ISSN 1688-2806

Universidad de la República
Facultad de Ingeniería

O-RAN: Puerta de entrada para nuevos investigadores

Tesis presentada a la Facultad de Ingeniería de la Universidad de la República por

Ing. Juan Navarro Gutiérrez

en cumplimiento parcial de los requerimientos para la finalización de la carrera de Maestría en Ingeniería Eléctrica.

Dirección de Tesis

Dra. Claudina Rattaro ..... Universidad de la República
Dr. Alberto Castro ..... Universidad de la República

Tribunal

Ing. Natalia Pignataro ..... Antel
Dr. Carlos Astudillo ..... UNICAMP
Dr. Matias Richart ..... Universidad de la República

Dirección Académica

Dra. Claudina Rattaro ..... Universidad de la República

Montevideo
miércoles 4 febrero, 2026

O-RAN: Puerta de entrada para nuevos investigadores, Ing. Juan Navarro Gutiérrez.

ISSN 1688-2806

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.2).
Contiene un total de 153 páginas.
Compilada el miércoles 4 febrero, 2026.
http://iie.fing.edu.uy/

Agradecimientos

A Carolina por estar siempre.

A mi familia, por inculcarme valores fundamentales, enseñarme el significado del compromiso y acompañarme en cada desafío.

A mis tutores Claudina y Alberto por el acompañamiento, motivación y apoyo durante la realización de este trabajo.

A la Universidad de la República y la educación pública.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

La evolución tecnológica de las redes móviles ha sido una constante en el ámbito de las telecomunicaciones, basta comparar las capacidades de las redes de hace quince años con las prestaciones disponibles en la actualidad para evidenciar avances sustanciales en eficiencia, rendimiento y servicios. Sin embargo, este progreso no se ha visto acompañado por una transformación equivalente en las políticas internas ni en el modelo comercial del ecosistema móvil, donde persiste una estructura dominada por los mismos actores tradicionales: las operadoras y los fabricantes de equipamiento. Las diferencias entre ambos sectores se han mantenido a lo largo del tiempo debido a sus intereses contrapuestos: mientras las operadoras buscan mayor interoperabilidad, apertura y flexibilidad para optimizar costos y diversificar proveedores, los fabricantes tradicionales tienden a mantener sistemas monolíticos y cerrados que sostengan su modelo de integración vertical. Esta divergencia ha limitado históricamente la adopción de arquitecturas más abiertas y modulares en las redes móviles.

En este contexto surge el enfoque Open RAN (O-RAN), una iniciativa que propone desagregar funcionalmente la red de acceso de radio y definir interfaces abiertas y estandarizadas. Su objetivo es habilitar un ecosistema más diverso, interoperable y competitivo, permitiendo la integración de componentes de distintos proveedores y fomentando la innovación en capas software, algoritmos de optimización y arquitectura de red. O-RAN se presenta así como una respuesta concreta a las limitaciones del modelo tradicional, promoviendo una evolución no solo tecnológica, sino también estructural en el despliegue y operación de las redes móviles.

Dado que O-RAN es una tecnología relativamente reciente y cuya información disponible aún no se encuentra ampliamente consolidada, esta tesis aborda un estudio teórico-práctico de O-RAN, con el propósito de reunir conocimiento que hoy está disperso entre documentos técnicos, especificaciones, implementaciones y experiencias de laboratorio. Se analizan en detalle los componentes fundamentales del ecosistema O-RAN, así como su arquitectura y especificaciones clave, para luego avanzar hacia la evaluación práctica de distintas implementaciones en entornos de laboratorio. Un aspecto central de este trabajo es la selección y el uso de herramientas libres, de bajo costo, bajo consumo y escalables, con el objetivo de democratizar la investigación y reducir las barreras de entrada para quienes cuentan con recursos limitados. De esta manera, se demuestra que es posible realizar experimentación sobre O-RAN sin depender de equipamiento costoso, ampliando así las posibilidades de investigación en ámbitos académicos y de pequeña escala.

Finalmente, se documenta de forma sistemática el proceso de instalación, configuración y uso de estas plataformas, proporcionando una guía clara y reproducible para futuros desarrollos en el área.

Tabla de contenidos

Capítulo 1

Introducción

Históricamente, los principales involucrados en todo lo relacionado con las redes móviles han sido los fabricantes de equipamiento y las operadoras. Aunque ambos colaboren activamente en el desarrollo de estándares y tecnologías para la red móvil a través de organizaciones como la 3GPP[1](*Third Generation Partnership Project*), la realidad es que tienen intereses contrapuestos. Por ejemplo las operadoras van a preferir sistemas abiertos e interoperables mientras que los fabricantes suelen impulsar características propietarias, del mismo modo, a las operadoras les conviene tener arquitecturas desagregadas pero los fabricantes favorecen soluciones *end-to-end*. Además, aunque algunas operadoras sean de las empresas más grandes e influyentes a nivel mundial (China Mobile, Orange, AT&T, Verizon, etc.), globalmente son cientos de empresas, mientras que el 95 %[2] del mercado mundial de equipamiento para la RAN está repartido entre 5 empresas (Huawei, Ericsson, Nokia, ZTE y Samsung). Esto pone a las operadoras en una situación de desventaja y dependencia frente a los fabricantes.

Estas tensiones que surgen desde los intereses económicos de las partes impactan directamente en la innovación, ya que la casi nula competencia y el hermetismo hacen que el resto de actores (particulares, academia, pequeñas empresas, etc.) sean casi que únicamente espectadores. Sí se puede estudiar, entender, analizar y proponer infinidad de soluciones y mejoras, pero difícilmente lleguen al usuario final sin que un fabricante decida adoptar dicha innovación en sus equipos. Por otra parte, el acceso a bancos de pruebas (*testbeds*) es limitado, ya que nuevamente se depende de fabricantes privados.

El panorama comenzó a cambiar con la llegada de la virtualización de las funciones de red, que permite virtualizar determinados componentes de la red móvil. En un principio, la virtualización de funciones de red se centró en el *core* de la red. Desde el punto de vista de las operadoras, fue un avance, aunque seguían dependiendo de los fabricantes para el software que implementaba los diferentes componentes del núcleo de red; ahora podían correrlo en hardware no propietario.

[1]https://www.3gpp.org/

[2]https://techblog.comsoc.org/2025/02/18/omdia-huawei-increases-global-ran-market-share-due-to-china-hegemony/

A pesar de estos avances en el núcleo de la red, la RAN siguió bajo el mismo paradigma: equipos cerrados y propietarios con poca o nula interoperabilidad y dependencia del fabricante.

El siguiente hito significativo en este camino hacia la apertura de las redes móviles fue el surgimiento, a partir de 2014, de plataformas de emulación como Open Air Interface (OAI)[3], srsRAN[4] o Open5GS[5]. Estas plataformas favorecieron, sobre todo, al ámbito académico, ya que ahora se podían realizar despliegues de redes móviles en el laboratorio de manera sencilla y relativamente económica. A partir de mediados de la década pasada hubo un auge en la producción de artículos científicos sobre las redes móviles, sobre todo 5G y LTE[6]. A pesar de todo esto, en los despliegues comerciales la situación de la RAN seguía casi sin cambios.

Apoyado en todo lo anterior es que por 2014/2015 comienza a germinar la idea de abrir la RAN, en principio fue a través de consorcios regionales de operadoras pero al ver que era una causa común en 2018 se formalizó la Open RAN Alliance[7], consorcio global fundado por las principales operadoras[8] para lograr el objetivo de tener una RAN abierta.

En este punto es importante preguntarse qué es y qué implica tener una RAN abierta. Una RAN abierta es una RAN que no depende de un fabricante, es una RAN interopereable entre diferentes proveedores, es una RAN desagregada y basada en la virtualización de las funciones de red, es una RAN en la que cualquier empresa grande o pequeña puede prestarle servicios, es una RAN que facilita la innovación y el desarrollo de nuevas tecnologías y aplicaciones. Todo esto implica un fuerte esfuerzo por estandarizar y definir normas y reglas, así como la correcta documentación y publicación de dichos estándares. Todas estas tareas recaen en la Open RAN Alliance, que no demoraron en comprender que un cambio de esta escala necesita de todos los actores, es por ello que en la actualidad la Open RAN Alliance esta conformada por operadoras, universidades, empresas privadas e incluso fabricantes de equipamiento.

Aunque es una tecnología relativamente nueva se ha avanzado bastante en estos años, la Open RAN Alliance ya ha publicado varios documentos con estándares y definiciones, en la literatura académica hay publicaciones del tema y han aparecido algunas soluciones comerciales asociadas, pero justamente al ser todo tan reciente la puerta de entrada a los investigadores a este tema no es tan visible como con otros tópicos. Aunque se encuentra información al respecto, no abunda.

Ya hablando del ámbito local, no se encontraron publicaciones sobre O-RAN en Colibrí[9] ni publicaciones asociadas a universidades nacionales en internet[10].

[3] https://openairinterface.org/

[4] https://www.srslte.com/

[5] https://open5gs.org/

[6] https://www.sciencedirect.com/science/article/pii/S0308596122000301

[7] https://www.o-ran.org/about

[8] https://mediastorage.o-ran.org/overview/O-RAN.Overview-of-the-O-RAN-ALLIANCE-presentation.pdf

[9] https://www.colibri.udelar.edu.uy

[10] Búsqueda realizada en octubre de 2025 utilizando el buscador del sitio Colibrí y buscadores de internet filtrando por “ORAN”, “O-RAN”, “Open RAN”. Así mismo se utilizó

1.0.1. Objetivos

El objetivo de este trabajo innovador a nivel nacional es facilitarles la entrada a O-RAN a otros investigadores. La idea es que quien quiera comenzar a trabajar con O-RAN comience leyendo este documento, de esta forma se ahorra el tiempo de recabar y filtrar información para concentrarse en la tarea especifica sobre O-RAN que vaya a desarrollar. Para ello se trabajó con un enfoque teórico-práctico. Primero se explica en profundidad la teoría (involucrados, arquitectura, componentes, interfaces, protocolos, etc.) para luego pasar a una descripción detallada de cómo desplegar varios tipos de plataformas de prueba (se abordan desde opciones todo en uno, pasando por plataformas simuladas, hasta testbeds que emulan la red real) y cómo utilizarlas. Esto es fundamental, ya que, como se verá en los próximos capítulos, una de las ideas disruptivas de O-RAN es la posibilidad de desarrollar aplicaciones para realizar determinadas tareas puntuales en la red. Estas aplicaciones podrían ser desarrolladas por cualquier programador con los conocimientos necesarios. Incluso se entiende que el desarrollo de este tipo de aplicaciones será el principal motivo de estudio de O-RAN, por lo que contar con un entorno donde realizar pruebas y prácticas de laboratorio es fundamental.

Es importante aclarar que, aunque en este documento se muestran comparaciones prácticas de diferentes plataformas, el objetivo del trabajo no es compararlas, sino presentar un conjunto de opciones para que el investigador pueda decidir cuál es la indicada según sus requerimientos.

1.0.2. Metodología y contenido

A nivel general el trabajo se puede dividir en dos partes, una teórica y otra práctica. Para el marco teórico se recabo información de artículos científicos, documentación oficial y libros de la temática. De todas esas fuentes se compararon definiciones y puntos de vista para así reflejarlos de la mejor manera en este documento. Además se trato de simplificar algunos conceptos que aun no están del todo claro y de profundizar en otros que ya están lo suficientemente maduros.

Para la parte práctica que consiste en el desarrollo y prueba de *testbeds* se comenzó realizando un análisis del estado del arte, o sea que implementaciones se han realizado hasta el momento. Luego se tomaron las mas prometedoras y se trataron de implementar siguiendo la documentación de cada una de ellas. En todos los casos hubieron que hacer ajustes o modificaciones para lograr desplegarlas sin errores. Para esto se estudio la documentación de las herramientas utilizadas y sobre todo experimentación práctica. Una vez que se obtuvo un resultado estable y replicable se procedió a documentar detalladamente todos los pasos.

Cabe destacar que para la realización de este trabajo se consultaron mas de 50 fuentes principales y mas de 70 fuentes secundarias. Este esfuerzo permitió contrastar enfoques, identificar tendencias recientes y asegurar que el análisis se apoye en evidencia sólida y actualizada.

En relación al documento, en el capítulo 2 se hace un repaso por evolución de

la herramienta ChatGPT para ampliar la búsqueda

la RAN a lo largo de la historia ya que es fundamental conocer dicha evolución para entender cómo se llega a O-RAN y cómo al mismo tiempo O-RAN recicla la mayoría de los conceptos. Luego en el mismo capítulo se introduce a la Open RAN Alliance profundizando en sus grupos de trabajo y como se organizan. Finalmente, el capítulo termina recopilando definiciones de O-RAN de otros autores y ofrece una definición propia.

En el capítulo 3 se explica detalladamente la arquitectura de O-RAN tanto física (componentes, interfaces, etc.) como a nivel de protocolos utilizados. Además, se introducen las xApps y las rApps (aplicaciones propias o de terceros que corren dentro de O-RAN). En el capítulo 4 se realiza un relevamiento de testbeds desarrollados por otros investigadores y se hace un breve repaso de las herramientas utilizadas en la mayoría de ellos.

En el capítulo 5 se explica la arquitectura y se detalla cómo instalar y utilizar cinco implementaciones diferentes de testbeds; luego, en el capítulo 6, con el objetivo de mejorar la comprensión de tres de estas implementaciones, se presentan experimentos prácticos para mostrar su utilidad, versatilidad y aplicabilidad.

Finalmente, el capítulo 7 concluye el trabajo con reflexiones finales y perspectivas a futuro.

Capítulo 2

Open RAN

En este capítulo se dará respuesta a la pregunta: ¿Qué es Open RAN? Para ello, es indispensable comenzar por hacer un breve repaso histórico de la RAN y ver cómo definen Open RAN otros autores. Luego, se presentará al principal gestor de Open RAN, la Open RAN Alliance, para responder finalmente la pregunta planteada.

2.1. Evolución de la RAN

Históricamente, la red móvil puede dividirse en tres partes bien definidas: los equipos de usuario o *user equipment* (UE), la red de acceso por radio o *radio access network* (RAN) y la red de núcleo o *core network* (CN).

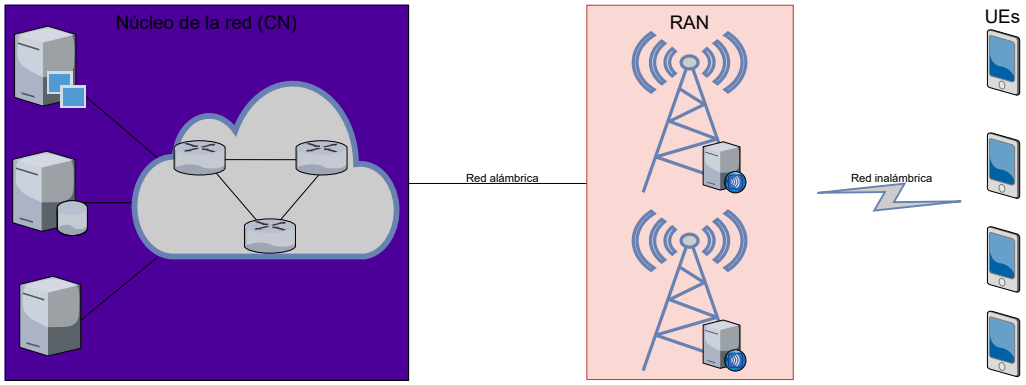

Figura 2.1: Arquitectura genérica de la red móvil

Equipos de Usuario (UE): Los UEs son dispositivos móviles, como smartphones, tablets y otros dispositivos de Internet de las Cosas (Internet of Things, IoT), que consumen los servicios proporcionados por la CN. Red de Acceso por Radio (RAN): La RAN es la interfaz entre los UEs y la red central. A través de estaciones base, la RAN proporciona conectividad inalámbrica y gestiona el acceso de los dispositivos a la red. Red de Núcleo (CN): La CN es el centro de la red móvil,

responsable de gestionar los servicios críticos, como la autenticación de usuarios, la movilidad (cambio de celdas sin interrupciones), la interconexión con otras redes (internet y redes telefónicas) y la administración de datos. En redes modernas, como 5G, el núcleo se ha transformado en una arquitectura basada en funciones de red virtualizadas (VNFs), lo que permite una mayor flexibilidad y escalabilidad.

La interacción entre estos componentes permite que los UEs se conecten a través de la RAN y accedan a los servicios y recursos gestionados en la CN.

Desde la tercera generación móvil (3G), la 3GPP[1] (*3rd Generation Partnership Project*) define los componentes de la RAN y la CN, así como las interfaces que interconectan dichos elementos y el stack de protocolos. A continuación se presenta brevemente cómo ha ido evolucionando la RAN desde 3G hasta el actual 5G haciendo énfasis en los aspectos que más adelante participan en la idea de Open RAN.

3G

La red móvil de tercera generación, también conocida como *Universal Mobile Telecommunications System* (UMTS), surge a principios de los 2000 como la evolución de GSM (2G).

La tecnología 3G representa un avance significativo en comparación con 2G, ofreciendo mejoras en la velocidad de transmisión de datos, la capacidad de la red y la eficiencia. A diferencia de 2G, que estaba optimizado principalmente para llamadas de voz y mensajes de texto, 3G permite una transmisión de datos considerablemente más rápida, lo que habilita servicios de internet móvil, videollamadas y una experiencia de navegación web más fluida.

La RAN de 3G o UTRAN (*UMTS Terrestrial RAN*) está compuesta por dos elementos, el NodeB y el controlador de la red de radio (*Radio Network Controller*, RNC). Además, se definen las interfaces Uu entre el UE y el NodeB, la luB entre el NodeB y el RNC, y las interfaces luCS y luPS entre el RNC y el core. Por último, se cuenta con la interfaz lur que interconecta los RNC. En la figura 2.2 se muestra el diagrama de red de 3G.

En el NodeB se implementan las funciones de capa física y parte de la función de capa 2, conocida como control de acceso al medio (*Medium Access Control*, MAC). En el RNC, la capa 2 tiene el resto del control de acceso al medio, y el control del enlace de radio (*Radio Link Control*, RLC), mientras que en capa 3 está la función de gestión de recursos de radio (*Radio Resources Management*, RRM).

Cada RNC administra un grupo de NodeB, por administrar se entiende la coordinación y control de todos los aspectos operativos de estos NodeB, incluyendo la asignación y gestión de los recursos de radio, la supervisión de la transmisión de datos, la ejecución de handovers, el control de potencia, la implementación de medidas de seguridad como el cifrado de datos, y la optimización de la calidad de servicio (QoS). Además, el RNC se encarga de la gestión de la movilidad de los usuarios dentro de la red y asegura una comunicación eficiente entre los UE y la red de núcleo a través de las interfaces correspondientes.

Los NodeB, por otro lado, se encargan de la transmisión y recepción de señales

[1] https://www.3gpp.org/

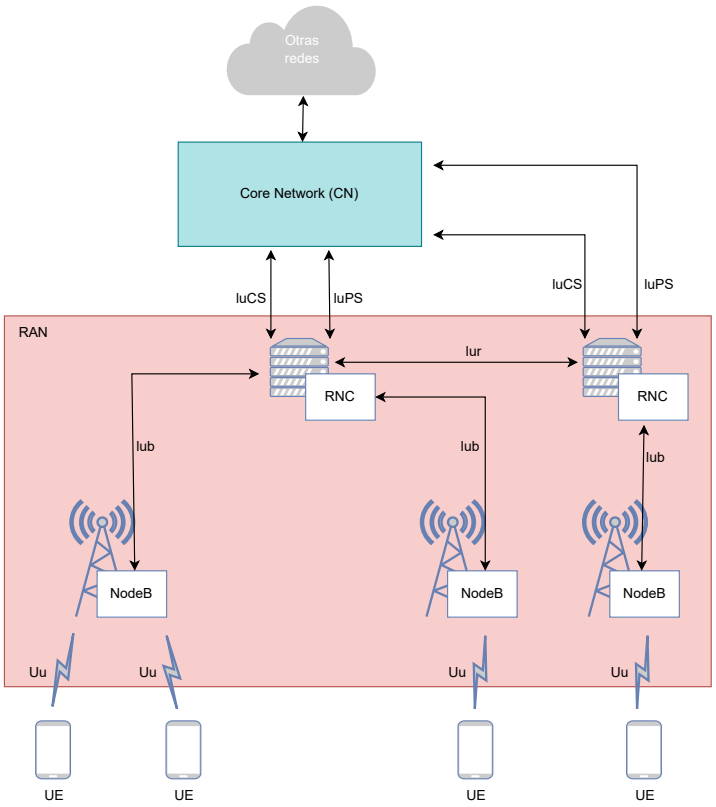

Figura 2.2: Arquitectura 3G con sus tres componentes: CN, RAN y UEs, donde se detallan los elementos que conforman la RAN y las interfaces.

de radio desde y hacia los UE, de la modulación y demodulación, entre otras funciones de radio, así como de reportar medidas al RNC.

En base a la descripción anterior, se puede decir que el RNC es más *inteligente* que el NodeB, por lo tanto, tiene mayores requerimientos de hardware (procesamiento, memoria, etc.).

Respecto a las interfaces, la UTRAN se compone de dos interfaces internas, la lub que es la interfaz lógica que conecta el NodeB con el RNC, y la lur que es la interfaz lógica que conecta diferentes RNC entre sí. Estas interfaces definen su propio protocolo de aplicación (NBAP para la lub y RNSAP para la lur). A nivel de transporte, ambas trabajan sobre ATM (*Asynchronous Transfer Mode*).

También se definen otras dos interfaces que comunican la UTRAN con el exterior. La Uu es la interfaz de radio entre el NodeB y los UE mediante WCDMA (*Wideband Code Division Multiple Access*). La interfaz lu (también se basa en la red de transporte ATM) comunica el RNC con el core de la red, se puede dividir lógicamente en dos, luCS que conecta el RNC con la red de circuitos y la luPS que comunica el RNC con la red de paquetes.

4G

A principios de 2010 llega al mercado la tecnología móvil conocida como 4G o LTE (*Long Term Evolution*). LTE trae consigo mejoras significativas respecto a 3G, teniendo mejor *throughput*, latencia, eficiencia espectral, etc., además de simplificar el *core* de la red gracias a que está pensado como una red de paquetes.

La RAN de LTE, también conocida como E-UTRAN (*Evolved UTRAN*) se compone de un elemento análogo al NodeB de 3G, ahora llamado eNodeB o eNB, y no hay elemento equivalente al RNC, 4G concentra la *inteligencia* de la RAN en el eNB. En la figura 2.3 se representa la RAN de 4G, con el eNB y las interfaces que lo conectan con los UE y con el *core*.

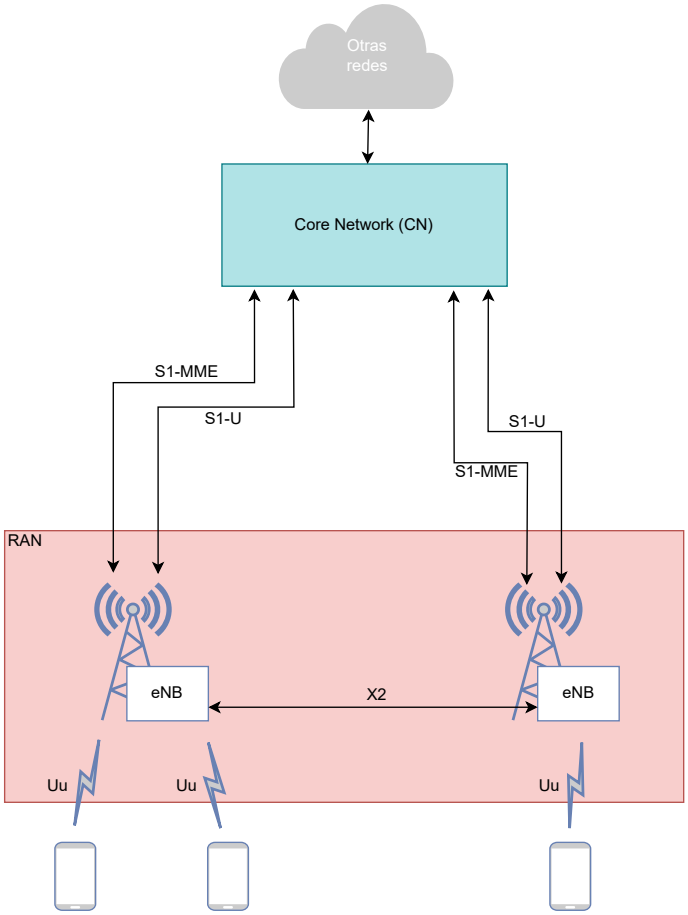

Figura 2.3: Arquitectura 4G con el CN, la RAN, la UE y las interfaces que permiten la comunicación entre los componentes.

En el eNB se concentra todo el stack de protocolos involucrado en la comunicación por radio, tanto para el plano de usuario (en naranja en la figura 2.4)

como para el plano de control (en rojo en la figura 2.4). Por plano de usuario (*user plane*) se entiende el conjunto de procesos y funciones que gestionan y transportan los datos del usuario a través de la red; estos datos de usuario pueden ser contenido web, video, voz, etc. Por plano de control (*control plane*) se entiende el conjunto de funciones y procesos responsables de establecer, gestionar y mantener las conexiones de red a través de las cuales circularán los datos de los usuarios.

Saber el funcionamiento del eNB es sinónimo de conocer las funciones que realiza cada protocolo del stack; por este motivo, a continuación se describirán las principales características y funciones de los protocolos involucrados en la comunicación de radio (**PHY**, **MAC**, **RLC**, **PDCP** y **RRC**) [44] [19]. No se profundizará en los protocolos que conectan la RAN con el *core* de la red, ya que no es el foco de este trabajo.

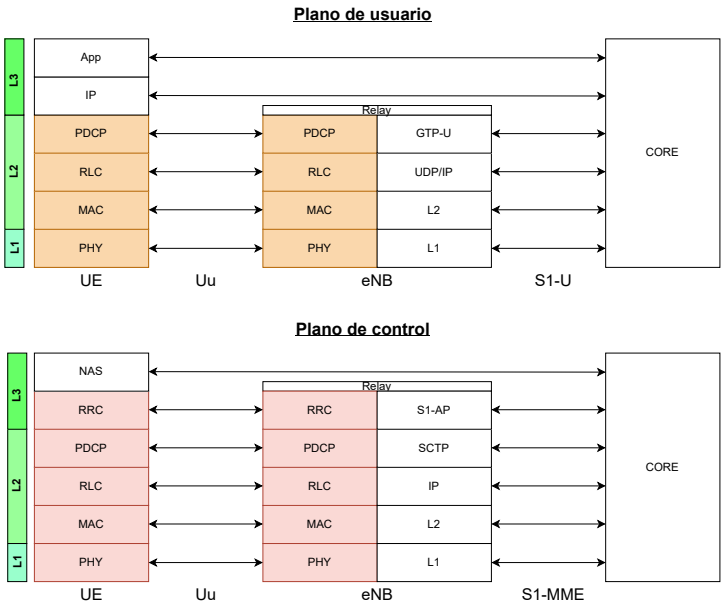

Figura 2.4: Stack de protocolos para 4G tanto para el plano de control como para el plano de usuario. En naranja y rojo los protocolos que efectivamente trabajan a nivel de la RAN.

- **PHY**: la capa física se encarga de la emisión y recepción de ondas electromagnéticas desde y hacia el aire. Entre sus funciones se encuentran la modulación, la demodulación, la codificación adaptativa, el control de potencia, la gestión de MIMO (*Multiple Input Multiple Output*) y las mediciones del canal, entre otras.
- **MAC**: es un protocolo de capa 2 que mapea los canales lógicos que tiene contra el protocolo superior (RLC) y por la cual recibe datos con canales de transporte que conectan la MAC con PHY. Además se encarga del *scheduling*, multiplexación y demultiplexación, medir el volumen de tráfico y de la detección y corrección de errores con HARQ.

- **RLC**: el Control del Enlace de Radio (*Radio Link Control*) es un protocolo de capa 2 que asegura el envío y recepción de paquetes PDCP entre el eNB y los UE. Se encarga de la segmentación y concatenación de paquetes de la capa superior para ajustarse al tamaño requerido por la capa inferior, también realiza el proceso inverso, o sea la reconstrucción de paquetes. Otras de sus funciones destacables son el ordenamiento de paquetes (excepto durante el *handover*), eliminación de duplicados y corrección de errores mediante ARQ (*Automatic Repeat Request*). RLC tiene tres modos de funcionamiento: 1) modo transparente, donde los paquetes pasan a través de la capa RLC sin mayor intervención, se utiliza para algunas señales de control, 2) modo no *acknowledged*, donde se aplican las funciones de segmentación, ordenamiento, detección de duplicados pero sin control de errores, es utilizado en tráfico sensible al retraso como VoIP, y 3) modo *acknowledged*, es igual al anterior pero agrega el control de errores, este modo es útil para tráfico sensible a los errores pero tolerante al *delay*, por ejemplo la navegación web.

- **PDCP**: el protocolo PDCP (capa 2) (*Packet Data Convergence Protocol*) es la que brinda el punto de acceso a los servicios de radio tanto al plano de usuario (por ejemplo a IP) como al plano de control (por ejemplo a RLC). Tiene tres funciones principales que son la compresión y descompresión de encabezados, el encriptado y desencriptado de datos, y la reordenación y secuenciación de PDUs durante el handover.

- **RRC**: este protocolo de capa 3 solo participa en el plano de control, y como su nombre indica, *Radio Resource Control*, se encarga de controlar y administrar los recursos de radio entre el eNB y los UE. Entre sus funciones se encuentra la difusión de parámetros del sistema, *paging*, señalización del *handover*, gestión de los SRB (*Signaling Radio Bearers*) y DRB (*Data Radio Bearers*), señalización para el establecimiento y liberación de la conexión, entre otras.

Otra idea que introduce LTE, aunque de forma primitiva, y que será clave en 5G y en O-RAN, es la desagregación. Esto es la división de una unidad funcional en unidades más simples que interactúan entre sí. Por ejemplo, como se ve en la figura 2.5 el eNB de LTE se divide en dos unidades/equipos/funciones, que son la BBU (*BaseBand Unit*) y la RU (*Radio Unit*) conectados entre si mediante la interfaz de *FrontHaul* (FH).

5G

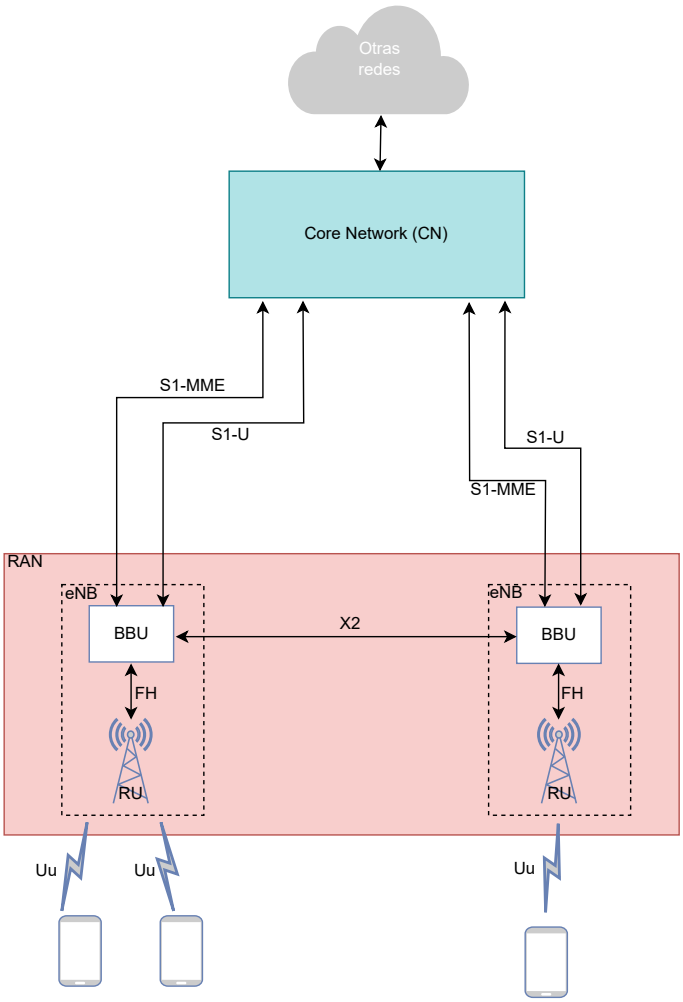

Figura 2.5: Arquitectura 4G con desagregación del eNB, separando el eNB en la BBU y RU.

5G NR (*New Radio*) es el último estándar de telefonía móvil de la 3GPP desplegado comercialmente, además se puede decir que es el estándar base de O-RAN. Mientras 4G era desplegado por las operadoras de todo el mundo se comenzó a trabajar en 5G, que en cuyas primeras definiciones se establecieron metas de velocidades (hasta 20Gbps) y latencia (menores a 1 ms)[2]. En 2018 la 3GPP publicó la primera versión del estándar de 5G conocido como NR *New Radio* (Rel. 15) y un año después comenzaron los primeros despliegues comerciales.

Respecto a la arquitectura de la red, si se ve en forma genérica como en la figura 2.6 solo difiere en el nombre de los componentes e interfaces con 4G, el eNB pasa a

[2]Especificaciones de la ITU IMT-2020

llamarse gNB, la interfaz X2 ahora es Xn, y las interfaces que conectan la interfaz con el núcleo son N2 y N3 en vez de S1-MME y S1-U. En la estructura interna del núcleo de la red también hay cambios significativos respecto a 4G aunque no se profundizará en ello en este trabajo.

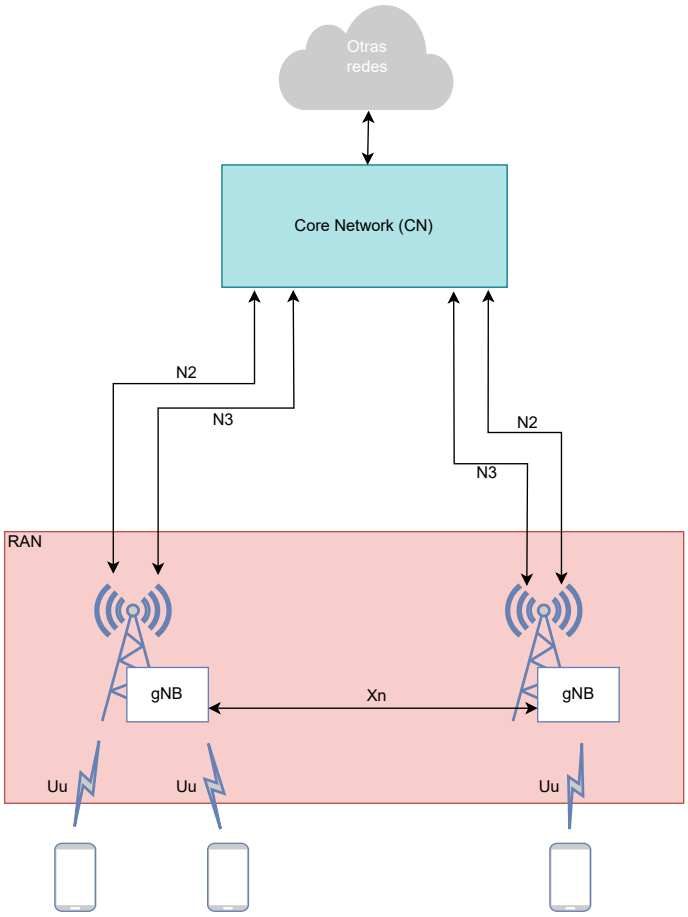

Figura 2.6: Arquitectura 5G donde se muestra la interconexión entre los tres componentes principales, CN, RAN y UE.

En cuanto a la experiencia de usuario las mejoras más significativas claramente son el mayor ancho de banda y la menor latencia, lo que no sólo mejora el uso que ya se venia haciendo de la red móvil sino que permite incorporar otros dispositivos y sistemas. También es un pilar fundamental de 5G las comunicaciones del tipo mMTC (*massive machine type communications*), esto es una enorme cantidad de dispositivos y sensores con bajos requisitos de *throughput* pero que necesitan una conectividad constante y confiable [20] [36].

Otras características técnicas destacadas de 5G (que gracias a ellas se llega a los niveles de *throughput* y latencia mencionados) son el uso de MIMO masivo, la

posibilidad de tener aplicaciones que requieran comunicaciones ultra confiables de baja latencia (URLLC), la capacidad de trabajar en varios rangos de frecuencia (incluso con ondas milimétricas), menor consumo, *beamforming*, mayor densidad de radio bases, mejora en los sistemas de *slicing*, mayor desagregación, utilización de diferentes *splits* y posibilidad de tener los sistemas distribuidos. Sobre los tres últimos puntos, que están fuertemente relacionados se profundizará a continuación ya que son relevantes en O-RAN.

Comenzando por la desagregación, 5G divide el gNB en tres elementos, la unidad central (*Central Unit*, CU), la unidad distribuida (*Distributed Unit*, DU) y la unidad de radio (*Radio Unit*, RU). Al tener dichos componentes separados es indispensable definir interfaces de comunicación entre ellos, en este caso la intefaz F1 entre el CU y el DU, y la interfaz de Fronthaul entre el DU y el RU. Todo lo anterior se ve representado en la figura 2.7. Existe un nivel más de desagregación que consiste en dividir el CU en dos partes, una que se encarga del plano de control y otra del plano de usuario, comunicándose entre sí mediante la interfaz E1. Otro aspecto importante es la *cardinalidad* de las conexiones, por definición un CU se puede conectar a uno o varios DU, pero un DU solo se conecta a un CU, algo similar sucede entre el DU y el RU, un DU puede controlar varios RU, pero un RU solo es controlado por un DU.

Como se vio, antes de 5G, la arquitectura clásica de la RAN era tener un componente de control y un componente de radio que en general se ubicaban en el sitio de la radio base, por ejemplo en 4G, la BBU en la base de la antena y la RU en la parte superior de la misma. Gracias al mayor nivel de desagregación y la forma en la que lo hace, 5G permite distribuir los sistemas, esto significa que se puede tener CU, DU y RU todo junto en la radio base, o tener el CU en un sitio central conectado a varios DU en sitios secundarios que a su vez se conectan con varias RU ubicadas en las radio bases. Es importante ver que todas las combinaciones son posibles, cada una con sus ventajas y desventajas, pero la elección queda del lado del operador que elegirá la mas conveniente según la situación. Otra ventaja de la desagregación es que la misma facilita la virtualización de las funciones de red, este aspecto es fundamental en O-RAN.

Hasta el momento no se ha mencionado que funciones cumple el CU, DU y RU, esto se debe a que no se pueden definir las funciones de estos elementos sin hablar de los *puntos de split*. En términos generales, estos *puntos de split* [35] [14] [55] [26] indican qué funciones del stack de protocolos se ejecutan en cada componente, y dependiendo de que partes del stack se implemente es que función cumple el CU, DU y RU. En 5G la 3GPP define que *puntos de split* son posibles, para ello divide algunos de los protocolos en dos, la parte *alta* y *baja* del mismo, por ejemplo en *High-MAC* se implementan algunas subfunciones del protocolo MAC y en *Low-MAC* otras. En la figura 2.8 se ven los ocho posibles splits de 5G definidos por la 3GPP.

La elección del split depende de varios factores pero de ellos se pueden destacar:

- **Requisitos de QoS**: algunos split son mejores en temas de latencia, otros en temas de *throughput*, etc. Un split bajo permite situar físicamente las funciones más sensibles al tiempo más cerca de la antena, lo que es crítico

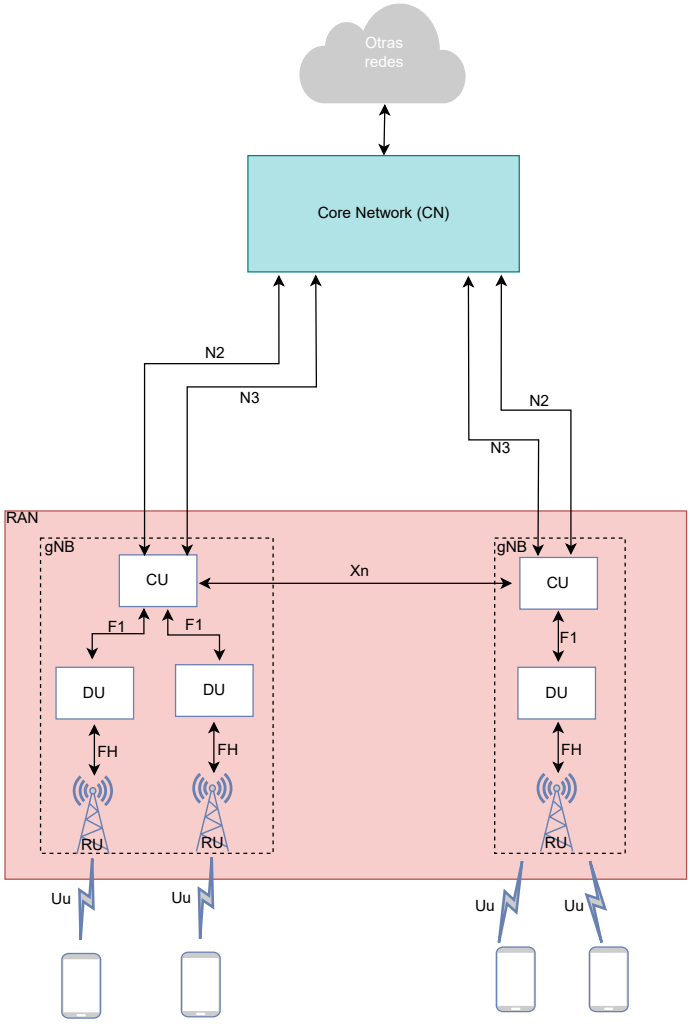

Figura 2.7: Arquitectura de la RAN 5G con desagregación en el gNB, este se divide en el CU, DU y RU.

para servicios de ultra baja latencia (URLLC). En cambio, splits más altos permiten una mayor eficiencia en coordinación y pueden beneficiar servicios de alta capacidad. Por ejemplo según [33] la latencia y el jitter del enlace de fronthaul pueden convertirse en el factor limitante cuando se eligen splits con mayor centralización de funciones.

- **Densidad vs cobertura**: en despliegues densos donde se requieren altas capacidades y coordinación entre celdas optar por un split que centralice más funciones (más alto) puede permitir optimizar el aprovechamiento del espectro. En entornos de mayor cobertura donde el enlace de transporte es

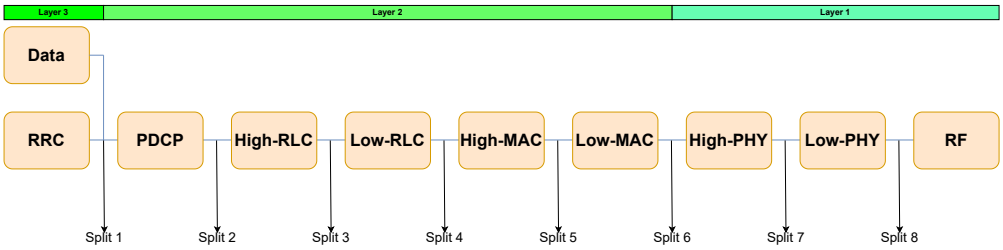

Figura 2.8: Puntos de split definidos por la 3GPP

más débil o donde la latencia del enlace puede ser mayor un split bajo puede resultar beneficioso.

- **Recursos de hardware disponibles**: cuántas mas funciones de red se implementen en un mismo nodo, mas recursos de cómputo consumirá.
- **Enlaces**: splits altos requieren enlaces de mayor capacidad que los splits bajos, sobre todo en el *fronthaul*.

6G

Aunque aun está en la etapa de investigación y desarrollo la ITU ya ha publicado el marco de referencia IMT-2030 para definir los requisitos, estándares y directrices de las tecnologías móviles de sexta generación (6G) que se espera sean implementadas hacia el año 2030.

La tecnología 6G promete una transformación profunda en las comunicaciones inalámbricas, extendiendo los límites de velocidad, latencia, cobertura y capacidades de inteligencia artificial en comparación con 5G. Con velocidades potenciales de hasta 1 Tbps y una latencia extremadamente baja de hasta 1 microsegundo, el objetivo de 6G es permitir aplicaciones avanzadas que requieren altísimas tasas de transferencia de datos y una respuesta casi instantánea. Este avance en la velocidad y latencia también conlleva un cambio en la arquitectura de las redes, donde el uso de frecuencias de terahercios (THz), que abarcan entre 100 GHz y 3 THz, se prevé como crucial para obtener estas capacidades sin precedentes.

El 6G está diseñado para proporcionar una cobertura verdaderamente global, combinando redes terrestres y satelitales para garantizar que tanto las zonas urbanas como las áreas remotas y submarinas puedan acceder a la red. Este sistema incluirá satélites de baja órbita (LEO), estaciones en el mar y sensores submarinos conectados a estaciones terrestres, ampliando la cobertura a áreas que las redes móviles anteriores no alcanzaban. Esta extensión de la cobertura es esencial para aplicaciones de “Internet de Todo” (IoE), que permitirán la conexión de millones de dispositivos y sensores en ubicaciones inaccesibles para 5G.

En cuanto a la integración de inteligencia artificial, 6G adoptará una arquitectura “nativa de IA” donde los algoritmos de inteligencia artificial y aprendizaje automático no solo optimizarán la administración de la red en tiempo real sino que también serán capaces de aprender de patrones de uso, predecir demandas

y auto optimizar su rendimiento. Esta arquitectura incluye gemelos digitales que replicarán la red en un entorno virtual para realizar simulaciones y pruebas de escenarios complejos sin impactar en la red física. Estos gemelos digitales permitirán también la generación de datos sintéticos para el entrenamiento de modelos de IA, manteniendo la red en niveles óptimos de seguridad y eficiencia energética.

Además, 6G introducirá una arquitectura sin celdas ("cell-less architecture"), eliminando las fronteras de las celdas tradicionales y permitiendo una conectividad continua sin interrupciones. Esto es especialmente útil para dispositivos en movimiento, como los vehículos autónomos, que podrán conectarse a la red sin interrupciones de cambio de celda. Esta arquitectura, junto con una gestión de recursos más inteligente y distribuida, soportará mejor el uso intensivo de recursos en aplicaciones como la realidad extendida (XR), que incluye realidad aumentada y virtual, y permitirá experiencias de interacción más inmersivas y de alta resolución sin interrupciones.

Las aplicaciones de 6G abarcan desde ciudades inteligentes y salud avanzada hasta vehículos autónomos y comunicación holográfica. Por ejemplo, en el ámbito de la salud, 6G facilitará cirugías remotas con un control en tiempo real, permitiendo que especialistas en cualquier parte del mundo operen a pacientes de forma precisa. En ciudades inteligentes, la infraestructura conectada a 6G permitirá la gestión en tiempo real del tráfico, la seguridad y el uso de energía, mejorando significativamente la calidad de vida. Además, 6G habilitará un "Internet Táctil", permitiendo la transmisión de sensaciones táctiles junto con audio y video, lo cual es clave para aplicaciones en robótica avanzada y telepresencia [49] [51] [12] [30].

2.2. Open RAN Alliance

Como se mencionó en el capitulo 1, históricamente las operadoras de telefonía móvil debían adquirir el hardware y software necesario para desplegar su red a un selecto grupo de fabricantes, siendo este hardware y software cajas negras para la operadora. Esto tiene varias desventajas (poca o nula interoperabilidad, poco margen de maniobra para requerimientos particulares, mayores costos, menor innovación, etc). Por estos motivos es que desde el inicio de la telefonía celular se buscó ir en la dirección de la estandarización, un gran avance fue la creación de la 3GPP que estandarizó las diferentes tecnologías celulares, pero esto no solucionó el problema mencionado ya que internamente los equipos tenían interfaces o procesos propietarios. Con el surgimiento y evolución de la virtualización se dio otro paso importante para lograr el objetivo, incluso con el pasar de los años se logro cierto nivel de apertura en el *core* de la red, pero la RAN seguía siendo fuertemente propietaria.

Fue así que las grandes operadoras comenzaron a agruparse para buscar una solución conveniente (para ellas). De los grupos que surgieron hay dos que destacan, la **C-RAN** China y la **XRAN** en Europa, Estados Unidos y Japón. Rápidamente el trabajo de estos grupos fue migrando de la RAN abierta a la RAN abierta, inteligente y virtualizada.

Al tener objetivos e intereses similares, ambos grupos decidieron fusionarse en

febrero de 2018 para formar la **O-RAN Alliance**, siendo sus miembros fundadores AT&T, China Mobile, NTT Docomo, Orange y T-Mobile [2] [3] [4].

Actualmente la alianza tiene mas de 300 miembros, desde otras grandes operadoras como Vodafone o Verizon, empresas como Intel, nVidia, Microsoft, Apple y Google, organismos varios como el NIST, el Centro Nacional de Ciber Seguridad y la NSA, Universidades de varios países y hasta fabricantes de equipamiento para la RAN como Nokia y Siemens[3].

Legalmente funciona bajo la figura de *asociación* dentro de la jurisdicción alemana. Su máximo órgano es la Asamblea General de Miembros que se reúne anualmente o excepcionalmente. Esta asamblea elige, cada dos años, hasta 10 miembros de la Junta Directiva (que de por si ya tiene 5 miembros fijos de los organismos fundadores). La Junta Directiva es quien gestiona la alianza. Apoyando a la Junta Directiva está el Comité Ejecutivo, este órgano se encarga de la agenda, presentación de propuestas, recomendaciones, etc. Además es el nexo entre el Comité Técnico y la Junta Directiva. El Comité Técnico decide y orienta sobre los aspectos técnicos de O-RAN, es el encargado de aprobar las especificaciones. El comité técnico esta compuesto por miembros y participantes, y se divide en los Grupos de Trabajo y los Grupos de Enfoque. Finalmente se tiene la Oficina de la O-RAN Alliance que da soporte administrativos a los órganos mencionados.

Existen dos formas de participar en la alianza, siendo miembros o participantes. Los miembros solo pueden ser operadoras de telefonía móvil. Los participantes por otro lado puede ser cualquier persona u organismo. Hay dos categorías de participantes, contribuyentes y académicos. Aunque los organismos de decisión están conformados por miembros, los participantes pueden participar en el desarrollo de especificaciones y en todas las etapas de los proyectos e iniciativas de la alianza.

Porque es de particular interés para este trabajo, a continuación se nombrará y dará una pequeña descripción de los Grupos de Trabajo (*Working Group*, WG) y de los grupos de enfoque (*Focus Group*, FG)[4]. Varios de los nombres de estos grupos de trabajo responden a componentes de la arquitectura que se verán en detalle en el siguiente capítulo.

Grupos de Trabajo

- **WG1: Use Cases and Overall Architecture Work Group**: se encarga de la arquitectura de O-RAN y los casos de uso
- **WG2: The Non-Real-Time RAN Intelligent Controller and A1 Interface Work Group**: su tarea es definir el Non-RT RIC para apoyar la gestión inteligente de recursos de radio (no en tiempo real).
- **WG3: The Near-Real-Time RIC and E2 Interface Work Group**: este grupo trabaja en el Near-RT RIC, este elemento permite el control y optimización casi en tiempo real de los recursos de la RAN. Además define la interfaz con la cual va a interactuar (interfaz E2).

[3]Lista completa de miembros: https://www.o-ran.org/membership
[4]https://www.o-ran.org/about

- **WG4: The Open Fronthaul Interfaces Work Group**: El objetivo de este grupo de trabajo es proporcionar interfaces de fronthaul verdaderamente abiertas e interoperables.
- **WG5: The Open F1/W1/E1/X2/Xn Interface Work Group**: se encarga de proporcionar especificaciones *multi-vendor* y proponer mejoras para las interfaces de la 3GPP F1/W1/E1/X2/Xn.
- **WG6: The Cloudification and Orchestration Work Group**: busca desacoplar el software de la RAN del hardware con el objetivo de poder utilizar software que implemente las diferentes funciones en hardware genérico.
- **WG7: The White-box Hardware Work Group**: El objetivo de este Grupo de Trabajo es especificar y liberar un diseño de referencia completo para fomentar una plataforma de software y hardware desacoplada.
- **WG8: Stack Reference Design Work Group**: su tarea es desarrollar la arquitectura de software, el diseño y el plan de lanzamiento para la Unidad Central O-RAN (O-CU) y la Unidad Distribuida O-RAN (O-DU) basado en las especificaciones de O-RAN y 3GPP para el stack de protocolos NR
- **WG9: Open X-haul Transport Work Group**: este grupo de trabajo se centra en la red de transporte (equipos, medios físicos, protocolos de control y gestión, etc)
- **WG10: OAM Work Group**: es responsable de los requisitos de OAM (*Operation, Administration and Management*) y de la interfaz O1.
- **WG11: Security Work Group**: se centra en los aspectos de segurar de la RAN.

Grupos de Enfoque

- **SDFG: Standard Development Focus Group**: elabora las estrategias de estandarización de la O-RAN Alliance ademas de coordinar con otros organismos que desarrollan estándares.
- **IEFG: Industry Engagement Focus Group**: busca promover y acelerar la adopción de O-RAN por parte de la industria.
- **OSFG: Open Source Focus Group**: gestiona la comunidad de desarrollo de software de código abierto de la O-RAN Alliance y coordina con otras comunidades de desarrollo libre.
- **TIFG: Testing and Integration Focus Group**: se encarga de probar e integrar las especificaciones de los diferentes grupos de trabajo verificando su funcionamiento e interoperabilidad.
- **SuFG: Sustainability Focus Group**: su tarea es optimizar el consumo de energía, reducir el impacto ambiental y crear redes móviles más eficientes en términos de energía y respetuosas con el medio ambiente.

- **nGRG: next Generation Research Group**: se centra en la investigación de principios de RAN abiertos e inteligentes en 6G y futuros estándares de red.

Aparte de los grupos de trabajo y de enfoque, el otro jugador clave en la O-RAN Alliance es la O-RAN Software Community (OSC). La OSC surge de un acuerdo de colaboración entre la O-RAN Alliance y la Linux Foundation[5] para apoyar la creación de software para la RAN. La OSC se alinea con la arquitectura y especificaciones de O-RAN para crear o utilizar código ya existente de otros proyectos con el fin de lograr una solución que pueda ser utilizada en la industria[6][7]. Dicho de otra forma la comunidad busca encontrar una solución de software libre para todas las funciones de red de O-RAN.

El trabajo de la OSC se divide en proyectos, donde cada uno de ellos busca implementar una función de red o elemento de O-RAN, a continuación se da una breve descripción de los más significativos [34] [8].

- **RICAPP**: incluye ejemplos de xApps y otras aplicaciones que pueden utilizarse para pruebas, integración y demostraciones.
- **RIC**: desarrolla la plataforma para el Near-RT RIC para la ejecución de xApps con soporte limitado para las interfaces O1, A1 y E2.
- **NONRTRIC**: colabora con los proyectos OAM, SMO, AI/ML y SIM para ofrecer las funcionalidades del SMO y de las interfaces O1 y O2. Tambien desarrolla las interfaces A1 y R1 asi como las capacidades para soportar rApps y el Non-RT RIC.
- **OAM**: proporciona una implementación de referencia según los documentos de O-RAN OAM (WG1).
- **OCU**: tiene como objetivo desarrollar una versión inicial de software centrada en la integración del Near-RT RIC y el OCU mediante la interfaz E2. No se desarrolla propiamente el OCU, se utiliza una imagen desarrollada por Radisys[8].
- **ODUHIGH**: implementación de los protocolos de capa dos en el ODU.
- **ODULOW**: implementación de los protocolos de capa uno en el ODU.
- **INF**: este proyecto provee referencias para realizar una implementación del tipo open-source de la infraestructura "Edge-Cloud" cumpliendo con las especificaciones de O-RAN.
- **SIM**: provee simuladores de la interfaces para probar las funciones de red.

[5]https://www.linuxfoundation.org/
[6]https://www.o-ran.org/software
[7]https://o-ran-sc.org/about/
[8]https://www.radisys.com/

- **INT**: testea el funcionamiento del resto de los proyectos, tanto en solitario como en conjunto.
- **SMO**: implementa la interfaz O1 y la funciones de nivel superior del SMO.
- **AI/ML Framework**: se desarrolla un framework para llevar a cabo todas las tareas relacionadas con AI/ML, entre ellas el entrenamiento, despliegue, inferencia y administración de modelos.

2.3. ¿Qué es Open RAN?

Antes de proponer una definición propia es importante ver qué dicen los expertos al respecto, a continuación se presentan los comentarios realizados en cuatro fuentes, dos artículos y dos libros sobre el tema.

Polese et al. [31] definen a la Open RAN como "el nuevo paradigma para la RAN del futuro" y luego agrega "Open RAN se basa en componentes desagregados, virtualizados y basados en software, conectados a través de interfaces abiertas y bien definidas, e interoperables entre diferentes proveedores. La desagregación y virtualización permiten implementaciones flexibles, basadas en principios nativos de la nube. Esto aumenta la resiliencia y reconfigurabilidad de la RAN. Las interfaces abiertas e interoperables también permiten a los operadores integrar diferentes proveedores de equipos, lo que abre el ecosistema RAN a actores más pequeños".

Por otro lado Marinova et al. [48] afirman que Open RAN "está revolucionando el espacio de las telecomunicaciones al introducir un marco basado en los conceptos de virtualización y apertura" y que "O-RAN fomenta componentes de RAN virtualizados y desagregados conectados a través de interfaces abiertas basadas en especificaciones de la Alianza O-RAN".

Wim Rouwet en su libro *Open Radio Access Network (O-RAN) Systems Architecture and Design* [45] dice que "el objetivo declarado de O-RAN es romper la naturaleza cerrada de las implementaciones actuales de la red de acceso por radio" para luego afirmar que "al abrir la implementación 3GPP, O-RAN pretende desacoplar las implementaciones de hardware y software, permitiendo que los proveedores (de hardware, software y sistemas) se enfoquen en proporcionar componentes en lugar de soluciones completas", también hace una analogía entre Open RAN y las redes definidas por software (SDN) diciendo que "el objetivo es seguir lo que ocurrió en las redes definidas por software y redes cableadas (piense en conmutadores, enrutadores y firewalls), que han pasado de implementaciones propietarias y centradas en hardware a una implementación centrada en software que se ejecuta en hardware de propósito general, Arm, RISC-V o x86."

Finalmente Wong et al. en el libro *Open RAN The Definitive Guide* [18] responde a la pregunta ¿Qué es Open RAN? diciendo que "Open RAN es el movimiento en las telecomunicaciones inalámbricas para desagregar hardware y software, abrir interfaces y reducir costos", luego agrega que "con más proveedores de hardware y software dentro del ecosistema de Open RAN, la competencia ayudará a la innovación acelerando el desarrollo de las redes inalámbricas"

Entonces, en base a todo lo recabado en este capítulo, el autor de esta tesis formula la siguiente definición de O-RAN:

Open RAN (O-RAN) es una tecnología/concepto innovador y revolucionario en el mundo de las telecomunicaciones que surge de los mismos operadores de red (unidos en la O-RAN Alliance) con el objetivo de abrir la red de acceso por radio sin *reinventar la rueda*, o sea aprovechando y aprendiendo del ecosistema existente de la 3GPP con 5G y 4G. Se basa en componentes de software virtualizados y desagregados conectados a través de interfaces de software abiertas y bien definidas con el fin de permitir la interoperabilidad. De esta forma se rompe el paradigma actual de soluciones monolíticas y todo en uno (cajas negras) que ofrecen los proveedores de equipos de telecomunicaciones pasando a una filosofía donde los componentes de hardware y software provienen de múltiples proveedores permitiendo una mayor flexibilidad, eficiencia y competitividad. Además busca agregarle *inteligencia* a la red con el objetivo de mejorar la gestión y administración.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 3

Arquitectura de O-RAN

En este capítulo se presentarán y describirán los diferentes módulos o componentes de O-RAN así como las interfaces que los interconectan. El eje central del capítulo será la figura 3.1, la cual muestra los elementos de la RAN y su interconexión a través de las distintas interfaces. También se muestra la cardinalidad de las conexiones (o sea si un determinado elemento se conecta con uno o varios de otro), los dominios temporales de los bucles de control (a la izquierda) y el dominio geográfico de los elementos (a la derecha). La principal fuente bibliográfica de este capítulo son las especificaciones de los grupos de trabajo de la O-RAN Alliance [1] apoyada con los libros [45] y [18] y los artículos [31], [48] y [56].

Si se comienza a recorrer la figura desde la parte superior, en violeta, se ve el CN (*core network*), de él salen las interfaces de la 3GPP (S1-U, S1-MME, N2 y N3) hacia el eNB y gNB según corresponda. Como ya se mencionó no se profundizará en estos aspectos.

[1]Imagen basada en el diagrama del libro [18] cap 3 pag 25

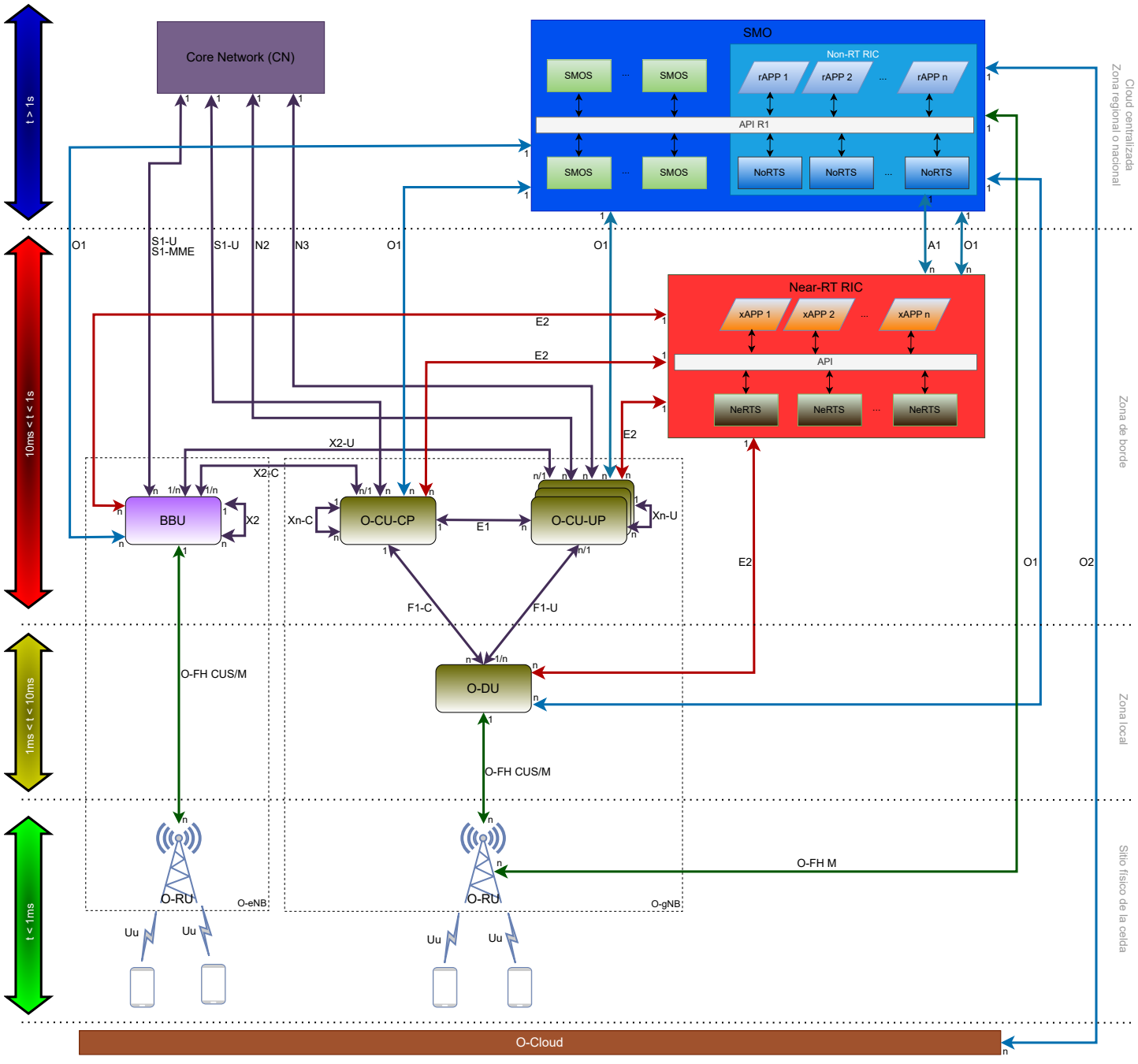

Figura 3.1: Arquitectura de O-RAN[1], detallando sus componentes (núcleo de la red, SMO, Non-RT RIC, Near-RT RIC, O-eNB, O-gNB y O-Cloud), enlaces, interfaces (E2, O1, O2, A1, O-FH y las 3GPP), zonas temporales (a la izquierda del diagrama) y zonas geográficas (a la derecha del diagrama).

En azul se representa el SMO (*Service Management and Orchestration*) que se compone de los SMOS (*SMO Services*) y del Non-RT RIC (*No Real Time Radio Inteligent Controller*) que a su vez esta formado por varios servicios y las rApps. La conexión interna entre los elementos se da gracias a la API R1. Del SMO salen (o llegan) las interfaces O1, O2 y opcionalmente la O-FH M, por otro lado del Non-RT RIC la interfaz que lo comunica directamente con el Near-RT RIC es la A1.

El Near-RT RIC (*Near Real Time Radio Inteligent Controller*), en rojo, se conforma de manera similar al SMO, de servicios y xApps. La comunicación entre ellos también es mediante una API. El Near-RT RIC interactúa con otros componentes a través de la interfaz E1. Por otro lado se conecta directamente con el SMO a través de la interfaz O1 y con el Non-RT RIC con la A1.

En el siguiente nivel se tiene el O-eNB y el O-gNB (análogos de O-RAN al eNB y gNB). Comenzando con el O-eNB, el mismo se ve desagregado en lo que sería la BBU y el O-RU (*Radio Unit*), conectados mediante las interfaces O-FH CUS/M. El O-eNB se vincula con el SMO utilizando la interfaz O1 y el Near-RT RIC con la interfaz E1. El resto de las interfaces que se ven en el diagrama son las propias de la 3GPP.

El O-gNB también se encuentra desagregado, en este caso en el O-CU-CP (*Central Unit - Control Plane*), O-CU-UP (*Central Unit - User Plane*) y O-DU (*Distributed Unit*). Al igual que en 5G la conexión entre estos elementos se da mediante interfaces de la 3GPP (E1, F1 y Xn). En conjunto el O-gNB se enlaza con el SMO utilizando la interfaz O1 y con el Near-RT RIC a través de la interfaz E2. De la misma forma que el O-eNB, el O-gNB se une al O-RU con la interfaces O-FH CUS/M. Como todos los componentes del O-eNB y O-gNB tienen un enlace con el Near-RT RIC utilizando la interfaz E2, es común ver en la literatura referirse a ellos como Nodos E2.

Finalmente en la parte inferior del diagrama, y como si estuviera por fuera (más adelante se verá el motivo) está el O-Cloud, este elemento es controlado directamente por el SMO mediante la interfaz O2.

A continuación, se llevará a cabo un análisis detallado de cada uno de los elementos e interfaces de O-RAN, describiendo en profundidad su función dentro del sistema, así como los mecanismos y procesos que implementan para garantizar el cumplimiento de sus objetivos y contribuir al funcionamiento eficiente de la arquitectura.

3.1. SMO

El SMO típicamente se despliega en una nube centralizada (regional o nacional) y es el elemento con mayor necesidad de recursos de la RAN (procesamiento, memoria, etc). En pocas palabras su función es administrar la RAN, ya sea directamente o a través de otros elementos. Las operaciones clave que el SMO debe soportar son:

- Proveer las funciones FCAPS (*Fault, Configuration, Accounting, Performan-*

ce, Security) a todas las funciones de red de la RAN. Las funciones FCAPS son un modelo de gestión de redes y servicios de telecomunicaciones que divide las actividades de administración en la gestión de fallos (detectar, registrar, resolver y documentar fallas), gestión de configuración (controlar y administrar la configuración de los dispositivos), gestión de contabilidad (conocer el uso de los recursos), gestión de desempeño (monitorizar y optimizar el desempeño de la red) y gestión de seguridad (gestión de usuarios, accesos, cifrado, amenazas, etc) [16].

- Implementar el Non-RT RIC
- Gestionar el O-Cloud

Sobre la cardinalidad del SMO, en general se tiene un SMO por red (u operador). Cada SMO controlará varios nodos E2 y Near-RT RICs.

3.1.1. SMOS

Las funciones recién descritas se implementan en el SMO mediante los SMO Services (SMOS). La estructura del SMO con sus SMOS trata de aproximarse lo más posible a una arquitectura basada en servicios. En la figura 3.2 se representa el SMO y algunos de los SMOS, a continuación se dará una breve explicación de cada uno.

- **RAN NF PM (*RAN Network Function Performance Management*)**: monitorea el rendimiento de los nodos E2 en tiempo real para que sus consumidores puedan detectar o anticipar problemas. La recopilación de datos se realiza través de las interfaces O1 y/o O-FH.
- **RAN NF FM (*RAN Network Function Fault Management*)**: recopila datos de fallos de las diferentes funciones de red (por ejemplo alarmas). También utiliza las interfaces O1 y/o O-FH.
- **RAN NF CM (*RAN Network Function Configuration Management*)**: proporciona mecanismos para que otros SMOS realicen cambios en la configuración de algún elemento de la RAN, sobre todo de los nodos E2.
- **TE&IV (*Topology and Inventory*)**: para que el SMO pueda aplicar un esquema OAM (Operación, Administración y Mantenimiento) correctamente es importante tener un inventario de los recursos disponibles y la forma en la que están conectados entre ellos. Este SMOS proporciona información actualizada sobre los recursos de la red y sus relaciones.
- **SME (*Service Management and Exposure*)**: se utiliza para descubrir, registrar y exponer servicios. Permite que diferentes componentes de la RAN como los RIC expongan sus capacidades a otros elementos. Es fundamental para facilitar la comunicación entre diferentes proveedores y consumidores de servicios, permite que las políticas y métricas de la red sean compartidas y applicadas de forma dinámica en toda la RAN.

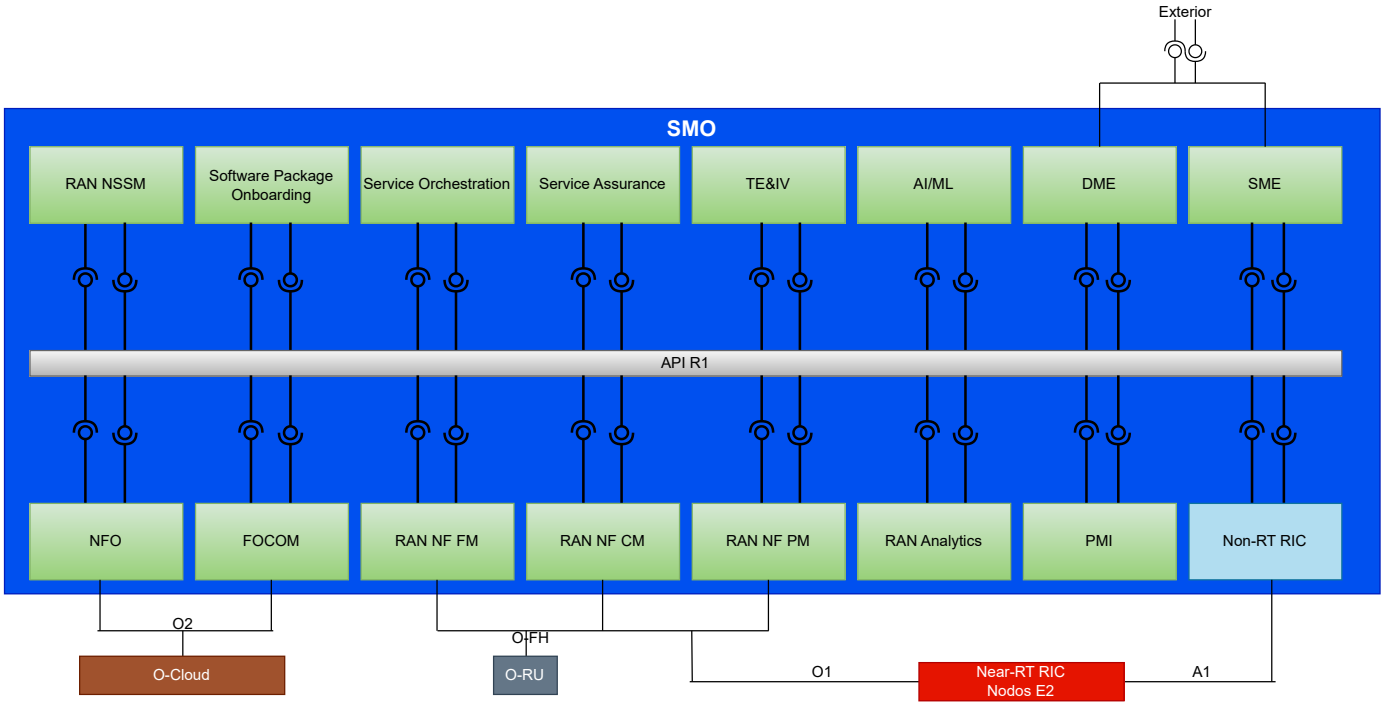

Figura 3.2: Arquitectura del SMO con sus SMOS. En la misma se aprecia la arquitectura productor/consumidor y como todos los SMOS se conectan a través de la interfaz R1, además se muestra a que SMOS llegan las interfaces externas.

- **DME (*Data Management and Exposure*)**: tiene como objetivo gestionar y exponer de manera eficiente los datos de la red. A través de este servicio, los operadores pueden acceder a datos clave sobre el rendimiento y el comportamiento de la red, y compartir esa información entre las distintas partes interesadas, incluyendo los controladores inteligentes y otros servicios de la infraestructura. El DME facilita la recopilación, almacenamiento y procesamiento de grandes volúmenes de datos generados por los componentes de la red. Estos datos pueden incluir métricas sobre el uso de recursos, el rendimiento de los enlaces de comunicación, y otros parámetros de calidad de servicio. Con esta información, los operadores pueden tomar decisiones informadas sobre la gestión de la red y la optimización de la experiencia del usuario, así como aplicar políticas de red adaptativas en tiempo real a través de interfaces.

- **NFO (*Network Function Orchestration*)**: es responsable de la automatización y coordinación de los recursos del O-Cloud utilizados para el despliegue de la O-RAN.

- **FOCOM (*Federated O-Cloud Orchestration and Management*)**: se encarga de gestionar y coordinar múltiples instancias de O-Cloud (que se encuentran distribuidas) para que en ellas se ejecuten las diferentes funciones

de red y trabajen de forma sincronizada y coordinada. Permite gestionar de forma centralizada y unificada los recursos distribuidos en las distintas O-Cloud.

- **RAN NSSM (*RAN Network Slice Subnet Management*)**: gestiona el *network slice*, cada slice se maneja como una instancia llamada NSSI (network subnet slice instance) y este SMOS se encarga de crearlas, activarlas, desactivarlas, obtener datos, etc.
- **RAN Analytics**: consume datos y servicios del SMO para procesarlos y analizarlos (según su lógica y algoritmos) para generar datos que permitan el análisis de la RAN como pueden ser predicciones, estadísticas o recomendaciones.
- **Service Orchestration** automatiza la gestión de la RAN. Su principal función es supervisar y coordinar de manera autónoma las operaciones necesarias para cumplir con los requisitos específicos de los servicios de red.
- **Service Assurance**: se encarga de mantener los niveles de garantía requeridos para los servicios Ent-to-end. Para esto recopila información de las NF involucradas y en caso que el nivel de garantía caiga por debajo de cierto umbral toma acciones correctivas.
- **Software Package Onboarding**: se encarga de la instalación, verificación de seguridad, validación y extracción de paquetes de software que implementen funciones de red, xApps, rApps, etc.
- **AI/ML**: maneja los modelos de entrenamiento disponibles y permite a otros SMOS utilizarlos.
- **PMI (*Policy Management and Information*)**: es donde están definidas las reglas, lineamientos o condiciones (políticas) que rigen el comportamiento y las acciones de todos los elementos de la O-RAN. Cumple el rol de punto de administración de políticas (definición del NIST[2]), o sea que tiene capacidades de registrar, anular, notificar políticas entre otras funciones.

3.2. Non-RT RIC

El controlador inteligente de radio no en tiempo real es uno de los nuevos elementos que trae O-RAN. Aunque funciona dentro del SMO como un conjunto de SMOS es común pensarlo como una entidad independiente. Su nombre *no en tiempo real* viene dado por la latencia o tiempo que le lleva un bucle cerrado de control, en este caso en ordenes de tiempo mayores a un segundo.

Los bucles cerrados de control (que también aplican a otros elementos) son procesos que constan de tres etapas ordenadas (recolección de datos, procesamiento y toma de decisiones) que están en un bucle constante. El tiempo que demora uno

[2]https://csrc.nist.gov/glossary/term/policy_administration_point

de estos ciclos determina el dominio temporal. En un ejemplo práctico el Non-RT RIC podría pedirle o recolectar datos del Near-RT RIC a través de la interfaz A1, procesarlos y aplicar alguna configuración determinada.

Internamente el Non-RT RIC tiene dos partes bien definidas, el Non-RT RIC Framework: funcionalidad que termina lógicamente la interfaz A1 y expone los servicios requeridos a las rApps a través de su interfaz R1; y el Non-RT RIC Applications (rApps): aplicaciones modulares que aprovechan la funcionalidad expuesta por el Non-RT RIC Framework para realizar la optimización de la RAN y otras funciones.

3.2.1. rApps

Las rApps son aplicaciones (generalmente desarrolladas por terceros) que se alojan en el Non-RT RIC con el objetivo de cumplir una función específica. Tienen a su disposición información de toda la red a través de la interfaz interna R1 que la comunica con el exterior mediante las interfaces O1, O2 y A1 así como con todas las SMOS del SMO. Al estar en el nivel más alto de la RAN en general aplican políticas y configuraciones de alto nivel que afectan toda la red. Es fundamental tener en cuenta el dominio temporal en el que trabajan, en ese aspecto están orientadas a funciones que requieren un análisis o procesamiento mas largo. Una de las funciones típicas que se programan en rApps son la optimización de la red a gran escala y el análisis predictivo, a menudo basadas en modelos de *machine learning* e inteligencia artificial.

3.3. Near-RT RIC

El Near-RT RIC es una función de red que se agrega a la RAN cuyo propósito es proveer de funciones de control y optimización casi en tiempo real (latencias de procesamiento de entre 10 ms y 1 s) mediante la recolección de datos y ejecución de acciones a través de la interfaz E2.

El SMO y el Non-RT RIC lo controlan mediante las interfaces O1 y A1 respectivamente. También tiene una interfaz (Y1) para que consumidores externos (la operadora) pueda extraer directamente información estadística o de comportamiento de la red para su análisis.En general se ubica en el borde de la red y puede administrar varios nodos E2, un operador puede tener varios Near-RT desplegados.

Internamente se reconocen dos partes, las xApps y la plataforma (*Near-RT RIC Platform*), a su vez la plataforma está compuesta por varias funciones predeterminadas. La comunicación interna entre las xApps y las funciones de la plataforma se realiza mediante una API. En la figura 3.3 se muestra un diagrama con la estructura interna del Near-RT RIC.

La función del Near-RT RIC es monitorear métricas y condiciones de la RAN en tiempos muy breves con el objetivo de realizar ajustes inmediatos de los parámetros de red (asignación de recursos de radio, potencia de transmisión, etc). Al detectar cambios en las condiciones de red, como congestión o degradación de la calidad, el

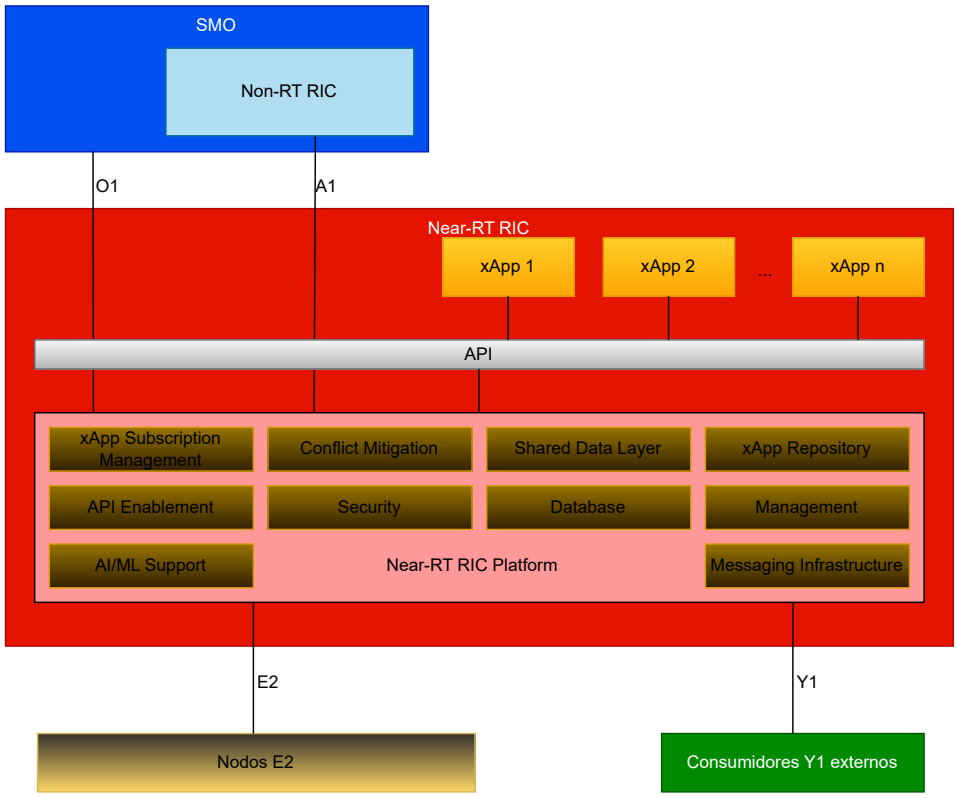

Figura 3.3: Arquitectura interna del Near-RT RIC donde se pueden ver las xApps y las funciones predeterminadas de la plataforma, interconectadas entre ellas mediante una API.

Near-RT RIC puede actuar rápidamente para redistribuir los recursos y optimizar la experiencia del usuario final.

El monitoreo y ajustes mencionados son realizados por aplicaciones de terceros (xApps) que se apoyan en la infraestructura del Near-RT RIC (plataforma) y la interfaz E2.

3.3.1. xApps

Las xApps son aplicaciones diseñadas para ejecutarse sobre el Near-RT RIC, brindando una capa de inteligencia y control en tiempo casi real sobre los recursos y parámetros de la RAN. Están destinadas a optimizar el rendimiento de la red de forma dinámica, utilizando datos detallados y realizando ajustes rápidos que permitan responder a condiciones variables, como la congestión de red, cambios en la demanda de usuarios, y otros eventos relevantes.

Las xApps son módulos independientes, lo que significa que pueden ser desarrolladas y proporcionadas por terceros. Esta modularidad fomenta la innovación y permite que proveedores o desarrolladores externos creen aplicaciones específicas que cumplan con diferentes objetivos de optimización de la red.

Las xApps interactúan con la plataforma Near-RT RIC a través de interfaces y APIs bien definidas. Estas interfaces permiten a las xApps acceder a datos de la red en tiempo real, recibir eventos específicos, y ejecutar acciones de control sobre la RAN. La comunicación se realiza principalmente a través de la interfaz E2 para interactuar con los nodos de la RAN, y las interfaces O1 y A1 para cumplir con políticas de optimización establecidas por el SMO y el Non-RT RIC.

3.3.2. Plataforma

Como ya se mencionó, la plataforma del Near-RT RIC brinda funciones de soporte para que las xApps puedan realizar su tarea, en la figura 3.3 se pueden ver las principales funciones de soporte que se encuentran en la plataforma. A continuación se describen dichas funciones:

- **xApp Subscription Management**: permite a las xApps y a la plataforma Near-RT RIC gestionar suscripciones a datos específicos que se comparten entre diferentes xApps. Esta función permite que una xApp se suscriba a ciertos datos generados por otra xApp, definiendo los criterios para recibir esos datos (como los eventos que desencadenarán el envío de información, el mecanismo de suscripción, etc).

- **Conflict Mitigation**: detecta y resuelve conflictos entre las solicitudes de múltiples xApps que intentan optimizar o modificar parámetros de la RAN. Debido a que cada xApp puede tener objetivos específicos que a veces resultan en acciones contradictorias, esta función es crucial para mantener la coherencia y efectividad en las optimizaciones de la red. Las especificaciones O-RAN resaltan tres clases diferentes de conflictos. 1) Conflictos directos: ocurren cuando dos o más xApps solicitan ajustes diferentes para el mismo parámetro de la RAN. Por ejemplo si una xApp quiere subir la potencia de transmisión y otra reducirla. 2) Conflictos indirectos: surgen cuando las acciones de una xApp afectan indirectamente el objetivo de otra xApp. Por ejemplo si una xApp realiza ajustes en la inclinación de una antena para mejorar la señal en un determinado sector pero otra xApp había disminuido la potencia para mejorar la interferencia inter-celda. 3) Conflictos implícitos: estos son los más difíciles de detectar, ya que las interacciones entre las xApps no son evidentes. Por ejemplo, optimizar un parámetro para mejorar el rendimiento de ciertos usuarios puede tener efectos secundarios no deseados en otro grupo. Para resolver estos conflictos hay tres enfoques principales, la coordinación previa a la acción (se aplican reglas o prioridades previo a la ejecución de la configuración), la verificación posterior (luego de ejecutar una acción se monitorean los efectos) y el aprendizaje automático (en algunos casos, Conflict Mitigation puede emplear enfoques de aprendizaje automático, como el aprendizaje por refuerzo, para predecir el impacto potencial de las acciones de las xApps y decidir cuál es la mejor opción para la red).

- **Shared Data Layer**: proporciona un sistema para la gestión y compartición de datos entre las xApps y la propia plataforma Near-RT RIC. Este sistema permite que las xApps compartan, almacenen y accedan a datos en tiempo real o casi real, optimizando la coordinación y el rendimiento en la red. Facilita la interoperabilidad y coordinación entre múltiples xApps, ya que actúa como un punto central de intercambio y actualización de datos, permitiendo que las xApps trabajen con información consistente. Además, mejora la escalabilidad y eficiencia de la RAN, al reducir la duplicación de datos y minimizar la latencia en el acceso a la información.

- **xApp Repository**: mantiene una lista actualizada de las xApps con el objetivo de saber cuáles pueden recibir determinadas políticas desde el Non-RT RIC, dicho de otra forma sabe que xApps son compatibles con las políticas definidas SMO y Non-RT RIC.

- **API Enablement**: esta funcionalidad brinda soporte para el registro, descubrimiento y consumo de las APIs dentro del alcance Near-RT RIC.

- **Security**: está diseñada para proteger la integridad y confidencialidad de los datos y la infraestructura de la red. Esta función busca evitar que xApps maliciosas o no autorizadas puedan acceder, modificar o exportar información sensible de la red o realizar acciones no autorizadas sobre la RAN.

- **Database**: es donde se almacena información relacionada a la RAN y los dispositivos de usuario. Las xApps tiene disponible estos datos a través del Shared Data Layer.

- **Management**: es capaz de detectar fallas, configurar parámetros de funcionamiento, llevar un log con toda la información relevante, hacer seguimiento del ciclo de trabajo, recolectar información de gestión entre otras tareas. En general el SMO es quien consume esta información para poder aplicar políticas y configuraciones sobre el Near-RT RIC.

- **AI/ML Support**: ofrece un soporte completo del flujo de trabajo de AI/ML (entrenamiento, inferencia, modelos, etc) para que las xApps implementen algoritmos de AI/ML. El uso de ninguna, parte o todas las funcionalidades de este módulo es opcional y dependerá del diseño de la xApp.

- **Messaging Infrastructure**: la infraestructura de mensajería interna conecta xApps, funciones del Near-RT RIC platform y terminaciones de interfaz entre sí. Las especificaciones no imponen ninguna tecnología específica pero enumeran los requisitos y funcionalidades que este subsistema debe proporcionar. La infraestructura de mensajería interna debe admitir el registro, descubrimiento y eliminación de endpoints y proporcionar APIs para enviar y recibir mensajes, ya sea a través de comunicaciones punto a punto o mecanismos de publicación/suscripción. También proporciona enrutamiento y robustez para evitar la pérdida de datos interna.

3.4. rApps y xApps: ejemplo

Habiendo visto ya los dos controladores de radio inteligentes (Non-RT RIC y Near-RT RIC) es fácil ver que una de las mayores ventajas del modelo de O-RAN y lo que da más valor agregado son las rApps y xApps. Resulta sumamente útil y novedoso que cada operador pueda programar su RAN para que actué de cierta forma personalizada, teniendo en cuenta su situación puntual. Otro tema novedoso e interesante es que el modelo de O-RAN permite que otros actores aparte de los grandes proveedores de hardware se involucren en el negocio de las *telcos*, por ejemplo una pequeña empresa de desarrollo de software puede programar xApps o rApps personalizadas para una operadora. Con el objetivo de explicar este punto a continuación se dará un ejemplo práctico de uso e interacción entre rApps y xApps con el objetivo de facilitar su comprensión.

Ejemplo

Se va a realizar un concierto en un estadio donde se espera que la cantidad de usuarios conectados a la red en la zona aumente drásticamente durante varias horas. Este incremento de usuarios puede provocar una congestión en las celdas alrededor del estadio, lo que afecta la calidad de servicio de los usuarios.

A nivel de las rApps, se tiene una que recopila y analiza datos históricos de las celdas cercanas al estadio (teniendo en cuenta que seguramente hayan habido otros eventos similares en el pasado), esta rApp debe buscar patrones de tráfico, ubicación de los usuarios, demanda de servicios específicos, etc.

Con todo ese análisis, la misma rApp u otra puede utilizar algoritmos de aprendizaje automático para predecir la demanda, calcular qué lugares estarán más sobrecargados, la necesidad de recursos, etc.

Finalmente otra rApp toma todo lo anterior y define políticas y ajustes necesarios para mitigar los efectos de la acumulación de usuarios. Estas políticas son enviadas al Near-RT RIC mediante la interfaz A1.

Es importante destacar que todo lo anterior se realizó antes del concierto, no durante el mismo (no en tiempo real).

Durante el evento el Near-RT RIC ejecuta xApps que aplican las políticas (y si es necesario las corrigen) de forma dinámica y casi en tiempo real respondiendo a los cambios instantáneos de la demanda de la red.

Por ejemplo una xApp debe estar controlando en tiempo real la congestión de las celdas, si una celda se acerca a su capacidad máxima, la xApp puede redistribuir automáticamente parte del tráfico hacia celdas adyacentes que tengan menos carga, o aumentar la potencia de transmisión en sectores específicos para optimizar la cobertura.

Otra xApp implementa políticas de priorización de calidad de servicio, priorizando el tráfico de servicios esenciales, como llamadas de emergencia o aplicaciones de salud, sobre otros servicios menos críticos, como el streaming de video de redes sociales.

Una xApps puede utilizar algoritmos de IA para analizar patrones de movi-

miento de los usuarios dentro del estadio. Esto les permite hacer ajustes proactivos en las celdas y sectores donde se espera un aumento en la concentración de usuarios (por ejemplo, en el área de salida después del evento).

Pensar en algo así con el esquema clásico es casi imposible ya que habría que pedirle al fabricante del equipamiento que realice un desarrollo puntual para un escenario puntual. Con O-RAN las opciones son infinitas, cada operador puede desarrollar él mismo o a través de un tercero las rApps o xApps que desee para satisfacer sus necesidades puntuales.

3.5. O-gNB

Como se mencionó en el capitulo 2, al momento de diseñar la solución para O-RAN se tomó como punto de partida la RAN de 5G. Esto es evidente cuando se mira la figura 3.1 y se enfoca en el O-gNB, claramente se ve que es igual al gNB de 5G compuesto por el O-CU (análogo al CU), el O-DU (análogo al DU) y el O-RU (análogo al RU). La diferencia entre las figuras 2.7 y 3.1 está en que el O-CU se descompone en dos entidades, una para el plano de control (O-CU-CP) y otra para el plano de usuario (O-CU-UP), aunque esto no es exclusivo de O-RAN.

Hablando de la conectividad de los elementos, el O-CU-CP y el O-CU-UP se conectan entre sí mediante la interfaz de la 3GPP E1, mientras que el O-CU con el O-DU a través de la interfaz F1 (también de la 3GPP). Otras interfaces definidas por la 3GPP que aparecen son la Xn que une a los O-CU entre sí y la X2 que conecta el O-CU con la BBU de 4G. El O-DU se enlaza con el O-RU utilizando la interfaz de *FrontHaul* O-FH CUS (*Control/User/Synchronization plane*) y O-FH M (*Management plane*).

Tanto el SMO como el Near-RT RIC pueden gestionar y controlar el O-CU y O-DU mediante la interfaz O1 para el SMO y E2 para el Near-RT RIC. El SMO también necesita comunicarse con el O-RU, O-RAN define dos formas de hacerla. Una híbrida, donde existe una conexión directa entre el SMO y O-RU mediante la interfaz O-FH M, y otra jerárquica donde no existe conectividad directa, si no que la comunicación se realiza a través del O-DU.

La ubicación geográfica de estos componentes va a depender del despliegue del que se esté hablando, pero como un O-CU controla varios O-DU, y un O-DU puede gestionar varios O-RU es normal que el O-CU este en el borde de la red, similar al Near-RT RIC, el O-DU ya en la zona local cerca de las celdas y el O-RU en el centro de la celda a la que da cobertura.

Cómo también se mencionó en el capitulo 2, para definir el funcionamiento de cada entidad hay que definir qué punto de split se utiliza, cabe destacar que el stack de protocolos se hereda de 4G y 5G. En O-RAN el punto de split es un tema importante, ya que debe optimizar el equilibrio entre los requisitos de latencia y el ancho de banda de la red de transporte, teniendo en cuenta esto, O-RAN entiende que el split mas conveniente entre O-DU y O-RU es el que se conoce como 7.2x. En la figura 3.4 se repsenta el split 7.2x y como se distribuye el stack de protocolos en los componentes del O-gNB.

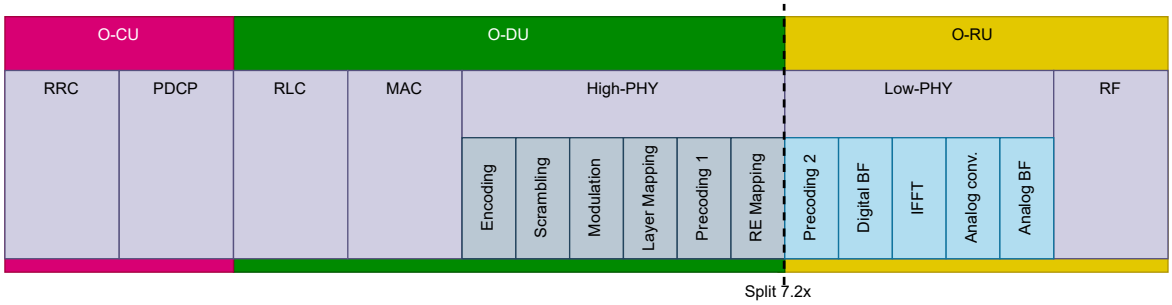

Figura 3.4: Representación del split 7.2x mostrando en que elemento de la red se implementa cada parte del *stack*.[3].

Como este *split* realiza parte del procesamiento de la señal en el O-DU se reduce la cantidad de datos que debe enviarse por la interfaz de *Fronthaul* disminuyendo los requisitos de ancho de banda y costos de transporte, además esta división de tareas, donde las mas intensivas se realizan en el O-DU mejora la latencia, ya que el O-DU por naturaleza tiene mayor poder de procesamiento que el O-RU. Otro factor a tener en cuenta es la necesidad de 5G de la instalación masiva de antenas, al descargar tareas complejas en el O-DU, la la O-RU se simplifica, haciéndola más asequible y fácil de desplegar[4]. Además tener la mayoría de las tareas en el O-DU facilita la introducción de nuevas características a futuro mediante actualizaciones de software sin necesidad de modificar el hardware.

En la figura 3.4 también se observa el split entre el O-CU y O-DU (split 2). Este split facilita la segmentación lógica entre el O-CU y O-DU ya que permite centralizar el O-CU y desplegar cerca del borde al O-DU. Entre las ventajas que ofrece se destaca la centralización del control, lo que facilita la movilidad, la coordinación de servicios, la gestión de seguridad y la virtualización de funciones en el O-CU. Además, permite que el O-DU permanezca cercano al usuario, optimizando las funciones sensibles a la latencia. Sin embargo, también presenta desventajas. Por ejemplo requiere una interfaz F1 con baja latencia y suficiente capacidad, por lo que depende fuertemente de la calidad del transporte. Además, desplaza parte del procesamiento al O-DU, que necesita mayor capacidad de cómputo que en splits más altos. En resumen, el split 2 equilibra centralización y eficiencia en movilidad, pero sus exigencias de transporte y su menor grado de desagregación pueden limitar su uso en despliegues con enlaces de backhaul/fronthaul más restringidos.

3.6. O-eNB

Como la red LTE aún es ampliamente utilizada y se requiere interoperabilidad se define el O-eNB como la versión de O-RAN para la clásica radio base de LTE eNB. En este caso las capas RRC, PDCP, RLC, MAC y High-PHY se concentran

[3]Imagen basada en el artículo https://la.mathworks.com/discovery/o-ran.html
[4]https://www.5gtechnologyworld.com/open-ran-functional-splits-explained/

en el BBU (desplegado en una nube local), mientras que al igual que para el O-gNB, el O-RU implementa la capa Low-PHY y las funciones de RF.

Respecto a las interfaces incluye las definidas por la 3GPP (X2-C y X2-U para comunicarse con otros O-eNB y O-gNB) y agrega las interfaces O1 contra el SMO y E2 contra el Near-RT RIC.

3.7. O-Cloud

El último elemento que se ve en el diagrama de la figura 3.1 es el O-Cloud. Como se puede observar el mismo esta "por fuera" de la arquitectura, esto es porque el O-Cloud no es un componente puntual en sí como lo son el resto, si no que es la representación de la plataforma de virtualización distribuida. Comprende un conjunto de nodos de infraestructura física que alojan las funciones de red de O-RAN.

Como se ha mencionado en este capítulo, las funciones de red pueden estar geográficamente ubicadas en lugares distantes, esto significa que en cada lugar (celda, borde de al red, centro de datos principal, etc) debe haber una plataforma de virtualización. Como la RAN trabaja de forma coordinada es fundamental que este sistema distribuido sea gestionado por un mismo elemento. Esta tarea recae sobre el SMO que mediante la interfaz O2 administra todas estas plataformas. Otro tema no menor es la coordinación temporal, es imperativo que los relojes de cada sistema estén sincronizados.

Como último comentario es importante destacar que los recursos necesarios para virtualizar cada función de red deben estar bien escalados. No es lo mismo el hardware necesario para virtualizar el SMO (que requiere una gran cantidad de memoria y núcleos de CPU para desarrollar su tarea) que el O-DU (que seguramente no tenga tantos requisitos de CPU y memoria como el SMO, pero si necesite aceleración de hardware para trabajar con frecuencias altas).

3.8. Interfaces

Hasta el momento se han introducido y explicado todos los elementos que componen la arquitectura de O-RAN, en esta sección se verán las interfaces que permiten la comunicación entre dichos componentes, detallando su funcionamiento, características, tipos de mensajes, etc. Nuevamente la figura 3.1 sirve como guía y es fundamental para entender las interfaces de O-RAN. Como ya se mencionó solo se profundizará en las nuevas interfaces definidas por O-RAN.

3.8.1. E2

La interfaz E2 es una interfaz abierta y lógica que conecta el Near-RT RIC con los nodos E2, cada nodo E2 se conecta solo a un Near-RT RIC, pero un Near-RT RIC se puede conectar a varios nodos E2. La interfaz le permite al Near-RT RIC controlar y gestionar los recursos de radio entre otras funcionalidades, además a

través de la interfaz E2 el Near-RT RIC recolecta datos, métricas e información de gestión de la RAN, ya sea periódicamente o utilizando *triggers*. Es importante destacar que debe cumplir con los requisitos de latencia casi en tiempo real. Los componentes que permiten la comunicación entre el Near-RT RIC y los nodos E2 son el E2AP (*E2 Application Protocol*) y el E2SM (*E2 Service Model*).

El **E2AP (*E2 Application Protocol*)** opera en el plano de control de la interfaz E2 y su función principal es estructurar y gestionar el intercambio de mensajes de control y monitoreo entre el Near-RT RIC y los nodos E2. Define que los procedimientos de señalización que permiten monitorear y controlar los nodos E2.

E2AP define varias primitivas o procedimientos funcionales que permiten la comunicación, los mas destacados son:

- **RIC Subscription**: establece suscripciones para eventos específicos en el nodo E2, permitiendo al Near-RT RIC recibir informes cuando ocurren esos eventos.
- **RIC Indication**: transmite información de eventos y métricas desde el nodo E2 hacia el Near-RT RIC.
- **RIC Control**: permite al Near-RT RIC enviar comandos de control al nodo E2 para aplicar configuraciones.
- **RIC Query**: para realizar consultas de información relacionada con la RAN o con usuarios específicos.
- **E2 Setup**: establece la conexión inicial entre el Near-RT RIC y el nodo E2.
- **E2 Node Configuration Update**: permite actualizar la configuración de los nodos E2.
- **E2 Removal**: facilita la desconexión de un nodo E2 de la interfaz E2 cuando ya no es necesario o cuando se requiere una reasignación.

Los mensajes E2AP se codifican mediante ASN.1 permitiendo representar los datos de manera compacta y eficiente. Para intercambiar los mensajes, E2AP utiliza en capa de transporte el protocolo SCTP y en capa de red el protocolo IP (IPv4 o IPv6). Es común cifrar a nivel de capa de red mediante IPsec.

El **E2SM (*E2 Service Model*)** estructura la información y los procedimientos para que el Near-RT RIC pueda interactuar de manera efectiva con las funciones implementadas en los nodos E2. El E2SM establece los elementos específicos que deben incluirse en los mensajes de la E2AP para poder manejar servicios y funciones específicos en los nodos E2. Los servicios básicos implementados son:

- **REPORT**: permite al Near-RT RIC recibir información periódica o basada en eventos desde el nodo E2. Esto es común en casos de monitoreo de métricas de rendimiento.

- **INSERT**: habilita al Near-RT RIC configurar el nodo E2 para que detenga temporalmente un procedimiento o proceso y envíe un informe al RIC antes de continuar.
- **CONTROL**: es utilizado por el Near-RT RIC para enviar comandos que permiten ajustar o modificar el comportamiento de un nodo E2
- **POLICY**: permite al Near-RT RIC definir y aplicar políticas de comportamiento dentro del nodo E2.

Estos servicios básicos se combinan para formar los modelos de servicio específicos, O-RAN define al menos tres:

- **E2SM-KPM**: modelo que permite la monitorización de indicadores clave de rendimiento (KPIs), proporciona reportes periódicos de métricas relacionadas con el desempeño de la red y los nodos E2.
- **E2SM-RC**: facilita ajustes en los niveles de acceso, control de llamadas, y distribución de recursos para usuarios específicos o para celdas en la red. El modelo incluye procedimientos para comandos en tiempo casi real que ayudan a mantener la calidad de servicio (QoS) en condiciones variables de carga.
- **E2SM-CCC**: se centra en la configuración y el control de los parámetros operativos de la RAN. Este modelo permite al Near-RT RIC ajustar configuraciones de la celda, como el tamaño de la celda, potencia de transmisión, y configuraciones de handover.
- **E2SM-NSSAI**: permite al Near-RT RIC manejar la selección y administración de Network Slices en la RAN.

Resumiendo, E2AP es el protocolo de capa de aplicación que opera sobre la interfaz E2, el mismo provee ciertas primitivas que posibilitan la comunicación entre el Near-RT RIC y los nodos E2. Los mensajes que se intercambian (orden, estructura, etc) son definidos en base a modelos de servicio, cada modelo de servicio no es mas que una secuencia de servicios básicos que buscan un determinado fin. En la figura 3.5 se muestra la estructura de un paquete E2.

Con el fin de aclarar las interacciones o relaciones entre el E2AP y el E2SM se dará el siguiente ejemplo:

El Near-RT RIC desea monitorear cierto parámetro de un nodo E2. Primero el Near-RT RIC inicia un procedimiento de suscripción (RIC Subscription) de E2AP para recibir reportes de rendimiento específicos de la celda. Utiliza el servicio REPORT del E2SM-KPM para configurar las métricas que necesita monitorear. Si en algún momento el nodo E2 quiere notificar de este valor al Near-RT RIC envía automáticamente un RIC Indication al Near-RT RIC con los datos que especificó el modelo de servicios E2SM-KPM, particularmente en el servicio REPORT.

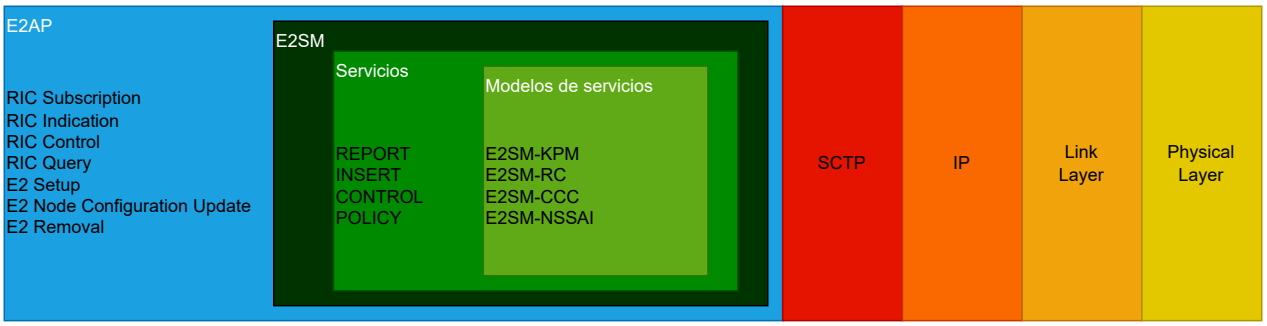

Figura 3.5: Estructura del paquete E2.

3.8.2. O1

La interfaz O1 permite la comunicación de gestión y orquestación de los nodos E2 y el Near-RT RIC con el SMO. Facilita las funciones de gestión, como la configuración, supervisión de fallos, control del rendimiento, y transferencia de archivos. Basada en estándares existentes como los definidos por 3GPP (TS 28.532 y TS 28.545) y el uso de protocolos seguros, la interfaz O1 emplea NETCONF para configuraciones y actualizaciones, HTTPS para el envío de notificaciones en formato JSON y SFTP para la transferencia de archivos. En transporte se utiliza TCP y en capa de red el protocolo IP.

La arquitectura de la interfaz O1 se basa en el concepto de elementos gestionados, donde cada nodo E2 o el Near-RT RIC es uno de ellos. A su vez cada elemento gestionado tiene servicios administrados (*Managed Services* o MnS). Estos MnS son componentes que facilitan la gestión y supervisión de la red de forma modular utilizando servicios específicos para cada *"tipo"* de administración. En otras palabras, cada elemento gestionado interactúa con el SMO a través de los MnS bajo el concepto de productor y consumidor.

Cada MnS está diseñado para realizar tareas específicas de gestión (configuración, monitoreo de fallos, recolección de datos de rendimiento, etc.), y cada uno sigue un protocolo estándar (como NETCONF o HTTP/HTTPS, SFTP) para garantizar la interoperabilidad y la seguridad. Los MnS definidos por O-RAN son los siguientes:

- **Provisioning Management Services**: facilita la configuración y modificación de atributos de objetos gestionados en la red. Utiliza el protocolo NETCONF para manejar configuraciones de manera segura y estructurada.
- **Fault Supervision Management Services**: permite la detección, reporte y seguimiento de alarmas y fallos en los elementos de la red.
- **Performance Assurance Management Services**: recolecta y analiza datos de rendimiento, con opciones para transmisión en tiempo real o mediante reportes en formato de archivo para el monitoreo continuo de KPIs y estado de la red

- **File Management Services**: facilita la transferencia segura de archivos entre el SMO y los elementos gestionados, los archivos pueden ser logs, archivos de configuiración, actualizaciones de firmware, etc.

- **Subscription Control and Notification Services**: permite que los consumidores de MnS se suscriban a notificaciones específicas, como alarmas o eventos de estado, para recibir actualizaciones en tiempo real sobre cambios en la red

En las especificaciones de O-RAN [1] se detalla como se debe realizar cada transacción a través de los MnS. Una representación del paquete O1 se puede ver en la figura 3.6

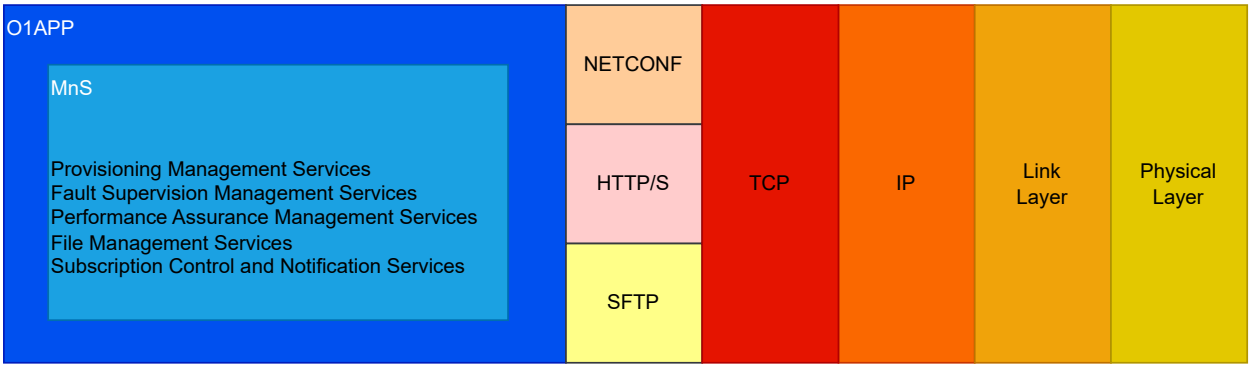

Figura 3.6: Estructura del paquete O1.

3.8.3. A1

La interfaz A1 está diseñada para conectar el Non-RT RIC directamente con el Near-RT RIC permitiendo la comunicación y colaboración entre estos dos componentes, en general el Non-RT RIC la utiliza para enviarle directrices y datos al Near-RT RIC para optimizar la operación de la RAN.

Los datos se intercambian utilizando objetos JSON sobre HTTPS, TCP e IP. Esto permite el intercambio de información de forma segura y estructurada. En la figura 3.7 se muestra una representación del paquete A1.

En capa de aplicación se definen tres servicios:

- **Policy Management**: le permite al Non-RT RIC definirle políticas al Near-RT RIC que guíen las operaciones que realiza este último sobre los nodos E2. Las políticas se crean y gestionan de forma declarativa, es decir, se definen en términos de objetivos a alcanzar sin especificar los pasos concretos para lograrlos. Esto le da al Near-RT RIC flexibilidad en cómo cumplir estos objetivos.

- **Enrichment Information**: posibilita el envío de información de enriquecimiento desde el Non-RT RIC al Near-RT RIC. Este tipo de información, que podría incluir ubicaciones de usuarios o patrones de tráfico, es valiosa para mejorar las decisiones de control en la red. Se entiende por información enriquecida a datos adicionales (complementarios a los datos estándar de la red) que permiten optimizar funciones y ajustes. La fuente de esta información puede ser interna (generada en el Non-RT RIC) o externa (facilitada desde el exterior).

- **AI/ML Model Management**: es una función específica para gestionar modelos de aprendizaje automático. Este servicio permite que el Non-RT RIC administre y distribuya modelos de AI/ML al Near-RT RIC para optimizar la toma de decisiones y el control de recursos en la RAN. Aunque nada impide entrenar los modelos directamente en el Near-RT RIC, hay consenso en que el Non-RT RIC es un entorno mas adecuado. Respecto a al ejecución de los modelos es indistinto donde se realiza, siempre y cuando se respeten las latencias máximas de los bucles de control.

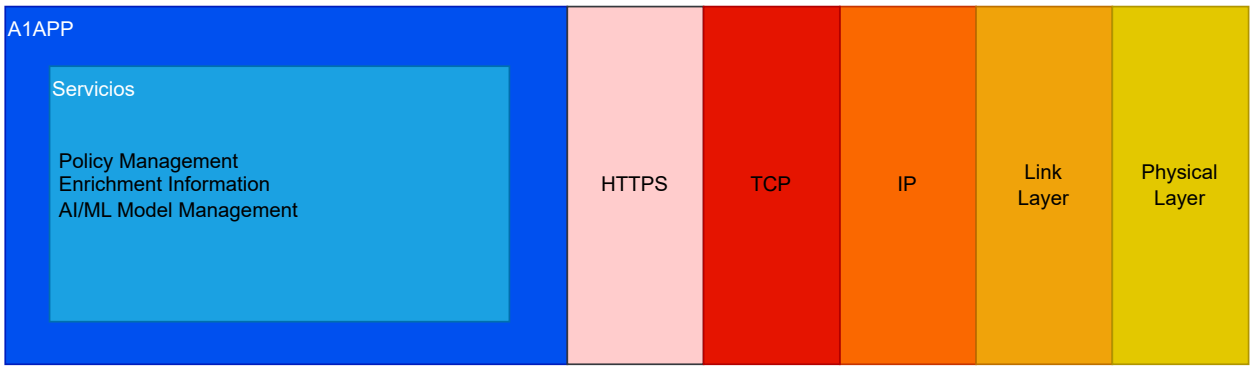

Figura 3.7: Estructura del paquete A1.

En el ejemplo visto en este capítulo sobre el funcionamiento de rApps y xApps, la interfaz A1 juega un papel fundamental ya que por esta interfaz es que se comunican las rApps con las xApps.

3.8.4. O2

La interfaz O2 proporciona una conexión lógica, abierta y segura entre el SMO y el O-Cloud, permitiendo la administración de la infraestructura del O-Cloud y la gestión del ciclo de vida de los despliegues de funciones de red dentro de O-Cloud. Esta interfaz es extensible, lo cual facilita la adición de nuevas funciones sin modificar los protocolos o procedimientos. Se estructura en dos bloques funcionales: el IMS (*Infrastructure Management Services*) y el DMS (*Deployment Management Services*).

El IMS es responsable de brindarle al SMO, particularmente a la SMOS FOCOM, una interfaz para gestionar los recursos de infraestructura (hardware, software, capacidad de procesamiento, memoria, etc) dentro del O-Cloud. Entre las tareas que permite está la gestión de inventario (inventario de recursos físicos y lógicos disponibles), el monitoreo (datos de telemetría en tiempo real de los recursos del O-Cloud), gestión del ciclo de vida de los recursos, notificación de fallos (enviar y recibir alertas) y administración del software (sistemas operativos, firmware, gestión de actualizaciones).

El DMS se encarga de la administración y orquestación del ciclo de vida de los despliegues de funciones de red. Se enfoca específicamente en administrar los despliegues de las funciones de red, asegurando que las aplicaciones y funciones de red operen de manera óptima en el entorno distribuido de O-Cloud. Físicamente conecta la SMOS NFO con el O-Cloud. Las funciones que cumple son las de instanciar y terminar funciones de red, monitorear y recuperar funciones de red, escalar vertical y horizontalmente, administrar actualizaciones y parches del software que implementan las funciones de red, entre otras.

Lo mencionado en este apartado puede verse representado en la figura 3.8.

3.8.5. Y1

Y1 es una interfaz lógica y abierta diseñada para habilitar el análisis de la RAN en tiempo casi real a través del Near-RT RIC, dicho de otra forma, la interfaz Y1 le permite a consumidores externos autorizados conectarse al Near-RT RIC para extraer datos y analizarlos fuera del contexto de la RAN. Es importante que sea una interfaz abierta y bien definida para que sea accesible por varios tipos de consumidores, ademas debe ser extensible (debe poder incorporar nuevos servicios y tipos de datos sin necesidad de modificar los protocolos o procedimientos). La información se envía en formato JSON, mediante HTTPS/TCP/IP.

Los servicios principales que soporta son:

- **Y1 RAI Subscription**: permite la suscripción de consumidores a los datos que quiera recibir.
- **Y1 RAI Query**: le permite al consumidor realizar consultas puntuales de datos, no requiere suscripción.

3.8.6. FrontHaul

La interfaz de FrontHaul (FH) es aquella que conecta el O-DU con el O-RU, y opcionalmente el SMO con el O-RU. Lógicamente se divide según el tipo de información que transporta, si es en el plano de usuario O-FH U, si es en el plano de control O-FH C, si es en el plano de sincronismo O-FH S y finalmente si es en el plano de gestión O-FH M. Es común ver los planos de control, usuario y sincronismo juntos (O-FH CUS) y el plano de gestión por separado. Para referirse a la interfaz FH completa se utilizará el término O-FH CUS/M.

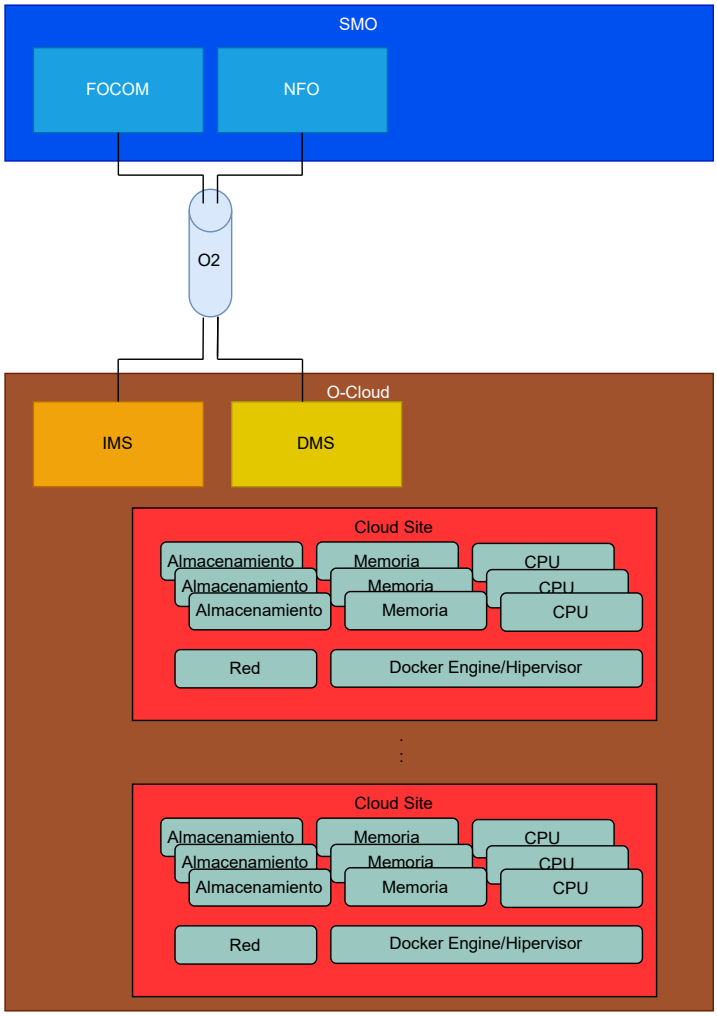

Figura 3.8: Representación de la interfaz O2, donde se ve que SMOS del SMO se comunica con el IMS e IDS dentro del O-cloud. Además se representa como varios grupos de recursos de hardware y software (*cloud sites*) conforman (y son administrador por) el O-Cloud.

Esta interfaz permite distribuir las funcionalidades de capa física entre el O-DU y O-RU, por esto sus funciones y características están fuertemente ligadas al *split* funcional elegido, que en el caso de O-RAN es el 7.2x. En la figura 3.9 se muestra la ubicación de la interfaz dentro de este split.

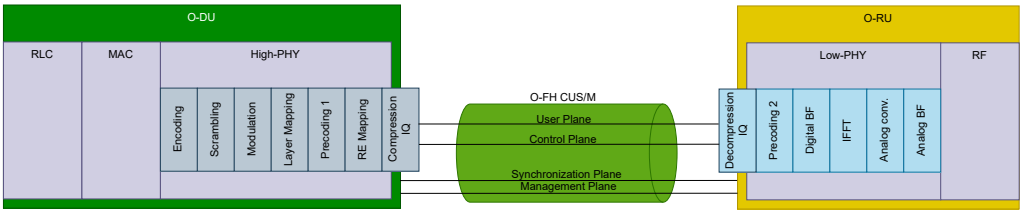

Figura 3.9: Representación de la interfaz O-FH CUS/M.

3.8.7. O-FH C

El plano de control tiene la función de manejar los datos de control asociados con la configuración y operación del O-RU y el O-DU. Este plano se centra en la transmisión de comandos y configuraciones necesarios para el procesamiento de datos de usuario (plano de usuario). Los mensajes intercambiados pueden ser enviados tanto para el enlace descendente como para el enlace ascendente dependiendo de las necesidades de configuración.

Se utiliza encapsulado de transporte como eCPRI o IEEE 1914.3 (RoE), lo que permite el envío de mensajes sin ser concatenados con los del plano de usuario, asegurando así que ambos flujos se mantengan separados dentro de la misma interfaz Ethernet. Esta separación es importante porque permite que el tráfico crítico del plano de control, que incluye configuraciones de sincronización y parámetros de control en tiempo real, no se vea afectado por la carga de datos del plano de usuario. Está habilitado el uso de etiquetas VLAN para diferenciar y priorizar tráfico. Opcionalmente se puede utilizar IP y UDP si se opera en redes más complejas. En la figura 3.10 se representa el stack del plano de control.

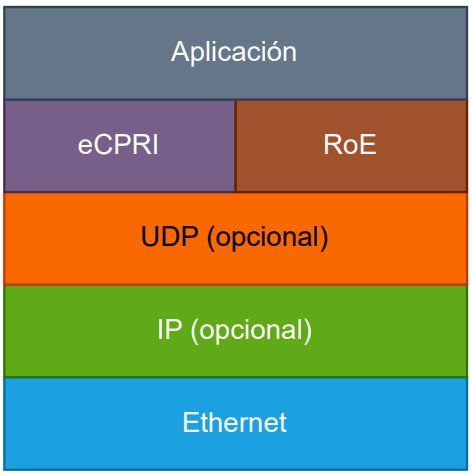

Figura 3.10: Stack de protocolos de O-FH C.

3.8.8. O-FH U

Se encarga de transmitir los datos de usuario (generados en su origen por el núcleo de la red o por el UE) en forma de muestras IQ (fase y cuadratura) entre el O-DU y O-RU. Comparte varias de las características del plano de control, como lo son el uso de doble capa de encabezado (eCPRI o RoE + aplicación), mismo stack (figura 3.10), uso de prioridades y clases de tráfico.

Suma la posibilidad de comprimir datos (compresión de muestras IQ), la fragmentación y ensamblaje, ya que el tamaño de los datos puede ser mayor que el tamaño máximo de paquetes (MTU) y el uso de ventanas de tiempo para segurar que los mensajes lleguen de manera precisa y sincrona. También es importante destacar que para los datos de usuario no se utilizan reconocimientos (ACK/NACK).

Antes de enviar cada paquete de datos de usuario, se configuran los parámetros de red a través del canal de control, o sea que cada mensaje del plano de usuario se antecede de mensajes del plano de control. Hay cierta excepciones donde los parámetros de transmisión se configuran estáticamente mediante el plano de gestión.

3.8.9. O-FH S

El plano de sincronismo es esencial para garantizar que el O-DU y el O-RU mantengan una alineación precisa en el tiempo y la frecuencia, cosa que es fundamental cuando se trabaja con sistemas de radio. Su objetivo es distribuir información de tiempo, frecuencia y fase para que los O-DU y O-RU estén alineados en estos parámetros.

Los protocolos utilizados para distribuir la información de sincronización son PTP y SyncE, ambos funcionando directamente sobre Ethernet (ver figura 3.11).

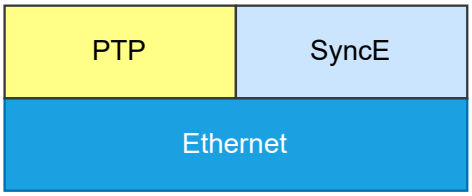

Figura 3.11: Stack de protocolos de O-FH S.

La O-RAN Alliance da cuatro arquitecturas posibles (ver figura 3.12) para la red de sincronismo:

- **LLS-C1**: establece un enlace punto a punto (a nivel de sincronismo) entre el O-DU y O-RU, donde el O-DU actúa como maestro. Este es el escenario más directo para distribuir la sincronización de tiempo y frecuencia hacia el O-RU y es la configuración mas básica que se puede tener.
- **LLS-C2**: la configuración LLS-C2 en el plano de sincronización extiende la configuración punto a punto de LLS-C1 para permitir la inclusión de

switches Ethernet en la ruta de sincronización entre el O-DU y el O-RU. En este caso los switches deben ser compatibles con los protocolos PTP y SyncE para poder funcionar como *boundary clock* o *transparent clock*. Qué modo usar y la cantidad de saltos máxima de saltos dependerá del máximo error o *drift* admitido.

- **LLS-C3**: representa una topología en la cual el O-DU no forma parte de la cadena de sincronización hacia el O-RU. En su lugar, la sincronización de frecuencia y tiempo se distribuye mediante la red de fronthaul desde una fuente central de sincronización. Tiene la mismas limitantes a nivel de red que LLS-C2, pero esta arquitectura facilita tener relojes primarios redundantes.

- **LLS-C4**: en este mecanismo el O-RU y O-DU se sincronizan localmente utilizando una fuente de referencia local sin la participación de la red de transporte. Este enfoque es útil para entornos donde la red de fronthaul no puede cumplir con los estrictos requisitos de sincronización de O-RAN. La fuente de sincronismo mas común en este caso es GNSS (Sistema Global de Navegación por Satélite). Es importante destacar que para utilizar esta topología de sincronismo en general se requieren relojes locales de mayor calidad para minimizar el drift.

3.8.10. O-FH M

El plano de gestión es responsable de todas las operaciones de gestión (no en tiempo real) entre el O-RU y el O-DU, u opcionalmente, entre el SMO y el O-RU. Su objetivo principal es permitir la configuración, supervisión y administración del hardware y software del O-RU, así como el mantenimiento y las actualizaciones necesarias para su operación continua en la RAN.

A diferencia de los otros planos, el de gestión utiliza un variado stack de protocolo para realizar sus tareas, siendo el principal NETCONF, que opera sobre SSH o TLS para mas seguridad. El modelado de datos se hace mediante el lenguaje YANG. Esto no quita que se utilicen otras herramientas bien conocidas en la gestión de redes y protocolos de red en general como SNMP, ICMP, DHCP, entre otros, en la figura 3.13 se muestra el stack completo.

Las principales funciones que se realizan sobre esta interfaz se listan a continuación:

- **Configuración Inicial y Arranque**: facilita el arranque del O-RU al configurarse con parámetros básicos, como la dirección IP, la asignación de VLAN y la sincronización de tiempo.

- **Actualización de Parámetros**: permite ajustar ciertos parámetros críticos en base a las métricas recolectadas, como la potencia de transmisión o las configuraciones de frecuencia.

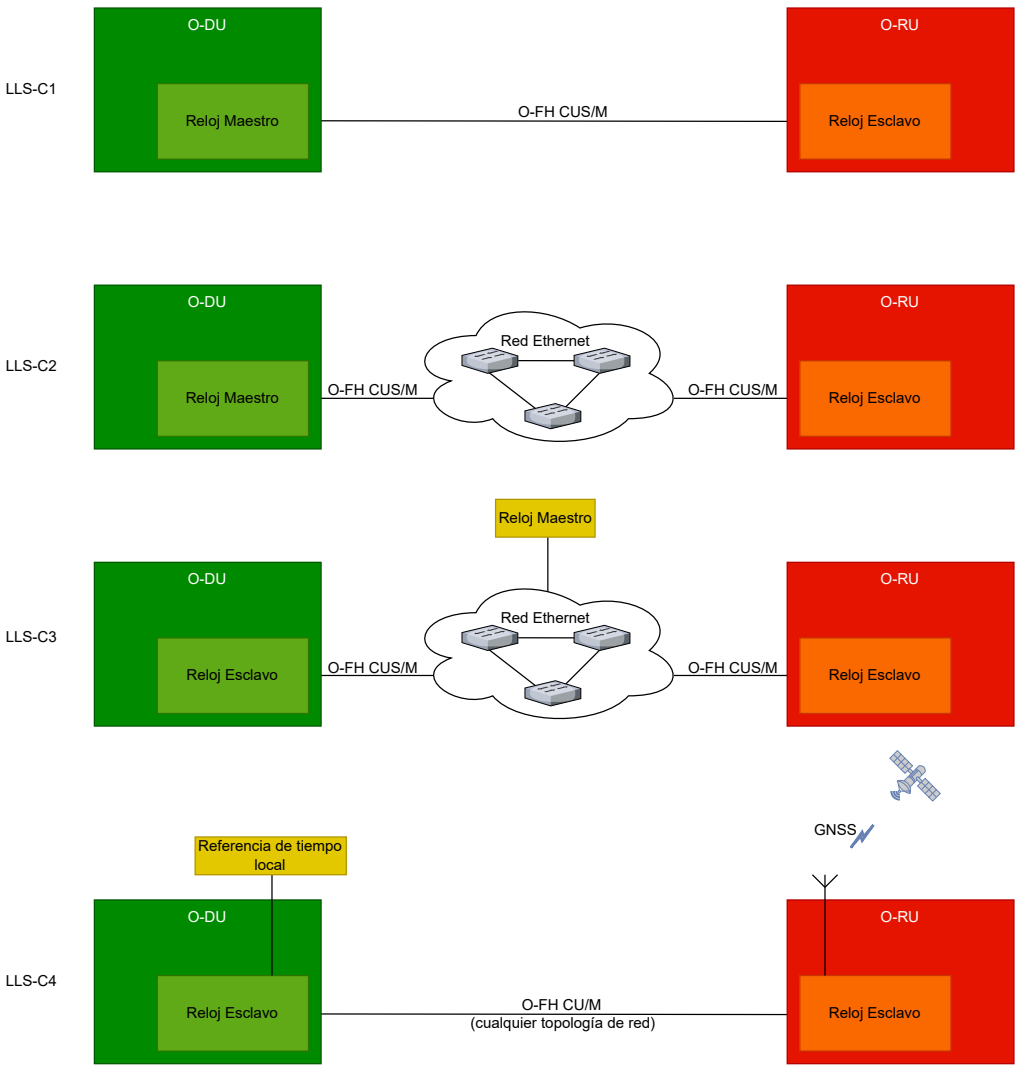

Figura 3.12: Topologías de sincronismo LLS-C1, LLS-C2, LLS-C3 y LLS-C4.

- **Monitoreo de Estado y Métricas de Desempeño**: supervisa indicadores clave de rendimiento (KPIs) y otros parámetros del O-RU, como la temperatura, el uso de CPU y las tasas de error en la transmisión.

- **Notificación de Eventos**: cuando ocurre un fallo o se supera un umbral crítico, envía notificaciones de eventos y alarmas a sistemas superiores (O-DU o SMO), permitiendo acciones de mantenimiento.

- **Registro de Fallos**: registra en logs los fallos del sistema.

- **Gestión de Certificados**: utiliza un sistema de gestión de certificados para garantizar que las comunicaciones de gestión estén cifradas y autenticadas, evitando accesos no autorizados al O-RU.

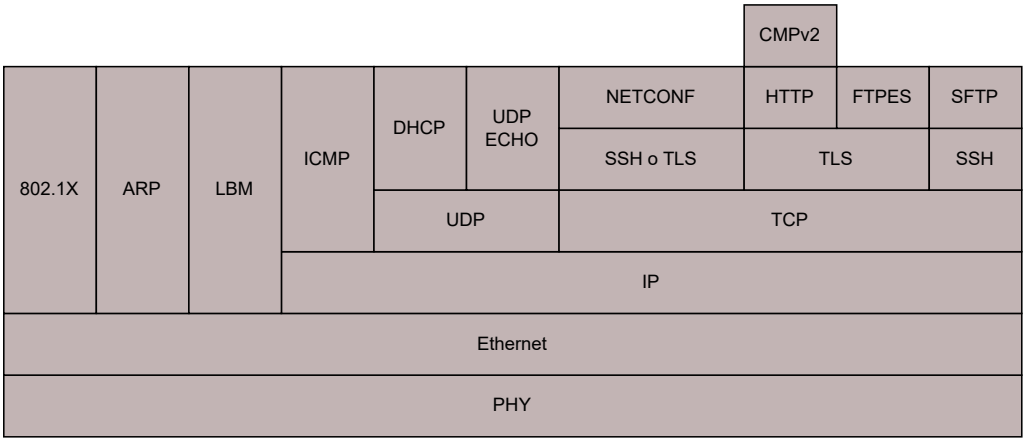

Figura 3.13: Stack de protocolos de O-FH M.

- **Control de Acceso y Autenticación**: NETCONF y SSH incluyen mecanismos de autenticación que limitan el acceso a usuarios autorizados, manteniendo la integridad y seguridad del sistema de gestión.
- **Gestión de Software y Actualizaciones**: facilita la instalación de nuevas versiones de software, incluyendo parches y actualizaciones necesarias para el O-RU.

3.9. Conclusiones del capítulo

En este capítulo se presentaron los elementos que componente la arquitectura de O-RAN (SMO, Non-RT RIC, Near-RT RIC, O-CU, O-DU y O-RU) así como las interfaces que los interconectan (E2, O1, A1, O2, Y1 y FH). De los componentes se profundizó en los elementos y funciones internas de cada uno detallando las interacciones entre ellos. De las interfaces se vio qué componentes comunican, qué información se intercambia a través de ellas y el stack de protocolos utilizado para dicho intercambio así como las primitivas y mensajes propios de O-RAN.

En este capítulo también se introdujeron las xApps y rApps, primero como elementos que componen los RICs y luego a través de ejemplos como entidades que interactúan con la red móvil.

En los siguientes capítulos de “bajarán a tierra” estos conceptos a través de la práctica ya que se indicará como instalar y utilizar distintas plataformas de prueba así como la realización de algunos experimentos demostrativos.

Capítulo 4

Antecedentes de implementaciones

Este capítulo se divide en dos secciones, en la primera se realiza una exploración de antecedentes (*estado del arte*) de implementaciones totales o parciales de O-RAN realizadas por la comunidad científica. De ellos se analizará qué escenarios manejan, qué componentes utilizan, las funcionalidades que despliegan, qué recursos de hardware fueron necesarios, etc. Luego en la segunda sección se realiza una breve introducción a tres herramientas que no surgen con O-RAN pero que sí son ampliamente utilizadas en los *testbeds*: srsRAN, Open Air Interface (OAI) y Open5gs.

4.1. Implementación de testbed O-RAN

El primer *paper* que se mencionará, (*Challenges and Lessons Learned During Private 5G Open RAN Deployments* [23]), no es una implementación directa de una maqueta, banco de prueba o *testbed* si no que es un análisis de los desafíos que han tenido diferentes intentos de despliegues de O-RAN así como las lecciones aprendidas de los mismos. El estudio se hizo en 2023 tomando en cuenta al menos cinco despliegues reales. Sobre las dificultades encontradas los autores comienzan diciendo que casi todos los *testbeds* se desarrollaron en fases, comenzando por las más sencillas y dejando las más complicadas para fases posteriores. Continúa afirmando que uno de los principales retos fue la integración a nivel de hardware y software, que resultó más compleja de lo esperado, generando un impacto negativo en el rendimiento del sistema, como tasas de datos inferiores a las anticipadas debido a una optimización insuficiente. Esta complejidad también aumentó el tiempo y los recursos necesarios, lo que derivó en un proceso menos eficiente en términos de costos y retrasos en el cronograma de implementación. Otro obstáculo significativo fue la falta de habilidades técnicas específicas de los recursos humanos para llevar a cabo la tarea. Los equipos que dependieron exclusivamente de sus recursos internos subestimaron la escala y complejidad del despliegue. Adicionalmente, los proveedores de hardware mostraron reticencia a realizar pruebas de interoperabilidad, lo que reveló problemas de compatibilidad entre productos de diferentes proveedores, especialmente en la interacción entre componentes críticos como el O-CU y O-DU.

Esta falta de interoperabilidad limitó la cantidad de pruebas funcionales realizadas en esta fase inicial. La configuración manual de los sistemas 5G SA también fue un desafío importante, debido a la escala y complejidad de las tareas necesarias. Esto subrayó la necesidad de avanzar hacia plataformas de prueba automatizadas para reducir los tiempos de configuración y aumentar la eficiencia. Finalmente, la integración del RIC (Near-RT y Non-RT) quedó rezagada en comparación con otros componentes de O-RAN, dado que su complejidad requiere fases adicionales de prueba y desarrollo. En esta etapa inicial, se priorizaron otras funcionalidades clave, dejando el RIC como un enfoque central para futuras fases del desarrollo. Pasando a las lecciones aprendidas, se identificó la necesidad de adquirir nuevas habilidades, ya que gran parte de los ingenieros y expertos actuales carecen de la capacitación adecuada para trabajar con O-RAN, especialmente en entornos 5G SA. Además, se enfatizó la importancia de realizar pruebas exhaustivas para garantizar la interoperabilidad y cumplimiento de los estándares, dado que algunos productos no soportan interfaces abiertas como se afirma. La configuración manual resultó ser un proceso desafiante, evidenciando la necesidad de automatizar tanto las pruebas como las configuraciones. Asimismo, se destacó la necesidad de establecer acuerdos de nivel de servicio a nivel de sistema, ya que la validación de componentes individuales no garantiza el éxito de extremo a extremo en un entorno de múltiples proveedores. Finalmente, se señaló que la gestión post-despliegue de O-RAN es un desafío continuo que requiere actualizaciones, mejoras y resolución de problemas constantes, para lo cual se recomienda adoptar metodologías ágiles como CI/CD (*Continuous Integration/Continuous Deployment*) para optimizar los procesos.

El objetivo del trabajo *An Open, Programmable, Multi-vendor 5G O-RAN Testbed with NVIDIA ARC and OpenAirInterface* [54] es desarrollar y evaluar un *testbed* de redes privadas 5G basadas en O-RAN llamado por los autores X5G. Una vez que se analiza el *paper* se ve que logran implementar una red 5G desagregada, pero no se implementa ningún RIC (aunque dicen que X5G está diseñado para integrarse a ellos). Propiamente de O-RAN, lo más cercano que utilizan es el *split* 7.2x. Otra particularidad que tiene este trabajo es que divide la implementación del O-DU en dos equipos diferentes, quedando RLC y MAC en uno, y High-PHY en otro. La interacción entre el "DU-High" y "DU-Low" se logra mediante la interfaz *Functional Application Platform Interface* (FAPI)[1]. El *core* de Open Air Interface corre en un servidor DELL R750 (56 núcleos, 256 GB de RAM), el CU y DU-High son implementaciones de OAI mientas que el DU-Low está basado en el NVIDIA Aerial SDK[2]. Para todo el gNB utiliza un servidor Gigabyte E251-U70 (CPU Intel Xeon Gold 6240R de 24 núcleos y 96 GB de RAM) en el cual se instalan dos GPU NVIDIA A100 y una tarjeta de red Mellanox CX6-DX. En el RU se implementa mediante una radio base Foxconn RPQN7801 que opera en la banda de los 3.7, 3.8 GHz y tiene cuatro antenas montadas. Para la sincronización temporal se agregan un reloj PTP Qulsar QG-2 como reloj maestro. Finalmente la conectividad es brindada por un switch DELL S5248F que ademas funciona como *boundary clock*.

[1] https://scf.io/en/documents/222_5G_FAPI_PHY_API_Specification.php
[2] https://developer.nvidia.com/aerial

En el trabajo titulado *Accelerating Open RAN Research Through an Enterprise-scale 5G Testbed* [6] los autores reconocen la importancia de los *testbeds* académicos pero afirman que dichos trabajos no son realistas en términos de fidelidad y estabilidad comparado con un sistema real, por este motivo ellos se plantean crear un *testbed* de O-RAN de escala empresarial. El banco de pruebas combina soluciones avanzadas de hardware y software para abordar limitaciones de plataformas previas, ofreciendo realismo, escalabilidad y flexibilidad. En el hardware, destacan ocho servidores HPE Telco DL110 Gen10 con procesadores Xeon 6338N y aceleradores Intel ACC100, switches Arista para los enlaces, unidades de radio Foxconn 4×4, y un reloj PTP Qulsar Qg2 para sincronización. Además, cuenta con dispositivos de usuario como Raspberry Pi con módems 5G, smartphones y dispositivos IoT. En el software, utiliza Linux CBL Mariner[3] optimizado para cargas en tiempo real, junto con soluciones vRAN como Intel FlexRAN para la capa física y la solución 5G de CapGemini[4] para L2/L3, gestionadas mediante Kubernetes y contenerización. Además integra herramientas como el framework Janus para telemetría y control en tiempo real, y automatiza despliegues con Helm charts, simplificando la experimentación y configuraciones avanzadas. Los autores consideran que los resultados obtenidos validan la funcionalidad del *testbed*, cumpliendo con su propósito de proporcionar un entorno de investigación realista y estable para O-RAN. Respecto a este *paper* es importante destacar que en nigún momento hace mención de los RIC.

En el *paper TinyRIC: Supercharging O-RAN Base Stations with Real-time Control* [27], los autores presentan TinyRIC, una plataforma que incorpora un control en tiempo real (RT-RIC) dentro de las radio bases de O-RAN. Esto aborda la carencia de capacidades RT en las arquitecturas existentes de O-RAN, optimizando tareas críticas como la asignación de recursos y el manejo del scheduler MAC a intervalos de tiempo extremadamente reducidos (del orden de microsegundos). TinyRIC introduce aplicaciones ligeras denominadas tApps, que posibilitan flexibilidad en la gestión de red y escalabilidad. Además, se incluye una arquitectura de mensajería interna denominada Mailbox, que simplifica la comunicación entre componentes. Los resultados experimentales destacan la viabilidad del sistema para escenarios de control RT, mostrando una latencia mínima y alta adaptabilidad frente a condiciones de red variables. Aunque en base a lo documento por los autores es una propuesta más que interesante la de agregar un controlador en tiempo real, en el documento no se profundiza como se implementó el mismo.

Ante la falta (según los responsables del estudio) de una plataforma de pruebas completa y estable (*end-to-end*) los autores del trabajo *Open AI Cellular (OAIC): An Open Source 5G O-RAN Testbed for Design and Testing of AI-Based RAN Management Algorithms* [53] proponen una solución llamada OAIC como banco de pruebas. OAIC se divide en dos partes, OAIC-C y OAIC-T. OAIC-C es el marco de desarrollo diseñado para implementar los controladores de radio inteligentes (Non-RT y Near-RT) así como todo el entorno 5G de manera estandarizada y abierta. Para el *core* de la red, el O-gNB y los UE utiliza una versión modificada

[3]https://es.wikipedia.org/wiki/CBL-Mariner
[4]https://www.capgemini.com/

de las soluciones brindadas por srsRAN[5]. El Near-RT RIC y el agente para la interfaz E2 está basado en el desarrollo de la O-RAN *Software Community*. De la OSC también se tomaron ejemplos de xApps. En el *paper* no se hace mención de la implementación utilizada para el SMO ni el Non-RT RIC, pero explorando la documentación del proyecto[6] se puede ver que sí existe aunque no se dan mayores detalles. El OAIC-T es un marco automatizado para probar y validar el funcionamiento de los RIC, sobre todo el funcionamiento de las xApps y rApps. OAIC-T facilita la configuración automática de entornos de prueba, la ejecución de pruebas y la generación de informes detallados sobre el desempeño. Además, emplea métodos de IA para controlar las entradas y parámetros, permitiendo evaluar el rendimiento del sistema en diferentes condiciones de tráfico y estado de la red. Es importante mencionar que todo el despliegue se realiza mediante Kubernetes, lo que hace este *testbed* sencillo de implementar y fácil de escalar. En la prueba que realizan los autores no se especifica que características de hardware de cómputo utilizaron, solo que se utilizó la infraestructura de CORNET (*Cognitive Radio Network*) de la universidad de Virginia Tech[7]. Los unicos datos de hardware que se dan es que los enlaces de red son de 10 Gbps y que se usaron SDR USRP X310 y N310. Finalmente, al igual que el trabajo anterior, propone la creación del RT RIC, este RIC alojaría zApps para el control en tiempo real de la RAN.

En el artículo *Prototyping Next Generation O-RAN Research Testbeds with SDRs* [52] se busca prototipar una plataforma de pruebas de O-RAN utilizando hardware comercial y software libre. Este *testbed* integra software de código abierto y hardware COTS para evaluar la interoperabilidad entre los componentes de O-RAN, con especial énfasis en la integración del Near-RT RIC a través de la interfaz E2. Se basa en srsRAN extendido con un agente E2 compatible con la arquitectura O-RAN. Esto permite la comunicación con el Near-RT RIC, utilizando xApps para optimizar el control de la RAN en tiempo casi real. Se implementaron dos xApps: KPIMON, que recopila métricas de rendimiento de la red (*Physical Resource Blocks* o PRBs, número de conexiones activas, QoS, etc.), y RAN Slicing, que gestiona la asignación de recursos en un entorno de slicing dinámico. El hardware utilizado incluye computadoras con procesadores Intel Core i9 de 11ª generación, 64 GB de RAM y GPUs Nvidia RTX A-5000, junto con radios SDR USRP B205-mini, B210, X310 y N310. La infraestructura se despliega con Kubernetes para facilitar la virtualización y escalabilidad de los servicios de O-RAN. El paper describe en detalle la implementación del flujo de trabajo E2, desde la configuración inicial hasta la interacción de las xApps con el RIC mediante mensajes E2AP en formato ASN.1. En la prueba realizada, se validó la interoperabilidad entre el Near-RT RIC y la RAN basada en srsRAN, confirmando que el testbed es funcional para la experimentación con control inteligente de la red en un entorno O-RAN. Se destaca la necesidad de seguir avanzando en la estandarización de frameworks abiertos para mejorar la adopción de O-RAN en entornos reales.

El trabajo *CloudRIC: Open Radio Access Network (O-RAN) Virtualization*

[5]https://www.srsran.com/

[6]https://openaicellular.github.io/oaic/index.html

[7]https://wireless.vt.edu/research/facilities.html

with Shared Heterogeneous Computing [46] no se trata directamente de una implementación de un banco de pruebas de O-RAN, si no que atiende a un problema intrínseco de los despliegues de O-RAN: el alto costo energético y de aceleradores de hardware en el O-DU. En un esquema clásico, cada O-DU tiene aceleradores de hardware (generalmente GPUs o FPGAs) dedicados. Los autores de este *paper* proponen la plataforma *CloudRIC* que permite compartir este hardware entre varios O-DU además de utilizar CPUs normales para disminuir costos y consumo sin comprometer la fiabilidad. *CloudRIC* introduce dos componentes clave en la arquitectura O-RAN, el *AAL Broker*, que gestiona dinámicamente el acceso de múltiples O-DU a los recursos computacionales compartidos, y un *Real-Time RAN Intelligent Controller* que ajusta la asignación de radio en función de la carga computacional. Este enfoque permite lograr un balance óptimo entre costo, consumo energético y confiabilidad. El sistema se implementó sobre una infraestructura de O-Cloud con GPUs NVIDIA V100 y CPUs Intel Xeon Gold 6240R, utilizando modelos de IA para la asignación de tareas en tiempo real. Las pruebas demostraron que *CloudRIC* logra mejoras de 3x en eficiencia energética y 15x en eficiencia de costos en comparación con los enfoques tradicionales.

La siguiente publicación estudiada es la del NIST[8] (*National Institute of Standards and Technology*) titulada *Blueprint for Deploying 5G O-RAN Testbeds: A Guide to Using Diverse O-RAN Software Stacks* [28]. Ante la necesidad de establecer bancos de pruebas O-RAN completos y replicables, los autores presentan una guía detallada para desplegar un *testbed* de 5G O-RAN desde cero. Este trabajo ofrece una visión integral de la arquitectura O-RAN, describe diferentes *stacks* de software para cada componente y proporciona ejemplos de implementación tanto en escenarios agregados como desagregados. El documento detalla la instalación de diversos software de código abierto utilizados en la comunidad O-RAN, como srsRAN para el gNB y UE, Open5GS para el core de la red, FlexRIC y OSC Near-RT RIC para el control inteligente de la RAN, y herramientas de simulación de la interfaz E2. El *testbed* se implementó en servidores con procesadores Intel Xeon Gold y Core i9, junto con radios SDR Ettus USRP B210 y X310. Se exploraron múltiples topologías de despliegue, desde configuraciones centralizadas en un solo servidor hasta arquitecturas desagregadas en varios servidores interconectados. También se validaron distintas formas de conexión entre el gNB y los UE, incluyendo conexiones simuladas (ZeroMQ[9]) y reales en radiofrecuencia con hardware SDR. También se da una herramienta de automatización que facilita la instalación y configuración del banco de pruebas, reduciendo el error humano y acelerando los tiempos de despliegue. Se realizaron pruebas funcionales de conectividad y rendimiento, incluyendo *ping tests*, mediciones de *throughput* con iperf3[10] y validación de xApps en el Near-RT RIC. Los resultados demostraron la viabilidad del testbed para experimentación e investigación en entornos 5G O-RAN abiertos. Es importante destacar que en el trabajo no se realiza un despliegue del Non-RT RIC, los autores dejaron esta implementación para agregar en el futuro.

[8] https://www.nist.gov/
[9] https://zeromq.org/
[10] https://software.es.net/iperf/

El trabajo *Testbed development: An intelligent o-ran-based cell-free mimo network* [11] presenta un *testbed* cell-free MIMO (las celadas tradicionales se reemplazan por un conjunto de muchos RUs distribuidos que cooperan para atender a todos los usuarios de forma simultanea) capaz de operar con COTS UE 5G. Se compone del stack 5G de srsRAN modificado junto con Open5GS y el Near-RT RIC de la OSC. Como hardware de computo utilizaron un servidor Dell PowerEdge R7525 con 64 núcleos de CPU, ademas para la parte de radio se instalaron USRP X310 y un reloj OctoClock NI. También incorporan una xApp para la asociación inteligente de antenas basado en *deep reinforcement learning* que ajusta dinámicamente los grupos de antenas según mediciones de KPM. Según los autores se demostró con éxito la operación de una red Cell-Free MIMO integrada en arquitectura O-RAN, capaz de servir equipos 5G comerciales y ejecutar MU-MIMO uplink gracias a las modificaciones realizadas en la capa MAC y PHY de srsRAN además de la incorporación de la xApp inteligente.

En *Campus5G: A Campus Scale Private 5G Open RAN Testbed* [15] los autores afirman ser el primer despliegue a escala de un campus de una red 5G con soporte para O-RAN. La propuesta destaca por su cumplimiento total con O-RAN, su uso de RUs comerciales y su operación en un entorno urbano exterior. El *testbed* Campus5G se construyó sobre una infraestructura hardware totalmente compatible con O-RAN, compuesta por 20 unidades de radio comerciales (12 Benetel RAN650 y 8 VVDN Mid-Power RU) operando en la banda n77 a 100 MHz y MIMO 4×4. Para el procesamiento se desplegaron seis servidores telco HP DL110 G10 Plus equipados con aceleradores Intel ACC100 y enlaces agregados de hasta 200 Gbps por nodo, complementados por un servidor GPU para AI/ML, un servidor de gestión y switches Arista dedicados tanto para el fronthaul como para la red de control. La sincronización temporal requerida por el estándar O-RAN se garantizó mediante un PTP Grandmaster Viavi Qg 2 y una red de fibra óptica instalada específicamente para cumplir con las restricciones estrictas de latencia del fronthaul. Sobre esta base se ejecutó una plataforma software flexible construida sobre Ubuntu, automatización mediante Ansible y orquestación con Kubernetes y Helm. Se utilizaron varias herramientas de software (como srsRAN y OpenAirInterface), además de soluciones comerciales basadas en Intel FlexRAN así como un núcleo Open5GS. También se integraron RICs estándar (O-RAN SC RIC, FlexRIC) y de investigación (EdgeRIC, Janus/jBPF), junto con xApps como KPIMON y aplicaciones basadas en IA, habilitando una plataforma altamente programable.

El equipo de la Northeastern University presenta en su paper *AutoRAN: Automated and Zero-Touch Open RAN Systems* [29] un framework integral de automatización cuyo objetivo es el despliegue, configuración y prueba zero-touch de redes 5G basadas en O-RAN. Los autores identifican que los sistemas celulares modernos han incrementado significativamente su complejidad debido a la coexistencia de múltiples microservicios, la fragmentación del stack RAN, la necesidad de gestionar radios heterogéneas y la presencia de aceleradores hardware especializados. Estas fuentes de complejidad representan un problema especialmente crítico para redes privadas 5G, donde empresas sin experiencia telco deben operar infraestructuras avanzadas. Para abordar estas limitaciones, AutoRAN se fundamenta en

un conjunto de abstracciones cloud-native que permiten virtualizar computación, red y aceleración hardware, exponer funciones RAN como microservicios, y gestionar el sistema mediante un enfoque declarativo que implementa el concepto de RAN-Infrastructure-as-Code. El diseño de AutoRAN se compone de un cluster OpenShift con 13 nodos heterogéneos que incluyen CPU x86 y ARM, GPUs NVIDIA L40, A100 y GH200, y radios comerciales (Foxconn) y SDRs (NI/Ettus). A nivel software integra múltiples soluciones como OpenAirInterface, NVIDIA ARC-OTA, srsRAN, un núcleo 5G Open5GS y RICs de la OSC, todo ello gestionado con herramientas como Tekton, ArgoCD, Podman, SR-IOV, y una cadena de compilación multi-arquitectura para soportar ARM y x86. Los resultados experimentales muestran que AutoRAN puede desplegar una RAN completa en menos de 60 segundos y alcanzar hasta 1.6 Gbps en pruebas con 15 UEs simulados usando RuSIM.

Los trabajos titulados *O-RAN with Machine Learning in ns-3* [17] y *Programmable and Customized Intelligence for Traffic Steering in 5G Networks Using Open RAN Architectures* [25] cuya particularidad es que presentan despliegues de O-RAN en el simulador ns-3[11]. En el primer trabajo se presenta una extensión del simulador ns-3 que modela la arquitectura de O-RAN permitiendo simular redes móviles bajo el paradigma de O-RAN, incluso integrarlos con otros módulos de ns-3 como por ejemplo los de aprendizaje automático. En el modelo propuesto, el Near-RT RIC se implementa como un contenedor lógico que agrupa todos los componentes funcionales necesarios para su operación. Entre estos se encuentran: un repositorio de datos para el almacenamiento de información, módulos lógicos que permiten la ejecución de xApps, un módulo de mitigación de conflictos, generadores de reportes, y la terminación de la interfaz E2. Para probar el modelo diseñaron un experimento basado en la gestión del *handover* en redes LTE donde probaron (entre otras cosas) que utilizando O-RAN y *machine learning* se reduce significativamente la perdida de paquetes en relación al esquema tradicional de las redes móviles. En el segundo trabajo se diseñó un módulo para ns-3 que permite conectar una simulación de ns-3 a un Near-RT RIC real. En este caso los autores utilizan una versión antigua del Near-RT RIC de la OSC junto con una versión específica de ns-3. Para conciliar la diferencia entre el tiempo que maneja el Near-RT RIC (tiempo del mundo real) y el tiempo en la simulación (tiempo discreto) ns-3 almacena la hora Unix al inicio de la simulación y la utiliza como marca de tiempo base. Cada vez que se envía un mensaje E2 al RIC, el simulador suma el tiempo de simulación transcurrido a la marca de tiempo base, asegurando coherencia en ambos lados de la relación de precedencia entre eventos. Para probar el sistema diseñaron un experimento cuyo objetivo es optimizar el direccionamiento de tráfico a través de una xApp logrando una mejora sustancial en el rendimiento de la red.

El paper [40] presenta FCT O-RAN, un testbed privado 5G de extremo a extremo diseñado e implementado en el Singapore Institute of Technology (SIT). La motivación del trabajo surge de los desafíos prácticos asociados a la desagregación del RAN, la coexistencia de múltiples proveedores y la necesidad de gestionar los servicios 5G mediante arquitecturas software dinámicas y programables. En este

[11] https://www.nsnam.org/

contexto, FCT O-RAN constituye una plataforma experimental representativa de despliegues industriales, integrando componentes comerciales de Microsoft Affirmed y ENEA para el 5G Core, CU/DU de Radisys basados en Intel FlexRAN, y unidades de radio indoor Foxconn y outdoor Benetel, todas interconectadas a través de interfaces O-RAN. La virtualización de las funciones de red se realizó sobre servidores HP ProLiant DL110 y DL360. Para optimizar el despliegue de las RUs, los autores desarrollan un digital twin del campus utilizando Wireless InSite, generando un modelo 3D para análisis de propagación, selección de ubicaciones y predicción de métricas como RSSI, SINR y throughput. Posteriormente validan el testbed mediante walk tests con un Samsung S22 instrumentado con QualiPoc.

Finalmente se mencionarán tres trabajos que utilizan O-RAN para aplicaciones distintas a dar servicios móviles directamente.

El artículo *Prototyping O-RAN Enabled UAV Experimentation for the AERPAW Testbed* [32] explora el uso de la arquitectura de O-RAN para el control de vehículos aéreos no tripulados (UAVs). En el trabajo se despliega el *core* 5G utilizando la solución de Open5GS[12] y el gNB de OAI[13]. En los UAVs se utiliza el UE de OAI. De O-RAN se implementa el Near-RT RIC y la interfaz E2 de FlexRIC también de OAI. En el Near-RT RIC corren xApps para monitorizar la dinámica de la red, estos datos son almacenados en una base de datos SQLite. Todo el experimento corre sobre la plataforma AERPAW[14] (*Aerial Experimentation and Research Platform for Advanced Wireless*). Esta plataforma cuenta con un ecosistema compuesto por una infraestructura distribuida que incluye UAVs equipados con radios definidos por software (SDRs), gNBs, nodos fijos y móviles, una red de fibra óptica de baja latencia y capacidad de cómputo. AERPAW permite realizar pruebas tanto en entornos reales como simulados. Hablando del hardware para el prototipado y pruebas, se utilizaron computadoras Intel NUC13 (Intel Core i7-10710U de 6 núcleos con 32GB de RAM), SDR USRP B210, amplificadores (Mini-Circuits ZHL-15W-422-S+, Mini-Circuits ZVE-8GX+ y Mini-Circuits ZX60-83LN12+) filtros (Mini-Circuits VLF-4400+ y Mini-Circuits VBFZ-3590+) y antenas (Mobile Mark RM-WB1-DN-B1K y Octane Wireless SA-1400-5900).

Luego en el trabajo *AI-Open-RAN for Non-Terrestrial Networks* [13] se propone una arquitectura unificada siguiendo los principios de O-RAN y la 3GPP para tener redes de acceso radio inteligentes en entornos no terrestres (NTN), como satélites y plataformas aéreas. Los autores redefinen las interfaces internas del RAN para que puedan operar sobre enlaces NTN con altas variaciones de movilidad, retardo y doppler. El trabajo implementa un *testbed* 5G SA con OAI (gNB y UE) implementado en una OAI BOX y mediante un módulo Quectel 5G con antenas integradas. Utilizando estos elementos realizaron una captura de datos (KPIs) para luego entrenar un modelo LSTM (*Long Short-Term Memory*) con el objetivo de predecir KPIs futuros.

Por último el paper *Development of open radio access networks (O-RAN) for real-time obotic teleoperation* [24] busca aprovechar los beneficios de O-RAN para

[12]https://open5gs.org/
[13]https://openairinterface.org/
[14]https://aerpaw.org/

el control de un brazo robótico a distancia y demostrar la viabilidad de O-RAN para aplicaciones de alta confianza y baja latencia. Utilizan srsRAN para el gNB y UEs, una versión del RIC OSC y SDRs USRP X310. La plataforma ejecuta una xApps de monitoreo y control que releva datos del sistema para mantener la comunicación estable con latencias menores a 50 ms.

En resumen se se analizaron 17 trabajos relacionados a implementaciones de O-RAN, desde artículos publicados hace algunos años hasta otros en proceso de publicación a noviembre de 2025, lo que demuestra el interés y dedicación de la academia a O-RAN. En la tabla 4.1 se resumen los trabajos analizados.

4.2. Herramientas

Como el lector habrá notado, un factor común en la mayoría de las plataformas de pruebas detalladas anteriormente es el uso de OAI, srsRAN y Open5GS, ya sea una en exclusividad o combinándolas. Por este motivo ahora se dará una breve introducción a cada una.

4.2.1. srsRAN

srsRAN (anteriormente conocido como srsLTE) es un conjunto de herramientas de software de código abierto orientadas a la investigación, el desarrollo y la experimentación en redes móviles. Su principal característica es que implementa, en software, los componentes fundamentales de un sistema de comunicaciones móviles 4G LTE y, más recientemente, 5G. Gracias a ello, investigadores, operadores y desarrolladores pueden desplegar y probar redes móviles en entornos de laboratorio o de campo, sin depender de equipos propietarios.

El proyecto está desarrollado y mantenido por Software Radio Systems (SRS) y se ha consolidado como una de las plataformas de referencia en el ámbito académico e industrial para explorar tecnologías basadas en Open RAN. Entre sus módulos más relevantes se encuentran el eNodeB/gNodeB, el EPC/5GC y el UE, lo que permite montar una red completa con hardware estándar y radios definidas por software (SDR).

La filosofía de srsRAN está alineada con el movimiento open source: transparencia en la implementación, flexibilidad para modificar el código y posibilidad de adaptarlo a distintos escenarios, desde pruebas de laboratorio hasta despliegues pre-comerciales. De esta manera, se convierte en una herramienta clave para experimentar con arquitecturas abiertas, validar nuevas propuestas de investigación y acelerar la adopción de soluciones interoperables en el ecosistema de Open RAN [5] [50] [43].

4.2.2. OAI

OpenAirInterface (OAI) es una iniciativa de código abierto orientada al desarrollo y experimentación de tecnologías de comunicación móvil basadas en los

Trabajo	Tipo de implementación	Stack / componentes	RIC	Hardware	Aporte principal
Challenges and Lessons Learned During Private 5G Open RAN Deployments	Análisis de despliegues reales	5G SA multivendor	No implementado	No detallado	Identificación de desafíos prácticos, interoperabilidad y necesidad de automatización
An Open, Programmable, Multi-vendor 5G O-RAN Testbed with NVIDIA ARC and OAI	Testbed 5G privado	OAI Core, CU/DU, NVIDIA Aerial SDK	No	Servidores Dell y Gigabyte, GPU A100, RU Foxconn	Testbed desagregado con split 7.2x y aceleración GPU
Accelerating Open RAN Research Through an Enterprise-scale 5G Testbed	Testbed escala empresarial	FlexRAN, CapGemini 5G, Kubernetes	No	HPE DL110, Intel ACC100, RU Foxconn	Testbed realista, estable y escalable a nivel enterprise
TinyRIC: Supercharging O-RAN Base Stations with Real-time Control	Prototipo de control RT	Arquitectura propietaria ligera	RT-RIC embebido	No detallado	Control en tiempo real a nivel MAC con latencias de microsegundos
Open AI Cellular (OAIC)	Testbed end-to-end automatizado	srsRAN, OSC RIC, Kubernetes	Near-RT y Non-RT	USRP X310/N310, infraestructura CORNET	Framework completo para desarrollo y testeo de xApps y rApps
Prototyping Next Generation O-RAN Research Testbeds with SDRs	Testbed experimental	srsRAN extendido + E2 Agent	Near-RT	Intel i9, GPUs RTX A5000, USRP	Validación del flujo E2 y control inteligente vía xApps
CloudRIC: O-RAN Virtualization with Shared Heterogeneous Computing	Arquitectura experimental	O-Cloud + IA	RT-RIC	GPU V100, Intel Xeon	Optimización de costos y energía compartiendo aceleradores
Blueprint for Deploying 5G O-RAN Testbeds (NIST)	Guía de implementación	srsRAN, Open5GS, FlexRIC, OSC	Near-RT	Intel Xeon, USRP B210/X310	Testbed replicable con múltiples topologías y automatización
Intelligent O-RAN-based Cell-Free MIMO Testbed	Testbed Cell-Free MIMO	srsRAN modificado, Open5GS	Near-RT	Dell R7525, USRP X310	Integración de Cell-Free MIMO con xApps basadas en DRL
Campus5G: A Campus Scale Private 5G Open RAN Testbed	Testbed a escala campus	srsRAN, OAI, FlexRAN, Open5GS	Near-RT	Servidores HP, RU Benetel/VVDN	Primer despliegue O-RAN campus-wide totalmente compliant
AutoRAN: Automated and Zero-Touch Open RAN Systems	Framework de automatización	OAI, srsRAN, Open5GS, OpenShift	OSC RIC	Cluster heterogéneo CPU/GPU/ARM	Despliegue zero-touch y RAN-as-Code
O-RAN with Machine Learning in ns-3	Simulación	ns-3 + modelo O-RAN	Near-RT simulado	Simulado	Extensión de ns-3 para evaluar O-RAN con ML
Programmable Intelligence for Traffic Steering using O-RAN	Simulación híbrida	ns-3 + Near-RT RIC real	Near-RT	Simulado	Integración ns-3 con RIC real para control de tráfico
FCT O-RAN Testbed (SIT)	Testbed industrial	Radisys, FlexRAN, Open5GS	No especificado	HP DL110/DL360, RU Foxconn/-Benetel	Validación end-to-end con digital twin y walk tests
O-RAN Enabled UAV Experimentation	Testbed UAV	OAI, Open5GS, FlexRIC	Near-RT	Intel NUC, USRP B210	Aplicación de O-RAN para control de UAVs
AI-Open-RAN for Non-Terrestrial Networks	Testbed NTN	OAI gNB/UE	No	OAI BOX, Quectel 5G	O-RAN aplicado a redes no terrestres con IA
O-RAN for Real-Time Robotic Teleoperation	Testbed robótico	srsRAN, OSC RIC	Near-RT	USRP X310	Control robótico remoto con baja latencia

Tabla 4.1: Resumen de trabajos académicos sobre implementaciones y plataformas O-RAN vistos en este capítulo.

estándares 3GPP. A diferencia de soluciones propietarias, OAI permite a la comunidad académica, la industria y las instituciones públicas desplegar, modificar y validar de manera transparente componentes de red móvil que abarcan desde LTE hasta 5G NR.

El proyecto es impulsado por la OpenAirInterface Software Alliance (OSA), conformada por centros de investigación (como EURECOM) y empresas colaboradoras. La plataforma incluye implementaciones completas de los principales bloques funcionales: UE, eNB/gNB y EPC/5GC. Esto posibilita la creación de redes end-to-end en entornos de laboratorio o en despliegues de prueba, utilizando hardware comercial y SDRs

Desde el punto de vista arquitectónico, OAI busca mantener una estricta adherencia a las especificaciones 3GPP, pero al mismo tiempo introduce mecanismos que facilitan la experimentación en entornos de investigación, como interfaces abiertas, flexibilidad en la configuración de la pila de protocolos y soporte para escenarios virtualizados. Además, OAI se ha convertido en una de las plataformas de referencia en el ecosistema Open RAN, al permitir evaluar la separación entre unidades centralizadas (CU), distribuidas (DU) y de radio (RU), así como su interoperabilidad con equipos heterogéneos.

En la literatura académica, OAI ha sido descrito como un motor de innovación en el ámbito de las redes móviles abiertas, ya que democratiza el acceso a implementaciones funcionales de LTE y 5G NR, promoviendo tanto la enseñanza como el prototipado rápido de nuevas tecnologías. Este enfoque convierte a OAI en un recurso estratégico para explorar el futuro de las comunicaciones más allá de 5G y en la transición hacia 6G [22] [21] [39].

4.2.3. Open5gs

Open5GS es una implementación de código abierto del núcleo de red para comunicaciones móviles, compatible tanto con LTE como con 5G. Su objetivo es ofrecer una plataforma flexible, accesible y totalmente funcional que permita desplegar y experimentar con arquitecturas de red móvil en entornos de laboratorio, académicos y de investigación, utilizando hardware convencional y software libre.

El proyecto, desarrollado en lenguaje C y mantenido por una comunidad activa, proporciona todos los elementos principales del núcleo: MME, HSS, SGW, PGW en el caso de LTE, y AMF, SMF, UPF, PCF, AUSF, UDM en el caso de 5G. Esto permite realizar pruebas de redes móviles en modo Standalone (SA) y Non-Standalone (NSA), facilitando la evaluación de funcionalidades avanzadas como segmentación de red (network slicing), movilidad, autenticación de usuarios y gestión del plano de usuario.

Desde el punto de vista arquitectónico, Open5GS intenta mantener un equilibrio entre apegarse a las especificaciones 3GPP y ofrecer simplicidad de uso. Esto permite que los investigadores y estudiantes se concentren en probar algoritmos, prototipos o nuevas funciones sin depender de la rigidez de soluciones comerciales. Asimismo, su integración con emuladores de equipos de usuario y de acceso radio lo convierte en una herramienta muy útil para construir escenarios de prueba de

extremo a extremo.

La literatura reciente ha mostrado que Open5GS constituye una opción de bajo costo para implementar núcleos 5G. Al evaluar su rendimiento frente a otras alternativas de código abierto, como Free5GC y OAI, estudios comparativos han analizado métricas como latencia del plano de control, estabilidad del sistema y eficiencia en el uso de recursos, confirmando que Open5GS ofrece un balance sólido entre simplicidad de despliegue y rendimiento en escenarios de investigación y prueba [37] [7].

4.2.4. OSC Near-RT RIC

Es la implementación del Near-RT RIC desarrollada por la OSC. El despliegue se realiza mediante *Kubernetes* y consta de varios *pods* que implementan las diferentes funciones del Near-RT RIC, estos pods están divididos en tres *namespaces*, en la figura 5.1 se pueden ver los *pods* y *namespaces* desplegados así como la interacción entre ellos. A continuación se hará una breve descripción de cada uno.

- **Namespace ricplt**
  - **rtmgr**: encargado del enrutamiento de mensajes entre servicios utilizado el protocolo *RIC Message Router* (RMR).
  - **submgr**: gestiona las suscripciones de xApps a los servicios E2.
  - **e2mgr**: administra las conexiones E2 con los nodos de la RAN.
  - **e2term**: punto de terminación de la interfaz E2, es a donde se conectan los nodos E2.
  - **appmgr**: administra el ciclo de vida de las xApps (registro, arranque, monitoreo, etc).
  - **redis**: base de datos para almacenar información de configuración y estado.
  - **alarmanager**: detecta, gestiona y publica las alarmas generadas por los distintos componentes del Near-RT RIC.
  - **vespamgr**: gestiona las métricas y alarmas hacia sistemas externos, especialmente aquellos basados en VES (*VNF Event System*) definido por ONAP (generalmente usado en el Non-RT RIC).
  - **a1mediator**: implementa la interfaz A1.
  - **o1mediator**: implementa la interfaz O1
  - **kong**: se utiliza como punto de entrada desde el exterior a los servicios del Near-RT RIC, de esta forma se evita exponer directamente todos los servicios.
  - **prometheus**: sistema de monitoreo encargado de recolectar y almacenar metricas internas del Near-RT RIC.

    - **influxdb**: sistema de monitoreo que puede ser utilizado por una xApp para almacenar las métricas recolectadas.

- **Namespace ricxapp**

    - **xApp**: en este *testbed* las xApp son desplegadas como pods dentro del *namespace ricxapp*.

- **Namespace ricinfra**

    - **tiller**: utilizado por Helm v2 para el despliegue.

4.2.5. FlexRIC

Este Near-RT RIC es un desarrollo propio del OAI (en lo formal es un proyecto distinto, pero ambos son tutoreados por la Open Air Interface Software Alliance) llamado FlexRIC[15] (también conocido como Mosaic5g FlexRIC) que incorpora todas las funciones necesarias en una sola aplicación.

Según el artículo [47] FlexRIC no es directamente el Near-RT RIC de O-RAN, si no que es un SDK (*Software Development Kit*) que permite construir controladores SD-RAN (*Software-Defined Radio Access Network*) especializados y orientados a servicios para redes 5G, particularmente uno de estos controladores podría ser el Near-RT RIC de O-RAN. Dicen tener una arquitectura más ligera, modular y flexible basada en dos componentes (servidor y agente) que junto con aplicaciones internas (iApps) y modelos de servicio permiten programar las funciones deseadas en la RAN.

En lugar de imponer un diseño cerrado y pesado (basado en contenedores, Kubernetes y microservicios como hace la referencia de O-RAN), proporciona las herramientas básicas para que desarrolladores construyan controladores Near-RT RIC especializados y optimizados a su caso de uso.

4.3. Conclusiones del capítulo

Ver la cantidad de artículos y trabajos dedicados o relacionados con plataformas de prueba para O-RAN demuestra lo vigente que está el tema en el ámbito académico y el interés que despierta en los investigadores. Incluso, si se analizan algunos de los nombres de los autores, es posible ver que hay grupos de investigación dedicándose casi exclusivamente a O-RAN. Del análisis también se desprende el consenso general acerca de la necesidad de contar con un *testbed* para trabajar con esta tecnología. Por otro lado, la evolución y periodicidad de las publicaciones dejan en claro que es un tema que hay que seguir de cerca para mantenerse actualizado. Como conclusión final, se puede afirmar que la mayoría de las plataformas utilizan sistemas bien conocidos en el ámbito de las redes móviles (srsRAN, OAI y Open5GS). Esto representa una ventaja, ya que la curva de aprendizaje en la

[15] https://gitlab.eurecom.fr/mosaic5g/flexric

utilización de las herramientas se concentra en pocas aplicaciones y, además, si el equipo de trabajo ya viene trabajando con redes móviles tradicionales, seguramente le resulten familiares.

Capítulo 5

Experimentación con implementaciones

Luego de presentar teóricamente O-RAN y de ver las implementaciones realizadas por otros actores toca primero replicar las implementaciones de algunos de los autores mencionados, y luego modificar algunas de ellas para agregarles elementos faltantes, probarlas en otras infraestructuras o adaptarlas a necesidades particulares. Las implementaciones seleccionadas son aquellas que pueden ser replicadas con escasos recursos siempre teniendo en cuenta el objetivo de este trabajo, ser el primer contacto con O-RAN. Para facilitar la identificación y poder referenciarlas fácilmente cada implementación se irá numerando (*Implementación 1, Implementación 2, etc*).

5.1. Implementación 1

La primera implementación que se verá será la propuesta por los autores de [28] (NIST) que desarrollaron una herramienta llamada *O-RAN-Testbed-Automation*[1] que instala un *testbed* casi completo de O-RAN de forma automática en un único nodo. Ofrecen dos opciones, la primera que se verá en esta implementación se compone de srsRAN junto con el Near-RT RIC de OSC y otra con OAI y el Near-RT RIC FlexRIC que se verá en la implementación 2.

Arquitectura

En el *tesbed* se despliega una red móvil basada en el *core* de Open5gs, el gNB de srsRAN y UE de srsRAN. Respecto al Near-RT RIC se utiliza un despliegue de Kubernetes que implementa el Near-RT RIC de la OSC. En la figura 5.1 se puede ver un diagrama de la arquitectura.

De la parte de la red móvil no se profundizará en la descripción ya que son herramientas conocidas y no es el objetivo de este trabajo. Lo que es importante comentar es que el gNB incluye soporte para la interfaz E2 y que el canal inalámbrico es simulado con ZeroMQ[2] (biblioteca de mensajería asíncrona de alto

[1] https://github.com/usnistgov/O-RAN-Testbed-Automation/tree/main
[2] https://zeromq.org/

rendimiento que permite la comunicación eficiente y escalable entre aplicaciones distribuidas mediante distintos patrones de comunicación).

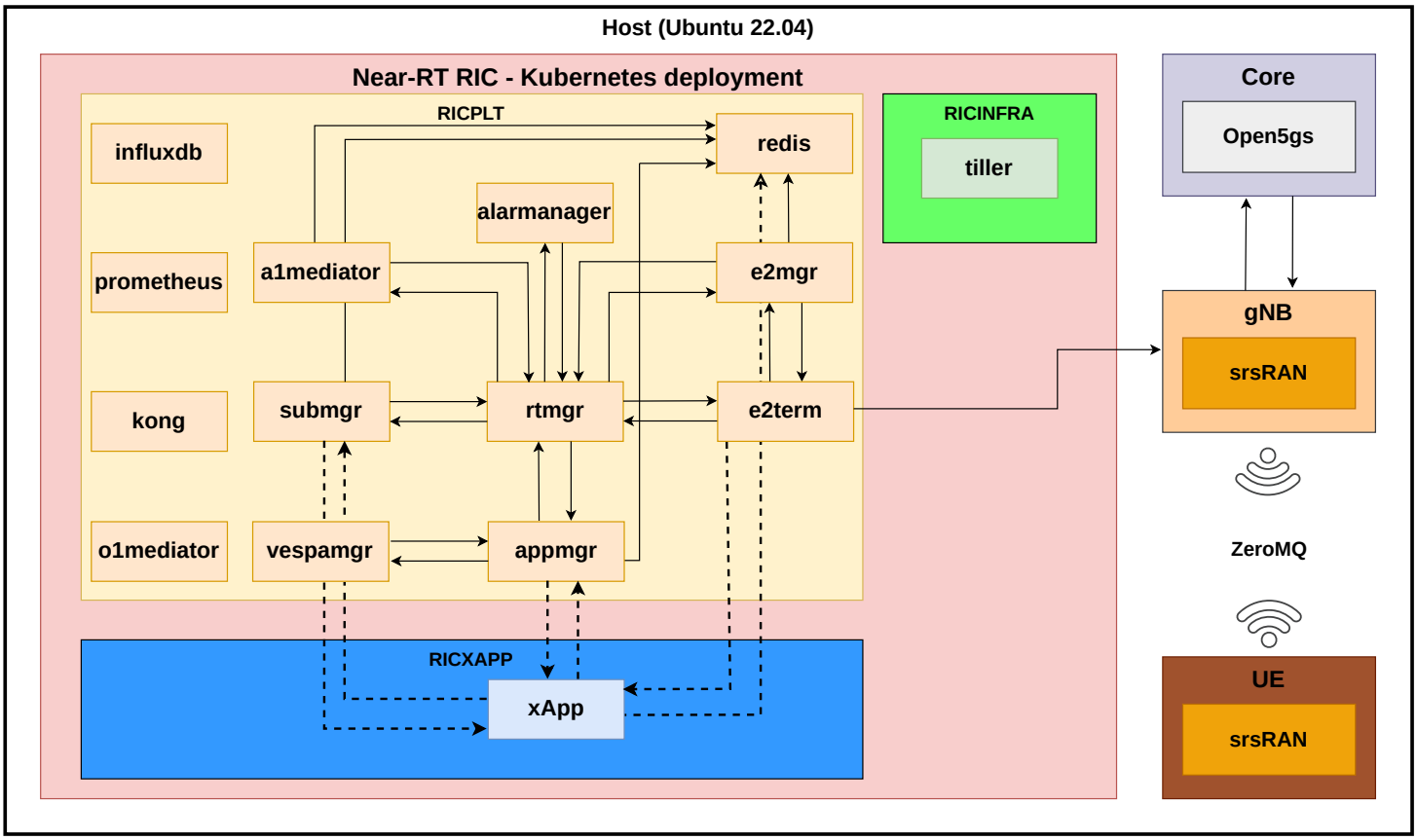

Figura 5.1: Arquitectura de la solución propuesta en la implementación 1. En el host se instala el Near-RT RIC de OSC como un despliegue de Kubernetes que consta de varios pods en tres namespaces. Además directamente sobre el host se instala el gNB y UE de srsRAN junto con el core de Open5gs.

Requisitos

- **Sistema operativo**: recomiendan utilizar Ubuntu Desktop 22.04
- **Disco duro**: 35 GB o mas
- **RAM**: 6144 MB o mas
- **CPUs**: 6

Además si se utiliza VirtualBox[3] se deben realizar las siguientes configuraciones al momento de crear la máquina virtual:

- Habilitar I/O APIC

[3]https://www.virtualbox.org/

- Habilitar PAE/NX
- Habilitar VT-x/AMD-V anidado
- Elegir interfaz de paravirtualización por defecto
- Habilitar la paginación anidada
- Dar toda la memoria de video posible
- Marcar unidad de estado sólido

Instalación

Una vez creada la máquina virtual e instalado el sistema operativo es necesario instalar las *Guest Additions* de VirtualBox, para esto en la ventana de la máquina virtual ir a **Dispositivos** → **Insertar imagen de CD con los componentes de invitado** y correr la siguiente secuencia de comandos:

```
$ sudo mkdir /media/cdrom
$ sudo mount /dev/cdrom /media/cdrom
$ cd /media/cdrom
$ sudo ./VBoxLinuxAdditions.run
$ sudo adduser $USER vboxsf
```

Luego instalar algunas dependencias y git:

```
$ sudo apt-get install -y dkms build-essential
     linux-headers-generic linux-headers-$(uname -r)
$ sudo apt-get install -y git
```

Clonar el repositorio:

```
$ git clone https://github.com/USNISTGOV/O-RAN-Testbed-
   Automation.git
```

Finalmente se debe ejecutar el script *full_install.sh* ubicado en el directorio raíz del proyecto y esperar que finalice todo el proceso. La instalación puede demorar varios minutos (dependiendo de los recursos del sistema) y necesita descargar paquetes e imágenes de docker.

```
$ cd O-RAN-Testbed-Automation
$ ./full_install.sh
```

Utilización

Al iniciar el equipo automáticamente arrancarán todos los *pods* correspondientes al Near-RT RIC, es importante aguardar a que todos los *pods* estén en estado *Running*, esto se puede verificar a través del comando:

```
$ k9s -A
```

o

```
$ kubectl get pods --all-namespaces
```

Los elementos de la red móvil se deben iniciar manualmente, esto se puede hacer de forma automática mediante el script *run.sh* ubicado en la raíz del proyecto, o se puede iniciar cada componente por separado ejecutando el script *run.sh* ubicado dentro del directorio asociado a cada elemento (`O-RAN-Testbed-Automation/5G_Core_Network` para el *core*, `O-RAN-Testbed-Automation/Next_Generation_Node_B` para el gNB y `O-RAN-Testbed-Automation/User_Equipment` para el UE).

La forma de obtener información del sistema es a través de los logs. En el caso del Near-RT RIC cada *pod* tiene su log al cual se puede acceder mediante el comando:

```
$ kubectl logs <POD> -n <NAMESPACE> -f
```

Por otra parte, los logs del *core*, gNB y UE se escriben en archivos de logs ubicados en la carpeta logs dentro del directorio correspondiente a cada elemento.

A nivel de red el *core* crea una interfaz virtual llamada *ogstun*, mientras que el UE crea un *network namespace* llamado *ue1*, esta configuración permite ejecutar ambos elementos en el mismo host pero lógicamente separados. Para "acceder" al UE se debe ejecutar:

```
$ sudo ip netns exec ue1 bash
```

De esta forma si por ejemplo se quiere traficar datos entre el UE y el *core* en sentido ascendente con iperf3 se debe ejecutar en una terminal el comando:

```
$ iperf3 -s -B <IP_ogstun>
```

y en el shell del UE (posterior a entrar al bash del namespace):

```
$ iperf3 -c -B <IP_ogstun>
```

Luego de tener todo iniciado y andando es útil verificar que el gNB se conectó a través de la interfaz E2 al Near-RT RIC, para esto se puede utilizar alguno de los siguientes comandos:

```
$ kubectl exec -it <NOMBRE_POD_e2mgr> -n ricplt -- curl
   http://localhost:3800/v1/e2t/list
$ curl -X GET http://<IP_POD_e2mgr>:3800/v1/nodeb/states |
   jq
```

El *testbed* esta hecho para soportar varias xApps por defecto, para ver o instalar las xApps disponibles dirigirse al directorio `O-RAN-Testbed-Automation/RAN_`

`Intelligent_Controllers/Near-Real-Time-RIC/additional_scripts`, en esta carpeta se encuentran (entre otros) los scripts para instalar y desplegar en Kubernetes las xApps por defecto. Las xApps incluidas en el desarrollo son[4]:

- **KPI Monitoring**:

  Esta xApp (también conocida como KPIMON) encarga de recolectar indicadores clave de desempeño (Key Performance Indicators o KPIs) de los nodos E2 (eNBs o gNBs) mediante la suscripción al *service model* E2SM-KPM, estándar que describe la semántica y el formato de los mensajes de monitoreo. El proceso se inicia con el envío de un mensaje de suscripción (*RIC Subscription Request*) en el que se especifican las métricas de interés, el alcance (por celda o por usuario) y la periodicidad de los reportes. Una vez aceptada la suscripción, los nodos E2 transmiten de manera periódica mensajes (*RIC Indication Messages*) que contienen datos de rendimiento, incluyendo utilización de recursos de radio, *throughput*, calidad de señal y niveles de carga de celda, etc.

  KPIMON procesa y decodifica estos mensajes empleando las definiciones ASN.1 del *service model*, y posteriormente almacena los resultados en una base de datos en memoria (Redis), optimizada para consultas rápidas por otras xApps o por mecanismos de control de mayor nivel. Este diseño facilita la implementación de esquemas de control en lazo cerrado, en los cuales la información recolectada se utiliza para tomar decisiones de optimización de parámetros de red, como la reasignación dinámica de recursos o la dirección inteligente del tráfico (*traffic steering*). Además KPIMON tiene integración nativa en entornos orquestados por Kubernetes lo que asegura su facil despliegue y alta disponibilidad. Esta xApp no solo sirve para recolectar datos y mostrarlos sino que también actúa como un nexo fundamental para aplicaciones de inteligencia artificialy automatización[5][6].

- **Anomaly Detection xApp**:

  Está concebida para detectar comportamientos atípicos o degradaciones en las métricas de las celdas o los equipos de usuario y emitir alertas hacia la función de *traffic steering*. Según la documentación de la comunidad O-RAN SC, esta aplicación opera iterativamente cada 10 milisegundos, extrayendo información de los UE desde una base de datos (en la versión actual utiliza InfluxDB) y generando predicciones sobre anomalías, que luego son enviadas al módulo de *traffic steering*, el cual confirma recepción mediante un mensaje ACK.

- **Traffic Steering xApp**:

[4]https://github.com/usnistgov/O-RAN-Testbed-Automation/tree/main/RAN_Intelligent_Controllers/Near-Real-Time-RIC

[5]https://hackmd.io/@abdfikih/BkIeoH9D0

[6]https://github.com/o-ran-sc/ric-app-kpimon-go

Coordina la asignación dinámica de usuarios a celdas con el objetivo de maximizar la eficiencia espectral y la calidad de experiencia. Su funcionamiento inicia con la recepción de políticas transmitidas por el Non-RT RIC a través de la interfaz A1 que establecen criterios de decisión tales como umbrales de ganancia mínima de rendimiento requeridos para justificar un cambio de celda. De forma continua, la xApp escucha los reportes de la xApp *Anomaly Detection* para identificar UEs que presentan degradación de servicio. Para cada uno de ellos, emite solicitudes a la xApp *Quality of Experience Predictor* con el fin de obtener estimaciones de *throughput* en la celda actual y en celdas vecinas. Con base en estas predicciones y en la política recibida, la xApp evalúa el beneficio potencial de reorientar al usuario y decide si se justifica un *handover*. Finalmente, la decisión de control es enviada a la xApp *RIC Control* que la convierte en un mensaje E2 de control para su ejecución en el nodo E2. De este modo la xApp cierra el ciclo de optimización al transformar información de red en acciones concretas, habilitando un control adaptativo de recursos radio en tiempo casi real y contribuyendo a la mejora global del desempeño de la red[7][8].

- **Quality of Experience Predictor xApp**:

  Trabaja en conjunto con la xApp KPIMON y *traffic steering*. Su cometido principal es estimar de forma anticipada el rendimiento esperado (*throughput*) tanto para enlace descendente como ascendente de los UEs dados en base en métricas de red disponibles. El resultado de esas predicciones es enviado nuevamente al módulo de *traffic steering*, que puede usarlo para decidir hacia qué celda o portadora dirigir cada usuario con el fin de optimizar la calidad de servicio percibida[9][10].

- **RIC Control xApp**:

  La xApp RIC Control es el componente encargado de transmitir las decisiones de control tomadas por aplicaciones de optimización o automatización (otras xApps) que operan sobre el Near-RT RIC. Recibe solicitudes de control de otros módulos (por ejemplo la xApp de *traffic steering*) y las traduce a mensajes de control conformes al estándar E2SM-RC. Para ello la xApp construye el mensaje de control con la semántica y estructura definidas por el modelo de servicio correspondiente, realiza su codificación en ASN.1 y lo transmite al nodo E2 a través del bus de mensajería RMR[11][12].

- **Hello World xApp**:

[7] https://docs.o-ran-sc.org/projects/o-ran-sc-ric-app-ts/en/latest/user-guide.html
[8] https://github.com/o-ran-sc/ric-app-ts
[9] https://docs.o-ran-sc.org/projects/o-ran-sc-ric-app-qp/en/latest/overview.html
[10] https://github.com/o-ran-sc/ric-app-qp
[11] https://docs.o-ran-sc.org/projects/o-ran-sc-ric-app-rc/en/latest/overview.html
[12] https://github.com/o-ran-sc/ric-app-rc

El *testbed* incluye dos xApp de pruebas, una realizada en Python y otra en Rust[13][14].

El espíritu original de este desarrollo es tener un despliegue de O-RAN totalmente automatizado, por este motivo las configuraciones (red, parámetros, etc) viene predefinida para que todo funcione. Puede ser necesario en algunos casos tener que editar dicha configuración, si esto es necesario todos los componentes de la red móvil se configuran desde los archivos ubicados en el directorio `config` dentro de la carpeta de cada elemento. En el caso del Near-RT RIC la tarea es más compleja, ya que habría que editar los *charts* de Helm ubicados en `O-RAN-Testbed-Automation/RAN_Intelligent_Controllers/Near-Real-Time-RIC/ric-dep`.

Uno de los casos en los que es necesario tocar la configuración del gNB y UEs es si se desea utilizar más de un UE en la red. Por defecto ZeroMQ no permite conectar directamente más de un UE, si se desea hacerlo es necesario incorporar un "intermediario". Los desarrolladores de srsRAN sugieren utilizar un script de gnuradio y realizar ciertas modificaciones a la configuración del gNB y UE. En la nota al pie[15] se deja el instructivo.

La implementación no contempla la ejecución de múltiples gNBs.

Conclusiones

Esta implementación resulto ser fácil de instalar y poner en marcha, siguiendo los pasos de la documentación se logró armar la infraestructura simulada y hacer que los diferentes elementos se vean y comuniquen entre sí. Funcionar con srsRAN y Open5gs se entiende como una ventaja ya que son sistemas ampliamente utilizados y muy bien documentados, por otro lado tener el Near-RT RIC "oficial" de la OSC también es positivo, y que sea basado en microservicios desplegados mediante Kubernetes un *plus*. El principal inconveniente es que todos estos proyectos avanzan por caminos separados, no existe una sincronización natural lo que puede dar problemas. Por ejemplo durante las pruebas realizadas para este trabajo se detectó que al instalar la xApp kpimon-go el pod nunca entraba en el estado *Running*. *Debugeando* se vio que el problema estaba cuando la xApp intentaba registrarse contra el nodo E2 (el gNB). Si se utiliza la herramienta *e2simulator* (simula un nodo E2) no había problema, pero contra el gNb si. Por este motivo y luego de recolectar la mayor información posible se planteó un *issue*[16] en el GitHub de los desarrolladores. Estos analizaron el problema y descubrieron que el gNB ya no soporta una métrica que antes si soportaba, seguramente debido a alguna actualización de srsRAN. Modificaron el código e hicieron un *commit* que deshabilita la conexión de la xApp al gNB para que no se caiga la xApp pero se

[13] https://github.com/o-ran-sc/ric-app-hw-python

[14] https://github.com/o-ran-sc/ric-app-hw-rust

[15] https://docs.srsran.com/projects/project/en/latest/tutorials/source/srsUE/source/index.html#multi-ue-emulation

[16] https://github.com/usnistgov/O-RAN-Testbed-Automation/issues/2

perdió la recolección de métricas. El equipo informó que cuando pudieran iban a investigar y buscar una solución al problema.

En resumen, aunque es una implementación prometedora puede incurrir en desfasajes de desarrollo que cause problemas de interoperabilidad.

5.1.1. Implementación 1.1

Al terminar de analizar la implementación 1 es evidente la falta del SMO y del Non-RT RIC, esto será una constante en todas las implementaciones ya que el desarrollo de este elemento ha tenido retrasos en relación al Near-RT RIC. De todas formas la OSC tiene abierto y en desarrollo el proyecto del Non-RT RIC[17]. Los autores del *testbed* utilizado en la implementación 1 han adaptado el *release L* del Non-RT RIC para poder sumarlo opcionalmente a la arquitectura desplegada.

En esta documentación se ve como implementación 1.1 ya que solo es compatible con la implementación 1 y como se verá a continuación no está del todo "aceitada".

El *release L* del Non-RT RIC de la OSC tiene los siguientes componentes y funciones[18]:

- **Non-RT-RIC Control Panel**: interfaz gráfica que permite gestionar y operar el Non-RT RIC
- **Non-RT-RIC (Spring Cloud) Service Gateway**: pasarela para enrutar solicitudes entre microservicios.
- **Non-RT-RIC (Kong) Service Exposure Prototyping**: permite exponer las funciones del Non-RT RIC a servicios externos o rApps.
- **A1 Policy Management Service**: componente principal de la gestión de políticas A1, recibe, almacena y distribuye las políticas del Near-RT RIC.
- **rApp Manager**: gestiona las rApps.
- **Information Coordinator Service**: recibe datos desde fuentes externas y coordina el flujo de información enriquecida hacía las rApps o los Near-RT RICs.
- **DMaaP/Kafka Information Producer Adapters**: se utiliza para recibir datos de fuentes externas.
- **Service Manager**: controla y coordina los servicios en ejecución del Non-RT RIC.
- **NONRTRIC CAPIF Core (Service Registry)**: es donde se descubren, documentan y registran los servicios disponibles.

[17] https://github.com/o-ran-sc/nonrtric

[18] https://lf-o-ran-sc.atlassian.net/wiki/spaces/RICNR/pages/446758914/Release+L

- **Initial Non-RT-RIC App Catalogue**: catálogo de rApps disponibles en el RIC.
- **A1 Policy Controller / Adapter**: adaptador entre el A1 Policy Management y el Near-RT RIC.
- **A1 Interface Simulator**: simula un Near-RT RIC para pruebas.
- **RAN PM Functions**: funciones de gestión de rendimiento. Recolecta, procesa y expone datos de rendimiendo de la red.
- **Authentication Support (JWT Token Fetch)**: permite la autenticación segura entre componentes y de servicios expuestos.

Al igual que ocurría con el Near-RT RIC, cada una de estas funcionas se relaciona con un *pod* del despliegue de *Kubernetes* en el cual se implementa el Non-RT RIC.

Para instalar el Non-RT RIC primero hay que tener funcionando la implementación 1 base, luego dirigirse al directorio `O-RAN-Testbed-Automation/RAN_Intelligent_Controllers/Non-Real-Time-RIC/` y ejecutar el script *full_install.sh*:

```
$ ./full_install.sh
```

Esto comenzará a descargar las imagenes y desplegar los *pods* en el *namespace* `nonrtric`. Los *pods* inician automáticamente con el sistema, se pueden realizar las mismas verificaciones explicadas en la implementación 1.

Para iniciar la interfaz web se debe ejecutar:

```
$ ./run_control_panel.sh
```

Aunque se logró instalar y desplegar, así como acceder a la interfaz web del Non-RT RIC no se lo logró que funcionaran correctamente todos los *pods* correctamente. Además a pesar de varios intentos y reinicios instalar el Non-RT RIC generó conflictos en los *pods* del Near-RT RIC. Debido a lo anterior y al problema de compatibilidad descripto en la implementación 1 se decidió no profundizar en esta implementación. Queda claro que los esfuerzos de desarrollo aún están en el Near-RT RIC, tanto por parte de la O-RAN Alliance como de los desarrolladores particulares (como los del NIST), se entiende que por el momento hay falencias en este aspecto pero seguramente con el pasar del tiempo se irán solucionando e irán apareciendo nuevas implementaciones tal como pasó con el Near-RT RIC.

5.2. Implementación 2

Como ya se mencionó la herramienta *O-RAN-Testbed-Automation* del NIST está hecha para instalar de la forma más fácil posible un *testbed* todo en uno de O-RAN. En la implementación 1 se presentó la opción srsRAN con Near-RT RIC de OSC, ahora se verá la segunda opción que se ofrece, OAI junto a el Near-RT

RIC FlexRIC.

Arquitectura

En este caso el *testbed* despliega una red móvil utilizando como *core* Open5gs y como gNB y UE la implementación de OAI con soporte E2. Por otro lado el canal de aire es simulado mediante RFSimulator[19] también desarrollado por OAI. Nuevamente vale aclarar que no se profundizará sobre la red móvil. El esquemático de la arquitectura se puede ver en la figura 5.2.

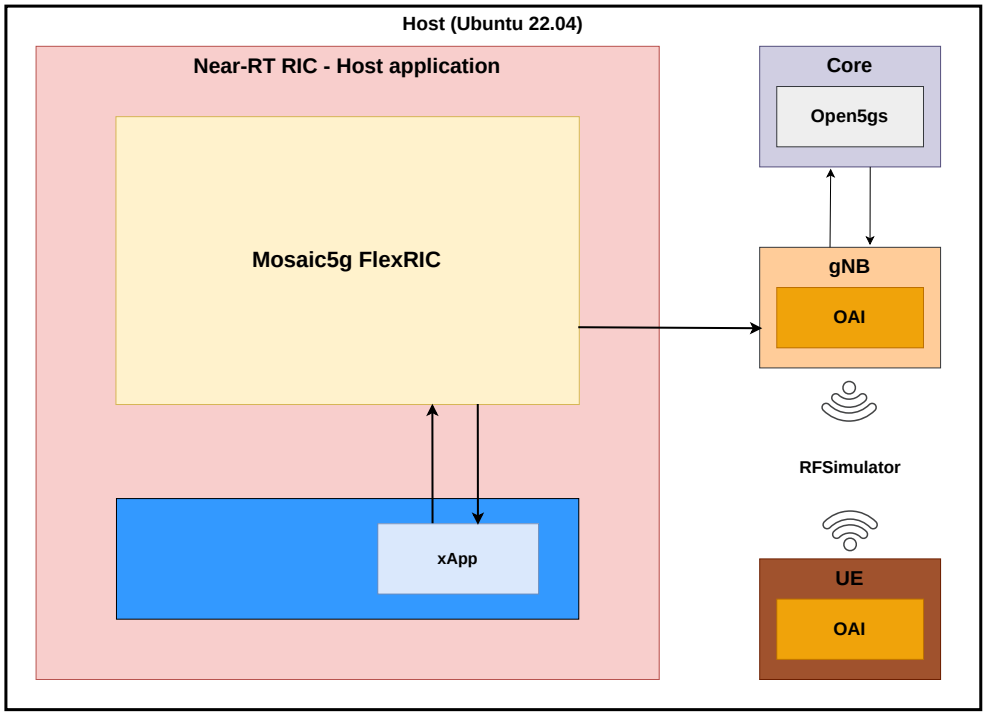

Figura 5.2: Arquitectura de la solución propuesta en la implementación 2. En el host se instala el Near-RT RIC FlexRIC de OAI. Además directamente sobre el host se instala el gNB y UE de OAI junto con el core de Open5gs.

Requisitos

Los desarrolladores indican los mismos requisitos mínimos necesarios que para la implementación 1.

Instalación

[19]https://gitlab.eurecom.fr/oai/openairinterface5g/-/blob/develop/radio/rfsimulator/README.md

Hasta la clonación del proyecto utilizando git los pasos son idénticos para la implementación 1 e implementación 2. Una vez clonado el proyecto, para la implementación 1 se trabajaba directamente sobre el directorio `O-RAN-Testbed-Automation`, en el caso de utilizar OAI, el directorio raíz será `O-RAN-Testbed-Automation/OpenAirInterface_Testbed`.

Por lo tanto para instalar:

```
$ cd O-RAN-Testbed-Automation/OpenAirInterface_Testbed
$ ./full_install.sh
```

La instalación descargará, compilará e instalará varios paquetes, el proceso completo puede demorar varios minutos. Una vez finalizada la instalación cada componente quedará pronto para usar en su respectivo directorio.

Utilización

Cada componente de la red tiene un directorio de trabajo, (`O-RAN-Testbed-Automation/ OpenAirInterface_Testbed/5G_Core_Network` para el *core*, `O-RAN-Testbed-Automation/OpenAirInterface_Testbed/Next_Generation_Node_B` para el gNB, `O-RAN-Testbed-Automation/ OpenAirInterface_Testbed/User_Equipment` para el UE) y `O-RAN-Testbed-Automation/OpenAirInterface_Testbed/RAN_Intelligent_Controllers/Flexible-RIC` para el Near-RT RIC. En todos los casos para iniciar los componentes se debe ejecutar el script *./run.sh* dentro de cada directorio siguiendo el siguiente orden: Core → gNB → UE → Near-RT RIC.

Esta solución permite ejecutar fácilmente varios UEs, simplemente se debe ejecutar el script *./run.sh* con el número de UE como parámetro, por ejemplo para ejecutar el UE 1:

```
$ ./run.sh
```

Para ejecutar el UE 2 se debe correr en otra terminal:

```
$ ./run.sh 2
```

y así con el resto de UEs.

Para cada UE se crea un *network namespace* llamado ue1, ue2, etc y para el *core* se crea la interfaz virtual *ogstun*, por lo tanto para simular tráfico entre UE y *core* primero se debe acceder al *bash* del *network namespace*:

```
$ sudo ip netns exec ue1 bash
```

y luego utilizando Iperf simular tráfico de *uplink*, *downlink* o ambos según se quiera.

Los componentes de la red guardan su salida en archivos de texto ubicados en la carpeta *logs* dentro de cada directorio. Otro directorio a destacar es el *configs* donde se almacena la configuración de cada elemento.

Como se comentó anteriormente, esta implementación utiliza la herramienta RFSimulator para simular el canal de aire. RFSimulator permite cargar modelos

de canal e incluso editarlo "en vivo", para todo esto hay que hacer algunas modificaciones en la configuración del UE y gNB. En el archivo *run.sh* del UE hay que editar la linea que lo ejecuta, la misma debe quedar:

```
sudo ip netns exec ue$UE_NUMBER ./nr-uesoftmodem -O
   "../../../../configs/ue$UE_NUMBER.conf" --rfsim --
   rfsimulator.serveraddr $HOSTNAME_IP --rfsimulator.
   options chanmod -r 106 --numerology 1 --band 78 -C
   3619200000
```

Mientras que el *run.sh* del gNB debe quedar:

```
sudo ./nr-softmodem -O "$SCRIPT_DIR/configs/gnb.conf" --
   rfsim --rfsimulator.options chanmod --gNBs.[0].
   min_rxtxtime 6
```

Además en los archivos de configuración del UE y gNB ubicado en el directorio *configs* se debe agregar al final inclusión al archivo que modela el canal:

```
@include "archivo_modelo_canal.conf"
```

Si se quiere editar el canal "en vivo" con la utilidad *telnetsrv* se debe agregar al final de la linea que ejecuta el gNB en el archivo *run.sh* las siguientes opciones:

```
--telnetsrv --telnetsrv.listenport 7000
```

luego acceder a la consola de RFSimulator con *telnet*:

```
telnet 127.0.0.1 7000
```

Puede pasar que la herramienta *telnetsrv* no esté compilada, por lo tanto previo a su utilización se debe realizar el siguiente procedimiento:

```
$ cd O-RAN-Testbed-Automation/OpenAirInterface_Testbed/
   Next_Generation_Node_B/openairinterface5g
$ source oaienv
$ cd cmake_targets
$ ./build_oai  --build-lib telnetsrv
```

Al igual que la implementación 1 esta implementación también trae algunas xApps predefinidas, a continuación se realiza una breve revisión de las mismas[20].

- **KPM Monitor xApp**:

  Funciona de forma similar a la xApp de monitoreo de la implementación 1 (se suscribe a métricas de los nodos E2 utilizando el modelo de servicio E2SM-KPM). Viene en tres versiones, una muestra las KPI en pantalla, otra las almacena en un archivo CSV y la última los guarda en una base de datos de InfluxDB.

[20]https://github.com/usnistgov/O-RAN-Testbed-Automation/tree/main/OpenAirInterface_Testbed/RAN_Intelligent_Controllers/Flexible-RIC

- **MAC + RLC + PDCP + GTP Monitor xApp**:

  Este xApp extiende la funcionalidad de monitoreo para obtener métricas más profundas de las capas MAC, RLC, PDCP y GTP del protocolo de datos. Usa modelos personalizados dentro de FlexRIC (codificación plana, no ASN.1) para capturar estadísticas como colas, retransmisiones, número de PDUs o paquetes descartados, volumen de tráfico, latencias internas, etc.

- **RIC Control xApp**:

  Al igual que en la implementación 1 es la xApp encargada de enviarle comandos a los nodos E2 para modificar su comportamiento dinámicamente, utilizando el modelo de servicio E2SM-RC.

- **RIC Control Monitor xApp**:

  Complementa al xApp de control con funciones de supervisión: monitorea las acciones que la xApp de control ejecuta, registra si fueron exitosas o no, y analiza el impacto de esas acciones en las métricas de la RAN. De ese modo se obtiene retroalimentación para evaluar los efectos del control, detectar fallas o ajustar políticas de control.

Conclusiones

Al igual que la implementación 1, este *testbed* fue sencillo de instalar y poner en marcha. Solo hubo un problema de conectividad interno (la conexión entre UE y *core* no pasaba físicamente por el gNB) que fue rápidamente resuelto por los desarrolladores a través de un *issue*[21] que se planteó. OAI es pionero en el desarrollo de este tipo de herramientas y al igual que srsRAN esta muy documentado y probado. La novedad en esta implementación es que no se utiliza el Near-RT RIC oficial de la OSC, si no que un desarrollo propio basado en el SDK FlexRIC. Esto por un lado tiene ciertas ventajas, según se demostró en [47] este Near-RT RIC es mas liviano y eficiente que el de la OSC, particularmente los desarrolladores de FlexRIC destacan el Near-RT RIC de OSC no llega a cumplir con los requerimientos temporales para aplicaciones de baja latencia mientras que el suyo sí. Además la herramienta permite adaptar el Near-RT RIC a las necesidades puntuales aunque tiene el costo asociado a la curva de aprendizaje necesaria para su correcta utilización. Otro punto a destacar es que como el desarrollo de OAI y FlexRIC están bajo la misma tutela se entiende que hay mayor coordinación que evite desfasajes y perdidas de compatibilidad. Hablando de las desventajas, la principal es que es un desarrollo independiente de la OSC, que es la entidad que marca el rumbo de O-RAN, dicho de forma más coloquial, no es el oficial.

[21] https://github.com/usnistgov/O-RAN-Testbed-Automation/issues/3

5.3. Implementación 3

Esta implementación está basada en la plataforma ns-O-RAN[22] que permite conectar una simulación de ns3 con un Near-RT RIC real. El detalle de la plataforma y algunos experimentos realizados con la misma se pueden ver en el artículo mencionado en el capitulo anterior [25].

Arquitectura

El sistema se compone de varios elementos independientes que trabajan en conjunto para lograr que redes móviles simuladas en ns3 interactúen con el Near-RT RIC real.

El primer componente es el mismo ns3, en este caso se utiliza la versión 3.36 (a junio de 2025 la última versión disponible es la 3.45). A ns3 se le instala el módulo *ns-o-ran-ns3-mmwave*[23] que es un *fork* del módulo *ns3-mmwave*[24]. Este último es un módulo conocido que permite simular redes 5G, el *fork* agrega las modificaciones necesarias para hacerlo compatible con O-RAN. Finalmente hay que instalar en ns3 el módulo *ns-O-RAN*[25] que extiende a *ns-o-ran-ns3-mmwave* para habilitar la terminación de múltiples interfaces E2 en los elementos de la simulación.

Por otro lado es necesario instalar la aplicación *o-ran-e2sim*[26] (que es un *fork* del proyecto *sim-e2-interface*[27] de la OSC). La función de esta aplicación es servidor como terminación E2 de la simulación y punto de conexión directo con el Near-RT RIC.

Finalmente se tiene al Near-RT RIC denominado por los autores como *Colosseum Near-Real-Time RIC*[28] o *ColORAN* [42] que es una versión reducida del Near-RT RIC *release bronze* de la OSC. Haciendo una analogía con la implementación 1 donde se despliega la última versión del RIC de la OSC, en este RIC solamente se implementan las siguiente funciones:

- **rtmgr**: encargado del enrutamiento de mensajes entre servicios utilizado el protocolo *RIC Message Router* (RMR).
- **e2mgr**: administra las conexiones E2 con los nodos de la RAN.
- **e2term**: punto de terminación de la interfaz E2, es a donde se conectan los nodos E2.
- **dbaas**: base de datos para almacenar información de configuración y estado.

[22] https://openrangym.com/ran-frameworks/ns-o-ran
[23] https://github.com/wineslab/ns-o-ran-ns3-mmwave
[24] https://github.com/nyuwireless-unipd/ns3-mmwave
[25] https://github.com/o-ran-sc/sim-ns3-o-ran-e2
[26] https://github.com/wineslab/o-ran-e2sim
[27] https://github.com/o-ran-sc/sim-e2-interface
[28] https://github.com/wineslab/colosseum-near-rt-ric

Al igual que en la implementación 1 las xApps corren como contenedores. En la figura 5.3 se muestra un diagrama de la arquitectura de esta solución.

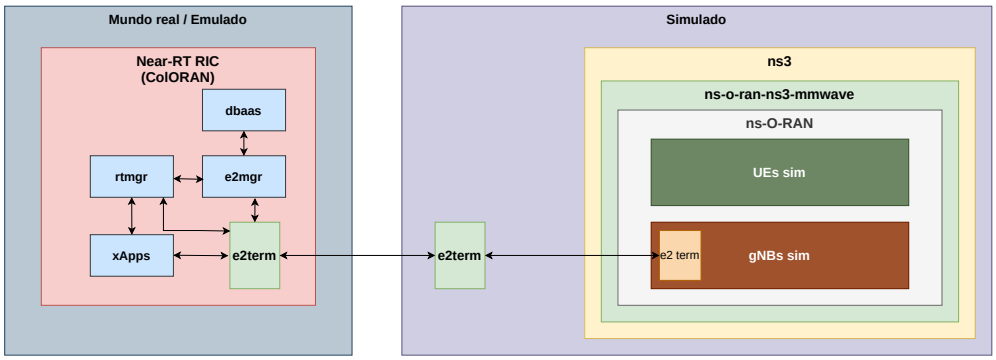

Figura 5.3: Arquitectura de la solución propuesta en la implementación 3 con un Near-RT RIC que corre en el mundo real y la RAN es enteramente simulada en ns3.

Requisitos

En ninguno de los documentos asociados a esta implementación dan requerimientos de hardware ni se explicitan las características técnicas asociadas a los equipos utilizados para las pruebas. En este trabajo se utilizó una maquina virtual de iguales características a las mencionadas en las implementaciones 1 y 2.

Instalación

Partiendo de una instalación limpia de Ubuntu 22.04 y elegido un directorio de trabajo que aquí se llamará `implementacion3` se deben instalar algunas dependencias:

```
$ apt-get install git
$ snap install docker
$ apt-get install -y build-essential git cmake libsctp-dev
   autoconf automake libtool bison flex libboost-all-dev
$ apt-get install g++ python3
```

Luego se debe proceder a descargar e instalar el Near-RT RIC *ColORAN*:

```
$ git clone -b ns-o-ran https://github.com/wineslab/
   colosseum-near-rt-ric
$ cd colosseum-near-rt-ric/setup-scripts
$ ./import-wines-images.sh
```

El siguiente paso es descargar y compilar *o-ran-e2sim*. Por como está configurado es necesario cambiar el nombre del directorio de destino a `oran-e2sim`:

```
$ git clone https://github.com/wineslab/ns-o-ran-e2-sim
   oran-e2sim
```

```
$ cd oran-e2sim/e2sim/
$ mkdir build
$ ./build_e2sim.sh 3
```

Finalmente queda descargar ns3 junto con los complementos necesarios y compilarlos:

```
$ git clone https://github.com/wineslab/ns-o-ran-ns3-mmwave
    ns-3-mmwave-oran
$ cd ns-3-mmwave-oran/contrib
$ git clone -b master https://github.com/o-ran-sc/sim-ns3-o
   -ran-e2 oran-interface
$ cd ..
$ ./ns3 configure --enable-examples --enable-tests
$ ./ns3 build
```

Una vez terminado todo el proceso, dentro del directorio `implementacion3` se deberán ver tres directorios: `colosseum-near-rt-ric`, `oran-e2sim` y `ns-3-mmwave-oran`.

Utilización

El primer paso para poner en funcionamiento el testbed es iniciar el Near-RT RIC, para esto hay que dirigirse al directorio `colosseum-near-rt-ric/setup-scripts` y ejecutar el script *start-ric-bronze.sh*:

```
$ ./start-ric-bronze.sh
```

Una vez que termina la ejecución se puede verificar que todos los contenedores estén funcionando usando el comando:

```
$ docker ps
```

Para ver los logs de los contenedores se utiliza el comando:

```
$ docker logs NOMBRE_CONTENEDOR -f
```

Para abrir una terminal dentro del contenedor se usa:

```
$ docker exec -it NOMBRE_CONTENEDOR -f
```

Vale aclarar que todos los comandos de *docker* son aplicables.

En caso de querer correr una xApp se debe iniciar en este momento, previo al inicio de la simulación. Como ya se mencionó las xApp en el RIC de la OSC funcionan dentro de contenedores.

Las simulaciones se ejecutan dentro del directorio `ns-3-mmwave-oran` con el comando:

```
$ ./ns3 run ARCHIVO_SIMULACION
```

En este trabajo no se va a explicar como se programa una simulación en ns3, pero si se comentará a continuación los parámetros que agrega *ns-O-RAN* a *mmwaves* para hacer que la red móvil clásica sea compatible con 5G. En breves palabras para hacer una simulación en ns3 se deben definir los elementos (UEs, radio bases, *core*, etc), configurarles parámetros de funcionamiento (configuración de red, modulación, *bitrate*, etc) y definir su comportamiento (si se están moviendo, como se mueven, como trafican datos, etc).

Para que la simulación sea compatible con O-RAN, o sea se conecte con el Near-RT RIC y envíe datos a través de la interfaz E2 se deben configurar los siguientes parámetros en los eNB y gNB:

- **g_e2lteEnabled (boolean)**: habilita la interfaz E2 en el eNB para enviar métricas KPM al RIC y recibir instrucciones de control.
- **g_e2nrEnabled (boolean)**: habilita la interfaz E2 en el gNB para enviar métricas KPM al RIC y recibir instrucciones de control.
- **g_e2du (boolean)**: activa el envío de métricas desde el DU (métricas asociadas a la capas RLC, MAC y PHY).
- **g_e2cuUp (boolean)**: activa el envío de métricas desde el plano de usuario del CU (métricas asociadas a la capa PDCP).
- **g_e2cuCp (boolean)**: Activa el envío de métricas desde el plano de control del CU.
- **g_reducedPmValues (boolean)**: limita la cantidad de datos que se envían desde los nodos E2 al RIC, útil para pruebas.
- **g_e2TermIp (string)**: dirección IP del Near-RT RIC, particularmente de la terminación E2. En el caso del Near-RT RIC utilizado en esta implementación es la IP del contenedor e2term (10.0.2.10 por defecto).
- **g_enableE2FileLogging (boolean)**: hace que todas las métricas recolectadas sean enviada a un archivo en vez de al Near-RT RIC, útil cuando no se tiene RIC o para testing.
- **g_indicationPeriodicity (double)**: define cada cuanto tiempo se envían los reportes E2, o sea cada cuanto se mandan los mensajes *E2 Indication* al Near-RT RIC.

Conclusiones

La idea y la solución dada en de esta implementación es novedosa y expande las posibilidades de experimentación con O-RAN ya que permite simular entornos y condiciones que en ambientes reales puede ser muy complicado. Depende del punto de vista si la mezcla de Near-RT RIC real con la RAN simulada es una oportunidad o una desventaja. Si se piensa solo en realizar simulaciones se entiende que no es la mejor opción ya que aunque los autores resuelven los temas de sincronización

temporal (y otros) se deben tener en cuenta ambos ambientes a la hora de hacer los experimentos. Ahora, si se quiere expandir un experimento real es sumamente útil, ya que se puede tener un despliegue real (similar a las implementaciones 1 y 2) junto con otro simulado utilizando el mismo Near-RT RIC. Seguramente haya que tener algunas consideraciones pero en base a lo investigado es posible.

Otra desventaja considerable es que se utilizan versiones del Near-RT RIC y ns3 antiguas y por el momento no hay posibilidad de actualizarlas, lo que hace que el sistema sea muy rígido y poco adaptable, por otro lado la instalación y configuración fue sencilla.

5.4. Implementación 4

La implementación 4 está basada en el módulo de ns3 *ORAN ns3* del NIST (artículo [17]) que a diferencia de la implementación 3 que simula parcialmente la arquitectura de O-RAN, permite realizar experimentos 100 % simulados de O-RAN.

Arquitectura

La arquitectura de la solución se puede ver desde dos puntos de vista, uno más "físico" donde se asocia la implementación con los elementos de la arquitectura de O-RAN y otro más a nivel de programación donde se ve como se implementa el sistema desde el punto de vista del código.

Comenzado por al arquitectura en un sentido más "físico", esta implementación tiene un Near-RT RIC que internamente se compone de los siguientes módulos (se utilizarán los nombres en inglés para mantener la coherencia con la documentación extraída del repositorio[29]):

- **Logic Module**: son los módulos que analizan los datos y toman acciones en base a los mismos, es el nombre que le dan a las xApp. El sistema necesita que haya al menos una xApp corriendo.
- **Data Repository**: se utiliza para unificar el mecanismo de almacenamiento de información, o sea dónde y cómo se guardan los datos. Además maneja la lectura de dichos datos por parte de otros módulos.
- **Conflict Mitigation**: este módulo recibo los comandos a ejecutar en los nodos E2 desde los *Logic Modules* y analiza la conveniencia y pertinencia de su ejecución en base a la lógica programada.
- **Query Triggers**: es donde se determinan las condiciones para invocar a un *Logic Modules* (por tiempo, por un evento, etc).
- **E2 RIC Terminator**: es el punto de conexión con los nodos E2, recibe los reportes y envía los comandos.

[29]https://github.com/usnistgov/ns3-oran

Por otro lado cada nodo E2, aparte de la implementación que hace a su función (UE, gNB, etc) tiene los siguientes módulos:

- **E2 Node Terminator**: comunica el nodo con el Near-RT RIC, recibe comandos y envia reportes.
- **Reporter**: observan valores del nodo y generan reportes, por ejemplo la posición o la intensidad de señal.
- **Report Trigger**: en este módulo se define cada cuánto o bajo qué evento se deben recopilar los datos.

En la figura 5.4 extraída directamente de la documentación de los desarrolladores se observa el esquema de la arquitectura descrita.

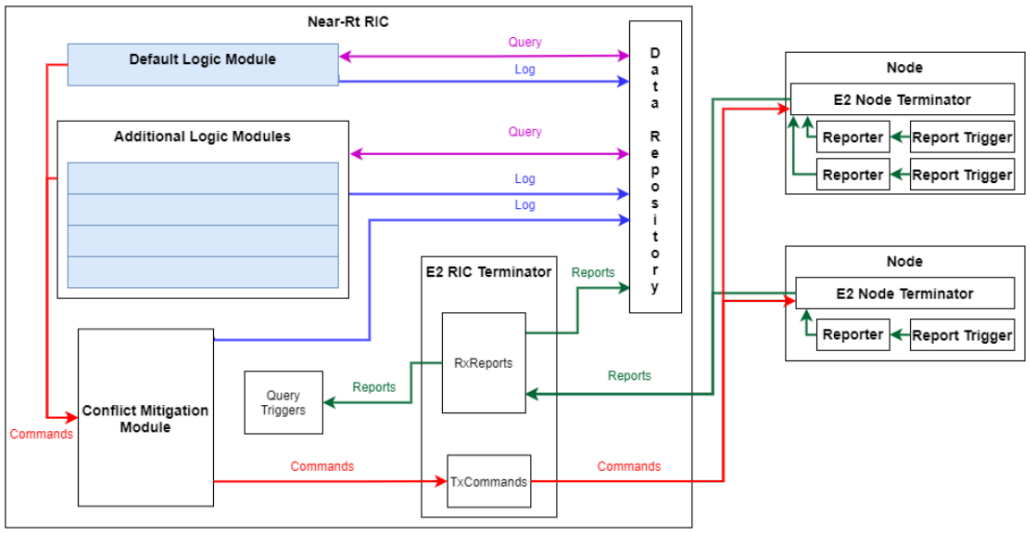

Figura 5.4: Arquitectura de la solución propuesta en la implementación 4. Se muestra el Near-RT RIC con sus módulos y como se comunican internamente, asi como los nodos E2 y su conexión con el Near-RT RIC.

Bajo la óptica del código, cada elemento mencionado se define en una clase (de C++), o sea que cada elemento termina siendo un objeto. El objeto Near-RT RIC se encarga de interconectar a los objetos que representan sus módulos (*Logic Module*, *Data Repository*, *Conflict Mitigation*, *Query Triggers* y *E2 RIC Terminator*). De la misma forma el objeto que representa al nodo E2 conecta entre si sus módulos (*E2 Node Terminator*, *Reporter* y *Report Trigger*).

Otro aspecto a destacar es que todas las clases base (menos la del Near-RT RIC, Nodo E2 y *E2 RIC Terminator*) le pueden heredar a clases secundarias que modifiquen o personalicen el funcionamiento. Para desarrollar una xApp, el código de la xApp debe heredar de la clase *Logic Module* y extender su funcionamiento con la lógica deseada, para definir la lógica del módulo de mitigación de conflictos se debe heredar de la clase del *Conflict Mitigation* y así con el resto de elementos.

Los elementos de información (tipos de mensajes que se pueden intercambiar) también son clases base. Existe dos tipos, *Report* y *Command*, entonces para generar un mensaje propio se debe heredar de estas.

También es importante mencionar que los desarrolladores proveen un *Helper* que facilita la instanciación, configuración y conexión de los elementos.

Por último es de destacar que ORAN ns-3 soporta *Machine Learning* (ML) de forma opcional, modular y acoplada a herramientas externas como ONNX[30] y PyTorch[31]. Gracias a esto se pueden desarrollar un *Logic Module* (xApp) con inferencia de ML, aunque hay que aclarar que el entrenamiento se debe hacer fuera de ns3.

Requisitos

En ninguno de los documentos asociados a esta implementación dan requerimientos de hardware ni se explicitan las características técnicas asociadas a los equipos utilizados para las pruebas. En este trabajo se utilizó una maquina virtual de iguales características a las mencionadas en las implementaciones 1 y 2.

Instalación

Como en el resto de las implementaciones se parte de una instalación limpia de Ubuntu 22.04, lo primero es instalar algunas dependencias y herramientas necesarias:

```
$ apt-get install cmake build-essential git libgsl-dev
   libgtk-3-dev libharfbuzz-dev libxml2 libxml2-dev
   libsqlite3-dev libeigen3-dev python3-dev python3-
   pygraphviz python3-pybind11 graphviz
```

Luego en el directorio de trabajo clonar el instalador de ns3 y moverse al directorio:

```
$ git clone https://gitlab.com/nsnam/ns-3-allinone.git
$ cd ns-3-allinone
```

Descargar la versión de ns3 deseada, en este caso la 3.42.

```
$ python3 download.py -n ns-3.42
```

Para terminar con la instalación de ns3 ejecutar los siguientes comandos:

```
$ cd ns-3.42
$ ./ns3 clean
$ ./ns3 configure --build-profile=debug --enable-examples
   --enable-tests
$ ./ns3 build
```

[30] https://onnx.ai/
[31] https://pytorch.org/

Se puede probar que todo esta correcto con el comando:

```
$ ./ns3 run hello-simulator
```

Luego de instalado ns3 resta agregar el módulo ORAN ns3:

```
$ cd contrib
$ git clone https://github.com/usnistgov/ns3-oran.git oran
$ cd ..
$ ./ns3 configure --enable-examples
$ ./ns3
```

Utilización

Al ser un ambiente completamente simulado, la utilización se resume a escribir el código de la simulación y correrla. Cada simulación va a depender de lo que el desarrollador necesite que haga. A continuación se realiza un punteo sobre los conceptos y elementos claves que deben estar en el código sin profundizar en una simulación particular en si.

El primer paso es incluir en el proyecto el módulo ORAN ns3, si se utiliza *cmake* esto se hace a través del archivo `CMakeLists.txt` agregando las siguientes lineas:

```
set(libraries_to_link "${libcore};${libinternet};${
   libmobility}")

if("oran" IN_LIST ns3-all-enabled-modules)
  list(APPEND libraries_to_link ${liboran})
  add_definitions(-DENABLE_ORAN)
endif()

build_lib_example(
  NAME my-openran-example
  SOURCE_FILES my-openran-example.cc
  LIBRARIES_TO_LINK
    ${libraries_to_link}
)
```

Lo anterior además de revisar si el módulo existe crea un macro para utilizar en el código y verificar la existencia de ORAN ns3. Luego ya en el archivo de la simulación hay que importar la librería del módulo:

```
#include <ns3/oran-module.h>
```

y comenzar a crear y configurar los objetos que cumplirán las funciones ya vistas. Para crear el Near-RT RIC:

```
Ptr<OranNearRtRic> nearRtRic = CreateObject<OranNearRtRic
   >();
```

Para crear el *Data Repository* de tipo SQLite[32] y configurar el archivo donde se guardará la información:

```
Ptr<OranDataRepository> dataRepository = CreateObject<
   OranDataRepositorySqlite>();
dataRepository->SetAttribute("DatabaseFile", StringValue("
   results.db"));
```

Para crear el objeto que represente el *Default Logic Module*:

```
Ptr<OranLm> defaultLm = CreateObject<
   OranLmLte2LteDistanceHandover>();
defaultLm->SetAttribute("NearRtRic", PointerValue(nearRtRic
   ));
```

Notar en el código anterior que el objeto se crea del tipo `OranLmLte2LteDistanceHandover` que es una clase que hereda de `OranLm`. Particularmente el tipo `OranLmLte2LteDistanceHandover` es una clase creada por los desarrolladores para implementar un tipo especifico de xApp, la idea es que cada uno desarrollo su clase en base a las necesidades.

Para instanciar el *Conflict Mitigation*:

```
Ptr<OranCmm> cmm = CreateObject<OranCmmSingleCommandPerNode
   >();
cmm->SetAttribute("NearRtRic", PointerValue(nearRtRic));
```

Nuevamente aquí se tiene que el objeto del *Conflict Mitigation* es de un tipo particular (`OranCmmSingleCommandPerNode`) que describe como resolver los conflictos a la hora de ejecutar los comandos, perfectamente se podría crear otro mecanismo con otra lógica para resolver estos conflictos.

Para crear el *E2 RIC Terminator*:

```
Ptr<OranNearRtRicE2Terminator> ricE2Terminator =
   CreateObject<OranNearRtRicE2Terminator>();
ricE2Terminator->SetAttribute("NearRtRic", PointerValue(
   nearRtRic));
ricE2Terminator->SetAttribute("DataRepository",
   PointerValue(dataRepository));
```

Para terminar de configurar el Near-RT RIC queda conectar todos los componentes a través del objeto que representa el Near-RT RIC:

```
nearRtRic->SetAttribute("DefaultLogicModule", PointerValue(
   defaultLm));
nearRtRic->SetAttribute("ConflictMitigationModule",
   PointerValue(cmm));
nearRtRic->SetAttribute("E2Terminator", PointerValue(
   ricE2Terminator));
```

[32]https://www.sqlite.org/

```
nearRtRic->SetAttribute("DataRepository", PointerValue(
    dataRepository));
```

Teniendo configurado el Near-RT RIC resta crear los nodos de red, crear los módulos de ORAN ns3 y enlazarlas. Por ejemplo por un lado se crea un *eNB*:

```
NodeContainer enbNodes;
enbNodes.Create(1);  // Crea 1 eNB (o gNB)
Ptr<Node> enbNode = enbNodes.Get(0);  // Este es tu '
    enbNode'
...
... configuración del eNB ...
...
```

Por otro lado se crea el *E2 Node Terminator* para nodos LTE:

```
Ptr<OranE2NodeTerminatorLteEnb> enbTerminator =
    CreateObject<OranE2NodeTerminatorLteEnb>();
enbTerminator->SetAttribute("NearRtRic", PointerValue(
    nearRtRic));
```

Y para que tenga sentido hay que crear al menos un *Reporter*:

```
Ptr<OranReporterLocation> locationReporter = CreateObject<
    OranReporterLocation>();
locationReporter->SetAttribute("Terminator", PointerValue(
    enbTerminator));
enbTerminator->AddReporter(locationReporter);
```

En este caso el *Reporter* es de tipo `OranReporterLocation` que viene predefinido con la solución, pero el programador puede crear el suyo.

Por último hay que agregar la configuración de ORAN ns3 al nodo creado, esto se hace de la siguiente forma:

```
enbTerminator->Attach(enbNode);
```

Una vez configurada toda la red O-RAN hay que "prender" el Near-RT RIC y activar la terminación E2 de los nodos:

```
Simulator::Schedule(Seconds(1), &OranNearRtRic::Start,
    nearRtRic);
Simulator::Schedule(Seconds(2), &OranE2NodeTerminatorLteUe
    ::Activate, ueTerminator);
```

Entre esta parte del código y el lanzamiento de la simulación (`Simulator::Run();`) es que se programa la simulación en si (eventos, tráfico, movimiento, etc).

En la documentación[33] se profundiza sobre como extender las clases base para crear *Logic Module*, *Data Repository*, *Conflict Mitigation*, *Query Triggers*, *E2 Node*

[33]https://github.com/usnistgov/ns3-oran/tree/master/doc

Terminator, *Reporter* y *Report Trigger* propios. No se agrega en esta documentación ya que se entiende que está bien explicado en el repositorio original y no aporta a entender el funcionamiento y arquitectura del sistema.

Respecto a las xApps (*Logical Modules*) incluidos, este *testbed* trae varios por defecto, la mayoría de ellos asociados al *handover*. Es importante aclarar que en este caso las xApps están asociadas a escenarios y no se presentan de forma aislada. A continuación se listan[34][35]:

- **Random Walk Example**:

  El Random Walk Example implementa un escenario de simulación minimalista orientado a demostrar la interacción entre nodos de ns-3 y el Near-RT RIC en un entorno O-RAN. En este ejemplo, los nodos siguen un patrón de movimiento aleatorio y reportan su posición periódicamente, permitiendo la recolección de datos sin procesamiento lógico ni emisión de comandos. La configuración se realiza mediante *OranHelper* que facilita la creación de un repositorio de datos con *backend* SQLite y la "instanciación" de módulos *No Operation* tanto para lógica y para la mitigación de conflictos, garantizando un comportamiento pasivo del sistema. Asimismo, se configura el Node E2 Terminator con un *Location Reporter* y un *periodic Report Trigger*, permitiendo el monitoreo continuo de la movilidad de los nodos. Este ejemplo resulta útil para evaluar el flujo de datos hacia el RIC y el funcionamiento básico de los componentes de la arquitectura sin introducir complejidad adicional.

- **LTE to LTE Distance Handover Example**:

  Presenta un escenario de simulación más complejo que ilustra la configuración manual de los modelos O-RAN sin recurrir al *OranHelper* ofreciendo mayor control sobre cada componente. La topología incluye dos eNBs y un UE que se desplaza de forma bidireccional entre ambas radio bases. Inicialmente, el UE está asociado al eNB más cercano, sin embargo, cuando se aproxima a la otra celda el *Logic Module* emite un *handover Command* transfiriendo la conexión al nuevo eNB. Además el método *ReverseVelocity* invierte periódicamente la dirección de movimiento del UE, permitiendo generar múltiples eventos de *handover* durante la simulación. Centrandose en la xApp esta recibe la posición del UE y en base a la distancia geométrica con los eNB decide si hace o no el *handover*.

- **LTE to LTE Handover With Helper Example**:

  Reproduce funcionalmente el mismo escenario que el ejemplo anterior, pero aprovecha el uso de *OranHelper* para simplificar la configuración y despliegue de los modelos de la arquitectura O-RAN.

[34]https://github.com/usnistgov/ns3-oran?tab=readme-ov-file#running-the-examples

[35]O-RAN Module Release oran-1.3 National Institute of Standards and Technology (NIST)

- **LTE to LTE Handover With LM Processing Delay Example**:

  Mantiene la misma topología y lógica funcional que el ejemplo con *OranHelper*, pero introduce un elemento adicional de realismo al configurar un retardo de procesamiento en el *logic module*. Este retardo se modela mediante una distribución normal. Este ejemplo resulta particularmente valioso para estudiar el impacto de latencias de procesamiento en la toma de decisiones de *handover* y su efecto en el desempeño global de la red.

- **LTE to LTE Handover With LM Query Trigger Example**:

  Amplía el escenario previo al incorporar un mecanismo de activación personalizada en el Near-RT RIC, diseñado para iniciar el proceso de consulta al *logic module* en el momento en que se reciben reportes que cumplen con determinadas condiciones. Esta lógica permite detectar de manera temprana situaciones en las que un *handover* podría ser necesario, como cuando el usuario se aproxima a una celda vecina y desencadenar de inmediato la toma de decisiones. De esta forma, el ejemplo ilustra cómo los disparadores personalizados pueden mejorar la capacidad de reacción del sistema y optimizar el tiempo de respuesta, contribuyendo a una gestión de movilidad más eficiente en redes LTE.

- **Keep Alive Example**:

  Muestra el funcionamiento del mecanismo de *keep alive* en entornos O-RAN. utilizando un único nodo que se desplaza en línea recta y reporta periódicamente su posición al Near-RT RIC. A pesar de estos reportes, el nodo es marcado con frecuencia como inactivo debido a la configuración del intervalo de envío de mensajes de registro, lo que provoca su "desregistración" temporal. El escenario está diseñado con parámetros de tiempo cuidadosamente ajustados para evidenciar el proceso de supervisión de actividad de los nodos, mostrando cómo la frecuencia de las verificaciones en el RIC y el intervalo de registros del nodo afectan su estado de conexión. De esta manera, el ejemplo permite analizar el impacto de los tiempos de registro y detección de inactividad sobre la estabilidad de la comunicación, resaltando consideraciones importantes para el diseño de redes donde la latencia y la confiabilidad son críticas.

- **Data Repository Example**:

  Ejemplo de uso de la API de repositorio de datos para almacenar y recuperar información en el contexto de una simulación O-RAN.

- **LTE to LTE ML Handover Example**:

  Explora el uso de modelos de aprendizaje automático a través de ONNX o PyTorch, para la toma de decisiones de *handover* en entornos LTE. El escenario incluye cuatro UEs y dos eNBs, con dos de los UEs permaneciendo dentro de la cobertura exclusiva de cada celda y los otros dos desplazándose en la zona de solapamiento. Durante la simulación, se recopilan datos de distancia y de pérdida de paquetes de todos los UEs, los cuales son procesados

por un modelo de *machine learning* que determina la asignación óptima de celdas para minimizar la pérdida de paquetes en el sistema. Este enfoque permite analizar el potencial de las técnicas de aprendizaje automático para mejorar la eficiencia de la gestión de movilidad en tiempo real. Además, el ejemplo incluye herramientas para la generación de datos de entrenamiento y la creación de modelos personalizados, facilitando el estudio de soluciones basadas en inteligencia artificial para la optimización de redes LTE.

Conclusiones

Esta solución 100 % simulada permite realizar casi cualquier tipo de experimento en el ámbito de O-RAN dentro de ns3. Es una herramienta sumamente útil y personalizable que da el marco para experimentar con O-RAN, particularmente con el desarrollo de algoritmos para las xApps en entornos que serían complejos de tener en el mundo real. Al momento de escribir este documento el proyecto se encuentra mantenido por sus autores y es compatible con la ante-penúltima versión de ns3. Instalarla y utilizar los códigos de ejemplo que dan los desarrolladores es relativamente sencillo, aunque hay que recalcar que si se desconoce de ns3 y C++ su utilización se ve muy limitada y la curva de aprendizaje es grande.

Tal vez la principal desventaja es que se apoya en el módulo LTE de mmWave[36] . Esto quiere decir que la red móvil simulada solo puede ser 4G y no es compatible con redes 5G. Estudiantes de la Facultad de Ingeniería de la UdelaR [10] que están trabajando con esta implementación consultaron a los desarrolladores[37] sobre este tema, y la respuesta fue que no tenían planes de adaptarlo a 5G, ya sea utilizando el mismo mmWave o 5G Lena[38] . Esta situación los motivó a analizar la viabilidad de realizar dicha adaptación, trabajo que, a diciembre de 2025, se encuentran desarrollando.

5.5. Implementación 5

La última implementación que se verá en este trabajo busca emular de la forma más aproximada posible la realidad dando el suficiente margen de maniobra para poder adaptar el *testbed* a los recursos disponibles. Originalmente esta implementación se iba a basar fuertemente en el trabajo del NIST [28] ya que en la implementación 1 y 2 demostró funcionar correctamente el *Automation Tool* y en el documento describen como desplegar el *testbed* variando la desagregación de los elementos. Al momento de realizar el despliegue surgieron dos inconvenientes principales: 1) no se utilizan las últimas versiones de los sistemas involucrados; 2) se encontraron inconsistencias al variar las versiones respecto con los resultados esperados. Por este motivo, aunque se utiliza como fuente el trabajo mencionado,

[36] https://github.com/nyuwireless-unipd/ns3-mmwave
[37] https://github.com/usnistgov/ns3-oran/issues/12
[38] https://5g-lena.cttc.es

se sumó la documentación de srsRAN[39] y el trabajo realizado en la Facultad de Ingeniería de la Universidad de la República titulado PortaRAN [41].

Arquitectura

Como se mencionó la idea principal de esta implementación es tener un sistema emulado y flexible a la hora de desplegar. La emulación se logra utilizado el *core* de Open5GS y el gNB de srsRAN. El UE puede ser emulado utilizando el de srsRAN o real usando un COTS UE. Como Near-RT RIC es posible utilizar el *release-i* de la O-RAN SC o un *branch* específico de FlexRIC. Si se habla de las interfaces, la comunicación entre gNB y RIC, y gNB y *Core* puede ser interna si corre todo en un mismo equipo o por red (Ethernet o Wifi) si corren en equipos distintos. La interfaz entre el UE y el gNB depende de que tipo de UE se utilice, en caso de usar el UE de srsRAN se puede simular el canal con ZMQ o a través de SDRs (uno en el UE y otro en el gNB) se puede usar el canal real. En caso de usar un UE COTS se debe conectar el gNB a un SDR y se utiliza el aire como canal. Finalmente y relacionado con lo anterior también hay flexibilidad en como *hostear* los diferentes elementos, puede correr todo en un mismo servidor, o se puede dividir en varios. En la figura 5.5 se resumen todas estas posibilidades.

No se profundiza en la descripción de cada solución ya que todas fueron previamente explicadas.

Requisitos

Para definir los requisitos primero hay que tener claro que tipo de desligue se quiere realizar ya que puede variar de un solo equipo o maquina virtual (desligue mas sencillo) hasta 3 o 4 equipos con SDRs y dispositivos COTS en los casos más complejos. También varía según el tamaño del escenario, sobre todo según la cantidad de UEs a emular.

Para tener como referencia, en una computadora de escritorio con un CPU Intel Core i7-4790 de 4 núcleos y 32 GB de RAM se logró correr un escenario completo utilizando un UE emulado y ZMQ en un experimento, y un UE COTS con SDR en el gNB en otro experimento.

En todos los casos el sismtema operativo recomendado es una instalación limpia y sin actualizar de Ubuntu Desktop 22.04.5 LTS[40].

Instalación

Al ser un *testbed* desagregado todos los elementos deben instalarse por separado. El punto de partida para todos los componentes es el mismo, una instalación limpia de Ubuntu Desktop 22.04.5 LTS con las actualizaciones automáticas desactivadas.

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

[40] https://releases.ubuntu.com/jammy/

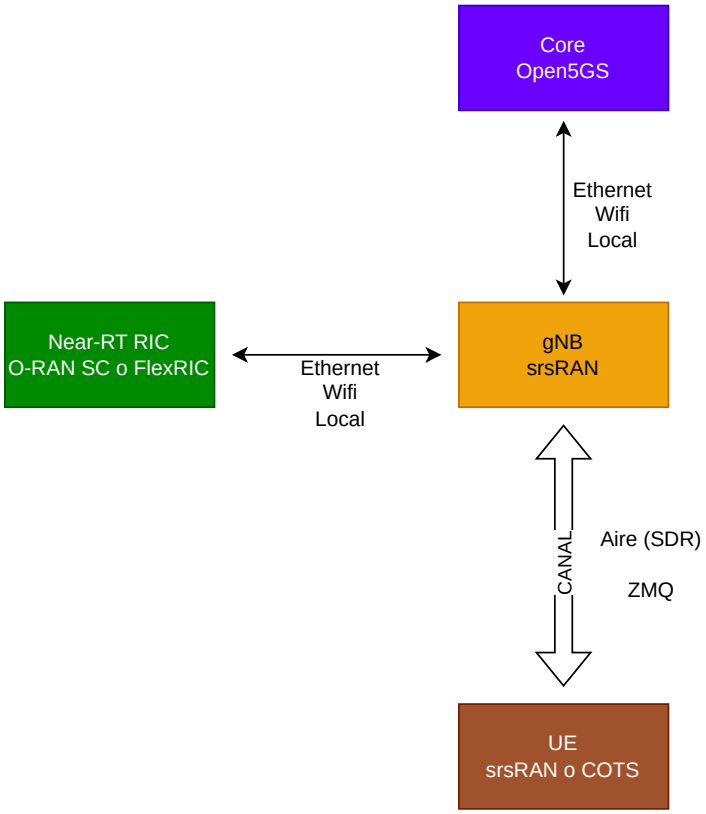

Figura 5.5: Arquitectura de la solución propuesta en la implementación 5. Esta solución no solo permite tener mas de una opción para el Near-RT RIC y el UE si no que permite desplegar cada elemento de forma independiente pudiendo estar todos en un mismo equipo, cada uno en un equipo independiente o agrupados de cualquier forma posible.

gNB

Se comienza instalando las dependencias:

```
$ sudo apt-get install cmake make gcc g++ pkg-config
   libfftw3-dev libmbedtls-dev libsctp-dev libyaml-cpp-dev
   git
```

Luego se instala el el diver del SDR y la librería ZMQ:

```
$ sudo add-apt-repository -y "deb http://uy.archive.ubuntu.
   com/ubuntu jammy-updates main restricted universe
   multiverse"

$ sudo add-apt-repository -y "deb http://security.ubuntu.
   com/ubuntu jammy-security main restricted universe
```

```
   multiverse"

$ sudo apt-get update

$ sudo apt-get install libuhd-dev uhd-host libzmq3-dev
```

Si efectivamente se van a utilizar SDRs en el *testbed* es necesario descargar la base de firmwares:

```
$ sudo uhd_images_downloader
```

El siguiente paso es descargar el proyecto srsRAN desde su repositorio:

```
$ git clone https://github.com/srsran/srsRAN_Project.git
```

Particularmente al momento de escribir este documento el último *release* es el 25_04.

Crear el directorio *build* en la raíz del repositorio y correr el cmake:

```
$ cd srsRAN_Project
$ mkdir build
$ cd build
$ cmake ../ -DENABLE_EXPORT=ON -DENABLE_ZEROMQ=ON
```

Verificar que en la salida del comando cmake se vea algo como lo siguiente:

```
...
-- FINDING ZEROMQ.
-- Checking for module 'ZeroMQ'
--   No package 'ZeroMQ' found
-- Found libZEROMQ: /usr/local/include, /usr/local/lib/
   libzmq.so
...
```

Finalmente compilar e instalar:

```
$ make -j $(nproc)
$ sudo make install
```

UE

En el caso de querer emular un UE es posible utilizar el srsUE del proyecto srsRAN_4G. Para su instalación se comienza instalando las dependencias.

```
$ sudo apt-get install build-essential cmake libfftw3-dev
   libmbedtls-dev libboost-program-options-dev libconfig++-
   dev libsctp-dev
```

Luego se instala el el diver del SDR y la librería ZMQ:

```
$ sudo add-apt-repository -y "deb http://uy.archive.ubuntu.
   com/ubuntu jammy-updates main restricted universe
   multiverse"

$ sudo add-apt-repository -y "deb http://security.ubuntu.
   com/ubuntu jammy-security main restricted universe
   multiverse"

$ sudo apt-get update

$ sudo apt-get install libuhd-dev uhd-host libzmq3-dev
```

El siguiente paso es descargar el proyecto srsRAN_4G desde su repositorio:

```
$ git clone https://github.com/srsRAN/srsRAN_4G.git
```

Particularmente al momento de escribir este documento el último *release* es el 23_11.

Crear el directorio *build* en la raíz del repositorio y correr el cmake:

```
$ cd srsRAN_4G
$ mkdir build
$ cd build
$ cmake ../
```

Verificar que en la salida del comando cmake se vea algo como lo siguiente:

```
...
-- FINDING ZEROMQ.
-- Checking for module 'ZeroMQ'
--   No package 'ZeroMQ' found
-- Found libZEROMQ: /usr/local/include, /usr/local/lib/
   libzmq.so
...
```

Finalmente compilar e instalar:

```
$ make -j $(nproc)
$ sudo make install
```

Core

Para el *core* se utilizará el *deploy* de Open5GS en Docker que viene integrado con srsRAN, por lo que el unico requisito para tener el *Core* es tener correctamente instalado el Docker Engine:

```
$ sudo apt-get install -y ca-certificates curl gnupg
$ sudo install -m 0755 -d /etc/apt/keyrings
```

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
    sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
$sudo chmod a+r /etc/apt/keyrings/docker.gpg
$ 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
$ sudo apt-get install -y docker-ce docker-ce-cli
   containerd.io docker-buildx-plugin docker-compose-plugin
```

NearRT RIC - FlexRIC

Se inicia instalando las dependencias:

```
$ sudo apt-get install swig libsctp-dev python3 cmake-
   curses-gui python3-dev pkg-config libconfig-dev
   libconfig++-dev
```

Luego se clona el proyecto y se pasa a la rama *br-flexric* (la rama por defecto del repositorio es *dev*).

```
$ git clone https://gitlab.eurecom.fr/mosaic5g/flexric.git
$ cd flexric
$ git checkout br-flexric
```

Previo a compilar hay que asegurarse que se esta trabajando con GCC versión 10. Si se tiene otra versión:

```
$ sudo apt install -y gcc-10 g++-10
$ sudo update-alternatives --install /usr/bin/gcc gcc /usr/
   bin/gcc-11 110 --slave /usr/bin/g++ g++ /usr/bin/g++-11
$ sudo update-alternatives --install /usr/bin/gcc gcc /usr/
   bin/gcc-10 100 --slave /usr/bin/g++ g++ /usr/bin/g++-10
$ sudo update-alternatives --config gcc
```

El último comando permite elegir la versión por defecto de GCC a utilizar por el sistema, elegir la 10.

Resta compilar:

```
$ mkdir build
$ cd build
$ cmake -DKPM_VERSION=KPM_V3_00 -DXAPP_DB=NONE_XAPP ../
$ make
$ sudo make install
```

Observación: si al momento de hacer el cmake da un error con swig (necesita la versión 4.1) aplicar los siguientes comandos:

```
$ sudo apt install -y build-essential libpcre2-dev
$ cd /tmp
$ curl -LO https://sourceforge.net/projects/swig/files/swig
   /swig-4.1.1/swig-4.1.1.tar.gz
$ tar xzf swig-4.1.1.tar.gz
$ cd swig-4.1.1
$ ./configure
$ make -j"$(nproc)"
$ sudo make install
$ sudo apt purge -y swig swig4.0
$ hash -r
```

Eliminar el directorio *build*, crearlo de vuelta y volver a ejecutar el cmake y make.

NearRT RIC - OSC

En este caso el Near-RT RIC corre como un despliegue de contenedores a través de Docker Compose. El primer paso es instalar Docker Engine tal como se hizo para el *Core*.

Finalmente se clona el repositorio, al momento de escribir este documento se utiliza el *i-release*.

```
$ git clone https://github.com/srsran/oran-sc-ric
```

Utilización

A continuación se explica como iniciar cada elemento, el orden de inicio se debe respetar y es: Core →Near-RT RIC →gNB →UE →xApp.

Core

Al utilizar el despliegue en docker compose provisto por srsRAN, iniciar el *core* es sencillo, simplemente dirigirse a la carpeta `srsRAN_Proyect/docker` y ejecutar:

```
$ sudo docker compose up --build
```

En lo que hay que prestar mas atención es en la configuración, la misma se encuentra en el directorio `srsRAN_Proyect/docker/open5gs`. Particularmente son de interes dos archivos, el `open5gs.env` y `open5gs-5gc.yml`.

En el primero se configuran varios parámetros de interés como la IP del *Core*, la red que se le asigna a los UEs y el nombre de la red, este último parámetro hay que recordarlo a la hora de configurar el UE COTS.

En el segundo archivo se configuran los distintos elementos del *Core* (AMF, UPF, etc), con ayuda de la documentación de open5gs es posible personalizar la configuración, pero el parámetro a recordar para luego configurar en el UE COTS y gNB es el `plmn_id`. Los archivos de configuración por defecto se adjuntan al repositorio de este trabajo.

Open5GS incorpora una web de administración que por defecto se expone en el puerto 9999 con usuario `admin` y contraseña `1423`. Desde esta interfaz se configuran los APN y UEs que pueden conectarse a la red. En el siguiente capítulo se verá un ejemplo concreto.

Observación: junto con el *Core* se incluye en el despliegue de Docker *Telegraf* y *Grafana* para capturar y visualizar datos del *Core*.

Near-RT RIC

En el caso de FlexRIC, desde el directorio `flexric` ejecutar:

```
$ ./build/examples/ric/nearRT-RIC
```

Para el RIC de la OSC, ir al directorio `oran-sc-ric` y ejecutar:

```
$ docker compose up
```

Puede ser necesario logearse a docker para bajar alguna de las imágenes, para esto se utiliza el comando `docker login` y se siguen las instrucciones.

gNB

Previo a iniciar el gNB es recomendable ajustar algunos parámetros del host

```
#Deshabilitar governador y setear para performante
$ echo performance | sudo tee /sys/devices/system/cpu/cpu*/
   cpufreq/scaling_governor >/dev/null
#Deshabilitar DRM KMS
$ echo N | sudo tee /sys/module/drm_kms_helper/parameters/
   poll >/dev/null
# Setear parámetros de red
$ sudo sysctl -w net.core.wmem_max=33554432
$ sudo sysctl -w net.core.rmem_max=33554432
$ sudo sysctl -w net.core.wmem_default=33554432
$ sudo sysctl -w net.core.rmem_default=33554432
# Activar forwarding
$ sudo sysctl -w net.ipv4.ip_forward=1
#Crear rutas
$ sudo ip ro add 10.45.0.0/16 via 10.53.1.2
$ sudo iptables -t nat -A POSTROUTING -o <INTERFAZ DE RED>
   -j MASQUERADE
```

Notar que al momento de crear las rutas el ejemplo utiliza las IPs por defecto del sistema cuando corre todo un el mismo host, en caso de desagregar los elementos utilizar las IPs que correspondan.

Una vez preparado el host para iniciar el gNB utilizar el siguiente comando desde la carpeta srsRAN_Proyect:

```
$ sudo ./build/apps/gnb/gnb -c <ARCHIVO DE CONFIGURACION>
```

El archivo de configuración a utilizar va a depender de que tipo de implementación este desplegando, para facilitar los primeros pasos con este *testbed* en el gitlab del proyecto[41] se dejan dos ejemplos base, uno para utilizar el UE de srsRAN (`gnb_oran_zmq.conf`) y otro para utilizar un UE COTS (`gnb_oran_sdr.conf`).

Los archivos están comentados para facilitar su configuración y adaptación a las necesidades puntuales, por ejemplo ajustar IPs de los elementos. Por mas detalle sobre la configuración dirigirse a la documentación de srsRAN.

UE srsRAN

El primer paso es agregar el *namespace* de red:

```
$ sudo ip netns add ue1
```

Luego ejecutar el UE desde la carpeta srsRAN_4G:

```
$ sudo build/srsue/src/srsue <ARCHIVO DE CONFIGURACION>
```

Un ejemplo del archivo de configuración se adjunta al repositorio[42].

Luego de iniciado el UE, antes de comenzar a hacer los experimentos hay que hacer unos ajustes, para esto se debe entrar *namespace*:

```
$ sudo ip netns exec ue1 bash
```

Y una vez dentro correr los siguientes comandos:

```
#Deshabilitar governador y setear para performante
$ echo performance | sudo tee /sys/devices/system/cpu/cpu*/
   cpufreq/scaling_governor >/dev/null
#Deshabilitar DRM KMS
$ echo N | sudo tee /sys/module/drm_kms_helper/parameters/
   poll >/dev/null
# Setear parámetros de red
$ sudo sysctl -w net.core.wmem_max=33554432
$ sudo sysctl -w net.core.rmem_max=33554432
$ sudo sysctl -w net.core.wmem_default=33554432
$ sudo sysctl -w net.core.rmem_default=33554432
# Agregar ruta para que el UE llegue al Core
ip route add default via 10.45.1.1 dev tun_srsue
# Agregar DNS
$ sudo resolvectl dns tun_srsue 8.8.8.8
$ sudo echo "nameserver 8.8.8.8" >> /etc/resolv.conf
```

A partir de este punto se puede utilizar el UE de srsRAN en la red móvil.

Observación: en caso de querer agregar mas UEs de srsRAN aplica el mismo mecanismo que en la implementación 1 para agregar mas UEs.

[41] https://gitlab.fing.edu.uy/mobile-optical-networks/testbed/mobile-network
[42] https://gitlab.fing.edu.uy/mobile-optical-networks/testbed/mobile-network

UE COTS

Para utilizar UEs reales (celulares) o equipos similares es necesario programar una tarjeta SIM con los parámetros adecuados para conectarse a la red desplegada. Para esto en este trabajo se utilizan las SIMs programables de Open-Cells[43] que se programa a través del software UICC.

Para esto colocar la SIM en la placa grabadora y conectarla por USB al equipo, luego dirigirse al directorio donde se descargó el programa y correr el siguiente comando:

```
$ sudo ./program_uicc --adm 12345678 --imsi <IMSI> --isdn
   00000001 --acc 0001 --key 6874736969202073796
   d4b2079650a73 --opc 504f20634f6320504f50206363500a4f -
   spn "<NOMBRE DE LA RED>" --authenticate --noreadafter
```

Los parámetros que se deben recordar para configurar luego en el *Core* son el IMSI, la KEY y el OPC. El nombre de la red debe coincidir con el configurado en el *Core*.

Para mas detalles dirigirse a la documentación de Open-Cells[44].

La configuración dentro del dispositivo varía según el modelo, pero generalmente hay que configurar el APN (con el mismo que se introduce cuando se agrega el dispositivo al *Core*, un ejemplo de esto se verá en los casos de uso), habilitar el roaming de datos y poner el modo en solo 5G.

Siguiendo con el mismo esquema que en las implementaciones anteriores resta realizar un inventario de las xApps que vienen por defecto en cada Near-RT RIC.

- FlexRIC: las xApps de ejemplo están en el directorio `flexric/examples/xApps` y están organizadas según el lenguaje de programación en el que están desarrolladas, hay en C, GO y Python.
  - **c/control**: usa el *service model* E2SM-RC para enviar acciones de control al gNB, por ejemplo configurar cuotas de PRB por *slice* o parámetros radio.
  - **c/monitor**: se conecta al near-RT RIC y se suscribe a métricas de la RAN mediante E2SM-KPM para recolectar y mostrar indicadores por UE o celda.
  - **c/helloworld**: consulta al RIC los nodos E2 conectados y los lista por consola.
  - **c/mwc2024**: escenario complejo que maneja el handover entre dos celdas usando E2SM-RC en base a métricas recolectadas a través de E2SM-KPM.
  - **go/flexpolicy**: permite aplicar politicas A1 a la RAN

[43] https://open-cells.com/

[44] https://open-cells.com/index.php/uiccsim-programing/

- **go/helloworld**: obtiene la lista de nodos E2 conectados y la muestra en pantalla.
- **go/monitor/oran**: se suscribe a los nodos E2 usando E2SM-KPM para obtener métricas de los mismos.
- **go/monitor/curst**: igual que el anterior pero sin utilizar los modelos de servicio definidos por O-RAN, utiliza uno propio que se empaqueta dentro de E2AP.
- **go/prb_utilization**: calcula periódicamente los PRBs utilizado en *downlink* por los UE, utiliza un modelo de servicio propio sobre E2AP.
- **go/slice_moni_ctrl**: xapp para monitorizar y controlar *slices*.
- **go/slice_term_ui**: muestra información de los *slices* a través de una interfaz de consola.
- **python3/control**: obtiene información de los UEs y *slices* de la red, permite crear, eliminar o modificar *slices* así como asignar UEs a determinados *slices*.
- **python3/helloworld**: idem a los *helloworld* anteriores.
- **python3/interactive**: xApp interactiva (a través de la terminal) que permite ver métricas de la red y manejar *slices*.
- **python3/monitor**: xApp de monitoreo, hay una versión que utiliza modelos de servicios definidos por O-RAN (E2SM-KMP) y otra que utiliza uno propio, siempre sobre E2AP.

- RIC OSC: en este caso las xApps se encuentran en el directorio `oran-sc-ric/xApps/python`, por el momento solo tiene ejemplos en Python.

  - **kpm_mon_xapp.py**: se conecta al Near-RT RIC y se suscribe a E2SM-KPM para recibir indicaciones (métricas) de KPIs desde el nodo E2.
  - **simple_ccc_xapp.py**: esta xApp envía mensajes de control a un nodo E2 usando el modelo de servicio E2SM-CCC para variar el limite de PRBs permitido.
  - **simple_mon_xapp.py**: versión de monitoreo básica.
  - **simple_rc_ho_xapp.py**: ejemplo simple para ordenar la realización de un *Handover* entre dos celdas predefinidas utilizando E2SM-RC.
  - **simple_rc_xapp.py**: a través de E2SM-RC esta xApp fija las cuotas de PRBs por *slice*.
  - **simple_xapp.py**: utiliza E2SM-KPM para monitorizar el tráfico de un UE, cuando este supera un cierto umbral envía un mensaje de control vía E2SM-RC para limitar el tráfico de ese UE a través de la limitación de los PRBs asignados.

Conclusiones

Sin duda esta es la plataforma de pruebas más "realista" de todas las vistas en este trabajo. La ventaja clara es que las pruebas realizadas sobre este *testbed* van a tener un comportamiento extrapolable a un despliegue real, además puede escalar dinamicamente (correr todo en un solo PC de escritorio o tener todo distribuido en varios servidores con SDRs). Lo anterior también es una de sus desventajas, realizar pruebas con varios UEs y gNBs implica una inversión grande en equipamiento. Otro punto débil (o al menos así se entiende según la experiencia durante el desarrollo de esta tesis) es la sensibilidad a los cambios de versiones. Como se comentó al inicio la idea era seguir el instructivo del trabajo realizado por el NIST (octubre de 2024), pero a la fecha (octubre 2025) algunas actualizaciones hicieron que no se pueda reproducir con facilidad y fuera necesario realizar adaptaciones para utilizar las últimas versiones de los sistemas involucrados. Vale la pena recordar que srsRAN dio un problema similar en la implementación 1.

5.6. Conclusiones del capítulo

En este capítulo se vieron cinco implementaciones de *testbeds* para trabajar y experimentar con O-RAN. De cada uno de ellos se detalló su arquitectura, componentes que lo integran, inventario de xApps, cómo se instalan y cómo se utilizan con el objetivo de que el lector pueda reproducirlos fácilmente. Se buscó tener plataformas de pruebas variadas que permitan realizar experimentos con ópticas diferentes, por ejemplo pruebas de conceptos masivas simuladas en ns3 con la implementación 4, dar los primero pasos en un ambiente emulado completo con las implementaciones 1 y 2 o realizar experimentos más realistas y distribuidos con la implementación 5. A modo de resumen en la tabla 5.1 se sintetizan las principales características de cada implementación.

Los resultados de este capítulo fueron parcialmente incluidos en un articulo científico que fue presentado y aceptado en la conferencia 13th Wireless Days Conference[45] realizada entre el 1 y 3 de diciembre de 2025 en Niterói, Brasil.

En el siguiente capítulo se realizarán pruebas de concepto con algunas de estas implementaciones para que en base a ejemplos prácticos se pueda entender aún mejor la utilidad y pertinencia de cada implementación.

[45] https://wireless-days.dnac.org/

Implementación	Stack (Core / RAN)	RIC	Radio / Canal	Ventajas	Limitaciones
1 (NIST)	Open5GS / srsRAN	OSC Near-RT RIC (Kubernetes)	ZeroMQ (canal simulado)	Despliegue sencillo; utiliza herramientas de código abierto ampliamente adoptadas; OSC RIC oficial	Sensible al cambio de versiones de los elementos
1.1 (NIST ext.)	Idem impl. 1	OSC Non-RT RIC (Release L)	ZeroMQ	Incluye soporte para rApps y gestión de políticas A1	Fallan varios pods; conflicto con el Near-RT RIC; no está listo para producción
2 (NIST)	Open5GS / OAI gNB & UE	FlexRIC	RFSimulator (OAI)	Liviano; soporta xApps con requisitos temporales estrictos; buena integración con OAI	No es oficial; curva de aprendizaje de FlexRIC
3 (ns-O-RAN)	simulación de ns-3 / basado en el módulo mmWave	ColORAN (OSC RIC limitado)	Simulado en ns-3	Mezcla un RIC real con la RAN simulada; permite experimentos flexibles	Usa versiones antiguas; rígido, poco adaptable
4 (ns3-ORAN)	Simulación completa en ns-3	RIC lógico (módulos para xApps, datos y conflict mitigation)	Completamente simulado en ns-3	Simulado, altamente flexible; soporta integración de ML con ONNX/PyTorch	Requiere experiencia con ns-3 & C++; curva de aprendizaje; sólo simulaciones, sólo LTE.
5 (srsRAN)	Open5GS / srsRAN	OSC Near-RT RIC / FlexRIC	ZeroMQ o canal real (aire)	Alta personalización; desde despliegues sencillos a complejos; permite usar elementos emulados o reales	A mayor tamaño mayor complejidad y requerimientos de hardware

Tabla 5.1: Comparación de las implementaciones analizadas.

Capítulo 6

Pruebas de casos de uso

A esta altura del documento, se ha realizado un recorrido exhaustivo por la evolución de las redes móviles con especial atención en la red de acceso por radio (RAN), se ha introducido el concepto de O-RAN y su principal impulsor, la O-RAN Alliance, y se ha descrito en detalle su arquitectura, explicando las funciones y características de cada uno de sus componentes. Asimismo, se han relevado diversos antecedentes que documentan el uso de *testbeds* desarrollados por otros investigadores, junto con la presentación de algunos de ellos y las instrucciones necesarias para su instalación y utilización. El siguiente objetivo consiste en seleccionar algunos de estos *testbeds* y llevar a cabo pruebas o experimentos que permitan validar su funcionalidad y evaluar su utilidad práctica.

En este capítulo se abordarán tres implementaciones que, al momento de redactar este documento, se consideran las más prometedoras: la implementación 2, la implementación 4 y la implementación 5 del capítulo anterior. Para cada una de ellas se diseñará un escenario específico y acorde al tipo de implementación que permita realizar pruebas y verificaciones orientadas a evaluar su desempeño y funcionalidad. A los experimentos correspondientes se los denominará *caso de uso 1*, *caso de uso 2* y *caso de uso 3*, respectivamente.

6.1. Caso de uso 1

Basado en la implementación 2 este caso constará de la red móvil emulada completa utilizando el *core* de Open5GS, un gNB de OAI y tres UE de OAI. A esto se le sumará el Near-RT RIC de FlexRIC al que se le instalará una xApp de monitoreo que recopila métricas de los UE y del enlace de los UE con el gNB, estas métricas se muestran en un *dashboard* de Grafana[1]. El canal inalámbrico entre los UE y gNB es simulado con RFSimulator. En la figura 6.1 se representa la arquitectura a utilizar.

Se realizaran los siguientes experimentos:

- **Experimento 1**: tráfico de *Uplink* y *Downlink* por separado entre un UE

[1] https://grafana.com/

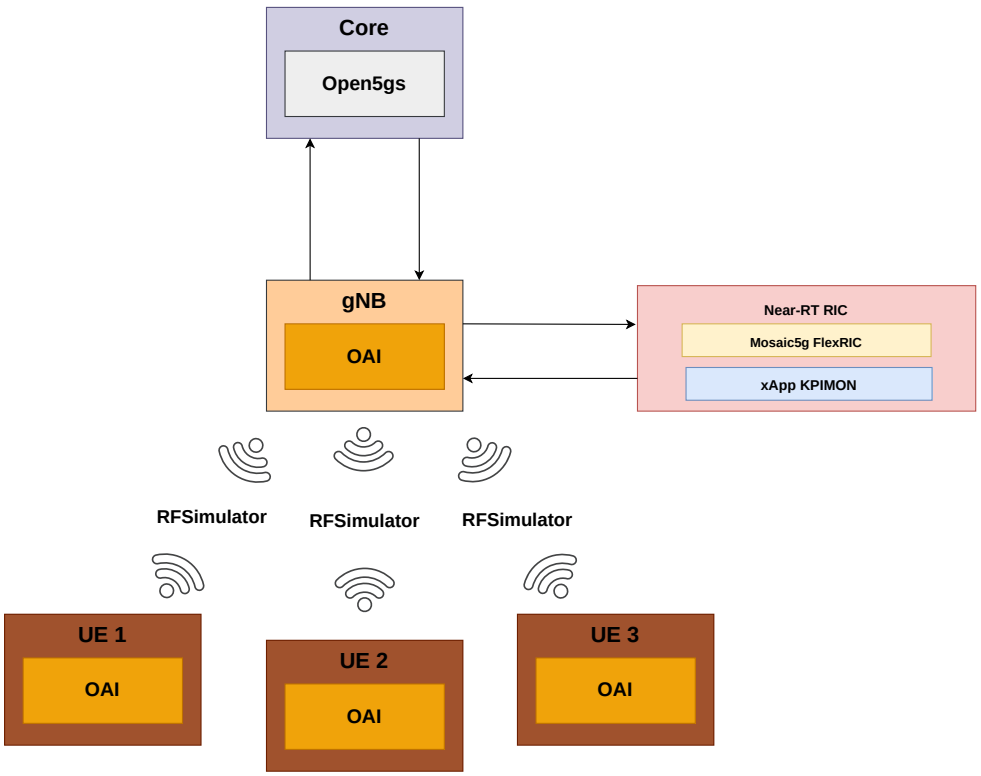

Figura 6.1: Arquiterctura del caso de eso 1 con la red móvil compuesta por el core, un gNb y tres UE además del Near-RT RIC.

y el *core* a máxima capacidad.

- **Experimento 2**: tráfico de bidireccional entre el UE1 y el *core* a diferentes velocidades junto con tráfico en un solo sentido entre el UE2 y el UE3 a una velicidad definida.
- **Experimento 3**: tráfico *Uplink* entre un UE y el *core* a máxima capacidad modificando el canal con RFSimulator.

En todos los casos el objetivo será observar en el panel de Grafana los datos capturados por la xApp. Además cuando no se indique lo contrario el modelo del canal utilizado sera AWGN sin atenuación, ruido -10 dB, sin variación temporal, completamente síncrono y sin *delay spread* (configuración en figura 6.2).

Experimento 1

Luego de iniciar todos los componentes tal cual se indicó en el capitulo anterior hay que iniciar la xApp, para esto también se hace uso de la *Automation Tool del NIST*, simplemente se debe ejecutar el script `start_grafana_with_csv_xapp_kpm_moni.sh` ubicado en `O-RAN-Testbed-Automation/OpenAirInterface_Testbed/`

```
model 1 rfsimu_channel_ue0 type AWGN:
model owner: not set
nb_tx: 1    nb_rx: 1    taps: 1 bandwidth: 40000000.000000    sampling: 61440000.000000
channel length: 1    Max path delay: 0.000000    ricean fact.: 0.000000    angle of arrival: 0.000000 (randomized:No)
max Doppler: 0.000000    path loss: 0.000000  noise: -10.000000 rchannel offset: 0    forget factor; 0.000000
Initial phase: 0.000000   nb_path: 10
taps: 0   lin. ampli. : 1.000000    delay: 0.000000
```

Figura 6.2: Configuración predeterminada del cana simulado con RFSimulator.

`RAN_Intelligent_Controllers/Flexible-RIC/additional_scripts`. Esto pone en ejecución la xApp y levanta en el puerto 3000 del equipo la consola de Grafana para ver los datos. Hay que hacer unas configuraciones menores en Grafana para visualizarlos, en la documentación de la solución se explica claramente.

En la figura 6.3 se puede ver el panel de Grafana antes de iniciar cualquier intercambio de información.

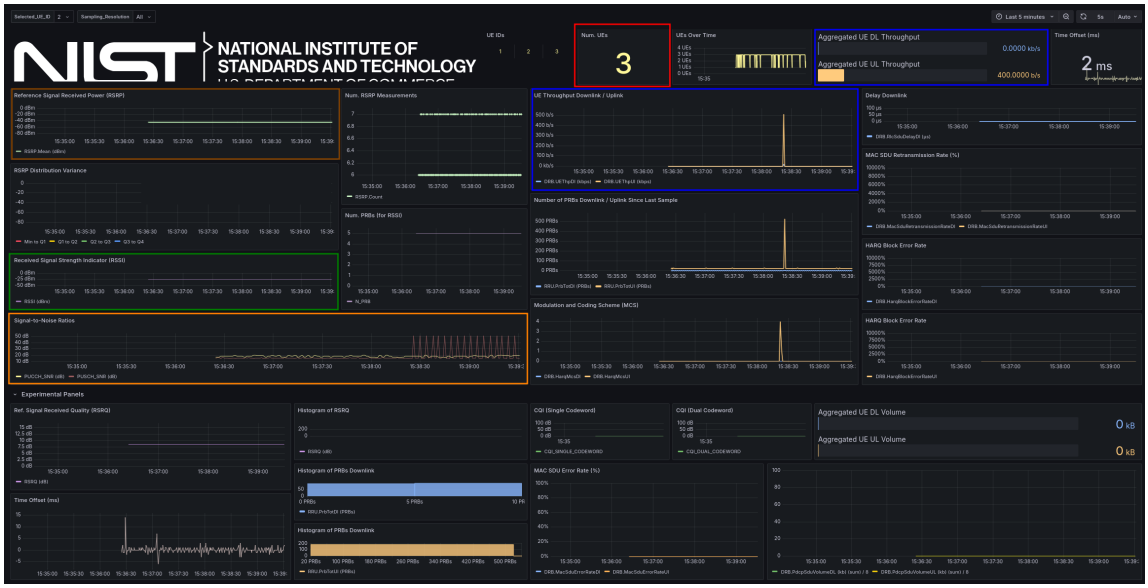

Figura 6.3: Panel de Grafana de la xApp de monitoreo con todo el sistema activo sin transmitir datos. En azul las medidas de tráfico, en rojo la cantidad de UEs conectados, en marrón el RSRP, en verde el RSSI y en naranja el SNR.

Lo primero a destacar es que la xApp efectivamente detecta tres UEs conectados a la red, luego que las métricas de tráfico (6.4) están casi en 0bps, lo que tiene sentido ya que el único tráfico existente es el de control. Si se analizan otros valores como el RSSI, SNR o RSRP (6.5) y se comparan con los logs del gNB se puede verificar que dichos valores son correctos. Lo mismo sucede con el resto de parámetros. En resumen la xApp está capturando correctamente estas métricas del gNB a través del Near-RT RIC.

Siguiendo con el experimento se procede a enviar paquetes TCP desde el UE al *core* (*uplink*) a máxima capacidad. Para esto se utiliza la herramienta *iperf*:

Figura 6.4: Gráfica de troughput para el sistema activo sin transmitir.

Figura 6.5: RSSI, SNR y RSRP para el sistema activo sin transmitir.

```
# EN EL CORE
$ iperf -s -B 10.45.0.1

#EN EL UE
$ iperf -c 10.45.0.1 -B 10.45.0.2
```

Se trafic\u00f3 durante 60 segundos, se pararon 60 segundos y luego se dejo transmitiendo de corrido, en la figura 6.6 se muestran los datos capturados.

Luego se realizó lo mismo pero invirtiendo el sentido del tráfico (lo que implica intercambiar donde se ejecutan los *iperf*). En la figura 6.7 se visualizan los datos capturados por la xApp.

Aquí se destaca que los datos de tráfico reportados por *iperf* coinciden con

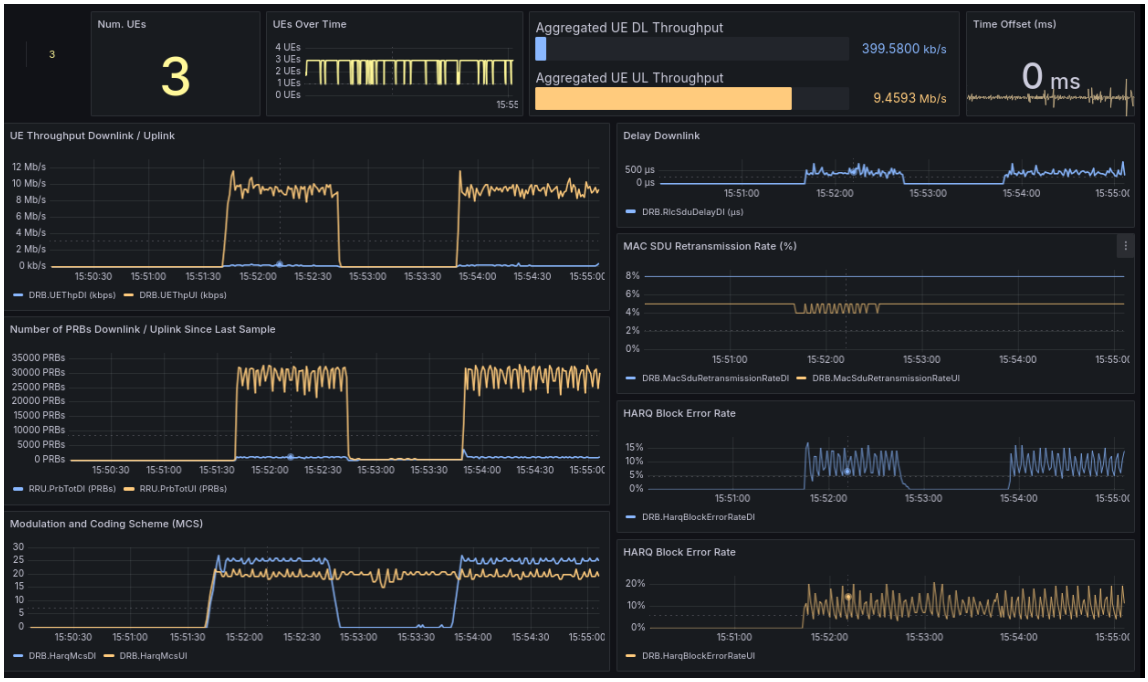

Figura 6.6: Parte del panel de Grafana de la xApp de monitoreo con todo el sistema activo transmitiendo del UE al core.

Figura 6.7: Parte del panel de Grafana de la xApp de monitoreo con todo el sistema activo transmitiendo del core al UE.

los capturados por la xApp, además también se puede apreciar que otros valores relacionados a la transmisión de datos como el número de PRBs, el HARQ BLER o el esquema de modulación y codificación entre otros parámetros varían en concordancia con la transmisión de datos.

Experimento 2

Este experimento es una extensión del anterior, aquí se busca probar el manejo de tráfico en varios sentidos y a distintas velocidades fijas. Se configuro el sistema para que el UE1 transmita hacía el *Core* a 3 Mbps mientras que le *Core* le transmite al UE1 a 5 Mbps. Por otro lado el UE2 transmite datos hacía el UE3 a una tasa de 8 Mbps. En la figura 6.8 se muestran las estadísticas de tráfico recolectadas por la xApp para cada nodo.

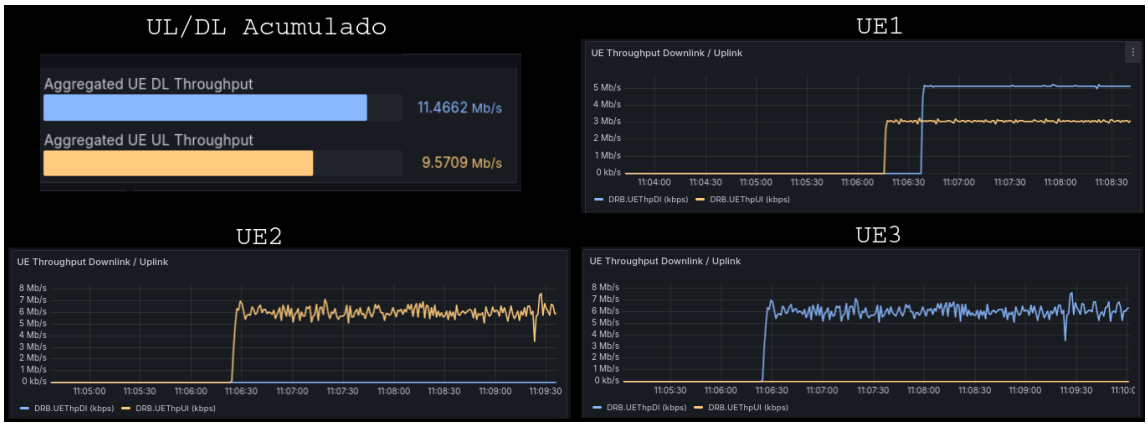

Figura 6.8: Estadisticas de tráfico para cada nodo junto al acumulado. En azul el *downlinnk* y en amarillo el *uplink*.

Dado el experimento se debería ver un acumulado de 11 Mbps en *uplink* y 13 Mbps en *downlink*, pero vemos que los valores medidos son aproximadamente 9.5 Mbps en *uplink* y 11.5 Mbps en *downlink*. Mirando las gráficas de tráfico individual para cada nodo se ve que el UE1 esta transmitiendo y recibiendo a las velocidades programadas, mientras que el UE2 transmite a 6.5 Mbps (aproximadamente) y el UE3 recibe a la misma velocidad, con estos valores si cierra la cifra registrada en el acumulado.

Experimento 3

La idea de esta prueba es experimentar con el RFSimulator para ver si la modificación de algún parámetro del canal se ve reflejada en los datos recolectados. En este caso se eligió tomar el canal por defecto e ir aumentando el *path loss* de acuerdo a la siguiente secuencia: 0 dB → 3 dB → 10 dB → 20 dB → 25 dB → 30 dB → 35 dB → 38 dB → 41 dB. La transmisión se configuró a 12 Mbps desde el UE1 al *Core*. Para no saturar con información el experimento se analizarán solo alguno de los parámetros capturados por la xApp, los elegidos fueron el *Throughput* (figura

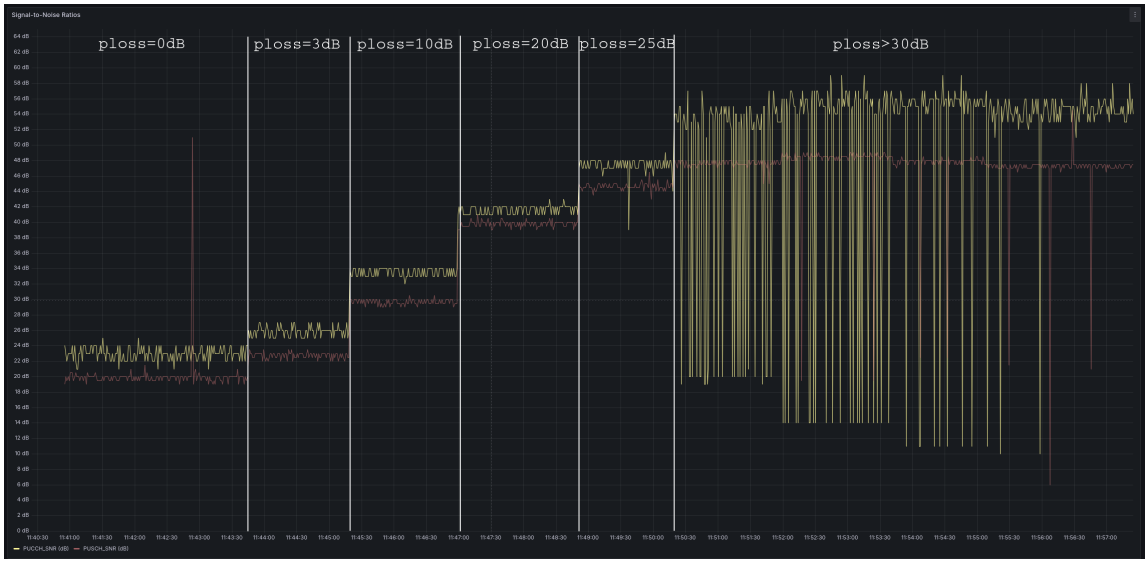

Figura 6.9: SNR (PUCCH SNR en amarillo, PUSCH SNR en rojo) medido durante la variación del path loss.

6.10), la relación señal a ruido (6.9), el *bit error rate* (figura 6.12) y el esquema de modulación (figura 6.11).

Comenzando con la SNR, su comportamiento es el esperable hasta $ploss = 30dB$, para valores mayores los datos capturados dejan de tener sentido. No se encontró una explicación para este fenómeno en las pruebas realizadas.

El *Throughput* y el esquema de modulación se pueden analizar juntos, es fácil ver (y totalmente esperable) que el *Throughput* cae cuando el sistema debe utilizar modulación con menor eficiencia espectral pero más robusto para hacer frente a la degradación del canal.

Finalmente el BER del sistema también es coherente, al principio se mantiene constante, incluso se ve como bajar la tasa de codificación surge efecto ya que la BER no aumente. Luego y como era esperable el canal ya está tan degradado que la BER comienza a subir hasta llegar al 100 %.

En resumen, dado el tipo de tráfico configurado y el modelo de canal utilizado los datos obtenidos a través de la xApp de monitoreo son consistentes con lo esperado.

Conclusiones

En todos los experimentos realizados se obtuvieron resultados razonables y esperados. Es importante recordar que el objetivo de los experimentos no es probar un determinado caso si no ver la viabilidad del *testbed*. Este sistema permite modelar sistemas sencillos (un gNB y pocos UEs) junto con el Near-RT RIC y las xApps. En este caso particular se probó una xApp de monitoreo y se verificó que los datos obtenidos a través de la interfaz E2 son congruentes con el experimento realizado. Factores como distintos tipos de tráfico, diferentes orígenes y destinos

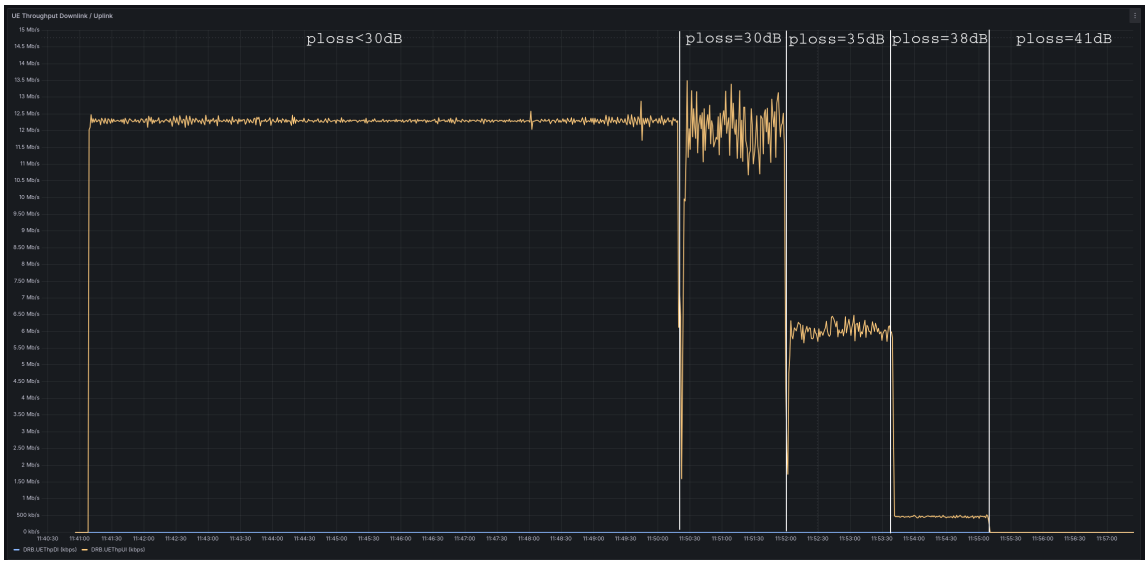

Figura 6.10: Variación de throughput entre el UE y el core en relación al path loss

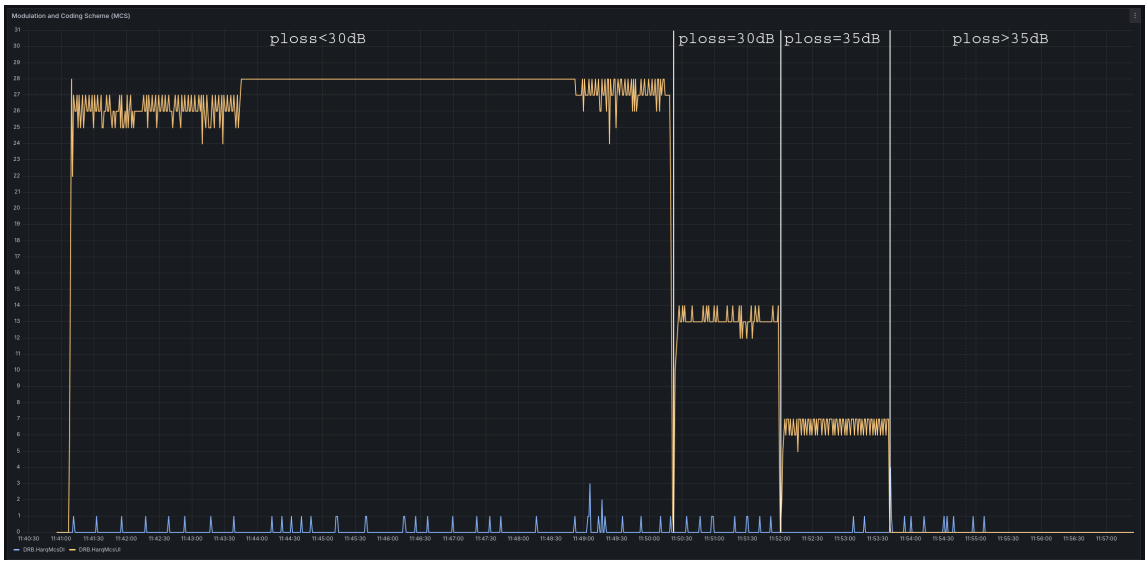

Figura 6.11: Esquema de modulación y codificación de la transmisión elegido en función del estado del canal.

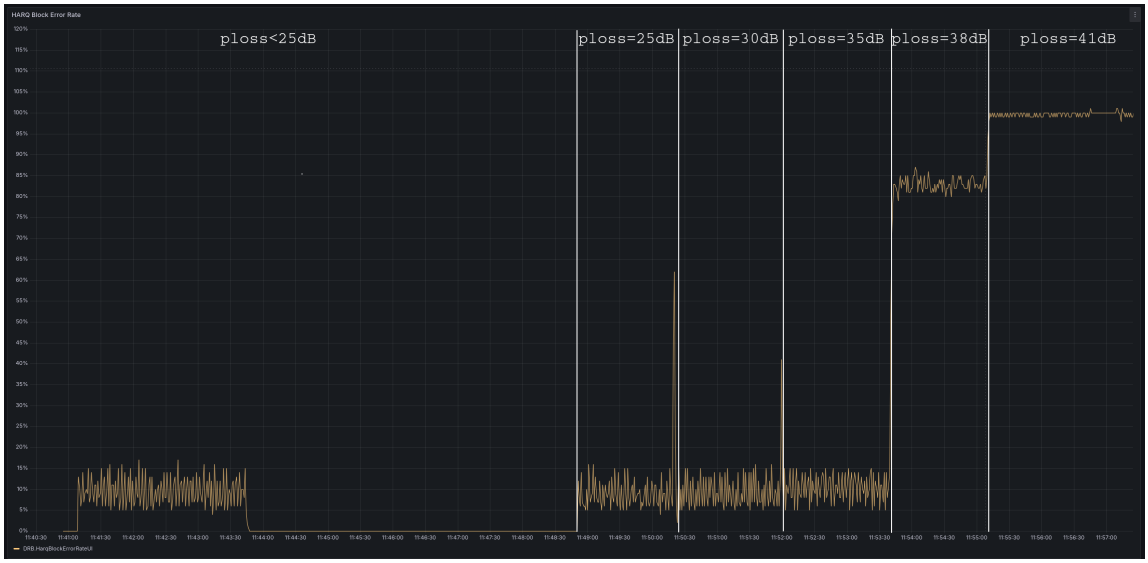

Figura 6.12: Tasa de error de bit del canal en función del path loss

así como variaciones del canal son reflejadas en los valores obtenidos mediante la xApp. Por su facilidad y funcionalidad (en la escala que se mencionó) lo hacen un *testbed* ideal para introducirse en el mundo de O-RAN y para realizar pruebas de concepto sencillas de manera fácil, rápida y con pocos recursos.

Gracias a la investigación de esta tesis este tipo de *testbed* ya fue utilizado por estudiantes de grado en un curso de las carreras de nuestra facultad.

6.2. Caso de uso 2

En este caso de uso se utiliza la implementación 4 (modulo del NIST para ns3). Perfectamente se podrían realizar los mismos experimentos que para el caso de uso 1, pero en miras de demostrar la versatilidad de este ambiente simulado se trabajará con el *HandOver* (HO).

En LTE los algoritmos de HO (A1, A2, A3, A4, A5, B1 y B2) toman la decisión de si cambiar a otra celda o no en base a parámetros de radio medidos por el UE (RSRP, RSRQ o SINR). De estos mecanismos, uno de los más utilizados es el A3. Este procedimiento se activa cuando la potencia de señal medida (RSRP o RSRQ) de una celda vecina supera a la de la celda servidora en un margen definido (offset de A3) durante un tiempo configurado (*Time-to-Trigger*). Una vez cumplida la condición el UE informa el eNB al cual esta conectado, este evalúa la situación y decide si realizar el HO o no. En caso afirmativo se inicia la señalización entre eNodeBs y luego se le ordena al UE conectarse a la nueva celda.

Utilizar cualquier otro parámetro que no sea de radio para evaluar la viabilidad de realizar el HO sería imposible en una red LTE pura. Por ejemplo se podría pensar en evaluar la distancia geométrica del UE a las radio bases más cercanas. El dato de la posición no es un parámetro que se intercambia entre el UE y gNB

de forma natural, aquí es donde entra en juego O-RAN. Es totalmente viable tener una xApp que conozca la posición de los eNB y solicite a través del Near-RT RIC y su interfaz E2 el dato de la posición geográfica a los UE. En base a estos datos la xApp puede correr un algoritmo que decida si realizar el HO o no.

Experimento

El experimento a realizar en este caso de uso se basa en una xApp (o como lo llaman en este modulo de ns3 un *logic module*) que realiza el HO al eNB más cercano basándose solo en la distancia geométrica. Además se comparará en la misma corrida como es el HO de un UE si utiliza A3 o la xApp. La trayectoria que siguen los UEs es aleatoria o configurable y el número de UEs a desplegar es variable.

También podrían ser variables la cantidad de eNB, para simplificar se utilizan tres. En la figura 6.13 se muestra el esquema del experimento. En todos los casos, los UE comienzan conectados al eNB1.

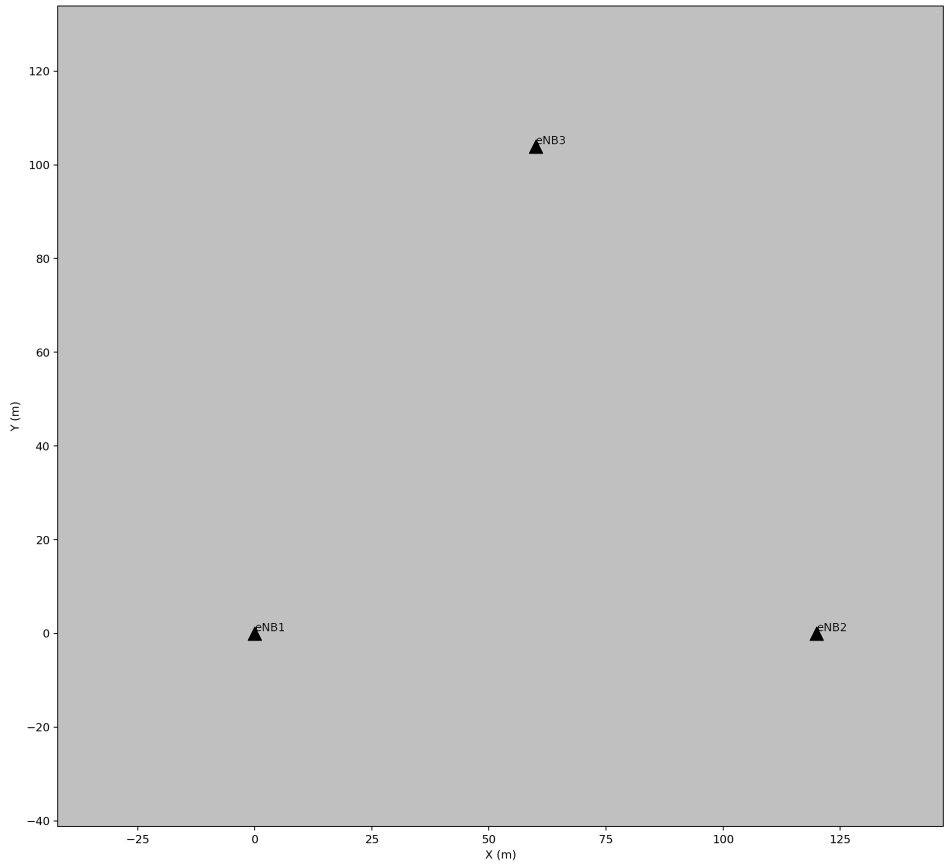

Figura 6.13: Esquema del experimento del caso de uso 2. Se ven los 3 eNB ubicados formando un tríangulo. Todo el fondo gris es la zona de circulación de los UEs.

El código de la simulación (puede verse el gitlab del proyecto[2]) fue adaptado de un experimento realizado por los autores del modulo con la asistencia de ChatGPT [38]. El código no está optimizado ya que la idea es mostrar las cualidades del sistema no un ejemplo en particular.

En la figura 6.14 se ve una corrida con cuatro UEs. Cada línea de color es la trayectoria recorrida por cada uno, con una **I** se marca el inicio y con una **F** el final. Sobre cada trayectoria se observan puntos negros que indican que ahí se realizó un HO por A3 y rosados si se realizó un HO por distancia. En los puntos se indica la celda de origen y la celda de destino.

Observación: en este sencillo ejemplo todos los UE comienzan conectados al eNB1, por este motivo a veces puede aparecer una HO del eNB1 al eNB1.

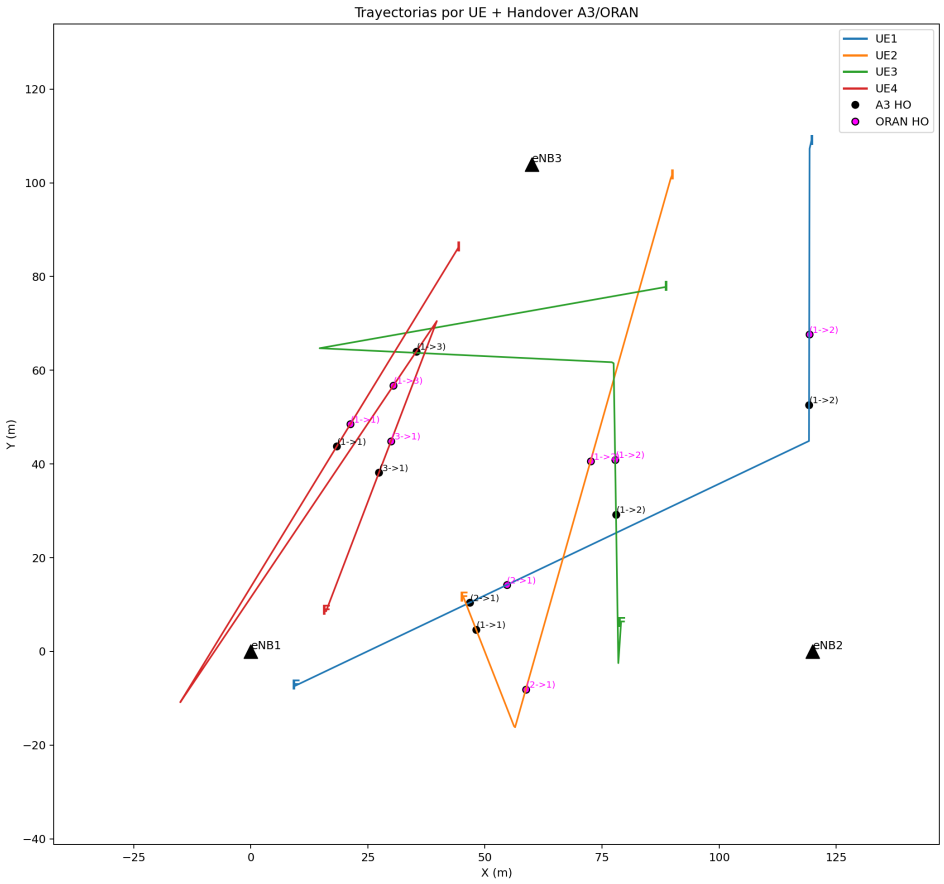

Figura 6.14: Simulación con 4 UEs

Además es posible imprimir en pantalla cualquier otra información que se considere útil, por ejemplo un log donde se muestra información sobre los HO como se ve en la figura 6.15.

[2]https://gitlab.fing.edu.uy/mobile-optical-networks/testbed/mobile-network

```
20.0209 [oran] HO EndOk IMSI=4 RNTI=47 -> Cell 1 at
   (21.2803,48.5108)
22.4889 [a3] HO EndOk IMSI=4 RNTI=47 -> Cell 1 at
   (18.3894,43.7873)
38.0209 [oran] HO EndOk IMSI=1 RNTI=1 -> Cell 2 at
   (119.244,67.6704)
52.0889 [a3] HO EndOk IMSI=1 RNTI=1 -> Cell 2 at
   (119.213,52.6159)
53.0209 [oran] HO EndOk IMSI=2 RNTI=2 -> Cell 2 at
   (72.5548,40.5929)
83.0209 [oran] HO EndOk IMSI=3 RNTI=3 -> Cell 2 at
   (77.8081,40.9259)
86.0209 [oran] HO EndOk IMSI=4 RNTI=5 -> Cell 3 at
   (30.4775,56.7553)
89.6889 [a3] HO EndOk IMSI=4 RNTI=5 -> Cell 3 at
   (35.3668,64.0157)
91.2889 [a3] HO EndOk IMSI=3 RNTI=3 -> Cell 2 at
   (77.9918,29.215)
95.0209 [oran] HO EndOk IMSI=1 RNTI=48 -> Cell 1 at
   (54.6554,14.1783)
95.0209 [oran] HO EndOk IMSI=2 RNTI=49 -> Cell 1 at
   (58.7434,-8.08633)
99.2889 [a3] HO EndOk IMSI=1 RNTI=48 -> Cell 1 at
   (46.7965,10.4363)
104.021 [oran] HO EndOk IMSI=4 RNTI=50 -> Cell 1 at
   (29.9461,44.8679)
106.889 [a3] HO EndOk IMSI=4 RNTI=49 -> Cell 1 at
   (27.3805,38.1491)
115.289 [a3] HO EndOk IMSI=2 RNTI=50 -> Cell 1 at
   (48.1217,4.68793)
```

Figura 6.15: Log de la simulación donde se muestra (de derecha a izquirda): marca de tiempo, tipo de HO, resultado del HO, identificador de usuario (IMSI), identificador de radio temporal (RNTI), celda de destino del HO, coordenadas geométricas del lugar donde se realiza el HO.

Como ya se mencionó, si se quieren agregar más UEs a la simulación solo hace falta indicar el numero deseado en un parámetro del código, por ejemplo en la figura 6.16 se ve el resultado del experimento con ocho UEs.

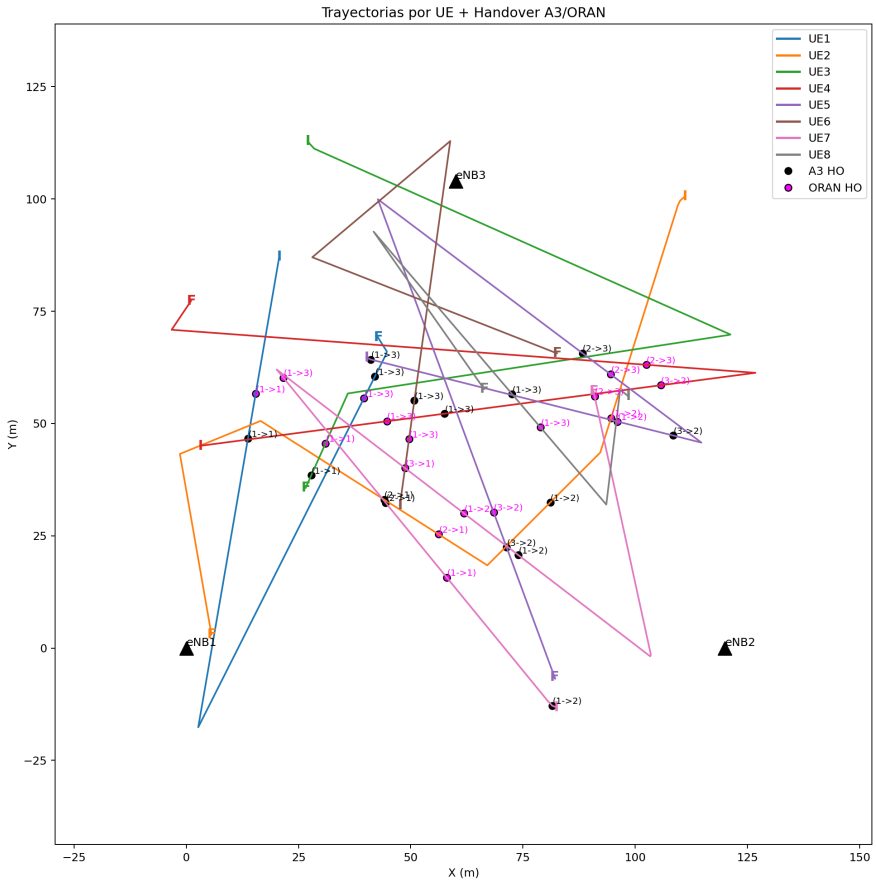

Figura 6.16: Simulación con 8 UEs

Aunque los experimentos se esforzaron en mostrar la versatilidad y utilidad del ambiente simulado es importante verificar que los resultados sean al menos coherentes. Si se toma el ejemplo con 4 UEs, se puede ver como el HO por distancia se realiza en casi todos los casos al pasar el punto equidistante de la trayectoria respecto a los eNBs, y en general el HO A3 se realiza poco después. Esto tiene sentido ya que en este ejemplo los parámetros de radio analizados también dependen unicamente de la distancia (hay linea de vista) por lo que el umbral se cumple al pasar el punto equidistante, y debido al *offset* el HO por A3 se realiza momentos después que el HO por distancia.

Conclusiones

Esta pequeña demostración práctica de este *testbed* muestra lo útil y potente que es. Permite realizar experimentos casi sin importar la escala (solo dependiendo del hardware donde corra la simulación) de manera sencilla. Además permite realizar experimentos o pruebas que si se utilizará una arquitectura emulada o real sería muy complicado. Por ejemplo aquí variar la trayectoria o algún parámetro de la red es simplemente editar el código y volver a correr la simulación, en un caso

real implica tomar nuevamente todos los datos y reconfigurar todo el sistema. Este ejemplo puntual de probar dos mecanismos de HO en idénticas condiciones hubiera sido muy costoso en una plataforma real o emulada. Analizando los aspectos negativos, el primero (que ya se mencionó en el capítulo anterior) es la curva de aprendizaje de ns3 y C++. Por otro lado es una simulación, que en muchos casos se puede parecer a la realidad pero nunca lo es. Además como se vio imperfecciones en el código puede llevar a resultados engañosos o erróneos. Más específicamente hablando de O-RAN, las xApp no son trasladables a un ambiente real, las mismas se codifican en C++ para ns3, o sea que para implementarlas en un RIC real deben ser reprogramadas. Finalmente recordar que solo brinda soporte para LTE. En resumen es un *tesbed* ideal para pruebas de concepto rápidas o para evaluar la viabilidad de un experimento mayor previo a realizarlo en un ambiente mas realista, también es muy útil para probar la lógica de las xApps y si es viable su utilización.

6.3. Caso de uso 3

El último caso de uso se basará en la implementación 5 y se realizarán tres experimentos. En el primero se utilizará el UE emulado de srsRAN conectado al gNB mediante ZMQ y como Near-RT RIC se desplegará el RIC de OSC con la xApp simple_xapp.py. Se aprovechará este experimento para mostrar como capturar paquetes E2AP con Wireshark[3]. En el segundo experimento el UE será un celular Samsung A33 que se conectará a la radio base a través de un SDR B200. El Near-RT RIC usado será Flexric con la xApp de monitoreo. En el tercer experimento se combinarán los dos anteriores para tener una red móvil emulada con dos gNB, uno conectado a UEs COTS a través de un SDR y otro a UEs srsRAN. En todos los casos (a no ser el COTS UE) todos los elementos correrán en el mismo host (Core i7-4790 de 4 núcleos y 32GB de RAM).

Experimento 1

Este experimento utilizará un *stack* de elementos emulados (srsRAN para gNB y UE), open5gs para el *core* y el Near-RT RIC de la OSC. El canal se emulará con ZMQ. En la figura 6.17 se puede ver la arquitectura del experimento con sus componentes.

[3]https://www.wireshark.org/

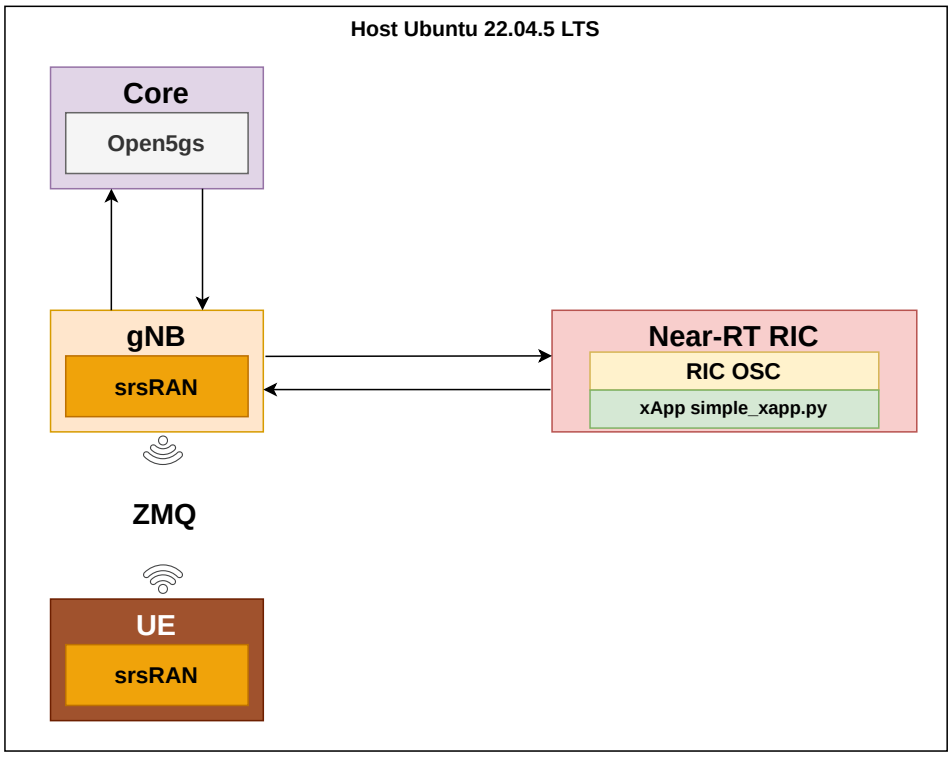

Figura 6.17: Arquitectura del experimento, dentro del host corren todos los elementos, el canal se simula con ZMQ.

La xApp a probar es `simple_xapp.py`. Esta xApp se caracteriza por utilizar dos modelos de servicio en la misma aplicación, el E2SM-KPM para recolectar métricas y el E2SM-RC para realizar acciones de control. La xApp monitorea el tráfico de un determinado UE, cuando los MB transmitidos supera cierto umbral (20 MB por defecto) se realiza una acción de control para limitar el número de PRBs asignados a dicho UE y así limitar el tráfico.

Por defecto se limita el número de PRBs a 10, esto hace que se pierda la conexión entre el UE y el gNB. En este caso se limitó el número de PRBs a 39, para esto editar el archivo `xApps/python/simple_xapp.py`, en la linea `self.max_prb_ratio1 = 10` cambiar 10 por 39. El primer paso es iniciar todos los componentes tal como se explicó en el capitulo 5, en este punto debería haber conectividad entre el UE y el *core* (verificar con ping e iperf) así como con los logs. Para lanzar la xApp se utiliza el siguiente comando desde el directorio `oran-sc-ric`:

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

Con la xApp corriendo sin realizar transmisiones en pantalla se puede ver (figura 6.18) que en el periodo de granularidad (1 segundo para este caso) no se transmitieron datos (`DRB.RlcSduTransmittedVolumeDL`). Además el acumulado

```
------------------------------------------------------------------

Data Monitoring:
  E2SM_KPM RIC Indication Content:
  -ColletStartTime:  2025-10-25 22:44:14
  -Measurements Data:
  --UE_id: 0
  ---granulPeriod: 1000
  ---Metric: DRB.RlcSduTransmittedVolumeDL, Value: 0.0 [MB]

Control Logic:
 Tx Data Stats:
  UE ID: 0, Max PRB Ratio: n/a, TXed Data [MB]: 0.0
------------------------------------------------------------------
```

Figura 6.18: Vista de la xApp cuando no hay transmisiones entre el Core y el UE.

(`TXed DATA`) también es cero.

Una vez que se comienza a transmitir desde el *core* al UE utilizando Iperf se puede ver como la xApp detecta dicha transmisión, en la figura 6.19 se ve como el `DRB.RlcSduTransmittedVolumeDL` es constante (esperado), y el `TXed DATA` va creciendo.

Al superar el umbral de `TXed DATA` definido (20 MB) se dispara la acción de control, en este caso limitar el número de PRBs asignado, esto se ve en la figura 6.20.

Para verificar que efectivamente se ha disminuido la cantidad de PRBs se tiene la figura 6.21 que muestra la salida del comando Iperf, allí se aprecia una clara caída en la velocidad de transmisión, además en la figura 6.22 también se ve que el valor de `DRB.RlcSduTransmittedVolumeDL` es menor.

Como se comentó en la introducción de este caso de uso se utilizará este experimento para mostrar la captura de paquetes E2AP. Esto tiene dos utilidades principales, 1) validar que el sistema emulado efectivamente utiliza las interfaces y protocolos definidos por O-RAN, 2) ver el intercambio de mensajes y su contenido es de suma utilidad para los investigadores. La prueba se realizó con las versiones 4.2 y 4.6 (última al momento de escribir este documento), estas versiones ya soportan el protocolo E2AP, para versiones mas viejas (pero siempre superior a 4.0) puede ser necesario realizar algunos ajustes. El primer caso a analizar es la suscripción del gNB (IP 10.0.2.1) al Near-RT RIC (IP 10.0.2.10), si se ve el intercambio de paquetes E2AP, al conectar el gNB (figura 6.23) se encontrarán 3 paqueres del tipo `E2setupRequest` y las respectivas respuestas.

No.	Time	Source	Destination	Protocol	Length	Info
435	11.457804402	10.0.2.1	10.0.2.10	E2AP	650	E2setupRequest
441	11.458028699	10.0.2.1	10.0.2.10	E2AP	474	E2setupRequest
509	11.461522949	10.0.2.10	10.0.2.1	E2AP	126	E2setupResponse
511	11.461708540	10.0.2.10	10.0.2.1	E2AP	134	E2setupResponse
519	11.471218701	10.0.2.1	10.0.2.10	E2AP	826	E2setupRequest
555	11.474683884	10.0.2.10	10.0.2.1	E2AP	142	SACK (Ack=1, Arwnd=16777216) , E2setupResponse

Figura 6.23: Captura de paquetes E2AP del registro del gNB contra el Near-RT RIC.

Para entender porque hay tres solicitudes de conexión hay que recordar que

```
------------------------------------------------------------------

Data Monitoring:
  E2SM_KPM RIC Indication Content:
  -ColletStartTime:  2025-10-25 22:44:19
  -Measurements Data:
  --UE_id: 0
  ---granulPeriod: 1000
  ---Metric: DRB.RlcSduTransmittedVolumeDL, Value: 3.8 [MB]

Control Logic:
 Tx Data Stats:
  UE ID: 0, Max PRB Ratio: n/a, TXed Data [MB]: 12.2
------------------------------------------------------------------

Data Monitoring:
  E2SM_KPM RIC Indication Content:
  -ColletStartTime:  2025-10-25 22:44:20
  -Measurements Data:
  --UE_id: 0
  ---granulPeriod: 1000
  ---Metric: DRB.RlcSduTransmittedVolumeDL, Value: 3.8 [MB]

Control Logic:
 Tx Data Stats:
  UE ID: 0, Max PRB Ratio: n/a, TXed Data [MB]: 16.0
------------------------------------------------------------------

Data Monitoring:
  E2SM_KPM RIC Indication Content:
  -ColletStartTime:  2025-10-25 22:44:21
  -Measurements Data:
  --UE_id: 0
  ---granulPeriod: 1000
  ---Metric: DRB.RlcSduTransmittedVolumeDL, Value: 3.8 [MB]

Control Logic:
 Tx Data Stats:
  UE ID: 0, Max PRB Ratio: n/a, TXed Data [MB]: 19.8
------------------------------------------------------------------
```

Figura 6.19: Vista de la xApp cuando se está transmitiendo entre el Core y el UE.

para O-RAN el O-DU, O-CU-CP y O-CU-UP son nodos E2 independientes, y en este caso en la configuración del gNB se indicó que los tres se registren ante el Near-RT RIC. Si se inspecciona un poco mas en profundidad el `E2setupRequest` del O-DU (paquete Nro 519) se puede ver que el O-DU anuncia los modelos de servicio que maneja (E2SM-KPM y E2SM-RC) y de estos modelos de servicio las métricas o parámetros disponibles, entre ellos los utilizados por la xApp (`DRB.RlcSduTransmittedVolumeDL` y `Max PRB Policy Ratio`). En la figura 6.24 se muestran los campos del paquete E2AP mencionados.

El otro elemento que genera mensajes E2AP es la xApp, el la figura 6.25 se ven los paquetes capturados durante todo el ciclo de vida de la xApp utilizada en

```
----------------------------------------------------------------

Data Monitoring:
  E2SM_KPM RIC Indication Content:
  -ColletStartTime:  2025-10-25 22:44:21
  -Measurements Data:
  --UE_id: 0
  ---granulPeriod: 1000
  ---Metric: DRB.RlcSduTransmittedVolumeDL, Value: 3.8 [MB]

Control Logic:
 Tx Data Stats:
  UE ID: 0, Max PRB Ratio: n/a, TXed Data [MB]: 19.8
----------------------------------------------------------------

Data Monitoring:
  E2SM_KPM RIC Indication Content:
  -ColletStartTime:  2025-10-25 22:44:22
  -Measurements Data:
  --UE_id: 0
  ---granulPeriod: 1000
  ---Metric: DRB.RlcSduTransmittedVolumeDL, Value: 3.8 [MB]

Control Logic:
 Tx Data Stats:
  UE ID: 0, Max PRB Ratio: n/a, TXed Data [MB]: 23.5
    23.5 MB of data transmitted to UE --> Switch Max PRB limit
    --->Send RIC Control Request to E2 node ID: gnbd_001_001_00019b_0 for UE ID: 0, PRB_min: 1, PRB_max: 39
----------------------------------------------------------------
```

Figura 6.20: Vista de la xApp cuando se alcanza el umbral y se dispara el mecanismo de control

```
------------------------------------------------------------
Client connecting to 10.45.1.2, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  1] local 10.45.1.1 port 42064 connected with 10.45.1.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-1.0000 sec  4.63 MBytes  38.8 Mbits/sec
[  1] 1.0000-2.0000 sec  3.38 MBytes  28.3 Mbits/sec
[  1] 2.0000-3.0000 sec  4.62 MBytes  38.8 Mbits/sec
[  1] 3.0000-4.0000 sec  2.75 MBytes  23.1 Mbits/sec
[  1] 4.0000-5.0000 sec  2.75 MBytes  23.1 Mbits/sec
[  1] 5.0000-6.0000 sec  4.12 MBytes  34.6 Mbits/sec
[  1] 6.0000-7.0000 sec  2.75 MBytes  23.1 Mbits/sec
[  1] 7.0000-8.0000 sec  1.29 MBytes  10.8 Mbits/sec
[  1] 8.0000-9.0000 sec  1.43 MBytes  12.0 Mbits/sec
[  1] 9.0000-10.0000 sec  1.43 MBytes  12.0 Mbits/sec
[  1] 10.0000-11.0000 sec  1.37 MBytes  11.5 Mbits/sec
[  1] 11.0000-12.0000 sec  1.30 MBytes  10.9 Mbits/sec
[  1] 12.0000-13.0000 sec  1.37 MBytes  11.5 Mbits/sec
[  1] 13.0000-14.0000 sec  1.37 MBytes  11.5 Mbits/sec
[  1] 14.0000-15.0000 sec  1.37 MBytes  11.5 Mbits/sec
[  1] 15.0000-16.0000 sec  1.37 MBytes  11.5 Mbits/sec
[  1] 16.0000-17.0000 sec  1.18 MBytes  9.90 Mbits/sec
```

Figura 6.21: Salida del comando Iperf, en el intervalo 7-8 se ve como cae la velocidad de transmisión.

```
----------------------------------------------------------------------

Data Monitoring:
  E2SM_KPM RIC Indication Content:
  -ColletStartTime:  2025-10-25 22:44:25
  -Measurements Data:
  --UE_id: 0
  ---granulPeriod: 1000
  ---Metric: DRB.RlcSduTransmittedVolumeDL, Value: 1.7 [MB]

Control Logic:
 Tx Data Stats:
  UE ID: 0, Max PRB Ratio: 39, Total TXed Data [MB]: 4.0
----------------------------------------------------------------------

Data Monitoring:
  E2SM_KPM RIC Indication Content:
  -ColletStartTime:  2025-10-25 22:44:26
  -Measurements Data:
  --UE_id: 0
  ---granulPeriod: 1000
  ---Metric: DRB.RlcSduTransmittedVolumeDL, Value: 1.6 [MB]

Control Logic:
 Tx Data Stats:
  UE ID: 0, Max PRB Ratio: 39, Total TXed Data [MB]: 5.6
----------------------------------------------------------------------
```

Figura 6.22: Vista de la xApp posterior a la reducción de PRBs asignados.

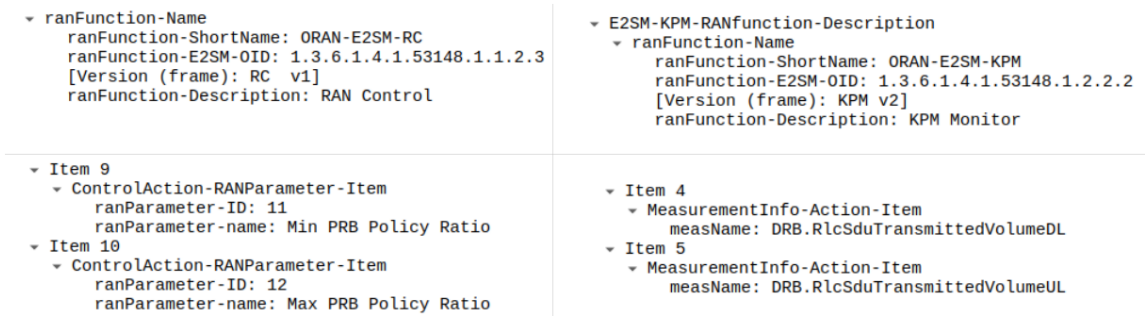

Figura 6.24: En la parte superior izquierda se presenta la definición de E2SM-RC mientras que a la derecha la de E2SM-KPM. En la parte inferior se ven los parámetros asociados a cada E2SM utilizados por la xApp.

No.	Time	Source	Destination	Protocol	Length	Info
462	22.768706402	10.0.2.10	10.0.2.1	E2AP	162	RICsubscriptionRequest
463	22.768808861	10.0.2.1	10.0.2.10	E2AP	114	SACK (Ack=0, Arwnd=16777216) , RICsubscriptionResponse
510	24.046502567	10.0.2.1	10.0.2.10	E2AP	170	RICindication
536	25.218850173	10.0.2.1	10.0.2.10	E2AP	170	RICindication
568	26.315433005	10.0.2.1	10.0.2.10	E2AP	170	RICindication
607	27.415950235	10.0.2.1	10.0.2.10	E2AP	170	RICindication
645	28.590791291	10.0.2.1	10.0.2.10	E2AP	170	RICindication
667	29.636271691	10.0.2.1	10.0.2.10	E2AP	170	RICindication
677	29.638163508	10.0.2.10	10.0.2.1	E2AP	202	SACK (Ack=6, Arwnd=16777216) , RICcontrolRequest
678	29.638479908	10.0.2.1	10.0.2.10	E2AP	102	SACK (Ack=1, Arwnd=16777216) , RICcontrolAcknowledge
743	31.852725041	10.0.2.1	10.0.2.10	E2AP	170	RICindication
789	33.666240637	10.0.2.1	10.0.2.10	E2AP	170	RICindication
841	35.111260159	10.0.2.1	10.0.2.10	E2AP	170	RICindication
883	36.149264098	10.0.2.1	10.0.2.10	E2AP	170	RICindication
911	37.476981529	10.0.2.1	10.0.2.10	E2AP	170	RICindication
957	38.939361198	10.0.2.1	10.0.2.10	E2AP	170	RICindication
999	39.651164323	10.0.2.10	10.0.2.1	E2AP	86	RICsubscriptionDeleteRequest
1000	39.651275770	10.0.2.1	10.0.2.10	E2AP	186	SACK (Ack=2, Arwnd=16777216) , RICindication
1017	39.856399446	10.0.2.1	10.0.2.10	E2AP	86	RICsubscriptionDeleteResponse

Figura 6.25: Captura de parquetes E2AP durante todo el ciclo de vida de la xApp usada en este experimento..

el ejemplo. Los primeros paquetes (462 y 463) muestran la suscripción de la xApp al parámetro `DRB.RlcSduTransmittedVolumeDL` ofrecido por el gNB.

Luego viene una secuencia de mensajes del tipo `RICindication` que son las notificaciones del gNB a la xApp con el valor en ese instante de `DRB.RlcSduTransmittedVolumeDL`. En el paquete 677 se dispara la acción de control (`RICcontrolRequest`) para limitar los PRBs, el siguiente paquete (688) es el ACK por parte del gNB a dicha solicitud. Luego se siguen recibiendo datos hasta que la xApp solicta la desuscripción al gNB (paquetes 999, 1000 y 1017). En este caso no se mostrarán capturas del contenido del paquete ya que Wireshark aun tiene problemas decodificando la información de algunos tipos de mensajes E2AP (faltan definiciones ASN.1 de los E2SM) y la muestra en hexadecimal, de todas formas si se decodifican estas secuencias se obtienen los valores esperados, por ejemplo en un mensaje `RICindication` se ve claramente el valor de `DRB.RlcSduTransmittedVolumeDL` enviado.

Experimento 2

Una de las características destacadas de la implementación 5 es la posibilidad de utilizar UEs COTS y el gNB con SDR. Para demostrar esta capacidad en este experimento se conectará a la red un celular Samsung A33 y se utilizará una xApp de monitoreo para ver el tráfico de este equipo. También se variará de Near-RT RIC (solamente por motivos demostrativos), utilizando en este caso FlexRIC, es importante tener en cuenta que perfectamente se podría utilizar el de la OSC. En la figura 6.26 se representa la arquitectura utilizada.

El primer paso es programar la SIM para conectarse a la red, para esto se utiliza la aplicación de Open-Cells `uicc`, como se mencionó en el capitulo anterior los parámetros que luego deben configurarse en el *core* son el IMSI, KEY, OPC y SPN. A continuación se muestra el comando utilizado y los parámetros grabados en la tarjeta SIM.

```
$ sudo ./program_uicc --adm 12345678 --imsi 555550100001101
    --isdn 00000001 --acc 0001 --key 6874736969202073796
```

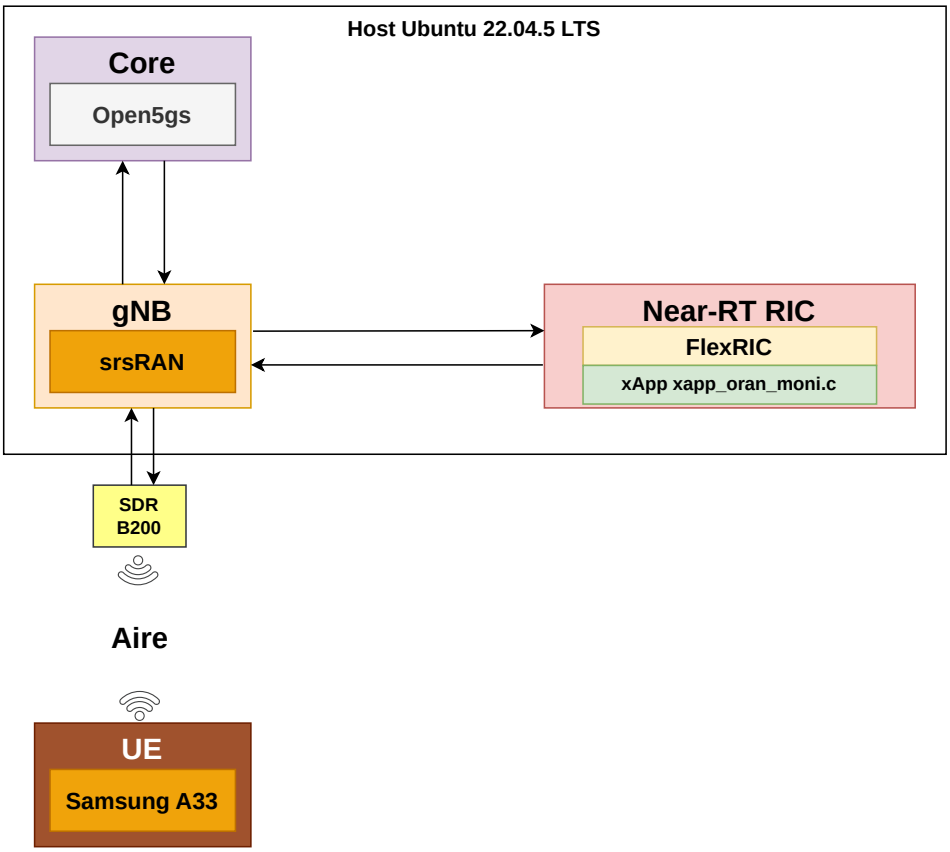

Figura 6.26: Arquitectura del experimento, dentro del host corre el core, gNB y Near-RT RIC, el UE se conecta al gNB de forma inalámbrica a través de un SDR.

```
d4b2079650a73 --opc 504f20634f6320504f50206363500a4f -
spn "ORANnet" --authenticate --noreadafter
```

El siguiente paso es registrar los datos de la SIM en el *core* para permitirle conectarse a la red. Open5GS incluye un administrador web que permite registrar fácilmente nuevos usuarios, el mismo se expone en el puerto 9999 y su usuario y contraseña por defecto son `admin` y `1423` respectivamente. En la página principal agregar un nuevo suscriptor y llenar los datos `IMSI`, `Subscriber Key (K)*` y `Operator Key (OPc/OP)*` con los datos IMSI, KEY y OPC grabados en la SIM. Más abajo en la misma ventana en el campo `DNN/APN*` elegir un nombre para el APN que será configurado en el UE y finalmente en el campo *UE IPv4 Address* la IP que se le asignará. Es importante que el parámetro SPN (nombre de la red) grabado en la SIM coincida con el configurado en el *core* y que la IP asignada al UE sea de la misma red que la configurada en Open5GS.

Una vez finalizada el registro se pueden iniciar el resto de componentes (Near-RT RIC y gNB) como se explicó en el capítulo anterior.

Observación: se puede verificar si el gNB detectó correctamente el SDR con los comandos `uhd_find_devices` y `uhd_usrp_probe`.

Luego de iniciar todos los componentes de la red se puede encender el celular y conectarlo a al red móvil, en este caso ORANnet.

La xApp de monitoreo a utilizar se ejecuta desde el directorio `flexric` con el siguiente comando:

```
$ ./build/examples/xApp/c/monitor/xapp_oran_moni -c ./
   xapp_mon_e2sm_kpm.conf
```

El archivo `xapp_mon_e2sm_kpm.conf` le configura la xApp indicándole la dirección del Near-RT RIC, el *service model* a utilizar y qué métricas solicitar, en este caso `DRB.UEThpDl` y `DRB.UEThpUl` (*throughput* de bajada y subida respectivamente).

Como en los demás experimentos primero se probó la xApp sin transmitir datos (figura 6.27) a modo de control. De los pares de valores mostrados el superior es la velocidad de bajada y el de abajo la velocidad de subida.

```
KPM-v3 ind_msg latency = 494866204418731849 µs from E2-node type 7 ID 411
meas record REAL_MEAS_VALUE value 0.000000
meas record REAL_MEAS_VALUE value 0.000000
KPM-v3 ind_msg latency = 496083415331287348 µs from E2-node type 7 ID 411
meas record REAL_MEAS_VALUE value 0.000000
meas record REAL_MEAS_VALUE value 0.000000
KPM-v3 ind_msg latency = 494331369323230207 µs from E2-node type 7 ID 411
meas record REAL_MEAS_VALUE value 55.000000
meas record REAL_MEAS_VALUE value 21.000000
KPM-v3 ind_msg latency = 496157185691563464 µs from E2-node type 7 ID 411
meas record REAL_MEAS_VALUE value 0.000000
meas record REAL_MEAS_VALUE value 0.000000
```

Figura 6.27: Salida de la xApp de monitoreo cuando no se están transmitiendo datos

Para realizar las pruebas de tráfico se instaló la aplicación *Speed Test* en el celular, esta aplicación mide el máximo *throughput* posible de bajada y subida. En la figura 6.28 se ven los datos capturados por la xApp durante la prueba de bajada y en la figura 6.29 durante la subida.

```
KPM-v3 ind_msg latency = 3140529449917355819 µs from E2-node type 7 ID 411
meas record REAL_MEAS_VALUE value 20459.000000
meas record REAL_MEAS_VALUE value 370.000000
KPM-v3 ind_msg latency = 3139773303746025880 µs from E2-node type 7 ID 411
meas record REAL_MEAS_VALUE value 18943.000000
meas record REAL_MEAS_VALUE value 335.000000
KPM-v3 ind_msg latency = 3139681090799180853 µs from E2-node type 7 ID 411
meas record REAL_MEAS_VALUE value 20735.000000
meas record REAL_MEAS_VALUE value 460.000000
KPM-v3 ind_msg latency = 3139330681598369443 µs from E2-node type 7 ID 411
meas record REAL_MEAS_VALUE value 18245.000000
meas record REAL_MEAS_VALUE value 457.000000
```

Figura 6.28: Salida de la xApp de monitoreo cuando se está evaluando la velocidad de bajada.

```
KPM-v3 ind_msg latency = -831455987577649113 µs from E2-node type 7 ID 411
meas record REAL_MEAS_VALUE value 803.000000
meas record REAL_MEAS_VALUE value 27168.000000
KPM-v3 ind_msg latency = -853587095059476809 µs from E2-node type 7 ID 411
meas record REAL_MEAS_VALUE value 672.000000
meas record REAL_MEAS_VALUE value 21898.000000
KPM-v3 ind_msg latency = -850175215988207494 µs from E2-node type 7 ID 411
meas record REAL_MEAS_VALUE value 666.000000
meas record REAL_MEAS_VALUE value 28157.000000
KPM-v3 ind_msg latency = -869466164676405597 µs from E2-node type 7 ID 411
meas record REAL_MEAS_VALUE value 729.000000
meas record REAL_MEAS_VALUE value 32260.000000
```

Figura 6.29: Salida de la xApp de monitoreo cuando se está evaluando la velocidad de subida.

En ambos casos es clara la diferencia de tráfico medida, además cuando se comparó con los valores obtenidos en la aplicación los mismos coinciden.

Observación: como en todos los casos los archivos de configuración utilizados se encuentran en el repositorio del proyecto.

Experimento 3

En este experimento se combinan los dos anteriores para tener una arquitectura (figura 6.30) que permite tener mas de un gNB en la red móvil, además de combinar el uso de UEs reales con UEs emulados.

En este caso las xApps se suscribirán a métricas de los dos gNBs. En las pruebas realizadas se traficaron datos entre ambos UEs así como entre cada UE e internet.

Conclusiones

A través de los experimentos realizados es posible confirmar la versatilidad de esta implementación, además de que es la mas cercana a la realidad. En este caso se uso un solo host para alojar todos los servicios, pero a partir de estos ejemplos es fácil extrapolar la configuración para tener todos los componentes desagregados. Como se mencionó el tamaño y tipo de despliegue depende de los recursos disponibles. Otro aspecto que se destaca es el de la posibilidad de usar los dos Near-RT RICs más conocidos en la actualidad, sobre todo el oficial de la OSC (recordar que en la implementación 1 daba problemas). Por último mencionar la biblioteca de xApps disponibles, que no solo son útiles para realizar pruebas como se hizo aquí, si no para tomar de ejemplo y punto de partida para el desarrollo de xApps propias.

6.4. Conclusiones del capítulo

En este capítulo se probaron las tres implementaciones consideradas más prometedoras, con el objetivo de ayudar a visualizar la utilidad y el ámbito de aplicación de cada una de ellas. De forma complementaria, también se permite inferir qué

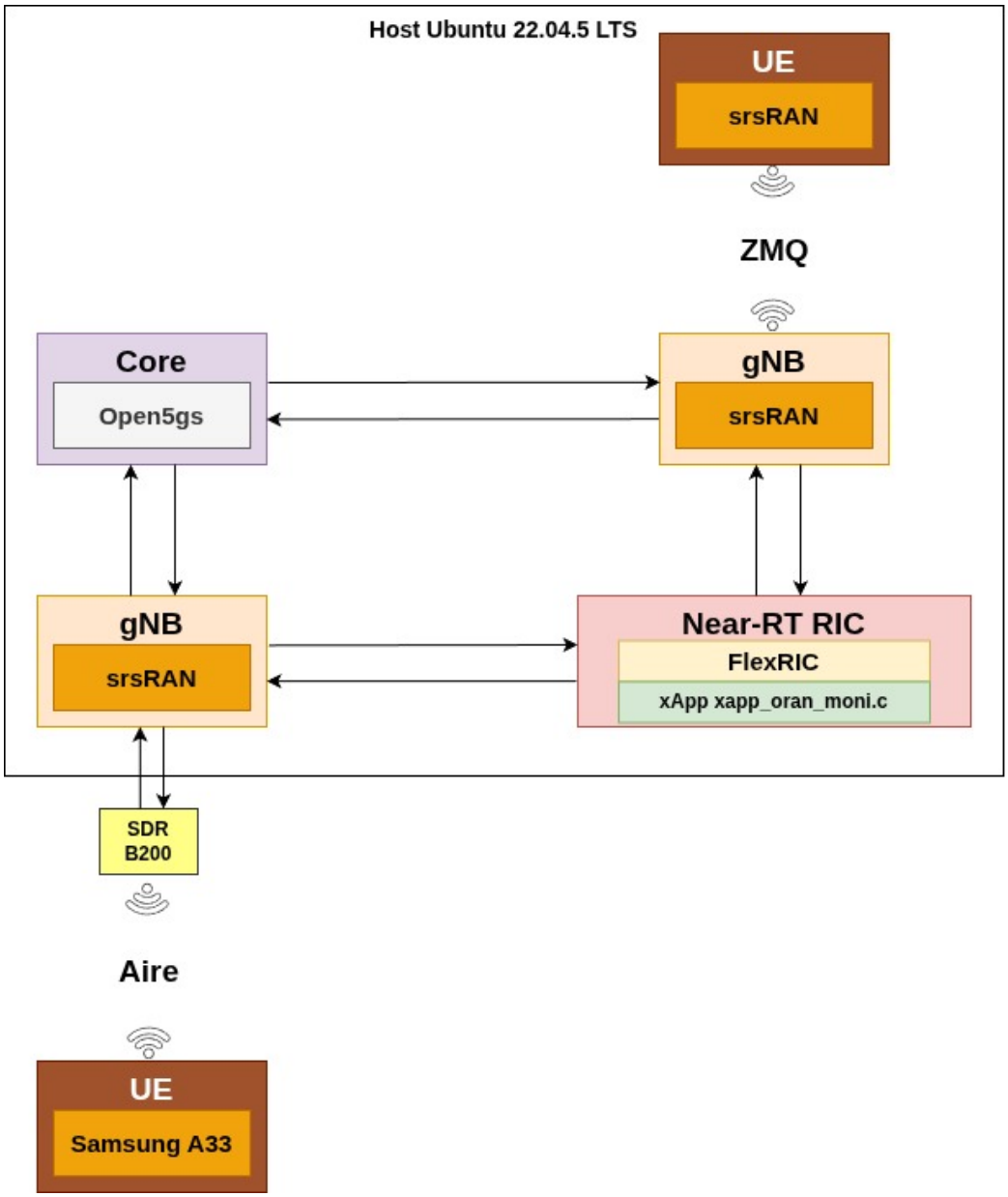

Figura 6.30: Arquitectura del experimento 3 con dos gNB, uno conectado a UEs COTS y otro a UEs emulados

implica la realización de un experimento (dificultad, conocimientos previos necesarios, etc.), así como los resultados que pueden esperarse. Las tres implementaciones analizadas en este capítulo abarcan un espectro que va desde configuraciones sencillas hasta escenarios de alta complejidad, incluyendo entornos completamente emulados, ambientes simulados y despliegues con un alto grado de realismo operativo, lo que permite evaluar su aplicabilidad en distintos niveles de exigencia técnica y contextos de uso. Este capítulo, junto con el anterior, proporciona todo lo necesario para que el investigador se sumerja en la práctica con O-RAN.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 7

Conclusiones y Trabajos Futuros

7.1. Conclusiones

El objetivo de este trabajo es ser la puerta de entrada a muchas otras investigaciones sobre Open RAN. Hasta ahora y como se ve en la cantidad de bibliografía consultada en este documento, interiorizarse en O-RAN era una tarea que requería dedicar mucho tiempo a la recopilación y filtración de información, esto se debe a lo relativamente nuevo que es el tema y a la falta de otros trabajos, sobre todo a nivel nacional.

La idea es que cualquier investigador con conocimientos básicos de redes móviles pueda comprender claramente la motivación de O-RAN, su arquitectura, como interactúan los componentes y que además pueda llevar a la práctica diversos experimentos a través de los *testbeds*.

Para lograr esto se comenzó haciendo un repaso de la evolución de la RAN, desde 2G hasta 6G. Aunque seguramente quienes lleguen a trabajar con O-RAN lo tengan estudiado es importante tenerlo claro porque O-RAN no reinventa la rueda, todo lo que puede usarse del sistema actual se recicla. Luego se presentó al principal promotor: la Open RAN Alliance, y se explicó como trabajan en la redacción de los estándares a traves de los grupos de trabajo, los grupos de enfoque y la OSC.

Una vez que se tuvieron los conceptos previos necesarios se pasa a definir O-RAN, para ello primero se repasaron las definiciones dadas por otros investigadores para luego formular la propia. Con esta definición se buscó englobar todos los conceptos claves de O-RAN de la manera más breve posible.

El capítulo 3 es tal vez uno de los mas importantes de este trabajo, ya que ahí se definió la arquitectura de O-RAN. El capítulo gira entorno a la figura 3.1 donde se muestran todos los elementos de la red móvil 5G tradicional adaptados para O-RAN junto con los RICs y las interfaces. Cada componente e interfaz se definió en detalle ahondando en sus funcionalidades, interacciones y componentes internos, también se explicaron los protocolos y mecanismos de intercambio de información. Finalmente se introdujo lo que aquí se considera una de las mayores oportunidades que brinda O-RAN para la comunidad académica y profesional, la

posibilidad de que la RAN ejecute aplicaciones personalizadas, las xApps y rApps.

Como para realizar cualquier trabajo con O-RAN y sobre todo para desarrollar xApps y rApps no alcanza solo con la teoría desde un principio se consideró que este trabajo debía dar como insumo una plataforma de pruebas junto con la guía práctica de como instalarla y utilizarla. Este trabajo comenzó con lo desarrollado en el capítulo 4 que fue una investigación del *estado del arte* sobre las diferentes plataformas de prueba utilizadas o desarrolladas por otros grupos de investigación, de esta recopilación se llegó a dos conclusiones 1) todos estos *testbeds* aunque distintos se apoyaban en las mismas plataformas de emulación de redes móviles (srsRAN, OAI, Open5GS, etc) por lo que era importante introducirlas en el documento y 2) se vio la necesidad de presentar más de una plataforma de pruebas, no solo porque existían, si no que se entendió que cada una se ajustaba a necesidades diferentes.

Luego de este repaso se paso a efectivamente probar cada uno de estos *testbeds* (capitulo 5) llegando a instalar y probar en profundidad 5 de ellos llamados internamente implementaciones. La implementación 1 y 2 son bancos de pruebas emulados desarrollados por el NIST basados en srsRAN, OAI, Open5GS, FlexRIC y el RIC de la OSC con la característica de ser auto desplegables a través de un script dado. Esto permite probar casi todas las configuraciones experimentales posibles de manera sencilla y con poca configuración. Es ideal para dar los primeros pasos y hacer una evaluación rápida sobre qué conjunto de herramientas es más conveniente para el caso. Aquí también se vio el que tal vez sea el principal problema en la actualidad de estos sistemas de prueba que es la sensibilidad a los cambios de versiones de sus componentes.

La implementación 3 es un caso particular e interesante ya que es una combinación de un Near-RT RIC real (emulado) conectado a una red móvil simulada en ns3. Es verdad que hoy en día los componentes que conforman el *testbed* están desactualizados pero la forma de encarar el problema puede resultar útil para aquellos experimentos donde lo importante sea probar xApps reales en entornos complejos de emular, por ejemplo casos con muchos UEs.

La siguiente implementación vista, la 4, también fue desarrollada por el NIST pero esta vez como un módulo para ns3. Esto permite realizar experimentos completamente simulados. Como se mencionó en el capítulo 5 esto permite realizar pruebas que de forma emulada sería muy complejo (varios UEs, varios eNBs, dispositivos moviéndose, condiciones especiales del canal, etc). El problema es que el módulo implementa la lógica de O-RAN, pero no implementa literalmente O-RAN. Por ejemplo las métricas solicitadas por una xApp son solicitadas por el Near-RT RIC a través de una interfaz con los nodos E2, pero ese intercambio no utiliza el protocolo E2AP ni los modelos de servicio definidos. Por eso este *testbed* es útil para probar casos de uso, lógicas y hacer pruebas de concepto de xApps, pero las mismas no son directamente trasladables a un sistema real.

Finalmente se presentó la implementación 5, este es un *testbed* emulado que utiliza las mismas herramientas que las implementaciones 1 y 2 pero en esta oportunidad se instala cada una por separado. Esto complejiza la instalación y configuración pero se gana libertad para personalizar la plataforma de pruebas. Puede

instalarse todo en un equipo, cada componente en un equipo independiente o mezclarlos como se requiera. Además permite personalizar la configuración para utilizar UEs emulados o reales, así como cambiar casi cualquier configuración de radio, seguridad, etc. Aquí que tanto acercarse a la realidad va a depender de los recursos disponibles.

De todos las plataformas de prueba se realizó un inventario de las xApps de ejemplo que traen, esto es importante no solo para probarlas y ver analizar los procesos que ejecutan, si no que sirven como punto de partida para el desarrollo de nuevas xApps.

Es importante recalcar que hay que tener en cuenta los cambios de versiones, seguramente en algún tiempo los pasos indicados para realizar los despliegues tengan variaciones que sería útil tener identificadas y documentadas.

No es menor mencionar también que el gran ausente en todos los *testbeds* es el SMO y el Non-RT RIC, solo se encontró una aplicación limitada para la implementación 1. Aunque la OSC se encuentra trabajando y si se revisan sus repositorios hay avances y versiones del SMO y Non-RT RIC se entiende que a las mismas les falta madurez para integrarse con el resto del ecosistema. Sin duda que uno de los trabajos a futuro es la integración del SMO y Non-RT RIC una vez que exista una aplicación utilizable.

Para facilitar la comprensión de las diferentes plataformas, analizar que se puede esperar de cada una y colateralmente ver el funcionamiento de O-RAN, en el capítulo 6 se tomaron las implementaciones 2 4 y 5 y se realizaron algunos experimentos (casos de uso) en ellas. En el caso de uso 1 se emuló una red móvil con un gNB y 3 UEs. En dicha red se dejo corriendo una xApp de monitoreo que releva varias métricas de la comunicación entre los UEs y el gNB, se experimentó variando el tráfico (*throughput* y dirección) y se verificó que los datos mostrados sean coherentes.

En el caso de uso 2 se simuló una red con 3 eNBs y varios UEs moviéndose en un área designada, la simulación se armó para que los UEs realicen el *handover* clásico de LTE por un lado y un *handover* por distancia geométrica a las radio bases utilizando una xApp. Este ejemplo muestra que aunque el sistema simulado no sea trasladable al mundo real, la versatilidad de experimentos que se pueden realizar hace que merezca la pena utilizarlo.

Finalmente en el caso de uso 3 se utilizo la implementación 5 para realizar dos experimentos, uno con un UE emulado y otro con un UE real. En el primer caso se utilizó una xApp que evaluaba métricas de la comunicación y en base a esos valores tomaba la decisión de limitar el tráfico del UE, en la segunda se monitorearon valores de tráfico del UE real a través de una xApp de monitoreo.

En todos los casos se explicó paso a paso como se desplegaron los casos de uso para que puedan reproducirse fácilmente. Es importante aclarar que el objetivo de estos experimentos no es el experimento en si (por ejemplo no se analizó cual tipo de *handover* era mejor), si no que la idea era mostrar experimentos variados con cada implementación para demostrar la utilidad de cada una y que se puede llegar a esperar de ellas.

O-RAN ofrece un mundo de oportunidades para los investigadores, este es solo

el primer paso. Al terminar este documento uno habrá entendido O-RAN y será capaz de realizar pruebas básicas.

Es difícil predecir con certeza cuál será el futuro de O-RAN y hasta qué punto logrará consolidarse dentro del ecosistema móvil global. Su adopción dependerá de factores técnicos, económicos y regulatorios que aún están en evolución, así como de la disposición de la industria a adoptar modelos más abiertos y colaborativos. Sin embargo, los avances recientes, el crecimiento de los sistemas abiertos y el interés creciente de la comunidad científica indican que O-RAN tiene el potencial de transformar de manera significativa la forma en que se diseñan, despliegan y operan las redes móviles. En este sentido, los resultados y experiencias presentados en esta tesis buscan aportar una base sólida para futuros desarrollos, investigaciones y pruebas que continúen explorando las posibilidades de esta arquitectura emergente.

7.2. Trabajos Futuros

Si se piensa el inicio en la investigación en O-RAN como una trilogía, este sería su primera parte. Como se dijo en mas de una oportunidad el gran potencial está en las xApps y rApps, por eso la evolución natural de este proyecto (su "segunda parte") es la realización de un instructivo dedicado al desarrollo de xApps y rApps, profundizando en que aspectos tener en cuenta, como se despliegan, como hacer que interactúen entre ellas, etc. Al momento de escribir este documento un grupo de estudiantes de grado [10] se encuentra trabajando en un proyecto dedicado a las xApps tomando como punto de partida este trabajo. Los estudiantes están trabajando en el diseño, desarrollo e implementación de xApps, desde sencillas aplicaciones de monitoreo hasta la aplicación de algoritmos de *machine learning*. Además están tratando de adaptar la implementación 4 para que soporte 5G y no solo LTE como lo hace actualmente. Aquí se destaca que esta tesis fue el punto de partida para dichos estudiantes.

La trilogía se puede completar con el desarrollo de una plataforma de pruebas de gran envergadura, aunque aquí las limitaciones son sobre todo presupuestales ya que mínimamente se necesitarían servidores de grado empresarial, varios SDRs, red de fibra, etc. Lo positivo es que como se vio en software el gasto sería nulo, al software utilizado en este trabajo se podría sumar un hipervisor como Proxmox[1] que facilitaría enormemente la administración y viene con algunas características que pueden resultar de utilidad como *software defined networks* nativa, conexión directa de hardware a maquinas virtuales, gestión de usuarios, entre otras o ir directamente por la arquitectura DevOps mediante Docker y/o Kubernetes. En este trabajo se muestra como tener plataformas "personales", fáciles de desplegar y con un gran potencial, pero para pasar al siguiente nivel es indispensable un despliegue que se aproxime al caso real de una operadora.

Por otra parte, resulta fundamental incorporar O-RAN en la capacitación de los futuros profesionales del área, ya que la falta de recursos humanos debidamente formados frente a un cambio tecnológico de esta magnitud puede representar una

[1] https://proxmox.com/en/

barrera significativa para su adopción. La carencia de conocimiento especializado no solo retrasa la implementación de estas tecnologías, sino que también puede relegar al país en términos de competitividad, innovación y desarrollo dentro del ecosistema de las telecomunicaciones. En este aspecto se utilizó la implementación 2 de este trabajo para desarrollar un laboratorio sobre O-RAN en la asignatura Redes de Acceso, siendo la primer experiencia con O-RAN en la enseñanza de grado en esta institución.

O-RAN no estará sola en el ecosistema de las telecomunicaciones, sino que deberá convivir e integrarse con otras redes existentes, por lo que investigar sobre este tópico también resulta fundamental. En este sentido, un estudiante de posgrado [9] se basa en los resultados preliminares de esta tesis con el objetivo de crear un *testbed* de O-RAN utilizando la red GPON como *midhaul*. Dicho *testbed* debió ser implementado en una institución extranjera, ya que no se cuenta con el equipamiento adecuado en la UdelaR.

El campo de investigación asociado a O-RAN es amplio y tiene múltiples líneas de trabajo futuras. Entre ellas se destacan la incorporación de nuevos casos de uso, como aplicaciones de realidad virtual, desarrollo de *xApps*, escenarios de IoT y experimentación con *network slicing*, una de las capacidades más prometedoras de las redes 5G y 6G. Asimismo, resulta de interés profundizar en mecanismos de control automático de la red (*closed-loop control*), así como en la aplicación de técnicas de inteligencia artificial y aprendizaje automático para optimización dinámica de recursos, predicción de tráfico y detección de anomalías. Por otra parte, se identifican como áreas clave de investigación el análisis de seguridad en arquitecturas O-RAN, la interoperabilidad entre proveedores, la evaluación de métricas de calidad de servicio y experiencia de usuario, y el estudio de eficiencia energética bajo distintos escenarios de operación.

Independientemente de la línea de investigación que se desee abordar en el ámbito de O-RAN, existe un punto en común fundamental: la necesidad de comprender en profundidad el funcionamiento de su arquitectura y disponer de una plataforma donde sea posible realizar pruebas y validaciones experimentales. En este sentido, el presente trabajo se posiciona como un punto de partida transversal, proporcionando tanto los fundamentos conceptuales como una infraestructura básica que habilita futuras investigaciones y desarrollos sobre O-RAN.

En resumen, si se va a trabajar con O-RAN, empezar por aquí!

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] O-RAN Alliance. Especificaciones de los grupos de trabajo de la o-ran alliance. URL: https://specifications.o-ran.org/specifications.

[2] O-RAN Alliance. Governance of o-ran alliance e.v. in compliance with wto principles. Technical report, O-RAN Alliance, 2023. Accedido: octubre 24, 2024. URL: https://mediastorage.o-ran.org/white-papers/O-RAN.O-RAN-ALLIANCE-in-Compliance-with-WTO-Principles-white-paper-2023-07-v02.pdf.

[3] O-RAN Alliance. Copyright © by the o-ran alliance e.v. 1 o-ran alliance overview. Technical report, O-RAN Alliance, 2024. Accedido: octubre 24, 2024. URL: https://mediastorage.o-ran.org/white-papers/O-RAN.Overview-of-the-O-RAN-ALLIANCE-presentation.pdf.

[4] O-RAN Alliance. Sitio web de la o-ran alliance, 2025. URL: https://www.o-ran.org/.

[5] Ruan P. Alves, João Guilherme A. da S. Alves, Mikael R. Camelo, Wilker O. de Feitosa, Victor F. Monteiro, and Francisco Rodrigo P. Cavalcanti. Experimental comparison of 5g sdr platforms: srsran x openairinterface. *arXiv preprint arXiv:2406.01485*, 2024. Disponible en línea, consultado en arXiv.

[6] Paramvir Bahl, Matthew Balkwill, Xenofon Foukas, Anuj Kalia, Daehyeok Kim, Manikanta Kotaru, Zhihua Lai, Sanjeev Mehrotra, Bozidar Radunovic, Stefan Saroiu, Connor Settle, Ankit Verma, Alec Wolman, Francis Y. Yan, and Yongguang Zhang. Accelerating open ran research through an enterprise-scale 5g testbed. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '23, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3570361.3615745.

[7] Maria Katarine Santana Barbosa, Marcelo Silva, Ednelson Cavalcanti, and Kelvin Dias. Open-source 5g core platforms: A low-cost solution and performance evaluation. *arXiv preprint arXiv:2412.21162*, 2024. Compara Open5GS, Free5GC y OAI en testbeds 5G SA; latencia, rendimiento y consumo de recursos.

[8] Fransiscus Asisi Bimo, Ferlinda Feliana, Shu-Hua Liao, Chih-Wei Lin, David F. Kinsey, James Li, Rittwik Jana, Richard Wright, and Ray-Guang

Cheng. OSC Community Lab: The Integration Test Bed for O-RAN Software Community . In *2022 IEEE Future Networks World Forum (FNWF)*, pages 513–518, Los Alamitos, CA, USA, October 2022. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/FNWF55208.2022.00096, doi:10.1109/FNWF55208.2022.00096.

[9] Valentín Brienza. Trabajo en desarrollo. Trabajo en desarrollo, 2025.

[10] Melgar Cerecetto, Conti. Trabajo en desarrollo. Trabajo en desarrollo, 2025.

[11] Yi Chu, Mostafa Rahmani, Josh Shackleton, David Grace, Kanapathippillai Cumanan, Hamed Ahmadi, and Alister Burr. Testbed development: An intelligent o-ran-based cell-free mimo network. *IEEE Communications Magazine*, 63(6):74–81, 2025. Licensed under CC BY 4.0. doi:10.1109/MCOM.001.2400574.

[12] Ramraj Dangi, Gaurav Choudhary, Nicola Dragoni, Praveen Lalwani, Utkarsh Khare, and Souradeep Kundu. 6g mobile networks: Key technologies, directions, and advances. *Telecom*, 4(4):836–876, 2023.

[13] Tri Nhu Do. AI-Open-RAN for Non-Terrestrial Networks. *arXiv preprint arXiv:2511.11947*, 2025. URL: https://arxiv.org/abs/2511.11947.

[14] Parallel Wireless Eugina Jordan. Open ran functional splits, explained, 2021. URL: https://www.5gtechnologyworld.com/open-ran-functional-splits-explained/.

[15] Andrew E. Ferguson, Ujjwal Pawar, Tianxin Wang, and Mahesh K. Marina. Campus5G: A Campus Scale Private 5G Open RAN Testbed. *arXiv preprint arXiv:2506.23740*, 2025. URL: https://arxiv.org/abs/2506.23740.

[16] Jorge Gallo. Gestión integral de redes y servicios de telecomunicaciones [material del curso], 2023. Facultad de Ingeniería, Universidad de la República.

[17] Wesley Garey, Richard A. Rouil, Evan Black, Tanguy Ropitault, and Weichao Gao. O-ran with machine learning in ns-3, 2023-06-28 04:06:00 2023. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=936291, doi:10.1145/3592149.3592157.

[18] Sridhar Rajagopal Rittwik Jana Ian C. Wong, Aditya Chopra. *Open RAN The Definitive Guide*. Wiley, New Jersey, USA, 1 edition, 2024.

[19] Jesús Cáceres Jimenez. Planificación radioeléctrica de una red lte, 2014. URL: https://biblus.us.es/bibing/proyectos/abreproy/90018/fichero/TFG_Jes%C3%BAs_C%C3%A1ceres_Jim%C3%A9nez.pdf.

[20] Zoran Blazevic Josip Lorincz, Amar Kukuruzovic. A comprehensive overview of network slicing for improvingthe energy efficiency of fifth-generation networks. *Sensor*, 24(3242), 2024. URL: https://doi.org/10.3390/s24103242.

[21] Florian Kaltenberger, Tommaso Melodia, Irfan Ghauri, Michele Polese, Raymond Knopp, Tien Thinh Nguyen, Sakthivel Velumani, Davide Villa, Leonardo Bonati, Robert Schmidt, Sagar Arora, Mikel Irazabal, and Navid Nikaein. Driving innovation in 6g wireless technologies: The openairinterface approach. *arXiv preprint arXiv:2412.13295*, 2024. Incluye descripción teórica de la arquitectura, componentes y visión hacia 6G para OAI.

[22] Florian Kaltenberger, Tien-Thinh Nguyen, Navid Nikaein, Raymond Knopp, and otros. Openairinterface: Democratizing innovation in the 5g era. *Computer Networks*, 176:107284, 2020. doi:10.1016/j.comnet.2020.107284.

[23] Xhafer Krasniqi, Edmond Hajrizi, and Besnik Qehaja. Challenges and lessons learned during private 5g open ran deployments. In *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pages 1–6, 2023. doi:10.1109/ICECCME57830.2023.10252662.

[24] Flavius-Filip Lacatus, Michael Siller, Luca Mastri, Noelia Gay, Genaro Vargas-Schaffer, and Georges Kaddoum. Development of Open Radio Access Networks (O-RAN) for Real-Time Robotic Teleoperation. *arXiv preprint arXiv:2502.14728*, 2025. URL: https://arxiv.org/abs/2502.14728.

[25] Andrea Lacava, Michele Polese, Rajarajan Sivaraj, Rahul Soundrarajan, Bhawani Shanker Bhati, Tarunjeet Singh, Tommaso Zugno, Francesca Cuomo, and Tommaso Melodia. Programmable and customized intelligence for traffic steering in 5g networks using open ran architectures. *IEEE Transactions on Mobile Computing*, pages 1–16, 2023. doi:10.1109/TMC.2023.3266642.

[26] Line M. P. Larsen, Aleksandra Checko, and Henrik L. Christiansen. A survey of the functional splits proposed for 5g mobile crosshaul networks. *IEEE Communications Surveys & Tutorials*, 21(1):146–172, 2019. doi:10.1109/COMST.2018.2868805.

[27] Chang Liu, Gopalasingham Aravinthan, Ahan Kak, and Nakjung Choi. Tinyric: Supercharging o-ran base stations with real-time control. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '23, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3570361.3615743.

[28] Peng Liu, Kyehwan Lee, Fernando Cintron, Simeon Wuthier, Bhadresh Savaliya, Douglas Montgomery, and Richard Rouil. Blueprint for deploying 5g o-ran testbeds: A guide to using diverse o-ran software stacks, 2024-10-23 04:10:00 2024. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=958753, doi:10.6028/NIST.TN.2311.

[29] Stefano Maxenti, Ravis Shirkhani, Maxime Elkael, Leonardo Bonati, Salvatore D'Oro, Tommaso Melodia, and Michele Polese. AutoRAN: Automated and Zero-Touch Open RAN Systems. *arXiv preprint arXiv:2504.11233*, 2025. URL: https://arxiv.org/abs/2504.11233.

[30] Pawan Meena, Monti Babulal Pal, Praphula Kumar Jain, and Rajendra Pamula. 6g communication networks: Introduction, vision, challenges, and future directions. *Wireless Personal Communications*, 125(3):1097–1123, 2022.

[31] Salvatore D'Oro Stefano Basagni y Tommaso Melodia Michele Polese, Leonardo Bonati. Understanding o-ran: Architecture, interfaces,algorithms, security, and research challenges. *IEEE Comumunications Surveys and Tutorials*, 25(2):1376–1411, 2023. URL: https://doi.org/10.1109/COMST.2023.3239220.

[32] Joshua Moore, Aly Sabri Abdalla, Charles Ueltschey, and Vuk Marojevic. Prototyping o-ran enabled uav experimentation for the aerpaw testbed, 2024. URL: https://arxiv.org/abs/2411.04027, arXiv:2411.04027.

[33] Ghizlane Mountaser, Maliheh Mahlouji, and Toktam Mahmoodi. Latency bounds of packet-based fronthaul for cloud-ran with functionality split, 2019. URL: https://arxiv.org/abs/1904.13108, arXiv:1904.13108.

[34] OSC O-RAN Alliance. Documentación de la osc. URL: https://docs.o-ran-sc.org/en/latest/.

[35] Benetel Olli Andersson. Functional splits: the foundation of an open 5g ran, 2021. URL: https://www.5gtechnologyworld.com/functional-splits-the-foundation-of-an-open-5g-ran/.

[36] Seizo Onoe. Open the new world of 5g. *IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pages 5–8, 2018. URL: 10.1109/ASSCC.2018.8579288.

[37] Open5GS Project. Open5gs documentation. https://open5gs.org/open5gs/docs/, 2023. Documentación oficial de Open5GS; funciones implementadas, guías de configuración, ejemplos con UERANSIM etc.

[38] OpenAI. Chatgpt, 2025. Accedido: agosto 2025. URL: https://chat.openai.com.

[39] OpenAirInterface Software Alliance. Openairinterface documentation. https://openairinterface.org/oai-resources/, 2023. Documentación oficial del proyecto OAI, consultado en línea.

[40] Amogh PC, Nagamuthu Vignesh, Yiyang Pei, Neelakantam Venkatarayalu, Pedro Henrique Amorim Rezende, Shyam Babu Mahato, and Sumei Sun. FCT O-RAN: Design and Deployment of a Multi-Vendor End-to-End Private 5G Testbed. *arXiv preprint arXiv:2509.01891*, 2025. URL: https://arxiv.org/abs/2509.01891.

[41] F Pedreira and B. Tió. Portaran: Implementación de un demostrador portable de red móvil 5g basado en srsran. Tesis de grado, Universidad de la República (Uruguay), Facultad de Ingeniería, 2025. En español.

[42] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. ColO-RAN: Developing Machine Learning-based xApps for Open RAN Closed-loop Control on Programmable Experimental Platforms. *IEEE Transactions on Mobile Computing*, pages 1–14, July 2022.

[43] Somreeta Pramanik, Adlen Ksentini, and Carla Fabiana Chiasserini. Characterizing the computational and memory requirements of virtual rans. *arXiv preprint arXiv:2201.00673*, 2022. Test-bed virtual RAN usando srsRAN; estudio de requerimientos de cómputo y memoria.

[44] Frank Rayal. Lte in a nutshell protocol architecture, 2017. URL: `https://frankrayal.com/wp-content/uploads/2017/02/LTE-in-a-Nutshell-Protocol-Architecture.pdf`.

[45] Wim Rouwet. *Open Radio Access Network (O-RAN) Systems Architecture and Design*. Academic Press, Elsevier, Londres, UK, 1 edition, 2022.

[46] Leonardo Lo Schiavo, Gines Garcia-Aviles, Andres Garcia-Saavedra, Marco Gramaglia, Marco Fiore, Albert Banchs, and Xavier Costa-Perez. Cloudric: Open radio access network (o-ran) virtualization with shared heterogeneous computing. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '24, page 558–572, New York, NY, USA, 2024. Association for Computing Machinery. `doi:10.1145/3636534.3649381`.

[47] Robert Schmidt, Mikel Irazabal, and Navid Nikaein. Flexric: an sdk for next-generation sd-rans. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '21, page 411–425, New York, NY, USA, 2021. Association for Computing Machinery. `doi:10.1145/3485983.3494870`.

[48] Alberto Leon-Garcia Simona Marinova. Intelligent o-ran beyond 5g: Architecture,use cases, challenges, and opportunities. *IEEE Access*, 12:27088–27114, 2024. URL: `10.1109/ACCESS.2024.3367289`.

[49] Jaspreet Singh, Gurpreet Singh, and Niyati Vashisht. Evaluating 6g network technology principles and applications: A review. *3rd International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)*, pages 1–7, 2023.

[50] Software Radio Systems (SRS). srsran documentation. `https://docs.srsran.com/`, 2023. AGPLv3, documentación oficial de srsRAN.

[51] Nassima Toumi and Toni Dimitrovski. Ai-native architecture for 6g networks and services with model dependencies. *2024 European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 901–904, 2024.

[52] Pratheek S. Upadhyaya, Aly S. Abdalla, Vuk Marojevic, Jeffrey H. Reed, and Vijay K. Shah. Prototyping next-generation o-ran research testbeds with sdrs, 2022. URL: https://arxiv.org/abs/2205.13178, arXiv:2205.13178.

[53] Pratheek S. Upadhyaya, Nishith Tripathi, Joseph Gaeddert, and Jeffrey H. Reed. Open ai cellular (oaic): An open source 5g o-ran testbed for design and testing of ai-based ran management algorithms. *IEEE Network*, 37(5):7–15, 2023. doi:10.1109/MNET.2023.3320933.

[54] Davide Villa, Imran Khan, Florian Kaltenberger, Nicholas Hedberg, Rúben Soares Da Silva, Anupa Kelkar, Chris Dick, Stefano Basagni, Josep M. Jornet, Tommaso Melodia, Michele Polese, and Dimitrios Koutsonikolas. An open, programmable, multi-vendor 5g o-ran testbed with nvidia arc and openairinterface. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2024. doi:10.1109/INFOCOMWKSHPS61880.2024.10620908.

[55] Parallel Wireless. 5g functional splits. URL: https://www.parallelwireless.com/wp-content/uploads/5G-Functional-Splits-V3.pdf.

[56] N. Li Z J. Huang H. Ding C.L. Y. Huang, Q. Sun. Validation of current o-ran technologies and insights on the future evolution. *IEEE*, 42(2):487–505, 2024. URL: https://doi.org/10.3390/s24103242.

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.
Compilado el miércoles 4 febrero, 2026.
http://iie.fing.edu.uy/