UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

FACULTAD DE INGENIERIA

Facultad de Ingeniaría - Universidad de la República

Proyecto de grado

Despliegue y evaluación de una plataforma de Software Defined Networks para redes inalámbricas

Autores:
Eric Brinckhaus
Agustín Echeverría

Supervisores:
Matías Richart
Eduardo Grampín

Informe de Proyecto de Grado presentado al Tribunal Evaluador como requisito de graduación de la carrera Ingeniería en Computación

Montevideo, 22 de junio de 2021

Despliegue y evaluación de una plataforma de Software Defined Networks para redes inalámbricas

Resumen

A medida que pasa el tiempo, internet se expande cada vez más y actualmente una enorme cantidad de dispositivos de distinto tipo y funcionalidades se encuentran conectados a internet. Sin embargo, las redes IP tradicionales son difíciles de administrar.

Las redes tradicionales presentan dificultades al configurar políticas de reenvío, balanceo de carga o reconfiguraciones por fallas o cambios. Para hacer frente a estos problemas surgen las redes definidas por software (SDN), un paradigma que separa el plano de control de la red de los dispositivos, promoviendo la centralización lógica en un controlador e introduciendo la capacidad de programar la red.

En la actualidad, la cantidad de diversos dispositivos con capacidad de conectarse a Internet, y en particular de conectarse a redes inalámbricas, aumenta el volumen de tráfico trasmitido. Esto hace que sea de gran interés distribuir recursos a demanda o reconfigurar la red de manera dinámica. Con esta finalidad es que se adapta el paradigma SDN a redes inalámbricas permitiendo la virtualización de recursos y de esta forma lograr configurar dinámicamente canales, ofrecer distintos tipos de servicio o reasociar clientes de forma eficiente.

Sobre este tema, en el año 2016, se realizó un proyecto de grado donde se realizó un relevamiento de estado del arte de donde se concluyó que la herramienta *5G-Empower* era la más completa en su momento para implementar SDN inalámbrico, a pesar de estar muy inmadura.

En este proyecto se definirán los conceptos principales y se realizará un estudio de la situación actual de SDN en redes inalámbricas. Una vez realizado este análisis, se buscará entender a fondo el funcionamiento y capacidades del *framework 5G-EmPower*, para luego desplegar una red inalámbrica utilizando esta herramienta. Por último, se planteará un caso de uso de interés, para el cual se desarrollará y evaluará una aplicación de control basada en un algoritmo de optimización que permita el reconocimiento de tráfico con preferencia dentro de una red. Esta aplicación se implementará y desplegará sobre el ambiente *5G-EmPower*.

Palabras Clave

Redes definidas por software, WiFi, 5G-Empower, Heurísticas, Slicing

Glosario

ACL Access Control List. 29

AP Access Point. 10–14, 22, 34, 74

API Application Programming Interface. 5, 7, 9–11, 14, 21, 25

ARP Address Resolution Protocol. 4

BSSID Basic Service Set Identifier. 12, 18, 80

DHCP Dynamic Host Configuration Protocol. 26, 29, 47

DNS Bomain Name System. 26

DSCP Differentiated Services Code Point. 20, 24, 29, 30, 47, 81

IoT Internet of Things. 2

LTE Long Term Evolution. 17, 18, 21, 27

LVAP Light Virtual Access Point. 10, 11, 13, 16–19, 22–25, 27, 33–38, 40, 43–47, 51, 54, 80, 83, 84

MCS Modulation and Coding Scheme. 8, 43

NAT Network Address Translation. 26

OSPF Open Shortest Path First. 1

QoS Quality of Service. 14, 17

RFC Request for Comments. 1

RSSI Received Signal Strength Indicator. 24, 34, 36, 38, 44, 45, 47, 54, 56, 62, 68

SDN Software Defined Network. 1–8, 15, 20, 21, 25, 29, 30, 32, 67

SSID Service Set Identifier. 20, 24, 27, 29, 77, 80

VAP Virtual Access Point. 13, 84

WTP Wireless Termination Point. 13, 16–19, 22–26, 28, 33–38, 45–49, 51–55, 58, 59, 61, 64, 68, 69, 74, 75, 80, 81

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La incorporación de nuevas funcionalidades y servicios en las redes de comunicaciones está fuertemente condicionada a los tiempos de mercado de los fabricantes de equipos especializados. El avance en el diseño y la producción de *hardware* genérico, en un constante proceso de disminución de costos y multiplicación de capacidad de cómputo, está desplazando el esfuerzo en el desarrollo de funcionalidades de red desde el *hardware* hacia el software.

Las redes de telecomunicaciones separan las funcionalidades del Plano de Datos (típicamente, reenviar datos desde un dispositivo a otro), de las del Plano de Control, que incluyen todas las funciones que permiten manipular la información adecuadamente. Históricamente las funciones del Plano de Control se implementan mediante protocolos distribuidos que se ejecutan en todos los dispositivos de red, como por ejemplo los protocolos de enrutamiento. Esto impone la necesidad de estandarizar los protocolos, de forma que equipos de diversos fabricantes puedan interoperar; por ejemplo, el protocolo OSPF está especificado en la RFC 2328 del IETF (*Internet Engineering Task Force*), un documento de casi 250 páginas.

Esto implica que incorporar una nueva funcionalidad al Plano de Control tiene el costo de promover la estandarización para que sea implementable en una red con múltiples dispositivos. Asimismo, como se mencionó anteriormente, los fabricantes deben invertir en procesos de desarrollo con viabilidad económica, lo que pone a los usuarios y operadores de telecomunicaciones en una posición de dependencia. Desde hace algunos años ha ganando terreno la idea de desacoplar funcionalidades del Plano de Control de los dispositivos e implementarlas en dispositivos de cómputo externo, lo que ha dado lugar al paradigma de las Redes Definidas por Software (*Software Defined Networks* o SDN).

SDN es un enfoque arquitectónico de red, que se basa en centralizar la lógica en un controlador (o un conjunto de), que permiten la administración y el control de los recursos de la red. En una arquitectura SDN, los dispositivos funcionan de forma autónoma con un conocimiento limitado del estado de la red. Con el tipo de control

centralizado que ofrece una red de este estilo, la administración, la restauración, la seguridad y las políticas de ancho de banda pueden ser manejadas de manera inteligente y optimizadas, otorgando una visión integral de la red. En este contexto, un operador de red o grupo de usuarios puede implementar una funcionalidad específica sin tener que esperar los tiempos de estandarización y desarrollo típicos de la industria; además, y gracias al desarrollo de abstracciones adecuadas, programadores no especializados pueden desarrollar aplicaciones que definen funcionalidades del Plano de Control, como por ejemplo un balanceador de carga en una red.

Por otro lado, existen múltiples soluciones parciales para el acceso móvil a servicios de voz y datos (por ejemplo 4G y WiFi), descoordinadas y poco eficientes desde el punto de vista de los recursos, y que resulta poco transparente para los usuarios. En este contexto, es de interés estudiar la oportunidad de atacar estos problemas basándose en los paradigmas de SDN y virtualización de funciones orquestadas sobre *hardware* genérico, que ponga en primer plano las demandas de los usuarios y los intereses de los proveedores. Si bien SDN es un campo explorado ampliamente sobre redes cableadas, el panorama no es el mismo para redes inalámbricas. En este tipo de redes aún existe *hardware* muy variado, y esto acompañado de la existencia de diferentes tecnologías de acceso que hacen la estandarización más difícil, generando una desafío adicional.

En este proyecto nos interesa continuar y extender el estudio del proyecto de grado realizado en el año 2016 (Chamlian *et al.* (2016)), donde se realizó una análisis inicial del estado del arte de SDN en redes WiFi y se evaluó un pequeño prototipo de una plataforma de SDN para WiFi. En particular, se desea realizar un despliegue mayor de una plataforma específica denominada 5G-EmPower (*5g-EmPOWER Wiki* (s.f.)), evaluando en detalle sus funcionalidades. Esta plataforma consiste en agentes que se implementan e instalan en los dispositivos inalámbricos, típicamente *Access Points* y/o *routers WiFi* que permitan la comunicación con el controlador, y define una API para la especificación de funcionalidades, que se programan en el controlador utilizando un amplio rango de lenguajes, que pueden ser especializados, de propósito general, y/o basados en la web.

Además se desarrollarán aplicaciones del plano de control específicas con el objetivo de evaluar la plataforma y el despliegue. En este sentido, parte de los objetivos es desarrollar aplicaciones que permitan realizar diferentes operaciones sobre el tráfico de una red WiFi. Por ejemplo, para proveer prioridad a mensajes de alerta en despliegues de IoT o para contar con conexión prioritaria para un despliegue de robots autónomos.

El presente proyecto tiene como objetivo general realizar un despliegue mayor de una plataforma específica denominada 5G-EmPower (*5g-EmPOWER Wiki* s.f.), evaluando en detalle sus funcionalidades. Además se plantearon los siguientes obje-

tivos específicos:

- Actualizar el relevamiento de plataformas o *frameworks* de SDN en redes inalámbricas.
- Comprender el funcionamiento de la plataforma 5G-Empower.
- Desplegar una red inalámbrica de prueba donde se instale la plataforma 5G-Empower.
- Desarrollar, desplegar y evaluar aplicaciones que agreguen funcionalidades bajo el paradigma SDN en redes inalámbricas.

Como resultados de este proyecto se espera contar con:

- Un documento de relevamiento de estado del arte en SDN.
- Un despliegue de una red inalámbrica gestionada por la plataforma elegida.
- Documentación sobre las funcionalidades que ofrece dicha plataforma.
- Un desarrollo en este ambiente en donde se aprecie un beneficio para una parte interesada.

Los siguientes capítulos de este documento se estructuraran de la forma que se describe a continuación. En el Capítulo 2, se presenta el paradigma SDN y se describen las principales herramientas disponibles para redes inalámbricas. En el Capítulo 3, se especifican las prestaciones y características de la plataforma 5G-Empower. Se detalla el despliegue realizado de una red 5G-Empower en el Capítulo 4. En el Capítulo 5, se propone un caso de uso que agregue funcionalidades en la red. Además, se describe el problema a resolver para cumplir con este caso de uso y se presentará una solución desarrollable. Por último se comprobarán los resultados sobre la red desplegada. Las conclusiones y trabajo a futuro se encuentran en el Capítulo 6.

Capítulo 2

Antecedentes: Software Defined Networking

Dado que SDN y en particular SDN en un medio inalámbrico, son conceptos claves en este proyecto, en este capítulo se explica en detalle el modelo SDN, más específicamente SDN en medio inalámbrico, y se realiza un relevamiento de plataformas existentes. El objetivo de esto último es entender el contexto en el que se inscribe la plataforma Empower que utilizaremos en el proyecto.

2.1. Motivación

Las redes de computadoras están constituídas típicamente por un gran numero de dispositivos de red, como routers y switches, en donde participan diferentes y complejos protocolos de red. Las redes IP tradicionales son complejas y difíciles de administrar. Es dificultoso configurar la red de acuerdo a políticas predefinidas y reconfigurarla para responder a fallas, cargas variables y cambios. Además, las redes tradicionales agrupan los planos de control y de datos.

El plano de datos realiza el envío sin estado y modifica paquetes basándose en tablas definidas. Asimismo, mantiene las estadísticas a nivel de paquete. El plano de control por su parte se encarga de dibujar la topología de la red y nutrir las tablas que usa el plano de datos, por ejemplo tablas de enrutamiento o tablas ARP.

La idea de una red programable ha sido propuesta para facilitar la evolución en las redes. En particular, *Software Defined Networking* (SDN) es un nuevo paradigma en el área de redes de computadoras, en el que el cambio principal con respecto al paradigma tradicional es que el plano de control será centralizado en una sola entidad (controlador). La idea principal es otorgarle a los desarrolladores y administradores de red la posibilidad de gestionar los recursos de una red de una manera sencilla, similar al manejo de memoria o recursos computacionales. De esta forma

los dispositivos de enrutamiento no abarcarán la toma de decisiones, dedicándose únicamente al plano de datos.

Un esquema de SDN permite una configuración de la red dinámica y programáticamente eficiente al mismo tiempo que permite un mejor desempeño y monitoreo de los componentes en la red. Las principales desventajas de una red SDN es que presenta un único punto de falla y además que es necesario reconfigurar toda estructura de la red para implementar SDN, en un ambiente ya funcional.

2.2. Modelo SDN

Esta sección esta basada en los trabajos realizados por (Nunes Astuto *et al.*, 2014), (Kreutz *et al.*, 2014) y (Mamushiane *et al.*, 2018).

El controlador se comunica utilizando una API con los dispositivos de enrutamiento, en SDN a esta API se le llama interfaz *Southbound*, mientras que la API con la que se comunica con las aplicaciones de red se le llama interfaz *Northbound*.

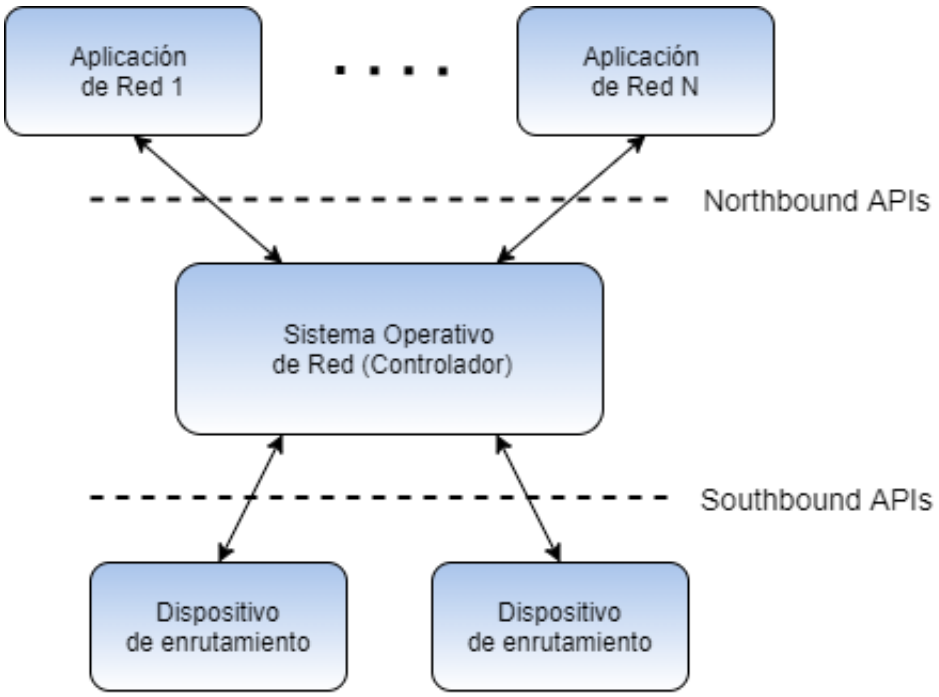

Figura 2.1: Modelo SDN.

En el modelo SDN (Ver Figura 2.1), siempre se representan a los dispositivos de enrutamiento por debajo del controlador y a las aplicaciones de red por arriba. Es por esto que se le denomina interfaces de *Southbound* y *Northbound* ya que corresponden a la dirección del flujo de comunicación entre el controlador y las distintas partes. Estos conceptos son importantes en SDN por lo que de aquí en más, todos los diagramas serán representados de la misma manera, las aplicaciones de red arriba del controlador y los dispositivos de enrutamiento por debajo.

Se ha buscado un protocolo estándar para definir las interfaz *Southbound* y también se está buscando uno para la interfaz *Northbound* (para este último aún no se ha logrado llegar a un estándar). Han surgido varias propuestas, pero la que tuvo más soporte y que actualmente es la mayoritariamente usada en redes SDN es el protocolo OpenFlow.

2.3. Protocolo OpenFlow

OpenFlow es un protocolo estandarizado para interactuar con los comportamientos de ruteo de switches de distintas marcas. Este protocolo permite programar tablas de flujo (una tabla con reglas asociadas con ciertas acciones) en un switch compatible con OpenFlow. Un switch *OpenFlow* consiste en una tabla de flujos que utiliza un canal seguro por el cual se comunicará con el controlador asociado, y cuya finalidad es la de configurar y actualizar las tablas de flujo de la manera indicada por el controlador.

El comportamiento de un switch *OpenFlow* es el siguiente:

Cuando recibe un paquete, comprueba en su tabla de flujos si ya hay alguna regla para ese tipo de paquete (para esa dirección, puerto y protocolo). Si hay una regla, entonces toma la acción correspondiente, en caso contrario envía un paquete llamado *"Packet-IN"* al controlador con el paquete que recibió.

Al paquete *"Packet-IN"* se le puede asignar un *Buffer ID*, el cual indica que el switch guarda ese paquete en su buffer con el ID indicado. Este *Buffer ID* sirve para que luego el controlador le indique la acción a tomar con el paquete que tiene almacenado en ese buffer ID y no es necesario mandar el paquete nuevamente.

El controlador luego de analizar el paquete, le enviará al switch un paquete llamado *"Packet-OUT"* con la acción a tomar para ese paquete y/o un paquete *"Flow Modification"* para modificar la tabla de flujos. El paquete *Packet-OUT* además de contener la acción, puede contener el paquete original o el *Buffer ID* si es que fue provisto por el switch. El paquete *Flow-MOD* le indica al switch que debe crear una nueva entrada en su tabla de flujos para saber qué acción tomar con paquetes similares en el futuro. Este paquete puede contener un *Buffer ID* para indicar al switch que esta acción debe tomarla con ese paquete también (en este caso no sería necesario mandar un paquete *Packet-OUT*). Además del buffer id y de la acción, los paquetes *Flow-MOD* contienen:

- **Match**: En este campo se manda la máscara, puerto, y/o dirección IP que deberán cumplir los paquetes para tomar esta acción.
- **Idle Timeout**: Si el switch no recibe paquetes de este tipo por esta cantidad de tiempo, se deberá eliminar esta entrada en la tabla de flujos.

- **Hard Timeout**: Luego de esta cantidad de tiempo eliminar esta entrada de la tabla de flujos.
- **Priority**: Prioridad de la entrada. Si un paquete matchea con más de una entrada de su tabla de flujos, tomará la acción de la entrada con más prioridad.

Además, si en los cabezales de *Timeout* del paquete *Flow-Mod* se envían valores en 0, esos campos no serán tomados en cuenta y la entrada permanecerá en la tabla hasta que el control envíe la instrucción de remover esa entrada. Una vez que el switch recibe alguno de estos paquetes tomará la acción requerida.

Notar que existen dos maneras de que el controlador llene la tabla de flujos de un switch (se pueden combinar ambas):

- De manera **reactiva**, que es la descrita anteriormente, cuando el controlador recibe un paquete de un switch actualiza su tabla de flujos.
- De manera **proactiva**, el controlador manda paquetes de actualización de entradas al switch aunque este no le haya enviado un paquete.

El controlador obtiene la información de la topología y estado de la red a partir de la información proporcionada por cada dispositivo. Con esta información el controlador decide qué decisiones tomar y qué entradas/acciones enviar a los switches.

La performance del controlador *OpenFlow* está sujeta a la implementación del mismo. Dentro de los controladores más populares se encuentran: *Ryu*, *ONOS*, *Floodlight* y *OpenDaylight*.

- **Ryu** (*Controlador Ryu* s.f.): es un framework implementado completamente en *Python* el cual incluye APIs para crear aplicaciones de gestión y control. Tambien incluye un conjunto de aplicaciones ya implementadas.

- **ONOS** (Berde *et al.*, 2014): es una plataforma distribuida, enfocada en aspectos de performance, escalabilidad y disponibilidad en redes de gran tamaño.

- **OpenDaylight** (Medved *et al.*, 2014): es un controlador implementado en Java con gran foco en seguridad, escalabilidad, estabilidad y performance. Su *Southbound* API brinda soporte a una gran variedad de servicios como *OpenFlow*, PCE-P, I2RS y NETCONF entre otros.

2.4. SDN Inalámbrico

Dado el extenso uso de las redes inalámbricas hoy en día, es de gran interés extender los conceptos de SDN hacia una red de este tipo. Como primer punto, se debería extender el paradigma SDN, para que las interfaces *southbound* no solo cumplan funciones de controlar flujos de datos sobre la capa IP, sino que interesa

realizar cambios sobre parámetros físicos de una interfaz. Por ejemplo, al aplicar SDN al medio inalámbrico, interesa generar reglas para asignar ciertos flujos a distintos canales usando un MCS (Modulation and Coding Scheme) en particular.

Tanto por las condiciones variantes del canal, como cuando un cliente o dispositivo cambia su estado en la red, ya sea por una nueva conexión, una desconexión o una resignación a otro nodo, la topología se actualiza. Durante esta reconfiguración es importante que el servicio a los clientes no se vea afectado, además de minimizar el *overhead* generado por los mensajes de control.

Un problema importante que presentan las redes inalámbricas al intentar aplicar SDN, es que las redes de este tipo son muy heterogéneas, y por lo tanto las interfaces *southbound* deben brindar compatibilidad a tecnologías variadas y a los diversos protocolos utilizados por las mismas.

2.4.1. Relevamiento de plataformas para SDN en redes inalámbricas

Por los motivos mencionados, existen diferentes soluciones de implementación de SDN en redes inalámbricas. A continuación se listarán algunos de los controladores de SDN en redes inalámbricas más conocidos. En esta sección, describiremos brevemente cada uno y nombraremos sus principales ventajas.

Para la redacción de esta sección nos vamos a centrar en el trabajo realizado por (Dezfouli *et al.*, 2018).

2.4.1.1. DenseAP

La arquitectura propuesta por *DenseAP* (Murty, Padhye, Chandra *et al.*, 2008) aborda los problemas de las WLAN densamente desplegadas sobre el control de asociaciones y la asociación de canales. Los dos principales componentes de esta arquitectura son *DenseAPs* y *DenseController*.

- Los **DenseAPs** se instalan en los APs y ejecutan una "máquina virtual" Windows que inicializa un demonio, el cual enviará periódicamente reportes al *Dense Controller*. Estos reportes incluyen una lista de clientes, condición del canal, valores de interferencia o una lista de clientes solicitando acceder a la red, entre otra información.

- El **DenseController** permite a los programadores de red configurar parámetros y enviar información al demonio para aplicar una configuración.

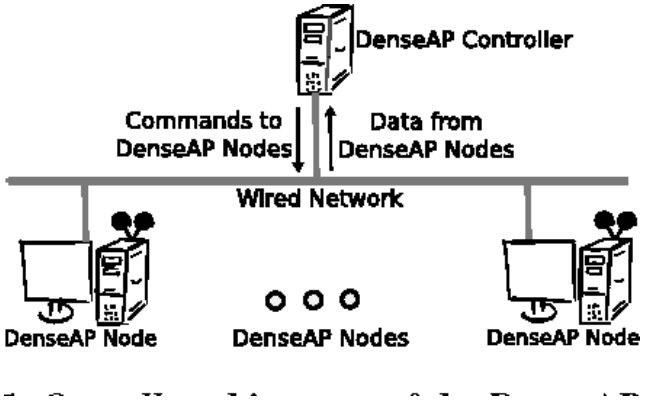

Figura 2.2: Arquitectura de un sistema DenseAP. (Murty, Padhye, Chandra *et al.*, 2008)

La principal ventaja de este framework es que ofrece configurabilidad de la red así como la obtención de estadísticas. La gran desventaja, es que no brinda la posibilidad de extender los mecanismos de control que provee.

2.4.1.2. Trantor

La arquitectura propuesta en *Trantor* (Murty, Wolman *et al.*, s.f.) es una versión mejorada de *DenseAP*, solo que ahora su protocolo *South-bound* extiende los mensajes de control con los clientes, además de con los APs. Esto permite ofrecer un monitoreo más completo del servicio que se está brindando teniendo estadísticas reales del cliente. *Trantor* provee un conjunto de APIs para controlar la asociación, el canal y la tasa de transmisión entre otros. Según (Dezfouli *et al.*, 2018), es en principio un buen diseño de la arquitectura, sin embargo aún no se ha llegado a un mecanismo de control que se beneficie del uso de las funciones brindada por las APIs.

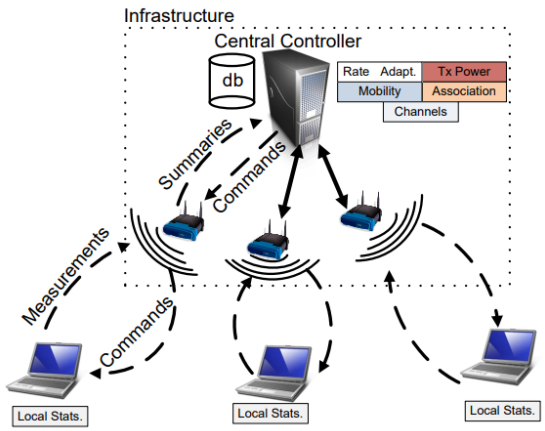

Figura 2.3: Arquitectura de Tantor. (Murty, Wolman *et al.*, s.f.)

2.4.1.3. Dyson

El *framework Dyson* (Murty, Padhye, Wolman *et al.*, 2010) es una versión extendida de *Trantor*, por lo que habilita tanto a los cliente como a los APs a comunicarse con el controlador, con la diferencia que divide a los clientes en dos grupos: los *Dyson-aware* y los *legacy*, donde solo los cliente *Dyson-aware* se pueden comunicar con el controlador. A partir de la información obtenida por las APIs, *Dyson* establece un mapa de red el cual contiene los siguientes componentes: localización de APs y clientes (utilizando cierto algoritmo), un grafo dirigido con información de las conexiones entre clientes y APs, utilización de los canales, y una base de datos donde se guardan todas las medidas recabadas. La API de *Dyson* incluye primitivas para configurar el canal, la potencia y tasa de transmisión de APs y clientes entre otras. También incluye primitivas para manejar la asociación entre APs y clientes. Para los clientes *legacy*, ofrece una API más reducida muy similar a la ofrecida por *DenseAP*.

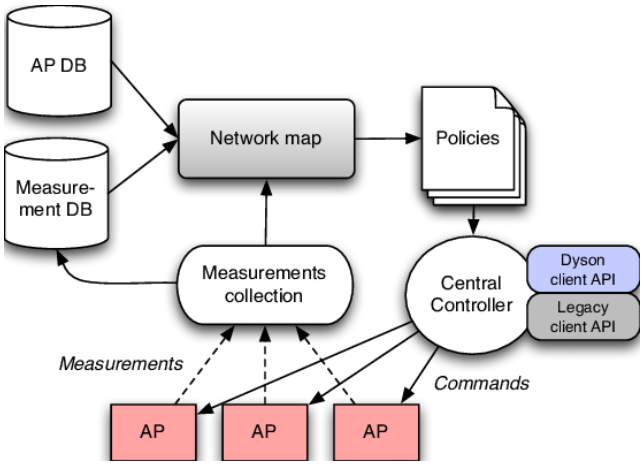

Figura 2.4: Arquitectura de Dyson. (Murty, Padhye, Wolman *et al.*, 2010)

2.4.1.4. Odin

El *framework Odin* (Suresh *et al.*, 2014) se basa en la virtualización de APs en un AP físico y para esto, introduce el concepto de LVAP, que explicaremos en Capítulo 3. Esto permite particionar la red (*slicing*) para brindar servicios de distinta calidad a distintos clientes. La arquitectura tiene principalmente tres componentes: *Odin Controller*, *Odin Agent* y las aplicaciones.

- **Odin controller** presenta una vista general al programador de status de los APs, clientes y OpenFlow switches de la red.
- **Odin Agent** se ejecuta en los APs y permite la comunicación mediante un protocolo propio south-bound. Las operaciones que son de tiempo crítico por

ejemplo la transmisión de ACKs, son realizadas por los APs mientras que las operaciones en donde el tiempo no es crítico, son manejadas por el controlador.

- **Applications**: las aplicaciones son implementadas en el controlador y utilizan la información provista por los *Odin Agents*, *OpenFlow* y SNMP (*Simple Network Management Protocol*). Las APIs proveen primitivas para implementar varios mecanismos de control por ejemplo control de asociaciones, balanceamiento de carga, y traslados de un cliente desde un AP hacia otro (de aquí en adelante llamados "handovers").

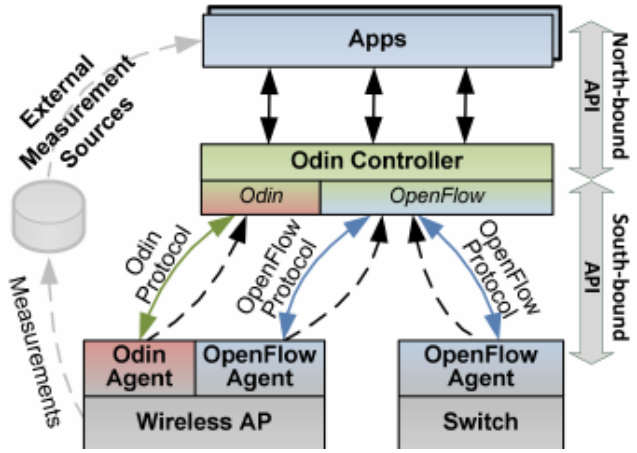

Figura 2.5: Arquitectura de Odin. (Suresh *et al.*, 2014)

2.4.1.5. Empower

Empower (Riggio, Rasheed *et al.*, 2013) se basa en el concepto de LVAP introducido por *Odin*, para facilitar la movilidad de clientes. Además de los LVAP, define otras abstracciones fundamentales para el controlador:

- **Resource Pool**: es un conjunto de *Resource Blocks* identificados por una frecuencia y un ancho de banda. Se usa para representar la relación entre un LVAP con un AP.

- **Channel quality and interference map** proveen estadísticas de la calidad de los enlaces entre clientes y APs. Es de mucha utilidad al tratar de optimizar los recursos en una red.

- **Port**: define la configuración del enlace entre un cliente y el AP asignado.

Estas abstracciones son expuestas a los programadores mediante una API Python, la cual les permite crear aplicaciones para gestionar la red.

2.4.1.6. BIGAP

Al igual que *Odin*, *BIGAP* (Zubow *et al.*, 2016) utiliza el concepto de virtualización pero lo hace de manera inversa. El objetivo de este framework es proveer un handover de clientes sobre los APs de forma transparente para los clientes. Para esto realiza la virtualización de forma contraria a *Odin*, mapea todos los APs físicos de la red a un solo AP virtual (por lo que no permite *slicing*). *BIGAP* asume que cada AP tiene dos interfaces inalámbricas, una para operar como AP y la otra para la recolección de estadísticas el cual periódicamente escucha los paquetes en todos los canales. Esta información va a ser utilizada para proveer un rápido mecanismo de control de asociaciones. Además, *BIGAP* requiere que todos los APs compartan el mismo BSSID para reducir la complejidad de handovers y para evitar colisiones, cada AP deberá seleccionar un canal que no esté siendo usado por sus *"two-hop neighborhood"*, estos son los APs que están a dos o menos nodos de distancia. Asociar un cliente a otro AP se realiza con un comando de cambio de canal y transfiriendo el estatus de la conexión del cliente al nuevo AP. *BIGAP* provee un protocolo *south-bound* para la colección de estadísticas y el control de asociamiento.

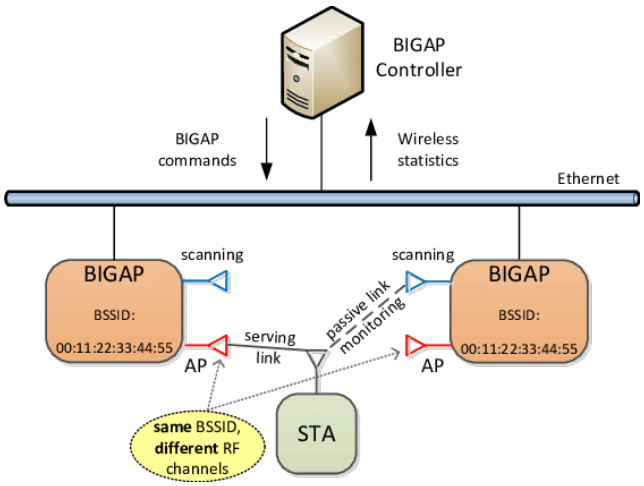

Figura 2.6: Arquitectura de BIGAP. (Zubow *et al.*, 2016)

2.4.1.7. AeroFlux

AeroFlux (Schulz-Zander *et al.*, s.f.) ataca el problema de que por flujo, o por paquete, los mecanismos de control deberían tener un delay corto y limitado. La arquitectura propone dividir el plano de control en dos capas: el plano *Global Control* y el plano *Near-Sighted Control*.

- **Global Control** maneja las operaciones donde el tiempo no es crítico y operaciones que requieren un conocimiento global de la red, por ejemplo, autenticación o balanceamiento de carga, entre otros.

- **Near-Sighted Control** son controladores situados cerca de los APs para realizar operaciones donde el tiempo es crítico, por ejemplo, control de tasa de transmisión y ajuste de potencia por paquete, entre otros.

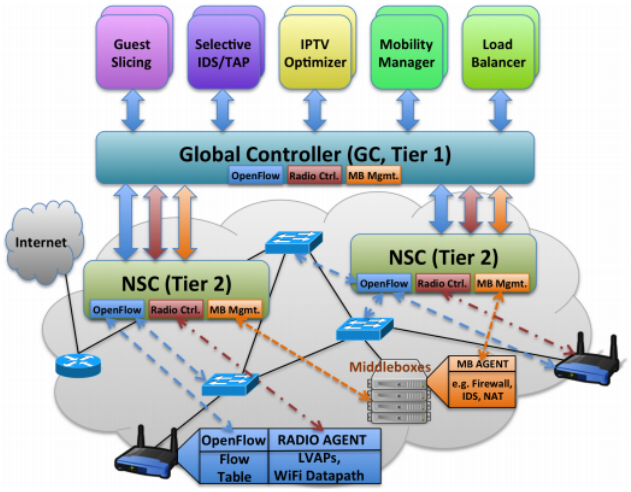

Figura 2.7: Arquitectura de AeroFlux. (Schulz-Zander *et al.*, s.f.)

2.4.1.8. CloudMAC

CloudMAC (Vestin *et al.*, s.f.) descompone las operaciones de los AP en dos módulos y utiliza las tablas de *switching* de *OpenFlow* para gestionar la comunicación entre los módulos. Los principales componentes de esta arquitectura son: APs, VAPs, un switch *OpenFlow* y un controlador *OpenFlow*.

Los APs solo reenvían las tramas MAC y manejan las operaciones de tiempo crítico, mientras que el resto de funcionalidades sobre las tramas MAC, son realizadas por las VAPs las cuales residen en una infraestructura de computación en la nube. Las VAPs son instancias de sistemas operativos ejecutadas en un hipervisor. Cada VAP puede contener una o más tarjetas de red inalámbricas, donde cada tarjeta es implementada como un driver en la VAP.

Las máquinas virtuales se conectan con los APs físicos mediante túneles de capa dos y un switch *OpenFlow*, donde su tabla de *forwarding* representa la forma en que se comunicarán. Mientras *Odin* necesita cambiar de LVAP a otro WTP para realizar el cambio de AP, en *CloudMAC* basta con cambiar la tabla de *forwarding* del switch OpenFlow. Una desventaja de esta arquitectura es que las operaciones manejadas por las VAPs tendrán mayor delay, pero por esto sólo se reenvían a las VAPs las operaciones que no requieren tiempo crítico.

Además esta arquitectura es capaz de realizar *downlink* programado configurando el switch OpenFlow usando tasas de ajuste o algoritmos de partición de intervalos de tiempo.

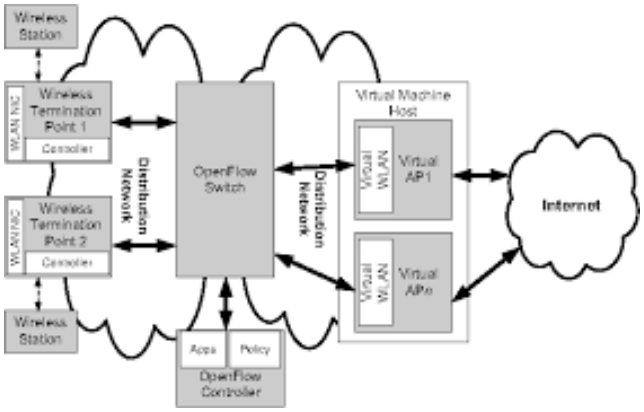

Figura 2.8: Arquitectura de CloudMAC. (Vestin *et al.*, s.f.)

2.4.1.9. Ethanol

Ethanol (Moura *et al.*, 2020), al igual que *CloudMAC*, mueve la mayoría de operaciones MAC de los APs al controlador. Sin embargo *Ethanol* no usa *OpenFlow*, ya que los autores consideran que no es un protocolo apto para proveer QoS en una red inalámbrica, por lo que implementaron su propio protocolo para complementar *OpenFlow*. Por lo tanto, un *Ethanol AP* debe proveer dos interfaces: una interfaz *Ethanol*, a través de un *Ethanol Agent* y una interfaz *OpenFlow*.

Ethanol provee los detalles de las implementaciones de APIs siguiendo un enfoque orientado a objetos, define entidades (*Entities*) que son objetos físicos o virtuales los cuales pueden ser configurados o consultados. Cada entidad tiene sus propiedades, por ejemplo un AP es una entidad física que contiene en sus propiedades el intervalo de tramas beacon entre otras. El controlador se comunica con las entidades mediante interfaces *get* y *set*. Además las entidades pueden tener eventos en los cuales el controlador tome determinadas acciones al respecto.

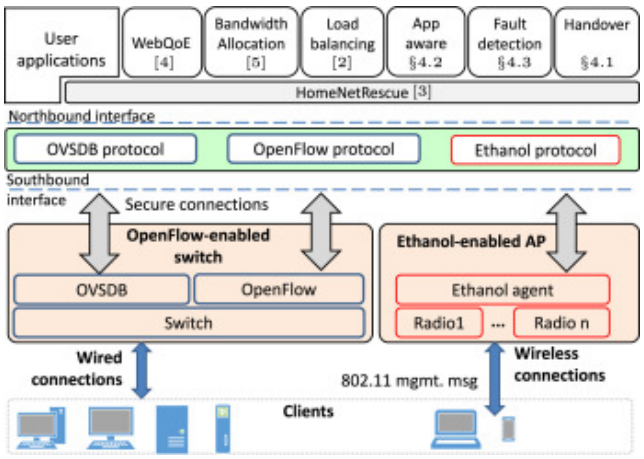

Figura 2.9: Implementación de AP Ethanol. (Moura *et al.*, 2020)

Capítulo 3

5G-EmPOWER

En este capítulo se propone un análisis más extenso sobre el framework de SDN inalámbrico conocido como *5G-EmPOWER* ((Riggio, Rasheed *et al.*, 2013), (*5g-EmPOWER Wiki* s.f.)), una solución alternativa a aquellas vistas en en el Capítulo 2.

A modo de introducción, *5G-EmPOWER* es un sistema operativo de red diseñado específicamente para redes inalámbricas. Como se vio en el Capítulo 2, *OpenFlow* es el protocolo más utilizado para la implementación de redes SDN, sin embargo es programable a muy bajo nivel lo cual implica un mayor esfuerzo para los programadores de red. Durante los años de origen de Empower ya existían soluciones propuestas para un manejo a más alto nivel sobre redes cableadas. Sin embargo la situación no era así para redes inalámbricas y esto motivó a la realización de un *framework* flexible que provea una interfaz sobre un lenguaje de alto nivel para su programación.

El *framework* se basa en la utilización de abstracciones programables de alto nivel para la gestión de redes inalámbricas. Estas abstracciones cubren aspectos de gestión de estados, asignación de recursos, monitoreo de la red y reconfiguración de la red:

- **Gestión de estados**: Las abstracciones deben permitir a los programadores describir el comportamiento deseado de la red. Además el sistema de los dispositivos de enrutamiento, también llamados agentes, deberá ser capaz de configurar el dispositivo para generar el comportamiento deseado. Estos cambios en la configuración de los dispositivos tiene que ser la misma independiente del modelo, marca y tecnología del dispositivo.

- **Asignación de recursos**: Las abstracciones deben permitir a los programadores tener una visión global de los recursos de la red además de permitir la utilización de cualquier modelo de asignación de recursos, desde acceso aleatorio hasta acceso programado.

- **Monitoreo de la red**: Las abstracciones deben ser capaces de brindar una visión global del estado de la red usando consultas con primitivas de alto nivel, como por ejemplo cambios en la topología o estadísticas de la red. Esta implementación deberá ser transparente al desarrollador sin importar el método para obtener esta información, ya sea por polling o por eventos.

- **Reconfiguración de la red**: Estas abstracciones deben separar las operaciones que se ejecutan en los agentes de las tareas ejecutadas en el controlador.

3.1. Abstracciones programables

En esta sección describiremos las abstracciones más importantes del framework ((Riggio, Marina *et al.*, 2015)), estas son *Light Virtual Access Point* (LVAP), *Resource Pool*, *Interference Map y Channel Quality* y *Port*. Un LVAP representa el estado de un cliente conectado de forma inalámbrica en la red el cual tiene asociado un conjunto de *Resources Blocks*, que representan la porción mínima de recursos que se le puede asignar a un cliente. Los LVAPs y WTPs manejan un conjunto de *Resources Blocks* llamado *Resource Pool*. La abstracción Port modela la reconfiguración de los *Resources Blocks* asignados a los distintos LVAPs y WTPs. Por último Los *Interference Map y Channel Map* modelan la calidad de los enlaces entre dos WTPs o entre un LVAP con un WTP.

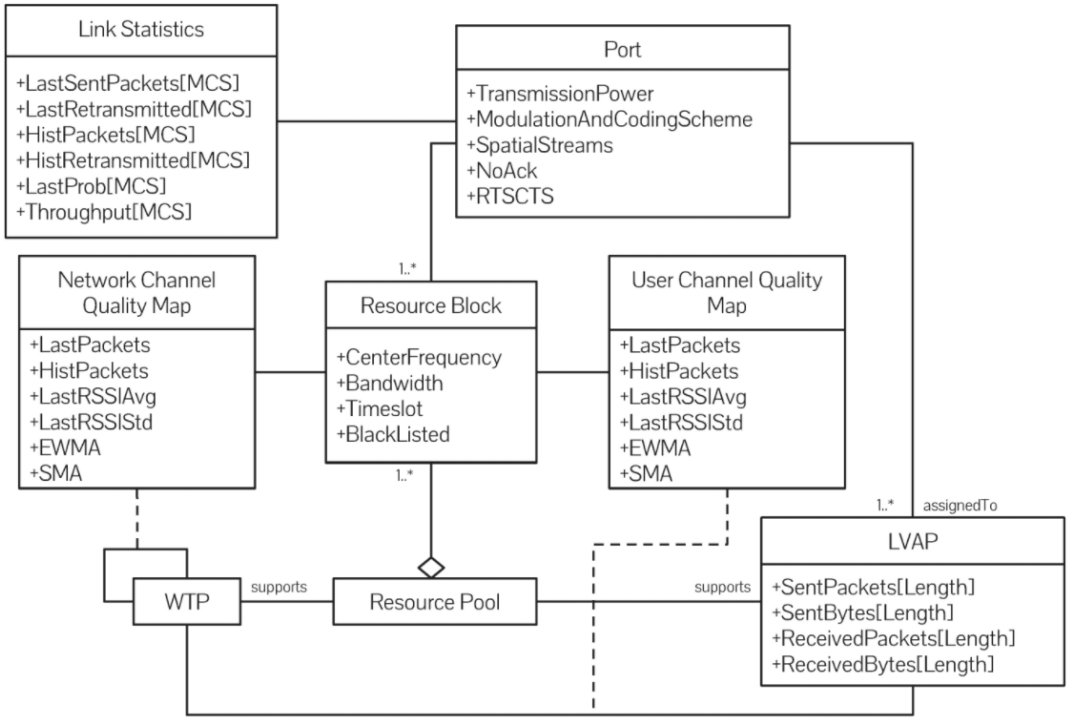

Figura 3.1: Relaciones entre las abstracciones programables. ((Riggio, Marina *et al.*, 2015))

3.1.1. Light Virtual Access Point

Como se mencionó, uno de los aspectos que deben cubrir las abstracciones del framework es que no dependan de la implementación de cada dispositivo. Cada tecnología tiene sus propias implementaciones, por ejemplo de QoS o de como realizar los *handovers*, por lo que los programadores deben modificar sus implementaciones dependiendo de la tecnología que usen los dispositivos.

Las LVAPs proveen una interfaz de alto nivel para manejar el estado de los clientes en una red inalámbrica, por lo que la abstracción resuelve todos los problemas que dependen de la tecnología como la asociación, la autenticación, el *handover* y la asignación de recursos. Esto permite que las aplicaciones sean portables y no se deben reimplementar al cambiar de tecnología, por ejemplo se puede usar la misma implementación para una red WiFi y para una red LTE.

Cuando un cliente accede a una red mediante una trama *probe request* (para obtener información de un AP utilizando el protocolo de redes inalámbricas 802.11) la trama será reenviada al controlador y este generará un LVAP para ese cliente, en el WTP donde se generó la solicitud, además de responder con una trama *probe response*. Por otro lado como el controlador tiene el conocimiento de la red, puede decidir no generar una LVAP si considera que no tiene los recursos necesarios para asignarle al nuevo cliente. Cada WTP va a tener tantos LVAPs como clientes co-

nectados a él. Cada LVAP tiene un identificador único, por lo que se puede pensar que es como un *access point* virtual que tiene su propio BSSID. El uso de LVAPs implica que si un cliente cambia de WTP, se debe realizar únicamente el *handover* del LVAP por lo que queda transparente para el lado del cliente ya que mantiene el estado de la sesión.

Esto además tiene la gran ventaja de que como el controlador tiene el conocimiento de toda la red, puede cambiar los LVAP de WTP para mejorar rendimiento y asignar mejor los recursos y para el cliente será transparente además de que recibirá una conexión de mejor calidad. Por otro lado se puede tener varios WTP asociados a un mismo cliente con la intención de que todos reciban los paquetes enviados por el cliente, aunque solo uno le responderá, esto genera robustez, ya que si el WTP asignado no pudo recibir el paquete por algún motivo, el controlador lo recibirá gracias a otro WTP y podrá manejarlo correctamente.

3.1.2. Resource Pool

Existen principalmente dos estrategias para asignar recursos en una red inalámbrica, el acceso programado (usado por redes LTE) y acceso aleatorio (usado por redes WiFi). Esta abstracción debe ser capaz de proveer ambas estrategias a los programadores de redes. El *Resource Pool* está formado por un conjunto de *Resources Blocks*. Un *Resource Block* se representa como una tupla $\langle f, t \rangle$ donde f es la banda de frecuencia y t es un intervalo de tiempo. A su vez una banda de frecuencia es representada por una tupla $\langle c, b \rangle$, donde c es el centro de frecuencia y b es el ancho de banda. Estos *Resources Blocks* serán asignados a LVAPs, y puede ocurrir que el mismo *Resource Block* esté asignado a más de un LVAP. Este es un escenario probable en redes WiFi, ya que usan acceso aleatorio y por esto también se utilizan mecanismos de retransmisión para manejar las colisiones. Lo que no es posible en redes WiFi es asignar un LVAP a más de un WTP por lo que los *Resources Blocks* de un LVAP deben pertenecer al mismo WTP.

3.1.3. Channel Quality and Interference Map

Es muy importante para los programadores de la red conocer la calidad brindada por cada LVAP, como también la interferencia que está teniendo cada cliente, por ejemplo fijándose en la tasa de paquetes perdidos. Esta abstracción sirve para brindar dicha información a los programadores para que pueden asignar los recursos de la mejor manera posible a los clientes.

El *Channel Quality Map* es un grafo dirigido que representa la calidad entre los distintos LVAPs y WTPs sobre los *Resources Blocks* disponibles, donde los vértices son los WTPs y LVAPs y existe un arista entre dos vértices $(a \rightarrow b)$ si a está en el

rango de comunicación de b sobre el mismo *Resource Block*.

Por otro lado el *Interference Map* es un grafo donde los vértices son las conexiones entre LVAPs y WTPs (por lo tanto lo que era una arista en el grafo anterior, ahora es un vértice) y se genera una arista entre dos vértices ($a \rightarrow b$) si el transmisor del enlace a está en el rango de interferencia del receptor del enlace b sobre un *Resource Block* dado.

Notar que las aristas de estos grafos tienen pesos asignados por lo que a partir de ellos se pueden calcular distintos valores que pueden ser utilizados para asignar LVAPs con WTPs, por ejemplo si se fija un cierto nivel de interferencia (basado en los requerimientos del LVAP), se puede calcular el *signal-to-interference-plus-noise-ratio* (SINR) y si el valor obtenido es mayor al nivel de interferencia fijado, se agrega el resource Block a un conjunto, llamémosle M de posibles asignaciones. Luego basta con tomar los elementos de M para asignar *Resources Blocks* a un LVAP.

3.1.4. Port

Esta abstracción se centra en las características del vínculo entre las LVAP y WTP. Al encontrarnos en un ambiente de conexiones inalámbricas, las características que deben ser tenidas en cuenta son diferentes a las de redes cableadas. Por este motivo es que en función del estado actual del canal, se debe poder modificar la potencia de transmisión, el esquema de modulación y codificación (también llamado MCS, sirve para determinar la velocidad de los datos intercambiados en una conexión inalámbrica) además de la configuración MIMO (define cómo son manejadas las ondas de transmisión y recepción en antenas).

Un Port entonces es una tupla $\langle p, m, a \rangle$, donde los tres valores que representan estas características para cada LVAP, p es la potencia de transmisión, m es el conjunto de MCS disponibles y a es la configuración MIMO (número de *spacial-streams*). Dado que esto define la configuración entre una LVAP y WTP, un WTP manejará tantos Ports como LVAPs tenga a su cargo.

3.2. Slicing

5G-EmPOWER ofrece también la infraestructura necesaria para el particionamiento virtual de una red, también conocido como *Slicing*. En (Richart, Baliosian, Serrat y Gorricho, 2016) *Slicing* se define como "cualquier forma de partición o combinación de un conjunto de recursos de red y presentarlos a los usuarios de manera que cada usuario, a través de su conjunto de recursos particionados o combinados tiene una vista única e independiente de la red. Los recursos pueden ser fundamentales (nodos, enlaces) o derivados (topologías) y se pueden virtualizar de forma

recursiva."

Este particionamiento ofrece la posibilidad de una diferenciación de servicio estando en las mismas condiciones físicas. Para ello, el *framework* ofrece la creación de *Slices*, las cuales son identificadas por el par SSID (nombre de la red) y un número identificador. A una Slice se le asignan un conjunto de propiedades que todo tráfico que cumpla la condición de la Slice tendrá. Estas propiedades son:

- **amsdu_aggregation**: Recibe un valor booleano y en caso de ser verdadero, se utilizará la agregación A-MSDU para las tramas Ethernet de 802.11n.

- **quantum**: Un valor en microsegundos que corresponde al tiempo que se le otorgará en su turno de transmisión de tramas.

- **sta_scheduler**: El algoritmo a utilizar para la tarea de *Scheduling*.

Un paquete IP se identifica como perteneciente a una Slice mediante el cabezal DSCP (*Differentiated Services Code Point*). En dicho campo, el paquete debe tener el identificador asignado a la Slice dentro de la red. Es pertinente mencionar que DSCP utiliza los primeros 6 bits del byte del cabezal Differentiated Services (DS), siendo los últimos 2 bits ignorados. De este modo, si se quiere identificar un paquete para la Slice 1, debería llevar el valor 0x4 en el campo.

Dada la forma de utilización de las Slices, no existe forma de hacer un mapeo Cliente-Slice. Es entonces que en (Riggio y Estefanía Coronado, 2018) se presenta una plataforma de pruebas que es una extensión de 5G-EmPOWER como una posible implementación de esta característica. La principal característica de dicha solución es la utilización de un nuevo controlador, llamado *Backhaul Controller*, que se encarga de agregar el tag DSCP a todos los paquetes que cumplan con ciertas condiciones, por ejemplo IP/Puerto destino y/o origen. Este *Backhaul Controller*, es un controlador de SDN cableado, en ese caso en particular se usa el controlador *Ryu* para etiquetar los paquetes que ingresan a la red. Es importante mencionar que los paquetes que se intercambian internamente no son etiquetados. Únicamente son etiquetados aquellos paquetes que ingresan a la red Empower de forma externa ya sea desde internet, producidos por el controlador o por otro dispositivo externo a la red Empower.

Para cumplir con los requerimientos de las Slices se define en los agentes una abstracción programable *TrafficRule* que, considerando los datos de las Slices creadas en el controlador, tomara decisiones al momento de enrutar los paquetes. En (Riggio y Estefanía Coronado, 2018) se describe el comportamiento de esta abstracción incluyendo los algoritmos de encolamiento y des-encolamiento de paquetes en los APs.

3.3. Arquitectura y componentes

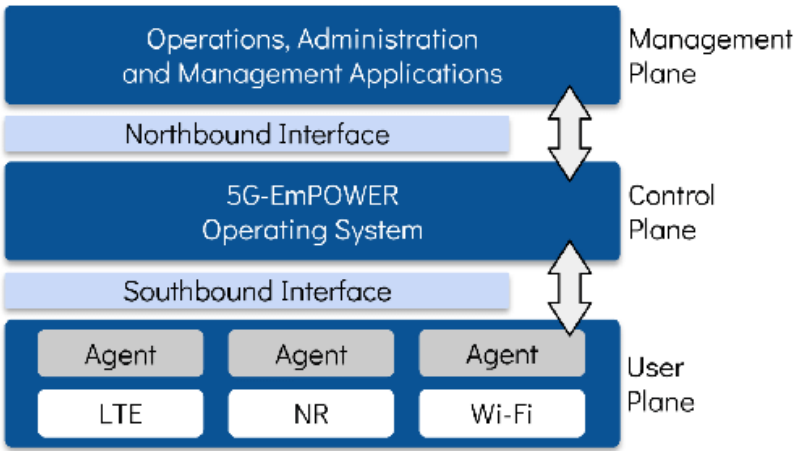

Figura 3.2: Vista de alto nivel de la arquitectura de EmPOWER. (Estefanía Coronado *et al.*, 2019)

La arquitectura de 5G-EmPOWER está compuesta por tres capas: una capa de usuario, una capa de control y otra capa de gestión (Ver Figura 3.2). La capa de usuario abarca los elementos de red en la RAN, esto incluye tanto a los puntos de acceso de WiFi (APs), como los nodos eNB LTE. La capa de control consiste del sistema operativo 5G-EmPOWER, el cual se comunica con los Agentes 5G-EmPOWER (va a existir un agente por dispositivo en la RAN). El agente 5G-EmPOWER recibe comandos del sistema operativo 5G-EmPOWER usando el protocolo *OpenEmpower* (Sección 3.3.1), que actua como *Southbound* API (al igual que *OpenFlow* para redes cableadas), proporcionando estadísticas y eventos de la red. Las aplicaciones de administración y gestión residen en la capa de gestión, comunicándose con la capa de control usando la *Northbound* API (Sección 3.3.4).

3.3.1. Interfaz Southbound: protocolo OpenEmpower

Si bien *OpenFlow* es el protocolo más utilizado y aceptado para SDN, sus características están centradas en redes cableadas y es de compleja adaptación para controlar redes inalámbricas. Por este motivo es que *OpenEmpower* (Estefanía Coronado *et al.*, 2019) fue desarrollado como un protocolo que permite el manejo remoto de elementos RAN sin asumir el tipo de conexión. Puede ser usado tanto en Wi-fi APs como en nodos LTE. Este protocolo se ubica sobre TCP y puede utilizar el protocolo TLS. El protocolo *OpenEmpower* se basa en tres tipos de eventos distintos:

- **Eventos individuales**: Son peticiones independientes de otras instrucciones, que dan instrucción o piden información por parte del controlador al dispositivo

que corre el agente. Como ejemplos de este tipo de evento están la consulta de las capacidades del dispositivo, o la asignación de una LVAP a un diferente WTP.

- **Eventos programados**:Son peticiones del controlador al agente para que mande periódicamente información de algún tipo, como ser un reporte cantidad de paquetes intercambiados que cumplan cierta condición.
- **Evento disparado**: El controlador puede decirle al agente que en caso que se cumpla una cierta condición, se mande un mensaje de aviso al controlador.

Todos los mensajes *OpenEmpower* empiezan con un cabezal común, en el cual se especifica la versión del protocolo, el tipo de evento, el largo del mensaje, el identificador del elemento RAN, el identificador de la celda/interface (dependiendo si es un eNB o AP), un identificador transaccional (*TransactionID*) y un número de secuencia. Dada la naturaleza de los eventos, los mensajes y sus respuestas son asíncronas, por lo cual el protocolo utiliza un campo *TransactionID* para identificar la respuesta con su solicitud.

El cabezal en común es seguido por un cabezal de evento, en el cual se especifica el tipo de acción, un código operacional (*opcode*) y en el caso de los eventos programados, se incluye el intervalo de tiempo. Luego del cabezal de evento, se coloca el cuerpo del mensaje. (Ver Figura 3.3)

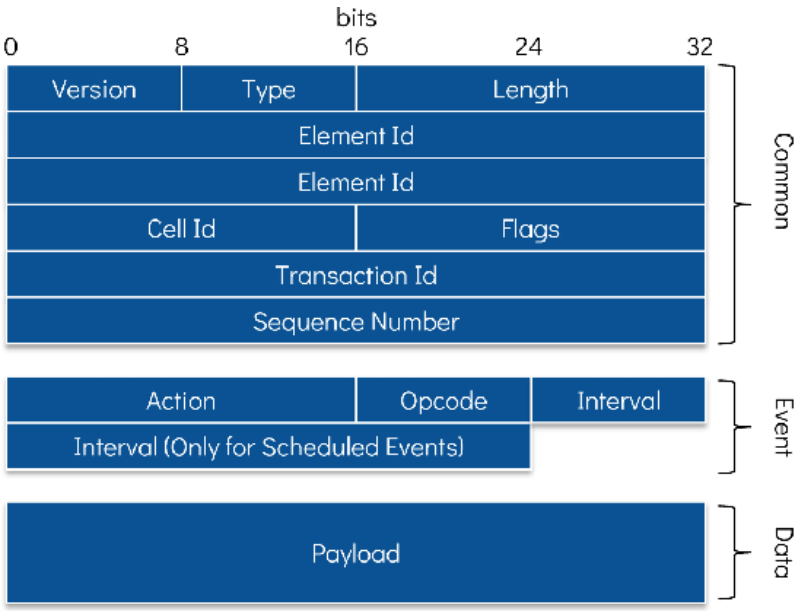

Figura 3.3: Estructura de un mensaje OpenEmpower. (Estefanía Coronado *et al.*, 2019)

Por último hay que aclarar que la implementación de este protocolo para el uso en redes Wi-Fi es soportado en APs basados en el sistema operativo *OpenWRT*.

3.3.1.1. Mensajes OpenEmpower

Si bien el protocolo OpenEmpower se menciona en Estefanía Coronado *et al.* (2019), no hay documentación disponible, por lo que este análisis se basa en inspección del código.

El protocolo OpenEmpower define un conjuntos de mensajes que intercambian el controlador con los agentes para el correcto funcionamiento de la red. Este conjunto de mensajes incluye mensajes *core* (por ejemplo los mensajes *Hello*, mensajes *Probe*, entre otros) , mensajes para le gestión de LVAPs, mensajes para definir las políticas de transmisión, mensajes para recolección de estadísticas, etc.

La lista completa de los mensajes definidos por el protocolo OpenEmpower se encuentra en Capítulo A.3.

3.3.2. Agente EmPower

El agente 5G-EmPOWER (*Repositorio git del agente empower* s.f.) es el encargado de gestionar la capa de usuario. A grandes rasgos, un Agente es un router *OpenWRT* el cual tiene instalado y configurado el paquete de *Empower-LVAP-Agent*. El agente es capaz de serializar y deserializar paquetes *OpenEmpower*, y utilizando gestores de eventos (uno para los eventos individuales y programados y otro para los eventos disparados) ejecuta comandos sobre el router. El agente está implementado sobre la arquitectura *Click Modular Router*. Para más información sobre este *framework* ver (Kohler *et al.*, 2000).

3.3.3. Controlador

La función del controlador (*Repositorio git del controlador empower* s.f.) es reconocer y centralizar todos los requerimientos que llegan desde las aplicaciones de red, para luego comunicarle los cambios a los agentes. Para cumplir con su función, el controlador mantiene un modelo propio del estado de la Red actual.

Esto lo logra por medio de la implementación de los siguientes conceptos:

- **WTP**: Una clase donde se almacena información sobre los Access Points, como ser su dirección física, una descripción legible o información de los diferentes bloques que lo componen.

- **Block**: Existe una clase llamada *ResourceBlock* que modela los diferentes bloques que tiene un WTP. Allí se guardan como atributos una instancia del WTP al que pertenece, la dirección física, su canal y su banda entre otros datos.

Además, se creó una clase llamada *ResourcePool* que extiende una lista, cuya funcionalidad es ser una lista de instancias de *ResourceBlock*. Esta extensión provee métodos interesantes para una lista de bloques:

- filter_by_band(band): Función que retorna la lista filtrada por la banda recibida por parámetro.
- filter_by_channel(channel): Función idéntica a la anterior, solo que esta vez utilizando el canal como filtro.
- sort_by_rssi(addr): Método que ordena la lista basándose en el valor de RSSI medido desde el dispositivo recibido por parámetro.
- first() y last(): Devuelven la instancia de *ResourceBlock* que esta en primer o último lugar.

- **LVAP**: Esta clase guarda la información de un cliente. Algunos de los datos que se tienen son su dirección física, su SSID e incluso el estado en el que se encuentra. Sin embargo, lo que más interesa de esta implementación son sus atributos *wtp* y *blocks*, que representan el WTP y el *ResourceBlock* al cual el cliente está asignado. Esto es de particular interés porque modificando los valores de estos atributos se logra realizar un *handover* del cliente. En el atributo *wtp* se puede almacenar una instancia de la clase mencionada en el punto anterior, mientras que en *blocks* se puede guardar una instancia de *ResourcePool* donde el primer bloque de la lista será el bloque que utilizará el cliente para bajada de datos.

- **Slice**: Aqui se modela una *slice* por medio de una clase que almacena un conjunto de propiedades y el ID de la slice, que será su identificador para diferenciar tráfico usando el cabezal *DSCP*. El conjunto de propiedades se compone de *amsdu aggregation*, *quantum*, y el tipo de *scheduler* (hay tres opciones: *Round Robin*, *Deficit Round Robin* o *Airtime Deficit Round Robin*). Por último, una *slice* puede tener un comportamiento diferente en WTPs distintos, por lo que también se tiene un mapa clave-valor, siendo la clave la dirección física de un WTP y el valor una instancia de conjunto de propiedades.

Para tener un control general sobre la Red existe una clase llamada *EmpowerProject*, donde se almacenan referencias a las clases mencionadas anteriormente. Es entonces que se tiene una lista de WTPs, LVAPs y Slices, acompañados de métodos de manipulación de *slices*, llamados *upsert_wifi_slice* y *delete_wifi_slice*, cuyas funciones son crear/editar y borrar *slices* respectivamente.

3.3.4. Interfaz Northbound y Aplicaciones de Red

Un *framework* basado en SDN debe otorgar a los programadores de red una interfaz de alto nivel que le permita obtener datos de la red además de actuar sobre ella. En el caso de 5G-EmPOWER existen dos posibilidades de interactuar con el controlador: por intermedio de una *API Rest* (*Empower Rest API* s.f.) o implementando una aplicación *Python* extendiendo la librería ofrecida por el sistema Empower.

Para la implementación de una aplicación nativa, dentro del ambiente de ejecución se debe crear un nuevo directorio en la ubicación de las aplicaciones del código que implementa el controlador. Allí se debe crear un archivo *Python*, donde se debe implementar una clase que extienda a *EWiFiApp*. Una aplicación corre sobre un contexto de ejecución, que es una instancia de la clase Proyecto vista en la Sección 3.3.3. Entonces extenderse de la clase mencionada anteriormente otorga acceso a una variable de la clase llamada *context*, donde se tiene una referencia a la instancia de *Project* donde se esta ejecutando. Esto permite a una aplicación acceder a todas las instancias de WTPs, LVAPs y Slices y todo aquel método que pertenezca a *Project*. Además de todos esos datos, se hereda una variable *blocks* que lista todos los bloques de todos los WTPs.

La clase que describe nuestra aplicación debe implementar, aparte del constructor, una función *loop* y una función *launch*. En la función *loop* se encontrará el código que se ejecutará efectivamente cada vez que se ejecute la aplicación, mientras que en *launch* se debe retornar una instancia de la clase.

Por otra parte, también se pueden implementar métodos de callback que serán invocados automáticamente bajo ciertos escenarios:

- **handle_client_leave(lvap)**: Es llamado cuando un cliente deja la red sin importar si era un cliente del proyecto donde se está ejecutando la app o no.
- **handle_lvap_leave(lvap)**: Es invocado cuando un cliente deja la red.
- **handle_client_join(lvap)**: Es llamado cuando un cliente se une a la red, no necesariamente al proyecto donde está corriendo nuestra app.
- **handle_lvap_join(lvap)**: Es invocado cuando un cliente se une a la red.
- **handle_wtp_down(wtp)**: Se llama cuando un WTP se desconecta del controlador.
- **handle_wtp_up(wtp)**: Es llamado cuando un WTP se conecta con el controlador.

Capítulo 4

Despliegue de una red gestionada por EmPOWER

Luego de haber investigado y de comprender la herramienta Empower, realizamos un despliegue de una red inalámbrica gestionada por Empower. En este capítulo se detallan las configuraciones y pasos que se siguieron para desplegar la red, y en particular la explicación de la configuración de la red, del controlador y de los WTPs, como asimismo la implementación de *slicing* en la red desplegada.

4.1. Configuración de la red

Al desplegar una red gestionada por Empower hay que tener en cuenta un aspecto importante: los WTPs cuando se conectan al controlador no tienen una IP, y cuando un cliente se conecta a la red, ni el controlador ni ninguno de los WTP le asignan una IP a dicho cliente, por lo que es necesario contar con un servidor DHCP para la asignación de IPs. Por otro lado, si se desea que la red tenga acceso a internet se debe contar con un servidor que se encargue de implementar NAT y DNS. Estas funcionalidades las provee cualquier *router* que brinda internet en un hogar, por lo que basta con conectar este *router* a la red Empower.

Dicho esto, además de contar con un servidor que implemente DHCP, DNS y NAT (*router* del hogar), necesitamos un *switch* con los puertos necesarios para conectar el *router* del hogar, el controlador y los WTPs. En nuestro caso usamos un *switch* TP-Link de 8 puertos.

En la siguiente figura se muestran los componentes necesarios para desplegar una red con un controlador y dos WTP. Las lineas implican que hay una conexión cableada.

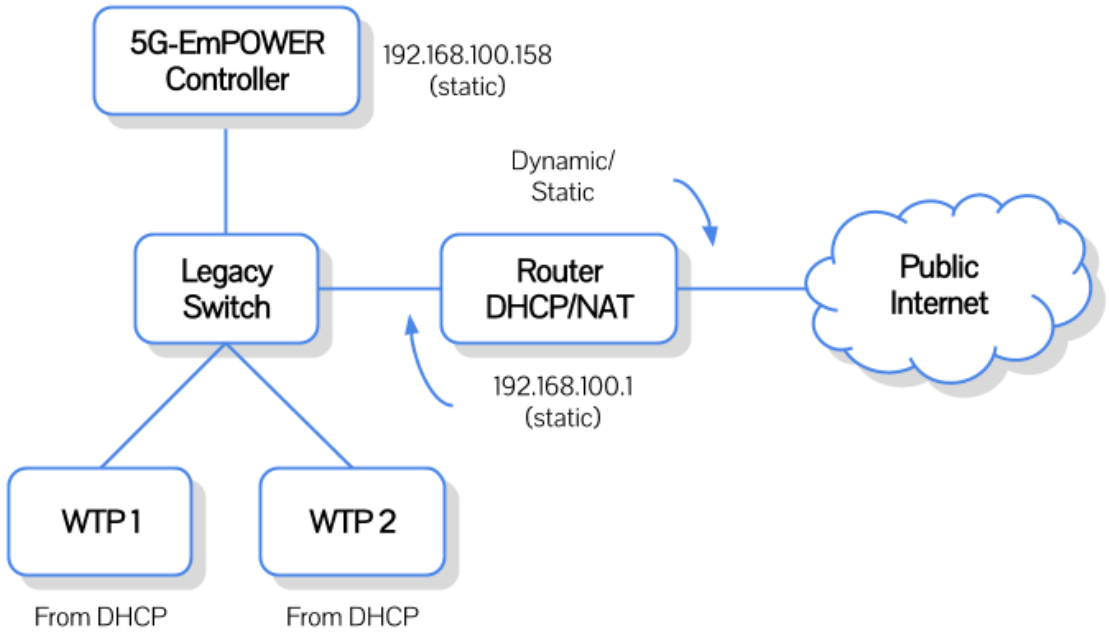

Figura 4.1: Configuración de una red simple. (*Red de referencia* s.f.)

4.2. Configuración del controlador

Para configurar el controlador primero tuvimos que entender la estructura del código. El controlador esta compuesto principalmente por dos módulos: *empower-runtime* y *empower-core*. En el módulo *core* se encuentran los gestionadores de usuarios, proyectos y acceso a la base de datos, así como algunas clases comunes, por ejemplo la clase SSID y la clase *EtherAddress* que representa una dirección MAC. Por otro lado, en el módulo *runtime* se encuentran los gestionadores de Slices, WTPs, aplicaciones, *Workers*, Wifi ACLs, Redes LTE y LVAPs y sus conexiones. Además, en el módulo *runtime* se encuentra la interfaz web para interactuar con el controlador. La clase *empower-runtime* dentro de este módulo es la encargada de levantar el controlador.

En un principio el módulo *core* estaba dentro del módulo *runtime* y por las fechas en que se comenzó este proyecto los fundadores de Empower decidieron separar el *core* del *runtime*. Esto nos trajo problemas ya que en la rama *master* del repositorio de Empower, no había una versión estable ni tampoco una etiqueta con el código antes de comenzar a separar el *core* del *runtime*. Por esto decidimos realizar los cambios necesarios para que el controlador funcione con el modulo *core* dentro del *runtime*, como funcionaba previamente.

Por último, se decidió usar *Docker* para no depender del sistema operativo ni de características de la computadora donde se ejecute el controlador. También se partió

de la base de que Empower tiene un repositorio con imágenes *Docker*, en particular un *Dockerfile* para levantar el controlador. En este punto se presentó otro problema: la imagen *Docker* del repositorio Empower no funcionaba por lo que se creó una imagen propia *Docker* para levantar el controlador. Este archivo se puede descargar de nuestro repositorio (*Repositorio git del proyecto* s.f.).

Por lo tanto para levantar el controlador Empower, se debe abrir una terminal donde esta situado el *Dockerfile* y ejecutar los siguientes comandos:

```
$ docker build --build-arg CACHE_DATE="$(date)" -t empower-runtime .
$ docker run --net=host --rm --privileged -it empower-runtime
```

Al iniciar *Docker* no se levanta el controlador automático (si se desea este comportamiento basta con descomentar la linea "ENTRYPOINT" del *Dockerfile*) y hay que levantarlo ejecutando el siguiente comando en el contenedor:

```
$ /bin/sh entry.sh
```

Este *script* "entry.sh" inicializa los servicios de las bases de datos y ejecuta "empower-runtime.py" para levantar el controlador.

4.3. Configuración de un WTP

Para configurar el WTP, se instaló sobre un router el sistema OpenWRT con el paquete del agente Empower ya instalado. Para generar esta imagen OpenWRT, se clonaron los repositorios (*Repositorio git para compilar imagen del agente* s.f.) y (*Repositorio git con las configuraciones del agente* s.f.). Luego, en una terminal sobre el directorio empower-openwrt, se ejecutaron los siguientes comandos:

```
$ ./scripts/feeds update -a
$ ./scripts/feeds install -a
### reemplazar wdr4300 si se usa otro router
$ ln -s ../empower-configs/empower-files-wdr4300/ files
### menu interactivo para crear la imagen
$ make menuconfig
$ make -j 3
```

Por otra parte, Empower tiene un repositorio con imágenes ya creadas para algunos *routers* (*Repositorio con imagenes OpenWRT del agente Empower* s.f.). En este caso, como el modelo de los routers utilizados es TP-Link WDR 4300, que tiene imagen ya creada en el repositorio Empower, se descargó la imagen de ese repositorio aunque también se hizo la prueba de construir la imagen usando los comandos previamente indicados.

Una vez instalado el agente, se realizó una configuración mínima en el router. Se editó el archivo *empower* del directorio */etc/config/*, modificando la "master ip" por la IP del controlador (IP de la interfaz conectada al switch). Por esto, es recomendable que el *router* DHCP le asigne siempre la misma dirección IP al controlador.

Para realizar este último punto es importante tener en cuenta que a pesar de ser un sistema OpenWRT, al conectar el router a una computadora, la IP del router no es 192.168.1.1 si no que es 192.168.250.1, además que la contraseña no es *root*, si no que es *empower*.

Por último, luego de haber editado el archivo /etc/config/empower, se hace necesario reiniciar el servicio *empower* ejecutando:

```
$ /etc/init.d/empower restart
```

4.4. Inicializar la red

Luego de haber configurado los WTPs y estar ejecutando el controlador Empower, se puede crear una o más redes virtuales. Para crear una red virtual basta con ingresar un SSID y una descripción de la red.

Antes o después de haber creado la red, se debe especificar una lista con las direcciones MAC de los WTPs que se van a utilizar. Los WTPs al estar configurados con el agente, mandarán constantemente mensajes *Hello* al controlador, pero solo cuando se configuran en el controlador es que esos WTPs pasan a ser utilizables en la red.

Por último, dado que el controlador no soporta autenticación de clientes mediante WPA, se debe especificar una Access Control List (ACL), que no es más que una lista de direcciones MAC. Solo los clientes a los que se les ingresó su MAC en la ACL podrán ver el SSID de la red y por lo tanto ingresar a ella.

Para ver en detalle como levantar una red, ver la Sección A.1

4.5. Implementación Slicing

Para la utilización de las funcionalidades de Slicing, es necesario crear una o más Slices. A la hora de crearlas se debe ingresar las características de cada una de ellas, como el *quantum*, y el algoritmo de *Scheduling*.

Como se detalló en la Sección 3.2, un paquete IP es categorizado por la plataforma de 5G-EmPOWER mediante el cabezal DSCP (*Differentiated Services Code Point*). En dicha sección se menciona un trabajo donde se propone una forma de asignar automáticamente ese cabezal en base a ciertas condiciones usando un controlador SDN, en particular *Ryu*. Debido a que no es el objetivo de este trabajo instalar

un controlador SDN para etiquetar paquetes, se decidió seguir por otro camino con el objetivo de asignar tráfico a una Slice u otra.

Es entonces que para enviar tráfico a hacia la red 5G-EmPOWER se eligió utilizar iPerf3 (*sitio iperf3* s.f.), una herramienta de consola que permite generar tráfico de manera sencilla. Esta herramienta brinda, entre varias opciones, la posibilidad de enviar tráfico con el cabezal DSCP que se le indique, por lo que su uso para esta situación es de gran auxilio.

La instalación de dicha herramienta es relativamente sencilla en todo ámbito. Para ambientes Linux se puede realizar por línea de comando dependiendo de la distribución, mientras que para sistemas Windows existe un ejecutable que se puede descargar desde la página oficial. En las pruebas realizadas también se tomaron routers OpenWRT, los cuales pueden descargar el paquete de iPerf3 desde su interfaz web.

La utilización de esta herramienta se basa en que desde un lado debe haber un servidor escuchando y, en contraparte un cliente le envía tráfico. Para levantar el servidor basta con ejecutar el siguiente comando:

```
$ iperf3 -s -p {puertoServidor}
```

Mientras que para enviar tráfico se ejecuta lo siguiente:

```
$ iperf3 -c {IPServidor} -p {puertoServidor} --dscp {ValorDSCP}
```

Existen también opciones para especificar el protocolo de transporte, *bandwidth* y muchas otras opciones que pueden ser útiles a la hora de hacer pruebas sobre la plataforma. Considerando que DSCP utiliza solo los primeros 6 bits del byte del cabezal por lo que si se quiere enviar tráfico etiquetado con la Slice 1, se debe enviar 4 en “ValorDSCP“.

Capítulo 5

Caso de Uso: Diferenciación de Servicio

Luego de desplegada una red Empower y tener un conocimiento de la herramienta, el siguiente objetivo de este proyecto era desarrollar, desplegar y evaluar una aplicación que sea de utilidad para una red SDN inalámbrica.

5.1. Problema y heurística propuesta

Para desarrollar una aplicación útil, la misma debe mejorar algún aspecto en la red o resolver algún problema que sea de interés. Es por esto que antes de explicar la aplicación vamos a presentar el problema que pretende solucionar la aplicación desarrollada.

5.1.1. Descripción del problema

La funcionalidad que se tomó como objetivo es la de ofrecer a los clientes de una red una tasa de transferencia mínima garantizada. En un escenario de este estilo existirán clientes pertenecientes a diferentes grupos en donde cada grupo tendrá una promesa de tasa de bajada, es decir desde el AP al cliente.

Por su parte, como se vio en el Capítulo 3 algunas de las funcionalidades más interesantes que ofrece Empower, son las de *Slicing* a través de repartir *airtime* implementado con *quantums* y *round robin*, además del manejo de la conexión de los clientes sobre los APs.

5.1.2. Modelado del problema

En este escenario se les promete una tasa de transferencia de bajada mínima en promedio a cada *slice*. Cada cliente pertenece exclusivamente a una *slice* y, depen-

diendo de lo otorgado a ella, tendría una tasa de bajada mínima garantizada. Para la implementación se asume que el algoritmo conoce una lista con cada *slice* y la tasa mínima a garantizar, y otra lista con cada cliente y la *slice* a la que pertenece. De este modo entonces se calcula qué tasa se le prometió a cada cliente.

Para cumplir con el objetivo, la idea desarrollada se basó en usar diferentes métricas y estadísticas acerca del tráfico y su calidad que pueden ser extraídas de los agentes. En base a esta información es que podemos tomar acciones, como ser el traslado de una LVAP hacia un WTP donde tendrá mejor rendimiento, o también la diferenciación del servicio entre clientes, otorgando más *airtime* a aquellos que se considere pertinente.

El problema de administración de la red entonces se puede formular como un problema de optimización. Para ello se deben definir parámetros del escenario, una función objetivo, un conjunto de restricciones y un conjunto de variables de decisión.

En primer lugar se tiene que tener en claro el escenario al cual nos enfrentamos. Esto implica conocer la tasa prometida para todas las slices de la red, la cual se representará de la manera $Tprom_s$, siendo s la *slice*. Por otra parte, también es necesario conocer el estado del canal entre clientes y WTPs. Esta relación se representará de la forma C_i^j, siendo i el cliente y j el WTP.

Lo siguiente a definir son las variables de decisión del problema. En este caso se reducen a dos:

A_i^j: con valor de 0 o 1, representa si el cliente i está conectada con el WTP j.

R_s^j: es el porcentaje de *airtime* que se le otorgará a la *slice* s en el WTP j.

Pasando ahora a las restricciones del problema:

- Para todo LVAP i tiene que existir un $A_i^j = 1$ y es único (todo LVAP está en exactamente 1 WTP): $\forall i, \exists! A_i^j : A_i^j = 1$, donde i es un LVAP.
- Todos los porcentajes de *airtime* son un valor entre 0 y 1: $\forall j, \forall s, 0 \leq R_s^j \leq 1$, donde j es un WTP y s es una *Slice*.
- La suma de porcentajes de airtime dado un WTP debe ser menor o igual a 1: $\forall j \sum_{s=0}^{S} R_s^j \leq 1$, donde S es el conjunto de *slices* y j es un WTP.
- Para todo cliente, su tasa real es mayor a igual a la prometida: Siendo $Treal_i$ la tasa real de el cliente i, $\forall i Treal_i \geq Tprom_i$.

Por último, lo que queda por definir es la función objetivo, la cual es simplemente maximizar la tasa recibida por todos los LVAPs, lo cual se representa de la siguiente manera:

Siendo S el cunjunto de *Slices*, I el conjunto de clientes, y J el conjunto de APs,

$$max \sum_{s=0}^{S} \sum_{i=0}^{I} Treal_i \tag{5.1}$$

Como se mencionó, la tasa de transferencia va a depender de las asignaciones de

los clientes a los WTPs y del *airtime* que tiene ese cliente en el WTP, por lo que se puede reformular la sumatoria anterior de la siguiente manera:

$$max \sum_{s=0}^{S} \sum_{i=0}^{I} C_i^j \cdot R_s^j \cdot A_i^j \tag{5.2}$$

Como se trata en (Richart, Baliosian, Serrat, Gorricho y Aguero, 2019), en donde se plantea un problema muy similar al descripto en esta sección y se presenta una solución para un único AP, no se trata el caso de múltiples APs. Hallar una solución exacta a este problema es muy complejo de resolver, ya que C es variable y desconocida, solo se puede conocer su valor instantáneo. Encontrar una solución a este problema está por fuera del alcance de este proyecto. En la Sección 5.1.3 se expondrá una heurística para aproximar una solución.

5.1.3. Descripción de la heurística

La heurística propuesta es un algoritmo de reglas. Dependiendo del estado actual del cliente, el algoritmo deberá decidir si tomar una acción o no. Si el algoritmo decide tomar una acción, el algoritmo debe ser capaz de determinar qué acción tomar.

Para implementar la heurística, se tuvieron en cuenta las acciones que se pueden realizar en el controlador EmPOWER así como las estadísticas que se pueden obtener de los agentes. En resumen el agente provee estadísticas sobre RSSI (que en este contexto es la forma de representar la calidad del canal), tasas de transmisión de LVAPs y porcentaje de uso de los WTPs, mientras que en el controlador se pueden tomar acciones como realizar un *handover* (implica el traslado de un cliente desde un WTP a otro) o modificar el *quantum* de una *slice* para cualquier WTP. Los detalles de implementación se pueden ver en la Sección 5.2. La presente sección está centrada en la descripción de la heurística.

El primer problema a resolver fue qué estadísticas usar para detectar que hay un problema en la red. Es importante tener en cuenta que una aplicación en un ambiente Empower se ejecuta automáticamente cada un intervalo de tiempo a determinar. La propuesta para el algoritmo que se ejecuta en cada intervalo fue, primero para cada LVAP, calcular la diferencia entre la tasa de transferencia efectiva ($Tactual$) del cliente y la tasa intentada ($Tintent$) desde la última iteración hasta la iteración actual. Si la diferencia es mayor al 10 % de $Tintent$ y además $Tactual$ es menor a su tasa prometida ($Tprom$), entonces implica que hay un cliente con el que no se está cumpliendo con su contrato (mínima tasa de transferencia prometida) y por lo tanto el algoritmo deberá tomar alguna acción.

Si en cambio, el cliente está teniendo una diferencia en las tasas mayor al 10 %,

pero su tasa de transferencia actual es mayor a la prometida, entonces no se realizará ningún cambio ya que se está cumpliendo con el contrato del cliente.

Por último si se detecta que el cliente tiene una diferencia en la tasa de transferencia actual e intentada menor al 10 %, no se debe realizar ninguna acción ya que el cliente no está presenciando problemas mayores.

En resumen esta parte del algoritmo se puede apreciar en el Algoritmo 1:

Algorithm 1: evaluarRed()

```
1  foreach slice in Slices do
2  |  Tprom ← rateProm(slice)
3  |  foreach lvap in LVAPs do
4  |  |  Tactual ← rateActual(lvap)
5  |  |  Tintent ← rateIntentado(lvap)
6  |  |  if Tintent − Tactual = Tintent · 0,1 then
7  |  |  |  if Tactual < Tprom then
8  |  |  |  |  tomarAccion(slice, lvap, Tprom, Tintent, Tactual)
9  |  |  |  else
10 |  |  |  |  doNothing    ▷ Cliente tiene una tasa mayor a la prometida
11 |  |  |  end
12 |  |  else
13 |  |  |  doNothing              ▷ Cliente no presencia problemas
14 |  |  end
15 |  end
16 end
```

Para solucionar el problema de que un LVAP no esté gozando de la tasa prometida se visualizan dos posibles soluciones: por un lado es posible moverlo de WTP donde pueda tener más recursos disponibles, y por otro lado se puede garantizar más *airtime* a la *slice* a la cual pertenece.

En el algoritmo propuesto se prioriza la búsqueda de un nuevo WTP sobre el cambio de *quantums* en *slices*. En primer lugar se buscará un WTP adecuado para el traslado y en caso de no encontrar un candidato, se procederá con el ajuste de *quantums* para la mejora del servicio.

Algorithm 2: tomarAccion(slice, lvap, Tprom, Tintent, Tactual)

```
1 if intentarHandover(slice, lvap, Tprom, Tintent, Tactual) then
2 |  print('Se realizó un Handover')
3 else if intentarQuantum(slice, lvap, Tprom, Tintent, Tactual) then
4 |  print('Se modificaron quantums')
5 else
6 |  doNothing                    ▷ Red saturada
7 end
```

Para decidir si se realiza un *handover*, primero filtramos los WTPs que tengan

buen RSSI sobre el LVAP, de esta manera se sabe que no se deberá mover el cliente a un WTP donde recibirá mala señal. El mínimo RSSI aceptado es otro parámetro configurable del algoritmo. Una vez que se tienen los bloques filtrados, hay que decidir qué parámetro priorizar para elegir el WTP "óptimo".

Notar que utilizar la tasa de transferencia del WTP (suma de todas las tasas de transmisión a los clientes del WTP) y elegir el WTP con menos tasa no es posible debido a que esta medida no representa la cantidad de recursos disponibles. Por ejemplo, si el WTP tiene clientes relativamente lejos, entonces tendrá en total poca tasa de transferencia, pero usan más tiempo de aire por lo que puede estar consumiendo más recursos que el WTP actual.

Como se quiere encontrar el WTP que tenga más recursos disponibles, se filtran los WTPs (que ya fueron filtrados por RSSI) por porcentaje de uso, donde solo quedan los WTPs que tienen menos porcentaje de uso de transmisión que el WTP actual del LVAP. Si se encuentra algún WTP con menos uso, se trasladará el LVAP al WTP que tenga menos porcentaje de uso.

Realizar solo esta heurística tiene el problema de que si todos los WTP tienen LVAPs transmitiendo a su máxima capacidad, no se podría realizar el *handover*. Sin embargo, es posible que esos LVAP que están saturando los WTPs lo hacen teniendo una tasa de transferencia más alta a la prometida por lo que se podría repartir los recursos de ese WTP con la finalidad de que todos las tasas de transmisión prometidas se cumplan.

Por este motivo, en el caso de que todos los WTPs estén con más uso que el WTP actual, se revisará la tasa de transferencia de cada LVAP que no esté en el actual WTP. Se calcula la tasa de transferencia total de LVAPs que están pasadas (tasas de transferencia por encima de sus tasas de transferencia prometidas, a la suma de estas tasas de transferencia pasadas se le llamará $Textra$) para cada WTP. Por ejemplo si en un WTP hay 2 LVAPs a los cuales se les transmite de forma efectiva a 10Mbits/s y la tasa de transferencia prometida de cada uno es 2Mbits/s, entonces el WTP está "pasado" por 16 Mbits/s respecto a su tasa prometida ($Textra = 16Mbits/s$). Luego se filtran los WTP que tengan $Textra$ un 20 % mayor a la tasa de transferencia actual del cliente . Luego en caso que haya más de un WTP candidato que cumplan con esa condición, se elige el WTP que tenga mayor $Textra$.

Con esta solución aún existen problemas que ocurren en ciertos escenarios. El primero de ellos es que se genere un traslado masivo hacia un WTP. Dado que la evaluación de cada LVAP se hace independientemente del resto de ellas, para toda aquella LVAP que precise movimiento siempre se eligirán los mismos WTPs candidatos dentro de una misma iteración del algoritmo. Si no se controla, esto puede provocar que en una sola iteración muchas LVAPs sean colocados en un mismo WTP, lo que en consecuencia genera una saturación del mismo y se vuelve al paso inicial,

donde los clientes no van a llegar a su tasa de transferencia prometida. Si no se controlan estos casos, puede llegar a ocurrir que luego de varias iteraciones no se haya mejorado la calidad del servicio.

Para contrarrestar entonces este problema, se agregó un máximo de traslados hacia un WTP en una iteración. Esto pasa a ser un parámetro configurable del algoritmo que, establecido lo suficientemente bajo, controla que no ocurra este tipo de situaciones. Sin embargo, es pertinente aclarar que el uso de esta restricción puede llevar a un retraso en los traslados necesarios. El valor de este parámetro dependerá del tamaño de la red.

Otro problema que puede existir es un efecto "ping-pong", en donde uno o más LVAPs vayan saltando entre los mismos WTPs en varias iteraciones del algoritmo. Este caso se presenta habitualmente cuando a nivel general se prometió más tasa a los clientes de la que la infraestructura actual pueda garantizar. En consecuencia, los LVAPs comienzan a cambiarse de un WTP a otro, sin obtener su resultado esperado.

En este caso, como medida preventiva se decidió guardar un registro de todos los *handovers* que el algoritmo haya hecho. De este modo, cuando se analice un WTP para ser un posible destino, primero se mirará el historial. Si se detecta que agregando este WTP como destino, se repite una secuencia de *handovers*, no se lo tendrá en cuenta como candidato.

En el algoritmo 3 se presenta un pseudocódigo de la etapa del algoritmo "intentar

un handover".

Algorithm 3: intentarHandover(slice, lvap, Tprom, Tintent, Tactual)

```
1  posibles_handovers ← []
2  actual_usage ← wtpUsage(lvap.wtp)
3  foreach wtp in WTPs, donde RSSI(lvap, wtp) >RSSI_MININMO do
4  |   wtp_usage ← wtpUsage(wtp)
5  |   if wtp_usage >actual_usage and !pingPong(wtp) and
   |    !maxHandovers(wtp) then
6  |   |   posibles_handovers.push(wtp)
7  end
8  if largo(posibles_handovers) >0 then
9  |   wtp_opt ← getMinUsage(posibles_handovers)
10 |   handover(lvap, wtp_opt)
11 else
12 |   foreach wtp in WTPs, donde RSSI(lvap, wtp) >RSSI_MININMO do
13 |   |   Textra ← getExtraRate(wtp)
14 |   |   if Textra >Tprom * 1.2 and !pingPong(wtp) and
   |   |    !maxHandovers(wtp) then
15 |   |   |   posibles_handovers.push(wtp)
16 |   end
17 |   if largo(posibles_handovers) >0 then
18 |   |   wtp_opt ← getMaxTasaExtra(posibles_handovers)
19 |   |   handover(lvap, wtp_opt)
20 end
```

Si con ninguna de las dos opciones, ya sea filtrando WTPs por uso o por tasa de transferencia pasada, se encontró un WTP candidato para realizar un *handover*, entonces se analiza la alternativa, que sería la modificación del *quantum* de la *slice* en la que se encuentra el LVAP para que consuma más *airtime* e intentar llegar a la tasa de transferencia prometida. Por otro lado, si en el WTP del LVAP, existe al menos un LVAP el cual está teniendo una tasa de transferencia mayor a la prometida, entonces se decrementa el *quantum* de esa *slice* en ese WTP. En este punto se utilizaron cuatro parámetros nuevos en el algoritmo. El porcentaje del *quantum* que se incrementa/decrementa y el máximo/mínimo de *quantum* posible en un WTP. En la versión final del algoritmo se utilizó tanto para el incremento como decremento del *quantum*, el 10 % de su *quantum* actual, como máximo *quantum* posible 16500 y como mínimo *quantum* posible 5000. El pseudocódigo de la función "intentarQuantum"

esta representado en Algoritmo 4.

Algorithm 4: intentarQuantum(slice, lvap, Tprom, Tintent, Tactual)

```
1 quantum_actual ← quantumActual(slice, lvap.wtp) if quantum_actual
    <QUANTUM_MAXIMO then
2   aumentarQuantum(slice, wtp)
3   foreach cliente in lvap.wtp do
4     Textra ← getExtraRate(cliente)
5     slice_cliente ← getSlice(cliente)
6     if Textra >rateProm(slice_cliente) and quantumActual(slice_cliente,
        lvap.wtp) >QUANTUM_MINIMO then
7       decrementarQuantum(slice_cliente, lvap.wtp)
8   end
```

Si el *quantum* ya es máximo entonces no se toma acción. Esto se puede dar en varias situaciones, por ejemplo si se promete demasiada tasa, o si la topología de la red no es la mejor (se tienen WTPs lejos de los clientes que son desperdiciados) entre otras razones.

Por ultimo, llegado al caso en que se promete más de lo que se puede ofrecer con los recursos disponibles, no existe una solución viable. En este caso, el sistema detecta esta situación y podría generar una alerta al administrador para que se tomen acciones.

En este capítulo se han visto los diferentes métodos que componen la heurística propuesta en este trabajo. En el siguiente capitulo (Capítulo 5.2) se abordarán los detalles de arquitectura e implementación relevantes.

5.2. Arquitectura e implementación de la solución

En este capítulo se explica en primera instancia la arquitectura de la solución propuesta para abordar el problema mediante la heurística vista en la Sección 5.1 y luego algunos detalles de su implementación.

Previo a la descripción de la arquitectura se destaca la decisión de implementar dos aplicaciones para modularizar lo que hubiera sido en un principio una sola aplicación. Por un lado se implementó la aplicación *WifiNetworkStats* que se encarga de obtener estadísticas sobre la red y guarda los resultados en una base de datos para que luego puedan ser consumidos por distintas aplicaciones. Esto permite que al realizar otro algoritmo para la toma de decisiones y acciones sobre la red, no sea necesario implementar la parte de obtención de datos nuevamente. Basta con levantar una instancia de esta aplicación y luego consumir los datos de la base de

datos.

La otra aplicación *WifiNetworkManager* es la que implementa la heurística y por lo tanto la encargada de consumir las estadísticas provistas por la aplicación *WifiNetworkStats* y tomar acciones sobre la red.

5.2.1. Arquitectura

Las aplicaciones fueron implementadas directamente sobre el controlador que ya provee una capa para implementar aplicaciones, que mediante las APIs *python* pueden consumir los manejadores del *Runtime*, como por ejemplo el manejador de la base de datos, manejador de los LVAPs o *slice*, etc. Esto permite que las aplicaciones puedan consultar el estado de la red usando los paquetes del protocolo EmPOWER y además modificarla, por ejemplo realizando *handovers*. Como manejador de base de datos se utilizó *Influx DB*, en particular se usó la clase *InfluxDBClient* de la librería *influxdb* de *python*.

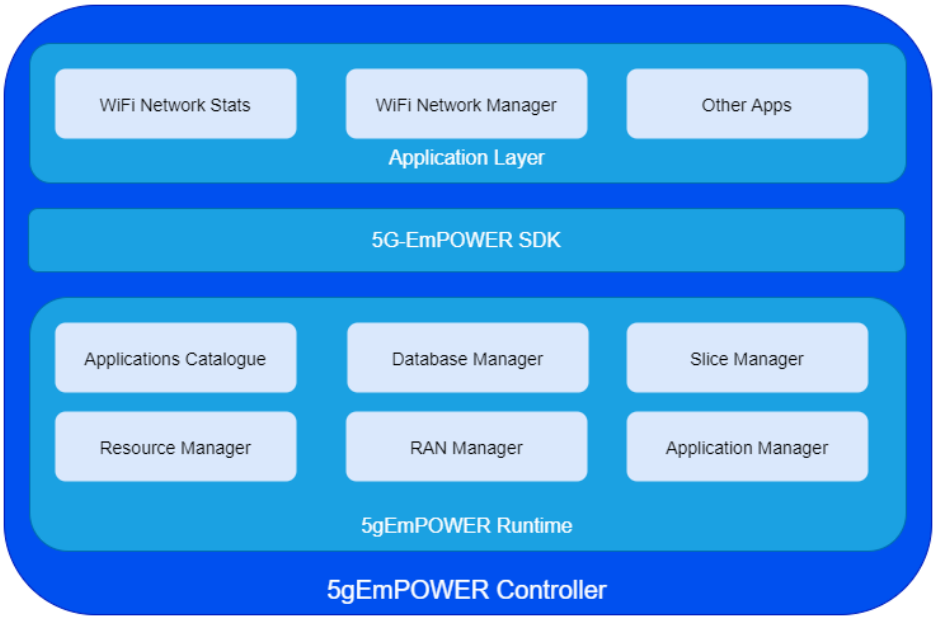

Figura 5.1: Arquitectura de la solución.

Bajo el *framework* 5G-Empower, para desarrollar una aplicación que utilice y modifique datos de la red se tiene que implementar una nueva clase con ciertas características. Para empezar se crea un directorio en el proyecto del controlador, más específicamente en la ruta "empower/apps". Dentro de este directorio se tiene el archivo que contiene la aplicación.

La implementación en sí de una aplicación consiste en crear una clase con el nombre que le se le desee dar, la cual debe ser una clase heredada de "EWifiApp" (ya mencionado en el capítulo Capítulo 3.3.4). Cuando se crea una instancia de

la aplicación para un proyecto, se llama a la función “launch”, la cual debe estar presente en la clase y cuya única instrucción es retornar una nueva instancia de la misma. Tanto esta función, como el creador de la clase reciben como parámetros el ID del servicio generado y el contexto en el que se encuentra. El atributo *context* también se explicó en la sección Capítulo 3.3.4.

Al crear una instancia de la aplicación, se le pasan los parámetros de ejecución que en este caso solo se utiliza un parámetro “every”, que es la frecuencia con la que se desea que se ejecute. Cuando es el momento de ejecutar la aplicación, se invoca otra función que debe estar presente, la función “loop”. El parámetro *every* está representado en milisegundos, entonces si levantamos una instancia de la aplicación con el parámetro en 2000 implica que se ejecutará la función *loop* de la aplicación cada 2 segundos.

Al implementar las aplicaciones de tipo *EWiFiApp*, la aplicación hereda varios métodos y atributos que permiten su funcionamiento, por ejemplo la clase *EApp* tiene los métodos *stop()* y *start()* para levantar o borrar instancias de la aplicación. Por otro lado de la clase *EService* la aplicación hereda los métodos para consultar y escribir en la base de datos entre otros, y es la clase que guarda los parámetros, el identificador y el contexto de la aplicación, que contiene todos los datos del proyecto en el que se instanció la aplicación.

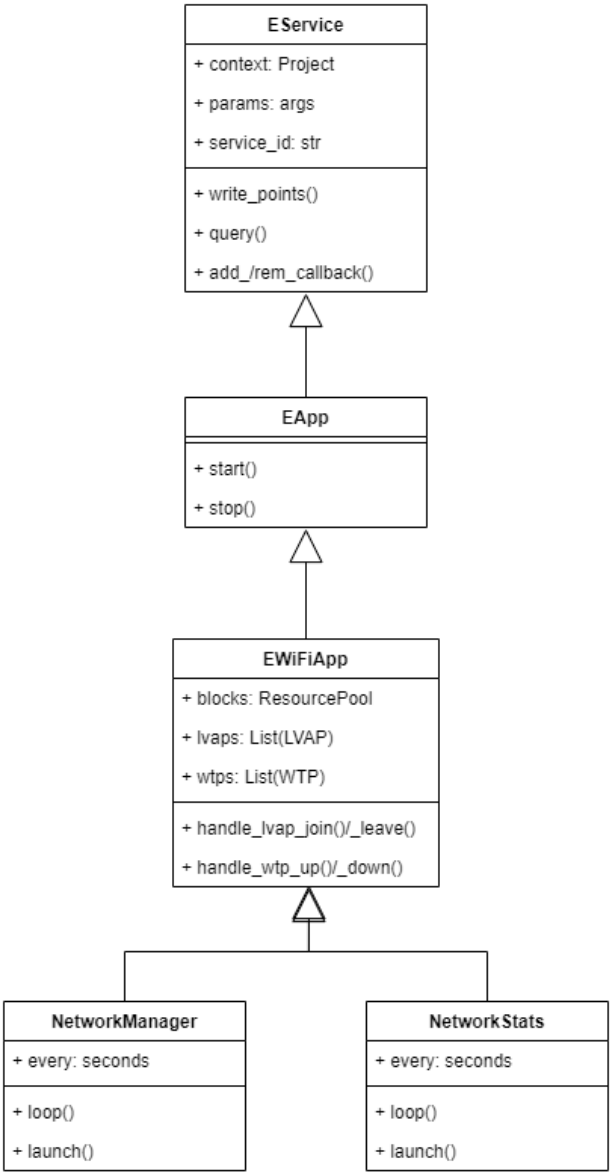

Figura 5.2: Diagrama de clases de las aplicaciones.

Las aplicaciones son expuestas por el controlador y luego mediante línea de comandos o directamente en el *webui* se pueden generar/borrar instancias de las aplicaciones asociadas a un proyecto. En este caso solo se necesita una instancia de la aplicación ejecutando, pero para otros casos se puede tener más de una instancia de la misma aplicación con distintos parámetros ejecutando al mismo tiempo.

5.2.2. Implementación

En esta sección se mencionan algunos aspectos relevantes respecto a la implementación de cada aplicación.

5.2.2.1. Aplicación WifiNetworkStats

Esta aplicación consiste en obtener toda la información de la red que sea necesaria y luego ponerla a disposición de la aplicación de manejo.

Los tipos de información utilizada de los agentes fue la siguiente:

- *WIFI Rate Control Stats* - Nos proporciona información sobre la tasa de transferencia actual de un LVAP así como los índices MCS utilizados.

  Formato de respuesta:

```
{
    "rate": "Rate en unidades de 500kbps o usando el índice MCS",
    "prob": "Probabilidad [0-18000]" ,
    "cur_prob": "Probabilidad actual [0-18000]" ,
    "cur_tp": "Bitrate estimado",
    "last_attempts": "Últimos intentos" ,
    "last_successes": "Últimos intentos con éxito" ,
    "hist_attempts": "Total de intentos",
    "hist_successes": "Total de intentos con éxito"
}
```

- *Packet Bin Counter* - Informa sobre la cantidad y el tamaño de los paquetes enviados por una LVAP. En base a estos números se infiere la tasa que se le intenta transmitir a un LVAP.

  Formato de respuesta: Consta de tres atributos, un *array* llamado *stats* y dos atributos llamados *nb_tx* y *nb_rx*, que representan la cantidad de objetos del arreglo que forman parte de la transmisión y recepción respectivamente. Ambos son números enteros cuya suma indica el largo del arreglo. Cada objeto del arreglo contiene un tamaño y una cantidad, lo cual indica el tamaño de la trama y cuantas veces fue enviada. Por lo tanto, si se suman todas las cantidades del arreglo hasta la posición *nb_tx*, obtendremos la cantidad de paquetes transmitidos por el LVAP.

```
{
    "nb_tx": 2,
    "nb_rx": 1,
    "stats": [
        {
            "size": 1386,
            "count": 6
        }
        {
            "size": 1450,
            "count": 209056
        }
        {
            "size": 66,
```

```
                "count": 100349
            }
        ]
}
```

- *Channel Quality Map* - Aquí se recibe, entre otros datos, el RSSI del bloque indicado con todos los dispositivos que encuentra.

  Formato de respuesta:

```
{
    "addr": "Dirección MAC del lvap o ap, dependiendo si es UCQM o NCQM",
    "last_rssi_std": "RSSI estandar durante la ultima ventana en dBm",
    "last_rssi_avg": "RSSI promedio durante la ultima ventana en dBm",
    "last_packets": "Cantidad de tramas en la última ventana",
    "hist_packets": "Número total de tramas",
    "mov_rssi": "RSSI en movimiento en dBm"
}
```

- *WIFI Channel Stats* - Lo obtenido aquí indica el porcentaje de uso de los recursos del *Resource Block* que se haya indicado.

  Formato de respuesta (devuelve los últimos 100 datos de este tipo):

```
{
    "timestamp": "Timestamp del agente",
    "tx": "Porcentaje de uso en transmisión",
    "rx": "Porcentaje de uso en recepción",
    "ed": "Porcentaje de uso en detección de energía"
}
```

Al inicio de la aplicación es necesario definir todos los paquetes que se utilizarán para la comunicación con los agentes. Luego de declarados los paquetes e inicializadas las variables a utilizar, al ejecutar la aplicación se envían todos los paquetes necesarios a los agentes pidiendo información. También se les asigna una función de *callback* a cada uno de los llamados para que procese la respuesta en el momento que llegue.

El objetivo de las funciones de *callback* es procesar las respuestas para luego disponer los valores en la base de datos, ingresando cada grupo de estadísticas en su propia tabla para su posterior uso por la aplicación de manejo de red. Por ejemplo, el *callback* de la respuesta de *Packet Bin Counter*, debe recorrer la respuesta (la estructura de este mensaje ya fue explicada anteriormente) y calcular el total de bytes y paquetes que se le intentaron transmitir al LVAP. Además, como la respuesta anterior está almacenada, se puede calcular la tasa (tanto de bytes como de paquetes) que se intentó transmitir en el intervalo de una respuesta a otra. Finalmente, se guardan estas estadísticas en una tabla de la base de datos.

5.2.2.2. Aplicación WifiNetworkManager

Esta aplicación consume las estadísticas obtenidas por la aplicación *WifiNetworkStats*, las analiza y toma decisiones al respecto.

En el planteamiento de la heurística, se parte de que se tienen tablas con información sobre las relaciones *Slices - LVAPs* y *Slices - Tasas prometidas*. Para almacenar estos datos se implementó un *script* en *Python*, que guarda estos datos en la base de datos *InfluxDB* para luego consultar esa tabla desde la aplicación. Por tanto este *script* se debe ejecutar antes de iniciar la aplicación.

A lo largo del desarrollo de la aplicación se trató de minimizar la cantidad de consultas a la base de datos, por lo que al iniciar la aplicación se consultó una sola vez las tablas con los datos de tasas prometidas y *slices* y se almacenaron los resultados en memoria local. Al inicio de la aplicación también se guardaron en memoria local las variables de la clase inicializadas. Los parámetros del algoritmo como por ejemplo, la cantidad máxima de *handovers* permitidos al mismo WTP, en la misma iteración, el RSSI mínimo aceptable para realizar *handover*, etc son definidos como constantes.

Como fue indicado en la Sección 5.1.3, para desarrollar la heurística se implementaron varias funciones auxiliares, entre ellas:

- *rateProm(slice)*, que devuelve la tasa prometida a *slice*.
- *rateActual(lvap)*, que devuelve la tasa de transferencia actual de *lvap*, usando las estadísticas *WiFi Rate Control*.
- *rateIntentado(lvap)*, que devuelve la tasa que se intentó transmitir al LVAP, usando las estadísticas *WiFi Rate Control*.
- *wtpUsage(wtp)*, que devuelve el porcentaje de uso dedicado a transmisión en WTP, usando las estadísticas *WIFI Channel Stats*.
- *pingPong(wtp, lvap)*, mirando el historial de *handovers* del LVAP la función retorna *true*, si al agregar el WTP a la lista de *handovers* se generan secuencias consecutivas en el historial. Esta lista solo toma los últimos 100 segundos.
- *maxHandovers(wtp)*, retorna si ya se realizaron la cantidad máxima de *handovers* a WTP en esta iteración.
- *getMinUsage(posibles_handovers)*, devuelve el WTP, con menos uso dentro de *posibles_handovers*
- *getExtraRate(wtp/lvap)*, devuelve $Textra$ de WTP o LVAP.

- *getMaxTasaExtra(posibles_handovers)*, devuelve el WTP con mayor $Textra$ dentro de *posibles_handovers*
- *handover(lvap, block)*, realiza una reasignación de LVAP a *block*.
- *quantumActual(slice, wtp)*, retorna el valor de *quantum* de *slice* en WTP.

Para realizar los cambios de valor de *quantums* se utilizó la primitiva *upsert_wifi_slice* a la cual se le pasa un objeto *Slice* el cual actualiza la *slice* con los parámetros que se le establecieron. El objeto *Slice* tiene las siguientes propiedades:

```
{
    "slice_id": "Slice ID",
    "properties": {
        "amsdu_aggregation": "False",
        "quantum": "quantum",
        "sta_scheduler": "Round-Robin"
    },
    "devices": ["devices"]
}
```

Donde en *devices* se especifica un objeto similar con las propiedades particulares para un WTP especifico.

Por último, notar que idealmente la aplicación *WiFiNetworkStats* debe ejecutarse con mayor frecuencia que la aplicación *WifiNetworkManager*.

En ambas aplicaciones se agregaron *logs* que escriben periódicamente. Estos *logs* se utilizarán luego para analizar el resultado de las pruebas, debido a que en los mismos se guardan el valor de los *quantums* y las acciones que toma el algoritmo entre otros datos.

El código del controlador modificado, incluyendo el código de ambas aplicaciones, se puede descargar del repositorio *git*: (*Repositorio git del proyecto* s.f.).

En la siguiente sección se harán distintas ejecuciones, comparaciones y análisis de los resultados sobre la heurística desarrollada.

5.3. Evaluación

El objetivo de esta sección es presentar la evaluación de la solución propuesta.

Para esta etapa del proyecto se decidió separar la evaluación en dos partes. En primer lugar se quiso verificar las decisiones individuales que toma el algoritmo, corriéndolo en un período breve sobre un escenario específicamente diseñado para evaluar cada decisión.

Por otra parte, también se evaluó el algoritmo a gran escala, creando escenarios con múltiples clientes y flujos cambiantes, con la finalidad de evaluar una posible mejora de servicio en un tiempo prolongado.

Ambas instancias de evaluación cumplen ciertas condiciones base:

- La aplicación de recolección de estadísticas se ejecuta cada 2 segundos.
- Todos los clientes de la prueba que se ejecute comenzarán conectados a un mismo WTP.
- El número máximo de traslados de LVAPs hacia un mismo WTP en una sola iteración del algoritmo es de uno.
- El nivel mínimo necesario de RSSI para hacer un traslado es de 170.

Dado que Empower utiliza el campo DSCP para identificar la *slice* a la que pertenece el tráfico, utilizaremos la funcionalidad provista por *Iperf* para establecer el cabezal DSCP.

Para realizar las pruebas se ejecutará el Controlador EmPOWER sobre una laptop con sistema operativo Ubuntu 18.04 LTS, donde corre el controlador dentro de un contenedor *Docker*. Para el armado de la red se tendrá un *switch*, un *Router* que funcione como servidor DHCP, un *Router* que se utilizará para generar trafico *iperf3*, dos WTP (Routers TP-Link TL-WDR4300, con sistema OpenWRT 19.07-SNAPSHOT, con el agente EmPOWER instalado) y cuatro clientes que son Routers TP-Link TL-WDR4300, con sistema OpenWRT 19.07.3. Los dispositivos que se usan como WTP tienen una interfaz 802.11n con una capacidad máxima de 20 Mbit/s aproximadamente, según lo apreciado bajo una red Empower. La versión de *iperf3* utilizada por los clientes es 3.7. Para algunas pruebas se utilizará solo una parte del despliegue, es decir menos clientes y/o menos WTPs.

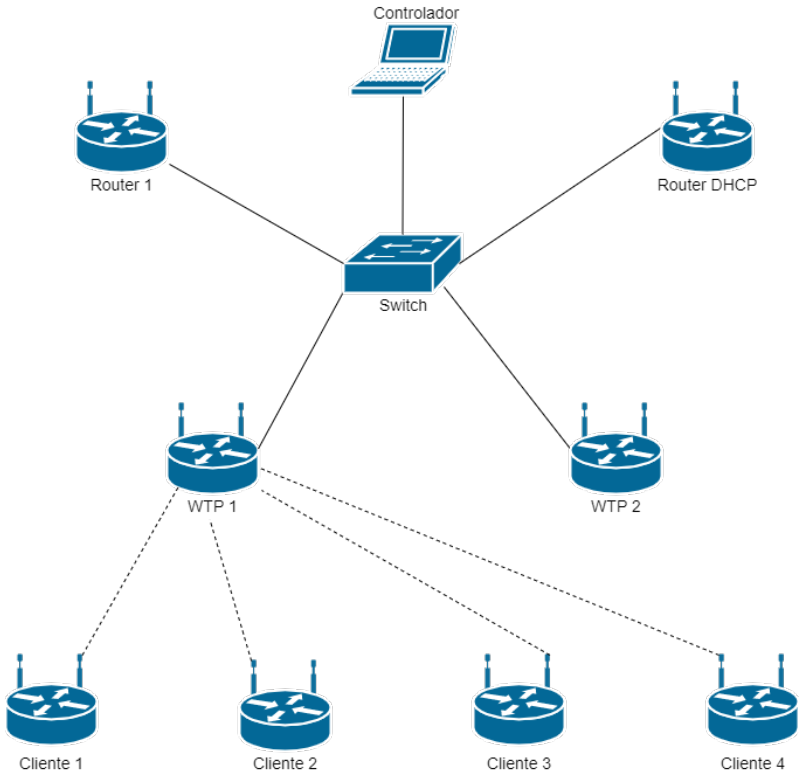

Figura 5.3: Despliegue de red de evaluación.

Sobre cada escenario se hará una ejecución con las aplicaciones desarrolladas en ejecución, y otra sin ninguna aplicación de control o utilizando algún otro algoritmo que tenga incluído el controlador. Para la presentación de los resultados se hará una comparación presentando los datos de ambas ejecuciones.

Para esta etapa de evaluación, se le realizaron algunos cambios a las aplicaciones, de manera tal que se guarden en un archivo todas las modificaciones sobre la red y el momento en que fueron realizadas. También se almacenará en un archivo el porcentaje de uso de cada bloque de cada WTP y los *quantum* de cada Slice. Estas últimas estadísticas se generaron en la aplicación de estadísticas por lo que se guardaron esos datos cada 2 segundos.

En ambas ejecuciones se guardará tanto los resultados finales como los resultados segundo a segundo de *Iperf*. Gracias a esto podemos comparar los resultados finales para ver las tasas de transmisión promedio y el promedio de paquetes perdidos para evaluar la eficacia y eficiencia del algoritmo. Al tener los datos en intervalos de un segundo de *Iperf*, se podrá graficar la tasa de transferencia y el porcentaje de paquetes perdidos a través del tiempo, y apreciar si en los momentos que el algoritmo tomó acciones mejoró la calidad de la red.

Se ejecutarán 3 escenarios base, que buscan probar como impactan las decisiones del algoritmo propuesto sobre una red mínima. Para evaluar el resultado se compararán las tasas de transmisión en el transcurso del tiempo con la misma ejecución

del escenario pero sin utilizar el algoritmo.

Luego se ejecutarán dos escenarios completos, donde el tráfico transmitido a los distintos clientes varía según el tiempo. En este sentido se pretende evaluar el uso del algoritmo propuesto en un escenario más próximo a la realidad y con mayor número de variables en juego. Igualmente los intervalos de tiempo están pensados para que el algoritmo se comporte de manera esperada. En este caso se recabarán más estadísticas que en los escenarios base. Los detalles de cada escenario se especificarán en su respectiva sección.

Para cada escenario se realizaron múltiples ejecuciones, donde los resultados siempre fueron similares. En este informe se eligió una instancia representativa.

5.3.1. Escenarios Base

Para todos los escenarios descritos a continuación se establecieron los siguientes valores de configuración:

- La aplicación de toma de decisiones se ejecuta cada 10 segundos
- Valor de *quantum* inicial para todo cliente en todo WTP es de 12000 ms
- Valor máximo asignable de *quantum* a un cliente en algún WTP es de 16500 ms
- Valor mínimo asignable de *quantum* a un cliente en algún WTP es de 8500 ms

5.3.1.1. Escenario Base 1

Presentación En este escenario se utilizarán dos clientes y un WTP. Cada cliente pertenece a una Slice distinta. Ambos clientes estarán a una distancia del WTP que no presente inconvenientes, de aproximadamente 0,5m entre sí. El cliente 1 tendrá una promesa de 5 Mbit/s, mientras que para el cliente 2 será de 15 Mbit/s.

En este escenario a ambos clientes se les intentará transmitir a 15 Mbit/s, luego de varias iteraciones el algoritmo debería incrementar el *quantum* de la Slice 2 al máximo y la del 1 al mínimo para darle más recursos al cliente que se le prometió más tasa de transferencia. El objetivo de este escenario es entonces comprobar el aumento y descenso de *quantums* por parte del algoritmo.

Resultados El comportamiento del algoritmo fue el esperado. Cada vez que se ejecutó el algoritmo, aumentó el *quantum* de la Slice 2 y decrementó el *quantum* de la Slice 1. Esto lo repitió hasta llegar al máximo *quantum* posible para la Slice 2 y el mínimo para la Slice 1. En este caso la Slice 2 finalizó con un *quantum* de 15972 ms y la Slice 1 finalizó con un *quantum* de 8748 ms. Esto implica que al final el cliente 2 tuvo que haber tenido casi el doble de tasa de transmisión que el cliente 1.

Como muestra la Figura 5.4 esto se cumple. A medida que pasan las iteraciones, el cliente 2 va teniendo más tasa de transferencia hasta llegar al máximo, donde duplica la tasa percibida por el cliente 1. Debido a esto, la tasa de transmisión del cliente 2 se aproxima al objetivo prometido, mientras se sigue cumpliendo con la tasa prometida al cliente 1.

Al ejecutarlo sin el algoritmo ambos clientes reciben constantemente alrededor de 10 Mbit/s cada uno, como se ve en Figura 5.5.

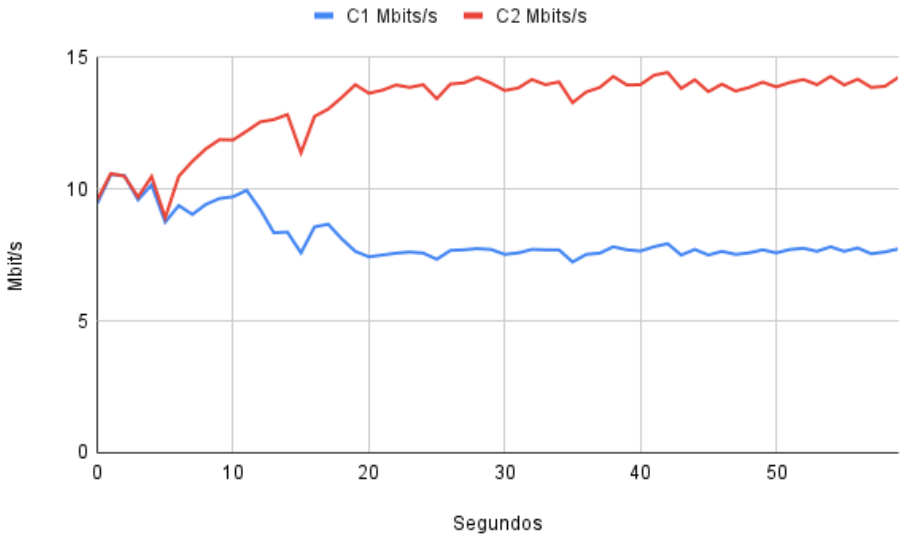

Figura 5.4: Resultados de Escenario Base 1 con algoritmo.

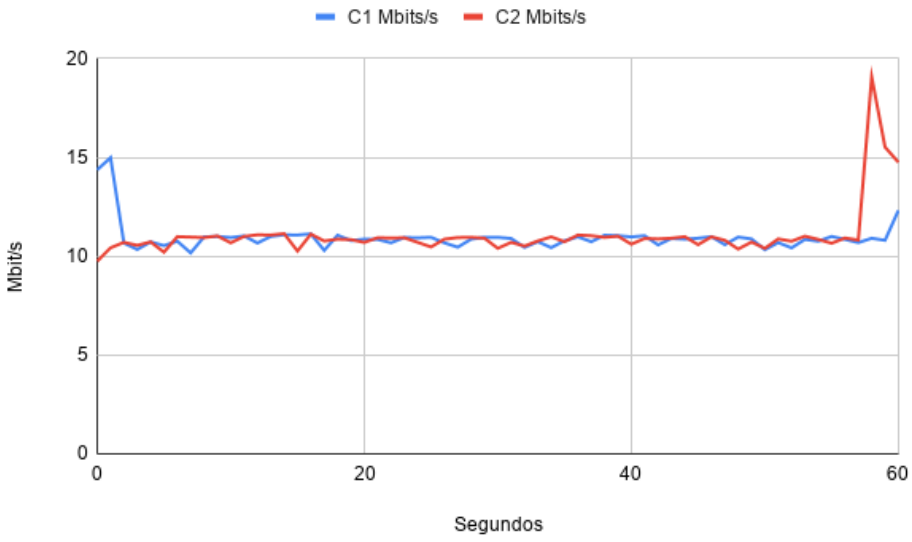

Figura 5.5: Resultados de Escenario Base 1 sin algoritmo.

5.3.1.2. Escenario Base 2

Presentación Para este escenario se utilizarán un cliente y dos WTP. Un WTP estará situado en las mismas condiciones de cercanía del Escenario Base 1 con respecto al cliente, mientras que el otro WTP estará situado a una distancia significativamente mayor. El cliente comenzará conectado al WTP lejano, y se le promete una tasa de transferencia de 10 Mbit/s.

En este escenario se le transmitirá al cliente 10 Mbit/s. En el escenario se comprobará la habilidad del algoritmo de realizar un traslado entre los WTP, para así asignar el LVAP a un mejor WTP y que pueda recibir la tasa de transmisión prometida.

Resultados En este escenario, al estar el cliente asignado a un WTP lejano y tener una señal de baja calidad, no pudo cumplir con la tasa de transmisión prometida. Por lo tanto, en la primera iteración, el algoritmo decidió realizar un handover, asignando el cliente a un WTP con mejor señal. Luego de realizar el handover la tasa de transferencia del cliente aumenta, habiendo alcanzar el valor prometido como se ve en Figura 5.6.

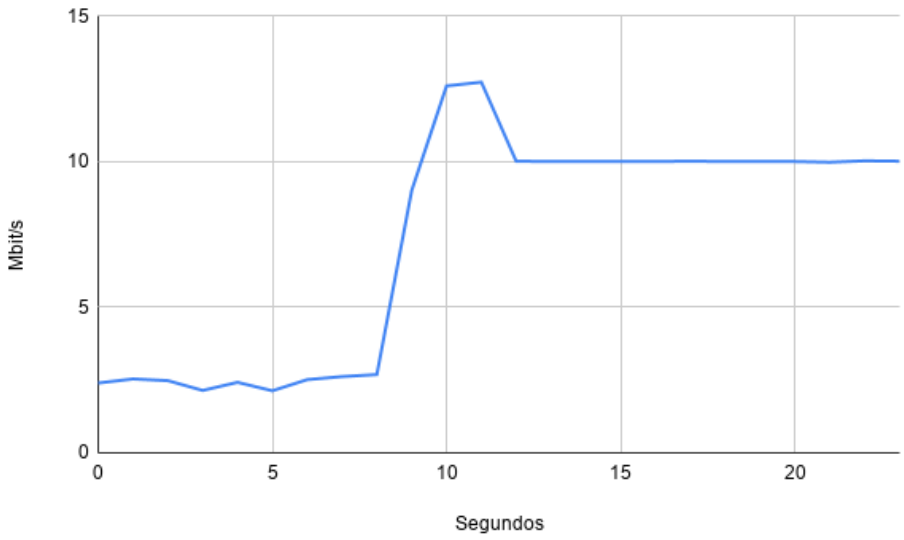

Figura 5.6: Resultados de Escenario Base 2 con algoritmo.

5.3.1.3. Escenario Base 3

Presentación En este escenario se usarán dos clientes y dos WTP. Cada cliente pertenece a una Slice distinta, y comenzarán conectados al mismo WTP. Ambos clientes estarán a una distancia del WTP que no presente inconvenientes y tienen la misma tasa de transferencia prometida de 15 Mbit/s.

En este escenario ambos clientes tratarán de transmitir a 15 Mbit/s. El algoritmo deberá realizar el traslado para uno de los clientes, y de esta forma quedará asignando únicamente un cliente a cada WTP. En esa nueva configuración se espera que ambos clientes obtengan su tasa de transmisión prometida.

Resultados Este escenario es similar al anterior de la Sección 5.3.1.3, excepto que en el caso actual la señal entre WTP y clientes es buena en ambos casos. El problema en este escenario es que ambos clientes están asignados a un mismo WTP y cuando se les transmite 15 Mbit/s a cada uno se satura el WTP, lo que impide cumplir con las promesas. En la primera iteración el algoritmo traslada uno de los clientes a un WTP ocioso, en este caso al otro WTP que no estaba siendo utilizado. Luego del traslado ambos clientes reciben el tráfico prometido como se puede apreciar en la Figura 5.7.

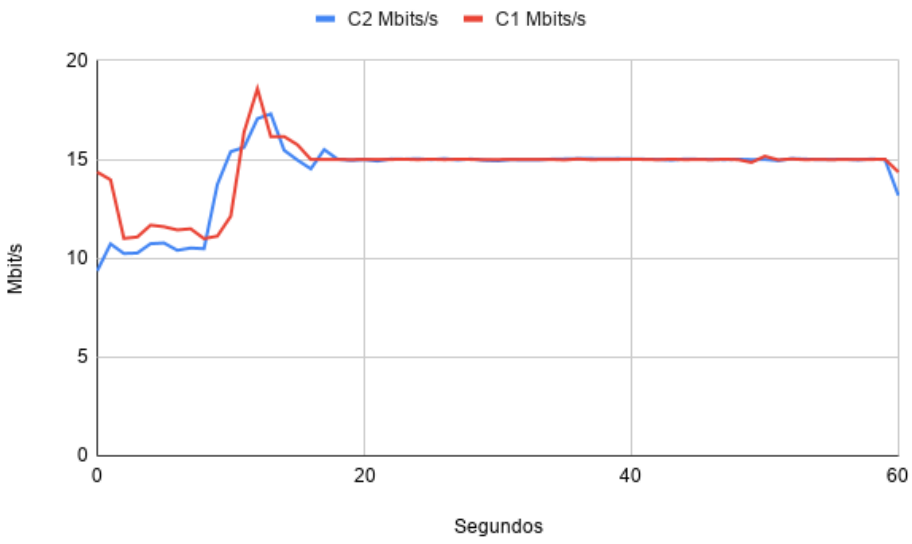

Figura 5.7: Resultados de Escenario Base 3 con algoritmo.

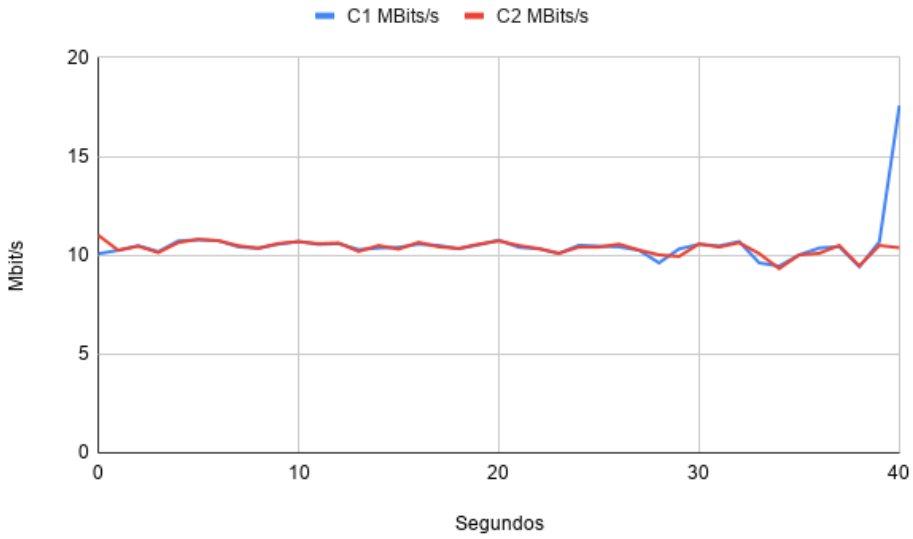

Figura 5.8: Resultados de Escenario Base 3 sin algoritmo.

5.3.2. Escenarios Completos

A diferencia de los escenarios presentados en la Sección 5.3.1, en estos casos se buscará analizar el algoritmo presentado a una escala más abarcativa y realista. Por esta razón en la ejecución de los escenarios más completos se modificaron algunos de los parámetros configurables para permitir al algoritmo más de margen, y que con ello se logre estabilizar antes de tomar una decisión nuevamente:

- La aplicación de toma de decisiones se ejecuta cada 6 segundos
- Valor de quantum inicial para todo cliente en todo WTP es de 10000 ms
- Valor máximo asignable de quantum a un cliente en algún WTP es de 16500 ms
- Valor mínimo asignable de quantum a un cliente en algún WTP es de 5000 ms

Además, los clientes siempre comenzarán en el mismo WTP, que será en ambos casos el llamado WTP 1.

Por otra parte, la configuración espacial de la red para este grupo de escenarios se mantiene siempre igual, y la idea es que no haya conflictos por grandes distancias o interferencia. Se intentó mantener la ubicación de los componentes lo más próximo a lo que se puede ver en la Figura 5.9.

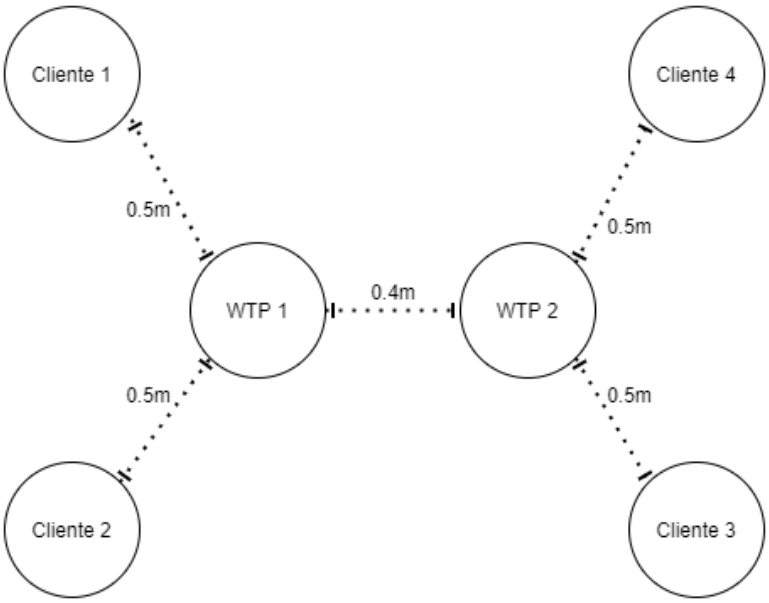

Figura 5.9: Configuración aproximada para escenarios completos.

A modo de comparación no se utilizará una ejecución del escenario sin ningún algoritmo corriendo, sino que se utilizará una aplicación de movilidad por RSSI, la cual viene incluída en el controlador 5G-EmPower. La idea general de dicha aplicación es realizar traslados cuando se detecta que para un LVAP existe un WTP con mejor señal.

5.3.2.1. Escenario Completo 1

Presentación En este primer escenario se contará con cuatro clientes:

- **Cliente 1**: Pertenece a Slice 1 y tiene una promesa de 2 Mbit/s
- **Cliente 2**: Pertenece a Slice 2 y tiene una promesa de 5 Mbit/s
- **Cliente 3**: Pertenece a Slice 3 y tiene una promesa de 10 Mbit/s
- **Cliente 4**: Pertenece a Slice 4 y tiene una promesa de 15 Mbit/s

Se definen además 6 intervalos de tiempo, cada uno durará 100 segundos, donde se variará la tasa ofrecida a cada uno de los clientes. En la Tabla 5.1 se muestran los detalles de cuánto se intenta transmitir a cada cliente en cada intervalo.

Previo a la ejecución de los casos, se razona un comportamiento esperado:

- **Intervalo 1**: El cliente 4 no podrá cumplir con su tasa de transferencia prometida. El algoritmo buscará un nuevo WTP destino y moverá este cliente a otro WTP (WTP 2).

Intervalo	Tiempo	Rate C1	Rate C2	Rate C3	Rate C4
1	0	15	0	0	15
2	100	15	10	10	15
3	200	15	5	5	5
4	300	10	10	10	15
5	400	2	5	10	15
6	500	10	10	10	10
7	600	0	0	0	0

Tabla 5.1: Tasa ofrecida en Mbit/s a cada cliente en Escenario Completo 1.

- **Intervalo 2**: Al comenzar este intervalo, los clientes 1, 2 y 3 están asignados al WTP 1 mientras que el cliente 4 está en el WTP 2. El cliente 4 llega a su tasa de transmisión prometida saturando el router por lo que no realizará acciones. Los clientes 2 y 3 no llegarán a su tasa de transferencia prometida en un principio pero no serán trasladados ya que el WTP 2 está saturado y no tiene capacidad sobrante. Por lo tanto a medida que pasen las iteraciones se decrementará el *quantum* del cliente 1 hasta llegar al mínimo, el *quantum* del cliente 2 oscilará sobre un punto medio, mientras que el cliente 3 aumentará su *quantum* hasta el máximo permitido.

- **Intervalo 3**: En el intervalo 3 el WTP 1 va a estar saturado mientras que el WTP 2 va a tener menos uso por lo que en la primera iteración el cliente 2 debería ser trasladado al WTP 2. Luego quedaría el WTP1 saturado pero todos los clientes llegarán a su tasa de transferencia prometida. El cliente 1 recibirá al final del intervalo alrededor de 10 MBit/s.

- **Intervalo 4**: Se debe notar que los dos WTP estarán saturados. En la primera iteración el algoritmo debería mover el cliente 2 al WTP1. Una vez realizado el *handover*, se repite el escenario del intervalo 2. Finalizando los clientes 1, 2 y 4 con tasas iguales a las prometidas y el cliente 3 con una tasa de transmisión un poco por debajo de lo prometido.

- **Intervalo 5**: En este intervalo no deberían haber cambios, todos los clientes deberían poder recibir sus tasas prometidas.

- **Intervalo 6**: En este último intervalo, el algoritmo debería mover el cliente 2 al WTP 2 ya que tiene menos porcentaje de uso que su WTP actual. Luego de varias iteraciones se ajustarán los *quantums* en el WTP2 hasta que el cliente 4 reciba 10 MBit/s y el cliente 2 reciba 5 MBits/s. En el WTP1 los *quantums* ya están ajustados por lo que una vez que se realice el *handover* del cliente 2, el cliente 3 debería recibir 10 MBit/s mientras que el cliente 1 debería recibir 5 MBit/s.

Resultados Los resultados de las ejecuciones muestran que en general el algoritmo propuesto identificó cuáles de los clientes debieron recibir cierta prioridad sobre otros. Siendo los clientes 1 y 2 los que menos tasa de transferencia tienen prometida, no advierten un beneficio del algoritmo, sino que en cambio obtienen tasas peores en promedio que con el algoritmo de asignación por RSSI. Si bien en la Figura 5.10 se ve un intercambio de las mayores tasas a lo largo de los intervalos para ambos algoritmos, en la Figura 5.11 se aprecia claramente un deterioro del servicio con el uso del algoritmo presentado. De todos modos es importante destacar que en las ejecuciones con el algoritmo corriendo, salvo breves picos de inestabilidad (como en el segundo 330 de la Figura 5.10), ambos clientes recibieron una tasa de transmisión mayor o igual a la que tenían prometida.

Analizando los resultados desde el punto de vista de los clientes con mayor promesa de recursos, el algoritmo les brindó en general un mejor servicio. Para los clientes 3 y 4 el algoritmo de asignación por RSSI no les otorga la tasa de transferencia prometida, a menos que el cliente intente transmitir muy por debajo de ese valor, tal como se puede ver en el intervalo 220-330 de las Figuras 5.12 y 5.13. De igual modo, se evidencia que utilizando el algoritmo propuesto no se logró llegar en todo momento a la tasa de transmisión prometida a estos clientes. Esto se puede apreciar en el intervalo 110-220 de la Figura 5.12 y también en los intervalos 550-660 de ambas Figuras.

Otra apreciación inferida de los resultados es que existen instancias en las que los clientes pierden conexión cuando se realiza un *handover*. Observando el instante 330 de la Figura 5.10 o el instante 350 de la Figura 5.13 se visualiza que los clientes en cuestión no recibieron datos, aunque ese fallo de conexión se resuelve sin necesidad de intervención en los momentos inmediatos subsiguientes.

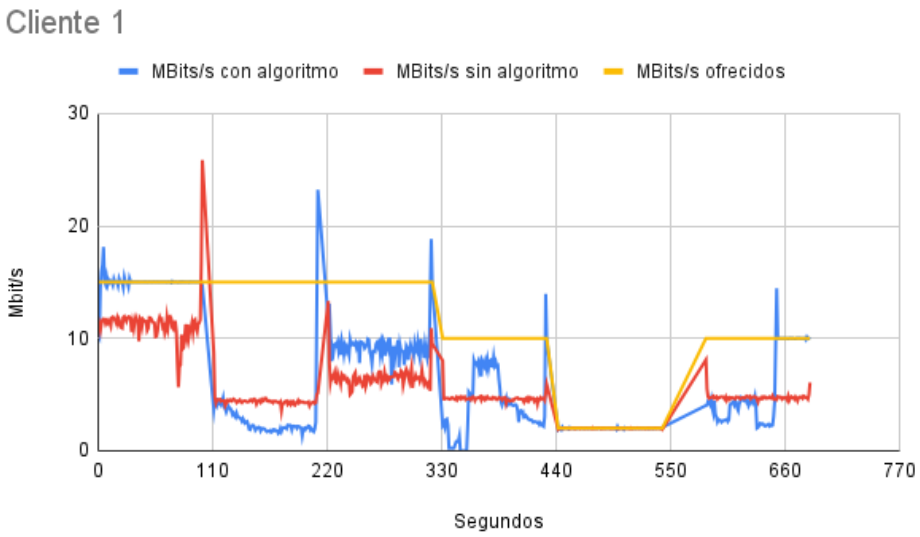

Figura 5.10: Escenario completo 1: Mbit/s recibido por cliente 1. Tasa prometida: 2 Mbit/s

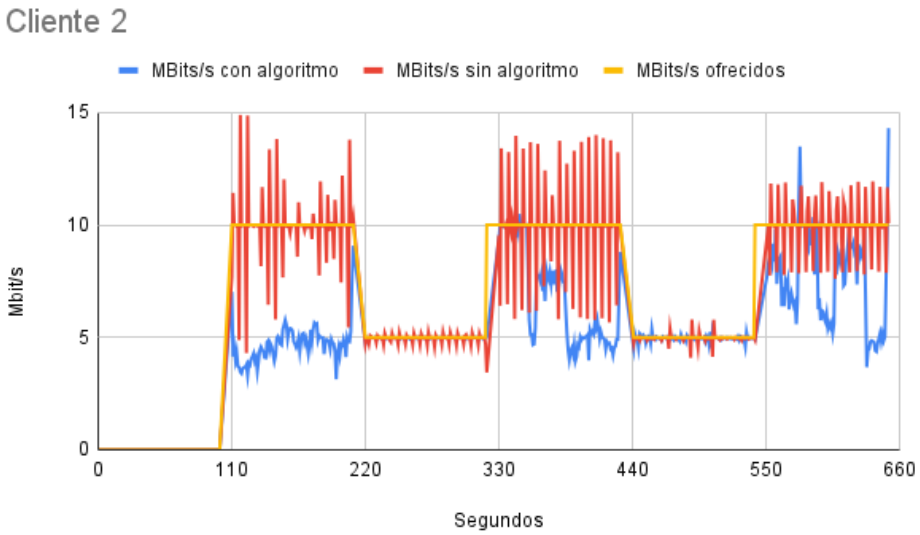

Figura 5.11: Escenario completo 1: Mbit/s recibido por cliente 2. Tasa prometida: 5 Mbit/s

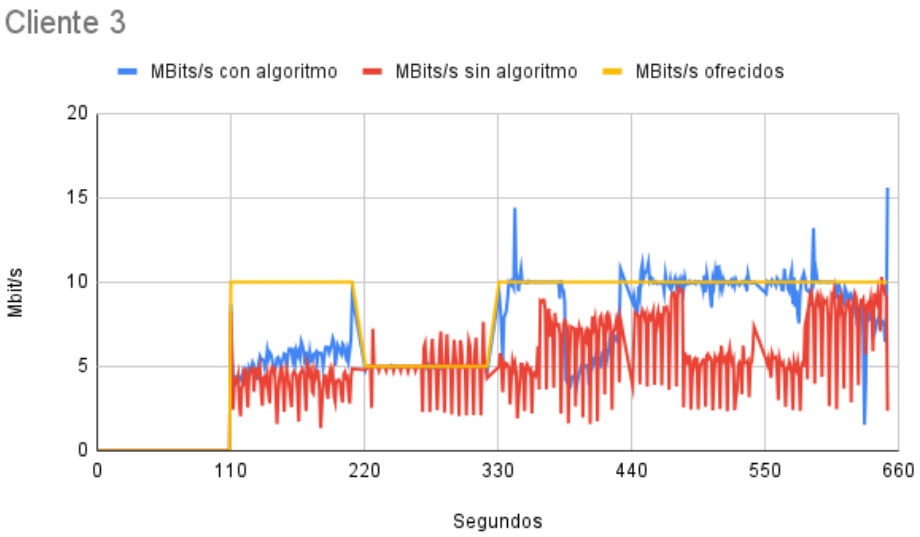

Figura 5.12: Escenario completo 1: Mbit/s recibido por cliente 3. Tasa prometida: 10 Mbit/s

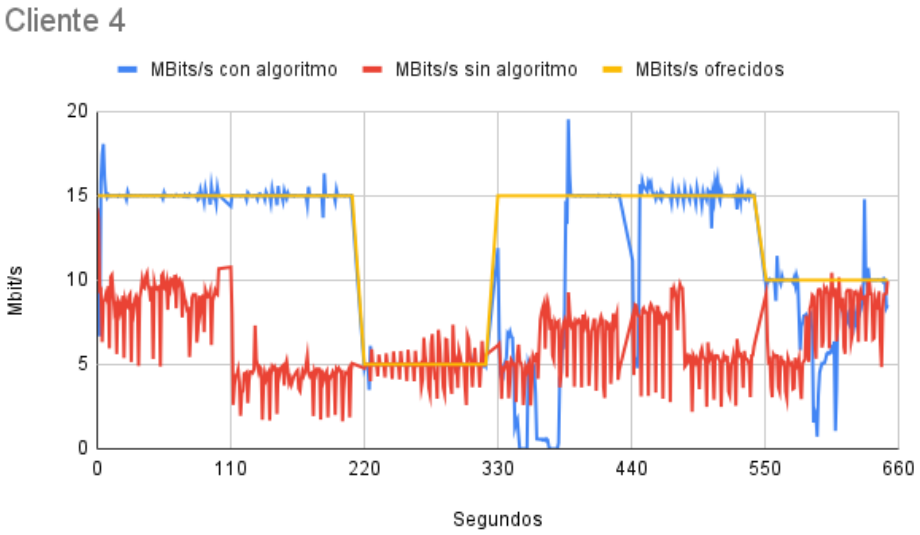

Figura 5.13: Escenario completo 1: Mbit/s recibido por cliente 4. Tasa prometida: 15 Mbit/s

Examinando ahora los *quantums* y las asignaciones de cada cliente a los WTPs se puede llegar a una conclusión similar a la anterior, y es que el algoritmo enunciado detectó un orden de prioridad entre los clientes. Cuando existe competencia por los recursos, el algoritmo le asigna más *airtime* a aquellos clientes que tienen mayores promesas. Un ejemplo de esto ocurre en el intervalo 110-220, donde la Figura 5.16 muestra que los clientes 1, 2 y 3 están en el WTP 1. En la Figura 5.14 se ve como se le asigna un *quantum* mayor a los clientes 2 y 3, mientras que el cliente 1 sufre una disminución del suyo.

Otro aspecto a destacar es que en general el algoritmo maneja las asignaciones de clientes a WTPs de modo razonablemente aceptable, intentando dejar un WTP con el cliente 4 y el otro con los restantes clientes. Esto tiene sentido debido a que la suma de las promesas de ellos (17 Mbit/s) es similar a la del cliente 4 individualmente considerado (15 Mbit/s).

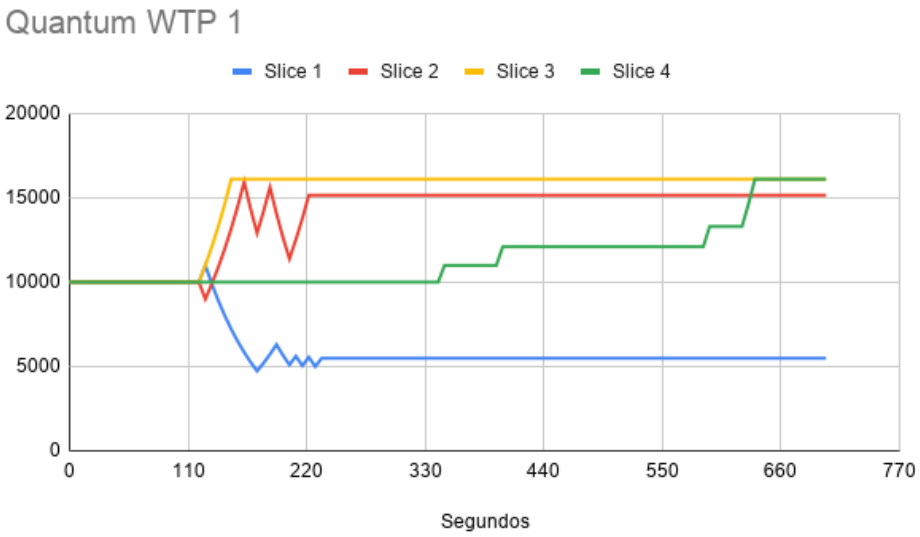

Figura 5.14: Escenario completo 1: Asignación de *quantums* en WTP 1.

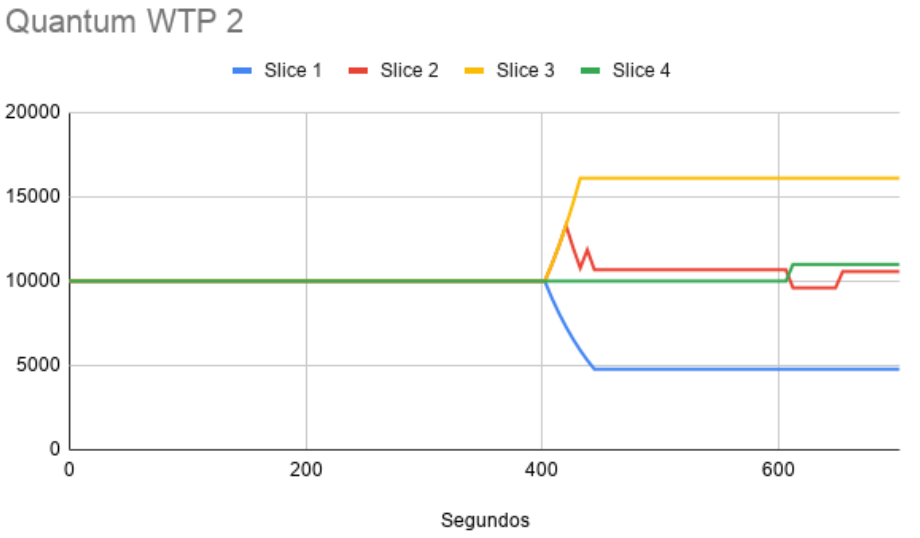

Figura 5.15: Escenario completo 1: Asignación de *quantums* en WTP 2.

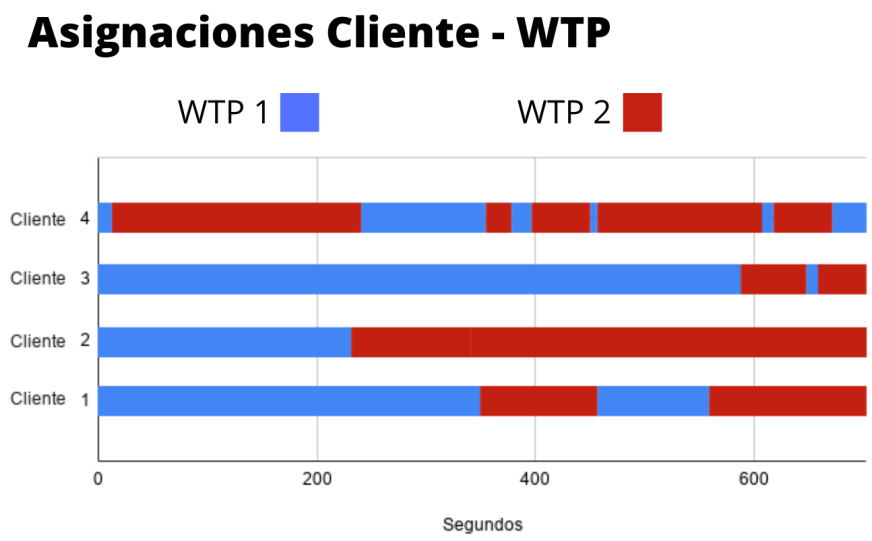

Figura 5.16: Escenario completo 1: Asignación de clientes a WTPs.

Como un indicador de qué tan beneficioso fue para los clientes el uso del algoritmo, se formuló el siguiente indicador:

$$Eval_{clientN} = avg(T^p_{lograda} - min(T^p_{prom}, T^p_{int})), \forall p \in P \tag{5.3}$$

Se puede medir para cada cliente este indicador teniendo las tasas logradas ($T^p_{lograda}$), prometidas (T^p_{prom}) e intentadas (T^p_{int}) para todo período p, de 1 segundo de duración, perteneciente a toda la ejecución.

Los índices de satisfacción para este escenario se pueden ver en la Tabla 5.2. Se aprecia allí que utilizando el algoritmo propuesto la mayoría de los clientes tiene un valor mayor en comparación. Cabe resaltar que cuanto mayor es la promesa, mayor fue la magnitud de la diferencia positiva.

A su vez, la columna de ganancia indica el aumento porcentual percibido por cada cliente. Siendo I_{CA} el índice de satisfacción con algoritmo y I_{SA} el mismo índice pero sin algoritmo, el nuevo valor se calculó de la siguiente manera:

$$\%incremento = \frac{(I_{CA} - I_{SA})}{|I_{SA}|} * 100 \tag{5.4}$$

En este escenario, los clientes 3 y 4 son los que en teoría deberían apreciar una ganancia positiva, y es efectivamente lo ocurrido, recibiendo ambos más de un 50 % de ganancia. En cambio, el cliente 2 es quien se ve más afectado por el uso del algoritmo, ya que su tasa de transmisión es la que se redistribuyó entre aquellos que más la precisaban. Sin embargo, con el algoritmo el índice se mantiene mayor a 0, lo que indica que en promedio se está cumpliendo con la promesa solicitada. Por su parte el cliente 1 no se vio significativamente afectado.

Cliente	Indice con algoritmo	Indice sin algoritmo	Ganancia
1	4.476	3.617	23.749 %
2	0.671	2.442	-72.523 %
3	-1.248	-2.824	55.807 %
4	-1.505	-6.253	75.932 %

Tabla 5.2: Índice de Satisfacción en Escenario Completo 1.

Intervalo	Tiempo	Rate C1	Rate C2	Rate C3	Rate C4
1	0	5	5	5	15
2	100	15	15	10	10
3	200	10	10	10	10
4	300	5	5	15	15
5	400	0	0	0	0

Tabla 5.3: Tasa ofrecida en Mbit/s a cada cliente en Escenario Completo 2.

5.3.2.2. Escenario Completo 2

Presentación El segundo escenario utilizará nuevamente cuatro clientes. Los clientes 1 y 2 pertencerán a la Slice 1 y se les promete 5 Mbit/s, mientras que los clientes 3 y 4 pertenecientes a la Slice 2 tienen una promesa de 15 Mbit/s.

En este caso existirán 4 intervalos de tiempo y la especificación de las tasas transmitidas se puede ver en la Tabla 5.3

El comportamiento esperado en este caso es el siguiente:

- **Intervalo 1**: Al comenzar con la transmisión de datos, el WTP 1 se saturará ya que no podrá soportar un flujo de 30 Mbits/s. Lo más probable es que todos los clientes ni siquiera lleguen a lo intentado, por lo que se realizará un traslado de alguno de ellos. El algoritmo va recorriendo los clientes en orden, por lo que el cliente 1 es el que tiene mayor probabilidad de ser transferido.

- **Intervalo 2**: Este intervalo comenzará con un cliente del Slice 1 en un solo WTP. En el otro WTP habrá demasiado tráfico, por lo que se hará otro *handover*, solo que esta vez será con algún cliente de la Slice 2, debido a que estos son los que no pueden cumplir con lo prometido. De esta manera, los clientes 1 y 3 quedarán juntos en un WTP.

- **Intervalo 3**: En este período quizás no se llegue a transmitir 20 Mbits/s en cada WTP, pero tampoco se hará ningún traslado, puesto que no habrá capacidad de sobra en ningún WTP como para aceptar un nuevo cliente. Sin embargo, sí se priorizarán los clientes de la Slice 2, por lo que sus quantums se incrementarán.

- **Intervalo 4**: Aquí ocurrirá algo similar al intervalo anterior, por lo que la

única acción a tomar sería seguir acentuando las diferencias de *quantum* entre clientes de Slice 2 y 1.

Resultados Nuevamente los resultados de las ejecuciones arrojan que el algoritmo propuesto provoca un orden de prioridad entre los clientes de la red. La Figura 5.17 no demuestra grandes diferencias en comparación con el algoritmo de asignación por RSSI, sin embargo en la Figura 5.18 sí se percibe un peor servicio en promedio utilizando el algoritmo desarrollado. De todos modos, todos los clientes de la Slice 1 reciben en todo momento una tasa de transferencia mayor o igual a la prometida.

En cambio, los clientes de la Slice 2 se ven beneficiados en gran medida por la utilización del algoritmo presentado. Aún así no se logra que ambos clientes de esta Slice transmitan a los valores prometidos, pero esto se debe a una particularidad del escenario, ya que en ciertos intervalos se tiene una tasa ofrecida conjunta mayor a la capacidad de ambos WTPs.

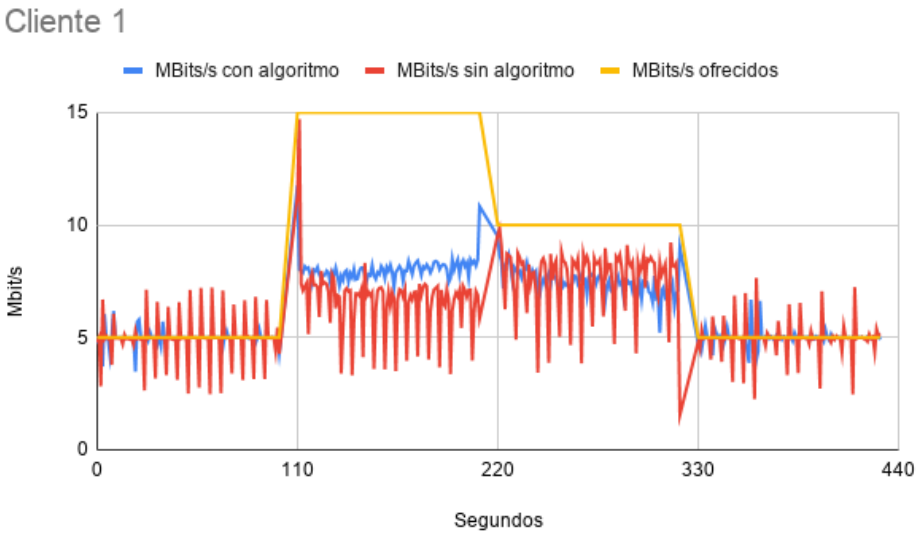

Figura 5.17: Escenario completo 2: Mbit/s recibido por cliente 1. Tasa prometida: 5 Mbit/s

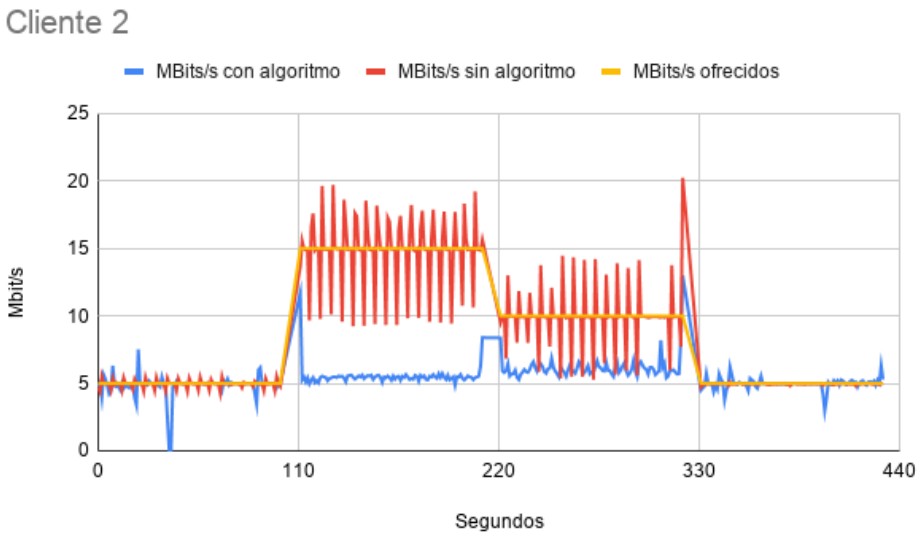

Figura 5.18: Escenario completo 2: Mbit/s recibido por cliente 2. Tasa prometida: 5 Mbit/s

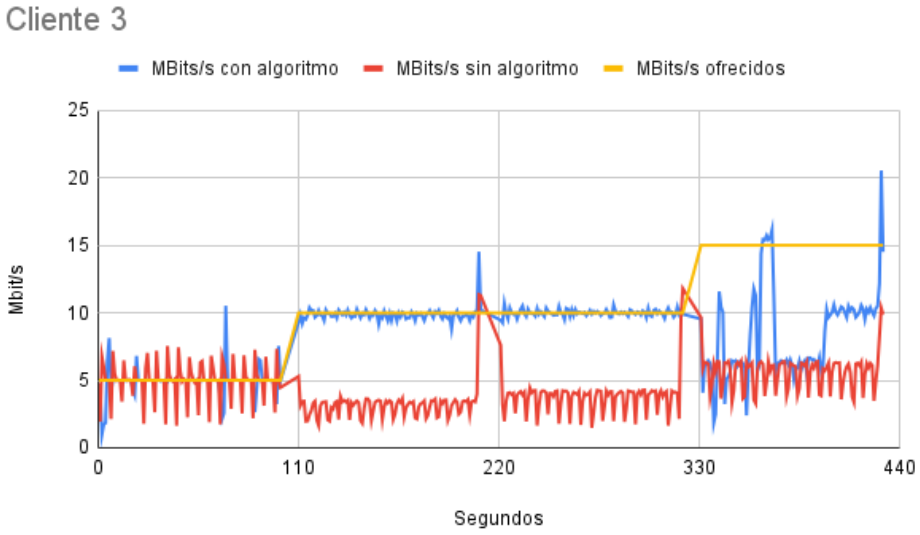

Figura 5.19: Escenario completo 2: Mbit/s recibido por cliente 3. Tasa prometida: 15 Mbit/s

Figura 5.20: Escenario completo 2: Mbit/s recibido por cliente 4. Tasa prometida: 15 Mbit/s

Observando la Figura 5.23 se detecta el patrón seguido por el algoritmo a la hora de decidir qué clientes enviar a cuál WTP. En este caso realiza el cambio más lógico al comienzo, asignando un cliente de cada Slice a cada WTP. De este modo siempre y cuando los clientes de las mismas Slices intenten transmitir tasas parecidas se podría generar una especie de balance entre los WTPs.

El análisis de las Figuras 5.21 y 5.22 también muestra la prioridad que se les otorgó a los clientes de la Slice 2 en ambos WTPs.

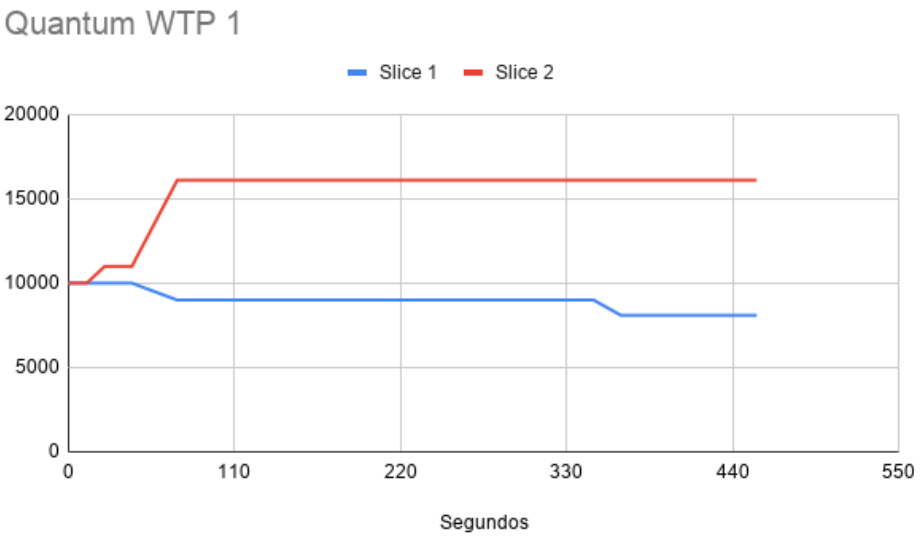

Figura 5.21: Escenario completo 2: Asignación de *quantums* en WTP 1.

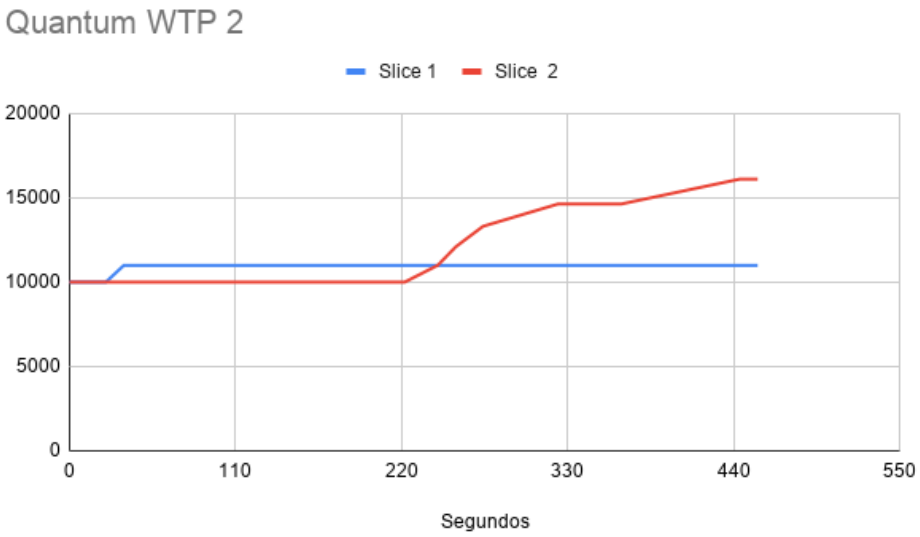

Figura 5.22: Escenario completo 2: Asignación de *quantums* en WTP 2.

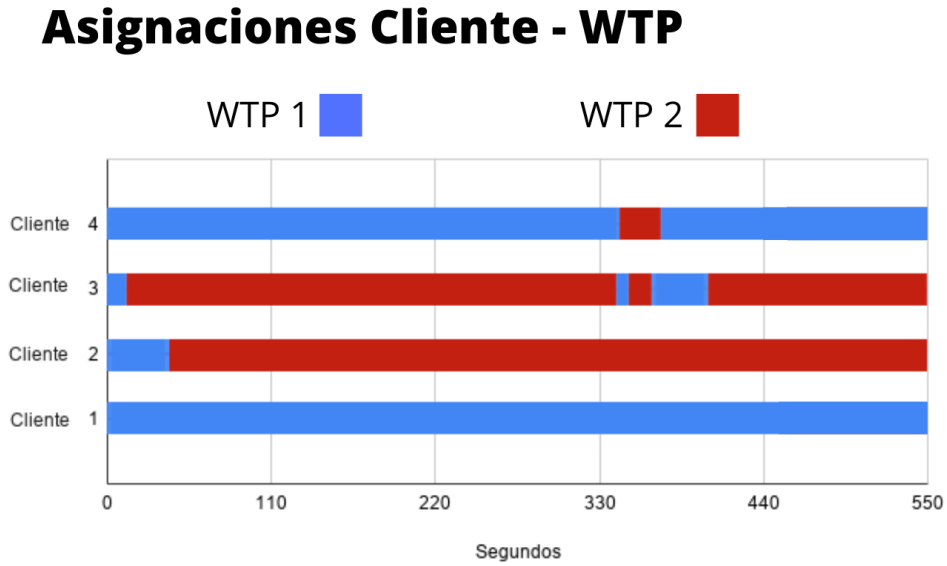

Figura 5.23: Escenario completo 2: Asignación de clientes a WTPs.

Como un indicador de qué tan beneficioso fue para los clientes el uso del algoritmo, utilizamos el mismo que en el escenario 1: Ecuación (5.3).

Los índices de satisfacción para este escenario se muestran en la Tabla 5.4. Los cliente tuvieron un índice de satisfacción mayor usando el algoritmo propuesto, y en particular se aprecia mayor diferencia en los clientes con mayores promesas. Por su parte, los porcentajes de ganancia se comportan de manera similar al escenario previo, pues pudo compensar lo percibido por los clientes 3 y 4 mediante la baja de calidad de servicio otorgado al cliente 2, y cumpliendo igualmente con la tasa prometida.

Cliente	**Índice con algoritmo**	**Índice sin algoritmo**	**Ganancia**
1	1.380	1.043	32.311 %
2	0.373	3.672	-89.842 %
3	-1.691	-5.590	69.750 %
4	-3.974	-7.207	44.859 %

Tabla 5.4: Índice de Satisfacción en Escenario Completo 2.

Capítulo 6

Conclusiones y trabajo a futuro

6.1. Conclusiones

En este proyecto se actualizó el relevamiento de herramientas de SDN en redes inalámbricas, donde se pudo ver que aún no se dispone de un estándar para este nuevo paradigma. Cada año surgen nuevos *frameworks* con sus particularidades, ventajas y desventajas.

Como objetivo de este proyecto, se propuso profundizar sobre la herramienta 5G-Empower para posteriormente desarrollar y desplegar una aplicación en la misma. Al investigar sobre este controlador se advirtió que el repositorio se actualiza constantemente (no hay una versión estable), lo cual genera dificultades para usar este *framework* en un ambiente de producción. Por otro lado, el código tiene ciertas diferencias con las características mencionadas en los *papers* que lo describen y hay una ausencia de documentación oficial, por lo cual se gace necesario analizar el código fuente para comprender su funcionamiento y crear nuevas aplicaciones. Un aspecto negativo del *framework* es la implementación de los agentes, los cuales usan *Click Modular Router* como implementación del camino de datos, herramienta que no ha recibido mantenimiento en los últimos años (*Repositorio Click* s.f.) y que agrega importante *overhead*. Una alternativa sería modificar el *firmware* del AP. De las últimas publicaciones realizadas por los autores 5G-EmPOWER, se lee que se está dejando de lado la implementación del agente WiFi y se está enfocando sobre la implementación del agente *lte*, y recientemente comenzaron a trabajar en la simulación de una red Empower en *ns3*.

Como aspectos positivos, es resaltable que la *webui* es intuitiva y elegante lo cual facilita la ejecución de aplicaciones y gestión de red mediante este método. Sobre esa base y de videos de *demo* publicados por los autores, fue posible desplegar una red Empower sin mayores dificultades. Una ventaja importante es que el controlador presenta varias aplicaciones integradas que sirven de ejemplo y base para la

implementación de nuevas aplicaciones. Por otro lado, el *framework* provee la opción de extender el agente para agregar nuevas funcionalidades. En el presente caso no fue necesario porque se está basado en los paquetes definidos por el protocolo *OpenEmpower* para realizar las aplicaciones.

Teniendo en cuenta las funcionalidades del controlador y tomando como base el trabajo realizado en (Richart, Baliosian, Serrat, Gorricho y Aguero, 2019), se propuso una heurística para cumplir con promesas de *bit-rate* garantizado a clientes, utilizando *slicing* en la red. Usar SDN para implementar la heurística parece una prometedora opción ya que desde el controlador se tiene una visión global de la red y se puede controlar cada WTP de forma individual. Además, la abstracción LVAP hace que los *handovers* sean transparentes para los clientes. Aprovechando que Empower brinda estas funcionalidades y además soporta la funcionalidad de *slicing* por *airtime*, se desarrolló la heurística utilizando esta herramienta.

Se logró implementar un algoritmo de control a nivel global que modifica parámetros de los WTP de forma individual y además, se se demostró que el *slicing* por *airtime* trabajó de forma correcta permitiendo una diferenciación en el servicio a los clientes como se esperaba. Durante las pruebas se comprobaron las mejoras en las tasas prometidas usando la heurística en comparación con un algoritmo de asignación de WTP por RSSI. Adicionalmente se cumplió durante la mayor parte del tiempo con las promesas de los clientes, siempre y cuando fueran posibles. Se concluye que se alcanzó una solución que además de servir para demostrar el uso de Empower es útil y plantea una solución a un problema existente.

Los resultados demostraron la utilidad de implementar SDN para gestionar una red inalámbrica. Al tener una visión general de la red y poder manipular los APs individualmente, se puede modificar el uso de los recursos de la red dinámicamente en base a las necesidades de la situación. Es de particular interés remarcar que este algoritmo se implementó en un lenguaje de alto nivel, donde no se tuvo que considerar las particularidades del *hardware* de los dispositivos.

Finalmente, se puede concluir que se cumplieron todos los objetivos planteados para este proyecto.

6.2. Trabajo a futuro

Luego de haber explicado y evaluado la implementación del algoritmo, es oportuno mencionar algunas posibles mejoras o alternativas que se le pueden realizar al algoritmo, así como plantear otros temas que pueden ser de interés para continuar la investigación en el futuro.

Respecto al algoritmo propuesto, la implementación siempre decide tomar una acción sobre el cliente al cual no se le está cumpliendo la tasa prometida. Una

posible modificación al algoritmo sería evaluar si no es mejor realizar acciones sobre los clientes que están en el mismo WTP que este cliente. Por otro lado, en lugar de procesar de a un cliente a la vez, se podría analizar las estadísticas de todos los clientes, para posteriormente tomar acciones.

Con foco ahora en las estadísticas que usa el algoritmo para tomar decisiones, se podría agregar algún dato sobre las colas de paquetes de los WTPs, por ejemplo el tiempo de espera que tuvieron para ser transmitidos. Con esta estadística se tendrán datos más certeros del estado y carga actual de los WTPs. Otra mejora interesante sería agregar una acción nueva, además de realizar *handovers* y modificar *quantums*, podría ser de interés evaluar la opción de que el algoritmo pueda asignar los clientes a nuevos canales. Para esto sería necesario contar con estadísticas de los canales para saber cuál de ellos sería el óptimo. Para cualquiera de las implementaciones mencionadas en este párrafo que se realicen sobre el *framework* EmPOWER sería necesario extender el agente.

Otro posible cambio para analizar, que constituye un problema relativamente complejo, sería simular la acción (*handover* o cambio de *quantum*) mediante otra heurística antes de ejecutarla para saber de antemano cuál sería su impacto. En este caso se podrían simular varias acciones para finalmente elegir la mejor y ejecutarla.

Complejizando aún más el problema, podría interesar la posibilidad de que el algoritmo utilice métodos de aprendizaje automático, donde dado un estado de la red puede aprender a detectar si hay problemas y luego tomar una decisión al respecto. Esto conlleva un análisis complejo, de modo que en una primera instancia habría que plantear las siguientes cuestiones previas: analizar la función objetivo, estudiar la posibilidad de tener un conjunto de datos para entrenar y evaluar, cómo se medirá la eficiencia del algoritmo y qué modelo de aprendizaje utilizar.

Por último, seria de utilidad realizar una investigación y puesta a punto de los *framework* que tienen capacidad de simulación sobre alguna herramienta, por ejemplo en *ns3*. Esto sería de gran ayuda para probar algoritmos y heurísticas ya que permite simular de forma simple escenarios complejos.

Bibliografía

Chamlian, A., Telfeyan, G. & Piquerez, N. (2016). Software Defined Networking en redes Inalámbricas.

5g-EmPOWER Wiki. (s.f.). https://github.com/5g-empower/5g-empower.github.io/wiki (Accessed: 05.04.2021)

Nunes Astuto, B., Mendonça, M., Nam Nguyen, X., Obraczka, K., Turletti, T., Astuto Nunes, B. A., Mendonca, M. & Nguyen, X.-N. (2014). A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *16*(3). https://doi.org/10.1109/SURV.2014.012214.00180Ãŕ

Kreutz, D., Ramos, F. M. V., Verissimo, P., Rothenberg, C. E., Azodolmolky, S. & Uhlig, S. (2014). Software-Defined Networking: A Comprehensive Survey. http://arxiv.org/abs/1406.0440

Mamushiane, L., Lysko, A. & Dlamini, S. (2018). A comparative evaluation of the performance of popular SDN controllers. *IFIP Wireless Days*, *2018-April*, 54-59. https://doi.org/10.1109/WD.2018.8361694

Controlador Ryu. (s.f.). https://ryu.readthedocs.io/en/latest/getting_started.html#what-s-ryu (Accessed: 11.05.2021)

Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W. & Parulkar, G. (2014). Onos, 1-6. https://doi.org/10.1145/2620728.2620744

Medved, J., Varga, R., Tkacik, A. & Gray, K. (2014). OpenDaylight: Towards a model-driven SDN controller architecture. *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, WoWMoM 2014*. https://doi.org/10.1109/WoWMoM.2014.6918985

Dezfouli, B., Sheth, J. & Radi, M. (2018). A Review of Software-Defined WLANs: Architectures and Central Control Mechanisms. (1).

Murty, R., Padhye, J., Chandra, R., Wolman, A. & Zill, B. (2008). Designing High Performance Enterprise Wi-Fi Networks. https://www.researchgate.net/publication/220832110

Murty, R., Wolman, A., Padhye, J. & Welsh, M. (s.f.). An Architecture for Extensible Wireless LANs.

Murty, R., Padhye, J., Wolman, A. & Welsh, M. (2010). Dyson: An Architecture for Extensible Wireless LANs. https://www.researchgate.net/publication/234778369

Suresh, L., Huehn, T., Schulz-Zander, J., Sarrar, N., Feldmann, A., Hühn, T., Merz, R. & Berlin, T. (2014). Programmatic Orchestration of WiFi Networks Minstrel-Blues View project Programmatic Orchestration of WiFi Networks. https://www.researchgate.net/publication/263010363

Riggio, R., Rasheed, T. & Granelli, F. (2013). EmPOWER: A testbed for Network Function Virtualization research and experimentation. *SDN4FNS 2013 - 2013 Workshop on Software Defined Networks for Future Networks and Services*. https://doi.org/10.1109/SDN4FNS.2013.6702538

Zubow, A., Zehl, S. & Wolisz, A. (2016). BIGAP - Seamless handover in high performance enterprise IEEE 802.11 networks. *Proceedings of the NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 445-453. https://doi.org/10.1109/NOMS.2016.7502842

Schulz-Zander, J., Sarrar, N., Schmid, S. & Berlin, T. U. (s.f.). AeroFlux: A Near-Sighted Controller Architecture for Software-Defined Wireless Networks.

Vestin, J., Dely, P., Kassler, A., Bayer, N., Einsiedler, H. & Peylo, C. (s.f.). ACM MobiCom 2012 Poster: CloudMAC-Towards Software Defined WLANs.

Moura, H., Alves, A. R., Borges, J. R., Macedo, D. F. & Vieira, M. A. (2020). Ethanol: A Software-Defined Wireless Networking architecture for IEEE 802.11 networks. *Computer Communications*, *149*, 176-188. https://doi.org/10.1016/j.comcom.2019.10.010

Riggio, R., Marina, M. K., Schulz-Zander, J., Kuklinski, S. & Rasheed, T. (2015). Programming Abstractions for Software-Defined Wireless Networks. *IEEE Transactions on Network and Service Management*, *12*(2), 146-162. https://doi.org/10.1109/TNSM.2015.2417772

Richart, M., Baliosian, J., Serrat, J. & Gorricho, J. L. (2016). Resource Slicing in Virtual Wireless Networks: A Survey. *IEEE Transactions on Network and Service Management*, *13*(3), 462-476. https://doi.org/10.1109/TNSM.2016.2597295

Riggio, R. & Coronado, E. [Estefanía]. (2018). Lasagna : Programming Abstractions for End – to – End Slicing in Software – Defined WLANs.

Coronado, E. [Estefanía], Khan, S. N. & Riggio, R. (2019). 5G-EmPOWER: A software-defined networking platform for 5G radio access networks. *IEEE Transactions on Network and Service Management*, *16*(2), 715-728. https://doi.org/10.1109/TNSM.2019.2908675

Repositorio git del agente empower. (s.f.). https://github.com/5g-empower/empower-lvap-agent (Accessed: 05.04.2021)

Kohler, E., Morris, R., Chen, B., Jannotti, J. & Kaashoek, M. F. (2000). The click modular router. *ACM Transactions on Computer Systems*, *18*(3), 263-297. https://doi.org/10.1145/354871.354874

Repositorio git del controlador empower. (s.f.). https://github.com/5g-empower/empower-runtime (Accessed: 27.03.2021)

Empower Rest API. (s.f.). https://github.com/5g-empower/5g-empower.github.io/wiki/REST-API (Accessed: 05.04.2021)

Red de referencia. (s.f.). https://github.com/5g-empower/5g-empower.github.io/wiki/Reference-Network-setup (Accessed: 05.04.2021)

Repositorio git del proyecto. (s.f.). https://github.com/ericbrinckhaus/empower-runtime-modified (Accessed: 05.04.2021)

Repositorio git para compilar imagen del agente. (s.f.). https://github.com/5g-empower/empower-openwrt (Accessed: 05.04.2021)

Repositorio git con las configuraciones del agente. (s.f.). https://github.com/5g-empower/empower-configs (Accessed: 05.04.2021)

Repositorio con imagenes OpenWRT del agente Empower. (s.f.). https://github.com/5g-empower/5g-empower.github.io/wiki/Pre-built-WTP-Firmwares (Accessed: 05.04.2021)

sitio iperf3. (s.f.). https://iperf.fr/ (Accessed: 03.05.2021)

Richart, M., Baliosian, J., Serrat, J., Gorricho, J. L. & Aguero, R. (2019). Guaranteed Bit Rate Slicing in WiFi Networks. *IEEE Wireless Communications and Networking Conference, WCNC*, *2019-April*. https://doi.org/10.1109/WCNC.2019.8885636

Repositorio Click. (s.f.). https://github.com/kohler/click (Accessed: 20.05.2021)

Apéndice A

Anexo

Incluimos en el Anexo un manual de como desplegar una red Empower, teniendo los WTPs con el agente instalado y en una máquina corriendo el contenedor *Docker* con el controlador Empower, como se explica en la Sección 4.2. El archivo *Dockerfile*, se puede descargar desde el repositorio (*Repositorio git del proyecto* s.f.).

También incluimos el listado completo de los mensajes definidos por el protocolo OpenEmpower, obtenidos investigando el código del agente y controlador.

A.1. Crear una red usando el WebUI de Empower

Una vez corriendo el controlador Empower, se puede acceder a la herramienta Web de gestión de recursos mediante la url *localhost:8888*. Una vez allí el usuario deberá iniciar sesión. Existen tres cuentas creadas por defecto:

Username	**Password**	**Type**
root	root	Administrator
foo	foo	User
bar	bar	User

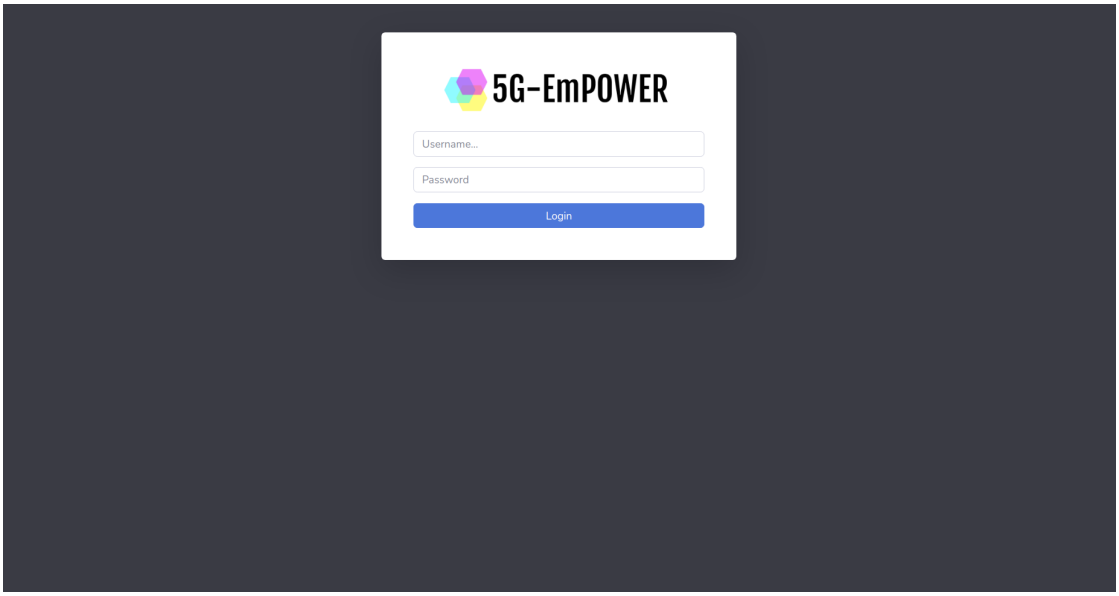

Figura A.1: Pantalla login Empower WebUI.

En primera instancia, para añadir WTPs y crear un proyecto hay que loguearse como administrador. Los pasos para incluir un WTP en el ambiente Empower se pueden ver en las siguientes capturas:

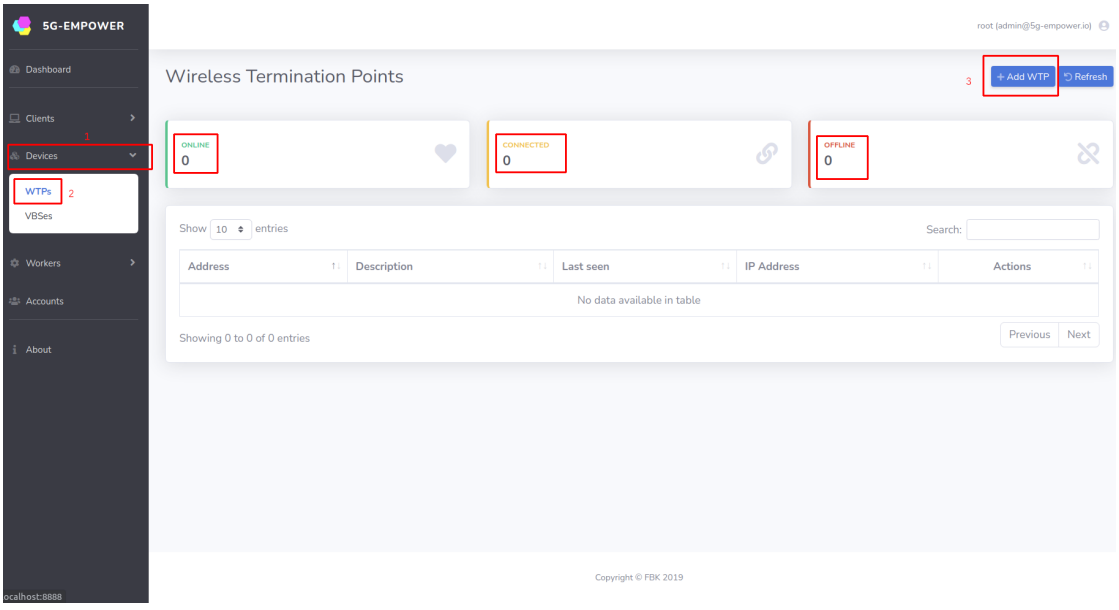

Figura A.2: Pantalla gestión de WTPs.

Accediendo al menú “Devices“->“WTPs“ se puede ver la lista y cantidad de AP conectados a la red. Allí existe la posibilidad de crear una nueva conexión.

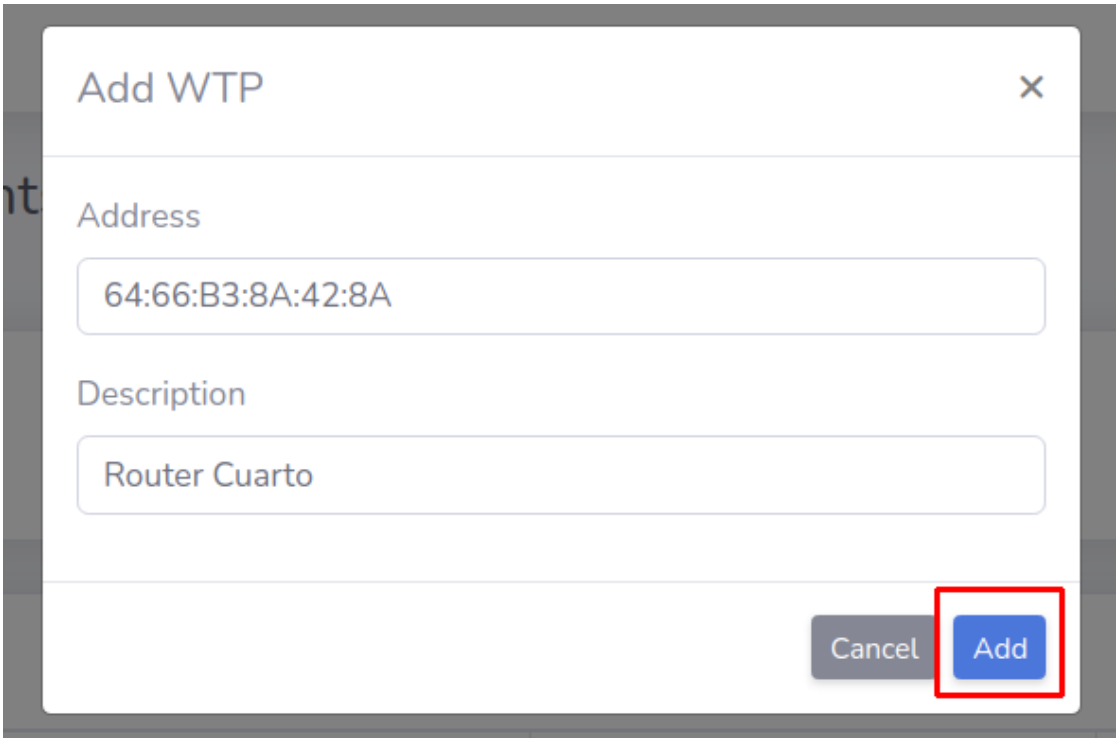

Figura A.3: Pantalla agregar WTP success.

Para agregar un nuevo WTP a la red, basta con especificar la dirección MAC y una descripción.

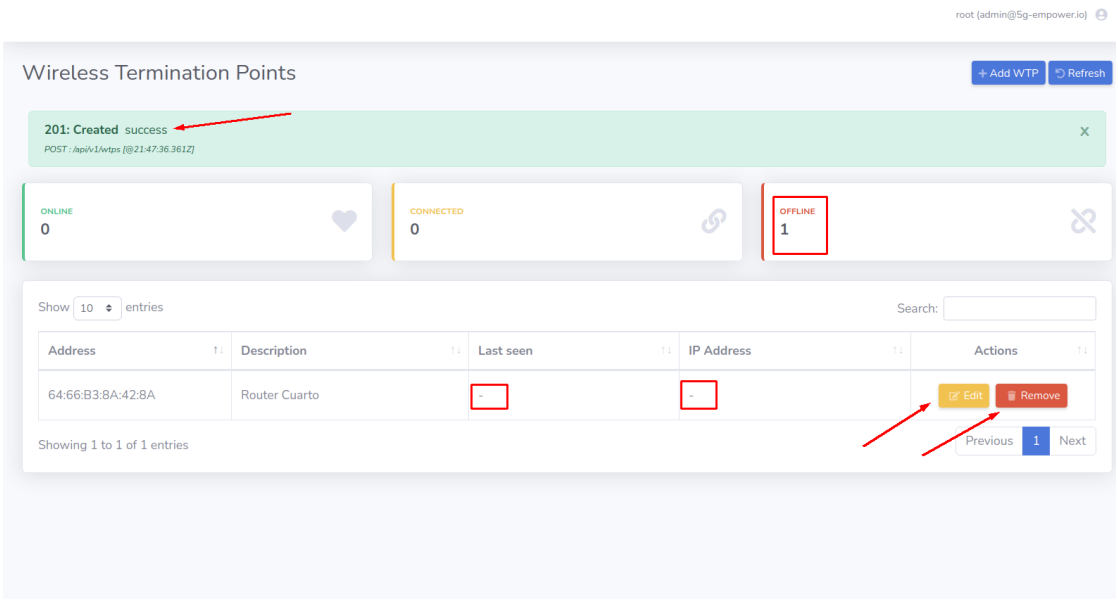

Figura A.4: Pantalla login Empower WebUI.

Una vez ingresado el WTP se percibirá como *Offline* en primera instancia, pero al cabo de unos segundos y refrescando la página se logrará conectar. También existe la posibilidad de editar su descripción, e incluso eliminarlo de la red.

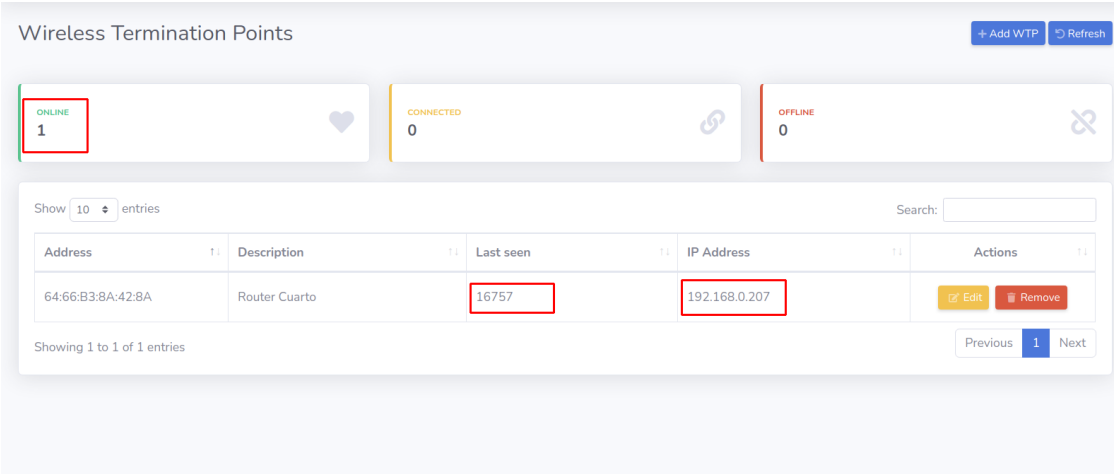

Figura A.5: Pantalla agregar WTP con WTP online.

En la misma pantalla también se puede observar su dirección IP asignada así como la fecha y hora de la última comunicación.

El flujo de creación de un proyecto es algo similar:

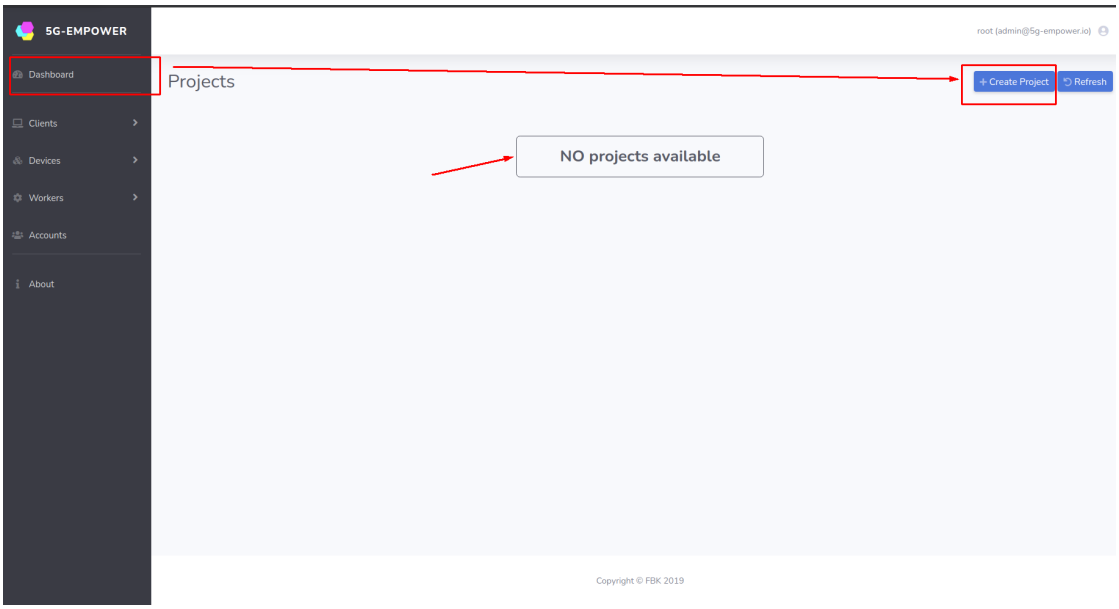

Figura A.6: Pantalla gestión de proyectos.

Se debe acceder a la pagina inicial, o “Dashboard“, donde se verá un listado con todos los proyectos existentes y estará la posibilidad de crear uno nuevo.

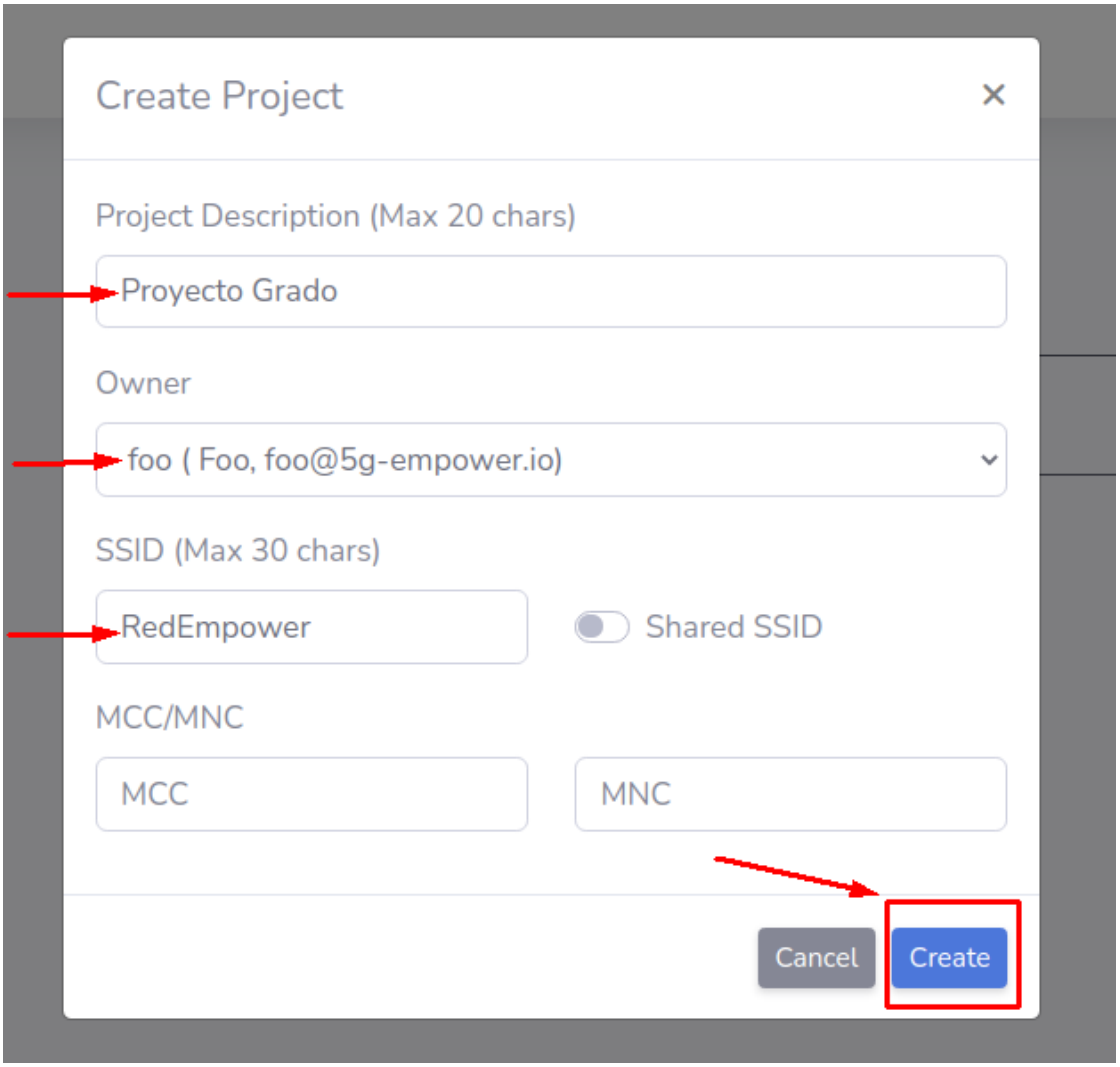

Figura A.7: Pantalla crear proyecto.

Para la creación de un proyecto se debe otorgar una descripción, el usuario dueño del mismo y el SSID, que será el nombre visible a los clientes de la red Wi-Fi. Solo los usuarios de tipo “Usuario“ pueden ser dueños de un proyecto.

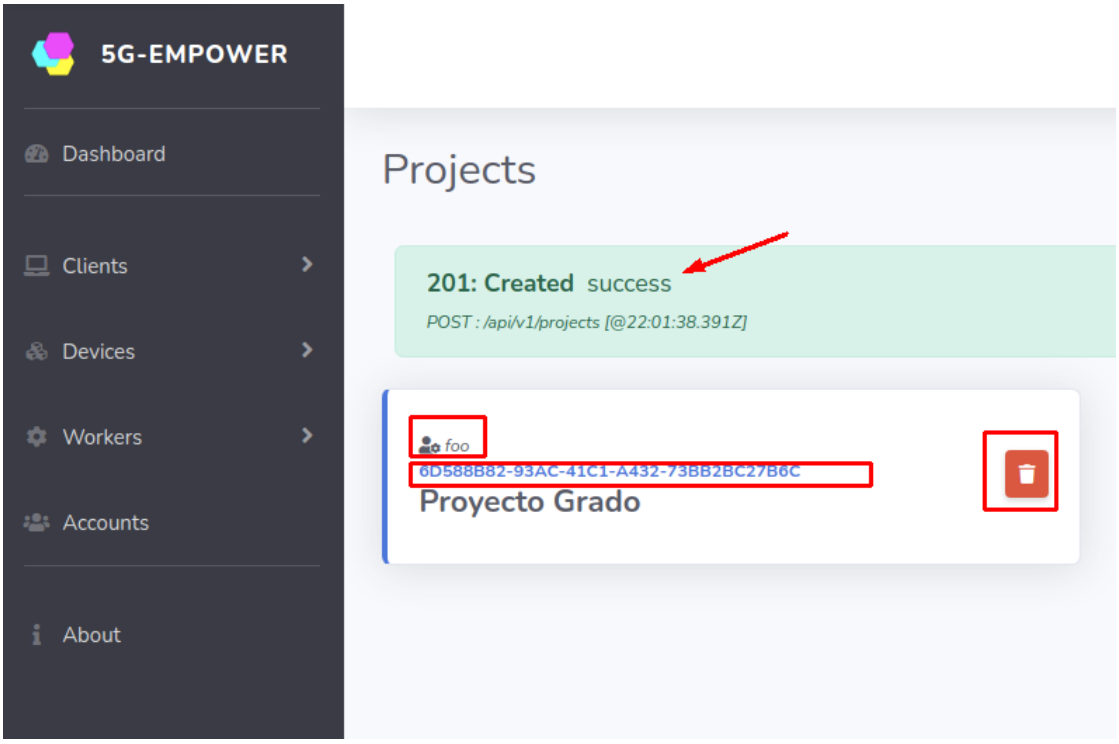

Figura A.8: Pantalla crear proyecto success.

Una vez creado el proyecto se le asignará un ID automáticamente y se podrá visualizar en el listado.

Luego de haber creado estas dos entidades, hay que iniciar sesión con el usuario dueño del proyecto para gestionarlo. En el caso de las imagen A.7 el usuario dueño del proyecto es “foo“.

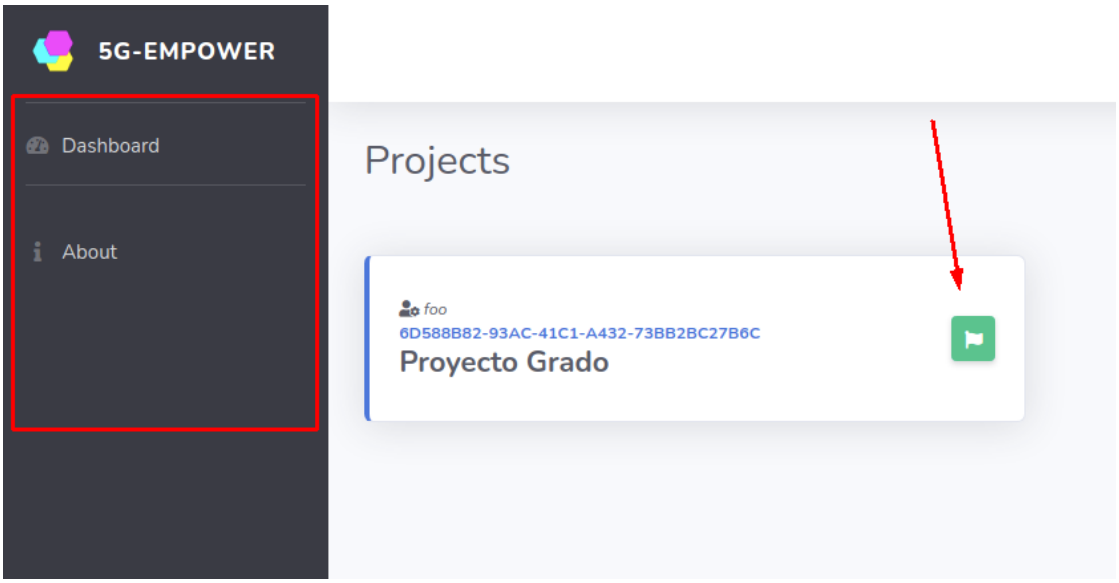

Figura A.9: Pantalla dashboard proyectos para usuarios de tipo Usuario.

Un usuario normal verá en la pagina inicial un listado con todos sus proyectos,

pudiendo seleccionar uno para editar o visualizar sus detalles. Si no tiene ningún proyecto a su nombre, no podrá realizar ninguna acción.

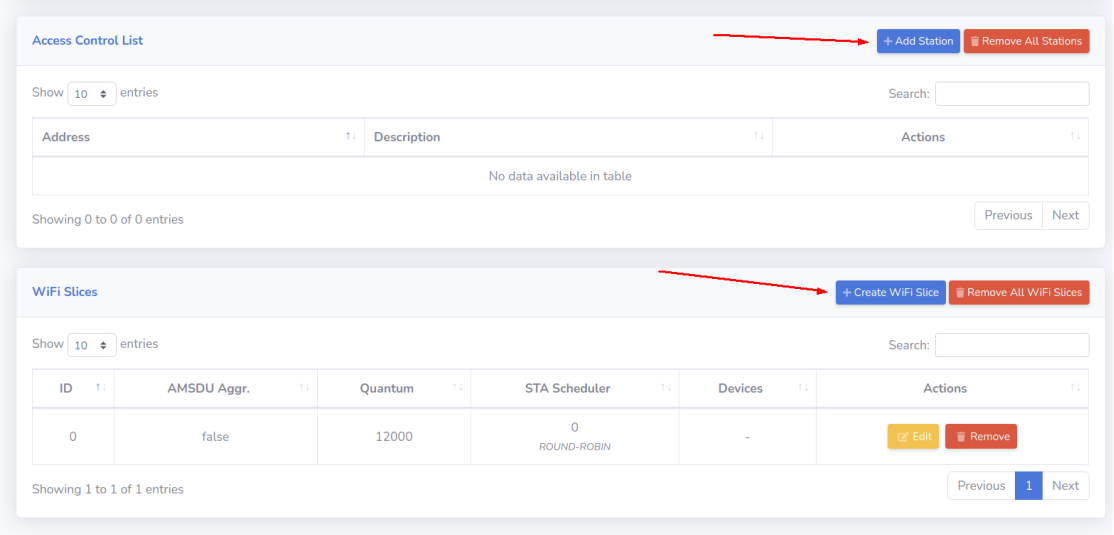

Figura A.10: Pantalla gestión ACL y Slices.

La red Wi-Fi asociada al proyecto tiene una lista de control de acceso, en donde se llevará el registro de que dispositivos tienen permiso de conectarse a la red. También existe una lista con las *Slices* pertenecientes a la red, donde existe la *Slice* con identificador 0 por defecto, aunque está la posibilidad de crear nuevas.

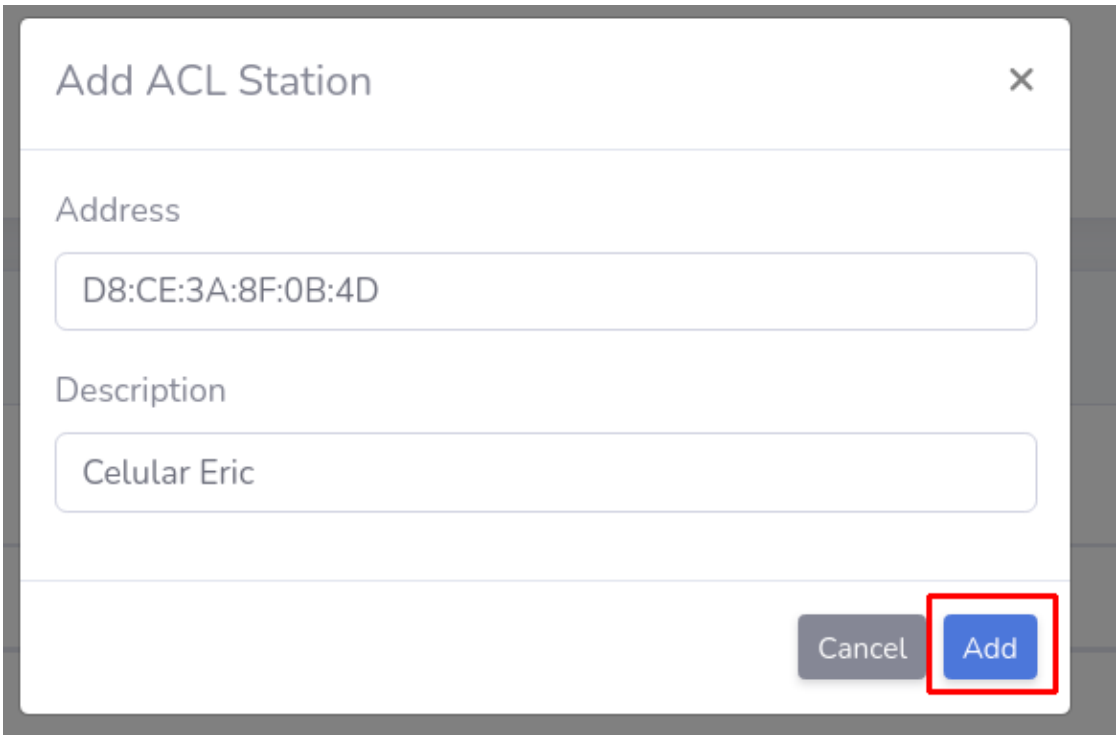

Figura A.11: Pantalla agregar ACL.

En la pantalla de creación de una entrada a la lista de control de acceso, se debe

especificar la dirección física, además de una descripción en lenguaje natural para luego identificar el dispositivo.

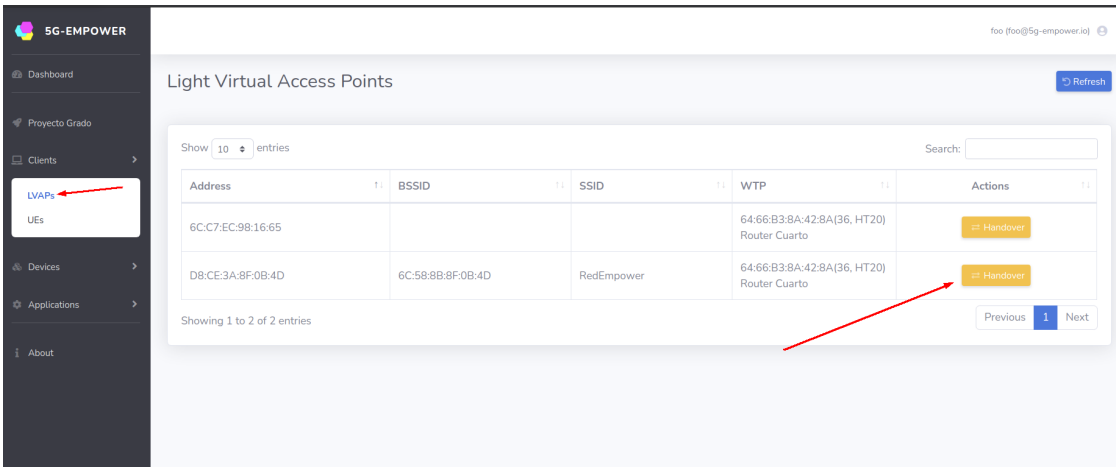

Figura A.12: Pantalla gestión LVAPs.

Una vez ingresados los dispositivos con acceso se podrán ver y editar sus detalles. Al navegar al menú “Clients“ ->“LVAPs“ se puede ver una lista de los dispositivos conectados en la red. Para cada dispositivo se puede ver el WTP y bloque a la cual esta conectado el dispositivo así como su BSSID. En la misma pantalla se brinda la opción de realizar un *Handovers* a los dispositivos. La existencia de valores en las columnas BSSID y SSID indica si el dispositivo está conectado a la red o no.

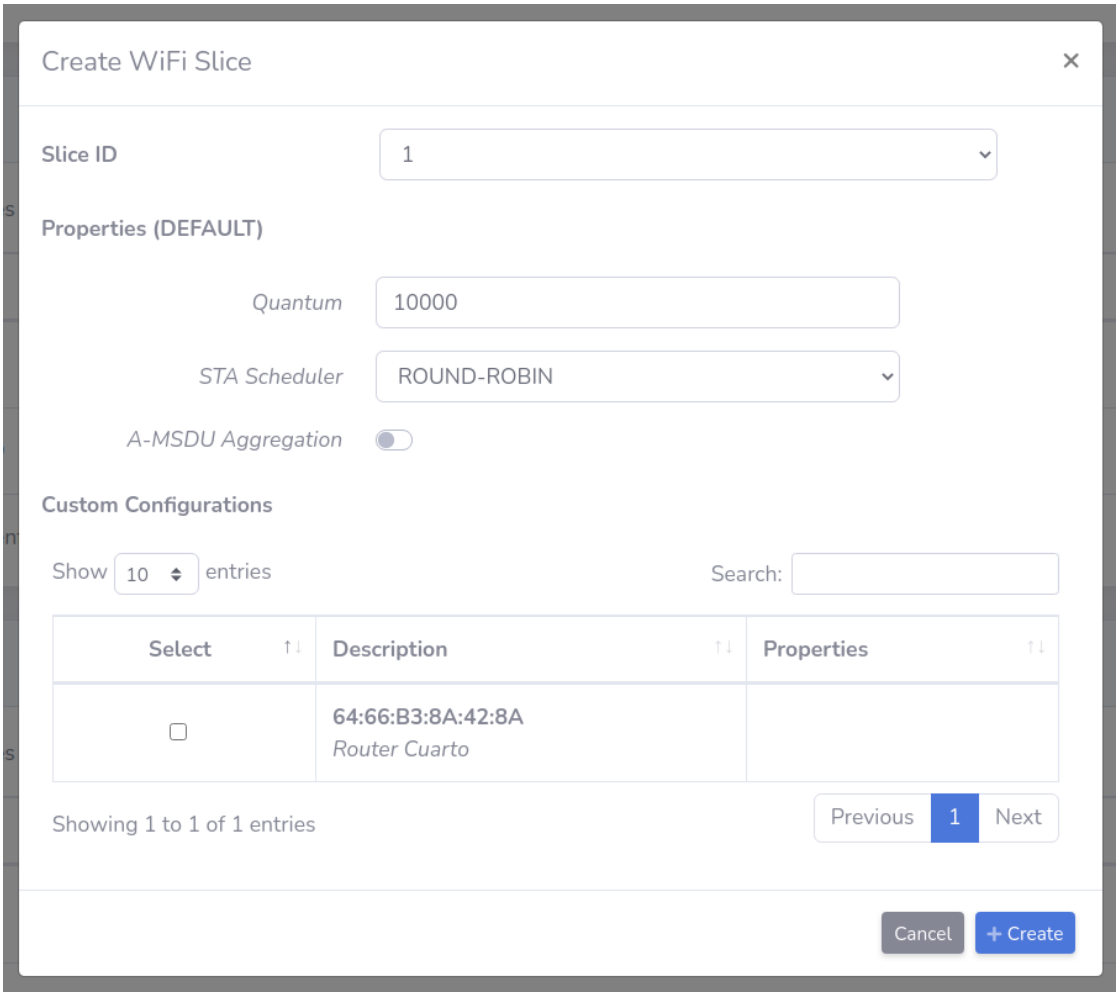

Figura A.13: Pantalla para crear una Slice.

Por su parte, al crear una *slice*, se le debe asignar en primer lugar un ID, que será luego el valor de DSCP que se utilizará para reconocer los paquetes IP. También se debe especificar las características de la *slice* (*quantum*, *scheduler type* y *A-MSDU aggregation*) y seleccionar los WTPs en donde se aplicarán estas reglas, si no se selecciona ningun WTP, se aplica a todos los WTPs de la red.

A.2. Levantar las aplicaciones creadas

Dada la naturaleza de la aplicación de control propuesta en este proyecto, debe existir información en la base de datos previo al comienzo de la ejecución de la misma. Allí deben existir entradas que indiquen por una parte todos los clientes pertenecientes a cada *slice*, y por otro las tasas prometidas a cada *slice*.

Cuando se quiera levantar el controlador, previo a correr el servicio se debe popular la base. Esto se hace ejecutando el *script Python* llamado *scipt_db.py* que se encuentra en el directorio raíz del repositorio ya mencionado. Para agregar clientes nuevos, cambiar valores de tasas prometidas o mover clientes de *slices* basta con

editar el archivo *scipt_ db.py* y ejecutarlo de nuevo.

La ejecución de una aplicación dentro del contexto Empower se hace desde el menú 'Applications'->'Catalog' teniendo un proyecto seleccionado:

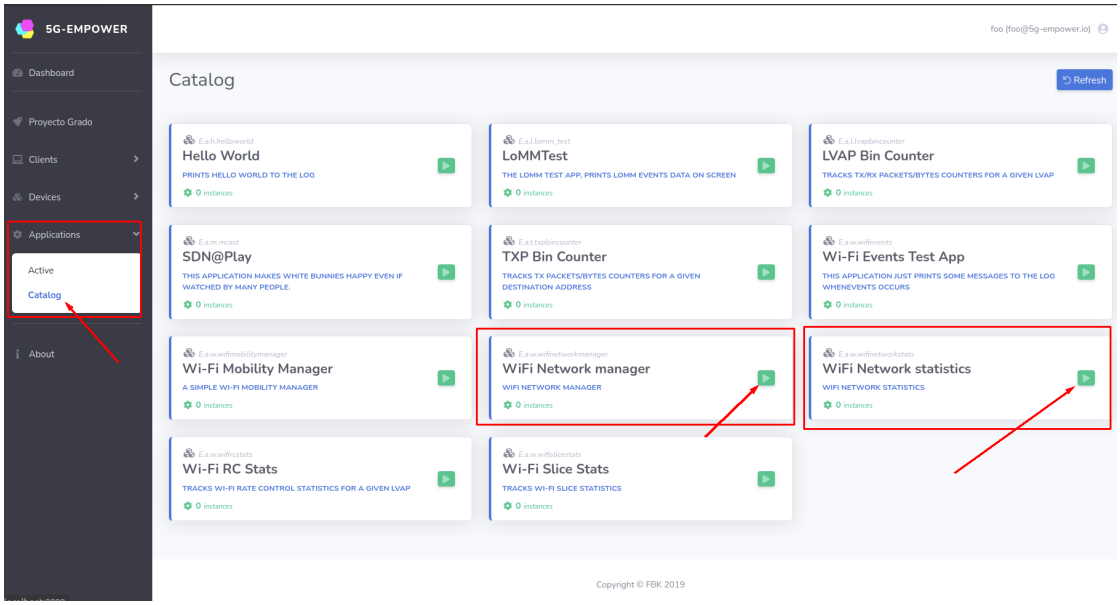

Figura A.14: Pantalla catalogo de aplicaciones.

Allí existe un listado de todas las aplicaciones disponibles para ejecutar con la opción de iniciar cualquiera de ellas.

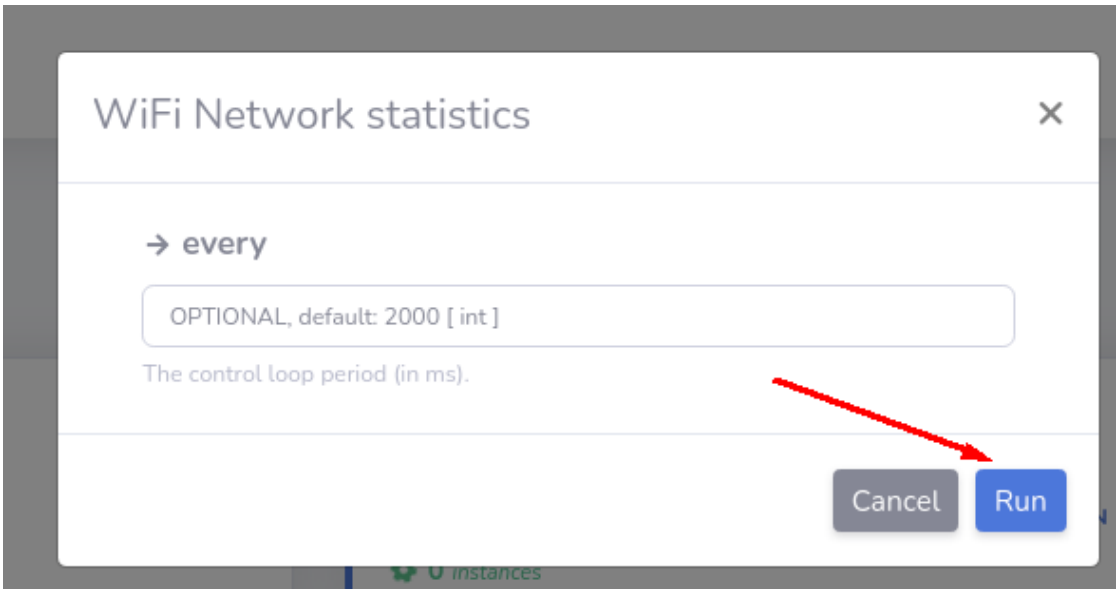

Figura A.15: Pantalla levantar instancia de una aplicación.

Al intentar ejecutar una aplicación se deberá ingresar el valor de todos los parámetros necesarios. Para cada uno de ellos existe la posibilidad de ser requerido u opcional, y se mostrará el valor por defecto en caso de dejar el campo vacío.

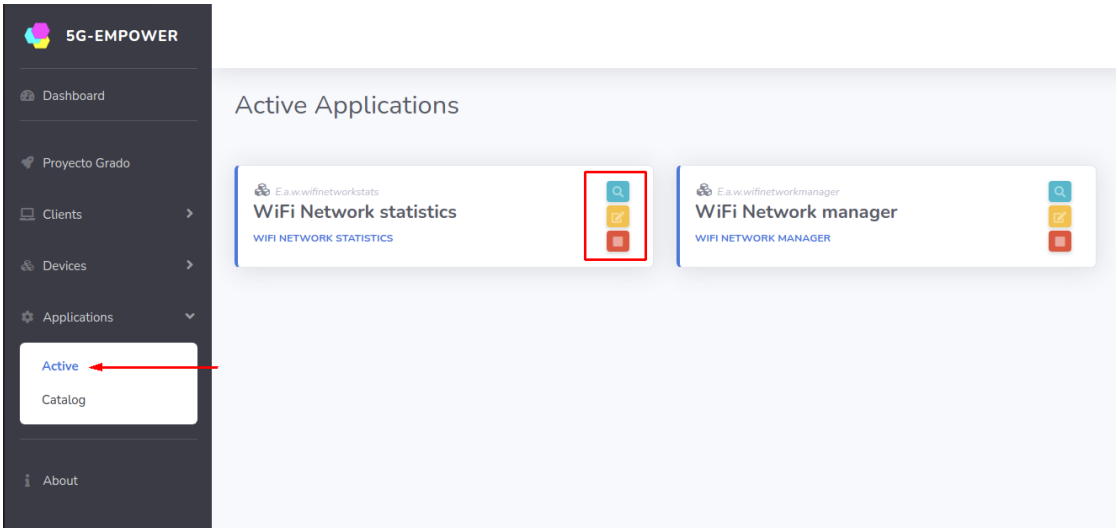

Figura A.16: Pantalla aplicaciones activas.

Una vez corriendo la aplicación, ingresando al menú 'Active' se podrán ver todas las aplicaciones en ejecución. También se puede cortar la aplicación o modificar sus parámetros durante la ejecución.

A.3. Mensajes OpenEmpower

Los mensajes que definen el protocolo OpenEmpower son los siguientes:

- **Mensajes Core** (0x01 - 0x3F)
  Mensajes base:

  HELLO_REQUEST = 0x01, // ac → wtp
  HELLO_RESPONSE = 0x02, // wtp → ac
  CAPS_REQUEST = 0x03, // ac → wtp
  CAPS_RESPONSE = 0x04, // wtp → ac
  PROBE_REQUEST = 0x05, // wtp → ac
  PROBE_RESPONSE = 0x06, // ac → wtp
  AUTH_REQUEST = 0x07, // wtp → ac
  AUTH_RESPONSE = 0x08, // ac → wtp
  ASSOC_REQUEST = 0x09, // wtp → ac
  ASSOC_RESPONSE = 0x0A, // ac → wtp

  Mensajes LVAP:

  ADD_LVAP = 0x0B, // ac → wtp
  ADD_LVAP_RESPONSE = 0x0C, // ac → wtp
  DEL_LVAP = 0x0D, // ac → wtp
  DEL_LVAP_RESPONSE = 0x0E, // ac → wtp

STATUS_LVAP = 0x0F, // wtp → ac
LVAP_STATUS_REQ = 0x10, // ac → wtp

Mensajes VAP:

ADD_VAP = 0x11, // ac → wtp
DEL_VAP = 0x12, // ac → wtp
STATUS_VAP = 0x13, // wtp → ac
VAP_STATUS_REQ = 0x14, // ac → wtp

Mensajes políticas de transmisión:

SET_PORT = 0x15, // ac → wtp
DEL_PORT = 0x16, // ac → wtp
STATUS_PORT = 0x17, // wtp → ac
PORT_STATUS_REQ = 0x18, // ac → wtp

Mensajes *Slices*:

SET_SLICE = 0x19, // ac → wtp
DEL_SLICE = 0x1A, // ac → wtp
STATUS_SLICE = 0x1B, // wtp → ac
SLICE_STATUS_REQ = 0x1C, // ac → wtp

- **Mensajes de Workers** (0x40 - 0x7F)
  Mensajes *Channel Quality Maps*:

  UCQM_REQUEST = 0x40, // ac → wtp
  UCQM_RESPONSE = 0x41, // wtp → ac
  NCQM_REQUEST = 0x42, // ac → wtp
  NCQM_RESPONSE = 0x43, // wtp → ac

  Mensajes estadísticas Wi-Fi:

  WIFI_STATS_REQUEST = 0x4A, // ac → wtp
  WIFI_STATS_RESPONSE = 0x4B, // wtp → ac

  Mensajes estadísticas de las *Slices*:

  SLICE_STATS_REQUEST = 0x4C, // ac → wtp
  SLICE_STATS_RESPONSE = 0x4D, // wtp → ac

- **Mensajes de Primitivas** (0x80 - 0xCF)
  Mensajes estadísticas de LVAP:

  LVAP_STATS_REQUEST = 0x80, // ac → wtp
  LVAP_STATS_RESPONSE = 0x81, // wtp → ac

  Mensajes contadores de paquetes y Bytes:

COUNTERS_REQUEST = 0x82, // ac → wtp
COUNTERS_RESPONSE = 0x83, // wtp → ac

Mensajes contadores de paquetes y Bytes transmitidos:

TXP_COUNTERS_REQUEST = 0x84, // ac → wtp
TXP_COUNTERS_RESPONSE = 0x85, // wtp → ac

Mensajes *Triggers*:

ADD_RSSI_TRIGGER = 0x86, // ac → wtp
RSSI_TRIGGER = 0x87, // ac → wtp
DEL_RSSI_TRIGGER = 0x88, // ac → wtp

Mensajes resumen de paquetes:

ADD_SUMMARY_TRIGGER = 0x8A, // ac → wtp
SUMMARY_TRIGGER = 0x8B, // ac → wtp
DEL_SUMMARY_TRIGGER = 0x8C, // ac → wtp

- **Mensajes Extras** (0xE0 - 0xFF)
  Mensajes *IGMP*:

  IGMP_REPORT = 0xE0, // wtp → ac
  INCOMING_MCAST_ADDRESS = 0xE1, // wtp → ac