



UNIVERSIDAD DE LA REPÚBLICA
FACULTAD DE INGENIERÍA



Adaptación Dinámica de Network Slicing en Transporte de Redes 5G Mediante SDN: Diseño, Implementación y Análisis Experimental

TESIS PRESENTADA A LA FACULTAD DE INGENIERÍA DE LA
UNIVERSIDAD DE LA REPÚBLICA POR

Pablo Bertrand

EN CUMPLIMIENTO PARCIAL DE LOS REQUERIMIENTOS
PARA LA FINALIZACIÓN DE LA CARRERA DE
MAESTRÍA EN INGENIERÍA ELÉCTRICA.

DIRECCIÓN DE TESIS

Claudina Rattaro Universidad de la República
Federico La Rocca Universidad de la República

TRIBUNAL

Germán Capdehourat Universidad de la República
Javier Carmona Universidad de Extremadura
José Joskowicz Universidad de la República

DIRECCIÓN ACADÉMICA

Claudina Rattaro Universidad de la República

Montevideo
23 de diciembre, 2025

Adaptación Dinámica de Network Slicing en Transporte de Redes 5G Mediante SDN: Diseño, Implementación y Análisis Experimental, Pablo Bertrand.

ISSN 1688-2806

Esta tesis fue preparada en L^AT_EX usando la clase iietesis (v1.2).

Contiene un total de 150 páginas.

Compilada el martes 3 febrero, 2026.

<http://iie.fing.edu.uy/>

Agradecimientos

En primer lugar, quiero agradecer a Claudina Rattaro por su rol en la dirección de esta tesis: por su gran disposición en todas las etapas, paciencia en las definiciones de alcance y claridad para enmarcar y abordar los problemas técnicos. También destaco su esfuerzo en la gestión y acompañamiento de la pasantía, que se pudo adaptar a mis tiempos y que fue una motivación importante para este trabajo.

Agradezco a Federico La Rocca por su parte en la dirección académica desde el comienzo de la maestría y por sus aportes en el área del aprendizaje automático.

Quiero agradecer también a la Universidad de Extremadura y a David Cortés-Polo por recibirme en su equipo y darme un lugar en el desarrollo de su investigación. Agradezco especialmente a Antonio Hidalgo por todo su apoyo durante y luego de terminada la pasantía.

Agradezco al Centro Universitario Regional del Este y a Juan Navarro por proveer la infraestructura usada en la última etapa de este trabajo, sin la cual no habría sido posible implementar un sistema de estas dimensiones.

Por último, un agradecimiento especial a mi familia, por sostener y acompañar este proceso.

Esta página ha sido intencionalmente dejada en blanco.

A Helena.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

Este trabajo presenta el diseño, implementación y evaluación de un testbed 5G orientado al estudio de la asignación dinámica de recursos en el dominio de transporte, combinando técnicas de network slicing y software-defined networking. El objetivo es analizar cómo distintos mecanismos de adaptación pueden modificar el ancho de banda de segmentos de red con diferentes requerimientos de servicio, para aprovechar los recursos no utilizados por la red sin dejar de cumplir con los niveles de servicio establecidos.

El testbed integra un núcleo 5G, nodos de acceso radio y una población configurable de dispositivos de usuario, junto con un plano de transporte programable y controlado por software. La arquitectura, totalmente basada en herramientas de código abierto, permite escalar el número de nodos de acceso y de segmentos de red, y proporciona un entorno unificado para monitorear, clasificar y manipular flujos en tiempo real. La primera contribución del trabajo es la construcción y documentación de esta plataforma, que reproduce el comportamiento de una red 5G y permite evaluar la asignación de recursos y la respuesta dinámica ante cambios de carga.

Sobre esta infraestructura se desarrollan dos mecanismos alternativos de adaptación. El enfoque reactivo, basado en umbrales simples, ajusta los recursos de cada segmento por escalones ante variaciones de consumo. El enfoque predictivo usa modelos de aprendizaje automático, entrenados con datos obtenidos del sistema a partir de tráfico realista, para anticipar el throughput de corto plazo. Se diseñan funciones de pérdida asimétricas para orientar que los eventuales errores por la aleatoriedad se den por sobre-estimación y se analiza su impacto en el ancho de banda reservado (para aprovechar el excedente) y la proporción de sub-predicción (para cumplir las garantías del servicio prioritario). Los modelos desarrollados se integran en un ciclo operativo de medición, predicción y reconfiguración ejecutado cada segundo.

Los resultados confirman que el sistema aísla correctamente el tráfico entre segmentos y que los modelos predictivos permiten estimar el throughput de corto plazo en segmentos con demanda variable, respondiendo a los cambios en la carga del transporte sin afectar al resto de la red. Además, el escalamiento a múltiples nodos y a un mayor número de dispositivos muestra que el testbed puede servir para evaluar estrategias de gestión distribuida de recursos.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Agradecimientos	I
Resumen	V
Glosario	XI
1. Introducción	1
1.1. Motivación	2
1.1.1. ¿Por qué Network Slicing?	3
1.1.2. Necesidad de experimentación	4
1.2. Contribuciones	5
1.3. Sobre el documento	6
2. Network slicing	7
2.1. Arquitectura de alto nivel	8
2.2. Definiciones y estandarización	9
2.3. Orquestación y ciclo de vida	12
2.3.1. Ciclo de vida del slice	12
2.3.2. Coordinación	13
2.3.3. Procedimientos de gestión	13
2.4. Slicing por dominio	15
2.4.1. Slicing en la RAN	15
2.4.2. Slicing en el núcleo	16
2.4.3. Slicing en la red de transporte	16
2.5. Slices: ¿estáticas o dinámicas?	17
2.6. Alcance y evaluación	18
3. Antecedentes de testbeds y posicionamiento de la propuesta	19
3.1. Revisiones bibliográficas	19
3.2. Testbeds de slicing en RAN y núcleo	22
3.3. Testbeds de slicing en transporte	25
3.4. Comparación y diferenciales	29
4. Tecnologías y herramientas utilizadas	31
4.1. Núcleo de red 5G	31
4.1.1. Fundamentos y arquitectura	32

Tabla de contenidos

4.1.2.	Funciones de red	33
4.1.3.	Herramienta elegida	36
4.2.	Acceso radio 5G	37
4.2.1.	Fundamentos de radio	37
4.2.2.	Componentes de la red	39
4.2.3.	Herramientas elegidas	43
4.3.	Virtualización	44
4.3.1.	Fundamentos	44
4.3.2.	Herramienta elegida	45
4.4.	Redes definidas por software	45
4.4.1.	Fundamentos	45
4.4.2.	Herramienta elegida	46
4.5.	Switching	46
4.5.1.	Fundamentos	46
4.5.2.	Herramienta elegida	47
4.6.	Resumen de componentes elegidos	47
5.	Solución base implementada	49
5.1.	Arquitectura del testbed	50
5.1.1.	Configuración	51
5.1.2.	Generación de tráfico	53
5.2.	Adaptación dinámica de slices	54
5.2.1.	Adaptación reactiva: Umbrales	54
5.2.2.	Adaptación predictiva: LSTM	57
6.	Escalamiento de la solución	63
6.1.	Arquitectura del testbed	65
6.1.1.	Configuración	66
6.1.2.	Recursos	70
6.2.	Perfiles de tráfico	71
6.2.1.	Tráfico de video	72
6.2.2.	Tráfico de audio	73
6.2.3.	Tráfico de navegación web	73
6.3.	Adaptación dinámica de slices	74
6.3.1.	Criterios por slice	74
6.3.2.	Generación de datasets	76
6.3.3.	Entrenamiento y evaluación de modelos LSTM	79
6.3.4.	Predicción Conformal	87
6.4.	Implementación en el sistema completo	91
7.	Conclusiones y trabajo futuro	95
7.1.	Conclusiones	95
7.2.	Trabajo futuro	96

A. Guía de configuración del testbed	99
A.1. Introducción	99
A.2. Configuración de Proxmox	99
A.3. Configuración de VMs	101
A.4. Configuración de Software	103
A.4.1. OPEN5GS	103
A.4.2. OpenDayLight	109
A.4.3. OVS	110
A.4.4. UERANSIM	113
A.4.5. srsRAN	117
A.4.6. Configuración UE físico	118
A.4.7. Interacción externa con ODL	119
Referencias	121
Índice de tablas	129
Índice de figuras	130

Esta página ha sido intencionalmente dejada en blanco.

Glosario

- 3GPP** 3rd Generation Partnership Project. 1
- 4G** Fourth Generation Mobile Network. 2
- 5G** Fifth Generation Mobile Network. 1
- 5GC** 5G Core. 32
- AAC** Advanced Audio Codec. 73
- AF** Application Function. 34
- AMF** Access and Mobility Management Function. 8
- API** Application Programming Interface. 4
- AUSF** Authentication Server Function. 34
- BSS** Business Support System. 14
- CAPEX** Capital Expenditure. 3
- CSMF** Communication Service Management Function. 8
- CSV** Comma-Separated Values. 76
- CU** Central Unit. 39
- CUPS** Control and User Plane Separation. 33
- DU** Distributed Unit. 39
- DW** Deutsche Welle. 72
- DWDM** Dense Wavelength Division Multiplexing. 26
- E2E** End-to-End. 23
- eMBB** enhanced Mobile Broadband. 1
- ENBPI** Ensemble Batch Prediction Intervals. 88

Glosario

ETSI European Telecommunications Standards Institute. 20

FHD Full High Definition. 57

gNB gNodeB. 5

GRE Generic Routing Encapsulation. 47

GSMA Global System for Mobile Communications Association. 3

GTP GPRS Tunnelling Protocol. 71

HLS HTTP Live Streaming. 72

HTTP Hypertext Transfer Protocol. 32

IETF Internet Engineering Task Force. 17

IP Internet Protocol. 16

JSON JavaScript Object Notation. 14

KPI Key Performance Indicator. 17

KVM Kernel-based Virtual Machine. 45

LSTM Long Short-Term Memory. 57

LTE Long Term Evolution. 2

MAC Medium Access Control. 40

MANO Management and Orchestration. 9

mMTC massive Machine-Type Communications. 1

MPLS Multiprotocol Label Switching. 26

MSE Mean Squared Error. 79

MVNO Mobile Virtual Network Operator. 4

NAS Non-Access Stratum. 40

NEF Network Exposure Function. 34

NF Network Function. 8

NFV Network Functions Virtualization. 20

NFVI Network Functions Virtualization Infrastructure. 44

NRF Network Repository Function. 33

NRM Network Resource Models. 13

NSA Non-Standalone. 2

NSI Network Slice Instance. 8

NSID Network Slice Instance Descriptor. 24

NSMF Network Slice Management Function. 9

NSSF Network Slice Selection Function. 8

NSSI Network Slice Subnet Instance. 8

NSSMF Network Slice Subnet Management Function. 9

NWDAF Network Data Analytics Function. 28

OAI OpenAirInterface. 22

ODL OpenDaylight. 5

ONAP Open Network Automation Platform. 21

ONOS Open Network Operating System. 22

OPEX Operational Expenditure. 3

OSM Open Source MANO. 21

OSS Operations Support System. 12

OVS Open vSwitch. 5

PCF Policy Control Function. 11

PDCP Packet Data Convergence Protocol. 40

PDU Protocol Data Unit. 11

PHY Physical Layer. 40

PLMN ID Public Land Mobile Network Identifier. 42

QoS Quality of Service. 11

RAN Radio Access Network. 6

REST Representational State Transfer. 14

RLC Radio Link Control. 40

Glosario

- RMSE** Root Mean Squared Error. 59
- ROADM** Reconfigurable Optical Add-Drop Multiplexer. 26
- RRC** Radio Resource Control. 27
- RTT** Round-Trip Time. 23
- RU** Radio Unit. 39
- S-NSSAI** Single Network Slice Selection Assistance Information. 9
- SA** Standalone. 2
- SBA** Service-Based Architecture. 8
- SDAP** Service Data Adaptation Protocol. 40
- SDN** Software-Defined Networking. 5
- SDR** Software-Defined Radio. 22
- SLA** Service-Level Agreement. 3
- SlaaS** Slice as a Service. 4
- SMF** Session Management Function. 8
- TR** 3GPP Technical Report. 11
- TS** 3GPP Technical Specification. 9
- UDM** Unified Data Management. 34
- UE** User Equipment. 5
- UPF** User Plane Function. 8
- URLLC** Ultra-Reliable and Low-Latency Communications. 1
- USRP** Universal Software Radio Peripheral. 43
- VIM** Virtual Infrastructure Manager. 23
- VM** Virtual Machine. 23
- VNF** Virtual Network Function. 16
- VNFD** Virtual Network Function Descriptor. 23

Capítulo 1

Introducción

El desarrollo de la quinta generación de comunicaciones móviles surge como respuesta a un conjunto de requisitos de servicio más diversos que los contemplados por las generaciones anteriores. 5G se diseña como un sistema capaz de soportar categorías de servicio con perfiles claramente diferenciados en términos de tasa de datos, latencia, densidad de dispositivos y confiabilidad. 3GPP identifica tres familias de servicios fundamentales: *enhanced Mobile Broadband* (eMBB), *Ultra-Reliable and Low-Latency Communications* (URLLC) y *massive Machine-Type Communications* (mMTC) [1]. Estas categorías sirven para el diseño de las funciones de red y para la planificación de los mecanismos de gestión de recursos en 5G.

La arquitectura de una red móvil se organiza tradicionalmente en tres dominios, como muestra la Figura 1.1: acceso, núcleo y transporte. El dominio del acceso comprende el conjunto de radio bases, que pueden tener distintos niveles de desagregación y que se encargan de proveer conectividad inalámbrica. El núcleo se define por funciones modulares, orientadas a servicios como la autorización y autenticación de usuarios, tarificación y señalización, entre otros. El dominio de transporte interconecta estos elementos y permite que los flujos de usuario y de señalización circulen entre los componentes del acceso, el núcleo y las redes externas.

En contraposición de esta visión por dominios, también puede concebirse el

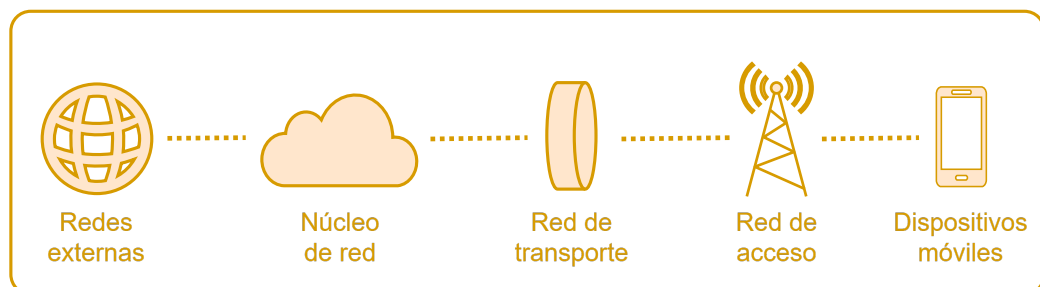


Figura 1.1: Arquitectura general de la red móvil: el dispositivo móvil se conecta por el medio inalámbrico a la red de acceso, que llega al núcleo de la red mediante la red de transporte.

Capítulo 1. Introducción

sistema como una única red de extremo a extremo (o *end-to-end*) que incluye funciones distribuidas en los tres dominios. Esto implica que mecanismos como la asignación de recursos, el establecimiento de sesiones o la aplicación de políticas pueden involucrar información combinada del acceso, el núcleo y el transporte, lo que abre la puerta a nuevos niveles de orquestación que puedan ver la red como un todo y tengan más información para la gestión de recursos.

El pasaje de 4G a 5G supone una evolución en varios sentidos. LTE fue diseñado para un escenario dominado por servicios de banda ancha móvil, con una arquitectura de núcleo más monolítica y un plano de control centralizado, con mecanismos de gestión de recursos optimizados para evitar congestión en escenarios de datos móviles, pero no necesariamente para adaptarse a otros tipos de servicios. En cambio, 5G adopta una arquitectura más modular, con funciones separadas y en interfaces estandarizadas entre componentes. Esto permite separar las responsabilidades del plano de control y del plano de usuario para ser más flexible en la gestión de las funciones de red.

La transición entre generaciones de tecnologías móviles acarrea problemas asociados a factores como costos y compatibilidades, por lo que la estandarización de 5G admite un despliegue intermedio llamado *Non-Standalone* o NSA [2], donde el núcleo LTE continúa realizando parte de las funciones de control mientras que el acceso radio pasa a 5G. Esta modalidad permite acelerar los primeros despliegues y reducir la inversión inicial, ya que reutiliza componentes ya operativos del sistema LTE. Sin embargo, también impone limitaciones: al mantener el núcleo LTE, ciertos mecanismos de gestión, coordinación y control siguen sujetos a las capacidades de LTE. Por este motivo, las funciones que requieren una autonomía plena del núcleo 5G (incluyendo varios aspectos de la segmentación dinámica) solo se habilitan en despliegues *Standalone* o SA, donde el sistema opera sobre el núcleo 5G.

Estas diferencias en la arquitectura y en los modos de despliegue condicionan el grado de control que puede ejercerse sobre los recursos de la red y, en particular, sobre su capacidad de operar mecanismos como la segmentación dinámica. La adopción inicial de NSA retrasó en parte la posibilidad de experimentar con modelos dinámicos, ya que la red sigue dependiendo de funciones y limitaciones de LTE. Con la progresiva transición hacia despliegues SA, la arquitectura de 5G habilita un espacio más amplio para explorar esquemas de asignación y gestión que integren información del acceso, del núcleo y del transporte. Este trabajo se posiciona en ese contexto: desarrollar un entorno que permite evaluar, de manera experimental, enfoques de gestión más flexibles que aprovechen las propiedades de la arquitectura 5G, durante esta etapa en la que la transición aún no es una realidad en la mayor parte de las operadoras.

1.1. Motivación

¿Qué es la red móvil comercial para sus usuarios? La conectividad es algo que damos por sentado, una *utility* más: como con la electricidad o el agua, no festejamos que funcione pero consideramos muy grave que no lo haga. Esto, en

los contextos de competencia de mercado, también la convierte en un *commodity*, en el sentido de comportarse como un servicio básico y de bajo costo en el que podríamos cambiar de proveedor indistintamente.

Esto ha hecho que, en los últimos años, la industria Telco se vea en un escenario de bajos márgenes donde, a pesar de necesitar grandes inversiones para mantenerse al ritmo de la evolución tecnológica, opera solamente como una infraestructura de datos que otros servicios usan y monetizan [3].

En respuesta, las prácticas de la industria llevan años promoviendo la transformación del proveedor de datos a la plataforma de tecnología, que pueda dar servicios digitales y sea capaz de aprovechar su propia infraestructura. Para llevar adelante esta evolución, llamada muchas veces como el pasaje “de Telco a *Tech-co*”, existen varias estrategias. Algunas de ellas surgen de alianzas como el TM Forum [4] para estandarizar interfaces de los servicios de operación y de negocios que corren sobre la red y facilitar la interoperabilidad e integración de nuevos sistemas que ayuden a aprovechar la infraestructura. Otros surgen de los propios estándares, como el *Network Slicing*.

1.1.1. ¿Por qué Network Slicing?

Network Slicing se refiere a la segmentación virtual de los recursos de una red. La técnica se posiciona en el mercado de telecomunicaciones gracias a la necesidad de los operadores de soportar una gama amplia de servicios y requerimientos comerciales. Análisis del sector, como los publicados por GSMA [5], destacan la eficiencia de costos como uno de los principales impulsores de esta evolución. Al permitir que múltiples redes virtuales compartan una misma infraestructura física, la segmentación permite a los operadores reducir el gasto de capital (CAPEX) y de operaciones (OPEX) mediante la reducción de los recursos que eran necesarios para atender servicios o verticales de manera separada. Esta eficiencia se ve además potenciada por la asignación dinámica de recursos habilitada mediante automatización [6], donde se presenta la posibilidad de adaptar en tiempo real sus configuraciones de segmentación ante variaciones en la demanda.

Más allá de las consideraciones económicas, la capacidad de garantizar Acuerdos de Nivel de Servicio (SLA) adaptados a diferentes requerimientos es la clave para dar lugar al *slicing*. El trabajo de GSMA sobre plantillas de SLA [7] muestra cómo la segmentación permite aislación y reserva de recursos, ajustadas a métricas de desempeño como el ancho de banda, latencia o la densidad de dispositivos. Esta diferencia resalta para las industrias emergentes, como se explica en especificaciones de 3GPP tratadas en la sección siguiente. Por ejemplo, los segmentos de eMBB están diseñados para maximizar la capacidad, soportando aplicaciones como transmisión de contenidos multimedia y redes densas en zonas urbanas. Los segmentos de URLLC, definidos por 3GPP para la automatización industrial de próxima generación y el control de vehículos autónomos, necesitan garantizar una latencia inferior a un milisegundo. Por su parte, los escenarios de mMTC enfrentan el desafío de conectar miles de millones de dispositivos de bajo consumo en áreas extensas, priorizando la duración de batería y la cobertura. Además, las comuni-

Capítulo 1. Introducción

caciones *Vehicle-to-Everything* requieren segmentos diseñados para la movilidad, con el fin de mantener enlaces continuos y seguros entre vehículos e infraestructuras [8]. Finalmente, las verticales de la Industria 4.0 demandan *slices* flexibles para satisfacer las necesidades de manufactura y automatización, un caso de uso ampliamente documentado en investigaciones patrocinadas por fabricantes [9].

Junto a estos impulsores técnicos y específicos por vertical, el *slicing* también da lugar a nuevos modelos comerciales y de provisión de servicios, destacando el denominado *Slice-as-a-Service* (SlaaS), en los que el aprovisionamiento de *slices* abstrae los segmentos de red como servicios programables bajo demanda para clientes empresariales, permitiendo a las organizaciones adquirir conectividad segmentada con SLAs acordados [10]. Estos modelos extienden el rol del operador más allá del proveedor de red, transformándolo en una plataforma de servicios que da lugar a ecosistemas donde la conectividad puede venderse directamente a verticales industriales, integrarse mediante APIs, o bien ofrecerse como segmentos para operadores móviles virtuales (MVNOs) o redes privadas. Proveedores de la industria ya implementan plantillas de segmentos y flujos de trabajo automatizados, lo que simplifica los procesos de aprovisionamiento y gestión del ciclo de vida, reduciendo las barreras de entrada para empresas y socios tecnológicos [11].

Estos impulsores y modelos de negocio configuran un escenario en el que el *network slicing* se convierte en habilitador de oportunidades de mercado y diversidad de servicios [12].

1.1.2. Necesidad de experimentación

El diseño de redes móviles ha estado tradicionalmente guiado por el principio de prepararse para el peor caso. Esta práctica, aunque en general es efectiva para mitigar la congestión en horas pico, genera períodos largos en los que los recursos de la red quedan subutilizados [13]. En este sentido, las configuraciones estáticas de segmentos de red no son suficientes: también derivan en un uso ineficiente de los recursos asignados inicialmente [14]. Asignar una porción fija de capacidad a un servicio implica asumir que ese servicio utilizará esa capacidad de forma sostenida, algo que rara vez pasa en escenarios reales.

A su vez, hay una brecha importante entre las capacidades teóricas de la segmentación de red dinámica y la práctica actual en despliegues comerciales. La adopción del *slicing* dinámico sigue siendo limitada: varios operadores han comenzado con configuraciones estáticas sobre núcleos 4G, mientras que los mecanismos dinámicos siguen en etapas de demostración o pruebas de concepto [15]. Lograr una asignación flexible y apoyada en técnicas de orquestación más complejas es uno de los caminos más prometedores para mejorar la eficiencia de las redes 5G [16].

Este contexto da lugar a explorar técnicas de automatización en la asignación de recursos con soluciones experimentales. Contar con un *testbed* que permita evaluar mecanismos de reasignación dinámica de recursos da la posibilidad de estudiar estrategias para afrontar esta problemática de recursos desaprovechados.

En ese marco, este trabajo se motiva por dos necesidades complementarias: por un lado, construir una plataforma 5G de extremo a extremo que permita experi-

mentar con *slicing* y, por otro, estudiar mecanismos de asignación que respondan mejor a la variación del tráfico que los esquemas estáticos.

1.2. Contribuciones

La contribución principal de este trabajo es la creación de un *testbed* 5G de extremo a extremo que combina un núcleo 5G, múltiples nodos de radio gNB, una población configurable de dispositivos de usuario (UE) y una red de transporte controlada mediante redes definidas por software o SDN. Este entorno no sólo reproduce la arquitectura de una red 5G real, sino que permite observar, medir y modificar el comportamiento del plano de usuario bajo distintas condiciones de carga.

El *testbed* implementado incluye mecanismos de clasificación de tráfico a nivel de conmutación con Open vSwitch (OVS), reglas explícitas para aislar flujos entre *slices* y una capa de control sobre el controlador de SDN OpenDayLight (ODL) que permite ajustar dinámicamente los parámetros de las colas (o *queues*) utilizadas por el plano de usuario. Esto permite validar el aislamiento entre *slices*, el impacto de las decisiones de reasignación de recursos y la consistencia del comportamiento de cada gNB cuando opera de forma independiente. La capacidad de escalar la arquitectura a múltiples gNBs y múltiples *slices* usando herramientas abiertas es uno de los resultados prácticos más relevantes del trabajo.

A partir de este entorno, el trabajo propone un mecanismo de adaptación dinámica para *slices* de transporte basado en la predicción del *throughput* futuro. Para eso se generan datasets a partir de tráfico real descargado por los dispositivos móviles a través de la red 5G y se entrenan modelos de aprendizaje automático, con el foco de evaluar si esta capacidad de predicción resulta efectiva para la asignación de recursos en tiempo real bajo diferentes perfiles de tráfico multimedia, en particular en *slices* prioritarios.

Como ejemplo de aplicación que saca provecho del *testbed*, se desarrolla una metodología para ajustar las políticas de asignación a partir de criterios que incluyen la definición de ventanas de observación, la construcción de datasets independientes para cada tipo de tráfico, el diseño de funciones de pérdida asimétricas para priorizar la sobrepredicción en *slices* prioritarios, y la integración de los modelos en un sistema completo que combina medición, predicción y reasignación de recursos cada segundo. Esta metodología permite estudiar el efecto de distintos esquemas de penalización y cuantificar sus consecuencias sobre métricas como el RMSE y el porcentaje de subpredicción. Finalmente, se calcula el margen a agregar para asegurar con cierta probabilidad que la subpredicción no va a pasar del límite establecido.

Esto se hace disponible para su reproducción en un repositorio público, de donde pueden seguirse las instrucciones para implementar total o parcialmente el *testbed* (en la medida en que los recursos lo permitan). También se da acceso a los datasets utilizados y los scripts para entrenar y utilizar los modelos aquí explicados.

1.3. Sobre el documento

El resto de este documento se organiza en base a los capítulos aquí explicados, desde conceptos fundamentales hasta la implementación del sistema completo.

El Capítulo 2 presenta los elementos centrales del *network slicing*: sus motivaciones, las principales definiciones establecidas por los estándares y los procedimientos y limitaciones identificadas en la literatura para enmarcar el trabajo. También se introduce cómo se plasma la técnica en cada dominio de la red y cómo se puede evaluar.

El Capítulo 3 revisa distintos antecedentes de implementaciones prácticas relacionadas con el *slicing* en la red de acceso radio (RAN), en el núcleo y en la red de transporte, poniendo especial atención en los *testbeds* desarrollados en años recientes. Esta revisión permite comparar enfoques, relevar las oportunidades y justificar la necesidad de una implementación que integre acceso, núcleo y transporte en una única plataforma con control de tráfico.

El Capítulo 4 describe las tecnologías utilizadas para construir el sistema: el núcleo 5G empleado, el entorno de radio, los componentes de virtualización y los mecanismos SDN para el plano de transporte. El objetivo es definir la función que cumple cada herramienta y cómo se articula dentro del diseño general, justificando las elecciones y decisiones de diseño.

El Capítulo 5 detalla la solución base desarrollada sobre un único gNB, incluyendo la arquitectura, la generación de tráfico y los primeros mecanismos de adaptación de recursos. A partir de eso, el Capítulo 6 presenta el proceso de escalar la solución a múltiples gNBs y múltiples *slices*, generando datasets con tráfico real, entrenando modelos predictivos y evaluando su desempeño dentro del sistema operativo completo.

Finalmente, el Capítulo 7 resume las conclusiones del trabajo y plantea posibles líneas de trabajo futuro, tanto en la ampliación del *testbed* como en el perfeccionamiento de las estrategias de adaptación basadas en predicción.

Capítulo 2

Network slicing

El concepto de *network slicing*, por novedoso que suene el nombre o reciente que sea la asociación a 5G, puede entenderse como una extensión del paradigma de virtualización de redes, ampliamente explorado en la literatura desde hace décadas [17]. La idea consiste en dividir una infraestructura física común en múltiples redes lógicas que comparten recursos de manera controlada, con la opción de dar a cada una de ellas niveles diferenciados de calidad de servicio. Lo diferente del enfoque en el marco de 5G es la estandarización formal por parte de 3GPP, que define procedimientos, interfaces y modelos de gestión para la creación y operación de *slices*.

En términos de virtualización, es posible distinguir entre los tipos de virtualización *hard* y *soft*. El primero se refiere a particiones de hardware, donde los recursos quedan asignados de manera exclusiva a cada entidad. Un ejemplo de esto es el uso de bocas diferentes de un equipo como conexiones rígidas de redes diferentes. El segundo tipo consiste en técnicas de virtualización basadas en software que permiten compartir los mismos recursos físicos entre múltiples entidades de manera dinámica y programable. El *network slicing* definido en 5G corresponde a esta última categoría, ya que se apoya en mecanismos software para la asignación flexible de recursos en los diferentes dominios de la red.

La separación de recursos, incluso de manera estática, ya resulta suficiente en muchos escenarios para diferenciar servicios y operar funciones en paralelo. Por ejemplo, la asignación fija de un ancho de banda mínimo garantiza la coexistencia de flujos de datos sin interferencias. No obstante, la posibilidad de implementar *slicing* dinámico aumenta la eficiencia del sistema al permitir que los recursos se ajusten en tiempo real a la demanda, pudiendo dar lugar a nuevos flujos con los recursos no utilizados.

Con el objetivo de organizar el análisis del *Network Slicing*, este capítulo se estructura en seis apartados. La Sección 2.1 presenta la arquitectura de alto nivel de los componentes de *slicing* en la red, con apoyo en diagramas basados en estándares de 3GPP. En la Sección 2.2 se detallan las definiciones y procesos de estandarización vinculados a la tecnología. Posteriormente, la Sección 2.3 aborda la orquestación y el ciclo de vida de los *slices*, desde su preparación hasta su terminación. La Sección 2.4 se centra en el *slicing* por dominio, diferenciando los

Capítulo 2. Network slicing

mecanismos específicos de RAN, núcleo y transporte. La Sección 2.5 discute la distinción entre *slices* estáticos y dinámicos, en función de las posibilidades de asignación de recursos. Finalmente, la Sección 2.6 establece el alcance del *Network Slicing* en el presente trabajo y los criterios de evaluación que se aplicarán a las *slices* de transporte en el marco de la experimentación.

2.1. Arquitectura de alto nivel

Como se introdujo, el *network slicing* en redes 5G se define como la capacidad de instanciar múltiples redes lógicas independientes sobre una infraestructura física compartida. Según la especificación técnica 23.501 de 3GPP [18], cada red lógica, denominada *Network Slice Instance* (NSI), ofrece un conjunto de funcionalidades que puede incluir acceso, transporte y núcleo. Una NSI se compone a su vez de *Network Slice Subnet Instances* (NSSI), correspondientes a dominios particulares.

Esta descomposición modular permite que un NSSI de un dominio pueda ser reutilizado en varias NSI. Por ejemplo, un NSSI de núcleo que incluya funciones de AMF, SMF y UPF (funciones de autenticación, manejo de sesiones y de plano de usuario, respectivamente, descritas en detalle en el Capítulo 4) puede servir simultáneamente a diferentes *slices*, siempre que compartan los mismos requerimientos funcionales. En cambio, en el dominio de RAN, un NSSI puede estar configurado con políticas específicas de asignación de espectro o de programación de recursos de radio, mientras que en transporte los NSSIs permiten definir caminos con restricciones particulares de latencia o capacidad [19].

La arquitectura de 5G introduce la *Service-Based Architecture* (SBA) en el núcleo de red. Bajo este enfoque, las funciones de red (*Network Functions* o NFs) se comunican mediante interfaces de servicios, lo que permite la instanciación y el descubrimiento de funciones. Este principio es importante para el *slicing*, ya que permite que las NFs se asocien a diferentes identificadores y sean compartidas o dedicadas según la necesidad del *slice*. Así, funciones de control pueden operar de forma compartida entre *slices*, mientras que el UPF se instancia típicamente de manera dedicada para aislar tráfico del plano de usuario.

Un elemento fundamental en este modelo es el *Network Slice Selection Function* (NSSF), definido por 3GPP como la función que asiste en la selección de *slice* durante el procedimiento de registro y en la modificación posterior de las suscripciones. El NSSF determina qué AMF puede gestionar los identificadores de *slice* solicitados por el UE, considerando la disponibilidad de *slice* en el área de servicio.

La gestión de *slice* se articula en torno a tres funciones jerárquicas que, por limitaciones de las herramientas a explicar en la Sección 2.6, no se incluyen en este *testbed*, pero que es pertinente analizar:

- La *Communication Service Management Function* (CSMF), que traduce los requisitos de servicios en requerimientos de *slice*. Es decir, actúa como interfaz con los clientes o verticales, recibiendo sus necesidades de nivel de servicio (por ejemplo, latencia o ancho de banda) y traduciendo esos requerimientos

2.2. Definiciones y estandarización

en especificaciones de *slice* que luego entregará al siguiente nivel.

- La *Network Slice Management Function* (NSMF), responsable de coordinar y orquestar instancias de NSI, descomponiéndolas en sub-instancias (NSSI) por dominio. Además, orquesta los flujos entre CSMF (nivel superior) y las NSSMF (nivel inferior).
- Las *Network Slice Subnet Management Functions* (NSSMF), que gestionan los NSSIs dentro de cada dominio específico (RAN, núcleo y transporte). Cumple funciones en cada dominio, como instanciación, escalado, reparación y finalización de los NSSIs. Interactúa con controladores o funciones específicas de cada dominio, alineándose con las órdenes del NSMF.

Esta jerarquía se ilustra en la Figura 2.1, que representa la arquitectura de gestión de *network slicing* conforme a 3GPP. Observando el dominio de gestión y orquestación (*MANagement and Orchestration* o MANO), en la parte superior se encuentra el CSMF, que recibe las especificaciones de nivel de servicio y las traduce en requerimientos técnicos. En el nivel intermedio, el NSMF administra las NSI y actúa como orquestador, descomponiendo las demandas hacia los dominios de red. En el nivel inferior, las NSSMF gestionan el ciclo de vida de los NSSIs, interactuando con funciones y controladores de dominio. Esta estructura refleja el modelo de gestión de *network slice* definido por 3GPP [20].

Por último, la arquitectura contempla la exposición de capacidades mediante APIs estandarizadas. Estas interfaces permiten ofrecer *network slicing* como un servicio (el mencionado SaaS), de modo que los clientes verticales puedan solicitar perfiles de servicio sin interacción directa con los parámetros internos de red. El NSMF es el encargado de traducir estas solicitudes en configuraciones técnicas, mientras que los NSSMF aplican las acciones concretas en cada dominio.

2.2. Definiciones y estandarización

La estandarización del *network slicing* es lo que permite la interoperabilidad de esta tecnología en redes 5G. La propia especificación TS 23.501, en su sección 5.15, establece los fundamentos de la selección, suscripción y operación de *slice*. A partir de ahí, se definen los mecanismos esenciales que permiten al UE, a la red de acceso y al núcleo 5G coordinar el uso de recursos segmentados.

Identificación de slice

El estándar introduce el *Single Network Slice Selection Assistance Information* (S-NSSAI) como identificador fundamental de un *slice*. El S-NSSAI está compuesto por el *Slice/Service Type* (SST), que determina la categoría de servicio soportada (eMBB, URLLC, MIoT), y un *Slice Differentiator* (SD) opcional, que diferencia múltiples instancias de *slice* con el mismo SST. Estos valores conforman el lenguaje común mediante el cual se identifican y seleccionan *slices* en cualquier despliegue 5G.

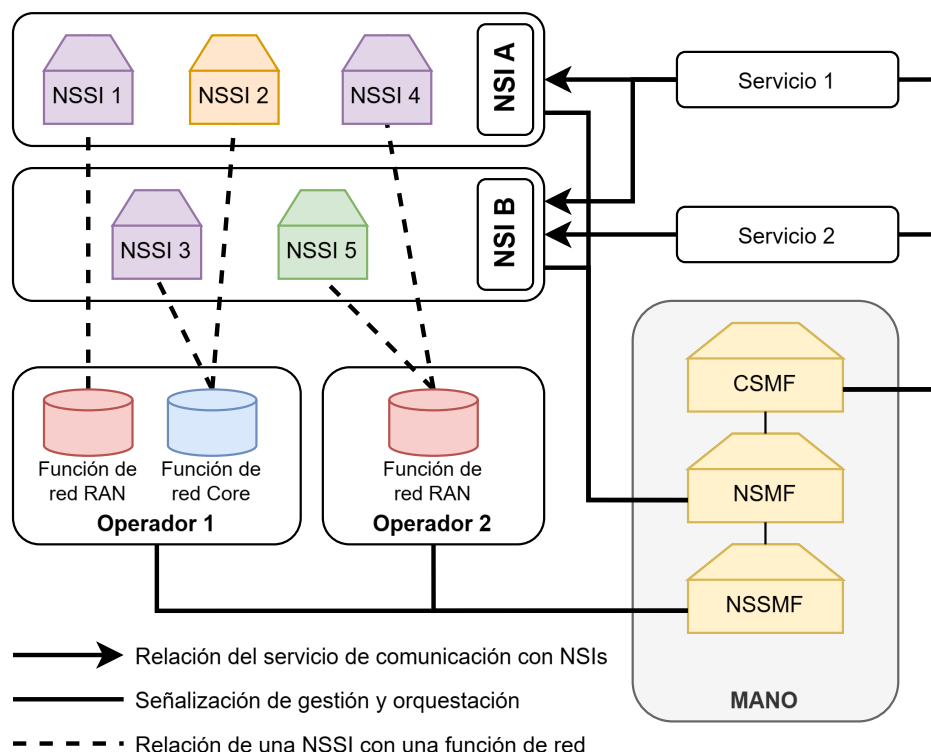


Figura 2.1: Jerarquía de gestión de *network slicing* en 5G, en la que se establecen las relaciones entre NSI, NSSI, orquestación, funciones de red y servicios.

La información contenida en el S-NSSAI permite además la interoperabilidad en escenarios en los que pueda no existir correspondencia directa entre *slices* ofrecidos en la red y el requerido por el dispositivo del cliente. Para eso, se define el mapeo de S-NSSAIs, de manera de que el abonado reciba un servicio aunque el *slice* original no esté disponible.

Suscripción del UE

El estándar diferencia tres tipos de NSSAI: el *Subscribed* NSSAI, definido en la red de origen y almacenado en el perfil del abonado, el *Configured* NSSAI, almacenado en el UE y recibido durante el registro, y el *Allowed* NSSAI, que representa el subconjunto efectivamente habilitado en la red. Estos tipos pueden actualizarse según la disponibilidad de *slice* en la red. En particular, TS 23.501 especifica los procedimientos mediante los cuales el AMF envía actualizaciones de *Configured* NSSAI al UE cuando cambian las condiciones de la red.

La inclusión de estas distinciones busca que la gestión de *slices* no se limite a la suscripción estática, sino que sea flexible frente a variaciones de topología, de roaming o de planificación de recursos.

Selección de funciones de red

TS 23.501 también establece la selección de funciones de red asociadas a los *slices*. El componente central es la NSSF, que guía la selección de *slice* durante el registro inicial. El NSSF determina qué función de gestión de sesiones debe encargarse de un conjunto solicitado de S-NSSAIs y coordina re-ubicaciones si una función existente no soporta todos los *slices* requeridos.

En el plano de usuario, cada sesión PDU (o conexión de datos que permite al UE intercambiar información con la red 5G) debe estar asociada a un S-NSSAI. Esto asegura que el tráfico de datos se encamine por las funciones de red correspondientes al *slice* solicitado. La especificación contempla procedimientos tanto para el establecimiento inicial como para la modificación de sesiones PDU, de modo que los recursos de red se mantengan alineados con los requisitos de servicio.

Procedimientos detallados

Mientras que TS 23.501 establece la arquitectura y los principios funcionales, el documento complementario TS 23.502 [21] especifica los procedimientos de señalización. Allí se describen en detalle los mensajes intercambiados entre UE, AMF, NSSF y otras funciones para implementar las operaciones de registro, selección de AMF, modificación de NSSAI y establecimiento de sesiones PDU.

Por ejemplo, el procedimiento de registro inicial incluye pasos en los que el dispositivo transmite su *Requested* NSSAI, el AMF consulta al NSSF, y este devuelve una lista de *slices* soportados junto con la recomendación de AMF adecuado. Esta descripción exhaustiva asegura que las implementaciones de diferentes fabricantes sigan la misma lógica y sean interoperables en la práctica.

Control de políticas

El marco de estandarización también vincula el *network slicing* con el control de políticas y facturación a través de 3GPP TS 23.503 [22]. Este documento establece cómo las reglas de QoS y facturación se aplican de manera diferenciada por *slice*.

En este sentido, el *Policy Control Function* (PCF) y las funciones de facturación se integran con el identificador S-NSSAI para aplicar políticas a nivel de sesión PDU. TS 23.503 busca que las decisiones de políticas se mantengan consistentes con la selección y operación de *slices* definida en TS 23.501.

Evolución del estándar

Antes de la formalización en TS 23.501, 3GPP desarrolló el documento TR 28.801 [23], un estudio preliminar que introdujo conceptos de gestión de *slice*, como las instancias NSI y NSSI, así como las funciones NSMF y NSSMF. Este reporte técnico sirvió como base para los trabajos posteriores por el grupo de trabajo SA5 [24] y preparó el terreno para la serie TS 28.53x. Esta serie de estándares constituye el complemento de gestión a la definición funcional de TS 23.501:

Capítulo 2. Network slicing

- TS 28.530 establece los principios de gestión del ciclo de vida de *slice*, con las fases de preparación, instanciación, operación y terminación.
- TS 28.531 define los modelos de recursos de red (*Network Resource Models*) aplicables a *slicing*.
- TS 28.532 describe los procedimientos de gestión (creación, modificación y eliminación) de NSI y NSSI.
- TS 28.533 detalla los requisitos de interfaz entre funciones de gestión de *slice* y sistemas OSS.

Si bien el detalle del ciclo de vida de *slices* se aborda a continuación, se mencionan aquí las referencias de 3GPP para mostrar que la estandarización del *network slicing* incluye un marco completo de gestión, que permite presentarlo como una tecnología interoperable y gestionable en el ecosistema 5G.

2.3. Orquestación y ciclo de vida

Los principios del *network slicing* descritos en TS 23.501 encuentran su complemento en la serie normativa TS 28.53x, que define cómo dichos *slices* pueden ser gestionados y orquestados a lo largo de su vida. Es decir que estas especificaciones permiten trasladar a la práctica la arquitectura: mientras que TS 23.501 establece la lógica de identificación, suscripción y selección de *slices*, la serie TS 28.53x, normaliza el ciclo de vida de los *slices*.

2.3.1. Ciclo de vida del slice

El ciclo de vida de un *slice* puede representarse como en la secuencia que muestra la Figura 2.2. El diagrama expone un flujo lineal desde la preparación hasta la terminación, colocando cada fase en un bloque conectado que enfatiza su dependencia del anterior. Esta representación muestra que los *slices* no son entidades estáticas, sino instancias que cambian en el tiempo bajo control de funciones de gestión.

La fase de preparación aparece en la figura como el primer bloque. En ella se asocian perfiles de servicio a perfiles de *slice*, definiendo los parámetros técnicos que luego guiarán la creación. De esta fase emergen insumos formales, como plantillas de *slice* y asignaciones iniciales de SLA, que constituyen la base de la siguiente etapa.

En el segundo bloque, el diagrama representa la fase de comisionamiento o instanciación, donde se crean las instancias de *slice* en los diferentes dominios de red. Este proceso implica la provisión coordinada de funciones de red virtualizadas y rutas controladas por SDN, configurando y asignando recursos según las necesidades del *slice*.

El tercer bloque corresponde a la fase de operación, que en un escenario normal se extiende como la etapa más prolongada del ciclo de vida. Esto refleja que la

2.3. Orquestación y ciclo de vida



Figura 2.2: Fases del ciclo de vida de un *network slice*, desde su creación hasta su eliminación.

operación continua requiere monitorización, gestión de rendimiento y detección de fallos. Aquí aplica la interacción con sistemas de analítica, como la *Management Data Analytics Function*, que alimentan la automatización y otros procesos de control.

Finalmente, el diagrama muestra la fase de terminación como el bloque de cierre del ciclo. En él, los recursos son liberados y puestos a disposición, y la instancia de *slice* se elimina del gestor.

2.3.2. Coordinación

Como se presentó en la Sección 2.1, la arquitectura de gestión se organiza en tres niveles: CSMF, NSMF y NSSMF. Lo restante con respecto a ellos es precisar cómo estas funciones intervienen en el ciclo de vida.

Durante la fase de preparación, el CSMF actúa como traductor de los requisitos del cliente en parámetros técnicos concretos. En la etapa de comisionamiento, el NSMF orquesta la creación de la instancia de *slice*, coordinando a los NSSMF que instancian cada uno en su respectivo dominio. En la operación, el NSMF supervisa el comportamiento del *slice*, mientras que los NSSMF gestionan en cada dominio los ajustes derivados de la monitorización. Finalmente, en la terminación, el NSMF organiza la revocación de configuraciones y los NSSMF liberan los recursos de manera controlada.

Este modelo busca que cada fase del ciclo de vida se ejecute en coherencia con la arquitectura introducida anteriormente.

2.3.3. Procedimientos de gestión

La gestión del ciclo de vida del *network slice* se define en un conjunto de normas ya analizadas para establecer modelos de información e interfaces entre sistemas. Como se vio, el TS 28.531 establece los modelos de recurso de red (NRM)

Capítulo 2. Network slicing

que describen los objetos necesarios para representar *slice* en la capa de gestión. Estos modelos, como se explicó, permiten que las funciones de gestión manipulen instancias de *network slice* como entidades formalmente definidas.

Entre los objetos más relevantes se encuentran los introducidos NSI, NSSI y perfiles de *slice* (para cubrir parámetros derivados de los servicios, como latencia máxima tolerada o *throughput*).

Los NRMs, implementables con protocolos como REST o formatos como JSON, proporcionan el vocabulario común para que diferentes funciones de gestión y sistemas OSS/BSS puedan interoperar.

El estándar TS 28.532 complementa a TS 28.531 definiendo los procedimientos que deben implementarse sobre los objetos de información. Estos procedimientos formalizan las operaciones que permiten gestionar el ciclo de vida de *slice*.

Entre los procedimientos más relevantes se tienen:

- Creación de NSI y NSSI: especifica cómo el NSMF solicita a un NSSMF la creación de una sub-instancia de *slice* en un dominio particular, utilizando plantillas predefinidas y recursos disponibles.
- Modificación de NSI y NSSI: permite ajustar parámetros en la fase de operación, como ampliar capacidad, modificar configuraciones de QoS o reasignar recursos en caso de fallos.
- Escalado: define operaciones de *scale-in* y *scale-out* sobre funciones virtualizadas, incrementando o reduciendo la capacidad de un *slice* en función de la demanda.
- Supervisión de fallas y alarmas: especifica procedimientos de notificación hacia el NSMF en caso de fallos detectados en un NSSI, de manera de que se pueda activar un proceso automatizado para reparar o subsanar.
- Terminación de NSI y NSSI: establece cómo hacer la terminación de manera ordenada de las instancias, liberando recursos y actualizando los estados en los NRMs para reflejar la disponibilidad de la infraestructura.

El TS 28.533 completa el marco de gestión describiendo las interfaces de integración entre funciones de gestión de *slice* y los sistemas de OSS y BSS de los operadores. Su objetivo es que la gestión de *slices* no quede confinada a plataformas aisladas, sino que se integre en los procesos administrativos y de negocio.

Entre los aspectos que regula se encuentran:

- Interfaces de orquestación: define los puntos de referencia mediante los cuales el NSMF y los NSSMF exponen capacidades al OSS. Estas interfaces siguen el sistema de gestión basada en servicios, utilizando APIs RESTful con codificación JSON.
- Gestión de SLA y facturación: especifica cómo los atributos definidos en los NRMs (por ejemplo, *throughput* garantizado o latencia máxima) se exponen a los BSS para soportar, por ejemplo, la diferenciación de facturación por *slice*.

2.4. Slicing por dominio

- Integración con gestión de fallas: establece cómo los datos recolectados por las funciones de gestión se transmiten a sistemas OSS para permitir supervisión unificada de la red.

Así, se busca que los sistemas de operación y negocio de los operadores puedan interactuar con las nuevas funciones de *slicing* sin necesidad de desarrollos ad hoc.

2.4. Slicing por dominio

Hasta la sección anterior, este capítulo abordó el *network slicing* como un todo, pensando en las interacciones que tiene que haber a nivel de red, de funciones de red y de servicios para operar de forma transparente con diferentes tipos de usuarios.

Lo siguiente es ver cómo eso se introduce en la propia red: al dividir la red móvil en acceso, transporte y núcleo, cada dominio implementa el *slicing* de maneras muy diferentes y, en principio, independientes.

2.4.1. Slicing en la RAN

El *slicing* en el dominio radio asigna identificadores de *slice* a los flujos de tráfico, de manera que cada sesión PDU queda asociada a un S-NSSAI y posteriormente a un gestor de QoS en el plano de acceso. Esta asociación permite que el tráfico se gestione con las propiedades que el *slice* requiere.

Para la segmentación de los recursos de radio, el *Radio Resource Management* (a cargo de la gestión de cada nodo) usa algoritmos de *scheduling* y control de admisión para distribuir los recursos espectrales entre *slices* de manera de que se cumplan simultáneamente varios SLA. Estudios experimentales en bancos de pruebas de RAN han mostrado la viabilidad de implementar *schedulers* sensibles al *slice* y controles de admisión diferenciados, capaces de mantener la estabilidad de QoS bajo condiciones de alta carga [25].

La virtualización y desagregación del RAN, definidas en general para 5G pero también impulsadas por el marco OpenRAN de la O-RAN Alliance [26], han ampliado las posibilidades del *slicing* al permitir que las funciones de control y usuario se desplieguen como componentes de software. Esto habilita escenarios donde las funciones centralizadas y distribuidas de la red de acceso (definidas en el Capítulo 4) pueden asignarse dinámicamente a *slices* específicos, facilitando el aislamiento entre servicios [27]. Sobre esta base, los operadores pueden definir políticas de *slice* en la RAN que incluyan asignaciones mínimas de ancho de banda, umbrales de latencia y parámetros de cobertura, además de capacidades como la reconfiguración del espectro o la priorización temporal de *slices* según la demanda.

Un beneficio de esta práctica es que permite que servicios tan distintos como comunicaciones masivas de banda ancha móvil y aplicaciones de misión crítica co-existan en una misma infraestructura sin comprometer la experiencia de usuario. Asimismo, la flexibilidad derivada de la virtualización permite reducir los tiempos de provisión de servicios y de adaptación de los recursos disponibles a las necesidades del mercado [28]. No obstante, la implementación tiene retos para nada

Capítulo 2. Network slicing

menores, comenzando por la variabilidad propia del canal de radio y la movilidad de los usuarios, que agregan complejidad al cumplimiento de SLAs. Además, la virtualización de la RAN introduce nuevas fuentes de latencia en enlaces de *fronthaul* y requiere interoperabilidad entre equipos de múltiples proveedores (dando lugar, una vez más, a OpenRAN para prosperar).

2.4.2. Slicing en el núcleo

En el núcleo 5G, el *slicing* se implementa mediante la selección y asignación de funciones de red en el plano de control y de usuario de acuerdo con los S-NSSAIs solicitados. La función de selección de *slice* determina qué funciones de control puede gestionar los *slice* de un usuario, asociándolo a una función de plano de usuario específica. Este mecanismo permite que un *slice* de baja latencia se asocie, por ejemplo, a funciones de usuario más cercanos al borde de la red, reduciendo la demora en la transmisión, mientras que *slices* de mayor ancho de banda se asocian a funciones con mayor capacidad de cómputo en centros de datos grandes. La diferenciación también se extiende al control de políticas y facturación, mediante la PCF que aplica reglas particulares a cada *slice*.

El valor de esta estrategia está, por un lado, en que los operadores pueden desplegar *slices* con instancias de funciones de red totalmente dedicadas a servicios críticos con latencias mínimas. Por otro lado, está en la flexibilidad de las funciones de red virtualizadas (VNFs) que permite escalar los recursos de un *slice* en función de la demanda, optimizando el uso de infraestructura [29]. También se habilita la creación de modelos de negocio diferenciados, ya que los parámetros de cada *slice* pueden asociarse a políticas de facturación específicas, lo que abre la puerta a servicios premium o industriales especializados [30].

Al mismo tiempo, esta flexibilidad trae el desafío de que múltiples instancias de SMF y UPF agregan complejidad de señalización y aumenta el riesgo de interferencias si el aislamiento no se implementa correctamente. También hay que considerar que la operación multi-*slice* añade sobrecarga en la gestión, al requerir monitoreo y orquestación de decenas o cientos de instancias virtualizadas que deben mantenerse coherentes en tiempo real.

2.4.3. Slicing en la red de transporte

El transporte es el tercer dominio en la materialización del *slice*. El concepto de *transport slicing* consiste en dividir los recursos de los enlaces entre gNB y núcleo (*backhaul*), entre unidades centralizadas y distribuidas (*midhaul*) y entre unidades distribuidas y radios (*fronthaul*) en *slices* que proveen capacidad a los servicios que soportan. Esto se logra mediante el uso de SDN y técnicas de ingeniería de tráfico, que permiten configurar rutas dedicadas y gestionarlas de forma dinámica en respuesta a las condiciones de la red [31]. También hay implementaciones en las que un controlador de transporte de *slice* es quien orquesta estos recursos en redes IP y ópticas [19].

Los beneficios del *slicing* en transporte son necesarios para cumplir con el

2.5. Slices: ¿estáticas o dinámicas?

SLA de un servicio. Aunque la RAN y el núcleo asignen recursos diferenciados, sin un transporte capaz de sostener la latencia y el *throughput* requeridos, el *slice* no puede cumplir con sus compromisos. Al asignar rutas específicas, implementar mecanismos de priorización y monitorear indicadores clave de rendimiento (KPIs) por *slice*, los operadores ganan control sobre el rendimiento. La integración de transporte óptico e IP bajo modelos comunes de *slicing*, además, permite reutilizar infraestructura y usar mecanismos de automatización para reducir costos operativos [32].

Aunque 3GPP considera al transporte como parte de las sub-instancias de *slice*, no define qué atributos deben gestionarse ni cómo mapear los parámetros de QoS del plano de usuario sobre los enlaces físicos. Este vacío obliga a coordinar con organismos como IETF para establecer mecanismos que gestionen el aislamiento y desempeño [33]. La heterogeneidad tecnológica y de proveedores en redes de transporte agrega dificultades, ya que no siempre es posible aplicar mecanismos uniformes de monitoreo de KPIs en entornos multi-dominio [34].

El *slicing* de transporte es el principal interés de este trabajo, apoyado en cómo el SDN puede ayudar en el control para regular los cuellos de botella y asegurar recursos a *slices* prioritarios en una red de múltiples servicios con requerimientos diferentes.

2.5. Slices: ¿estáticas o dinámicas?

La implementación de un *network slice* plantea la dicotomía de poder concebirse como una *slice* cuyos recursos quedan fijos una vez aprovisionados, o como una *slice* cuyos recursos se ajustan para responder a variaciones de demanda. Cabe aclarar que, por la definición de ciclo de vida usada por 3GPP y explicada en este capítulo, toda *slice* es “dinámica” en al menos algunos de sus procesos: aprovisionamiento, modificación y terminación. Por tanto, la idea de “*slice* estático” no significa ausencia de cambios, sino que una vez establecido, sus recursos no se ajustan.

Sin embargo, en la práctica algunos despliegues optan por fijar los recursos de la *slice* tras su creación. Por ejemplo, en el *slicing* estático se suelen asignar recursos dedicados fijos, como ancho de banda o canales ópticos, que no se modifican salvo cambios administrativos mayores [35]. En redes privadas, el operador puede configurar *slices* con recursos fijos para una vertical específica, sin ajustes frecuentes en el transporte o núcleo, para evitar complejidad operacional y sobrecostos de señalización [36].

Por el otro lado, hay beneficios al usar *slices* que permiten adaptación dinámica de recursos. Cuando los recursos asignados a un *slice* no se usan totalmente, pueden ser reutilizados temporalmente en otros servicios de menor prioridad o *best effort*, aprovechando los activos del sistema. Por ejemplo, se puede comparar un enfoque basado en reservas fijas con uno basado en participación (“share-based”), mostrando que los métodos dinámicos pueden mejorar significativamente el uso de recursos sin romper con los SLA, siempre que los mecanismos de control respondan con suficiente rapidez [37]. Asimismo, se han propuesto esquemas que ajustan el

uso de subcanales u otros bloques de recurso en función de la carga, manteniendo los requisitos de latencia y *throughput* de cada *slice* [38].

En el presente trabajo se adopta una posición dinámica: se consideran *slices* que pueden ser configuradas de manera variable, con capacidad de adaptación dinámica en el transporte, respetando los SLA definidos. Con esta decisión se quiere dar lugar a la exploración de métodos para guiar esa adaptación dinámica, basados tanto en respuestas al comportamiento instantáneo de los *slices* como en la mirada en la historia de ese comportamiento para predecir su futuro.

2.6. Alcance y evaluación

A partir de lo analizado en las secciones anteriores, es importante precisar el alcance de este trabajo en relación con la tecnología de *network slicing*. Tal como se discutió, la estandarización 3GPP prevé la creación y gestión de *slices* de extremo a extremo que abarcan los dominios de acceso radio, núcleo y transporte. Sin embargo, en este trabajo se delimita el análisis al dominio de transporte, considerando configuraciones dinámicas, y sin abordar los procesos de aprovisionamiento y terminación. Estas restricciones responden a las limitaciones de las herramientas y plataformas seleccionadas para la experimentación, que se presentan en detalle en el Capítulo 4. En consecuencia, el estudio se centra en la caracterización y evaluación de *slices* en el plano de transporte, sin extenderse a la orquestación de extremo a extremo y, por tanto, sin poder evaluar las demás etapas del ciclo de vida del slice ni la modificación transversal de los recursos en los diferentes dominios.

Sobre esta base, la evaluación de las *slices* de transporte sigue los criterios de evaluar, en primer lugar, el aislamiento entre *slices*, entendido como la capacidad de que el tráfico cursado en un *slice* no ocupe los recursos reservados para otro *slice*. La metodología para medir este aislamiento se basa en pruebas de carga diferenciada: se induce congestión en un *slice* y se verifica, por un lado, que los demás *slices* no sean conscientes de ese tráfico (no se curse en sus interfaces ni UPF) y, por el otro, que el *throughput* en los demás *slices* permanezca en el valor esperado, sin modificaciones en las pérdidas.

En segundo lugar, se verifica que el *throughput* asignado a cada *slice* cumpla con los SLA definidos. En el caso de este trabajo, se pueden adaptar parámetros específicos para tráfico multimedia, en particular para servicios de video o audio, pero se contempla también un criterio general para cualquier tipo de tráfico.

Un tercer criterio de evaluación es la puntualidad en la adaptación del *throughput*. No basta con que eventualmente se asignen recursos adicionales: la adaptación debe hacerse en una ventana de tiempo suficientemente corta como para evitar cortes en la transmisión. En la práctica, esto significa medir la latencia de reacción del *slice* frente a variaciones en la carga, con el objetivo de validar que los mecanismos de control cumplen con la continuidad del servicio.

Esta metodología es la que se contempla a lo largo del trabajo, para evaluar y comparar los escenarios que se desarrollan en los capítulos posteriores.

Capítulo 3

Antecedentes de testbeds y posicionamiento de la propuesta

El desarrollo de *network slicing* en redes 5G ha generado y sigue motivando una amplia producción científica, orientada tanto a la definición de estándares de la industria como a la validación práctica mediante plataformas experimentales. En este capítulo se presentan los antecedentes más relevantes para el presente trabajo, con énfasis en iniciativas que implementan *testbeds* o mecanismos de segmentación basados en SDN. El objetivo es situar la propuesta de este trabajo en el contexto del estado del arte, identificando similitudes, diferencias y huecos que justifican el aporte de la investigación.

La revisión se organiza en cuatro apartados. En primer lugar, se presentan los estudios de carácter de síntesis o *survey*, en los que se identifican las principales líneas de investigación y desafíos pendientes. Después, se describen los *testbeds* centrados en la segmentación de recursos en los dominios de acceso y núcleo de red, incluyendo tanto implementaciones abiertas como propuestas analíticas. Luego se revisan los trabajos que han explorado el *slicing* en el dominio de transporte, ámbito en el que se enmarca esta propuesta. Finalmente, se hace una comparación global que resume las principales características de los enfoques revisados y permite resaltar el valor agregado del *testbed* desarrollado en los capítulos siguientes.

3.1. Revisiones bibliográficas

El uso de artículos de revisión es una práctica establecida para mapear el estado del arte en campos amplios como el del *network slicing* en 5G. En contextos donde coexisten líneas teóricas, prototipos experimentales y proyectos industriales, los *surveys* permiten, por un lado, sintetizar conceptos, arquitecturas y estándares y, por otro, identificar tendencias tecnológicas y herramientas populares. Para esta sección se ha seguido un criterio dirigido a priorizar trabajos que, además de hacer una revisión del estado del arte, se enfoquen en el desarrollo o evaluación de *testbeds* o en estudios empíricos sobre implementaciones de *slicing*, de modo que la síntesis sea pertinente para la discusión posterior sobre entornos de experimentación.

Capítulo 3. Antecedentes de testbeds y posicionamiento de la propuesta

De acuerdo con este criterio, los trabajos analizados se seleccionaron por presentar un resumen sobre *slicing*, describir o comparar plataformas de experimentación (*testbeds* o pruebas de concepto), o aportar evidencia reciente sobre desempeño de aplicaciones desplegadas sobre *slices*.

Esmaily & Kralevska: Small-Scale 5G Testbeds for Network Slicing Deployment: A Systematic Review (2021)

El trabajo de Esmaily y Kralevska [39] se centra en implementaciones pequeñas para desplegar *network slicing*. Los autores justifican su estudio en la utilidad de esos *testbeds* para laboratorios universitarios y grupos de investigación con recursos limitados, en contraste con los grandes proyectos que requieren inversiones mayores. El objetivo es proporcionar un resumen útil para investigadores para reproducir y extender *testbeds* de *slicing* con herramientas *open-source*.

Se describe el procedimiento de búsqueda y selección bibliográfica, detallando bases de datos consultadas, palabras clave usadas y criterios de inclusión y exclusión. Además, proponen un esquema de evaluación por criterios primarios y secundarios para valorar qué tan idóneo es cada *testbed*, como su compatibilidad con el NFV MANO de ETSI, su capacidad de emular RAN y núcleo, costo y documentación. Esta metodología es uno de los aportes más prácticos del artículo, porque convierte la revisión en una herramienta de evaluación para quien quiere seleccionar un *testbed*.

El trabajo organiza su exposición en cuatro bloques:

1. Catálogo de software para los componentes del MANO, para emulación de RAN y para la virtualización del núcleo
2. Definición de criterios de diseño para los *testbeds*
3. Inventario y descripción detallada de *testbeds* existentes (con sus objetivos, tecnologías y experimentos)
4. Comparación, resumen de retos y direcciones futuras

Entre los resultados, los autores muestran una clasificación de paquetes y utilidades que mejor se ajustan a cada bloque MANO, un conjunto de criterios de diseño que funcionan como lista de verificación para la implementación de *testbeds*, y una tabla comparativa de *testbeds* que presenta las soluciones principales y los problemas más comunes. Asimismo, el trabajo enfatiza que, si bien algunos *testbeds* son limitados frente a los proyectos a gran escala, dan una aproximación realista y económica para validar funcionalidades de *slicing* en el laboratorio.

Barakabitze et al.: 5G network slicing using SDN and NFV: taxonomy, architectures and future challenges (2019)

Barakabitze et al. [40] presentan un *survey* sobre las arquitecturas y los retos asociados al *network slicing*, con énfasis en SDN y NFV. Su trabajo cubre fundamentos técnicos, iniciativas industriales y proyectos colaborativos, dando una foto del panorama tecnológico de 2019.

3.1. Revisiones bibliográficas

El texto se separa en requerimientos de 5G, paradigma de software en redes, tecnologías habilitadoras (SDN, NFV y contenedores), iniciativas industriales, orquestación y gestión de extremo a extremo y, finalmente, un capítulo dedicado a desafíos abiertos y líneas de investigación.

Entre las contribuciones se encuentran una comparación de arquitecturas de *slicing* (modelos centralizados vs. modelos distribuidos), una revisión de proyectos industriales y pruebas de concepto que ejemplifican la adopción de SDN/NFV para *slicing*, y el avance de los orquestadores relevantes de software libre (principalmente OSM [41] y ONAP [42]).

En su carácter abarcativo, el trabajo sacrifica la posibilidad de ahondar sobre *testbeds* concretos: las referencias a pruebas de concepto tienen pocos detalles y los resúmenes de proyectos no incluyen métricas. No obstante, su resumen de retos es una referencia relevante en el contexto de *slicing* porque ayuda a enmarcar algunas problemáticas importantes, como la coordinación entre la orquestación de NFV y control SDN, o el cumplimiento de SLAs en la red móvil.

Moreira et al.: Resource Allocation Influence on Application Performance in Sliced Testbeds (2024)

Moreira et al. [43] presentan un estudio experimental enfocado en entender cómo la asignación de recursos de CPU y memoria en *testbeds* de *slicing* influye en el rendimiento de aplicaciones concretas. El trabajo se basa en experimentar en dos *testbeds* y medir efectos de combinaciones de CPU y RAM sobre la latencia de operaciones de lectura y escritura en una aplicación de base de datos. Su aporte relevante para esta sección es mostrar que la variabilidad de resultados entre *testbeds* es significativa y que lo obtenido por un experimento puede variar en otras implementaciones de la misma maqueta en tanto haya variaciones en el hardware usado, por más pequeñas que sean.

El estudio combina niveles de CPU (1 vCPU vs 4 vCPU) y memoria (2 GB vs 8 GB) y mide latencias de operaciones de lectura/escritura sobre nodos desplegados dentro de *slices* en ambos *testbeds*. El trabajo documenta los generadores de carga, la configuración de Kubernetes, las plantillas de *slice*, los tiempos de despliegue y los resultados. Un hallazgo importante es que la influencia relativa de CPU y RAM sobre la latencia no es uniforme entre los *testbeds*: en uno de ellos la CPU explica un porcentaje de la variación de latencia mientras que en el otro la influencia es un porcentaje diferente.

Las conclusiones de la revisión son más concretas que en otros trabajos. En primer lugar, se llega a que la elección de plantillas de recursos para *slices* debe calibrarse en función del *testbed*. Además, los tiempos de despliegue varían entre infraestructuras, lo que condiciona la escalabilidad de despliegues que buscan ser dinámicos.

Aunque el trabajo es reciente y riguroso, su alcance es muy acotado, abordando solo dos *testbeds*, una aplicación y combinaciones discretas de recursos. Por tanto, sus resultados son representativos para el diseño de experimentos, pero no pretenden generalizar sobre todos los *testbeds* o aplicaciones.

3.2. Testbeds de slicing en RAN y núcleo

Distintos grupos de investigación han presentado propuestas de plataformas abiertas para experimentar con *network slicing* en el plano de acceso radio y en el núcleo de red. A continuación se analizan tres de los desarrollos más representativos, que exploran despliegues para la gestión de de estos componentes en redes 5G.

Pineda et al.: Design and Analysis of an Open-Source SDN-based 5G Standalone Testbed (2023)

El trabajo [44] propone un marco para desplegar un *testbed* 5G SA con integración de capacidades SDN y orientado a su reproducción por la comunidad. Los autores buscan ofrecer no solo la descripción del software abierto usado, sino también dos escenarios de uso (uno completamente virtualizado y otro híbrido con equipos de radio definida por software o SDR) para evaluar la viabilidad del *slicing* y la posibilidad de aplicar políticas de tráfico mediante SDN en un entorno SA.

La metodología combina la integración de proyectos *open-source* con experimentación basada en métricas de rendimiento. Tras revisar alternativas y criterios de selección, los autores eligen un conjunto de proyectos de software libre y documentan en detalle el proceso de despliegue. Construyen dos escenarios: uno totalmente virtualizado, con la herramienta UERANSIM [45] para RAN y UE; y otro híbrido, en el que srsRAN [46] opera sobre hardware SDR para incluir radio real. Para recolectar métricas realizan pruebas de *throughput* y latencia, evaluando además configuraciones de *network slicing* y el uso de políticas SDN para ajustar el tráfico. El documento describe también decisiones de diseño (como reemplazar ONOS [47] por OpenDaylight [48] debido a la disponibilidad de *plug-ins*) y detalla procedimientos de instalación y parametrización de cada componente.

La arquitectura propuesta es modular y está pensada para favorecer la reproducibilidad. Sus bloques principales son:

- Núcleo 5G: Open5GS [49] como implementación del núcleo *standalone*. Los autores justifican su elección frente a alternativas como OAI [50], priorizando documentación y compatibilidad.
- RAN y UE: UERANSIM en el escenario virtualizado y srsRAN (sobre SDR) en el escenario híbrido, para incluir radio real. El uso de UERANSIM se fundamenta en su flexibilidad y facilidad de integración.
- Plano de control: OpenDaylight como controlador central, tras pruebas iniciales con ONOS. La decisión se basa en la mayor disponibilidad de *plug-ins* y en su soporte en entornos OpenStack [51].
- Plano de datos: Open vSwitch [52] como elemento compatible con SDN para el forwarding. Se describen reglas y manipulación de tablas para aplicar políticas y delimitar flujos por *slice*.

3.2. Testbeds de slicing en RAN y núcleo

Los autores destacan la separación de funciones en máquinas virtuales (VMs), el uso inicial de OpenStack y la importancia de documentar cada paso de la instalación para facilitar que sea replicable. Señalan, además, que su *testbed* no incorpora por ahora una solución MANO completa.

El experimento se organiza en escenarios comparativos. Las métricas principales son *throughput* y latencia (en la forma de *Round-Trip Time* o RTT), medidas bajo distintos perfiles de carga y con configuraciones de *slicing* aplicadas. Entre los resultados, se reporta que el escenario híbrido (srsRAN sobre SDR) alcanzó mejor latencia que el escenario totalmente virtualizado. Esta comparación permite discutir el impacto del *stack* virtualizado y de las políticas de *slicing* sobre el retardo.

Esta es la principal referencia utilizada para el despliegue de este trabajo, gracias a que comparte criterios de herramientas y a que sus objetivos están alineados a la intención de implementar *slices*. Cabe aclarar que los autores no agregan configuraciones sobre SDN a esta implementación, por lo que no llega a implementarse un *slicing* de transporte y, por eso, se lo evalúa en esta sección de *slicing* de núcleo.

Esmaily et al.: A Cloud-based SDN/NFV Testbed for E2E Network Slicing (2020)

Los autores presentan 5GIIK [53], un *testbed* en la nube para *network slicing* de extremo a extremo, con énfasis en la integración de orquestador y la incorporación dinámica de VNFs. El objetivo es construir una plataforma completa basada en herramientas *open-source*, priorizando la orquestación y la posibilidad de reproducir experimentos en múltiples dominios.

La metodología combina la selección de herramientas *open-source*, el despliegue distribuido y experimentos iniciales para validar el ciclo de vida de los *slices*. 5GIIK se implementa siguiendo el marco ETSI NFV-MANO: OSM como orquestador, OpenStack como gestor de infraestructura virtual (VIM), y distintas VNFs para los dominios RAN y núcleo. La infraestructura se distribuye físicamente en varias sedes universitarias conectadas por un *backbone*, lo que permite evaluar la instanciación remota y la orquestación entre dominios geográficamente separados. La instanciación de *slices* se organiza con OSM ejecutando las llamadas al VIM para desplegar servicios.

Los componentes principales de 5GIIK son:

- VIM: OpenStack como capa de virtualización para VNFs y unidades de banda base, formando una arquitectura de Cloud-RAN.
- MANO: OSM gestiona la orquestación de descriptores de VNF (VNFDs) y el ciclo de vida de servicios. Su elección se justifica por compatibilidad multi-VIM y menor consumo de recursos frente a alternativas.
- RAN: se usan de srsRAN y OpenAirInterface, detallando la definición del archivo descriptor para núcleo y acceso LTE.
- Transporte: se apoya en la red de campus y el *backbone* del operador para interconectar dominios.

Capítulo 3. Antecedentes de testbeds y posicionamiento de la propuesta

La arquitectura está pensada para operar con múltiples VIM, con descriptores que permiten parametrizar el ciclo de vida y las métricas a recolectar vía VIM. El documento detalla cómo se configuran estos descriptores y qué parámetros pueden personalizarse (CPU, memoria, redes virtuales y puntos de conexión).

Los autores muestran experimentos iniciales centrados en la creación de *slices* mediante OSM, el monitoreo en tiempo real de VMs y el aprovisionamiento de VNFs en distintos VIMs. Se verifican conexiones TCP y comunicaciones UE-servidor en escenarios de prueba para comprobar la integridad funcional de los despliegues. También se comparan criterios de diseño frente a otros *testbeds*, destacando la capacidad de aprovisionamiento dinámico y de operación sobre múltiples VIMs. La evaluación se concentra en validar el flujo entre VNFD y descriptores de instancias de *slice* (NSID) más que en medidas de rendimiento.

Del trabajo se destaca el uso de descriptores para facilitar el aprovisionamiento de *slices*, la orquestación con OSM que resultó eficaz para instanciación en distintos VIMs y para aprovisionar VNFs en tiempo real (pero que, de todas formas, requiere un esfuerzo considerable en la definición de descriptores y en pruebas de compatibilidad), y en la distribución geográfica al desplegar RAN y núcleo en sedes distintas, ya que permite evaluar latencias y efectos de transporte más cercanos a los reales.

Ye et al.: End-to-End Quality of Service in 5G Networks: Examining the Effectiveness of a Network Slicing Framework (2018)

El artículo [54] propone un marco para la implementación de QoS para *network slicing* en 5G. El objetivo es diseñar métodos de división dinámica de recursos y modelos de optimización del uso de recursos de CPU y ancho de banda.

Los autores desarrollan dos mecanismos. El primero es el *slicing* dinámico de recursos radioeléctricos: en el dominio inalámbrico plantean dividir el ancho de banda entre radio bases de manera dinámica según la carga instantánea, con un controlador virtual que ajusta las proporciones para maximizar una función de utilidad atada a restricciones de servicio. Este modelo se formula como un problema de optimización de asignación de ancho de banda entre *macro* y *small cells*.

El segundo mecanismo reparte CPU y ancho de banda entre flujos que atraviesan un nodo NFV, para lo cual se definen perfiles de recursos y se busca minimizar colas y balancear flujos con distintos perfiles. A partir de análisis teóricos y simulaciones, muestran reducciones de retardos de cola.

El trabajo no incluye un *testbed* experimental, pero propone un método de orquestación que gestiona los parámetros de radio y cableados. Los modelos usan información como ubicación de dispositivos, estado del canal y carga por celda para tomar las decisiones.

La validación se hace con simulaciones que modelan tráfico y perfiles de recursos. Los resultados muestran que el segundo mecanismo puede reducir los retardos de cola en el enlace de salida de un nodo NFV y que el *slicing* dinámico de ancho de banda entre celdas mejora el desempeño de la red frente a asignaciones estáticas.

3.3. Testbeds de slicing en transporte

La capa de transporte es la encargada de llevar el tráfico de cada *slice* con las garantías definidas por los SLA. En consecuencia, la investigación en *slicing* de transporte aborda varios aspectos: abstracción y orquestación de recursos, algoritmos de asignación, mecanismos SDN para provisión y aplicación de políticas, y validación de aislamiento y KPIs en entornos reales o emulados.

En esta sección se analizan cuatro trabajos que contribuyen a este conjunto de problemas desde perspectivas diferentes: una implementación de SDN, la validación experimental en el *testbed* óptico de escala realista, un estudio sobre las limitaciones de herramientas *open-source* para simular 5G que incluye consideraciones sobre el transporte conectado a RAN y núcleo, y un *testbed* de extremo a extremo enfocado en abordar la automatización de la red.

Es discutible que algunos de estos trabajos trabajan con la RAN y el núcleo en el afán de ir de extremo a extremo, como también se puede plantear que algunos trabajos del apartado anterior segmentan el transporte con SDN. El criterio utilizado para separar los trabajos fue el foco y metodología de *slicing*, según los cuales se puede discutir un dominio por sobre los demás, dando lugar a aquellos que, como en esta sección, traen más aportes al estudio de *slicing* de transporte.

Fanibhare et al.: SDN-Driven Network Slicing Enabling Scenario-Based Applications in Tactile Internet (2024)

El artículo [55] presenta TINetS3, una propuesta orientada a los requisitos exigentes del “Internet Táctil”: latencias muy bajas, alta confiabilidad y suficiente *throughput*. Los autores sostienen que este tipo de aplicaciones requiere *slices* a medida, aprovisionados y controlados bajo demanda, por lo que su contribución se centra en tres escenarios concretos: *slicing* de topología, de servicios y de emergencias. El trabajo busca demostrar que algoritmos basados en SDN (con OVS) pueden materializar estos *slices* y cumplir con los KPI definidos.

Estos algoritmos se implementan sobre una arquitectura con SDN que programa reglas en switches basados en Open vSwitch. La validación se realiza en un emulador que reproduce cargas heterogéneas y permite medir *throughput* y RTT por *slice*, comparando los resultados con los objetivos de diseño.

La arquitectura incluye un orquestador de *slices* que toma decisiones de asignación, comunica reglas a un controlador SDN y aplica configuraciones en los elementos de *forwarding*. El documento describe cómo las políticas de *slicing* se traducen en flujos y colas en OVS. El despliegue se basa en emulación, lo que facilita reproducir condiciones de tráfico diferentes, aunque no contempla pruebas en hardware.

La evaluación se organiza en experimentos por escenario. Las métricas relevantes son *throughput* y RTT, medidas en *slices* UDP y TCP bajo distintos perfiles de tráfico. Los resultados muestran que los *slices* configurados de acuerdo con los perfiles cumplen con las garantías previstas: por ejemplo, cuando se asigna ancho de banda y restricciones de RTT, las mediciones se mantienen dentro de los rangos esperados. En el escenario de emergencia se observa cómo RTT y *throughput* se

Capítulo 3. Antecedentes de testbeds y posicionamiento de la propuesta

ajustan según la prioridad aplicada. También se compara el rendimiento de los algoritmos con trabajos previos, destacando una mejor adaptación de topología o asignación sin necesidad de re-aprovisionar VNFs.

De los experimentos se concluye, en primer lugar, que la combinación de SDN y OVS permite crear *slices* con condiciones apropiadas para aplicaciones del Internet Táctil. En segundo lugar, que clasificar los escenarios facilita el diseño de algoritmos especializados, más efectivos que enfoques genéricos. Finalmente, que la validación en emulación resulta útil para iterar con distintos patrones de tráfico, aunque deja pendiente el análisis en redes físicas con variaciones en capas 1 y 2. Los autores también señalan que la falta de técnicas de aprendizaje automático limita la adaptación proactiva, y sugieren que incorporar predicción podría mejorar la eficiencia.

Vilalta et al.: Experimental Validation of Resource Allocation in Transport Network Slicing Using the *Adrenaline* Testbed (2020)

Vilalta et al. [56] validan experimentalmente una arquitectura de *network slicing* en el dominio de transporte óptico, que integra además la orquestación de recursos de nube y transporte. El objetivo es mostrar que un planificador de *slices* puede asignar recursos de red y de nube cumpliendo requisitos y reduciendo el bloqueo de solicitudes. También buscan comparar algoritmos de asignación de recursos en el *testbed Adrenaline* [57].

El trabajo propone algoritmos para asignación de recursos y la validación práctica en *Adrenaline*, que incluye capa óptica DWDM, nodos ROADM, transporte IP para redes metro y un plano SDN para orquestación. El flujo seguido es el siguiente:

1. Llegada de una solicitud de *slice* con requisitos de latencia, VNFs y topología.
2. Consulta de NetSlice Planner sobre el inventario de recursos de red y nube.
3. Ejecución del algoritmo de asignación.
4. Provisión e instanciación de los recursos (enlaces ópticos, enlaces MPLS o VNFs en OpenStack).

Se describe la implementación del planificador, las APIs utilizadas y la traducción de las decisiones lógicas a caminos físicos, incluyendo la selección de rutas ópticas y la asignación de longitudes de onda.

De los experimentos se concluye, en primer lugar, que la abstracción de recursos ópticos y MPLS permite al planificador operar sobre un inventario unificado. En segundo lugar, que la ubicación de VNFs en el borde produce RTT más bajos y con menor variación. Finalmente, que la integración de MANO y OSM con SDN hace posible el aprovisionamiento a costa de mayor complejidad operativa, lo que requiere herramientas de gestión adecuadas.

Rouili et al.: Evaluating Open-Source 5G SA Testbeds: Unveiling Performance Disparities in RAN Scenarios (2024)

Aunque este trabajo [58] está centrado principalmente en evaluar desempeño de herramientas de simulación y emulación de RAN, su contribución es relevante para la sección de transporte por la interacción entre dominios: los resultados resaltan cómo la elección entre simulador y emulador incide en métricas que atraviesan el transporte (*throughput* y RTT) y en la utilización de CPU y energía. Los autores buscan cuantificar disparidades entre escenarios para orientar la búsqueda de herramientas a usar según el propósito experimental.

De los tres escenarios implementados, el primero es usando UERANSIM para la emulación de gNB y UE, sin implementar capa física. El segundo es con RFSimulator [59], una emulación con mayor detalle del *stack* de radio. El tercero es utilizando OAI sobre hardware SDR, siendo el único que trabaja con la interfaz inalámbrica. Cada escenario dispone de un núcleo 5G SA *open-source* y nodos con conectividad de transporte que permiten medir efectos *end-to-end*. La metodología incluye instrumentación, pruebas en múltiples ubicaciones y escenarios de carga.

Los autores optan por OAI para el gNB en el escenario SDR por compatibilidad y usan UERANSIM y RFSimulator para escenarios más costosos dado que son virtualizados sobre hardware general.

Los experimentos muestran algunas diferencias. Por ejemplo, UERANSIM tiene limitaciones en el modelado de la capa física. RFSimulator demanda mucha CPU y es más errático en su ejecución. OAI sobre SDR produce medidas más cercanas a condiciones reales pero introduce inestabilidades, como la caída del gNB bajo *uplink* intenso o inconsistencias en las conexiones RRC, que a su vez afectan el comportamiento observado en transporte en forma de pérdidas y variaciones de RTT. Las métricas de consumo energético y cobertura del radio también se analizan y se muestran distribuciones, a partir de lo cual recomiendan prudencia al usar emulación para evaluar QoS, ya que los resultados pueden diferir notablemente con el despliegue de SDR.

Los autores señalan como limitaciones la dependencia de *releases* en desarrollo en OAI y la necesidad de comparar estos entornos con redes comerciales a escala para relativizar los resultados.

Chouman et al.: A Modular, End-to-End Next-Generation Network Testbed: Towards a Fully Automated Network Management Platform (2024)

El trabajo [60] presenta un *testbed* con una plataforma de gestión de red orientada a la automatización total para entornos 5G. Sus objetivos son definir la arquitectura de un sistema conforme a 3GPP, proponer un diseño para *testbeds* que integren RAN, núcleo, UEs y aplicaciones, implementar y evaluar elementos de gestión y analítica dentro del sistema, y proporcionar una base para investigación en automatización y gestión. El artículo se orienta a construir una plataforma que sea implementable con tecnologías *open-source* y que permita experimentación reproducible.

La metodología del artículo combina revisión y síntesis de implementaciones

Capítulo 3. Antecedentes de testbeds y posicionamiento de la propuesta

previas, diseño de la arquitectura, implementación de componentes como el núcleo 5G y monitoreo, y experimentación para evaluar la capacidad de emular funciones como registros de UE, señalización y forwarding del plano de datos. Se comparan los proyectos *open-source* de núcleo Open5GS y free5GC [61], y se integran herramientas de orquestación (Kubernetes [62]) y monitoreo (Prometheus [63]) para validar el enfoque de gestión automatizada. El punto es demostrar que un *testbed* de software abierto puede mantener conformidad con 3GPP mientras permite acciones automáticas guiadas por telemetría.

El documento describe una arquitectura detallada y descompuesta en módulos, de donde se destaca que:

- Se analizan alternativas open-source para la implementación del núcleo y se justifica la elección de Open5GS con el cumplimiento del *Release* 16 del estándar. En su implementación proponen ejecutar funciones de red como procesos ejecutables en VM o contenedores, según el módulo.
- El *testbed* contempla integración con gNBs reales (con srsRAN) o emulados (con UERANSIM). Se describe cómo conectar un gNB al núcleo y cómo emular múltiples dispositivos en un host.
- Se implementa la NWDAF (Network Data Analytics Function) para soportar analítica y automatización, incluyendo funciones de recolección de telemetría, entrenamiento de modelos y toma de decisiones automatizadas. NWDAF es señalado como elemento central para la aplicación de inteligencia en la gestión por ser una forma simple de exponer la información del núcleo para la toma de decisiones.
- Se discuten las prácticas de orquestación de Kubernetes para contenedores y Prometheus para monitoreo, además de la necesidad de integrar un MANO y controladores SDN para orquestar servicios. Se plantea un sistema de automatización que incluye despliegue, monitoreo, análisis y acciones correctivas.

Para la evaluación, se describe la comunicación completa de planos de control y de usuario desde el registro de UE hasta el establecimiento de sesiones, para demostrar la fidelidad del entorno y la integración entre RAN y núcleo. El documento muestra trazas para confirmar la conformidad con 3GPP. Luego, se hace una evaluación de la integración de NWDAF: se muestran casos de uso donde la analítica de la función de red observa métricas y dispara acciones como ajustes de configuración.

Finalmente, se hace un análisis comparativo de implementaciones: en la sección de revisión se comparan implementaciones *open-source* para justificar las decisiones de diseño del *testbed*. Sin embargo, no se agregan experimentos comparando rendimiento (como *throughput*), sino que se centra en la demostración de la gestión.

Por otro lado, se reconoce una limitante en el alcance de la validación, ya que la evaluación es funcional pero no incluye estudios de rendimiento que cuantifiquen los límites de la plataforma bajo cargas reales.

3.4. Comparación y diferenciales

El análisis de los antecedentes revisados permite reconocer los aportes de cada iniciativa y situar el valor agregado de este trabajo en relación con las propuestas existentes. Los *testbeds* de Pineda [44], Esmaily [39], Ye [54], Fanibhare [55], Rouili [58] y Chouman [60] dan perspectivas complementarias sobre cómo implementar *network slicing* en distintas capas de la red. Algunos de ellos priorizan la perspectiva de extremo a extremo, otros se concentran en dominios específicos como transporte o RAN, y varios aportan metodologías para demostrar la viabilidad del paradigma. Sin embargo, ninguno reúne simultáneamente una pila 5G SA completa, un control de transporte basado en SDN, y mecanismos dinámicos de adaptación de los recursos en un entorno reproducible como el que se plantea en este trabajo.

El *testbed* de Pineda [44] es el antecedente más cercano al integrar Open5GS, UERANSIM, srsRAN, ODL y OVS en un despliegue que cubre escenarios virtualizados y mixtos. Sus aportes principales son la integración de una red 5G SA y la demostración de que políticas SDN pueden modelar el *slicing*. Frente a eso, este trabajo incorpora mecanismos dinámicos de adaptación que dan un paso más sobre la configuración estática de *slices* mostrada en este antecedente, con más slices y más dispositivos.

Los trabajos de Esmaily [39] y Ye [54] presentan limitaciones sobre las que es fácil proponer un diferencial. El primero hace una comparación de *testbeds* de pequeña escala y criterios de diseño, pero no es una implementación de *slicing* 5G. El segundo desarrolla un modelo analítico para implementar QoS con recursos compartidos, sin materializarlo en una plataforma. En contraste, esta propuesta avanza hacia la validación de la adaptabilidad de los *slices* en condiciones de tráfico real.

Fanibhare [55] introduce TINetS3, donde se explora la versatilidad de SDN para crear *slices* basados en escenarios de *Tactile Internet*. Aunque evidencia cómo los algoritmos pueden adaptar recursos en un entorno emulado, no integra un núcleo 5G ni evalúa tráfico móvil extremo a extremo. Rouili [58], por su parte, muestra la pertinencia del *optical slicing* para redes de transporte, destacando la capacidad de segmentar caminos ópticos mediante controladores. Ambos trabajos muestran la aplicabilidad de SDN en transporte, pero no lo integran en una red *standalone* ni lo validan con tráfico real como en este trabajo.

Finalmente, Chouman [60] presenta un *testbed* modular y conforme a 3GPP, orientado a la automatización mediante herramientas de analítica. Su fortaleza es la visión de integración de monitoreo y control automático, con foco en la orquestación. No obstante, su evaluación no incorpora escenarios de tráfico real ni mecanismos específicos de adaptación dinámica de *slices*. En ese sentido, el presente trabajo se posiciona como un complemento empírico que aporta un mecanismo sencillo pero efectivo para la asignación de recursos en transporte bajo condiciones de video o audio reales.

La comparación puede sintetizarse en la Tabla 3.1, donde se contrastan los principales criterios entre los antecedentes y este trabajo. La tabla muestra cómo,

Capítulo 3. Antecedentes de testbeds y posicionamiento de la propuesta

Tabla 3.1: Comparación de testbeds revisados con respecto a este trabajo.

Trabajo	Dominios cubiertos	Tecnologías principales	Nivel de orquestación	Innovación principal	Limitaciones frente a este trabajo
Pineda et al. [44]	RAN + Núcleo + Transporte	Open5GS, UE-RANSIM, srsRAN, ODL, OVS	SDN con integración básica	Primer stack SA abierto con SDN aplicado a slicing	Sin adaptación dinámica de slices; menor aislamiento entre planos
Esmaily & Kravetska [39]	Análisis comparativo	OSM, OpenStack	NFV-MANO	Criterios de diseño para testbeds pequeños	No implementa un testbed propio
Ye et al. [54]	Teórico (RAN + NFV)	Modelos analíticos	No aplica	Framework analítico para asignación de recursos	No implementa un testbed propio
Fanibhare et al. [55]	Transporte emulado	ODL, OVS	SDN	Algoritmos de slicing por escenarios (topología, emergencia)	No incluye 5G SA ni tráfico real
Rouili et al. [58]	Transporte óptico	<i>Adrenaline</i> , ONOS	SDN	Validación de slicing óptico en laboratorio	Sin integración con Core/RAN ni tráfico real
Chouman et al. [60]	RAN + Núcleo + Transporte	Open5GS, NWDAF, Prometheus, srsRAN/UE-RAN-SIM	Orquestación del core con analítica	Integración de telemetría y automatización	No aplica adaptación dinámica simple
Este trabajo	RAN + Núcleo + Transporte	Open5GS, UE-RANSIM/srsRAN, ODL, OVS, Proxmox VE	SDN + adaptación dinámica	Stack SA completo y escalable con mecanismos de predicción de tráfico	—

a diferencia de los demás, este *testbed* combina una red 5G completa y reproducible, un plano de control SDN y mecanismos de adaptación dinámica, lo que define su aporte más relevante.

Este análisis recoge los trabajos que revisan o implementan *testbeds* de redes móviles y posiciona a este trabajo como un diferencial en el uso del stack SA y controlador SDN para soportar múltiples *slices* en múltiples nodos de radio y dispositivos. También introduce muchas de las herramientas utilizadas por otros trabajos en los distintos dominios, dando lugar al estudio de esas tecnologías y herramientas en el próximo capítulo.

Capítulo 4

Tecnologías y herramientas utilizadas

El estándar 5G introduce *network slicing* como diferenciador para las nuevas generaciones de redes móviles, con todos los beneficios y aplicaciones presentados en el Capítulo 2. En este apartado se describen las tecnologías involucradas en cada dominio de una red 5G *Standalone*, que deben ser contempladas e implementadas en este paradigma de segmentación virtual.

Junto con cada tecnología a integrar al *testbed* utilizado en este trabajo, se plantea la pregunta: ¿qué herramienta existente es la mejor para cumplir esa función en esta red? Cada tramo de la red cuenta con varias opciones para elegir, y en cada caso se busca fundamentar la elección por los motivos que correspondan, pero hay un lineamiento a fijar desde esta introducción, que es la elección de software libre.

La definición del uso de software libre se apoya en la concepción del avance tecnológico como un ámbito colaborativo que, al igual que la ciencia, requiere del intercambio de conocimiento. Todos los componentes de este trabajo pertenecen al ecosistema *open source* para ser reproducibles en cualquier otro proyecto en cualquier contexto, dando, por un lado, auditabilidad al trabajo y sus contribuciones y, por otro, la posibilidad de seguir construyendo sobre el mismo.

Las tecnologías a describir se separan en núcleo 5G, RAN 5G, controlador SDN, switch, y virtualizador del ambiente, con las respectivas herramientas seleccionadas para realizar sus funciones.

4.1. Núcleo de red 5G

El núcleo de red (o *Core Network*) es un componente central encargado de gestionar la conectividad, la movilidad y la provisión de servicios de la red. Es donde se coordinan las funciones de control y de usuario, para que cada dispositivo conectado reciba los recursos y la política de acceso que le corresponden. A diferencia de las generaciones anteriores, el núcleo 5G se diseña desde su origen como una arquitectura completamente basada en software, desacoplada del hardware subyacente y preparada para operar sobre infraestructuras virtualizadas o en la nube.

Capítulo 4. Tecnologías y herramientas utilizadas

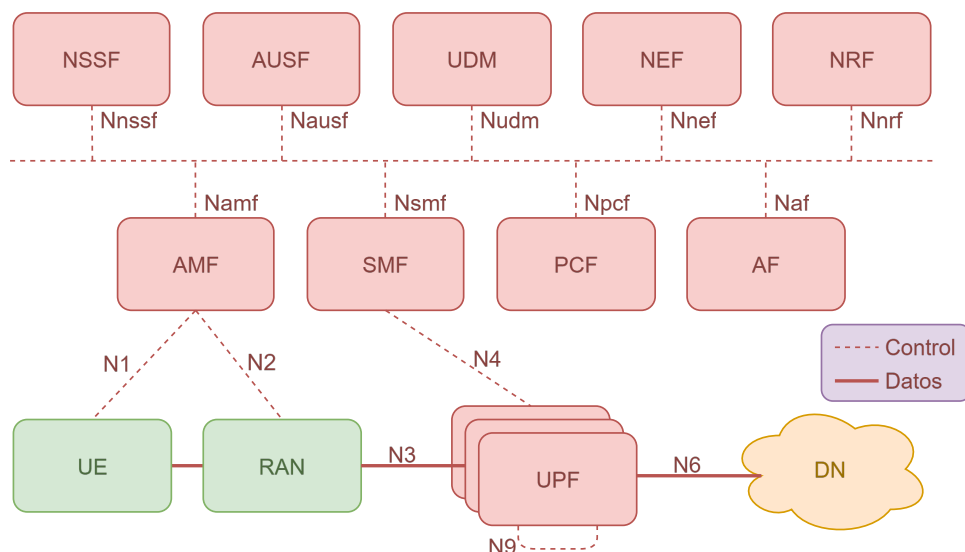


Figura 4.1: Arquitectura de funciones de red del núcleo 5G, incluyendo las funciones del plano de control, las réplicas de funciones de plano de usuario para los distintos slices y las conexiones con la red radio y redes externas.

Su rol es fundamental para la operación de los distintos dominios de red: actúa como intermediario entre la RAN y los servicios de datos o aplicaciones finales, gestionando tanto las sesiones de usuario como la autenticación, la movilidad y cobranza. Gracias a su diseño modular, cada función del núcleo puede escalarse o actualizarse de forma independiente, permitiendo adaptar la red a las necesidades de los *slices* definidos.

4.1.1. Fundamentos y arquitectura

La Figura 4.1 muestra un diagrama de las funciones de red que conforman el núcleo 5G, diferenciando las conexiones que pertenecen al plano de control y al plano de usuario o de datos. En esta arquitectura se observan las principales funciones que serán definidas en más detalle en la sección siguiente. El modelo es introducido por 3GPP bajo el concepto de *Service-Based Architecture* o arquitectura basada en servicios [64].

A diferencia de las generaciones anteriores de redes móviles, donde los elementos del núcleo se comunicaban mediante interfaces dedicadas punto a punto, el núcleo 5G adopta este modelo orientado a servicios. En él, cada función de red puede ofrecer o consumir servicios a través de APIs estandarizadas, eliminando la rigidez de las conexiones entre nodos y permitiendo que cada NF exponga sus capacidades de manera modular, lo que da escalabilidad y flexibilidad a la gestión.

La comunicación entre funciones se basa en el protocolo HTTP y usa el modelo REST, que define un conjunto de métodos para acceder y manipular recursos a través de interfaces web. Los métodos más comunes usados en las APIs del 5GC

son:

- GET: para obtener información de un recurso sin modificarlo.
- POST: para crear un nuevo recurso o iniciar una operación específica.
- PUT: para actualizar un recurso existente.
- DELETE: para eliminar un recurso identificado.

El proceso de comunicación entre NFs sigue un esquema de descubrimiento de servicios. Cada función consumidora identifica qué otras funciones ofrecen el servicio requerido a través del *Network Repository Function* (NRF), que actúa como repositorio del sistema. Este mecanismo da cierta independencia a las funciones con respecto a la topología, permitiendo que nuevas instancias de red se incorporen al sistema sin necesidad de configuraciones estáticas.

Gracias a la comunicación basada en servicios, las funciones de red pueden instanciarse o reemplazarse dinámicamente, sin necesidad de volver a configurar el sistema completo. Esta modularidad también permite compartir funciones entre distintos *slices*, optimizando los recursos y facilitando algunas automatizaciones. Además, la interoperabilidad del modelo SBA con entornos virtualizados favorece la gestión del núcleo, en la que la red puede adaptarse a las demandas de usuarios y servicios.

Otro aspecto diferencial, implícito en la Figura 4.1, es la separación entre el plano de control y el plano de usuario, conocida como *Control and User Plane Separation* o CUPS. Este principio permite que las funciones que gestionan la señalización operen independientemente de las funciones que manejan el tráfico de usuario. Esta separación facilita la distribución del tráfico en distintos planos de usuario, clave para la implementación y orquestación de *slices*.

4.1.2. Funciones de red

Las funciones de red del núcleo 5G son definidas en el mencionado estándar TS 23.501 y materializan la implementación de CUPS en la medida en que cada una de las propias funciones solo puede pertenecer a uno de los planos: todas las funciones se manejan en el plano de control a excepción de la UPF. El objetivo de este apartado es recorrer una a una las funciones y entender sus roles en la red, para luego ver cómo interactúan en el establecimiento de una sesión PDU en la red para un *slice* dado.

Las funciones de red del núcleo 5G pueden describirse de la siguiente manera:

- **NSSF (Network Slice Selection Function)**: determina qué conjunto de *network slices* o instancias de *slice* son adecuadas para un usuario o servicio en función de sus parámetros. Su tarea es informar a la AMF qué *slice* debe asignarse para que cada solicitud de conexión se encamine a través del UPF correspondiente.

Capítulo 4. Tecnologías y herramientas utilizadas

- **UDM (Unified Data Management Function)**: gestiona los datos de suscripción de los usuarios, como los perfiles, credenciales y políticas de acceso. Provee a las funciones de red los datos necesarios para autorizar, autenticar y configurar sesiones. Centraliza la gestión de identidad en el núcleo, dando este tipo de información sobre usuarios a las demás funciones.
- **AUSF (Authentication Server Function)**: se encarga de autenticar a los dispositivos y usuarios que intentan acceder a la red. Utiliza las credenciales de suscripción proporcionadas por la UDM para generar y validar los vectores de autenticación. También participa en la protección de la información de identidad del usuario y en los procesos criptográficos asociados a la conexión.
- **NEF (Network Exposure Function)**: permite que servicios externos y aplicaciones (servicios B2B, por ejemplo) accedan de manera limitada a ciertas capacidades de la red, como información de sesión, políticas o métricas de rendimiento. Lo hace a través de APIs estandarizadas.
- **NRF (Network Repository Function)**: mantiene un catálogo de todas las funciones de red disponibles en el núcleo, junto con la información de sus servicios y direcciones. De este modo, cada función puede descubrir a las demás, eliminando la necesidad de conexiones estáticas.
- **AMF (Access and Mobility Management Function)**: administra el acceso de los dispositivos y la movilidad de los usuarios en la red. Se ocupa del registro inicial, la autenticación, la gestión de ubicación y los traspasos entre celdas o *handovers*. Es responsable de la comunicación inicial entre el UE y el resto del núcleo, actuando como punto de entrada al plano de control.
- **SMF (Session Management Function)**: coordina el establecimiento, modificación y liberación de las sesiones. Define las rutas de datos y selecciona el UPF adecuado para cada sesión, además de gestionar la asignación de direcciones IP y parámetros de calidad de servicio.
- **PCF (Policy Control Function)**: define y aplica las políticas de control que regulan el comportamiento del tráfico, la priorización de servicios y la asignación de recursos. La PCF proporciona reglas de control de flujo y QoS a la SMF y a la AMF, para que cada usuario o servicio cumpla las políticas definidas por el operador.
- **AF (Application Function)**: representa aplicaciones que interactúan con el núcleo de red para solicitar políticas específicas, como requisitos de QoS o condiciones de servicio. Aunque no forma parte directa del plano de control, el AF comunica sus necesidades a través de la PCF o la NEF para adaptar el comportamiento de la red según las demandas del servicio.
- **UPF (User Plane Function)**: es la única función perteneciente al plano de usuario. Su tarea principal es el reenvío del tráfico de datos de los usuarios.

4.1. Núcleo de red 5G

Actúa como punto de interconexión entre el núcleo y las redes externas, estableciendo túneles para cada sesión. También permite la separación y aislamiento del tráfico entre diferentes *slices* o servicios, mediante la instanciación de más funciones UPF paralelas que se comunican por separado con la red externa y aíslan la comunicación entre *slices* desde el punto de vista del núcleo.

No todas las funciones descritas intervienen en un proceso normal de establecimiento de sesión PDU. La Figura 4.2 presenta un diagrama de flujo de ese establecimiento, numerando las etapas que se recorren y señalando las funciones involucradas en cada una. A continuación se describe en mayor detalle el intercambio que ocurre en cada etapa:

1. El UE hace una solicitud para establecer una sesión PDU. Esta solicitud incluye el identificador S-NSSAI para indicar el *slice* deseado. La solicitud pasa a través de la RAN y se dirige a la AMF, responsable de gestionar la movilidad y el acceso del dispositivo.
2. Al recibir la solicitud, la AMF verifica si el *slice* solicitado está permitido para ese suscriptor. Para esto, la AMF consulta a la NSSF, que determina qué NSI o qué conjunto de *slices* son adecuados para el dispositivo.
3. Con esa información, la AMF selecciona la SMF apropiada para el *slice*.
4. La SMF se comunica con la UDM para obtener los datos de suscripción necesarios, incluyendo la identidad del usuario, sus credenciales de seguridad y los *slices* permitidos en su perfil. La UDM confirma que el usuario está autorizado para el *slice* seleccionado.
5. La SMF consulta a la PCF para obtener las políticas asociadas al *slice*, incluyendo reglas QoS, prioridades de tráfico y restricciones de uso específicas del *slice*.
6. Se establece la sesión entre la SMF y el UPF correspondiente al *slice* autorizado. La SMF gestiona la creación del túnel de datos y asegura que el tráfico del usuario se encamina a través del UPF que pertenece al mismo NSI.
7. La SMF se comunica con la RAN para asignar los recursos de radio necesarios. En este punto, la RAN puede adaptar la configuración de radio según los parámetros del *slice*.
8. La RAN gestiona los recursos de radio del dispositivo para que la asignación cumpla las políticas del *slice*.
9. La RAN responde a la SMF confirmando la disponibilidad de los recursos y la correcta configuración del *slice* en el plano de acceso.
10. Se establece la comunicación entre la SMF y el UPF para configurar el túnel con la RAN, de manera de que el tráfico fluya únicamente a través de los recursos del *slice* autorizado.

Capítulo 4. Tecnologías y herramientas utilizadas

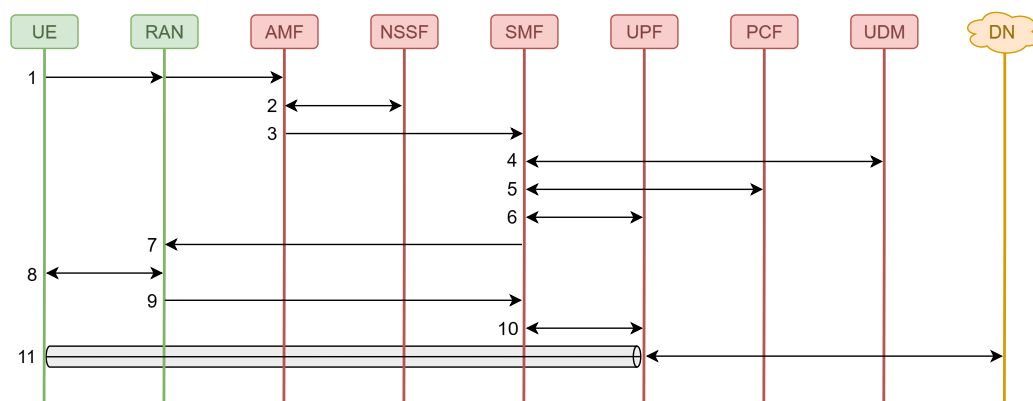


Figura 4.2: Flujo para el establecimiento de una sesión PDU en 5G.

11. Se crea el túnel de datos desde el dispositivo del usuario hasta el UPF del *slice* correspondiente, completando el establecimiento de la sesión.

4.1.3. Herramienta elegida

Para la implementación del núcleo 5G en el *testbed* se consideraron tres opciones disponibles: Free5GC, Open5GS y OAI-5G CN [65]. Todas son utilizadas o mencionadas en los antecedentes técnicos analizados.

Se eligió Open5GS para la implementación final, que combina facilidad de integración y un desarrollo activo por parte de la comunidad, lo que la convierte en una opción sólida para entornos de experimentación.

Open5GS implementa todas las funciones principales del plano de control y del plano de usuario definidas por el estándar 3GPP, como AMF, SMF, UPF, PCF, UDM, AUSF, NRF y NSSF. Cada función se ejecuta de manera independiente, lo que permite desplegarlas en máquinas virtuales o contenedores separados y ajustar los recursos asignados según lo necesario. Se debe notar, sin embargo, que el despliegue por defecto de la herramienta es de todas las funciones en una única máquina o contenedor, por lo que la separación requiere un esfuerzo adicional. En el caso de este trabajo, se ejecutan todas las funciones de control en una única máquina virtual, mientras que cada función de usuario se separa en una máquina distinta.

La elección de Open5GS también está respaldada por su adopción en varios trabajos previos. En particular, Pineda *et al.* [44] lo utilizaron como núcleo en su plataforma basada en SDN, destacando su estabilidad y su integración fluida con OpenDaylight y Open vSwitch. Su trabajo mostró que Open5GS ofrece un buen equilibrio entre completitud de funcionalidades y facilidad de despliegue.

4.2. Acceso radio 5G

La red de acceso radio es la parte de la infraestructura que conecta a los UE con el núcleo de la red. En 5G esta función recae sobre el *gNodeB* y es el primer punto donde se aplican las decisiones de calidad de servicio y aislamiento de *network slicing*. La RAN tiene espacio de toma de decisiones propias: es capaz de asignar portadoras, configurar *slots* y administrar la programación para que los diferentes flujos sean tratados de acuerdo a sus SLA.

La RAN en 5G se diseñó para poderse distribuir mejor en el espacio: sus elementos pueden dividirse en unidades con diferente jerarquía y conectarse mediante interfaces normalizadas para permitir despliegues desagregados, o pueden desplegarse de manera centralizada en un único punto por nodo, dando lugar a varios modelos según la topología del operador [66]. Esta modularidad permite ubicar algunas funciones cerca del borde para, por ejemplo, reducir latencias, o enviar otras a nubes privadas o públicas para optimizar recursos. La distribución de la RAN es un factor a considerar para la implementación de *slices* con requisitos distintos en la misma infraestructura física.

Desde la perspectiva del núcleo, la RAN es la intermediaria entre el UE y las funciones de red, transmitiendo tanto la señalización de control como el tráfico de usuario. Las interfaces N1, N2 y N3 vistas en la Figura 4.1 establecen estos vínculos: N1 para la señalización entre UE y AMF, N2 entre RAN y funciones de control del núcleo, y N3 para el transporte de datos hacia los UPF. En las subsecciones siguientes se desarrollan primero los fundamentos relevantes y, a continuación, la descripción del gNB y del UE, junto con la interacción requerida entre ellos para conectar el dispositivo a la red.

4.2.1. Fundamentos de radio

En 5G, el acceso radioeléctrico se conoce como *New Radio* o NR y marca diferencias importantes respecto a las generaciones anteriores. A diferencia de las arquitecturas previas, NR puede operar en un rango muy amplio de frecuencias, desde menos de 1 GHz hasta bandas milimétricas por encima de los 50 GHz [67].

En la capa física de NR, los recursos se organizan en un plano de tiempo y frecuencia llamado *resource grid*. Cada punto de esa grilla representa un *resource element* o RE, que es la unidad más pequeña que puede asignarse a una transmisión. En el eje del tiempo, los RE se agrupan en símbolos OFDM, y en el eje de la frecuencia, se agrupan en bloques de recursos (*resource blocks* o RB), que contienen 12 sub-portadoras contiguas.

Una de las innovaciones más relevantes de NR es que ya no existe un único espaciamiento entre sub-portadoras, sino que este parámetro puede escalarse. En LTE el valor era fijo en 15 kHz, pero en NR puede multiplicarse por un factor 2^{μ} , tomando valores de 15, 30, 60, 120 o 240 kHz, en lo que se conoce como sistema de numerologías. Gracias a este diseño, una misma red puede operar con diferentes configuraciones de tiempo y frecuencia, asignando numerologías distintas a distintos servicios o *slices*. Por ejemplo, un *slice* pensado para IoT puede usar una

Capítulo 4. Tecnologías y herramientas utilizadas

numerología baja, mientras que un servicio URLLC puede usar una numerología alta para reducir la duración del símbolo OFDM y así disminuir la latencia.

El marco temporal también sigue esa lógica de flexibilidad. Cada cuadro o trama de radio NR tiene una duración de 10 ms, dividido en 10 sub-tramas de 1 ms. Sin embargo, dentro de cada sub-trama, la cantidad de *slots* depende de la numerología: cuanto mayor es el espaciamiento entre sub-portadoras, menor es la duración del *slot*, lo que permite mayor granularidad para asignar recursos.

Un concepto clave dentro de esta estructura es el de *Physical Resource Block* o PRB, que representa la unidad mínima de asignación de recursos dentro del plano frecuencia-tiempo. Cada PRB abarca 12 sub-portadoras consecutivas en el dominio de la frecuencia y un número determinado de símbolos OFDM en el dominio del tiempo, generalmente correspondiente a un *slot*. La cantidad de PRBs disponibles en una portadora depende tanto del ancho de banda como de la numerología seleccionada. Por ejemplo, con un espaciamiento de 15 kHz, una portadora de 20 MHz contiene 106 PRBs, mientras que al duplicar el espaciamiento a 30 kHz, esa misma portadora admite solo 51 PRBs. Los PRBs son la base sobre la cual el planificador o *scheduler* distribuye los recursos de radio entre los distintos usuarios o servicios, y esto se puede observar y modificar en algunas herramientas *open-source* de RAN disponibles.

NR puede operar tanto en modo *Frequency Division Duplex* (FDD) como en *Time Division Duplex* (TDD). Las bandas bajas suelen usar FDD, donde los enlaces de subida y bajada se separan en frecuencia. En cambio, en las bandas más altas (sobre todo las milimétricas) se usa TDD, que alterna los intervalos de transmisión y recepción en el tiempo. TDD permite aprovechar mejor el espectro, aunque tiene mayores requerimientos a nivel del sincronismo temporal de la red para evitar interferencias.

En cuanto a los canales de la capa física, NR mantiene la separación entre canales lógicos, de transporte y físicos. Los canales físicos son los que se transmiten por el aire, e incluyen el *Physical Downlink Shared Channel* para datos de usuario, el *Physical Downlink Control Channel* para la señalización de control, y el *Physical Uplink Shared Channel* para la transmisión desde el dispositivo hacia la red. Además, hay señales de sincronización y referencia como el *Synchronization Signal Block* o las *Demodulation Reference Signals*, que permiten al UE sincronizarse y estimar las condiciones del canal.

Otro aspecto importante del diseño NR es el soporte para las nuevas tecnologías de antenas. La capa física fue pensada para funcionar con sistemas MIMO (*Multiple Input Multiple Output*) masivos, con decenas o hasta cientos de elementos de antena. Esto permite formar haces que concentran la energía de transmisión hacia direcciones específicas, mejorando la eficiencia espectral, especialmente en bandas milimétricas donde las pérdidas son mayores. A esa técnica se la denomina *beamforming* y, para implementarla, NR incluye señales de referencia específicas llamadas *CSI-RS* y *SRS* que permiten estimar el canal y ajustar la dirección de los haces en tiempo real.

La calidad de la comunicación inalámbrica en un entorno 5G depende de varios factores físicos. Entre ellos, el ruido y la interferencia son inevitables, y su impacto

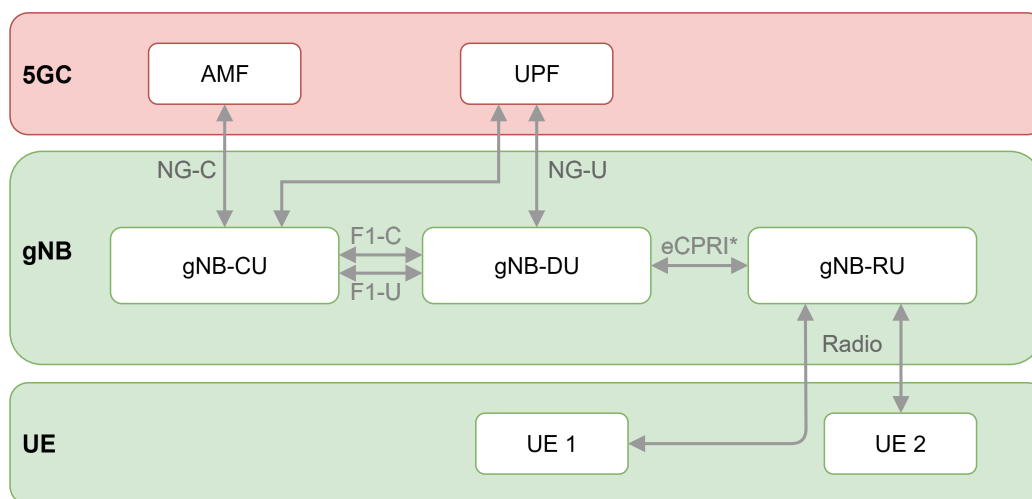


Figura 4.3: Arquitectura general de RAN en 5G: desagregación del gNB en unidades centralizada, distribuida y de radio, con sus conexiones hacia el núcleo de la red (NG-C y NG-U) y hacia los dispositivos (inalámbrica).

crece con el ancho de banda y la densidad de dispositivos conectados. Además, las señales de radio están sujetas a efectos de propagación como la atenuación con la distancia, la reflexión, la refracción y la difracción, que alteran la potencia recibida y pueden provocar *fadings* rápidos o lentos según el entorno. El fenómeno de multicamino, por su parte, genera interferencias constructivas o destructivas cuando varias copias de la misma señal llegan con diferentes retardos, afectando la relación señal a ruido. Estos son factores a tener presentes en cualquier despliegue inalámbrico, incluyendo experimentos de laboratorios en *testbeds* como este.

4.2.2. Componentes de la red

La arquitectura de RAN de 5G se define en la mencionada norma TS 38.401. Su diseño busca hacer más flexible y escalable la gestión de los recursos de radio, aprovechando la posibilidad de virtualización y separación de funciones.

A diferencia de las generaciones anteriores, en 5G se descompone el nodo de acceso en distintas funciones que pueden desplegarse de manera centralizada o distribuida, incluso sobre entornos virtualizados. Esto facilita adaptar la red según la carga de tráfico, la ubicación de los usuarios o las necesidades de cada servicio.

En la Figura 4.3 se muestra la arquitectura general de la RAN. Allí se observa que el elemento principal es el *gNodeB* o gNB, que da conectividad a los UE a través de la interfaz radio y se comunica con el núcleo 5G mediante la interfaz NG.

Cada gNB puede dividirse en tres partes: la *Central Unit* (CU), la *Distributed Unit* (DU) y la *Radio Unit* (RU). Estas unidades se describen a continuación, y sus conexiones se realizan mediante interfaces estandarizadas (F1 entre CU y DU y eCPRI o similares entre DU y RU), lo que permite decidir qué funciones mantener cerca del usuario y cuáles centralizar en la nube.

Capítulo 4. Tecnologías y herramientas utilizadas

Nodo gNB

El *gNodeB* es el punto de acceso radio de la red 5G. Administra los recursos del espectro, establece y mantiene las conexiones con los UEs, coordina los *handovers* entre celdas y vela porque la comunicación cumpla los parámetros de calidad definidos para cada servicio.

El gNB se comunica con el núcleo de red 5G a través de la interfaz NG, que tiene dos componentes: NG-C, para la señalización y control hacia la función AMF, y NG-U, para el tráfico de usuario hacia la función UPF.

Internamente, el gNB se organiza en tres módulos:

- **CU (Central Unit):** gestiona las funciones de control, como el manejo de conexiones RRC (*Radio Resource Control*), la configuración de las celdas y la coordinación con el núcleo de red.
- **DU (Distributed Unit):** se encarga de las operaciones que requieren baja latencia, como el procesamiento de los protocolos de capa de enlace. La totalidad de protocolos del *stack* incluidos en la DU se define según la implementación que se elija para el *fronthaul*.
- **RU (Radio Unit):** maneja la conversión digital-analógica, la amplificación y la transmisión de señales de radio, interactuando directamente con las antenas.

Equipo de usuario UE

El *User Equipment* es el dispositivo del usuario final que se conecta a la red 5G, que puede ser, por ejemplo, un teléfono, módem o equipo IoT. Su función es iniciar la conexión, autenticarse ante el núcleo, intercambiar datos y mantener la sesión activa mientras el usuario se desplaza o cambia de celda.

El UE implementa el *stack* de protocolos NR, que incluye:

- **Capa PHY (física):** responsable de la transmisión y recepción de señales sobre el aire.
- **Capa MAC (Medium Access Control):** asigna los recursos radio y regula el acceso al canal compartido.
- **Capa RLC (Radio Link Control):** garantiza la entrega de los datos mediante retransmisiones y control de flujo.
- **Capa PDCP (Packet Data Convergence Protocol):** realiza la compresión de cabeceras, cifrado y control de integridad.
- **Capa SDAP (Service Data Adaptation Protocol):** asigna los flujos de datos a los identificadores de calidad de servicio.
- **Capa NAS (Non-Access Stratum):** gestiona el registro del dispositivo, la autenticación y la creación de sesiones PDU con el núcleo de red.

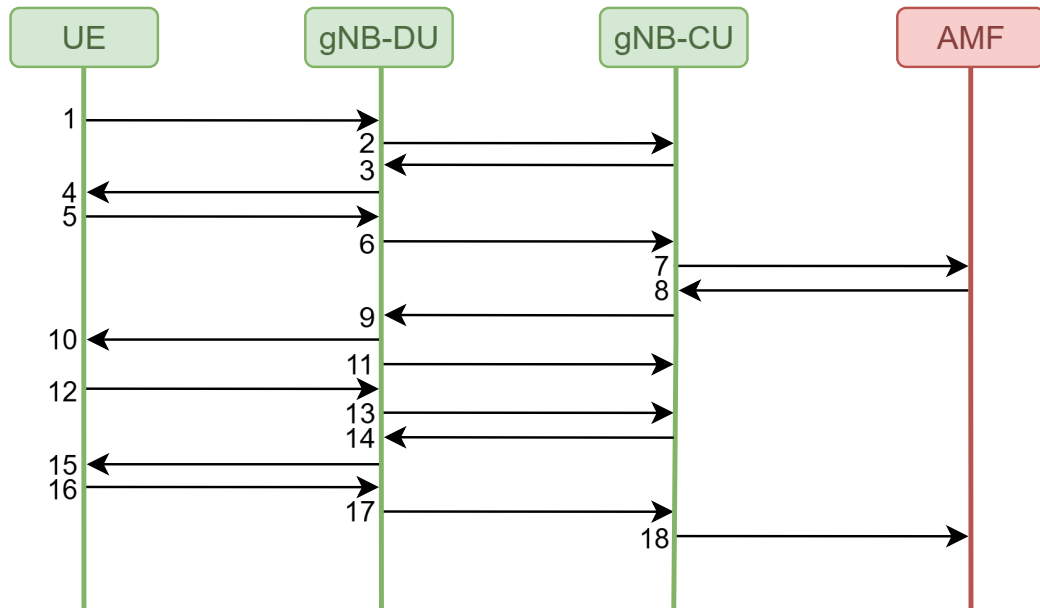


Figura 4.4: Procedimiento de acceso inicial de un UE en la RAN 5G.

Estas capas trabajan en conjunto con las del gNB para establecer y mantener la comunicación, adaptándose a las condiciones de radio y al tipo de servicio en curso.

Interacción entre UE, gNB y núcleo

La interacción entre el equipo de usuario, el gNB y el núcleo 5G sigue el procedimiento de acceso inicial definido en el estándar. En la Figura 4.4 se presenta el flujo completo de señalización. A continuación, se describen las etapas principales del proceso en el que un dispositivo pasa a tener una conexión activa con el núcleo 5G.

1. El UE envía un mensaje *RRCSetupRequest* al DU, iniciando la conexión con la red. Este mensaje incluye información sobre la identidad del usuario y el tipo de servicio que solicita.
2. El DU recibe la solicitud y, si el UE es admitido, genera la configuración de capa baja correspondiente al dispositivo. A continuación, encapsula el mensaje RRC dentro de un *INITIAL UL RRC MESSAGE TRANSFER* y lo envía al CU. Si se trata de un dispositivo de capacidad reducida (*Reduced Capability UE*, en general usado para IoT), se indica en el mensaje una *NR RedCap UE indication*.
3. El CU asigna un identificador *F1AP ID* al UE y genera un mensaje *RRC-Setup* dirigido hacia el dispositivo, que viaja a través del DU. La respuesta se encapsula en un *DL RRC MESSAGE TRANSFER* para su envío por *downlink*.

Capítulo 4. Tecnologías y herramientas utilizadas

4. El DU reenvía al UE el mensaje *RRCSetup*, conteniendo los parámetros de configuración necesarios para completar la conexión de radio.
5. El UE responde al DU con el mensaje *RRC CONNECTION SETUP COMPLETE*, confirmando la aplicación de la configuración enviada.
6. El DU encapsula este mensaje dentro de un *UL RRC MESSAGE TRANSFER* y lo reenvía al CU, completando el intercambio inicial de señalización RRC entre el dispositivo y la red.
7. El CU envía un mensaje *INITIAL UE MESSAGE* al AMF, notificando que un nuevo usuario fue detectado en la red y que se estableció con éxito la comunicación RRC.
8. El AMF responde al CU con un *INITIAL CONTEXT SETUP REQUEST*, ordenando la creación del contexto de usuario necesario para la autenticación, la autorización y la gestión de movilidad.
9. El CU envía al DU un mensaje *UE CONTEXT SETUP REQUEST* para establecer el contexto correspondiente en la unidad distribuida. En este mensaje puede incluirse también el *SecurityModeCommand* con los parámetros de seguridad para cifrado y autenticación. Además, el CU agrega el identificador PLMN ID del operador.
10. El DU reenvía el *SecurityModeCommand* al UE, indicando que debe activar los algoritmos de cifrado y autenticación especificados por la red.
11. El DU envía el mensaje *UE CONTEXT SETUP RESPONSE* al CU, confirmando que la creación del contexto en la unidad distribuida fue exitosa.
12. El UE responde al comando de seguridad con un mensaje *SecurityModeComplete*, notificando que los parámetros de seguridad se aplicaron correctamente y que la conexión puede continuar bajo las nuevas condiciones cifradas.
13. El DU encapsula el mensaje RRC dentro de un *UL RRC MESSAGE TRANSFER* y lo envía nuevamente al CU.
14. El CU genera un mensaje *RRCReconfiguration* para ajustar los parámetros de conexión, asignar recursos adicionales o activar servicios específicos de red. Este mensaje se encapsula dentro de un *DL RRC MESSAGE TRANSFER*.
15. El DU transmite el mensaje *RRCReconfiguration* al UE, que recibe las nuevas instrucciones para adaptar su configuración de radio y protocolos.
16. El UE responde con un mensaje *RRCReconfigurationComplete* al DU, confirmando que la reconfiguración fue aplicada correctamente.
17. El DU encapsula nuevamente el mensaje en un *UL RRC MESSAGE TRANSFER* y lo reenvía al CU como parte del cierre del proceso de establecimiento de contexto.

18. Finalmente, el CU envía un *INITIAL CONTEXT SETUP RESPONSE* al AMF, indicando que el establecimiento del contexto de usuario y de la conexión RRC se completó con éxito. En este punto, el UE se considera registrado y listo para iniciar sesiones PDU o transmisión de datos de usuario.

Este procedimiento completa el acceso inicial, validando que el usuario esté autenticado, que los recursos estén asignados correctamente y que el tráfico fluya bajo las políticas y parámetros definidos para su sesión.

4.2.3. Herramientas elegidas

Para implementar la red de acceso en el *testbed*, se consideraron tres herramientas de software libre ampliamente utilizadas: OpenAirInterface, UERANSIM y srsRAN. Finalmente, se optó por dos herramientas que permiten cubrir tanto la emulación virtual como la conexión inalámbrica real: UERANSIM y srsRAN. La primera se utiliza para recrear de forma virtual el comportamiento del UE y del gNB, mientras que la segunda permite desplegar una celda 5G física usando equipos SDR y conectar terminales comerciales reales. Ambas herramientas son ampliamente usadas en investigación y desarrollo.

UERANSIM es una plataforma de emulación que reproduce protocolos del *stack* NR definidos por 3GPP hasta las funciones NAS. Esto permite simular la interacción entre el UE y el gNB en un entorno controlado. Su arquitectura facilita la conexión con el núcleo 5G, como Open5GS, a través de las interfaces estandarizadas N1, N2 y N3. Gracias a esto, es posible analizar procesos como el registro, el establecimiento de sesiones PDU o la asignación de *slices*, sin necesidad de contar con hardware físico. En los trabajos revisados en el capítulo de antecedentes, UERANSIM aparece de forma recurrente en *testbeds* dedicados a validar funciones del núcleo o experimentar con controladores SDN, lo que confirma su estabilidad y confiabilidad en entornos de desarrollo académico.

Por otro lado, srsRAN permite dar el paso desde la emulación hacia la operación con señales de radio reales. En este contexto, los dispositivos de radio definidos por software permiten implementar funciones de transmisión y recepción de señales mediante software sobre hardware genérico programable, lo que aporta flexibilidad para la experimentación y el prototipado rápido en entornos de laboratorio. Esta característica los convierte en una opción especialmente adecuada para *testbeds* de investigación, donde es necesario modificar parámetros físicos y de enlace sin requerir equipamiento propietario. La compatibilidad de srsRAN con dispositivos SDR como los modelos de la familia USRP lo convierten en una herramienta muy práctica para implementar un gNB en laboratorio. En la primera parte de este trabajo, srsRAN cumple el rol de gNB *all-in-one*, es decir que implementa tanto CU como DU y RU, encargándose de la capa física NR y del control de acceso. Esto hace posible observar el comportamiento de la interfaz aire, medir la calidad del enlace y analizar cómo influyen las condiciones radio en el desempeño general de la red, sin abordar las complejidades de desagregar el *fronthaul* en alguna de sus variantes. En la segunda parte del trabajo se descarta este componente ya

Capítulo 4. Tecnologías y herramientas utilizadas

que no es viable implementar muchos dispositivos en hardware en el ambiente de laboratorio.

Varios de los *testbeds* revisados también utilizaron srsRAN como base para sus celdas 5G. En todos los casos se destacó su facilidad de integración, su estabilidad y su conformidad con las especificaciones 3GPP.

4.3. Virtualización

La virtualización es una tecnología que permite abstraer los recursos físicos de un sistema como la CPU, el almacenamiento o las interfaces de red, para presentarlos de manera flexible y controlada a uno o varios entornos virtuales. En lugar de depender directamente del hardware, las aplicaciones y servicios se ejecutan dentro de máquinas virtuales, cada uno con su propio sistema operativo y recursos asignados. Esto permite que múltiples entornos independientes convivan sobre un mismo equipo físico, aprovechando mejor los recursos y mejorando la portabilidad y la administración del sistema.

4.3.1. Fundamentos

En el contexto de redes 5G, la virtualización facilita el despliegue de funciones de red, su escalamiento y su migración entre nodos. Esta flexibilidad es necesaria para los escenarios de *network slicing*, donde los recursos deben adaptarse a distintas demandas de servicio. La arquitectura de virtualización [68] se organiza en tres capas: la infraestructura de virtualización (*Network Functions Virtualization Infrastructure* o NFVI), las funciones de red virtualizadas y el bloque de gestión y orquestación. Cada una cumple un rol específico y se comunica mediante interfaces estandarizadas.

Virtualización de hardware

La virtualización de hardware es el punto de partida de toda la arquitectura. Su objetivo es dividir los recursos físicos de un servidor (procesamiento, memoria y almacenamiento) para que puedan ser compartidos de forma segura entre distintas máquinas virtuales. El componente que hace esto posible es el hipervisor, un software que se sitúa entre el hardware y las máquinas virtuales, administrando la asignación de recursos y el aislamiento entre entornos.

Gracias al hipervisor, un mismo servidor físico puede ejecutar varios sistemas operativos independientes, lo que permite crear laboratorios de red complejos sin necesidad de contar con múltiples equipos. Este principio también puede aplicarse en entornos de nube privada o híbrida, donde los recursos se distribuyen entre varios servidores físicos interconectados, manteniendo una vista unificada de los recursos.

4.4. Redes definidas por software

Virtualización de funciones de red

La virtualización de funciones de red o NFV es una aplicación directa del concepto anterior al dominio de las telecomunicaciones. En lugar de depender de equipos dedicados como firewalls, balanceadores de carga o routers, NFV permite que esas funciones se ejecuten como software sobre servidores estándar. Esto no solo reduce los costos de infraestructura, sino que también hace posible desplegar, escalar o reemplazar funciones de red de forma más rápida.

Dentro de la arquitectura NFV, las funciones virtualizadas se denominan VNFs y se ejecutan sobre la infraestructura de virtualización NFVI. La gestión y el control de todo este entorno recaen sobre el sistema MANO, que coordina el ciclo de vida de las VNFs, supervisa el estado de los recursos y permite automatizar su configuración. Esta lógica de desacoplamiento entre hardware y software es la que permite que una red 5G pueda adaptarse a nuevas demandas, como activar un nuevo *slice* o aumentar la capacidad de procesamiento de una función sin interrumpir el servicio.

4.3.2. Herramienta elegida

Para la implementación del entorno virtualizado en este trabajo se consideraron dos opciones: VMWare ESXi [69] y Proxmox Virtual Environment [70]. Entre ellas se optó por Proxmox, una plataforma de virtualización de código abierto que combina máquinas virtuales basadas en KVM y contenedores. Esta herramienta tiene una interfaz web intuitiva y soporte para redes virtuales, gestión de almacenamiento y snapshots, entre otros beneficios que la convierten en una opción muy práctica.

La alternativa de VMWare es la seleccionada por el principal antecedente de este trabajo [44], que provee instrucciones detalladas de cómo hacer el despliegue con ella. Sin embargo, el acceso a ESXi era limitado al momento de implementarlo por lo que debió descartarse. Además, se valora que Proxmox sea basado en Debian por sobre el sistema de Kernel propietario que usa ESXi.

Proxmox permite armar topologías complejas en pocos pasos, integrando tanto el plano de gestión como el de datos. En el *testbed* desarrollado, se utiliza para alojar las instancias de las funciones de red como el núcleo 5G, el controlador SDN o el *switch* virtual, con aislamiento entre ellas y control centralizado de los recursos.

4.4. Redes definidas por software

4.4.1. Fundamentos

El concepto de *Software-Defined Networking* surge como una respuesta a la rigidez de las arquitecturas tradicionales, donde cada equipo de red combina el plano de control y el de reenvío dentro del mismo dispositivo. En estas arquitecturas, modificar una política de encaminamiento o implementar un nuevo servicio

Capítulo 4. Tecnologías y herramientas utilizadas

requiere configurar todos los equipos involucrados.

SDN propone separar el plano de control y el plano de datos. Esta separación permite centralizar la inteligencia de red en un elemento de software denominado controlador, que mantiene una visión global de la topología y puede programar dinámicamente el comportamiento de los dispositivos de red. La comunicación entre ambos planos se realiza mediante interfaces estandarizadas conocidas como *Southbound Interfaces* (SBI) [71], siendo OpenFlow una de las más reconocidas.

Por otro lado, las aplicaciones o sistemas de gestión que interactúan con el controlador lo hacen a través de las *Northbound Interfaces* (NBI), que exponen la red como un conjunto de recursos programables. Este enfoque simplifica la automatización, el despliegue de servicios bajo demanda y la adaptación de políticas de calidad de servicio o seguridad en tiempo real. De esta manera, SDN hace a una red más flexible y controlable por software, alineada con el objetivo de orquestación de la infraestructura 5G.

4.4.2. Herramienta elegida

Entre las implementaciones disponibles de controladores SDN, OpenDaylight destaca por su madurez y compatibilidad con una amplia gama de protocolos. Desarrollado por la Linux Foundation, tiene una arquitectura basada en micro-servicios que facilita la integración de nuevos módulos y la personalización de funcionalidades según el entorno de despliegue.

Otros controladores considerados fueron ONOS y Ryu [72]. El primero, que también es gestionado por la Linux Foundation, se vio en el análisis de antecedentes que es frecuentemente puesto en segundo plano frente a OpenDayLight por menor disponibilidad de *plug-ins*. Ryu tiene la ventaja de dar mayor libertad en la programación del controlador y las interacciones, y de hecho fue utilizado en parte de la línea de investigación [73], pero no se mantuvo en este trabajo por haber dejado de contar con soporte y desarrollo.

En el contexto de este trabajo, se seleccionó OpenDaylight por su implementación del protocolo OpenFlow [74] y su compatibilidad con el *plug-in* OVSDb [75], lo que permite gestionar switches virtuales como Open vSwitch. Además, varios de los *testbeds* analizados en los antecedentes usan OpenDaylight como componente de control SDN, que a su vez cuenta con una comunidad extensa, soporte continuo y actualizaciones frecuentes.

4.5. Switching

4.5.1. Fundamentos

El *switching* es una de las funciones elementales en cualquier red, ya que es el proceso mediante el cual se reenvían paquetes entre diferentes interfaces de red de acuerdo con su dirección de destino. En redes tradicionales, los switches son dispositivos físicos dedicados que operan principalmente en capa 2, conmutando tramas Ethernet mediante tablas de direcciones MAC aprendidas de forma dinámica. Sin

4.6. Resumen de componentes elegidos

embargo, con la virtualización y las redes definidas por software, este concepto se ha extendido hacia implementaciones completamente lógicas, capaces de ejecutarse en servidores estándar sin necesidad de hardware especializado.

En un entorno de red moderno, como una red 5G desplegada sobre infraestructura virtualizada, el switching es necesario para conectar las diferentes funciones de red y las máquinas virtuales que las alojan. En este contexto, los switches virtuales reproducen el comportamiento de un switch físico, pero dentro del hipervisor o del sistema operativo anfitrión. Esto permite crear topologías de red flexibles y aplicar reglas de reenvío o filtrado directamente en software.

El concepto de switching virtual también es clave para el control centralizado propio de las arquitecturas SDN: los controladores, como OpenDaylight, pueden comunicarse con los switches virtuales mediante protocolos estándar como OpenFlow para modificar dinámicamente las reglas de switching. De esta manera, se elimina la necesidad de configurar cada equipo por separado y facilita la automatización de políticas de tráfico.

4.5.2. Herramienta elegida

Para implementar la función de switching en el *testbed*, se seleccionó Open vSwitch, una plataforma ampliamente adoptada en entornos de virtualización y SDN. OVS es un switch virtual de código abierto con soporte para varios protocolos de control y que puede integrarse tanto con hipervisores (como KVM o Proxmox) como con controladores SDN.

Una de sus principales ventajas es su capacidad para operar en modo *kernel space*, lo que permite alcanzar un rendimiento cercano al de los switches físicos, mientras mantiene la flexibilidad de un entorno definido por software. OVS soporta funcionalidades como el etiquetado VLAN, el uso de túneles GRE para interconectar dominios virtuales, y el control individual de colas y flujos.

Además, la mayoría de los *testbeds* de *slicing* revisados en los antecedentes adoptan OVS como componente del plano de datos, validando su elección en este trabajo.

4.6. Resumen de componentes elegidos

Estas elecciones terminan de definir el *stack* tecnológico establecido para la implementación de este *testbed*, que queda como en la Tabla 4.1, donde se detallan además las versiones utilizadas en cada caso.

De esta manera, se da paso a la implementación inicial y pruebas del sistema descritas en el próximo capítulo y a cómo se escala más adelante, manteniendo esta selección de herramientas en el proceso.

Capítulo 4. Tecnologías y herramientas utilizadas

Tabla 4.1: Listado de herramientas y versiones seleccionadas para el testbed por tecnología.

Componente de la red	Herramienta elegida	Versión
Ambiente de virtualización	Proxmox VE	8.2.2 (kernel 6.8.12-8-pve)
5G Core Network	Open5GS	2.7.2
RAN Virtual	UERANSIM	3.2.7
RAN Física	srsRAN	24.10.1
Controlador SDN	OpenDayLight	karaf 0.8.4
Switch Virtual	Open vSwitch	2.13.8

Capítulo 5

Solución base implementada

El *testbed* inicial se diseña con el objetivo de evaluar mecanismos de control y adaptación de recursos en un entorno 5G basado en SDN, utilizando una red de extremo a extremo. Es decir que esta primera implementación valida la correcta integración de los varios componentes involucrados que se presentaron y justificaron en el capítulo anterior, y también implementa los ajustes dinámicos de recursos en una primera instancia para confirmar que la solución es viable para los objetivos del trabajo.

Este sistema se organiza en torno a dos *slices* de transporte: uno prioritario y otro *best effort*. El *slice* prioritario será quien transporte video a demanda desde una plataforma de *streaming* hasta su visualización en un UE físico, mientras que el *slice best effort* comienza con tráfico genérico que usa los recursos que sobran del *slice* prioritario.

El criterio de *slicing* dinámico se centra exclusivamente en el plano de transporte, aprovechando las capacidades del *software-defined networking* y la segmentación por flujos de Open vSwitch. Esto permite aplicar reglas diferenciadas de enrutamiento y de asignación de ancho de banda para cada tipo de servicio, sin extender la segmentación al plano de control ni al acceso radio. Cabe aclarar que también se realiza *slicing* estático del núcleo al separar los flujos en distintos UPFs, aunque no se varían sus recursos por limitaciones de la herramienta por lo que no se tiene en cuenta para esta experimentación.

En este capítulo se presenta el contenido correspondiente a la implementación base del *testbed*: se comienza por desarrollar la arquitectura utilizada, incluyendo la configuración de los componentes y el tráfico de usuario utilizado. Se ejecuta, como primer mecanismo de adaptación dinámica de recursos, el ajuste reactivo que usa umbrales para aumentar o disminuir el *throughput* del *slice* y asegurar que el *slice* prioritario tenga los recursos siempre que los necesita y que deje de tenerlos cuando no los está utilizando. Se introduce también el segundo mecanismo de ajuste, cuyo objetivo es el mismo pero con la implementación de un modelo de aprendizaje automático para aprender el comportamiento del tráfico prioritario en el tiempo para predecir los recursos que va a requerir a continuación y dárselos por adelantado.

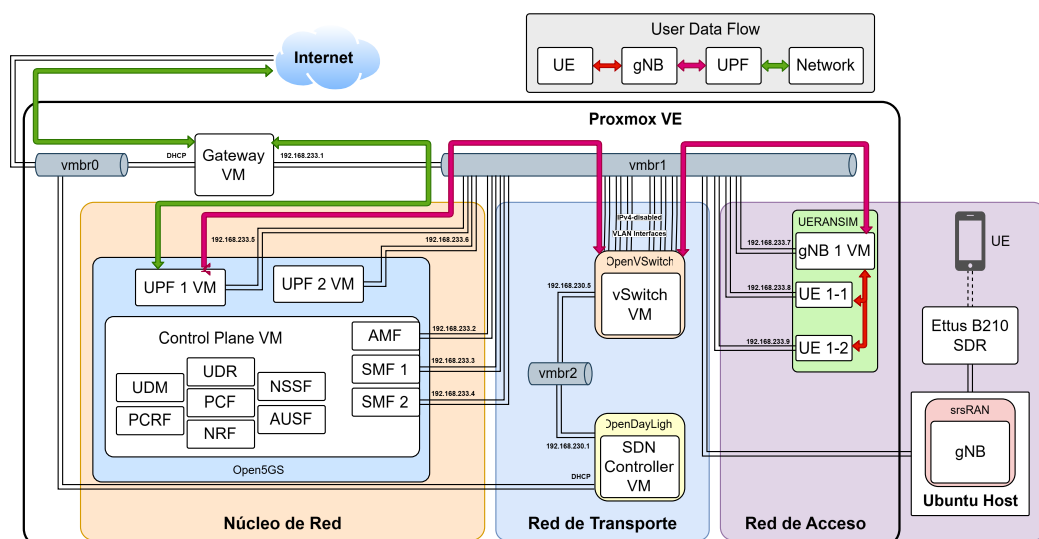


Figura 5.1: Arquitectura inicial del testbed: un dispositivo real conectado a srsRAN por medio de SDR, dos dispositivos virtuales conectados a un gNB virtual y dos planos de usuario que definen los dos slices.

5.1. Arquitectura del testbed

La Figura 5.1 muestra la arquitectura general implementada en el entorno de prueba. La infraestructura se desplegó sobre un entorno de virtualización Proxmox VE, que permitió crear y administrar las máquinas virtuales correspondientes a cada función de red. Dentro de este entorno se implementaron los componentes seleccionados en el capítulo anterior:

- **Core 5G:** Implementado con Open5GS y resaltado en celeste en la Figura 5.1, se dividió en tres máquinas virtuales: una para el plano de control (que agrupa las funciones AMF, SMF, NRF, UDM, AUSF, PCF y NSSF) y dos para el plano de usuario, representando a los dos *slices* definidos. Esta separación facilita el direccionamiento independiente del tráfico de cada servicio.
- **gNB y UEs virtuales:** Para la emulación virtual de la comunicación de radio, se utilizó UERANSIM (en verde en la Figura 5.1), configurado con una máquina virtual para el gNB y dos para los UEs. Cada UE se asoció a uno de los dos *slices*, reproduciendo un escenario con diferentes requisitos de servicio.
- **Controlador SDN:** Se usó OpenDaylight como controlador a cargo de gestionar el flujo de datos en el plano de transporte, como se ve en amarillo en la Figura 5.1. La comunicación entre el controlador y el switch virtual se realizó a través del protocolo OpenFlow. El controlador tiene la particularidad de ser accesible desde el exterior (para interfacear directamente sobre

5.1. Arquitectura del testbed

lecturas y configuraciones). Su única otra conexión es con el switch OVS a controlar.

- **vSwitch:** Open vSwitch operó como conector entre los distintos dominios del *testbed* (ver cuadro anaranjado en la Figura 5.1). Fue configurado para distinguir el tráfico entre ambos *slices* mediante el uso de flujos etiquetados y reglas específicas para cada interfaz virtual. Tiene conexión con todos los componentes de la red por los que circula tráfico de control o datos, solo no pudiendo acceder directamente al bridge Linux de salida (*vmbr0*). Sí accede a los otros dos bridge Linux del sistema: *vmbr1*, que interconecta todas la VMs de Open5GS y UERANSIM separadas en VLANs, y *vmbr2*, por el que se hace la gestión SDN sobre el switch.
- **Gateway:** Un nodo adicional actuó como puerta de enlace hacia Internet, permitiendo la salida del tráfico de los UEs a redes externas.
- **gNB y UE reales:** En configuraciones híbridas, el *testbed* puede operar con una radio base física implementada con srsRAN y un SDR Ettus B210 [76], permitiendo validar las pruebas de conectividad con un teléfono celular comercial conectado mediante un canal inalámbrico. Este componente puede verse externo al entorno virtual, en rojo en la Figura 5.1.

5.1.1. Configuración

La configuración del *testbed* se realizó siguiendo las instrucciones detalladas en la guía de implementación del Apéndice A. A continuación se resume el procedimiento ejecutado en cada etapa: virtualización y despliegue de máquinas, configuración del plano de datos y del controlador SDN, definición de las colas para los *slices*, configuración del núcleo Open5GS y puesta en marcha de la RAN emulada y la RAN real.

Preparación del entorno

Se crearon las máquinas virtuales correspondientes (plano de control Open5GS, dos UPF, controlador OpenDaylight, nodo OVS y nodos RAN) y se definieron los *bridges* en el *host* para segregar tráfico de gestión y de datos. Se asocian múltiples interfaces a un *bridge* y la activación de las mismas con los comandos de sistema de red y **netplan**, así como la instalación de Open vSwitch en el *host*:

```
sudo apt install openvswitch-switch -y
sudo ovs-vsctl add-br br0
sudo ip addr add 192.168.233.1/24 dev br0
sudo ip link set br0 up
sudo ovs-vsctl add-port br0 ens19
```

Estas operaciones ponen al switch en el centro del intercambio de toda la red, para que OVS sea consciente de todo el tráfico de usuario y pueda tanto dar la información pertinente a ODL como implementar las políticas que se le indiquen.

Capítulo 5. Solución base implementada

Integración de OVS y ODL

El switch OVS se conecta a OpenDaylight mediante los comandos estándar para establecer el controlador, manager y la versión de OpenFlow:

```
sudo ovs-vsctl set-controller br0 tcp:<ODL_IP>:6633
sudo ovs-vsctl set-manager ptcp:6640
sudo ovs-vsctl set bridge br0 protocols=OpenFlow13
```

Tras aplicar los puertos al *bridge* se añade un flujo por defecto para permitir el encaminamiento sin reglas del tráfico que no coincida con las condiciones específicas:

```
sudo ovs-ofctl add-flow br0 -O OpenFlow13
"table=0,priority=100,actions=normal"
```

Creación de flujo y colas

Para implementar la diferenciación por *slices* en el plano de transporte se configuraron colas asociadas a los puertos relevantes. Los comandos usados en el *testbed* para crear QoS y colas y asociarlas a la interfaz del gNB (ejemplo con la interfaz **enp2s1**) son los siguientes:

```
sudo ovs-vsctl -- \
  set port enp2s1 qos=@qosgnb -- \
  --id=@qosgnb create qos type=linux-htb \
  queues:10=@slice1 \
  queues:20=@slice2 -- \
  --id=@slice1 create queue other-config:max-rate=30000000 -- \
  --id=@slice2 create queue other-config:max-rate=10000000
```

de donde se puede ver que se asocia un identificador para cada cola y un *throughput* máximo (**max-rate**) en bits por segundo. Tras la creación de estas colas, la verificación se realiza con:

```
sudo ovs-vsctl list queue
sudo ovs-vsctl list qos
sudo ovs-vsctl list port
```

La primera cola (ID 10) se inicializa con 30 Mbps y la segunda (ID 20) con 10 Mbps. Estos valores son el estado inicial y pueden modificarse durante la ejecución.

Una vez creadas las colas, se añaden reglas OpenFlow que mapean tráfico procedente de determinados puertos (por ejemplo, los puertos asociados a los UPF) a las colas creadas. Los ejemplos documentados en la guía son:

```
sudo ovs-ofctl add-flow br0 -O OpenFlow13 \
  "table=0,priority=200,in_port=9,actions=set_queue:10,normal"
sudo ovs-ofctl add-flow br0 -O OpenFlow13 \
  "table=0,priority=200,in_port=4,actions=set_queue:20,normal"
```

5.1. Arquitectura del testbed

Estas reglas prioritarias se procesan antes que la regla general y permiten imponer la limitación de *bitrate* por *slice* en función del `in_port` asociado al UPF. Se recomienda identificar el número de `in_port` mediante

```
sudo ovs-ofctl -O OpenFlow13 show br0
```

y comprobar la operación con

```
sudo ovs-ofctl -O Openflow13 dump-flows br0
```

Configuración de Open5GS

El despliegue del núcleo en tres VMs se hace usando una para el plano de control y dos para los UPF (uno por cada *slice*). Los archivos de configuración contienen las direcciones PFCP y GTP-U y la información de S-NSSAI que asocia cada UPF a su *slice* correspondiente. En el apéndice se muestran estas configuraciones y el script utilizado para inicializar las funciones del plano de control (llamadas a `open5gs-nrfd`, `open5gs-amfd`, `open5gs-smfd -c /etc/open5gs/smf1.yaml`, etc.).

Configuración de UERANSIM y srsRAN

La RAN emulada se configuró con UERANSIM (y en la variante híbrida se utilizó srsRAN para el gNB físico). Los archivos de configuración del gNB y de los UEs (plantillas `open5gs-gnb.yaml` y `open5gs-ue.yaml`) se adaptaron para apuntar a las IPs del núcleo y para fijar el S-NSSAI correspondiente a cada UE, de modo que cada UE se conecte en el *slice* esperado. Además, se registra a los suscriptores en la interfaz web de Open5GS (por ejemplo, los IMSI de los UEs utilizados).

Verificación y puesta en marcha

Tras la puesta en marcha, se verifica la conectividad extremo a extremo mediante los procedimientos de registro, el establecimiento de sesiones y la comprobación del ruteo de tráfico. Para validar la lógica de adaptación se realizan lecturas periódicas de bytes transmitidos por puerto vía RESTCONF y se ejecutan scripts que realizan PUT sobre los recursos OVS identificados, comprobando que los valores de `max-rate` cambian efectivamente y que las métricas reflejan las modificaciones.

5.1.2. Generación de tráfico

Para caracterizar el comportamiento de los *slices* se utilizaron dos tipos de tráfico en esta etapa. Por un lado, se generó tráfico homogéneo con el objetivo de evaluar el rendimiento sostenido y la reacción de las colas bajo cargas constantes. Esto se materializó con la función `iperf` que genera tráfico de prueba que ocupa todo el ancho de banda disponible.

Por otro lado, se reprodujeron descargas reales de contenido de video a demanda desde la plataforma YouTube [77] con el UE físico conectado al gNB con srsRAN, para analizar el impacto de las variaciones temporales de *bitrate* y las necesidades de adaptación en un escenario de consumo multimedia real.

Capítulo 5. Solución base implementada

Los experimentos con estos flujos permitieron comparar métricas entre el *slice* prioritario y el *slice best effort*, así como estudiar la eficacia de los mecanismos de adaptación implementados en el plano de transporte.

5.2. Adaptación dinámica de slices

En este trabajo se propone un sistema (implementado en scripts de Python) para automatizar el monitoreo de tráfico y adaptación de *bitrate*, por medio de la API que expone ODL para el uso del controlador. Para la adaptación de *slicing* se requiere cumplir con algunas condiciones de base:

- Poder leer o estimar el *throughput* de las interfaces entre UPF y gNB en tiempo de ejecución.
- Poder modificar el *bitrate* máximo de esas interfaces en tiempo de ejecución.
- Contar con algún algoritmo de decisión que se encargue de adaptar los *bitrate* de las colas correspondientes a cada *slice*.

Los primeros dos puntos requieren la validación del uso de la API REST que provee ODL. El último punto puede implementarse con la complejidad que se desee, atado a la disponibilidad de recursos de cómputo para el procesamiento de la información. El escenario desplegado en este *testbed* inicial incluye dos aproximaciones: con un algoritmo básico que define el *bitrate* del *slice* prioritario en base a un umbral, y con un algoritmo que utiliza una red neuronal para predecir el tráfico del *slice* y adaptar el *bitrate* de forma predictiva.

5.2.1. Adaptación reactiva: Umbrales

La primera aproximación consiste de un script básico que, apoyado en funciones de lectura de bytes traficados (método GET al *endpoint* de estadísticas) y de escritura de máximo *bitrate* permitido (método PUT al *endpoint* de configuración), monitorea el *throughput* que pasa por una interfaz y ajusta el *bitrate* que se le da a cada *slice*.

Implementación

El criterio de adaptación se basa en un loop infinito que lee los bytes traficados en un instante, luego espera cinco segundos para leerlo de nuevo (la API se actualiza cada tres o cuatro segundos), y promedia la diferencia de valores para estimar cuántos bytes por segundo pasaron por la interfaz. El loop calcula ese *throughput* para el *Slice 1* y lo compara con un umbral predefinido: si el tráfico por el *Slice 1* se mantiene por debajo del umbral, los *bitrates* de las colas permanecen como en el estado inicial. Si, en algún momento, el *throughput* del *Slice 1* supera el valor de ese umbral, los *bitrates* de las colas se adaptan, dándole un mayor *bitrate* máximo

5.2. Adaptación dinámica de slices

al *Slice 1* (que es prioritario) y dejando la capacidad restante para el *Slice 2* (*best effort*).

El script sigue una lógica como la presentada por el pseudocódigo a continuación:

```
Inicializar tasas de transmisión:
     $Bitrate_1 \leftarrow$  nivel bajo Slice 1
     $Bitrate_{total} \leftarrow$  ancho de banda total disponible
     $Bitrate_2 \leftarrow Bitrate_{total} - Bitrate_1$ 
ConfigurarColas( $Bitrate_1$ ,  $Bitrate_2$ )

Umbral  $\leftarrow$  nivel de tráfico que dispara la reasignación
estado  $\leftarrow$  bajo
while el sistema esté activo do
    Throughput  $\leftarrow$  MedirThroughput(Slice 1)
    if Throughput > Umbral y estado =bajo then
         $Bitrate_1 \leftarrow$  nivel alto Slice 1
         $Bitrate_2 \leftarrow Bitrate_{total} - Bitrate_1$ 
        ConfigurarColas( $Bitrate_1$ ,  $Bitrate_2$ )
        estado  $\leftarrow$  alto
    else if Throughput  $\leq$  Umbral y estado =alto then
         $Bitrate_1 \leftarrow$  tasa inicial
         $Bitrate_2 \leftarrow Bitrate_{total} - Bitrate_1$ 
        ConfigurarColas( $Bitrate_1$ ,  $Bitrate_2$ )
        estado  $\leftarrow$  bajo
    end if
    Esperar(intervalo de muestreo)
end while
```

Este es el primer abordaje al problema, con el objetivo secundario de probar la capacidad de automatizar los procesos de lectura y escritura. Con este programa corriendo exitosamente, es posible dar paso a un criterio más apropiado para la adaptación del *throughput* en las colas.

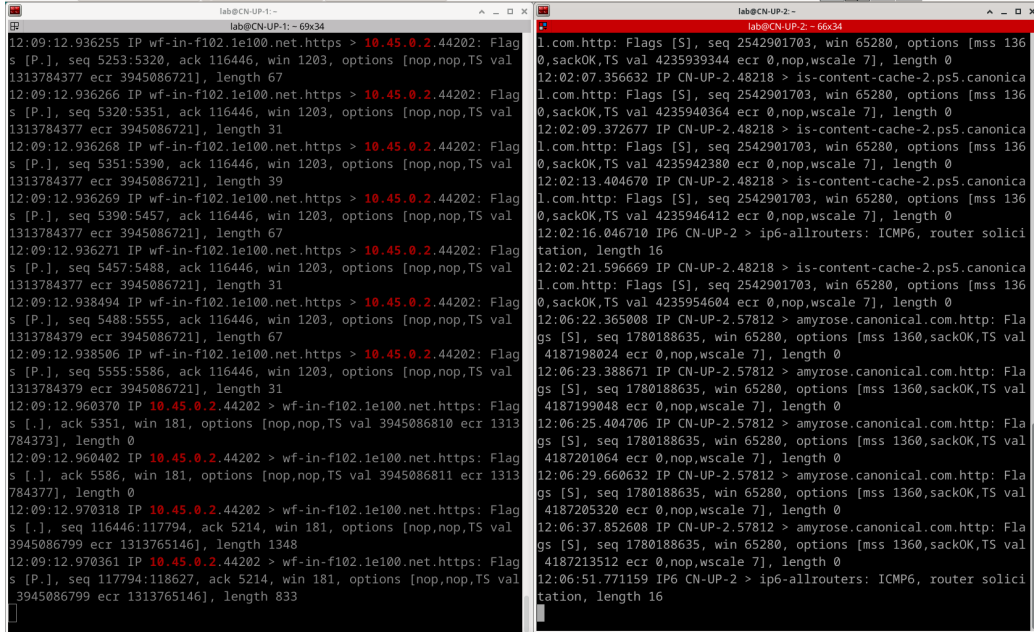
Se desprende del pseudocódigo, también, el apoyo en funciones auxiliares de lectura y escritura, generadas en base al análisis de la API de ODL y a la documentación de comandos provista con OVSDb.

Evaluación

El primer punto a evaluar en todos los casos es el correcto aislamiento de tráfico entre los slices implementados. En cada caso, esto se hace en dos componentes de la red: entre UPFs y entre enlaces de transporte.

El primero se muestra en la Figura 5.2 y se realiza ejecutando la función `tcpdump` en las máquinas virtuales que alojan a las funciones de plano de usuario. En la Figura se puede verificar que, para horarios posteriores a las 12:09, solo el

Capítulo 5. Solución base implementada



```
lab@CN-UP-1:~$ tcpdump -i any -s 0 -w -
12:09:12.936255 IP wf-in-f102.1e100.net.https > 10.45.0.2.44202: Flag
s [P.], seq 5253:5320, ack 116446, win 1203, options [nop,nop,TS val
1313784377 ecr 3945086721], length 67
12:09:12.936266 IP wf-in-f102.1e100.net.https > 10.45.0.2.44202: Flag
s [P.], seq 5320:5351, ack 116446, win 1203, options [nop,nop,TS val
1313784377 ecr 3945086721], length 31
12:09:12.936268 IP wf-in-f102.1e100.net.https > 10.45.0.2.44202: Flag
s [P.], seq 5351:5390, ack 116446, win 1203, options [nop,nop,TS val
1313784377 ecr 3945086721], length 39
12:09:12.936269 IP wf-in-f102.1e100.net.https > 10.45.0.2.44202: Flag
s [P.], seq 5390:5457, ack 116446, win 1203, options [nop,nop,TS val
1313784377 ecr 3945086721], length 67
12:09:12.936271 IP wf-in-f102.1e100.net.https > 10.45.0.2.44202: Flag
s [P.], seq 5457:5488, ack 116446, win 1203, options [nop,nop,TS val
1313784377 ecr 3945086721], length 31
12:09:12.938494 IP wf-in-f102.1e100.net.https > 10.45.0.2.44202: Flag
s [P.], seq 5488:5555, ack 116446, win 1203, options [nop,nop,TS val
1313784379 ecr 3945086721], length 67
12:09:12.938506 IP wf-in-f102.1e100.net.https > 10.45.0.2.44202: Flag
s [P.], seq 5555:5586, ack 116446, win 1203, options [nop,nop,TS val
1313784379 ecr 3945086721], length 31
12:09:12.960370 IP 10.45.0.2.44202 > wf-in-f102.1e100.net.https: Flag
s [I.], ack 5351, win 181, options [nop,nop,TS val 3945086810 ecr 1313
784373], length 0
12:09:12.960402 IP 10.45.0.2.44202 > wf-in-f102.1e100.net.https: Flag
s [I.], ack 5586, win 181, options [nop,nop,TS val 3945086811 ecr 1313
784377], length 0
12:09:12.970318 IP 10.45.0.2.44202 > wf-in-f102.1e100.net.https: Flag
s [I.], seq 116446:117794, ack 5214, win 181, options [nop,nop,TS val
3945086799 ecr 1313765146], length 1348
12:09:12.970361 IP 10.45.0.2.44202 > wf-in-f102.1e100.net.https: Flag
s [P.], seq 117794:118627, ack 5214, win 181, options [nop,nop,TS val
3945086799 ecr 1313765146], length 833

lab@CN-UP-2:~$ tcpdump -i any -s 0 -w -
1.com.http: Flags [S], seq 2542901703, win 65280, options [mss 136
0,sackOK,TS val 4235939344 ecr 0,nop,wscale 7], length 0
12:02:07.356632 IP CN-UP-2.48218 > is-content-cache-2.ps5.canonica
1.com.http: Flags [S], seq 2542901703, win 65280, options [mss 136
0,sackOK,TS val 4235940364 ecr 0,nop,wscale 7], length 0
12:02:09.372677 IP CN-UP-2.48218 > is-content-cache-2.ps5.canonica
1.com.http: Flags [S], seq 2542901703, win 65280, options [mss 136
0,sackOK,TS val 4235942380 ecr 0,nop,wscale 7], length 0
12:02:13.404670 IP CN-UP-2.48218 > is-content-cache-2.ps5.canonica
1.com.http: Flags [S], seq 2542901703, win 65280, options [mss 136
0,sackOK,TS val 4235946412 ecr 0,nop,wscale 7], length 0
12:02:16.046710 IP6 CN-UP-2 > ip6-allrouters: ICMP6, router solici
tation, length 16
12:02:21.596669 IP CN-UP-2.48218 > is-content-cache-2.ps5.canonica
1.com.http: Flags [S], seq 2542901703, win 65280, options [mss 136
0,sackOK,TS val 4235954604 ecr 0,nop,wscale 7], length 0
12:06:22.365008 IP CN-UP-2.57812 > amyrose.canonical.com.http: Fla
gs [S], seq 1780188635, win 65280, options [mss 1360,sackOK,TS val
4187198024 ecr 0,nop,wscale 7], length 0
12:06:23.388671 IP CN-UP-2.57812 > amyrose.canonical.com.http: Fla
gs [S], seq 1780188635, win 65280, options [mss 1360,sackOK,TS val
4187199048 ecr 0,nop,wscale 7], length 0
12:06:25.404706 IP CN-UP-2.57812 > amyrose.canonical.com.http: Fla
gs [S], seq 1780188635, win 65280, options [mss 1360,sackOK,TS val
4187201064 ecr 0,nop,wscale 7], length 0
12:06:29.660632 IP CN-UP-2.57812 > amyrose.canonical.com.http: Fla
gs [S], seq 1780188635, win 65280, options [mss 1360,sackOK,TS val
4187205320 ecr 0,nop,wscale 7], length 0
12:06:37.852608 IP CN-UP-2.57812 > amyrose.canonical.com.http: Fla
gs [S], seq 1780188635, win 65280, options [mss 1360,sackOK,TS val
4187213512 ecr 0,nop,wscale 7], length 0
12:06:51.771159 IP6 CN-UP-2 > ip6-allrouters: ICMP6, router solici
tation, length 16
```

Figura 5.2: Ejecución de tcpdump en ambos slices. El izquierdo (UPF 1) ve pasar el tráfico desde y hacia el UE (con IP 10.45.0.2) mientras que el derecho (UPF 2) solo vio tráfico administrativo tres minutos antes.

UPF 1 (indicado en el encabezado como “CN-UP-1”) contiene tráfico, que además incluye la dirección de la red tunelizada “10.45...”. El otro plano, en cambio, no muestra tráfico desde las 12:06:51, cuando enviaba tráfico de gestión a un dominio de Canonical [78].

El segundo componente a evaluar es el trayecto o enlace entre gNB y UPF, para lo cual se hace uso del switch virtual que interconecta ese enlace: ODL permite leer la cantidad de bytes transcurridos por cada cola de OVS mediante la API, por lo que esta prueba se reduce a hacer una lectura en cada *slice* (en cada cola) antes y después de hacer pasar tráfico por solo una de ellas. Un resultado favorable, como el obtenido en este caso, muestra una diferencia de bytes traficados en el *slice* deseado, mientras que el número de bytes que pasaron por el otro *slice* permanece igual.

Como se presentó en la Sección 2.6, hay dos aspectos más a evaluar: que el máximo *throughput* asignado sea apropiado para lo requerido por el *slice*, y que esa asignación se haga en el tiempo correcto. Es decir, que la magnitud y retardo de la asignación de recursos no deteriore el servicio a transmitir (en este caso, la reproducción del video).

La Figura 5.3 es una herramienta usada a lo largo del capítulo para evaluar estos aspectos: la superposición del *throughput* en función del tiempo con el *max.rate* en función del tiempo. Esto permite ver, por un lado, que el flanco de subida del umbral, correspondiente con el momento de la respuesta de la adaptación, se corresponda con el flanco de subida de *throughput* por descarga del buffer del video. En este sentido, el muestreo utilizado en esta etapa cada 5 segundos da un resul-

5.2. Adaptación dinámica de slices

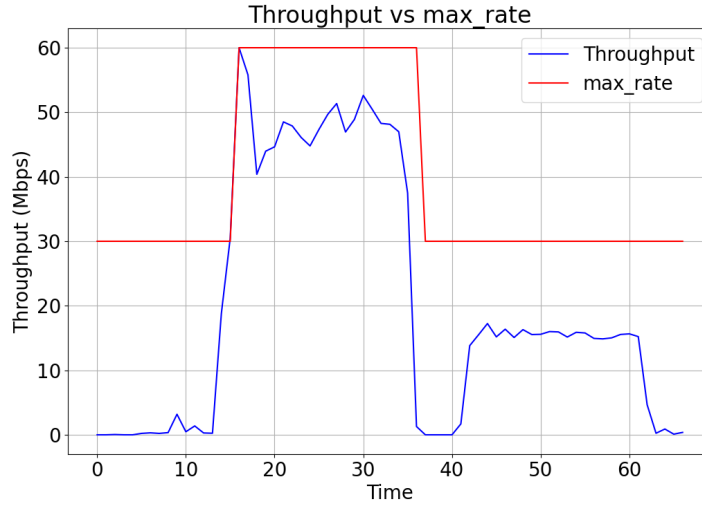


Figura 5.3: Tráfico y bitrate máximo asignado por umbrales al slice prioritario.

tado unívoco: o bien los flancos coinciden en una muestra, o bien el ajuste se hace 5 segundos tarde, afectando con seguridad al video. La Figura muestra cómo, en este caso, coinciden.

Por el otro lado, el aspecto remanente a evaluar es si la magnitud (la altura) de los flancos es suficiente. La elección de umbrales es de 30 y 60 Mbps y el servicio utilizado es video a demanda en resolución 1080p o FHD. Por este motivo, cambiar de umbral es suficiente para validar que la magnitud es apropiada para la carga del buffer. Si, en cambio, se usaran umbrales más escalonados o videos de resoluciones más altas, sería necesario hacer la evaluación de los umbrales sin efectivamente aplicarlos, para que el valor del *throughput* total requerido sea el real y no el acotado por el umbral.

La otra herramienta a utilizar, más difícil de documentar, es el aprovechamiento del método de *streaming* HTTP adaptativo que tienen las plataformas como YouTube para definir lo que significa en los hechos que los recursos sean “apropiados” para el servicio: que el reproductor nunca deba ajustar la resolución a una más pequeña por falta de *throughput* en la red para cargar el buffer de video antes de la reproducción. Esto es, que el video no deba bajar su calidad durante la reproducción, lo cual queda explícito en el reproductor de YouTube en el indicador de resolución del video y es, por tanto, un método replicable para evaluar.

5.2.2. Adaptación predictiva: LSTM

En contraparte, y como paso siguiente en la automatización del proceso de asignación de recursos de *slice*, se propone un mecanismo de adaptación de *bitrate* para optimizar la asignación de *throughput* mediante el uso de un modelo de *Long Short-Term Memory* o LSTM que, al igual que el método anterior, monitoree continuamente el *throughput* de cada *slice* a través de la API del controlador SDN,

Capítulo 5. Solución base implementada

pero que ahora sea capaz de predecir los valores futuros de *throughput* a corto plazo utilizando el modelo LSTM y ajuste el *bitrate* asignado en consecuencia.

La selección del modelo se hizo en base a una revisión de la literatura relativa a la predicción de tráfico en el aprovisionamiento de *slices*. Un primer antecedente directo del enfoque adoptado se presenta en el trabajo de Labonne et al. [79], donde se compara LSTM con otros dos modelos con el objetivo de predecir la utilización futura del enlace con varios segundos de anticipación. A partir de una red simulada, se construye un conjunto de datos recolectados cada 3 segundos, y se entrena cada modelo mediante validación cruzada. Los resultados muestran que LSTM obtiene el mejor desempeño, con errores inferiores al 3 % en la predicción del uso futuro del enlace.

Por otra parte, Rayyis et al. [80] proponen un esquema de predicción de recursos en redes 5G basado en una arquitectura que combina LSTM con redes neuronales convolucionales para mejorar la precisión de la predicción. El modelo se entrena con datos históricos de uso de red y se evalúa mediante métricas estándar de regresión, reportando errores bajos. El uso de LSTM se justifica por su capacidad para modelar dependencias temporales en secuencias de tráfico, lo cual resulta adecuado para escenarios de predicción de demanda en redes móviles.

En contraste con los enfoques basados en redes recurrentes, Khudhus et al. [81] abordan la predicción de parámetros de QoS en *slices* 5G mediante modelos clásicos de aprendizaje automático, específicamente regresión lineal, árboles de decisión y *Random Forest*. A partir del conjunto de datos utilizado, los autores observan que *Random Forest* tiene el mejor desempeño entre los modelos evaluados.

Finalmente, Zhou et al. [82] presentan un esquema de escalamiento dinámico de *slices* asistido por predicción, donde se usa un método de suavizado exponencial [83] para anticipar la demanda de tráfico, mediante el cual se le da más peso a las muestras recientes que a las anteriores para hacer la predicción. A partir de sus resultados se hace el escalamiento de funciones de red virtualizadas para reducir costos y consumo energético. Si bien el enfoque permite una adaptación predictiva, el método limita su capacidad para capturar patrones complejos de tráfico.

De esta revisión se concluye que LSTM es un modelo apropiado para la aplicación, por lo que se procede a entrenarlo e implementarlo en este sistema.

Implementación

La metodología es análoga a la anterior pero con el agregado del modelo: en primer lugar, se consulta periódicamente el controlador SDN para recuperar el *throughput* de cada *slice*. Estos datos se introducen en el modelo LSTM, que genera una predicción del *throughput* para el siguiente instante de tiempo. En segundo lugar, con base en el valor predicho, se toma una decisión sobre la asignación adaptativa de *bitrate*, ajustando los recursos de manera dinámica para evitar congestión o sub-utilización. Finalmente, estos ajustes se implementan a través de la API del controlador.

El modelo LSTM se entrena con datos históricos de *throughput* de múltiples capturas de tráfico, aprendiendo las dependencias temporales para proporcionar predicciones. Para abordar un caso de uso con tráfico real y apuntar a simplificar el

5.2. Adaptación dinámica de slices

entrenamiento del modelo, se selecciona el mismo tipo único de tráfico: contenido VoD en la forma de videos de YouTube. El comportamiento del tráfico sigue una lógica relativamente iterativa: picos de *throughput* en la carga del buffer de video, seguidos por valles de *throughput* mientras este buffer se reproduce.

Para llevar a cabo ese entrenamiento, el primer paso es generar el set de datos necesario. En este caso, se seleccionó un video a demanda de YouTube de larga duración, se comenzó a descargar desde el UE físico (por conexión de radio con el SDR) sobre el *Slice* 1, almacenando en un archivo de texto todos los puntos relevados, conformados por dos datos: tiempo y *throughput*. De esta manera, la serie temporal del comportamiento de *throughput* a lo largo del video evidencia los mencionados picos de carga de buffer que usa YouTube y muchas plataformas de streaming para la reproducción de video.

Luego, estos datos se ingresan en una red neuronal: el modelo se compone de tres capas LSTM con 100, 50 y 50 unidades ocultas, respectivamente y, finalmente, una capa de salida densa. Estos valores fueron los obtenidos al experimentar con varios tamaños de capas del modelo, basados en lo sugerido para una predicción similar de tráfico en función del tiempo en la bibliografía revisada [79].

Evaluación

Nuevamente, se evalúa la asignación de slices en base a su capacidad de aislar y predecir el tráfico. En primer lugar, se repiten y aprueban las validaciones de aislamiento, viendo en las trazas de UPF que el tráfico de video del *Slice* 1 no pasa por el plano de usuario 2. También se verifica que la interfaz del switch que se conecta al UPF tampoco ve ese tráfico. Estos resultados son de esperar puesto que no cambia la arquitectura del *testbed*.

Para la evaluación de la predicción del tráfico, se introduce el error cuadrático medio o RMSE (por sus siglas en inglés), una métrica que mide la diferencia promedio entre los valores que el modelo predice y los valores reales observados. Esto indica qué tan lejos están, en promedio, las predicciones del modelo respecto a la realidad, dando más peso a los errores grandes. Se calcula tomando la raíz cuadrada del promedio de los errores al cuadrado, lo que facilita interpretarlo en las mismas unidades que la variable que se está prediciendo.

En esta parte del trabajo, el RMSE resulta una medida razonable porque el objetivo es evaluar qué tan bien el modelo puede anticipar el *throughput* del *slice*, que es una variable continua y sujeta a variaciones rápidas. Al penalizar más los errores grandes, el RMSE permite identificar si el modelo falla significativamente en ciertos momentos de alta o baja carga, algo muy importante cuando se busca predecir picos de tráfico o caídas abruptas en el rendimiento.

Con esa medida de evaluación clara, se presenta el primer resultado evaluado para la predicción de *throughput*, como se muestra en la Figura 5.4: aunque el modelo logró anticipar el momento de la mayoría de los principales picos de tráfico, tendió sistemáticamente a subestimar o sobrestimar sus magnitudes.

Varias iteraciones con varios tipos de tráfico llevaron a conclusiones similares, por lo que se adoptó una estrategia de medición con mayor resolución usando la herramienta *nload*, configurada para tomar muestras del *throughput* cada 0,5

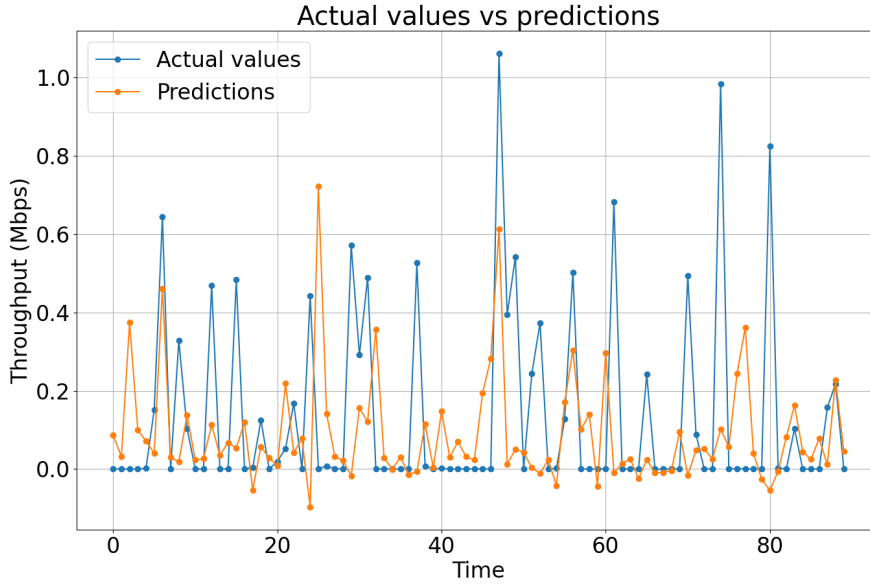


Figura 5.4: Tráfico y bitrate asignado por LSTM al slice prioritario, muestreando cada 5 segundos.

segundos. Esta herramienta se implementa haciendo una conexión `ssh` en paralelo al intercambio con la API del controlador, el cual mantiene la tarea de cambiar la configuración del switch pero no la de leer su tráfico. Se relevan así unas diez mil muestras, que se dividen en 80 % para entrenamiento y 20 % para validación.

Al incorporar estas mediciones más detalladas en el proceso de entrenamiento, la red neuronal mostró una mejora en la precisión: logró mantener el momento de los picos y reflejar con fidelidad sus magnitudes como se muestra en la Figura 5.5, alcanzando un RMSE de 2,6 con datos sin normalizar, lo que equivale al 5,25 % del rango total de los valores. Este cambio de enfoque, necesario por las limitaciones que se enfrentaban, demuestra que mayores frecuencias de muestreo son efectivas para construir un modelo capaz de asignar recursos de red con esta precisión.

Finalmente, se evalúa la demora que existe entre los picos estimados y los picos reales de tráfico. La figura muestra que la diferencia es mínima, coincidiendo exactamente en la mayoría de los casos o fallando en una posición de muestra, lo que equivale a 0,5 segundos y no tiene efecto sobre el video.

Estas pruebas avalan el modelo para la predicción de tráfico de video a demanda, con lo cual se concluye esta primera aplicación funcional sobre un *testbed* 5G SA completo y controlado por SDN, con un UE real que descarga video por la interfaz de aire en un *slice* dinámico aprovisionado por algoritmos automáticos. Este *testbed* fue útil para la familiarización con las distintas herramientas seleccionadas y, en particular, con la orquestación vía SDN con sus beneficios y complicaciones.

El resultado de este capítulo fue presentado en una conferencia del área de las telecomunicaciones y posteriormente publicado [84], con la validación adicional

5.2. Adaptación dinámica de slices

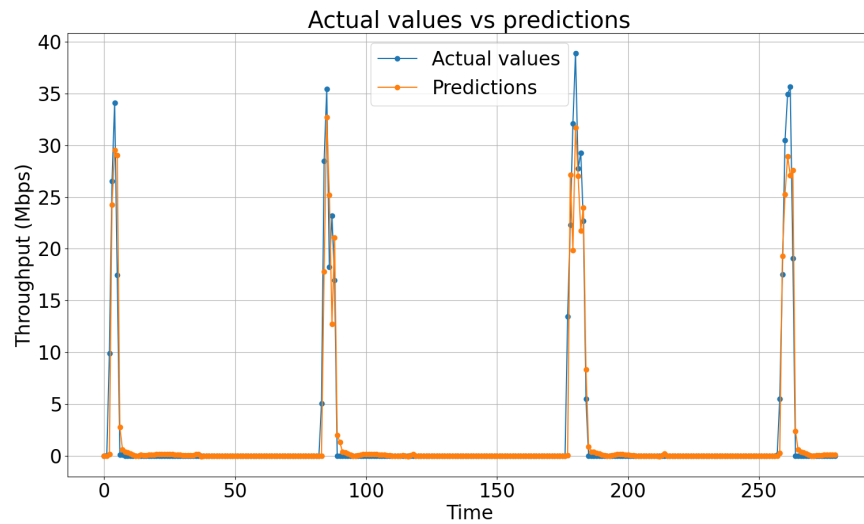


Figura 5.5: Tráfico y bitrate asignado por LSTM al slice prioritario, muestreando cada 0,5 segundos.

de haber sido distinguido con el premio a *Best Paper* del evento. Otro resultado derivado de este trabajo, que fue presentado en una conferencia [85] pero que queda por fuera del alcance de esta tesis, es la clasificación de tráfico de distintos tipos de descarga de video en este mismo escenario.

De todas formas, la predicción se lleva a cabo en un solo gNB, en un *slice* dedicado solamente a esa descarga y sin otros tráficos presentes. Estas limitaciones llaman a dar los siguientes pasos hacia la generalización de la solución para otros casos de uso.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Escalamiento de la solución

Con el modelo validado para un cierto tráfico en una red mínima de prueba, el siguiente paso a tomar es dar relevancia a la solución mediante el escalamiento del *testbed* hacia un tamaño más realista, con menos particularidades del escenario probado y más cercano a un escenario real, como puede ser el de una red privada.

Para esto, se definen varias direcciones en las que extender el trabajo. En primer lugar, se agrega un tercer *slice* que se manejará de manera independiente a los otros dos que se mantienen en sus roles de prioritario adaptable y de *best effort*. En segundo lugar, se generaliza la aplicación al agregar más gNBs: ninguna red consiste de un único nodo de radio, por lo que se agregan dos nodos más que deberán manejarse por separado. En tercer lugar, se agregan dispositivos de usuario: se pasa de 2 a 20 UEs, divididos en *slices* por un lado y en gNBs por el otro. UEs de un mismo *slice* pueden estar separados en gNBs distintos y UEs de un mismo gNB pueden corresponder a diferentes *slices*. Finalmente, se generaliza también el tráfico: se pasa del modelado de tráfico de un único dispositivo en descarga activa de video a demanda, al uso alternado de 20 UEs descargando tráfico de video, audio y páginas web, con duraciones y pausas aleatorias.

Con todos estos criterios, se llega a la simulación de una red como la de la Figura 6.1, en la que puede verse cómo el tráfico de cada *slice* se separa para ir a los UEs de cada gNB, por lo que la gestión de los recursos en cada caso va a depender del uso de ese gNB, además de los criterios preestablecidos para el *slice*.

Sin embargo, las generalidades agregadas en pos del realismo de la solución no vienen exentas de excepciones y limitantes. Algunas a destacar son:

- Al incrementar diez veces la cantidad de dispositivos, queda fuera del alcance el uso de dispositivos reales, pasando a un escenario totalmente virtualizado: los 20 dispositivos consisten en máquinas virtuales Ubuntu 20.04 con el software UERANSIM operando en modalidad UE 5G, usando la interfaz de red `uesimtun0` (creada por esa aplicación) para el tráfico a encapsular y tratar como parte de la red móvil.
- Esta virtualización acarrea una consecuencia adicional: UERANSIM, que emula el *stack* de radio para la conexión con el gNB, no tiene la capacidad de variar las condiciones del canal inalámbrico. A diferencia del uso de `srsUE`

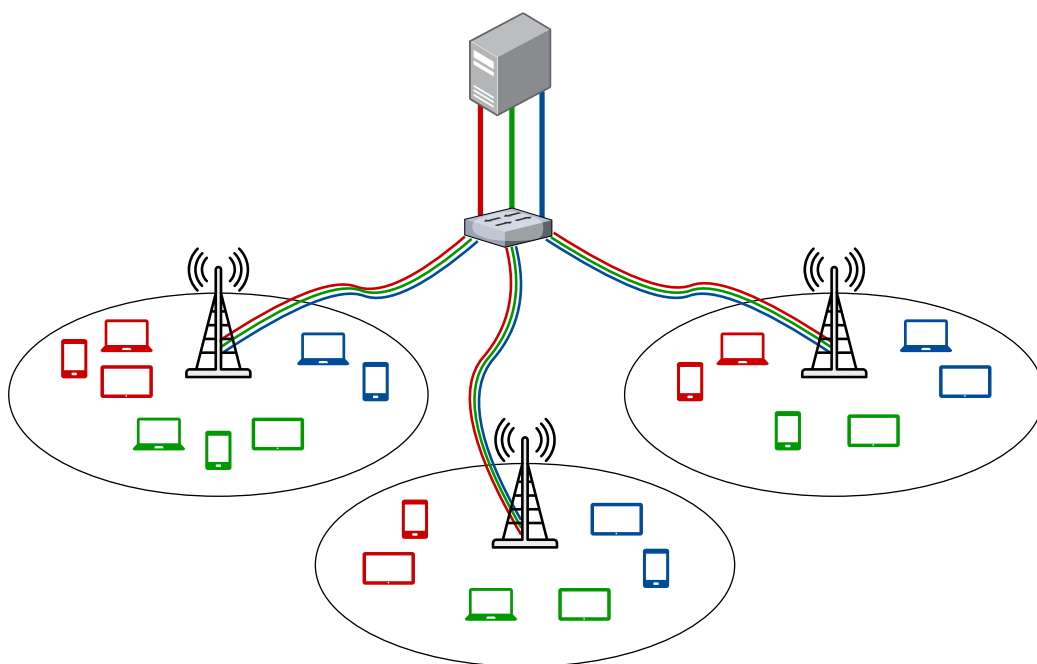


Figura 6.1: Red a simular con escalamiento: el tráfico de los tres slices que sale del núcleo se encamina por el switch a cada uno de los tres sitios de radio para atender a cada servicio en cada sitio.

para esta misma emulación, que utiliza GNU Radio [86] para el pasaje de muestras y que puede incorporar *throttle* para emular pérdidas, UERANSIM mantiene siempre la integridad de toda la comunicación. De todas maneras, se opta por esta última por tener menor consumo de recursos y por tanto facilitar la escalabilidad de UEs.

- El tráfico, como se mencionó, se extiende a tres tipos comunes en dispositivos móviles. Si bien se agrega tanto diversidad de tráfico como aleatoriedad en duraciones de descarga y tiempos de espera, como se va a desarrollar en este capítulo, se termina limitando a tres modalidades de descarga, reproducibles mediante scripts, que se van a repetir para el mismo UE durante todo el experimento.
- Los dispositivos simulados se acotan a 20 por una limitación en la definición del switch virtual utilizado: las interfaces de red de la VM, conectadas al *bridge* que une todos los dispositivos involucrados, tienen un límite a nivel del Proxmox que aloja al sistema, por lo que se define la cota superior en el supuesto de que el alcance de este experimento incluya un único switch para el transporte.

El resto del capítulo describe la arquitectura final de la solución, con las nuevas configuraciones incorporadas y los recursos asignados; luego pasa a detallar los nuevos perfiles de tráfico involucrados, los criterios a considerar para la asignación

6.1. Arquitectura del testbed

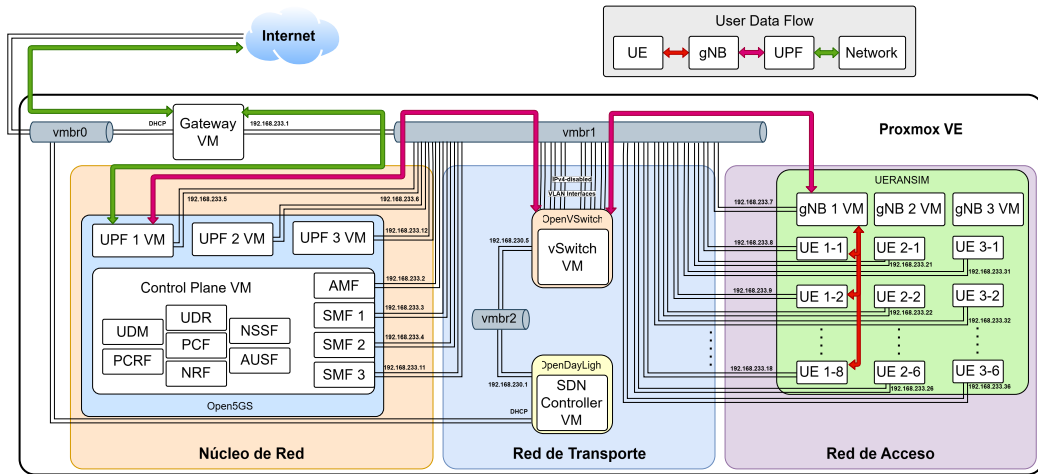


Figura 6.2: Arquitectura final del testbed: se incorpora una tercera UPF con su SMF, dos gNBs y varios UE virtualizados. Tres gNBs con tres UPFs definen nueve flujos diferenciados de slice.

de recursos a los *slices* y, finalmente, lo referente a entrenamiento y evaluación de los modelos de adaptación del *throughput*. Para esto último, se mantiene la selección de la red neuronal LSTM, pero se formaliza el problema y se prueban diferentes funciones de pérdida con nuevos criterios a tomar en un escenario de tráfico con mayor aleatoriedad.

6.1. Arquitectura del testbed

La Figura 6.2 muestra la arquitectura del sistema extendido. La versión final del *testbed* mantiene la misma lógica general de la arquitectura anterior, pero con los mencionados cambios tanto en el número de dispositivos como en la organización de los flujos. La más visible es la incorporación de un tercer *slice* completo, con su propio par SMF-UPF, que permite extender la experimentación a un caso más general, donde coexisten tres tipos de tráfico con características distintas y políticas de asignación diferentes.

En esta nueva estructura, cada gNB mantiene tres flujos asociados, uno por cada *slice*, conformando un total de nueve flujos en el *testbed*. Esta disposición hace que los tres gNB se encuentren en igualdad de condiciones, y la lógica de operación es análoga pero diferenciada en cada uno. Es decir, las reglas de cada *slice* son transversales a los gNBs, pero la predicción de ese mismo *slice* va a depender del tráfico que pasa, por lo que va a depender de las condiciones de cada gNB.

Para identificar de forma unívoca cada uno de los nueve flujos descendentes, se definen reglas en OVS que combinan el puerto de entrada, correspondiente al UPF asignado, con la dirección IP de destino, correspondiente al gNB que atiende a cada UE.

De este modo, cada flujo queda determinado por el par (in_port, nw_dst), lo cual permite clasificarlo y asignarlo a una cola específica. Por ejemplo, la siguiente

Capítulo 6. Escalamiento de la solución

regla asigna todo el tráfico IP proveniente del UPF1 (puerto 6) y con destino al gNB1 (IP 192.168.233.7) por la cola 110:

```
sudo ovs-ofctl add-flow br0 -O OpenFlow13 "table=0,priority=300,
in_port=6,ip,nw_dst=192.168.233.7,actions=set_queue:110,normal"
```

Reglas análogas se aplican para los demás pares (UPF, gNB), completando los nueve flujos experimentales definidos en la arquitectura.

Si se quisiera agregar un nuevo gNB, bastaría con incorporar tres flujos adicionales (uno por *slice*) manteniendo las mismas reglas de asignación. En ese sentido, la arquitectura no depende del número de gNB presentes, sino que replica un patrón de conexión.

6.1.1. Configuración

Uno de los ajustes más relevantes de esta etapa fue la lectura de estadísticas de tráfico. En el despliegue inicial, se explicó que la recolección de métricas con baja latencia dependía de la herramienta externa `nload` que, si bien era útil, implicaba la conexión al sistema por `ssh` para el control de flujos. En la versión final se logró integrar completamente la lectura a través de la API de OpenDaylight, manteniendo la granularidad en el relevamiento del *throughput*.

Para lograrlo, se modificó el archivo de configuración llamado `org.opendaylight.openflowplugin.cfg` ajustando el parámetro `basic-timer-delay` a un valor de 1000 ms. Con esto se consiguió disminuir significativamente el tiempo de *polling* de los datos de OpenFlow, logrando una captura de métricas más fluida con respecto a las cuatro segundos iniciales. Este cambio permitió mantener la consistencia en las predicciones que se había logrado con `nload` sin perder el control mediante la API.

Siguiendo con las modificaciones en configuraciones, varios archivos debieron ser editados en VMs existentes o creados en VMs nuevas, según la asignación de IPs vista en la Figura 6.2.

Un primer ejemplo de modificación se encuentra en el `amf.yaml`, que debe ahora aceptar un nuevo *slice* en el sistema:

```
plmn_support:
- plmn_id:
  mcc: 001
  mnc: 01
  s_nssai:
  - sst: 1
    sd: 000001
  - sst: 1
    sd: 000002
  - sst: 1
    sd: 000003
```

Análogamente, y en la misma máquina correspondiente al plano de control, se modifica la función de *slicing* `nssf.yaml` para el mismo objetivo:

6.1. Arquitectura del testbed

```
- uri: http://127.0.0.124:7777
  s_nssai:
    sst: 1
    sd: 000003
```

También en CN-CP, debe generarse un nuevo archivo `smf3.yaml` con el fin de levantar esa tercera función de manejo de sesiones para el nuevo *slice*. Para eso, y con respecto a los demás SMF configurados, se deben modificar las IP según el diagrama para referenciar tanto a la nueva interfaz de esta máquina que va a corresponder a este SMF, como a la IP de la que será la nueva UPF:

```
pfcf:
  server:
    - address: 192.168.233.11
  client:
    upf:
      - address: 192.168.233.12
      dnn: internet
gtpu:
  server:
    - address: 192.168.233.11
...
session:
  - subnet: 10.47.0.0/16
    gateway: 10.47.0.1
    dnn: internet
...
info:
  - s_nssai:
    - sst: 1
      sd: 000003
      dnn:
        - internet
```

Véase que también se distingue el nuevo *slice* con el parámetro SD en 000003 y la nueva subred con la que se manejará el túnel GTP entre UE y UPF para todos los dispositivos del *slice*, la 10.47.0.0/16.

Este UPF se despliega en la nueva VM mediante un nuevo `upf.yaml`, modificando parámetros similares:

```
upf:
  pfcf:
    server:
      - address: 192.168.233.12
  gtpu:
    server:
```

Capítulo 6. Escalamiento de la solución

```
- address: 192.168.233.12
session:
- subnet: 10.47.0.0/16
  gateway: 10.47.0.1
  dnn: internet
  dev: ogstun
```

Pasando a las modificaciones de UERANSIM, se modifican los archivos UERANSIM/config/open5gs-gnb.yaml en cada una de las dos nuevas VMs de gNBs. En cada una se fija la IP propia, la del AMF, y el soporte a los tres *slices*. A continuación se ve el ejemplo con el gNB-2:

```
linkIp: 192.168.233.20
ngapIp: 192.168.233.20
gtpIp: 192.168.233.20

amfConfigs:
- address: 192.168.233.2
  port: 38412

slices:
- sst: 1
- sst: 1
  sd: 1
- sst: 1
  sd: 2
- sst: 1
  sd: 3
```

Finalmente, se consideran los ajustes en todos los UEs nuevos, distinguiendo las dos características que permiten mapear a qué flujo de tráfico pertenecen: su gNB y su *slice*. En este ejemplo, se ve el UE-22, que pertenece al gNB-2 (distinguible por su IP) y al *slice* 2 (como indica el parámetro SD):

```
gnbSearchList:
- 192.168.233.20
...
sessions:
- type: 'IPv4'
  apn: 'internet'
  slice:
    sst: 1
    sd: 2
configured-nssai:
- sst: 1
  sd: 2
```

6.1. Arquitectura del testbed

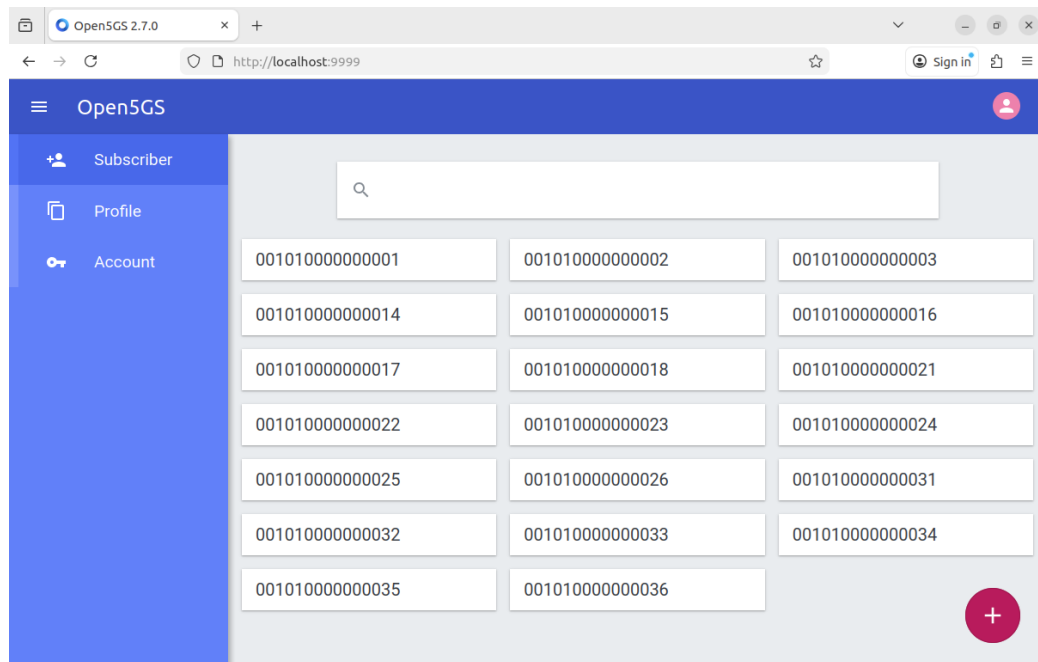


Figura 6.3: Interfaz gráfica del núcleo Open5GS con los dispositivos añadidos.

```
default-nssai:  
- sst: 1  
  sd: 2
```

El cambio final a contemplar es el agregado de un IMSI único para cada UE y, en contraparte, su adición al núcleo a través del plano de control. Para esto, es fácil utilizar la interfaz gráfica que se despliega en la VM del plano de control, ingresando al puerto 9999 del `localhost`, como puede verse en la Figura 6.3.

El despliegue del *testbed* completo se hace con algunos automatismos que simplifican la operación para el usuario, tanto para facilitar la reproducibilidad del *testbed* como para reducir los tiempos de preparación de cada prueba. La automatización se divide en dos niveles: uno asociado al despliegue y destrucción de los servicios principales, y otro destinado a la ejecución de los UEs y sus perfiles de tráfico.

En el primer nivel, se desarrollaron dos scripts en Python que permiten lanzar y detener ordenadamente los servicios de las VMs involucradas. Estos incluyen los componentes de núcleo, las tres funciones de plano de usuario y los tres nodos gNB. De esta manera, todo el entorno puede levantarse en pocos segundos con configuraciones consistentes y predefinidas, garantizando que cada experimento parta de un estado limpio.

En el segundo nivel, scripts de Bash en cada VM de UE se encargan de activar las instancias de UERANSIM que se conectan a los gNB correspondientes. Cada UE queda así vinculado a un único tipo de tráfico y, por tanto, a un *slice* determinado. Los scripts que generan el tráfico propiamente dicho se explican en la sección siguiente, donde se detalla el comportamiento de cada perfil.

Capítulo 6. Escalamiento de la solución

6.1.2. Recursos

La Tabla 6.1 recoge cómo se asignaron disco, memoria y vCPUs a cada VM dentro de Proxmox VE. Esa asignación define la base sobre la cual corren las funciones de control (AMF, SMF, etc.), las funciones de usuario (UPF 1, UPF 2, UPF 3) y las instancias de gNB y UE.

Tabla 6.1: Lista de VMs y sus atributos en Proxmox VE

VM ID	Nombre	Disco (GB)	Memoria (GB)	vCPUs
1000	OVS	24.00	6.00	4
1001	GW	16.00	4.00	2
1002	Controller	32.00	8.00	2
1003	CN-CP	32.00	8.00	3
1004	CN-UP-1	32.00	6.00	3
1005	CN-UP-2	32.00	6.00	3
1006	CN-UP-3	32.00	6.00	3
1010	gNB-1	24.00	4.00	2
1020	gNB-2	24.00	4.00	2
1030	gNB-3	24.00	4.00	2
1111	UE-11	16.00	2.00	1
1112	UE-12	16.00	2.00	1
1113	UE-13	16.00	2.00	1
1114	UE-14	16.00	2.00	1
1115	UE-15	16.00	2.00	1
1116	UE-16	16.00	2.00	1
1117	UE-17	16.00	2.00	1
1118	UE-18	16.00	2.00	1
1121	UE-21	16.00	2.00	1
1122	UE-22	16.00	2.00	1
1123	UE-23	16.00	2.00	1
1124	UE-24	16.00	2.00	1
1125	UE-25	16.00	2.00	1
1126	UE-26	16.00	2.00	1
1131	UE-31	16.00	2.00	1
1132	UE-32	16.00	2.00	1
1133	UE-33	16.00	2.00	1
1134	UE-34	16.00	2.00	1
1135	UE-35	16.00	2.00	1
1136	UE-36	16.00	2.00	1
TOTAL	-	592.00	96.00	46

Las VMs del plano de control agrupan las funciones de señalización, mientras que las VMs que implementan los UPF manejan el procesamiento del tráfico de

usuario. Esta separación refleja parte de la arquitectura de Open5GS y facilita el aislamiento por *slices*: los recursos asignados a cada VM condicionan la capacidad de cada componente para procesar sesiones o flujos de datos.

Es conveniente marcar que los valores de la tabla son recursos asignados, no mediciones de uso. Es decir que describe la capacidad disponible en cada VM sin evaluar cómo se comportó el consumo durante una prueba concreta.

Los puntos más sensibles al aumento de carga suelen ser las funciones que procesan el plano de usuario, por su papel en el encaminamiento y tratamiento de paquetes. Por eso, al escalar el número de UEs o al aumentar fuertemente los bitrates de perfiles como el de video, la capacidad de los UPF y, en general, la CPU disponible en los hosts físicos, se transforma en un factor limitante a tener en cuenta. Estas son consideraciones de diseño y planificación para justificar el dimensionamiento mostrado en la tabla.

6.2. Perfiles de tráfico

El tráfico utilizado en la primera aproximación al problema en el Capítulo 5, si bien refleja realismo en la medida en que se trata de tráfico de video a demanda real, disponible en internet y perfectamente comparable al de cualquier video descargable por usuarios normales (en contraposición con el tráfico generado, por ejemplo, con *iperf*), parte de dos premisas poco generalizables para el *slice*: solo hay un video y no se detiene.

Por un lado, al tener un único UE en el *slice*, el tráfico de este *es* el tráfico del UE, por lo que se simplifica enormemente el modelado del tráfico: solo existen picos de descarga de buffer y valles de tiempo de *playback* en los que se reproduce ese buffer. Por otro lado, todo el tiempo considerado para entrenamiento y validación es tiempo de descarga, por lo que tampoco hay aleatoriedad en tiempos de pausa.

De ahí la propuesta de generalización de tráfico introducida al comienzo del capítulo: dividir cada *slice* en tipos de tráfico real diferente (video, audio y navegación web), y agregar aleatoriedad a cada uno, en la forma de largos aleatorios de espera entre iteraciones y, si aplica, de reproducción.

Una dificultad agregada en la resolución del tráfico de descarga es que las aplicaciones utilizadas en cada perfil hacen uso, por defecto, de la interfaz de red con salida hacia el *Gateway* y luego internet. El problema de esto es que no pasa por el flujo de la red móvil, por lo que no respeta el túnel GTP por el gNB ni el uso de su UPF, lo que implica que en efecto no se use el *slice* ni se monitoree por la fila correcta del switch para medir *throughput* o limitar bitrate.

A continuación se presenta la implementación de cada uno. Tienen en común la ejecución mediante script en forma de loop con tiempos de espera, además de las limitantes ya mencionadas de que cada dispositivo se dedica a un solo tipo de tráfico y que los tiempos de espera se dan en ciertos márgenes necesarios para el contexto de esta experimentación.

Capítulo 6. Escalamiento de la solución

6.2.1. Tráfico de video

Este fue el perfil más desafiante de implementar. En las primeras pruebas se buscó utilizar herramientas como VLC [87] y `yt-dlp` [88] para generar tráfico de video a demanda desde YouTube. Sin embargo, ninguna de las dos herramientas logró asociarse correctamente a la interfaz `uesimtun0`, lo que imposibilitaba que el tráfico pasara por el plano de usuario del sistema 5G. En el caso de VLC, a pesar de la flexibilidad para reproducir flujos en distintos formatos, la herramienta ignora el parámetro de interfaz y envía siempre el tráfico por la ruta por defecto del sistema operativo. Por su parte, `yt-dlp` dependía de VLC para la reproducción directa del contenido, lo que llevaba al mismo problema.

Una alternativa intermedia consistió en el uso de `ffmpeg` [89] para realizar descargas parciales de videos bajo demanda, intentando emular el comportamiento real de descarga y reproducción alternadas. En esas pruebas se introdujeron pausas forzadas en la descarga para generar el efecto de reproducción del buffer, pero el modelo no resultó convincente: durante las pausas, el tráfico caía a cero exacto, lo que no representa bien el comportamiento real de los servicios de video, donde siempre hay cierto nivel de actividad de fondo. Además, los buffers de descarga correspondían al `ffmpeg` intentando descargar todos los paquetes posibles en el tiempo en el que se ejecuta, en lugar de descargar solo lo que se debería reproducir a continuación, por lo que tampoco se emulaba fielmente esa parte.

Finalmente, se optó por un flujo de video en vivo, lo que resolvió varios de los problemas anteriores. Se eligió el canal DW TV, que transmite de forma continua las 24 horas y está disponible en formato HLS (HTTP Live Streaming) [90] con múltiples resoluciones, entre ellas 720p, lo que permitió seleccionar una calidad media para los experimentos. Al no estar alojado en YouTube, evita los inconvenientes de compatibilidad que presentan VLC y `yt-dlp`, y permite mantener un tráfico continuo y realista, siempre circulando a través de `uesimtun0`.

El flujo de video se descarga con `ffmpeg`, usando la opción `-re` para que la lectura del flujo se haga en tiempo real (respetando el ritmo de reproducción) y la opción `-map` para fijar la resolución en 720p. La ejecución completa del perfil se realiza mediante un script en bucle que alterna períodos de espera y de descarga, con duraciones aleatorias dentro de márgenes definidos. El comportamiento general de este tipo de script se ilustra en el siguiente pseudocódigo, que se utiliza tanto para este perfil como para el de audio:

Inicializar variables:

contador $\leftarrow 0$

while el sistema esté activo **do**

$t_{\text{descarga}} \leftarrow \text{rand}(30, 60)$

según el valor de *contador*:

caso 0–14:

$t_{\text{espera}} \leftarrow \text{rand}(30, 60)$

contador $\leftarrow \text{contador} + 1$

```

caso 15–19:
     $t_{\text{espera}} \leftarrow \text{rand}(1, 20)$ 
     $\text{contador} \leftarrow \text{contador} + 1$ 
caso 20:
     $t_{\text{espera}} \leftarrow \text{rand}(30, 60)$ 
     $\text{contador} \leftarrow 0$ 

    sleep( $t_{\text{espera}}$ )
    reproducción( $t_{\text{descarga}}$ )
end while

```

En este esquema, las duraciones de descarga y espera varían aleatoriamente dentro de rangos predefinidos, de modo que el tráfico no presenta un patrón fijo y se asemeja al comportamiento irregular de un usuario real. Cada iteración del bucle equivale a un ciclo de reproducción continua, con intervalos de descanso que simulan la pausa o el cambio de contenido.

6.2.2. Tráfico de audio

El perfil de tráfico de audio se basa en el mismo modelo de ejecución mostrado en el algoritmo de video, manteniendo la lógica de alternancia entre períodos de espera y descarga con duraciones aleatorias. En este caso, se sustituye la fuente de video por un flujo de audio proveniente de un servidor público **Icecast**, disponible en forma continua las 24 horas.

El objetivo de este perfil es representar el comportamiento de un servicio de transmisión de voz o música en tiempo real, como una llamada o una radio en línea. Para ello, se seleccionó la versión del flujo codificada a 64 kbps, que ofrece una calidad equivalente a la de una comunicación de voz comprimida con códecs como Opus o AAC. Este bitrate permite generar una carga constante y representativa, sin saturar el ancho de banda del *slice*, y reflejando de manera razonable la ocupación típica de un flujo de audio sostenido.

El tráfico resultante se mantiene estable y de baja intensidad en comparación con el de video, pero suficiente para observar la respuesta del *slice* y de los mecanismos de control de ancho de banda ante un flujo continuo. El uso de la misma estructura de script facilita, además, la comparación entre perfiles, dado que las diferencias en comportamiento provienen únicamente del tipo y volumen del tráfico, no de la lógica de ejecución.

6.2.3. Tráfico de navegación web

El perfil de navegación web busca representar el comportamiento de un usuario que visita diferentes sitios, con pausas intermedias que simulan los tiempos de lectura o exploración antes de pasar a otra página. En este caso, se diseñó un script en **bash** que descarga contenido web a través de la interfaz **uesimtun0**, con esperas aleatorias y selección variable de destinos.

Capítulo 6. Escalamiento de la solución

Durante las primeras pruebas se intentó implementar este tráfico con `wget`, pero no fue posible forzar su asociación a la interfaz `uesimtun0`, lo que llevó a reemplazarlo por `curl`. Esta herramienta permitió controlar explícitamente la interfaz de salida y, además, ofreció mayor flexibilidad para manejar tiempos de espera, extracción de recursos y control de errores.

El script está estructurado en grupos de páginas. Cada grupo corresponde a un conjunto de sitios relacionados (por ejemplo, artículos de Wikipedia, secciones de un portal de noticias o páginas de la Facultad de Ingeniería). Antes de iniciar un grupo, el script espera un tiempo aleatorio de entre 30 y 60 segundos, simulando el momento de inactividad previo a comenzar una sesión de navegación. Luego, dentro de cada grupo, las páginas se visitan en orden aleatorio, descargando su contenido principal y algunos recursos secundarios como hojas de estilo, imágenes o pequeños scripts, pero evitando archivos multimedia grandes como videos o audios.

Entre cada página visitada se introduce otra espera aleatoria, de entre 10 y 20 segundos, que representa el tiempo de lectura o de interacción del usuario antes de pasar a la siguiente página del mismo grupo. Este patrón de alternancia entre descargas y pausas genera un tráfico intermitente y variado, más cercano al comportamiento real del uso web cotidiano, pero manteniendo controlados los márgenes de carga y frecuencia de conexión.

El resultado es un perfil de tráfico irregular, con ráfagas breves de actividad seguidas de períodos cortos de inactividad. Este tipo de tráfico contrasta con los perfiles de video y audio, y resulta útil para analizar cómo responde el *slice* ante flujos de carácter no continuo y con patrones menos predecibles.

6.3. Adaptación dinámica de slices

6.3.1. Criterios por slice

Para la adaptación dinámica de los recursos se definieron tres *slices* con comportamientos y objetivos distintos: *slice* prioritario, *slice* garantizado y *slice Best Effort*. Cada uno agrupa un tipo de tráfico diferente, con políticas de asignación que buscan reflejar el comportamiento esperado de una red heterogénea y, al mismo tiempo, permitir que la infraestructura responda a la carga variable de cada servicio.

Slice prioritario: video

El *slice* prioritario está orientado a los flujos de video. Este tráfico representa el consumo más exigente en términos de ancho de banda y, en general, es el que más impacto tiene sobre la capacidad de la red cuando la demanda crece. Por esa razón, se decidió darle el nivel más alto de prioridad en el proceso de asignación de recursos.

La política de este *slice* es simple: si los recursos del sistema lo permiten, siempre se asigna la cantidad necesaria para satisfacer la demanda estimada por el modelo predictivo. No se establecen límites superiores fijos, sino que el ancho de

6.3. Adaptación dinámica de slices

banda se ajusta dinámicamente según las predicciones del modelo LSTM asociado. En la práctica, esto implica que cuando hay disponibilidad de capacidad, este *slice* puede absorber la mayor parte del ancho de banda total.

Esta decisión busca reproducir el comportamiento de servicios de video bajo demanda, en los que el retardo o la degradación de la calidad son perceptibles y afectan directamente la experiencia del usuario. En consecuencia, este *slice* recibe siempre prioridad sobre los demás, tanto en la etapa de predicción como en la asignación efectiva de recursos.

Slice garantizado: audio

El *slice* garantizado agrupa el tráfico de audio, que si bien tiene un requerimiento de ancho de banda mucho menor, no puede interrumpirse ni degradarse por completo ante congestión. Por eso, aunque su prioridad de asignación está por debajo del *slice* de video, se le reserva un mínimo de 0,5 Mbps que nunca puede ser reducido.

Este umbral garantiza que, aun en condiciones de alta ocupación, las comunicaciones de voz sigan siendo posibles. El modelo LSTM que gobierna su comportamiento utiliza las mismas estrategias de predicción que el *slice* prioritario, pero sus resultados se interpretan de forma distinta: el ancho de banda asignado puede aumentar si la predicción lo sugiere, pero nunca disminuir por debajo del límite mínimo.

En este sentido, el *slice* garantizado cumple un rol de equilibrio entre flexibilidad y confiabilidad. No busca competir con el *slice* de video por capacidad, pero asegura que exista un nivel de servicio básico constante, independiente del estado del resto de la red. Esta política refleja la lógica de priorizar la continuidad de la comunicación frente a la calidad o la velocidad.

Slice Best Effort: navegación web

El *slice Best Effort* concentra el tráfico de navegación web y, potencialmente, otros servicios sin requisitos estrictos de retardo o ancho de banda. Su asignación se realiza únicamente con los recursos remanentes, una vez que los otros *slices* han recibido lo que necesitan.

El objetivo de este *slice* no es maximizar la velocidad de descarga, sino permitir el uso de la capacidad sobrante. En términos prácticos, su ancho de banda puede variar significativamente a lo largo del tiempo, reflejando la naturaleza elástica del tráfico web. Cuando la red está desocupada, este *slice* puede alcanzar bitrates altos y, cuando hay congestión, puede reducirse a cero.

Esta política se alinea con el principio del tráfico *Best Effort*: aprovechar lo disponible sin exigir reservas ni garantías. Además, al dejar su asignación completamente subordinada a las necesidades de los otros dos *slices*, se logra mantener una distribución de recursos coherente con la importancia relativa de cada tipo de servicio.

6.3.2. Generación de datasets

Para poder aplicar los modelos de predicción de tráfico que se describen más adelante, fue necesario construir datasets específicos para cada tipo de flujo involucrado. La idea de fondo es sencilla: si cada *slice* tiene un comportamiento característico, entonces es fundamental que el modelo aprenda a predecir ese comportamiento a partir de mediciones tomadas directamente sobre el sistema real. Por esa razón, todos los datasets utilizados en este trabajo provienen de capturas hechas en el *testbed* final, con la arquitectura escalada a tres *slices*, tres gNBs y los veinte UEs virtuales ya descritos en las secciones anteriores, por más que no todos ellos participen de cada adquisición.

Cada dataset se genera midiendo el *throughput* de un flujo de tráfico bien definido, siempre con tres UEs asociados al mismo gNB y ejecutando el tipo de tráfico que corresponde a ese *slice*. Esto permite capturar un comportamiento representativo de carga media dentro del escenario completo, sin caer en un caso trivial de un único UE, lo que terminaría dificultando el aprendizaje del modelo. Además, mantener la estructura de tres UEs por flujo ayuda a que las comparaciones entre los distintos perfiles sean más claras.

El procedimiento general para obtener los datos es el mismo en los dos casos:

1. Se levanta el tráfico real ejecutado por los UEs de uno de los gNBs.
2. Se identifica el flujo correspondiente usando las reglas de clasificación por puerto e IP ya mencionadas en la sección anterior.
3. Se consulta el valor instantáneo de *throughput* a través de la API expuesta por ODL.
4. Se registra ese valor en un archivo CSV junto con su marca de tiempo.

El muestreo se realiza cada un segundo, lo cual permite capturar las variaciones rápidas que tienen lugar en los perfiles de tipo video y navegación web. No se utiliza ninguna estructura de *timestamp* particular para el CSV, ya que en este contexto lo único relevante es el espaciado temporal constante entre mediciones y no la forma exacta del formato.

El archivo resultante contiene dos columnas: *throughput*, expresado en Mbps, y tiempo, cuyo formato no se estandariza porque no es necesario para los procedimientos posteriores.

Al cabo de varias horas de medición continua, cada dataset termina acumulando una cantidad considerable de muestras, por arriba de las 35000 en ambos casos. A continuación se describe con mayor detalle cómo se utiliza cada dataset según el tipo de tráfico de los *slices* definidos previamente.

Dataset de tráfico de video

El tráfico de video es el que presenta mayor variabilidad y también el que trabaja con los anchos de banda más altos. Para generar su dataset, se utilizaron tres UEs ejecutando el perfil de video tal como se definió en la sección anterior: un

6.3. Adaptación dinámica de slices

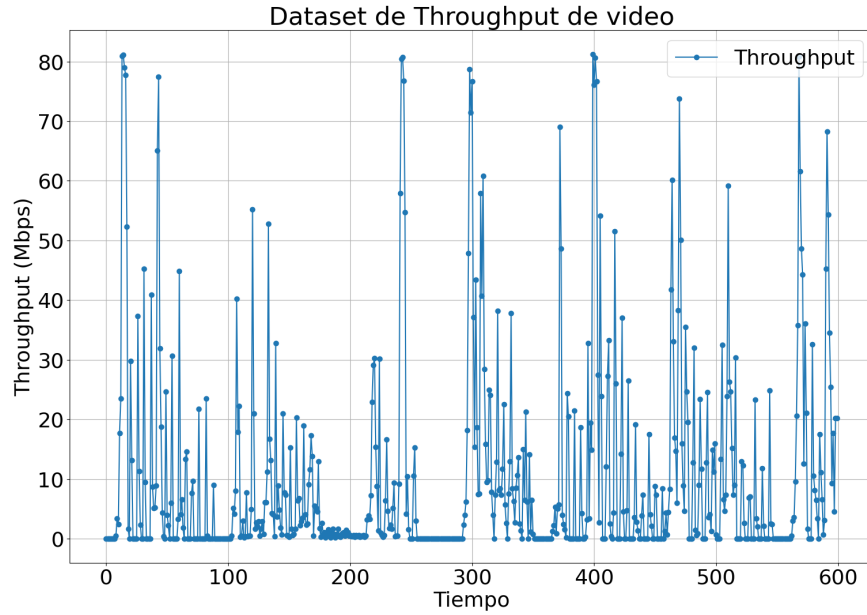


Figura 6.4: Fragmento del dataset del slice de video, muestreado cada un segundo.

stream en vivo de DW, en 720p, obtenido mediante `ffmpeg` con la opción `-re` para garantizar que la velocidad de descarga se mantenga consistente con el tiempo real del contenido.

Una vez que los tres UEs están conectados al gNB asignado y generando tráfico, se identifica el flujo correspondiente en OVS, usando las reglas de clasificación por puerto de entrada y dirección IP de destino ya explicadas en la arquitectura, y se comienza la captura del *throughput* sobre ese único flujo. Es importante notar que el dataset contiene el valor total del flujo que el UPF entrega al gNB, que naturalmente incorpora los picos y valles generados por los tres dispositivos en tiempos aleatorios.

El resultado es una serie temporal con una dinámica marcada, visible en la Figura 6.4, con períodos de consumo alto cuando los UEs descargan segmentos nuevos del *stream* y pequeñas caídas asociadas a pausas propias del protocolo HLS, además de las menos predecibles variaciones cuando alguno de los UEs entra en una fase de espera del script.

Este comportamiento es importante en el entrenamiento del LSTM, ya que obliga al modelo a captar tanto la tendencia dominante del flujo como sus oscilaciones cortas.

Dataset de tráfico de audio

El perfil de audio se generó también con tres UEs simultáneos, apuntando al *stream* Icecast elegido (disponible de forma continua y en 64 kbps) y utilizan-

Capítulo 6. Escalamiento de la solución

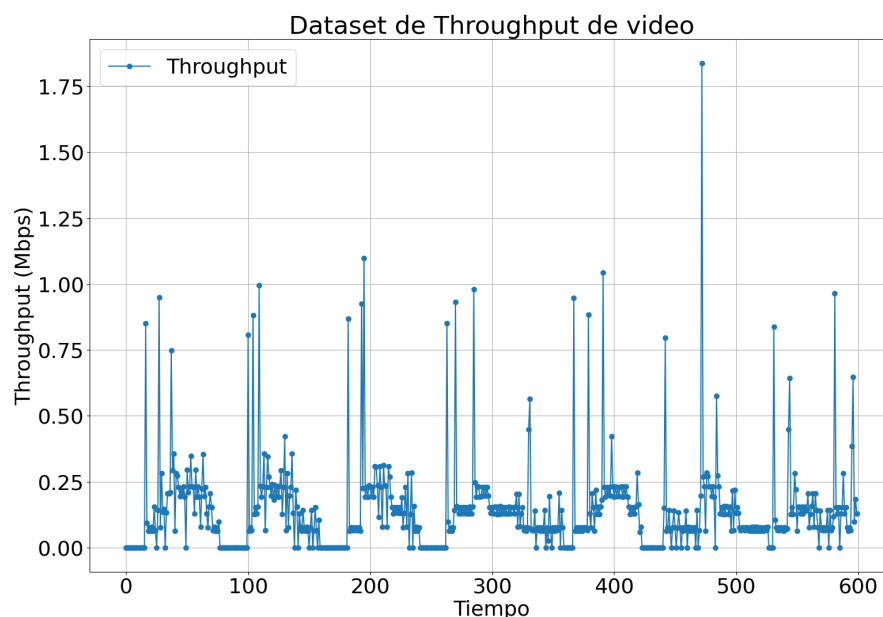


Figura 6.5: Fragmento del dataset del slice de audio, muestreado cada un segundo.

do exactamente la misma lógica de ejecución descrita para el video, pero con el volumen de datos y el comportamiento característicos de un flujo de voz.

La construcción del dataset sigue el mismo procedimiento: identificar el flujo correspondiente, consultar su *throughput* mediante API y registrar una muestra por segundo. La diferencia más relevante está en el tipo de variabilidad que se captura: mientras que en video el flujo puede tener saltos más bruscos, en audio los valores tienden a ser más estables con algunos picos aislados (Figura 6.5). Este dataset es el que se usa para entrenar el modelo asociado al *slice* con mínimo garantizado.

Una vez capturados los dos conjuntos de datos, cada uno se almacena en un archivo CSV independiente. No se realiza ninguna limpieza ni filtrado adicional más allá de la verificación obvia de que no haya filas incompletas, ya que las mediciones provienen directamente del flujo que ofrece ODL con el período de *polling* reducido a un segundo.

La decisión de mantener los datasets tal cual surgieron del sistema responde a dos motivos: evita introducir sesgos que puedan engañar al modelo, y preserva el comportamiento real del sistema, con toda su variabilidad. Siendo un escenario acotado por la naturaleza del *testbed*, el segundo punto es especialmente relevante. Los datasets utilizados, así como los scripts para relevarlos, quedan disponibles en el repositorio del proyecto.

A continuación se detalla cómo se utilizan estos datasets para el entrenamiento, validación y posterior despliegue de los modelos LSTM que definen el mecanismo de asignación dinámica de recursos en cada *slice*.

6.3.3. Entrenamiento y evaluación de modelos LSTM

El objetivo de esta sección es describir el proceso de entrenamiento de los modelos LSTM utilizados para predecir el comportamiento del tráfico de los *slices* de video y de audio. La lógica del entrenamiento es similar en ambos casos, aunque el análisis se desarrolla en mayor profundidad para el modelo de video, dado que es el más exigente en términos de variabilidad y de volumen sobre la asignación dinámica de recursos.

El entrenamiento del modelo comienza definiendo la ventana temporal utilizada como insumo del LSTM. En este caso, se usa una ventana fija de 120 valores consecutivos del *throughput* para predecir el valor siguiente. El número surge directamente del comportamiento de los scripts de tráfico: tanto los tiempos de espera como los de descarga del contenido varían entre 30 y 60 segundos en la mayoría de los ciclos, por lo que una ventana de 120 segundos cubre un ciclo completo en el peor caso. Esto permite que el modelo tenga siempre a la vista una secuencia completa del patrón típico de subida (durante la descarga) y bajada (durante la espera) del *throughput*, evitando que la predicción se base en segmentos demasiado cortos.

Como no se aplica ningún tipo de normalización sobre los datos, los valores de *throughput* se utilizan tal cual fueron capturados. Esto fue para mantener la relación entre los valores reales del sistema y las predicciones del modelo, y evitar introducir distorsiones que afecten la interpretación de las salidas del modelo. El dataset completo (de más de 35,000 muestras en ambos casos) se divide en dos subconjuntos: 80 % para entrenamiento y 20 % para validación.

El modelo usado en todos los experimentos sigue la misma estructura que en el capítulo anterior. Se usa un LSTM implementado en Keras [91], seguido de una capa densa que produce un único valor de salida (el *throughput* estimado para el siguiente instante). El optimizador utilizado es Adam [92], sin ajustes especiales. Más allá del uso de *ReLU* [93] como activación interna, no se incorporan capas adicionales. La idea es mantener una estructura lo más simple posible, de manera de facilitar la interpretación de los resultados.

Una vez entrenado el modelo base utilizando la función de pérdida estándar MSE (error cuadrático medio), el siguiente paso es evaluar si esta configuración inicial es apropiada para el caso del *slice* de video. Esta es una instancia particularmente importante porque, al ser el *slice* prioritario, cualquier error que resulte en una asignación insuficiente de recursos puede tener un impacto directo en la experiencia del usuario. La primera evaluación mostró un resultado que, a primera vista, parece razonable: la Figura 6.6 muestra cómo se ajusta la predicción al conjunto de prueba, con un valor de RMSE de 13,01, relativamente bajo para una serie tan variable que se maneja en un rango de 0 a unos 100 Mbps.

También puede verse en la Figura 6.7 que el *Scatter* de esta predicción contra los valores reales se aproxima a lo esperado en este caso: los puntos se agrupan alrededor de la línea que iguala los valores.

Sin embargo, al observar el porcentaje de subpredicción (es decir, la proporción de veces en las que el modelo predice un valor menor al real) se ve que la métrica de error esconde otro problema: tener 36,5 % de los valores de *throughput* asignados

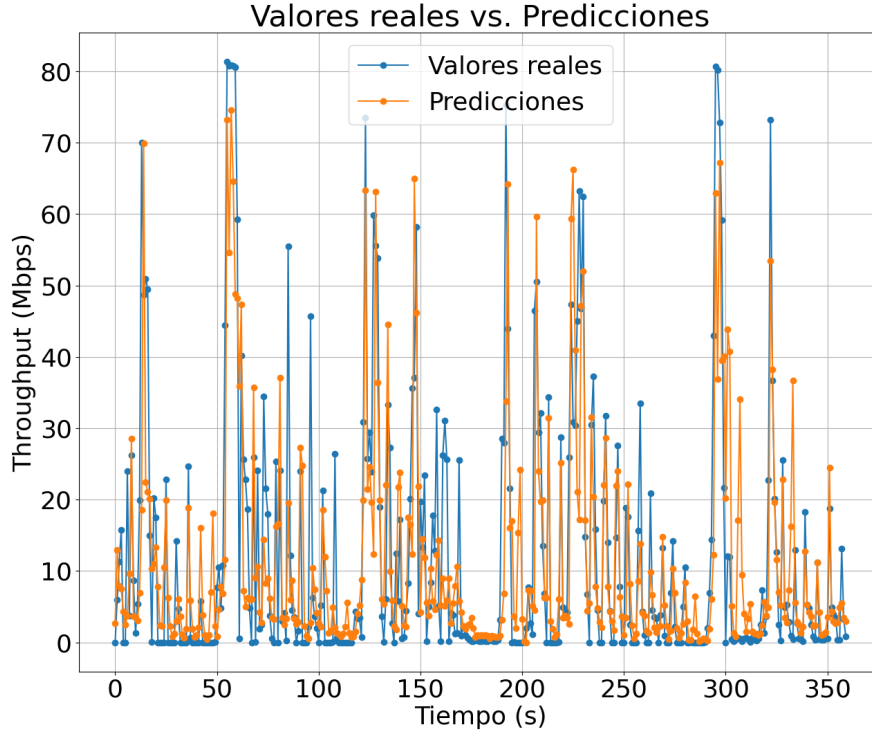


Figura 6.6: Comparación de las predicciones del modelo inicial contra los datos de validación en función del tiempo.

por debajo del ancho de banda requerido resulta demasiado alto para ser aceptable en un *slice* que, por definición, debería adelantarse a los picos de tráfico y asignar los recursos suficientes.

Esta observación agrega a la evaluación el criterio de priorizar la sobrepredicción permite explorar alternativas dentro del modelo LSTM: dado que es preferente que, en caso de haber errores, se estime por encima del valor real que por debajo, es posible ajustar la función con la que se penalizan los errores durante el entrenamiento. El MSE penaliza simétricamente los errores positivos y negativos, lo que, en principio, no es ideal cuando el problema es desalentar la subpredicción. Por lo tanto, la pregunta natural es: ¿qué función de pérdida es mejor para penalizar más un tipo de error que otro?

Para abordar esto, se planteó inicialmente una función de pérdida asimétrica con dos regiones diferenciadas. Siendo $e = y_{\text{pred}} - y_{\text{real}}$ el error instantáneo, se define una función de pérdida (o *Loss*) por tramos:

$$L(e) = \begin{cases} \alpha e^2, & \text{si } e < 0 \text{ (subpredicción)} \\ \beta e^2, & \text{si } e \geq 0 \text{ (sobrepredicción)} \end{cases}$$

donde α y β son parámetros ajustables. Cuando $\alpha > \beta$, la función penaliza más los errores por subpredicción que los de sobrepredicción. Este tipo de pérdida mantiene

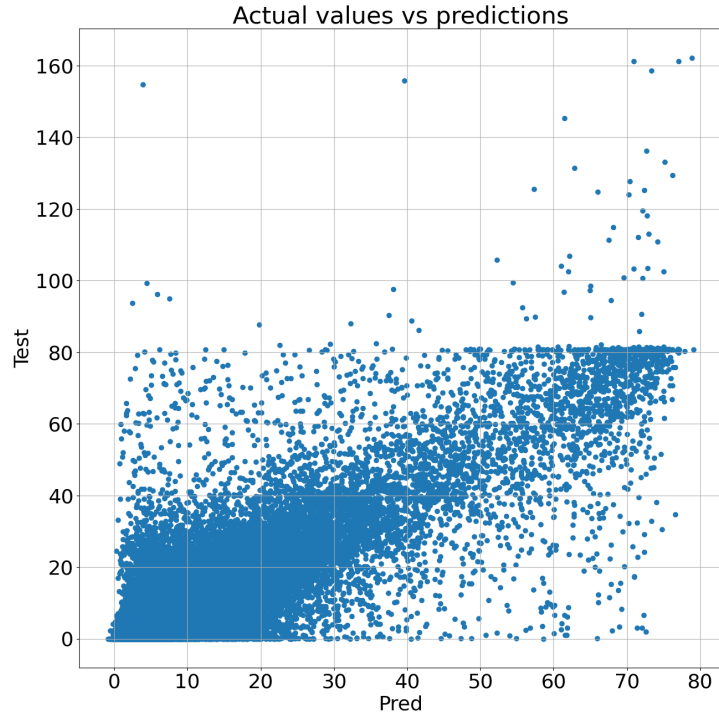


Figura 6.7: Scatter de las predicciones del modelo inicial contra los datos de validación.

la estructura del MSE, pero introduce un peso diferente según el signo del error.

A partir de esta formulación, se probaron algunas combinaciones de valores para α y β , evaluadas siguiendo el mismo procedimiento: se entrenó el modelo desde cero, se aplicó al conjunto de validación y se recolectaron las mismas métricas, además de las gráficas correspondientes. La Figura 6.8 muestra el resultado de usar $\alpha = 3$ y $\beta = 1$.

En ese caso, el RMSE fue un poco peor con 13,7, pero el porcentaje de subpredicción bajó al 25,8%, lo que representa una mejora con respecto al intento inicial y da cierta validez al método de ajuste de función de pérdida. De todas formas, se requiere una mayor penalización de la subpredicción para minimizar aun más su porcentaje de aparición, aunque sea al costo de una menor precisión en la asignación.

Aquí se introduce una métrica más a considerar, que lleva una correlación con el RMSE pero que da un mejor reflejo de lo que interesa evaluar. En el contexto de la asignación dinámica de recursos para optimizar el uso de estos sin comprometer el nivel de servicio, lo que interesa minimizar no es el RMSE (que indica con qué precisión se ajusta la predicción, independientemente de si el error se hace por encima o por debajo del valor real), sino el bitrate asignado. Es decir que, dado un cierto nivel de cobertura garantizado, será deseable usar el modelo que menos

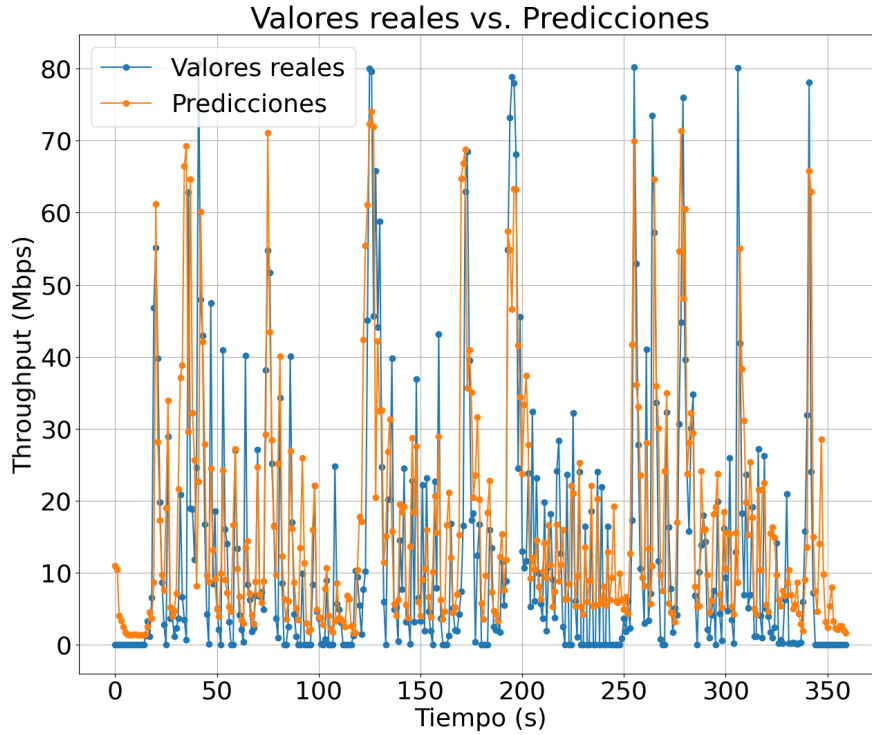


Figura 6.8: Comparación de las predicciones del modelo con función de pérdida cuadrática asimétrica contra los datos de validación en función del tiempo.

bitrate le dé al *slice* prioritario, para que este pueda ser aprovechado en otros servicios.

En este sentido, la primera función asimétrica para penalizar la subpredicción asigna un bitrate promedio de 14,9 Mbps frente a los 11,2 Mbps asignados por el modelo original con pérdida MSE. Es esperable que, en la medida en la que se busque obtener menos predicciones por debajo de los valores reales, este promedio aumente.

La siguiente iteración ensayada para seguir reduciendo la subpredicción es una función de pérdida completamente lineal a ambos lados, pero nuevamente con pendientes diferentes según el signo del error. En este caso, la idea es simplificar la forma de la función de pérdida para que el modelo responda más directamente a los desvíos. La definición general es:

$$L(e) = \begin{cases} \alpha |e|, & \text{si } e < 0 \\ \beta |e|, & \text{si } e \geq 0 \end{cases}$$

donde se vuelve a utilizar el criterio $\alpha > \beta$ para penalizar más la subpredicción que la sobrepredicción. Para este experimento se probaron los valores $\alpha = 5$ y $\beta = 1$.

El resultado numérico de esta variante es que el porcentaje de subpredicción baja de forma notable, alcanzando 18,5 %. Sin embargo, el RMSE sube a 16,18,

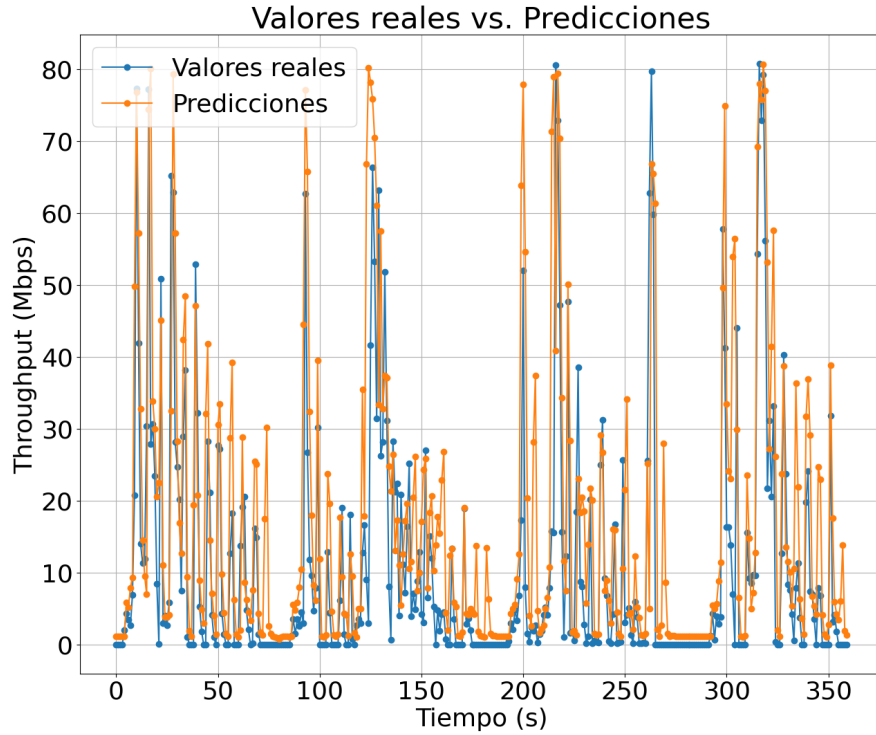


Figura 6.9: Comparación de las predicciones del modelo con función de pérdida lineal asimétrica contra los datos de validación en función del tiempo.

lo que se ve reflejado claramente en las gráfica de la Figura 6.9. En particular, la serie temporal muestra que el modelo pierde la capacidad de seguir los picos altos. Esto es esperable cuando se usa una función de pérdida lineal, ya que el modelo no recibe un castigo adicional por errores grandes, de modo que tiende a suavizar más de lo deseable. Para un flujo tan cambiante como el de video, este efecto resulta contraproducente y termina afectando la precisión del ajuste.

Se puede destacar de esta iteración que el bitrate promedio es de 12,0 Mbps, que representa una reducción frente a la anterior aún mejorando la subpredicción. De todas formas, esta última sigue siendo demasiado alta para la aplicación.

En la siguiente alternativa se explora una combinación de las dos funciones: mantener una penalización cuadrática para los errores negativos, de modo de castigar fuertemente cualquier estimación por debajo del valor real, mientras que para los errores positivos se utiliza una función lineal, que evita penalizar en exceso las veces en las que el modelo asigna un valor superior al necesario. La función toma entonces la forma:

$$L(e) = \begin{cases} \alpha e^2, & \text{si } e < 0 \\ \beta |e|, & \text{si } e \geq 0 \end{cases}$$

La primera variante de este tipo que se probó utiliza los coeficientes $\alpha = 3$ y $\beta =$

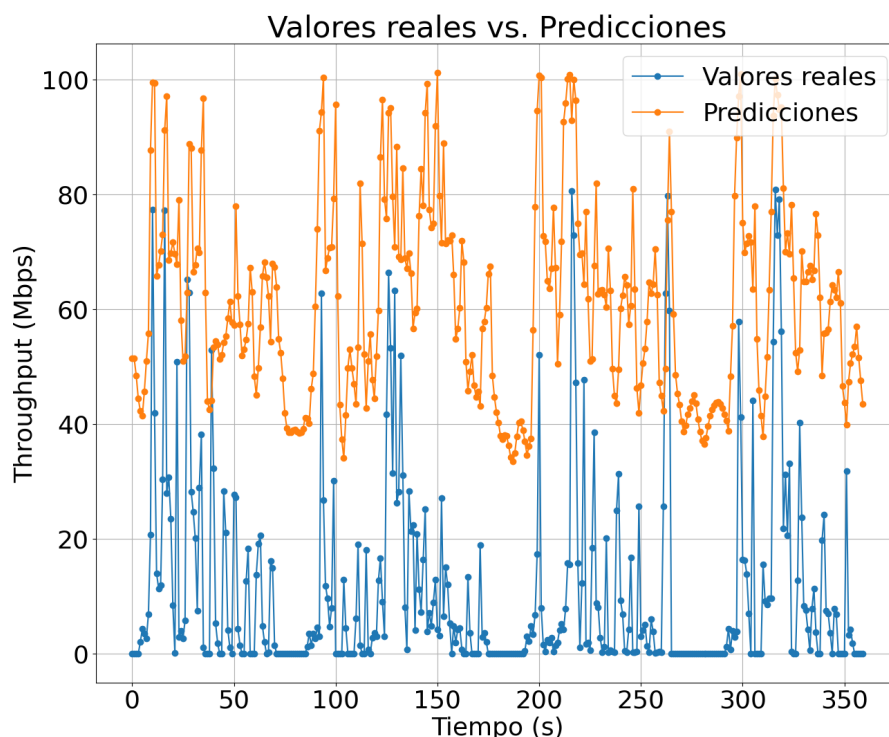


Figura 6.10: Comparación de las predicciones del modelo con función de pérdida asimétrica (cuadrática con factor 3, lineal con factor 1) contra los datos de validación en función del tiempo.

1. En este caso, el porcentaje de subpredicción cae prácticamente a cero (0,7%), lo cual demuestra que la función logra exactamente lo que se pretende: desalentar casi por completo que el modelo quede por debajo del valor real. Sin embargo, esto tiene un costo evidente: el RMSE crece hasta 53,77, y la serie temporal confirma que la predicción tiende a irse demasiado arriba en varios tramos (ver Figura 6.10). El modelo garantiza sobrepredicción, pero con un excedente de ancho de banda que no se aprovecha en el *slice*, promediando 62,6 Mbps asignados.

El comportamiento es coherente con que una penalización tan fuerte del lado negativo empuja al modelo a una solución en la que asegura que el error se haga siempre por encima del valor real, pero sin distinguir adecuadamente entre los picos auténticos del flujo y las variaciones menores. Esto no deja de ser mejor que simplemente asignar de forma fija el máximo disponible al *slice*, pero no es lo más adecuado para una asignación que busca cierto equilibrio entre disponibilidad y eficiencia.

Es interesante ver cómo esto también se visualiza en el gráfico de *Scatter* de la Figura 6.11. Los puntos que antes seguían la diagonal que equipara los valores de predicción con los de validación, ahora tienen un fuerte corrimiento hacia la

6.3. Adaptación dinámica de slices

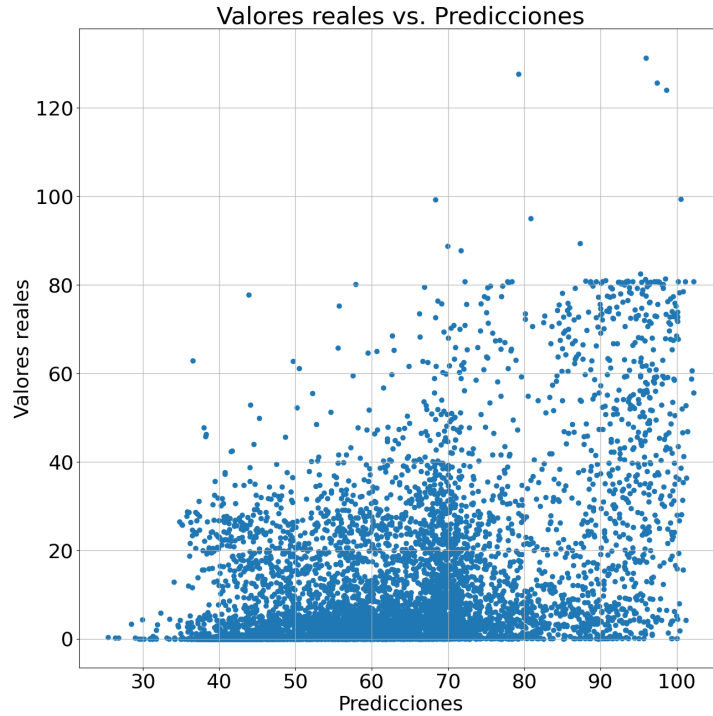


Figura 6.11: Scatter de las predicciones del modelo con función de pérdida asimétrica (cuadrática con factor 3, lineal con factor 1) contra los datos de validación.

derecha al probar el caso opuesto.

Para mejorar este balance, se hace una prueba adicional reduciendo la penalización cuadrática del lado negativo a $\alpha = 2$, manteniendo $\beta = 1$ en el lado positivo. Esta pequeña modificación busca permitir algo más de flexibilidad al modelo, sin dejar de privilegiar la sobrepredicción.

El resultado mostrado en la Figura 6.12: la subpredicción sube levemente a 4,6 %, que sigue siendo un valor muy bajo, y el RMSE baja de forma significativa respecto del caso anterior, ubicándose en 33,05. Si bien este error sigue siendo mayor que el obtenido en los casos puramente cuadráticos o puramente lineales, llega a un compromiso más equilibrado entre ambos criterios: evita la subpredicción en la inmensa mayoría de los casos, lo cual se estableció como prioritario, sin llegar a una sobreasignación tan pronunciada como en la variante anterior, en la forma de un bitrate promedio de 35,2 Mbps.

Igualmente, el modelo sigue siendo conservador en cuanto a la priorización de no subasignar por sobre el ajustar lo más posible la predicción, por lo que el análisis pasa a enfocarse en comparar estas variantes desde la perspectiva operativa de la asignación dinámica. Por un lado, una sobrepredicción muy marcada se acerca a garantizar que el *slice* de video nunca se quede sin ancho de banda, pero a costa

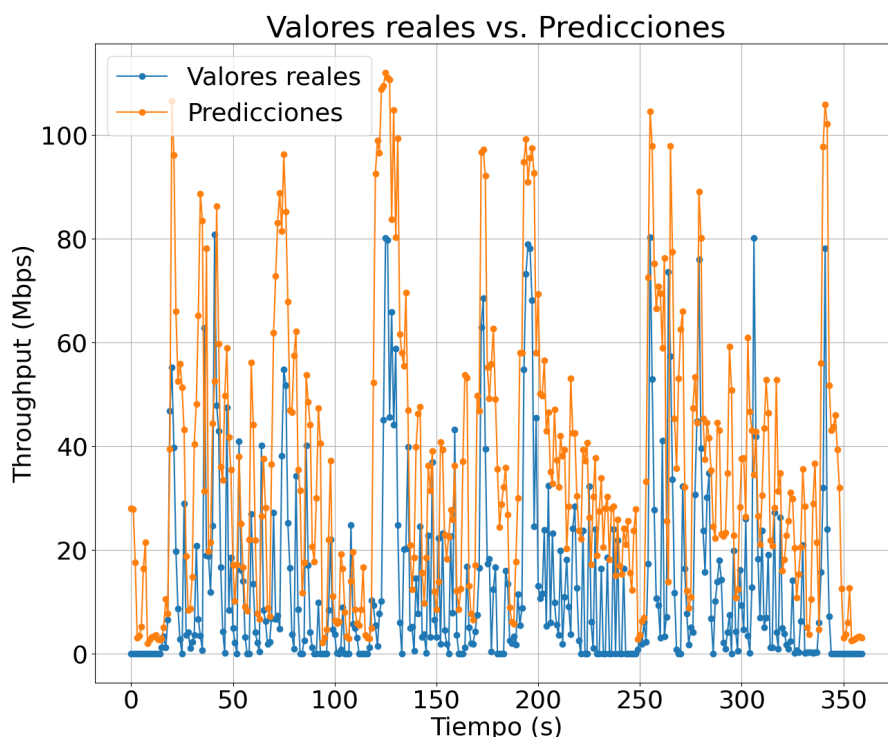


Figura 6.12: Comparación de las predicciones del modelo con función de pérdida asimétrica (cuadrática con factor 2, lineal con factor 1) contra los datos de validación en función del tiempo.

de asignar más de lo necesario, lo cual afecta el objetivo inicial de aprovechar recursos no utilizados. Por otro lado, un bitrate promedio bajo con una cobertura alta significa que la asignación se acerca más a lo realmente requerido, con el riesgo de que, con frecuencia, los picos intermedios o grandes se subestimen, alterando el flujo de tráfico (definido como prioritario) en la medida en que un reproductor de video no recibe todo lo que necesita para reproducirlo sin interrupciones.

La función mixta con $\alpha = 2$ y $\beta = 1$ representa el punto medio más razonable entre estas dos prioridades. Un 4,6 % de subpredicción es lo suficientemente bajo como para considerarse seguro en un sistema que prefiere error hacia arriba, y un bitrate promedio asignado de 35,2 Mbps resulta aceptable. Este valor refleja el compromiso asumido: garantizar que el *slice* prioritario tenga los recursos necesarios, incluso si esto implica aceptar una predicción algo más gruesa en términos absolutos.

Por esta razón, la variante con función cuadrática para errores negativos y función lineal para errores positivos, con coeficientes $\alpha = 2$ y $\beta = 1$, es seleccionada como la mejor entre los ensayos realizados en esta etapa.

Una vez finalizada la experimentación con el modelo de video y seleccionada la

6.3. Adaptación dinámica de slices

función de pérdida más adecuada para este caso, se repite un procedimiento similar para el dataset de audio, aunque con un alcance menor. La razón es sencilla: como se mencionó en la definición de los *slices*, el modelo de audio tiene un impacto diferente en el proceso de asignación de recursos: se le asigna un mínimo garantizado y su variabilidad es más acotada. En consecuencia, no resulta necesario repetir el análisis completo ni realizar comparaciones detalladas entre múltiples funciones de pérdida. Basta con tomar la que mejor desempeño mostró en el caso de video y verificar, por medio de un entrenamiento equivalente, que el modelo también logra capturar adecuadamente la dinámica del tráfico de audio.

Los mismos métodos se utilizan para validar este modelo: los resultados obtenidos de subpredicción y RMSE son 12 % y 0,169, respectivamente.

Finalmente, vale aclarar que ambos modelos, una vez entrenados, se almacenan y se utilizan posteriormente dentro del sistema de asignación dinámica de recursos, cuyo funcionamiento se describe en la sección siguiente. El rol del modelo se limita a producir un valor estimado del *throughput*. El criterio de cómo esa estimación se traduce en una decisión de asignación pertenece por completo al mecanismo definido para cada *slice*.

6.3.4. Predicción Conformal

Con la definición final del criterio por el cual se quiere acercar la predicción lo más posible al valor real con cierta garantía de que no va a estar por debajo de este, el último paso de esta metodología es formalizarlo de forma de convertir esa predicción en un modelo probabilístico.

La reducción de sesgo se busca ahora mediante la denominada predicción conformal [94], que permite obtener un intervalo tal que, con probabilidad $1 - \alpha$, el valor futuro de *throughput* se encuentre dentro de dicho intervalo. Es importante notar que esta metodología puede utilizarse con cualquier predictor, por lo que se mantiene válida si se cambia la técnica de modelado.

En este caso se define $\alpha = 0,05$ para obtener una garantía de cobertura del 95 % y así dimensionar el bitrate sin subestimar la demanda. Se utiliza la variante *split* conformal, separando los datos en tres partes: una primera para entrenar el modelo LSTM, la segunda para calibrar el ancho del intervalo, y la tercera para validar con el mismo tamaño que se hicieron las pruebas de las partes anteriores, de manera de que los resultados sean comparables.

Tras el entrenamiento del estimador $\hat{f}$ de la primera etapa, se evalúan los valores reales y_i y sus predicciones $\hat{f}(x_i)$ sobre el conjunto de calibración D_2 y se calculan los residuos absolutos

$$r_i = |y_i - \hat{f}(x_i)| \quad i \in D_2$$

que se ordenan de menor a mayor. Sea $n_2 = |D_2|$ el tamaño de la calibración y $k = \lceil (1 - \alpha)(n_2 + 1) \rceil$. Entonces q se define como el k -ésimo residual más pequeño entre los r_i . Intuitivamente, q es un umbral tal que solo una fracción controlada de los errores históricos queda por encima de él. La construcción mediante el índice k garantiza, bajo la hipótesis de intercambiabilidad (o “*Exchangeability*”, referido

Capítulo 6. Escalamiento de la solución

a que la distribución de un vector es la misma, independientemente del orden elegido) entre los puntos de calibración y un futuro punto, una cobertura mínima de $1 - \alpha$ en una muestra finita.

El intervalo conformal para una nueva entrada x queda entonces

$$C(x) = [\hat{f}(x) - q, \hat{f}(x) + q],$$

y, en este caso, para evitar la subestimación del *throughput*, se utiliza el límite superior

$$\hat{y}(x) = \hat{f}(x) + q.$$

Con esto se obtiene una asignación de bitrate que, salvo en hasta un 5 % de los casos, superará el valor real del *throughput*. La decisión deja de depender de ajustes arbitrarios de la función de pérdida y pasa a tener una justificación basada en la distribución de errores del modelo.

Naturalmente, dado que el percentil se computa en base a los residuos definidos, se toma como modelo predictor aquel que arrojó los menores errores de RMSE, que es usando la función de pérdida MSE.

Los resultados se pueden ver en la Figura 6.13. El q calculado es de 29,8 Mbps, agregando ese margen al bitrate asignado (la predicción) y pasando de un índice de subpredicción de 33,53 % a 2,33 %. Así, el bitrate promedio que se asigna es de 41,2 Mbps entre la predicción base y el margen q .

Estos resultados muestran que es posible controlar la subpredicción bajo un umbral $(1 - \alpha)$. Sin embargo, se impone un intervalo constante q para toda la señal, independientemente de la variabilidad temporal del tráfico. Esto genera que en regiones de menor volatilidad exista una mayor sobreasignación de ancho de banda.

La metodología considerada para intentar compensar esto, también presentada en la bibliografía, es el principio de adaptabilidad local del error, que consiste en que los errores del estimador puedan variar de acuerdo a la varianza del tráfico. En escenarios de baja variación, el estimador tiende a cometer errores menores, por lo cual el intervalo podría reducirse y, por el otro lado, en regiones con cambios abruptos o picos de demanda, el intervalo debe ampliarse para mantener el nivel de cobertura.

La librería MAPIE [95] implementa funciones de predicción con intervalos dinámicos para series temporales utilizando regresores o redes neuronales. Su estrategia se basa en estimar un conjunto de residuales en ventana móvil y recalculando un percentil con un esquema del tipo *Ensemble Batch Prediction Intervals* (ENBPI). No obstante, MAPIE entrena el modelo y ajusta la variación de manera conjunta en cada ejecución, lo cual dificulta su implementación en este caso, donde se quiere separar las fases de entrenamiento del modelo LSTM y de operación en el sistema completo.

Por este motivo se adopta una implementación manual inspirada en ENBPI, preservando la calibración inicial del percentil y permitiendo actualización en línea a partir del residuo recién observado.

Sea f_θ el modelo previamente entrenado y (x_t, y_t) la muestra observada en tiempo t . Se define el residuo instantáneo como:

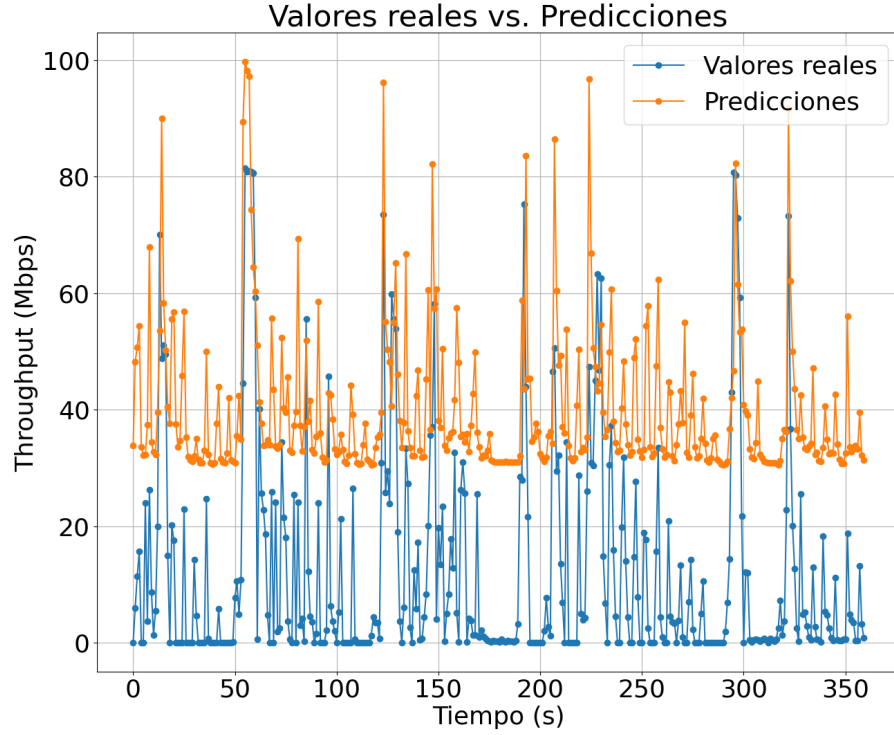


Figura 6.13: Comparación de las predicciones del modelo con función de pérdida mse y percentil conformal fijo contra los datos de validación en función del tiempo.

$$r_t = |f_\theta(x_t) - y_t|$$

Durante la calibración inicial se calcula un conjunto de residuales:

$$\mathcal{R}_0 = \{r_1, r_2, \dots, r_m\}$$

y a partir de ellos se estima el percentil conformal inicial:

$$q_0 = \text{Percentil}_{(1-\alpha)}(\nabla_0)$$

A diferencia del método anterior, ENBPI actualiza el intervalo a medida que llegan nuevas muestras. Esto es, para cada nuevo instante se extiende el conjunto de residuales:

$$\mathcal{R}_t = \mathcal{R}_{t-1} \cup \{r_t\}$$

y se recalcula

$$q_t = \text{Percentil}_{(1-\alpha)}(\mathcal{R}_t)$$

Capítulo 6. Escalamiento de la solución

El intervalo final utilizado para asignación de recursos se expresa como:

$$\hat{y}_t = f_\theta(x_t) + q_t$$

Un punto a notar que ENBPI introduce una estimación explícita de varianza σ_t dependiente del régimen local, mientras que en esta implementación esa variabilidad queda absorbida en q_t , simplificando la actualización. Cada q_t es así un estimador empírico del error histórico acumulado.

De esta manera, el algoritmo implementado queda de la forma:

```
Calibrar modelo con residuales  $\mathcal{R}_0$ 
 $q_0 \leftarrow \text{Quantile}_{1-\alpha}(\mathcal{R}_0)$ 
for  $t = 1 \rightarrow T$  do
   $\hat{y}_t \leftarrow \hat{y}_t + q_{t-1}$ 
  if  $y_t$  observado then
     $r_t \leftarrow |\hat{y}_t - y_t|$ 
    Agregar  $r_t$  a  $\mathcal{R}_{t-1}$  manteniendo ventana fija
     $q_t \leftarrow \text{Quantile}_{1-\alpha}(\mathcal{R}_t)$ 
  else
     $q_t \leftarrow q_{t-1}$ 
  end if
end for
```

Este esquema produce intervalos adaptativos que se reducen en periodos de estabilidad del tráfico y se expanden ante variaciones bruscas. Sin embargo, el resultado obtenido de aplicar este criterio en el sistema implementado no es el esperado: variando entre 24,07 y 39,95 Mbps, el valor de q termina promediando en 31,43 Mbps para los datos relevados, sin presentar una mejora relevante en cobertura que apenas sube de 96,7% a 97%. Más notablemente, el objetivo de disminuir el bitrate asignado (para desperdiciar menos recursos en un escenario similar de cobertura) no se cumple, aumentando el promedio de asignación a 42,7 Mbps.

De todas formas, esta metodología puede ser pertinente para otros tipos de tráfico con mayor varianza relativa, por lo que la herramienta queda disponible junto con el código principal, por más que no vaya a utilizarse para este escenario en particular.

Con el resultado final obtenido, se presenta la Tabla 6.2 mostrando los resultados clave de los experimentos realizados, incluyendo funciones de pérdida simétricas y asimétricas y el agregado de la predicción conformal (con margen fijo y variable).

De estos resultados, se toma el modelo con pérdida simétrica en base a la función MSE y con el margen de predicción conformal fijo para implementar en el sistema final. Si bien no es el que mejor aprovecha los recursos no utilizados por no ser el de menor *throughput* asignado para la cobertura de más de 95%, tiene el respaldo estadístico de dar un intervalo de confianza para cumplir con esta

6.4. Implementación en el sistema completo

Tabla 6.2: Comparación de los resultados obtenidos con distintas funciones de pérdida durante el entrenamiento del modelo LSTM. Se incluyen el RMSE, el bitrate promedio asignado y el índice de subpredicción para funciones simétricas y asimétricas, mostrando el compromiso entre ajuste, aprovechamiento de recursos y garantía de servicio.

Función de pérdida del modelo	RMSE (Mbps)	Asignación promedio (Mbps)	Índice de subpredicción (%)
MSE	13,01	11,2	36,5
$L(e) = \begin{cases} 3e^2, & e < 0 \\ e^2, & e \geq 0 \end{cases}$	13,74	14,9	25,8
$L(e) = \begin{cases} 5 e , & e < 0 \\ e , & e \geq 0 \end{cases}$	16,18	12,0	18,5
$L(e) = \begin{cases} 3e^2, & e < 0 \\ e , & e \geq 0 \end{cases}$	53,77	62,6	0,7
$L(e) = \begin{cases} 2e^2, & e < 0 \\ e , & e \geq 0 \end{cases}$	33,05	35,2	4,6
MSE con CP fijo	32,68	41,2	3,3
MSE con CP variable	34,24	42,7	3,0

condición que es fundamental para dar el servicio prioritario.

6.4. Implementación en el sistema completo

Una vez definidos los modelos y los criterios de asignación para cada *slice*, el paso final es integrarlos en el sistema real y comprobar que el comportamiento del *testbed* coincide con lo esperado. Esta etapa combina dos aspectos: por un lado, verificar que el aislamiento entre *slices* se cumple de forma estricta en los flujos de tráfico; por otro, validar que la asignación dinámica de recursos efectivamente modifica el ancho de banda disponible en cada *slice* según las predicciones del modelo y las reglas definidas.

La primera prueba consiste en confirmar que el tráfico de los *slices* 2 y 3 no aparece en ningún momento en la interfaz del UPF asociado al *slice* 1. Esto se verifica directamente mediante capturas con `sudo tcpdump -i ogstun` mientras se generan flujos reales desde UEs del *slice* de audio y del *slice* de navegación. En todos los casos, las capturas del UPF 1 muestran únicamente el tráfico del *slice* de video, sin ningún paquete proveniente de los otros *slices*. Se repitió la prueba en los demás UPFs y el resultado fue idéntico: no aparece tráfico cruzado, lo que confirma que la clasificación inicial funciona como corresponde.

La segunda comprobación se hace directamente en OVS usando el volcado completo de reglas con:

```
sudo ovs-ofctl dump-flows br0
```

Capítulo 6. Escalamiento de la solución

```
lab@fing-ovs:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows br0
[sudo] password for lab:
cookie=0x0, duration=14066.146s, table=0, n_packets=5110852, n_bytes=7294869112, priority=300,ip,in_port=emp2s1,nw_dst=192.168.233.7 actions=set_queue:110,NORMAL
cookie=0x0, duration=14066.132s, table=0, n_packets=0, n_bytes=0, priority=300,ip,in_port=emp2s1,nw_dst=192.168.233.20 actions=set_queue:210,NORMAL
cookie=0x0, duration=14066.118s, table=0, n_packets=0, n_bytes=0, priority=300,ip,in_port=emp2s1,nw_dst=192.168.233.30 actions=set_queue:310,NORMAL
cookie=0x0, duration=14066.103s, table=0, n_packets=0, n_bytes=0, priority=300,ip,in_port=emp2s2,nw_dst=192.168.233.7 actions=set_queue:120,NORMAL
cookie=0x0, duration=14066.089s, table=0, n_packets=0, n_bytes=0, priority=300,ip,in_port=emp2s2,nw_dst=192.168.233.20 actions=set_queue:220,NORMAL
cookie=0x0, duration=14066.075s, table=0, n_packets=0, n_bytes=0, priority=300,ip,in_port=emp2s2,nw_dst=192.168.233.30 actions=set_queue:320,NORMAL
cookie=0x0, duration=14066.061s, table=0, n_packets=0, n_bytes=0, priority=300,ip,in_port=emp2s5,nw_dst=192.168.233.7 actions=set_queue:130,NORMAL
cookie=0x0, duration=14066.047s, table=0, n_packets=0, n_bytes=0, priority=300,ip,in_port=emp2s5,nw_dst=192.168.233.20 actions=set_queue:230,NORMAL
cookie=0x0, duration=14066.033s, table=0, n_packets=0, n_bytes=0, priority=300,ip,in_port=emp2s5,nw_dst=192.168.233.30 actions=set_queue:330,NORMAL
cookie=0x0, duration=14281.015s, table=0, n_packets=19701277, n_bytes=16603522534, priority=100 actions=NORMAL
lab@fing-ovs:~$
```

Figura 6.14: Impresión de flujos en la interfaz de OVS. Todos mantienen cero bytes de tráfico, a excepción del flujo correspondiente a la prueba y la regla de tráfico genérico.

El análisis de estas reglas permite identificar rápidamente qué flujo corresponde a cada *slice*. Por ejemplo, el flujo asociado a la cola 110 (que corresponde al *slice* 1 del gNB 1) muestra claramente su *in_port* de origen (el UPF 1) y la dirección IP de destino (el gNB 1). Durante una prueba simple, como enviar `ping` a `google.com` desde un UE del *slice* 1 a través de `uesimtun0`, es fácil ver cómo el campo *n_bytes* del flujo relevante aumenta mientras no se modifica en los demás. Esto confirma que la clasificación por puerto e IP definida en la arquitectura efectivamente separa los nueve flujos sin ambigüedades.

El siguiente paso es incorporar los modelos al funcionamiento en tiempo real del sistema. Cada segundo, el software externo ejecuta la siguiente secuencia: primero, lee el valor de *throughput* expuesto por ODL promediándolo con la lectura del segundo anterior. Luego, actualiza la ventana de 120 valores utilizada por el modelo y aplica el modelo correspondiente para obtener la predicción del instante siguiente y, finalmente, escribe la nueva asignación en la API de ODL. Este proceso ocurre de manera continua mientras el sistema está activo.

La lógica de asignación es distinta según el *slice*. En el caso del *slice* de video, se toma directamente el valor que predice el modelo. En el *slice* de audio, se utiliza también la predicción, pero respetando el mínimo garantizado definido previamente (de manera que nunca quede por debajo de ese umbral). En el *slice* de navegación web, el valor asignado resulta simplemente del ancho de banda remanente una vez descontados los otros dos *slices*. Todo este proceso se ejecuta en paralelo para cada gNB, de forma independiente, lo que permite que el sistema escale sin necesidad de coordinación central adicional.

Para comprobar que la asignación dinámica efectivamente impacta en el *slice best effort*, se ejecutó `iperf` entre un UE del *slice* 3 y la VM de Gateway, suplantando el tráfico de navegación web. Dado que el enlace se establece a través de `uesimtun0`, el tráfico atraviesa el gNB, el UPF correspondiente y finalmente OVS, de modo que cae en la cola asignada a ese *slice*. Como `iperf` toma todo el ancho de banda disponible para enviar o recibir tráfico, este método permite observar de manera clara cómo los recursos tomados por la herramienta disminuyen cuando el *slice* de video aumenta su asignación y vuelven a incrementarse cuando la demanda del *slice* prioritario baja. Esta prueba simple ilustra que el mecanismo de reasignación es efectivo y actúa sobre tráfico real en tiempo de ejecución.

Aunque sería posible complementar estas observaciones con métricas adicionales como latencia, este aspecto no se analiza en profundidad porque el foco del trabajo está puesto en la asignación de ancho de banda entre *slices*.

Con esto se muestra que el sistema implementado en el *testbed* funciona de

6.4. Implementación en el sistema completo

punta a punta: el aislamiento entre *slices* se cumple, la clasificación por flujo opera como se espera, y los modelos modifican efectivamente los recursos asignados según sus predicciones y las reglas definidas para cada tipo de tráfico.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 7

Conclusiones y trabajo futuro

La construcción del *testbed*, las decisiones de diseño y los experimentos que lo acompañan fueron guiados por una misma intención: entender, con un nivel razonable de detalle, cómo podría funcionar un mecanismo de asignación dinámica de recursos en una red 5G que opere en condiciones realistas.

El trabajo no apunta a resolver en su totalidad los problemas abiertos, sino a aportar una experiencia concreta y reproducible sobre cómo se articulan distintas tecnologías abiertas cuando se las integra para estudiar el comportamiento de los *slices* y su adaptación. Con esa perspectiva, las conclusiones que siguen resumen los principales aprendizajes obtenidos y los aspectos del sistema que pudieron validarse experimentalmente.

7.1. Conclusiones

El trabajo presentado desarrolla y valida un entorno experimental para estudiar la asignación dinámica de recursos en redes móviles 5G, con énfasis en la segmentación por *slices* y en la interacción entre el acceso, el núcleo y el transporte. El primer resultado relevante es la construcción de un *testbed standalone* de extremo a extremo, que integra un núcleo 5G completo, múltiples gNBs, funciones de transporte programables mediante SDN y un conjunto de UEs virtuales capaces de generar tráfico realista. Este entorno se compone de herramientas abiertas, se documenta de manera integral y se hace disponible en este [repositorio](#), lo que facilita su reproducción y extensión. La arquitectura final habilita la incorporación sistemática de nuevos gNBs y UEs, manteniendo criterios uniformes para la gestión de recursos y el aislamiento entre *slices*.

Sobre este *testbed* se implementan dos mecanismos complementarios de adaptación de recursos. El primero es de tipo reactivo y se basa en umbrales simples, lo que permite modificar la asignación ante variaciones abruptas en el consumo de los *slices*. El segundo es de tipo predictivo y utiliza modelos LSTM entrenados a partir de mediciones reales del tráfico. El diseño y la validación de estos modelos se realiza íntegramente dentro del entorno experimental, lo que permite evaluar su comportamiento en un escenario controlado pero representativo del

Capítulo 7. Conclusiones y trabajo futuro

funcionamiento de una red móvil.

En el caso del *slice* de video, se comprueba que un modelo LSTM simple es capaz de anticipar el *throughput* de corto plazo de un flujo compuesto por varios UEs, siempre que se trabaje con ventanas temporales que capturen un ciclo completo del comportamiento del tráfico. La experimentación también muestra que la función de pérdida utilizada durante el entrenamiento tiene un impacto directo sobre la adecuación del modelo a los requisitos del sistema. Ajustando la función de pérdida para penalizar de forma diferenciada los errores según su signo, es posible priorizar la reducción de la subpredicción a costa de un aumento en el error medio. Esta estrategia se valida experimentalmente y permite seleccionar una configuración que equilibra la sobreasignación con la precisión necesaria para operar en el *testbed*.

La integración de predicción conformal permitió reemplazar el ajuste heurístico de la función de pérdida por un mecanismo estadístico capaz de controlar explícitamente la subpredicción. El modelo LSTM conserva la capacidad de estimar el *throughput* futuro, mientras que la calibración sobre los residuos provee un margen cuantificable para asegurar el nivel de cobertura deseado. En el primer experimento, el uso de un percentil fijo logró reducir la subpredicción a un valor cercano al 2%, con el costo de un incremento constante en el ancho de banda asignado. Luego, la variante adaptativa mantuvo la cobertura pero no mejoró el aprovechamiento de recursos en este escenario, sugiriendo que la técnica puede ser útil para garantizar umbrales de seguridad pero que su eficiencia depende de la variabilidad temporal del tráfico.

Desde el punto de vista operativo, el sistema integrado permite confirmar algunos puntos: se valida el aislamiento entre *slices* a nivel del UPF y del plano de transporte, se verifica que el tráfico generado por los UEs sigue el recorrido completo de la arquitectura y se observa la respuesta efectiva del *slice* de menor prioridad ante la reasignación dinámica. Estos resultados confirman que la interacción entre los modelos de predicción, el plano de control SDN y la infraestructura 5G se comporta de acuerdo con los criterios definidos para cada *slice*.

7.2. Trabajo futuro

El problema y la virtud de implementar una red móvil de extremo a extremo es que se abren muchas más aristas de las que es posible abordar. En cada esquina del camino de implementación se han señalado las limitaciones y las decisiones que debieron tomarse para seguir una metodología sin salir del alcance, y en cada una de esas curvas se abrió un camino que queda por recorrer.

Por conjeturas ya explicadas, el único *slicing* abordado fue el de transporte, mientras los dominios de acceso y núcleo siguen sin segmentarse. El acceso radio tiene actualmente muy pocas opciones de herramientas de código abierto que permitan manipular los recursos de radiofrecuencia, y menos aún orquestarlo. Un ejemplo en el que esto es posible es mediante la modificación del módulo de *Scheduler* de OpenAirInterface, que puede ser adaptado para variar la asignación de recursos por dispositivo o por grupos de dispositivos en respuesta a alguna lógi-

ca. Por su parte, srsRAN admite una segmentación fija de los PRB que asigna por *slice*. En ningún caso se encontró la capacidad de orquestar estas asignaciones desde afuera de las propias herramientas, lo que hoy define un límite en lo que se puede implementar, pero que no está libre de futuras versiones o de desarrollos particulares en el futuro.

El *slicing* de núcleo también tiene limitaciones propias de la herramienta. En el caso de Open5GS, no se cuenta con la opción de crear o destruir UPFs (*slices*) en tiempo de ejecución, por lo que un ciclo de vida con aprovisionamiento y desmantelamiento no sería viable. Sin embargo, otros trabajos [96] han tomado la opción de preparar y disponibilizar una cantidad de VMs con recursos para encaminar el tráfico, y que solo se utilicen ante una necesidad de tráfico, emulando el comportamiento de aprovisionamiento. Otra ruta a recorrer para el núcleo puede ser un despliegue sobre Kubernetes, que no requiera de la interacción con la aplicación de Open5GS (los `yaml` quedarían intactos) para variar los recursos asignados al UPF. Esto bien podría ser orquestado desde el exterior con plataformas apropiadas para la gestión de recursos como OpenStack, respondiendo a la necesidad de unificar la gestión del *slice* con la lógica que hoy maneja solo recursos de transporte.

Más caminos se desprenden de los cuestionamientos de topología: ninguna red de transporte real, más allá de alguna red privada pequeña, consiste solo de un switch. La generalización de la red de transporte debería complejizar significativamente la arquitectura con el agregado de otros switches, routers, redundancias, y demás consideraciones conocidas. La algoritmia asociada ganaría mucho en complejidad, para lo cual el ya instalado controlador de SDN sería un gran aliado. Sin embargo, también se descentralizaría el procesamiento de flujos que hoy hace OVS en un único switch, haciendo escalable la solución de considerar cada flujo de cada *slice* de cada gNB por separado.

Por último, y ya bordeando campos complementarios al de las redes móviles: de lo revisado en los antecedentes técnicos, se ve que es poco común la implementación de *testbeds* con tantos dispositivos como los 20 aquí dispuestos; tampoco se ve el uso de tráfico realista, perfilado con aleatoriedad que busque acercarse a la de consumo real. Estos flujos son fácilmente capturables por medio de la API armada para ese fin, lo que da lugar a aprovechar el sistema para generar datos de tráfico realista que permitan investigar mejores modelos para adaptar recursos, así como para otras aplicaciones innovadoras aplicables a este tipo de redes.

Estos son solo algunos de los trayectos que quedan fuera del alcance y que quedan planteados como posibles puntos de partida para trabajos futuros en esta línea de investigación.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice A

Guía de configuración del testbed

A.1. Introducción

Testbed adaptado para el uso de Proxmox como ambiente de virtualización, con el agregado de la configuración de todos los componentes para operar con Network Slicing, así como con el monitoreo y adaptación dinámica de bitrates en el switch intermedio. Esto da a la red capacidad de slicing dinámico a nivel de red de transporte.

En principio, se diseña el testbed para trabajar con dos slices: uno de tráfico prioritario, al que se le provee todo el ancho de banda que requiera, y otro best effort, que dispone de los recursos restantes.

También se brindan detalles para la implementación opcional de un gNB externo al host de Proxmox que ejecute srsRAN y se comunique directamente con un teléfono móvil (UE) real.

A.2. Configuración de Proxmox

A nivel de Proxmox VE se deben configurar máquinas virtuales y bridges Linux para el armado de la red.

Los bridges Linux deben tener las siguientes características:

Nombre	Red	Acepta VLAN	Tiene interfaz asociada
vmbr230	192.168.230.0/24	No	No
vmbr233	192.168.233.0/24	Sí	No(*)
vmbr0	Red local del router de salida	No	Sí, salida física del equipo o cluster hacia internet

(*) Puede agregarse una interfaz física a un segundo puerto para conectar un gNB externo.

Las máquinas virtuales deben ser creadas con la imagen ISO de Ubuntu Server 20.04. En base a la experiencia del sistema bajo estrés, se recomiendan las siguientes características:

Apéndice A. Guía de configuración del testbed

ID (Nombre)	Disco	RAM	Cores	Arquitectura de CPU	Interfaces de Red
100 (OVS)	24	4	4	Default	7
101 (GW)	16	4	2	Default	2
102 (Controller)	24	6	2	Default	2
103 (CN-CP)	24	6	2	kvm64	3
104 (CN-UP-1)	32	4	4	kvm64	1
105 (CN-UP-2)	32	4	4	kvm64	1
106 (gNB)	32	4	2	Default	1
107 (UE-1)	32	4	4	Default	1
108 (UE-2)	32	4	4	Default	1

En el proceso de creación de la máquina virtual, solo es posible agregar una única interfaz de red. Las demás deben agregarse accediendo a la información de la VM en Proxmox y, dentro del menú de “Hardware”, seleccionar “Network Devices” y “Add”.

Las características de las VM deberían configurarse como en la siguiente tabla, con la consideración de que la configuración de IP (última columna) no corresponde a Proxmox sino que se realizará después sobre cada VM:

VM	Bridge	VLAN	IP/Máscara	Gateway
OVS	vmbr230	N/A	192.168.230.5/24	192.168.230.0
OVS	vmbr233	101	N/A	N/A
OVS	vmbr233	1030	N/A	N/A
OVS	vmbr233	1031	N/A	N/A
OVS	vmbr233	1032	N/A	N/A
OVS	vmbr233	104	N/A	N/A
OVS	vmbr233	105	N/A	N/A
OVS	vmbr233	106	N/A	N/A
OVS	vmbr233	107	N/A	N/A
OVS	vmbr233	108	N/A	N/A
GW	vmbr0	N/A	DHCP	N/A
GW	vmbr233	101	192.168.233.1/24	N/A
Controller	vmbr0	N/A	DHCP	N/A
Controller	vmbr230	N/A	192.168.230.0/24	192.168.233.1
CN-CP	vmbr233	1030	192.168.233.2/24	192.168.233.1
CN-CP	vmbr233	1031	192.168.233.3/24	192.168.233.1
CN-CP	vmbr233	1032	192.168.233.4/24	192.168.233.1
CN-UP-1	vmbr233	104	192.168.233.5/24	192.168.233.1
CN-UP-2	vmbr233	105	192.168.233.6/24	192.168.233.1
gNB	vmbr233	106	192.168.233.7/24	192.168.233.1
UE-1	vmbr233	107	192.168.233.8/24	192.168.233.1
UE-2	vmbr233	108	192.168.233.9/24	192.168.233.1

A.3. Configuración de VMs

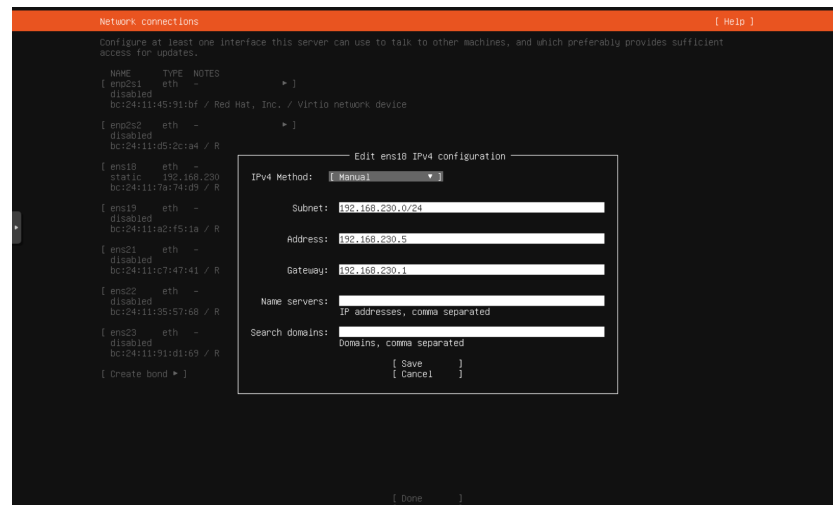


Figura A.1: Menú de configuración de Ubuntu Server.

A.3. Configuración de VMs

En esta sección se describe la configuración básica a realizar sobre las VMs, a nivel de sistema operativo y networking. Lo propio de cada componente Open Source utilizado para la red móvil se presenta en la sección siguiente.

Los pasos correspondientes a la configuración de cada VM son:

1. **Configuración de red**, consistente en dirección IP, máscara, gateway (según tabla anterior) y DNS (8.8.8.8, 8.8.4.4)

- a) Al inicializar una VM con Ubuntu Server, el menú de configuración de la Figura A.1 es desplegado.

Allí deben llenar los campos explicados. También deberían figurar las interfaces de red agregadas desde Proxmox en la siguiente pantalla (ver Figura A.2).

- b) Para el caso de los bridges hacia internet de las máquinas GW y Controller, debe setearse en netplan la interfaz con DHCP. Lo primero es revisar la configuración de red para saber qué interfaz utilizar. Para esto se puede usar “ip a”, tomar la interfaz nueva, y luego ingresar a la edición de netplan:

```
$ sudo nano /etc/netplan/00-installer-config.yaml
```

Allí agregar la nueva interfaz y la capacidad DHCP. Por ejemplo:

```
network:
  ethernets:
    ens19:
      dhcp4: yes
```

Apéndice A. Guía de configuración del testbed

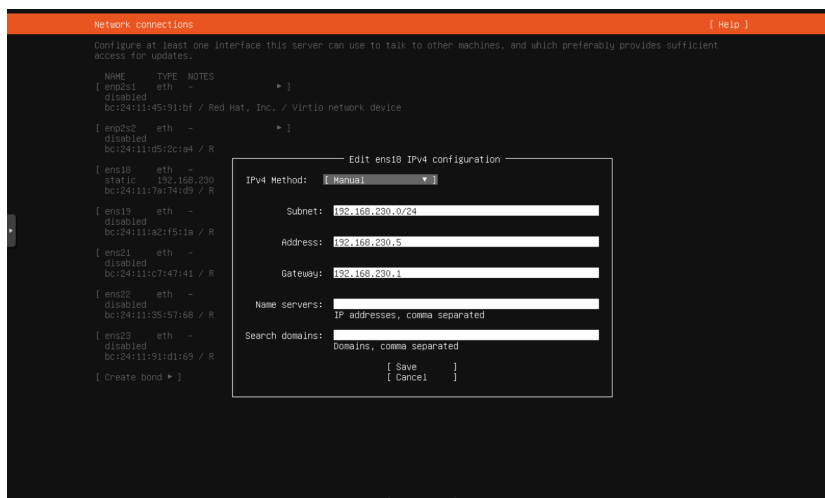


Figura A.2: Menú de configuración de Ubuntu Server.

Luego deben levantarse el enlace y activar el cliente DHCP:

```
$ sudo ip link set ens19 up
$ sudo dhclient ens19
```

Finalmente, puede probarse la salida a internet:

```
$ ping -c 3 google.com
```

Nota: para la asignación de IPs en VMs con múltiples interfaces, utilizar la GUI de Proxmox para asociar la MAC correspondiente a cada interfaz y así determinar qué configurar en cada una.

2. Actualización APT:

```
sudo apt update && sudo apt upgrade -y
```

3. Instalación de servidor SSH:

```
sudo apt install openssh-server -y
```

4. Configuración persistente de forwarding IPv4

a) `sudo nano /etc/sysctl.conf`

b) Remover el comentario (borrar “#”) de la línea que dice:

```
net.ipv4.ip_forward=1
```

c) La línea modificada se lee al iniciar el kernel, por lo que puede reiniciarse la VM o correr el siguiente comando por única vez:

A.4. Configuración de Software

```
sudo sysctl -w net.ipv4.ip_forward=1
```

5. Configuración persistente de NAT en iptables

- a) `sudo apt-get install iptables-persistent`
- b) `sudo iptables -t nat -A POSTROUTING -s 192.168.233.0/24
-o ens19 -j MASQUERADE`
(La IP/máscara es la red aguas abajo, y la interfaz corresponde a la salida a internet)
- c) `sudo netfilter-persistent save`

Los pasos 1-3 aplican a todas las VMs creadas, mientras que los pasos 4 y 5 aplican a todas a excepción de la RAN (gNB, UE-1 y UE-2).

A.4. Configuración de Software

Dado que, en la sección anterior, se instalaron servidores SSH en todas las VMs y se configuró su conectividad, todos los pasos de esta sección pueden hacerse desde algún host común vía SSH para facilitar el copiado y pegado.

A.4.1. OPEN5GS

Open5GS es el software elegido para ejecutar el núcleo de la red móvil. En este Testbed, el núcleo se desarrolla en tres VMs: una para el plano de control, una para un plano de usuario o de datos (User Plane Function o UPF del Slice 1) y otra para un segundo plano de usuario (Slice 2). Las VMs de UPF tienen una única interfaz de red mientras que la del plano de control tiene tres: una para la AMF, y dos para los SMF correspondientes a cada UPF.

Para la instalación de Open5GS se recomienda la documentación oficial. Se debe tener la consideración de que, según la versión de Ubuntu, varía un comando de instalación de MongoDB para el uso del core. La instalación de Open5GS debe hacerse en las tres VMs, independientemente de que cada una luego ejecute funciones distintas. Por otra parte, cabe resaltar que la instalación de MongoDB, solamente es necesaria realizarla en la máquina del plano de control.

Las nociones para la configuración de un segundo plano de usuario (definiendo el segundo slice a nivel de núcleo) fueron tomadas del tutorial [97], al que es posible referirse ante dudas.

CP

Cambios en la configuración del Plano de Control.

AMF: /etc/amf.yaml

Apéndice A. Guía de configuración del testbed

```
ngap:
  server:
    - address: 192.168.233.2
metrics:
  server:
    - address: 127.0.0.5
      port: 9090
guami:
  - plmn_id:
      mcc: 001
      mnc: 01
    amf_id:
      region: 2
      set: 1
tai:
  - plmn_id:
      mcc: 001
      mnc: 01
    tac: 1
plmn_support:
  - plmn_id:
      mcc: 001
      mnc: 01
    s_nssai:
      - sst: 1
        sd: 000001
      - sst: 1
        sd: 000002
      - sst: 1
security:
  integrity_order : [ NIA2, NIA1, NIA0 ]
  ciphering_order : [ NEA0, NEA1, NEA2 ]
network_name:
  full: Open5GS
  short: Next
amf_name: open5gs-amf0

NRF: /etc/open5gs/nrf.yaml

nrf:
  serving: # 5G roaming requires PLMN in NRF
    - plmn_id:
        mcc: 001
        mnc: 01
sbi:
  server:
```

A.4. Configuración de Software

```
- address: 127.0.0.10
  port: 7777
```

NSSF: /etc/open5gs/nssf.yaml

```
nsi:
- uri: http://127.0.0.10:7777
  s_nssai:
    sst: 1
    sd: 000001
- uri: http://127.0.0.10:7777
  s_nssai:
    sst: 1
    sd: 000002
```

SMF1: /etc/open5gs/smf1.yaml

```
logger:
  file:
    path: /var/log/open5gs/smf1.log
```

```
global:
  smf:
    sbi:
      server:
        - address: 127.0.0.24
          port: 7777
      client:
    pfcp:
      server:
        - address: 192.168.233.3
    gtpu:
      server:
        - address: 192.168.233.3
    client:
      upf:
        - address: 192.168.233.4
          dnn: internet
    metrics:
      server:
        - address: 127.0.0.4
          port: 9090
    session:
      - subnet: 10.45.0.1/16
        dnn: internet
  dns:
```

Apéndice A. Guía de configuración del testbed

```
- 8.8.8.8
- 8.8.4.4
mtu: 1400
info:
  - s_nssai:
    - sst: 1
      dnn:
        - internet
  tai:
    - plmn_id:
      mcc: 001
      mnc: 01
      tac: 1
```

SMF2: /etc/open5gs/smf2.yaml

```
logger:
  file:
    path: /var/log/open5gs/smf2.log
```

```
pfcf:
  server:
    - address: 192.168.233.9
  client:
    upf:
      - address: 192.168.233.5
      dnn: internet
```

```
gtpu:
  server:
    - address: 192.168.233.9
```

```
metrics:
  server:
    - address: 127.0.0.4
      port: 9090
```

```
session:
  - subnet: 10.46.0.1/16
    dnn: internet
```

```
dns:
  - 8.8.8.8
  - 8.8.4.4
```

```
mtu: 1400
info:
  - s_nssai:
    - sst: 1
      sd: 000002
      dnn:
```

```
    - internet
tai:
  - plmn_id:
      mcc: 001
      mnc: 01
  tac: 1
```

Script de despliegue del plano de control:

```
/usr/bin/open5gs-nrfd &
sleep 2
/usr/bin/open5gs-scpd &
sleep 2
/usr/bin/open5gs-amfd &
sleep 2
/usr/bin/open5gs-smfd -c /etc/open5gs/smf1.yaml &
/usr/bin/open5gs-smfd -c /etc/open5gs/smf2.yaml &
/usr/bin/open5gs-ausfd &
/usr/bin/open5gs-udmd &
/usr/bin/open5gs-udrd &
/usr/bin/open5gs-pcfd &
/usr/bin/open5gs-nssfd &
/usr/bin/open5gs-bsfd &
/usr/bin/open5gs-webuid &
```

UP1

UPF: /etc/open5gs/upf.yaml

```
upf:
  pfcp:
    server:
      - address: 192.168.233.4
  gtpu:
    server:
      - address: 192.168.233.4
  session:
    - subnet: 10.45.0.0/16
      gateway: 10.45.0.1
      dnn: internet
      dev: ogstun
  metrics:
    server:
      - address: 127.0.0.7
      port: 9090
```

Apéndice A. Guía de configuración del testbed

UP2

UPF: /etc/open5gs/upf.yaml

```
upf:
  pfcp:
    server:
      - address: 192.168.233.5
  gtpu:
    server:
      - address: 192.168.233.5
  session:
    - subnet: 10.46.0.0/16
      gateway: 10.46.0.1
      dnn: internet
      dev: ogstun
  metrics:
    server:
      - address: 127.0.0.7
        port: 9090
```

Registro de UEs en el CP

En la máquina del CP, será necesario registrar los UEs que se van a conectar a la red. Para ello debe estar corriendo el servicio `open5gs-webuid`. Si está siendo ejecutado correctamente, en un navegador se accede a `localhost:9999`.

Las credenciales para entrar son: **usuario:** admin, **password:** 1324. A continuación, será necesario añadir la tarjeta SIM como suscriptor de la red, accediendo a la sección *Subscriber* de la interfaz web de Open5GS ubicada en `localhost:9999`. Se pulsa el botón “+” y se introduce la información de la SIM.

La información introducida para el UE físico que se utilizará es la siguiente:

▪ UE físico

- IMSI: 001019876543210
- K: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
- OPc: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
- SST: 1
- SD: Vacío

▪ UE Emulado 1

- IMSI: 001010000000001
- K: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
- OPc: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF

- SST: 1
- SD: 1
- **UE Emulado 2**
 - IMSI: 0010100000000002
 - K: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
 - OPc: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
 - SST: 1
 - SD: 2

En función del SST y SD que se introduzca, el UE se conectará a un slice u otro. Eso sí, debe coincidir con la información introducida en UERANSIM.

A.4.2. OpenDayLight

El controlador SDN utilizado es OpenDayLight (ODL), instalado con Apache Karaf en su versión 0.8.4. Para este proceso se debe ejecutar lo siguiente en la VM Controller:

```
sudo apt-get -y update
sudo apt-get -y upgrade
sudo apt-get -y install unzip
sudo apt-get -y install openjdk-8-jre
sudo update-alternatives --config java
ls -l /etc/alternatives/java
echo 'export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre'
>> ~/.bashrc
source ~/.bashrc
echo $JAVA_HOME
sudo apt install curl -y
curl -XGET -O https://nexus.opendaylight.org/content/repositories/
opendaylight.release/org/opendaylight/integration/karaf/0.8.4/
karaf-0.8.4.zip
unzip karaf-0.8.4.zip
cd karaf-0.8.4/
./bin/karaf clean
```

Aquí se abre la terminal de ODL, sobre la cual se deben instalar los *features* de interés para el uso de ODL. En este Testbed se utilizan los necesarios para el uso de OVSDB, *plugin* descrito en esta guía. En la terminal de ODL, ejecutar:

```
feature:install odl-ovsdb-southbound-impl-ui
```

Con esto se instalarán varias dependencias, que pueden revisarse usando:

Apéndice A. Guía de configuración del testbed

```
feature:list -i
```

Las *features* instaladas ya deberían permitir el uso de la API para la lectura y escritura de configuraciones y estadísticas. Los métodos utilizados para esto se verán en la sección siguiente.

A.4.3. OVS

En OVS debe terminar de configurarse la red, a causa de las interfaces que van a conectarse en capa 2 con los hosts locales. Esto consiste en dos pasos. El primero es editar el netplan desde la terminal de SSH con:

```
sudo nano /etc/netplan/00-installer-config.yaml
```

y agregar todas las interfaces nuevas:

```
network:
  version: 2
  ethernet:
    ens18:
      dhcp4: false
      addresses:
        - 192.168.230.5/24 # Static IP for ens18
      gateway4: 192.168.230.1
      nameservers:
        addresses:
          - 8.8.8.8 # DNS servers (Google DNS)
    ens19:
      dhcp4: false # Disable DHCP (No IP)
      optional: true # Optional
    ens20:
      dhcp4: false
      optional: true
    ens21:
      dhcp4: false
      optional: true
    ens22:
      dhcp4: false
      optional: true
    ens23:
      dhcp4: false
      optional: true
    enp2s1:
      dhcp4: false
      optional: true
    enp2s2:
      dhcp4: false
      optional: true
```

A.4. Configuración de Software

```
enp2s3:
  dhcp4: false
  optional: true
enp2s4:
  dhcp4: false
  optional: true
```

Finalmente, aplicar los cambios con:

```
$ sudo netplan apply
```

Se puede validar que las interfaces estén todas en estado “UP” usando `ip a`. Si no lo estuvieran, se deben levantar de a una usando:

```
sudo ip link set enXXX up
```

Nota: Si se utilizó Ubuntu Desktop (en lugar de Server) para la VM del OVS, es importante desactivar el Network Manager y trabajar solamente con Netplan. Esto se debe a que el Manager genera sus propias rutas con mayor prioridad y mucho tráfico nunca ingresa en las colas establecidas por OVS, por lo que su bitrate no se limita como se espera.

Lo siguiente es hacer la instalación y configuración de OpenVSwitch en el host. Para esto:

```
$ sudo apt install openvswitch-switch -y
$ sudo apt install -y net-tools iproute2 bridge-utils iputils-ping
tcpdump traceroute
$ sudo ovs-vsctl add-br br0
$ sudo ip addr add 192.168.233.1/24 dev br0
$ sudo ip link set br0 up
$ sudo ovs-vsctl add-port br0 ens19
$ sudo ovs-vsctl add-port br0 ens18
$ sudo ovs-vsctl add-port br0 ens20
$ sudo ovs-vsctl add-port br0 ens21
$ sudo ovs-vsctl add-port br0 ens22
$ sudo ovs-vsctl add-port br0 ens23
$ sudo ovs-vsctl add-port br0 enp2s1
$ sudo ovs-vsctl add-port br0 enp2s2
$ sudo ovs-vsctl add-port br0 enp2s3
$ sudo ovs-vsctl add-port br0 enp2s4
```

Se agregan también los comandos para el controlador y el protocolo OpenFlow:

```
sudo ovs-vsctl set-controller br0 tcp:192.168.230.1:6633
sudo ovs-vsctl set-manager ptcp:6640
sudo ovs-vsctl set bridge br0 protocols=OpenFlow13
```

Apéndice A. Guía de configuración del testbed

En este punto es necesario un reboot de la VM para aplicar los puertos.

```
$ sudo ovs-ofctl add-flow br0 -O OpenFlow13  
"table=0,priority=100,actions=normal"
```

Con esto queda habilitado el flujo en general para todo el tráfico que no vaya a tener reglas. Sobre esa base se busca generar reglas en la tabla de OVS, con prioridad mayor, que permitan limitar el máximo bitrate de subida (UL) o de bajada (DL).

Para implementar dichas reglas se selecciona la interfaz correspondiente: si se quiere limitar el bitrate de UL, se deben establecer reglas sobre la interfaz del switch que va hacia el UPF; para la bajada, se establecen sobre la interfaz que va hacia el gNB. En este caso, se proveen los comandos para ambos tipos de tráfico, a ser ejecutados directamente sobre el switch. Esto no se hace escalable en la medida que se agregan gNBs a la red de acceso o switches a la de transporte, por lo que en ese caso se debería hacer a través del controlador.

Para limitar el tráfico de bajada, se ejecuta el siguiente comando en el switch:

```
$ sudo ovs-vsctl -- \  
set port enp2s1 qos=@qosgnb -- \  
--id=@qosgnb create qos type=linux-htb \  
queues:10=@slice1 \  
queues:20=@slice2 -- \  
--id=@slice1 create queue other-config:max-rate=30000000 -- \  
--id=@slice2 create queue other-config:max-rate=10000000
```

Se introdujo que el objetivo es utilizar una slice prioritaria (en este caso inicializada en 30 Mbps) y una slice best effort (en este caso en 10 Mbps). Estos valores iniciales se irán modificando en tiempo de ejecución.

Es posible verificar la creación de QoS y colas utilizando los siguientes comandos:

```
$ sudo ovs-vsctl list queue  
$ sudo ovs-vsctl list qos
```

En la lista de QoS se deberían listar las mismas IDs de colas que figuran en la lista de queue, asociadas a los números introducidos (en este caso, cola 10 para Slice 1 y cola 20 para Slice 2).

También puede validarse que el QoS con nuestras colas esté asociado al puerto correcto, utilizando el siguiente comando y buscando el ID en el parámetro “qos” del puerto:

```
$ sudo ovs-vsctl list port
```

Luego de creadas esas colas y asociadas a la interfaz del gNB (en el ejemplo “enp2s1”), se deben crear los flujos de tráfico asociados a esas colas:

A.4. Configuración de Software

```
$ sudo ovs-ofctl add-flow br0 -O OpenFlow13  
"table=0,priority=200,in_port=9,actions=set_queue:10,normal"
```

```
$ sudo ovs-ofctl add-flow br0 -O OpenFlow13  
"table=0,priority=200,in_port=4,actions=set_queue:20,normal"
```

Se puede observar que los últimos flujos agregados tienen mayor prioridad que el flujo original, por lo que el switch los revisa antes. Si se detecta que el tráfico proviene de la interfaz de un UPF (asociado al “in_port” en OVS), se ingresa en la regla y se limita el bitrate según la cola asociada al UPF. Para determinar a qué puerto OVS pertenece cada interfaz, buscar asociación de la MAC (Proxmox) con el número de in_port (en OVS, mediante el comando `sudo ovs-ofctl -O OpenFlow13 show br0`).

Estos se validan utilizando el siguiente comando, que también va a evidenciar si ya existió tráfico por esas colas (al menos unos paquetes de control es esperable que circulen luego de unos segundos, por lo que sirve para confirmar si la regla está operativa):

```
$ sudo ovs-ofctl -O OpenFlow13 dump-flows br0
```

Para limitar el tráfico de subida, se ejecuta análogamente:

```
$ sudo ovs-vsctl -- \  
set port ens23 qos=@qosupf1 -- \  
--id=@qosupf1 create qos type=linux-htb \  
queues:11=@slice1u -- \  
--id=@slice1u create queue other-config:max-rate=2000000000
```

```
$ sudo ovs-vsctl -- \  
set port enp2s4 qos=@qosupf2 -- \  
--id=@qosupf2 create qos type=linux-htb \  
queues:21=@slice2u -- \  
--id=@slice2u create queue other-config:max-rate=1000000000
```

Donde los puertos a elegir corresponden al UPF 1 y UPF 2 respectivamente, mientras que los flujos se asocian al puerto del gNB como sigue:

```
$ sudo ovs-ofctl add-flow br0 -O OpenFlow13  
"table=0,priority=200,in_port=1,actions=set_queue:11,normal"  
$ sudo ovs-ofctl add-flow br0 -O OpenFlow13  
"table=0,priority=200,in_port=1,actions=set_queue:21,normal"
```

A.4.4. UERANSIM

En la implementación de este Testbed se probaron dos proyectos Open Source para la red de acceso: srsRAN (con srsUE) y UERANSIM. Después de varias

Apéndice A. Guía de configuración del testbed

pruebas con ambos, se optó por un modelo híbrido en el que se use srsRAN para un gNB físico (explicado en la subsección siguiente) y UERANSIM para un nodo simulado, junto con sus respectivos UEs.

La instalación de UERANSIM es bastante simple, solo hay que instalar las dependencias, clonar el repositorio y compilar el código. En este enlace se muestra una guía.

En el directorio UERANSIM/build se encontrarán los binarios ejecutables del gNB (**nr-gnb**) y del UE (**nr-ue**). Por otra parte, en el directorio UERANSIM/config se encuentran archivos de configuración de ejemplo tanto del gNB como del UE. Los que utilizamos como base son `open5gs-gnb.yaml` y `open5gs-ue.yaml`.

A continuación se muestra cómo editar los archivos de configuración para adecuarlos a nuestro escenario.

gNB

```
mcc: '001'          # Mobile Country Code value
mnc: '01'           # Mobile Network Code value (2 or 3 digits)

nci: '0x000000010'  # NR Cell Identity (36-bit)
idLength: 32        # NR gNB ID length in bits [22...32]
tac: 1              # Tracking Area Code

linkIp: 192.168.233.7 # gNB's local IP address for Radio Link Simulation
                        # (Usually same with local IP)
ngapIp: 192.168.233.7 # gNB's local IP address for N2 Interface (Usually
                        # same with local IP)
gtpIp: 192.168.233.7  # gNB's local IP address for N3 Interface (Usually
                        # same with local IP)

# List of AMF address information
amfConfigs:
  - address: 192.168.233.2
    port: 38412

# List of supported S-NSSAIs by this gNB
slices:
  - sst: 1
  - sst: 1
    sd: 1
  - sst: 1
    sd: 2

# Indicates whether or not SCTP stream number errors should be ignored.
ignoreStreamIds: true
```

UE1

A.4. Configuración de Software

```
# IMSI number of the UE. IMSI = [MCC|MNC|MSISDN] (In total 15 digits)
supi: 'imsi-0010100000000001'
# Mobile Country Code value of HPLMN
mcc: '001'
# Mobile Network Code value of HPLMN (2 or 3 digits)
mnc: '01'
# SUCI Protection Scheme
protectionScheme: 0
# Home Network Public Key for protecting with SUCI Profile A
homeNetworkPublicKey:
'5a8d38864820197c3394b92613b20b91633cbd897119273bf8e4a6f4eec0a650'
# Home Network Public Key ID for protecting with SUCI Profile A
homeNetworkPublicKeyId: 1
# Routing Indicator
routingIndicator: '0000'

# Permanent subscription key
key: 'FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF'
# Operator code (OP or OPC) of the UE
op: 'FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF'
# This value specifies the OP type and it can be either 'OP' or 'OPC'
opType: 'OPC'
# Authentication Management Field (AMF) value
amf: '8000'
# IMEI number of the device. It is used if no SUPI is provided
imei: '356938035643802'
# IMEISV number of the device.
imeiSv: '4370816125816150'

# Network mask used for the UE's TUN interface to define the subnet size
tunNetmask: '255.255.255.0'

# List of gNB IP addresses for Radio Link Simulation
gnbSearchList:
- 192.168.233.7

# Initial PDU sessions to be established
sessions:
- type: 'IPv4'
  apn: 'internet'
  slice:
    sst: 1

# Configured NSSAI for this UE by HPLMN
configured-nssai:
```

Apéndice A. Guía de configuración del testbed

- sst: 1

Default Configured NSSAI for this UE

default-nssai:

- sst: 1

UE2

IMSI number of the UE. IMSI = [MCC|MNC|MSISDN] (In total 15 digits)

supi: 'imsi-001010000000002'

Mobile Country Code value of HPLMN

mcc: '001'

Mobile Network Code value of HPLMN (2 or 3 digits)

mnc: '01'

SUCI Protection Scheme

protectionScheme: 0

Home Network Public Key for protecting with SUCI Profile A

homeNetworkPublicKey:

'5a8d38864820197c3394b92613b20b91633cbd897119273bf8e4a6f4eec0a650'

Home Network Public Key ID for protecting with SUCI Profile A

homeNetworkPublicKeyId: 1

Routing Indicator

routingIndicator: '0000'

Permanent subscription key

key: 'FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF'

Operator code (OP or OPC) of the UE

op: 'FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF'

This value specifies the OP type and it can be either 'OP' or 'OPC'

opType: 'OPC'

Authentication Management Field (AMF) value

amf: '8000'

IMEI number of the device. It is used if no SUPI is provided

imei: '356938035643802'

IMEISV number of the device. It is used if no SUPI and IMEI is provided

imeiSv: '4370816125816150'

Network mask used for the UE's TUN interface to define the subnet size

tunNetmask: '255.255.255.0'

List of gNB IP addresses for Radio Link Simulation

gnbSearchList:

- 192.168.233.7

Initial PDU sessions to be established

sessions:

A.4. Configuración de Software

```
- type: 'IPv4'
  apn: 'internet'
  slice:
    sst: 1
    sd: 2

# Configured NSSAI for this UE by HPLMN
configured-nssai:
- sst: 1
  sd: 2

# Default Configured NSSAI for this UE
default-nssai:
- sst: 1
  sd: 2
```

Para ejecutar las instancias de gNB y UE basta con ejecutar los siguientes comandos en sus respectivas máquinas virtuales:

```
$ sudo ./nr-gnb -c open5gs-gnb.yaml
$ sudo ./nr-ue -c open5gs-ue.yaml
```

A.4.5. srsRAN

El nodo físico se implementa sobre un host distinto al del ambiente de Proxmox introducido al comienzo, que se conecta al mismo por un cable de red a un segundo puerto del host inicial.

srsRAN queda corriendo sobre el segundo host, que en este caso es un PC con Ubuntu Desktop 22 nativo. Ese host se conecta vía USB 3.0 al SDR de Ettus para configurar sus parámetros de radio y utilizarlo como RU. El UE utilizado es un teléfono móvil COTS, de marca Oppo, con una SIM grabada para este fin.

Cabe resaltar que es necesario configurar la VLAN e IP en la interfaz del Ubuntu Host externo conectada al servidor. Para ello se han utilizado los siguientes comandos:

```
sudo ip link add link <interfaz> name vlan105 type vlan id 105
sudo ip address add 192.168.233.6/24 dev vlan105
sudo ip link set vlan105 up
```

Para la instalación del srsRAN basta con seguir la guía de instalación. Una vez compilado el código, se accede al directorio `srsRAN_Project/build/apps/gnb` donde se encontrará el binario ejecutable del gNB. En este directorio se crea el siguiente archivo de configuración:

```
cu_cp:
  amf:
```

Apéndice A. Guía de configuración del testbed

```
    addr: 192.168.233.2
    port: 38412
    bind_addr: 192.168.233.6
    supported_tracking_areas:
      - tac: 1
        plmn_list:
          - plmn: "00101"
            tai_slice_support_list:
              - sst: 1

ru_sdr:
  device_driver: uhd
  #device_args: type=b200,num_recv_frames=64,num_send_frames=64
  srates: 23.04
  otw_format: sc12
  tx_gain: 80
  rx_gain: 40

cell_cfg:
  dl_arfcn: 632628
  band: 78
  channel_bandwidth_MHz: 20
  common_scs: 30
  plmn: "00101"
  tac: 1
  pci: 1

log:
  filename: /tmp/gnb.log
  all_level: warning

pcap:
  mac_enable: false
  mac_filename: /tmp/gnb_mac.pcap
  ngap_enable: false
  ngap_filename: /tmp/gnb_ngap.pcap
```

Para ejecutar el gNB basta con ejecutar el siguiente comando:

```
sudo ./gnb -c archivo_config.yaml
```

A.4.6. Configuración UE físico

Para poder conectar un teléfono a la red, es necesario hacer una configuración previa en el propio dispositivo. Esto es introducir la información del APN en la configuración de la red móvil del dispositivo.

A.4. Configuración de Software

Será necesario introducir la siguiente información del APN:

- MCC: 001
- MNC: 01
- APN: internet
- Protocolo de punto de acceso: IPv4

Por último, si el UE no se conectara a la red, sería necesario utilizar una aplicación que se encargue de forzar que el móvil utilice únicamente 5G. La aplicación es *5G Switch - Force 5G Only*.

Esta aplicación nos permite seleccionar el tipo de red al que quieres que se conecte el móvil.

A.4.7. Interacción externa con ODL

Se recomienda utilizar Postman inicialmente para probar:

- **Lectura de estadísticas.** Se utiliza el método GET para obtener la cantidad instantánea de bytes que han pasado por un flujo. El punto es leer esa cantidad con cierto período y hacer el promedio de la diferencia en ese período para estimar el throughput en (M)bps. En la etapa de validación, sólo interesa verificar que la lectura funciona. Para esto se selecciona el método GET, se deja el cuerpo de la solicitud vacío y se introduce el endpoint correcto. Un ejemplo de endpoint es el siguiente:

```
http://158.49.247.131:8181/restconf/operational/.opendaylight-  
inventory:nodes/node/openflow:206863090138899/node-connector/  
openflow:206863090138899:1/.opendaylight-port-statistics:flow-  
capable-node-connector-statistics
```

Si la formación del URL no está clara, se sugiere hacer un GET de directorios anteriores para conocer la ruta correcta. Por ejemplo:

```
http://158.49.247.131:8181/restconf/operational/.opendaylight-  
inventory:nodes/
```

- **Sobreescritura del bitrate para cierta cola.** Se utiliza el método PUT para modificar el máximo bitrate (**max-rate**) de la cola asociada a un flujo. La forma de elegir el nuevo bitrate se desprende de cómo se busque adaptarlo, por lo que en este punto basta con elegir cualquier número para el bitrate y validar que fue establecido. Para la escritura, se selecciona el método PUT y se arma el cuerpo como a continuación:

Apéndice A. Guía de configuración del testbed

```
{
  "ovsdb:queues": [
    {
      "queue-id": "queue://13b6b7a3-ea38-4680-a82f-278e2a73b489",
      "queue-uuid": "13b6b7a3-ea38-4680-a82f-278e2a73b489",
      "queues-other-config": [
        {
          "queue-other-config-key": "max-rate",
          "queue-other-config-value": "1000000"
        }
      ]
    }
  ]
}
```

En la URL se introduce el endpoint correspondiente a la cola que se quiere modificar, como el siguiente ejemplo:

```
http://158.49.247.131:8181/restconf/config/network-topology:network-topology/topology/ovsdb
```

El valor de la cola puede obtenerse de la revisión de un GET de directorios anteriores, aunque se recomienda acceder directamente a la consola del switch y ejecutar:

```
$ sudo ovs-vsctl list qos
```

De esta manera se puede tomar el ID de la cola deseada, asegurando que se corresponde con el QoS y puerto elegidos.

Para la validación, se modifica el método a GET, y de la URL solo se cambia “config” por “operational”, es decir (para el ejemplo):

```
http://158.49.247.131:8181/restconf/operational/network-topology:network-topology/topology/
```

También se puede validar directamente en el switch, ejecutando:

```
$ sudo ovs-vsctl list queue
```

Nota: los caracteres especiales que se usen como parte de los strings de ID y no como parte de la dirección deben traducirse a una codificación que pueda interpretar la API. Estrictamente, los ejemplos de la URL anterior sustituyen “:” por “%3A” y “/” por “%2F”.

Una vez verificada la capacidad de realizar esas operaciones, es posible pasar a ejecutar mecanismos automatizados para la adaptación del tráfico en tiempo real.

Referencias

- [1] Study on scenarios and requirements for next generation access technologies. Technical Report TR 38.913, 3rd Generation Partnership Project (3GPP), 2017.
- [2] 3rd Generation Partnership Project (3GPP). System architecture for the 5G System.
- [3] KPMG International. From telco to techco: Towards tomorrow's telecom. Technical report, KPMG, 2024. White paper (Spanish version).
- [4] TM Forum. Tm forum – accelerating industry transformation through collaboration. <https://www.tmforum.org/>, 2025.
- [5] GSMA. Network slicing use case requirements. https://www.gsma.com/solutions-and-impact/technologies/networks/gsma_resources/network-slicing-use-case-requirements-whitepaper/, 2018. Whitepaper.
- [6] Mika Uusitalo (Nokia). Network slicing: Enabling new business models. <https://www.nokia.com/blog/driving-tangible-monetization-and-business-growth-through-5g-network-slicing/>, 2025. Available Online.
- [7] GSMA. Generic network slice template version 9.0. https://www.gsma.com/solutions-and-impact/technologies/networks/gsma_resources/network-slicing-use-cases-requirements-2/, 2023. Use Case and SLA Templates.
- [8] Archive Market Research. 5g network slicing 2025-2033. <https://www.archivemarketresearch.com/reports/5g-network-slicing-364184>, May 2024.
- [9] Nokia. Network slicing. <https://www.nokia.com/network-slicing/>.
- [10] ONAP. Orchestration and automation of network slices. https://www.onap.org/wp-content/uploads/sites/20/2020/06/ONAP_Technical-Overview_061720.pdf, 2020. Whitepaper.
- [11] RADCOM. Ensure 5g network slicing quality with automated assurance. <https://radcom.com/ensure-5g-network-slicing-quality-with-automated-assurance>, 2025.

Referencias

- [12] E. Stefanski. 2025 wireless trends: Government initiatives, network slicing and more. <https://www.forbes.com/councils/forbestechcouncil/2025/02/10/2025-wireless-trends-government-initiatives-network-slicing-and-more/>, 2025.
- [13] 5g transport networks market report. <https://www.datainsightsmarket.com/reports/5g-transport-networks-466474>, 2024.
- [14] S. et al. Kukliński. Dynamic network slicing in 5g: Challenges and opportunities. <https://5g-drive.eu/wp-content/uploads/2021/09/a75-kukliński-paper.pdf>, 2021.
- [15] Network slicing advances slowly toward 2025. <https://www.fierce-network.com/wireless/network-slicing-slides-more-vigorously-2025>, 2025.
- [16] 5g network slicing market report. <https://www.snsinsider.com/reports/5g-network-slicing-market-4697>, 2024.
- [17] N. M. M. K. Chowdhury and R. Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.
- [18] 3GPP. Technical specification group services and system aspects; system architecture for the 5g system (5gs); stage 2. Technical Report TS 23.501, 3GPP, December 2019. Release 16.
- [19] Nokia. Transport slicing end-to-end 5g. <https://www.nokia.com/asset/206889>, 2019.
- [20] 3GPP. Management and orchestration; concepts, use cases and requirements. Technical Report TS 28.530, 3rd Generation Partnership Project (3GPP), 2023.
- [21] 3GPP. Procedures for the 5g system (5gs); stage 2. Technical Report TS 23.502, 3rd Generation Partnership Project (3GPP), 2023.
- [22] 3GPP. Policy and charging control framework for the 5g system. Technical Report TS 23.503, 3rd Generation Partnership Project (3GPP), 2023.
- [23] 3GPP. Management and orchestration; study on management and orchestration of network slicing for next generation network. Technical Report TS 28.801, 3rd Generation Partnership Project (3GPP), 2018.
- [24] 3GPP. Sa wg5 - management, orchestration and charging. <https://www.3gpp.org/3gpp-groups/service-system-aspects-sa/sa-wg5>.
- [25] K. Koutlia, D. Tsolkas, I. Samaras, P. Spapis, and N. Passas. Design and experimental validation of a software-defined ran testbed with slicing support. *arXiv preprint*, arXiv:1905.05729, 2019.
- [26] O-RAN Alliance. About o-ran alliance. Technical report, O-RAN Alliance.

- [27] A. Garcia-Saavedra and X. Costa-Perez. O-ran: Disrupting the virtualized ran ecosystem. *IEEE Communications Standards Magazine*, 5(4):96–103, Dec 2021.
- [28] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina. Network slicing in 5g: Survey and challenges. *IEEE Communications Magazine*, 55(5):94–100, 2017.
- [29] M. Richart, J. Baliosian, J. Serrat, and J.-L. Gorricho. Resource slicing in virtual wireless networks: A survey. *IEEE Transactions on Network and Service Management*, 13(3):462–476, 2016.
- [30] H. Zhang, N. Liu, X. Chu, K. Long, A. Aghvami, and V. C. M. Leung. Network slicing based 5g and future mobile networks: Mobility, resource management, and challenges. *IEEE Communications Magazine*, 55(8):138–145, 2017.
- [31] B. Martini, F. Paganelli, P. Cappanera, N. Bastianini, and P. Castoldi. Transport network slicing in 5g: Architecture, resource allocation, and challenges. *IEEE Communications Standards Magazine*, 2(3):60–67, 2018.
- [32] I. T. Haque, G. Marchetto, and R. Serral-Gracià. A survey of transport network slicing and future research directions. *IEEE Communications Surveys & Tutorials*, 23(2):1096–1128, 2021.
- [33] R. Muñoz, R. Vilalta, R. Casellas, R. Martínez, A. Mayoral, V. López, and D. King. Transport aspects of network slicing: Existing solutions and gaps. *IEEE Communications Standards Magazine*, 2(3):28–36, 2018.
- [34] M. Channegowda, D. Perez, R. Vilalta, R. Casellas, R. Martinez, V. Lopez, R. Munoz, and D. Lopez. Multi-domain and multi-vendor orchestration of end-to-end network slices. *IEEE Communications Magazine*, 55(8):116–123, 2017.
- [35] Ribbon Communications. Demystifying network slicing. <https://ribboncommunications.com/sites/default/files/2020-05/Demystifying-Network-Slicing-WP.pdf>, 2020.
- [36] Ericsson. Applied network slicing scenarios in 5g. Ericsson Technology Review, 2019.
- [37] Jia-You Lin, Ping-Hung Chou, and Ren-Hung Hwang. Dynamic resource allocation for network slicing with multi-tenants in 5g two-tier networks. *Sensors*, 23(10), 2023.
- [38] F. Song et al. Dynamic virtual resource allocation for 5g and beyond network slicing. *arXiv preprint*, 2020.
- [39] A. Esmaeily and K. Kralevska. Small-scale 5g testbeds for network slicing deployment: A systematic review. *Wireless Communications and Mobile Computing*, 2021.

Referencias

- [40] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges. *Computer Networks*, 167:106984, 2020.
- [41] ETSI. Open source mano (osm). <https://www.etsi.org/technologies/open-source-mano>. Available online.
- [42] The Linux Foundation. Onap — open network automation platform. <https://onap.org/>. Available online.
- [43] Rodrigo Moreira, Larissa F. Rodrigues Moreira, Tereza C. Carvalho, and Flávio de Oliveira Silva. Resource allocation influence on application performance in sliced testbeds. *arXiv preprint*, arXiv:2412.16716, 2024.
- [44] Diana Pineda, Ricardo Harrilal-Parchment, Kemal Akkaya, Ahmed Ibrahim, and Alexander Perez-Pons. Design and analysis of an open-source sdn-based 5g standalone testbed. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2023.
- [45] Alireza Gungor. Ueransim: Open source 5g ue and ran simulator. <https://github.com/aligungr/UERANSIM>. Available online.
- [46] Software Radio Systems. srsran project: Open source 4g/5g software radio suite. <https://www.srs.io/srsran>. Available online.
- [47] Linux Foundation. Onos: Open network operating system. <https://wiki.onosproject.org/>. Available online.
- [48] Linux Foundation. Opendaylight: Open source sdn controller platform. <https://www.opendaylight.org>. Available online.
- [49] Sukchan Lee and the Open5GS Community. Open5gs: Open source 5g core network project. <https://open5gs.org>. Available online.
- [50] OpenAirInterface Software Alliance. Openairinterface 5g radio access network project. <https://openairinterface.org/oai-5g-ran-project/>. Available online.
- [51] OpenInfra Foundation. Openstack – open source cloud computing infrastructure. <https://www.openstack.org/>.
- [52] The Linux Foundation. Open vswitch: An open virtual switch. Technical report. Available online.
- [53] Ali Esmaily, Katina Kravetska, and Danilo Gligoroski. A cloud-based sdn/nfv testbed for end-to-end network slicing in 4g/5g. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, 2020.
- [54] Qiang Ye, Junling Li, Kaige Qu, Weihua Zhuang, Xuemin Sherman Shen, and Xu Li. End-to-end quality of service in 5g networks: Examining the effectiveness of a network slicing framework. *IEEE Vehicular Technology Magazine*, 13(2):65–74, 2018.

- [55] Vaibhav Fanibhare, Nurul I. Sarkar, and Adnan Al-Anbuky. Tinets3: Sdn-driven network slicing enabling scenario-based applications in tactile internet. *IEEE Transactions on Network and Service Management*, 21(4):4639–4654, 2024.
- [56] R. Vilalta, R. Muñoz, R. Casellas, R. Martínez, F. Li, and P. Tang. Experimental validation of resource allocation in transport network slicing using the adrenaline testbed. *Photonics Netw. Commun.*, 40:82–93, 2020.
- [57] Centre Tecnològic de Telecomunicacions de Catalunya (CTTC). Adrenaline testbed: Experimental research testbed on high-performance and large-scale intelligent optical transport networks. <https://www.cttc.cat/adrenaline-testbed/>. Available online.
- [58] M. Rouili, N. Saha, M. Golkarifard, M. Zangoeei, R. Boutaba, E. Onur, and A. Saleh. Evaluating open-source 5g sa testbeds: Unveiling performance disparities in ran scenarios. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS) Workshops*, 2024.
- [59] OpenAirInterface. Rfsimulator. <https://gitlab.eurecom.fr/oai/openairinterface5g/-/blob/develop/radio/rfsimulator/README.md>. Available online4.
- [60] A. Chouman, D. M. Manias, and A. Shami. A modular, end-to-end next-generation network testbed: Towards a fully automated network management platform. *IEEE Transactions on Network and Service Management (submitted / preprint)*, 2024. Preprint (arXiv): <https://arxiv.org/abs/2403.15376>.
- [61] free5GC Project Contributors. free5gc: A free and open source 5g core network system. <https://www.free5gc.org/>. Available online.
- [62] Cloud Native Computing Foundation. Kubernetes – production-grade container orchestration. <https://kubernetes.io/>.
- [63] The Linux Foundation. Prometheus: Open-source systems monitoring and alerting toolkit. <https://prometheus.io/>. Available online.
- [64] 3GPP. Technical specification group services and system aspects; principles and guidelines for service-based interface (sbi) and application programming interface (api) specification (3gpp ts 29.501 version 16.8.0 release 16). Technical report, 3GPP, 2021.
- [65] OpenAirInterface Software Alliance. Openairinterface 5g core network (oai 5gc). <https://openairinterface.org/oai-5g-core-network-project/>. Available online.
- [66] 3rd Generation Partnership Project (3GPP). NR; NG-RAN; Architecture description (Release 18). Version 18.4.0.

Referencias

- [67] 3rd Generation Partnership Project (3GPP). NR; Physical channels and modulation (Release 18). Version 18.6.0.
- [68] Network functions virtualisation (nfv); architectural framework.
- [69] Broadcom. Vmware vsphere: Cloud infrastructure. <https://www.vmware.com/products/cloud-infrastructure/vsphere>. Available online.
- [70] Proxmox Server Solutions GmbH. Proxmox virtual environment (ve): Open source server virtualization platform. <https://www.proxmox.com/en/proxmox-ve>. Available online.
- [71] Open Networking Foundation. SDN Architecture. https://opennetworking.org/wp-content/uploads/2013/02/TR-522_SDN_Architecture_issue_1.1.1.pdf, June 2016. Available online.
- [72] Ryu SDN Framework Community. Ryu: Sdn framework. <https://ryu-sdn.org/>. Available online.
- [73] Antonio Hidalgo Pablo Bertrand. Testbed-5g-slicing-sdn: scriptscontrollerryu. <https://github.com/pablobertrand/Testbed-5G-Slicing-SDN/tree/main/scriptsControllerRyu>. Available online.
- [74] Open Networking Foundation. Openflow 1.3.4 switch specification. <https://opennetworking.org/wp-content/uploads/2013/07/OpenFlow1.3.4TestSpecification-Basic.pdf>. Available online.
- [75] The Linux Foundation. Ovsdb documentation. https://docs.opendaylight.org/_/downloads/ovsdb/en/latest/pdf/. Available online.
- [76] National Instruments. Usrp b210 sdr kit - dual channel transceiver (70 mhz-6 ghz). <https://www.ettus.com/all-products/ub210-kit/>. Ettus Research, a National Instruments Brand.
- [77] Inc. YouTube. Youtube. <https://www.youtube.com>. Available online.
- [78] Canonical Ltd. Canonical. <https://www.canonical.com>.
- [79] Maxime Labonne, Charalampos Chatzinakis, and Alexis Olivereau. Predicting bandwidth utilization on network links using machine learning. In *2020 European Conference on Networks and Communications (EuCNC)*, pages 242–247, 2020.
- [80] Anfal Musadaq Rayyis, Mohammad Maftoun, Maryam Khademi, Emrah Arslan, and Silvia Gaftandzhieva. Optimizing 5g resource allocation with attention-based cnn-bilstm and squeeze-and-excitation architecture. *Frontiers in Communications and Networks*, 2025.
- [81] Md. A. Khudhus, M. Humera Khanam, D. Vivekanandha Reddy, G. Keerthi, P. Srujigna, and P. Shanmuga Teja. Qos prediction of 5g network slices using ml models. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 2025.

- [82] Jinhe Zhou, Wenjun Zhao, and Shuo Chen. Dynamic network slice scaling assisted by prediction in 5g network. *IEEE Access*, 2020.
- [83] National Institute of Standards and Technology. What is exponential smoothing? <https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm>. Available online.
- [84] Pablo Bertrand, Claudina Rattaro, Antonio Hidalgo, David Cortés-Polo, and Jesús Calle-Cancho. Sdn-based slicing testbed for 5g networks. In *2025 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2025.
- [85] David Cortes-Polo, Antonio Hidalgo, Jesus Calle-Cancho, Pablo Bertrand, and Claudina Rattaro. Arquitectura para el slicing dinamico en redes privadas 5g. In *XVII Jornadas de Ingeniería Telemática (JITEL 2025)*, España, November 2025. Universidad de Extremadura.
- [86] GNU Radio. Gnu radio project. <https://www.gnuradio.org>. Available online.
- [87] VideoLAN. Vlc media player. <https://www.videolan.org/vlc/>. Available online.
- [88] yt-dlp contributors. yt-dlp: A command-line audio/video downloader. <https://github.com/yt-dlp/yt-dlp>. Available online.
- [89] FFmpeg project. Ffmpeg multimedia framework. <https://ffmpeg.org/>. Available online.
- [90] Inc. Apple. Rfc 8216: Http live streaming. <https://www.rfc-editor.org/rfc/rfc8216>, 2017. Available online.
- [91] Keras. Introduction to keras for engineers. https://keras.io/getting_started/intro_to_keras_for_engineers/. Available online.
- [92] Keras. Adam optimizer. <https://keras.io/api/optimizers/adam/>, 2023. Available online.
- [93] Keras. Relu activation layer. https://keras.io/api/layers/activation_layers/relu/. Available online.
- [94] Ryan Tibshirani. *Conformal Prediction*. Department of Statistics, University of California, Berkeley, 2023. Lecture notes for Advanced Topics in Statistical Learning, Spring 2023.
- [95] Quantmetry. Mapie: Model-agnostic prediction interval estimator in python. <https://mapie.readthedocs.io/>, 2024.
- [96] Kuan-Lin Lee, Chung-Nan Lee, and Ming-Feng Lee. Realizing 5g network slicing provisioning with open source software. In *2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSI-PA ASC)*, pages 1923–1930, 2021.

Referencias

- [97] s5uishida. Open5gs 5gc & ueransim ue / ran sample configuration - select upf based on s-nssai. https://github.com/s5uishida/open5gs_5gc_ueransim_snssai_upf_sample_config. GitHub repository.

Índice de tablas

3.1.	Comparación de testbeds revisados con respecto a este trabajo. . .	30
4.1.	Listado de herramientas y versiones seleccionadas para el testbed por tecnología.	48
6.1.	Lista de VMs y sus atributos en Proxmox VE	70
6.2.	Comparación de los resultados obtenidos con distintas funciones de pérdida durante el entrenamiento del modelo LSTM. Se incluyen el RMSE, el bitrate promedio asignado y el índice de subpredicción para funciones simétricas y asimétricas, mostrando el compromiso entre ajuste, aprovechamiento de recursos y garantía de servicio. .	91

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

1.1. Arquitectura general de la red móvil: el dispositivo móvil se conecta por el medio inalámbrico a la red de acceso, que llega al núcleo de la red mediante la red de transporte.	1
2.1. Jerarquía de gestión de <i>network slicing</i> en 5G, en la que se establecen las relaciones entre NSI, NSSI, orquestación, funciones de red y servicios.	10
2.2. Fases del ciclo de vida de un <i>network slice</i> , desde su creación hasta su eliminación.	13
4.1. Arquitectura de funciones de red del núcleo 5G, incluyendo las funciones del plano de control, las réplicas de funciones de plano de usuario para los distintos slices y las conexiones con la red radio y redes externas.	32
4.2. Flujo para el establecimiento de una sesión PDU en 5G.	36
4.3. Arquitectura general de RAN en 5G: desagregación del gNB en unidades centralizada, distribuida y de radio, con sus conexiones hacia el núcleo de la red (NG-C y NG-U) y hacia los dispositivos (inalámbrica).	39
4.4. Procedimiento de acceso inicial de un UE en la RAN 5G.	41
5.1. Arquitectura inicial del testbed: un dispositivo real conectado a srs-RAN por medio de SDR, dos dispositivos virtuales conectados a un gNB virtual y dos planos de usuario que definen los dos slices. . . .	50
5.2. Ejecución de tcpdump en ambos slices. El izquierdo (UPF 1) ve pasar el tráfico desde y hacia el UE (con IP 10.45.0.2) mientras que el derecho (UPF 2) solo vio tráfico administrativo tres minutos antes.	56
5.3. Tráfico y bitrate máximo asignado por umbrales al slice prioritario.	57
5.4. Tráfico y bitrate asignado por LSTM al slice prioritario, muestreando cada 5 segundos.	60
5.5. Tráfico y bitrate asignado por LSTM al slice prioritario, muestreando cada 0,5 segundos.	61
6.1. Red a simular con escalamiento: el tráfico de los tres slices que sale del núcleo se encamina por el switch a cada uno de los tres sitios de radio para atender a cada servicio en cada sitio.	64

Índice de figuras

6.2. Arquitectura final del testbed: se incorpora una tercera UPF con su SMF, dos gNBs y varios UE virtualizados. Tres gNBs con tres UPFs definen nueve flujos diferenciados de slice.	65
6.3. Interfaz gráfica del núcleo Open5GS con los dispositivos añadidos.	69
6.4. Fragmento del dataset del slice de video, muestreado cada un segundo.	77
6.5. Fragmento del dataset del slice de audio, muestreado cada un segundo.	78
6.6. Comparación de las predicciones del modelo inicial contra los datos de validación en función del tiempo.	80
6.7. Scatter de las predicciones del modelo inicial contra los datos de validación.	81
6.8. Comparación de las predicciones del modelo con función de pérdida cuadrática asimétrica contra los datos de validación en función del tiempo.	82
6.9. Comparación de las predicciones del modelo con función de pérdida lineal asimétrica contra los datos de validación en función del tiempo.	83
6.10. Comparación de las predicciones del modelo con función de pérdida asimétrica (cuadrática con factor 3, lineal con factor 1) contra los datos de validación en función del tiempo.	84
6.11. Scatter de las predicciones del modelo con función de pérdida asimétrica (cuadrática con factor 3, lineal con factor 1) contra los datos de validación.	85
6.12. Comparación de las predicciones del modelo con función de pérdida asimétrica (cuadrática con factor 2, lineal con factor 1) contra los datos de validación en función del tiempo.	86
6.13. Comparación de las predicciones del modelo con función de pérdida mse y percentil conformal fijo contra los datos de validación en función del tiempo.	89
6.14. Impresión de flujos en la interfaz de OVS. Todos mantienen cero bytes de tráfico, a excepción del flujo correspondiente a la prueba y la regla de tráfico genérico.	92
A.1. Menú de configuración de Ubuntu Server.	101
A.2. Menú de configuración de Ubuntu Server.	102

Esta es la última página.
Compilado el martes 3 febrero, 2026.
<http://iie.fing.edu.uy/>