Universidad Zaragoza
1542

Trabajo Fin de Grado

Integración de funcionalidades avanzadas de QoS en un entorno SDWLAN

Integration of QoS advanced functionalities in a SDWLAN environment

Autor

Sergio Espinosa Fumanal

Director

José Ruiz Mas

Departamento de Ingeniería Electrónica y Comunicaciones

Escuela de Ingeniería y Arquitectura

Universidad de Zaragoza

2025

A mi familia,
por su constante apoyo y confianza,
por enseñarme el valor del esfuerzo y acompañarme en cada paso.

A mis amigos,
porque no todo es trabajar ni estudiar: siempre queda tiempo para quedar, hacer deporte y tomar algo, y con vosotros todo eso sabe mejor.

A Cristina,
que me apoya incondicionalmente, me motiva cuando es necesario y apaga los fuegos cuando no todo va bien. Eres lo que me empuja a seguir adelante sabiendo que siempre habrá alguien a mi lado.

A mis compañeros,
que me han demostrado que lo que no saca uno solo, lo sacamos todos juntos. Porque no han faltado buenos momentos, dentro y fuera de clase.

A mis profesores,
en quienes he encontrado la motivación de aprender para resolver mis dudas sobre lo que, desde niño, siempre me ha parecido magia. Especialmente a Ángela y Pepe, por vuestra dedicación diaria y preocupación.

Por último, a mis abuelos,
que, estén o no presentes, me enseñan el valor de la vida y en quienes encuentro el reflejo de lo que realmente es el éxito.

Y a todas las personas que, de un modo u otro,
han formado parte de este camino.

RESUMEN

Actualmente, es prácticamente inimaginable una vida sin conexión a Internet. Con la aparición de nuevas tecnologías y el aumento del número de dispositivos electrónicos surge la necesidad de evolucionar los medios de acceso a Internet para mejorar la experiencia de usuario al mismo tiempo que los servicios consumidos se vuelven más exigentes. Más concretamente, son las conexiones inalámbricas las que han experimentado una evolución significativa en los últimos años, permitiendo mayor velocidad, menor latencia y una mejor eficiencia en la gestión de los recursos de red. Tecnologías como el avance hacia Redes Definidas por Software (*Software Defined Network, SDN*) y Virtualización de Funciones de Red *(Network Functions Virtualization, NFV*) han transformado la manera en que se gestionan y optimizan las infraestructuras inalámbricas. Bajo este contexto, surge la necesidad de desarrollar soluciones avanzadas que permitan mejorar la asignación de recursos en redes WLAN, garantizando calidad de servicio (QoS, *Quality of Service*) y eficiencia en entornos con alta demanda de conectividad.

Por ello, se plantea la incorporación de soluciones avanzadas en un entorno de Redes de Área Local Inalámbricas Definidas por Software (*Software-Defined Wireless Local Area Network, SDWLAN*) dentro de un proyecto de investigación centrado en la optimización de redes *WLAN* coordinadas basadas en arquitecturas programables.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

LISTA DE ACRÓNIMOS

ACK	*Acknowledge* frame*,* trama Wi-Fi
AMSDU	*Aggregated MAC Service Data Unit*
AP	*Access Point*
BSSID	*Basic Service Set Identifier*
CSA	*Channel Switch Announcement*
CSMA/CA	*Carrier Sense Multiple Access with Collision Avoidance*
DIFS	*Distributed Interframe Space*
DSCP	*Differentiated Services Code Point*, campo de la cabecera IP
GRE	*Generic Routing Encapsulation*
LVAP	*Light Virtual Access Point*
MBR	*Maximum Bit Rate*
MCS	*Modulation Coding Scheme*
MSDU	*MAC Service Data Unit*
NFV	*Network Functions Virtualization*
QoS	*Quality of Service*
RSSI	*Received Signal Strength Indicator*
SDN	*Software Defined Network*
SDWLAN	*Software Defined Wireless Local Area Network*
SDWN	*Software-Defined Wireless Network*
SIFS	*Short Interframe Space*
SLA	*Service Level Agreement*
SSID	*Service Set Identifier*
ToS	*Type of Service*, campo de la cabecera IP
WLAN	*Wireless Local Area Network*

GLOSARIO

Airtime	Tiempo de ocupación del medio inalámbrico durante una transmisión (o tiempo dado para realizar transmisiones).
Déficit	Tiempo (μs) que puede utilizar un *slice*/cola para transmitir
Handover	Proceso donde un cliente Wi-Fi cambia de un punto de acceso a otro, manteniendo su conexión y logrando una movilidad transparente sin interrupciones.
Nodo	Equipo físico donde se despliega uno o varios *pods*
Pesos	Ponderación usada por las colas para repartirse el *quantum* de un *slice*.
Pod	En *Kubernetes*, conjunto de contenedores que comparten recursos de red y almacenamiento
Quantum	Cantidad fija de tiempo (μs) que se añade periódicamente al déficit de un *slice*/cola garantizando tiempo de transmisión.
Scheduling	Algoritmos que definen cómo se extraen paquetes de las colas.
Slice	Porción o partición lógica de los recursos del punto de acceso para asignar recursos específicos a conjuntos de tráficos.
Thread	Hilo de ejecución que opera en paralelo al código principal
Throughput	Tasa de transmisión de datos efectiva lograda, fundamental para medir el rendimiento del sistema. Se mide en Mbps.

1. INTRODUCCIÓN

1.1 CONTEXTO Y UBICACIÓN DEL TRABAJO

El crecimiento de tecnologías emergentes y el aumento de dispositivos conectados han impulsado la evolución de las redes inalámbricas, lo que ha exigido nuevas soluciones que mejoren la eficiencia y la calidad del servicio. En este contexto, las arquitecturas basadas en SDN (*Software Defined Network*) y NFV (*Network Functions Virtualization*) han adquirido protagonismo como herramientas clave para optimizar la gestión de recursos en entornos WLAN (*Wireless Local Area Network*) cada vez más complejos.

En este trabajo se plantea la integración de distintas soluciones en un escenario *SDWN* (*Software-Defined Wireless Network*) como parte de un proyecto de investigación más amplio centrado en la optimización de redes *WLAN* coordinadas basadas en arquitecturas programables (*Software-Defined Wireless Local Area Network*, SDWLAN). En trabajos anteriores [1][2][3], se han abordado soluciones tanto desde el punto de vista de arquitecturas de red programables y virtualizadas, como soluciones específicas que incorporan funcionalidades básicas, hasta llegar a abordar soluciones más avanzadas relacionadas con la asignación de recursos de red y garantía de QoS. La finalidad de este trabajo es integrar estas funcionalidades, afrontando los desafíos inherentes al despliegue del escenario de trabajo y operación conjunta en un entorno real.

1.2 OBJETIVOS

El objetivo principal de este Trabajo Fin de Grado es integrar dos funcionalidades avanzadas, el algoritmo de control de tasa *Onoe* y un sistema de *slicing*, en un entorno *SDWLAN*. Esta integración busca mejorar la gestión de recursos inalámbricos y acercar el sistema a condiciones reales de funcionamiento, superando las limitaciones de entornos experimentales anteriores.

Para alcanzar este objetivo general, se establecen los siguientes objetivos específicos:

- Analizar y evaluar soluciones *SDWLAN* existentes, especialmente aquellas desarrolladas previamente en proyectos anteriores, con el fin de comprender su

arquitectura, sus ventajas y sus limitaciones.

- Comprender el funcionamiento de las funcionalidades a integrar, es decir, el algoritmo de control de tasa *Onoe* y el sistema de *slicing*, analizando tanto su diseño como su comportamiento esperado.
- Desplegar un entorno funcional *contenerizado*, en una infraestructura basada en *Kubernetes*, que permita aplicar las funcionalidades mencionadas sobre puntos de acceso reales.
- Implementar y adaptar el algoritmo de control de tasa *Onoe*, permitiendo que se ejecute en un entorno con múltiples dispositivos conectados y adaptándose dinámicamente a las condiciones del canal inalámbrico.
- Diseñar e integrar un sistema de *slicing* que gestione el *airtime*, garantizando una asignación justa y configurable de los recursos entre diferentes flujos o dispositivos conectados al punto de acceso.
- Identificar y mitigar conflictos o ajustes derivados de la convivencia entre ambas funcionalidades, garantizando que la integración conjunta de control de tasa y *slicing* no interfiera negativamente en el rendimiento global del sistema.
- Realizar pruebas experimentales en un entorno real, observando el comportamiento de la solución desplegada y evaluando su efectividad en diferentes situaciones de carga y uso.

En conjunto, este trabajo pretende continuar la investigación en redes SDN para entornos Wi-Fi para que pueda servir como base para futuras mejoras y funcionalidades más complejas.

1.3 HERRAMIENTAS

A continuación, se detallan las herramientas utilizadas en este trabajo.

1.3.1 Kubernetes

Kubernetes es un sistema de orquestación de contenedores diseñado para automatizar el despliegue y la gestión de infraestructuras de redes. En este proyecto, se emplea para construir la infraestructura SDWLAN, gestionando la ejecución de *pods* (unidad de despliegue más pequeña en *Kubernetes*) dentro de nodos (máquinas físicas donde se ejecutan los *pods*).

En este caso, los nodos son ordenadores de propósito general equipados con los periféricos necesarios, como las tarjetas Wi-Fi. Por lo tanto, *Kubernetes* facilita el despliegue dinámico de los *pods* requeridos en el escenario de trabajo (por ejemplo, un punto de acceso, un *router* o un controlador) y su eliminación cuando ya no sean necesarios.

Es importante destacar que los elementos clave del escenario, como los puntos de acceso y los *routers*, no son dispositivos físicos independientes, sino unidades de software (*pods*) desplegadas en nodos físicos que comparten recursos con el resto de *pods* desplegados en el nodo.

1.3.2 Docker

Docker es una herramienta que permite *empaquetar* aplicaciones junto con todas sus dependencias en *contenedores*, asegurando un funcionamiento consistente en cualquier entorno. Al ser una solución portátil, resulta de gran utilidad para utilizar y replicar la aplicación de manera remota en varios equipos simultáneamente.

En este proyecto, *Docker* se ha utilizado para *contenerizar* cada uno de los elementos de la red de nuestro escenario de trabajo. Principalmente, se ha trabajado en la *contenerización* del código en C que compone el punto de acceso *OdinAP*, para facilitar su ejecución en diferentes equipos.

Gracias a este enfoque, el despliegue del *software* es más ágil y escalable, permitiendo que se ejecute simultáneamente en varios dispositivos de manera eficaz.

1.3.3 Iperf

Para la generación de tráfico en el entorno Wi-Fi se ha utilizado *Iperf*, un *software* de *Linux* que posibilita la generación de tráfico desde un equipo a otro a partir de la dirección IP.

En este trabajo se utilizan opciones que permiten personalizar la generación de tráfico como la duración del flujo, el tamaño de los paquetes generados, la velocidad de transmisión y el puerto destino. La opción que permite especificar el puerto destino hará que, con la herramienta *iptables,* se puedan marcar diferentes tipos de tráfico usando el campo DSCP (*Differentiated Services Code Point*) de la cabecera IP.

1.3.4 Iptables

Iptables es la herramienta de *Linux* que se ha utilizado para asignar un valor determinado a DSCP. Para ello se utiliza la tabla *mangle*, con la que es sencillo modificar los paquetes que pasen por la interfaz de red de un dispositivo (para marcarlos con un valor determinado).

1.3.5 Matlab

Para llevar a cabo una valoración de los resultados y obtener unas medidas precisas se utiliza *Matlab* como herramienta de creación de representaciones gráficas. Con ella se analizan principalmente aspectos como el *throughput*, el porcentaje de *airtime* (referido al tiempo empleado para la transmisión de la información en el medio radio), el número de paquetes transmitidos, etc.

Es posible crear las gráficas mencionadas anteriormente gracias a la exportación de los resultados recopilados tras la ejecución de *Iperf*. Estos resultados se exportan en archivos de texto y son procesados a posteriori para observar cómo ha funcionado la prueba.

1.3.6 Wpa_Supplicant

Wpa_Supplicant es una herramienta de *software* utilizada principalmente en sistemas operativos *Linux* para gestionar la conexión de clientes Wi-Fi (STAs). Funciona como un cliente de asociación y autenticación que se puede ejecutar en segundo plano, encargándose de establecer y mantener la conexión entre el dispositivo y un punto de acceso inalámbrico.

Además, esta herramienta permite configurar los parámetros de la red con la que se va a establecer la conexión, permitiendo especificar el SSID (*Service Set Identifier*), la contraseña, el método de autenticación, etc.

En entornos de desarrollo o pruebas como SDWN, su presencia es esencial para conseguir el comportamiento de un cliente real que se conecta a la red desplegada.

1.3.7 Wireshark

Wireshark es una herramienta de análisis de protocolos de red ampliamente utilizada para capturar y examinar en detalle el tráfico que circula por una interfaz de red.

Permite observar y clasificar los paquetes según su protocolo, dirección de origen y destino, número de secuencia, tiempos de transmisión, entre muchos otros parámetros.

Concretamente, en este proyecto *Wireshark* se ha utilizado para registrar y analizar el comportamiento del sistema durante las pruebas de validación. La herramienta ha permitido obtener métricas relevantes como la tasa de transmisión de paquetes, la presencia de retransmisiones o la variación del tráfico en diferentes momentos de la ejecución. Estos datos han sido fundamentales para evaluar el rendimiento global del sistema y comprobar el impacto de las modificaciones implementadas en distintos escenarios de prueba.

1.4 ESTRUCTURA DE LA MEMORIA

En este primer capítulo se han introducido el contexto en el que se enmarca este TFG y los objetivos que se persiguen.

El capítulo 2, *Estado del arte - Antecedentes*, comienza con una visión general de la tecnología de red Wi-Fi y la infraestructura SDN a desarrollar, describiendo la evolución de su arquitectura e identificando sus partes fundamentales. Posteriormente, se explican detalladamente las funcionalidades de control de tasa (*Onoe*) y de segmentación de red (diferenciación de tráficos) a integrar en nuestro entorno de red Wi-Fi virtualizado.

El capítulo 3, *Escenario y aspectos de la integración*, describe el escenario de trabajo tanto desde un punto de vista físico como lógico, resaltando los aspectos más relevantes a considerar en la integración de las nuevas funcionalidades. Se abordan aspectos tales como la modificación del llenado del *buffer* del *kernel* (control de tasa), la aparición de nuevos tráficos o los efectos del *handover* en las funcionalidades integradas, todos relacionados con el hecho de trabajar en un entorno experimental que incorpora elementos de red Wi-Fi reales.

En el capítulo 4, *Pruebas y resultados*, se presentan las pruebas realizadas en nuestro entorno experimental real y los resultados obtenidos que validan la integración realizada. Finalmente, esta memoria concluye con un capítulo 5 de *Conclusiones y líneas futuras* propuestas a partir de este trabajo.

También se incluyen dos anexos que contienen información complementaria. El

Anexo A aporta información de índices de MCS (*Modulation Coding Scheme*) a considerar en las distintas versiones Wi-Fi y en el control de tasa aplicado. El Anexo B contiene información sobre los algoritmos de *scheduling* empleados en la parte de segmentación de red y diferenciación de tráficos.

2. ESTADO DEL ARTE - ANTECEDENTES

Este capítulo presenta los fundamentos teóricos y las soluciones existentes que sirven de base para el desarrollo del presente Trabajo Fin de Grado. Se abordarán los estándares de redes inalámbricas, las arquitecturas de Redes Definidas por Software para WLAN, y los algoritmos específicos de control de tasa y segmentación de red.

2.1 IEEE 802.11

El estándar IEEE 802.11 establece las especificaciones para redes inalámbricas de área local (WLAN). Desde su introducción en 1997, ha evolucionado para responder a las crecientes demandas de conectividad, ofreciendo mayor velocidad, capacidad y eficiencia. Este estándar opera en bandas de frecuencia como 2.4 GHz, 5 GHz y, más recientemente, 6 GHz, empleando técnicas de modulación como OFDM (*Orthogonal Frequency Division Multiplexing*) para optimizar la transmisión de datos en el medio inalámbrico. En este trabajo, se ha utilizado específicamente la versión 802.11n, conocida como *Wi-Fi 4*, operando en las bandas de 2.4 GHz y 5 GHz, aunque el sistema también es compatible con 802.11ac (*Wi-Fi 5*). Esta versión introduce mejoras como canales más anchos (hasta 40 MHz, y en versiones posteriores como 802.11ac, hasta 160 MHz) y MIMO (*Multiple Input Multiple Output*), permitiendo mayores tasas de transmisión y una mejor gestión del espectro.

El protocolo de acceso al medio (MAC) de 802.11 se basa en CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*), que coordina el acceso al canal mediante escucha previa y tiempos de espera aleatorios (*backoff*) para minimizar colisiones. El *airtime*, o tiempo de ocupación del medio inalámbrico, es un factor crítico en este contexto, ya que depende de la tasa de transmisión (definida por el esquema de modulación y codificación, MCS), las interferencias y los tiempos asociados al protocolo MAC. En entornos con alta densidad de dispositivos o tráfico variable, la gestión del

airtime puede volverse compleja, afectando al rendimiento y a la latencia de la red. Estas características son especialmente relevantes en aplicaciones que requieren garantía de QoS, como el *streaming* o las comunicaciones en tiempo real.

Versiones recientes como 802.11ax (*Wi-Fi 6*) incorporan avances como OFDMA (*Orthogonal Frequency Division Multiple Access*) y MU-MIMO (*Multi-User Multiple Input Multiple Output*) para mejorar la eficiencia y atender simultáneamente a múltiples dispositivos. A pesar de estos avances en recientes versiones Wi-Fi, la utilización de redes virtualizadas SDN sigue siendo limitada en arquitecturas basadas en 802.11. De ahí que surja la necesidad de explorar soluciones avanzadas Wi-Fi basadas en entornos virtualizados y *softwarizados*.

Para ello, es necesario contar previamente con funcionalidades básicas como el control de tasa para optimizar el uso del *airtime* al adaptar dinámicamente el MCS a las condiciones del canal, así como lograr una mayor eficiencia en la transmisión. A partir de ahí se pueden plantear funcionalidades avanzadas como el *slicing* que permitan atender con mayor prioridad a ciertos flujos que lo requieran, mejorando la asignación de dicho *airtime* según las necesidades específicas de los usuarios.

Estas funcionalidades tienen el objetivo de optimizar la asignación de recursos en la red, garantizando una distribución eficiente del tiempo de ocupación del medio inalámbrico (*airtime*) desde el AP (*Access Point*) hasta las estaciones conectadas (STAs). Es decir, el enfoque del trabajo se centrará en los flujos de tráfico *downlink*, cuyo *airtime* se calcula según lo indicado en la Ecuación 1.

$$Airtime = \overline{n_{Tx}} \cdot (\overline{T_{Backoff}} + T_{DIFS} + T_{DATA} + T_{SIFS} + T_{ACK})$$

Ecuación 1. Cálculo de Airtime medio

Más detalladamente, el *airtime* es el tiempo que requiere una trama *unicast* para enviarse por la interfaz inalámbrica y recibir la confirmación del ACK, teniendo en cuenta el número medio de retransmisiones.

Como se muestra en la Ecuación 2, el parámetro $\overline{n_{Tx}}$ (número medio de transmisiones, incluyendo la transmisión original) se calcula como la inversa de la probabilidad de recibir el ACK a una tasa determinada. Esta probabilidad dependerá de

la tasa; cuanto mayor sea la tasa, menor será la probabilidad de ACK y, cuanto menor sea la tasa, mayor será la probabilidad de ACK.

$$\overline{n_{Tx}} = 1/p_{ACK}(R)$$

Ecuación 2. Cálculo del número medio de transmisiones

El resto de los tiempos de la *Ecuación 1* pertenecen al protocolo de acceso a la red CSMA/CA: $T_{Backoff}$ es un tiempo aleatorio previo a la transmisión, T_{DIFS} es el *Distributed InterFrame Space*, T_{DATA} es el tiempo de transmisión que depende de la tasa de transmisión utilizada, T_{SIFS} es el *Short InterFrame Space* y T_{ACK} es el tiempo de transmisión de la trama ACK.

De esta forma, el sistema será capaz de adaptarse dinámicamente a las condiciones del canal modificando los parámetros característicos de las transmisiones inalámbricas que determinan la velocidad de transmisión. Al mismo tiempo, el sistema también podrá diferenciar entre distintas clases de tráfico y usuarios con el objetivo de repartir el *airtime* según criterios de prioridad de usuarios, inelasticidad de tráficos, etc.

2.2 ARQUITECTURAS SDN PARA REDES WLAN

2.2.1 APs virtuales y arquitecturas SDWN

Como se menciona en *[4],* SDN es un paradigma de diseño de redes que separa el plano de control del plano de datos, lo que permite una gestión centralizada, programabilidad y automatización. Esta separación mejora la agilidad, la escalabilidad y la eficiencia operativa, al permitir que el control de la red sea gestionado de forma lógica desde un único punto mediante *software*.

Una arquitectura SDN suele estar compuesta por tres planos principales: el plano de aplicación, el plano de control y el plano de datos. En el centro de esta arquitectura se encuentra un controlador centralizado, encargado de tomar decisiones sobre el funcionamiento de la red. Este controlador se comunica con los dispositivos del plano de datos mediante protocolos estandarizados como *OpenFlow*. Gracias al desacoplo entre el control y el envío de datos, el plano de control se encarga de la lógica y la toma de decisiones, mientras que el plano de datos se dedica exclusivamente al envío de paquetes.

Además, SDN aporta programabilidad a la red al abstraer el *hardware* subyacente y permitir el control a través de APIs. Las APIs *northbound* (que conectan el plano de control con el plano de aplicación) permiten que las aplicaciones definan políticas de alto nivel sobre el comportamiento de la red, mientras que las APIs *southbound* (que conectan el plano de control con el plano de datos) facilitan la interacción del plano de control con los dispositivos físicos, asegurando la correcta ejecución de instrucciones.

Suresh et al. [5] propuso *Odin*, una arquitectura SDWN diseñada para redes WLAN empresariales, que introduce el concepto de puntos de acceso virtuales para simplificar el desarrollo de aplicaciones y gestionar la movilidad de las estaciones (STAs) entre diferentes APs físicos. Esto se consigue mediante la abstracción denominada LVAP (*Light Virtual Access Point*). Cuando una STA se conecta por primera vez, el controlador SDN le asigna un LVAP único, en lugar de asociarse directamente a un AP físico. Este LVAP contiene un SSID virtual, un BSSID (*Basic Service Set Identifier*) virtual, la dirección MAC de la STA y su dirección IP.

Gracias a esta abstracción, cuando la STA se mueve fuera del alcance del AP, el controlador migra dinámicamente el LVAP a otro punto de acceso, manteniendo la misma identidad de red (SSID y dirección IP). Esto permite una movilidad totalmente transparente para la STA, sin necesidad de realizar procesos de reasociación o reautenticación.

Sin embargo, la arquitectura *Odin* propuesta en [5] tiene dos limitaciones principales: en primer lugar, asume que todos los APs operan en el mismo canal, lo que impide una planificación de canales eficaz y, en segundo lugar, se enfrenta a problemas de escalabilidad, ya que no es posible utilizar tramas *beacon* de difusión. En su lugar, el punto de acceso debe enviar tramas *beacon unicast* con una dirección MAC específica a cada STA (cada LVAP solo puede atender a una STA). Además, el punto de acceso debe ser capaz de generar los ACK (trama *Acknowledge*) Wi-Fi correspondientes para cada STA.

Para superar estas limitaciones y siguiendo este paradigma de AP virtual, en los últimos años se han propuesto varias propuestas de arquitectura para SDWN. La arquitectura *EmPOWER* introducida en [6] integra múltiples tecnologías de acceso radio y proporciona un conjunto de abstracciones de programación para modelar aspectos clave

de las redes inalámbricas tales como la movilidad y/o el control de tasa [7][8], pero sin abordar sus problemas de escalabilidad Wi-Fi. Ante esta limitación, la arquitectura *BIGAP* [9] propone utilizar un único BSSID global compartido por todos los APs de un mismo conjunto de servicio extendido (ESS). Desde el punto de vista de la STA, toda la red se percibe como si fuera un único punto de acceso, lo que reduce la sobrecarga de señalización y mejora la escalabilidad tanto en número de usuarios como en densidad de APs. Soluciones más recientes basadas en SDN [10][11][12] integran mecanismos de traspaso y soporte multicanal con herramientas de supervisión y otras funcionalidades inteligentes. Ejemplo de ello son el uso del CSA (*Channel Switch Announcement)* y el estándar IEEE 802.11h, lo que permite que los APs trabajen en canales distintos sin perder sincronización.

Cada una de las propuestas mencionadas emplea un controlador central responsable de la gestión de la red. Sin embargo, todas ellas asumen que los elementos de red, como los AP y los *routers* residen en la misma red física. Además, no abordan la gestión basada en SDN del segmento cableado, aspecto crítico cuando se desea que estos elementos se distribuyan en distintas ubicaciones físicas. Este control del segmento cableado basado en SDN debe permitir crear y gestionar la comunicación entre los puntos de acceso y el *router* a partir de túneles GRE (*Generic Routing Encapsulation*) cuando estos elementos están distribuidos en distintas ubicaciones físicas que pertenecen a redes IP distintas.

Nuestra arquitectura SDWN denominada *NeWLAN*, punto de partida y soporte de este trabajo, forma parte de la evolución descrita incorporando las mejoras mencionadas anteriormente, integrando mecanismos de movilidad avanzada, puntos de acceso virtuales, reducción de la sobrecarga de señalización mediante *BIGAP*, y control unificado tanto del segmento inalámbrico como del cableado. Esta combinación de elementos permite ofrecer una solución escalable, flexible y fácilmente desplegable para entornos reales con múltiples dispositivos y puntos de acceso distribuidos.

2.2.2 Arquitectura SDWN *NeWLAN*

Las principales características y novedades de la propuesta *NeWLAN* (Fig. 1) se resumen en esta sección. Estas incluyen la gestión basada en SDN del plano de datos dentro del segmento cableado de la red, la eliminación de la necesidad de infraestructura de red dedicada o servidores exclusivos, y el aprovechamiento de redes superpuestas

(*overlay networks*) e implementaciones *contenerizadas* para lograr una mayor flexibilidad y eficiencia. Asimismo, abarca la gestión de movilidad para soportar traspasos (*handovers*) dentro de la red de acceso Wi-Fi.

En trabajos anteriores, se buscaba integrar mecanismos de coordinación que mejoraran las capacidades de los puntos de acceso (APs) Wi-Fi gestionados de forma centralizada. Para ello se incluyeron herramientas de monitorización y funcionalidades adicionales que permitían una gestión de red inteligente utilizando APs comerciales de bajo coste. Al aprovechar la información de red recopilada por los controladores centrales, se demostró el potencial para tomar decisiones informadas sobre la asignación de recursos. Por tanto, para un funcionamiento adecuado, es esencial usar dos redes separadas: una para el control y otra para los datos.

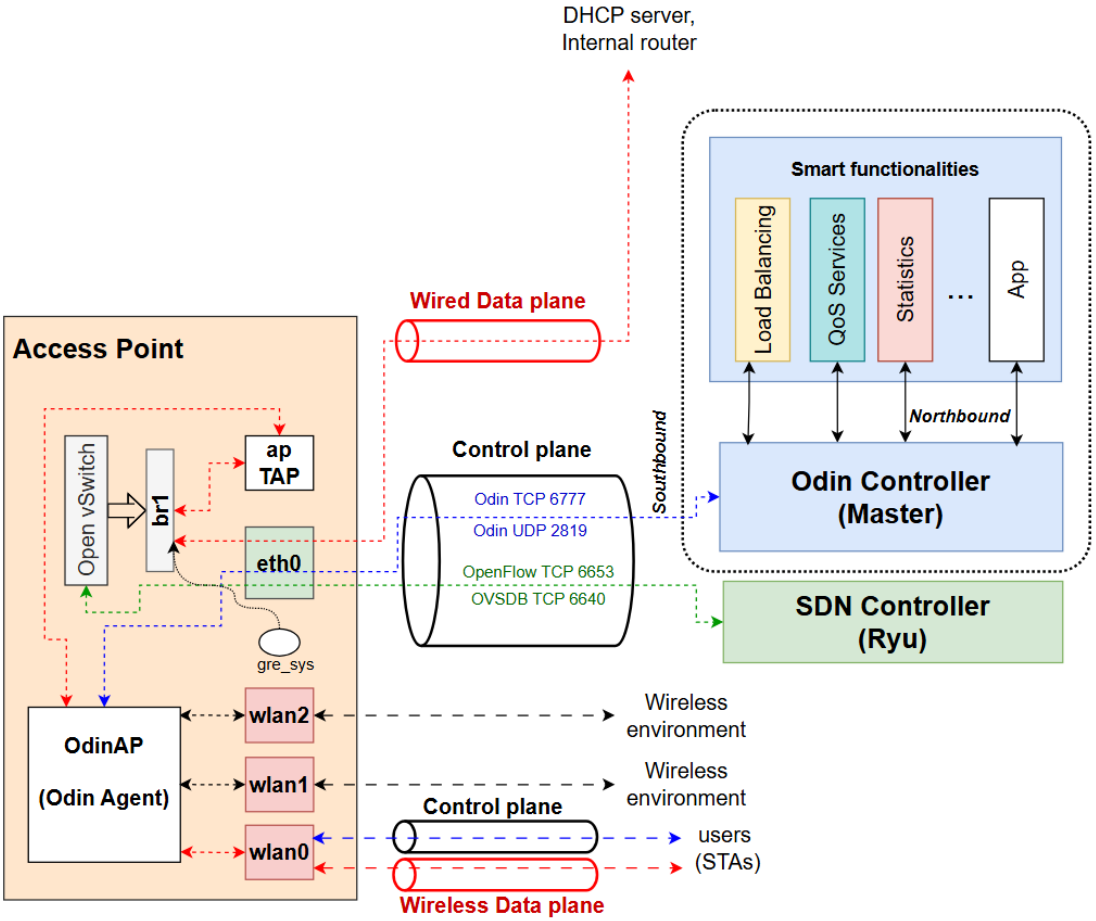


Figura 1. Componentes funcionales y protocolos. AP virtual y controladores

El plano de control se encarga de la funcionalidad radio y configura la conmutación y el encaminamiento dentro del plano de datos (Fig. 2). Para esto, se utilizan dos controladores centrales: un controlador *OpenFlow* estándar (basado en *Ryu*) para configurar el segmento cableado de la ruta de datos, y un controlador personalizado, *Odin*, para gestionar el segmento inalámbrico. El controlador *OpenFlow* supervisa los *switches* virtuales desplegados a lo largo de la ruta de datos, mientras que el controlador *Odin* interactúa con los *Agentes Odin* ubicados en los APs para gestionar todas las funcionalidades radio y las aplicaciones de gestión inalámbrica personalizadas.

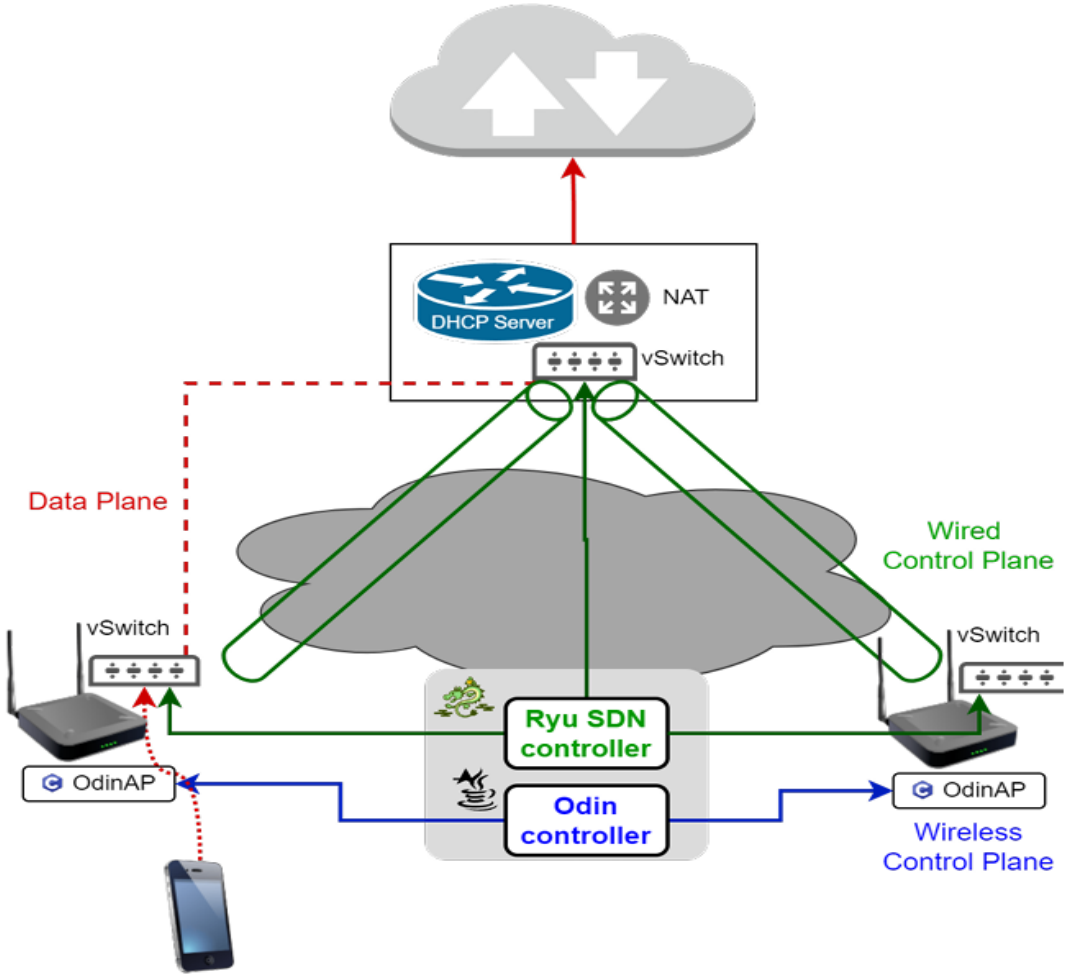


Figura 2. Plano de datos y plano de control

Plano de Datos

El plano de datos se encarga de establecer la conectividad para las STAs a través de puntos de acceso Wi-Fi. Las STAs que quieran conectarse se asocian con cualquiera

de los APs desplegados siguiendo los procedimientos inalámbricos estándar.

Una vez que la STA se asocia, se establece la ruta de datos inalámbrica, lo que permite la configuración de la ruta completa, incluyendo la asignación de direcciones IP mediante DHCP y la conexión cableada entre el AP y el *router* de acceso a Internet. Este *router* funciona como servidor DHCP y realiza traducción de direcciones de red (*NAT*), proporcionando acceso a Internet.

Para permitir el despliegue en cualquier ubicación, independientemente de la infraestructura de red subyacente, se establece una red *overlay* con una ruta de datos cableada tunelizada. Esta conexión utiliza túneles GRE configurados sobre las interfaces físicas.

Esta arquitectura se ilustra en las Fig. 1 y Fig. 2, y la configuración del plano de datos con túneles se detalla en Fig. 3.

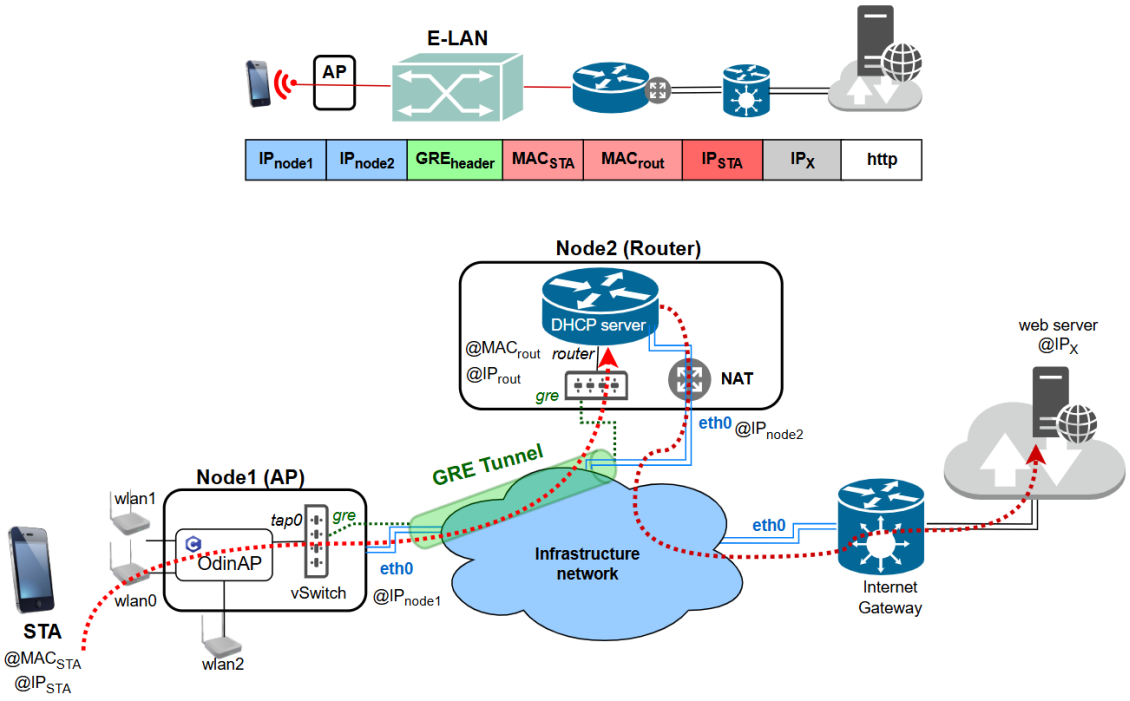


Figura 3. Plano de datos. Tráfico uplink: de la STA a Internet

Plano de Control

El enfoque de redes SDWN introduce la programabilidad a través de controladores centralizados que permiten la gestión de APs virtuales, *switches* y otros elementos de red,

optimizando la gestión de recursos.

El protocolo *Odin* supera las limitaciones de *OpenFlow* en la capa MAC de IEEE 802.11 como son la gestión de LVAP o la configuración de parámetros de transmisión inalámbrica. Este protocolo facilita la comunicación entre el controlador y los APs, manejando todos los intercambios de control y gestión. Utiliza conexiones TCP para información relacionada con la gestión como son la asociación, los traspasos o heterogeneidad de las redes WLAN y utiliza conexiones UDP para informes periódicos y notificaciones.

Los agentes *Odin*, que se ejecutan en los APs inalámbricos, proporcionan la información necesaria para que el controlador orqueste la red Wi-Fi y recupere métricas relevantes. Las operaciones críticas en tiempo real del protocolo MAC de Wi-Fi, como las confirmaciones IEEE 802.11 (ACKs), siguen siendo gestionadas por el *hardware* de la tarjeta Wi-Fi. Por otro lado, las funciones no críticas en tiempo real, como la gestión de asociaciones de clientes, se implementan en *software* tanto en el controlador como en los agentes.

La arquitectura utiliza la abstracción LVAP, que permite controlar a la STA y asegurar traspasos de nivel 2 sin interrupciones. Para reducir la sobrecarga en el medio inalámbrico, se implementa una solución basada en el concepto *BIGAP*. Esta arquitectura utiliza un único BSSID global compartido por todos los APs de un mismo conjunto de servicio extendido (ESS). Desde la perspectiva de la STA, toda la red se percibe como si fuera un único y gran AP, lo que reduce la sobrecarga de señalización inalámbrica y mejora la escalabilidad en redes Wi-Fi. Para lograr esto, *BIGAP* asigna diferentes canales de frecuencia radio a los APs y, durante el proceso de traspaso, aprovecha la funcionalidad DFS (*Dynamic Frequency Selection*) de IEEE 802.11 para que la STA perciba que el AP al que está conectada está simplemente cambiando de canal.

El establecimiento de la ruta de datos inalámbrica se logra cuando el agente *Odin* en el AP sigue los procedimientos estandarizados de asociación y autenticación de 802.11. Durante el intercambio de esta señalización, el agente *Odin* se comunica con el controlador *Odin* para registrar la STA en la red y crear el LVAP correspondiente. Esta información incluye la dirección IP asignada a la STA, lo que requiere señalización DHCP. El agente *Odin* interviene para finalizar la creación del LVAP capturando el

mensaje DHCP ACK y notificándolo al controlador *Odin*.

El controlador SDN *Ryu* establece y mantiene una red virtual en el segmento cableado. Esto lo hace interconectando *switches* virtuales. Las rutas de datos inalámbrica y cableada se controlan por separado, usando el controlador *Odin* para el segmento inalámbrico y el controlador SDN *Ryu* para el segmento cableado.

Durante la conexión física en el despliegue, al iniciar los APs y el *router*, los *switches* virtuales se conectan al controlador SDN *Ryu* y completan el *handshake* inicial de *OpenFlow*. Este proceso permite al controlador identificar las direcciones IP de los nodos que alojan las funciones virtuales e iniciar la creación de túneles GRE, que se añaden como puertos a los *switches* virtuales.

Para minimizar los retrasos causados por el restablecimiento de la ruta de datos cableada durante un traspaso, el controlador SDN *Ryu* debe ser notificado para actualizar las reglas de flujo y lograr un traspaso sin interrupciones (*seamless handover*). Cuando una STA se asocia con un nuevo AP, se crea una nueva regla de flujo en la tabla del *switch* virtual correspondiente. Para evitar esperar a que las reglas antiguas expiren, la implementación propuesta activa explícitamente las modificaciones necesarias de las reglas de flujo. El nuevo AP que atiende a la STA genera paquetes ARP "gratuitos" en nombre de la STA después del traspaso. Estos paquetes de difusión se reenvían al controlador SDN *Ryu*, que instruye a los *switches* virtuales relevantes para que inunden la red, actualizando proactivamente la tabla de flujo con la nueva ruta de reenvío, incluso en ausencia de tráfico *unicast*.

2.2.2.1 *OdinAP* – Punto de acceso

NeWLAN utiliza APs virtuales diseñados a medida, implementados como una aplicación *standalone* de espacio de usuario (*OdinAP*) desarrollada en C, lo que permite alcanzar un mayor rendimiento y *throughput* en la transmisión de datos. *OdinAP* está preparado para ser *contenerizado* (con *Docker*) y desplegado en ordenadores de propósito general, lo que facilita su uso en entornos distribuidos gestionados por plataformas como *Kubernetes*.

Para gestionar las comunicaciones tanto a nivel de datos como a nivel de control, *OdinAP* hace uso de varios *sockets*. Un *socket* es una interfaz *software* que permite el

intercambio de datos entre procesos, ya sea dentro del sistema o entre sistemas remotos a través de la red. Estos *sockets* nos permiten establecer conexiones TCP o UDP con otros equipos y son fundamentales para establecer comunicaciones con el controlador SDN *Ryu*.

Los sockets utilizados son los siguientes:

- Interfaz inalámbrica principal → Funciona en modo monitor para poder inyectar las tramas Wi-Fi en el medio inalámbrico y escuchar aquellas que deba transmitir por la interfaz *Ethernet* hacia el *router*.
- Interfaz inalámbrica auxiliar → Funciona en modo monitor para poder, entre otras funcionalidades, escuchar las tramas enviadas y recibidas por la interfaz inalámbrica principal. Ello permite monitorizar la transmisión de *OdinAP* para realizar el control de tasa[1].
- Interfaz inalámbrica auxiliar 2 → Funciona en modo monitor para poder escuchar en un canal dado las tramas enviadas por STAs pertenecientes a nuestra infraestructura Wi-Fi. Su función es monitorizar la transmisión para realizar los traspasos de STAs entre APs.
- *Socket* TCP (plano de control) → Permanece a la espera de conexiones TCP del controlador. Realiza el *three-way-handshake* y crea un nuevo *socket* TCP para crear la sesión TCP.
- *Socket* UDP (plano de control) → Envía información de control al controlador sobre UDP (notificaciones e informes periódicos).
- Socket de la interfaz TAP (plano de datos) → Recibe y envía tramas de datos por/a la interfaz fija (*Ethernet*) del AP (conexión con red cableada). Corresponde a la interfaz TAP de *OdinAP*.

[1] Lo ideal sería poder aplicar el algoritmo de control de tasa a partir de valores proporcionados por el *driver* a través del *Radiotap TX Flags*. Pero no todos los *drivers* lo implementan y por ello la utilización de esta interfaz para control de tasa permite ser independientes del *driver* de la tarjeta de red inalámbrica utilizada.

2.2.2.2 *Router* DHCP y NAT

Para proporcionar conectividad entre la red *WLAN* privada e internet utilizamos un *pod* que actúa como *router*. Además de encaminar el tráfico, este *pod* también actúa como servidor DHCP y realiza traducciones NAT. El servicio DHCP ofrece a los host direcciones de la red 192.168.137.0/24 excepto la 192.168.137.131, que está configurada de manera estática.

El *router* está configurado para traducir direcciones IP privadas a direcciones IP públicas con el fin de poder acceder a internet, pero no a la inversa. Además, este *pod* va a ser el encargado de marcar el tráfico, indicando el valor del campo DSCP que debe tener cada paquete que encamina. Este campo nos ayudará a diferenciar tráficos y aplicar políticas de *slicing* en los puntos de acceso.

2.3 *ONOE* – ALGORITMO DE CONTROL DE TASA

Onoe (*On Demand Transmission Opportunity Enhancement*) es el algoritmo de control de tasa implementado en un entorno experimental propio denominado *Inymon* [1], desarrollado con el fin de poder crear y probar soluciones en entornos Wi-Fi antes de su incorporación a entornos Wi-Fi con STAs reales. El algoritmo *Onoe* fue diseñado por *Mad Wifi* (*Multiband Atheros Driver for Wi-Fi*), un controlador de código abierto para sistemas *Linux*, especialmente para tarjetas inalámbricas que utilizan *chipsets* de *Atheros*.

Este algoritmo de control funciona en lazo abierto, es decir, que ajusta la tasa de transmisión basándose únicamente en sus propias mediciones y estadísticas, sin utilizar retroalimentación directa del receptor. Además, la implementación de este algoritmo se plantea como una solución independiente del *driver* de la tarjeta de red del punto de acceso, pues en ocasiones esta no proporciona los valores de pérdidas y retransmisiones.

Para obtener sus propias mediciones se utiliza una interfaz auxiliar, denominada también “interfaz espía”. Se trata de una segunda interfaz Wi-Fi física cuya función es monitorizar los números de secuencia de los paquetes capturados en el canal Wi-Fi donde se esté dando servicio. A partir de los números de secuencia, el sistema podrá saber si se ha retransmitido un paquete (repetición del número de secuencia) o si un paquete ha sido recibido correctamente (se observa un número de secuencia mayor al de dicho paquete). Esta monitorización no permite diferenciar si la retransmisión ha sido producida por una

colisión o por una pérdida, sin embargo, proporciona al sistema una adaptabilidad adecuada independientemente del dispositivo receptor.

Para realizar dicho control, *Onoe* se basa en un sistema de créditos evaluados en ventanas temporales de un tiempo dado. Al finalizar cada intervalo, se evalúa el rendimiento calculando estadísticas según el número de paquetes enviados correctamente y el número de retransmisiones. Se sumará un crédito si se obtiene un porcentaje de retransmisiones inferior a un valor dado (10% en nuestro caso). Cuando se acumulen 10 créditos, se aumentará la tasa de transmisión y se reiniciará el contador. En cambio, si el porcentaje de retransmisiones en dicha ventana temporal es superior a un valor dado (50% en nuestro caso), se restablecerán los créditos a 0 y se reducirá la tasa de transmisión. En consecuencia, este mecanismo disminuye la sensibilidad ante posibles variaciones rápidas de las condiciones del canal a cambio de una actualización más lenta de la tasa de transmisión.

Para aumentar o disminuir la tasa de transmisión se modifica dinámicamente el MCS, el número de *spatial streams* (flujos espaciales, para el uso de MIMO) y la codificación de canal. Para obtener los detalles completos de los diferentes MCS y su configuración se recomienda consultar la Tabla 8 del Anexo A.

Dado que *Onoe* es un algoritmo que se aplica de forma local, el control de tasa funcionará únicamente en un sentido de la comunicación (*downlink*). Este es el algoritmo (ya probado en [1]) que debemos integrar en el punto de acceso y evaluar con STAs reales en este TFG.

2.4 *SLICING* – SEGMENTACIÓN DE RED Y DIFERENCIACIÓN DE TRÁFICOS

Slicing o segmentación de red es el concepto utilizado para que la transmisión *downlink* en un entorno *SDWLAN* (desde el punto de acceso hasta la STA) se adapte a las necesidades de los tráficos y haga un uso eficiente de los recursos de la red. El objetivo de esta técnica es conseguir proporcionar un trato diferenciado a ciertos flujos de datos que envía el punto de acceso por el medio inalámbrico. La manera en la que se ha implementado permite aplicar diversos criterios para establecer distintos niveles de prioridad en la transmisión.

En este contexto, podríamos considerar tráfico prioritario a aquel que cumple

condiciones como:

- Tráfico inelástico: paquetes con restricciones temporales que requieren latencias bajas.
- Usuarios *premium*: paquetes asociados a usuarios con privilegios que desean tener una prioridad que mejore la experiencia de los servicios que estén utilizando.

Para poder realizar una identificación de los tipos de tráfico se necesita un elemento en la red que los marque previamente. En el escenario empleado, es el *router* el elemento encargado de realizar dicha clasificación. Esta se lleva a cabo utilizando el campo DSCP de la cabecera IP (Fig. 4), compuesto por 6 bits. Su utilidad es precisamente clasificar y gestionar el tráfico de red, permitiendo aplicar *scheduling* o políticas de calidad de servicio (QoS).

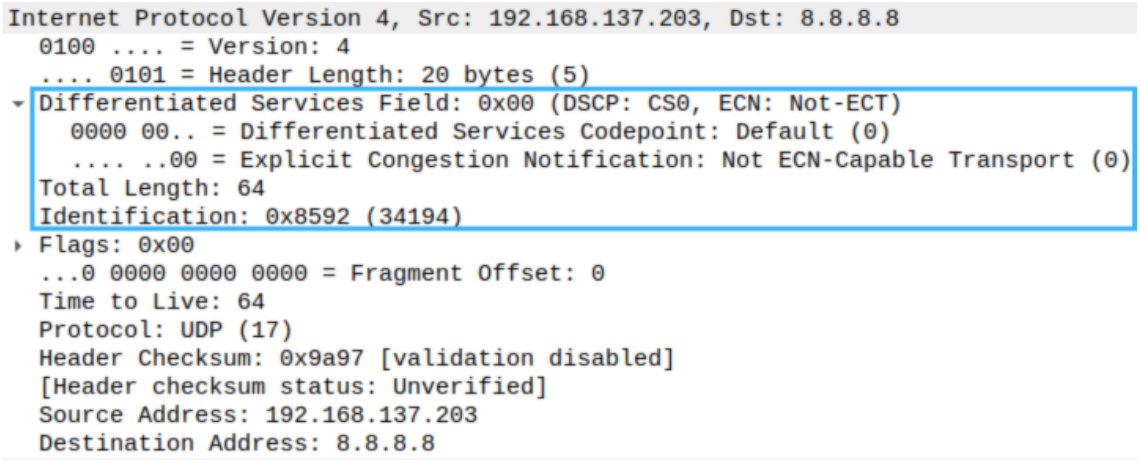


Figura 4. Estructura de la cabecera IP y DSCP

Para desarrollar el esquema de prioridades mencionado anteriormente se ha utilizado el entorno experimental *Inymon*, dando lugar a la propuesta de RAN *slicing* (sistema de *slices* y colas en entorno radio) descrita en [3]. La Fig. 5 representa el sistema desarrollado para varios puntos de acceso, extraída del trabajo mencionado, y que debemos integrar en el punto de acceso y evaluar con STAs reales en este TFG.

El término *slice* se puede entender como una porción o partición lógica de los recursos del punto de acceso que permite asignar recursos específicos a conjuntos de tráficos. Cada *slice* está formado por un conjunto de colas (un máximo de 8 para nuestro sistema) donde los paquetes quedarán a la espera de ser enviados por la interfaz

inalámbrica. De esta manera, cada paquete esperará en la cola y *slice* que le corresponda según la prioridad que se le haya asignado en el marcado. Para el desencolado de los paquetes, se utiliza un *thread* (hilo) que se ejecuta en paralelo al resto del código y cuyo objetivo es extraer los paquetes de las colas siguiendo las políticas establecidas de reparto de recursos. El funcionamiento de ese *thread* se detallará más adelante en las secciones posteriores.

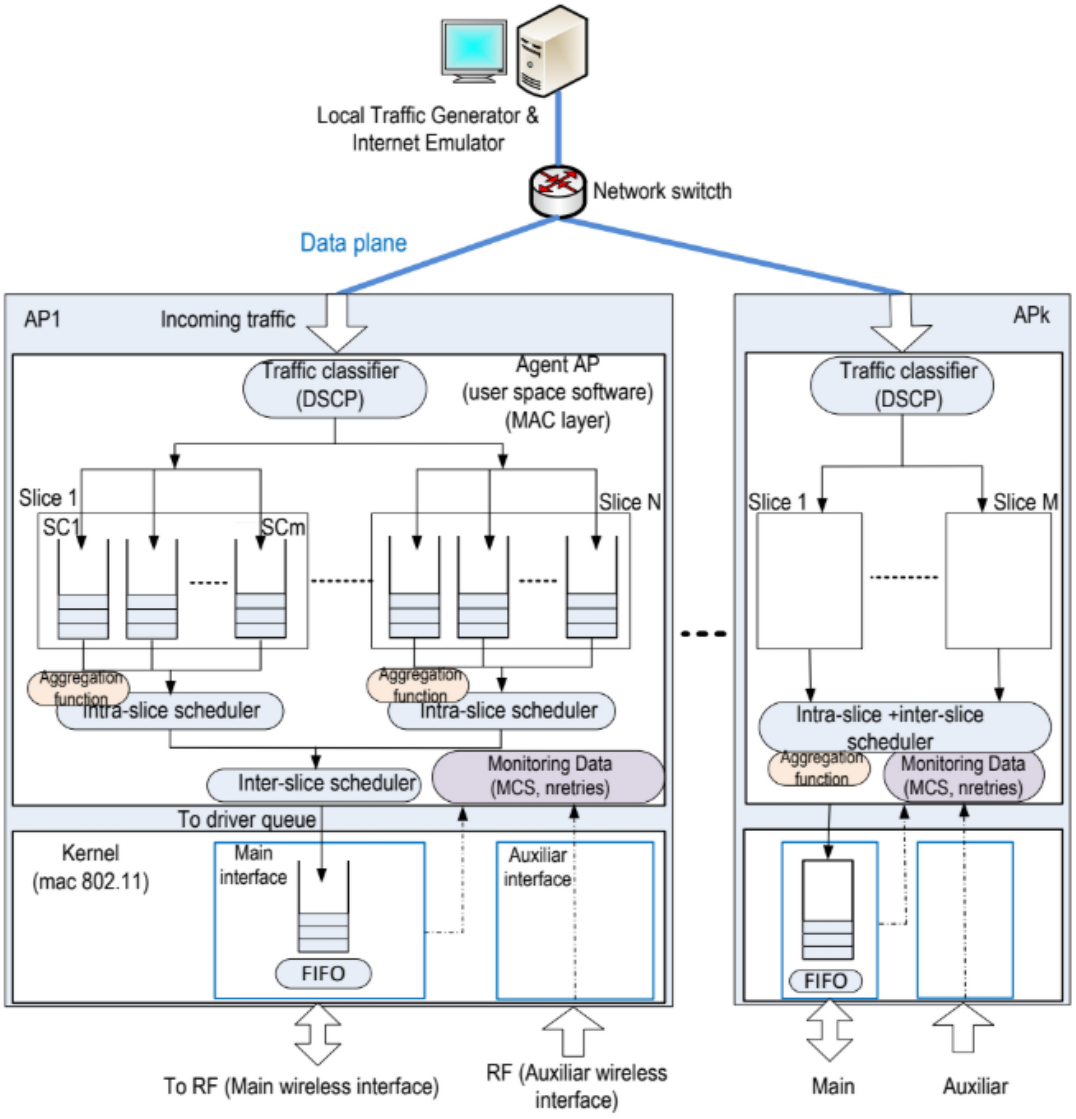


Figura 5. Arquitectura de la solución de slicing en el AP

2.4.1 Scheduling

Los algoritmos de *scheduling* empleados en este TFG (ver Anexo B) definen la forma en la que se extraen los paquetes de las colas y determinan cuál es el criterio de

reparto de *airtime* excedente, es decir, el *quantum* que no sea utilizado por los *slices* y colas. En el trabajo mencionado [3], se trabajó con dos algoritmos: *Round Robin* (RR) y *Airtime Deficit Weighted Round Robin* (ADWRR). En nuestro caso, el análisis se centrará en ADWRR por su mayor complejidad y flexibilidad en la asignación de recursos.

Según su ámbito de operación, es posible clasificar los algoritmos en 2 tipos:

- Algoritmo *inter-slice*, responsable de repartir los recursos entre los *slices* activos (aquellos que tienen colas con paquetes en espera).
- Algoritmo *intra-slice*, responsable de repartir el *quantum* y el *déficit* entre las colas activas de un mismo *slice*.

Ambos algoritmos son independientes entre sí, es decir, en la ejecución del sistema, se puede configurar un algoritmo determinado para el reparto de recursos *inter-slice* y otro algoritmo distinto para el reparto *intra-slice*.

2.4.1.1 ADWRR – Déficit, *quantum* y pesos

ADWRR es un algoritmo que, como su nombre indica, comparte los principios básicos de *Round Robin*, otro algoritmo más básico y conocido en la planificación y distribución de recursos. El objetivo de *Round Robin* es asegurar que todas las colas y *slices* tengan la misma oportunidad de ser servidas de manera justa y ordenada. Para ello, atenderá a cada cola o *slice* una cantidad fija de tiempo durante el cual puede ocupar el medio radio antes de pasar a la siguiente cola o *slice*. De esta manera, se atiende a las partes participantes de manera equitativa y siguiendo un orden circular. En el caso de ADWRR, el algoritmo evoluciona para gestionar las colas y *slices* de manera más controlada, pudiendo definir distintos criterios de reparto de recursos.

Para poder entender el funcionamiento del algoritmo ADWRR es imprescindible presentar primero los conceptos de déficit, *quantum* y pesos. Se define déficit como el tiempo en µs del que dispone una cola o *slice* para ocupar el medio inalámbrico. Cada cola o *slice* tiene un contador denominado "DC" (*Deficit Counter*) que almacenará el valor del déficit de cada estructura.

El *quantum* (Q) es una cantidad fija de tiempo asociada a un *slice* que se suma a su déficit. Como los *slices* se visitan siguiendo un esquema rotativo, se sumará dicha cantidad Q cada vez que se visite dicho *slice*. Este valor es específico para cada *slice* y

representa una garantía de tiempo de transmisión en el medio inalámbrico.

Una vez revisado el concepto de *quantum*, se puede explicar la función de los pesos. Los pesos son parámetros asignados a cada cola dentro de un *slice* y su función es distribuir el *quantum* asignado a dicho *slice*. Como se mencionó anteriormente, el *quantum* es un valor que se asigna a cada *slice*, por lo que los pesos se utilizan para repartir dicho *quantum* entre las colas que pertenecen a ese *slice*. Cada cola tiene un peso específico, y el *quantum* que le corresponde se calcula como se muestra en la Ecuación 3:

$$Q[s,i] = \left(\frac{W[s,i]}{\sum_{j\ no\ vacía\ \in s} W[s,j]}\right) \cdot Q[s]$$

Ecuación 3. Cálculo del quantum de una cola a partir del quantum del slice

- Q[s,i]: *Quantum* para la cola *i* del *slice s*
- Q[s]: *Quantum* asignado al *slice s*
- W[s,i]: Peso asignado a la cola *i* del *slice s*

Se puede observar que el *quantum* del que podrá disponer la cola *i* será el del *slice* multiplicado por un factor (menor o igual a 1) que se calculará ponderando el peso de la propia cola con la suma de los pesos de las colas activas, es decir, que necesiten transmitir paquetes. Por lo tanto, si una cola no está activa, no participará en el reparto de *quantum* y este se repartirá entre las colas que estén activas.

Atendiendo al funcionamiento del *scheduling*, existen varios algoritmos y cada uno de ellos realizará un reparto de recursos diferente. Sin embargo, todos ellos comparten un funcionamiento base que se explica a continuación[2].

El *thread* encargado de extraer los paquetes de las colas recorre secuencialmente las colas activas de manera rotativa. Cuando una cola tiene el turno, el hilo extrae paquetes hasta que esta se vacíe o su contador de déficit (DC) no disponga del valor suficiente para

[2] Ver Anexo B. Para un análisis más detallado del funcionamiento de estos algoritmos consultar [3]

enviar el siguiente paquete. Durante la transmisión, se calcula el *airtime* (el tiempo en microsegundos que el paquete ocupa en el medio) del paquete y se descuenta dicho valor del contador DC, ajustando así la disponibilidad de recursos para las siguientes transmisiones. En el momento en el que el *thread* no puede continuar extrayendo paquetes de la cola, el contador DC se reinicia a cero y se avanza a la siguiente cola activa del *slice*, y si no quedan más colas en ese *slice*, avanzará al siguiente. Cuando se avance a la última cola del último *slice* se volverá a empezar desde el principio.

Para entender mejor el funcionamiento del *scheduling*, la Tabla 1 plantea un ejemplo de *scheduling inter-slice*:

Tabla 1. Ejemplo de funcionamiento de scheduling

	SLICE 1	*SLICE* 2	*SLICE* 3
QUANTUM (µs)	3500	2500	4000

Cada *slice* tiene una asignación de *quantum* que le garantiza un porcentaje de ocupación mínimo en caso de saturación del enlace radio. A continuación, se muestra cómo se debe realizar el cálculo de dicho porcentaje.

$$Airtime(\%)_{S1} = \frac{3500}{3500 + 2500 + 4000} \times 100 = 35\%$$

$$Airtime(\%)_{S2} = \frac{2500}{3500 + 2500 + 4000} \times 100 = 25\%$$

$$Airtime(\%)_{S3} = \frac{4000}{3500 + 2500 + 4000} \times 100 = 40\%$$

Por lo tanto, si el tráfico del *slice* 3 aumenta, se puede garantizar que este utilice el 40% del tiempo de ocupación del canal durante la transmisión.

Sin embargo, en ciertos momentos, algunos *slices* pueden no requerir totalmente el tiempo de transmisión que se les ha garantizado, utilizando solo una parte de su asignación. En estos casos, el *airtime* no utilizado puede redistribuirse entre los *slices* activos que necesiten aumentar su *throughput* de salida para igualarlo con el *throughput* de entrada. De este modo, se evita el desperdicio de recursos y se optimiza el rendimiento del punto de acceso.

Para la redistribución del *airtime* excedente, se pueden emplear tres algoritmos. La principal diferencia entre ellos se encuentra en si la asignación de *quantum* es constante o dinámica en el tiempo. En el caso de una asignación dinámica, también varía el criterio de distribución, pudiendo priorizar ciertos *slices* o mantener un reparto equitativo entre todos.

En el caso del *scheduling intra-slice*, ocurre lo mismo, pero en lugar de ajustar el quantum, se modifica el valor de los pesos. Es decir, si una cola no requiere la totalidad de su peso asignado, este puede reducirse y reasignarse a otra cola que sí lo necesite.

A continuación, se describen más detalladamente estos algoritmos.

2.4.1.2 ADWRR - Pesos o quantums estáticos

Esta variante de ADWRR mantiene una asignación fija del valor del *quantum* o de los pesos, dependiendo si se aplica a nivel *inter-slice* o *intra-slice*. Para ilustrar su funcionamiento, se muestra en Fig. 6 un experimento en el que se representa el porcentaje de *airtime* utilizado por cada *slice* a lo largo del tiempo. A la derecha se representa el *quantum* asignado con respecto al tiempo, que permanece constante.

En este caso, se ha configurado el *quantum* de forma que el *slice* 1 tenga garantizado el 30 % del *airtime*, el *slice* 2 del 20 % y el *slice* 3 del 50 %. Durante los primeros 13 segundos del experimento, todos los *slices* hacen uso completo del *airtime* que les ha sido asignado, por lo que ninguno puede ceder recursos.

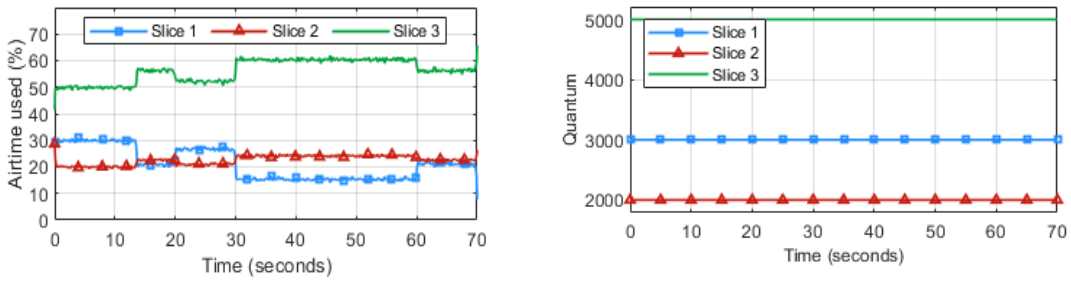


Figura 6. Reparto airtime. Asignación quantum estática

Sin embargo, en el instante t=13, el *slice* 1 reduce su demanda y pasa a utilizar solo el 20 % del *airtime*, liberando un 10 % que puede ser aprovechado por otros *slices* que aún se encuentren insatisfechos. Ese 10 % excedente se redistribuye

proporcionalmente entre los *slices* activos en ese momento (aquellos con paquetes pendientes en sus colas), es decir, los *slices* 2 y 3. La cantidad de *airtime* adicional que recibe cada uno dependerá de su *quantum* nominal, como se muestra en los siguientes cálculos.

$$Extra\ Airtime(\%)_{S2} = 10\% \times \frac{2000}{2000 + 5000} \approx 2.857\%$$

$$Extra\ Airtime(\%)_{S3} = 10\% \times \frac{5000}{2000 + 5000} \approx 7.143\%$$

2.4.1.3 ADWRR - Igual ratio de satisfacción

Esta variante del algoritmo ADWRR busca realizar un reparto del *airtime* excedente que consiga un grado de satisfacción similar para todos los *slices* o colas insatisfechas. En este caso, se define el ratio de satisfacción (*Degree of Satisfaction*, *DS*) como el cociente del *throughput* de salida R_{out} entre el *throughput* demandado por el flujo en el punto de acceso $R_{demanded}$. Cabe destacar que $R_{demanded}$ puede tomar valores superiores al *throughput* de entrada si la cola/*slice* ha almacenado paquetes durante un periodo de tiempo y ahora tienen los recursos necesarios de *airtime* para vaciar las colas al mismo tiempo que se envía el tráfico entrante.

El valor del *throughput* de salida R_{out} puede estar limitado por un SLA (*Service Level Agreement*) y se puede configurar con el parámetro MBR (*Maximum Bit Rate*). Este parámetro afectará al calcular el valor del ratio de satisfacción DS (ver Ecuación 4) tanto para los algoritmos *inter-slice* como para los *intra-slice*.

$$\mathrm{DS[s]} = \begin{cases} \dfrac{R_{out}[s]}{min(\,R_{demanded}[s]\,,\ MBR[s]\,)}\ ,\ \ si\ hay\ slices\ insatisfechos \\ \dfrac{R_{out}[s]}{R_{demanded}[s]}\ ,\ \ si\ todos\ los\ slices\ satisfechos \end{cases}$$

$$\mathrm{DS[s,i]} = \begin{cases} \dfrac{R_{out}[s,i]}{min(\,R_{demanded}[s,i]\,,\ MBR[s,i]\,)}\ ,\ \ si\ hay\ colas\ insatisfechas \\ \dfrac{R_{out}[s,i]}{R_{demanded}[s,i]}\ ,\ \ si\ todas\ las\ colas\ satisfechas \end{cases}$$

Ecuación 4. Cálculo del grado de satisfacción del slice 's' y de la cola 'i'

Cuando $MBR > R_{demanded}$, el MBR no influye en el cálculo, ya que no se alcanza el *throughput* límite de salida. Sin embargo, si $MBR < R_{demanded}$, el grado de satisfacción se calculará tomando MBR como referencia (pues se supone que la tasa de salida no puede superar dicho valor). En cambio, si tan solo hay una cola o *slice* activos en ese momento, se realizará siempre el cálculo teniendo en cuenta $R_{demanded}$ puesto que se omite MBR, permitiendo una mayor calidad de servicio cuando los recursos estén infrautilizados.

Cada segundo se monitorizan los valores de DS de cada *slice* y de cada cola. Si alguna de estas estructuras deja de utilizar los recursos garantizados, estos podrán ser reasignados a otra cola o *slice* que los necesite. La redistribución se realiza asignando los recursos a la estructura con el menor ratio de satisfacción DS, priorizando así aquellas con mayor necesidad de recursos. De este modo, se abandona el reparto proporcional basado en pesos y se adopta un enfoque centrado en las necesidades individuales de cada estructura.

Para llevar a cabo la distribución de recursos, se ajusta el *quantum* asignado a cada *slice Q[s]* en el caso del algoritmo *inter-slice*, o los pesos de las colas *W[s,i]* en los algoritmos *intra-slice*, lo que a su vez modifica el *airtime* recibido por cada cola *Q[s,i]*.

Según lo explicado anteriormente, un *slice* o una cola no pueden ceder recursos y recibir al mismo tiempo. Para que una estructura pueda ceder recursos, su grado de satisfacción (DS) debe ser igual a 1 o estar muy próximo a este valor, considerando un pequeño margen de tolerancia. Por otro lado, para recibir recursos de otras estructuras, debe ser la que presente el menor grado de satisfacción.

Una descripción más detallada del funcionamiento de este algoritmo se puede encontrar en el Anexo B.

2.4.1.4 ADWRR - Preferencia al índice menor

Esta variante de ADWRR busca redistribuir el *airtime* excedente entre las estructuras con un grado de satisfacción inferior a 1 (DS < 1), siguiendo un orden de prioridad basado en su índice. En el caso de un algoritmo *inter-slice*, los *slices* con índices más bajos tienen mayor prioridad; por ejemplo, el *slice* 1 tendrá preferencia sobre el *slice* 2. De manera similar, en un algoritmo *intra-slice*, la prioridad dentro de un *slice* se asigna

según el índice de las colas, donde la cola 0 será siempre la más prioritaria.

Por lo tanto, cuando se ceda *airtime*, este se asignará primero a las estructuras con menor grado de satisfacción, comenzando por el *slice* y cola de menor índice. Una vez que estas estructuras alcancen un nivel adecuado de satisfacción, se considerarán "satisfechas", permitiendo así continuar con la redistribución hacia las siguientes.

2.4.1.5 Particularidades de la implementación

En el sistema desarrollado, los 3 bits más significativos del campo DSCP se han dedicado al número de *slice*, mientras que los 3 bits menos significativos se dedican a las colas de dicho *slice*. En este caso, para simplificar la clasificación de tráfico, se ha realizado una asociación experimental entre el número del puerto destino y un valor DSCP determinado (ver Tabla 2).

Tabla 2. Asociación entre puerto destino y DSCP

Slice	Cola	DSCP	PUERTO	Slice	Cola	DSCP	PUERTO
0	0	0	5000	2	0	16	5016
0	1	1	5001	2	1	17	5017
0	2	2	5002	2	2	18	5018
0	3	3	5003	2	3	19	5019
0	4	4	5004	2	4	20	5020
0	5	5	5005	2	5	21	5021
0	6	6	5006	2	6	22	5022
0	7	7	5007	2	7	23	5023
1	0	8	5008	3	0	24	5024
1	1	9	5009	3	1	25	5025
1	2	10	5010	3	2	26	5026
1	3	11	5011	3	3	27	5027
1	4	12	5012	3	4	28	5028
1	5	13	5013	3	5	29	5029
1	6	14	5014	3	6	30	5030
1	7	15	5015	3	7	31	5031

Se ha utilizado la herramienta de *Linux iptables*, que facilita el proceso de marcado de paquetes. Este marcado se ha implementado en la interfaz del *router* que conecta con el punto de acceso. De este modo, en nuestras pruebas, todos los paquetes que lleguen al punto de acceso desde el *router* estarán ya marcados.

3. ESCENARIO Y ASPECTOS DE LA INTEGRACIÓN

3.1 ESCENARIO DE TRABAJO

Para describir el escenario de trabajo con el que se realizarán las pruebas, se va a considerar el mismo desde dos puntos de vista: el primero consiste en cómo está compuesto el escenario en cuanto a componentes físicos, hardware y periféricos; en el segundo, se verá dónde se ejecutan los *pods* lanzados por *Kubernetes* y las interfaces que se crean.

3.1.1 Escenario de trabajo - Nivel físico

El escenario experimental está formado por dos equipos de propósito general: un *PC Intel NUC* y una *Raspberry Pi 4*, ambos conectados a una misma red *Ethernet*, utilizando un *switch* como elemento de construcción de red LAN. Los dispositivos se encuentran dentro de la red IP 155.210.157.0/24, la cual dispone de un *router* con acceso a Internet. Además, incluye uno o más equipos de propósito general (*MiniPC*s) que albergarán distintas STAs o tarjetas de red inalámbricas.

Como se puede observar en Fig. 7, al *Intel NUC* se le han conectado tres tarjetas Wi-Fi mediante cables USB. Estas tarjetas serán utilizadas por el *pod* encargado de desempeñar la función de punto de acceso. La interfaz *wlan0* será la responsable de transmitir y recibir tramas Wi-Fi desde y hacia las estaciones conectadas a su red. En cambio, la interfaz *wlan1* actuará como interfaz auxiliar o "espía", capturando las tramas transmitidas por *wlan0* con el objetivo de recopilar estadísticas que permitan ajustar dinámicamente el MCS. Finalmente, la interfaz *wlan2* funciona en modo monitor para poder escuchar en un canal dado las tramas enviadas por STAs pertenecientes a nuestra infraestructura Wi-Fi. Su función es monitorizar la transmisión para realizar los traspasos

de STAs entre APs[3].

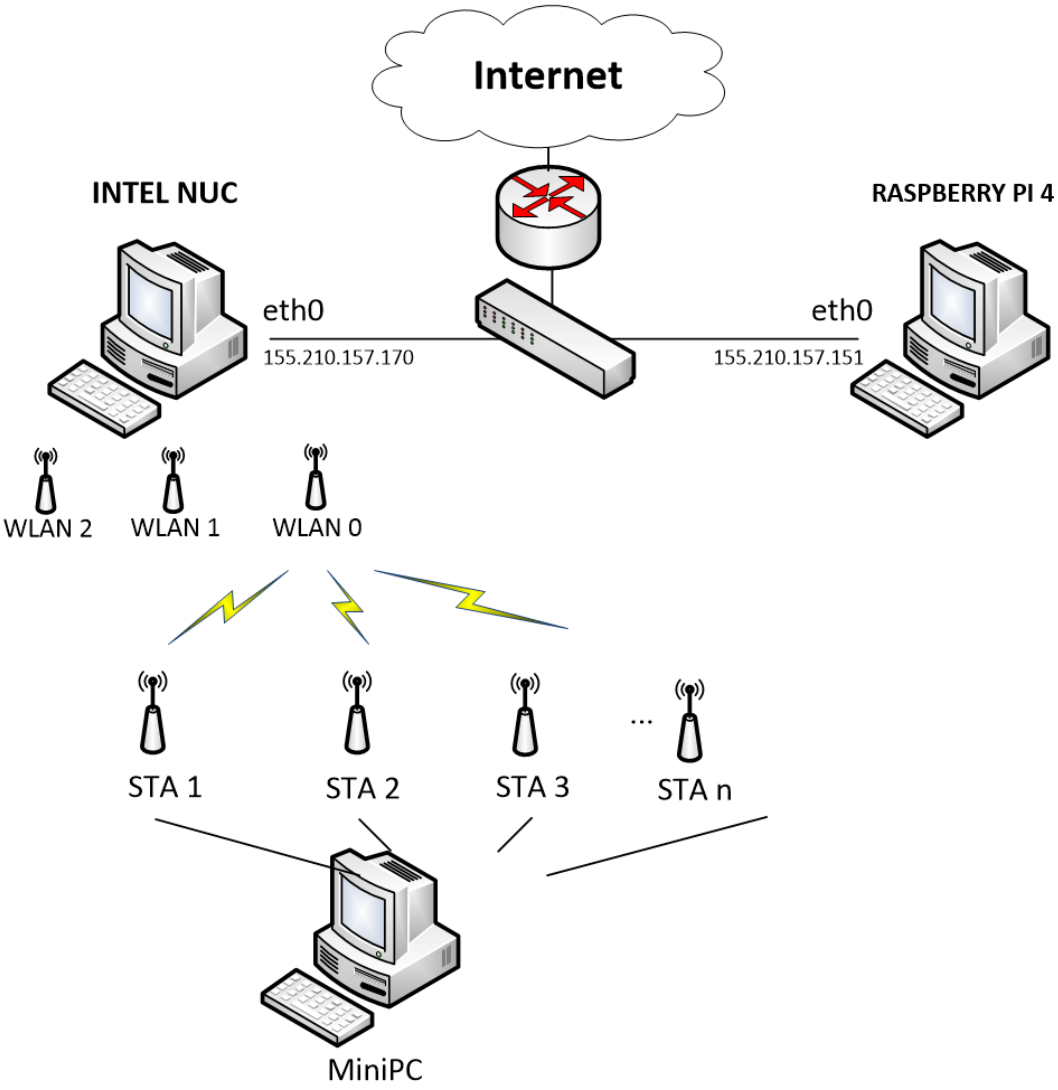


Figura 7. Esquema gráfico del escenario a nivel físico

Para la realización de las pruebas, se utilizan un número suficientemente grande de STAs en modo *managed* (cliente Wi-Fi) que se conectan a la red Wi-Fi desplegada. Los

[3] Esta función de monitorización podría ser proporcionada por las propias STAs si tuvieran soporte 802.11k. A día de hoy, no muchos dispositivos comerciales, aparte de los de Apple, tienen este tipo de soporte.

clientes Wi-Fi pueden estar ubicados en uno o más PCs de propósito general y se conectan vía USB.

3.1.2 Escenario de Trabajo - Nivel lógico

Una vez está definida la arquitectura *hardware* con la que se va a trabajar, se puede definir la arquitectura *software* o lógica (Fig. 8). Al haber trabajado con ordenadores de propósito general, es posible desplegar uno o más *pods* en un mismo equipo físico (nodo). Estos *pods* trabajan de forma aislada como si se tratase de máquinas distintas. Cada *pod* cuenta con su propia interfaz de red con la que se podrá comunicar de manera independiente. Para facilitar la conectividad entre *pods*, en los nodos se crean *bridges* e interfaces virtuales que agilizan el tráfico de red entre ellos.

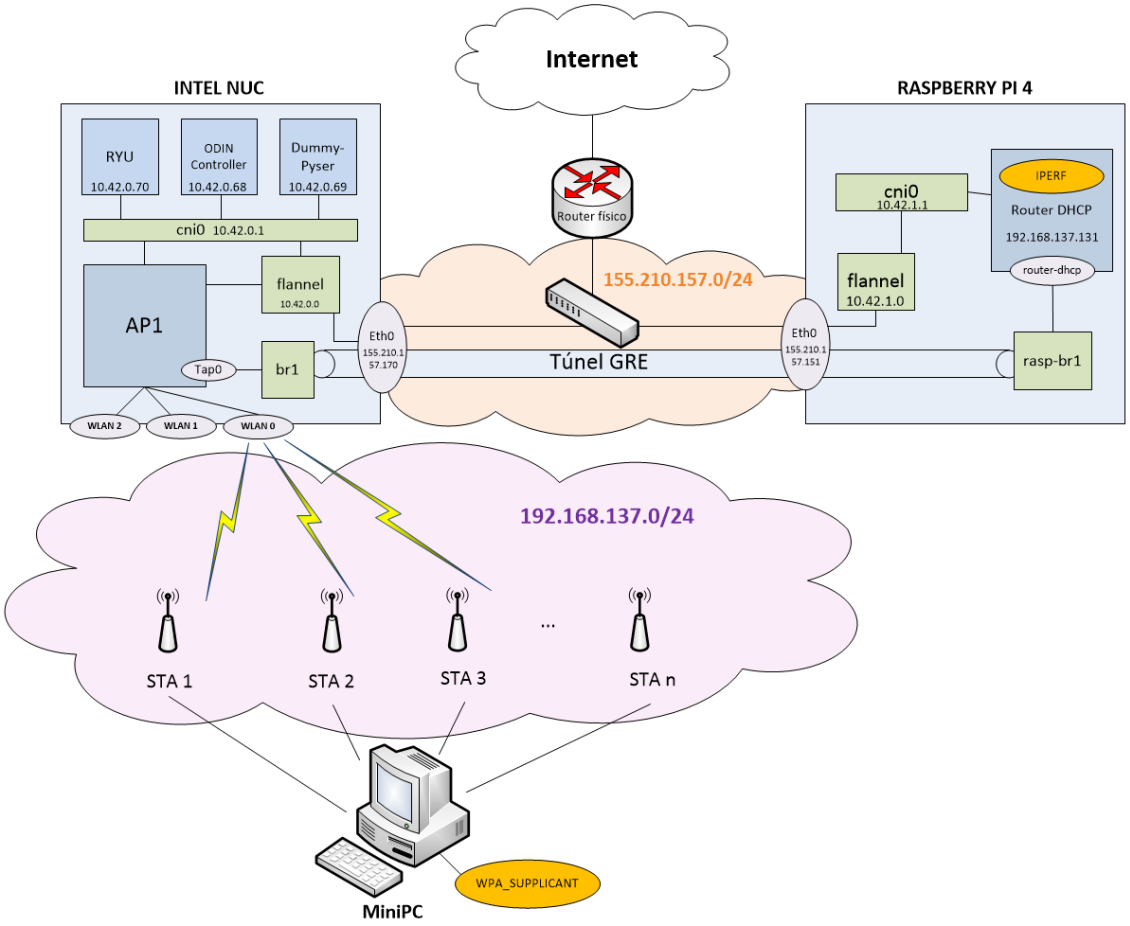


Figura 8. Esquema gráfico del escenario a nivel software

Los *pods* que se van a desplegar, mostrados en la Fig. 8, son los siguientes:

- AP1 – Punto de acceso Wi-Fi
- Controlador SDN *Ryu* – Gestiona la parte de red fija (crea y controla los túneles *Router-AP*) con *OpenFlow*
- Controlador *Odin* – Gestiona el segmento inalámbrico de la red desplegada (*handover*, asociación de la STA, autenticación, tabla de STAs conectadas, etc.)
- *Dummy Pyser* – Controlador de red. Funciona como servidor UDP para que el controlador de *Odin* conozca las direcciones IP de las interfaces de control de los APs.
- *Router* DHCP – *Router* que conecta la red inalámbrica a Internet

En este escenario, los *pods AP1*, controlador *Ryu*, controlador *Odin* y *Dummy Pyser* se despliegan en el *PC Intel NUC*, mientras que el *pod Router DHCP* se ejecuta en la *Raspberry Pi 4*. No obstante, esta distribución no es la única posible, ya que podrían haberse ubicado algunos *pods* en equipos físicos distintos. En este caso concreto, se ha optado por separar físicamente el *AP1* y el *Router DHCP* para simular un entorno lo más parecido posible a un despliegue real, donde es habitual que, en escenarios con múltiples puntos de acceso, el *router* se encuentre ubicado en un equipo distinto al del punto de acceso.

En el propio despliegue de los *pods*, también se crean nuevas interfaces de red. Estas son las que se muestran en la Fig. 8 en color verde.

- *cni0* –interfaz que conecta todos los *pods* del mismo nodo
- *br1* – interfaz que actúa como *bridge* para tunelizar el tráfico entre el punto de acceso y el *router*
- *flannel* – interfaz que define la red *overlay o* red superpuesta 10.42.0.0/16 que permite la comunicación de control entre los *pods* que se encuentran en nodos distintos.

De esta forma, se consigue el despliegue de la red Wi-Fi con máquinas virtuales, independiente de la red física, muchos menos equipos físicos y mayor flexibilidad y facilidad para incorporar nuevos *pods*.

Es importante señalar que, en la Fig. 7 y en la Fig. 8, se muestra también el *router*

de la red 155.210.157.0/24. Este dispositivo representa una máquina física independiente del *pod Router DHCP* desplegado. Conviene comprender las diferencias entre ambos elementos: por un lado, el pod *Router DHCP* es el que crea la red privada 192.168.137.0/24; por otro, el *router* físico de la red 155.210.157.0/24 proporciona salida a Internet y es utilizado como ruta por defecto por el *pod*, permitiendo el acceso a Internet desde la red WLAN privada.

Una vez desplegado el escenario de red, se conectan los clientes Wi-Fi, implementados mediante tarjetas Wi-Fi USB conectadas al equipo de propósito general denominado *MiniPC*. Para establecer la conexión con el punto de acceso se ha utilizado el *software wpa_supplicant*. Dada la diversidad de distribuciones y versiones de *Linux*, así como las particularidades de los controladores de las tarjetas de red, se optó por la versión 2.10 de *wpa_supplicant*, ya que se ha comprobado experimentalmente que ofrece un funcionamiento más estable y una mayor compatibilidad. A continuación, se muestra el archivo de configuración utilizado:

```
network={
        ssid="wi5sergio"
        key_mgmt=NONE
        scan_ssid=1
}
ctrl_interface=/run/wpa_supplicant
update_config=1
```

Este archivo de configuración de *wpa_supplicant* permite la conexión automática a una red Wi-Fi abierta llamada "*wi5sergio*", incluso si su SSID está oculto. Además, se habilita la posibilidad de que *wpa_supplicant* modifique este archivo si es necesario y se establece una interfaz de control para permitir la comunicación con herramientas externas como *wpa_cli*.

Por último, *Iperf* será la herramienta con la que el pod *Router DHCP* genere los distintos flujos UDP hacia las STAs para poder realizar las pruebas de tráfico deseadas.

3.2 ASPECTOS RELEVANTES A CONSIDERAR EN LA INTEGRACIÓN

Este trabajo se centra en la implementación de funcionalidades avanzadas en un entorno WLAN definido por *software*. Como ya ha sido comentado, dichas funcionalidades fueron desarrolladas anteriormente en un entorno de pruebas experimental conocido como *Inymon*, en el que se desarrollaron el algoritmo de control de tasa [1] y el sistema de *slicing* [3], respectivamente.

Inymon se desarrolló como un entorno de pruebas ágil en el que las modificaciones no supusieran un consumo de tiempo elevado, cuenta con las funciones básicas para un entorno Wi-Fi y se utiliza para desarrollar nuevas funcionalidades antes de incorporarlas al entorno real de *NeWLAN*, que es el entorno más completo.

Para lograr dicha sencillez en *Inymon*, se prescinde de algunas funciones fundamentales en Wi-Fi. Una de las primeras diferencias de *Inymon* es que no cuenta con el procedimiento de asociación entre el AP y la estación Wi-Fi. Es decir, el punto de acceso inyecta directamente como tramas Wi-Fi las tramas recibidas desde el *router* hacia las STAs. Para que el AP conozca la dirección MAC de las STAs conectadas, se modifica manualmente la tabla ARP, donde se apunta la correspondencia entre dirección IP y dirección MAC de cada una de ellas. Por esta razón, en *Inymon* tampoco se envían tramas *Beacon*, ya que no se sigue el procedimiento estándar de asociación y autenticación. Al no existir una asociación formal entre el AP y las estaciones, no es necesario anunciar la presencia de la red ni permitir que las STAs se conecten a ella de forma convencional. Esto provoca que las STAs no estén conectadas a ninguna red y, en consecuencia, no puedan generar tráfico. Por lo tanto, en *Inymon* solo se puede generar tráfico *downlink* (generado por el punto de acceso), y las STAs solo pueden escuchar las tramas dirigidas hacia ellas y devolver el ACK correspondiente si están configuradas en modo monitor. Todo ello implica que *Inymon* tampoco necesita tráfico de control, dado que la STA no está conectada a ninguna red. De este modo, este entorno experimental prescinde de protocolos como ARP y DHCP.

Debido a las diferencias mencionadas, es necesario tener en cuenta un conjunto de consideraciones al trasladar las funcionalidades del entorno experimental *Inymon* al entorno *NeWLAN*. Estas consideraciones han supuesto una serie de modificaciones que se explican a continuación.

3.2.1 Modificación del llenado del *buffer* del *kernel*

Como se ha descrito previamente, *Inymon* es un sistema sencillo y ágil con el que poder trabajar. Esa sencillez va asociada también a una menor carga computacional, sobre todo, al compararlo con *NeWLAN*.

Entre las distintas funciones que desempeña el *socket* de la interfaz *wlan1* del entorno de *NeWLAN*, destacan al menos dos de ellas:

- Detectar posibles pérdidas o colisiones y elaborar estadísticas en periodos de un segundo. Con las estadísticas se decide si las condiciones del canal son favorables como para aumentar el MCS, desfavorables como para reducirlo o adecuadas para el MCS utilizado. Esto constituye el algoritmo de control de tasa *Onoe* que opera en lazo abierto.
- Controlar la ocupación del *buffer* de salida del *kernel* de la tarjeta de red hasta un tamaño determinado.

Este control del *buffer* del *kernel* es necesario para evitar la pérdida de tramas por desbordamiento y, por otra parte, que estas tramas salgan con la información radio lo más actualizada posible. El límite de tamaño que se proponga ha de cumplir ambos aspectos y una vez establecido se gestiona por control de flujo. En este apartado, se describe el problema, pero antes de entrar en detalle es necesario adquirir una visión más global.

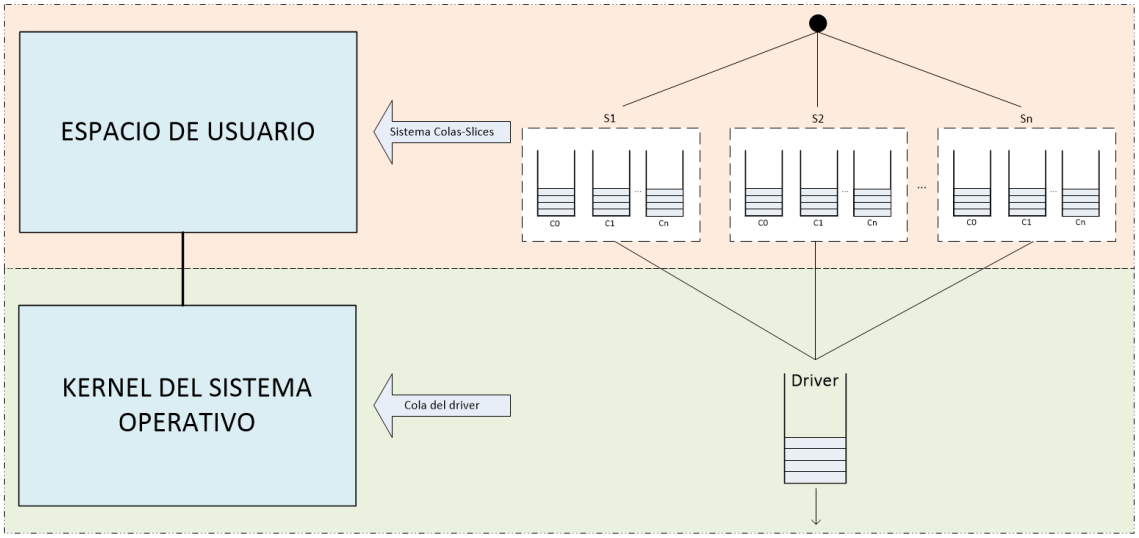


Figura 9. Representación del espacio de usuario y el kernel del S.O.

Para conseguir implementar la diferenciación de tráfico en los puntos de acceso, se diseñó en *Inymon* un sistema de colas y *slices* como se muestra en la Fig. 5. Este sistema opera en el espacio de usuario, es decir, que opera en la zona de memoria del sistema donde se ejecutan las aplicaciones y procesos que no forman parte del núcleo (*kernel*) del sistema operativo (ver Fig. 9).

Cuando un punto de acceso recibe un paquete dirigido hacia una de sus STAs asociadas, el paquete se encola en el *slice* y cola que le corresponda según su campo DSCP. Sin embargo, no es posible inyectar tráfico directamente desde el espacio de usuario. Antes, el paquete debe ser enviado a la cola del *driver* de la tarjeta de red.

El *buffer* del *driver* es una cola FIFO que opera en el *kernel* del sistema operativo. Su tamaño máximo es de 256 y si se desborda, pierde paquetes. Por esta razón, es necesario realizar un control desde el espacio de usuario para que los paquetes en cola no superen el tamaño máximo. El control del tamaño de la cola se realiza con la interfaz auxiliar 1, mencionada anteriormente, utilizando los números de secuencia de la cabecera IEEE 802.11 y funciona como se describe a continuación.

Un *contador A*, inicializado a cero, registra el número de paquetes que se han enviado desde el sistema de colas y *slices* hacia la cola del *driver*. Cada vez que se envía uno, el valor del *contador A* aumenta en una unidad. Utilizando la interfaz auxiliar, se compara el último número de secuencia registrado con el del paquete capturado en ese momento. A partir de esa comparación, se sabe cuántos paquetes se han detectado y cuántos no se han detectado.

La interfaz auxiliar *wlan1* monitoriza el canal Wi-Fi y registra el número de paquetes enviados por la interfaz principal en un *contador B*. En ocasiones, puede suceder que la interfaz espía no monitorice todas las tramas enviadas. Por ello, se utilizan los números de secuencia de la cabecera IEEE 802.11 de modo que al detectar un número 'X', todos los anteriores quedan confirmados. Es decir, al recibirse una trama con un número de secuencia determinado se puede afirmar que ésta ha salido del *buffer* del *driver* y también todas las anteriores.

Para controlar el tamaño de la cola del *driver*, la diferencia entre *Contador A* y *Contador B*, que da como resultado el tamaño del *buffer* del *driver* no puede superar el tamaño máximo permitido.

$$Contador\ A - Contador\ B\ \leq Tamaño\ Máximo\ Permitido$$

Ecuación 5. Control del tamaño del buffer del driver

En el ejemplo de la Fig. 10 se muestra una instantánea en la que el *contador A* toma un valor de 8, lo que indica que, desde que se desplegó el AP, se han enviado 8 paquetes a la cola del *driver*. En este ejemplo, el número de secuencia del primer paquete es 500, el cual ya ha sido capturado y procesado por la interfaz auxiliar. Debido a la alta densidad de tráfico, la interfaz espía no consigue capturar los paquetes con números de secuencia 501 y 502; sin embargo, sí captura el paquete con número de secuencia 503. El punto de acceso detecta la pérdida de paquetes (paquetes no monitorizados) comparando el último número de secuencia registrado (500) con el del paquete capturado (503), determinando así que se han perdido dos paquetes intermedios (no han sido monitorizados). Como resultado, el contador B se incrementa en 3 unidades, pasando de 1 a 4. Además, se verifica que el tamaño de la cola del *driver* se corresponde con la diferencia calculada según la Ecuación 5.

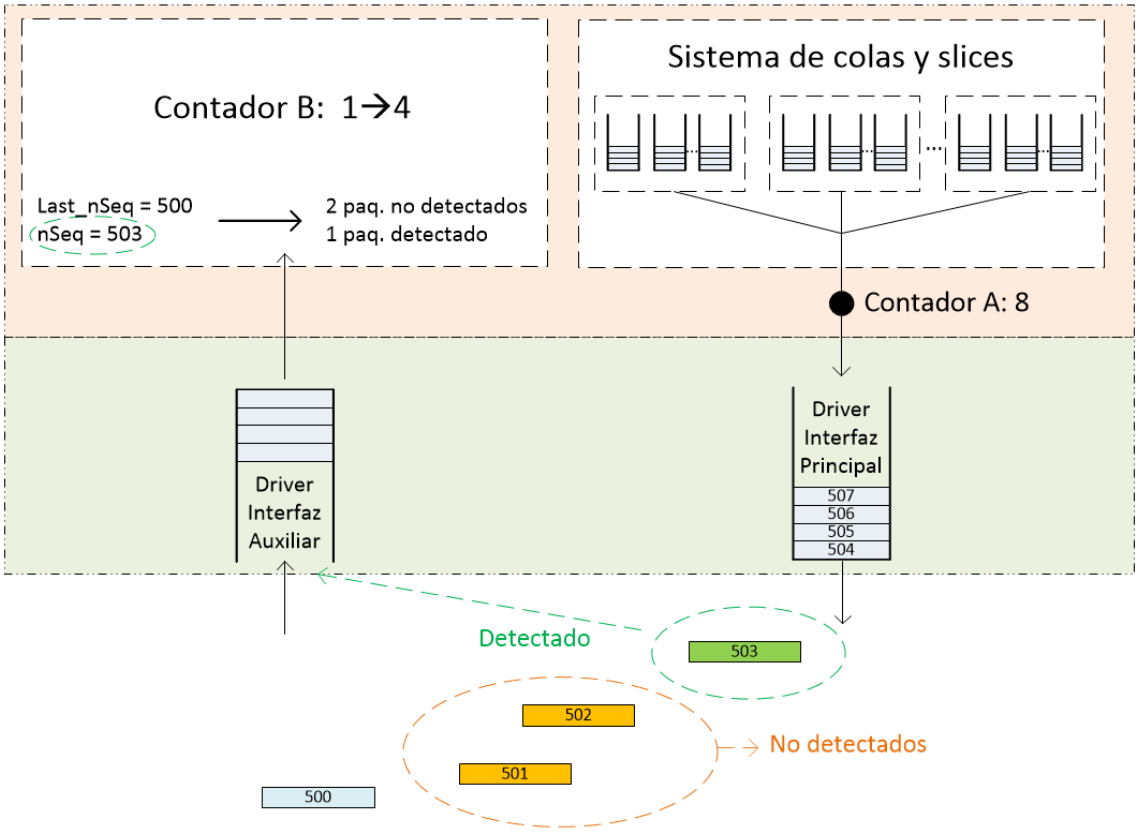


Figura 10. Esquema de funcionamiento - Control del tamaño del buffer del driver

Una vez visto el funcionamiento del algoritmo de llenado del *buffer* del *driver*, se plantea el problema encontrado en la incorporación del algoritmo de tasa variable *Onoe*. Inicialmente, para conseguir una respuesta más rápida a los cambios de MCS en la transmisión, se escogió un valor máximo del tamaño de la cola del *driver* muy reducido. Se consideró que 20 paquetes eran suficientes para soportar las pérdidas de la interfaz auxiliar, por lo tanto, *Inymon* establecía un tamaño máximo del *buffer* del *driver* de 20 paquetes.

Tras la implementación del algoritmo de control de tasa, se han realizado pruebas con alta densidad de tráfico y, en algunas realizaciones, la interfaz auxiliar no ha monitorizado secuencias de más de 20 paquetes seguidos. Esto se debe a que el punto de acceso debe llevar a cabo una gran cantidad de operaciones y cálculos adicionales, además del proceso de monitorización. La no detección de paquetes impide que paquetes nuevos entren al *buffer* del *driver*, lo que provoca que este se vacíe y deje de transmitir. Como consecuencia, la interfaz auxiliar no puede monitorizar paquetes posteriores a los 20 ya enviados, por lo que nunca se podrá dar paso a nuevos paquetes.

Dado que el tamaño máximo elegido previamente era muy reducido, se propone encontrar una solución aumentando su capacidad. El límite establecido por el *driver* de la tarjeta de red es de 256 paquetes, así que se puede elegir un valor mayor dentro de ese rango de valores.

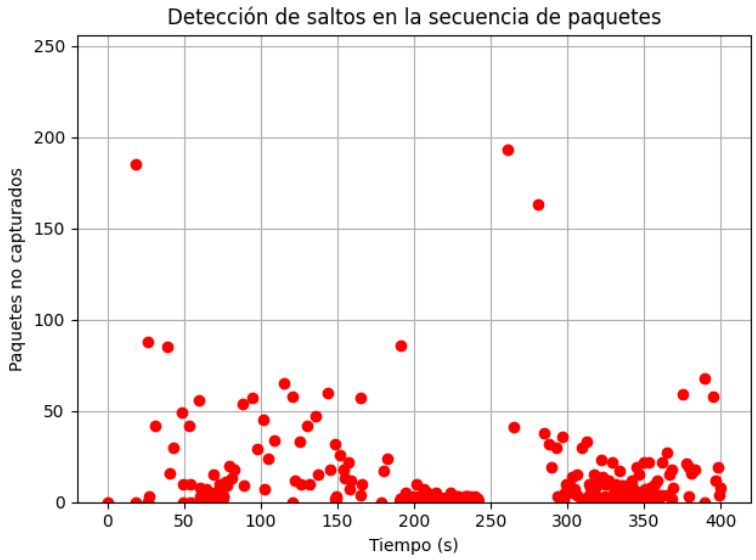


Figura 11. Paquetes no capturados en una realización de una prueba de slicing

Aunque no es necesario calcular un valor exacto para el tamaño óptimo del *buffer*, las pruebas realizadas han evidenciado que, bajo condiciones de alta carga, las pérdidas de tramas no monitorizadas pueden alcanzar valores cercanos a los 200 paquetes consecutivos. Esta observación se ve reflejada en Fig. 11, donde se representan los saltos detectados entre números de secuencia de la cabecera del estándar IEEE 802.11, lo cual permite visualizar la magnitud de las secuencias perdidas por la interfaz auxiliar.

Esto indica que, si bien un límite bajo puede ofrecer una respuesta más ágil ante cambios en el MCS, también puede generar bloqueos en el sistema cuando la monitorización no es capaz de seguir el ritmo del tráfico.

En función del estrés al que se someta el punto de acceso, pueden producirse variaciones significativas en la capacidad de monitorización, por lo que una solución más robusta consistiría en adaptar dinámicamente el tamaño del *buffer*. Como línea futura de trabajo, se propone reemplazar el umbral fijo por un mecanismo que ajuste el tamaño del *buffer* en tiempo real, permitiendo una mayor flexibilidad y resiliencia frente a variaciones en la carga del sistema. Sin embargo, por el momento se establece un umbral fijo de 200 paquetes que ha permitido desarrollar pruebas exhaustivas sin bloqueos.

3.2.2 Aparición de nuevos tráficos

Pasar de trabajar con *Inymon* a hacerlo con *NeWLAN* implica la aparición de nuevos tipos de tráfico que no estaban presentes en el entorno experimental anterior. Recordemos que *Inymon* es un entorno simplificado donde solamente se trabaja con el plano de datos. En cambio, *NeWLAN* incorpora tanto el plano de datos como el plano de control.

En la incorporación del algoritmo de *slicing* desarrollado en *Inymon*, solo se consideran las tramas de datos que llegan al punto de acceso por la interfaz *Ethernet* y deben salir por la interfaz Wi-Fi. Sin embargo, en *NeWLAN* se genera tráfico de control que también necesita ser clasificado y encolado en el sistema de *slicing*.

Los nuevos tráficos que se deben contemplar son:

- Tráfico DHCP
- Tráfico ARP
- Tráfico de gestión IEEE 802.11

En primer lugar, se debe valorar cuán importante es el tráfico de control para decidir cuántos recursos le vamos a asignar. Este tipo de tráfico es esencial porque es el que se encarga de realizar el intercambio de información antes de comenzar la transmisión de datos. Además, en condiciones normales, el tráfico de control representa una fracción muy pequeña en comparación con el tráfico de datos.

Teniendo en cuenta estos dos aspectos, se propone la solución de tratar este tráfico como prioritario. Si se retrasa su procesamiento, se estaría ralentizando el envío del tráfico de datos posterior. En cambio, si se le proporciona prioridad, no afecta negativamente al rendimiento del sistema, ya que el volumen de tráfico de control es tan bajo que no supone una carga significativa. Por tanto, se pueden priorizar las tramas de control sin perjudicar el comportamiento general del sistema en comparación con el rendimiento obtenido en *Inymon*. Para dar prioridad al tráfico de control, es necesario modificar el diseño del algoritmo de *slicing*. En el entorno de *Inymon*, el primer *slice* es el *slice* 0, que participa en el reparto de *airtime* durante la transmisión, compitiendo con el resto de *slices* activos.

En *NeWLAN*, se ha adoptado una solución diferente: el *slice* 0 se excluye del reparto de *airtime* y se reserva exclusivamente para el tráfico de control. Al quedar fuera de la competición con el resto de *slices*, se le otorga prioridad absoluta, lo que implica que, cuando llega un paquete al *slice* 0, este se sirve de forma inmediata, sin necesidad de esperar a que una estructura (cola o *slice*) termine su turno.

Para simplificar el tratamiento de este tipo de tráfico, se ha decidido utilizar una única cola dentro del *slice* 0. Esta solución se basa en que, al no haber espera, no tiene sentido aplicar una diferenciación interna de tráfico, ya que su impacto sería prácticamente inapreciable. No obstante, sería posible realizar dicha diferenciación si fuera conveniente en un futuro proyecto.

El resto de *slices* deben esperar a que el *slice* 0 termine de transmitir todos sus paquetes. No obstante, como se ha explicado anteriormente, esta espera es mínima, ya que el volumen de tráfico de control es muy reducido en comparación con el de datos. Esta afirmación se justifica de forma gráfica más adelante, en el capítulo 5, donde se muestra la pequeña proporción de *airtime* ocupada por el tráfico de control.

Tras haber decidido cómo tratar al tráfico de control, han surgido otros eventos imprevistos. Al conectar un cliente Wi-Fi a la red de *NeWLAN*, en ocasiones se han

capturado tramas en las que el intercambio de información DHCP aparece marcado con valores en el campo DSCP no contemplados.

En la captura de la Fig. 12 y Fig. 13 se observan paquetes marcados con un valor DSCP igual a 4. Este valor lo toma de manera no intencionada y no se utiliza en nuestro sistema, por lo tanto, el sistema de *slicing* de *NeWLAN* necesita conocer este nuevo caso para clasificarlo en el *slice* que le corresponde, es decir, en el *slice* 0.

Source	Destination	Protocol	Length	DSCP	Info
0.0.0.0	255.255.255.255	DHCP	392	4	DHCP Discover - Transaction ID 0x2dd5f66d
192.168.137.131	192.168.137.201	DHCP	395	4	DHCP Offer - Transaction ID 0x2dd5f66d
0.0.0.0	255.255.255.255	DHCP	392	4	DHCP Request - Transaction ID 0x2dd5f66d
192.168.137.131	192.168.137.201	DHCP	395	4	DHCP ACK - Transaction ID 0x2dd5f66d

Figura 12. Captura de Wireshark – Intercambio DHCP con valor DSCP 4

Este valor de DSCP es introducido por el *driver* de la tarjeta de red del cliente y, cuando el punto de acceso recibe una trama con ese valor DSCP, responde usando el mismo valor. Dado que el *driver* trabaja en el *kernel* del sistema operativo, no es posible modificar este valor desde el espacio de usuario (con *iptables*, por ejemplo) ya que el *driver* sobrescribirá cualquier modificación.

Para ello, se propone clasificar el tráfico de control identificando las cabeceras del paquete. Dado que resulta sencillo identificar las cabeceras de un paquete, se utilizarán para clasificar todo el tráfico de control, incluyendo DHCP y otros protocolos como ARP o IEEE 802.11, que utiliza tramas como *Beacon*, *Probe Request/Response*, *Authentication, Deauthentication*, *Association Request/Response*, *Reassociation Request/Response*, *Disassociation* y *Action*. Todos estos protocolos serán clasificados en el *slice* 0.

```
Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
▾ Differentiated Services Field: 0x10 (DSCP: Unknown, ECN: Not-ECT)
    0001 00.. = Differentiated Services Codepoint: Unknown (4)
    .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Total Length: 328
  Identification: 0x0000 (0)
▸ Flags: 0x00
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 128
  Protocol: UDP (17)
  Header Checksum: 0x3996 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 0.0.0.0
  Destination Address: 255.255.255.255
```

Figura 13. Captura de Wireshark – Cabecera IP del paquete DHCP Discover

3.2.3 Actualización del sistema de *slices* y colas por *handover*

Un aspecto clave en la integración del sistema es el impacto que produce el *handover* sobre la gestión de los *slices* y las colas. Para que un cliente Wi-Fi pueda cambiar de un punto de acceso a otro, es necesario realizar una gestión centralizada y coordinada de la información que cada AP recibe del cliente. Para ello, los APs monitorizan la potencia de señal recibida de cada cliente Wi-Fi mediante su interfaz inalámbrica *wlan2*, y es el controlador *Odin* el encargado de recopilar esa información y decidir cuál será el nuevo AP encargado de prestar servicio al cliente. Por este motivo, es necesario que el canal Wi-Fi en el que opera *wlan2* se ajuste dinámicamente al canal donde se encuentra el cliente que se desea monitorizar.

Como ya se ha comentado, se ha de tener en cuenta que el punto de acceso, a diferencia de *Inymon*, lleva a cabo una gran cantidad de operaciones y cálculos adicionales que no pueden ser dilatados en el tiempo. Cualquier instrucción que detenga la ejecución del programa puede llegar a generar resultados nocivos en funciones como, por ejemplo, el *slicing*.

La instrucción de cambio de canal es un caso concreto de lo que se ha mencionado. Esta orden se ejecutaba inicialmente mediante una llamada en C que utiliza la función *system()*, como se muestra en la Fig. 14.

```
1. system("iwconfig wlan2 channel {value}");
```

Figura 14. Instrucción de código bloqueante

La función *system()* realiza una llamada bloqueante al intérprete de comandos */bin/sh*, ejecutando, en este caso, el comando *iwconfig wlan2 channel {value}*. Esta llamada detiene la ejecución del programa principal hasta finalizar, lo que afecta negativamente a su rendimiento. Para superar esta limitación, se ha implementado una solución basada en *posix_spawn()*, que permite lanzar un nuevo proceso de manera más controlada y sin pasar por el intérprete de comandos. A diferencia de *system()*, *posix_spawn()* no bloquea la ejecución principal, lo que permite ejecutar el cambio de canal en paralelo sin interrumpir el funcionamiento del programa. Como resultado, se logra cambiar el canal de la interfaz *wlan2* de forma eficiente y sin afectar la capacidad

de respuesta del sistema. El código de esta implementación puede verse en Fig. 15.

Para medir el impacto de usar una u otra instrucción se implementa el código que puede verse en la Fig. 15. Los tiempos obtenidos validan la decisión tomada:

- *system()*: ~ 560 – 570 ms
- *posix_spawn()*: ~ 0.200 – 0.350 ms

```
1. /////////////// Measure the time taken to execute the command ////////////////////
2. struct timeval start, end;
3. gettimeofday(&start, NULL); // Start timing
4.
5. // Replace system() with posix_spawn() --> This creates a paralell process to
   execute the command
6. pid_t pid;
7. char *argv[] = {"/bin/sh", "-c", auxString, NULL}; // Execute the command using
   /bin/sh
8. extern char **environ; // Use the current environment variables
9.
10.     if (posix_spawn(&pid, "/bin/sh", NULL, NULL, argv, environ) != 0)
11.     {
12.         fprintf(stderr, ANSI_COLOR_BOLD_RED "[OdinAP-
            switch_channel_aux_interface] posix_spawn failed: %s\n" ANSI_COLOR_RESET,
            strerror(errno));
13.     }
14.
15.     gettimeofday(&end, NULL); // End timing
16.
17.     if (debugLevel % 10 >= 2)
18.     {
19.       if (debugTimestamp == 1) printMicroTime();
20.       long elapsedTime = (end.tv_sec - start.tv_sec) * 1000000L + (end.tv_usec -
                          start.tv_usec);
21.       fprintf(stderr,
                  "Command executed in %ld microseconds: %s\n",
                  elapsedTime, auxString);
22.     }
```

Figura 15. Instrucción de código no bloqueante con medición de tiempo de ejecución

Esto representa una reducción cercana al 99,95 % en el tiempo de espera, una

mejora crítica en entornos como los puntos de acceso Wi-Fi, donde es necesario gestionar múltiples tareas simultáneamente sin bloqueos. Esta optimización mejora la estabilidad del sistema, su rendimiento general y la calidad de servicio.

Otro aspecto importante es la gestión de tramas durante un *handover*. Cuando un cliente Wi-Fi cambia de canal, el sistema debe identificar que dicho cliente ya no debe recibir tramas a través del AP anterior y que estas deben redirigirse al nuevo AP asignado. Además, el AP de origen debe dejar de enviar tramas al cliente en el momento en que se le notifica que debe cambiar de canal, lo cual se realiza mediante tramas *CSA* definidas en el estándar IEEE 802.11. Si esta actualización no se lleva a cabo correctamente, las tramas seguirán enviándose por el canal anterior, generando múltiples retransmisiones, ya que el cliente no estará escuchando en esa frecuencia. Como consecuencia, las tramas que permanezcan en los *buffers* del AP antiguo deberán ser descartadas, y serán los protocolos correspondientes los encargados de gestionar su retransmisión.

Como se muestra en la Fig. 10, *OdinAP* implementa su sistema de *slices* y colas en el espacio de usuario, y las tramas extraídas de esta estructura se envían a la cola del *driver*. Como esta cola opera en el núcleo del sistema (*kernel*) y no permite eliminar paquetes una vez insertados, el único lugar viable para introducir lógica que descarte tramas obsoletas es el propio sistema de *slices* y colas en el espacio de usuario.

Para resolver este problema, se ha desarrollado un *thread* encargado de eliminar aquellas tramas que ya no deben enviarse. Este hilo recorre ordenadamente todas las colas del espacio de usuario, realizando las acciones indicadas en la Fig. 16. En el diagrama de flujo, el nodo (trama con metadatos) evaluado en cada momento se denomina *current*, y cada uno de estos nodos contiene un puntero al siguiente nodo al que se accede mediante *current→next*. Para gestionar las inserciones y eliminaciones de paquetes en la cola *q*, se utilizan dos punteros: *q→head*, que apunta al primer paquete, y *q→tail*, que apunta al último.

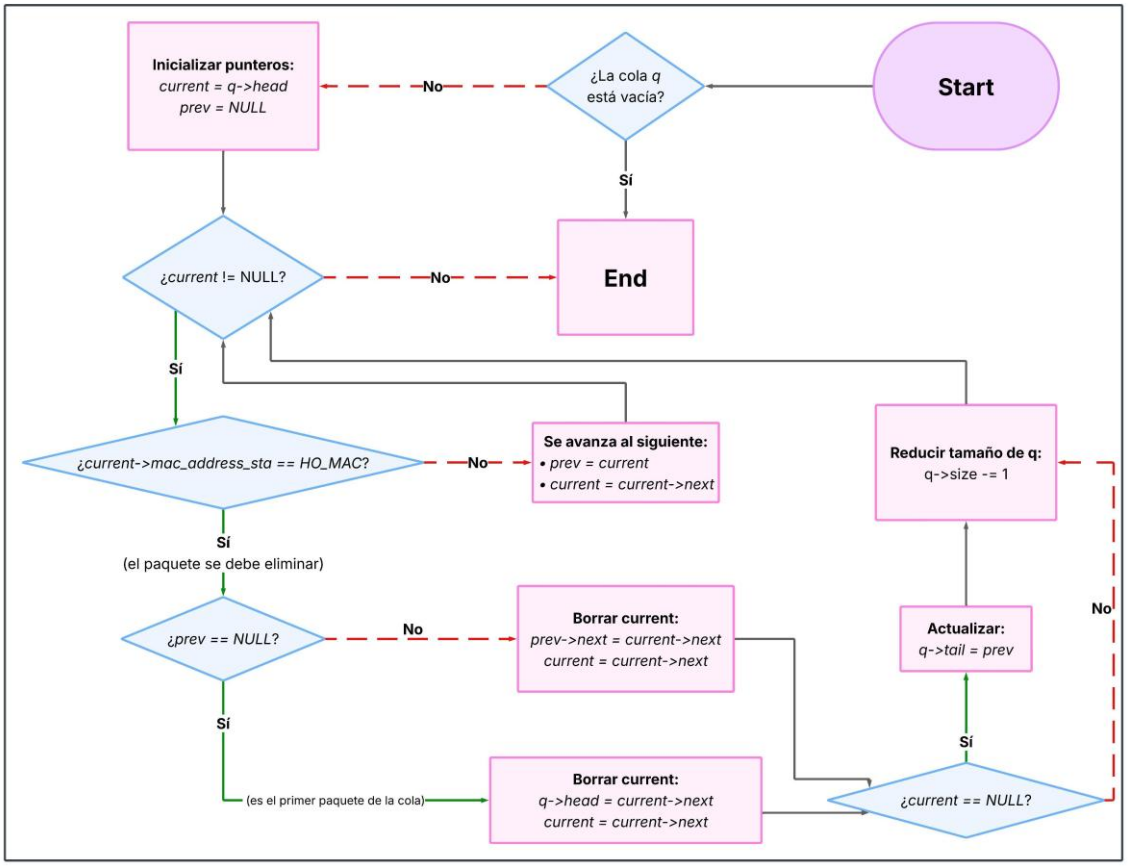


Figura 16. Diagrama de flujo del borrado de paquetes en las colas

Dado que se conoce la dirección MAC del cliente Wi-Fi que realiza el *handover*, esta se compara con la dirección de destino de cada trama en las colas. Si ambas direcciones coinciden, se elimina la trama correspondiente y se continúa con la siguiente.

Como este proceso se ejecuta en paralelo mediante un *thread*, ha sido necesario un análisis exhaustivo del código C en el entorno *OdinAP* para introducir mecanismos de sincronización (semáforos tipo *mutex*) que garanticen el acceso en exclusión mutua a *slices*, colas y nodos, evitando así interferencias con el *thread* encargado del *scheduling*.

4. PRUEBAS Y RESULTADOS

Para comprobar el correcto funcionamiento de las funcionalidades implementadas, se ha realizado una batería de pruebas que tiene como objetivo verificar que la integración se ha llevado a cabo satisfactoriamente. Para ello, se han repetido las pruebas bajo condiciones de tráfico generado similares a las utilizadas en el estudio de [13], realizado en *Inymon*. Concretamente, se han realizado cuatro experimentos: Experimento 0, Experimento 1, Experimento 2 y Experimento 3.

El Experimento 0 pretende verificar que la integración del sistema de *slicing* y del control de tasa se ha realizado correctamente, y que ambos pueden funcionar de manera simultánea sin problemas. Además, se incluye un análisis detallado del funcionamiento del algoritmo *ADWRR* con pesos estáticos.

Por otro lado, el Experimento 1 se centra en comprobar el comportamiento de las variantes del algoritmo *ADWRR* aplicadas al *scheduling intra-slice*. Para un mismo patrón de generación de tráfico, el experimento recoge los resultados obtenidos utilizando las siguientes variantes: pesos estáticos, igual ratio de satisfacción y priorización al índice menor. El Experimento 2 tiene un objetivo similar al del Experimento 1, pero centrándose en el funcionamiento del *scheduling inter-slice*.

Por último, el Experimento 3 pretende verificar el funcionamiento del sistema de *slicing* junto con el *handover*. Para ello, se duplica el escenario y un cliente Wi-Fi realiza el traspaso de un AP a otro.

Los resultados de estas pruebas han sido obtenidos a partir de la recopilación de datos durante la ejecución del código. Los datos se almacenan en ficheros de texto, que posteriormente son procesados mediante un *script* de *Matlab* para extraer las representaciones gráficas y estadísticas que se muestran en los siguientes apartados.

4.1 EXPERIMENTO 0 – CONTROL DE TASA Y ADWRR CON PESOS ESTÁTICOS

En el Experimento 0 se generaron ocho flujos UDP con el objetivo de comprobar el correcto comportamiento de las funcionalidades implementadas. Para ello, se emplearon las características mostradas en la Tabla 3. Se utilizaron seis clientes Wi-Fi situados dentro de un radio menor de dos metros con respecto al punto de acceso. Por lo

tanto, todos los clientes contaron con un buen RSSI (*Received Signal Strength Indicator*).

Tabla 3. Experimento 0. Parámetros de interés

	***Slice* 1**			***Slice* 2**		***Slice* 3**		
Quantum (*Qnom*)	3500μs			2500μs		4000μs		
Queues	**Queue 0**	**Queue 1**		**Queue 0**	**Queue 1**	**Queue 0**	**Queue 1**	**Queue 2**
Weight (*Wnom*)	50	50		30	70	50	30	20
STA id	*STA 1*	*STA 6*	*STA 3*	*STA 2*	*STA 3*	*STA 4*	*STA 5*	*STA 6*
MCS index	3	6	1	Variable	1	2	4	6
Iperf Rate (Time interval)	1.4 Mbps (0-50 s)	1.4Mbps (0-50 s)	1.4Mbps (0-50 s)	1.4Mbps (0-50 s)	1.4Mbps (0-50 s)	1.4 Mbps (0-10s), 0.5 Mbps (10-20 s), 1.4 Mbps (20-30 s), 0.5 Mbps (30-50 s)	1.4 Mbps (0-30s) 0.5 Mbps (30-50 s)	1.4 Mbps (0-30s) 0.5 Mbps (30-50 s)
UDP size (bytes)	250B	1250 B	650 B	500 B	250 B	250 B	250 B	400 B
Airtime (μs)	281.5μs	345.5μs	625.5μs	-	377.5μs	313.5μs	249.5μs	241.5μs
Airtime required parameters	*Band = 5GHz, BW = 20MHz, T_{DIFS} = 34μs, T_{SIFS} = 16μs., $T_{Backoff}$ = T_{slot}·CW/2 with CW = CWmin = 15 , T_{phyOH} (11a)=20μs,* *T_{ACK} (24Mbps) =28*μs, *T_{slot} = 9μs, T_{phyOH} (mixed) = 36μs + 4μs ($n_{antenna}$ – 1) , T_{phyOH} (greenfield) = 28μs + 4μs ($n_{antenna}$ – 1)* *MAC_{OH} = $L_{service}$ (2B) + L_{HMAC} (24B) + L_{QoS} (2B) + L_{FCS} (4B), L_{LLCOH} (8B)* *Ldata (xB) = IP (20B) + UDP (8B) + DATA*							

Cada cliente Wi-Fi tiene un valor de MCS configurado. Todos los clientes tienen MCS fijos, excepto la STA 2, que tiene un MCS variable controlado por el algoritmo de control de tasa *Onoe*. Así, la STA 2 se ha configurado para utilizar valores de MCS entre 7 y 15.

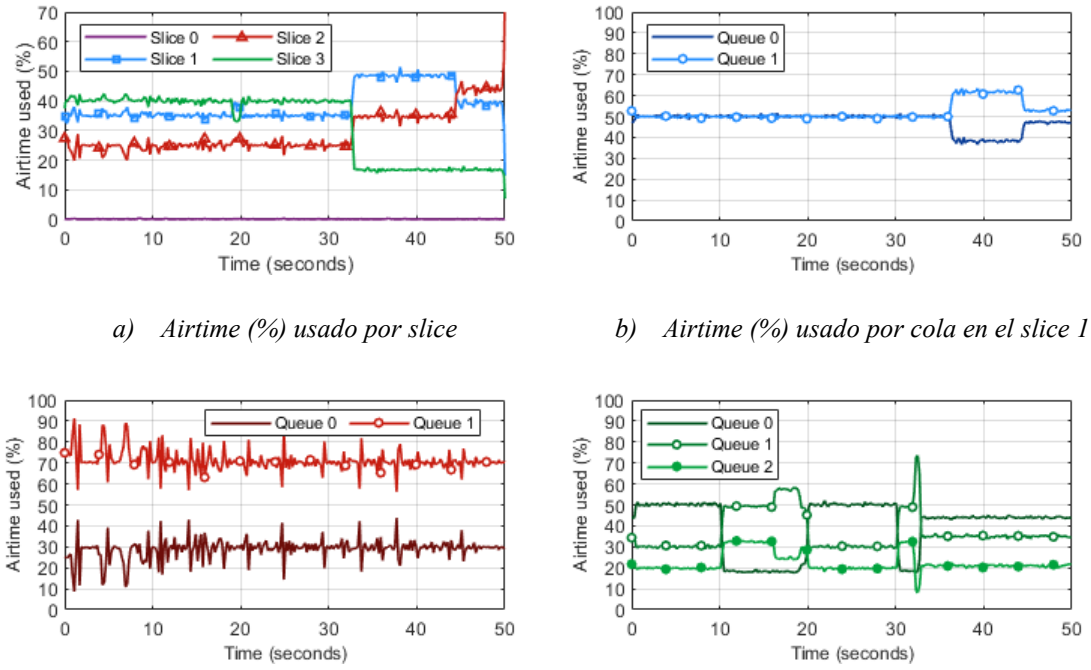
Este experimento utiliza el algoritmo ADWRR con pesos o quantum estáticos. Esto quiere decir que, como se ha comentado en el capítulo 2, los recursos excedentes se reparten proporcionalmente al *quantum*, que permanece fijo.

El tráfico se genera con la herramienta de Linux, *Iperf*, que genera paquetes UDP

siguiendo una distribución uniforme. Cada flujo UDP tiene un conjunto de parámetros que lo caracterizan: la tasa o *throughput* (a nivel de transporte), el tamaño de los paquetes, la duración, la dirección IP destino y el puerto destino. El puerto destino es el parámetro que define el criterio de clasificación, es decir, según el valor del puerto destino, los paquetes de un flujo se dirigen a un *slice* y cola determinados.

Se han configurado tres *slices* para el tráfico correspondiente al plano de datos: *slice* 1, *slice* 2 y *slice* 3. Los dos primeros utilizan dos colas: *Queue* 0 y *Queue* 1; y el tercer *slice* utiliza tres colas.

A continuación, realizadas las pruebas del Experimento 0, se analizan los resultados obtenidos de la Fig. 17. Si se presta atención a los datos desde t=0s hasta t=10s, se observa que el reparto de *airtime* entre *slices* cumple con las condiciones esperadas.



a) Airtime (%) usado por slice

b) Airtime (%) usado por cola en el slice 1

c) Airtime (%) usado por cola en el slice 2

d) Airtime (%) usado por cola en el slice 3

Figura 17. Experimento 0. Airtime consumido por cada slice y sus respectivas colas

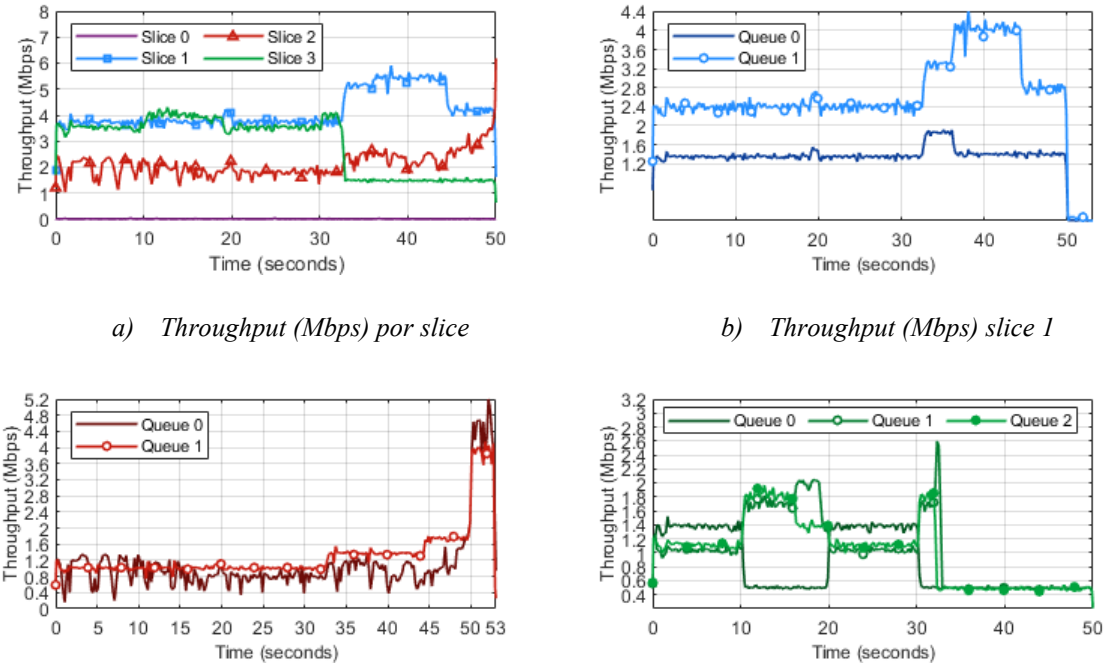
Los porcentajes de *airtime* consumidos por *slice* son: 35% para el *slice* 1, 25% para el *slice* 2 y 40% para el *slice* 3. Estos valores se corresponden con la configuración establecida para los valores de quantum: 3500µs, 2500µs y 4000µs, respectivamente. Esto confirma que el sistema es capaz de controlar el reparto de *airtime* en situaciones de

congestión, asignando recursos de forma proporcional al valor del *quantum* asignado a cada *slice*.

Además, se puede observar cómo el *slice* 0, destinado al tráfico de control, consume un porcentaje de *airtime* prácticamente imperceptible en comparación a los *slices* asignados al tráfico de datos. De esta manera, se comprueba la efectividad de la solución propuesta, ya que permite dar prioridad al tráfico de control sin que este interfiera ni limite los recursos disponibles para el tráfico de datos.

Para demostrar que, efectivamente, el sistema se encuentra en estado de congestión durante este intervalo, puede comprobarse en la Fig. 18 que ningún *slice* logra transmitir la tasa generada. En concreto, a cada uno de los *slices* entran flujos de *Iperf* a las siguientes tasas:

- ***Slice* 1:** 1.4 Mbps × 3 = 4.2Mbps
- ***Slice* 2**: 1.4 Mbps × 2 = 2.8Mbps
- ***Slice* 3:** 1.4 Mbps × 3 = 4.2Mbps



a) Throughput (Mbps) por slice

b) Throughput (Mbps) slice 1

c) Throughput (Mbps) slice 2

d) Throughput (Mbps) slice 3

Figura 18. Experimento 0. Throughput utilizado por cada slice y sus respectivas colas

Sin embargo, el *slice* 1 alcanza aproximadamente los 3.8 Mbps, el *slice* 2 ronda los

1.8 Mbps y el *slice* 3 se acerca a los 3.6 Mbps. Cabe destacar que, a lo largo de todo el experimento, el *airtime* consumido por el *slice* 2 presenta una mayor fluctuación, ya que tiene mayores desviaciones respecto a la media en comparación con el resto de *slices*. Este comportamiento se debe a que, en el *slice* 2, se ha utilizado una STA móvil con el objetivo de modificar dinámicamente su MCS mediante el algoritmo de control de tasa. La movilidad supone variaciones significativas en las condiciones del canal Wi-Fi, que, a su vez, provocan retransmisiones, tal como se muestra en la Fig. 19, lo que afecta a la estabilidad del *throughput* de la cola y, en consecuencia, a la del *slice*.

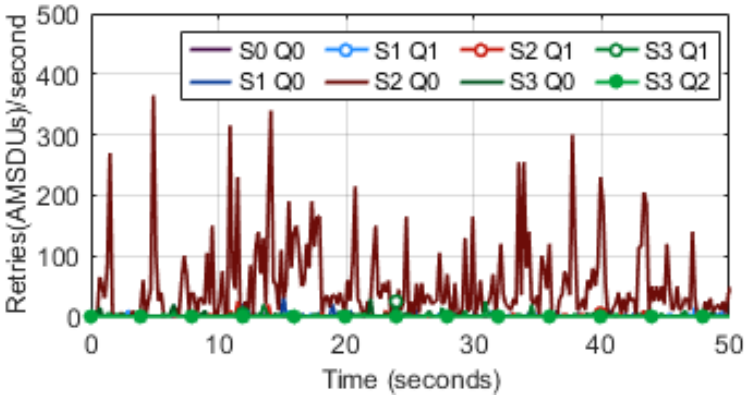


Figura 19. Experimento 0. Retransmisiones experimentadas

Atendiendo a los resultados obtenidos desde t=10s hasta t=20s, se observa en la Fig. 18.d una reducción del tráfico generado hacia la cola 0 del *slice* 3, pasando de 1.4 Mbps a 0.5 Mbps. Por lo tanto, el *airtime* consumido por esta cola se reduce, ya que ahora puede satisfacer completamente la nueva tasa de entrada. En la Fig. 17.d se observa el descenso del consumo de *airtime* del 50% inicial al 19% correspondiente a la nueva tasa de entrada.

Como la cola 1 y la cola 2 no recibían suficiente *airtime* para satisfacer su tasa de entrada, aprovechan los recursos que libera la cola 0. Dado que se emplea el algoritmo ADWRR con quantum y pesos estáticos, el reparto de *airtime* entre las colas 1 y 2 se realiza en proporción a los pesos asignados a cada una.

$$Extra\ Airtime(\%)_{C1} = (50 - 19)\% \times \frac{30}{30 + 20} = 18.6\%$$

$$Extra\ Airtime(\%)_{C2} = (50 - 19)\% \times \frac{20}{30 + 20} = 12.4\%$$

La nueva distribución de *airtime* en el *slice* 3 es aproximadamente: 19% la cola 0, 48.6% la cola 1 y 32.4% la cola 2. Esta nueva distribución permite que la cola 2 alcance un *throughput* de 1.8 Mbps (ver Fig. 18.d); sin embargo, la tasa de entrada de esa cola es de 1.4 Mbps. Esto indica que la cola 2 está vaciando su *buffer*, en el que había acumulado paquetes durante los primeros diez segundos debido a la falta de recursos suficientes en ese intervalo. En el segundo t=16s, esta cola consigue transmitir todos los paquetes acumulados y reduce su *throughput* a un valor igual a la tasa de entrada: 1.4 Mbps.

Al reducir el *throughput*, se reduce al mismo tiempo el consumo de *airtime*, lo que libera recursos para otra cola aún insatisfecha. Es el caso de la cola 1, que había acumulado más paquetes que la cola 2 y sigue vaciando su *buffer*. Como se muestra en la Fig. 18.d, desde el instante t=16s hasta el t=19s, aumenta su *throughput* y, en consecuencia, su consumo de *airtime*.

Mientras tanto, el consumo de *airtime* del *slice* 3 se ha mantenido constante, ya que, aunque una cola haya reducido su demanda, otra ha utilizado ese *airtime* en cuanto ha tenido la oportunidad. Por esta razón, el sistema se ha mantenido constantemente saturado y el reparto de *airtime* no ha variado.

En el periodo comprendido entre t=20s y t=30s el sistema recibe flujos UDP con las mismas características que en el primer periodo. En consecuencia, la cola 0 del *slice* 3 sirve los 1.4 Mbps que le llegan, mientras que la cola 1 y la cola 2 no disponen de recursos suficientes y acumulan paquetes en el *buffer*. El sistema no puede repartir suficientes recursos, se encuentra saturado.

A partir del instante t=30s hasta el final del experimento, se reduce el *throughput* de entrada a las colas del *slice* 3. Cada una de ellas pasa de recibir 1.4 Mbps a recibir 0.5 Mbps. En la Fig. 18.d se observa un periodo transitorio en el que las colas 1 y 2 incrementan su *throughput* para vaciar los paquetes acumulados en el anterior periodo. Como ocurrió anteriormente, la cola 2 logra vaciar su *buffer* antes que la cola 1. Finalmente, todas las colas estabilizan su *throughput* a 0.5 Mbps.

En ese instante, el *slice* 3 usa menos recursos *quantum* del que dispone, por lo que el déficit excedente se desecha y se continúa con el siguiente *slice*. Esto permite que el resto de *slices* puedan enviar paquetes más a menudo que en el estado de saturación, por lo tanto, aumentará el *airtime* de cada *slice* y, en consecuencia, el *throughput*.

La reducción del *airtime* del *slice* 3 es de, aproximadamente, un 23%. Este excedente se repartirá en el resto de *slices* que lo necesiten, de manera proporcional al *quantum*. Así, tal y como se refleja en la Fig. 17.a, el *slice* 1 se llevará un 13.4% y el *slice* 2 se llevará un 9.6%.

El *slice* 1 utiliza el 13.4% del *airtime* adicional para aumentar su *throughput* y vaciar el *buffer*, tanto de su cola 0 como de la cola 1. En torno al instante t=33s, aumenta el *throughput* de ambas colas manteniéndose el reparto de *airtime*, pues ambas siguen usando todos los recursos garantizados. A partir del instante t=36s, la cola 0, que recibe menos tráfico, consigue vaciar el *buffer* y ya no necesita utilizar todos los recursos. De esta forma, la cola 1 podrá utilizarlos para vaciar su *buffer* en un menor tiempo. Es en ese momento cuando, como se muestra en la Fig. 17.b, la cola 1 alcanza un 60% de consumo de *airtime*. Una vez la cola 1 haya vaciado el *buffer*, se reducirá su *throughput* al valor de la tasa de entrada, 2.8 Mbps. En ese instante, el *slice* 1 no necesitará utilizar los recursos recibidos, por lo que el *slice* 2, que sí que los necesita, podrá aprovecharlos.

Atendiendo al *throughput* conseguido por el *slice* 2 en la Fig. 18.c, se puede observar que en ningún instante del experimento este *slice* va a conseguir vaciar el *buffer* de alguna de sus colas. Se mantiene en todo momento en estado de saturación, con un reparto de *airtime* constante en media, a pesar de sus desviaciones ruidosas. Además, puede observarse en la Fig. 18.c cómo el *slice* 2 necesita tres segundos más para vaciar sus *buffers* por completo una vez ha dejado de recibir paquetes, puesto que en el instante t=50s el experimento termina. A partir del segundo 50, todos los *slices* han vaciado sus colas, excepto el *slice* 2, que empleará todos los recursos disponibles para vaciarlas.

En la Fig. 20 se observa la variación del MCS de la STA en cuestión, que ha conseguido índices más grandes cuando ha dispuesto de mejores condiciones del canal, e índices menores conforme se ha alejado del punto de acceso.

Analizando los resultados obtenidos en el Experimento 0, se puede observar la implementación correcta del algoritmo de control de tasa y del sistema de *scheduling* del algoritmo ADWRR con pesos y quantum estáticos.

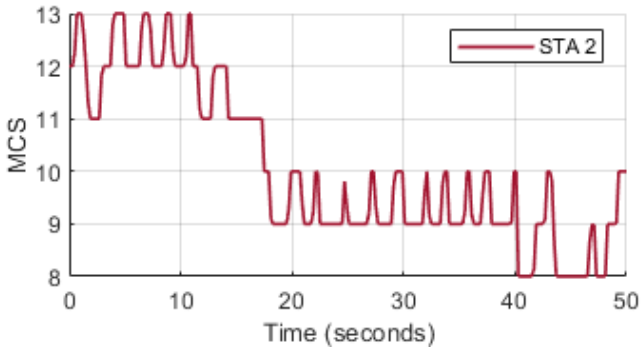


Figura 20. Experimento 0. Variación MCS de la STA 2 (slice 2, cola 0)

4.2 EXPERIMENTO 1 - VARIANTES DE ADWRR *INTRA-SLICE*

Este experimento se realiza con el objetivo de verificar el correcto funcionamiento de las distintas variantes del algoritmo de planificación ADWRR *intra-slice*. En este, se compara la variante del algoritmo de pesos estáticos, con el de igual ratio de satisfacción y con el de preferencia al índice menor para el reparto *intra-slice*. El objetivo es mostrar el correcto comportamiento de los algoritmos y demostrar que aseguran una distribución de recursos eficiente según el criterio seleccionado. Para el algoritmo ADWRR *inter-slice* se utiliza la variante de *quantum* estático. Los parámetros empleados en ambos casos se recogen en la Tabla 4.

En este experimento se generan nueve flujos UDP con una duración de 70 segundos, utilizando la agregación de tramas MAC (AMSDU, *Aggregated MAC Service Data Unit*). Esto permite obtener valores de *throughput* mayores.

Para este experimento se vuelven a utilizar tres *slices* dedicados exclusivamente al tráfico de datos. El *slice* 1 tendrá garantizado el 30% del *airtime*, el *slice* 2 el 20% y el *slice* 3 el 50%. Además, se introduce el concepto de *MBR* (*Maximum Bit Rate*), que actúa como un parámetro de limitación de la tasa de salida. El MBR es una característica propia de cada cola dentro de cada *slice* y puede estar definido por un SLA.

Tabla 4. Experimento 1. Parámetros de interés

	***Slice* 1**		***Slice* 2**			***Slice* 3**			
Quantum (*Qnom*)	3000μs		2000μs			5000μs			
Queues	**Queue 0**	**Queue 1**	**Queue 0**		**Queue 1**	**Queue 0**	**Queue 1**	**Queue 2**	**Queue 3**
Weight (*Wnom*)	120	80	140		60	70	60	40	30
MBR	2.5Mbps	2Mbps	2.5Mbps		2Mbps	3Mbps	2.5Mbps	2.5Mbps	2Mbps
STA id	STA 1	STA 3	STA 2	STA 1	STA 4	STA 1	STA 2	STA 6	STA 5
Priority index	0	1	0		1	0	1	2	3
MCS index	2	4	3	2	6	2	3	3	4
Iperf Rate (Time interval)	2.5 Mbps (0-70 s)	2 Mbps (0-70 s)	1 Mbps (0-70 s)	0.8 Mbps (0-70 s)	1.2 Mbps (0-70 s)	1.2 Mbps (0-10 s) 2,8 Mbps (10-40 s) 3,3 Mbps (40-70 s)	2.5Mbps (0-10 s) 2 Mbps (10-20 s) 1 Mbps (20-60 s) 1.3 Mbps (60-70 s)	1.3 Mbps (0-10 s) 1.6 Mbps (10-30 s) 2.3 Mbps (30-70 s)	1.25 Mbps (0-50 s) 0.5 Mbps (50-70 s)
UDP size (bytes)	1250B	250 B	200 B	350 B	700 B	350 B	350 B	200B	250B
Max Aggregation Size	1200B	1200B	1200B		1200B	1200B	1200B	1200B	1200B

Este parámetro MBR nos permite calcular el ratio de satisfacción que tiene una cola determinada, que se define como el ratio que compara la tasa conseguida frente a la tasa demandada (la tasa de entrada al AP) o al MBR:

- Si la tasa de entrada es menor que el MBR, el ratio de satisfacción se calcula como: $R_{achieved}/\ R_{demanded}$.
- Si la tasa de entrada es mayor que el MBR, se debe adaptar el concepto de satisfacción con las limitaciones establecidas por el SLA y se calcula como: $R_{achieved}/MBR$.

De esta forma, el ratio de satisfacción representa cómo de satisfecho están los flujos de una cola en relación con los recursos que podrían y deberían tener.

Una vez descritos los parámetros que se emplean en el experimento, se realiza el

siguiente análisis. Al observar la distribución de *airtime* de la Fig. 21, se puede comprobar que el sistema se encuentra en estado de saturación. Durante todo el experimento, los *slices* utilizan el porcentaje máximo de *airtime* garantizado en la configuración. De esta forma, resulta más sencillo profundizar en el análisis de los algoritmos de *scheduling intra-slice*.

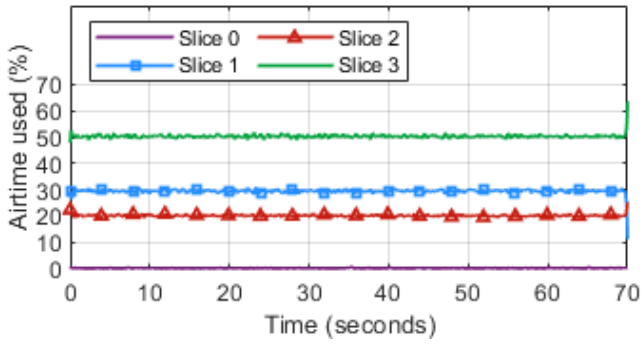
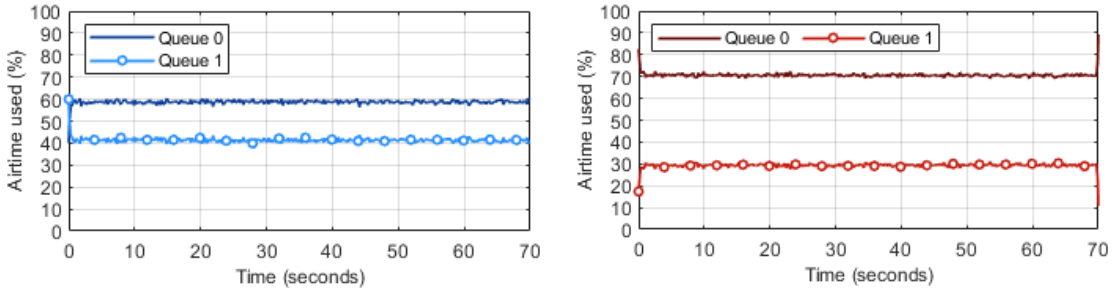


Figura 21. Experimento 1. Distribución de airtime por slice

En la Fig. 22, se observa el reparto de *airtime* dentro de los *slices* 1 y 2. En este experimento, la distribución *intra*-slice de ambos *slices* permanece constantemente en saturación, es decir, cada cola utiliza su *airtime* garantizado. Así, se procede a realizar el estudio del *slice* 3 en profundidad y sin la influencia de modificaciones en el reparto de recursos de otras estructuras.



a) Airtime slice 1 *b) Airtime slice 2*

Figura 22. Experimento 1. Distribución de airtime del slice 1 y slice 2

Para comprobar el correcto funcionamiento y comparar los algoritmos de scheduling intra-slice, se ha repetido el Experimento 1 en tres ocasiones: una vez con cada versión del algoritmo ADWRR. La Fig. 23 presenta los resultados obtenidos del *slice* 3 en cada realización. La Fig. 22.A1 presenta de forma visual de los patrones de

generación de tráfico del experimento para ese *slice*. Para el *scheduling inter-slice* se configura el algoritmo de quantum estático

En el intervalo que comprende desde t=0 hasta t=10 se observa que todas las STAs consiguen recibir el *throughput* deseado. A diferencia del porcentaje de *airtime* garantizado (C0:35%, C1:30%, C2:20%, C3:15%), el sistema distribuye el *airtime* excedente de la cola 0 a aquellas que lo necesiten, modificando así el reparto (C0:24%, C1:37%, C2:22%, C3:17%).

En el intervalo comprendido entre t=10 y t=20, tres de las cuatro colas del *slice* 3 reciben un *throughput* distinto al del periodo anterior (Fig. 23.A1). Este cambio provoca que ninguna de ellas esté en condiciones de ceder recursos, algo que puede comprobarse al observar las Fig. 23 A3, B3 y C3, donde se aprecia que todas las colas consumen el máximo *airtime* disponible según sus pesos. En estas condiciones, todas las variantes del algoritmo *ADWRR* implementadas arrojan resultados idénticos, ya que su funcionamiento se basa en repartir *airtime* excedente, el cual en este caso no existe.

En el instante $t = 20$, el *throughput* de la cola 1 (asociada a la STA 2) se reduce a la mitad (Fig. 23.A1). A partir de ese momento, el *scheduler* distribuye el *airtime* sobrante de la cola 1 entre las colas restantes. Si se aplica el algoritmo de pesos estáticos (Fig. 23.A), el comportamiento del sistema es similar al observado en el Experimento 0: las colas insatisfechas (cola 0 y cola 2) incrementan su *airtime* de forma proporcional a sus pesos. En el caso del algoritmo de prioridad al índice menor (Fig. 23.C), se observa un aumento en el *DS* de la cola con mayor prioridad, la cola 0 (Fig. 23.C2). No obstante, la reasignación de peso a la cola 0 no ocurre de forma inmediata, ya que el sistema calcula las estadísticas con una frecuencia de un segundo. Este retardo provoca que, al inicio del periodo, tanto la cola 0 como la cola 2 presenten un aumento de *DS* similar al observado con el algoritmo de pesos estáticos. Sin embargo, una vez el sistema ajusta los pesos y el algoritmo converge, se alcanza el reparto esperado.

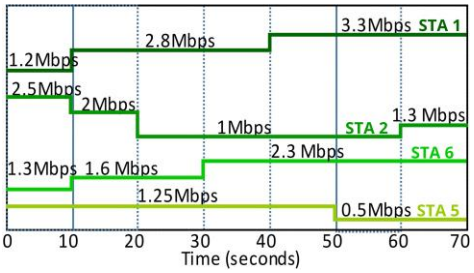
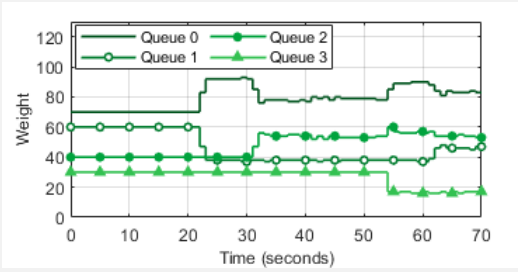
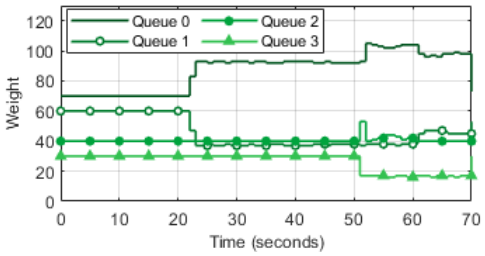
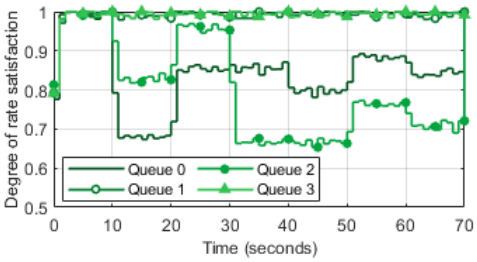
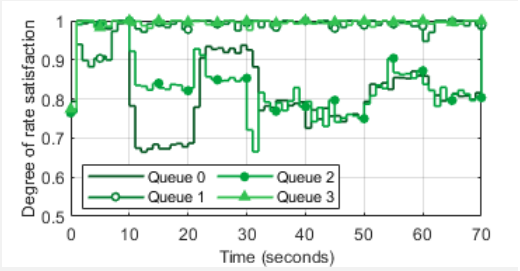
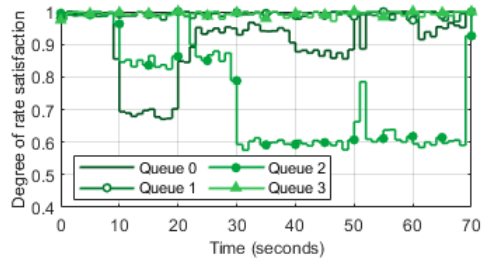
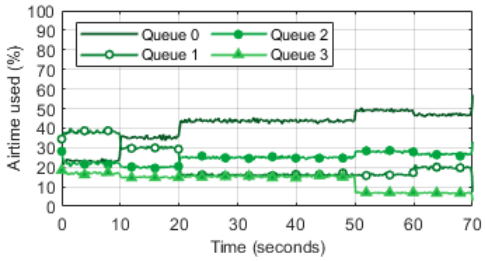
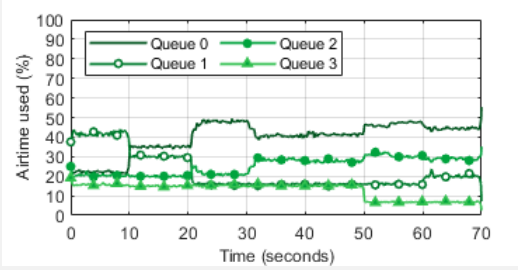
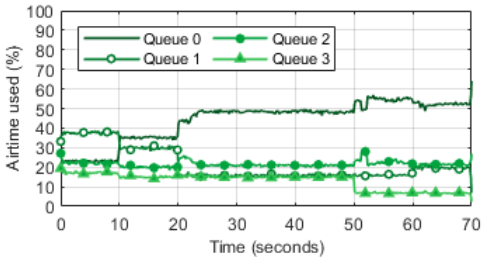
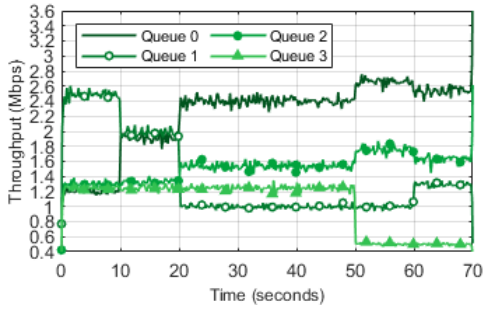
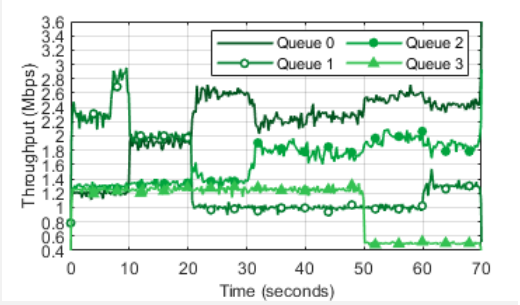
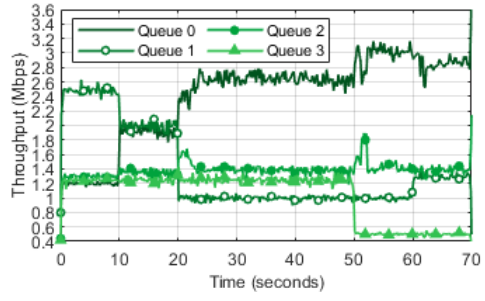
Pesos/Quantum estático	Igual ratio de satisfacción	Preferencia al índice menor
		
A1) Patrones de generación de tráfico slice 3	*B1) Evolución de los pesos slice 3*	*C1) Evolución de los pesos slice 3*
		
A2) Grado de satisfacción slice 3	*B2) Grado de satisfacción slice 3*	*C2) Grado de satisfacción slice 3*
		
A3) Reparto de airtime slice 3	*B3) Reparto de airtime slice 3*	*C3) Reparto de airtime slice 3*
		
A4) Throughput slice 3	*B4) Throughput slice 3*	*C4) Throughput slice 3*

Figura 23. Representación de estadísticas del slice 3 – Experimento 1

Entre t=30 y t=50 se observan ligeras variaciones. En $t = 30$ aumenta la tasa de entrada de la cola 2 y, en t=40, la de la cola 1. Bajo el algoritmo de pesos estáticos (Fig. 23.A), la cola 2 experimenta una disminución en su grado de satisfacción, ya que no logra aumentar su *throughput* al ya encontrarse utilizando tanto su *airtime* nominal como el excedente previamente asignado. El algoritmo de igualación del *DS* reajusta los pesos para que las colas insatisfechas mantengan un valor de *DS* similar (Fig. 23, B2). Es importante señalar que, nuevamente, existe un retardo entre la detección del *DS* y la reconfiguración de los pesos, lo que genera breves periodos transitorios donde el comportamiento del sistema se asemeja al del algoritmo de pesos estáticos. En el caso del algoritmo de prioridad por índice, se incrementa el peso de la cola 0 para mejorar su *DS*, manteniendo a la cola 2 (con un índice superior) insatisfecha.

En el instante t=50, la cola 3 reduce su *throughput*, liberando recursos que pueden ser aprovechados por el resto de las colas. El algoritmo de pesos estáticos reparte nuevamente el *airtime* de forma proporcional, mientras que el algoritmo de igual *DS* reajusta los pesos para mantener la equidad entre colas insatisfechas. Por su parte, el algoritmo de prioridad al índice menor continúa asignando más peso a la cola 0. Cabe destacar que, tras t=50, la cola 0 recibe suficiente peso como para alcanzar un *throughput* igual a su *MBR* (3 Mbps). No obstante, en t=60, la cola 1 incrementa su *throughput* y recupera parte de los recursos previamente cedidos a la cola 0. Esta variación se refleja en la Fig. 23, C1, donde se observa una reducción del peso asignado a la cola 0 en favor de la cola 1.

4.3 EXPERIMENTO 2 – VARIANTES DE ADWRR *INTER-SLICE*

Este experimento se realiza con el objetivo de verificar el correcto funcionamiento de las distintas variantes del algoritmo de planificación ADWRR *inter-slice*[4]. En este, se compara la variante del algoritmo de pesos estáticos, con el de igual ratio de satisfacción y con el de preferencia al índice menor en el reparto de *airtime* a nivel de *slice*. El objetivo

[4] Para el algoritmo ADWRR *intra-slice* se utiliza la variante de igual ratio de satisfacción

es mostrar el correcto comportamiento de los algoritmos y demostrar que aseguran una distribución de recursos eficiente según el criterio seleccionado. Los parámetros empleados se recogen en la Tabla 5.

Tabla 5. Experimento 2. Parámetros de interés

	***Slice* 1**		***Slice* 2**			***Slice* 3**			
Quantum (*Qnom*)	3000μs		2000μs			5000μs			
Queues	**Queue 0**	**Queue 1**	**Queue 0**		**Queue 1**	**Queue 0**	**Queue 1**	**Queue 2**	**Queue 3**
Weight (*Wnom*)	120	80	140		60	70	60	40	30
MBR	2.5Mbps	2Mbps	2.5Mbps		2Mbps	3Mbps	2.5Mbps	2.5Mbps	2Mbps
STA id	STA 1	STA 3	STA 2	STA 1	STA 4	STA 1	STA 2	STA 6	STA 5
Priority index	0	1	0		1	0	1	2	3
MCS index	2	4	3	2	6	2	3	3	4
Iperf Rate (Time interval)	2.5 Mbps (0-10 s) 1 Mbps (10-60 s) 2 Mbps (60-70s)	2 Mbps (0-20 s) 3 Mbps (20-30 s) 1 Mbps (30-70 s)	1.6 Mbps (0-70 s)	1.2 Mbps (0-70 s)	1.2 Mbps (0-20 s) 2 Mbps (20-70 s)	1.2 Mbps (0-10 s) 2,8 Mbps (10-40 s) 3,3 Mbps (40-70 s)	2.5Mbps (0-10 s) 2 Mbps (10-20 s) 1 Mbps (20-60 s) 1.3 Mbps (60-70 s)	1.3 Mbps (0-10 s) 1.6 Mbps (10-30 s) 2.3 Mbps (30-70 s)	1.25 Mbps (0-50 s) 0.5 Mbps (50-70 s)
UDP size (bytes)	1250B	250 B	200 B	350 B	700 B	350 B	350 B	200B	250B
Max Aggregation Size	1200B	1200B	1200B		1200B	1200B	1200B	1200B	1200B

De forma similar al Experimento 1, en el Experimento 2 se generan nueve flujos UDP con una duración de 70 segundos y vuelve a utilizarse la agregación de tramas. Dichos flujos son, en una parte, similares a los del Experimento 1 ya que tan solo se han modificado la duración y el *throughput* de los flujos UDP del *slice* 1 y el *slice* 2. El resto de parámetros tienen el mismo valor.

Para este experimento, se vuelven a utilizar tres *slices* dedicados exclusivamente al tráfico de datos. El *slice* 1 tendrá garantizado el 30% del *airtime*, el *slice* 2 el 20% y el *slice* 3 el 50%. De forma similar al Experimento 1, se utilizan los conceptos de MBR y

grado de satisfacción.

En Fig. 24.A1 se muestran los patrones de generación de tráfico del *slice* 1 y del *slice* 2. Como los flujos UDP para el *slice* 3 son los mismos que para el experimento anterior, se puede consultar cuando sea conveniente en Fig. 23.A1. A continuación, se analizan los resultados obtenidos en cada uno de los algoritmos ADWRR.

Desde el inicio del experimento hasta t=10, el sistema se encuentra saturado. Todos los *slices* usan el quantum nominal asignado y no pueden ceder nada. Por lo tanto, el *airtime* consumido por cada uno se reparte proporcionalmente al quantum (S1:30%, S2:20%, S3:50%).

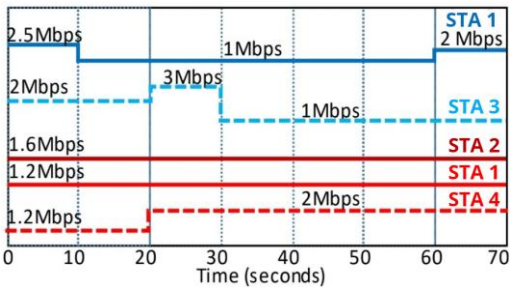
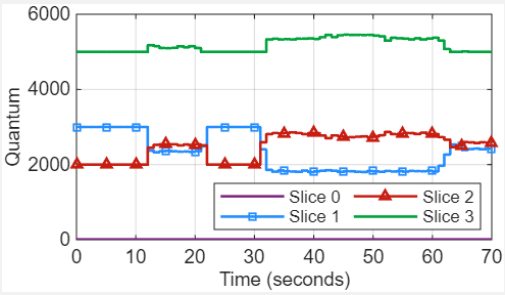
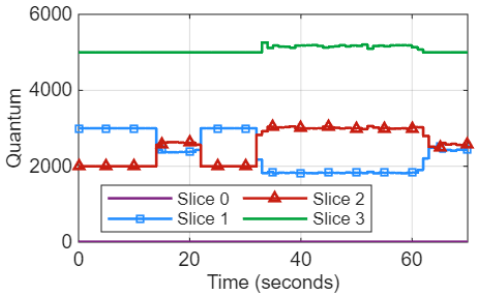
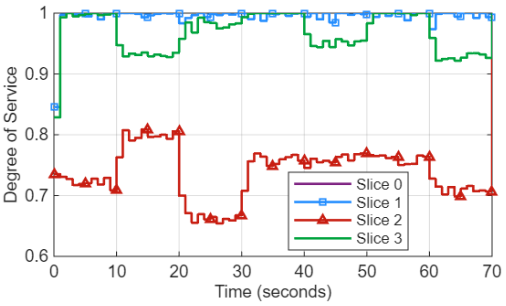
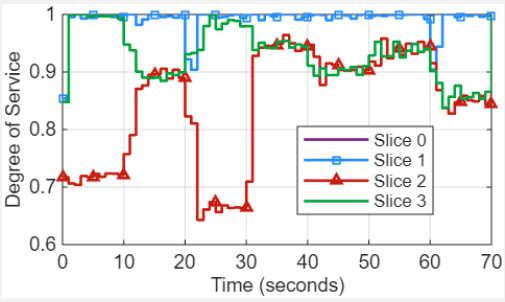
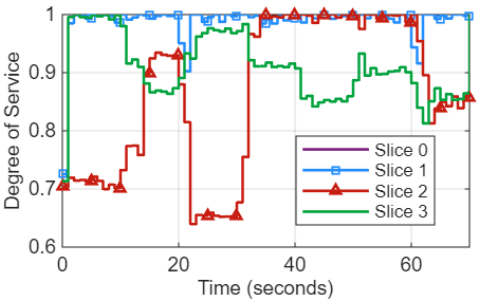
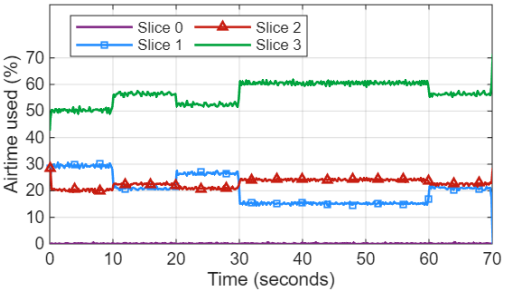
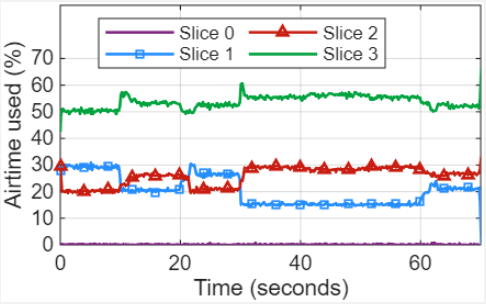
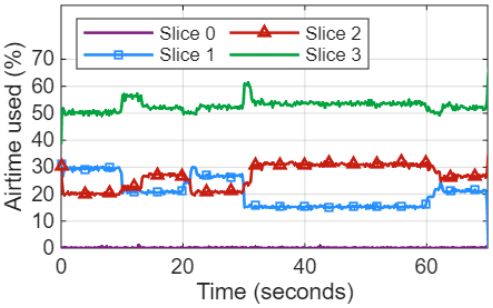
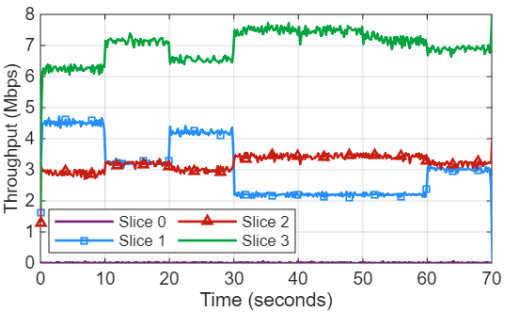
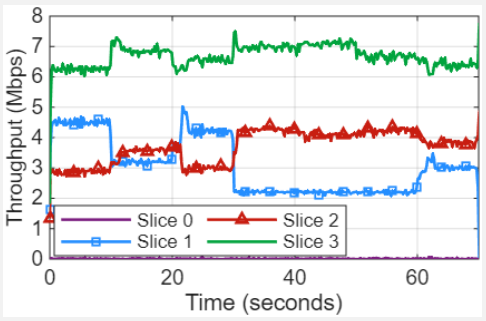
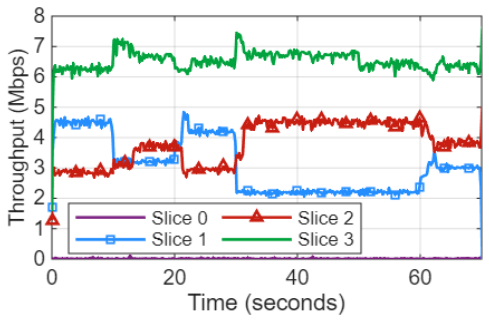
En el intervalo desde t=10 hasta t=20, el *throughput* de la cola 0 del *slice* 1 disminuye (Fig. 24 A3, B3, C3). Esto reduce el consumo de *airtime* del *slice* 1 y le permite ceder recursos al resto un 10% de *airtime*. Cuando se usa ADWRR con pesos estáticos el *slice* 2 consigue un 3.75% y el *slice* 3, que ha aumentado su demanda de *airtime*, recibe un 6,25%, que resulta al calcular la proporción con el quantum nominal entre los dos *slices*. Para cuando se utiliza el mismo ratio de satisfacción, se comprueba en Fig. 24.B2 que el *slice* 2 y el *slice* 3 alcanzan un mismo DS ajustando dinámicamente su *quantum* (Fig. 24.B1). Es preciso explicar que el grado de satisfacción del *slice* 3 se reduce en este intervalo porque aumenta su *throughput* un total de 1.4 Mbps. Por lo tanto, se alcanza un *throughput* de entrada de 7.65 Mbps, que no se consigue transmitir y el DS baja hasta un 90% (6.9 Mbps transmitido / 7.65 Mbps entrada). Al usar la variante de preferencia al índice menor, se observa en Fig. 24.C1 un aumento en el *slice* 2, que es el que menor índice tiene. Cabe destacar que en t=10 existe un corto desfase temporal entre el cambio de los flujos UDP y la modificación de los pesos. Esto, como en ocasiones anteriores, produce momentáneamente un ajuste de *airtime* no intencionado.

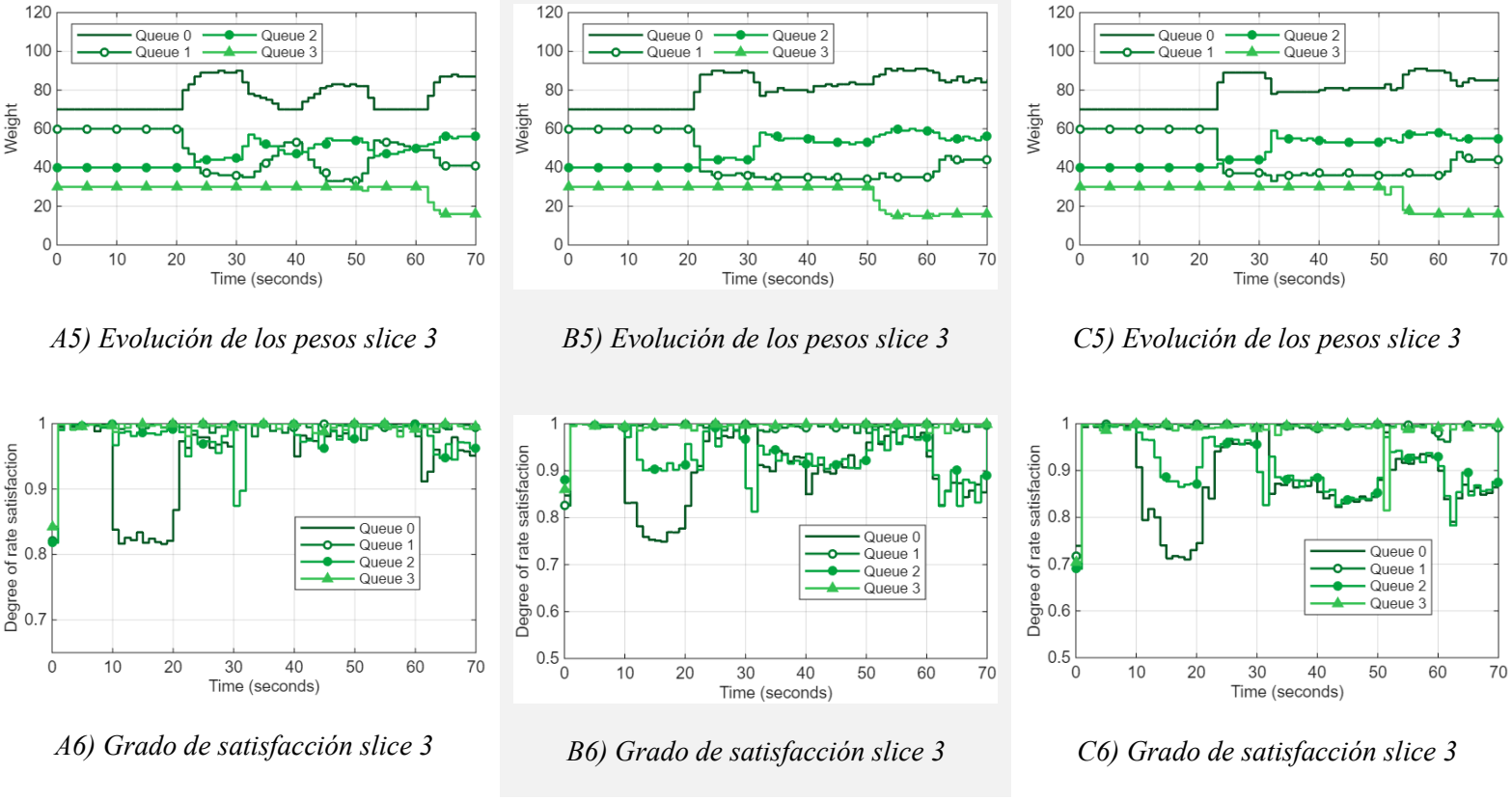
Entre t=20 y t=30, el *throughput* de entrada del *slice* 1 aumenta 1 Mbps, el del *slice* 2 aumenta 0.8 Mbps y el del *slice* 3 disminuye 1 Mbps (Fig. 24.A1, Fig. 23.A1). Para todas las versiones de ADWRR, el *slice* 1 recupera casi todos los recursos que había prestado (Fig. 24 B1 y C1) y con ello consigue transmitir los 4 Mbps del *throughput* de entrada (Fig. 24 A4, B4, C4). Por lo tanto, el *slice* 2 ve reducido su *airtime* y no consigue transmitir los 4.8 Mbps de *throughput* de entrada (Fig. 24 A4, B4, C4). Por último, el *slice* 3 disminuye el *throughput* de entrada y pierde gran parte del *airtime* cedido

anteriormente por el *slice* 1. Por lo tanto, este *slice* disminuye su *throughput* y consigue prácticamente transmitir el *throughput* de entrada (ver ratio de satisfacción Fig. 24 A2, B2, C2). Como en este periodo de 10 segundos no se ceden recursos, los resultados obtenidos son prácticamente los mismos para las tres pruebas.

Desde t=30 hasta t=50, el *slice* 1 vuelve a ceder recursos al disminuir su *throughput* de entrada 2 Mbps. De nuevo se realiza un reparto de *airtime* proporcional al *quantum* nominal para ADWRR con *quantum* constante (Fig. 24.A3) y las otras dos pruebas distribuyen los recursos siguiendo su criterio de reparto. Para cuando se utiliza el igual ratio de satisfacción, el *slice* 2, que es el más insatisfecho (Fig. 24.B), recibe gran parte del *quantum* del *slice* 1 hasta igualar su DS con el *slice* 3. El *slice* 3 también recibe *quantum* y, al aumentar el *throughput* de entrada al mismo tiempo, se reduce su DS. En t=40 el *slice* 3 aumenta el *throughput* de entrada en 0.5 Mbps y esto se traduce en una reducción del DS de los *slices* insatisfechos (Fig. 24.B2) puesto que deben tomar valores similares. Para ADWRR con preferencia al índice menor, se le asigna todo el *quantum* al *slice* 2 hasta que esté totalmente satisfecho (Fig. 24.C2). Los recursos que todavía queden libres se repartirán al *slice* 3, que no podrá transmitir el *throughput* de entrada en ningún momento de este periodo.

En los últimos 20 segundos del experimento, el *slice* 3 disminuye su demanda de *throughput* en 0.5 Mbps (Fig. 23.A1). Esto le permite mejorar su DS utilizando en las 3 pruebas los mismos recursos (Fig. 24 A2, B2, C2). Por esta razón, las estadísticas del resto de *slices* permanecen constantes excepto en Fig. 24.B2, donde el *slice* 2 consigue aumentar levemente su DS por la política de igual ratio de satisfacción. En el instante t=60, se produce un aumento de demanda de *throughput* en el *slice* 1 y el *slice* 3. Por ello, el *slice* 1 se ve obligado a recuperar parte de los recursos cedidos (Fig. 24 B1, C1) y el resto de *slices* verán afectado su DS debido a esta modificación (Fig. 24 B2, C2).

Pesos/Quantum estático	Igual ratio de satisfacción	Preferencia al índice menor
		
A1) Patrones de generación de tráfico	*B1) Evolución de los pesos*	*C1) Evolución de los pesos*
		
A2) Grado de satisfacción	*B2) Grado de satisfacción*	*C2) Grado de satisfacción*
		
A3) Reparto de airtime	*B3) Reparto de airtime*	*C3) Reparto de airtime*
		
A4) Throughput	*B4) Throughput*	*C4) Throughput*



A5) Evolución de los pesos slice 3

B5) Evolución de los pesos slice 3

C5) Evolución de los pesos slice 3

A6) Grado de satisfacción slice 3

B6) Grado de satisfacción slice 3

C6) Grado de satisfacción slice 3

Figura 24. Representación de estadísticas – Experimento 2

4.4 EXPERIMENTO 3 – FUNCIONAMIENTO DE *SLICING* CON *HANDOVER*

Para evaluar el funcionamiento del *slicing* junto con el de *handover*, se ha ajustado el escenario de trabajo. En este experimento se utilizan dos puntos de acceso ubicados a una distancia suficiente para que la STA realice el *handover* en el punto medio entre ambos APs. Como se muestra en la Fig. 25, se emplean tres equipos físicos: dos *PCs Intel NUC* y una *Raspberry Pi 4*. Se ha introducido un *PC Intel NUC2* como el dispositivo que alberga el segundo punto de acceso. La STA 4, que comenzará conectada al AP del *PC Intel NUC1*, se utiliza como terminal móvil.

Para este experimento se utiliza la configuración de *quantum* y pesos estáticos tanto en el *scheduling intra-slice* como en el *inter-slice*. Aprovechando esta configuración, se ha tratado de aproximar la ocupación del enlace a la del Experimento 0, generando flujos de *Iperf* similares. Para obtener resultados comparables, se ha ajustado el *throughput* de algunos flujos UDP con el fin de compensar las diferencias en los MCS de las STAs utilizadas en este experimento respecto a las del Experimento 0.

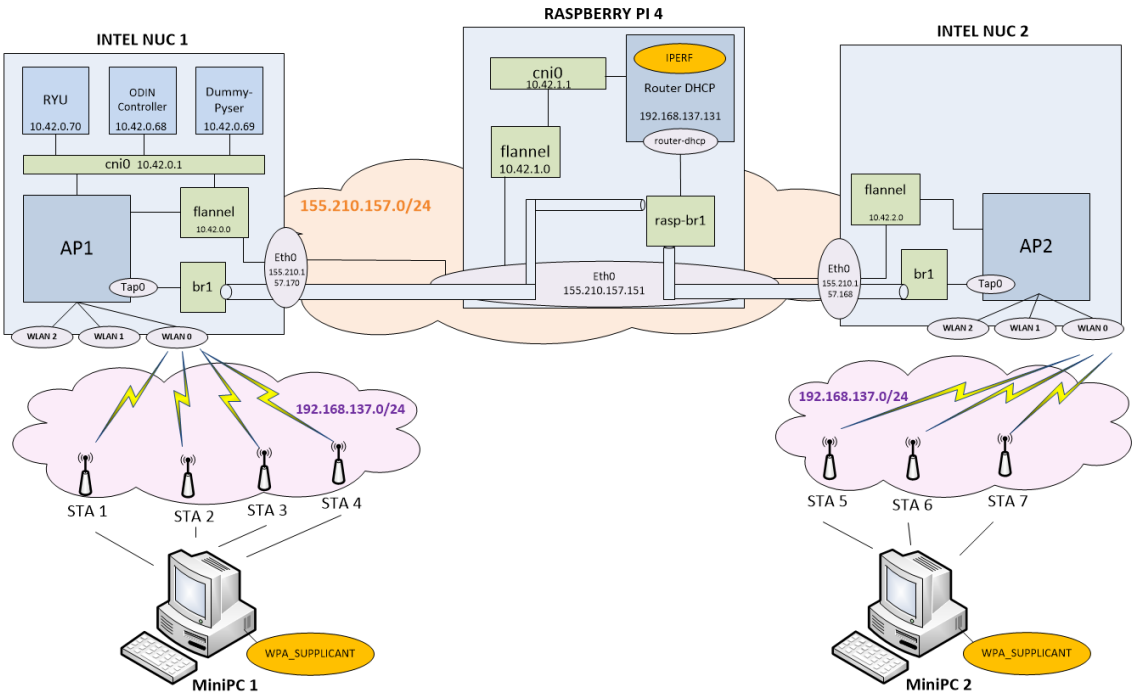


Figura 25. Escenario de trabajo para handover a nivel lógico

La Tabla 6 recoge los parámetros relevantes del AP1, que opera en el canal 36, mientras que la Tabla 7 muestra los del AP2, que opera en el canal 48. Ambas tablas incluyen los flujos correspondientes a la STA 4. Aunque estos flujos (de la STA 4) aparezcan reflejados en las dos tablas, solo serán enviados por el AP que esté prestando servicio a la STA 4 en cada momento.

El resto de las STAs se reparten equitativamente: la mitad se asocian con el AP1 y la otra mitad con el AP2. De este modo, se consigue que ambos puntos de acceso transmitan flujos UDP equivalentes, permitiendo así una comparación equilibrada del rendimiento entre ellos.

Tabla 6. Experimento 3. Parámetros de interés AP1

	Slice 1			**Slice 2**		**Slice 3**					
Quantum (*Qnom*)	3500µs			2500µs		4000µs					
Queues	**Queue 0**	**Queue 1**		**Queue 0**	**Queue 1**	**Queue 0**		**Queue 1**		**Queue 2**	
Weight (*Wnom*)	50	50		30	70	50		30		20	
STA id	*STA 2*	*STA 3*	*STA 1*	*STA 2*	*STA 1*	*STA 2*	*STA 4*	*STA 3*	*STA 4*	*STA 3*	*STA 4*
MCS index	3	6	1	3	1	3	4	6	4	6	4
Iperf Rate (Time interval)	1.4 Mbps (0-100 s)	1.4Mbps (0-100 s)	1.4Mbps (0-100 s)	1.4Mbps (0-100 s)	1.4Mbps (0-100 s)	1.6 Mbps (0-10s), 0.55 Mbps (10-20 s), 0.68 Mbps (20-100 s)	1.2 Mbps (20-80 s)	0.68 Mbps (0-100s)	0.9 Mbps (0-100s)	0.6 Mbps (0-100s)	0.9 Mbps (0-100s)
UDP size (bytes)	250B	1250 B	650 B	500 B	250 B	250 B	250 B	250 B	250 B	400 B	400 B

Tabla 7. Experimento 3. Parámetros de interés AP2

	Slice 1			**Slice 2**		**Slice 3**					
Quantum (*Qnom*)	3500µs			2500µs		4000µs					
Queues	**Queue 0**	**Queue 1**		**Queue 0**	**Queue 1**	**Queue 0**		**Queue 1**		**Queue 2**	
Weight (*Wnom*)	50	50		30	70	50		30		20	
STA id	*STA 16*	*STA 17*	*STA 15*	*STA 16*	*STA 15*	*STA 16*	*STA 4*	*STA 17*	*STA 4*	*STA 17*	*STA 4*
MCS index	3	6	1	3	1	3	4	6	4	6	4
Iperf Rate (Time interval)	1.4 Mbps (0-100 s)	1.4Mbps (0-100 s)	1.4Mbps (0-100 s)	1.4Mbps (0-100 s)	1.4Mbps (0-100 s)	1.6 Mbps (0-10s), 0.55 Mbps (10-20 s), 0.68 Mbps (20-100 s)	1.2 Mbps (20-80 s)	0.68 Mbps (0-100s)	0.9 Mbps (0-100s)	0.6 Mbps (0-100s)	0.9 Mbps (0-100s)
UDP size (bytes)	250B	1250 B	650 B	500 B	250 B	250 B	250 B	250 B	250 B	400 B	400 B

Los flujos UDP de la Tabla 6 y Tabla 7 comienzan en el mismo instante de manera sincronizada en los dos puntos de acceso. En t=45 la STA 4 realiza el *handover* del AP1 al AP2 y en t=80 vuelve al AP1. A continuación, se realiza un análisis de los resultados

obtenidos que se muestran en la Fig. 26.

Dado que la STA 4 recibe tres flujos UDP con un *throughput* total de 3 Mbps y que estos flujos solo se envían a través del AP que la atiende en cada momento, el *slice* 3 únicamente estará en condiciones de ceder recursos cuando la STA 4 se encuentre conectada al otro AP (ver Fig. 26 A1 y B1). Esto se refleja en el comportamiento del AP1 durante los intervalos t=0 a t=45 y t=80 a t=100, periodos en los que este AP da servicio a la STA 4. De forma inversa, el *slice* 3 del AP2 no podrá liberar recursos entre t=45 y t=80, ya que en ese intervalo es el encargado de transmitir los flujos a dicha estación. Cabe destacar que estos periodos pueden extenderse ligeramente si existen paquetes acumulados en las colas, ya que estos deben ser transmitidos antes de que el sistema esté en condiciones de ceder *airtime* a otros *slices*.

Centrando la atención en el instante del handover en t=45, se observa en la Fig. 26.A1 una caída en el consumo de *airtime* causada por la salida de la STA 4. Al mismo tiempo, el AP2 comienza a servir a la STA 4, lo que incrementa su consumo de *airtime*. En algunas gráficas se puede observar que, cuando la STA 4 no está presente, el sistema consume menos recursos y los *slices* con mayor demanda pueden aumentar su *quantum*. Por esta razón, se producen también cambios en otros *slices* que no transmiten los paquetes dirigidos a la STA 4 (ver Fig. 26 A2, A3, B2, B3).

Centrando la atención en el *slice* 3 durante el *handover*, se aprecia en la Fig. 26 A5 y B5 una disminución del *throughput* en el AP1 y un aumento en el AP2. En el momento del *handover* en el AP1, ninguna de las colas está en condiciones de ceder *airtime*, ya que todas utilizan sus recursos máximos asignados (C0: 50 %, C1: 30 %, C2: 20 %). Sin embargo, justo después del *handover*, se produce una bajada en el *throughput*, acompañada de una redistribución del *airtime* (Fig. 26.A4), en la que se obtiene un 45 % para la cola 0, un 35 % para la cola 1 y un 20 % para la cola 2.

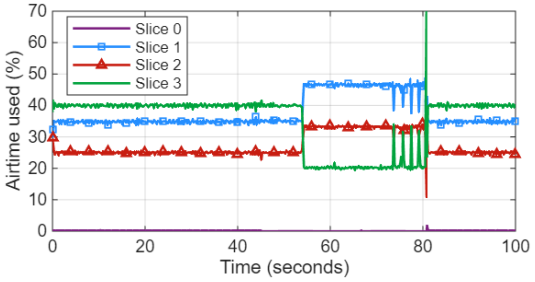
Esa misma distribución es la que presentan las colas del *slice* 3 unos instantes antes del *handover*. Como los flujos UDP han sido replicados en ambos APs (excepto los de la STA 4), se observa un comportamiento simétrico en el reparto del *airtime*, dado que, tras el *handover*, el reparto obtenido es el que corresponde al caso en el que ninguna cola está en condiciones de ceder recursos (de nuevo, C0: 50 %, C1: 30 %, C2: 20 %).

En la Fig. 26 A6 y B6 se puede ver un aumento del número de retransmisiones en

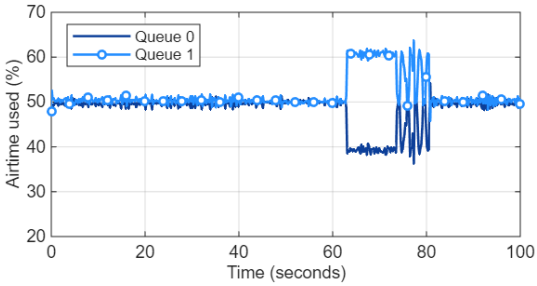
el instante del *handover* en t=45. En el AP1, estas retransmisiones se prolongan más en el tiempo, debido a que la STA 4 se aleja progresivamente del punto de acceso, reduciendo la potencia de señal y dificultando la comunicación. Además, el hecho de que el MCS permanezca constante durante toda la prueba para todas las STAs impide adaptarse a las condiciones variables del canal. Unos instantes después, también se producen retransmisiones en el AP2, aunque más concentradas en el tiempo y de mayor volumen. Esto ocurre porque, cuando *OdinAP* notifica a la STA que debe cambiar de canal, esta no lo hace de forma inmediata, lo que puede generar retardo e incluso periodos en los que la STA no escucha ni el canal nuevo ni el anterior (en el cambio de canal). Como resultado, se producen retransmisiones debido a que la STA no genera los ACKs correspondientes a las tramas que comienzan a enviarse por el nuevo canal. Estas retransmisiones tienen un impacto negativo en el *throughput* de salida de ambos APs, ya que se consume *airtime* en el envío de tramas que no llegan a su destino.

Más adelante, en t=80, se produce el mismo evento, pero en sentido inverso: la STA 4 se desplaza del AP2 al AP1. Nuevamente, se observa en la Fig. 26 B5 y B6 una caída del *throughput* en todas las colas, provocada por las retransmisiones generadas al alejarse del punto de acceso. En el AP1, además, se registran algunas variaciones en el *throughput* en los instantes previos al *handover*. Este comportamiento se debe a un cúmulo de retransmisiones de origen desconocido en el *slice* 1, que alteran la estabilidad del reparto de *airtime*. Una vez finalizado el *handover*, vuelven a aparecer retransmisiones debido al retardo con el que la STA 4 cambia de canal. Finalmente, cuando la STA empieza a escuchar en el nuevo canal, se recupera la estabilidad en el sistema de reparto.

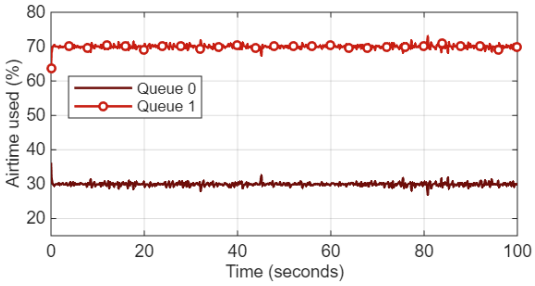
AP1 – Canal 36



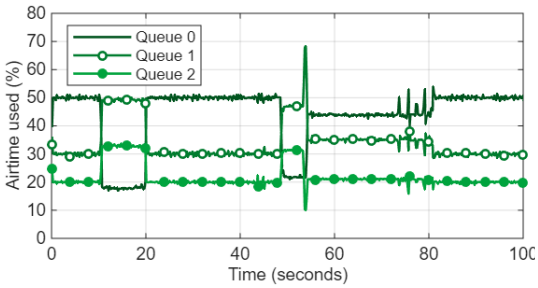
A1) Airtime por slice, AP1



A2) Airtime slice 1, AP1

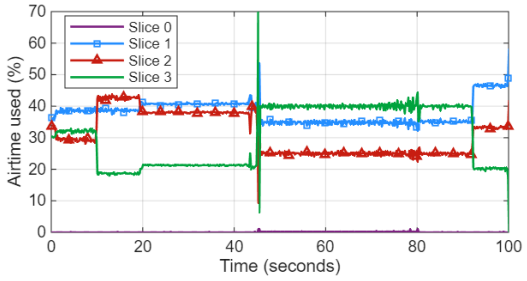


A3) Airtime slice 2, AP1

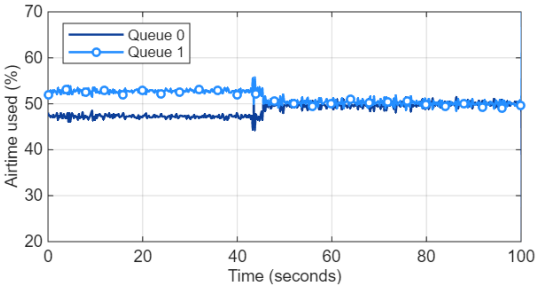


A4) Airtime slice 3, AP1

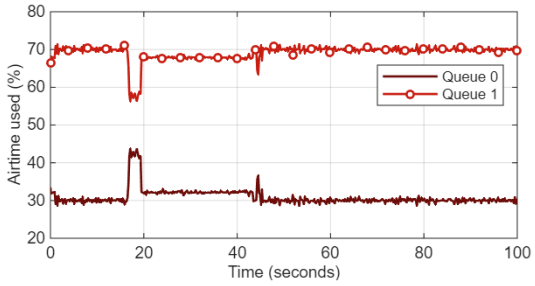
AP2 – Canal 48



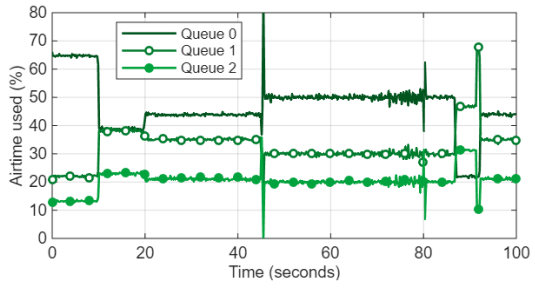
B1) Airtime por slice, AP2



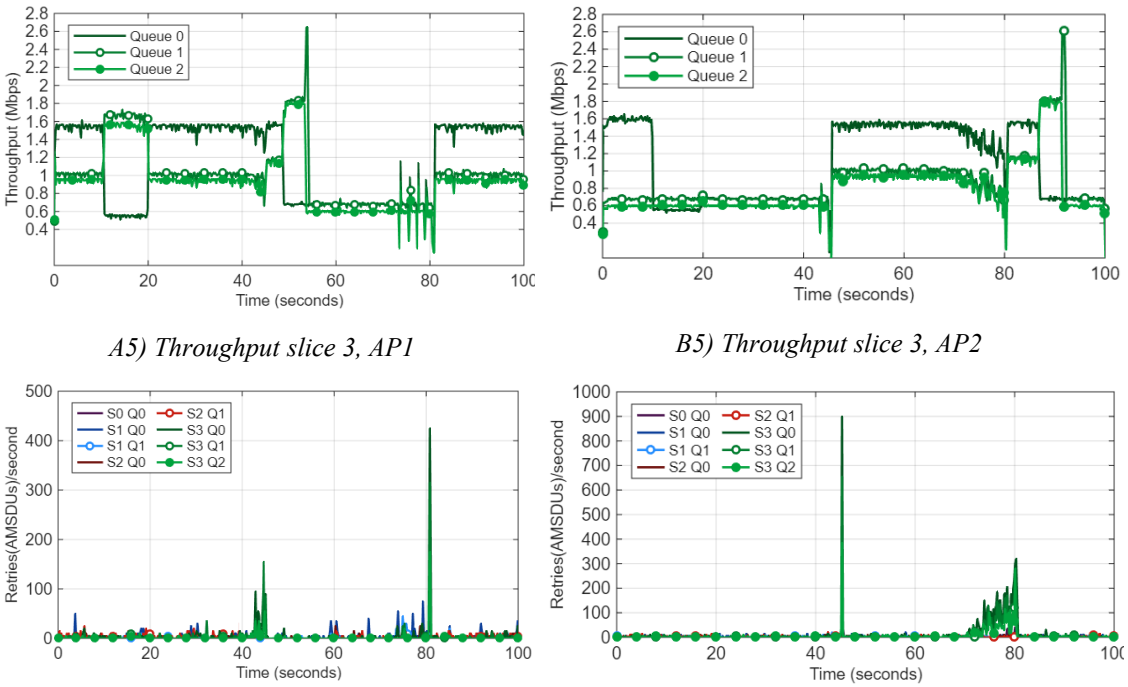
B2) Airtime slice 2, AP2



B3) Airtime slice 2, AP2



B4) Airtime slice 3, AP2



A5) Throughput slice 3, AP1

B5) Throughput slice 3, AP2

A6) Retransmisiones de AMSDU, AP1

B6) Retransmisiones de AMSDU, Ap2

Figura 26. Representación de gráficas – Experimento 3

Como se ha mencionado anteriormente, las STAs pueden presentar tiempos de cambio de canal distintos a lo esperado, lo que genera retransmisiones. Concretamente, en las pruebas se han utilizado interfaces Wi-Fi de dos fabricantes. Uno de ellos es *Realtek*, que presenta un esquema temporal de cambio de canal como el que se muestra en Fig. 27. El otro fabricante es *Mediatek*, cuyas interfaces, además de funcionar como cliente Wi-Fi, permiten crear interfaces virtuales en modo monitor para capturar tráfico en el canal activo. Las tarjetas *Mediatek* presentan un patrón temporal como el mostrado en Fig. 28. En ambas figuras, los intervalos señalados con una cruz son los intervalos en los que la STA no devuelve ACKs, mientras que los intervalos señalados con un *tick* verde son en los que sí que se responde el ACK.

En ambos casos, se observa que las STAs presentan intervalos de duración considerable sin enviar ACKs, lo que deteriora los resultados al provocar pérdidas innecesarias y complicar la estabilidad del proceso de *handover*. El hecho de que cada STA tenga su propia implementación (distintos *drivers*) dificulta la programación de *OdinAP* y desaprovecha recursos dedicados a tramas que no se van a recibir. Este problema será objeto de una línea futura que tendrá como objetivo encontrar la solución

que permita minimizar el número de retransmisiones hasta que la STA esté disponible para escuchar y transmitir.

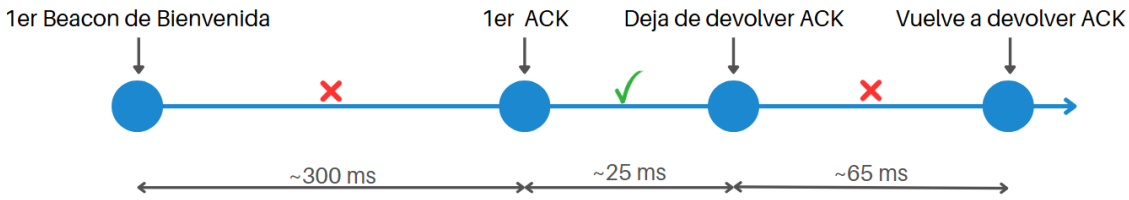


Figura 27. Diagrama de tiempos de la antena Wi-Fi Realtek

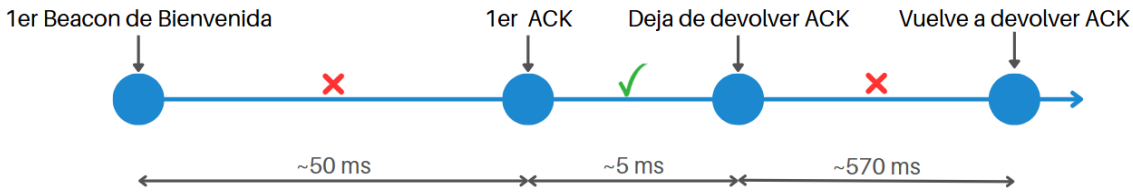


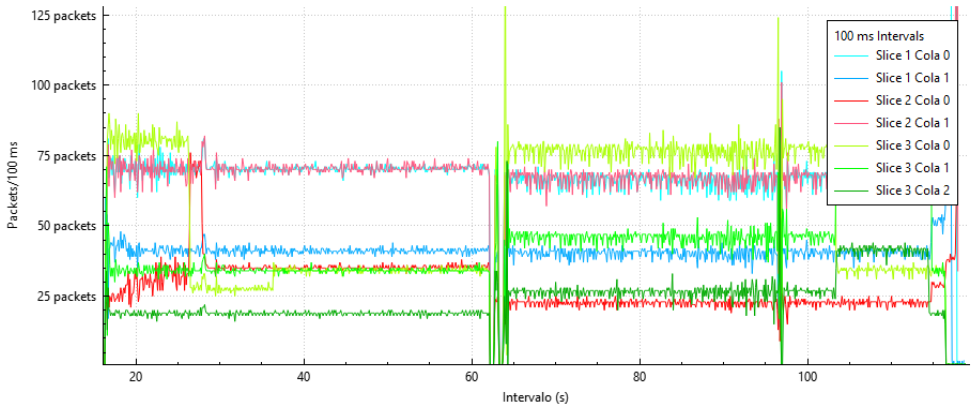
Figura 28. Diagrama de tiempos de la antena Wi-Fi Mediatek

Por otra parte, para evidenciar el impacto real que pueden tener ciertos retardos en momentos críticos como el *handover*, se analiza a continuación el efecto de utilizar funciones bloqueantes y no bloqueantes en *OdinAP*. En concreto, se estudia cómo afecta este comportamiento al sistema de *scheduling* cuando un equipo (ya sea un AP o una STA) deja de transmitir o recibir tramas Wi-Fi. Este aspecto, tratado en el capítulo 3 (apartado 3.2.3, *Actualización del sistema de slices y colas por handover*), supuso una mejora significativa: la sustitución de *system()* por *posix_spawn()* redujo en un 99,95 % el tiempo necesario para ejecutar el comando de cambio de canal. Esta alternativa permite lanzar el proceso en paralelo, minimizando su impacto sobre el resto del programa[5].

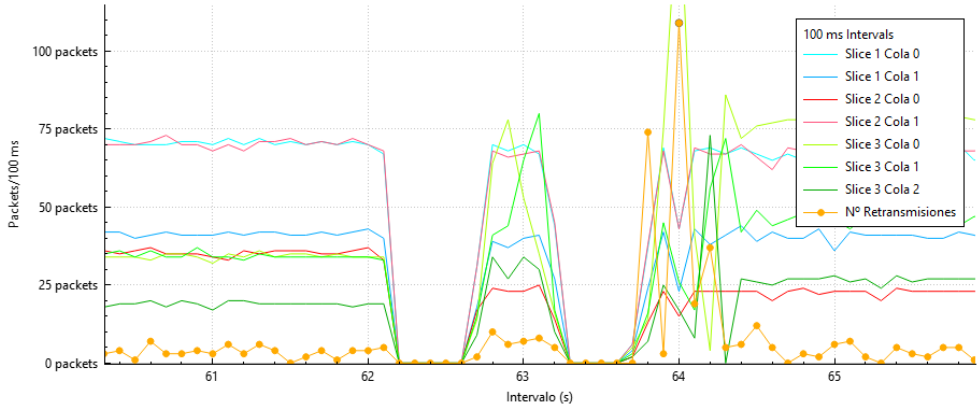
Se han recogido datos que permiten realizar la comparación del rendimiento del sistema antes y después de esta mejora. En esta ocasión se ha utilizado *Wireshark* como

[5] Resaltar que los resultados vistos hasta ahora en este apartado utilizan la alternativa no bloqueante *posix_spawn()*.

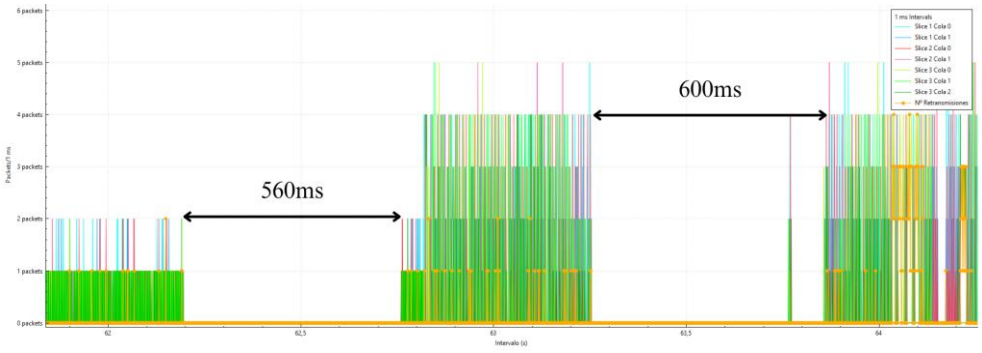
herramienta de monitorización, análisis y extracción de resultados, representando frente al tiempo el número de paquetes enviados con una resolución variable según el caso.



a) Paquetes enviados por cada 100 ms por wlan0



b) Zoom de paquetes enviados por cada 100 ms en el instante crítico



c) Zoom de paquetes enviados por cada 1 ms en el instante crítico

Figura 29. Estadísticas obtenidas con la llamada a la instrucción bloqueante

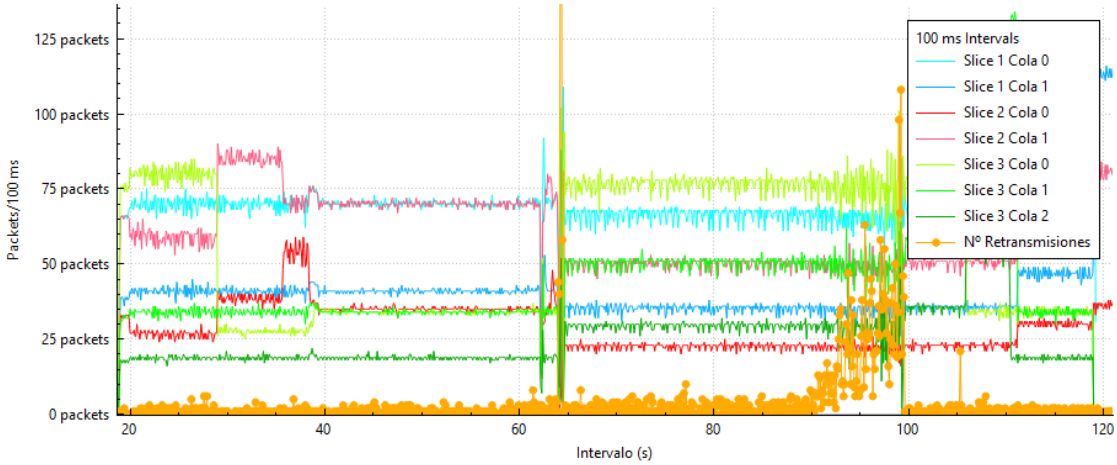
La Fig. 29 muestra los resultados del Experimento 3 al utilizar la instrucción bloqueante *system()* para cambiar el canal de la interfaz *wlan2*. En ella se representan los paquetes enviados durante el experimento, capturados desde la interfaz *wlan1* del AP2, que actúa como punto de acceso receptor de la STA en movimiento. La subfigura A recoge la visión global del experimento; la B amplía el instante en que el controlador *Odin* ordena a AP2 monitorizar a la STA en su canal original justo antes del *handover*; y la C muestra ese mismo intervalo con mayor resolución temporal (1 ms), lo que permite un análisis más detallado del comportamiento del sistema.

En Fig. 29.c se ven dos periodos de 560 y 600 ms en los que no se envían paquetes: el primero aparece por cambiar la interfaz *wlan2* al canal a monitorizar y el segundo por volver al canal original. En estos intervalos no hay retransmisiones, lo que indica que estas no son las responsables la bajada del *throughput*; las retransmisiones ocurren después del segundo periodo de silencio, cuando la STA monitorizada, tras realizar el *handover*, entra en el canal del AP2. La monitorización de la STA con la interfaz auxiliar *wlan2* en otro canal no debería causar un desvanecimiento tan prolongado en la transmisión.

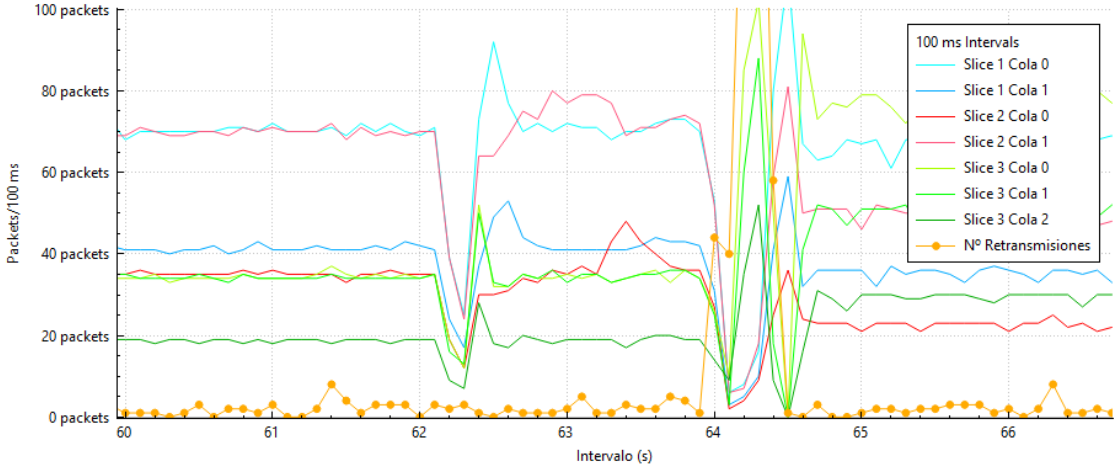
En la Fig. 30 se presentan los resultados obtenidos al utilizar la instrucción *posix_spawn()* para ejecutar el cambio de canal durante el *handover*. Como en casos anteriores, la Fig. 30.a muestra el experimento completo y la Fig. 30.b destaca el intervalo de mayor interés. Además, se han añadido las Fig. 30.c1 y Fig. 30.c2, donde se representan con mayor nivel de detalle el primer y el segundo desvanecimiento detectados, respectivamente.

Gracias a la mejora introducida, se observa una notable reducción en la duración de estos silencios. En concreto, el primer periodo pasa de los 560 ms anteriores a tan solo 111 ms (sumando 46 ms y 65 ms, como se ve en la Fig. 30.c1), y el segundo se reduce de 600 ms a 150 ms (sumando 80 ms y 70 ms, ver Fig. 30.c2). En ambos casos, el punto de acceso recupera su capacidad de transmisión en un tiempo mucho menor, logrando reducir los periodos de inactividad hasta en un 80 %. No obstante, a pesar de la mejora, todavía se observa un tiempo de parada no deseado, ya que el punto de acceso debería ser capaz de seguir transmitiendo sin interrupciones. Por tanto, la resolución de este error no

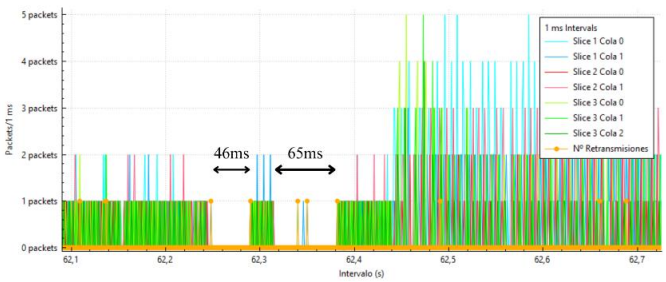
solo mejora significativamente el rendimiento, sino que también revela nuevas limitaciones del sistema que deberán abordarse en futuras líneas de trabajo.



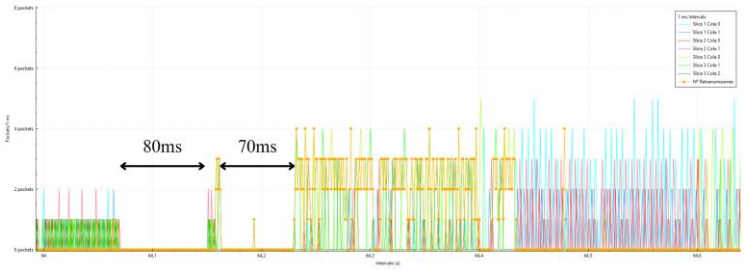
a) Paquetes enviados por cada 100 ms por wlan0



b) Zoom de paquetes enviados por cada 100 ms en el instante crítico



c1) Zoom del primer instante de bajada de throughput en b) (~ t=62.3). Paquetes enviados cada 1ms



c2) Zoom del segundo instante de bajada de throughput en b) (~ t=64.1). Paquetes enviados cada 1ms

Figura 30. Estadísticas obtenidas con la llamada a la instrucción no bloqueante

Pese a esta mejora, sigue observándose que justo en el momento en que *OdinAP* ejecuta el *handover* (t=64.3), se producen retransmisiones debidas al breve periodo de silencio en el que la estación no responde.

La diferencia de tiempos entre ambas soluciones demuestra la importancia de evitar llamadas bloqueantes en un entorno tan sensible como el del punto de acceso Wi-Fi, donde se manejan múltiples tareas críticas en tiempo real. Esta mejora ha permitido una mayor estabilidad y un comportamiento más robusto del sistema en general.

5. CONCLUSIONES Y LÍNEAS FUTURAS

En este trabajo se ha desarrollado e integrado una solución que combina un algoritmo de control de tasa (*Onoe*) con un sistema de *slicing* dentro de una arquitectura SDWLAN. El objetivo ha sido dotar al entorno *NeWLAN* de capacidades avanzadas de gestión de recursos inalámbricos, utilizando condiciones de tráfico intensivo. La integración se ha validado mediante una serie de pruebas experimentales en múltiples escenarios, demostrando la robustez y funcionamiento del sistema resultante.

5.1 CONCLUSIONES

A continuación, se enumeran los principales logros y conclusiones alcanzados durante el desarrollo del proyecto:

→ Integración exitosa de *slicing* y control de tasa: Se ha verificado el correcto funcionamiento simultáneo del sistema de *slicing* con el algoritmo *Onoe*. Esta combinación permite una gestión avanzada del *airtime* y una adaptación dinámica del MCS en función de las condiciones del canal, lo que es clave para garantizar la QoS en entornos con alta demanda.

→ Gestión eficiente del *airtime* mediante *slicing*: El sistema ha demostrado ser capaz de asignar recursos de la forma esperada en condiciones normales y de congestión. Además, el *airtime* no utilizado por colas con baja demanda se redistribuye eficazmente entre aquellas que requieren más capacidad, maximizando el rendimiento del punto de acceso.

→ Priorización efectiva del tráfico de control: Se ha reservado el *slice* 0 exclusivamente para el tráfico de control, quedando fuera del reparto de *airtime* con los *slices* de datos. Esta decisión ha demostrado ser eficaz: el tráfico de control (como DHCP, ARP y tramas de gestión IEEE 802.11) consume un porcentaje mínimo del *airtime*, permitiendo su procesamiento inmediato sin afectar al tráfico de datos. También se ha implementado una funcionalidad para redirigir correctamente tramas DHCP (u otros tráficos de control) con valores DSCP no previstos.

→ Validación del comportamiento de las opciones de *scheduling* ADWRR: Las pruebas realizadas con distintas variantes del algoritmo ADWRR —*quantum* estático,

igual ratio de satisfacción y preferencia por el índice menor— han validado su funcionamiento y la correcta distribución de recursos:

La variante con *quantum* estático distribuye los recursos excedentes proporcionalmente al *quantum* nominal entre las estructuras (colas o *slices*) activas que lo necesiten.

La variante de igual ratio de satisfacción equilibra dinámicamente los recursos excedentes entre las estructuras activas que necesiten más recursos para igualar su grado de satisfacción.

La variante con preferencia al índice menor garantiza primero la satisfacción de las colas prioritarias, aquellas con menor índice.

→ Gestión del *handover* y reducción de interrupciones: El sistema de *slicing* es compatible con el *handover*. Ha mejorado la estabilidad y continuidad del servicio y, aunque este proceso genera retransmisiones —debido al retardo en el cambio de canal de las STAs y a la ausencia temporal de ACKs—, la sustitución de la instrucción bloqueante *system()* por *posix_spawn()* ha reducido hasta un 80 % los periodos de inactividad del AP (de ~560–600 ms a ~111–150 ms). Esta optimización resulta esencial para conseguir una respuesta más fluida en entornos Wi-Fi sensibles al tiempo y plantear nuevas líneas de mejora.

Además de los logros técnicos alcanzados, es importante destacar que este trabajo se ha desarrollado dentro de un proyecto de investigación activo, en un entorno técnico complejo que combina tecnologías de distintos niveles. Desde aspectos de alto nivel como la *contenerización* de servicios, el despliegue de *pods* en *Kubernetes* o la coordinación entre nodos distribuidos, hasta componentes de bajo nivel como la programación en C o la modificación directa de campos en las cabeceras de los paquetes de red, el sistema requiere una comprensión integral y detallada. Esta multidisciplinariedad ha supuesto un esfuerzo añadido al trabajo realizado, que ha ido mucho más allá de implementar funcionalidades: ha exigido una integración cuidadosa en un entorno previamente diseñado, ya en funcionamiento y en evolución constante. Todo ello refuerza el valor del resultado obtenido y sienta una base sólida para futuras mejoras.

5.2 LÍNEAS FUTURAS

A partir de los resultados obtenidos y los desafíos identificados durante el desarrollo de este trabajo, se proponen las siguientes líneas de investigación para seguir mejorando la funcionalidad y el rendimiento del entorno SDWLAN:

→Adaptación dinámica del tamaño del *buffer* del *kernel*: Actualmente se utiliza un umbral fijo de 200 paquetes para el *buffer* del *driver* del *kernel*. Sin embargo, la capacidad de monitorización puede variar según la carga de tráfico, lo que podría provocar bloqueos si el umbral es demasiado bajo o deteriorar el funcionamiento del control de tasa si es demasiado alto, al permitir el envío de paquetes con un MCS desactualizado. Como mejora futura, se plantea implementar un mecanismo que ajuste dinámicamente el tamaño del *buffer* en tiempo real. Esto permitiría una mayor flexibilidad y resiliencia del sistema frente a variaciones de carga y condiciones del canal, mejorando la eficiencia en la transmisión de paquetes.

→Minimización de las retransmisiones durante el *handover*: Los experimentos han revelado que las STAs pueden presentar tiempos variables de cambio de canal y periodos prolongados sin envío de ACKs, lo que afecta negativamente al rendimiento y genera retransmisiones innecesarias. Se propone investigar las causas de estos periodos de inactividad y diseñar soluciones que reduzcan su impacto. Una posible solución podría consistir en enviar periódicamente tramas ARP *request* dirigidas a la IP de la STA en proceso de *handover*, utilizando una dirección MAC de destino *broadcast*. Al recibir un ARP *reply*, se confirmaría que la STA ya está activa, permitiendo mejorar la eficiencia del proceso sin necesidad de provocar retransmisiones adicionales.

→ Eliminación de los tiempos de inactividad durante la monitorización previa al *handover*:
En el diseño actual, cuando el punto de acceso cambia el canal de su interfaz auxiliar para monitorizar a una STA que va a realizar un *handover*, se produce un periodo de inactividad en el que el AP que monitoriza deja de transmitir. Aunque esta pausa ya se ha reducido significativamente, sigue teniendo un impacto en el rendimiento del sistema. Como línea futura, se plantea eliminar por completo estos tiempos de bloqueo mediante técnicas de paralelización más avanzadas con el objetivo de mantener la transmisión continua del AP sin interrupciones.

→ Integración de la clasificación de tráfico con el controlador SDN *Ryu*: El marcado de tráfico mediante DSCP se ha realizado en el *router* mientras, paralelamente, se ha desarrollado el marcado de paquetes con *Ryu*. Una posible mejora es integrar esta función en *NeWLAN*, lo que permitiría una gestión más centralizada, programable y dinámica de las políticas de QoS en función del estado de la red y de las aplicaciones activas.

BIBLIOGRAFÍA

[1] Manuel Rivas Morillo. “Implementación de algoritmo de control de tasa y desarrollo del entorno experimental para su evaluación”. Trabajo de Fin de Grado, Universidad de Zaragoza, 2024.

[2] Julia Santacruz Lacambra. “Despliegue de redes WLAN coordinadas basadas en arquitecturas programables y virtualizadas”. Trabajo de Fin de Grado, Universidad de Zaragoza, 2024.

[3] Sara Ibáñez Alloza. “Propuesta y desarrollo de mecanismos para el RAN *Slicing* en redes WLAN 5G en entorno real experimental”. Trabajo de Fin de Grado, Universidad de Zaragoza, 2024.

[4] M. Canales, J. Santacruz, J. R. Gállego, J. Ruíz-Mas, Á. Hernández-Solana, J. Navajas, y J. Ortín, “*Leveraging Kubernetes for Automated Deployment and Orchestration in Virtualized Wi-Fi Networks*”, *IEEE Transactions on Network and Service Management*. En revisión (enviado el 6 de marzo de 2025).

[5] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, “*Towards programmable enterprise WLANs with Odin*,” in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN ’12. New York, NY, USA: *Association for Computing Machinery*, 2012, p. 115–120. [Online]. Available: https://doi.org/10. 1145/2342441.2342465

[6] R. Riggio, M. K. Marina, J. Schulz-Zander, S. Kuklinski, and T. Rasheed, “*Programming abstractions for software-defined wireless networks*,” *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 146–162, 2015.

[7] E. Coronado, R. Riggio, J. Villalón, and A. Garrido, “*Joint mobility management and multicast rate adaptation in software-defined enterprise WLANs*,” *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 625–637, 2018.

[8] E. Coronado, S. N. Khan, and R. Riggio, "*5G-EmPOWER: A software-defined networking platform for 5G radio access networks,*" *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 715–728, 2019.

[9] A. Zubow, S. Zehl, and A. Wolisz, "*BIGAP — Seamless handover in high-performance enterprise IEEE 802.11 networks*," in *NOMS 2016 – 2016 IEEE/IFIP Network Operations and Management Symposium*, 2016, pp. 445–453.

[10] J. Saldana, R. Munilla, S. Eryigit, O. Topal, J. Ruiz-Mas, J. Fernández-Navajas, and L. Sequeira, "*Unsticking the Wi-Fi client: Smarter decisions using a software-defined wireless solution,*" *IEEE Access*, vol. 6, pp. 30917–30931, 2018.

[11] J. Saldana, J. Ruiz-Mas, J. Fernández-Navajas, J. L. S. Riaño, J.-P. Javaudin, J.-M. Bonnamy, and M. Le Dizes, "*Attention to Wi-Fi diversity: Resource management in WLANs with heterogeneous APs,*" *IEEE Access*, vol. 9, pp. 6961–6980, 2021.

[12] J. Lucas Vieira, D. Mosse, and D. Passos, "*LEAF: Improving handoff flexibility of IEEE 802.11 networks with an SDN-based virtual access point framework,*" *IEEE Transactions on Network and Service Management*, vol. 21, no. 6, pp. 6630–6642, 2024.

[13] A. Hernández-Solana, J. Ruiz, M. Canales, J. Fernández-Navajas, J. Gallego, y S. Ibáñez-Alloza, *"Practical Challenges of Implementing Slicing in 5G SDN WLAN Networks", IEEE Transactions on Network and Service Management*. *Major revision* (enviado el 10 de enero de 2025).

ANEXO A – TABLA DE INFORMACIÓN DE ÍNDICES MCS

Tabla 8. Información de índices MCS

MCS Index - 802.11n and 802.11ac

802.11n 802.11ac

HT MCS Index	VHT MCS Index	Spatial Streams	Modulation	Coding	20MHz Data Rate No SGI	20MHz Data Rate SGI	40MHz Data Rate No SGI	40MHz Data Rate SGI	80MHz Data Rate No SGI	80MHz Data Rate SGI	160MHz Data Rate No SGI	160MHz Data Rate SGI
0	0	1	BPSK	1/2	6.5	7.2	13.5	15	29.3	32.5	58.5	65
1	1	1	QPSK	1/2	13	14.4	27	30	58.5	65	117	130
2	2	1	QPSK	3/4	19.5	21.7	40.5	45	87.8	97.5	175.5	195
3	3	1	16-QAM	1/2	26	28.9	54	60	117	130	234	260
4	4	1	16-QAM	3/4	39	43.3	81	90	175.5	195	351	390
5	5	1	64-QAM	2/3	52	57.8	108	120	234	260	468	520
6	6	1	64-QAM	3/4	58.5	65	121.5	135	263.3	292.5	526.5	585
7	7	1	64-QAM	5/6	65	72.2	135	**150**	292.5	325	585	650
	8	1	256-QAM	3/4	78	86.7	162	180	351	390	702	780
	9	1	256-QAM	5/6	n/a	n/a	180	**200**	390	**433.3**	780	866.7
8	0	2	BPSK	1/2	13	14.4	27	30	58.5	65	117	130
9	1	2	QPSK	1/2	26	28.9	54	60	117	130	234	260
10	2	2	QPSK	3/4	39	43.3	81	90	175.5	195	351	390
11	3	2	16-QAM	1/2	52	57.8	108	120	234	260	468	520
12	4	2	16-QAM	3/4	78	86.7	162	180	351	390	702	780
13	5	2	64-QAM	2/3	104	115.6	216	240	468	520	936	1040
14	6	2	64-QAM	3/4	117	130.3	243	270	526.5	585	1053	1170
15	7	2	64-QAM	5/6	130	144.4	270	**300**	585	650	1170	1300
	8	2	256-QAM	3/4	156	173.3	324	360	702	780	1404	1560
	9	2	256-QAM	5/6	n/a	n/a	360	**400**	780	**866.7**	1560	1733.3
16	0	3	BPSK	1/2	19.5	21.7	40.5	45	87.8	97.5	175.5	195
17	1	3	QPSK	1/2	39	43.3	81	90	175.5	195	351	390
18	2	3	QPSK	3/4	58.5	65	121.5	135	263.3	292.5	526.5	585
19	3	3	16-QAM	1/2	78	86.7	162	180	351	390	702	780
20	4	3	16-QAM	3/4	117	130	243	270	526.5	585	1053	1170
21	5	3	64-QAM	2/3	156	173.3	324	360	702	780	1404	1560
22	6	3	64-QAM	3/4	175.5	195	364.5	405	n/a	n/a	1579.5	1755
23	7	3	64-QAM	5/6	195	216.7	405	**450**	877.5	975	1755	1950
	8	3	256-QAM	3/4	234	260	486	540	1053	1170	2106	2340
	9	3	256-QAM	5/6	260	288.9	540	**600**	1170	**1300**	n/a	n/a

ANEXO B – PSEUDOCÓDIGO DE ALGORITMOS ADWRR

A continuación, se presenta el pseudocódigo que muestra el funcionamiento de los algoritmos *slicing* en este trabajo. Se muestran el proceso de encolado y desencolado, el algoritmo ADWRR *inter-slice* e *intra-slice* y los distintos algoritmos de redistribución de *airtime* (igual ratio de satisfacción y preferencia al índice menor).

Los algoritmos han sido extraídos de [3], donde se explica el desarrollo de estos en el entorno experimental *Inymon*.

Algoritmo 1 Proceso de encolado

▷ q_{head}: paquete en la cabeza de la cola
▷ q_{tail}: paquete al final de la cola
▷ q_{size}: tamaño de la cola

Se guardan todos los valores necesarios relativos al paquete en la variable $nodo$
if q_{head} == NULL **then** ▷ Si la cola está vacía el nuevo paquete es el único
    $q_{head} = nodo$
    $q_{tail} = nodo$
else ▷ Si ya había paquetes, se añade al final de la cola
    $q_{tail} -> next = nodo$
    $q_{tail} = nodo$
end if
q_{size} += 1

Algoritmo 2 Proceso de desencolado

▷ q_{head}: paquete en la cabeza de la cola
▷ q_{tail}: paquete al final de la cola
▷ q_{size}: tamaño de la cola
▷ $nodo$: paquete desencolado

$nodo = q_{head}$ ▷ Se guarda el paquete a desencolar en una variable
if $q_{head} == q_{tail}$ **then** ▷ Si solo hay un paquete en la cola, se vacía
    q_{head} = NULL
    q_{tail} = NULL
else ▷ Si ya hay más de un paquete, actualiza el que está en cabeza
    $q_{head} = q_{head} -> next$
end if
q_{size} -= 1
return $nodo$

Tabla 9. Variables ADWRR

Variable	Descripción
$airtime[s,i]$	Airtime (tiempo de vuelo) estimado del primer paquete de la cola i del slice s (sin retransmisiones)
$Q[s]$	Quantum del slice s (configurado por el controlador)
$W[s,i]$	Peso de la cola i del slice s (W_{nom} configurado por el controlador)
$Deficit[s]$	Déficit del slice s
$Deficit[s,i]$	Déficit de la cola i del slice s
$Deficit_{retries}[s]$	Déficit acumulado del slice s debido a las retransmisiones
$Deficit_{retries}[s,i]$	Déficit acumulado de la cola i del slice s debido a las retransmisiones
N_{slices}	Número de slices
$N_{queue}[s]$	Número de colas del slice s

Algoritmo 3 Inter-slice *s*cheduling ADRR

```
1:  s = 0
2:  loop every
3:      Mientras la ocupación física del buffer del driver sea el umbral, espera

        #Actualiza el déficit del slice s si tiene paquetes que enviar
4:      if Slice[s].empty == false then    ▷ Las colas de s tienen paquetes

            ▷ Actualiza el déficit con el quantum y el déficit del slice s debido a las retransmisiones anteriores. Deficit_retries siempre es ≥ 0
5:          Deficit[s] += Q[s] - Deficit_retries[s]
6:          Deficit_retries[s] = 0

            #Envía los paquetes del slice mientras Deficit[s]>0 y haya paquetes con airtime más pequeño que su Deficit[s]
7:          ALGORITMO INTRA-SLICE para servir las colas (Alg. 4 ó 8)
8:      end if

        #Continúa con el siguiente slice
9:      if s < (N_slices - 1) then s++
10:     else s = 0
11: end loop
```

Algoritmo 4 Algoritmo Intra-Slice para servir las colas del slice. Opción ADRR

#Suma el déficit de todas las colas del slice *s*
$w_{QueueNoEmpty}[s] = \sum_{\forall i_{empty} \in s} W[s,i]$

for cada cola i no vacía del slice s **do**
  $Q[s,i] = (W[s,i]/w_{QueueNoEmpty}) * Q[s]$
  $Deficit[s,i]$ += $Q[s,i] - Deficit_{retries}[s,i]$
  $Deficit_{retries}[s,i] = 0$
end for

#Proceso de desencolado
$q_{ini} = Slice[s].Queue\_index$ ▷ Guarda el índice inicial de la cola para continuar la ronda
$i = q_{ini}$
while $Deficit[s] > 0$ **do**
  while $Deficit[s,i] > 0$ and $Queue[s,i]$.empty == false **do**
    Mientras la ocupación física del buffer del driver sea el umbral, espera

    #Opción de no agregación de paquetes
    ▷ Calcula el airtime del primer paquete de *Queue[s,i]*
    $airtime[s,i] = Queue[s,i]$.airtime(head packet)
    $npAggr = 1$

    #Opción de agregación de paquetes
    ▷ Calcula el número de paquetes que pueden ser agregados (*npAggr)* basado en el *Deficit[s,i]*, el máximo número de la agregación y el número de paquetes de la cola (A-MSDU)
    $npAggr = Queue[s,i]$.MaxAggregationAllowed($Deficit[s,i]$)
    $airtime[s,i] = Queue[s,i]$.airtime($npAggr$)

    if $airtime[s,i] \leq Deficit[s,i]$ **then**
      ▷ Desencola el paquete de la cabeza (A-MSDU) de la cola i (uno o más si pueden ser agregados) del slice s
      $Queue[s,i]$.dequeue($npAggr$)
      Construye la trama agregada con los paquetes desencolados
      Send(frame)
      $Deficit[s]$ -= $airtime[s,i]$
      $Deficit[s,i]$ -= $airtime[s,i]$

      if $Queue[s,i]$.empty == true **then**
        ▷ El último paquete de la cola *i* ha sido enviado. Su déficit es redistribuido proporcionalmente entre las colas activas basado en su quantum
        $w_{QueueNoEmpty}[s] = \sum_{\forall i \in s} W[s,i]$
        for cada cola j no vacía del slice s **do**
          $Deficit[s,j]$+=$(W[s,j]/w_{QueueNoEmpty}[s])$*$Deficit[s,i]$
        end for
        $Deficit[s,i] = 0$
        break ▷ Siguiente cola
      end if
    else break ▷ Siguiente cola
    end if
  end while

  #Continúa con la siguiente cola del slice *s*
  if $i <(N_{queues}[s]$ - 1) **then** i++
  else $i = 0$
  end if
  if $i == q_{ini}$ **then**
    $Slice[s].Queue\_index = i$
    break ▷ Siguiente slice
  end if
  if $Slice[s]$.empty == true **then** $Deficit[s] = 0$ ▷ Siguiente slice
end while

REDISTRIBUCIÓN *INTRA-SLICE*

Tabla 10. Variables de la redistribución Intra-Slice

Variable	Descripción
T	Intervalo de actualización
$airtime_T[s,i]$	Airtime estimada de todos los paquetes servidos por la cola i del slice s en el intervalo T, donde $n_{retries}$ es el número de retransmisiones. $\sum_{\forall k \in T} \left(1 + n_{retries}[s,i,k] \cdot airtime[s,i,k]\right)$
$airtime_T[s]$	Airtime estimado de todas las colas del slice s. $\sum_{\forall i \in s} airtime_T[s,i]$
$airtime_T$	Airtime estimado para todas los slices. $\sum_{\forall s} airtime_T[s]$
$MBR[s,i]$	Tasa Máxima de Bits (Maximum Bit Rate) de la cola i del slice s.
$R_{demanded}[s,i]$	Tasa de datos en bps a transmitir de la cola i.
$R_{achieved}[s,i]$	Tasa de datos efectiva en bps (nivel IP) de la cola i.
$S_{unsatisfied}[s]$	Grupo de colas insatisfechas del slice s.
$S_{underutilized}[s]$	Grupo de colas del slice s que tienen pesos para ceder.
$W_{nom}[s,i]$	Peso nominal de la cola i del slice s (configurado por el controlador).
$W[s,i]$	Peso usado por la cola i del slice s en el intervalo T. $W_{nom}[s,i]$ por defecto. $W_{nom}[s,i]$ - $W_{ceded}[s,i]$ si la cola i da peso a otras colas insatisfechas del mismo slice $s$lice s. $W_{nom}[s,i]$ + $W_{added}[s,i]$ si la cola i recibe peso de colas satisfechas del mismo slice s.
$W_{total}[s]$	Suma de los pesos de las colas del slice s. Esto se mantiene constante en todos los intervalos de tiempo. $\sum_{\forall i \in s} W_{nom}[s,i]$
$W_{ceded}[s,i,j]$	Peso cedido por la cola i a la cola j del slice s. Tiene el mismo valor que $W_{added}[s,j,i]$
$W_{ceded}[s,i]$	Peso total cedido por la cola i del slice s. $\sum_{\forall j \in S_{unsatisfied}[s]} W_{ceded}[s,i,j]$
$W_{added}[s,i,j]$	Peso recibido por la cola i de la cola j del slice s. Mismo valor que $W_{ceded}[s,j,i]$
$W_{added}[s,i]$	Peso total recibido por la cola i del slice s. $\sum_{\forall j \in S_{underutilized}[s]} W_{added}[s,i,j]$
$W^{exp}_{ceded}[s,i]$	Peso extra esperado de la cola i para ser cedido.
$W^{need}_{added}[s,i]$	Peso requerido por la cola i del slice s para estar satisfecha.
$\frac{W[s,i]}{W_{total}[s]}$	Porcentaje de airtime esperado (en el AP) usado por la cola i del slice s respecto a la asignación de su slice.
$DS[s,i]$	Grado de satisfacción (Degree of Satisfaction) de la cola i del slice s (0=totalmente insatisfecho, 1=totalmente satisfecho). $R_{achieved}[s,i]/R_{demanded}[s,i]$ si $R_{demanded}[s,i] < MBR[s,i]$ $R_{achieved}[s,i]/MBR[s,i]$ si $R_{demanded}[s,i] > MBR[s,i]$
$IDS^{exp}[s,i]$	El incremento de $DS[s,i]$ de la cola i del slice s esperado para el incremento de una unidad de peso.

Algoritmo 5 Redistribución Intra-Slice del quantum no usado

#Identificar colas insatisfechas del slice s

▷ $DS[s,i]$: Grado de satisfacción de la cola i del slice s

▷ $IDS^{exp}[s,i]$: El aumento de *DS* esperado para el incremento de una unidad de peso

1: **for** cada cola $i \in$ slice s **do**
2: $DS[s,i] = \max\left(\frac{R_{achieved}[s,i]}{R_{demanded}[s,i]}, \frac{R_{achieved}[s,i]}{MBR[s,i]}\right)$
3: **if** $DS[s,i] < 1$ **then**
4: Añade i al grupo de $S_{unsatisfied}[s]$ y guarda el par ($W[s,i]$, $DS[s,i]$)
5: **end if**
6: $IDS^{exp}[s,i] = DS[s,i]/W[s,i]$
7: $ds[s,i] = DS[s,i]$ ▷ Variable que guarda la tasa de satisfacción esperada
8: **end for**

#Estima *DS* si el peso fuera el nominal

▷ $S_{ceded}[s,i]$: grupo de colas que han cedido recursos a la cola $i \in s$

▷ $W_{ceded}[s,i,j] = W_{added}[s,j,i]$: recursos cedidos por la cola i a la $j \in s$

▷ $W_{added}[s,i] = \sum_{\forall j \in S_{ceded}[s,i]} W_{ceded}[s,j,i]$

▷ $W[s,i] = W_{nom}[s,i] + W_{added}[s,i] - W_{ceded}[s,i]$

9: **for** cada cola $i \in$ slice s cuyo $W[s,i] > W_{nom}[s,i]$ **do**
10: **for** cada cola $j \in S_{ceded}[s,i]$ **do**
11: $W[s,j]$ += $W_{ceded}[s,j,i]$
12: $ds[s,j] = \min(ds[s,j] + IDS^{exp}[s,j] * W_{ceded}[s,j,i], 1)$
13: $W[s,i]$ -= $W_{added}[s,i,j]$
14: $ds[s,i] = \max(ds[s,i] - IDS^{exp}[s,i] * W_{added}[s,i,j], 0)$
15: **end for**
16: **if** $ds[s,i] < 1$ **then** Añade la cola i al grupo $S_{unsatisfied}[s]$
17: **else** Elimina a la cola i del grupo $S_{unsatisfied}[s]$
18: **end for**

▷ Los pesos han sido reseteados, por lo que $W[s,i] = W_{nom}[s,i]\ \forall i \in s$
$W_{ceded}[s,i,j] = W_{added}[s,i,j] = W_{ceded}[s,i] = W_{added}[s,i] = 0\ \forall i,j \in s$

#Identificar colas infrautilizadas del slice s

▷ $excess[s,i]$: porcentaje de la porción de recursos que teóricamente corresponden a la cola i y no están ocupados

19: **for** cada cola $i \notin S_{unsatisfied}[s]$ **do**
20: $excess[s,i] = \left(\frac{W[s,i]}{W_{total}[s]} - \frac{airtime_T[s,i]}{airtime_T[s]}\right) / \left(\frac{W[s,i]}{W_{total}[s]}\right)$
21: **if** $excess[s,i] > \alpha$ **then**
22: Añade a la cola i al grupo $S_{underutilized}[s]$
23: $W^{exp}_{ceded}[s,i] = (excess[s,i]\text{-}\alpha) * W[s,i]$ ▷ Peso extra para ceder
24: **end if**
25: **end for**

#Opción Redistribuir pesos para conseguir una tasa de satisfacción similar

26: $w_{ceded} = \sum_{\forall i \in S_{underutilized}[s]} W^{exp}_{ceded}[s,i]$

▷ Distribuir w_{ceded} entre las colas $\in S_{unsatisfied}[s]$

27: **while** $w_{ceded} > 0$ **do**
28: Encuentra la cola $j \in S_{unsatisfied}$[s] con menor $ds[s,j]$
29: Encuentra la cola $i \in S_{underutilized}[s]$ con mayor $W^{exp}_{ceded}[s,i]$
30: **if** $\lceil W_{nom}[s,i] * \beta \rceil < w_{ceded}$ **then** $step = \lceil W_{nom}[s,i] * \beta \rceil$
31: **else** $step = w_{ceded}$
32: w_{ceded} -= $step$
33: $W^{exp}_{ceded}[s,i]$ -= $step$
34: $W_{ceded}[s,i]$ += $step$ and $W_{ceded}[s,i,j]$ += $step$
35: $W_{added}[s,j]$+= $step$ and $W_{added}[s,j,i]$ += $step$
36: $W[s,i]$ -= $step$
37: $W[s,j]$ += $step$
38: $ds[s,j]$ += $IDS^{exp}[s,j]$ * step
39: **if** $ds[s,j] \geq 1$ **then** Elimina la cola j del grupo $S_{unsatisfied}[s]$
40: **if** $S_{unsatisfied}[s] = \emptyset$ **then**
41: **if** $R_{demanded}[s,j] > MBR[s,j]$ para alguna j **then**
42: Redefine $S_{unsatisfied}[s]$ con $R_{demanded}$ real (*repite líneas 1-7*)
43: **if** $S_{unsatisfied}[s]$ sigue vacío **then** Salir de *while*
44: **end if**
45: **end if**
46: **end while**

#Opción Redistribuye el peso siguiendo una prioridad

47: Calcula el W requerido por cada cola j para estar satisfecha $W^{need}_{added}[s,j]$
48: **for** cada cola $i \in S_{underutilized}[s]$ ordenada de forma descendiente según su $W^{exp}_{ceded}[s,i]$ **do**
49: **while** $W^{exp}_{ceded}[s,i] > 0$ **do**
50: Iterar las colas del grupo $S_{unsatisfied}[s]$ en orden de prioridad
51: **if** $W^{need}_{added}[s,j] > W^{exp}_{ceded}[s,i]$ **then**
52: $W^{need}_{added}[s,j]$ -= $W^{exp}_{ceded}[s,i]$
53: $w = W^{exp}_{ceded}[s,i]$
54: **else**
55: $W^{need}_{added}[s,j] = 0$
56: $w = W^{need}_{added}[s,j]$
57: **end if**
58: $W^{exp}_{ceded}[s,i]$ -= w
59: $W_{ceded}[s,i]$ += w and $W_{ceded}[s,i,j] = w$
60: $W_{added}[s,j]$ += w and $W_{added}[s,j,i] = w$
61: $W[s,i]$ -= w
62: $W[s,j]$ += w
63: $ds[s,j]$ += $IDS^{exp}[s,j]$ * w
64: **if** $ds[s,j] \geq 1$ **then** Elimina la cola j del grupo $S_{unsatisfied}[s]$
65: **if** $S_{unsatisfied}[s] = \emptyset$ **then**
66: **if** $R_{demanded}[s,j] > MBR[s,j]$ para alguna j **then**
67: Redefine $S_{unsatisfied}[s]$ con el valor real $R_{demanded}$ (repite *líneas 1-7*)
68: **if** $S_{unsatisfied}[s]$ sigue vacío **then** Salir de *while*
69: **end if**
70: **end if**
71: **end while**
72: **end for**

REDISTRIBUCIÓN *INTER-SLICE*

Tabla 11. Variables de la redistribución Inter-Slice

Variable	Descripción
$MBR[s]$	Máxima Tasa de Bit (Maximum Bit Rate) del slice s
$R_{demanded}[s]$	Tasa de datos en bps a transmitir del slice s
$R_{achieved}[s]$	Tasa de datos efectiva en bps (nivel IP) del slice s
$S_{unsatisfied}$	Grupo de slices insatisfechos
$S_{underutilized}$	Grupo de slices que tienen pesos para ceder
$Q[s]$	Quantum usado por el slice s en el intervalo T, $Q_{nom}[s]$ por defecto. $Q_{nom}[s]$ - $Q_{ceded}[s]$ si el slice s reduce su quantum para dárselo a otros slices que están insatisfechos. $Q_{nom}[s]$ + $Q_{added}[s]$ si el slice s recibe quantum cedido por una o más slices satisfechas.
$Q_{nom}[s]$	Quantum nominal del slice s (configurado por el controlador)
Q_{total}	Suma el quantum de los slices. El valor se mantiene constante en todos los intervalos de tiempo $\sum_{\forall s} Q_{nom}[s]$
$Q_{ceded}[s,j]$	Quantum cedido por el slice s al slice j. Mismo valor que $Q_{added}[j,s]$
$Q_{ceded}[s]$	Quantum total cedido por el slice i $\sum_{\forall j \in S_{unsatisfied}} Q_{ceded}[s,j]$
$Q^{exp}_{ceded}[s]$	Quantum extra esperado del slice $s$para ser cedido
$Q_{added}[s,j]$	Quantum recibido por el slice s del slice j. Mismo valor que $Q_{ceded}[j,s]$
$Q_{added}[s]$	Quantum total que ha recibido el slice s $\sum_{\forall j \in S_{underutilized}} Q_{added}[s,j]$
$Q^{need}_{added}[s]$	Quantum requerido por el slice s para estar satisfecho
$\frac{Q[s]}{Q_{total}}$	Porcentaje de airtime (en el AP) que se espera que use el slice s respecto al uso total
$DS[s]$	Grado de satisfacción del slice s (0=totalmente insatisfecho, 1=totalmente satisfecho) $R_{achieved}[s]/R_{demanded}[s]$ if $R_{demanded}[s] < MBR[s]$ $R_{achieved}[s]/MBR[s]$ if $R_{demanded}[s] > MBR[s]$
$IDS^{exp}[s]$	Incremento de $DS[s]$ esperado para un incremento en una unidad de quantum

Algoritmo 6 Inter-Slice Redistribución del quantum no usado

#Identificar slices insatisfechos
▷ $DS[s]$: Grado de satisfacción del slice s
▷ $IDS^{exp}[s]$: El incremento del DS esperado por el incremento de una unidad de quantum
for cada slice s **do**
  $DS[s] = \max\left(\frac{R_{achieved}[s]}{R_{demanded}[s]}, \frac{R_{achieved}[s]}{MBR[s]}\right)$
  if $DS[s] < 1$ **then**
    Añadir slice s al grupo $S_{unsatisfied}$ y guardar par ($Q[s]$, $DS[s]$)
  end if
  $IDS^{exp}[s] = DS[s]/Q[s]$
  $ds[s] = DS[s]$ ▷ Variable para guardar la tasa de satisfacción esperada
end for

#Estimar *DS* si el quantum es el nominal
▷ $S_{ceded}[s]$: grupo de slices que han cedido recursos al slice s
▷ $Q_{ceded}[s, j] = Q_{added}[j, s]$: recursos cedidos por el slice s to j
▷ $Q_{added}[s] = \sum_{\forall j \in S_{ceded}[s]} Q_{ceded}[j, s]$
▷ $Q[s] = Q_{nom}[s] + Q_{added}[s] - Q_{ceded}[s]$
for cada slice s cuyo $Q[s] > Q_{nom}[s]$ **do**
  for cada slice $j \in S_{ceded}[s]$ **do**
    $Q[j]$ += $Q_{ceded}[j, s]$
    $ds[j] = \min(ds[j] + IDS^{exp}[j] * Q_{ceded}[j, s], 1)$
    $Q[s]$ -= $Q_{added}[s, j]$
    $ds[s] = \max(ds[s] - IDS^{exp}[s] * Q_{added}[s, j], 0)$
  end for
  if $ds[s] < 1$ **then** Añade el slice s al grupo $S_{unsatisfied}$
  else Elimina al slice s del grupo $S_{unsatisfied}$
end for
▷ Los valores de quantum han sido reseteados, $Q[s] = Q_{nom}[s]\ \forall s$
  $Q_{ceded}[s, j] = Q_{added}[s, j] = Q_{ceded}[s] = Q_{added}[s] = 0\ \forall s, j$

#Identificar slices infrautilizados
▷ $excess[s]$: porcentaje de la porción de recursos que teóricamente corresponden al slice s y no están ocupados
for each slice $s \notin S_{unsatisfied}$ **do**
  $excess[s] = \left(\frac{Q[s]}{Q_{total}} - \frac{airtime_T[s]}{airtime_T}\right) / \left(\frac{Q[s]}{Q_{total}}\right)$
  if $excess[s] > \alpha$ **then**
    Añade el slice s al grupo $S_{underutilized}$
    $Q^{exp}_{ceded}[s] = (excess[s]\text{-}\alpha) * Q[s]$ ▷ Peso extra para ceder
  end if
end for

#Opción Resdistribuir quantums para conseguir una tasa de satisfacción similar
$q_{ceded} = \sum_{\forall s \in S_{underutilized}} Q^{exp}_{ceded}[s]$

▷ Distribute q_{ceded} among slices $\in S_{unsatisfied}$
while $q_{ceded} > 0$ **do**
  Find slice $j \in S_{unsatisfied}$ with lower $ds[j]$
  Find $s \in S_{underutilized}$ with the highest $Q^{exp}_{ceded}[s]$

  if $\lceil Q_{nom}[s] * \beta \rceil < q_{ceded}$ **then** $step = \lceil Q_{nom}[s] * \beta \rceil$
  else $step = q_{ceded}$
  q_{ceded} -= $step$
  $Q^{exp}_{ceded}[s]$ -= $step$
  $Q_{ceded}[s]$ += $step$ and $Q_{ceded}[s, j]$ += $step$
  $Q_{added}[j]$+= $step$ and $Q_{added}[j, s]$ += $step$
  $Q[s]$ -= $step$
  $Q[j]$ += $step$
  $ds[j]$ += $IDS^{exp}[j]$ * step
  if $ds[j] \geq 1$ **then** Remove j to $S_{unsatisfied}$ group

  if $S_{unsatisfied} = \emptyset$ **then**
    if $R_{demanded}[j] > MBR[j]$ para alguna j **then**
      Redefine $S_{unsatisfied}$ with real $R_{demanded}$ (*repite líneas 1-7*)
      if $S_{unsatisfied}$ still empty **then** Exit *while*
    end if
  end if
end while

#Opción Redistribuye quantum siguiendo una prioridad

47: Calcula el Q requerido por cada slice j para estar satisfecho $Q^{need}_{added}[j]$
48: **for** cada slice $s \in S_{underutilized}$ ordenado de forma descendiente según su $Q^{exp}_{ceded}[s]$ **do**
49: **while** $Q^{exp}_{ceded}[s] > 0$ **do**
50: Itera los slices del grupo $S_{unsatisfied}$ en orden de prioridad
51: **if** $Q^{need}_{added}[j] > Q^{exp}_{ceded}[s]$ **then**
52: $Q^{need}_{added}[j]$ -= $Q^{exp}_{ceded}[s]$
53: $q = Q^{exp}_{ceded}[s]$
54: **else**
55: $Q^{need}_{added}[j] = 0$
56: $q = Q^{need}_{added}[j]$
57: **end if**
58: $Q^{exp}_{ceded}[s]$ -= q
59: $Q_{ceded}[s]$ += q and $Q_{ceded}[s, j] = q$
60: $Q_{added}[j]$ += q and $Q_{added}[j, s] = q$
61: $Q[s]$ -= q
62: $Q[j]$ += q
63: $ds[j]$ += $IDS^{exp}[s, j]$ * q
64: **if** $ds[j] \geq 1$ **then** Elimina j del grupo $S_{unsatisfied}$
65: **if** $S_{unsatisfied} = \emptyset$ **then**
66: **if** $R_{demanded}[j] > MBR[j]$ para alguna j **then**
67: Redefine $S_{unsatisfied}$ con el $R_{demanded}$ real (*repite líneas 1-7*)
68: **if** $S_{unsatisfied}$ sigue vacío **then** Salir de *while*
69: **end if**
70: **end if**
71: **end while**
72: **end for**