este trabajo es de tipo cascada. Esta estrategia permite estructurar el desarrollo de forma secuencial (análisis, diseño, implementación, experimentación y evaluación), de forma que cada etapa se cierra antes de iniciar la siguiente. Esto facilita controlar si se van cumpliendo los objetivos establecidos.

Inicialmente, se introduce el contexto del trabajo, centrado en la arquitectura abierta y desagregada propuesta por la *O-RAN Alliance*. Con ello, se pretende situar al lector en el marco conceptual en el que se desarrolla este estudio. A continuación, se definen los objetivos perseguidos, que guiarán el desarrollo del proyecto y servirán como referencia para evaluar los resultados obtenidos. Seguidamente, se realiza una revisión del estado del arte, presentando las soluciones más relevantes y recientes que han adoptado el enfoque propuesto por O-RAN.

Tras esto, se exponen los fundamentos teóricos necesarios para la comprensión de esta arquitectura, del RIC (en concreto, del Near-RT RIC), la interfaz E2 y las *xApps*. Esta parte teórica es fundamental para garantizar

una posterior implementación sólida y coherente. Además, se incluye una planificación temporal del proyecto, así como una estimación preliminar de los recursos y costes asociados a su ejecución.

Una vez consolidados los conocimientos técnicos necesarios y definidos los objetivos, se plantea el escenario a implementar y se seleccionan, de entre las opciones investigadas, las dos soluciones más prometedoras en función de su relevancia, soporte comunitario y grado de madurez. A continuación, dicho escenario se implementa en ambas soluciones como se muestra en el capitulo 5.

Posteriormente, se realiza un análisis conceptual de ambas soluciones, basado en las observaciones obtenidas durante la fase de implementación. A la alternativa no seleccionada se le aplica una prueba de concepto orientada a validar su funcionamiento básico. Por su parte, sobre la opción más prometedora se llevan a cabo pruebas más exhaustivas, se incorpora una funcionalidad adicional desarrollando una nueva *xApp* para control de *Physical Resource Block*s (PRBs) y se procede al despliegue de una *xApp* de control en un entorno real, más allá del entorno simulado inicial, con el fin de evaluar su comportamiento en condiciones más cercanas a un escenario operativo.

Como broche final, se analizan en detalle los resultados obtenidos, evaluando aspectos como la interoperabilidad, el rendimiento y la adaptabilidad del sistema. A partir de este análisis, se extraen conclusiones que permiten valorar el grado de éxito alcanzado y se proponen futuras líneas de trabajo que permitan extender la solución planteada.

1.4. Estructura de la Memoria

En esta sección se detallan las partes de las que consta la presente memoria, describiendo de forma general lo realizado en cada capítulo.

- **Capítulo 1. Introducción:** consta de los siguientes apartados:
  - **Contexto y Motivación:** se presenta una breve introducción del contexto tecnológico actual y los motivos que han impulsado la elaboración de esta investigación.
  - **Objetivos y Alcance del Proyecto:** se exponen las metas principales que se persiguen con el trabajo, así como los resultados obtenidos hasta su finalización.
  - **Metodología:** se ofrece una visión general del enfoque de trabajo adoptado, explicando de forma resumida las distintas etapas seguidas durante el desarrollo del proyecto.

  - **Estructura de la Memoria:** se presenta un resumen del contenido que conforma cada uno de los capítulos que integran este documento.

- **Capítulo 2. Estado del Arte:** en este capítulo se detalla:

  - **Casos de Uso:** se describen los principales escenarios de aplicación, incluyendo ejemplos representativos de cada uno.
  - **Proyectos Relevantes:** se analizan iniciativas en curso o finalizadas recientemente que implementan la infraestructura 5G O-RAN más RIC. De entre todas, se eligen las dos más prometedoras.

- **Capítulo 3. Fundamentos Teóricos:** se abordan los conceptos clave necesarios para comprender la arquitectura y el funcionamiento de las redes 5G, con especial énfasis en su evolución hacia entornos abiertos como O-RAN. Este capítulo se divide en 3 partes:

  - **Fundamentos de las redes 5G:** se presentan los principios básicos de las redes móviles de quinta generación, incluyendo sus objetivos, características clave y casos de uso representativos.
  - **Arquitectura 5G:** se describe en detalle la arquitectura técnica de las redes 5G, incluyendo los componentes del núcleo de red (*5G Core*), la red de acceso radioeléctrico (RAN), y aspectos relevantes de la capa física y la interfaz aérea.
  - **Evolución hacia O-RAN:** se analiza la transición hacia arquitecturas abiertas como O-RAN, explicando sus diferencias con las redes 5G tradicionales, y profundizando en elementos clave como los controladores RICs (Near-RT RIC y Non-RT RIC) y el protocolo E2, los cuales habilitan una mayor inteligencia, interoperabilidad y control en la red.

- **Capítulo 4. Planificación y Costes:** se detallan tres secciones principales:

  - **Planificación Temporal:** se presenta una descripción del cronograma de tareas realizadas durante el proyecto.
    - *Diagrama de Gantt:* se incluye un diagrama en el que se muestra el tiempo dedicado a cada tarea o fase en relación con el tiempo total disponible para el proyecto.
  - **Recursos:** se especifican los recursos necesarios:
    - *Recursos Humanos:* se enumeran los recursos humanos implicados, junto con su presupuesto asignado.
    - *Recursos Hardware:* se detallan los recursos hardware utilizados y su presupuesto.

        - *Recursos Software:* se describen las herramientas y software utilizados, junto con los costes asociados.
    - **Estimación de Costes:** se presenta la suma de todos los recursos empleados, lo que da como resultado la estimación del coste global del trabajo realizado.
    - **Modificaciones en la Planificación Inicial:** se presentan los costes añadidos al proyecto debidos al despliegue del escenario de pruebas físico.

- **Capítulo 5. Diseño e Implementación:** este capítulo describe los pasos seguidos para el diseño y puesta en marcha de las dos alternativas o escenarios virtuales. Se divide en 3 partes:
    - **Contexto Práctico:** se presenta una imagen y una explicación detallada del escenario global que se busca implementar. Todo el entorno se desarrolla dentro de una máquina virtual, incluyendo la configuración *Internet Protocol* (IP) y demás elementos necesarios para el correcto funcionamiento del sistema.
    - **Despliegue de la arquitectura funcional de OAIC:** se describe la infraestructura, detallando las posibilidades de implementación del Near-RT RIC y sus funcionalidades. Además, se abordan aspectos como la configuración de la red y las opciones disponibles para la integración de estos componentes en el entorno.
    - **Implementación del escenario srsRAN:** de igual modo al escenario anterior, se explica la implementación del sistema en este nuevo entorno, describiendo la configuración de la red y cómo se establece la conexión entre los diferentes componentes del sistema.

- **Capítulo 6. Pruebas y Experimentación:** se divide en 5 apartados:
    - **Entorno de Pruebas:** en base a lo observado en el capítulo anterior, se describen las pruebas que se van a realizar en cada entorno.
    - **OAIC. Prueba de Funcionalidad:** se valida la funcionalidad básica del sistema. Se evalúa su comportamiento y rendimiento en condiciones controladas, con el fin de asegurar que las funcionalidades clave operen como se espera.
    - **srsRAN. Monitorización y Control:** se realizan pruebas centradas en la supervisión y gestión del sistema, con enfoque en el monitoreo y control de la red mediante el RIC. Este apartado incluye:

- *Consideraciones:* antes de entrar en detalle con las pruebas a realizar, se comentan ciertos aspectos técnicos de los escenarios de implementación que será importante tener en cuenta a la hora de entender los resultados obtenidos.
- *Monitorización de Métricas:* se realizan pruebas orientadas a evaluar las capacidades de supervisión de las dos implementaciones del Near-RT RIC (*Flexible Radio Access Network Intelligent Controller* (FlexRIC) y *Open Radio Access Network Software Community Radio Access Network Intelligent Controllerr* (ORAN-SC-RIC)), analizando cómo cada una gestiona y reporta las métricas asociadas al comportamiento de la red.
  - Seguimiento con FlexRIC.
  - Seguimiento con ORAN-SC-RIC.
- *Control de Recursos Radioeléctricos:* se efectúan experimentos para evaluar el grado de control que cada implementación del Near-RT RIC puede ejercer sobre los recursos radio de la red.
  - Verificación del control de PRBs en un escenario con varios UEs.
  - Implementación de una nueva *xApp* para monitorización de métricas y control de PRBs con ORAN-SC-RIC. Esta nueva *xApp* se basa en la anterior, permite detectar congestión por medio del *Key Performance Measurements* (KPM) y reaccciona cambiando el uso de recursos radio de la red.
- *Comparativa:* se analizan las diferencias de desempeño entre las pruebas de monitoreo y control de ambas implementaciones de Near-RT RIC, con el fin de identificar qué soluciones y configuraciones resultan más eficientes.

- **Estudio del procedimiento E2 en Wireshark:** a modo complementario se analiza una traza de Wireshark con el fin de ver en detalle el procedimiento E2, identificando los mensajes intercambiados entre el Near-RT RIC y la gNB. Este análisis permite comprobar el correcto establecimiento de la asociación E2, el intercambio de mensajes de control y de información, y verificar la reacción de las *xApps* ante determinados eventos observados en la red.
- **Escenario real:** se analiza el funcionamiento de una de las *xApps* de control de PRBs en un escenario real, esto es, usando *Next Unit of Computing*s (NUCs) para el UE y la parte RAN + *core*, y haciendo uso de un *Software-Defined Radio* (SDR) para tener un canal radio *over-the-air*, no simulado.

  - **Complicaciones encontradas a lo largo del proyecto:** se detallan las dificultades técnicas y operativas que surgieron durante el desarrollo de los escenarios de prueba, así como las soluciones adoptadas para superar los obstáculos encontrados.

- **Capítulo 7. Conclusiones y Líneas Futuras:** este capítulo final está compuesto por 2 secciones:

  - **Conclusiones:** se realiza un breve análisis de los resultados obtenidos tras las pruebas realizadas en el capítulo anterior, destacando las principales conclusiones derivadas de dichos resultados.
  - **Trabajos Futuros:** se exploran posibles líneas de trabajo que permitan mejorar el rendimiento de esta infraestructura.

Capítulo 2

Estado del Arte

En este capítulo se presentan algunos de los principales proyectos y aplicaciones actuales basados en la arquitectura O-RAN junto con el RIC en redes 5G. Tras analizar su relevancia e impacto en los distintos ámbitos, se destacan las implementaciones de código abierto disponibles en la actualidad, así como la selección de dos de ellas para su uso en el desarrollo de este proyecto.

2.1. Proyectos y Aplicaciones Reales

La adopción de la arquitectura O-RAN junto con el uso del RIC está permitiendo el desarrollo de soluciones innovadoras en diferentes sectores, más allá del ámbito puramente técnico de las telecomunicaciones. A continuación, se describen algunos proyectos representativos que ilustran cómo estas tecnologías se están aplicando en casos reales, desde la salud y la industria hasta las ciudades inteligentes, el transporte y la agricultura.

1. **OpenCare5G** [3] : el proyecto OpenCare5G de 2023, impulsado por el Hospital de Clínicas de la Universidad de São Paulo, busca mejorar la atención médica remota en Brasil, especialmente en regiones rurales y comunidades amazónicas. Basado en redes privadas 5G y la arquitectura O-RAN, ofrece alta fiabilidad y baja latencia para conectar dispositivos médicos y servicios de telemedicina, facilitando diagnósticos casi en tiempo real en áreas con infraestructura limitada. La arquitectura se muestra en la figura 2.1.

   Como se observa en la figura 2.1, se implementa una red 5G O-RAN conectada a un núcleo 5G en el *datacenter* del banco Itaú, Brasil. Actualmente, no se ha integrado ningún RIC, ya que la fase inicial se centró en la conectividad básica y pruebas de casos clínicos. En fases futuras se planea incluirlo para optimizar recursos y mejorar el rendimiento de la red.

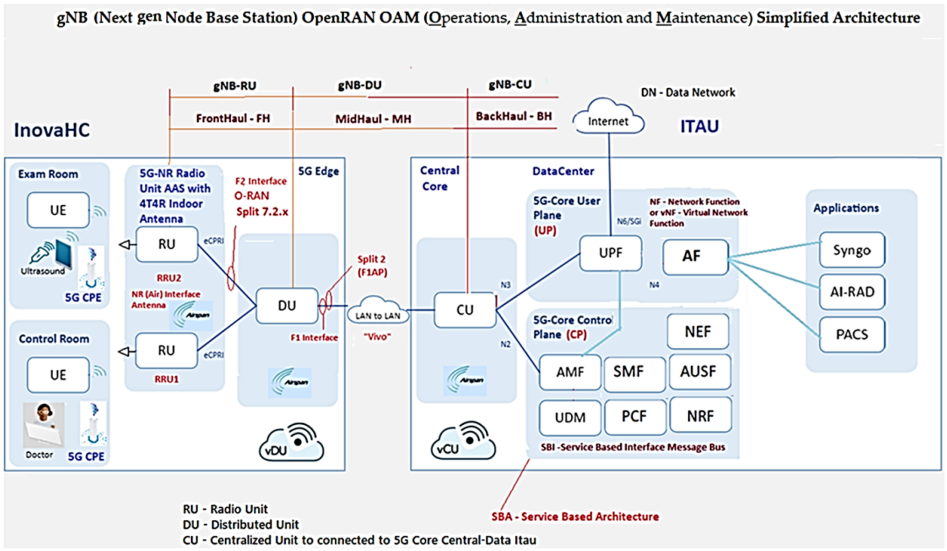


Figura 2.1: Arquitectura de OpenCare5G [3].

Durante las pruebas, se ha logrado transmitir ecografías y estudios de imagen con baja latencia (alrededor de 40 ms) y buena calidad, permitiendo la colaboración remota entre médicos. Entre los beneficios destacan el acceso a salud especializada, diagnósticos más rápidos, reducción de costes, mayor seguridad y el impulso a un ecosistema tecnológico nacional, con proyección para aplicaciones futuras como cirugía remota e AI.

2. **Red O-RAN en Polonia para Industria 4.0** [11] : este proyecto surge a finales de 2023 como una iniciativa pionera para impulsar la transformación digital del sector industrial en Polonia mediante el despliegue de la primera red privada O-RAN 5G para la Industria 4.0. Lanzada oficialmente en Cracovia, esta red marca el inicio de una nueva era de conectividad móvil avanzada con fines industriales en el país. La instalación, que opera en el *Hub4Industry* del Centro de Robótica Astor, se desarrolla conjuntamente por Hubraum (la incubadora tecnológica de Deutsche Telekom), IS-Wireless, Benetel y T-Mobile.

   Esta red sirve como plataforma clave para habilitar aplicaciones avanzadas de la Industria 4.0, como la automatización de procesos industriales mediante el control y monitoreo en tiempo real de máquinas y robots, así como la conectividad masiva de dispositivos como sensores IoT, sistemas de visión artificial o brazos robóticos. Además, proporciona un entorno de pruebas o *sandbox* para la formación, innovación y el desarrollo de nuevas aplicaciones tecnológicas, permitiendo a *star-*

tups, pymes e instituciones experimentar con soluciones 5G adaptadas a las necesidades del sector industrial.

En la misma línea, IS-Wireless continúa investigando a día de hoy en la incorporación de esta solución en otros países como Alemania [12], al mismo tiempo que intenta mejorarla con grupos de trabajo paralelos, como es el caso de la mejora de funcionalidades del RIC enfocadas al entorno industrial [13].

3. **5G *Mobile oRAN for Dense Environments* (MoDE)** [14] : esta iniciativa, nacida en 2023 y con previsión de fin en el año 2025, surge como una colaboración entre Virgin Media O2, Mavenir, VMware y la Universidad de Surrey, respaldada por el gobierno del Reino Unido a través de *UK Telecoms Innovation Network* (UKTIN). Su objetivo principal es demostrar cómo la arquitectura O-RAN puede mejorar la gestión del tráfico móvil en entornos con alta densidad de usuarios, como estadios y centros de eventos. Se enfoca en el uso de tecnologías avanzadas, como el RIC, optimización de capacidad, ahorro de energía y la implementación de infraestructuras virtualizadas para ofrecer un servicio superior, incluso en áreas densamente pobladas. Este enfoque ha sido probado con éxito en el estadio *Allianz Arena* de Múnich, demostrando su efectividad en un entorno de alta demanda de conectividad móvil.

4. ***Advanced Wireless Research Platform for Digital Agriculture & Rural Environments* (ARA) O-RAN** [15] : la plataforma ARA ha sido designada por la *O-RAN Alliance* como un centro de pruebas e integración abiertas. Esta designación ha impulsado la creación de ARA O-RAN, un sublaboratorio especializado en pruebas de O-RAN, establecido dentro del *ARA Wireless Living Lab*.

   Este proyecto se centra en el avance de tecnologías inalámbricas diseñadas para satisfacer las necesidades específicas de las comunidades rurales y el sector agrícola. Su arquitectura (véase la figura 2.2) se construye sobre el sistema operativo de nube *OpenStack*, utilizando un esquema de aprovisionamiento de recursos basado en contenedores, el cual integra componentes de software como el *Open Air Interface* (OAI) para la implementación de 5G y FlexRIC para el Near-RT RIC. Los usuarios interactúan con el sistema a través del Portal ARA, donde pueden reservar recursos.

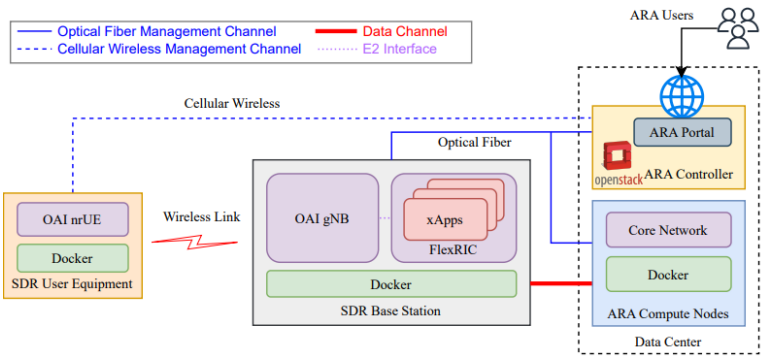


Figura 2.2: Arquitectura ARA O-RAN [15].

El entorno de pruebas (figuras 2.3 y 2.4) se basa en el despliegue al aire libre, con *Base Station*s (BSs) y UEs ubicados en entornos agrícolas reales. Además, ofrece un entorno de pruebas en interiores equipado con una extensa red SDR.

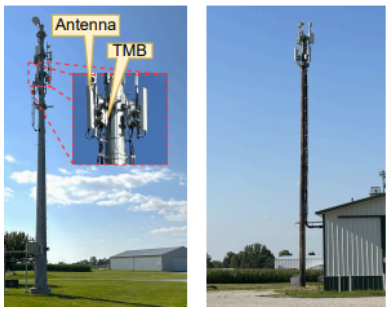


Figura 2.3: BS ARA O-RAN [15].

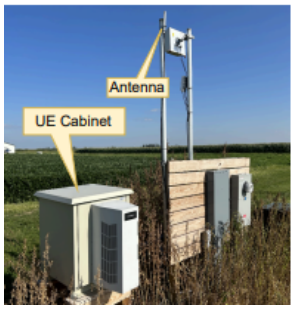


Figura 2.4: UE ARA O-RAN [15].

5. ***Deep Learning*, O-RAN y RIC en el transporte inteligente** [16] : uno de los problemas principales que enfrentan los sistemas vehiculares inteligentes es la interrupción de la comunicación debido a los *handovers*, que ocurren cuando un vehículo cambia de estación base debido a su alta movilidad. Esto afecta negativamente a la calidad de los servicios, especialmente en aplicaciones sensibles como la transmisión de video o las actualizaciones por aire. Como solución, este artículo, presentado a finales de 2024, propone el uso de la arquitectura O-RAN junto con el Near-RT RIC y modelos de aprendizaje profundo para optimizar y tomar decisiones en tiempo real, minimizando la degradación de la calidad de servicio causada.

La plataforma utilizada en esta investigación se basa en la *O-RAN*

Software Community (OSC), un entorno de código abierto respaldado por la colaboración de la *O-RAN Alliance* y la *Linux Foundation*. La arquitectura se lleva a cabo mediante el simulador *Objective Modular Network Testbed in C++* (OMNeT++), al que se le integra el módulo simu5G para simular el comportamiento de la estación base o gNB (véase la figura 2.5).

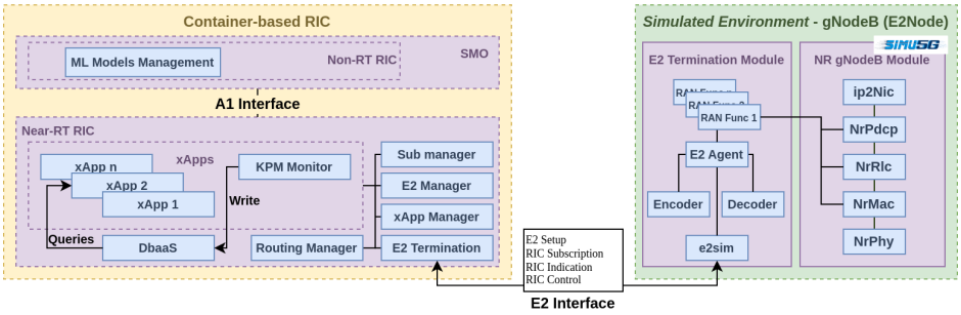


Figura 2.5: OMNeT++ [16].

En este marco, se desarrollan *xApps* que se ejecutan en el Near-RT RIC para gestionar de forma eficiente las redes de acceso radio y optimizar los procesos de *handover* mediante el uso de modelos de *Deep Learning*. El bucle de aprendizaje se muestra en la figura 2.6, como se observa hay una *xApp* de gestión de traspasos (*HO Mgmt xApp*) que supervisa a los usuarios y determina si es necesario realizar un *handover*. También, se tiene otra *xApp* de predicción de calidad de servicio (*QP xApp*) que genera predicciones a petición de la (*HO Mgmt xApp*), permitiendo anticiparse proactivamente a los *handovers*. Además, hay un módulo o modelo de entrenamiento que genera modelos de *deep learning* utilizando una base de datos, y transfiere los modelos entrenados a la *QP xApp*.

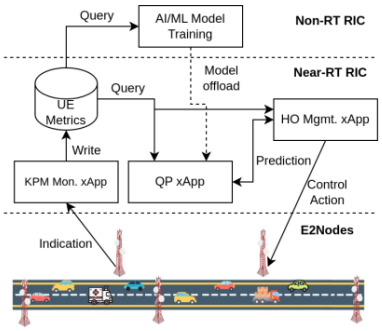


Figura 2.6: Solución para el problema del *handover* [16].

Los resultados de las pruebas mostraron que la propuesta logró una

mejora significativa en parámetros como el *Channel Quality Indicator* (CQI), el rendimiento, la latencia y la congelación de sesión en comparación con el procedimiento estándar de *handover* del *3rd Generation Partnership Project* (3GPP) [17].

6. **O-RAN en *Telcom Infra Proyect*** [18] y [19]. *Telecom Infra Project* es una comunidad global de organizaciones líderes que colaboran para transformar la forma en que se desarrolla e implementa la infraestructura de telecomunicaciones. Uno de sus proyectos, OpenRAN, tiene como objetivo promover la innovación y la adopción de soluciones interoperables en redes RAN. Para ello, busca alianzas sólidas en toda la industria con organizaciones como 3GPP [17], *Open Networking Foundation* (ONF) o la *O-RAN Alliance*, que comparten una visión similar y trabajan para acelerar la desagregación y la innovación.

Como se ha podido observar, la infraestructura que se estudia en este trabajo fin de máster es muy prometedora e innovadora, mostrando grandes beneficios en áreas tan importantes como la salud, industria, transporte o agricultura, además de en las propias redes de tráfico de datos de usuarios, durante los anteriores 2 años.

2.2. Implementaciones Disponibles

1. **Aether** [20] es una plataforma 5G de código abierto diseñada para entornos de borde o *edge computing*, desarrollada por la ONF [21]. Es de código abierto y es posible acceder a su GitHub [22] y a una guía detallada para su desarrollo [23]. Su arquitectura se basa en dos componentes principales:

   - **SD-Core** (véase la figura 2.7) es un núcleo móvil nativo de la nube que cumple con los estándares del 3GPP [17], compatible con despliegues 4G/LTE, 5G-*Non-Standalone* (NSA) y 5G-*Standalone* (SA). Ofrece una*Application Programming Interface* (API) que permite a los operadores gestionar suscriptores, definir parámetros de calidad de servicio para los distintos slices, modificar la configuración de las funciones de red en tiempo de ejecución y recopilar datos de telemetría.
   - **SD-RAN** (véase la figura 2.8) es una implementación abierta y basada en software de una red de acceso radio dividida, que sigue la arquitectura definida por la O-RAN. Incorpora un Near-RT RIC nativo de la nube y desarrollado sobre el controlador de ONF, y un conjunto de *xApps* de referencia. SD-RAN no solo se alinea con las especificaciones actuales de O-RAN, sino que también contribuye activamente a su evolución.

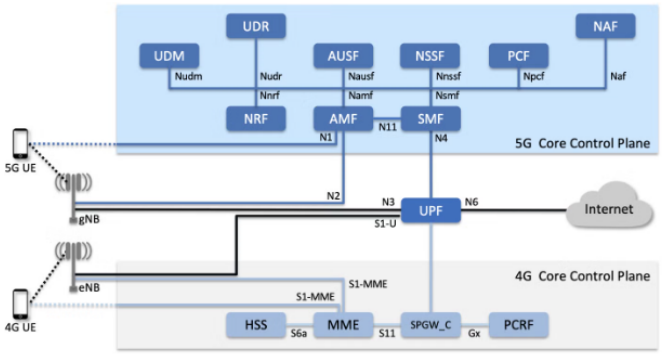


Figura 2.7: SD-CORE [20].

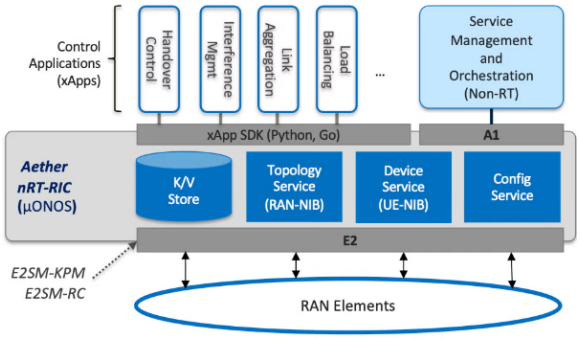


Figura 2.8: SD-RAN [20].

2. **ns-O-RAN** [24] es otra plataforma de simulación de código abierto que combina una pila de protocolos 4G/5G en el simulador ns-3 con una interfaz E2 compatible con O-RAN. Esta herramienta mejora la recopilación de datos y las pruebas de xApp, facilitando soluciones de AI y ML. En la figura 2.9 se puede ver la arquitectura planteada.

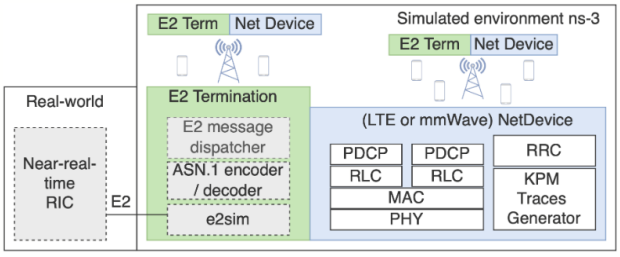


Figura 2.9: Arquitectura ns-O-RAN [25].

Soporta la recopilación de métricas en diversos escenarios y aplicaciones, y permite un control de bucle cerrado mediante modelos *E2 Service Model* (E2SM) para gestionar el tráfico y la movilidad.

3. **OAIC** es un proyecto emulado basado en OAI [9] y que dispone de software para desplegar:

   - **Red troncal (CN)**: se implementa con *OpenAirCN* [26]. Empezó ofreciendo soporte para un *core* 4G y actualmente para 5G-SA, el cual es empleado en este trabajo. La infraestructura de este núcleo de red se puede observar en la figura 2.10.

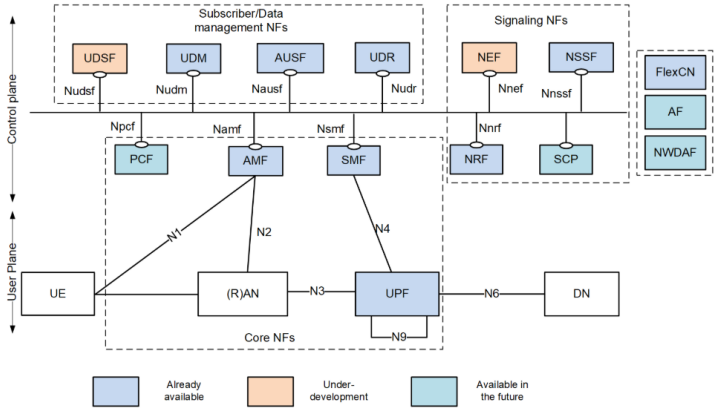


Figura 2.10: OAI CN [26].

   - **Red de Acceso Radio (RAN)**: *OpenAir5G* permite configurar diversos parámetros de la gNB, de los UEs y del simulador de canal *RFSimulator* [27] para 5G-SA. El esquema radio se puede apreciar en la figura 2.11. Aunque *OpenAir5G* no es completamente parte de la arquitectura O-RAN, comparte principios de una RAN *open-source*, permitiendo la interoperabilidad con soluciones O-RAN y facilitando la construcción y simulación de redes 5G de manera flexible, por lo que se estudia en este trabajo.

Además, junto con FlexRIC [29] para el RIC, permite simular redes 5G completas. Fue desarrollado en 2020, en el marco de la iniciativa de *OpenRAN*, permitiendo la integración de múltiples proveedores y componentes. Esta plataforma está diseñada para ser una herramienta útil tanto para investigadores como para operadores de telecomunicaciones que necesitan probar soluciones de red 5G en escenarios de simulación antes de su implementación real. Se puede consultar su repositorio de GitHub [30] y una guía a su implementación [31].

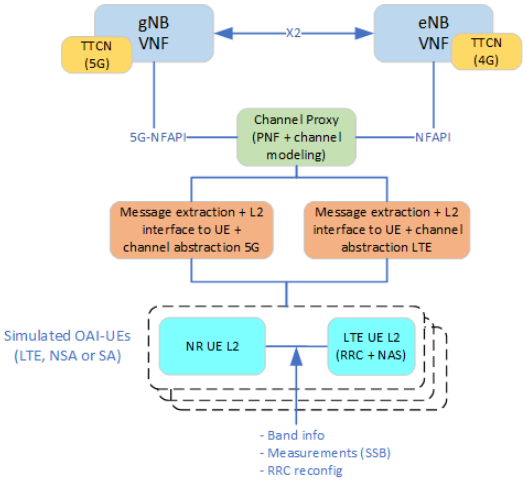


Figura 2.11: Red radio OAI [28].

4. **srsRAN** es un proyecto de código abierto iniciado en 2017, primero ofreciendo soluciones para 4G, después 5G-NSA y ahora 5G-SA, y desarrollado por la empresa irlandesa *Software Radio Systems* (SRS) [32], la cual apuesta por soluciones ligeras y portátiles con interfaces de radio definidas por software que favorecen la adaptabilidad y la simplicidad en entornos experimentales y de despliegue.

   En la práctica, utiliza *Open 5 Generation System* (Open5GS) [33] como núcleo (figura 2.12) y su propia solución completa para la parte radio (la conexión entre el UE y la gNB se lleva a cabo por medio del simulador *Zero Message Queue* (ZeroMQ) [34]) con la opción de usar FlexRIC [29] o una versión modificada del ORAN-SC-RIC [35] para el control de la red. Véase la arquitectura radio en la figura 2.13.

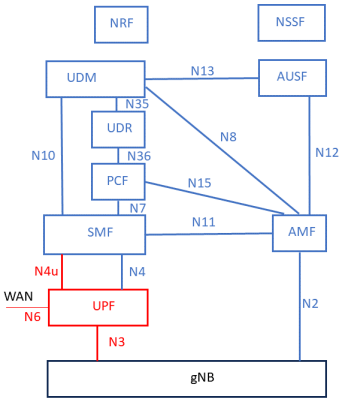


Figura 2.12: Open5GS *core* [36] .

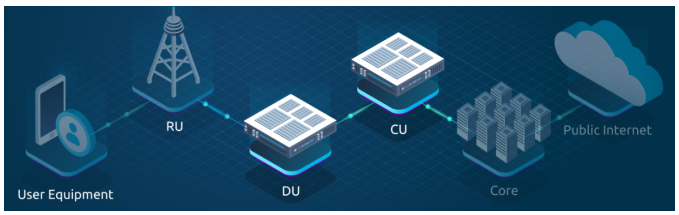


Figura 2.13: srsRAN radio [8] .

Actualmente sigue en constante desarrollo con la presentación de *workshops* cada año donde se incluyen funcionalidades nuevas al ecosistema. En [8] es posible encontrar tanto un enlace a su repositorio, actualizado a día de hoy, como links a la documentación. En este último se incluyen numerosos apartados con guías de implementación, configuraciones de ejemplo, dudas comunes, etc.

Tras estudiar las cuatro opciones disponibles, se ha decidido implementar tanto la plataforma OAIC como la de srsRAN. Ambas ofrecen ventajas significativas, como su activa comunidad de desarrolladores, constante actualización de sus repositorios y la inclusión regular de nuevos *workshops*, lo que asegura que siempre estén a la vanguardia en cuanto a funcionalidades y compatibilidad con las últimas tecnologías. Además, ambas soluciones emplean herramientas de terceros, lo que facilita la interoperabilidad entre diferentes proveedores de software y fomenta un entorno flexible y modular. Sin embargo, entre las dos, la opción más prometedora es srsRAN. Es por esto que se ha decidido que sea la plataforma principal para las pruebas y desarrollos más profundos, mientras que la implementación de OAIC complementa el entorno de pruebas, proporcionando una valiosa alternativa para explorar diferentes configuraciones y escenarios.

Capítulo 3

Fundamentos Teóricos

Este capítulo presenta los conceptos fundamentales necesarios para comprender el funcionamiento y la evolución de las redes 5G. Se abordan tanto su estructura técnica como su evolución hacia arquitecturas *open-source*, desagregadas e inteligentes.

3.1. Fundamentos de las redes 5G

La quinta generación de redes móviles, conocida como 5G, surge como un salto revolucionario en la tecnología móvil, adentrándose en sectores como la industria 4.0, la automoción, la movilidad, el transporte, el sistema sanitario y el ecosistema del entretenimiento. Para todos estos ámbitos, organizaciones como la *International Telecommunication Union* (ITU) [37] o 3GPP [17] han lanzado muchas iniciativas que requieren definir bien los escenarios disponibles que se tienen, los cuales se definen en el *International Mobile Telecommunications 2020* (IMT-2020) de la ITU.

3.1.1. Escenarios

Se definen tres grandes escenarios de uso que guían el diseño de 5G, como se comentó en el capítulo 1:

- ***Enhanced Mobile Broadband* (eMBB)**: enfocado en el acceso multimedia, de servicios y datos por parte de los usuarios. Casos típicos incluyen *streaming* en UHD/4K o realidad aumentada. Requiere altas tasas de datos y cobertura amplia.
- ***Ultra-Reliable & Low Latency Communications* (URLLC)**: comunicaciones de ultra fiabilidad y latencia extremadamente baja. Aplicaciones como automatización industrial, cirugía remota o conducción autónoma dependen de este perfil.

- ***Massive Machine-Type Communications* (mMTC)**: orientado a conectar un gran número de dispositivos con poco tráfico individual pero alta densidad de conexión. Usos como ciudades inteligentes, redes eléctricas inteligentes y sensores distribuidos son representativos.

Estas tres categorías son las más representativas y hacen una recopilación global de los principales casos de uso o escenarios (véase la figura 3.1).

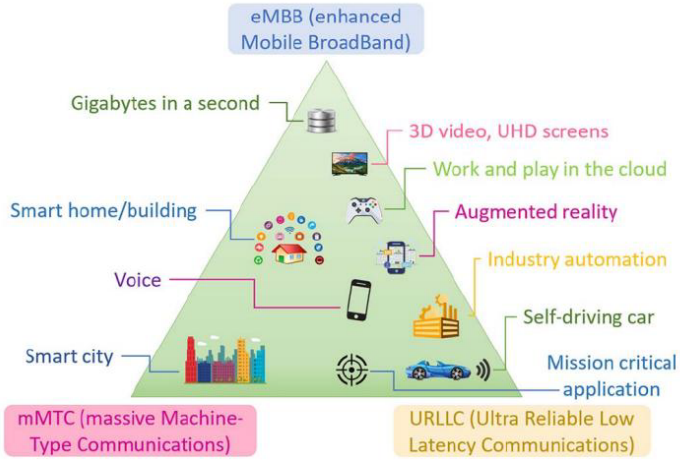


Figura 3.1: Escenarios de uso del 5G [38].

3.1.2. Requisitos técnicos

La recomendación *International Telecommunication Union - Radiocommunication* (ITU-R) M.2410 [39] establece los requisitos clave que deben cumplir las tecnologías 5G/6G. Algunos de los principales *Key Performance Indicator*s (KPIs) se pueden ver en la figura 3.2 son:

- **Velocidad pico de datos**: hasta 20 Gbps para eMBB.
- **Latencia**: menor de 4 ms para eMBB y hasta 1 ms para URLLC.
- **Densidad de conexión**: hasta un millón de dispositivos por km^2 para mMTC.
- **Movilidad**: soporte para usuarios en movimiento a velocidades de hasta 500 km/h.
- **Eficiencia espectral**: tres veces superior a la de IMT-Advanced.

Como conclusión, se espera que el tráfico global crezca entre 10 y 100 veces entre 2020 y 2030, con un aumento significativo del tráfico móvil de vídeo y *Machine to Machine* (M2M), destacando la asimetría del tráfico, donde el enlace descendente representa la mayor parte del tráfico de datos.

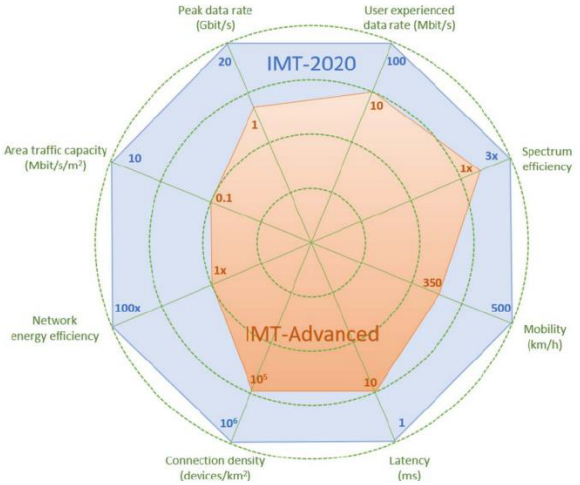


Figura 3.2: Requisistos técnicos del 5G [38].

3.2. Arquitectura 5G

Tras haber analizado de forma breve la aparición y los objetivos fundamentales de la tecnología móvil 5G, es importante profundizar en su arquitectura de red, que constituye el esqueleto sobre el que se sustentan las capacidades y mejoras que esta nueva generación móvil promete. La arquitectura de 5G incluye tanto el núcleo de la red o *core*, que gestiona el control y la administración de los servicios, como la infraestructura de acceso radio o RAN, que facilita la comunicación entre los dispositivos móviles y las estaciones base. Además, cabe destacar que la capa física, la cual especifica características radio importantes para la conexión entre las estaciones base y el usuario, también juega un papel muy importante, por lo que también se lleva a cabo un repaso de ella.

Como concepto inicial, la arquitectura 5G tiene los mismos elementos que las generaciones anteriores: un UE, la RAN y la CN, como se muestra en la figura 3.3:

Figura 3.3: Arquitectura general 5G [40].

Además, se definen 2 tipos de despliegue 5G:

- **Arquitectura No Autónoma o NSA**: en este modelo, el acceso radio 5G se integra con la infraestructura 4G ya existente, usando la red de acceso *Long Term Evolution* (LTE) y el núcleo *Evolved Packet Core* (EPC). Esto permite aprovechar las ventajas de la nueva radio 5G, como una menor latencia, sin necesidad de reemplazar la red completa. Sin embargo, en esta configuración solo se ofrecen servicios de 4G. En esta opción, pensada como una etapa transitoria hacia el 5G completo, la estación base de LTE, conocida como *Evolved Node B* (eNB), actúa como nodo maestro y la estación base 5G, denominada *Evolved Next Generation Node B* (en-gNB), como nodo secundario, comunicándose entre sí mediante la interfaz X2. En la figura 3.4 izquierda es posible observar esta arquitectura.
- **Arquitectura Autónoma o SA**: en este caso, el acceso radio 5G se conecta directamente al núcleo 5G o *5G Core* (5GC), sin depender de ningún elemento de la red 4G. Esta configuración permite ofrecer todo el conjunto de servicios diseñados para el 5G desde su primera fase. Las estaciones base 5G o gNB se conectan entre ellas mediante la interfaz Xn y se comunican con el núcleo 5G a través de la interfaz NG. Este despliegue (véase la figura 3.4 derecha) representa una red 5G completamente independiente.

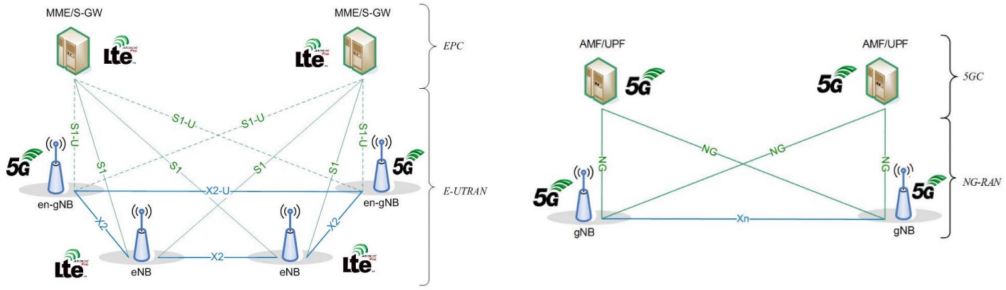


Figura 3.4: Arquitectura 5G NSA [41].

A continuación, se explican en más detalle cada una de las partes de la arquitectura 5G SA, que es la que actualmente se utiliza.

3.2.1. *Core* 5G

La arquitectura 5GC sigue un modelo *Service-Based Architecture* (SBA), donde las funciones de red o *Network Function* (NF) ofrecen servicios a otras

funciones o consumidores autorizados a través de interfaces comunes. Este enfoque modular facilita la virtualización, permitiendo que las funciones se implementen de forma distribuida, escalable y redundante. Así, los servicios pueden prestarse desde distintas ubicaciones, mejorando la resiliencia y permitiendo la continuidad del servicio incluso durante tareas de mantenimiento.

En la figura 3.5 se puede ver la estructura del *core* 5G, formada por las diferentes funciones de red:

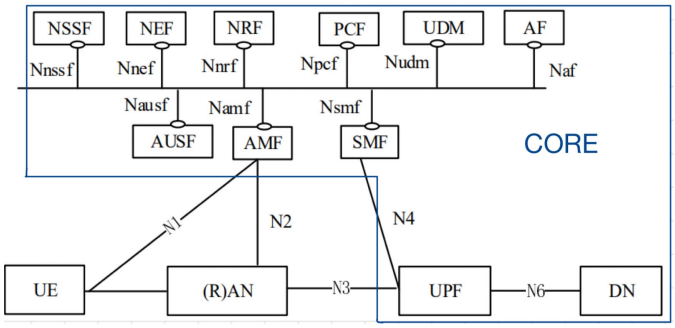


Figura 3.5: *Core* 5G SA [41].

- ***Authentication Server Function* (AUSF)**: proporciona el servicio de autenticación, gestionando las credenciales y validando el acceso de los dispositivos a la red.

- ***Access and Mobility Management Function* (AMF)**: gestiona el control de acceso, la movilidad entre celdas y la autenticación de los usuarios. Interactúa con la RAN y otras funciones de red para soportar la movilidad y el *Network Slicing*. Además, se encarga de la selección de la *Session Management Function* (SMF).

- ***Session Management Function* (SMF)**: se encarga de la gestión de sesiones de los usuarios, asignando direcciones IP, y gestionando el tráfico hacia el *User Plane Function* (UPF). También maneja la *Quality of Service* (QoS) y las políticas de tráfico.

- ***User Plane Function* (UPF)**: procesa y enruta el tráfico del plano de usuario, aplicando políticas de red y generando informes de tráfico. Interactúa con el núcleo de la red y la red de datos.

- ***Network Slice Selection Function* (NSSF)**: permite seleccionar las *slices* adecuadas para cada usuario, facilitando el uso de múltiples *slices* simultáneamente.

- ***Network Exposure Function* (NEF)**: proporciona la capacidad de exposición externa de las funciones de red, incluyendo monitorización, aprovisionamiento y enrutamiento de tráfico.
- ***Network Repository Function* (NRF)**: facilita la gestión de las funciones de red, incluyendo el registro, deregistro y la autorización y descubrimiento de las NF.
- ***Policy Control Function* (PCF)**: gestiona las políticas de acceso y QoS para garantizar la calidad del servicio y la gestión de tráfico.
- ***Unified Data Management* (UDM)**: almacena los datos de los suscriptores, generando y gestionando las credenciales de autenticación y autorizando el acceso a la red. Utiliza la función *User Data Repository* (UDR) (no explícita en la figura 3.5) como repositorio central para almacenar y recuperar dicha información de suscripción.
- ***Application Function* (AF)**: interactúa con otras funciones de red del plano de control según los servicios y propiedades de la red.

3.2.2. Radio 5G

La arquitectura de la red de acceso en 5G o *Next Generation Radio Access Network* (NG-RAN) (véase la figura 3.6) se caracteriza por tener una estructura simplificada. Consiste en una única entidad, la gNB, que se conecta al núcleo de la red 5G (5GC) mediante la interfaz NG. Esta interfaz, que se corresponde con la interfaz S1 en 4G, gestiona la movilidad de los usuarios y el establecimiento de sesiones. Además, la gNB puede establecer comunicación con otros gNB a través de la interfaz Xn, que se deriva de la interfaz X2 de la red 4G. La interfaz Xn proporciona varias funciones esenciales, como la gestión de la configuración, la preparación y cancelación de *handovers*, la recuperación de contexto de usuario, y la gestión del ahorro de energía, entre otras. También permite la conectividad dual y la transferencia de datos entre nodos.

En algunos casos, el gNB interactúa con el eNB de la red 4G mediante la interfaz X2, que ofrece funciones como la conectividad para la gestión de movilidad de usuarios entre las redes 4G y 5G. Específicamente, la interfaz X2 permite la transferencia de datos entre nodos, la actualización de contexto de usuario y el control de la conectividad dual.

Por otro lado, la gNB también se conecta a los dispositivos de usuario o UE mediante la interfaz *New Radio* (NR), que no se muestra explícitamente en la figura, y proporciona los recursos radioeléctricos necesarios para el acceso a la red.

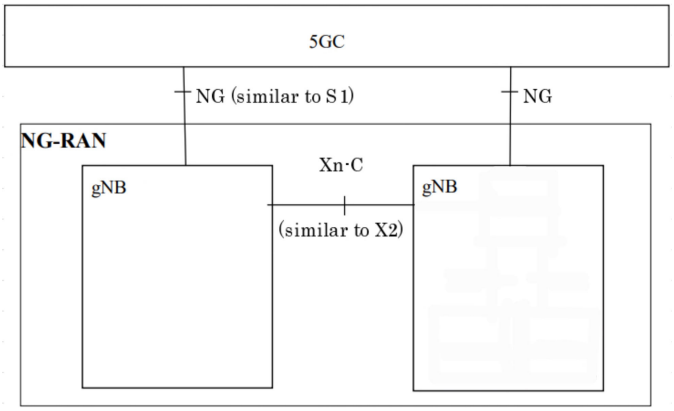


Figura 3.6: *Radio* 5G SA [41].

Algunas de las funciones principales de la estación base son:

1. **Gestión de recursos de radio**: esto incluye el control de las portadoras de radio, el control de admisión de radio, la gestión de la movilidad de las conexiones y la asignación dinámica de recursos para los UE, tanto en el enlace ascendente como descendente (programación).

2. **Seguridad**: se encarga de la compresión de la cabecera IP, así como del cifrado y la protección de la integridad de los datos que transitan por la red.

3. **Selección de AMF**: en situaciones donde no es posible determinar el enrutamiento a una AMF desde la información proporcionada por el UE, la gNB selecciona una AMF para gestionar la conexión.

4. **Enrutamiento de datos y control**: los datos del plano de usuario se enrutaran hacia el UPF, mientras que la información del plano de control se dirige hacia la AMF.

5. **Configuración y liberación de conexiones**: es responsable de la configuración y la liberación de las conexiones entre el UE y la red.

6. **Mediciones y movilidad**: configura las mediciones y genera informes de mediciones que son necesarios para la gestión de la movilidad y la programación eficiente de la red.

7. **Gestión de QoS**: gestiona el flujo de QoS y realiza el mapeo adecuado a las portadoras de radio de datos, asegurando que se mantengan los estándares de calidad del servicio.

8. **Interfuncionamiento NR y *Evolved Universal Terrestrial Radio Access* (E-UTRA)**: asegura un estrecho interfuncionamiento

entre la interfaz NR de 5G y E-UTRA de LTE, garantizando la interoperabilidad y el funcionamiento de la red 5G junto con las redes 4G existentes.

Actualmente, se está adoptando cada vez más una nueva solución conocida como O-RAN, la cual se comenta en la sección 3.3.1.

3.2.3. Pila de protocolos

La pila de protocolos organiza las funciones de comunicación en capas bien definidas, tanto para el plano de control como para el plano de usuario (véase las figuras 3.7 y 3.8).

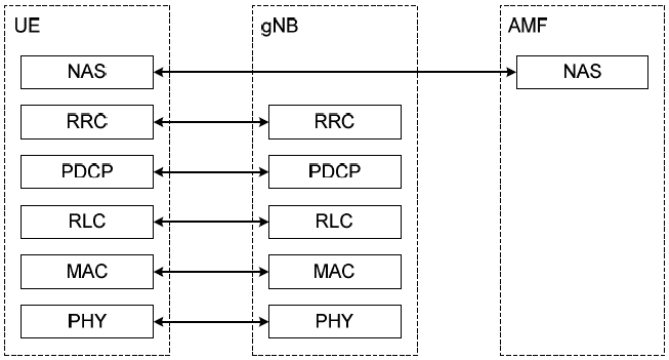


Figura 3.7: Pila de protocolos en el plano de control 5G [41].

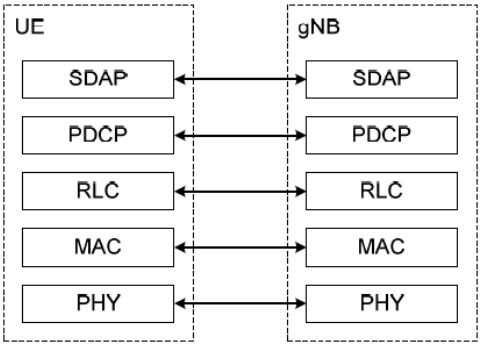


Figura 3.8: Pila de protocolos en el plano de usuario 5G [41].

Aquellos protocolos o capas comunes a ambos planos son:

- ***Physical Layer* (PHY)**: encargada de la modulación/demodulación y transmisión/recepción de la señal sobre la interfaz de radio, más adelante se profundiza en esta capa.

- ***Medium Access Control* (MAC)**: gestiona el acceso a los recursos radio y la multiplexación de datos. Además, se encarga de la corrección de errores mediante *Hybrid Automatic Repeat reQuest* (HARQ), el mapeo entre canales lógicos y canales de transporte, la notificación de información de planificación, el manejo de prioridades entre UEs mediante planificación dinámica, el manejo de prioridades entre canales lógicos de un mismo UE mediante priorización y la inserción de *padding* cuando es necesario.

- ***Radio Link Control* (RLC)**: proporciona servicios como transferencia de *Protocol Data Unit*s (PDUs) de capas superiores, segmentación, control de errores y reensamblado de datos.

- ***Packet Data Convergence Protocol* (PDCP)**: en el plano de usuario proporciona numeración de secuencia, compresión y descompresión de cabeceras (*RObust Header Compression (ROHC) only*), transferencia de datos de usuario, reordenamiento y detección de duplicados, enrutamiento de PDUs (en caso de *split bearers*), retransmisión de *Service Data Unit*s (SDUs), cifrado y descifrado, etc. En el plano de control también ofrece protección de integridad con cifrados, detección de duplicados y transferencia de datos de control.

Por su lado, en el plano de control se tienen:

- ***Radio Resource Control* (RRC)**: se encarga de aspectos relacionados con el control de la conexión radio, incluyendo la gestión de la conexión RRC entre el UE y el NG-RAN, la configuración y liberación de portadores de señalización (*Signaling Radio Bearer* (SRB)) y portadores de datos (*Data Radio Bearer* (DRB)), funciones de movilidad como el traspaso y la transferencia de contexto, la gestión de la calidad de servicio (QoS), la recuperación de fallos en el enlace radio y la transferencia de mensajes *Non-Access Stratum* (NAS).

- ***Non-Access Stratum* (NAS)**: gestiona funciones no relacionadas directamente con la red de acceso, como la autenticación, la gestión de la movilidad y el control de seguridad, siendo transportado de forma transparente por la red.

Mientras, en el plano de usuario:

- ***Service Data Adaptation Protocol* (SDAP)**: encargado de mapear los flujos de QoS con los portadores de datos radio, así como de marcar el identificador de flujo de QoS, denominado *QoS Flow Identifier* (QFI), en los paquetes tanto de bajada como de subida.

Capa Física

Dentro de las pilas de protocolos vistas, la capa más baja común a ambos planos es la capa física o PHY. Esta capa es especialmente crítica, ya que se encarga de la modulación, demodulación, codificación, transmisión y recepción de las señales en la interfaz de radioes decir, entre la gNB y los UE. A continuación, se profundiza en el papel y los mecanismos específicos de la capa física.

Modulación

En 5G NR, la modulación utilizada para la transmisión en el enlace descendente o *Downlink* (DL) es *Cyclic Prefix Orthogonal Frequency Division Multiplexing* (CP-OFDM), al igual que en LTE, permitiendo una alta eficiencia espectral y compatibilidad con *Multiple Input Multiple Output* (MIMO). Para el enlace ascendente o *Uplink* (UL), 5G NR admite dos formas de onda: CP-OFDM, que se emplea para señales de control y también para datos cuando se requiere soporte multicapa (varios flujos simultáneamente), y *Discrete Fourier Transform spread Orthogonal Frequency Division Multiplexing* (DFT-s-OFDM), que se utiliza principalmente para la transmisión de datos de usuario en configuraciones de una sola capa.

Frecuencias

En cuanto al rango espectral empleado, se soporta dos rangos de frecuencias como se puede ver en la tabla 3.1.

Rango de frecuencias [MHz]	**BW de canal [MHz]**
FR1 : 410 – 7125	5, 10, 15, 20, 25, 30, 40, 50, 60, 80, 90, 100
FR2: 24250 – 52600	50, 100, 200, 400

Tabla 3.1: Rango de frecuencias en 5G NR.

Estructura de trama

Una trama 5G tiene la estructura mostrada en la figura 3.9 y una duración fija de 10 ms, dividida en 10 subtramas de 1 ms cada una. Dependiendo del valor de *Subcarrier Spacing* (SCS) (representa la distancia en frecuencia entre subportadoras adyacentes y puede tomar el valor 15, 30, 60, 120 o 240 kHz), cada subtrama puede contener un número variable de ranuras (véase la tabla 3.2).

Cada ranura contiene 14 símbolos *Orthogonal Frequency Division Multiplexing* (OFDM), todos precedidos por un prefijo cíclico, excepto para el caso de SCS de 60 kHz, en el cual también puede usarse un prefijo cíclico extendido, que es aproximadamente cuatro veces más largo y reduce el número de símbolos a 12 por ranura. Este prefijo extendido está diseñado

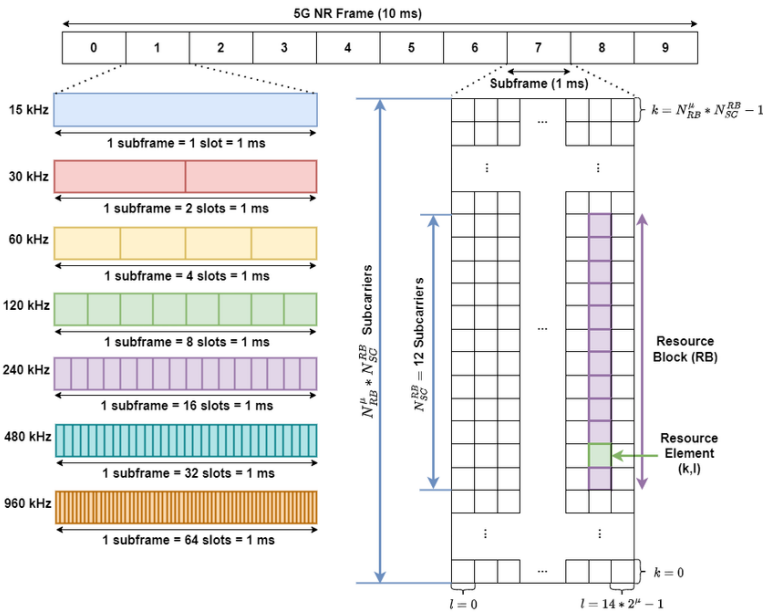


Figura 3.9: Estructura de trama en 5G [42].

SCS [kHz]	**Nº de ranuras por subtrama (1 ms)**
15	1
30	2
60	4
120	8
240	16

Tabla 3.2: Relación entre el espaciado de subportadoras (SCS) y el número de ranuras por subtrama en 5G NR.

para entornos con alta dispersión temporal, como celdas de gran tamaño. La duración de los símbolos y del prefijo cíclico está inversamente relacionada con el SCS: a mayor espaciado entre subportadoras, menor será la duración del símbolo OFDM. Además, 1 subportadora y 1 símbolo OFDM forman un *Resource Element* (RE), mientras que 12 subportadoras constituyen un *Resource Block* (RB).

En cuanto a la asignación de recursos temporales, se admite tanto *Frequency Division Duplex* (FDD) como *Time Division Duplex* (TDD) usando la misma estructura de trama. En TDD, cada símbolo OFDM dentro de una ranura puede configurarse como de enlace ascendente (UL), enlace descendente (DL) o flexible.

Por último, se destaca que aunque lo habitual es que una transmisión

ocupe una ranura completa, 5G NR permite realizar transmisiones en fracciones de ranura, con un mínimo de dos símbolos OFDM, lo cual es ideal para aplicaciones de URLLC.

Canales y señales físicas

En la tabla 3.3 se presentan las principales características de los canales físicos utilizados en 5G NR. Estos canales permiten la transmisión de datos, control y sincronización entre el dispositivo y la red, empleando técnicas como la modulación, la codificación y la asignación eficiente de recursos.

Dirección	Canal Físico	Modulación	Codificación de Canal
DL	PDSCH	QPSK, 16QAM, 64QAM, 256QAM	LDPC
DL	PBCH	QPSK	Código Polar
DL	PDCCH	QPSK	Código Polar
UL	PUSCH	QPSK, 16QAM, 64QAM, 256QAM, pi/2 BPSK cuando se selecciona DFT-s-OFDM	Código LDPC
UL	PUCCH	pi/2-BPSK, BPSK, QPSK dependiendo del formato de PUCCH y tamaño del bit de información	Dependiendo del UCI: código de repetición, código polar, código simple, etc
UL	PRACH	-	-

Tabla 3.3: Resumen de las características de los canales físicos en 5G.

En el enlace descendente se definen tres tipos principales de canales físicos:

- **Physical Downlink Shared Channel (PDSCH)** se emplea para la transmisión de datos descendentes.
- **Physical Broadcast Channel (PBCH)** se utiliza para los sistemas de información broadcast.
- **Physical Downlink Control Channel (PDCCH)** transmite la información de control y se usa para decisiones de *scheduling* entre el PDSCH y PUSCH.

Al igual que en el ascendente se tienen:

- **Physical Uplink Shared Channel (PUSCH)** se usa para la transmisión de datos en el enlace ascendente.

- **Physical Uplink Control Channel (PUCCH)** transporta la información de control ascendente, incluye HARQ, *scheduling requests* e información del estado del canal descendente.

- **Physical Random Access Channel (PRACH)** solicita la conexión mediante *random access* del UE.

En cuanto a las señales empleadas en los canales mencionados, hay algunas exclusivas del enlace descendente:

- ***Channel State Information Reference Signal* (CSI-RS)** transmite la información del estado del canal para la estimación de calidad.

- ***Primary Synchronization Signal* (PSS)** se emplea para la sincronización primaria en el enlace descendente.

- ***Secondary Synchronization Signal* (SSS)** se utiliza para la sincronización secundaria en el enlace descendente.

Una que sólo se emplea en ascendente:

- ***Sounding Reference Signal* (SR-S)** se usa como señal de referencia para sondeo en el enlace ascendente.

Y dos de ellas que son flexibles:

- ***Demodulation Reference Signals* (DM-RS)** se emplea como señal de referencia para la demodulación.

- ***Phase-Tracking Reference Signals* (PT-RS)** se utiliza para el seguimiento de fase en la transmisión.

3.3. Evolución de las redes 5G

En las redes móviles tradicionales, la arquitectura de acceso radio (RAN) se basaba en una RU conectada a una BBU. Sin embargo, con la evolución en 5G, esta arquitectura ha sido mayormente reemplazada por modelos más flexibles y desagregados. En este contexto (véase la figura 3.10), el nodo de nueva generación gNB puede dividirse funcionalmente en una *O-RAN Radio Unit* (O-RU), una *O-RAN Central Unit* (O-CU) y una o varias *O-RAN Distributed Unit* (O-DU), las cuales se comunican entre sí mediante la interfaz F1. Esta interfaz proporciona servicios clave como la gestión del contexto del UE, la transferencia de mensajes del plano de control y funciones de paginación.

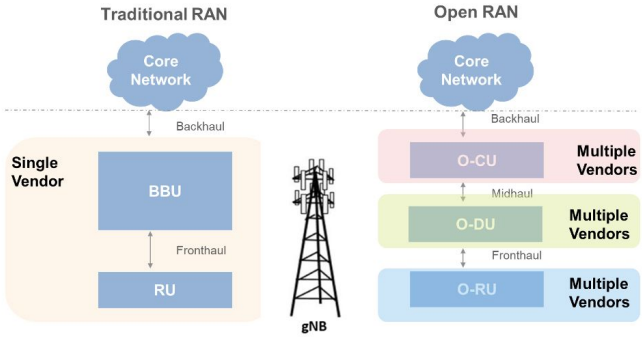


Figura 3.10: Arquitectura RAN vs. O-RAN [43].

3.3.1. O-RAN

La arquitectura O-RAN, propuesta por la *O-RAN Alliance* [44], es el resultado de un esfuerzo global iniciado en 2018 por operadores líderes con el objetivo de transformar las redes de acceso radio RAN tradicionales en infraestructuras más abiertas, virtualizadas e interoperables.

La arquitectura lógica propuesta se puede observar en la figura 3.11. Descompone los elementos tradicionales de la RAN en múltiples funciones distribuidas e inteligentes y se divide principalmente en dos dominios: el *Service Management and Orchestration* (SMO) y las O-RAN NFs.

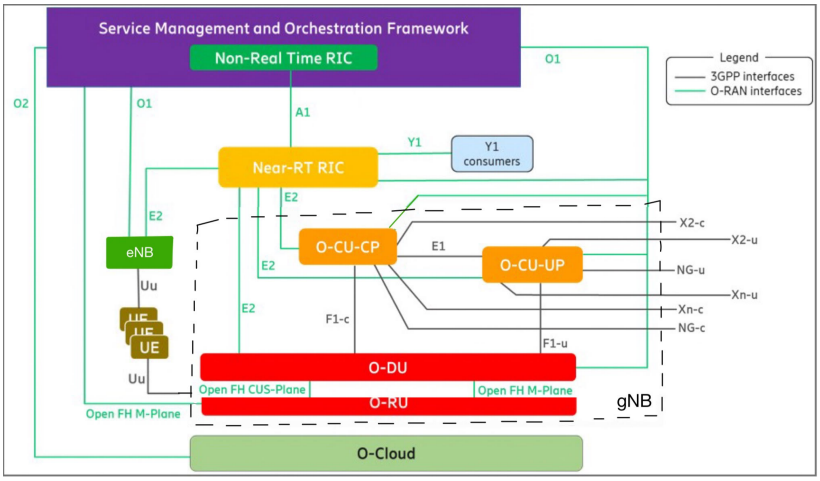


Figura 3.11: Arquitectura lógica de O-RAN.

- ***Service Management and Orchestration*** **(SMO)**: proporciona funciones de gestión del ciclo de vida, configuración, monitoreo y orquestación de los elementos de red. Está basado en la SBA, que permite

la interoperabilidad mediante interfaces basadas en servicios estándar, donde los elementos del SMO pueden actuar como productores y consumidores de servicios. Esto facilita la validación de los servicios, la especificación de APIs y modelos de datos, y la identificación de servicios comunes que pueden ser producidos por un solo productor, como autenticación y gestión de datos. Dentro de éste se encuentra:

- ***Non Real Time Radio Access Network Intelligent Controller*** **(Non-RT RIC)**: este elemento se encarga del control no en tiempo real (mayor a 1 segundo), proporcionando políticas, directivas de optimización y entrenamiento de modelos de AI. Además, gestiona aplicaciones llamadas *rApps*. Se comunica con el Near-RT RIC mediante la interfaz A1.

- **O-RAN NFs**

  - ***Near Real Time Radio Access Network Intelligent Controller*** **(Near-RT RIC)**: realiza control y optimización casi en tiempo real (10 ms a 1 s) a través de *xApps*. Se comunica con el Non-RT RIC mediante la interfaz A1 y con las funciones de red a través de la interfaz E2.
  - ***O-RAN Central Unit - Control Plane*** **(O-CU-CP)**: gestiona el plano de control, incluyendo la movilidad, el establecimiento de sesiones y la gestión del contexto de usuarios, con interfaces hacia 5GC y otros elementos de la red.
  - ***O-RAN Central Unit - User Plane*** **(O-CU-UP)**: maneja el plano de usuario, transportando datos entre el núcleo de red y el O-DU.
  - ***O-RAN Distributed Unit*** **(O-DU)**: gestiona funciones como MAC, RLC y parte de la capa física (high-PHY y/o low-PHY), y se conecta con la O-RU a través de la interfaz *Open Fronthaul*. Se comunica con el SMO a través de la interfaz O1 para tareas de gestión, y con el RIC en tiempo cercano real mediante la interfaz E2. El O-DU puede dividirse en DU-high y DU-low. DU-high contiene las funciones de capa 2 y coordina con la unidad de control mediante la interfaz F1, manejando control de celda, gestión de usuarios y tunelización de datos. DU-low se encarga de tareas de capa física, incluyendo programación UL/glsDL, sincronización temporal, manejo de señales de referencia y uso de aceleradores de hardware para codificación/decodificación. La comunicación entre DU-high y DU-low se realiza mediante FAPI+.
  - ***O-RAN Radio Unit*** **(O-RU)**: realiza la conversión digital-analógica y la transmisión/recepción de señales RF. Se conecta al O-DU a través de la interfaz *Open Fronthaul*.

- **Plataforma de infraestructura O-Cloud**: es la plataforma de computación en la nube que alberga los elementos virtualizados de O-RAN, como el Near-RT RIC, O-CU y O-DU (el O-RU aún no está virtualizado). O-Cloud gestiona los recursos y la carga de trabajo mediante interfaces O2 y proporciona soporte para la orquestación de la infraestructura.

Como se ha podido ver, en esta nueva arquitectura entran muchos tipos de interfaces, por lo que a continuación se comenta brevemente de qué se encarga cada una en el contexto 4G/5G:

Interfaz	Descripción
A1	Se encuentra entre el Non-RT RIC y el Near-RT RIC. Soporta gestión de políticas, información de enriquecimiento y gestión de modelos de AI.
O1	Utilizada por el SMO para la gestión de las funciones de red O-RAN.
O2	Se encuentra entre el SMO y O-Cloud, proporcionando servicios de gestión de despliegue y de infraestructura.
E1	Entre nodos O-CU-CP y O-CU-UP, reutilizando principios de 3GPP para interoperabilidad.
E2	Conecta el Near-RT RIC con un nodo E2 (como gNB o eNB). Explicado en detalle en la sección 3.3.3.
Open Fronthaul Interface	Conecta el O-DU y el O-RU, incluyendo planos de sincronización y gestión.
F1-c	Conecta O-CU-CP y O-DU para gestión del control y transmisión de datos.
F1-u	Entre O-CU-UP y O-DU, gestiona el plano de usuario.
NG-c	Conecta O-CU-CP con AMF en el 5GC.
NG-u	Transmite datos de usuario entre O-CU-UP y UPF del 5GC.
X2-c	Transmisión de información del plano de control entre eNB y gNB en redes mixtas.
X2-u	Transmisión de información del plano de usuario entre eNB y gNB en redes mixtas.
Xn-c	Transmisión de control entre gNBs en 3GPP 5G, también utilizado en O-RAN.
Xn-u	Permite la transmisión de datos de usuario entre gNBs en 5G.
Uu	Conecta el UE con el eNB o gNB.
Y1	Proporciona a consumidores suscripciones o solicitudes de análisis de RAN del Near-RT RIC.
R1	Interfaz basada en servicios entre las *rApps* y el *framework* del Non-RT RIC.

Tabla 3.4: Interfaces clave en la arquitectura O-RAN

Una vez analizada la arquitectura O-RAN con la que se va a trabajar, es posible identificar una serie de beneficios clave [45] que esta propuesta tecnológica aporta.

Una de las principales ventajas es la reducción de costes, tanto de capital (CAPEX) como operativos (OPEX). Al basarse en interfaces *open-source*, O-RAN evita la dependencia de un único proveedor, lo que permite integrar equipos de distintos fabricantes. Esta apertura favorece un entorno más competitivo y reduce los costes asociados a licencias y a soluciones propietarias. Además, el uso de hardware y software de código abierto acelera la innovación y fomenta un ecosistema más dinámico.

En términos de eficiencia y rendimiento de red, O-RAN introduce mecanismos avanzados de automatización que permiten gestionar los recursos de manera inteligente, con mínima intervención humana. Su capacidad de control en tiempo real permite optimizar de forma continua el rendimiento de la red, lo que se traduce en una mejor experiencia para el usuario final, incluso en escenarios de alta complejidad. Funcionalidades como el balanceo de carga, la gestión de la movilidad o el ahorro energético pueden adaptarse dinámicamente mediante la colaboración entre los distintos RICs.

Por último, otro beneficio destacable es la incorporación ágil de nuevas funciones. Al estar construida sobre una infraestructura nativa de la nube, se facilita la implementación de nuevas capacidades mediante simples actualizaciones de software.

3.3.2. Inteligencia en O-RAN 5G: RIC

La arquitectura O-RAN introduce inteligencia distribuida en la red a través del RIC. Este elemento puede ser de dos tipos según los requisitos de latencia y tiempo de respuesta:

Non-RT RIC

El Non-RT RIC dentro del marco SMO se estructura a través de diversas funciones lógicas (véase la figura 3.12). Estas permiten la gestión eficiente de servicios y *rApps*, abarcando desde la configuración y el acceso a información de rendimiento, hasta la exposición de servicios y la facilitación de la comunicación.

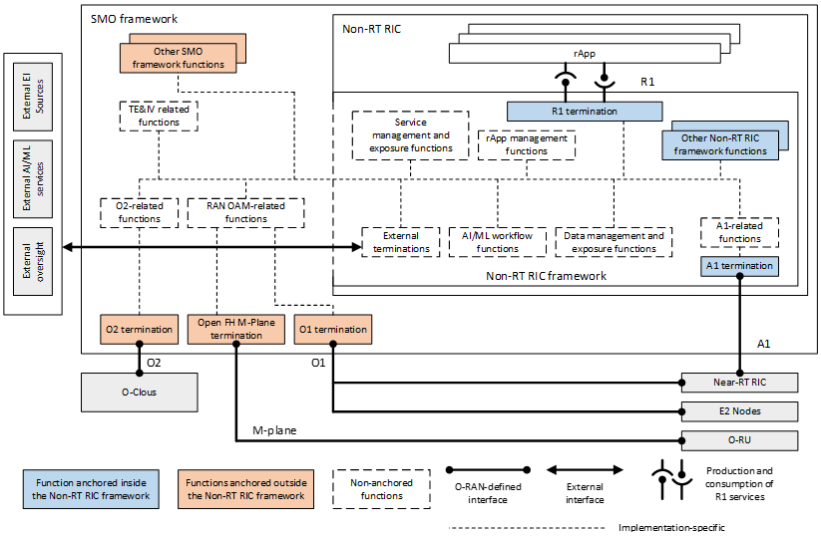


Figura 3.12: Arquitectura del Non-RT RIC [7].

Algunos de los bloques de funciones que se tienen en este RIC son los siguientes:

- ***Service Management and Exposure Functions***
  - Habilita funciones como el registro de servicios, descubrimiento de servicios, notificaciones de servicios, autorización, autenticación, soporte de comunicación.
  - Permite que consumidores externos descubran y utilicen servicios expuestos por el Non-RT RIC.
- ***rApp Management Functions***
  - Habilita la gestión de las *rApps* dentro del marco del SMO/Non-RT RIC.
  - Permite configurar las *rApps* para acceder a información de fallos y rendimiento reportada por ellas y facilitar la funcionalidad de registros (*logging*).
  - Usa la interfaz R1 para proporcionar un acceso coherente a estas funcionalidades para todas las *rApps*, independientemente de su proveedor.
- ***AI/ML Functions***
  - Soporta el entrenamiento, validación, despliegue y gestión del ciclo de vida de modelos de AI/ML.

- ***Data Management and Exposure Functions***
  - Recoge, almacena y expone datos de múltiples fuentes incluyendo el SMO, el Near-RT RIC y los nodos E2.
  - Proporciona preprocesamiento y transformación de datos para consumo de modelos de AI/ML.
  - Garantiza la gobernanza de datos, control de acceso y gestión de metadatos.
- **A1 *Interface***
  - Gestiona la interfaz A1 hacia el Near-RT RIC.
  - Transmite políticas, modelos de ML e información de enriquecimiento desde el Non-RT RIC al Near-RT RIC.
  - Soporta mecanismos de retroalimentación y monitoreo de operaciones relacionadas con A1.
- **R1 *Interface***
  - Proporciona la interfaz entre el Non-RT RIC y las *rApps*.
  - Habilita comunicaciones seguras, registro de servicios y mecanismos de solicitud/respuesta.

Como se ha podido ver, el Non-RT RIC gestiona los servicios/tareas dentro de la red y las *rApps*, facilitando la monitorización del rendimiento y la integración de modelos de AI/ML, sin establecer una conexión directa con los nodos E2. Por otro lado, también interactúa con el Near-RT RIC a través de la interfaz A1, transmitiendo las políticas y datos de enriquecimiento que ha adquirido. Estos datos permiten optimizar la gestión y el control en tiempo real de los nodos E2 de la red.

Near-RT RIC

Como se mostró en la figura 3.11, el Near-RT RIC tiene conexión por medio de 4 interfaces distintas al SMO, al Non-RT RIC, los consumidores y el nodo E2 (gNB). En esta sección, se describen algunas de las funciones que tiene en su interior, y que se aplican por medio de las interfaces descritas. Estas funcionalidades se pueden ver en la figura 3.13.

- **Base de datos y *Service Data Layer* (SDL)**
  - La base de datos proporciona almacenamiento persistente y acceso eficiente a datos relacionados con la O-RAN y los UE.
  - El SDL permite que las *xApps* se suscriban, lean y escriban datos en la base de datos de manera consistente.

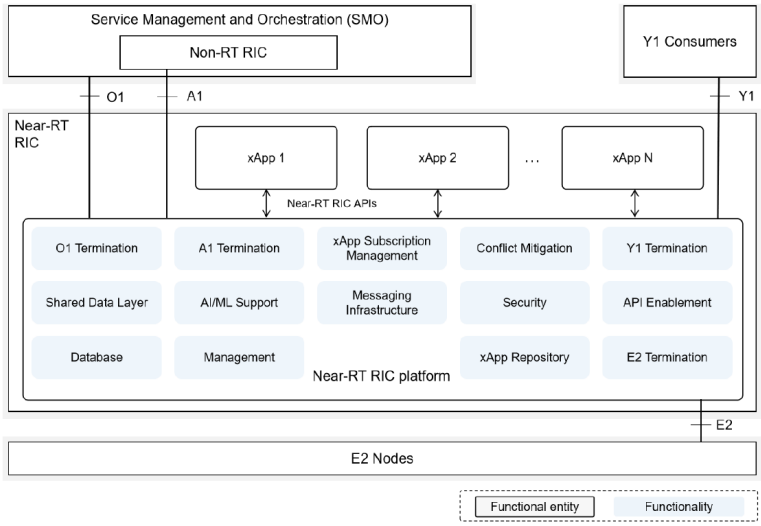


Figura 3.13: Arquitectura del Near-RT RIC [6].

- ***xApp Suscription Management***
  - Administra las suscripciones de las *xApps* a los datos de E2.
  - Consolida suscripciones duplicadas para reducir redundancias y mejorar la eficiencia.
  - Controla el acceso a los datos y permite auditoría de las suscripciones activas.
- ***Conflict Mitigation***
  - Detecta solapamiento de mensajes entre diferentes *xApps*.
  - Resuelve conflictos para mantener la coherencia en las configuraciones de red.
- ***Messaging Infrastructure***
  - Facilita el descubrimiento, registro y eliminación de los *endpoints*.
  - Provee robustez ante pérdida de datos y asegura manejo correcto de mensajes obsoletos.
- ***Security***
  - Protege los datos sensibles de la red ante posibles abusos o comportamientos maliciosos de las *xApps*.
- ***Operations, Administration and Maintenance* (OAM)**
  - Engloba funciones de gestión de fallos, configuración, contabilidad, rendimiento y seguridad.

  - Incluye soporte para *logging*, trazas, análisis y monitoreo de métricas.
  - Utiliza la interfaz O1 para comunicación con el SMO.

- ***Interfaces***
  - **E2**: gestiona la conexión *Stream Control Transmission Protocol* (SCTP) con los nodos E2 y el intercambio de mensajes *E2 Application Protocol* (E2AP).
  - **A1**: facilita la comunicación con el Non-RT RIC para el intercambio de políticas y datos de enriquecimiento.
  - **O1**: establece conexión con el SMO para tareas relacionadas con la gestión de red.
  - **Y1**: provee datos analíticos de O-RAN a consumidores externos mediante la interfaz Y1.

- ***API Enablement***
  - Soporta el registro, descubrimiento y autenticación de APIs para las *xApps*.
  - Administra eventos, suscripciones y compatibilidad entre servicios y *xApps*.

- ***AI/ML Support***
  - ***Data Pipelining***: Procesa y prepara datos para entrenar modelos de AI/ML.
  - ***Model Management***: Almacena, versiona y despliega modelos de AI/ML.
  - ***Training***: Entrena modelos directamente dentro del entorno del RIC.
  - ***Inference***: Ejecuta inferencias basadas en los modelos entrenados.

- ***xApp Repository Function***
  - Mantiene una lista de *xApps* candidatas para la terminación A1, permitiendo el envío de políticas A1 o actualizaciones de política según el tipo de política y las políticas del operador.
  - Mantiene los tipos de políticas soportados en el Near-RT RIC, basados en los tipos de política que admiten las *xApps* registradas y las políticas del operador.
  - Realiza control de acceso de las *xApps* a los tipos A1-E1 solicitados, conforme a las políticas definidas por el operador.

Después de haber revisado las funcionalidades clave del Near-RT RIC y sus interfaces, es importante destacar que la interfaz E2 juega un papel central en la arquitectura global del sistema. Mientras que otras interfaces permiten la comunicación con el SMO, el Non-RT RIC y los consumidores, la interfaz E2 es la que establece la conexión directa con la O-RAN, permitiendo la gestión y control en tiempo real de los nodos de la red, como las gNB. Dado que la O-RAN se encarga de la gestión de los recursos de radio, profundizar en la interfaz E2 es esencial para entender cómo el Near-RT RIC interactúa con la infraestructura de red para la toma de decisiones en tiempo real.

3.3.3. Interfaz E2: Enlace entre RIC y O-RAN

Es una interfaz abierta entre el Near-RT RIC y los nodos E2 como la gNB. Permite al Near-RT RIC:

- Controlar la gestión de recursos de radio.
- Recoger métricas de la O-RAN (de forma periódica o por eventos).

Para llevar a cabo las tareas mencionadas, utiliza identificadores estándar de 3GPP para gNB y clases de QoS. Además, en cuanto a los UEs, introduce un *User Equipment-Identification* (UE-ID) como un identificador uniforme que protege la privacidad de estos.

Lógicamente, la interfaz se compone de la pila de protocolos mostrada en la figura 3.14. Se tiene IP, sobre él SCTP, y sobre este:

- ***E2 Application Protocol* (E2AP)**: protocolo de control base para la coordinación entre el Near-RT RIC y los nodos E2.
- ***E2 Service Model* (E2SM)**: modelo que define funciones específicas como recopilación de métricas o control RAN. Los mensajes E2AP contienen los mensajes E2SM.

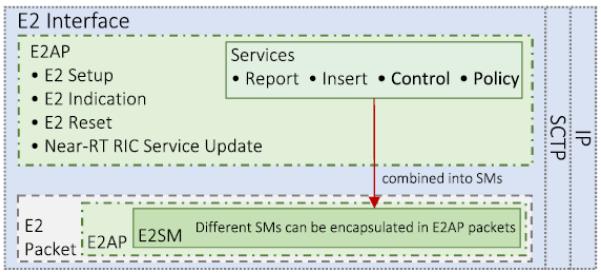


Figura 3.14: Pila de protocolos de la interfaz E2 [46].

Funcionamiento

Mediante el uso de mecanismos de publicación-suscripción, los nodos E2 pueden publicar sus datos y las *xApps* pueden suscribirse a ellos a través de la interfaz E2. Esto permite separar claramente las capacidades de cada nodo y definir cómo interactúan las *xApps* con la red radio.

En el nivel más bajo, el E2AP se encarga de gestionar la configuración, el reinicio y la notificación de errores de la interfaz E2, así como del intercambio de capacidades entre el nodo E2 y el Near-RT RIC. Un ejemplo de este proceso se muestra en la figura 3.15. Primero, se establece una conexión SCTP entre el nodo E2 y el Near-RT RIC, el cual conoce de antemano la dirección IP y el puerto de la terminación E2 del Near-RT RIC. A continuación, el nodo E2 envía una solicitud de configuración (*E2 Setup Request*) en la que comunica las funciones RAN que soporta, junto con su configuración e identificadores. El Near-RT RIC procesa esta información y responde con un mensaje de configuración (*E2 Setup Response*), estableciendo así la conexión.

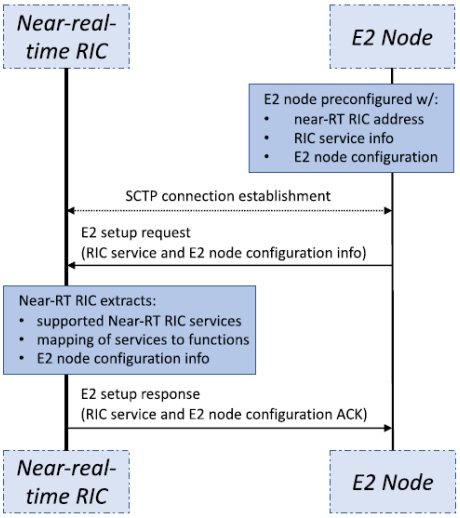


Figura 3.15: Establecimiento de sesion E2 [46].

Una vez configurada la interfaz, el E2AP habilita cuatro servicios fundamentales que pueden combinarse de distintas formas para crear los modelos de servicio E2SM, los cuales se usarán con las posteriores *xApps*:

- ***E2 Report***: para el envío periódico o bajo demanda de métricas y datos de telemetría.
- ***E2 Insert***: para notificar eventos (como *handovers*) desde el nodo E2 hacia el Near-RT RIC.

- ***E2 Control***: para que el Near-RT RIC pueda enviar comandos de control directamente al nodo.
- ***E2 Policy***: para que el nodo ejecute políticas definidas por el Near-RT RIC de forma autónoma tras una suscripción.

El mensaje de modelo de servicio se transmite en el *payload* de los mensajes E2AP codificado usando *Abstract Syntax Notation One* (ASN.1). Actualmente, hay 6 tipos de modelos de servicio [47]:

- ***E2 Service Model Network Interface* (E2SM-NI)**: expone servicios de funciones RAN relacionados con:
  - Exposición de interfaces de red.
  - Modificación de mensajes entrantes y salientes.
  - Ejecución de políticas que alteran el comportamiento de la red.
- ***E2 Service Model Key Performance Measurement Monitor* (E2SM-KPM) v1**:
  - Reportes periódicos del rendimiento de celdas en O-DU.
  - Reportes de rendimiento por celda o UE en O-CU-CP.
  - Reportes del rendimiento de portadoras en O-CU-UP.
- ***E2 Service Model Key Performance Measurement Monitor* (E2SM-KPM) v2**: amplía las capacidades de la versión anterior (versión 1) incluyendo:
  - Exposición de mediciones disponibles.
  - Reportes periódicos basados en suscripciones del Near-RT RIC.
- ***E2 Service Model RAN Control* (E2SM-RC)**:
  - Información de control RAN y contexto de UE.
  - Modificación e inicio de procesos y mensajes de control RAN.
  - Ejecución de políticas que afectan el comportamiento de control RAN.
- ***E2 Service Model Cell Configuration and Control* (E2SM-CCC)**:
  - Exposición de información de configuración a nivel nodo y celda.
  - Inicio de configuraciones y controles sobre parámetros de nodo y celda, como la selección del ancho de banda.
- ***E2 Service Model Logical Link Control* (E2SM-LLC)**:

- Exposición de información de capas L1 y L2.
- Inicio del control de capas inferiores.

De entre los modelos de servicio descritos anteriormente, los más comúnmente utilizados en entornos reales son aquellos orientados al monitoreo y al control, específicamente E2SM-KPM y E2SM-RC. A continuación, se presentan ejemplos representativos de procedimientos asociados a cada uno de estos modelos, con sus respectivos flujos de mensajes.

Es necesario llevar a cabo una suscripción por parte de la *xApp*. Este proceso permite al Near-RT RIC indicar qué tipo de información desea recibir o qué acciones desea controlar en el nodo E2. En la figura 3.16 se muestra el procedimiento general de suscripción de una *xApp*.

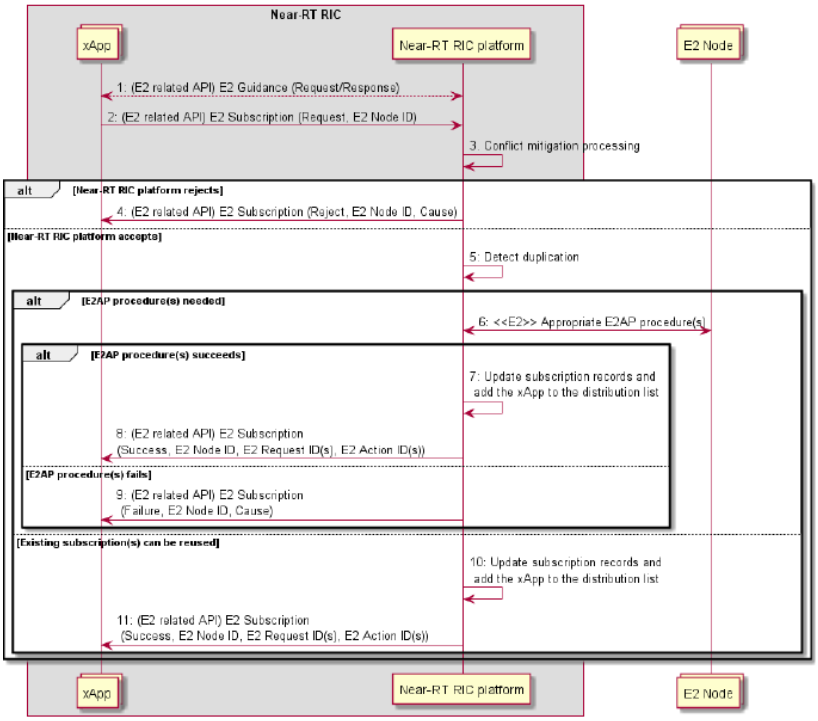


Figura 3.16: Suscripción de una *xApp* con E2 [6].

Este procedimiento comienza cuando la *xApp*, previamente autorizada, identifica la necesidad de recibir ciertas métricas o llevar a cabo ciertos eventos en un nodo E2 concreto. En este punto, la *xApp* puede (opcionalmente) solicitar orientación al Near-RT RIC para validar su solicitud. A continuación, envía una petición de suscripción que especifica la función RAN objetivo, el evento que desencadena la acción y la lista de acciones que desea realizar.

Una vez recibida la solicitud, el Near-RT RIC lleva a cabo los siguientes pasos generales:

1. **Evaluación de la solicitud:** valida el formato y contenido de la suscripción y verifica si la *xApp* está autorizada.
2. **Gestión de conflictos:** comprueba si la nueva suscripción entra en conflicto con otras ya existentes y, si es necesario, aplica mecanismos de mitigación.
3. **Verificación de duplicados:** determina si ya existe una suscripción activa con los mismos parámetros.
4. **Procedimiento E2AP:** si no hay duplicados, el Near-RT RIC inicia un procedimiento E2AP con el nodo E2. Este procedimiento permite configurar la suscripción en el nodo mediante el intercambio de mensajes estandarizados (como *RIC Subscription Request* y *RIC Subscription Response*), dentro de los cuales irá encapsulada información E2SM-KPM o E2SM-RC.
5. **Actualización de registros:** si el procedimiento es exitoso, se actualizan los registros de suscripción y se asocia la *xApp* a los identificadores correspondientes.
6. **Respuesta a la *xApp*:** se notifica el éxito o fallo del proceso; en caso de éxito, se incluyen los identificadores de la suscripción y acciones asignadas.

Este proceso general de suscripción se adapta según el tipo de servicio o función E2 que la *xApp* desea utilizar. Por ejemplo, en el caso de la monitorización de métricas, se utiliza el modelo de servicio E2SM-KPM. En ese contexto, el nodo E2 anuncia las métricas disponibles para exposición mediante descripciones estructuradas en este tipo de modelo. La *xApp*, al estar interesada en ciertas métricas, envía su solicitud de suscripción al Near-RT RIC, especificando qué indicadores de rendimiento desea recibir y bajo qué modalidad: periódica o basada en eventos. Esta solicitud se encapsula en un mensaje *RIC Subscription Request* con contenido específico del E2SM-KPM. Una vez que el nodo E2 acepta la suscripción (vía *RIC Subscription Response*), comienza a enviar las métricas solicitadas en mensajes de tipo *E2 Indication*. Este flujo especializado se ilustra en la figura 3.17.

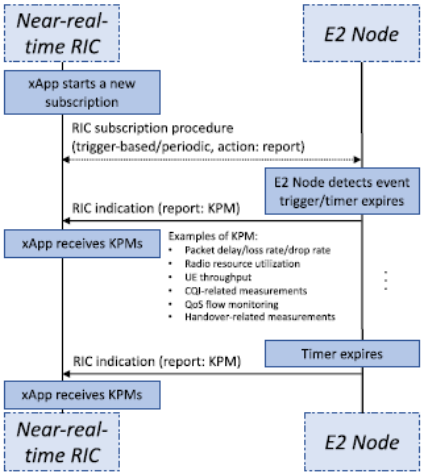


Figura 3.17: Monitorización de métricas con E2 [46].

Por otro lado, una vez desplegada la *xApp*, el control de funciones como la asignación de recursos se realiza a través de un ciclo en el que: primero, la *xApp* recibe información del estado de la red mediante mensajes E2SM-KPM recolectados a través de la interfaz E2 (por ejemplo, PRB solicitados, medidas de latencia y rendimiento), y después computa acciones de control para dirigir al nodo *Distributed Unit* (DU) en la asignación dinámica de recursos, como los PRBs, entre diferentes UEs. Este procedimiento se muestra en la figura 3.18.

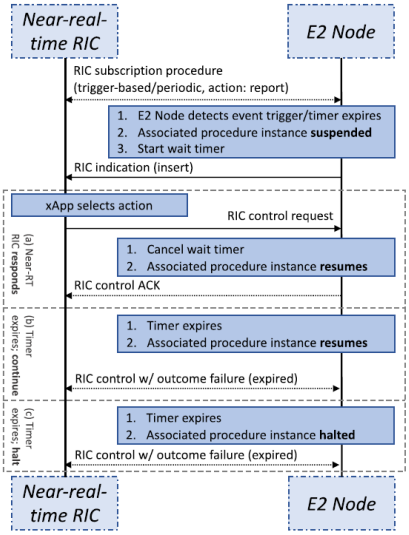


Figura 3.18: Control de recursos con E2 [46].

El procedimiento de subscripción de una *xApp* y los modelos de servicio E2SM-KPM y E2SM-RC son los más importantes, pero existe una gran variedad de procedimientos adicionales. Más información sobre esta interfaz y su comportamiento se puede encontrar en [6] y [46].

Capítulo 4

Planificación y Costes

En este capítulo se detallan las distintas etapas que conforman el desarrollo del proyecto, incluyendo una descripción general de cada una y el tiempo estimado dedicado a su ejecución. Asimismo, se presentan los recursos utilizados, tanto humanos como materiales (hardware y software), junto con una estimación de los costes asociados a cada categoría. Finalmente, se ofrece una visión global del presupuesto total del trabajo, incluyendo un desglose de los costes derivados de las modificaciones incorporadas respecto a la planificación inicial.

4.1. Planificación Temporal

En esta primera sección se presentan las diferentes fases y tareas del proyecto; además de su duración y su posición a lo largo del tiempo disponible.

1. **Fase 1. Propuesta de proyecto**
    - **Planteamiento del proyecto:** elaboración de la propuesta del proyecto en colaboración entre el tutor y el estudiante (20 días).
    - **Presentación y aceptación del proyecto**: exposición formal de la propuesta y aprobación definitiva por parte del centro académico (10 días).
2. **Fase 2. Revisión bibliográfica**
    - **Análisis del estado del arte:** análisis de los principales usos y beneficios de la arquitectura propuesta (O-RAN + RIC), incluyendo ejemplos de proyectos/implementaciones relevantes a nivel internacional. Toma de decisiones sobre las soluciones escogidas. (8 días).
    - **Estudio de los fundamentos teóricos:** comprensión profunda de los conceptos teóricos esenciales relacionados con el tema (15 días).

- **Revisión de fuentes y referencias:** organización, verificación y recopilación final de toda la bibliografía consultada (5 días).

3. **Fase 3. Diseño e implementación**

   - **Análisis de la infraestructura objetivo:** se estudian los requisitos del entorno de red a implementar, definiendo las características hardware y software necesarias (3 días).
   - **Diseño e implementación de una red O-RAN + RIC mediante OAIC:** se lleva a cabo la creación del primer escenario virtual utilizando *OpenAirInterface*. Se configura la infraestructura necesaria para integrar el Near-RT RIC, incluyendo la definición de los elementos funcionales, las conexiones entre componentes y los parámetros de red (IP, interfaces, sincronización, etc.) (5 días).
   - **Diseño e implementación de una red O-RAN + RIC mediante srsRAN:** se lleva a cabo la creación del segundo escenario virtual. Al igual que en caso anterior, se configura la infraestructura necesaria para integrar el Near-RT RIC en la red O-RAN (5 días).

4. **Fase 4. Pruebas y experimentación**

   - **Definición del entorno de pruebas:** se detallan los escenarios experimentales definidos a partir del diseño descrito en la fase anterior. Se establecen los criterios de prueba y se definen los objetivos de evaluación para cada uno de los entornos implementados (2 días).
   - **Validación funcional con OAIC:** se comprueba el funcionamiento básico del sistema, evaluando su rendimiento y comportamiento bajo condiciones controladas. El objetivo es verificar que las funcionalidades principales operan correctamente según lo esperado (8 días).
   - **Experimentación con srsRAN:** se elije esta iniciativa como la más completa, por lo que con ella se realizan pruebas de monitoreo y control de red mediante el Near-RT RIC, evaluando la recopilación de métricas con FlexRIC y ORAN-SC-RIC. Además, se implementa control de PRBs para analizar la capacidad de gestión del sistema. Finalmente, se comparan los resultados de ambas implementaciones del Near-RT RIC para identificar la más eficiente (25 días).

5. **Fase 5. Escritura y entrega de la memoria**

- **Redacción de la memoria:** esta actividad se inicia una vez finalizada la fase de pruebas. Durante su desarrollo, se documentan de forma estructurada y clara los conocimientos adquiridos, así como los resultados obtenidos a lo largo del proyecto (42 días).
- **Entrega de la memoria:** solicitud de evaluación del proyecto a través del correo y entrega de la memoria del estudiante por Prado y la memoria del tutor (1 día).
- **Defensa del proyecto**: elaboración de la defensa del trabajo realizado y exposición ante el Tribunal escogido (10 días).

En la figura 4.1 se presenta de forma ordenada y cronológica el conjunto de actividades del proyecto, detallando su fecha de inicio, de finalización y su duración en días. Esto permite ofrecer una visión general y estructurada de la planificación comentada.

Nombre	Fecha de inicio	Fecha de fin	Duración
TFM Near-RT RIC	16/9/24	10/7/25	288
Fase 1. Propuesta de proyecto	16/9/24	4/12/24	78
Planteamiento del proyecto	16/9/24	5/10/24	20
Presentación y aceptación del proyecto	25/11/24	4/12/24	10
Fase 2. Revisión bibliográfica	5/2/25	14/3/25	38
Análisis del estado del arte	5/2/25	12/2/25	8
Estudio de los fundamentos teóricos	17/2/25	3/3/25	15
Revisión de fuentes y referencias	10/3/25	14/3/25	5
Fase 3. Diseño e implementación	17/3/25	4/4/25	19
Análisis de la infraestructura objetivo	17/3/25	19/3/25	3
Diseño e implementación mediante OAIC	24/3/25	28/3/25	5
Diseño e implementación mediante srsRAN	31/3/25	4/4/25	5
Fase 4. Pruebas y experimentación	7/4/25	18/5/25	39
Definición del entorno de pruebas	7/4/25	8/4/25	2
Validación funcional con OAIC	9/4/25	16/4/25	8
Experimentación con srsRAN	23/4/25	18/5/25	25
Fase 5. Escritura y entrega de la memoria	19/5/25	10/7/25	53
Redacción de la memoria	19/5/25	29/6/25	42
Entrega de la memoria	30/6/25	30/6/25	1
Defensa del proyecto	1/7/25	10/7/25	10

Figura 4.1: Resumen de la temporización de las fases y tareas del proyecto.

Además, se incluye a continuación un diagrama de Gantt (figura 4.2) en el que se representan gráficamente las distintas fases y tareas, así como su distribución a lo largo del periodo previsto. Este tipo de diagrama permite visualizar de forma clara la secuencia temporal de las actividades, su duración, solapamientos y dependencias. Gracias a esta representación, es posible identificar fácilmente los hitos más relevantes y supervisar el avance del proyecto en función de los plazos establecidos.

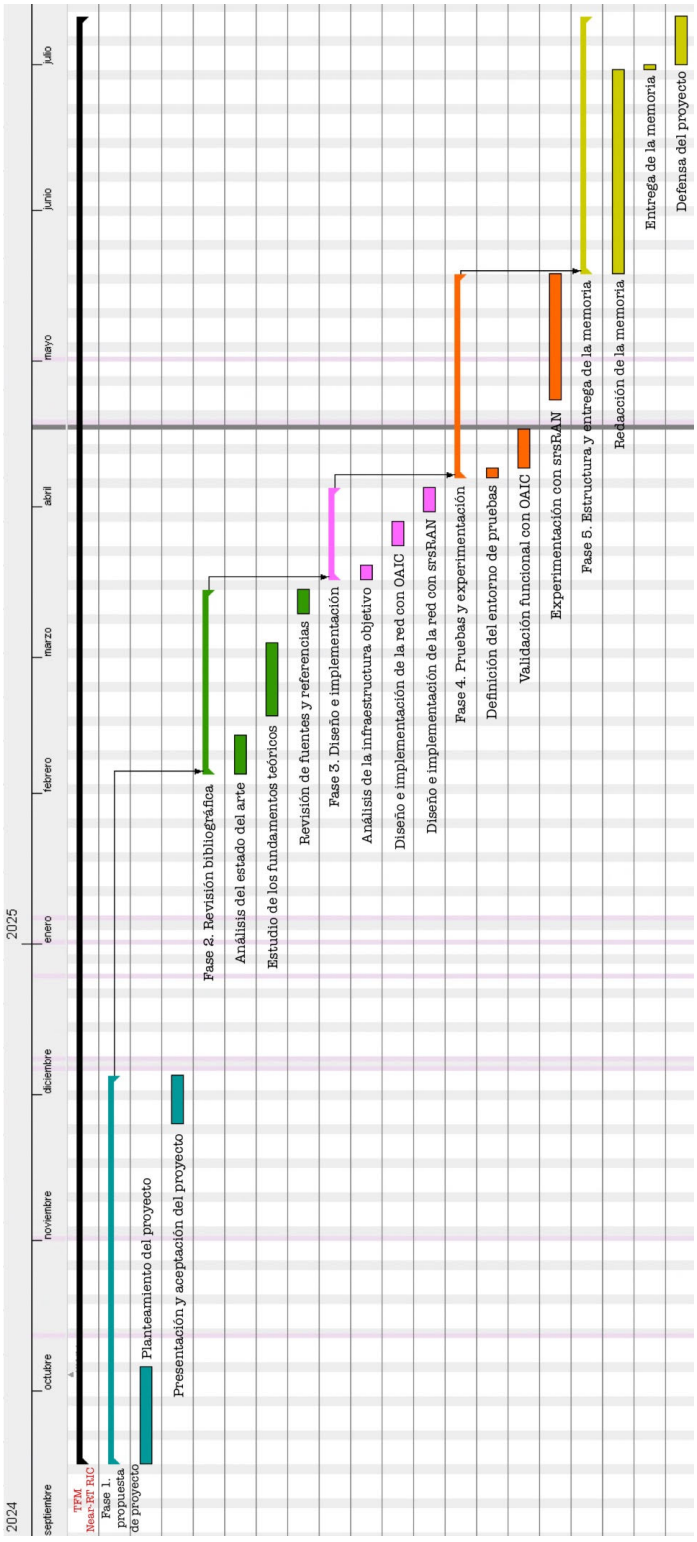


Figura 4.2: Diagrama de Gantt.

4.2. Recursos y Coste

A continuación, se presentan los recursos humanos, hardware y software empleados en el proyecto, junto con el coste total y desglosado a cada uno de ellos.

4.2.1. Recursos Humanos

Dentro de esta categoría de recursos se consideran las personas involucradas en el desarrollo del proyecto, así como el tiempo dedicado por cada uno. Los individuos que han intervenido en la realización final del trabajo son:

- **Alejandra Oliver Boada**. Estudiante del Máster en Ingeniería de Telecomunicación en la Universidad de Granada. Asume el cargo de autora del proyecto.

- **Jorge Navarro Ortiz**. Profesor Titular de Universidad en el Departamento de Teoría de la Señal, Telemática y Comunicaciones. Asume el cargo de tutor del proyecto.

- **Félix Delgado Ferro**. Investigador Predoctoral en *Wireless and Multimedia Networking Lab* - UGR. Asume el cargo de cotutor del proyecto.

El tiempo invertido por el tutor y cotutor se ha destinado a la realización de tutorías con la estudiante, la recopilación de información relevante, y el seguimiento y revisión tanto del desarrollo técnico del proyecto como de la elaboración de la presente memoria.

Respecto a las horas de dedicación por parte de la estudiante, en la tabla que se presenta a continuación (tabla 4.1) se recogen las distintas tareas llevadas a cabo, junto con el número de horas asignado a cada una, y el total del tiempo empleado en la ejecución completa del proyecto.

La tabla 4.2 presenta un resumen del tiempo de dedicación de cada uno de los participantes en el proyecto, acompañado del coste por hora correspondiente según su función o cargo, así como el cálculo del presupuesto total asociado a los recursos humanos que se han necesitado finalmente según las necesidades que han ido surgiendo a lo largo de los meses de búsqueda de información y desarrollo experimental. Esta información permite estimar con mayor precisión el impacto económico del personal involucrado y facilita una planificación más realista de los recursos disponibles.

Tarea	Horas Dedicadas
Planteamiento del proyecto	30
Presentación y aceptación del proyecto	10
Análisis del estado del arte	40
Estudio de fundamentos teóricos clave	80
Revisión de fuentes y referencias	40
Análisis de la infraestructura objetivo	20
Diseño e implementación de red O-RAN + RIC con OAIC	50
Diseño e implementación de red O-RAN + RIC con srsRAN	50
Definición del entorno de pruebas	40
Validación funcional con OAIC	40
Experimentación con srsRAN (monitoreo, y control de PRB)	70
Redacción de la memoria	150
Entrega de la memoria	2
Defensa del proyecto	20
TOTAL: 642 horas	

Tabla 4.1: Tabla resumen de horas dedicadas a las diferentes tareas.

Persona	Horas Trabajadas	Precio/hora (€)	Coste Total (€)
Jorge Navarro Ortiz	20	50	1000
Félix Delgado Ferro	20	35	700
Alejandra Oliver Boada	642	20	12840
TOTAL: 14540 €			

Tabla 4.2: Coste total de recursos humanos.

4.2.2. Recursos Hardware

Como herramienta hardware única se ha empleado un portátil de uso personal modelo ASUS ZenBook 14, cuyas características se observan en la tabla 4.3.

Ha resultado fundamental contar con un ordenador portátil que presentase buenas prestaciones, permitiendo ejecutar de forma simultánea los distintos programas y máquinas virtuales necesarios, sin comprometer el rendimiento ni la velocidad de trabajo. Teniendo en cuenta que la vida útil estimada de un portátil con plena amortización es de cinco años, y que en este proyecto se ha utilizado durante un año, en la tabla 4.4 se detalla el coste proporcional al precio de adquisición del equipo (620,13€).

Modelo	ASUS ZenBook 14
Procesador	AMD Ryzen 7 5700U (8 núcleos-16 hilos/4.3 GHz/12 MB caché)
Tarjeta Gráfica	AMD Radeon R7 Graphics Integrada
Almacenamiento	SSD M.2 NMVe PCIe 3.0 de 512 GB
Memoria RAM	16 GB SO-DIMM LPDDR4x
Sistema Operativo	*Windows* 11

Tabla 4.3: Características del portátil utilizado.

Elemento	**Precio (€)**	**Vida Media**	**Uso**	**Coste Proporcional (€)**
Portátil ASUS ZenBook 14	620,13	5 años	1 año	124,03

Tabla 4.4: Coste total de recursos hardware.

4.2.3. Recursos Software

Los programas y herramientas software empleados son los siguientes:

- **Sistema Operativo *Windows* 11 *Home*** [48]: sistema operativo creado por *Microsoft* y utilizado en el equipo portátil que soportará las redes virtuales configuradas a través de *VirtualBox*.

- ***Oracle Virtual Machine* (VM) *VirtualBox* 7.0.6** [49]: herramienta de virtualización compatible con varias plataformas, diseñada para arquitecturas x86/ amd64. Es un software de código abierto que facilita la creación y gestión de máquinas virtuales, permitiendo la virtualización de diversos sistemas operativos. Las máquinas virtuales se pueden administrar a través de una interfaz gráfica, y ofrecen múltiples opciones de personalización y funcionalidad.
  Esta herramienta ha sido utilizada para configurar las redes virtuales descritas en el capítulo 5.

- **Sistema Operativo *Ubuntu* 20.04 *Long Term Support* (LTS)** [50]: sistema operativo que se utiliza en la máquina virtual donde se implementa la solución OAIC. La etiqueta LTS indica que esta versión goza de una mayor estabilidad, soporte a largo plazo y actualizaciones de seguridad durante un periodo extendido en comparación con las versiones estándar. Aunque la versión más reciente de LTS es la 22.04, se opta por usar la 20.04 para asegurar que se tenga soporte a los incidentes que puedan ocurrir en la fase de experimentación; además, se infiere más diversidad.

- **Sistema Operativo *Ubuntu* 22.04 LTS** [51]: sistema operativo que se instala en la máquina virtual con la implementación de la solución 2, srsRAN.

- ***Grafana* 10.1.5** [52]: plataforma abierta y de código libre que permite visualizar datos a través de gráficas y tablas. Puede instalarse en el equipo o utilizarse vía navegador en su versión *Grafana Cloud*. Su interfaz permite configurar paneles con gráficos que muestren los datos que se deseen. Es útil para interpretar información de diversas fuentes, como bases de datos como *Prometheus*,*My Structured Query Language* (MySQL) o *InfluxDB*.
  En este proyecto, se emplean *Grafana* e *InfluxDB* para crear gráficos con los resultados obtenidos en los escenarios virtuales.

- ***Wireshark* 4.4.5** [53]: herramienta gratuita para el análisis de protocolos que captura paquetes de red y permite un análisis detallado de las comunicaciones. Se utiliza para revisar los mensajes intercambiados y diagnosticar problemas en la red.
  Se emplea para analizar la comunicación en las redes virtuales creadas.

- ***Overleaf*** [54]: editor colaborativo en línea basado en la nube para redactar y editar documentos en LaTeX. Es usado para escribir y publicar documentos científicos, ofreciendo plantillas de revistas y un historial de cambios para facilitar la gestión del documento, además de permitir la colaboración en tiempo real.
  Este editor se utiliza para la redacción de esta memoria.

- ***GanttProject* 3.3.3316** [55]: software libre multiplataforma que facilita la planificación de proyectos mediante la creación de gráficos Gantt. Ha sido utilizado para crear el diagrama de Gantt mostrado en la figura 4.2.

- ***Zotero* 7.0.15** [56]: gestor bibliográfico gratuito y de código abierto, compuesto por una extensión para navegadores que permite guardar rápidamente información de sitios web y una aplicación de escritorio para organizar, citar y gestionar referencias.
  Se ha utilizado en este proyecto para generar la bibliografía, gracias a su capacidad para exportar en formatos como BibTeX.

La mayoría de las herramientas software utilizadas a lo largo del proyecto son de código abierto, lo cual ha permitido minimizar significativamente los costes asociados. El único gasto adicional en este ámbito corresponde a la licencia del sistema operativo *Windows 11 Home*, instalado en el equipo portátil descrito en la sección 4.2.2.

Cabe señalar que el dispositivo incluía originalmente la versión *Windows 10 Home*, pero con el objetivo de prolongar su vida útil y garantizar una mejor amortización a largo plazo, se llevó a cabo una actualización de sistema. El coste de esta licencia asciende a 145€. Un desglose detallado de los costes asociados al software se muestra en la tabla 4.7:

Software	**Precio (€)**
Sistema Operativo *Windows* 11 *Home*	145
Oracle VM *VirtualBox* 7.0.6	0
Sistema Operativo *Ubuntu* 20.04 LTS	0
Sistema Operativo *Ubuntu* 22.04 LTS	0
Repositorio srsRAN 5G	0
Repositorio srsRAN 4G	0
Repositorio OAIC	0
Grafana 10.1.5	0
Wireshark 4.4.5	0
Overleaf	0
GanttProject 3.3.3316	0
Zotero 7.0.15	0
TOTAL: 145 €	

Tabla 4.5: Coste Total de recursos software.

4.2.4. Estimación de Costes

Seguido de esto, se realiza una estimación del coste total del proyecto, el cual se detalla en la tabla 4.6:

Recursos	**Coste Total (€)**
Humanos	9660
Software	145
Hardware	124,03
TOTAL: 9929.03 €	

Tabla 4.6: Coste total del proyecto.

Así, el coste total de proyecto es de **9929.03 €**.

4.2.5. Modificaciones en la Planificación Inicial

En el mes de Junio de 2025 se pensó en utilizar una serie de equipos disponibles en la Universidad para hacer una serie de pruebas con instrumentación real, con el fin de que no fuese todo virtualizado y el proyecto se acercase un poco más a la realidad encontrada en proyectos internacionales.

Los equipos utilizados tenían un periodo de amortización de unos 5 años, y fueron usados únicamente 1 semana. En la tabla 4.7 se deja un resumen de su coste en caso de interés.

Elementos	Precio (€)
2x NUC 13ANH (i7, 16 GB de *Random Access Memory* (RAM), 512 GB de *Solid State Drive* (SSD))	2x900=1800
PC	2000
Tarjeta Quectel RM500Q-GL + placa de desarrollo RMU500EK	600
Caja de Faraday LBX1000 *Radiofrequency* (RF) Shielded Test Enclosure	2500
USRP-2901 (70MHz to 6GHz, 56 MHz Bandwidth, 2 Input/Output, USB 2.0, USB 3.0)	3600
TOTAL: 10500 €	

Tabla 4.7: Coste Total de los Equipos del Escenario Real.

El coste de este equipamiento, proporcional a un uso de 1 semana, sería de unos 40 €aproximadamente. Añadido al presupuesto mostrado en la sección 4.2.4, finalmente se tiene un coste de 11345.03 €.

Capítulo 5

Diseño e Implementación

En el presente capítulo se exponen los pasos seguidos en el diseño, despliegue y configuración del escenario 5G O-RAN + RIC con las dos implementaciones escogidas.

5.1. Contexto Práctico

La figura 5.1 muestra el escenario general del sistema que se va a desarrollar. Más adelante, cada implementación tendrá sus particularidades en cuanto a direccionamiento, conexiones internas, etc. Como se puede observar, ha sido completamente desplegado dentro de una máquina virtual basada en VirtualBox 7.0.6 [49]. Esta decisión permite facilitar la portabilidad del entorno y el aislamiento del mismo respecto al sistema anfitrión, lo que garantiza un mayor control sobre la instalación, configuración y pruebas en él mismo.

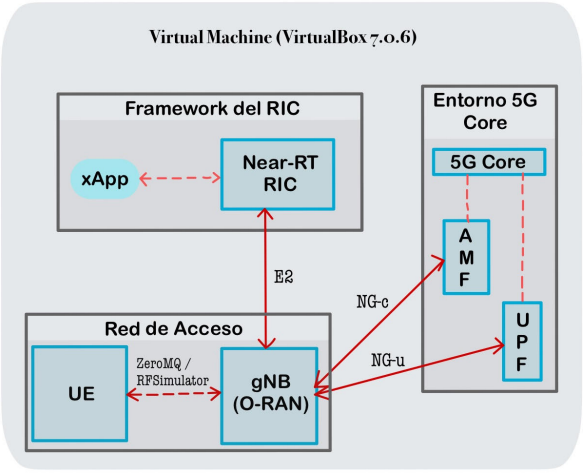


Figura 5.1: Escenario general de implementación.

Dentro de la máquina virtual, se pueden distinguir tres subsistemas principales:

- ***Framework* del RIC**: incluye el módulo Near-RT RIC y varias *xApps* (aplicaciones encargadas de realizar funciones de monitorización y/o control sobre la red) dependientes de la implementación de Near-RT RIC con la que se esté trabajando en cada momento. La comunicación entre el Near-RT RIC y las *xApps* se realiza de forma interna mediante una interfaz lógica, mientras que la conexión de éste con el gNB se establece a través de la interfaz E2.

- **Entorno 5G *Core***: está compuesto por los elementos fundamentales del núcleo de red 5G, como el SMF, PCF, AMF y UPF (estos dos últimos con conexión directa a la red de acceso mediante las interfaces NG-c y NG-u, correspondientes al plano de control y al plano de usuario, respectivamente. Estos módulos conforman la infraestructura central encargada de gestionar el acceso y la movilidad del sistema 5G.

- **Red de Acceso**: está conformada por el nodo gNB y el UE. La gNB actúa como intermediario entre el usuario y el núcleo de red, y se comunica con un UE simulado (con posibilidad de comunicarse con más de un usuario) a través de herramientas como ZeroMQ o *RFSimulator*, con las cuales se puede simular un canal físico sin requerir hardware como tal. Por otro lado, mantiene enlaces con el *core* 5G mediante las interfaces NG-c y NG-u.

Este entorno, completamente virtualizado, permite la evaluación funcional de los distintos componentes del sistema y su integración con las aplicaciones del Near-RT RIC, brindando un marco controlado para pruebas, validación y desarrollo de nuevas funcionalidades de red.

5.2. Despliegue de la arquitectura funcional de OAIC

A continuación, se describe el proceso seguido en la elaboración de un escenario virtual en el que se está utilizando la solución propuesta por OAIC siguiendo la guía [57] con ciertos cambios.

5.2.1. Instalación de la VM y dependencias

Como se indicó en la sección 5.1, toda la solución se ejecuta sobre una máquina virtual con Ubuntu 20.04 LTS [50]. Una vez creada la máquina, es necesario instalar una serie de dependencias fundamentales para compilar y ejecutar los distintos componentes del entorno OAIC.
A continuación, se enumeran aquellas más generales:

```
sudo apt install git vim tree net-tools libsctp-dev python3.8
cmake-curses-gui libpcre2-dev python-dev build-essential cmake
libfftw3-dev libmbedtls-dev libboost-program-options-dev
libconfig++-dev libtool autoconf python3-pip curl bison flex iperf
unzip
```

Figura 5.2: Instalación de librerías generales.

Instalación de SWIG

El entorno OAIC utiliza *Simplified Wrapper and Interface Generator* (SWIG) para permitir la interacción entre código C/C++ y Python en componentes como el FlexRIC. En particular, FlexRIC requiere SWIG versión 4.1 o superior para generar correctamente los enlaces con su API:

```
# Clona el repositorio de SWIG desde GitHub y cambia a su directorio
git clone https://github.com/swig/swig.git
cd swig

# Cambia a la rama de la versión 4.1
git checkout release-4.1

# Ejecuta el script de configuración de SWIG
./autogen.sh

# Configura la instalación de SWIG con el prefijo /usr/
./configure --prefix=/usr/

# Compila SWIG utilizando 8 hilos
make -j8

# Instala SWIG en el sistema
sudo make install
```

Figura 5.3: Instalación de SWIG desde código fuente.

Compilador GCC

FlexRIC no es compatible con versiones antiguas del compilador *gcc*, como la versión *gcc-11*, ya que contiene errores de compilación relacionados con la gestión de punteros y estructuras en el C++ moderno. Por este motivo, es necesario instalar *gcc-12* o *gcc-13* con:

```
sudo apt install gcc-10
sudo apt install g++-10
```

Figura 5.4: Instalación del compilador GCC.

Instalación de Docker

Dado que el entorno OAIC despliega el núcleo de la red 5G utilizando contenedores Docker, es necesario tener configurada esta herramienta en el sistema.

```
# Instala putty, ca-certificates y gnupg necesarios
sudo apt install -y putty ca-certificates gnupg

# Crea el directorio para almacenar las claves del repositorio
sudo install -m 0755 -d /etc/apt/keyrings

# Descarga y guarda la clave pública de Docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
    dearmor -o /etc/apt/keyrings/docker.gpg

# Cambia los permisos de la clave para que sea accesible
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Añade el repositorio oficial de Docker para Ubuntu
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/
    keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(.
    /etc/os-release && echo $VERSION_CODENAME) stable" | sudo tee /etc
    /apt/sources.list.d/docker.list > /dev/null

# Actualiza la lista de paquetes con el nuevo repositorio
sudo apt-get update

# Instala Docker Buildx y Docker Compose plugin
sudo apt install -y docker-buildx-plugin docker-compose-plugin
```

Figura 5.5: Instalación de Docker y configuración desde su repositorio oficial.

RFSimulator

RFSimulator es una herramienta que emula enlaces de radio en entornos virtualizados, permitiendo la transmisión entre entidades como gNBs y UEs sin necesidad de hardware físico y mediante la creación de un *socket Transmission Control Protocol (TCP)*. En este caso, su instalación se va a realizar junto con los contenedores Docker oficiales de OAI [27].

5.2.2. Implementación de la red

OAI 5G *Core*

En primer lugar, se debe instalar y configurar el *core* de la red 5G. Para ello, se hace uso de la solución *OpenAirCN* [26], desarrollada por la comunidad de *OpenAirInterface*. Esta solución implementa las funciones principales del núcleo 5G como contenedores Docker:

```
# Descarga el archivo zip con los recursos del tutorial OAI desde el repositorio oficial.
wget -O ~/oai-cn5g.zip https://gitlab.eurecom.fr/oai/openairinterface5g/-/archive/develop/openairinterface5g-develop.zip?path=doc/tutorial_resources/oai-cn5g

# Descomprime el archivo zip descargado en el directorio home del usuario.
unzip ~/oai-cn5g.zip

# Mueve los archivos del directorio descomprimido a un nuevo directorio llamado oai-cn5g.
mv ~/openairinterface5g-develop-doc-tutorial_resources-oai-cn5g/doc/tutorial_resources/oai-cn5g ~/oai-cn5g

# Elimina el archivo zip descargado y el directorio intermedio creado durante la descompresión.
rm -r ~/openairinterface5g-develop-doc-tutorial_resources-oai-cn5g ~/oai-cn5g.zip

# Cambia al directorio recién creado que contiene los recursos de OAI.
cd ~/oai-cn5g

# Ejecuta el comando Docker Compose para descargar las imágenes necesarias para la configuración de OAI.
sudo docker compose pull
```

Figura 5.6: Instalación del OAI 5G *Core Network* (CN).

Como se observa, una vez descargado el entorno, se ejecuta el comando `docker compose pull`, que descarga las imágenes Docker correspondientes a cada servicio definido en el archivo `docker-compose.yml`. Las imágenes actúan como plantillas que contienen el sistema base y las dependencias necesarias para crear los contenedores.

Para lanzar todos los servicios, se utiliza el comando `sudo docker compose up -d`. Esto crea y arranca los contenedores, que son instancias ejecutables generadas a partir de las imágenes previamente descargadas. Para comprobar que los contenedores están en ejecución, se puede usar `sudo docker ps -a`.

Entre los servicios desplegados se incluyen:

- **MySQL**: como base de datos principal.
- **IMS**: para la gestión del subsistema multimedia IP.
- **UDR**, **UDM**, **AUSF** y **NRF**: responsables de la gestión de datos de usuario, autenticación y descubrimiento de funciones.
- **AMF** y **SMF**: encargados de la gestión de acceso y sesión, respectivamente.

- **UPF**: implementa la función de plano de usuario.
- ***External Data Network* (EXT-DN)**: simula una red externa (por ejemplo, acceso a Internet).

Cada servicio se configura con una dirección IP fija dentro del rango `192.168.70.128/26`, lo cual permite una comunicación controlada y predecible entre contenedores dentro del entorno definido por Docker. Además, se establece la asignación de una IP en el rango 10.0.0.0/24 a los usuarios que se conecten a la red.

Red de Acceso OAI 5G y UE

En segundo lugar, se debe implementar la red de acceso. *OpenAirInterface* ofrece una solución de red de acceso *open-source* (muy cercana a O-RAN) que permite ajustar múltiples parámetros relacionados con la gNB, los UE y el simulador de canal *RFSimulator*.

El primer paso es descargar el repositorio [30] y compilarlo:

```
# Clona el repositorio de OpenAirInterface 5G en el directorio ~/oai
git clone https://github.com/openaicellular/openairinterface5G.git ~/oai

# Cambia al directorio del repositorio clonado
cd ~/oai

# Cambia a la rama específica para el taller OAI 2024
git checkout oaic_workshop_2024_v1

# Accede al directorio de los archivos de compilación de OpenAirInterface
cd ~/oai/cmake_targets/

# Ejecuta el script de construcción para la configuración de simulación con soporte para gNB y NR UE, incluyendo E2
./build_oai -I -w SIMU --gNB --nrUE --build-e2 --ninja
```

Figura 5.7: Descarga y preparación de la red de acceso OAI 5G.

Como se puede ver, se ejecuta el *script build_oai* y se le pasan las siguientes opciones:

- **-I**: esta opción indica que se debe realizar una instalación completa del entorno, asegurando que todas las dependencias necesarias sean gestionadas.
- **-w SIMU**: define el modo de trabajo como SIMU, para indicar que es una simulación, en lugar de una instalación real sobre hardware.

- –**gNB**: activa la construcción de la funcionalidad de gNB.
- –**nrUE**: activa la construcción para el NR UE.
- –**build-e2**: activa la construcción de la interfaz E2 para la comunicación entre la gNB y el Near-RT RIC.
- –**ninja**: se especifica el uso de *ninja*, un sistema de construcción más eficiente, que ayuda a acelerar el proceso de compilación mediante un manejo más optimizado de las dependencias y la construcción en paralelo.

El siguiente paso es lanzar la gNB y el UE. Para la gNB, se toma el fichero *gnb.sa.band78.fr1.106PRB.usrpb210.conf* (véase el Anexo A para más detalles sobre él). Se escoge este fichero ya que especifica operación en la banda 78, dentro de la subfranja FR1 en el rango de frecuencias. Esta banda, también conocida como n78, es una de las más utilizadas globalmente para la implementación de redes 5G debido a su equilibrio entre cobertura y capacidad. Además, la elección de 106 PRB en la configuración indica que se utilizará un ancho de banda específico para el canal de comunicación, correspondiente a un escenario con una capacidad media-alta. Aunque el nombre del fichero hace referencia al uso de *Universal Software Radio Peripheral* (USRP) B210, la configuración está pensada para un entorno de simulación utilizando `RFSimulator`.

Es posible visualizar todos los ficheros de configuración disponibles en [58]. Además, cabe resaltar que estos ficheros son ejemplos de configuraciones ya hechas, pero cada uno se puede modificar según las necesidades de su entorno.

Una vez hecha la compilación, para ejecutar la gNB se emplea el siguiente comando dentro del directorio `/oai/cmake_targets/ran_build/build`:

```
sudo ./nr-softmodem -O ../../../targets/PROJECTS/GENERIC-NR-5GC
/CONF/gnb.sa.band78.fr1.106PRB.usrpb210.conf
--gNBs.[0].min_rxtxtime 6 --rfsim --sa
```

Como se puede ver, se lanza el fichero de configuración comentado haciendo uso del ejecutable `nr-softmodem` con las siguientes opciones:

- –**gNBs.[0].min_rxtxtime**: esta opción establece el tiempo mínimo de recepción/transmisión de la señal para el primer gNB (índice [0]). Esto está relacionado con el tiempo de sincronización y la latencia de transmisión/recepción. En este caso, se establece a 6 ms.
- –**rfsim**: indica que el sistema no utilizará hardware radio real (como el USRP), sino que se simulará. Esto es útil para pruebas y simulaciones en lugar de trabajar con hardware físico real.

- –**sa**: habilita el modo *standalone* en el contexto de 5G, es decir, hace referencia a una arquitectura de red en la que el acceso de radio y el núcleo son independientes.

Hay una gran variedad de opciones disponibles, consultar el Anexo B para saber más.

Con la gNB corriendo, para ejecutar el UE se emplea el siguiente comando dentro del directorio `/oai/cmake_targets/ran_build/build`:

```
sudo ./nr-uesoftmodem -r 106 --numero\textit{log}y 1 --band 78
-C 3619200000 --rfsim --sa --uicc0.imsi 001010000000001
--rfsimulator.serveraddr 127.0.0.1
```

Como se observa, se emplea el ejecutable `nr-uesoftmodem`, el cual implementa la capa radio para un UE. En este caso, no se usa ningún fichero de configuración, sino que directamente al ejecutable se le pasan una serie de parámetros (véase el Anexo C). Los principales y los que se van a usar en este proyecto son los siguientes:

- **-r**: esta opción establece el ancho de banda en PRBs. En este caso se establece un ancho de banda de 106 PRBs, lo que equivale a 20 MHz según la figura 6.1

- –**numerology**: define la numerología utilizada para la comunicación. En este caso, se está usando la numerología 1, que especifica un espaciado de subportadora de 30 kHz, entre otros.

- –**band**: especifica la banda de operación 78, que corresponde a una frecuencia en el rango FR1 para 5G.

- **-C**: establece la frecuencia en *downlink* en Hz (en este caso, 3619.2 MHz).

- –**rfsim**: habilita la simulación radio.

- –**sa**: activa el modo *standalone*.

- –**uicc0.imsi**: establece el *International Mobile Subscriber Identity* (IMSI) 001010000000001 para el UE, que es un identificador único de suscripción para la red móvil.

- –**rfsimulator.serveraddr**: especifica la dirección del servidor de simulación radio, en este caso, la gNB a 127.0.0.1 (el puerto es por defecto 4043).

Al ejecutarlo, el UE se conecta a la red y recibe la IP 10.0.0.2 en la interfaz de túnel *oaitun_ue1*. En este caso, el entorno está diseñado de forma que todos los elementos del sistema se ejecutan en contenedores interconectados con direcciones IP estáticas dentro de una misma subred. Gracias a esto, se evita la necesidad de tener que realizar un *routing* adicional que facilite la comunicación entre ellos.

Near-RT RIC: FlexRIC

El último componente a instalar es el Near-RT RIC. En este caso, se emplea el FlexRIC. Para empezar, es necesario clonar el repositorio [29]:

```
# Clona el repositorio de FlexRIC desde GitHub
git clone https://github.com/openaicellular/flexric.git ~/flexric

# Cambia al directorio de FlexRIC
cd ~/flexric

# Cambia a la rama 'aymanfatich'
git checkout aymanfatich
```

Figura 5.8: Descarga de FlexRIC en la rama *aymanfatich* [59].

A continuación, se debe compilar:

```
# Crea un directorio 'build' para la compilación
mkdir build

# Cambia al directorio 'build'
cd build

# Ejecuta cmake para configurar la compilación
cmake ../

# Compila el proyecto utilizando múltiples núcleos (con 'nproc' para
    determinar el número de núcleos disponibles)
make -j`nproc`

# Instala el proyecto compilado en el sistema
sudo make install
```

Figura 5.9: Compilación de FlexRIC.

Se lanza ejecutando `./flexric/build/examples/ric/nearRT-RIC`.

En la guía oficial [57], se utiliza una rama diferente del repositorio de FlexRIC. Sin embargo, en este caso, se ha optado por emplear la rama de GitLab *aymanfatich* [59], ya que incluye la configuración necesaria para que la *xApp* pueda exportar métricas a Grafana. En particular, el fichero de configuración contiene las siguientes líneas:

```
[NEAR-RIC]
NEAR_RIC_IP = 127.0.0.1

[XAPP]
DB_DIR = /flexric/
DB_NAME = xapp_db
```

Figura 5.10: Fichero de configuración del FlexRIC en la rama *aymanfatich* [59].

En este archivo se especifica que el Near-RT RIC será accesible en la IP 127.0.0.1 (puerto 36421 por defecto) y que la *xApp* enviará los datos a una base de datos SQLite llamada *xapp_db*. Este proceso de exportación de métricas está gestionado en el repositorio, específicamente en el directorio *src/xApp/db*. Además, en el directorio raíz, donde también se encuentra el archivo de configuración, se incluye otro fichero denominado *grafana-dashboard.json*. Este último permite visualizar las métricas exportadas en Grafana.

Otro aspecto importante recae en la comunicación entre la gNB y este Near-RT RIC. Como se ha ido mencionando en los capítulos 1 y 3, esta comunicación se establece a través de la interfaz E2, la cual soporta una serie de modelos de servicio. En esta solución, esta interfaz soporta los modelos E2SM-KPM Y E2SM-RC conforme a las especificaciones de *O-RAN.WG3.E2SM-KPM-V02.03* y *O-RAN.WG3.E2SM-RC-V01.03* [60]. Estas especificaciones regulan el intercambio de métricas y eventos entre los componentes del sistema, lo que permite que el Near-RT RIC realice un monitoreo continuo y un control eficiente de las funciones de la gNB, optimizando así la gestión dinámica de la red 5G en tiempo real.

En cuanto a la monitorización de las distintas métricas de rendimiento, *O-RAN.WG3.E2SM-KPM-V02.03* está alineada con las especificaciones del *3GPP TS 28.552* [61]. Este marco proporcionado por 3GPP establece las directrices para la medición y evaluación de un gran número de KPIs dentro de las redes 5G. Entre ellos, aquellos con los que se está trabajando en esta solución son:

Indicador	Descripción
`DRB.AirIfDelayUl/Dl`	Tiempo de retardo en la interfaz aérea en enlace ascendente/descendente.
`DRB.RlcDelayUl`	Retardo introducido por la capa RLC en enlace ascendente.
`DRB.RlcPacketDropRateDl`	Tasa de pérdida de paquetes en la capa RLC en enlace descendente.
`DRB.RlcSduDelayDl`	Retardo de las SDU transmitidas por RLC en enlace descendente.
`DRB.UEThpUl/Dl`	*Throughput* del usuario en enlace ascendente/descendente.
`RRU.PrbAvailUl/Dl`	Número de bloques de recursos físicos disponibles en enlace ascendente/descendente.
`RRU.PrbTotUl/Dl`	Total de bloques de recursos físicos en enlace ascendente/descendente.
`RSRP (L1-RSRP)`	(Añadida por OAIC, no definida en *O-RAN KPM v2.03*) Potencia recibida de la señal de referencia a nivel de capa 1.
`RSRQ`	(Añadida por OAIC, no definida en *O-RAN KPM v2.03*) Calidad de la señal de referencia recibida.
`DRB.PdcpSduVolumeUl/DL`	Volumen de datos intercambiado (en kb) a nivel PDCP.

Tabla 5.1: Indicadores clave de rendimiento (KPI) en O-RAN

De entre los mencionados, hay algunos completamente incorporados, como `DRB.PdcpSduVolumeDL`, `DRB.PdcpSduVolumeUL`, `DRB.RlcSduDelayDl`, `DRB.UEThpDl` y `DRB.UEThpUl`; mientras que los demás continúan en desarrollo.

En cuanto al control, en *O-RAN.WG3.E2SM-RC-V01.03* se definen una gran variedad de configuraciones que permiten controlar diversos aspectos de la red. En la versión actual de la solución, se tienen capacidades de modificación del algoritmo de *pacing*, optimización de colas, clasificación de tráfico y gestión de *slices*, permitiendo la creación, eliminación y modificación de estos según las necesidades de la red.

Finalmente, se desea destacar que esta versión del FlexRIC contiene una serie de *xApps* con la que realizar las tareas de monitorización y control, las cuales serán comentadas en el capítulo 6.

De esta forma, con todos los componentes ejecutándose, se puede hacer

una simple prueba y comprobar que el UE y el *core* tienen comunicación entre sí en ambos sentidos, pasando por la gNB:

```
oaic@oaic-VirtualBox:~$ ping 192.168.70.135 -I oaitun_ue1
PING 192.168.70.135 (192.168.70.135) from 10.0.0.2 oaitun_ue1: 56(84) bytes of data.
64 bytes from 192.168.70.135: icmp_seq=1 ttl=63 time=10.9 ms
64 bytes from 192.168.70.135: icmp_seq=2 ttl=63 time=19.9 ms
64 bytes from 192.168.70.135: icmp_seq=3 ttl=63 time=20.6 ms
^C
--- 192.168.70.135 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 10.896/17.126/20.624/4.416 ms
oaic@oaic-VirtualBox:~$ sudo docker exec -it oai-ext-dn ping 10.0.0.2
[sudo] contraseña para oaic:
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=63 time=19.1 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=63 time=23.7 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=63 time=19.2 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=63 time=18.7 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 18.734/20.173/23.737/2.063 ms
```

Figura 5.11: *Ping* del UE al UPF del *core* y viceversa.

Arquitectura y direccionamiento

A continuación, se presenta una figura que muestra la arquitectura completa implementada y configurada, incluyendo el direccionamiento de los distintos componentes del sistema.

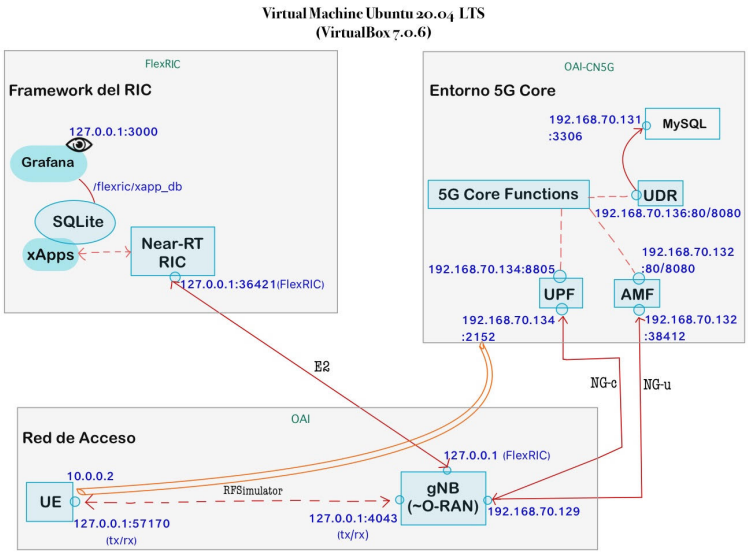


Figura 5.12: Arquitectura funcional con OAIC.

En la figura 5.12 se distinguen tres bloques principales:

- ***Framework* del RIC**:
  - Near-RT RIC, que se comunica con la gNB a través de la interfaz E2 utilizando la dirección 127.0.0.1:36421.

- *xApps*.
- Una base de datos SQLite.
- Grafana accesible en 127.0.0.1:3000.

- **Red de Acceso**:
  - gNB, que se comunica con el Near-RT RIC y con el UE mediante la interfaz local y está disponible en la dirección 192.168.70.129 para la conexión con el *core* 5G.
  - UE, el cual se comunica en local con la gNB y recibe la IP 10.0.0.2 al establecer la conexión completa con la red y crearse la sesión PDU.

- ***core* 5G**:
  - AMF accesible en 192.168.70.132 (puerto 38412 con la gNB y 8080 con otras funciones de red).
  - UPF accesible en 192.168.70.134 (puerto 2152 con la gNB y 8805 con otras funciones de red).
  - UDR accesible en 192.168.70.136 (puertos 80 y 8080).
  - Base de datos MySQL, la cual tiene conexión con el UDR y es accesible en la dirección 192.168.70.131:3306.

5.3. Implementación del escenario srsRAN

El segundo escenario hace uso de la solución propuesta por srsRAN. Seguidamente, se detalla el proceso seguido siguiendo la guía [62].

5.3.1. Instalación de la VM y dependencias

En este caso, todo el sistema se despliega sobre una máquina virtual Ubuntu 22.04 LTS. Además, al igual que en el caso anterior, es necesario instalar una serie de dependencias. Las más generales serían:

```
sudo apt install build-essential cmake make pkg-config libfftw3-dev
libsctp-dev libyaml-cpp-dev libmbedtls-dev libgtest-dev ccache
libbackward-cpp-dev libboost-program-options-dev libconfig++-dev
```

Figura 5.13: Instalación de las librerías básicas.

Aparte de los paquetes mencionados, se instala SWIG en esta nueva máquina, ya que es necesario para el uso de FlexRIC. Esto se puede hacer siguiendo los pasos mencionados en 5.2.1. También es necesario instalar

Docker, ya que el *core* de la red se implementa con una versión dockerizada de Open5GS (consultar la sección 5.2.1) y el compilador GCC en la versión 10 (sección 5.2.1). Por último, el simulador de canal radio ZeroMQ y los repositorios srsRAN 5G y srsRAN 4G también.

ZeroMQ

ZeroMQ [34] es una biblioteca de mensajería asíncrona que permite la comunicación eficiente entre procesos a través de *sockets*. Establece automáticamente dos sockets: uno de envío y otro de recepción, facilitando la implementación de patrones de comunicación de tipo publicación-suscripción. En este caso, su instalación se realiza mediante el paquete *libzmq3-dev* utilizando el siguiente comando: `sudo apt-get install libzmq3-dev`.

srsRAN Project (5G)

Se descarga el repositorio de srsRAN 5G [63], el cual contiene la gNB y una versión dockerizada de Open5GS, este último se usa como *core* 5G:

```
# Clona el repositorio de srsRAN
git clone https://github.com/srsran/srsRAN_Project.git

# Cambia al directorio del proyecto
cd srsRAN_Project

# Crea un directorio 'build' para la compilación
mkdir build

# Cambia al directorio 'build'
cd build

# Ejecuta cmake para configurar la compilación con opciones adicionales (
cmake ../ -DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON

# Compila el proyecto utilizando múltiples núcleos (con 'nproc' para determinar el número de núcleos disponibles)
make -j`nproc`
```

Figura 5.14: Compilación de srsRAN 5G.

srsRAN 4G

Este repositorio [64] contiene una instancia del UE que se conectará a la gNB en la red de acceso. Para instalarlo:

```
# Clona el repositorio de srsRAN 4G
git clone https://github.com/srsRAN/srsRAN_4G.git

# Cambia al directorio del proyecto
cd srsRAN_4G

# Crea un directorio 'build' para la compilación
mkdir build

# Cambia al directorio 'build'
cd build

# Ejecuta cmake para configurar la compilación
cmake ../

# Compila el proyecto
make

# Realiza las pruebas del proyecto
make test

# Instala el proyecto compilado en el sistema
sudo make install

# Ejecuta el script de instalación de configuraciones
srsran_install_configs.sh user
```

Figura 5.15: Compilación de srsRAN 4G.

5.3.2. Despliegue de la red

Open5GS *Core*

Para el núcleo de esta nueva red 5G se emplea Open5GS, una solución de código abierto escrita en lenguaje C que implementa tanto el 5GC como el EPC (este último no se necesita). En este caso, se emplea una versión dockerizada de proporcionada por srsRAN, la cual ya incluye la instalación y configuración del direccionamiento IP, entre otros aspectos esenciales para la operación del núcleo. Esto simplifica considerablemente el proceso de configuración, ya que esta versión se encarga de:

- Configuración automática de *Mobile Country Code* (MCC) y *Mobile Network Code* (MNC) (001/01).
- Configuración de la dirección de enlace del AMF y del UPF con la gNB. En este caso, viene por defecto una dirección de *loopback*, la cual se cambia a la IP 10.53.1.2.
- Establecimiento de las direcciones IP de las demás funciones de red (todas ellas en local, es decir, 127.0.0.X).
- Configuración de una IP virtual de todo el contenedor (10.45.1.1) con la que este se comunicará con el UE.

- Configuración de parámetros como el *Tracking Area Code* (TAC) (establecido a 7) o *Network Slice Selection Assistance Information* (NSSAI) (=1) para permitir la conectividad y la operación de la red 5G.
- Creación de una base de datos MongoDB en 127.0.0.1:9999 para almacenar la información de la red y sus suscriptores.
- Adición de un suscriptor con IMSI 001010123456780, el cual recibirá IP 10.45.1.2 cuando se establezca la sesión PDU.

Es importante destacar que en el fichero *docker-compose.yml* en el que se realiza toda esta configuración, se debe establecer de forma correcta la ruta al fichero de configuración de la gNB, para que la conexión entre la red de acceso y el núcleo de red sea satisfactoria (se hablará de este fichero y de su localización en la sección 5.3.2). Una vez establecida la configuración:

```
cd ./srsRAN_Project/docker
sudo docker compose up 5gc
```

Near-RT RIC

-FlexRIC

El siguiente componente es el Near-RT RIC. Esta solución permite operar con dos tipos, FlexRIC y ORAN-SC-RIC. En cuanto al FlexRIC, al igual que en el escenario OAIC, es necesario clonar el repositorio [29]:

```
# Clona el repositorio de FlexRIC desde GitHub y va a su directorio
git clone https://github.com/openaicellular/flexric.git
cd flexric
# Cambia a la rama 'br-flexric'
git checkout br-flexric
```

Figura 5.16: Descarga de FlexRIC en la rama 'br-flexric'.

A continuación, se debe compilar:

```
# Crea un directorio 'build' para la compilación
mkdir build
cd build
# Configura la compilación
cmake -DKPM_VERSION=KPM_V3_00 -DXAPP_DB=NONE_XAPP ../
# Compila el proyecto utilizando múltiples núcleos
make -j`nproc`
# Instala el proyecto compilado en el sistema
sudo make install
```

Figura 5.17: Compilación de FlexRIC.

Se lanza ejecutando `./flexric/build/examples/ric/nearRT-RIC`.

Esto arranca una instancia de este Near-RT RIC en la dirección 127.0.0.1: 36421, según su fichero de configuración:

```
SM_DIR = "/usr/local/lib/flexric/"
# supported name = NearRT_RIC, E2_Agent, E2_Proxy_Agent, xApp

Name = "NearRT_RIC"
NearRT_RIC_IP = "127.0.0.1"

E2_Port = 36421
E42_Port = 36422
```

Figura 5.18: Fichero de configuración del FlexRIC en la rama 'br-flexric'.

-ORAN-SC-RIC

Por otro lado, también es posible hacer uso del ORAN-SC-RIC. En concreto, se instala una versión mínima contenida en un entorno Docker propiedad de srsRAN, eliminando así la necesidad de Kubernetes o Helm.

Primero, es necesario clonar el repositorio correspondiente:

```
# Clona el repositorio del ORAN-SC-RIC
git clone https://github.com/srsran/oran-sc-ric
# Cambia al directorio del proyecto y crear los ficheros de
    configuración.
cd oran-sc-ric
./create_ric_config_files.sh
```

Figura 5.19: Clonación del repositorio de ORAN-SC-RIC.

Una vez descargado, se puede iniciar mediante Docker:

```
# Despliega la instancia de ORAN-SC-RIC usando Docker Compose
sudo docker compose up
```

Figura 5.20: Despliegue de ORAN-SC-RIC mediante Docker.

El último comando mostrado lanza una instancia mínima funcional de este Near-RT RIC que incluye todos los servicios esenciales en contenedores separados, comunicándose entre sí en una red Docker común. Estos servicios están definidos en el archivo *docker-compose.yml*:

- **dbaas** (IP: 10.0.2.12): contenedor Redis utilizado como base de datos para el RIC. Carga un módulo adicional al iniciar.

- **rtmgr_sim** (IP: 10.0.2.15): simulador del *Route Manager*, usado para pruebas de enrutamiento y mensajería.
- **submgr** (IP: 10.0.2.13): gestor de suscripciones de *xApps* a servicios del RIC. Se conecta con el dbaas para gestionar suscripciones.
- **e2term** (IP: 10.0.2.10): punto terminal del plano E2, encargado de gestionar la comunicación con los nodos de radio, en este caso, la gNB.
- **appmgr** (IP: 10.0.2.14): gestor de aplicaciones *xApps*, coordina la ejecución y ciclo de vida de las aplicaciones que corren en el RIC.
- **e2mgr** (IP: 10.0.2.11): manager del plano E2, coordina la gestión de las conexiones E2 y la configuración del sistema.
- **python_xapp_runner** (IP: 10.0.2.20): contenedor base para ejecutar *xApps* escritas en Python.

En cuanto al canal de comunicación entre la gNB y el Near-RT RIC, srsRAN emplea las especificaciones *O-RAN.WG3.E2SM-KPM-R003-v03.00* (con base en *O-RAN.WG3.E2SM-R003-v03.00*) y *O-RAN.WG3.E2SM-RC-R003-v03.00* [60]. Estas normas definen tanto el protocolo de transporte como la estructura y funcionalidad de los modelos de servicio que permiten al Near-RT RIC supervisar y actuar sobre el comportamiento de la gNB en tiempo casi real.

Para la recolección de métricas clave, se emplea el modelo de servicio *E2SM-KPM-R003*, que en esta implementación soporta todos los estilos de reporte definidos (*Service Styles* 1 a 5):

- ***Style 1. E2 Node Measurement***: utilizado para transportar reportes de mediciones a nivel de nodo E2, agregando métricas generales sin distinguir entre UEs.
- ***Style 2. E2 Node Measurement for a Single UE***: permite recolectar reportes de mediciones específicos para un único usuario (UE) de interés desde un nodo E2.
- ***Style 3. Condition-based, UE-level E2 Node Measurement***: facilita reportes por grupo de UEs cuyas métricas cumplen ciertas condiciones previamente definidas, diferenciando por tipo de medición.
- ***Style 4. Common Condition-based, UE-level Measurement***: similar al estilo 3, pero agrupando múltiples tipos de medición bajo condiciones comunes aplicadas a varios UEs.

- ***Style 5. E2 Node Measurement for Multiple UEs***: se enfoca en la recolección de métricas específicas para múltiples UEs seleccionados dentro del nodo E2.

Esto permite al sistema ofrecer una monitorización flexible de los parámetros de rendimiento, alineándose con lo establecido por la especificación *3GPP TS 28.552* [61]. Dentro del conjunto de indicadores disponibles, se incluyen todos los descritos en la solución OAIC, añadiendo:

- `DRB.RlcPacketDropRateUl`: tasa de éxito de paquetes en enlace ascendente.
- `DRB.PacketSuccessRateUlgNBUu`: tasa de pérdida de paquetes RLC en enlace descendente.
- `DRB.RlcSduTransmittedVolumeUl/DL`: volumen de SDUs transmitidas por RLC en enlace ascendente/descendente.

Cabe señalar que, al igual que con OAIC, muchos de ellos siguen en desarrollo, mientras que los que están ya configurados para ser usados en la plataforma son `DRB.UEThpUl/Dl` y `RRU.PrbAvailUl/Dl` y, en versiones concretas, `DRB.RlcSduTransmittedVolumeUl/Dl`.

Respecto a las capacidades de control, el modelo *E2SM-RC-R003* se implementa en su versión con soporte exclusivo para el estilo de servicio 2, el cual permite llevar a cabo configuraciones concretas como la disposición de celdas, *slices*, gestión de PRBs, etc.

En este contexto, se han desarrollado y desplegado diversas *xApps* sobre FlexRIC y ORAN-SC-RIC que permiten explotar las capacidades descritas, y que serán analizadas con detalle en el capítulo 6 mediante experimentación y evaluación de resultados.

Red de Acceso (srsRAN 5G + srsRAN 4G)

Con el núcleo 5G y el Near-RT RIC funcionando, srsRAN ofrece una solución de red de acceso de código abierto tanto para 5G como para 4G. En el caso de srsRAN 5G, el proyecto incluye la implementación de la gNB. Por otro lado, srsRAN 4G se enfoca en la implementación del UE. Ambas plataformas son compatibles con las interfaces y especificaciones de O-RAN, lo que permite una integración flexible y un control preciso sobre la infraestructura de acceso en sus respectivas generaciones de red.

Para poner en marcha la instancia de gNB, se debe acceder al directorio `srsRAN_Project/build/ apps/gnb`. En él, se debe crear el fichero de configuración que contiene las características que se desean para esta estación

base (en este caso se le llama *gnb_zmq.yaml)*. En el Anexo D, se muestra un ejemplo básico de configuración de la gNB, el cual se irá modificando en el capítulo 6 (si se desea ver todas las opciones de configuración que se pueden añadir a este fichero, consultar [65]). Entre los parámetros más importantes se encuentran:

- **cu_cp**: sección principal para configurar el plano de control de la unidad de control de la gNB. En ella, se incluye la IP del AMF como 10.53.1.2, el TAC a 7, 001/01 para el MCC/MNC (prueba internacional) y temporizadores de inactividad.
- **ru_sdr**: define el controlador de radio y sus parámetros dentro de la unidad radio. En este caso, se usa ZeroMQ para la conexión con el UE con dos *sockets*, uno para transmitir y otro para recibir. Además, se establece la tasa de muestreo, que viene determinada por el ancho de banda de canal y las ganancias de transmisión y recepción.
- **cell_cfg**: configura los parámetros de la celda como la banda NR 3. Se utiliza la banda n3 de tipo FDD, ya que la banda n78 empleada en la solución anterior, aunque ofrece mayores velocidades, no es compatible debido a que requiere modo TDD; a pesar de ello, se ha elegido esta banda ya que proporciona una configuración estable y es ampliamente soportada en implementaciones SA de 5G, facilitando las pruebas funcionales básicas del UE (en el capítulo 6 se variará entre las bandas FDD). Además, se configura el ancho de banda (10 MHz), espacio de subportadora (15 kHz), TAC (7), MCC/MNC (001/01), y configuraciones específicas de canales físicos como PDCCH, PRACH, PDSCH y PUSCH.
- ***log***: permite especificar el nivel de detalle del *log* y la ruta donde se almacenará el archivo de registro.
- **pcap**: habilita la captura de paquetes en distintas capas, especificando también las rutas donde se guardarán los archivos correspondientes.
- **e2**: activa y configura el agente E2, incluyendo la dirección y el puerto del Near-RT RIC (127.0.0.1:36421 para el FlexRIC y 10.0.2.10:36421 para el ORAN-SC-RIC), y la activación de módulos como KPM y *Radio Control* (RC).
- **metrics**: define la configuración del sistema de métricas. Permite establecer el periodo de reporte para las métricas en milisegundos y en formato *json*. También especifica la dirección IP y el puerto del servidor al que se enviarán dichas métricas (172.19.1.4:55555).

Como se puede observar en su fichero de configuración, la gNB implementa una separación simulada entre la *Central Unit* (CU) y la DU, haciendo

uso de las diferentes interfaces definidas en la arquitectura O-RAN, como por ejemplo la interfaz E2 entre la O-CU-CP y el Near-RT RIC. En el repositorio del proyecto, se incluyen ficheros de configuración específicos tanto para la CU como para la DU, permitiendo su personalización de forma independiente si se desea. No obstante, dado que el presente proyecto se centra en la integración y validación del Near-RT RIC, resulta suficiente emplear esta solución en la que la gNB simula internamente dicha separación, facilitando así las pruebas sin necesidad de una implementación física completamente distribuida.

Con el fichero de configuración creado, la estación base se pone en marcha con uno de estos dos comandos, dependientes del Near-RT RIC que se desee emplear:

```
# Con ORAN SC RIC
sudo ./gnb -c gnb_zmq.yaml e2 --addr="10.0.2.10" --bind_addr=
"10.0.2.1"

# Con FlexRIC
sudo ./gnb -c gnb_zmq.yaml e2 --addr="127.0.0.1" --bind_addr=
"127.0.0.1"
```

Las direcciones IP utilizadas en estos comandos corresponden a la configuración de red entre la estación base y el Near-RT RIC. En el caso de ORAN-SC-RIC, la dirección 10.0.2.10 representa su IP a la que se conectará la gNB, mientras que 10.0.2.1 es la IP local de la gNB en esa red virtual o puente. Para FlexRIC, se utilizan direcciones de *loopback* (127.0.0.1).

El siguiente paso es iniciar el UE. Como se ha mencionado, este forma parte del proyecto srsRAN 4G y al igual que con la gNB, requiere de un fichero de configuración (véase el Anexo E). En él se configura:

- **rf**: sección principal para configurar los parámetros de radiofrecuencia. Incluye el desplazamiento de frecuencia (0 Hz), las ganancias de transmisión y recepción (50 dB), la tasa de muestreo a 11.52 MHz con un ancho de banda de 10 MHz(que debe coincidir con la indicada en la gNB) y el número de antenas (1 antena). Además, especifica el dispositivo de radio (ZeroMQ) y los puertos de transmisión y recepción para la comunicación (2 *sockets*).
- **rat.eutra**: se define el número de portadora de bajada y el número de portadoras 4G.
- **rat.nr**: configura los parámetros para la red NR (5G), incluyendo la banda NR (banda 3) y el número de portadoras NR (1 portadora).

- **pcap**: habilita la captura de paquetes.
- ***log***: define el nivel de detalle de los *logs*. Especifica el nivel de registro para todo el sistema, la capa física, el límite de tamaño de los *logs* en formato hexadecimal y la ruta del archivo de registro.
- **usim**: configura los parámetros relacionados con el usuario, como el modo de operación (simulación), el algoritmo de autenticación (*Milenage*), la clave *Operator Configuration Parameter* (OPC), la clave secreta K y los identificadores IMSI e *International Mobile Equipment Identity* (IMEI).
- **rrc**: configura los parámetros del protocolo RRC, incluyendo la versión de liberación y la categoría del dispositivo UE.
- **nas**: configura los parámetros para la red de acceso no radio, especificando el nombre del *Access Point Name* (APN) (se establece a srsapn, un nombe ficticio que representa el perfil de acceso en el que se especifican ciertos parámetros para la conexión de datos, como la dirección IP, las reglas de enrutamiento, y otros detalles relacionados con la conexión a la red de datos del operador) y el protocolo utilizado (IPv4).
- **gw**: establece los parámetros de la pasarela de red, como el espacio de nombres de red (ue1), el nombre del dispositivo de red y la máscara de subred (/24).
- **gui**: habilita o desactiva la interfaz gráfica, en este caso desactivada.

Este fichero también tiene más opciones, las cuales se pueden investigar en [66].

Para poner en marcha el usuario, basta con crear el espacio de nombres que se define en el fichero de configuración descrito con el comando `sudo ip netns add ue1` y en el directorio *srsRAN_4G/build/srsue/src* ejecutar `sudo ./srsue ue_zmq.conf`.

Una vez ejecutado el comando, la conexión se ha establecido correctamente una vez que al UE se le asigna una dirección IP en la interfaz *tun_srsue*, lo cual se observa en el mensaje *PDU Session Establishment successful. IP: 10.45.1.2*. La conexión queda confirmada posteriormente con el mensaje *RRC NR reconfiguration successful.*

Recolección de métricas y visualización en Grafana

Como se ha comentado en la sección 5.3.2, la gNB manda las métricas un servidor de métricas en la dirección 172.19.1.4. Pues bien , dicho servidor de

métricas las almacena en una base de datos InfluxDB accesible en 172.19.1.5, de la cual lee los datos Grafana haciendo uso de la dirección 172.19.1.6. El usuario puede ver los resultados recopilados en 127.0.0.1:3300.
Todo esto queda definido en el archivo *docker-compose.yml* del directorio *docker* del repositorio, el mismo fichero donde se realiza toda la configuración del *core* 5G mencionada en la sección 5.3.2.
Para lanzar esta recopilación, se debe ejecutar el comando `sudo docker compose -f docker/docker-compose.yml up grafana` en el directorio raíz del repositorio srsRAN 5G.

Routing: Configuración de rutas para el tráfico entre el UE y el *core*

En este escenario, al haber tantas subredes mezcladas, es necesario llevar a cabo un cierto enrutamiento para permitir la comunicación bidireccional entre el UE y el *core* de la red (Open5GS).

Concretamente, se deben realizar dos cosas:

- Añadir una ruta que permita alcanzar el UE (10.45.0.0/16) a través de la interfaz virtual conectada al contenedor de Open5GS (10.45.1.1).
- Establecer una ruta por defecto en el espacio de nombres del UE, de modo que todo el tráfico generado por éste se dirija al *core* mediante la interfaz *tun_srsue* (el túnel de datos).

Para automatizar esta configuración, se ha creado el *script ip_config.sh*, que realiza los pasos anteriores de forma secuencial:

```
1 #!/bin/bash
2
3 # Añade ruta desde el host hacia la red del UE pasando por el
      contenedor del core
4 sudo route add -net 10.45.0.0 netmask 255.255.0.0 gw 10.53.1.2
5
6 # Añade la ruta por defecto en el espacio de nombres del UE
7 sudo ip netns exec ue1 ip route add default via 10.45.1.1 dev
      tun_srsue
```

Figura 5.21: *Script ip_config.sh* para configurar el enrutamiento entre el UE y el *core*.

Al ejecutar este *script*, la tabla de rutas del sistema tiene la estructura mostrada en la tabla 5.2 (cabe destacar que también se tendrían las columnas *Ref* y *Uso*, todas con valor 0, las cuales se omiten por simplicidad, ya que representan campos no relevantes).

Destino	Pasarela	Genmask	Indic	Métric	Interfaz
0.0.0.0	192.168.1.1	0.0.0.0	UG	100	enp0s8
10.0.2.0	0.0.0.0	255.255.255.0	U	0	br-3cd2761e2cf4
10.45.0.0	10.53.1.2	255.255.0.0	UG	0	br-df6dd1752908
10.53.1.0	0.0.0.0	255.255.255.0	U	0	br-df6dd1752908
169.254.0.0	0.0.0.0	255.255.0.0	U	1000	enp0s8
172.17.0.0	0.0.0.0	255.255.0.0	U	0	docker0
172.19.1.0	0.0.0.0	255.255.255.0	U	0	br-0b35fbbaa419
192.168.1.0	0.0.0.0	255.255.255.0	U	100	enp0s8

Tabla 5.2: Tabla de rutas IP del *host* tras ejecutar `ip_config.sh`.

Como se observa, la ruta por defecto (0.0.0.0/0) establece que cualquier paquete cuyo destino no coincida con una ruta más específica será enviado a la pasarela 192.168.1.1 a través de la interfaz *enp0s8*. La red 10.0.2.0/24 está conectada directamente a la interfaz *br-3cd2761e2cf4*, lo que indica que todo el tráfico con destino a esa red no requiere pasarela (tráfico gNB - ORAN-SC-RIC). En el caso de la red 10.45.0.0/16, se observa que su ruta pasa por la puerta de enlace 10.53.1.2 mediante la interfaz *br-df6dd1752908*. Esto significa que un UE con IP 10.45.1.2, al querer comunicarse con otro UE dentro de la red 10.45.0.0/16, deberá enviar los paquetes a través del *core*. Por otro lado, la red 10.53.1.0/24 está conectada directamente a la misma interfaz, indicando que las comunicaciones internas en esta subred no requieren intermediarios.
Las redes 172.17.0.0/16 y 172.19.1.0/24 (interfaces *docker0* y *br-0b35fbbaa419*, respectivamente) también están directamente conectadas y se utilizan para la red de contenedores relacionados con el almacenamiento y visualización de métricas. Finalmente, 169.254.0.0/16 una red reservada para autoconfiguración y también es accesible de forma directa por *enp0s8*.

Se comprueba con la herramienta *ping* que existe comunicación bidireccional entre el UE y el *core* de la red 5G:

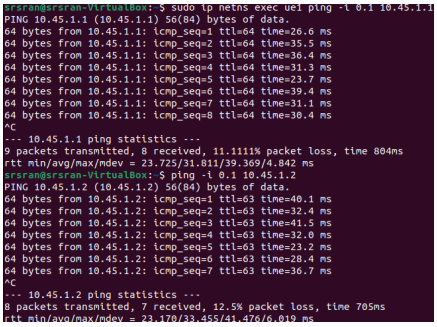


Figura 5.22: *Ping* del UE a la IP virtual del *core* y viceversa.

Arquitectura y direccionamiento

Al igual que con el escenario anterior, a continuación se presenta una figura que muestra la arquitectura completa implementada, configurada y virtualizada en Ubuntu 22.04 LTS, incluyendo la dirección de los distintos componentes del sistema.

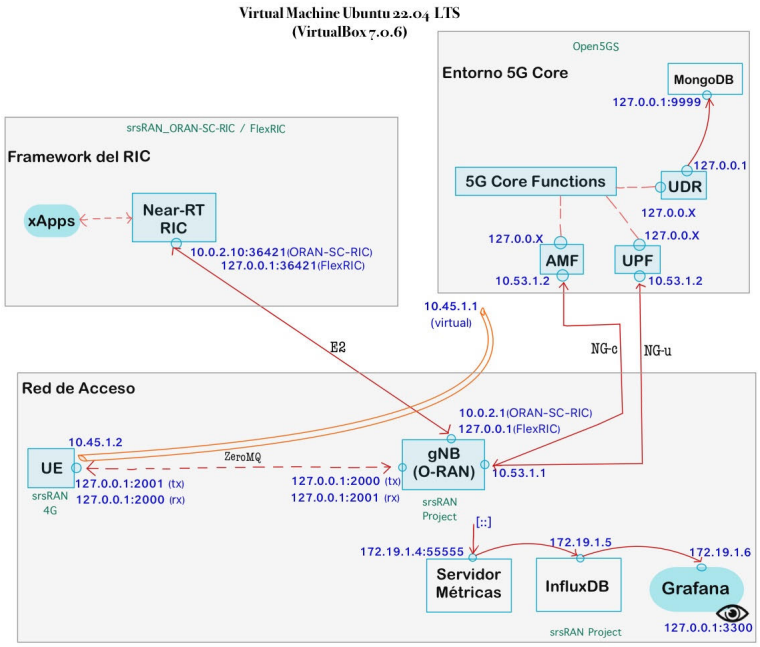


Figura 5.23: Arquitectura funcional con srsRAN.

Se distinguen tres bloques principales:

- ***Framework* del RIC**:
  - Near-RT RIC, que se comunica con la gNB a través de la interfaz E2 utilizando la dirección 127.0.0.1:36421 en el caso de FlexRIC o 10.0.2.10:36421 para ORAN-SC-RIC.
  - *xApps*.
- **Red de Acceso**:
  - gNB, que se comunica con el FlexRIC y con el UE mediante la interfaz local, con el ORAN-SC-RIC en la 10.0.2.1 y está disponible en la dirección 10.53.1.1 para la conexión con el *core* 5G.
  - Servidor de métricas que recibe de cualquier interfaz en la IP 172.19.1.4 las métricas de red y las envía una base de datos InfluxDB.

- Base de datos InfluxDB que alamacena las métricas recibidas del servidor de métricas. Está accesible en la dirección 172.19.1.5.
- Grafana, la cual toma las métricas de InfluxDB con la IP 172.19.1.6 y las pone a disposición del usuario en un *dashboard* en la dircción 127.0.0.1:3300.
- UE, el cual se comunica en local (mediante dos *sockets*) con la gNB y recibe la IP 10.45.1.2 al establecer la conexión completa con la red y crearse la sesión PDU.

- ***Core* 5G de Open5GS**:
  - AMF accesible en 10.53.1.2 para la gNB y EN 127.0.0.X para otras funciones de red.
  - UPF accesible en 10.53.1.2 para la gNB y EN 127.0.0.X para otras funciones de red.
  - UDR accesible en 127.0.0.X para otras funciones de red y en 127.0.0.1 (*loopback*) con la base de datos MongoDB.
  - Base de datos MongoDB, la cual gestiona las suscripciones de los usuarios y es alcanzable en 127.0.0.1:9999.

Capítulo 6

Pruebas y Experimentación

6.1. Entorno de Pruebas

Para llevar a cabo las pruebas descritas en este trabajo, se han implementado dos entornos distintos: uno basado en OAIC [30] y otro en srsRAN [8]. Cada uno de ellos ofrece unas funcionalidades diferentes.

El entorno OAIC se emplea para realizar una prueba básica de funcionalidad. Esta decisión se debe a que es una plataforma más cerrada y con una comunidad de soporte más reducida en comparación con srsRAN. En todo caso, en esta prueba se valida la conectividad y comportamiento general del sistema mediante el uso de la herramienta `iperf` [67] para la generación de tráfico, monitorizando métricas clave a través de FlexRIC [29] y visualizando los resultados en un panel de Grafana. Además, se modifican ciertos parámetros del canal radio para observar cómo se reflejan estos cambios en las métricas obtenidas.

Por otro lado, se ha elegido el entorno srsRAN como el escenario para realizar el mayor conjunto de pruebas debido a varios motivos. En primer lugar, cuenta con una comunidad de desarrolladores mucho más activa, lo que facilita el acceso a soporte técnico, documentación actualizada y resolución de incidencias a través de foros y repositorios. En segundo lugar, esta plataforma emplea versiones de sistema operativo más recientes (Ubuntu 22.04 LTS frente al 20.04 LTS de OAIC) lo que permite mayor compatibilidad con herramientas actuales o paquetes más actualizados. A nivel técnico, srsRAN también integra versiones más recientes de las especificaciones de la interfaz E2, lo que habilita una interacción más flexible y moderna entre la gNB y el Near-RT RIC, y por último permite trabajar con ORAN-SC-RIC y con FlexRIC.

Por todo lo anterior, con srsRAN se realizan pruebas en dos niveles

progresivos. Al igual que en el entorno OAIC, se emplea la herramienta `iperf` para generar tráfico durante las pruebas:

- **Pruebas de monitorización**: se varian condiciones de red estables y se analiza la capacidad de supervisión de ambas implementaciones del Near-RT RIC: ORAN-SC-RIC y FlexRIC. Se comparan las métricas obtenidas con cada uno, visualizadas también a través de Grafana, con el objetivo de identificar diferencias.
- **Pruebas de control de red**: se evalúa la asignación avanzada de PRBs a usuarios usando ORAN-SC-RIC, que en versiones previas cuenta con *xApps* para ello y se prueba con dos UE conectados simultáneamente. Mientras ORAN-SC-RIC controla PRBs estándar, FlexRIC promete manejar PRBs con *slicing*. Se utiliza el primer Near-RT RIC porque, aunque FlexRIC parece más prometedor, tanto su comunidad como la de srsRAN aún requieren alineación para trabajar conjuntamente en este escenario, proceso que está en desarrollo.

Finalmente, se incluye una sección comparativa entre las dos implementaciones del Near-RT RIC, destacando ventajas, limitaciones y eficiencia de cada una según los resultados obtenidos en las pruebas de supervisión y control.

6.1.1. Observaciones técnicas

Cada entorno de pruebas impone sus limitaciones de rendimiento, derivadas igualmente de su arquitectura interna (forma en que cada plataforma implementa y gestiona el plano de usuario, la virtualización del canal y el procesamiento de paquetes). Tanto en el caso del entorno basado en OAIC como en el entorno de srsRAN, se ha observado que el caudal máximo alcanzable se sitúa en torno a los 60 Mbps en simulación.

Por último, cabe destacar que a lo largo de este capítulo se irán realizando pruebas cambiando el ancho de banda (medido en PRBs). Es importante destacar que los PRBs asociados a cada ancho de banda dependen del SCS:

- Below 6 GHz:

SCS (kHz)	Channel bandwidth (MHz)												
	5	10	15	20	25	30	40	50	60	70	80	90	100
15	25	52	79	106	133	[160]	216	270	*n/a*	*n/a*	*n/a*	*n/a*	*n/a*
30	11	24	38	51	65	[78]	106	133	162	[189]	217	[245]	273
60	*n/a*	11	18	24	31	[38]	51	65	79	[93]	107	[121]	135

  - Values for 30, 70, 90 MHz CBW not fixed and not in UE spec yet

- Above 24 GHz:

SCS (kHz)	Channel bandwidth (MHz)			
	50	100	200	400
60	66	132	264	*n/a*
120	32	66	132	264

Figura 6.1: Relación PRBs-SCS-Ancho de banda [68].

Además, se menciona que todos los anexos que se irán comentando a lo largo del capítulo se pueden encontrar en mi GitHub [69].

6.2. OAIC. Prueba de Funcionalidad

Como se mencionó al inicio de este capítulo, el uso de OAIC se emplea para validar la funcionalidad básica del sistema. Se evalúa su comportamiento y rendimiento en condiciones controladas, con el fin de asegurar que las funcionalidades clave operen como se espera.

Como se mencionó en el capítulo 5, para lanzar la red completa se deben ejecutar los comandos mostrados en la tabla 6.1, los cuales permiten lanzar cada uno de los elementos necesarios de la red:

Componente	Comando de ejecución
Core 5G	`sudo docker compose up -d`
FlexRIC (Near-RT RIC)	`./flexric/build/examples/ric/nearRT-RIC`
gNB	`sudo ./nr-softmodem -O ../../../targets/PROJECTS/GENERIC-NR-5GC/CONF/gnb.sa.band78.fr1.106PRB.usrpb210.conf --gNBs.[0].min_rxtxtime 6 --rfsim --sa`
UE	`sudo ./nr-uesoftmodem -r 106 --numerology 1 --band 78 -C 3619200000 --rfsim --sa --uicc0.imsi 001010000000001 --rfsimulator.serveraddr 127.0.0.1`
xApp	`./build/examples/xApp/c/monitor/xapp_kpm_moni`
Generación de tráfico	`sudo docker exec -it oai-ext-dn iperf -u -t 100 -i 1 -fk -B 192.168.70.135 -b 100M -c 10.0.0.2`

Tabla 6.1: Resumen de comandos para la puesta en marcha del entorno con FlexRIC y OAIC.

En la sección 5.2 del capítulo 5, se definió un escenario base para la validación de la red, en el que se mantenían constantes los principales parámetros de configuración (simulando buenas condiciones), utilizando los comandos mostrados anteriormente para el despliegue de cada componente.

A continuación, se plantean una serie de pruebas funcionales orientadas a analizar cómo afectan diferentes parámetros físicos y de configuración a las métricas clave de la red. Para ello, se hará uso de la `xapp_kpm_moni` del FlexRIC (véase el Anexo F), ejecutando el comando `./build/examples/xApp/c/monitor/xapp_kpm_moni`, el cual permite monitorizar indicadores como:

- `DRB.PdcpSduVolumeDL` y `DRB.PdcpSduVolumeUL`: volumen de datos intercambiado (en kb) a nivel PDCP.
- `DRB.RlcSduDelayDl`: retardo medio de los paquetes a nivel RLC.
- `DRB.UEThpDl` y `DRB.UEThpUl`: throughput medio por usuario en enlace descendente y ascendente.

Se muestra la tabla 6.2, donde se puede ver un resumen de la configuración de algunas pruebas básicas a realizar. En los Anexo A, B y C es

posible encontrar cómo llevar a cabo estos escenarios de configuración con los ficheros y opciones disponibles.

Parámetro	Prueba 0 (Referencia)	Prueba 1 (Misma banda, BW reducido)	Prueba 2 (Mayor ganancia UE y atenuación)	Prueba 3 (Banda y SCS mayores, BW reducido)
Banda (3GPP)	n78	n78	n78	n257
Frecuencia central	3 600 MHz	3 600 MHz	3600 MHz	27 000 MHz
Subcarrier spacing (SCS)	30 kHz	30 kHz	30 kHz	120 kHz
Ancho de banda	106 PRBs	24 PRBs	106 PRBs	32 PRBs
Atenuación TX/RX	12 dB	12 dB	20 dB	12 dB
Ganancia UE (TX/RX)	50 dB	50 dB	70 dB	50 dB
Ganancia máx. RX gNB	114 dB	114 dB	114 dB	114 dB

Tabla 6.2: Configuraciones empleadas en la validación funcional con OAIC.

En particular, se han realizado estas pruebas generando tráfico en sentido descendente (DL) mediante la herramienta `iperf`, configurada para enviar un flujo de hasta 100 Mbps. No obstante, como se dijo en la sección 6.1.1, la red limita a 60 Mbps, por lo que esta cifra representa el máximo alcanzable.

Prueba 0. Referencia

Con el escenario de referencia ejecutado, se presentan los resultados obtenidos en la figura 6.2. En ella, se observa que tanto el volumen de SDU en subida (cantidad total de datos que entran al plano de usuario desde la capa IP) como el *throughput* correspondiente son prácticamente nulos, dado que se tiene un cliente `iperf` en el núcleo que genera tráfico hacia el usuario.
En la bajada (DL), el *throughput* se mantiene alrededor de los 60 Mbps de forma relativamente estable, aunque el volumen de SDU en PDCP presenta alta variabilidad, con picos de hasta 100 MB y caídas cercanas a 60 MB. Estas fluctuaciones se deben a la dinámica del tráfico y a la gestión de los *buffers* en las capas inferiores del protocolo. Además, el *delay* de las SDU en la capa RLC (cuánto tiempo permanece una unidad de datos en la capa RLC antes de ser transmitida o procesada) muestra variaciones significativas, causadas por la saturación del *buffer* derivada de la simulación del canal, lo que genera retrasos acumulados hasta la liberación de espacio.

Finalmente, se evidencia una detención del *throughput* y la latencia de las SDUs en la capa RLC, mientras que el volumen de estas en la capa PDCP aumenta considerablemente, alcanzando valores de hasta 10 MB. Esta situación corrobora la congestión progresiva en las capas superiores del protocolo, que impide la correcta transmisión de datos y genera acumulación en los *buffers*, lo que finalmente conduce a la paralización del flujo de datos.

Figura 6.2: Experimentación OAIC con configuración inicial.

Prueba 1. Misma banda y ancho de banda reducido

Si se reduce el ancho de banda del canal de 106 PRBs a 24 PRBs (equivalente a una reducción de 40 MHz a 10 MHz), el comportamiento observado se resume en la figura 6.3. Como se aprecia, tanto el volumen de SDUs como el *throughput* experimentan una caída significativa, pasando de valores medios de aproximadamente 80 MB y 60 Mbps a cerca de 15 MB y 14 Mbps, respectivamente. Sin embargo, ambos parámetros muestran una mayor estabilidad en esta configuración, ya que la reducción del ancho de banda limita la capacidad de transmisión del canal, lo que disminuye la cantidad de datos que pueden enviarse, pero a su vez reduce la congestión en los *buffers*.

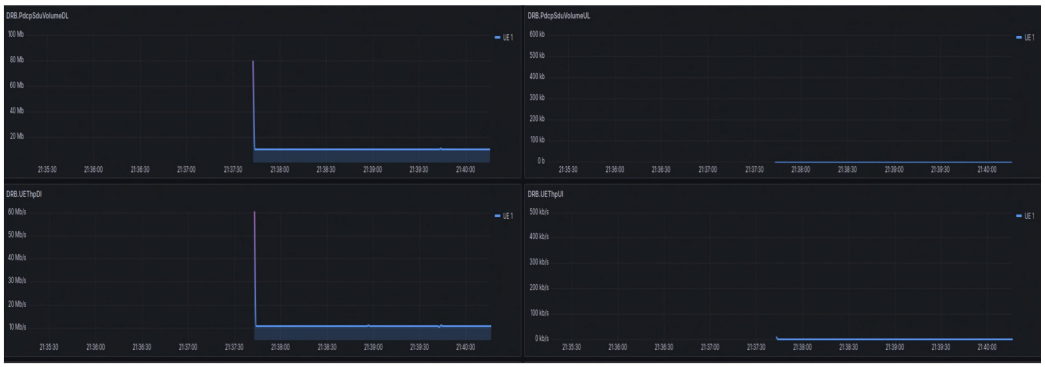


Figura 6.3: Experimentación OAIC con ancho de banda reducido.

Prueba 2. Misma banda, mayor ganancia del UE y mayor atenuación

Si se incrementa la atenuación en transmisión y recepción de 12 dB a 20 dB, compensándose este aumento con una ganancia adicional de 20 dB en el UE para ambos enlaces. En la figura 6.4 se observa que tanto el *throughput* como el volumen de SDU mantienen valores similares a los de la primera prueba. Sin embargo, la saturación del *buffer* de SDU ocurre mucho antes, provocando un corte temprano en el tráfico. Esto se debe a que el aumento de ganancia permite al UE recibir una mayor cantidad de paquetes, incrementando la carga sobre el *buffer*. Aunque la mayor ganancia compensa la atenuación del canal manteniendo la calidad del enlace, el volumen de datos entrante sobrepasa la capacidad de procesamiento del *buffer*, lo que provoca su saturación prematura y afecta a la continuidad del tráfico.
Cabe destacar que este comportamiento es una limitación de la simulación debido a los problemas comentados inicialmente; en un entorno real, el aumento de ganancia mejoraría la calidad de la señal y el rendimiento general del enlace.

Figura 6.4: Experimentación OAIC a mayor ganancia y atenuación.

Prueba 3. Banda y SCS mayores, ancho de banda reducido

Por último, en la figura 6.5 se analiza el rendimiento del sistema al operar en una banda de frecuencia superior, manteniendo la misma configuración que en la prueba de referencia, excepto por el ancho de banda, el cual se reduce de 106 a 32 PRBs. Se observa que el *throughput* ya no alcanza los 60 Mbps registrados en pruebas previas, ahora se tiene en torno a los 35 Mbps. Esta disminución en la tasa de transmisión se refleja también en un menor

volumen de SDU. Además, el retardo en la capa RLC SDU se mantiene constante pero con valores elevados, lo que lleva a pensar en la formación de colas de paquetes. Este comportamiento puede explicarse por la combinación de dos factores: el uso de frecuencias más altas hace que estas se vean más afectadas por la atenuación de canal que otras, lo que provoca una bajada de rendimiento; además, la reducción del ancho de banda también limita la capacidad máxima del canal.

Figura 6.5: Experimentación OAIC en una banda superior (SCS mayor) y ancho de banda reducido.

A partir de los resultados obtenidos, se observa que la *xApp* implementada en el Near-RT RIC permite monitorizar en tiempo real el tráfico y la calidad del enlace, facilitando la detección temprana de problemas como la congestión o la degradación de la conexión. Esta monitorización continua y detallada permite gestionar los recursos de forma más eficiente y proactiva, habilitando decisiones informadas y mecanismos de control dinámico que optimizan el rendimiento de la red y mejoran la experiencia del usuario.

6.3. srsRAN. Monitorización y Control

A continuación, se presentan pruebas centradas en la supervisión y gestión del sistema mediante el Near-RT RIC. La monitorización de métricas se estudia para las dos implementaciones de Near-RT RIC comentadas a lo largo del proyecto, mientras que el control se analiza bajo casos comunes en redes como el control de PRBs.

6.3.1. Consideraciones

Las pruebas presentadas en este trabajo se realizan sobre entornos simulados, donde el comportamiento temporal no es equivalente al tiempo real debido a las características propias de la emulación y la arquitectura del sistema. En el caso de OAIC, por cómo está implementada su arquitectura interna y por el uso de *RFSimulator*, la diferencia entre ambos tiempos no es muy grande. Sin embargo, en srsRAN al emplear mecanismos de comunicación entre componentes basados en ZeroMQ, el tiempo percibido en la emulación transcurre a una velocidad mucho menor que en un escenario real. Esto implica que, si en el mundo real un evento dura un tiempo `T`, en la emulación ese mismo evento puede extenderse a un tiempo `xT` con `x<1`, es decir, el tiempo en la simulación es "más lento" que en la realidad.

Esta ralentización temporal conlleva que los datos recogidos por el sistema durante la prueba, que se generan y monitorizan en tiempo real, se presenten en un intervalo de tiempo aparente menor. De esta forma, el rendimiento medido, en términos de *throughput*, puede parecer más alto del que correspondería en un entorno real. En otras palabras, al "comprimir" la información en un tiempo reducido, el sistema visualiza los mismos datos como si fueran transmitidos en menos tiempo, lo que provoca una sobreestimación relativa del *throughput*.

Además, esta situación se ve influida por la carga de datos generada en las pruebas. Cuanto menor es la cantidad de datos transmitidos, menos tiempo requiere el sistema para procesarlos, lo que reduce la diferencia entre el tiempo emulado y el tiempo real, acercando las métricas medidas a valores más representativos del comportamiento real. Por ejemplo, si se manda un flujo de datos con velocidad baja (como 1 Mbps), se tienden a mostrar resultados más consistentes, mientras que al aumentar la carga (por ejemplo, a 10 Mbps o más), el factor de desaceleración del tiempo se amplía, de forma que en un canal real se verían esos 10 Mbps y en el simulado más.

Como posible mitigación, se ha explorado el uso de técnicas como el *pacing* en `iperf3`, que permite espaciar la transmisión de paquetes para aproximar mejor las métricas de la emulación a las esperadas en un entorno real. Sin embargo, aunque esta estrategia mejora la correspondencia en valores bajos y medios, no logra eliminar completamente las diferencias causadas cuando el flujo es elevado. Por tanto, es importante tener en cuenta estas limitaciones al interpretar los resultados obtenidos, enfocándose en el análisis relativo de las métricas y su evolución bajo distintas condiciones de prueba, más que en los valores absolutos.

En la figura 6.6, se muestra una representación comparativa entre el

tráfico enviado (ajustado a lo que se esperaría en un sistema real) y el tráfico realmente observado en el entorno simulado. Esta figura se incluye con fines ilustrativos, para que el lector pueda contextualizar mejor las diferencias temporales y de rendimiento presentes en este tipo de entornos.

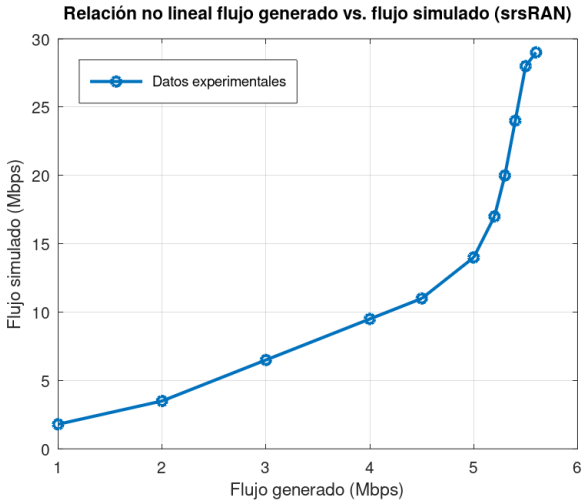


Figura 6.6: Diferencia temporal entre el canal físico real y el canal simulado con ZeroMQ.

6.3.2. Monitorización de Métricas

En este primer apartado, se analiza la capacidad de los distintos Near-RT RIC para proporcionar visibilidad sobre el estado de la red, con especial énfasis en el seguimiento de métricas clave para la toma de decisiones en tiempo casi real. La monitorización de estas resulta esencial para habilitar políticas de optimización dinámica y garantizar una operación eficiente y las *xApps* empleadas para ello se pueden ver en el Anexo F.

Seguimiento con ORAN-SC-RIC

Primero, se evalúa la capacidad de monitorización del Near-RT RIC ORAN-SC-RIC, a través del uso de una *xApp* que recopila el `DRB.UEThpUl/Dl` (kbps) y `RRU.PrbAvailUl/Dl`.

Por otro lado, la conexión con Grafana se realiza directamente desde la red de acceso, donde se visualizan métricas más relacionadas con la indicación de la calidad, como el *throughput* UL/DL, el *Modulation and Coding Scheme* (MCS) UL/DL, el CQI y la *Signal Noise Ratio* (SNR).

En concreto, se va a realizar el siguiente conjunto de pruebas haciendo uso de la *kpm_mon_xapp*, una *xApp* escrita en Python que se conecta al Near-RT RIC para suscribirse a reportes KPM de nodos E2 específicos, recopila métricas de rendimiento de la red según diferentes estilos de reporte, procesa y muestra en consola las indicaciones recibidas con datos detallados de mediciones por métrica y por UE:

Parámetro	**Prueba 0 (Referencia)**	**Prueba 1 (Misma banda con BW reducido)**	**Prueba 2 (Misma banda con BW reducido y mayor ganancia del UE)**	**Prueba 3 (Banda distinta)**
Banda (3GPP)	n3	n3	n3	n2
Frecuencia central	1 842.5 MHz	1 842.5 MHz	1 842.5 MHz	1 930 MHz
Subcarrier spacing (SCS)	15 kHz	15 kHz	15 kHz	15 kHz
Ancho de banda	52 PRBs	25 PRBs	25 PRBs	106 PRBs
Pérdidas	0 dB	0 dB	0 dB	0 dB
Ganancia gNB (TX/RX)	50 dB	50 dB	50 dB	50 dB
Ganancia UE (TX/RX)	50 dB	50 dB	80 dB	50 dB

Tabla 6.3: Configuraciones empleadas en la validación de la monitorización de ORAN-SC-RIC con srsRAN.

Como se observa, la prueba de referencia tiene una idealidad media, por lo que en su máximo no se alcanzarán los 60 Mbps limitantes, sino menos. Para poner en marcha el entorno, se recuerda:

Componente	**Comando de ejecución**
Core 5G	`docker compose up 5gc`
ORAN-SC-RIC (Near-RT RIC)	`docker compose up`
gNB	`sudo ./gnb -c gnb_zmq.yaml e2 --addr="10.0.2.10-bind_addr="10.0.2.1"`
UE	`sudo ip netns add ue1 & sudo ./srsue ue_zmq.conf`
Routing	`sudo ./ip_config.sh`
Grafana	`sudo docker compose -f docker-compose.yml up grafana`
xApp	`sudo docker compose exec python_xapp_runner ./kpm_mon_xapp.py --metrics=DRB.UEThpUl, DRB.UEThpDl,RRU.PrbAvailUl, RRU.PrbAvailDl`
Generación de tráfico	`sudo docker exec -it open5gs_5gc /bin/bash iperf -c 10.45.1.2 -u -b 100M -i 1 -t 100`

Tabla 6.4: Resumen de comandos para la puesta en marcha del entorno de monitorización con srsRAN.

A continuación, se puede ver el comportamiento de este sistema.

Prueba 0. Referencia

En la prueba 0, se realizó una conexión mediante `iperf` con un tráfico descendente de 100 Mbps. Como se muestra en la figura 6.7, la red alcanza rápidamente su capacidad máxima de transmisión, llegando a un tope de 29 Mbps, lo cual indica una saturación del enlace en esas condiciones debido al problema comentado de ZeroMQ. Por otro lado, se observan varios indicadores relevantes:

- El *Modulation and Coding Scheme* (MCS) utilizado es el 27, correspondiente a una modulación de orden $Q_m = 6$, un *Target Code Rate* de 910 y una eficiencia espectral aproximada de 5.3320 bits/s/Hz, lo cual refleja un uso eficiente del canal de radio.
- La *Signal Noise Ratio* (SNR) registrada es 68 dB, lo que indica una excelente calidad de señal.
- El *Block Error Rate* (BLER) es del 0 %, por lo que no se observan pérdidas de bloques de datos.
- El *Time Advance* (TA) es de 0 μs, lo que sugiere que el retardo de propagación es despreciable (obvio al ser una simulación).

En la parte derecha de la figura 6.7 se visualizan correctamente las métricas recolectadas por la *xApp*:

- En *uplink* están disponibles los 52 PRBs correspondientes a un ancho de banda de 10 MHz.
- En *downlink*, sentido en el que se realiza el envío de tráfico con `iperf`, únicamente quedan 8 PRBs disponibles, lo que evidencia el uso de estos para enviar datos.

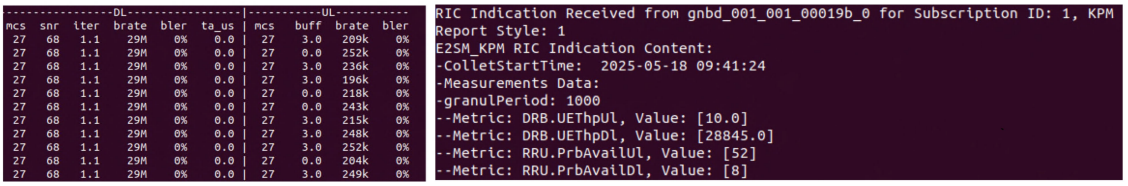


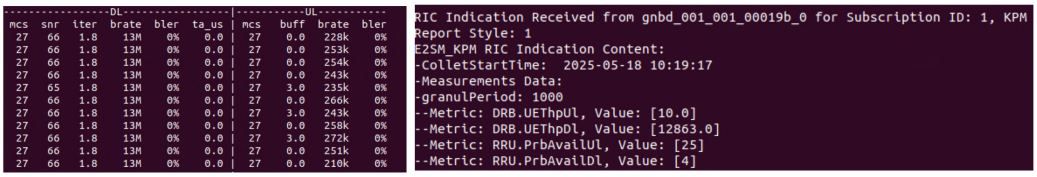
Figura 6.7: Experimentación srsRAN - ORAN-SC-RIC de referencia.

Al observar lo capturado en Grafana (véase la figura 6.8), se corrobora la exposición de los resultados obtenidos, además de una métrica adicional, el CQI. Este tiene el valor más alto que existe.

Figura 6.8: Experimentación srsRAN - ORAN-SC-RIC de referencia (Grafana).

Prueba 1. Misma banda y ancho de banda reducido

Si ahora se reduce el ancho de banda del sistema de 10 MHz (correspondientes a 52 PRBs) a 5 MHz (equivalentes a 25 PRBs), la SNR disminuye ligeramente, pasando de 68 dB a 66 dB, y el *throughput* máximo alcanzado cae a aproximadamente 13 Mbps. Este descenso en el rendimiento se debe principalmente a la reducción de recursos radioeléctricos disponibles, lo cual limita la capacidad del sistema para transmitir datos de manera eficiente. Además, se puede ver con los PRBs disponibles en *uplink* son los 25 totales que hay a repartir, mientras que en *downlink*, solo quedan 4.
Por otro lado, las demás métricas medidas no presentan cambios significativos. Esto se debe a que el entorno en el que se realiza la prueba es simulado y, por tanto, no varían factores como la movilidad, interferencias externas o condiciones del canal radioeléctrico.



DL						UL			
mcs	snr	iter	brate	bler	ta_us	mcs	buff	brate	bler
27	66	1.8	13M	0%	0.0	27	0.0	228k	0%
27	66	1.8	13M	0%	0.0	27	0.0	253k	0%
27	66	1.8	13M	0%	0.0	27	0.0	254k	0%
27	66	1.8	13M	0%	0.0	27	0.0	243k	0%
27	65	1.8	13M	0%	0.0	27	3.0	235k	0%
27	66	1.8	13M	0%	0.0	27	0.0	266k	0%
27	66	1.8	13M	0%	0.0	27	3.0	243k	0%
27	66	1.8	13M	0%	0.0	27	0.0	258k	0%
27	66	1.8	13M	0%	0.0	27	3.0	272k	0%
27	66	1.8	13M	0%	0.0	27	0.0	251k	0%
27	66	1.8	13M	0%	0.0	27	0.0	210k	0%

```
RIC Indication Received from gnbd_001_001_00019b_0 for Subscription ID: 1, KPM
Report Style: 1
E2SM_KPM RIC Indication Content:
-ColletStartTime:  2025-05-18 10:19:17
-Measurements Data:
-granulPeriod: 1000
--Metric: DRB.UEThpUl, Value: [10.0]
--Metric: DRB.UEThpDl, Value: [12863.0]
--Metric: RRU.PrbAvailUl, Value: [25]
--Metric: RRU.PrbAvailDl, Value: [4]
```

Figura 6.9: Experimentación srsRAN - ORAN-SC-RIC con ancho de banda reducido.

Figura 6.10: Experimentación srsRAN - ORAN-SC-RIC con ancho de banda reducido (Grafana).

Este comportamiento ha podido observarse en las figuras 6.9 y 6.10, donde se muestran tanto los parámetros del sistema como las métricas capturadas por la *xApp* a través de la plataforma Grafana.

Prueba 2. Misma banda, mayor ganancia del UE

Siguiendo con un ancho de banda reducido de 5 MHz, se evalúa el efecto de aumentar la ganancia del UE hasta los 80 dB. Como se puede observar, parámetros como el MCS, el BLER, el TA y la SNR permanecen sin cambios apreciables. Sin embargo, sí se observan variaciones en el *throughput* y en el número de PRBs utilizados. El *throughput* experimenta un incremento notable, pasando de 13 Mbps a unos 24 Mbps. Al mismo tiempo, el número de PRBs utilizados disminuye a tan solo 10. Este comportamiento sugiere una mayor eficiencia espectral, posiblemente atribuida a un uso más eficiente del planificador (*scheduler*) del gNB, que puede asignar los recursos radioeléctricos de manera más precisa cuando la ganancia es mayor, minimizando los PRBs necesarios para alcanzar un determinado rendimiento.

Tanto en las métricas capturadas por el ORAN-SC-RIC como en la visualización mediante Grafana se observa esta mejora de rendimiento, tal como se muestra en las figuras 6.11 y 6.12.

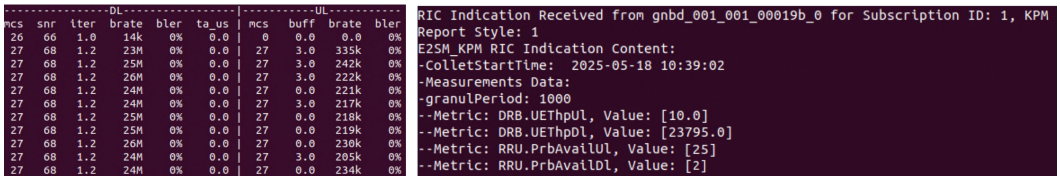


Figura 6.11: Experimentación srsRAN - ORAN-SC-RIC con ancho de banda reducido y aumento de la ganancia del UE.

Figura 6.12: Experimentación srsRAN - ORAN-SC-RIC con ancho de banda reducido y aumento de la ganancia del UE (Grafana).

Prueba 3. Banda mayor

Finalmente, si se modifica la configuración para operar en una banda distinta, pasando de la banda n3 (alrededor de 1 845 MHz) a la banda n2 (alrededor de 1 930 MHz), y se incrementa el ancho de banda de 10 MHz (52 PRBs) a 20 MHz (106 PRBs), se dispone del doble de recursos físicos para la transmisión de datos. Este aumento de capacidad se traduce directamente en una mejora del *throughput*, al permitir una mayor cantidad de información transmitida por unidad de tiempo.
Por otro lado, en condiciones reales, un aumento en la frecuencia de operación podría tener efectos adversos sobre la eficiencia del enlace debido al incremento de la atenuación. Sin embargo, en este entorno simulado, estos factores no tienen un impacto significativo, por lo que el principal efecto observado es positivo, gracias a la ampliación del ancho de banda y al mayor número de PRBs disponibles.

A pesar de esta mejora en capacidad, se observa una ligera disminución

en la SNR. Esta caída puede atribuirse al hecho de que, al duplicar el número de PRBs (de 52 a 106), el sistema no incrementa proporcionalmente la potencia total transmitida; entonces, la potencia disponible por subportadora se reduce, lo que se traduce en una menor SNR por PRB. Esto explica la ligera degradación observada, a pesar del entorno simulado sin presencia de ruido externo significativo.

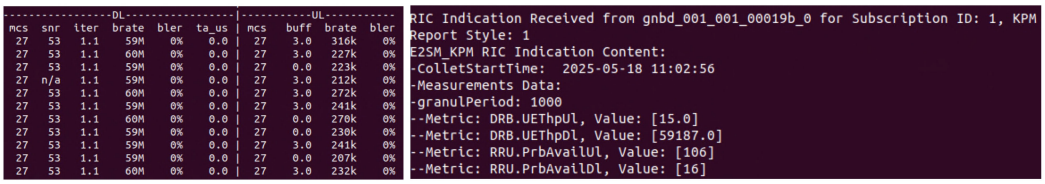


Figura 6.13: Experimentación srsRAN - ORAN-SC-RIC en banda n2 con aumento del ancho de banda.

Figura 6.14: Experimentación srsRAN - ORAN-SC-RIC en banda n2 con aumento del ancho de banda (Grafana).

El ORAN-SC-RIC junto con la *xApp* han demostrado precisión en la recolección y monitoreo en tiempo real de las métricas clave del sistema, facilitando una visión detallada del rendimiento y uso de recursos.

Seguimiento con FlexRIC

Una vez estudiados diferentes escenarios de red y la captura de métricas con ORAN-SC-RIC, se procede a realizar las mismas pruebas con la segunda implementación de un Near-RT RIC, el FlexRIC. Los pasos para la puesta en marcha son los mismos, salvo que ahora la ejecución del Near-RT RIC y de la *xApp* se hacen por medio de `./flexric/build/examples/ric/nearRT-RIC`

y `./flexric/build/examples/xApp/c/monitor/xapp_oran_moni -c ./xapp_mon_e2sm_kpm.conf`, respectivamente.
Además, las métricas a medir se pasan al FlexRIC por medio de un fichero de configuración. Su contenido puede verse en el Anexo G. Como se observa, con este Near-RT RIC es posible capturar de forma exitosa 2 métricas más, el volumen de SDU transmitidas en UL/DL y la tasa de pérdida de paquetes en DL.

En las figuras 6.15, 6.16, 6.17 y 6.18, es posible observar el comportamiento de la red y la toma de medidas del FlexRIC.

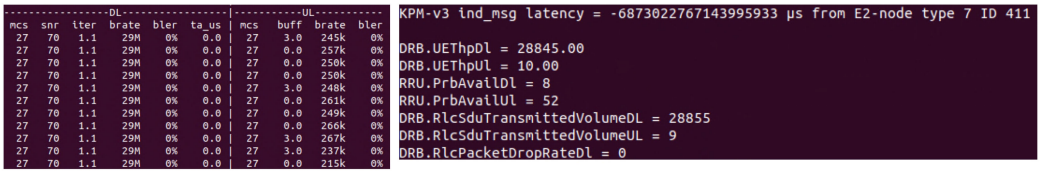


Figura 6.15: Experimentación srsRAN - FlexRIC de referencia.

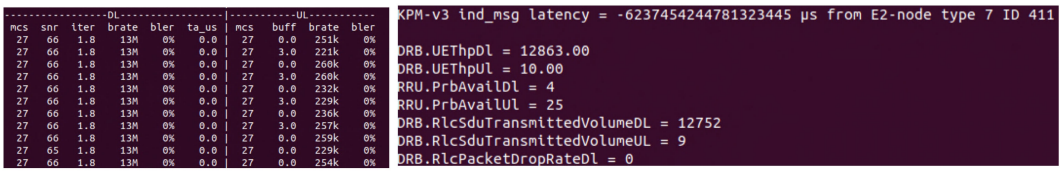


Figura 6.16: Experimentación srsRAN - FlexRIC con BW reducido.

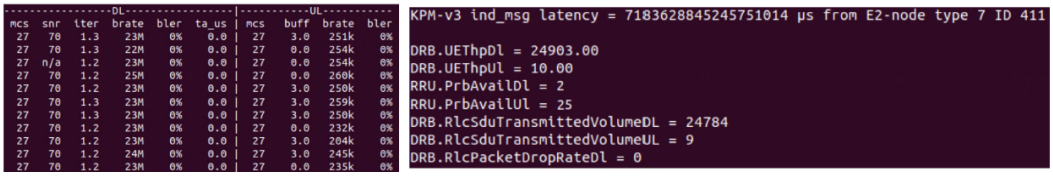


Figura 6.17: Experimentación srsRAN - FlexRIC con ancho de banda reducido y aumento de la ganancia del UE.

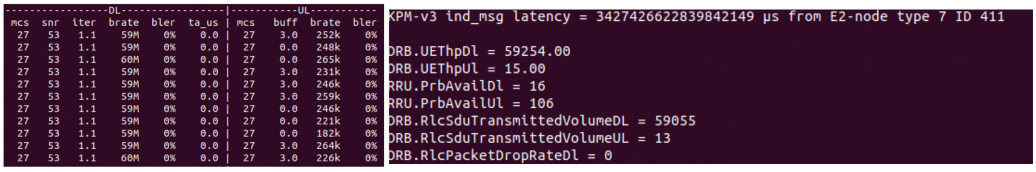


Figura 6.18: Experimentación srsRAN - FlexRIC en banda n2 con aumento del ancho de banda.

Como resulta evidente, la red se comporta de igual manera que en el caso de usar el ORAN-SC-RIC, ya que éste no influye en ella. Por otro lado, las medidas también se parecen bastante a las observadas en la red, a diferencia de que ahora es posible observar tanto la pérdida de paquetes en *downlink*, la cual resulta nula, como el volumen de SDU transmitidos.

6.3.3. Control de Recursos Radioeléctricos

En la sección 6.3.2 se analizaron las capacidades de monitorización de algunas métricas ofrecidas por las diferentes implementaciones del Near-RT RIC. A partir de ello, en esta sección se estudia el control de recursos radioeléctricos, centrándose en cómo dichas métricas pueden ser empleadas para tomar decisiones que optimicen el uso del espectro y mejoren la calidad del servicio.

El control de recursos radioeléctricos, como la asignación dinámica de PRBs, es una funcionalidad clave del Near-RT RIC que permite operar la red de forma más eficiente y adaptativa. En esta sección se presenta un escenario de este tipo y una implementación concreta de una *xApp* desarrollada sobre ORAN-SC-RIC, diseñada para detectar situaciones de congestión en el canal de bajada o DL con un UE y reaccionar ajustando automáticamente la cantidad de recursos asignados según el estado actual de la red.

Control de PRBs con ORAN-SC-RIC

Antes de profundizar en la experimentación y desarrollo de la *xApp*, es importante aclarar que la funcionalidad de control de recursos en cuanto a la asignación dinámica de PRBs, se basa en *commits* anteriores tanto del repositorio srsRAN [63] como en el de ORAN-SC-RIC [35].

Dejando destacado lo anterior, en este primer apartado se ha utilizado una *xApp* denominada *simple_xapp.py*, basada en el *framework* del Near-RT RIC ORAN-SC-RIC. Esta aplicación tiene como objetivo realizar un control dinámico de los recursos físicos asignados a los usuarios (PRBs) en función del volumen de datos transmitidos en el enlace descendente.

Para ver el código de esta *xApp* puede acceder al Anexo G. A continuación, se describe su funcionamiento:

- **Descripción general:** La *xApp* `MyXapp` se conecta a un nodo E2 para recibir datos de rendimiento de usuarios (UE) en forma de métricas KPM. Su función principal es gestionar el tráfico descendente (DL) de un UE, ajustando dinámicamente los recursos de red asignados según el volumen de datos transmitidos.

- **Inicialización:**
  - Inicializa variables como `ue_dl_tx_data` para almacenar los datos de tráfico transmitido por UE.
  - Establece límites predeterminados para la relación PRB y un umbral de datos de 4 MB para cambiar la asignación de recursos.
- **Suscripción a métricas KPM:**
  - La *xApp* se suscribe a las métricas KPM del nodo E2 utilizando un estilo de reporte (por defecto, estilo 4).
  - Recibe datos periódicos del volumen de datos transmitidos por cada UE.
- **Lógica de control:**
  - Si el volumen de datos transmitidos por un UE supera un umbral de 4 MB, se ajusta la relación PRB del UE.
  - La *xApp* envía una solicitud de control al nodo E2 para cambiar la relación PRB entre los valores mínimo y máximo.
- **Método de inicio:**
  - Utiliza el decorador `@xAppBase.start_function` para marcar la función de inicio.
  - Configura y suscribe a las métricas KPM del nodo E2, asegurando que se procesen y se ajuste el uso de recursos de manera continua.

Para testear la *xApp*, se deben arrancar los distintos componentes de la red en el siguiente orden:

1. **Lanzamiento del *core* 5G con Open5GS (dockerizado)**: se arranca el núcleo de red haciendo uso de la versión en contenedores. A diferencia de pruebas anteriores, en esta ocasión el fichero `.env` de configuración debe incluir la ruta a un fichero `.csv` que contiene los datos de suscripción de dos usuarios (IMSI, IMEI, OPC, etc.). De esta forma, ambos UEs están definidos desde el arranque del sistema.
2. **Puesta en marcha de ORAN-SC-RIC en su versión dockerizada**.
3. **Inicio de la gNB**: se lanza una instancia de la gNB con la dirección IP 10.0.2.10 del ORAN-SC-RIC y configurada para escuchar en la interfaz 10.0.2.1.

4. **Arranque de los dos UEs**: se lanzan dos instancias de UE. El primero sigue la configuración detallada en el Anexo E. El segundo UE utiliza un archivo de configuración similar, pero con las siguientes modificaciones:

   - Se cambian los puertos ZeroMQ: transmisión por el puerto 2201 y recepción por el 2200.
   - En la sección `[usim]` del fichero de configuración, se introducen los datos del segundo usuario (IMSI, IMEI, OPC, K, etc.).

   Ambos UEs permanecerán desconectados de la red hasta que se active el entorno *GNU Radio* [70].

5. **Configuración de los canales mediante *GNU Radio***: se debe diseñar un esquema `.grc` que defina los canales de transmisión (véase la figura 6.19:

   - Canal descendente (de gNB hacia los UEs) — diagrama superior.
   - Canal ascendente (de los UEs hacia la gNB) — diagrama inferior.

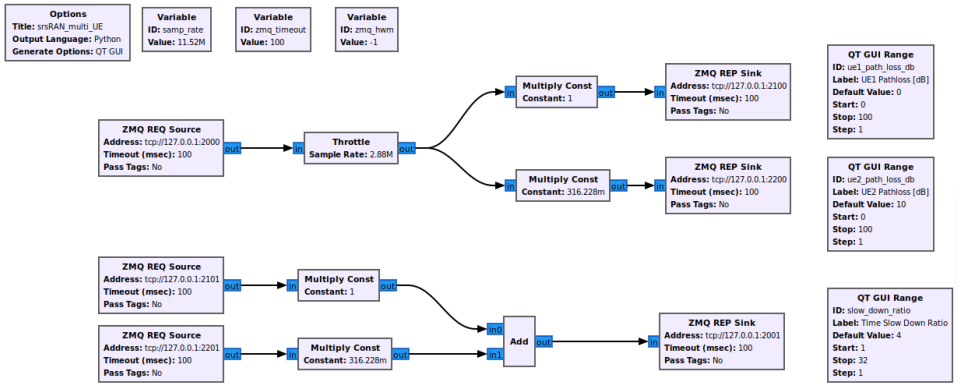


Figura 6.19: Esquema .grc para canal ascendente y descendente en GNU Radio.

   Este esquema comparte el canal físico simulado de ZeroMQ entre los dos UE y permite, en pruebas futuras, introducir condiciones como pérdida de señal (*pathloss*) o latencia. Sin embargo, para esta prueba concreta, estas opciones se mantendrán desactivadas.
   Para arrancarlo, se debe pulsar `Execute` en su interfaz gráfica. Seguidamente, ambos UEs se conectan automáticamente a la red, estableciéndose la sesión PDU de cada uno y adquiriendo las direcciones IP 10.45.1.2 y 10.45.1.3, respectivamente.

6. **Configuración del enrutamiento**: antes de iniciar ningún tipo de tráfico, se ejecuta el *script ip_config.sh* comentado en el capítulo 5 para establecer las rutas necesarias entre los distintos elementos de la red.

Con todos los componentes en marcha y las condiciones de simulación de referencia mostradas en la tabla 6.3, se puede proceder a generar tráfico descendente (actualmente esta solución soporta control únicamente en este sentido) desde el *core* hacia los UEs. Al hacer uso de `iperf`, no es necesario lanzar ningún servidor en escucha, por lo que directamente se lanzan clientes `iperf` desde el contenedor del *core* con:

```
sudo docker exec -it open5gs_5gc /bin/bash
UE 0: iperf -c 10.45.1.2 -u -b 5M -i 1 -t 100
UE 1: iperf -c 10.45.1.3 -u -b 5M -i 1 -t 100
```

Donde:

- `-c 10.45.1.x`: dirección IP del UE cooncreto.
- `-u`: indica uso del protocolo *User Datagram Protocol* (UDP).
- `-b 5M`: caudal de 10 Mbps.
- `-i 1`: resultados cada segundo.
- `-t 100`: duración de 100 segundos.

Finalmente, se lanza la *xApp* con el siguiente comando desde el directorio `oran-sc-ric/xApps/python/`:

```
sudo docker compose exec python_xapp_runner ./simple_xapp.py
```

Nota: *debido a los efectos del canal simulado, comentados en la sección 6.3.1, las métricas captadas por los mecanismos de monitorización de red no reflejan de forma exacta el tráfico real generado por las aplicaciones. Por ejemplo, cuando solo un UE envía tráfico, un flujo real de 10 Mbps puede aparecer como 29 Mbps en las métricas de la red. En el caso de dos UEs enviando simultáneamente 5 Mbps cada uno, el sistema puede mostrar aproximadamente 15 Mbps por usuario. Esto sugiere que el canal simulado tiene un límite práctico de unos 30 Mbps agregados (se usan las condiciones de referencia mostradas en la tabla 6.3). Cuando hay un único flujo, los retardos de cola y almacenamiento intermedio pueden acumular datos en las capas inferiores, generando picos aparentes de caudal mucho mayores. Con múltiples flujos, el canal se reparte y se estabiliza en torno a ese límite máximo, mostrando valores más próximos al tráfico real.*

En las figuras 6.20 y 6.21 se puede observar el comportamiento de esta *xApp*. Al inicio, marcado en verde en la figura 6.20, se observa que cada UE recibe aproximadamente 15 Mbps, lo que equivale a un total de 30 Mbps para ambos UEs, dado que el canal está limitado a esta velocidad por las

condiciones de simulación establecidas. Esta distribución de 15 Mbps por UE se debe a que los recursos del canal se comparten entre ambos UEs. Como se mencionó al principio de la sección 6.3.3, la funcionalidad de control se encuentra actualmente en desarrollo en las versiones más recientes de los repositorios de srsRAN [63] y ORAN-SC-RIC [35], pero está soportada en *commits* anteriores de estos (como el usado aquí) donde el *slicing* aún no ha sido implementado. Por lo tanto, en este contexto, se puede considerar que ambos UEs están en el mismo *slice* desde el inicio.
Cuando ambos UEs exceden los 4 MB acumulados, se activa el control, limitando su uso de PRBs entre el 1 % y el 10 % de los recursos disponibles en ese momento (al principio, siempre hay 52 PRBs disponibles debido a un ancho de banda de 10 MHz). Este control reduce el flujo a 6.8 MB, como se muestra en naranja. En la *xApp*, este momento se traduce en un uso de `max_prb_ratio1 = 1` y `cur_ue_max_prb_ratio = max_prb_ratio1 = 10`. Tras la activación del control, la métrica que registra este volumen de datos se resetea para ambos UEs (representado en color rosa), y el flujo de datos continúa con su funcionamiento normal.

En la figura 6.21, que muestra el flujo segundos después, se puede ver en amarillo como ambos vuelven a alcanzar unos 4.2 MB acumulados, por lo que salta de nuevo la lógica de control y permite el uso del 100 % de los PRBs disponibles en ese momento (`cur_ue_max_prb_ratio = max_prb_ratio2 = 100`). Este comportamiento demuestra la efectividad del mecanismo de control propuesto para mantener un reparto equilibrado de los recursos radio entre múltiples usuarios.

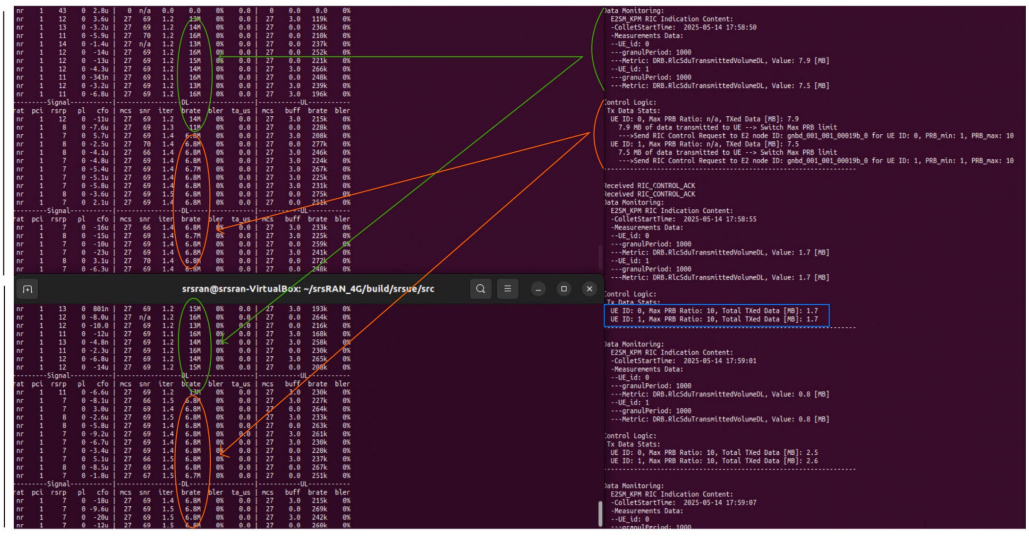


Figura 6.20: Funcionamiento de *simple_xapp.py* (I).

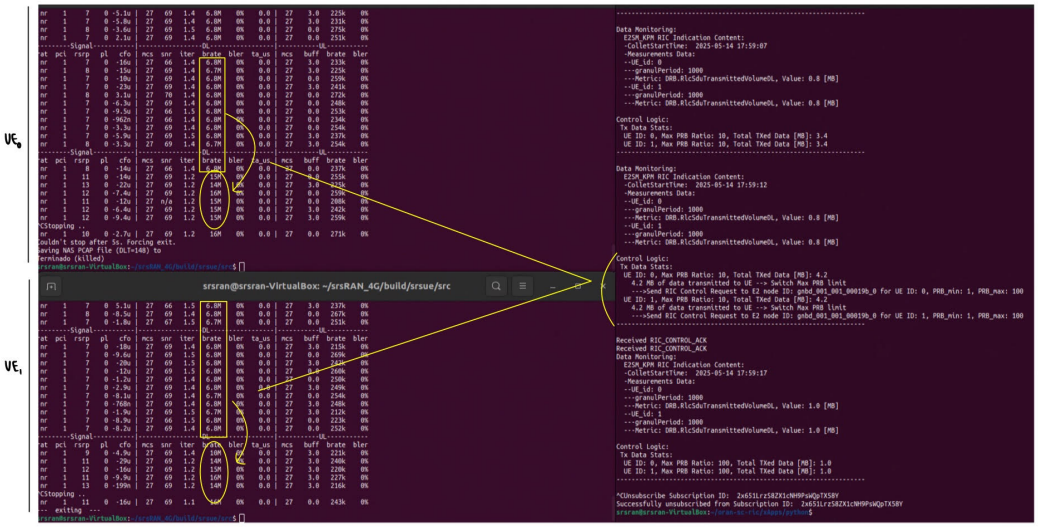


Figura 6.21: Funcionamiento de *simple_xapp.py* (II).

En el ejemplo mostrado inicialmente, se suponía que ambos UEs estaban en condiciones idénticas de canal. A continuación, se introduce una pequeña atenuación artificial de 2 dB mediante un bloque de *pathloss* en *GNU Radio* aplicado únicamente al UE 0, con el objetivo de observar cómo afecta esta desventaja a la asignación de recursos y al comportamiento del mecanismo de control implementado en la *xApp*. Dado que ambos UEs siguen estando en la misma *slice* de la gNB, el mecanismo de control se aplica sobre los recursos compartidos entre los dos UEs.

Tal y como se observa en la figura 6.22, remarcado en verde, al inicio el UE 0 experimenta una peor calidad de canal debido a la ausencia del *pathloss*, lo que se traduce en un menor rendimiento y, por tanto, en una menor tasa de transmisión efectiva. Como consecuencia, el UE 1, que tiene un mejor rendimiento inicial, alcanza antes el umbral de 4 MB acumulados. Esto activa el mecanismo de control, que reduce su porcentaje de uso de PRBs a entre un 1 % y un 10 % (zona marcada en naranja). Al disminuir la asignación de recursos al UE 1, parte de estos quedan disponibles para el UE 0, que comienza a beneficiarse de un mayor uso de PRBs, mejorando así su rendimiento.

Este intercambio de recursos provoca que, más adelante, sea el UE 0 quien alcanza el umbral de 4 MB y, en consecuencia, se le aplica el mismo mecanismo de control, limitando su acceso a los recursos. En paralelo, el UE 1 vuelve a alcanzar por segunda vez dicho umbral tras haber recuperado un mayor porcentaje de PRBs durante el descenso del otro usuario, y se le

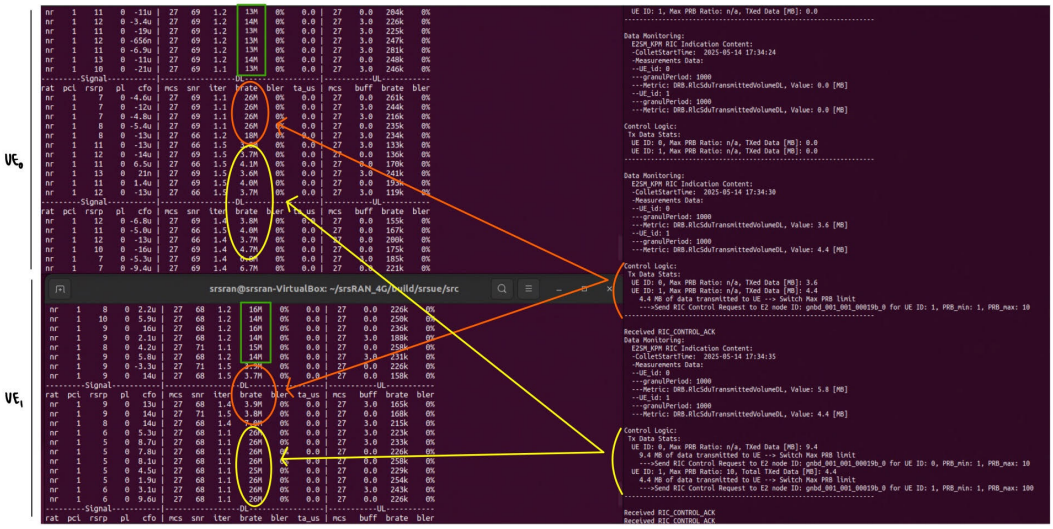


Figura 6.22: Funcionamiento de *simple_xapp.py* con *pathloss* (I).

vuelve a controlar el uso, pero esta vez se le da uso del 100 %. Finalmente, en la figura 6.23, se puede observar cómo, hacia el final del experimento (zona en rosa), el UE 1 alcanza nuevamente el umbral de datos acumulados antes que el UE 0, reflejo de su mayor rendimiento relativo. Como resultado, ambos usuarios se encuentran bajo control, con una asignación de recursos entre el 1 % y el 10 %, lo que provoca que sus rendimientos se igualen.

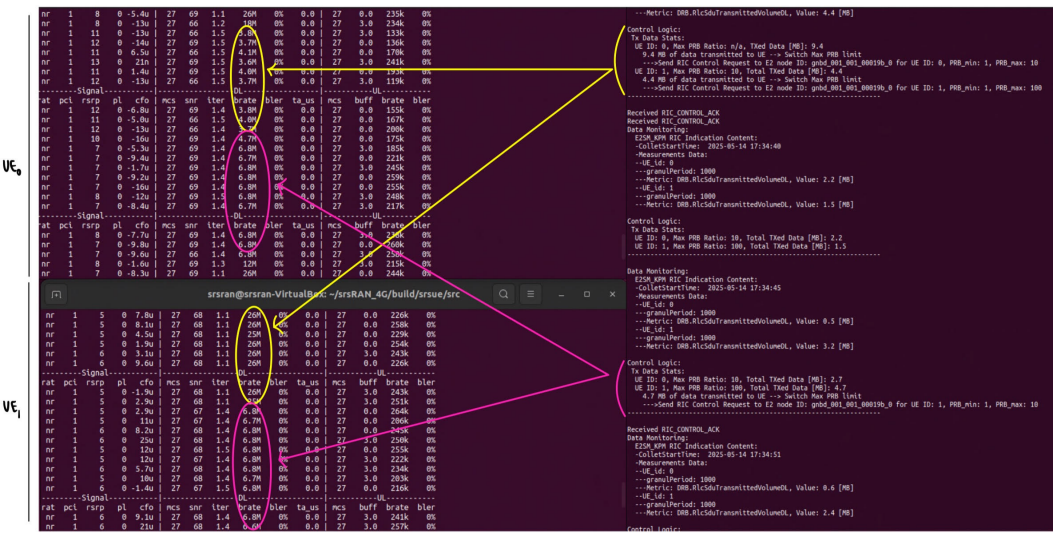


Figura 6.23: Funcionamiento de *simple_xapp.py* con *pathloss* (II).

Este comportamiento confirma de nuevo que la lógica de control aplicada por la *xApp* permite equilibrar de forma dinámica la asignación de recursos radio entre usuarios con diferentes condiciones de canal, penalizando temporalmente a aquellos que consumen más volumen de datos y favoreciendo a los que menos han transmitido, logrando así una mayor equidad en el uso de la red.

Implementación de una *xApp* para detección de congestión

Partiendo del enfoque descrito en la sección anterior, se ha desarrollado una nueva *xApp* orientada específicamente a la detección y mitigación de congestión en el canal de bajada (DL). Esta implementación también se basa en el *framework* del Near-RT RIC ORAN-SC-RIC y emplea la interfaz E2SM-KPM para la recolección periódica de métricas de rendimiento.

A diferencia de la anterior, cuyo objetivo era el control dinámico de los PRBs en función del volumen de datos transmitido, esta nueva *xApp* se centra en identificar condiciones de congestión de un (UE) y actuar en consecuencia. Para ello, analiza el tráfico descendente y los PRBs disponibles y, cuando detecta congestión, ajusta de manera dinámica la asignación máxima de PRBs para dicho usuario, con el fin de proteger la red.

La aplicación está implementada en Python, extendiendo la clase base *xAppBase*, que facilita la gestión de suscripciones, el procesamiento de informes y el envío de comandos de control al nodo E2. El funcionamiento de la *xApp*, junto con su código fuente completo, se presenta en el Anexo G y puede resumirse en las siguientes fases:

1. **Inicialización del entorno y parámetros**:
   - Al ejecutar `main.py`, se inicializan las variables globales:
     - `ue_th_dl`, `ue_prb_avail_dl`: métricas por UE, donde `ue_th_dl` representa el *throughput* en DL por UE y `ue_prb_avail_dl` representa los PRBs disponibles en DL por UE.
     - `cur_ue_max_prb_ratio`: estado actual de la cuota máxima de PRBs por UE.
   - Se definen los umbrales para detección de congestión:
     - `dl_tx_data_max_threshold_kbps = 28500` (kbps)
     - `dl_prb_avail_min_threshold = 15` (PRB)
   - Se establecen los parámetros de control:
     - `min_prb_ratio`: ratio mínimo de PRBs por UE, utilizado para definir el umbral inferior de recursos asignados al UE. Se establece al 1 %.

- `max_prb_ratio1`: ratio máximo de PRBs por UE en la primera fase de control, utilizado cuando se detecta congestión. Se establece al 10 %.
- `max_prb_ratio2`: ratio máximo de PRBs por UE en la fase de restauración, utilizado para incrementar progresivamente los recursos después de aliviar la congestión. Se establece al 100 %.

2. **Configuración y envío de la suscripción KPM**:
   - Se llama a la función `subscribe_report_service_style_4()` para establecer la suscripción:
     - Configura el nodo E2 como destino de la suscripción.
     - Define la periodicidad de los informes mediante el parámetro `report_period`.
     - Selecciona los UEs a monitorizar mediante el parámetro `matchingUeConds`.
     - Aplica un filtro por calidad de señal (por ejemplo, `ul-rSRP <threshold`).
     - Registra la función callback `my_subscription_callback()` para que se ejecute automáticamente cuando lleguen nuevos informes.
3. **Espera de mensajes → llega un** `Indication`:
   - El programa espera pasivamente la llegada de nuevos informes del nodo E2.
   - Cuando llega un nuevo mensaje `Indication`, se almacena en la variable `previous_indications` para poder acceder a las métricas de iteraciones anteriores.
   - Luego, se ejecuta automáticamente la función `my_subscription_callback()`.
4. **Procesamiento en la función** `my_subscription_callback`:
   - La función `my_subscription_callback()` realiza las siguientes tareas:
     - Extrae el cuerpo del mensaje de la indicación.
     - Itera por cada UE contenido en el informe.
     - Lee las métricas: `DRB.UEThpDl` (throughput en DL) y `RRU.PrbAvailDl` (PRBs disponibles).
     - Actualiza los diccionarios `ue_th_dl` y `ue_prb_avail_dl`.
5. **Evaluación de congestión y decisión de control**:
   - Se evalúan las métricas para detectar congestión:

- Si `DRB.UEThpDl` ¿umbral alto `dl_tx_data_max_threshold_kbps` y `RRU.PrbAvailDl` ¡umbral bajo `dl_prb_avail_min_threshold`, se detecta congestión.

6. **Aplicación de acción correctiva**:
   - Se llama a la función `control_slice_level_prb_quota()` para reducir la cuota máxima de PRBs del UE congestionado, aplicando `max_prb_ratio1`.
   - Se envía un mensaje de control al nodo E2 usando el protocolo E2SM-RC.
7. **Monitorización y restauración progresiva**:
   - En rondas siguientes (tras recibir más informes), si ya no hay congestión:
     - Se espera un tiempo (`sleep(10)` segundos).
     - Se aumenta progresivamente la cuota de PRBs (+2, +4, +4, +5), enviando nuevos mensajes de control con los valores restaurados.
     - Pasado un tiempo, se restablece al 100 % la cuota de asignación posible.
8. **Repite el ciclo con cada nuevo** `Indication`.

Este ciclo de monitorización y control se repite continuamente, permitiendo una respuesta adaptativa a las condiciones dinámicas de la red. La *xApp* imprime continuamente los valores de `ue_th_dl` y `ue_prb_avail_dl` por pantalla, y actúa tan pronto como se detecta congestión, aplicando las restricciones establecidas. En la figura 6.24 se muestra un diagrama de flujo aclaratorio de este comportamiento.

Para hacer una prueba con esta nueva *xApp*, se siguen prácticamente los mismos pasos que los comentados en la prueba anterior, con la diferencia de que ahora:

1. El *Core* 5G de Open5GS tiene ne su `.env` sólo un usuario.
2. No hace falta *GNU Radio*. Al lanzarse el UE, este se conecta ala red y recibe su IP 10.45.1.2.

Una vez lanzados correctamente todos los componentes de red, se genera el tráfico descendente desde el *core* hacia el UE. Primero, se inicia el servidor `iperf` de escucha en el UE y luego se lanza el cliente `iperf` dentro del contenedor de *Open5GS* ejecutando:

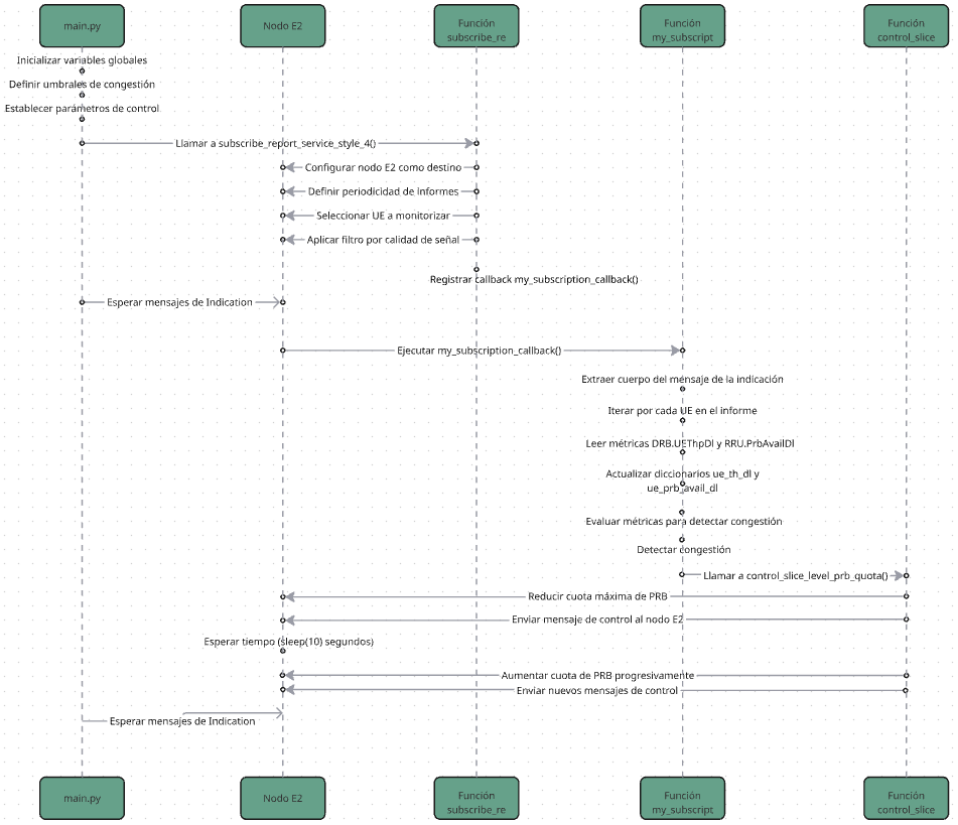


Figura 6.24: Diagrama de flujo sobre el funcionamiento de la *xApp* implementada.

```
sudo docker exec -it open5gs_5gc /bin/bash
iperf -c 10.45.1.2 -u -b 5M -i 1 -t 100
```

Como se observa, se va a empezar enviando un flujo de tráfico de 5 Mbps, con el objetivo de evaluar el comportamiento de la *xApp* en una situación en la que no se alcanza el umbral de congestión establecido (28 Mbps). Esta primera fase permite comprobar que la *xApp* simplemente monitoriza las métricas de red sin activar ningún mecanismo de control, ya que no se detecta congestión. Más tarde, se añade un segundo flujo de tráfico `iperf` paralelo con otros 10 Mbps para llevar al sistema al estado de çongestión simulada".

Nota: *los 5 Mbps enviados serán percibidos como aproximadamente 10-11 Mbps.*

En la figura 6.25 se puede ver cómo al inicio, cuando no hay tráfico, la *xApp* monitoriza las métricas mostrando 0 *throughput* y 52 PRBs disponibles

(52 PRBs correspondientes a 10 MHz de ancho de banda).

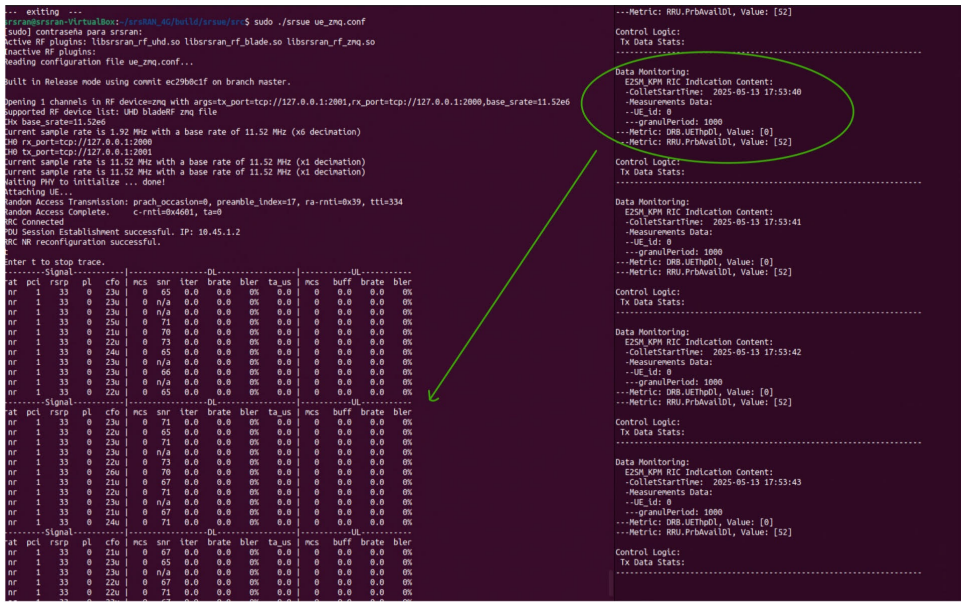


Figura 6.25: Condiciones iniciales en la *xApp*.

Por otro lado, si se inicia tráfico se puede observar el comportamiento esperado en la figura 6.26. Al inicio, el *throughput* se mantiene por debajo del umbral de congestión y la *xApp* no actúa sobre la asignación de recursos (rodeado en verde en la figura). Sin embargo, tras activar el segundo flujo de tráfico, el *throughput* total sube rápido hasta que los 28 Mbps y se detecta congestión (color naranja). En ese momento, la *xApp* activa su mecanismo de control y reduce inmediatamente el parámetro `cur_ue_max_prb_ratio` al valor inicial de recuperación `max_prb_ratio1 = 10` (representado en azul), observándose una reducción del tráfico a 6.9 Mbps. Esta acción significa que el UE afectado puede utilizar en este momento únicamente entre el 1 % (valor mínimo `min_prb_ratio`) y el 10 % de los PRBs disponibles en ese instante. Cabe destacar que si en el momento de la congestión hay 8, como se ve en la figura, en el instante en el que realiza la acción puede haber una ligera variación de estos, por lo que el UE usa entre el 1 %-10 % de 8-10 PRBs aproximadamente.

Durante los siguientes segundos, se inicia la fase de recuperación. En cada intervalo de 5 segundos (cada uno destacado de un color diferente), el valor de `cur_ue_max_prb_ratio` se incrementa en 2 unidades, luego 4, 4, y finalmente en 5 unidades, lo que permite al UE usar progresivamente un mayor porcentaje de los PRBs disponibles, permitiendo que el sistema se adapte gradualmente a las condiciones de congestión sin causar fluctuacio-

nes abruptas en el rendimiento. Al finalizar esta fase de restauración, el rendimiento alcanzado dependerá de los PRBs disponibles en ese momento. Es posible que se logre un rendimiento medio, como se observa en la figura, o bien que se alcance un estado de congestión si aún permanecen datos acumulados en el buffer de recepción del UE debido a un mal canal (el mismo motivo que la incoherencia del *throughput*). Este enfoque gradual permite evitar picos de demanda de recursos que puedan sobrecargar nuevamente la red, favoreciendo un equilibrio entre la capacidad de recuperación del UE y la estabilidad general del sistema.

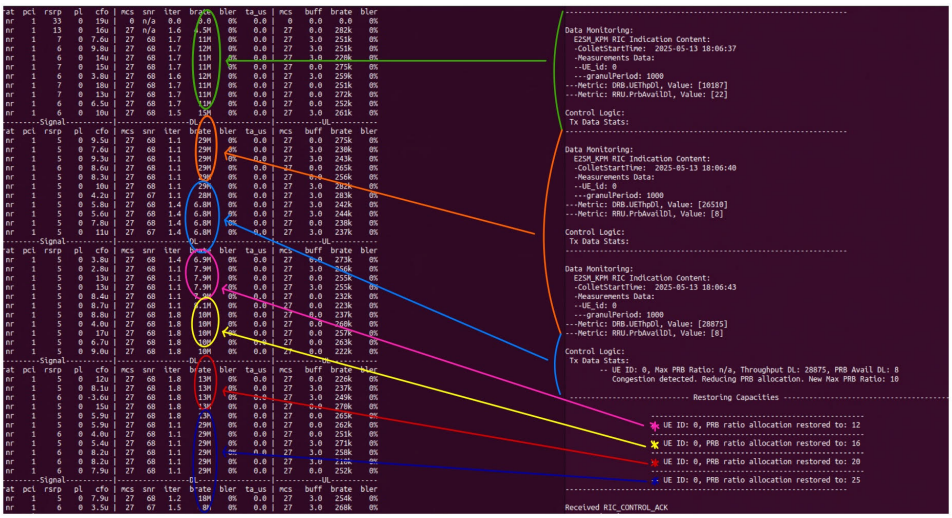


Figura 6.26: Funcionamiento de la *xApp*.

En resumen, este proceso de incremento progresivo de la asignación de recursos permite no solo restaurar el *throughput* del UE de manera controlada, sino también optimizar el uso de los PRBs disponibles en la red, mejorando la eficiencia del sistema sin comprometer su estabilidad general.

Una vez transcurrido el tiempo de recuperación gradual, se restablecerá la asignación de PRBs al 100 %, simulando que ha pasado un período de normalización y que la red puede volver a operar con su capacidad total. Al estar usando ZeroMQ en el canal simulado, es posible que haya paquetes en cuello de botella, y de nuevo lleguen todos a la vez y se vuelva a producir congestión, lo cual en un esceario real no ocurriría.

6.3.4. Comparativa

En esta sección se analizan las diferencias de desempeño entre las pruebas de monitorización y control realizadas con los dos entornos evaluados. Por un lado, el despliegue basado en OAIC que emplea FlexRIC exclusiva-

mente para recopilar métricas; por otro, la solución srsRAN la cual combina FlexRIC para la monitorización y ORAN-SC-RIC para llevar a cabo tareas de control de recursos.

Monitorización. Ambas plataformas registran con precisión las métricas fundamentales de la red; sin embargo, FlexRIC ofrece un conjunto de indicadores más amplio, lo que facilita un análisis detallado del estado del sistema, supervisando métricas adicionales tales como la tasa de pérdida de paquetes en sentido descendente o el volumen de datos recopilado en ambos sentidos. Esta mayor granularidad resulta especialmente útil en entornos de laboratorio o escenarios de investigación donde se requiere visibilidad exhaustiva del plano de usuario.

Control de recursos. En la vertiente de control, *ORAN-SC-RIC* ha demostrado estar preparado para gestionar la asignación dinámica de PRBs, proporcionando mecanismos básicos de orquestación radio. En la versión actual, *FlexRIC* todavía no implementa esa funcionalidad, ya que su foco principal sigue siendo educativo y de monitorización; no obstante, la hoja de ruta del proyecto contempla la incorporación de políticas de asignación de recursos e incluso la creación de *slices*, lo que situaría a *FlexRIC* en un nivel funcional por encima de la aproximación de *ORAN-SC-RIC* en futuros lanzamientos.

Síntesis. En el estado presente de desarrollo, *FlexRIC* destaca por la profundidad de monitorización que ofrece, mientras que *ORAN-SC-RIC* aporta capacidades de control que ya pueden emplearse en pruebas de optimización radio. La elección de una u otra solución dependerá, por tanto, de las necesidades del caso de uso: visibilidad detallada y extensible frente a control inmediato de recursos.

6.4. Escenario real

Con el objetivo de validar el funcionamiento del Near-RT RIC en un entorno no virtualizado, se ha diseñado un escenario realista en el que se integran todos los componentes clave de una red 5G en condiciones controladas pero físicas. El UE está soportado mediante un módem Quectel RM500Q-GL conectado a un NUC 13ANH, mientras que en otro NUC igual se ejecutan tanto la estación base (gNB), basada en srsRAN, como el *core* de Open5GS, en su versión nativa (no dockerizada). A este segundo NUC se conecta un SDR 2901, que se introduce en una jaula de Faraday con el fin de evitar radiaciones no deseadas en bandas licenciadas (5G), asegurando así el cumplimiento normativo. En este entorno, a diferencia de los escenarios virtuales, el canal radio no se simula mediante ZeroMQ o *RFsimulator*, sino

que se emplea un canal físico real, lo que permite estudiar el comportamiento de la red de manera más fiel a su operación en condiciones reales.

Además, en el mismo NUC que alberga tanto la gNB como el *core*, se despliega también el ORAN-SC-RIC, sobre el cual se ejecuta la *xApp simple_xapp.py*. Esta aplicación permite validar, en una única prueba integrada, tanto las funcionalidades de monitorización de métricas como las capacidades de control sobre los PRBs del sistema, verificando así la operatividad del Near-RT RIC en un entorno físico no virtualizado.

A continuación, se muestra una figura representativa del montaje llevado a cabo en las instalaciones de la Universidad, así como el direccionamiento empleado en dicho escenario. Cabe destacar que, en este caso, se emplea el *Public Land Mobile Network* (PLMN) 90170 en lugar del 00101 utilizado en los entornos simulados. Este último es el valor generalmente usado para experimentación interna, mientras que el 90170, aunque también pertenece a un rango especial reservado, fue utilizado por estar previamente configurado en componentes como el *core* debido a pruebas anteriores realizadas por el equipo de investigación. Al tratarse de un identificador privado, resultaba igualmente válido para el entorno de pruebas desplegado. Además, se configura la gNB para trabajar con un ancho de banda de 20 MHz y en banda 78, con el fin de tener un rendimiento representativo.

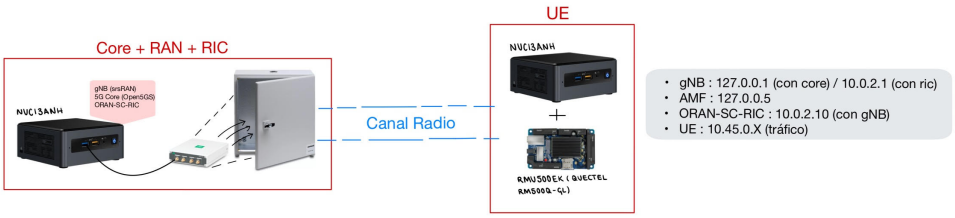


Figura 6.27: Montaje del escenario real.

Una vez completado el montaje descrito, se procede a realizar una prueba de conectividad. En la figura 6.28, se observa cómo la gNB se conecta correctamente al AMF, es decir, al núcleo de la red. Por otro lado, en la parte derecha de la misma figura, se muestra el UE, previamente registrado en la red, el cual recibe la dirección IP 10.45.0.1. A partir de este momento, se verifica la conectividad en la red mediante una prueba de *ping*, cuyos resultados se pueden ver en la parte inferior de la figura 6.28. Como se aprecia, la prueba se realiza con éxito, confirmando así el correcto funcionamiento de la red.

Una vez verificado el correcto funcionamiento del conjunto formado por el UE, la RAN y el CN, se procede a incorporar el Near-RT RIC, concretamente el ORAN-SC-RIC, a la red. Tal como se describe en el capítulo 5, este

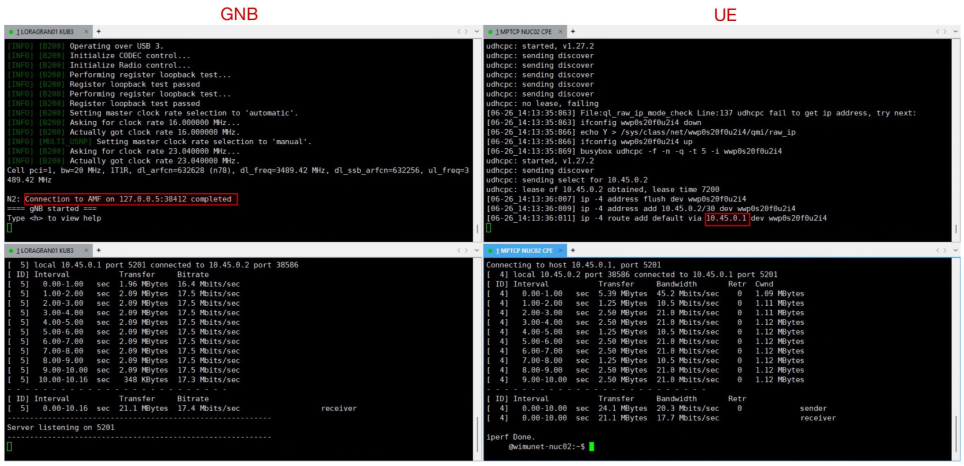


Figura 6.28: Prueba inicial de conectividad en la red.

componente se descarga y despliega siguiendo los pasos correspondientes. En la figura 6.29 se muestra el resultado de esta integración. Tras lanzar el RIC junto al resto de componentes del sistema, se observa que la gNB se conecta correctamente al núcleo de red, lo cual queda reflejado en el mensaje resaltado en rojo en este último. Además, en la parte derecha de la imagen se puede comprobar que el Near-RT RIC establece la conexión con dicha gNB y la acepta como nodo E2, identificándola como *gnbd_901_070_00019b_0*.

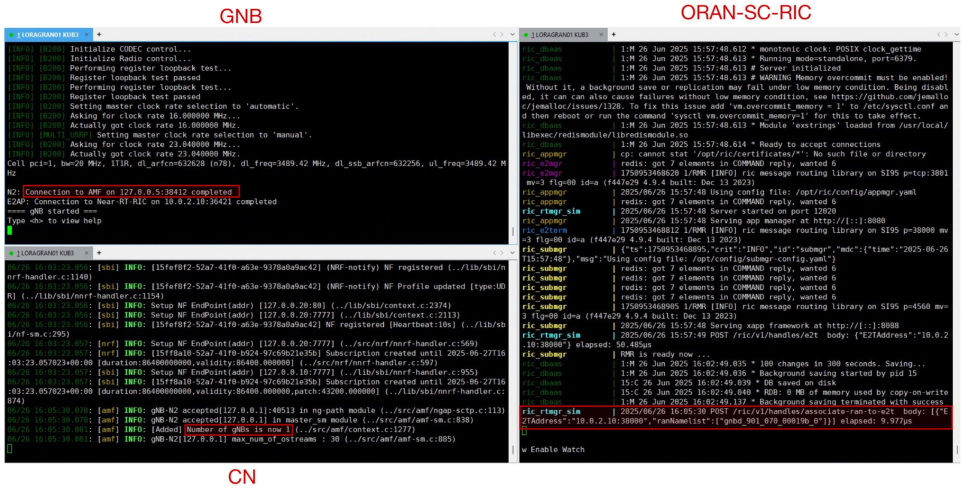


Figura 6.29: UE + RAN + CN + ORAN-SC-RIC.

Ahora es momento de hacer la prueba con la *xApp*. Al igual que se hizo con el escenario simulado, se lanza en primer lugar el *core*, seguido del Near-RT RIC. Con ambos elementos en ejecución, se pone en marcha la gNB y, posteriormente, el UE. Como se puede ver en la figura 6.30, este

último recibe una dirección IP dentro del rango 10.45.0.0/24. Una vez todo el sistema está operativo, se inicia una sesión de tráfico `iperf` con un caudal de 20 Mbps en el enlace descendente (DL) y, en paralelo, se ejecuta la *xApp simple_xapp.py*. Se recuerda que esta *xApp* monitoriza el volumen de SDUs y, al detectar que se han alcanzado los 4 MB, genera y envía un mensaje de control al nodo E2 para reducir el número de PRBs asignados.

Figura 6.30: Ejecución de *simple_xapp.py* en el escenario real.

Como se muestra en la figura 6.30, la *xApp* está monitorizando correctamente la métrica correspondiente al volumen de datos transmitidos. Además, en la traza de `iperf` se observa una tasa de transmisión de 20 Mbps, lo que valida el comportamiento esperado del sistema. Es importante destacar que, a diferencia de los entornos simulados donde los resultados de `iperf` no eran representativos a lo enviado debido a la discrepancia entre el tiempo simulado y el tiempo real, en este entorno físico los valores obtenidos sí reflejan con precisión el tráfico real transmitido.

Más adelante, al analizar las trazas obtenidas durante la ejecución (véase figura 6.31), se puede observar con claridad cómo el Near-RT RIC, a través de la *xApp* implementada, envía de manera efectiva mensajes de control hacia la gNB. Estos mensajes contienen instrucciones específicas para la reasignación dinámica de los recursos radioeléctricos, como los PRBs, en función de las métricas monitorizadas previamente. Por su parte, la gNB responde de forma inmediata confirmando la recepción y ejecución de dichas políticas, lo que demuestra no solo la correcta comunicación entre ambos elementos, sino también la capacidad de la red para adaptarse rápidamente a las condiciones cambiantes del entorno.

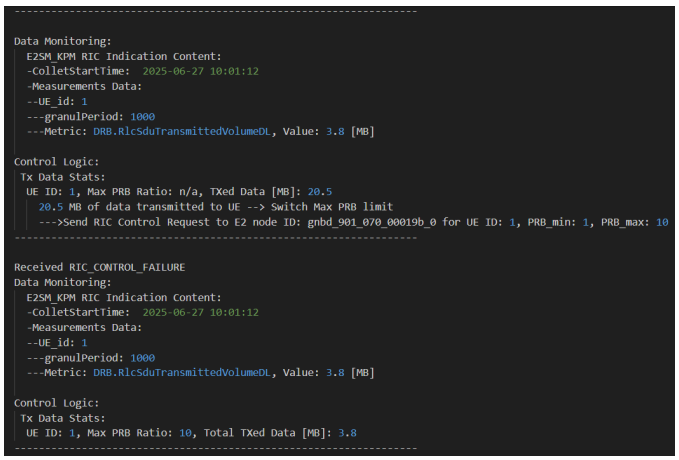


Figura 6.31: Intercambio de mensajes de control entre la *xApp* y la gNB.

Como se ha podido observar, el Near-RT RIC no solo aporta beneficios en escenarios simulados, sino que también demuestra su funcionalidad y eficacia en entornos físicos reales. En este caso, se ha logrado verificar tanto la correcta monitorización en tiempo casi real de métricas clave del sistema, como la capacidad de aplicar control dinámico sobre los recursos radioeléctricos, específicamente sobre los PRBs, a través de la *xApp simple_xapp.py*.

Esta validación práctica confirma que el ORAN-SC-RIC es capaz de gestionar y optimizar la red, lo que permite una mayor flexibilidad y adaptabilidad frente a las condiciones cambiantes del entorno radioeléctrico. Además, su implementación en un entorno no virtualizado, con hardware real y canal físico, aporta un nivel de realismo y fiabilidad que supera los resultados obtenidos en simulaciones.

6.5. Estudio del procedimiento E2 en Wireshark

Tras haber estudiado cómo las diferentes *xApps* son capaces de suscribirse a métricas mediante la comunicación entre el Near-RT RIC y la gNB, resulta interesante corroborar que dicho procedimiento se refleja en el intercambio de mensajes del protocolo encargado de ello, el protocolo E2. Este comportamiento fue descrito previamente en la sección 3.3.3 del capítulo 3 y en este caso se monitoriza el funcionamiento de la *xApp simple_xapp* de la sección 6.3.3.

En la figura 6.32 se observa que, en primer lugar, se establece la sesión E2. A continuación, una *xApp* envía una solicitud de suscripción. Posteriormente, la gNB comienza a enviar reportes al Near-RT RIC. Si la *xApp*

es de monitorización, simplemente recibe los datos sin tomar más acciones. En cambio, si se trata de una *xApp* de control, esta procesa los reportes y responde enviando acciones de control a la gNB, estableciendo así un ciclo continuo de interacción. Finalmente, la *xApp* puede decidir cancelar su suscripción cuando ya no desee seguir recibiendo información.

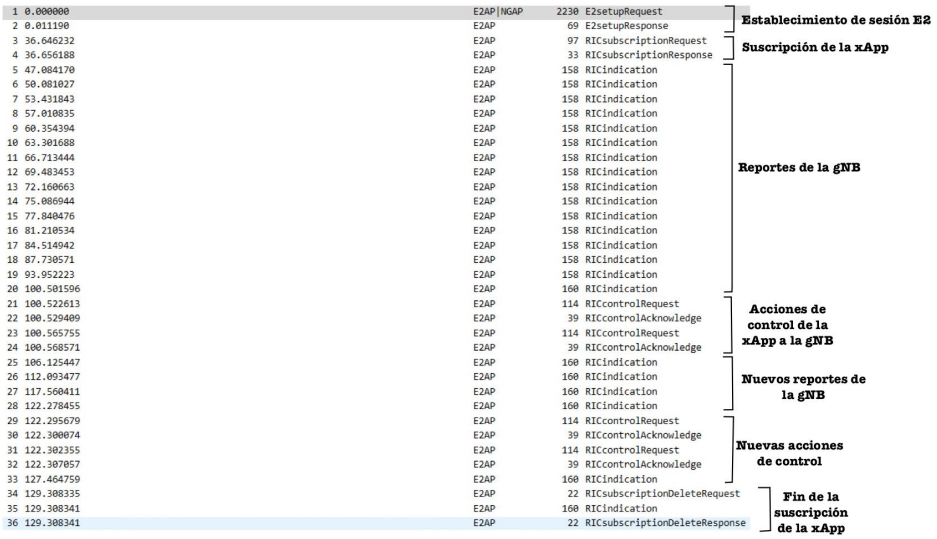


Figura 6.32: Procedimiento E2 en Wireshark.

Estos mensajes se pueden mirar de forma resumida con más detalle, por ejemplo, el caso del intercambio *E2 Setup Request / E2 Setup Response*. El primero de los mensajes es enviado desde la gNB al Near-RT RIC e incluye 4 partes diferenciadas. La primera de ellas, mostrada en la figura 6.33, incluye el identificador de transacción y el identificador de la gNB, en este caso.

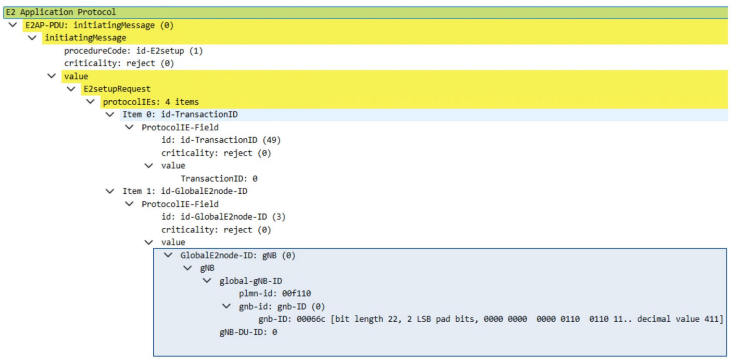


Figura 6.33: *E2 Setup Request* (I).

En la segunda parte (véase la figura 6.34), se observa cómo aparecen las funciones RAN permitidas, esto es, los modelos de servicio. Como se puede ver, se permite el modelo E2SM-KPM v2 y E2SM-RC v1. Dentro del primero se listan las métricas de reporte soportadas. Por otro lado, se permite el control de PRBs.

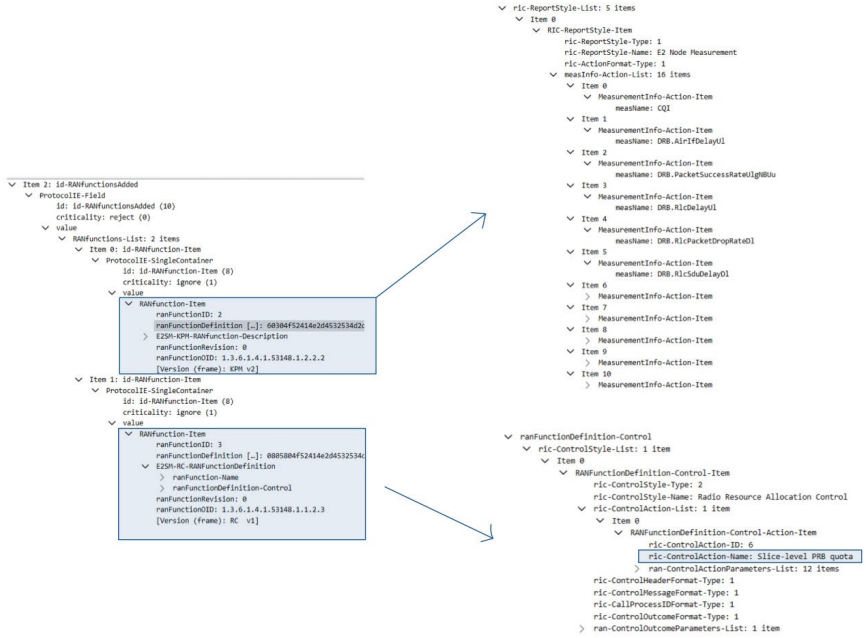


Figura 6.34: *E2 Setup Request* (II).

Finalmente, también la gNB también envía otra información de configuración, como el nombre de la AMF a la que está conectada. Esta parte puede verse en la figura 6.35.

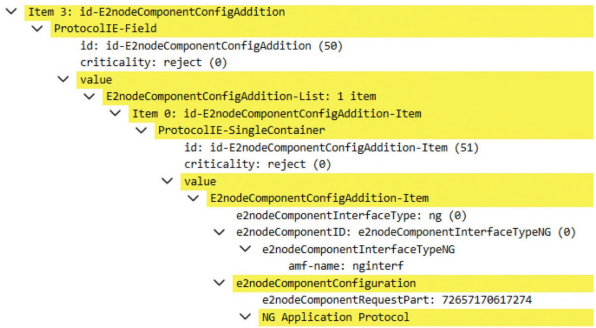


Figura 6.35: *E2 Setup Request* (III).

A este mensaje, el Near-RT RIC responde con un mensaje de confirmación, el *E2 Setup Response*.

Una vez establecida la comunicación E2 entre ambos componentes, la *xApp* puede querer iniciar una suscripción a alguna/s métrica/s. Para ello, envía un mensaje *E2 Ssuscription Request*, el cual es confirmado por la gNB con un *E2 Suscription Response*. En la figura 6.36 se puede ver esta petición.

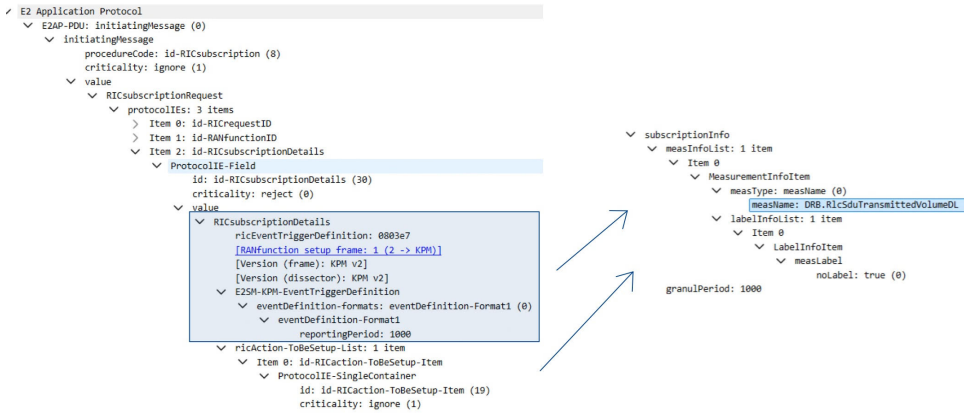


Figura 6.36: *E2 Suscription Request*.

Como se aprecia, la *xApp* indica que se quiere suscribir a la métrica *DRB.RlcSduTransmittedVolumeDL* y recibirla por medio de un estilo de reporte.

Una vez que la solicitud de suscripción es aceptada, la gNB comienza a enviar mensajes de tipo *RIC Indication* al Near-RT RIC. En el ejemplo analizado, había dos UEs conectados, por lo que, como se observa en la figura 6.37, dentro del campo *RICIndicationMessage* se incluye la métrica *DRB.RlcSduTransmittedVolumeDL* correspondiente a cada UE. Además, cada usuario está identificado mediante dos identificadores únicos.

- gNB-DU-UE-F1AP-ID: identificador único asignado por la DU de la gNB al UE en el protocolo F1AP.
- gNB-CU-UE-F1AP-ID: identificador único asignado por la CU de la gNB al UE también en F1AP.

Estos identificadores permiten que el sistema distinga entre los diferentes UEs conectados.

Figura 6.37: *RIC Indication.*

Si la *xApp* no tuviese configurado el control, simplemente recibiría esta métrica con el fin de usarla para monitorización de la red. En este caso, permite control, por lo que al recibir estos valores, decide enviar una acción de control, utilizando para ello un mensaje *RIC Control Request*. Este mensaje puede ver en la figura 6.38. En él, dentro del campo *RICControlMessage*, la *xApp* informa de los valores que manda para el control de PRBs, en este caso, 1 %, 10 % y 100 %. Como se observa en la figura 6.32, este mensaje se envía 2 veces, una para cada UE.

La lógica de control de la gNB sabrá si es la primera vez que el UE ha sobrepasado el umbral de la métrica propuesta, y en dicho caso responderá con un *E2 Control Acknowledge* con la acción de control que implementa (véase la figura 6.39).

Después, se mantiene este bucle de control hasta que la *xApp* puede decidir terminar esta suscripción, mediante un intercambio *E2 Suscription Delete Request / E2 Suscription Delete Response.*

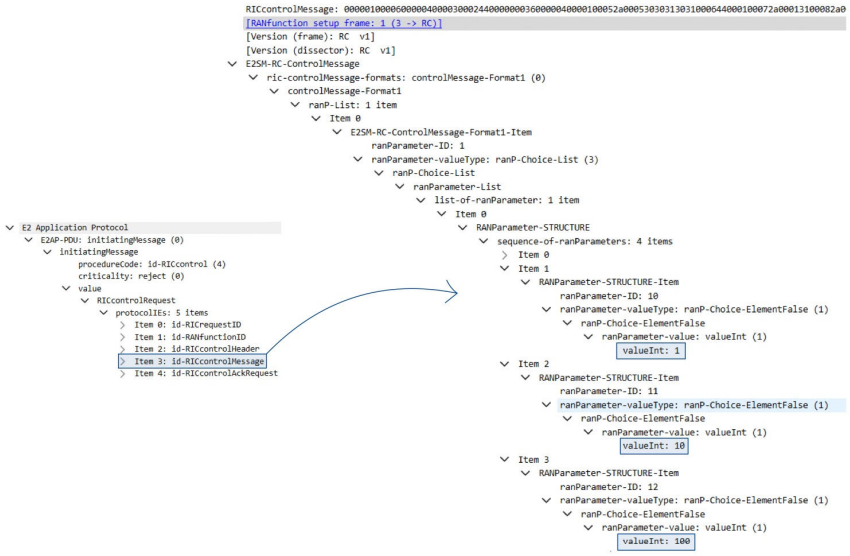


Figura 6.38: *RIC Control Request.*

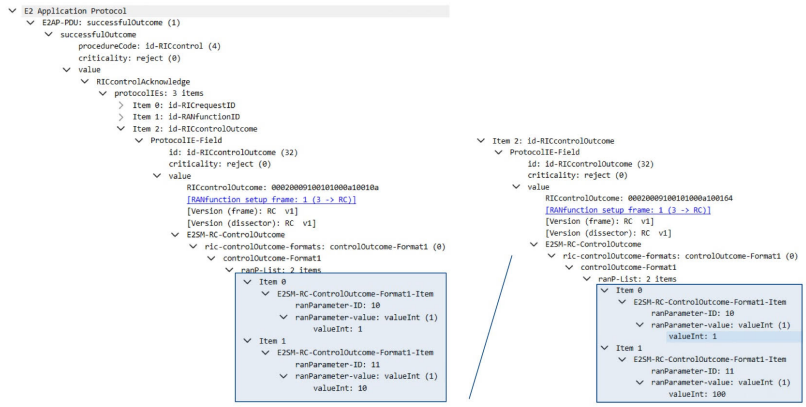


Figura 6.39: *RIC Control Acknowledge.*

6.6. Complicaciones y desafíos encontrados

Durante el desarrollo del presente proyecto, centrado en el análisis e implementación de funcionalidades 5G utilizando los entornos OAIC y srsRAN, surgieron complicaciones derivadas del estado de desarrollo en el que se encontraban estos marcos de trabajo. Desde enero hasta junio de 2025, tanto OAIC como srsRAN experimentaron actualizaciones prácticamente diarias en su código y documentación, al igual que otras herramientas empleadas

como el Near-RT RIC FlexRIC. Esta situación obligó a una adaptación constante de los entornos, incluyendo ajustes en versiones y módulos para garantizar la compatibilidad con los escenarios de prueba planteados. En particular, en el entorno srsRAN, utilizado como base principal de experimentación, fue necesario realizar un análisis y retroceso de versiones. Esto se debió a que ciertas funcionalidades clave, como la gestión de PRBs mediante el Near-RT RIC, aún no estaban completamente implementadas o presentaban inestabilidad, lo que llevó a validar manualmente funcionalidades específicas en varias versiones del software, con la consiguiente inversión de tiempo y esfuerzo.

Otro desafío fue el desarrollo y prueba de una nueva *xApp* orientada al control de recursos físicos en función de métricas obtenidas del entorno de red también supuso un nuevo reto. Esta aplicación requería un diseño nuevo y personalizado de la lógica de control presentada en otras *xApps* existentes debido al uso de nuevas métricas medidas a la hora de hacer el control de recursos.

Finalmente, la inclusión de un caso de estudio con dispositivos SDR también trajo consigo desafíos relacionados con el cambio de un entorno simulado a uno físico, como el ajuste de parámetros asociados al canal, la conectividad entre los componentes de red situados ahora en diferentes máquinas, etc. Esta última etapa supuso una capa más de complejidad técnica, pero también amplió el alcance y la aplicabilidad del trabajo realizado.

En resumen, más allá de las propias dificultades del análisis de la red y el desarrollo de nuevas funcionalidades como la *xApp* en srsRAN, el principal reto del proyecto ha estado en las constantes investigaciones y actualizaciones de los entornos utilizados, en especial srsRAN. Esto obligó a dedicar una cantidad considerable de tiempo a investigar la compatibilidad entre versiones, evaluar implementaciones parciales y adaptar configuraciones para permitir tanto el uso de funcionalidades ya existentes como la integración de otras desarrolladas específicamente para este proyecto.

Capítulo 7

Conclusiones y Líneas Futuras

A continuación, se exponen las conclusiones obtenidas tras la realización del proyecto, junto con los posibles trabajos futuros a realizar.

7.1. Conclusiones

Este proyecto se ha desarrollado en paralelo a la evolución constante de las herramientas *open-source* relacionados con O-RAN y el Near-RT RIC, lo que ha supuesto afrontar diversas dificultades técnicas y de integración a lo largo de su ejecución.

En una primera fase se revisaron y analizaron diferentes proyectos y desarrollos internacionales relacionados con la arquitectura abierta O-RAN y la implementación del Near-RT RIC, con el fin de comprender el estado del arte y las posibilidades que ofrece este nuevo paradigma para la gestión dinámica y programable de redes 5G.

Seguidamente, se profundizó en los conceptos fundamentales previos a la implementación práctica, abordando con detalle la evolución de las redes 5G desde la arquitectura tradicional de RAN hacia la arquitectura abierta O-RAN, y los cambios estructurales que esta transición implica. Se explicó la incorporación del RIC en sus dos variantes principales, el Near-RT RIC y el Non-RT RIC, haciendo especial énfasis en las aplicaciones asociadas, conocidas como *xApps* y *rApps*, respectivamente. Finalmente, se describió el protocolo E2, que es el encargado de gestionar el control y la monitorización de la red a través de estos elementos.

Con esta base teórica, se procedió a implementar dos escenarios simulados, haciendo uso de las plataformas srsRAN y OAIC. Este proceso im-

plicó la configuración detallada de los distintos componentes en ambos casos, adaptando cada entorno para su correcta operación. Tras analizar ambos escenarios una vez configurados, se vio que la plataforma OAIC permitía únicamente la integración con uno de los tipos de Near-RT RIC, el FlexRIC, limitando así las funcionalidades disponibles para el control y monitorización. Por su parte, srsRAN demostró una mayor versatilidad al permitir tanto la integración con FlexRIC como con ORAN-SC-RIC, además de una mayor comunidad de soporte o software más actualizado, lo que llevó a seleccionar éste como escenario principal para continuar con pruebas más avanzadas de monitorización y control dinámico.

Tras esto, se realizaron varias pruebas de funcionalidad con OAIC, las cuales constataron el funcionamiento del Near-RT RIC y de la red en sí, y más tarde se pasó a probar el comportamiento de srsRAN. En este último, se pudo comprobar cómo para las dos implementaciones del glsNear-RT RIC, se disponía de *xApps* capaces de monitorizar ciertas métricas como el *throughput*, los PRBs disponible, el volumen de SDUs transmitidas, etc. Además, se pudo llevar a cabo una *xApp* de control que permitía gestionar de forma eficiente los recursos radioeléctricos en escenarios donde se detectase congestión.

Finalmente, y como un avance no previsto en la planificación inicial del proyecto, se desarrolló un escenario real en el que se empleó un canal físico real mediante un dispositivo SDR. Este escenario permitió validar el funcionamiento del sistema en condiciones próximas a un entorno físico real, superando las limitaciones de los entornos puramente virtuales y demostrando la capacidad del Near-RT RIC para operar eficazmente en escenarios reales con tráfico y condiciones dinámicas.

Con ello, se concluye que los objetivos planteados al inicio del proyecto han sido alcanzados con éxito, sentando la base para futuros desarrollos e investigaciones en el ámbito de las arquitecturas abiertas y el control dinámico en redes móviles 5G/6G.

7.2. Líneas Futuras

A pesar de que los dos escenarios virtuales y el escenario real están funcionando correctamente, existen diversas mejoras y avances que pueden implementarse para optimizar y ampliar las capacidades del sistema:

- Dado que tanto OAIC como srsRAN se encuentran en continuo desarrollo, se prevé la incorporación futura de funcionalidades de *slicing* tanto en el FlexRIC como en la red radio y núcleo de OAIC, así como

en la parte radio de srsRAN. Esto permitirá una gestión más eficiente y segmentada de los recursos de red.

- Actualmente, las implementaciones se rigen por versiones de los estándares E2 que no están completamente actualizadas con las últimas especificaciones publicadas por la *O-RAN Alliance*. La actualización a versiones más recientes traerá nuevas funcionalidades y mejoras en el rendimiento y la interoperabilidad.

- Aunque srsRAN implementa métricas definidas en especificaciones concretas de *E2SM-KPM* y *E2SM-RC*, existen aún métricas y funcionalidades contempladas en estas especificaciones que no se han integrado en el proyecto. Se está trabajando en su incorporación para ampliar la capacidad de monitorización y control.

- Se pretende seguir desarrollando nuevas *xApps* que permitan realizar funciones más avanzadas y específicas para la gestión dinámica de la red, aumentando así el abanico de posibilidades para la optimización y adaptación en tiempo real.

- En el escenario real, se podrían introducir condiciones más complejas, como la simulación de obstáculos y variaciones en el canal radioeléctrico, con el objetivo de analizar cómo afectan estas casuísticas al comportamiento de la red y al desempeño del control dinámico.

En definitiva, estas líneas futuras abren un camino prometedor para enriquecer y robustecer la arquitectura abierta O-RAN con sistemas cada vez más completos, flexibles y adaptables, fortaleciendo el papel del Near-RT RIC como elemento clave en este tipo de redes.

Bibliografía

[1] Ndikumana, A., Nguyen, K. K., & Cheriet, M. (2024). *Age of Processing-Aware Offloading Decision for Autonomous Vehicles in 5G Open RAN Environment*. IEEE Transactions on Mobile Computing, 2024. doi: 10.1109/TMC.2023.3341082. Consultado el 29 de junio de 2025, desde https://ieeexplore.ieee.org/document/10352974

[2] Tselikis, C. J. (2023). *Automated and Secure Control of Industrial Internet of Things: from WSN to Open RAN Solutions*. Consultado el 29 de junio de 2025, desde https://ieeexplore.ieee.org/document/10183051

[3] de Oliveira, W., Batista, J., J. O. R., Novais, T., Takashima, S. T., Stange, L. R., Martucci, J., M., Cugnasca, C. E., & Bressan, G. (2023). *OpenCare5G: O-RAN in Private Network for Digital Health Applications*. Sensors, 2023. doi: 10.3390/s23021047. Consultado el 29 de junio de 2025, desde https://doi.org/10.3390/s23021047

[4] Witkowski, D. (2019, diciembre). *Bridging the Gap. 21st Century Wireless Telecommunications Handbook: Second Edition*. California, USA, Joint Venture Silicon Valley.

[5] Thieu, H.-T., Pham, V.-Q., Kak, A., & Choi, N. (2023). *Demystifying the Near-real Time RIC: Architecture, Operations, and Benchmarking Insights*. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2023. doi: 10.1109/INFOCOMWKSHPS57453.2023.10225852. Consultado el 29 de junio de 2025, desde https://ieeexplore.ieee.org/document/10225852

[6] O-RAN Work Group 3. (2025a). *Near-Real-time RAN Intelligent Controller and E2 Interface. Near-RT RIC Architecture* (O-RAN.WG3.TS.RICARCH-R004-v07.00). O-RAN ALLIANCE. Consultado el 29 de junio de 2025, desde https://specifications.o-ran.org/specifications

[7] ALLIANCE, O.-R. (2024). *Non-RT RIC: Architecture* (O-RAN.WG2.Non-RT-RIC-ARCH-R004-v06.00). O-RAN ALLIANCE. Consultado el 29 de junio de 2025, desde https://specifications.o-ran.org/specifications

[8] srsRAN. (2024a). *5G srsRAN*. Consultado el 29 de junio de 2025, desde https://www.srsran.com/5g

[9] OpenAirInterface. (2025a). *OpenAirInterface Homepage*. Consultado el 29 de junio de 2025, desde https://openairinterface.org/

[10] Başaran, O. T., Başaran, M., Turan, D., Bayrak, H. G., & Sandal, Y. S. (2023). *Deep Autoencoder Design for RF Anomaly Detection in 5G O-RAN Near-RT RIC via xApps*. IEEE BlackSeaCom. doi: 10.1109/ICC-Workshops57953.2023.10283501. Consultado el 29 de junio de 2025, desde https://ieeexplore.ieee.org/document/10283501

[11] IS-Wireless. (2023a). *Poland's First Open RAN 5G Network for Industry 4.0 Built Using Local Solutions*. Consultado el 29 de junio de 2025, desde https://www.is-wireless.com/news/polands-first-open-ran-5g-network-for-industry-4-0-built-using-local-solutions/

[12] IS-Wireless. (2023b). *IS-Wireless has been selected to provide private 5G for Industry 4.0 in Germany*. Consultado el 29 de junio de 2025, desde https://www.is-wireless.com/news/is-wireless-has-been-selected-to-provide-private-5g-for-industry-4-0-in-germany/

[13] IS-Wireless. (2024). *Smart Networks for a Smarter World: IS-Wireless Revolutionizes Telecom with RIC Innovations*. Consultado el 29 de junio de 2025, desde https://www.is-wireless.com/news/smart-networks-for-a-smarter-world-is-wireless-revolutionizes-telecom-with-ric-innovations/

[14] Virgin Media O2, Mavenir, VMWare & University of Surrey. (2024). *5G MoDE: Mobile oRAN for Dense Environments*. Consultado el 29 de junio de 2025, desde https://uktin.net/5GMoDE

[15] Zhang, T., Boateng, J. O., Islam, T. U., Ahmad, A., Zhang, H., & Qiao, D. (2024). *ARA-O-RAN: End-to-End Programmable O-RAN Living Lab for Agriculture and Rural Communities*. IEEE INFOCOM 2024 . doi: 10.1109/INFOCOMWKSHPS61880.2024.10620737. Consultado el 29 de junio de 2025, desde https://ieeexplore.ieee.org/abstract/document/10620737

[16] Santana, M., & Dias, K. L. (2024, Diciembre). *Open RAN-Enabled Deep Learning-Assisted Mobility Management for Connected Vehicles*. Consultado el 29 de junio de 2025, desde https://arxiv.org/pdf/2412.21161

[17] 3rd Generation Partnership Project (3GPP). (s.f.). *3GPP – The 3rd Generation Partnership Project*. Consultado el 29 de junio de 2025, desde https://www.3gpp.org/

[18] Telecom Infra Project. (2025). *Telecom Infra Project*. Consultado el 29 de junio de 2025, desde https://telecominfraproject.com/

[19] *Telecom Infra Project GitHub Repositories*. (s.f.). GitHub. Consultado el 29 de junio de 2025, desde https://github.com/Telecominfraproject

[20] Aether. (2025). *Aether Project – Cloud Native 5G*. Consultado el 29 de junio de 2025, desde https://aetherproject.org/

[21] Open Networking Foundation. (2025, abril). *Aether: The First Open Source 5G Connected Edge Platform*. Consultado el 29 de junio de 2025, desde https://opennetworking.org/aether/

[22] *Open Networking Lab GitHub Repositories* [Set of repositories with tools and frameworks developed by ON.Lab]. (2025, abril). GitHub. Consultado el 29 de junio de 2025, desde https://github.com/opennetworkinglab

[23] *Aether Documentation*. (2025). Aether Project. Consultado el 29 de junio de 2025, desde https://docs.aetherproject.org/master/index.html

[24] OpenRAN Gym. (2024a). *5G ns-3 with O-RAN near-RT RIC bronze version*. Consultado el 29 de junio de 2025, desde https://openrangym.com/tutorials/ns-o-ran

[25] OpenRAN Gym. (2024b). *ns-O-RAN: Open-Source 5G Simulation Platform*. Consultado el 29 de junio de 2025, desde https://openrangym.com/ran-frameworks/ns-o-ran

[26] *OAI 5G Core - AMF Module*. (2025, abril). GitLab. Consultado el 29 de junio de 2025, desde https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-amf/-/wikis/home

[27] *OpenAirInterface 5G RF Simulator*. (2025, abril). GitLab. Consultado el 29 de junio de 2025, desde https://gitlab.eurecom.fr/oai/openairinterface5g/-/blob/develop/radio/rfsimulator/README.md

[28] OpenAirInterface. (2025b). *OpenAirInterface 5G RAN Project*. Consultado el 29 de junio de 2025, desde https://openairinterface.org/oai-5g-ran-project/

[29] *FlexRIC by MOSAIC5G*. (2025, abril). GitLab. Consultado el 29 de junio de 2025, desde https://gitlab.eurecom.fr/mosaic5g/flexric

[30] *OAIC by OpenAI Cellular*. (2025, abril). GitHub. Consultado el 29 de junio de 2025, desde https://github.com/openaicellular/openairinterface5G

[31] OpenAI Cellular. (2025). *OAIC Quickstart Guide*. Consultado el 29 de junio de 2025, desde https://openaicellular.github.io/oaic/quickstart.html

[32] Software Radio Systems. (s.f.). *Software Radio Systems (SRS)*. Consultado el 29 de junio de 2025, desde https://srs.io/

[33] Open5GS. (2024). *Open5GS*. Consultado el 29 de junio de 2025, desde https://open5gs.org/

[34] ZeroMQ Project. (2025). *ZeroMQ Official Website*. Consultado el 29 de junio de 2025, desde https://zeromq.org/

[35] *O-RAN SC RIC by srsRAN*. (2025, abril). GitHub. Consultado el 29 de junio de 2025, desde https://github.com/srsran/oran-sc-ric

[36] Håkegård, J. E., Lundkvist, H., Rauniyar, A., & Morris, P. (2024). *Performance Evaluation of an Open Source Implementation of a 5G Standalone Platform*. IEEE Access. doi: 10.1109/ACCESS.2024.3367120. Consultado el 29 de junio de 2025, desde https://ieeexplore.ieee.org/document/10439170

[37] Unión Internacional de Telecomunicaciones. (s.f.). *UIT – Página oficial de la Unión Internacional de Telecomunicaciones*. Consultado el 29 de junio de 2025, desde https://www.itu.int/es/Pages/default.aspx#/es

[38] Navarro-Ortiz, J., Romero-Diaz, P., Sendra, S., Ameigeiras, P., Ramos-Muñoz, J. J., & López-Soler, J. M. (2020). *A Survey on 5G Usage Scenarios and Traffic Models*. IEEE Communications Surveys & Tutorials. doi: 10.1109/COMST.2020.2971781. Consultado el 29 de junio de 2025, desde https://doi.org/10.1109/COMST.2020.2971781

[39] Unión Internacional de Telecomunicaciones. (2017). *Informe UIT-R M.2410-2017 – Requisitos mínimos de rendimiento técnico para IMT-2020*. Consultado el 29 de junio de 2025, desde https://www.itu.int/pub/R-REP-M.2410-2017

[40] 3GPP. (s.f.). *5G System Overview*. Consultado el 29 de junio de 2025, desde https://www.3gpp.org/technologies/5g-system-overview

[41] ETSI. (s.f.). *3GPP TR 21.915 V15.0.0: Technical Specification Group Services and System Aspects; Release 15 Description*. Consultado el 29 de junio de 2025, desde https://www.etsi.org/deliver/etsi_tr/121900_121999/121915/15.00.00_60/tr_121915v150000p.pdf

[42] Flores, M. E., Poisson, D. D., Stevens, C. J., Nieves, A. V., & Wyglinski, A. M. (2023). *Implementation and Evaluation of a Smart*

Uplink Jamming Attack in a Public 5G Network. IEEE Access. doi: 10.1109/ACCESS.2023.3296701. Consultado el 29 de junio de 2025, desde https://doi.org/10.1109/ACCESS.2023.3296701

[43] MathWorks. (s.f.). *O-RAN and 5G: Enabling the Future of Wireless Networks*. Consultado el 29 de junio de 2025, desde https://www.mathworks.com/discovery/o-ran.html

[44] O-RAN Work Group 1. (2023). *O-RAN Architecture Description* (O-RAN.WG1.TS.OAD-R004-v13.00). O-RAN ALLIANCE. Consultado el 29 de junio de 2025, desde https://specifications.o-ran.org/specifications

[45] Li, C., Akman, A., Ong, L., Suciu, L., Sahin, B. Y., Li, T., Stjernholm, P., Voigt, J., Buldorini, A., Sun, Q., Li, M., Hirayama, H., Hannak, G., Costanzo, S., Sintorn, M., & Yang, J. (2020). *O-RAN Use Cases and Deployment Scenarios* (O-RAN.WG1.Use-Cases-and-Deployment-Scenarios-White-Paper-2020-02). O-RAN ALLIANCE. Consultado el 29 de junio de 2025, desde https://specifications.o-ran.org/specifications

[46] Polese, M., Bonati, L., D'Oro, S., Basagni, S., & Melodia, T. (2023). *Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges*. IEEE Communications Surveys & Tutorials. doi: 10.1109/COMST.2023.3240176. Consultado el 29 de junio de 2025, desde https://ieeexplore.ieee.org/document/10024837

[47] O-RAN Work Group 3. (2025b). *Near-RT RIC and E2 Interface.E2 Service Model (E2SM)* (O-RAN.WG3.TS.E2SM-R004-v07.00). O-RAN ALLIANCE. Consultado el 29 de junio de 2025, desde https://specifications.o-ran.org/specifications

[48] Microsoft Corporation. (s.f.). *Especificaciones y requisitos del sistema de Windows 11.* Consultado el 29 de junio de 2025, desde https://www.microsoft.com/es-es/windows/windows-11-specifications

[49] Oracle Corporation. (s.f.). *Oracle VM VirtualBox.* Consultado el 29 de junio de 2025, desde https://www.virtualbox.org/

[50] Canonical Ldt. (s.f.[a]). *Ubuntu 20.04 LTS*. Consultado el 29 de junio de 2025, desde https://ubuntu.com/blog/tag/20-04-lts

[51] Canonical Ldt. (s.f.[b]). *Ubuntu 22.04 LTS*. Consultado el 29 de junio de 2025, desde https://ubuntu.com/blog/tag/22-04-lts

[52] Grafana Labs. (s.f.). *Grafana: The open observability platform*. Consultado el 29 de junio de 2025, desde https://grafana.com/

[53] Wireshark Foundation. (s.f.). *Wireshark: The world's most popular network protocol analyzer*. Consultado el 29 de junio de 2025, desde https://www.wireshark.org/

[54] Overleaf. (s.f.). *Overleaf — Editor LaTeX online*. Consultado el 29 de junio de 2025, desde https://es.overleaf.com/project

[55] Thomas, A., & Barashev, D. (s.f.). *GanttProject: free project management tool for Windows, macOS, and Linux*. Consultado el 29 de junio de 2025, desde https://www.ganttproject.biz

[56] Zotero. (s.f.). *Your personal research assistant*. Consultado el 29 de junio de 2025, desde https://www.zotero.org/

[57] Alliance, O. C. S. (2024). *OAIC 2024 Workshop - OAI FlexRIC Documentation*. Consultado el 29 de junio de 2025, desde https://openaicellular.github.io/oaic/OAIC-2024-Workshop-oai-flexric-documentation.html

[58] *OpenAirInterface 5G Configuration Files*. (2025, mayo). GitHub. Consultado el 29 de junio de 2025, desde https://github.com/openaicellular/openairinterface5G/tree/develop/targets/PROJECTS/GENERIC-NR-5GC/CONF

[59] Mosaic5G. (2021). *FlexRIC - Branch: aymanfatich*. GitLab. Consultado el 29 de junio de 2025, desde https://gitlab.eurecom.fr/mosaic5g/flexric/-/tree/aymanfatich

[60] O-RAN Alliance. (s.f.). *O-RAN Specifications – Technical Documents and Standards*. Consultado el 29 de junio de 2025, desde https://specifications.o-ran.org/specifications

[61] 3rd Generation Partnership Project (3GPP). (2024). *5G; Management and orchestration; 5G performance measurements* (inf. téc. N.º TS 28.552 V16.6.0). 3GPP. Consultado el 29 de junio de 2025, desde https://www.etsi.org/deliver/etsi_ts/128500_128599/128552/16.06.00_60/ts_128552v160600p.pdf

[62] srsRAN. (2024b). *O-RAN NearRT-RIC and xApp*. Consultado el 29 de junio de 2025, desde https://docs.srsran.com/projects/project/en/latest/tutorials/source/near-rt-ric/source/index.html

[63] *Open source O-RAN 5G CU/DU solution from Software Radio Systems (SRS)*. (2025, 7 de mayo). GitHub. Consultado el 29 de junio de 2025, desde https://github.com/srsran/srsran_project

[64] *Open source SDR 4G software suite from Software Radio Systems (SRS)*. (2023, 23 de noviembre). GitHub. Consultado el 29 de junio de 2025, desde https://github.com/srsran/srsran_4g

[65] srsRAN. (2024c). *Configuration Reference*. Consultado el 29 de junio de 2025, desde https://docs.srsran.com/projects/project/en/latest/user_manuals/source/config_ref.html

[66] *Example Configuration File for srsUE in srsRAN*. (2023). GitHub. Consultado el 29 de junio de 2025, desde https://github.com/srsran/srsRAN_4G/blob/master/srsue/ue.conf.example

[67] Iperf. (s.f.). *Iperf Documentation*. Consultado el 29 de junio de 2025, desde https://iperf.fr/iperf-doc.php

[68] SemiEngineering. (s.f.). *5G New Radio Signal Design*. Consultado el 29 de junio de 2025, desde https://semiengineering.com/5g-new-radio-signal-design/

[69] Boada, A. O. (2025). *TFM Near-RT RIC*. GitHub. Consultado el 29 de junio de 2025, desde https://github.com/AlejandraOliver/TFM_Near-RT-RIC/tree/main

[70] GNU Radio Project. (s.f.). *GNU Radio – The Free & Open Software Radio Ecosystem*. Consultado el 29 de junio de 2025, desde https://www.gnuradio.org/

Anexo A

Fichero de configuración de la gNB en OAIC

En este apéndice se muestra el *script* básico para la configuración de la gNB en el entorno OAIC (se puede ver un comentario añadido con la explicación de qué indica cada parámetro).

El fichero de configuración se muestra a continuación:

```
Active_gNBs = ( "gNB-OAI"); # gNBs activos que se van a iniciar
Asn1_verbosity = "none";    # Nivel de detalle en ASN.1

gNBs =
(
 {
    //////// Identification parameters:
    gNB_ID    =  0xe00;                    # ID único del gNB
    gNB_name  =  "gNB-OAI";                # Nombre del nodo gNB
    tracking_area_code  =  1;              # Código de zona utilizado
        en la señalización con el AMF
    plmn_list = ({ mcc = 001; mnc = 01; mnc_length = 2; snssaiList =
        ({ sst = 1; }) }); # Configuración del PLMN (MCC/MNC)

    nr_cellid = 12345678L;                 # ID de celda de 36 bits

    //////// Physical parameters:

    do_CSIRS = 1;                # Activación de señales CSI-RS
    do_SRS = 1;                  # Activación de señales SRS
```

Figura A.1: Configuración del gNB y parámetros de red (I).

```
servingCellConfigCommon = (
{
  physCellId = 0;                       # Identificador físico de celda
  absoluteFrequencySSB = 641280;        # Frecuencia absoluta del SSB
  dl_frequencyBand = 78;                # Banda NR (aprox. 3.5 GHz)
  dl_absoluteFrequencyPointA = 640008;# Punto A para el portador DL
  dl_offstToCarrier = 0;                # Desviacion de A al portador DL
  dl_subcarrierSpacing = 1;           # Espaciado subportadora DL=30kHz
  dl_carrierBandwidth = 106;            # Ancho de banda DL (PRBs)

  initialDLBWPlocationAndBandwidth = 28875;   # Conf inicial DL BWP
  initialDLBWPsubcarrierSpacing = 1;          # Espaciado subp. DL BWP
  initialDLBWPcontrolResourceSetZero = 12;    # Recursos de control
      para PDCCH
  initialDLBWPsearchSpaceZero = 0;            # ID del espacio de bú
      squeda para PDCCH

  ul_frequencyBand = 78;                # Banda NR para UL
  ul_offstToCarrier = 0;                # Desviación UL
  ul_subcarrierSpacing = 1;             # Espaciado UL (30 kHz)
  ul_carrierBandwidth = 106;            # Ancho de banda UL (PRBs)
  pMax = 20;                            # Potencia máxima del UE (dBm)
  initialULBWPlocationAndBandwidth = 28875; # Configuración inicial UL
       BWP
initialULBWPsubcarrierSpacing = 1; # Espaciado subp. para el BWP en UL


  prach_ConfigurationIndex = 98;        # Índice de configuración PRACH
  prach_msg1_FDM = 0;                   # 0: un solo FDM
  prach_msg1_FrequencyStart = 0;        # Desviación de frecuencia PRACH
  zeroCorrelationZoneConfig = 13;       # Zona de correlación cero
  preambleReceivedTargetPower = -96;  # Potencia objetivo para preá
      mbulo
  preambleTransMax = 6;                 # Máximo de intentos de preá
      mbulo
  powerRampingStep = 1;                 # Paso de rampa de potencia
  ra_ResponseWindow = 4;                # Ventana de respuesta RACH
  ssb_perRACH_OccasionAndCB_PreamblesPerSSB_PR = 4; # Tipo de mapeo
      RACH/SSB
  ssb_perRACH_OccasionAndCB_PreamblesPerSSB = 14;   # Preámbulos por
      SSB
  ra_ContentionResolutionTimer = 7;   # Temporizador de resolución de
      colisiones
  rsrp_ThresholdSSB = 19;               # Umbral RSRP para selección SSB
  prach_RootSequenceIndex_PR = 2;       # Índice de la secuencia raíz
      primaria para PRACH
  prach_RootSequenceIndex = 1;          # Índice de la secuencia raíz
      secundaria para PRACH
  msg1_SubcarrierSpacing = 1,           # Espaciado para msg1 (30 kHz)
  restrictedSetConfig = 0,              # 0 = conjunto sin restricciones
  msg3_DeltaPreamble = 1;               # Delta para msg3
  p0_NominalWithGrant = -90;            # Valor nominal de potencia con
      grant

  pucchGroupHopping = 0;   # Saltos de grupo PUCCH deshabilitados
  hoppingId = 40;          # ID para hopping
  p0_nominal = -90;        # Potencia nominal

  ssb_PositionsInBurst_PR = 2; # Forma del bitmap de posiciones SSB
  ssb_PositionsInBurst_Bitmap = 1;  # Solo el primer SSB activo
  ssb_periodicityServingCell = 2;   # Periodicidad SSB (20ms)
  dmrs_TypeA_Position = 0;          # Posicion de DMRS tipo A
  subcarrierSpacing = 1;            # Espaciado para la celda
```

Figura A.2: Configuración del gNB y parámetros de red (II).

```
1
2   referenceSubcarrierSpacing = 1;  # Espaciado de subportadoras de
        referencia
3   dl_UL_TransmissionPeriodicity = 6;     # Periodicidad DL/UL (5ms)
4   nrofDownlinkSlots = 7;                 # Ranuras de bajada
5   nrofDownlinkSymbols = 6;               # Símbolos de bajada
6   nrofUplinkSlots = 2;                   # Ranuras de subida
7   nrofUplinkSymbols = 4;                 # Símbolos de subida
8
9   ssPBCH_BlockPower = -25;               # Potencia del bloque PBCH
10
11 }
12 );
13 SCTP :
14 {
15     SCTP_INSTREAMS  = 2; # Número de streams de entrada SCTP
16     SCTP_OUTSTREAMS = 2; # Número de streams de salida SCTP
17 };
18
19 amf_ip_address = ({ ipv4 = "192.168.70.132"; }); # IP del AMF
20
21 NETWORK_INTERFACES :
22 {
23     GNB_IPV4_ADDRESS_FOR_NG_AMF = "192.168.70.129/24"; # IP para
           interfaz NG con AMF
24     GNB_IPV4_ADDRESS_FOR_NGU = "192.168.70.129/24";    # IP para
           interfaz de usuario
25     GNB_PORT_FOR_S1U = 2152;                           # Puerto para
           trafico S1-U (GTP-U)
26 };
27
28 MACRLCs = (
29 {
30   num_cc = 1;                          # Componentes portadora
31   tr_s_preference = "local_L1";        # Transporte con capa L1 local
32   tr_n_preference = "local_RRC";       # Transporte con RRC local
33   pusch_TargetSNRx10 = 150;            # Objetivo SNR para PUSCH x10
34   pucch_TargetSNRx10 = 200;            # Objetivo SNR para PUCCH x10
35 });
36
37 L1s = (
38 {
39   num_cc = 1;
40   tr_n_preference = "local_mac";       # Comunicación local con MAC
41   prach_dtx_threshold = 120;           # Umbral de DTX para PRACH
42   pucch0_dtx_threshold = 100;          # Umbral de DTX para PUCCH0
43   ofdm_offset_divisor = 8;             # Control de sincronización
44 });
45
46 RUs = (
47 {
48   local_rf = "yes";                    # Indica si se usa RF local
49   nb_tx = 1;                           # Número de transmisores
50   nb_rx = 1;                           # Número de receptores
51   att_tx = 12;                         # Atenuación TX (dB)
52   att_rx = 12;                         # Atenuación RX (dB)
53   bands = [78];                        # Bandas soportadas
54   max_pdschReferenceSignalPower = -27; # Potencia máx de referencia
         PDSCH
55   max_rxgain = 114;                    # Ganancia máx de recepción
56   eNB_instances = [0];                 # Instancias eNB asociadas
57   bf_weights = [0x00007fff, 0x0000, 0x0000, 0x0000]; # Pesos de
         formación de haz
58   clock_src = "internal";              # Fuente de reloj
59 });
```

Figura A.3: Configuración del gNB y parámetros de red (III).

```
THREAD_STRUCT = (
{
  parallel_config = "PARALLEL_SINGLE_THREAD";  # Hilo único para procesamiento
  worker_config = "WORKER_ENABLE";             # Habilita worker threads
});

rfsimulator :
{
  serveraddr = "server";                    # IP del servidor RF simulado
  serverport = 4043;                        # Puerto del simulador
  options = ();                             # Opciones adicionales
  modelname = "AWGN";                       # Modelo de canal: AWGN
  IQfile = "/tmp/rfsimulator.iqs";          # Archivo de IQ para simulador
};

security = {
  ciphering_algorithms = ( "nea0" );         # Algoritmo de cifrado utilizado (nea0: sin cifrado real, util para pruebas).
  integrity_algorithms = ( "nia2", "nia0" );# Algoritmos de integridad : nia2 (128-EIA2 con AES), nia0 (sin integridad, solo pruebas).
  drb_ciphering = "yes";                    # Se aplica cifrado a los DRB (Data Radio Bearer).
  drb_integrity = "no";                     # No se aplica integridad a los DRB.
};

log_config :
{
  global_log_level = "info";    # Nivel global de logs: 'info' muestra informacion general sin ser demasiado detallada.
  hw_log_level = "info";        # Logs del hardware.
  phy_log_level = "info";       # Logs de la capa fisica.
  mac_log_level = "info";       # Logs de la capa MAC.
  rlc_log_level = "info";       # Logs de la capa RLC.
  pdcp_log_level = "info";      # Logs de la capa PDCP.
  rrc_log_level = "info";       # Logs de la capa RRC.
  ngap_log_level = "debug";     # Logs detallados para la capa NGAP (interfaz con el core).
  f1ap_log_level = "debug";     # Logs detallados para la interfaz F1AP (entre CU y DU).
};

e2_agent = {
  near_ric_ip_addr = "127.0.0.1";       # IP del Near-RT RIC con el que se comunica el agente E2.
  sm_dir = "/usr/local/lib/flexric/";   # Directorio donde se encuentran las shared libraries (Service Models) del agente E2.
};
```

Figura A.4: Configuración del gNB y parámetros de red (IV).

Anexo B

Opciones de configuración de la gNB en OAIC

En este apéndice se recopilan las opciones de configuración disponibles para la ejecución de la gNB en OAIC agrupadas por categorías:

B.1. Procesamiento, configuración UE/gNB, estimación de canal

- −−**rf-config-file**: archivo de configuración para el *front-end*.
- −−**thread-pool**: configuración del *pool* de hilos. Valor por defecto: 8 hilos flotantes.
- −−**phy-test**: prueba de capa PHY del UE (MAC deshabilitado).
- −−**do-ra**: prueba de procedimientos *Random Access* (RA) entre gNB y UE.
- −−**sa**: ejecuta el gNB en modo *standalone*.
- −−**sl-mode**: establece el modo NR *sidelink* (0: sin sidelink, 1: con cobertura/gNB, 2: sin cobertura/sin gNB).
- −−**usim-test**: usa algoritmo de autenticación *Exclusive OR* (XOR).
- −−**clock-source**: fuente de reloj del hardware (0: interno, 1: externo, 2: *Global Positioning System Disciplined Oscillator* (GPSDO)).
- −−**time-source**: fuente de tiempo del hardware (0: interno, 1: externo, 2: GPSDO).
- −−**tune-offset**: *offset* de sincronización *Local Oscillator* (LO) en Hz.

- −−**wait-for-sync**.
- `-C`: establece la frecuencia de *downlink* para todas las portadoras componentes.
- −−**CO**: establece el *offset* de *uplink* para todas las portadoras componentes.
- **-a**: *offset* de identificador de canal.
- **-d**: habilita estadísticas L1 y L2.
- `-q`: medición de tiempo de procesamiento por subtrama (modo LTE).
- −−**numerology**: numerología para 5G.
- −−**band**: índice de banda.
- −−**emulate-rf**: radiofrecuencia emulada habilitada (deshabilitada por defecto).
- −−**parallel-config**: nivel de paralelismo.
- −−`worker-config`: Configuración de *workers*.
- −−**noS1**: desactiva interfaz *S1 Interface* (S1).
- −−**rfsim**: ejecuta en modo simulador radio.
- −−**nbiot-disable**: desactiva *Narrowband IoT* (NB-IoT), aunque esté definido en el archivo de configuración.
- −−**chest-freq**: estimación de canal en frecuencia (0: interpolación lineal, 1: promedio por PRB).
- −−**chest-time**: estimación de canal en tiempo (0: usa último símbolo *Demodulation Reference Signal* (DMRS), 1: promedio por símbolo).
- −−**nsa**: activa el modo *non-standalone*.
- −−**node-number**: número de nodo (el nodo puede ser un UE).
- −−**usrp-tx-thread-config**: crea hilo adicional para transmisión USRP.
- −−**nfapi**: cambia el modo *Network Function API* (NFAPI) para NR.
- −−**non-stop**: vuelve al modo de sincronización tras 100 fallos PBCH consecutivos.
- −−**emulate-l1**: ejecuta en modo emulado L1 (capa PHY deshabilitada).

- −−**continuous-tx**: transmisión continua incluso en modo TDD.
- −−**disable-stats**: desactiva generación y persistencia global de estadísticas.
- −−**no-itti-threads**: no lanza hilos *Inter-Thread Interoperability* (ITTI), el procesamiento ocurre en el hilo llamador.
- −−**ldpc-offload-enable**: habilita descarga *Low-Density Parity-Check* (LDPC) a tarjeta *Advanced Micro Devices* (AMD) Xilinx T2 telco.
- −−**sync-ref**: el UE actúa como referencia de sincronización en *sidelink* (0: ninguno, 1: gNB, 2: GNSS, 4: local).
- **-A**: ajusta TA de la parte radio para compensar retrasos internos.
- **-E**: aplica tres cuartos de la frecuencia de muestreo (reduce velocidad de datos en *Universal Serial Bus* (USB)/*Peripheral Component Interconnect Express* (PCIe)).

B.2. Opciones de *logging* y servidores

- **-R**: habilita el log en línea.
- **-g**: establece el nivel global de log (0:error, 1:warn, 2:info, 3:debug, 4:trace).
- −−**telnetsrv**: inicia servidor Telnet embebido.
- −−**websrv**: inicia servidor Web embebido.
- −−**log-mem**: log en memoria.
- −−**telnetclt**: cliente Telnet.

B.3. Modo PHYTest

Es un modo especial de prueba utilizado en las implementaciones de redes móviles para evaluar, verificar y depurar el comportamiento de la capa física de un dispositivo o sistema.

- **-m**: MCS de *downlink* en modo PHYTest.
- **-l**: número de capas de *downlink* en PHYTest.
- **-L**: número de capas de *uplink* en PHYTest.
- **-t**: MCS de *uplink* en modo PHYTest.

- **-M**: número de PRBs para *Downlink Shared Channel* (DLSCH) en PHYTest.
- **-T**: número de PRBs para *Uplink Shared Channel* (ULSCH) en PHY-Test.
- **-D**: bitmap de slots DLSCH (slot 0 comienza en *Least Significant Bit* (LSB)).
- **-U**: bitmap de slots ULSCH (slot 0 comienza en LSB).
- −−**usrp-tx-thread-config**: crea hilo adicional para transmitir con USRP.
- −−**uecap_file**: ruta al archivo de capacidades del UE.

B.4. Parámetros Adicionales

- −−**Asn1_verbosity**.
- −−**Active_gNBs**.

Anexo C

Opciones de configuración del UE en OAIC

En este apéndice se recopilan las opciones de configuración disponibles para la ejecución del UE en OAIC agrupadas por categorías:

C.1. Procesamiento, configuración UE/gNB, ganancias, estimación de canal

- −−**usrp-args** : argumentos para identificar el USRP.
- −−**tx_subdev** : selección de subdispositivo de transmisión.
- −−**rx_subdev** : selección de subdispositivo de recepción.
- −−**dlsch-parallel** : número de hilos para procesamiento DLSCH (0 = sin paralelismo)
- −−**offset-divisor** : divisor para calcular el desfase del símbolo OFDM (por defecto: 8)
- −−**max-ldpc-iterations** : máximo de iteraciones del decodificador LDPC.
- −−**ldpc-offload-enable** : habilita *offload* de LDPC a tarjeta AMD Xilinx T2.
- −**V** : habilita *Value Change Dump* (VCD)
- −−**uecap_file** : ruta al archivo de capacidades del UE.
- −−**rrc_config_path** : ruta para configuración RRC.
- −−**reconfig-file** : archivo *reconfig.raw* en modo phy-test.

- −−**rbconfig-file** : archivo *rbconfig.raw* en modo phy-test.
- −−**ue-idx-standalone**.
- −−**ue-rxgain** : ganancia de recepción del UE.
- −−**ue-rxgain-off** : *offset* del amplificador externo del UE.
- −−**ue-txgain** : ganancia de transmisión del UE
- −−**ue-nb-ant-rx** : número de antenas receptoras del UE.
- −−**ue-nb-ant-tx** : número de antenas transmisoras del UE.
- −−**ue-scan-carrier** : escaneo completo de *Global Synchronization Channel* (GSCN) por el UE.
- −−**ue-fo-compensation** : habilita compensación de desfase de frecuencia en UE.
- −−**ue-max-power**: potencia máxima del usuario.
- −**r** : número de PRBs para modo el SA.
- −−**ssb** : subportadora de inicio.
- −−**if_freq** : frecuencia intermedia.
- −−**if_freq_off** : *offset* de frecuencia intermedia en UL.
- −−**chest-freq** : tipo de estimación de canal en frecuencia (0=interpolación, 1=media por PRB).
- −−**chest-time** : tipo de estimación de canal en tiempo (0=último símbolo DMRS, 1=media por símbolo).
- −−**ue-timing-correction-disable** : desactiva corrección temporal del UE.
- −−**SLC** : frecuencia de *sidelink* para todos los portadores.
- −−**num-ues**: número de usuarios.
- −−**ntn-koffset** : *offset* para *Non-Terrestrial Network* (NTN) (15 kHz SCS).
- −−**ntn-ta-common** : TA común para NTN en ms.
- −−**agc** : control automático de ganancia para UE.

C.2. Modo de ejecución, simulación, radiofrecuencia, interfaces, NSA

- −−**rf-config-file** : archivo de configuración de front-end.
- −−**thread-pool** : configuración del *pool* de hilos.
- −−**phy-test** : test del PHY del UE (MAC deshabilitado).
- −−**do-ra** : test con procedimientos RA (gNB/UE).
- −−**sa** : ejecuta el gNB en modo SA.
- −−**sl-mode** : modo *sidelink* NR (0=desactivado, 1=gNB, 2=sin gNB).
- −−**usim-test** : usa el algoritmo XOR para autenticación (modo test).
- −−**clock-source** : fuente de reloj (0=interno, 1=externo, 2=GPS-DO).
- −−**time-source** : fuente de tiempo (0=interno, 1=externo, 2=GPS-DO).
- −−**tune-offset** : *offset* de sintonización LO (en Hz).
- −−**wait-for-sync**.
- −**C** : frecuencia DL para todas las portadoras.
- −−**CO** : *offset* de UL para todos las portadoras.
- −**a** : *offset* de identificador de canal.
- −**d** : habilita *XForms* (XFORMS) y estadísticas L1/L2.
- −**q** : mide tiempos de procesamiento por subtrama.
- −−**numerology** : numerología para 5G.
- −−**band** : banda NR.
- −−**emulate-rf** : habilita la parte radio emulada (deshabilitada por defecto).
- −−**parallel-config** : configura nivel de paralelismo.
- −−**worker-config** : configura workers: `WORKER_DISABLE` / `WORKER_ENABLE`.
- −−**noS1** : desactiva la interfaz S1.
- −−**rfsim** : ejecuta en modo simulador radio.

- −−**nbiot-disable** : desactiva NB-IoT aunque esté en la configuración.
- −−**nsa** : habilita el modo *non-standalone*.
- −−**node-number**.
- −−**usrp-tx-thread-config** : crea hilo extra para transmisión USRP.
- −−**nfapi** : modo NFAPI para NR.
- −−**non-stop** : vuelve al modo SYNC tras 100 fallos PBCH.
- −−**emulate-l1** : ejecuta en modo L1 emulado (PHY desactivado).
- −−**continuous-tx** : transmisión continua, incluso en TDD (para evitar bugs USRP).
- −−**disable-stats** : desactiva generación y almacenamiento de estadísticas.
- −−**no-itti-threads** : ejecuta procesamiento de cola ITTI en hilo llamador.
- −−**sync-ref** : referencia de sincronización para UE (0=ninguna, 1=gNB, 2=*Global Navigation Satellite System* (GNSS), 4=tiempo local).
- −**A** : ajuste de avance temporal de la parte radio.
- −**E** : aplica 3/4 del ancho de banda (reduce el uso USB/*Peripheral Component Interconnect Express* (PCIE)).

C.3. *Logging* y servidores embebidos

- −**R** : habilita log online.
- −**g** : nivel global de log (0=error, 4=trace).
- −−**telnetsrv** : inicia servidor Telnet embebido.
- −−**websrv** : inicia servidor Web embebido.
- −−**log-mem**: logs en memoria.
- −−**telnetclt**: cliente Telnet.

C.4. Trazado y debug visual

- −−**T_port** : puerto para el trazador.
- −−**T_nowait** : no espera por el trazador, inicia de inmediato.
- −−**T_stdout** : muestra mensajes de *logging* por consola.

Anexo D

Fichero de configuración de la gNB en srsRAN

En este apéndice se muestra el *script* empleado para la configuración de la gNB en el entorno srsRAN (se puede ver un comentario añadido con la explicación de qué indica cada parámetro).

El fichero de configuración se muestra a continuación:

```
# Dirección del AMF al que se conecta la gNB.
cu_cp:
  amf:
    # Dirección o nombre del host del AMF.
    addr: 10.53.1.2
    # Puerto del AMF.
    port: 38412
    # Dirección local en la que se enlaza la gNB para el tráfico del
        AMF.
    bind_addr: 10.53.1.1
    # Áreas de seguimiento compatibles.
    supported_tracking_areas:
      - tac: 7
        # Lista de PLMNs compatibles dentro del área de seguimiento.
        plmn_list:
          - plmn: "00101"
            # Lista de slices soportados para el área de seguimiento.
            tai_slice_support_list:
              - sst: 1

  # Temporizador de inactividad para UE/PDU/DRB en segundos (1-7200).
  inactivity_timer: 7200
```

Figura D.1: Fichero de configuración de la gNB en srsRAN (I).

```
# Configuración de la RU (Radio Unit) utilizando ZMQ.
ru_sdr:
  # Nombre del driver de radiofrecuencia.
  device_driver: zmq
  # Argumentos del dispositivo, incluyendo puertos TX/RX y frecuencia de muestreo base.
  device_args: tx_port=tcp://127.0.0.1:2000,rx_port=tcp://127.0.0.1:2001,base_srate=11.52e6
  # Frecuencia de muestreo en MHz.
  srate: 11.52
  # Ganancia de transmisión.
  tx_gain: 75
  # Ganancia de recepción.
  rx_gain: 75

# Configuración de la celda NR.
cell_cfg:
  # Número absoluto de canal de RF para el enlace descendente.
  dl_arfcn: 368500
  # Banda NR utilizada.
  band: 3
  # Ancho de banda del canal en MHz.
  channel_bandwidth_MHz: 10
  # Espaciado entre subportadoras (en kHz).
  common_scs: 15
  # Identificador de red pública móvil transmitido por la gNB.
  plmn: "00101"
  # Código del área de seguimiento.
  tac: 7

  # Configuración del canal de control PDCCH.
  pdcch:
    common:
      # Índice de espacio de búsqueda cero (compatible con srsUE).
      ss0_index: 0
      # Índice de CORESET cero (compatible con srsUE). Significa que se usa la zona estándar donde van las órdenes de la red (DCI -Downlink Control Info), y es la que necesita srsUE para poder entenderlas.
      coreset0_index: 6
    dedicated:
      # Tipo de espacio de búsqueda: común.
      ss2_type: common
      # Formato de DCI desactivado (uso de fallback).
      dci_format_0_1_and_1_1: false

  # Configuración de acceso aleatorio PRACH.
  prach:
    # Índice de configuración PRACH (compatible con srsUE).
    prach_config_index: 1

  # Modulación y codificación del canal de datos PDSCH.
  pdsch:
    mcs_table: qam64

  # Modulación y codificación del canal de subida PUSCH.
  pusch:
    mcs_table: qam64

# Configuración del sistema de logging.
log:
  # Ruta del archivo de log.
  filename: /tmp/gnb.log
  # Nivel de log general.
  all_level: info
  # Tamaño máximo en bytes para logs en hexadecimal.
  hex_max_size: 0
```

Figura D.2: Fichero de confguración de la gNB en srsRAN (II).

```
# Configuración de capturas PCAP.
pcap:
  # Activar capturas de nivel MAC.
  mac_enable: false
  # Archivo de salida para capturas MAC.
  mac_filename: /tmp/gnb_mac.pcap
  # Activar capturas NGAP.
  ngap_enable: false
  # Archivo de salida para capturas NGAP.
  ngap_filename: /tmp/gnb_ngap.pcap
  # Activar capturas E2AP.
  e2ap_enable: true
  # Archivo para capturas E2AP en el DU.
  e2ap_du_filename: /tmp/gnb_du_e2ap.pcap
  # Archivo para capturas E2AP en el CU-CP.
  e2ap_cu_cp_filename: /tmp/gnb_cu_cp_e2ap.pcap
  # Archivo para capturas E2AP en el CU-UP.
  e2ap_cu_up_filename: /tmp/gnb_cu_up_e2ap.pcap

# Configuración de la interfaz E2.
e2:
  # Habilitar agente E2 en el DU.
  enable_du_e2: true
  # Habilitar módulo KPM (Key Performance Metrics).
  e2sm_kpm_enabled: true
  # Habilitar módulo RC (Radio Control).
  e2sm_rc_enabled: true
  # Dirección IP del RIC.
  addr: 127.0.0.1
  # Dirección local de enlace (solo necesaria si el RIC está en otra m
      áquina).
  #bind_addr: 127.0.0.100
  # Puerto del RIC.
  port: 36421

# Configuración de métricas internas.
metrics:
  # Periodicidad del informe RLC en milisegundos.
  rlc_report_period: 1000
  # Habilita la generación de métricas en formato JSON.
  enable_json_metrics: true
  # Dirección IP del servidor que recibirá las métricas.
  addr: 172.19.1.4
  # Puerto del servidor de métricas.
  port: 55555
```

Figura D.3: Fichero de confguración de la gNB en srsRAN (III).

Anexo E

Fichero de configuración del UE en srsRAN

En este apéndice se muestra el *script* empleado para la configuración del UE en el entorno srsRAN (se puede ver un comentario añadido con la explicación de qué indica cada parámetro).

El fichero de configuración se muestra a continuación:

```
# Configuración de RF: Ajustes de la frecuencia, ganancias y tasa de
    muestreo para el dispositivo de radio.
[rf]
# Desplazamiento de frecuencia (en Hz).
freq_offset = 0
# Ganancia de transmisión (en dB).
tx_gain = 50
# Ganancia de recepción (en dB).
rx_gain = 50
# Tasa de muestreo (en muestras por segundo).
srate = 11.52e6
# Número de antenas.
nof_antennas = 1

# Nombre del dispositivo RF (aquí usando ZMQ como ejemplo).
device_name = zmq
# Argumentos para el dispositivo, como puertos y tasa de muestreo.
device_args = tx_port=tcp://127.0.0.1:2001,rx_port=tcp
    ://127.0.0.1:2000,base_srate=11.52e6

# Configuración para EUTRA (LTE) - Parámetros de la red EUTRA.
[rat.eutra]
# Número de portadora de bajada (EUTRA ARFCN).
dl_earfcn = 2850
# Número de portadoras LTE.
nof_carriers = 0
```

Figura E.1: Fichero de configuración del UE en srsRAN (I).

```
# Configuración para NR (5G) - Parámetros de la red NR.
[rat.nr]
# Banda NR (5G).
bands = 3
# Número de portadoras NR.
nof_carriers = 1

# Configuración de PCAP: Habilitación de la captura de paquetes para diferentes interfaces.
[pcap]
# Desactiva la captura de PCAP.
enable = none
# Archivo de captura de MAC.
mac_filename = /tmp/ue_mac.pcap
# Archivo de captura de MAC para NR.
mac_nr_filename = /tmp/ue_mac_nr.pcap
# Archivo de captura para NAS.
nas_filename = /tmp/ue_nas.pcap

# Configuración de registros (logs): Detalles del archivo de logs y nivel de información.
[log]
# Nivel de registro para todo el sistema.
all_level = info
# Nivel de registro para la capa física.
phy_lib_level = none
# Límite de tamaño de registros en formato hexadecimal.
all_hex_limit = 32
# Ruta del archivo de logs.
filename = /tmp/ue.log
# Tamaño máximo del archivo de log (sin límite).
file_max_size = -1

# Configuración del USIM: Datos de la tarjeta SIM, como el algoritmo de autenticación.
[usim]
# Modo de operación (simulación).
mode = soft
# Algoritmo de autenticación.
algo = milenage
# Valor OPC para autenticación.
opc  = 63BFA50EE6523365FF14C1F45F88737D
# Clave secreta K para autenticación.
k    = 00112233445566778899aabbccddeeff
# IMSI del usuario.
imsi = 001010123456780
# IMEI del dispositivo.
imei = 353490069873319

# Configuración de RRC: Parámetros relacionados con el control de radio.
[rrc]
# Versión de liberación de RRC.
release = 15
# Categoría del dispositivo UE.
ue_category = 4

# Configuración de NAS: Parámetros para la red de acceso no radio.
[nas]
# Nombre del APN.
apn = srsapn
# Protocolo del APN (IPv4).
apn_protocol = ipv4
```

Figura E.2: Fichero de configuración del UE en srsRAN (II).

```
# Configuración de la pasarela de red: Ajustes para la red de usuario.
[gw]
# Espacio de nombres de red (namespace).
netns = ue1
# Nombre del dispositivo de red.
ip_devname = tun_srsue
# Máscara de subred para el dispositivo.
ip_netmask = 255.255.255.0

# Configuración de la interfaz gráfica (GUI): Habilitación de la GUI.
[gui]
# Desactiva la interfaz gráfica.
enable = false
```

Figura E.3: Fichero de configuración del UE en srsRAN (III).

Anexo F

xApps para monitorización de métricas

A continuación, se muestra modo informativo, las *xApps* empleadas para los escenarios de monitorización empleados en este trabajo.

F.1. FlexRIC en OAIC

En las figuras siguientes se muestra la *xapp_kpm_moni.c*:

```
#include "../../../../src/xApp/e42_xapp_api.h"
#include "../../../../src/util/alg_ds/alg/defer.h"
#include "../../../../src/util/time_now_us.h"
#include "../../../../src/util/alg_ds/ds/lock_guard/lock_guard.h"
#include "../../../../src/util/e.h"

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>

static
uint64_t const period_ms = 1000;

static
pthread_mutex_t mtx;
```

Figura F.1: *xapp_kpm_moni.c* (I).

```
static
void log_gnb_ue_id(ue_id_e2sm_t ue_id)
{
  if (ue_id.gnb.gnb_cu_ue_f1ap_lst != NULL) {
    for (size_t i = 0; i < ue_id.gnb.gnb_cu_ue_f1ap_lst_len; i++) {
      printf("UE ID type = gNB-CU, gnb_cu_ue_f1ap = %u\n", ue_id.gnb.
          gnb_cu_ue_f1ap_lst[i]);
    }
  } else {
    printf("UE ID type = gNB, amf_ue_ngap_id = %lu\n", ue_id.gnb.
        amf_ue_ngap_id);
  }
  if (ue_id.gnb.ran_ue_id != NULL) {
    printf("ran_ue_id = %lx\n", *ue_id.gnb.ran_ue_id); // RAN UE NGAP
        ID
  }
}

static
void log_du_ue_id(ue_id_e2sm_t ue_id)
{
  printf("UE ID type = gNB-DU, gnb_cu_ue_f1ap = %u\n", ue_id.gnb_du.
      gnb_cu_ue_f1ap);
  if (ue_id.gnb_du.ran_ue_id != NULL) {
    printf("ran_ue_id = %lx\n", *ue_id.gnb_du.ran_ue_id); // RAN UE
        NGAP ID
  }
}

static
void log_du_ue_id(ue_id_e2sm_t ue_id)
{
  printf("UE ID type = gNB-DU, gnb_cu_ue_f1ap = %u\n", ue_id.gnb_du.
      gnb_cu_ue_f1ap);
  if (ue_id.gnb_du.ran_ue_id != NULL) {
    printf("ran_ue_id = %lx\n", *ue_id.gnb_du.ran_ue_id); // RAN UE
        NGAP ID
  }
}
static
void log_cuup_ue_id(ue_id_e2sm_t ue_id)
{
  printf("UE ID type = gNB-CU-UP, gnb_cu_cp_ue_e1ap = %u\n", ue_id.
      gnb_cu_up.gnb_cu_cp_ue_e1ap);
  if (ue_id.gnb_cu_up.ran_ue_id != NULL) {
    printf("ran_ue_id = %lx\n", *ue_id.gnb_cu_up.ran_ue_id); // RAN UE
         NGAP ID
  }
}

typedef void (*log_ue_id)(ue_id_e2sm_t ue_id);

static
log_ue_id log_ue_id_e2sm[END_UE_ID_E2SM] = {
    log_gnb_ue_id, // common for gNB-mono, CU and CU-CP
    log_du_ue_id,
    log_cuup_ue_id,
    NULL,
    NULL,
    NULL,
    NULL,
};
```

Figura F.2: *xapp_kpm_moni.c* (II).

```
static
void log_int_value(byte_array_t name, meas_record_lst_t meas_record)
{
  if (cmp_str_ba("RRU.PrbTotDl", name) == 0) {
    printf("RRU.PrbTotDl = %d [PRBs]\n", meas_record.int_val);
  } else if (cmp_str_ba("RRU.PrbTotUl", name) == 0) {
    printf("RRU.PrbTotUl = %d [PRBs]\n", meas_record.int_val);
  } else if (cmp_str_ba("DRB.PdcpSduVolumeDL", name) == 0) {
    printf("DRB.PdcpSduVolumeDL = %d [kb]\n", meas_record.int_val);
  } else if (cmp_str_ba("DRB.PdcpSduVolumeUL", name) == 0) {
    printf("DRB.PdcpSduVolumeUL = %d [kb]\n", meas_record.int_val);
  } else {
    printf("Measurement Name not yet supported\n");
  }
}

static
void log_real_value(byte_array_t name, meas_record_lst_t meas_record)
{
  if (cmp_str_ba("DRB.RlcSduDelayDl", name) == 0) {
    printf("DRB.RlcSduDelayDl = %.2f [us]\n", meas_record.real_val);
  } else if (cmp_str_ba("DRB.UEThpDl", name) == 0) {
    printf("DRB.UEThpDl = %.2f [kbps]\n", meas_record.real_val);
  } else if (cmp_str_ba("DRB.UEThpUl", name) == 0) {
    printf("DRB.UEThpUl = %.2f [kbps]\n", meas_record.real_val);
  } else {
    printf("Measurement Name not yet supported\n");
  }
}

typedef void (*log_meas_value)(byte_array_t name, meas_record_lst_t
    meas_record);

static
log_meas_value get_meas_value[END_MEAS_VALUE] = {
    log_int_value,
    log_real_value,
    NULL,
};

static
void match_meas_name_type(meas_type_t meas_type, meas_record_lst_t
    meas_record)
{
  // Get the value of the Measurement
  get_meas_value[meas_record.value](meas_type.name, meas_record);
}

static
void match_id_meas_type(meas_type_t meas_type, meas_record_lst_t
    meas_record)
{
  (void)meas_type;
  (void)meas_record;
  assert(false && "ID Measurement Type not yet supported");
}
```

Figura F.3: *xapp_kpm_moni.c* (III).

```
typedef void (*check_meas_type)(meas_type_t meas_type,
    meas_record_lst_t meas_record);

static
check_meas_type match_meas_type[END_MEAS_TYPE] = {
    match_meas_name_type,
    match_id_meas_type,
};

static
void log_kpm_measurements(kpm_ind_msg_format_1_t const* msg_frm_1)
{
  assert(msg_frm_1->meas_info_lst_len > 0 && "Cannot correctly print
      measurements");

  // UE Measurements per granularity period
  for (size_t j = 0; j < msg_frm_1->meas_data_lst_len; j++) {
    meas_data_lst_t const data_item = msg_frm_1->meas_data_lst[j];

    for (size_t z = 0; z < data_item.meas_record_len; z++) {
      meas_type_t const meas_type = msg_frm_1->meas_info_lst[z].
          meas_type;
      meas_record_lst_t const record_item = data_item.meas_record_lst[
          z];

      match_meas_type[meas_type.type](meas_type, record_item);

      if (data_item.incomplete_flag && *data_item.incomplete_flag ==
          TRUE_ENUM_VALUE)
        printf("Measurement Record not reliable");
    }
  }

}

static
void sm_cb_kpm(sm_ag_if_rd_t const* rd)
{
  assert(rd != NULL);
  assert(rd->type == INDICATION_MSG_AGENT_IF_ANS_V0);
  assert(rd->ind.type == KPM_STATS_V3_0);

  // Reading Indication Message Format 3
  kpm_ind_data_t const* ind = &rd->ind.kpm.ind;
  kpm_ric_ind_hdr_format_1_t const* hdr_frm_1 = &ind->hdr.
      kpm_ric_ind_hdr_format_1;
  kpm_ind_msg_format_3_t const* msg_frm_3 = &ind->msg.frm_3;

  int64_t const now = time_now_us();
  static int counter = 1;
  {
    lock_guard(&mtx);

    printf("\n%7d KPM ind_msg latency = %ld [us]\n", counter, now -
        hdr_frm_1->collectStartTime); // xApp <-> E2 Node
```

Figura F.4: *xapp_kpm_moni.c* (IV).

```
    // Reported list of measurements per UE
    for (size_t i = 0; i < msg_frm_3->ue_meas_report_lst_len; i++) {
      // log UE ID
      ue_id_e2sm_t const ue_id_e2sm = msg_frm_3->meas_report_per_ue[i
          ].ue_meas_report_lst;
      ue_id_e2sm_e const type = ue_id_e2sm.type;
      log_ue_id_e2sm[type](ue_id_e2sm);

      // log measurements
      log_kpm_measurements(&msg_frm_3->meas_report_per_ue[i].
          ind_msg_format_1);

    }
    counter++;
  }
}

static
test_info_lst_t filter_predicate(test_cond_type_e type, test_cond_e
    cond, int value)
{
  test_info_lst_t dst = {0};

  dst.test_cond_type = type;
  // It can only be TRUE_TEST_COND_TYPE so it does not matter the type
  // but ugly ugly...
  dst.S_NSSAI = TRUE_TEST_COND_TYPE;

  dst.test_cond = calloc(1, sizeof(test_cond_e));
  assert(dst.test_cond != NULL && "Memory exhausted");
  *dst.test_cond = cond;

  dst.test_cond_value = calloc(1, sizeof(test_cond_value_t));
  assert(dst.test_cond_value != NULL && "Memory exhausted");
  dst.test_cond_value->type = OCTET_STRING_TEST_COND_VALUE;

  dst.test_cond_value->octet_string_value = calloc(1, sizeof(
      byte_array_t));
  assert(dst.test_cond_value->octet_string_value != NULL && "Memory
      exhausted");
  const size_t len_nssai = 1;
  dst.test_cond_value->octet_string_value->len = len_nssai;
  dst.test_cond_value->octet_string_value->buf = calloc(len_nssai,
      sizeof(uint8_t));
  assert(dst.test_cond_value->octet_string_value->buf != NULL && "
      Memory exhausted");
  dst.test_cond_value->octet_string_value->buf[0] = value;

  return dst;
}

static
label_info_lst_t fill_kpm_label(void)
{
  label_info_lst_t label_item = {0};

  label_item.noLabel = ecalloc(1, sizeof(enum_value_e));
  *label_item.noLabel = TRUE_ENUM_VALUE;

  return label_item;
}
```

Figura F.5: *xapp_kpm_moni.c* (V).

```
static
kpm_act_def_format_1_t fill_act_def_frm_1(ric_report_style_item_t const* report_item)
{
  assert(report_item != NULL);

  kpm_act_def_format_1_t ad_frm_1 = {0};

  size_t const sz = report_item->meas_info_for_action_lst_len;

  // [1, 65535]
  ad_frm_1.meas_info_lst_len = sz;
  ad_frm_1.meas_info_lst = calloc(sz, sizeof(meas_info_format_1_lst_t));
  assert(ad_frm_1.meas_info_lst != NULL && "Memory exhausted");

  for (size_t i = 0; i < sz; i++) {
    meas_info_format_1_lst_t* meas_item = &ad_frm_1.meas_info_lst[i];
    // 8.3.9
    // Measurement Name
    meas_item->meas_type.type = NAME_MEAS_TYPE;
    meas_item->meas_type.name = copy_byte_array(report_item->meas_info_for_action_lst[i].name);

    // [1, 2147483647]
    // 8.3.11
    meas_item->label_info_lst_len = 1;
    meas_item->label_info_lst = ecalloc(1, sizeof(label_info_lst_t));
    meas_item->label_info_lst[0] = fill_kpm_label();
  }

  // 8.3.8 [0, 4294967295]
  ad_frm_1.gran_period_ms = period_ms;

  // 8.3.20 - OPTIONAL
  ad_frm_1.cell_global_id = NULL;

#if defined KPM_V2_03 || defined KPM_V3_00
  // [0, 65535]
  ad_frm_1.meas_bin_range_info_lst_len = 0;
  ad_frm_1.meas_bin_info_lst = NULL;
#endif

  return ad_frm_1;
}

static
kpm_act_def_t fill_report_style_4(ric_report_style_item_t const* report_item)
{
  assert(report_item != NULL);
  assert(report_item->act_def_format_type == FORMAT_4_ACTION_DEFINITION);

  kpm_act_def_t act_def = {.type = FORMAT_4_ACTION_DEFINITION};

  // Fill matching condition
  // [1, 32768]
  act_def.frm_4.matching_cond_lst_len = 1;
  act_def.frm_4.matching_cond_lst = calloc(act_def.frm_4.matching_cond_lst_len, sizeof(matching_condition_format_4_lst_t));
  assert(act_def.frm_4.matching_cond_lst != NULL && "Memory exhausted");
```

Figura F.6: *xapp_kpm_moni.c* (VI).

```
  // Filter connected UEs by S-NSSAI criteria
  test_cond_type_e const type = S_NSSAI_TEST_COND_TYPE; //
      CQI_TEST_COND_TYPE
  test_cond_e const condition = EQUAL_TEST_COND; //
      GREATERTHAN_TEST_COND
  int const value = 1;
  act_def.frm_4.matching_cond_lst[0].test_info_lst = filter_predicate(
      type, condition, value);

  // Fill Action Definition Format 1
  // 8.2.1.2.1
  act_def.frm_4.action_def_format_1 = fill_act_def_frm_1(report_item);

  return act_def;
}

typedef kpm_act_def_t (*fill_kpm_act_def)(ric_report_style_item_t
    const* report_item);

static
fill_kpm_act_def get_kpm_act_def[END_RIC_SERVICE_REPORT] = {
    NULL,
    NULL,
    NULL,
    fill_report_style_4,
    NULL,
};

static
kpm_sub_data_t gen_kpm_subs(kpm_ran_function_def_t const* ran_func)
{
  assert(ran_func != NULL);
  assert(ran_func->ric_event_trigger_style_list != NULL);

  kpm_sub_data_t kpm_sub = {0};

  // Generate Event Trigger
  assert(ran_func->ric_event_trigger_style_list[0].format_type ==
      FORMAT_1_RIC_EVENT_TRIGGER);
  kpm_sub.ev_trg_def.type = FORMAT_1_RIC_EVENT_TRIGGER;
  kpm_sub.ev_trg_def.kpm_ric_event_trigger_format_1.report_period_ms =
       period_ms;

  // Generate Action Definition
  kpm_sub.sz_ad = 1;
  kpm_sub.ad = calloc(kpm_sub.sz_ad, sizeof(kpm_act_def_t));
  assert(kpm_sub.ad != NULL && "Memory exhausted");

  // Multiple Action Definitions in one SUBSCRIPTION message is not
      supported in this project
  // Multiple REPORT Styles = Multiple Action Definition = Multiple
      SUBSCRIPTION messages
  ric_report_style_item_t* const report_item = &ran_func->
      ric_report_style_list[0];
  ric_service_report_e const report_style_type = report_item->
      report_style_type;
  *kpm_sub.ad = get_kpm_act_def[report_style_type](report_item);

  return kpm_sub;
}

static
bool eq_sm(sm_ran_function_t const* elem, int const id)
{
  if (elem->id == id)
    return true;

  return false;
}
```

Figura F.7: *xapp_kpm_moni.c* (VII).

```
static
size_t find_sm_idx(sm_ran_function_t* rf, size_t sz, bool (*f)(
    sm_ran_function_t const*, int const), int const id)
{
  for (size_t i = 0; i < sz; i++) {
    if (f(&rf[i], id))
      return i;
  }
  assert(0 != 0 && "SM ID could not be found in the RAN Function List"
      );
}
int main(int argc, char* argv[])
{
  fr_args_t args = init_fr_args(argc, argv);

  // Init the xApp
  init_xapp_api(&args);
  sleep(1);

  e2_node_arr_xapp_t nodes = e2_nodes_xapp_api();
  defer({ free_e2_node_arr_xapp(&nodes); });

  assert(nodes.len > 0);

  printf("Connected E2 nodes = %d\n", nodes.len);

  pthread_mutexattr_t attr = {0};
  int rc = pthread_mutex_init(&mtx, &attr);
  assert(rc == 0);

  sm_ans_xapp_t* hndl = calloc(nodes.len, sizeof(sm_ans_xapp_t));
  assert(hndl != NULL);

  // START KPM
  int const KPM_ran_function = 2;

  for (size_t i = 0; i < nodes.len; ++i) {
    e2_node_connected_xapp_t* n = &nodes.n[i];

    size_t const idx = find_sm_idx(n->rf, n->len_rf, eq_sm,
        KPM_ran_function);
    assert(n->rf[idx].defn.type == KPM_RAN_FUNC_DEF_E && "KPM is not
        the received RAN Function");
    // if REPORT Service is supported by E2 node, send SUBSCRIPTION
    // e.g. OAI CU-CP
    if (n->rf[idx].defn.kpm.ric_report_style_list != NULL) {
      // Generate KPM SUBSCRIPTION message
      kpm_sub_data_t kpm_sub = gen_kpm_subs(&n->rf[idx].defn.kpm);

      hndl[i] = report_sm_xapp_api(&n->id, KPM_ran_function, &kpm_sub,
           sm_cb_kpm);
      assert(hndl[i].success == true);

      free_kpm_sub_data(&kpm_sub);
    }
  }
  // END KPM
  sleep(1800);
  for (int i = 0; i < nodes.len; ++i) {
    // Remove the handle previously returned
    if (hndl[i].success == true)
      rm_report_sm_xapp_api(hndl[i].u.handle);
  }
  free(hndl);
  // Stop the xApp
  while (try_stop_xapp_api() == false)
    usleep(1000);
  printf("Test xApp run SUCCESSFULLY\n");}
```

Figura F.8: *xapp_kpm_moni.c* (VIII).

F.2. ORAN-SC-RIC en srsRAN

Seguidamente, en las figuras se puede observar la *kpm_mon_xapp.py*, empleada en este caso para el monitoreo de métricas en el escenario srsRAN por parte del ORAN-SC-RIC.

```
#!/usr/bin/env python3

import argparse
import signal
from lib.xAppBase import xAppBase

class MyXapp(xAppBase):

    def __init__(self, config, http_server_port, rmr_port):
        super(MyXapp, self).__init__(config, http_server_port,
            rmr_port)

    def my_subscription_callback(self, e2_agent_id, subscription_id,
        indication_hdr, indication_msg, kpm_report_style, ue_id):
        if kpm_report_style == 2:
            print("\nRIC Indication Received from {} for Subscription
                ID: {}, KPM Report Style: {}, UE ID: {}".format(
                e2_agent_id, subscription_id, kpm_report_style, ue_id)
                )
        else:
            print("\nRIC Indication Received from {} for Subscription
                ID: {}, KPM Report Style: {}".format(e2_agent_id,
                subscription_id, kpm_report_style))

        indication_hdr = self.e2sm_kpm.extract_hdr_info(indication_hdr
            )

        meas_data = self.e2sm_kpm.extract_meas_data(indication_msg)

        print("E2SM_KPM RIC Indication Content:")
        print("-ColletStartTime: ", indication_hdr['colletStartTime'])
        print("-Measurements Data:")

        granulPeriod = meas_data.get("granulPeriod", None)

        if granulPeriod is not None:
            print("-granulPeriod: {}".format(granulPeriod))

        if kpm_report_style in [1,2]:
            for metric_name, value in meas_data["measData"].items():
                print("--Metric: {}, Value: {}".format(metric_name,
                    value))
        else:
            for ue_id, ue_meas_data in meas_data["ueMeasData"].items()
                :

                print("--UE_id: {}".format(ue_id))
                granulPeriod = ue_meas_data.get("granulPeriod", None)
```

Figura F.9: *kpm_mon_xapp.py* (I).

```
                if granulPeriod is not None:
                    print("---granulPeriod: {}".format(granulPeriod))

                for metric_name, value in ue_meas_data["measData"].
                    items():
                    print("---Metric: {}, Value: {}".format(
                        metric_name, value))

    # Mark the function as xApp start function using xAppBase.
        start_function decorator.

    @xAppBase.start_function

    def start(self, e2_node_id, kpm_report_style, ue_ids, metric_names
        ):
        report_period = 1000
        granul_period = 1000

        # use always the same subscription callback, but bind
            kpm_report_style parameter

        subscription_callback = lambda agent, sub, hdr, msg: self.
            my_subscription_callback(agent, sub, hdr, msg,
            kpm_report_style, None)

        if (kpm_report_style == 1):
            print("Subscribe to E2 node ID: {}, RAN func: e2sm_kpm,
                Report Style: {}, metrics: {}".format(e2_node_id,
                kpm_report_style, metric_names))
            self.e2sm_kpm.subscribe_report_service_style_1(e2_node_id,
                 report_period, metric_names, granul_period,
                subscription_callback)

        elif (kpm_report_style == 2):

            # need to bind also UE_ID to callback as it is not present
                 in the RIC indication in the case of E2SM KPM Report
                Style 2

            subscription_callback = lambda agent, sub, hdr, msg: self.
                my_subscription_callback(agent, sub, hdr, msg,
                kpm_report_style, ue_ids[0])

            print("Subscribe to E2 node ID: {}, RAN func: e2sm_kpm,
                Report Style: {}, UE_id: {}, metrics: {}".format(
                e2_node_id, kpm_report_style, ue_ids[0], metric_names)
                )
            self.e2sm_kpm.subscribe_report_service_style_2(e2_node_id,
                 report_period, ue_ids[0], metric_names, granul_period
                , subscription_callback)

        elif (kpm_report_style == 3):

            if (len(metric_names) > 1):
                metric_names = metric_names[0]
                print("INFO: Currently only 1 metric can be requested
                    in E2SM-KPM Report Style 3, selected metric: {}".
                    format(metric_names))

            # TODO: currently only dummy condition that is always
                satisfied, useful to get IDs of all connected UEs

            # example matching UE condition: ul-rSRP < 1000
```

Figura F.10: *kpm_mon_xapp.py* (II).

```
            matchingConds = [{'matchingCondChoice': ('testCondInfo', {
                'testType': ('ul-rSRP', 'true'), 'testExpr': 'lessthan
                ', 'testValue': ('valueInt', 1000)})}]

            print("Subscribe to E2 node ID: {}, RAN func: e2sm_kpm,
                Report Style: {}, metrics: {}".format(e2_node_id,
                kpm_report_style, metric_names))
            self.e2sm_kpm.subscribe_report_service_style_3(e2_node_id,
                 report_period, matchingConds, metric_names,
                granul_period, subscription_callback)


        elif (kpm_report_style == 4):

            # TODO: currently only dummy condition that is always
                satisfied, useful to get IDs of all connected UEs

            # example matching UE condition: ul-rSRP < 1000

            matchingUeConds = [{'testCondInfo': {'testType': ('ul-rSRP
                ', 'true'), 'testExpr': 'lessthan', 'testValue': ('
                valueInt', 1000)}}]

            print("Subscribe to E2 node ID: {}, RAN func: e2sm_kpm,
                Report Style: {}, metrics: {}".format(e2_node_id,
                kpm_report_style, metric_names))

            self.e2sm_kpm.subscribe_report_service_style_4(e2_node_id,
                 report_period, matchingUeConds, metric_names,
                granul_period, subscription_callback)

        elif (kpm_report_style == 5):

            if (len(ue_ids) < 2):
                dummyUeId = ue_ids[0] + 1
                ue_ids.append(dummyUeId)
                print("INFO: Subscription for E2SM_KPM Report Service
                    Style 5 requires at least two UE IDs -> add dummy
                    UeID: {}".format(dummyUeId))

            print("Subscribe to E2 node ID: {}, RAN func: e2sm_kpm,
                Report Style: {}, UE_ids: {}, metrics: {}".format(
                e2_node_id, kpm_report_style, ue_ids, metric_names))

            self.e2sm_kpm.subscribe_report_service_style_5(e2_node_id,
                 report_period, ue_ids, metric_names, granul_period,
                subscription_callback)
        else:
            print("INFO: Subscription for E2SM_KPM Report Service
                Style {} is not supported".format(kpm_report_style))

            exit(1)

if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='My example xApp')
    parser.add_argument("--config", type=str, default='', help="xApp
        config file path")
    parser.add_argument("--http_server_port", type=int, default=8092,
        help="HTTP server listen port")
    parser.add_argument("--rmr_port", type=int, default=4562, help="
        RMR port")
```

Figura F.11: *kpm_mon_xapp.py* (III).

```
    parser.add_argument("--e2_node_id", type=str, default='
        gnbd_001_001_00019b_0', help="E2 Node ID")
    parser.add_argument("--ran_func_id", type=int, default=2, help="
        RAN function ID")
    parser.add_argument("--kpm_report_style", type=int, default=1,
        help="xApp config file path")
    parser.add_argument("--ue_ids", type=str, default='0', help="UE ID
        ")
    parser.add_argument("--metrics", type=str, default='DRB.UEThpUl,
        DRB.UEThpDl', help="Metrics name as comma-separated string")

    args = parser.parse_args()
    config = args.config
    e2_node_id = args.e2_node_id # TODO: get available E2 nodes from
        SubMgr, now the id has to be given.
    ran_func_id = args.ran_func_id # TODO: get available E2 nodes from
         SubMgr, now the id has to be given.
    ue_ids = list(map(int, args.ue_ids.split(","))) # Note: the UE id
        has to exist at E2 node!
    kpm_report_style = args.kpm_report_style
    metrics = args.metrics.split(",")

    # Create MyXapp.
    myXapp = MyXapp(config, args.http_server_port, args.rmr_port)
    myXapp.e2sm_kpm.set_ran_func_id(ran_func_id)

    # Connect exit signals.
    signal.signal(signal.SIGQUIT, myXapp.signal_handler)
    signal.signal(signal.SIGTERM, myXapp.signal_handler)
    signal.signal(signal.SIGINT, myXapp.signal_handler)

    # Start xApp.
    myXapp.start(e2_node_id, kpm_report_style, ue_ids, metrics)
    # Note: xApp will unsubscribe all active subscriptions at exit.
```

Figura F.12: *kpm_mon_xapp.py* (IV).

F.3. FlexRIC en srsRAN

Se hace uso de *xapp_oran_moni.c* de FlexRIC, del mismo modo en la monitorización de métricas. Debido a su gran extensión en comparación con las demás *xApps* mostradas en este apéndice, no se presenta su contenido explícito, pero se puede encontrar en **??**. A ella se le pasa un archivo de configuración con las métricas a medir. A continuación, se puede ver este.

```
# Ruta del directorio donde están las Service Models (SM)
SM_DIR = "/usr/local/lib/flexric/"

# Nombre del xApp (debe coincidir con el soportado)
Name = "xApp"
```

Figura F.13: *xapp_mon_e2sm_kpm.conf* (I).

```
# Dirección IP del RIC y puerto E2
NearRT_RIC_IP = "127.0.0.1"
E42_Port = 36422

# Lista de Service Models a suscribir (en este caso, solo KPM)
Sub_ORAN_SM_List = (
    {
      # Nombre del SM: KPM (Key Performance Measurements)
      name = "KPM",
      # Intervalo de muestreo en milisegundos
      time = 1000,
      # Formato de codificación del SM (1 = ASN.1)
      format = 1,
      # Tipo de RAN al que se aplica (ngran_gNB_DU para gNB-DU)
      ran_type = "ngran_gNB_DU",

      # Lista de métricas que se desean monitorizar
      actions = (
            # Throughput DL por UE
            { name = "DRB.UEThpDl" },
            # Throughput UL por UE
            { name = "DRB.UEThpUl" },
            # PRBs disponibles en DL
            { name = "RRU.PrbAvailDl" },
            # PRBs disponibles en UL
            { name = "RRU.PrbAvailUl" },
            # Volumen RLC SDU transmitido en DL
            { name = "DRB.RlcSduTransmittedVolumeDL" },
            # Volumen RLC SDU transmitido en UL
            { name = "DRB.RlcSduTransmittedVolumeUL" },
            # Tasa de pérdida de paquetes en RLC DL
            { name = "DRB.RlcPacketDropRateDl" }
            )
    }
)
xApp_DB = {
    # Activación del almacenamiento de métricas en base de datos (ON/
        OFF)
    enable = "OFF"
    # Dirección IP del servidor de base de datos
    ip = "127.0.0.1"
    # Directorio local donde guardar la base de datos en modo OFF
    dir = "/tmp/"
    # Nombre del archivo donde se almacenarán los datos (modo local)
    filename = "testdb"
    # Nombre de usuario para conexión a base de datos (si se usa MySQL
        )
    username = "your_username" # if using mysql
    # Contraseña para conexión a base de datos (si se usa MySQL)
    password = "your_passwd" # if using mysql
}
```

Figura F.14: *xapp_mon_e2sm_kpm.conf* (II).

Anexo G

xApps para control de PRBs

En este apéndice se muestra la *simple_xapp.py xApp* implementada por la comunidad de srsRAN y la *simple_xapp_custom.py* desarrollada por la alumna, teniendo como base la anterior. Con ambas, se lleva a cabo un control de PRBs en el escenario srsRAN por medio del ORAN-SC-RIC. En las figuras G.1, G.2 y G.3 se muestra *simple_xapp.py*:

```
#!/usr/bin/env python3

import time
import datetime
import argparse
import signal
from lib.xAppBase import xAppBase

class MyXapp(xAppBase):
    def __init__(self, http_server_port, rmr_port):
        super(MyXapp, self).__init__('', http_server_port, rmr_port)
        self.ue_dl_tx_data = {}
        self.min_prb_ratio = 1
        self.max_prb_ratio1 = 10
        self.max_prb_ratio2 = 100
        self.cur_ue_max_prb_ratio = {}
        self.dl_tx_data_threshold_mb = 4

    def my_subscription_callback(self, e2_agent_id, subscription_id,
        indication_hdr, indication_msg, kpm_report_style, ue_id):
        indication_hdr = self.e2sm_kpm.extract_hdr_info(indication_hdr
            )
        meas_data = self.e2sm_kpm.extract_meas_data(indication_msg)

        print("Data Monitoring:")
        print("  E2SM_KPM RIC Indication Content:")
        print("  -ColletStartTime: ", indication_hdr['colletStartTime'
            ])
        print("  -Measurements Data:")
```

Figura G.1: *simple_xapp.py xApp* (I).

```
        for ue_id, ue_meas_data in meas_data["ueMeasData"].items():
            print("  --UE_id: {}".format(ue_id))
            granulPeriod = ue_meas_data.get("granulPeriod", None)
            if granulPeriod is not None:
                print("  ---granulPeriod: {}".format(granulPeriod))

            for metric_name, values in ue_meas_data["measData"].items
                ():
                print("  ---Metric: {}, Value: {:.1f} [MB]".format(
                    metric_name, sum(values)/8/1000))

                if (metric_name == "DRB.RlcSduTransmittedVolumeDL"):
                    if ue_id in self.ue_dl_tx_data:
                        # Reported in kbits, convert to MBs.
                        self.ue_dl_tx_data[ue_id] += sum(values)
                            /8/1000
                    else:
                        self.ue_dl_tx_data[ue_id] = sum(values)/8/1000
        print("")
        print("Control Logic:")
        print(" Tx Data Stats:")
        for ue_id, value in self.ue_dl_tx_data.items():
            cur_ue_max_prb_ratio = self.cur_ue_max_prb_ratio.get(ue_id
                , 0)
            if cur_ue_max_prb_ratio:
                print(f'  UE ID: {ue_id}, Max PRB Ratio: {
                    cur_ue_max_prb_ratio}, Total TXed Data [MB]: {
                    value:.1f}')
            else:
                print(f'  UE ID: {ue_id}, Max PRB Ratio: n/a, TXed
                    Data [MB]: {value:.1f}')
            if (value > self.dl_tx_data_threshold_mb):
                print(f"    {value:.1f} MB of data transmitted to UE
                    --> Switch Max PRB limit")
                cur_ue_max_prb_ratio = self.cur_ue_max_prb_ratio.get(
                    ue_id, self.max_prb_ratio2)
                new_ue_max_prb_ratio = self.max_prb_ratio2 if
                    cur_ue_max_prb_ratio == self.max_prb_ratio1 else
                    self.max_prb_ratio1
                # Reset collected TX data volume.
                self.ue_dl_tx_data[ue_id] = 0
                self.cur_ue_max_prb_ratio[ue_id] =
                    new_ue_max_prb_ratio
                print("    --->Send RIC Control Request to E2 node ID:
                     {} for UE ID: {}, PRB_min: {}, PRB_max: {}".
                    format(e2_agent_id, ue_id, self.min_prb_ratio,
                    new_ue_max_prb_ratio))
                self.e2sm_rc.control_slice_level_prb_quota(e2_agent_id
                    , ue_id, min_prb_ratio=self.min_prb_ratio,
                    max_prb_ratio=new_ue_max_prb_ratio,
                    dedicated_prb_ratio=100, ack_request=1)
        print("------------------------------------------------------"
            )
        print("")
```

Figura G.2: *simple_xapp.py xApp* (II).

```
    # Mark the function as xApp start function using xAppBase.
        start_function decorator.
    # It is required to start the internal msg receive loop.
    @xAppBase.start_function
    def start(self, e2_node_id, kpm_report_style, ue_ids, metric_names
        ):
        report_period = 1000
        granul_period = 1000

        subscription_callback = lambda agent, sub, hdr, msg: self.
            my_subscription_callback(agent, sub, hdr, msg,
            kpm_report_style, None)

        # Dummy condition that is always satisfied
        matchingUeConds = [{'testCondInfo': {'testType': ('ul-rSRP', '
            true'), 'testExpr': 'lessthan', 'testValue': ('valueInt',
            1000)}}]

        print("Subscribe to E2 node ID: {}, RAN func: e2sm_kpm, Report
             Style: {}, metrics: {}".format(e2_node_id,
            kpm_report_style, metric_names))

        self.e2sm_kpm.subscribe_report_service_style_4(e2_node_id,
            report_period, matchingUeConds, metric_names,
            granul_period, subscription_callback)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='My example xApp')
    parser.add_argument("--http_server_port", type=int, default=8090,
        help="HTTP server listen port")
    parser.add_argument("--rmr_port", type=int, default=4560, help="
        RMR port")
    parser.add_argument("--e2_node_id", type=str, default='
        gnbd_001_001_00019b_0', help="E2 Node ID")
    parser.add_argument("--ran_func_id", type=int, default=2, help="
        RAN function ID")
    parser.add_argument("--kpm_report_style", type=int, default=4,
        help="KPM Report Style ID")
    parser.add_argument("--ue_ids", type=str, default='0', help="UE ID
        ")
    parser.add_argument("--metrics", type=str, default='DRB.
        RlcSduTransmittedVolumeDL', help="Metrics name as comma-
        separated string")

    args = parser.parse_args()
    e2_node_id = args.e2_node_id # TODO: get available E2 nodes from
        SubMgr, now the id has to be given.
    ran_func_id = args.ran_func_id # TODO: get available E2 nodes from
         SubMgr, now the id has to be given.
    ue_ids = list(map(int, args.ue_ids.split(","))) # Note: the UE id
        has to exist at E2 node!
    kpm_report_style = args.kpm_report_style
    metrics = args.metrics.split(",")

    # Create MyXapp.
    myXapp = MyXapp(args.http_server_port, args.rmr_port)
    myXapp.e2sm_kpm.set_ran_func_id(ran_func_id)

    # Connect exit signals.
    signal.signal(signal.SIGQUIT, myXapp.signal_handler)
    signal.signal(signal.SIGTERM, myXapp.signal_handler)
    signal.signal(signal.SIGINT, myXapp.signal_handler)

    # Start xApp.
    myXapp.start(e2_node_id, kpm_report_style, ue_ids, metrics)
    # Note: xApp will unsubscribe all active subscriptions at exit.
```

Figura G.3: *simple_xapp.py xApp* (III).

Mientras, en las figuras G.4, G.5 y G.6 es posible observar el desarrollo de la nueva:

```
#!/usr/bin/env python3
# Importaciones necesarias para el funcionamiento de la xApp
import time
import signal
import argparse
from lib.xAppBase import xAppBase

# Clase principal de la xApp que hereda de xAppBase
class MyXapp(xAppBase):
    def __init__(self, http_server_port, rmr_port):
        super(MyXapp, self).__init__('', http_server_port, rmr_port)

        # Diccionarios para almacenar métricas de cada UE
        self.ue_th_dl = {}  # Throughput de bajada por UE
        self.ue_prb_avail_dl = {}  # PRBs disponibles en DL por UE

        # Parámetros de configuración de ratios de PRBs
        self.min_prb_ratio = 1
        self.max_prb_ratio1 = 10
        self.max_prb_ratio2 = 100
        self.cur_ue_max_prb_ratio = "n/a"

        # Umbrales para detectar congestión
        self.dl_tx_data_max_threshold_kbps = 28500
        self.dl_prb_avail_min_threshold = 15

    # Función de callback ejecutada al recibir datos del nodo E2
    def my_subscription_callback(self, e2_agent_id, subscription_id,
        indication_hdr, indication_msg, kpm_report_style, ue_id):
        indication_hdr = self.e2sm_kpm.extract_hdr_info(indication_hdr
            )
        meas_data = self.e2sm_kpm.extract_meas_data(indication_msg)

        # Procesamiento de datos recibidos por el UE
        for ue_id, ue_meas_data in meas_data["ueMeasData"].items():
            for metric_name, values in ue_meas_data["measData"].items
                ():
                if isinstance(values, list) and len(values) > 0:
                    values = values[0]
                if metric_name == "DRB.UEThpDl":
                    self.ue_th_dl[ue_id] = values
                if metric_name == "RRU.PrbAvailDl":
                    self.ue_prb_avail_dl[ue_id] = values

        # Lógica de control basada en métricas: detectar congestión y
            aplicar protección
        for ue_id, ue_meas_data in meas_data["ueMeasData"].items():
            print("  --UE_id: {}".format(ue_id))
            granulPeriod = ue_meas_data.get("granulPeriod", None)
            if granulPeriod is not None:
                print("  ---granulPeriod: {}".format(granulPeriod))
```

Figura G.4: *xApp* que adapta la asignación de PRBs en función del estado del canal (I).

```
            # Procesar las métricas
            for metric_name, values in ue_meas_data["measData"].items():
                print("---Metric: {}, Value: {}".format(metric_name, values))

                if isinstance(values, list) and len(values) > 0:
                    values = values[0]  # Tomar el primer valor si es una lista no vacía
                # Evaluar y almacenar las métricas
                if metric_name == "DRB.UEThpDl":
                    self.ue_th_dl[ue_id] = values
                if metric_name == "RRU.PrbAvailDl":
                    self.ue_prb_avail_dl[ue_id] = values
        print("")
        print("Control Logic:")
        print(" Tx Data Stats:")

        # Evaluar la condición de congestión
        for ue_id, _ in self.ue_th_dl.items():
            th_dl = self.ue_th_dl.get(ue_id, 0)
            prb_avail_dl = self.ue_prb_avail_dl.get(ue_id, 0)

            # Escenario de Congestión -> Medida de Protección
            if th_dl > self.dl_tx_data_max_threshold_kbps and prb_avail_dl < self.dl_prb_avail_min_threshold:
                new_ue_max_prb_ratio = self.max_prb_ratio1
                if self.cur_ue_max_prb_ratio != self.max_prb_ratio1:
                    previous_prb_ratio = self.cur_ue_max_prb_ratio
                    self.cur_ue_max_prb_ratio = new_ue_max_prb_ratio
                    print(f"        -- UE ID: {ue_id}, Max PRB Ratio: {previous_prb_ratio}, Throughput DL: {th_dl}, PRB Avail DL: {prb_avail_dl}")
                    print(f"           Congestion detected. Reducing PRB allocation. New Max PRB Ratio: {self.cur_ue_max_prb_ratio}")
                    self.e2sm_rc.control_slice_level_prb_quota(e2_agent_id, ue_id, min_prb_ratio=self.min_prb_ratio, max_prb_ratio=self.cur_ue_max_prb_ratio, dedicated_prb_ratio=100, ack_request=1)

                   # Restablecer a más PRB gradualmente
                    time.sleep(5)
                    print(f" ")
                    print(f" ---------------------------- Restoring Capacities ----------------------------------------")
                    print(f" ")
                    increments = [2, 4, 4, 5]
                    for increment in increments:
                        previous_prb_ratio = self.cur_ue_max_prb_ratio
                        self.cur_ue_max_prb_ratio += increment
                        print(f"                      -- UE ID: {ue_id}, PRB ratio allocation restored to: {self.cur_ue_max_prb_ratio}")
                        self.e2sm_rc.control_slice_level_prb_quota(e2_agent_id, ue_id, min_prb_ratio=self.min_prb_ratio, max_prb_ratio=self.cur_ue_max_prb_ratio, dedicated_prb_ratio=100, ack_request=1)
                        time.sleep(5)
                    print("----------------------------------------")
                    print("")
```

Figura G.5: *xApp* que adapta la asignación de PRBs en función del estado del canal (II).

```
                    # Esperar antes de restablecer todos los PRBs al principio
                    time.sleep(10)
                    previous_prb_ratio = self.cur_ue_max_prb_ratio
                    print(f" ---------------------------- Restoring All PRBs to Maximum ----------------------------")
                    self.cur_ue_max_prb_ratio = self.max_prb_ratio2  # Asignar todos los PRBs disponibles
                    print(f"                     -- UE ID: {ue_id}, PRB ratio allocation set back to 100%")
                    self.e2sm_rc.control_slice_level_prb_quota(e2_agent_id, ue_id, min_prb_ratio=self.min_prb_ratio, max_prb_ratio=self.cur_ue_max_prb_ratio, dedicated_prb_ratio=100, ack_request=1)
                    time.sleep(5)  # Tiempo de espera adicional si es necesario
                    print("----------------------------------------")
                    print("")

    # Función principal de arranque de la xApp
    @xAppBase.start_function
    def start(self, e2_node_id, kpm_report_style, ue_ids, metric_names):
        report_period = 1000
        granul_period = 1000

        # Crear callback de suscripción
        subscription_callback = lambda agent, sub, hdr, msg: self.my_subscription_callback(agent, sub, hdr, msg, kpm_report_style, None)

        # Condición para filtrar UEs con baja calidad de señal
        matchingUeConds = [{'testCondInfo': {'testType': ('ul-rSRP', 'true'), 'testExpr': 'lessthan', 'testValue': ('valueInt', 1000)}}]

        # Realizar la suscripción al nodo
        self.e2sm_kpm.subscribe_report_service_style_4(e2_node_id, report_period, matchingUeConds, metric_names, granul_period, subscription_callback)

# Ejecución principal del script
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("--http_server_port", type=int, default=8090)
    parser.add_argument("--rmr_port", type=int, default=4560)
    parser.add_argument("--e2_node_id", type=str, default='gnbd_001_001_00019b_0')
    parser.add_argument("--ran_func_id", type=int, default=2)
    parser.add_argument("--kpm_report_style", type=int, default=4)
    parser.add_argument("--ue_ids", type=str, default='0')
    parser.add_argument("--metrics", type=str, default='DRB.RlcSduTransmittedVolumeDL')

    args = parser.parse_args()
    e2_node_id = args.e2_node_id
    ran_func_id = args.ran_func_id
    ue_ids = list(map(int, args.ue_ids.split(",")))
    kpm_report_style = args.kpm_report_style
    metrics = args.metrics.split(",")
```

Figura G.6: *xApp* que adapta la asignación de PRBs en función del estado del canal (III).

```
    # Crear e iniciar la xApp
    myXapp = MyXapp(args.http_server_port, args.rmr_port)
    myXapp.e2sm_kpm.set_ran_func_id(ran_func_id)

    # Captura de señales para terminación limpia
    signal.signal(signal.SIGQUIT, myXapp.signal_handler)
    signal.signal(signal.SIGTERM, myXapp.signal_handler)
    signal.signal(signal.SIGINT, myXapp.signal_handler)

    myXapp.start(e2_node_id, kpm_report_style, ue_ids, metrics)
```

Figura G.7: *xApp* que adapta la asignación de PRBs en función del estado del canal (IV).