Escuela Politécnica Superior de Linares

Grado en ingeniería de tecnologías de telecomunicación

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

GESTIÓN DE CARGA EN EDGE COMPUTING PARA VIDEOJUEGOS ONLINE SENSIBLES A LATENCIA

Alumna: Carmen Torres Montijano

Tutora: Rocío Pérez de Prado

Depto.: Ingeniería de telecomunicación

Octubre, 2019

TABLA DE CONTENIDO

NDICE DE FIGURAS

ÍNDICE DE TABLAS

1 RESUMEN

1.1 Resumen

En el presente Trabajo de Fin de Grado se pretende analizar el funcionamiento de las redes *Edge Computing* o *Fog Computing* (término que se utilizará indistintamente a lo largo del documento) que se caracterizan por ser una evolución de las redes de *Cloud Computing* que permiten la ejecución de carga con sensibilidad a latencia. En particular, el TFG tiene el objetivo de analizar su funcionamiento en base a diferentes topologías y estrategias de planificación. Para ello, se estudiarán las bondades de esta tecnología y algunas de estas estrategias para su posterior implementación en el simulador *iFogSim*, el cual es popular para el estudio de este tipo de aplicaciones.

El caso de uso propuesto es el de estudiar la reducción de latencia en un videojuego online llamado *EEG Tractor Beam*. Se propondrán diferentes posibles topologías en las que se analizará el retraso y el consumo de red, entre otros parámetros.

Además, se estudiarán diversas estrategias de planificación y su integración en el simulador, para su posterior comparación con las estrategias ya implementadas.

1.2 Abstract

The main purpose of this thesis is to analyse the behaviour of *Edge Computing* or *Fog Computing* networks (term that will be used indistinctly all along in this document). Those are considered to be an evolution of Cloud Computing networks, that allows load balancing aiming a reduction of the end-to-end latency. To this effect, is necessary the study of some scheduling techniques with the final purpose of the implementation of those in *iFogSim* simulator.

EEG Tractor Beam is proposed as the online videogame to study. Different topologies on this videogame will be proposed to study the latency, network consumption and other parameters as well.

Furthermore, a variety of load management algorithms will be revised and their integration in the simulator. Finally, the relevant results of the simulations will be compared with the algorithms that are already implemented.

2 INTRODUCCIÓN

2.1 MOTIVACIÓN

El paradigma de Internet de las cosas (IoT) promete realizar innovaciones que podrían mejorar nuestra calidad de vida. Para ello, se quiere dar conectividad a muchos de nuestros objetos cotidianos – tales como frigoríficos, cámaras y semáforos– a Internet.

Con ello se abren las puertas a invenciones que facilitarán las interacciones entre humanos y objetos, las cuales permitirán la construcción de Smart Cities, infraestructuras y servicios que mejorarán la calidad de vida de las personas a través del análisis de los datos generados por estos. Para 2020, Cisco estima que habrá alrededor de 50 mil millones de dispositivos conectados a través del paradigma del IoT [1], lo que equivaldría a una cantidad de 6.58 dispositivos por persona, como se muestra en la Figura 1.

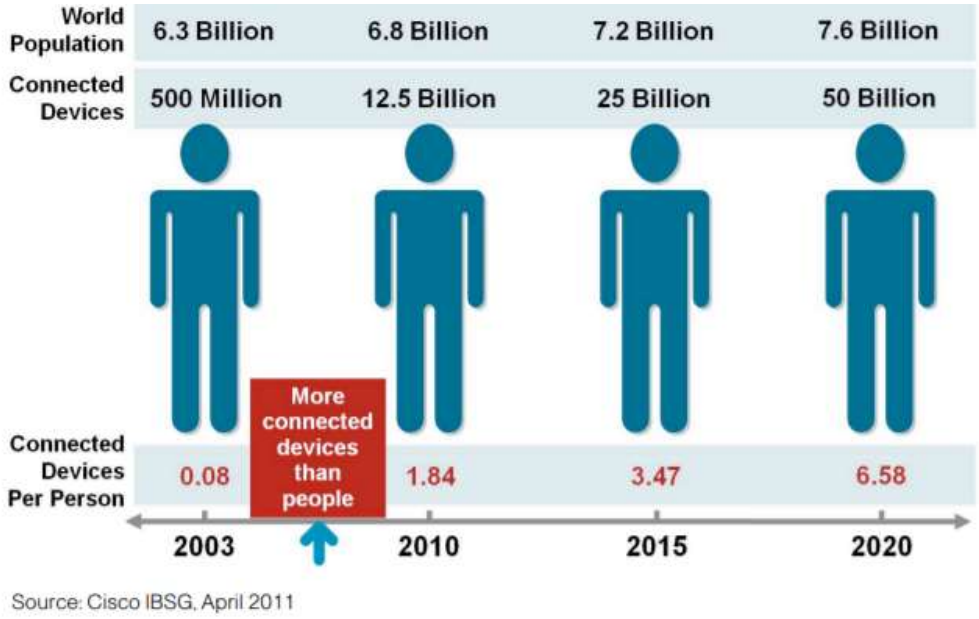


Figura 1: El avance de IoT. [1]

Cloud Computing es considerada actualmente la piedra angular en este tema, por ofrecer una infraestructura donde almacenar y procesar estos datos generados. Sin embargo, con millones de dispositivos conectados a Internet enviando constantemente datos a la nube, y con la futura evolución en el número de estos, no será viable para tomar decisiones que requieran ser en tiempo real.

Para algunas aplicaciones es inaceptable tener alta latencia (aplicaciones como servicios de emergencia o de salud), por lo que surge la necesidad de contar con algo más que servicios en la nube para procesar tal cantidad de datos. De esta forma, se comienza a hablar de *Edge Computing o Fog Computing*, el cual extiende los servicios centralizados

de la nube a los extremos de las redes de comunicación (los cuales serán dispositivos con más capacidad de procesamiento y más cercanos a la fuente emisora de datos) dando como resultado una reducción de la latencia a través de esta distribución de carga, proveyendo además de movilidad. Por lo tanto, es fundamental que la carga en estos dispositivos se gestione eficientemente para ofrecer una menor latencia final, siendo este el objetivo del presente Trabajo de Fin de Grado. En particular, se estudiará el caso de un videojuego online con requerimientos de baja latencia.

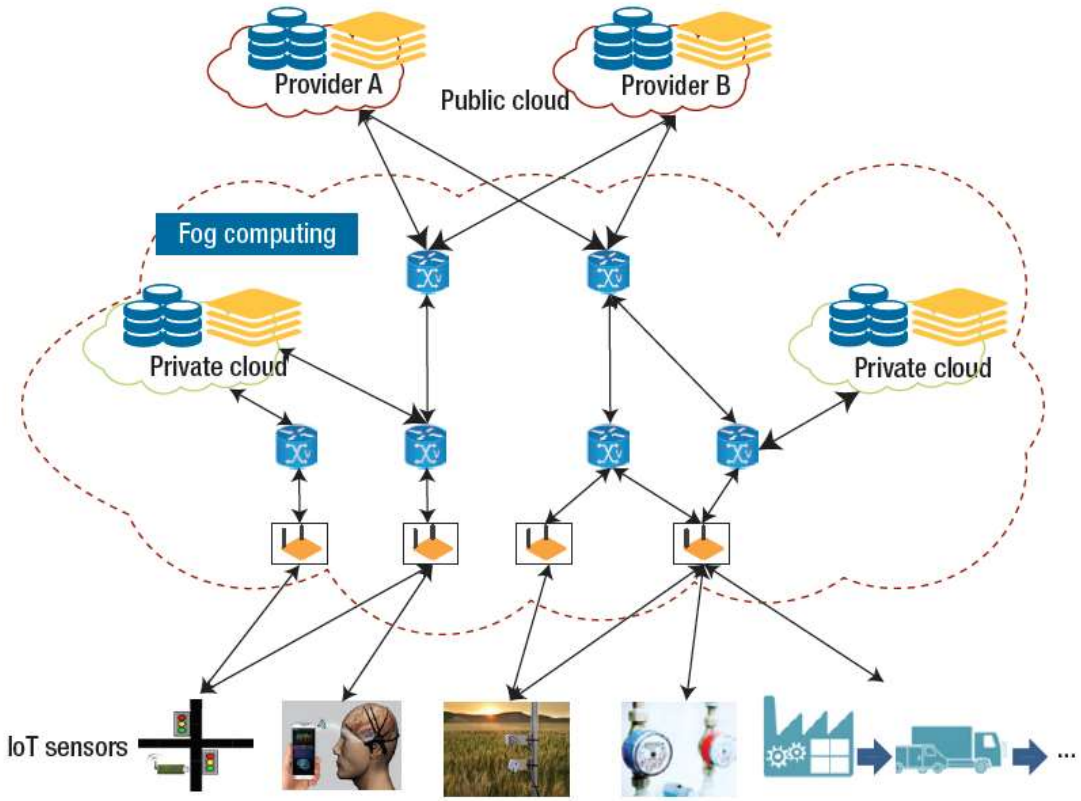


Figura 2: Procesamiento de datos distribuidos en Fog Computing. Los usuarios pueden utilizar sensores IoT para diferentes aplicaciones. Los datos recolectados serían procesados por los dispositivos Fog (entre los que están incluidos las pasarelas y nubes cercanas). [2]

2.2 Estructura del Trabajo de Fin de Grado

En este apartado se pretende dar una visión general de los temas que se cubrirán en este documento, con la idea de facilitar al lector la información que desea buscar.

2.2.1 Capítulo 1: Resumen

Se pretende dar una visión general del documento en pocas líneas.

2.2.2 Capítulo 2: Introducción

En la introducción se hace una visión general del panorama actual de computación y cuál es la motivación que conduce al desarrollo y estudio de otros sistemas.

2.2.3 Capítulo 3: Objetivos

Se sintetizan cuáles son los objetivos que se quieren alcanzar con este proyecto.

2.2.4 Capítulo 4: Materiales y métodos

En este apartado se detallará el funcionamiento del simulador utilizado y el de su predecesor, pues es necesario para un mejor entendimiento. Además, se aclarará cuáles serán los parámetros de simulación escogidos para la fase de pruebas y se hablará de la implementación de las estrategias escogidas.

2.2.5 Capítulo 5: Resultados y discusión

Con los parámetros definidos en el anterior capítulo, se presentarán y analizarán los resultados obtenidos de más interés. Se estudiarán un gran número de condiciones y parámetros que se sintetizarán para favorecer su comprensión.

2.2.6 Capítulo 6: Conclusiones

Se expondrán las lecciones aprendidas tras la elaboración de este proyecto tras analizar los casos de uso propuestos, las diversas topologías y estrategias de planificación implementadas.

2.2.7 Capítulo 7: Anexos

En este capítulo no se pretende añadir todos los resultados, pues alargaría y entorpecería la lectura de este documento. Por lo que se explicará cómo están estructurados los datos adjuntos de este proyecto para una eficaz consulta sobre ellos.

2.2.8 Capítulo 8: Referencias bibliográficas

Se citarán los trabajos consultados para la elaboración del proyecto de acuerdo a las normas básicas de estilo proveídas por la normativa de Trabajo de Fin de Grado de la Escuela Politécnica Superior de Linares.

3 OBJETIVOS

En el presente Trabajo de Fin de Grado se estudiarán técnicas de gestión de carga para reducir la latencia en sistemas sensibles a ella. En particular, en este caso el sistema sensible a latencia se trata de un videojuego online, *EEG Tractor Beam*, el cual implica una relación cerebro-computadora. Para la ejecución del juego se necesita un smartphone con Android y unos auriculares inalámbricos, los cuales procesan en tiempo real las señales EEG del estado cerebral del usuario. Una forma de minimizar la latencia es procesar las tramas de información lo más cerca posible de la fuente de datos. Por ello, se torna en un caso ideal de *Edge Computing*.

Los objetivos son:

- Estudio de sistemas de *Edge Computing*, estructuras del simulador de este tipo de tecnología, la plataforma del juego objeto de estudio, *EEG Tractor Beam* y estrategias de planificación.
- Estudio y diseño de topologías dentro del juego *EEG Tractor Beam*. También se integrarán en *iFogSim*.
- Estudio e implementación de estrategias de planificación en Java e integración en el simulador.

4 MATERIALES Y MÉTODOS

Este capítulo se queda dividido en dos partes; una primera parte consistente en el aprendizaje de todo lo referente a *Fog Computing*, desde sus precedentes (sección 4.1) a el propio simulador utilizado (sección 4.2). En este mismo capítulo se detallarán algunas extensiones realizadas para la puesta en marcha del mismo y de la elaboración de las diferentes topologías planteadas. La segunda parte se centra en el desarrollo de estrategias de planificación integradas en el simulador (sección 4.3).

Así, la estructura de este capítulo se resume de la siguiente forma:

- 4.1 – Estado del arte: En este apartado se hondará en los precedentes de *Fog Computing*, justificando por qué se considera como una alternativa a otros métodos de computación tradicionales.
- 4.2 – *Fog Computing*: Aprendizaje e introducción del simulador empleado, extensiones sobre el simulador, elaboración de diferentes topologías para su estudio.
- 4.3 – Implementación de las estrategias de planificación: Se explica el proceso de forma detallada del software desarrollado para este proyecto.

4.1 Estado del arte

4.1.1 Cloud Computing

Hasta ahora, como ya se ha mencionado antes en este documento, se ha confiado el almacenamiento e incluso el procesamiento en *Cloud Computing*.

Cloud Computing [3] distribuye la arquitectura, plataformas y aplicaciones como servicios, virtualizando todos estos y ofreciéndolos bajo demanda. Estos servicios son conocidos en la industria como *Infraestructure as a Service* (IaaS), *Platform as a Service* (PaaS) y *Software as a Service* (SaaS), dependiendo qué servicio están ofreciendo al cliente final.

Para los proveedores de servicios como Microsoft Azure o Amazon Web Services (AWS), aportan un modelo de infraestructura por el cual clientes particulares y empresas acceden a aplicaciones desde cualquier lugar y a cualquier momento bajo demanda. La dinámica de este servicio es que los clientes acceden y pagan sólo cuando los necesiten.

El poder de computación reside en un conjunto de data centers, que están equipados con los suficientes recursos para garantizar estos servicios a miles de usuarios. Suelen estar instalados en localizaciones remotas, por lo que hay una latencia inherente

en la utilización de estos. Con el incipiente aumento de dispositivos conectados a la red, se necesitará aumentar la cantidad de data centers para seguir siendo capaces de proveer los mismos servicios sin empeorar la calidad de estos.

Para probar estrategias diferentes de planificación se hace uso de simuladores. Un simulador que se usa para los entornos *Cloud* es *CloudSim*, de los mismos desarrolladores que el simulador que se ha escogido para este proyecto, *iFogSim*. En el siguiente apartado se hablará de sus características principales ya que, *iFogSim* es una extensión de este, por lo que comparten muchas clases.

4.1.1.1 Arquitectura. Diseño en capa

En la figura 3 se muestra la arquitectura en capas que sigue *Cloud Computing* [5]. En ellas, los recursos físicos del *Cloud* junto a un *middleware* (software situado entre un sistema operativo y las aplicaciones que permite la comunicación oculta entre ellos) forman la base para distribuir *IaaS* y *PaaS*. La capa superior se centra en *SaaS* haciendo uso de los servicios de capas inferiores. Los servicios de *PaaS* y *SaaS* suelen ser desarrollados por terceros.

- Aplicaciones *Cloud*.

En esta capa se incluyen aplicaciones que están directamente disponibles a usuarios finales. Estas aplicaciones son cubiertas por proveedores *Cloud* (proveedores SaaS) y accedidas por usuarios finales, ya sea por subscripción al servicio o pago por uso. Alternativamente, en esta capa, los usuarios pueden desarrollar sus propias aplicaciones.

- Middleware nivel de usuario.

En esta capa están los *frameworks* de software, como interfaces 2.0 web, que ayudan a los desarrolladores a crear interfaces. Además, provee de más tipos de herramientas para la creación de aplicaciones en la nube.

- Core de middleware.

Esta capa implementa servicios a nivel de plataforma que aporta un entorno de ejecución para alojar servicios de aplicaciones a nivel de usuario, como gestión de *SLA* dinámico. Un ejemplo conocido de servicio operando en esta capa es *Google App Engine.*

- Nivel de sistema.

El poder de computación es provisto por un conjunto de data centers que tienen desde cientos a miles de hosts. Contienen cantidades masivas de recursos que son virtualizados por servicios que permiten compartirlos entre usuarios.

Figura 3: Arquitectura en capas. [5]

4.1.2 Antecedentes: CloudSim

Para la realización de este proyecto, tal y como se ha explicado en capítulos anteriores, se ha hecho uso de un simulador bajo el lenguaje Java. El simulador escogido es *iFogSim,* no obstante, para entender parte de sus cualidades (las cuales serán detalladas más adelante), hay que hacer un estudio de este entorno de programación.

La complejidad de este simulador no reside sólo en su enorme extensión de clases, sino que también en que una buena parte de ellas vienen de su predecesor: *CloudSim* [5], un simulador desarrollado para estudiar entornos *Cloud*. Con el fin de entender el funcionamiento de *iFogSim* es fundamental conocer el funcionamiento básico de su

antecesor, así como la interrelación de algunas de sus clases, pues estas se han de ampliar con el fin de desarrollar nuevas estrategias de planificación.

CloudSim es un simulador de código abierto desarrollado en Java, el cual tiene como objetivo que el usuario pueda modelar, simular y experimentar con la infraestructura de *Cloud Computing* y de sus servicios en un entorno controlado. De esta forma, permite al desarrollador ajustar los cuellos de botella antes de emplearlos en *Cloud*s reales y experimentar con diferentes cargas y topologías de red.

4.1.2.1 Diseño e implementación de CloudSim

En esta sección, se proveerá de información brevemente detallada de las clases más relevantes que constituyen los pilares de este simulador [4], y, por tanto, los de *iFogSim*.

- *Cloud*Sim:

Es la clase principal, la cual es responsable de manejar las colas de eventos y controlar de forma secuencial la simulación de eventos. También comprueba que estos sean ejecutados en las respectivas identidades creadas con este fin. Los eventos en *Cloud*Sim están definidos con un identificador y una etiqueta. El identificador especifica a qué identidad está destinado dicho evento y la etiqueta representa el tipo de evento.

- *Cloudlet*:

Realiza la función de una tarea. Cuenta con una serie de parámetros que caracterizan las tareas que deben de ser ejecutadas en la aplicación. Algunos de sus parámetros más relevantes son: *Id* (un identificador que permitirá la asignación a otras clases) y *length* (número de instrucciones que dicha tarea ha de consumir para terminar su ejecución).

- *CloudletScheduler*

Es una clase abstracta que se puede extender para implementar diferentes estrategias que determinen como se reparte el poder de procesamiento de *Cloudlets* en una máquina virtual. Con el simulador vienen implementadas dos por defecto: *CloudletSchedulerSpaceShared* y *CloudletSchedulerTimeShared*.

- *Datacenter*

Esta clase simula el núcleo a nivel de servicios de infraestructura (hardware) que ofrecen los proveedores de servicios en la nube ya mencionados anteriormente. En otras palabras, representa un *CPD* (Centro de Procesado de Datos). Tienen una lista de hosts

en la que se almacenan todos los equipos destinados a realizar los procesamientos de tareas.

- *DatacenterBroker*

Esta clase está encargada de mediar las negociaciones entre los proveedores de *Cloud* y el usuario final. También se encarga de asignar la creación de las máquinas virtuales (a partir de ahora VM) para asignar las tareas a estas. Esta clase se puede extender para evaluar y probar nuevas estrategias.

- *Host*

Esta clase representa un recurso físico como capacidad de procesamiento o almacenamiento. Encapsula información importante como la cantidad de memoria, una lista de núcleos de procesamiento (que representan máquinas de varios núcleos), una estrategia de distribución del procesamiento de las VMs y estrategias para proveer memoria y ancho de banda a estas.

- *Pe*

Pe (Processing Element) representa una unidad CPU, definida en términos de Millones de Instrucciones Por Segundo (MIPS). La capacidad de MIPS desempeña un papel importante, pues, a mayor capacidad de MIPS en un host, más capacidad de cómputo, implicando un tiempo menor en lo que a ejecución de tareas se refiere.

- *NetworkTopology*

Esta clase contiene información sobre el comportamiento de la red (latencias) en la simulación.

- *Vm*

Esta clase modela una VM, la cual está gestionada y alojada por un componente *Cloud*. Cada VM tiene acceso a una componente que almacena las siguientes características: memoria accesible, procesador, tamaño de almacenamiento, etcétera.

- *VmAllocationPolicy*

Es una clase abstracta que representa una estrategia de aprovisionamiento que monitoriza una VM para alojarla en diferentes hosts. La principal funcionalidad de esta clase es seleccionar el host disponible en un datacenter que disponga de la memoria, almacenamiento y disponibilidad necesarias para alojarla.

- *VmScheduler*

Se trata de una clase abstracta implementada por un componente host que modela las estrategias requeridas para alojar los núcleos de procesamiento a las VMs. Las funcionalidades de esta clase pueden ser fácilmente modificadas para amoldarse a las estrategias de planificación para aplicaciones diseñadas por el desarrollador.

4.2 Fog Computing

4.2.1 Introducción

Se define *Fog Computing [5]* como un paradigma de computación distribuida que fundamentalmente extiende los servicios del *Cloud* a los límites de la red. Usa de forma continua los recursos del *Cloud* con los dispositivos limítrofes junto a su propia infraestructura. Facilita la gestión y programación de la capacidad de computación, almacenamiento y de la red entre data centers y dispositivos de usuarios finales. Esencialmente implica componentes de una aplicación ejecutándose tanto como en el *Cloud* como en los dispositivos situados entre el mismo *Cloud* y clientes finales (routers, switches inteligentes, etc). *Fog Computing* soporta movilidad, interfaces y recursos heterogéneos, interacción con la nube, análisis de datos distribuidos para conciliar los requerimientos de las aplicaciones que necesitan poca latencia con una distribución geográfica amplia y densa.

Hay unos cuantos beneficios asociados *a Fog Computing* que aseguran su éxito. El primero es la reducción del incremento del tráfico incontrolado, resultando en un control de la congestión y de la latencia general. Provee de una plataforma para el filtrado y para el análisis de los datos generados por los sensores utilizando los recursos de los dispositivos al límite de la red. Por ello, se reduce drásticamente el tráfico enviado al *Cloud* permitiendo alojar los filtros cerca de la fuente de datos. Una considerable reducción de la latencia de propagación es la siguiente ventaja importante de este paradigma, especialmente para aplicaciones críticas en las que se necesita procesado de datos a tiempo real. Por último, incluso con los ilimitados recursos virtuales de *Cloud Computing*, puede convertirse en una situación de cuello de botella si todos los datos sin procesar son enviados a un *Cloud* centralizado. *Fog computing* es capaz de filtrar y procesar una considerable cantidad de data proveniente de los dispositivos de los usuarios, consiguiendo que la arquitectura de procesado de datos sea distribuida y escalable.

En los siguientes apartados se procurarán algunos casos de uso en los que podría ser útil esta tecnología [6].

4.2.1.1 Sanidad y seguimiento de actividades

Fog Computing podría ser útil en servicios sanitarios donde el procesamiento a tiempo real es crítico. Un sistema propuesto que utiliza *Fog Computing* para detectar, predecir y prever cuando a un paciente le da un ataque cardíaco. Los algoritmos de detección temprana de infartos cerebrales hacen un uso dinámico a través de dispositivos *Fog* y recursos *Cloud*. Se realizaron pruebas y se concluyó con que este sistema tiene un tiempo de respuesta menor y consumía menos energía que sistemas que solo utilizaban el *Cloud*.

4.2.1.2 Realidad aumentada, sistemas cognitivos y videojuegos

Fog Computing juega un rol importante en aplicaciones de realidad aumentada, las cuales son muy sensibles a la latencia. Por ejemplo*, EEG Tractor Beam*, el juego propuesto de estudio en este proyecto, desempeña una clasificación del estado cerebral a tiempo real de los jugadores, y, después, clasifica los modelos en los servidores *Cloud* basándose en lecturas de electroencefalogramas que los sensores captan.

Un dispositivo *wearable* de asistencia cognitiva que utiliza las *Google Glass* ayuda a gente con agudeza mental reducida a realizar algunas tareas, incluyendo recordarles los nombres de personas que han conocido, pero no recuerdan. En esta aplicación, los dispositivos se comunican con el *Cloud* para tareas que toleran retraso como puede ser reporte de errores y logging. Para tareas que no toleran ningún retraso, el sistema transmite video desde las gafas a los dispositivos *Fog* para su procesado. Este sistema demuestra cómo el uso de dispositivos *Fog* cercanos disminuye la latencia extremo a extremo.

4.2.1.3 Servicios inteligentes

Fog Computing puede ser utilizado para servicios inteligentes, con el objetivo de mejorar la generación de energía y su consumo. En este tipo de entornos, los dispositivos Fog pueden informar de manera más detallada de su consumo energético a los usuarios de dispositivos móviles. Estos dispositivos *Fog* pueden calcular el coste del consumo eléctrico a lo largo del día y sugerir que fuente de energía es más económica a cierta hora o cuando apagar o encender ciertos electrodomésticos con el fin de minimizar su consumo.

4.2.2 Arquitectura

En la figura 4 se presenta una arquitectura de referencia [6] para *Fog Computing*. En la capa más baja, encontramos los sensores y también dispositivos *Edge* y *gateways*. Esta capa además incluye aplicaciones que pueden ser instaladas en los dispositivos finales para mejorar sus funcionalidades. Estos elementos se usarán en la capa de red para comunicarse entre ellos y entre ellos y el *Cloud*. La siguiente capa contiene los servicios y recursos de *Cloud*, que soporta la gestión y procesamiento de tareas IoT. Encima de la capa de *Cloud* tenemos el software de gestión de recursos que permite garantizar la QoS de las aplicaciones de *Fog Computing*. Finalmente, en la capa superior tenemos ciertas aplicaciones y soluciones para los usuarios finales.

Mirando dentro de la capa de Gestión de Recursos de Software, se implementan muchos servicios middleware para optimizar el uso de los recursos *Cloud* y *Fog* en nombre de las aplicaciones. El objetivo de estos servicios es reducir el coste del uso del *Cloud* al mismo tiempo que alcanzar niveles aceptables de latencia llevando la computación a los nodos *Fog*. Esto se consigue con un gran número de servicios trabajando juntos:

- *Flow and task placement (Reubicación de tareas y flujos)*

Este componente vigila si el *Cloud*, Fog y los recursos de red están disponibles para identificar los mejores candidatos para ejecutar estas tareas y estos flujos. Este componente se comunica con el servicio proveedor de recursos para indicar el número actual de flujos y tareas.

- *Knowledge Base*

Este componente almacena información histórica acerca de las demandas de la aplicación que pueden ser mediados por otros servicios para apoyar su proceso de decisiones.

- *Performance prediction*

Este servicio utiliza la información del anterior para estimar la eficiencia de los recursos disponibles del *Cloud*. Esta información es también usada por el *resource provisioning service* para decidir la cantidad de recursos que ofrecer en los casos en los que hay un gran número de tareas y flujos o cuando la eficiencia es baja.

- *Raw-Data Management*

Tiene acceso directo a las fuentes de datos y nos permite ver los datos. A veces para verlos debe ser a través de consultas. Otras veces puede ser un proceso más complejo y puede requierir *MapReduce*.

- *Monitoring*

Este servicio vigila la eficiencia y el estado de las aplicaciones y da esta información a otros servicios que la requieren.

- *Profiling*

Crea perfiles sobre cómo se consumen los recursos basándose en la información de *knowledge base* y *monitoring service*.

- *Resource Provisioning*

Este servicio es el responsable de adquirir los recursos para alojar las aplicaciones. Lo hace de forma dinámica ya que las aplicaciones y requerimientos cambian a lo largo del tiempo. La decisión de qué cantidad de recursos suministra se realiza en base a la información que proporcionan otros servicios, requerimientos de latencia de usuario y credenciales.

- *Security*

Facilita servicios de autentificación, autorización y criptografía según la requieran los servicios y aplicaciones.

Todos estos elementos y servicios descritos nos sirven de referencia. Existen aplicaciones *Fog* que pueden ser construidas sin todos estos elementos o que pueden ser construidas con algunos otros.

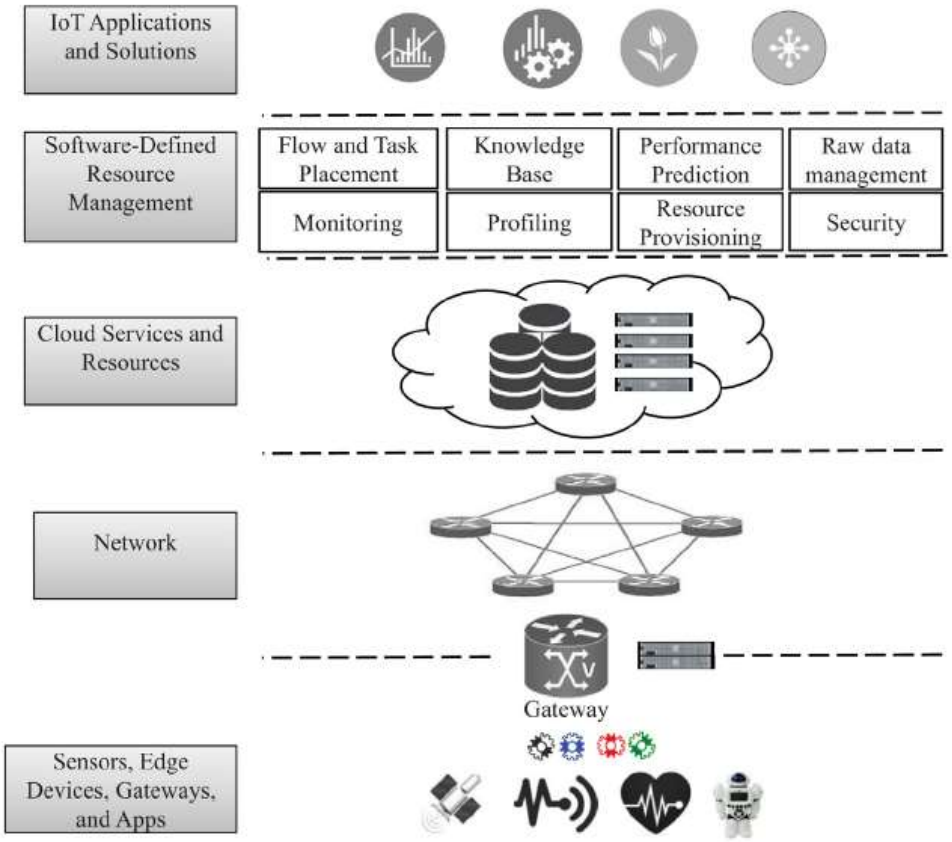


Figura 4: Arquitectura del Fog. [7]

4.2.3 Productos comerciales

Ya hay empresas que han apostado por empezar a comercializar productos basados en esta tecnología. En este apartado se verán algunos de los productos.

- CISCO IOx

La compañía Cisco es pionera en el campo de *Fog Computing*. De hecho, Cisco introdujo el término "*Fog Computing*". Su plataforma, conocida como IOx, es una combinación del sistema operativo líder de la industria, IOS, con el sistema operativo más popular de software libre, Linux [7].

Los routers se encargan de hacer visible la disponibilidad de la capacidad de computación y almacenamiento alojadas en unas VMs del sistema operativo.

Las aplicaciones que se ejecutan en la "niebla" (*Fog*) se pueden comunicar con dispositivos IoT que usen cualquier protocolo. Además, tiene implementado una consola para gestionar y monitorizar el rendimiento de la aplicación.

- LocalGrid

Esta plataforma de *Fog Computing* es un software que está integrado en dispositivos de red (switches, routers) y sensores. Esta plataforma se encarga de estandarizar y de hacer más seguras las comunicaciones entre todos los dispositivos [8], abaratando costes de customización.
La plataforma está localizada entre los dispositivos en el límite de la red y la nube. Además, todos los dispositivos se pueden comunicar con la nube a través de unos estándares de comunicación abiertos, consiguiendo una combinación de la niebla y la nube para resolver problemas más complejos.

4.2.4 Simuladores

En resumen, aunque hay varios simuladores que modelan escenarios IoT, *iFogSim* es el único que está diseñado e implementado para modelar el *Fog* junto IoT y el *Cloud*. Gracias a esta característica, permite una evaluación de estrategias de planificación para aplicaciones IoT como procesado a tiempo real en un ambiente controlado.
Además, al tratarse de un simulador de código abierto y al ser producto de colaboraciones entre varios desarrolladores, permite que se pueda customizar las clases según la aplicación que se va a desarrollar y contribuir a una comunidad para promover el avance y desarrollo.

4.2.5 iFogSim

Se ha decidido utilizar este simulador entre todos los demás por varias razones; para empezar, este simulador, al ser una extensión de *CloudSim*, ha sido utilizado y testeado por una amplia comunidad además de haber muchísimas estrategias de implementación ya estudiadas para este simulador en específico. También permite la composición jerárquica de dispositivos *Fog, Cloudlets y Clouds*, además concede la libertad a los programadores de poner retrasos entre aplicaciones a medida. Lo cual, es fundamental para nuestro estudio de la latencia.

4.2.5.1 Diseño e implementación

Para implementar las funcionalidades de la arquitectura de *iFogSim*, se han implementado las funcionalidades elementales de simulación de eventos de *CloudSim*. Entidades como

datacenters se comunican entre sí a través de mensajes (mandando eventos, más precisamente). Así pues, el núcleo de la capa de *CloudSim* es responsable de gestionar los eventos entre entidades *Fog* en *iFogSim*. En esta sección, se describen los detalles de estas clases y sus interacciones. La implementación del simulador está constituida por entidades y servicios simulados.

4.2.5.2 Clases principales

Además de las clases expuestas a continuación [9], otras clases de gran importancia son las anteriormente detalladas en *CloudSim*, puesto que son las clases compartidas de mayor importancia para el correcto funcionamiento del simulador.

- *FogDevice*

En esta clase se especifican las características de hardware y las conexiones de los dispositivos *Fog*, sensores y activadores. Esta clase es una extensión de la clase *PowerDatacenter* de *CloudSim*. Los métodos en esta clase definen como los recursos de un dispositivo *Fog* están planificados entre los módulos que se están ejecutando y como los módulos están desarrollados en ellos.

- *Sensor*

Contiene atributos que representan las características de un sensor, desde su conectividad hasta sus atributos de salida. Contiene una referencia para indicar a que *Gateway* está conectado y la latencia de conexión entre ellos. Además, define los argumentos de salida del sensor y la distribución que siguen los *tuples* (unidades mínimas de comunicación) al llegar a su destino y volver.

- *Tuple*

Son la unidad fundamental de comunicación entre entidades en el *Fog*. Son una herencia de la clase *Cloudlet* de *CloudSim*. Están caracterizados por el tipo de fuente y módulos destino. Los atributos de esta clase especifican los requerimientos de procesamiento (definidos como Millones de Instrucciones, MI) y la longitud del dato.

- *Actuator*

Esta clase modela un activador definiendo sus propiedades de conexión. Un atributo de esta clase identifica el *Gateway* por el que este está conectado y su latencia.

- *Application*

Una aplicación está modelada como un grafo directo, donde los vértices representan los módulos que realizan el procesamiento y los bordes determinan las dependencias entre ellos. Estas entidades quedan definidas por las clases: *AppModule*, *AppEdge* y *AppLoop*, que serán detalladas a continuación.

- *AppModule*

Representan los elementos de procesamiento de las aplicaciones *Fog*. Esta implementada como una extensión de *PowerVm* en *CloudSim*. Para cada *tuple* que llega, una instancia de *AppModule* lo procesa y genera los *tuples* de salida que serán enviados a los siguientes módulos del grafo.

- *AppEdge*

Denota una dependencia de datos entre un par de módulos y representa un borde del modelo de aplicación. Cada borde es caracterizado por el tipo de *tuple*, el cual queda especificado en el atributo *tupleType*.

- *AppLoop*

Es una clase adicional que tiene como propósito crear bucles de interés para el usuario. Es de especial interés para medir la latencia de extremo a extremo. Fundamentalmente se trata de una lista de módulos empezando desde el origen del bucle hasta el módulo donde el bucle termina.

- *Power Monitoring Service*

Además del procesamiento básico de las funcionalidades de *tuples*, hay servicios simulados disponibles. Cada dispositivo *Fog* tiene asociado un modelo de potencia. El método *executeTuple*() en la clase *FogDevice* contiene la lógica necesaria para el procesamiento de los *tuples* donde el modelo de potencia se usa para actualizar la consumición energética basada en los cambios en utilización de recursos.

4.2.5.3 Estrategias de planificación implementadas por defecto

El simulador contiene implementadas dos estrategias de planificación

- Ubicación en el *Cloud*

  Esta estrategia está basada en el modelo tradicional del *Cloud*, donde las implementaciones de todos los módulos se ejecutan en data centers. El bucle *sense-process-actuate* en dichas aplicaciones los sensores

están transmitiendo datos al *Cloud* donde son procesados y los activadores son informados en caso de requerir alguna acción por su parte.

- Ubicación en los bordes de la red (a partir de ahora, *Edgeward*)
  Favorece el desarrollo de los módulos de la aplicación cerca de los bordes de la red. Sin embargo, los dispositivos que están en estos bordes – como routers y puntos de acceso – pueden no tener capacidad computacional suficiente para albergar todas las operaciones de la aplicación. En esta situación, se envían estos procesos a dispositivos cercanos al *Cloud* e intenta ubicarlos en dispositivos alternativos. En el algoritmo adjunto en la figura 6 se demuestra el intercambio entre el *Fog* y la nube emplazando los módulos cerca del borde la red y el *Cloud*.

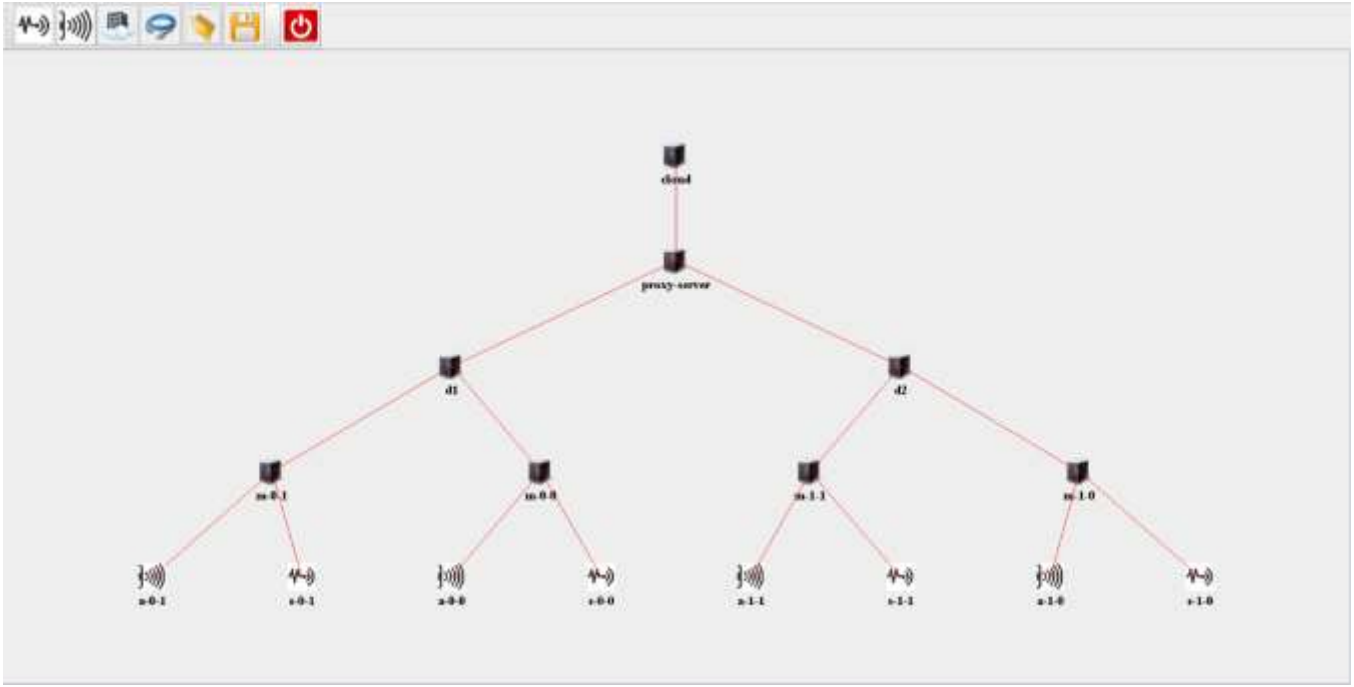
Algorithm 1: Edge-ward module placement

```
for p ∈ PATHS do Across all paths
    placeList := {};
    for Fog device d ∈ p do leaf-to-root traversal
        for module w ∈ app do
            if all predecessors of w are placed then
                add w to placeList ;
            end
        end
        for module θ ∈ placeList do
            if θ is already placed on device f ∈ p then
                Merge θ with its upstream instance ;
                f :=device holding merged instance ;
                while CPU_θ^req ≥ CPU_f^avail do
                    f := parent(f) ;
                end
                Place θ on device f ;
            end
            else if CPU_θ^req ≤ CPU_d^avail then
                Place θ on device d ;
            end
        end
    end
end
```

Figura 5: Pseudocódigo del emplazamiento Edgeward. [10]

4.2.5.4 Interfaz gráfica

Con el fin de facilitar la descripción de topologías físicas *iFogSim* tiene la clase *FogGUI*. Permite al usuario dibujar elementos físicos como dispositivos *Fog*, sensores, activadores y links de conexión. En ellas se pueden definir todos los parámetros utilizados



para la simulación; latencia entre links, MIPS, RAM de los dispositivos, entre otros. Las topologías creadas pueden guardarse y cargarse convirtiéndolas en formato JSON. No es el único método para construir las topologías, también se pueden programar. En la figura 7 se puede ver un ejemplo de una topología construida con GUI.

Con el fin de poder visualizar la configuración que se usará para este estudio, se ha creado la topología *vr_game_topo*, la cual muestra el caso de tener dos routers y dos Smartphone por cada uno de ellos. En el nivel más bajo se pueden ver los sensores, los cuales, siguiendo la nomenclatura del simulador son *actuators*.

Figura 6: Topología vr_game_topo

4.2.6 Caso de uso

El caso de uso propuesto para el proyecto es un juego online sensible a la latencia llamado *EEG Tractor Beam* [11] en el que se hace uso de interacción cerebro-ordenador. Para jugar al juego, cada jugador tiene que llevar unos auriculares *MINDO-4S* inalámbricos conectados a su smartphone. El juego se ejecuta en una aplicación Android en el smartphone de cada jugador. La aplicación procesa en tiempo real las señales EEG recogidas por los auriculares y calcula el estado cerebral del jugador.

En la pantalla de la aplicación, el juego muestra a todos los jugadores en círculo rodeando un objeto, el cual es el objetivo. Cada jugador tiene que ejercer una fuerza atractiva hacia el objetivo, que será medida como su nivel de concentración (se estima haciendo la media con la densidad espectral de potencia del electroencefalograma en las bandas alfa, beta y teta del jugador). Para ganar el juego, los jugadores tienen que intentar

atraer el objetivo hacia ellos mismos mientras evitan las probabilidades de que otros jugadores se hagan con él.

El procesamiento a tiempo real requiere que la aplicación esté alojada muy cerca de la fuente de datos – preferiblemente en el mismo smartphone. Sin embargo, ese despliegue no permitiría una cobertura global – que típicamente requiere un despliegue en la nube. Esta mezcla de objetivos contradictorios hace que esta aplicación sea un caso típico de *Fog Computing*.

4.2.6.1 Modelo de aplicación

En la figura 8 se presenta un grafo en el que se muestra las dependencias de los módulos de la *aplicación EEG Tractor Beam.* Consiste en 3 módulos principales que realizan el procesamiento de las tareas – *Client Concentration Calculator* and *Coordinator.* Estos módulos se han desarrollado usando la clase *AppModule*. En la figura se ven las dependencias entre estos, que han sido producto del desarrollo de la clase *AppEdge*. Finalmente se ha programado un bucle que nos muestra la latencia usando la clase *AppLoop*. La aplicación se alimenta de señales EEG recogidas por sensores y mostradas por activadores a tiempo real al jugador. Las funciones arriba mencionadas son*:*

- *Client*

  El módulo cliente interviene con el sensor y recibe las señales EEG sin procesar. Comprueba los valores de la señal en busca de discrepancias y descarta cualquier lectura inconsistente. Si esta señal es consistente, envía el valor al módulo *Concentration Calculator* para obtener el nivel de concentración del usuario. Cuando recibe este valor, lo muestra enviándolo al activador DISPLAY.

- *Concentration calculator*

  Este módulo es responsable de determinar el estado cerebral del usuario y calcular su nivel de concentración desde la señal que le envía el módulo *Client*. Después, le informa el valor medido para que el *display* pueda actualizar el valor.

- *Coordinator*

  Trabaja a nivel global y coordina el juego entre los distintos jugadores que pueden estar distribuidos geográficamente. Envía constantemente el estado del juego al módulo *client* de todos los jugadores en línea.

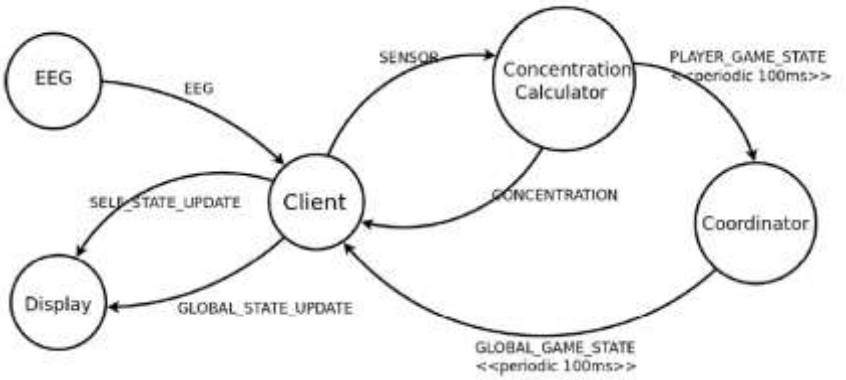


Figura 7: Dependencia entre módulos en EEG Tractor Beam. [12]

4.2.7 Procedimientos

En [10], documento que se ha usado como guía e inspiración en este proyecto, se presenta el caso de uso de *EEG Tractor Beam Game*, por lo que, para empezar a entender el funcionamiento del simulador se propone simular y llegar a los mismos resultados que dicho documento, cuya simulación la denominaremos como "caso base". Ya que en *iFogSim* viene implementado la topología del juego con una planificación en la nube, se decide hacer una comparativa con la planificación *Edgewards*, objetivo de estudio de este proyecto. Para así resaltar lo ventajoso que podría ser apoyarse en esta tecnología.

Después se implementará una amplia variación de topologías, en la que se irá variando las capacidades de cada componente para ver cómo estas características influyen sobre la latencia y otros objetivos (costes de ejecución en la nube, de la red o tiempo de ejecución), además se harán comparativas con la ejecución en el *Cloud*, para reafirmar la importancia de investigar sobre *Fog Computing*. Esto sería la primera parte del proyecto.

En la segunda parte, se estudiarán diferentes estrategias de planificación con el objetivo de implementar una y hacer comparativas sobre su rendimiento. Para ello ha sido necesario un estudio y búsqueda exhaustivas de *White Papers* en la red, tarea que no ha sido fácil debida a la escasez de información sobre este tema.

4.2.7.1 Configuración de parámetros para el caso base

En esta primera parte, como hemos dicho anteriormente, se tiene como objetivo la simulación del caso base y hacer una comparativa con la misma configuración en el caso *Cloud*. Para ello, usamos de referencia los valores que se especifican en [1], los cuales son:

TIPO DE TUPLE	CPU (MIPS)	N/W LENGTH
EEG	2000 (A) / 2500 (B)	500
_SENSOR	3500	500
PLAYER_GAME_STATE	1000	1000
CONCENTRATION	14	500
GLOBAL_GAME_STATE	1000	1000
GLOBAL_GAME_UPDATE	1000	500
SELF_STATE_UPDATE	1000	500

Tabla 1: Descripción de los bordes intermodulares.

TIPO DE DISPOSITIVO	CPU (GHz)	RAM (GB)	POTENCIA (W)
Cloud VM	3.0	4	107.339(M) 83.433(I)
WiFi Gateway	3.0	4	107.339(M) 83.433(I)
Smartphone	1.6	1	87.53(M) 52.44(I)
ISP Gateway	3.0	4	107.339(M) 83.433(I)

Tabla 2: Configuración de los dispositivos Fog.

Cada auricular está conectado a un smartphone vía Bluetooth. Estos, tienen acceso a internet a través de routers WiFi que están conectados a unas pasarelas ISP.

En las simulaciones, se tiene en cuenta dos tipos de auriculares con diferentes medias de tiempo y longitudes de tuples:

AURICULARES	TUPLE CPU	MEDIA DE TIEMPO INTER-ARRIVAL
A	2000 Millones de Instrucciones	10 milisegundos
B	2500 Millones de Instrucciones	5 milisegundos

Tabla 3: Configuración de los sensores.

A la vez, con el propósito de evaluar el rendimiento de *iFogSim*, se va variando el tamaño de las topologías; incrementando el número de routers mientras se mantiene el número de smartphones conectados a estos – Config 1, Config 2, Config 3, Config 4 y Config 5 – teniendo 1, 2, 4, 8 y 16 routers respectivamente.

Para la simulación del caso *Edgewards* se ha tenido que modificar la clase *VRGame,* la cual es la clase en la que viene implementado el juego del que se quiere estudiar. Ya que por defecto viene implementado con una planificación en la nube, se han realizado unas modificaciones con el fin de que trabaje con el caso *Fog.* A grandes rasgos consisten en el objeto *controller* – a través del cual podemos invocar los diferentes tipos de planificación – llame en este caso a la planificación *Edgeward* y pasarle los argumentos que requiere en su caso; *fogDevices, application y moduleMapping.* Para el caso de uso de planificación *Cloud* no es necesario este paso pues es la planificación por defecto.

De una forma general, para hacer la simulación de un caso de uso, se puede resumir a muy alto nivel en los siguientes pasos:

- Primero, los componentes físicos tienen que ser creados con la configuración deseada. Los parámetros configurables, entre otros son: RAM, capacidad de procesamiento (en millones de instrucciones por segundo, MIPS), coste de procesamiento por millones de instrucciones, ancho de banda de subida y de bajada, potencia cuando el dispositivo está ocupado (busy power) o cuando está libre (idle power) y nivel jerárquico – Los dispositivos que están en el nivel jerárquico más bajo son los que están asociados con los dispositivos IoT. Para fijar el intervalo de tiempo en el que los sensores capturan datos tenemos que inicializar el objeto *transmitDistribution*. Además, cuando se crean los sensores y activadores se requiere el identificador de la aplicación y del *bróker*.

- Después, se crean los componentes lógicos como *AppModule*, *AppEdge* y *AppLoop*. Mientras se crean los *AppModules*, sus configuraciones como el tipo de tuple, su dirección y CPU, vienen especificadas por objetos de *AppEdge*.

- Finalmente, los componentes de gestión (*ModuleMapping*) son inicializados para definir diferentes planificadores y políticas de reubicación de módulos. Los usuarios pueden considerar el total de

energía consumida, la latencia, el uso de la red, costes operacionales extendiendo la clase *ModuleMapping*. Basándose en la información de *AppEdges*, los requerimientos de *AppModule* deben de estar alineados con la especificación de cada tuple correspondiente y abastecerse con los recursos disponibles. Una vez que se han mapeado los dispositivos Fog, la información de los componentes físicos de la topología son enviados al objeto *Controller*, el cual, envía el sistema completo a *CloudSim* para empezar la simulación.

4.2.7.2 Ampliación de topologías

Para hacer un estudio más exhaustivo del funcionamiento y bondades de *Fog Computing*, se propone examinar el rendimiento de esta tecnología bajo diferentes condiciones. Con este fin, se han variado las capacidades de los distintos dispositivos que conforman la red para el caso de uso que nos ocupa, comprobando de esta manera como las limitaciones del hardware influyen sobre la latencia, consumo de recursos de red y *Cloud* y tiempo de ejecución de la aplicación.

Se irán variando las capacidades de los dispositivos y el número de estos que intervienen (en el caso de smartphones y routers) de forma incremental, con el objetivo visibilizar en pequeños pasos cómo la escalabilidad de la aplicación se desborda o mantiene.

Para hacer este estudio y asegurarnos de que se han cubierto todos los casos de estudios, se procedido de la siguiente forma:

Primero, se han enumerado los componentes que conforman la red y se quieren variar: número de jugadores (smartphones), número de routers WiFi, características computacionales (MIPS y RAM) del *Cloud*, de las pasarelas ISP, de las pasarelas y de los smartphones.

Si, por ejemplo, se quiere variar las características del *Cloud*, para graficar la repercusión de un aumento de capacidades de este de forma gradual, se decidió buscar cuales eran las características de unos servidores *Cloud* potentes, y, una vez obtenidas estas, variar desde las que están definidas para el caso base, hasta estas capacidades máximas. Aunque los valores introducidos de RAM puedan no tener sentido de esta forma (95 Gbytes), no es intención de este estudio hacerlo con valores comerciales. Lo que se quiere ver es cómo se comporta la red cuándo se incrementan de forma proporcional el hardware. Todas las características que se irán variando se harán en 5 configuraciones, con el propósito de ver su evolución progresivamente.

A continuación, veremos la justificación de los valores elegidos

A) Número de jugadores

También puede ser visto como el número de Smartphones en juego. Este, lo variaremos conforme se hace en la publicación: 1, 2, 4, 8, y 16 jugadores por cada router – Config 1, config 2, config 3, config 4, config 5 respectivamente – De esta forma, podemos ver como se colapsa la red cuando el número de jugadores se va incrementando.

B) Número de routers

Se incrementa a la misma razón que el número de jugadores. Hay que tener en cuenta en los análisis posteriores, que, por cada router, hay un número de jugadores asignado (el cual asignaremos nosotros) por lo que, cuando ambos valores son máximos, se puede llegar a alcanzar hasta 256 jugadores simultáneamente.

C) Características del *Cloud*

Las características que hemos considerado para nuestro estudio son los MIPS (Millions of Instructions per Second) and RAM (en Mbytes)

Los valores iniciales sobre los que se han trabajado son 44800 MIPS, y 40000 de RAM. Para los cuales, se han escogido los valores aplicados en las que usaron los desarrolladores de dicha aplicación [10]. Cómo las capacidades de hardware se han desarrollado desde entonces, se ha decidido aumentar estas a razones constantes.

Para decidir el límite superior de las capacidades de nuestros servidores *Cloud*, se ha hecho una búsqueda en la red para averiguar hasta qué capacidades de hardware llegan algunos de los servidores ya existentes, por lo que, para las capacidades máximas se han escogido la de unos servidores potentes.

De esta forma, las capacidades máximas se han establecido en 89600 MIPS y 262144 Mbytes de RAM.

Para ir variándolo en 5 configuraciones como se ha hecho con todos los demás dispositivos de red, se ha decidido variar desde las mínimas, aumentando a la misma razón hasta llegar a las capacidades máximas, con la intención de ver la variación de las características de la red y consumo poco a poco (aún que eso signifique asignar valores de RAM no comerciales en el mercado).

Esta forma de proceder será igual para la elaboración de las distintas configuraciones del ISP, routers WiFi y smartphones.

Por lo que las diferentes configuraciones para los servidores *Cloud* serían:

D) Características del ISP (Fog Server en la notación del simulador)

Para las cinco variaciones propuestas, las características de su hardware quedarían resumidas en la siguiente tabla:

Configuración	RAM (Mbytes)	MIPS (Millions of Instructions Per Second)
Config 1	4000	2800
Config 2	19384	3500
Config 3	34768	4200
Config 4	50152	4900
Config 5	65536	5600

Tabla 5: Mejora del hardware de las pasarelas ISP.

E) Características de los router WiFi (Dept)

Estas son las configuraciones propuestas para los router WiFi:

Configuración	RAM (Mbytes)	MIPS (Millions of Instructions Per Second)
Config 1	40000	44800
Config 2	95536	56000
Config 3	151072	67200
Config 4	206608	78400
Config 5	262144	89600

Tabla 4: Características del Cloud.

F) Características de los smartphones

Por último, estás son las características de hardware de los smartphones:

Configuración	RAM (Mbytes)	MIPS (Millions of Instructions Per Second)
Config 1	1000	2800
Config 2	1750	1250
Config 3	2500	1500
Config 4	3250	1750
Config 5	4096	2000

Tabla 7: Mejora del hardware de los smartphones.

Al disponer de tantos parámetros para su estudio, se quiere ver su influencia individual sobre otros, así como su influencia conjunta. Para ello, se hace una tabla en la que se especifica que dispositivo variará y a qué hoja de Excel corresponde su análisis.

Para una mayor clarificación, se adjunta un extracto de dicha tabla:

variar numero de jugadores	mantener número de switches	mantener caract. *CLOUD*	mantener caract. PROXY SERVER	mantener caract. DEPT	mantener caract. terminales	Result1
variar numero de jugadores	mantener número de switches	mantener caract. *CLOUD*	mantener caract. PROXY SERVER	mantener caract. DEPT	cambiar caract. terminales	Result2
variar numero de jugadores	mantener número de switches	mantener caract. *CLOUD*	mantener caract. PROXY SERVER	cambiar caract. DEPT	mantener caract. terminales	Result3
variar numero de jugadores	mantener número de switches	mantener caract. *CLOUD*	mantener caract. PROXY SERVER	cambiar caract. DEPT	cambiar caract. terminales	Result4
variar numero de jugadores	mantener número de switches	mantener caract. *CLOUD*	cambiar caract. PROXY SERVER	mantener caract. DEPT	mantener caract. terminales	Result5
variar numero de jugadores	mantener número de switches	mantener caract. *CLOUD*	cambiar caract. PROXY SERVER	mantener caract. DEPT	cambiar caract. terminales	Result6

Tabla 8: Extracto tabla de seguimiento de topologías.

En rojo queda marcado el parámetro que se quiere variar, en negro, se dejaran con las mínimas características de hardware.

Además, para cada análisis se ha tenido en cuenta los dos tipos de auriculares que recogen las señales EEG del usuario que tuvieron en cuenta en el desarrollo del juego. Las características del de estos auriculares vienen resumidos en la tabla 3.

Como si de una tabla de verdad se tratara, teniendo 6 distintos parámetros que se quieren estudiar, para estudiar todas las posibles combinaciones serán necesarios 2^6 = 64 análisis. Para cada uno de ellos, se ha analizado como esa combinación de variación de parámetros de los elementos de red que lo conforma influye sobre la latencia, el tiempo de simulación, el consumo de recursos en servidores *Cloud* y consumo de la red. Todo ello para las estrategias *Edgeward* y *Cloud*. En esta memoria solo se detallarán los análisis más interesantes, ya que es un análisis extenso. En cualquier caso, se incluirán en los anexos todos los demás resultados.

Más adelante, en la sección de discusión se tratarán los resultados más interesantes y en la sección de anexos, se hablará de cómo están estructuradas el resto de análisis para una más fácil comprensión de los resultados analizados.

4.3 Estrategias de planificación

En el presente apartado se procede a detallar la implementación de las estrategias de planificación elegidas, y, posteriormente, en el apartado de discusión, se compararán entre ellas y las ya implementadas por defecto en el simulador para visualizar cómo afecta la estrategia de planificación sobre la latencia y otros parámetros.

Conviene mencionar que, tras el meticuloso estudio de las diversas estrategias propuestas por diferentes desarrolladores, se ha observado que no hay literatura científica acerca de planificar las tareas intermodulares. Al tratarse el presente trabajo de tipo teórico o experimental, se decide estudiar como influiría el dotar de una planificación diferente a estas tareas. Con este objetivo, se han implementado dos planificaciones diferentes; *Priority Scheduling* y *Min Min Scheduling*.

a) *Priority Scheduling*

La implementación de esta planificación se ha basado en [13]. Se ha escogido porque la planificación de tareas basada en prioridad es bastante efectiva ya que satisface los requerimientos de los usuarios cumpliendo con QoS, minimizando el tiempo que las tareas están en cola para ser procesadas. Por lo

que, para este tipo de planificación nos estamos basando en un nivel más bajo; en las tareas que componen los módulos. Para la planificación de módulos se ha dejado el algoritmo por defecto *Edgeward*, ya que es bastante eficiente en términos de latencia.

El funcionamiento básico consiste en procesar las peticiones del módulo *Client* recibidas, las cuales pueden tener diferentes *deadlines* que necesitan ser cumplidas para la correcta ejecución de la tarea. Basado en este requerimiento, se crea un módulo encargado de asignar distintas prioridades.

Antes de presentar el pseudocódigo del algoritmo conviene tener en cuenta algunos de los parámetros que se han usado para el modelo:

- Niveles de prioridad: high (prioridad = 1), medium (prioridad = 2), low (prioridad = 3) y sus colas correspondientes: highQ, mediumQ y lowQ.

- Tiempo estimado de ejecución de la tarea i: estimatedServiceTime

- Retraso: Delay = (DeadLine + CT) (1
  Siendo CT el tiempo actual.

Una vez conocida la nomenclatura utilizada, queda más claro el pseudocódigo que define el algoritmo implementado:

```
1. //Delay máximo permitido no superior al tiempo estimado de servicio
   If delay = = estimatedServiceTime
        Return high;
2. // Delay máximo permitido entre los umbrales T1 Y T2
   Else If T1 < delay <=T2
        If priority == 1
             Return high
        Else if priority == 2
             Return medium
        Else if priority == 3
             Return low
3. // Delay máximo permitido mayor que el umbral T2
   Else delay > T2
        If priority == 1
             If highQ not full
                  Return high
             Else
                  Return medium
        Else if priority == 2
             If mediumQ not full
                  Return medium
             Else
                  Return low
        Else if priority == 3
             Return low
```

Las variables T1 y T2 en el algoritmo son periodos de tiempo significativos. Se usan para ordenar las tareas pendientes con respecto a su *deadline*. Mientras se decide a que cola se debería añadir la siguiente tarea, T1 siendo menor que T2, todas las tareas que tengan la menor *deadline* serán añadidas a la lista de prioridad máxima, como se puede observar en el punto 2. Todas las tareas que tengan una *deadline* más tardía se añadirán a otras colas, de esta forma, las de la lista de prioridad alta serán ejecutadas antes.

En el algoritmo, el *delay* máximo permitido se compara con *estimatedServiceTime* y contra dos valores umbrales. En el primer paso, *delay* es comparado con *estimatedServiceTime*. Si es menor o igual a este la tarea será asignada como prioridad alta. En el siguiente paso se compara con los valores umbrales T1 y T2, si se haya entre los dos, el nivel de prioridad asignado será acorde con el etiquetado anteriormente de la tarea. Finalmente, en el tercer paso, cuando el retraso es superior a T2 el nivel de prioridad corresponderá al asignado originalmente siempre y cuando la lista no esté llena, si lo está se asignará una prioridad menor. Esto se permite porque al tener la tarea un *delay* permitido mayor se puede categorizar de una prioridad menor cuando la cola de las tareas de prioridades altas está completa.

Además, se han tenido que modificar todas las funciones que manejaban las tareas del simulador, incluyendo en cada una de ellas las listas anteriormente creadas y el manejo de estas.

Para el desarrollo de dicha estrategia, se ha creado una clase nueva llamada *CloudletSchedulerPriority* en la que va toda la lógica anteriormente explicada, se ha modificado *tupleScheduler*, la cual ahora heredará de *CloudletSchedulerPriority* en lugar de *CloudletSchedulerTimeShared*. Además, la clase *fogDevice* también ha sido necesaria una pequeña modificación en el método *updateAllocatedMips* para gestionar los tuples y por último, se ha retocado la clase que modela el juego que estamos simulando; *VRGame*.

b) MinMin Scheduling

   Este algoritmo consiste en asignar las tareas más cortas (hablando en términos de tiempos de ejecución) a los dispositivos a los que les lleve menos tiempo en ejecutar la tarea. Este método optimiza el tiempo de ejecución de las tareas. [14].

   Se trata de un algoritmo sencillo pero rápido en procesamiento de tareas cuando esta es pequeña comparada con la más larga. Sin embargo, si el número de tareas largas es mayor que el de las tareas cortas resulta en un uso pobre de los recursos disponibles y un mayor tiempo de ejecución.

Para la implementación de esta estrategia de planificación, se ha creado una nueva clase llamada *MinminScheduling*, en la que se ha implementado la siguiente lógica de programación:

// Fase 1: Calculo del tiempo de ejecución de las tareas disponibles

1. **Para todas** las tareas (ti) de la lista L
2. **Para todos** los recursos (Rj)
3. Calcular el tiempo de ejecución: CTij = ETij + rj
4. **Fin** del bucle del paso 2
5. **Fin** del bucle del paso 1

// Fase 2: Asignar las tareas ti, con tiempo mínimo de ejecución a recursos con mínimo tiempo de esperado de ejecución de estas

6. **Para cada** tarea encontrar la que tenga menor tiempo de ejecución y recurso con el mismo
7. Asignar la tarea ti al recurso Rj que tenga mínimo tiempo de ejecución
8. Eliminar la tarea ti de la lista L
9. Actualizar el recurso Rj y tiempo para volver a estar disponible ri.
10. Actualizar el tiempo de ejecución de todas las tareas
11. Repetir pasos desde 6 al 10 hasta que todas las tareas estén mapeadas
12. **Fin** del bucle del paso 6

Se han modificado las clases *tupleScheduler* para que herede los atributos de esta nueva clase que se ha creado, *cloudSimTags* para añadir nuevas etiquetas que eran necesarias para el algoritmo:

Tag	Valor
VM_STATUS_READY	BASE + 49
VM_STATUS_BUSY	BASE + 50
VM_STATUS_IDLE	BASE + 51

Figura 8: Etiquetas nuevas.

Las etiquetas indican que una acción tiene que ser tomada o informada con un elemento de *CloudSim*. Para este caso, se necesitaban las anteriores etiquetas para informar del estado de las máquinas virtuales.

La creación de estas se ha realizado siguiendo las ya existentes en dicha hoja, con un valor base y los siguientes números no ocupados por otras acciones a realizar.

Además, se ha creado una clase adicional llamada *VmInstance* para tener una instancia de máquina virtual en la que se asigna que está libre.

5 RESULTADOS Y DISCUSIÓN

En este apartado se mostrarán y discutirán los resultados de las simulaciones previamente mencionadas.

5.1 Ampliación de topologías

Como previamente se había establecido, primero se quiere experimentar con el aumento de las características del hardware que compone la red en nuestro caso de uso, viendo su influencia sobre la latencia u otros objetivos. Aprovechando este análisis, compararemos los resultados de las simulaciones con otras de *Cloud Computing*, ya que es el paradigma de computación predilecto actualmente.

5.1.1 Resultados base

Como objetivo inicial se propuso llegar a los resultados de las simulaciones en las que se presentaba el simulador en el que se está basando este estudio [1]. Se considera que los resultados de este caso de uso son necesarios para ver el impacto de estas estrategias de planificación en términos de latencia o carga.

Para los resultados que se expondrán a continuación se dejará el número de jugadores por router WiFi constante en 4 y se irán variando el número de routers. Las características de los diferentes componentes se especificaron en el capítulo anterior.

5.1.1.1 Latencia en el bucle de control

El simulador cuenta con una clase para definir bucles de control para la evaluación de latencia en una serie de módulos a definir por el desarrollador. En este caso, se define el bucle para el conjunto de módulos más críticos en términos de latencia: aquellos en los que intervienen en el procesamiento de las señales EEG del usuario transformándolas en el estado actual del jugador en su smartphone. Disponer de latencia en esta serie de módulos dañaría la experiencia de juego ya que son los que intervienen directamente con el jugador. Los módulos que componen este bucle son: *EEG -> CLIENT -> CONCENTRATION CALCULATOR -> CLIENT -> DISPLAY.*

Se hacen pruebas con ambos tipos de auriculares, los cuales ya se detallaron sus especificaciones en el capítulo anterior. Las comparaciones y gráficas se ha decidido

hacerlas solo en uno de los casos, con el headset B, ya que se dan mayores latencias con estos.

En la tabla se puede observar como la latencia se reduce dramáticamente en el caso de la estrategia *Edgeward* donde los dispositivos *Fog* también son usados para la computación.

Configuración	*Edgeward* Headset A (ms)	*Edgeward* Headset B (ms)	*Cloud* Headset A (ms)	*Cloud* Headset B (ms)	Mejora (%)
Config 1	17.01	28.18	230.66	225.16	87.48
Config 2	18.44	28.11	230.53	225.28	87.52
Config 3	19.04	29.55	230.62	225.56	86.90
Config 4	19.18	29.40	889.94	2954.07	99.00
Config 5	18.64	28.75	4053.58	4569.87	99.37

Tabla 9: Latencia media en el bucle de control. Caso base.

En la tabla 9, la columna de mejora se realiza con la comparación de los peores escenarios, es decir, con el caso del headset B, ya que es el que resulta en una mayor latencia. En este caso es una comparativa entre *Cloud* y *Edge*.

En el gráfico de la figura 8 se observa que la latencia en el caso *Edge* es casi inapreciable en comparación con la del caso *Cloud*.

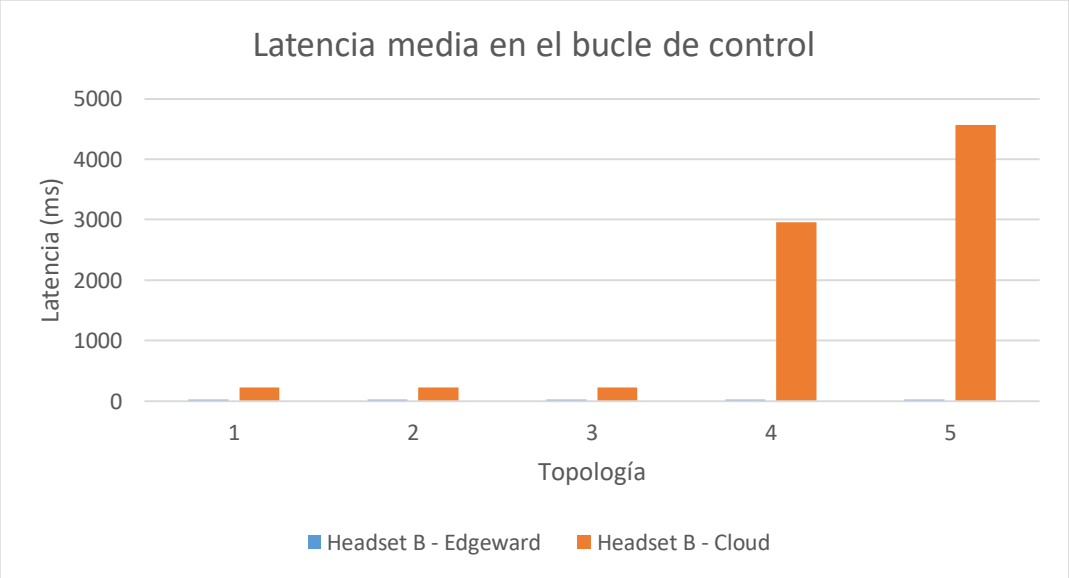


Figura 9: Gráfico de latencia media en el bucle de control. Caso base.

5.1.1.2 Uso de la red

La tabla 10 ilustra el consumo total de recursos de red para el juego *EEG Tractor Beam*. Incrementar el número de dispositivos conectados a la aplicación incrementa la carga de la red cuando solo se usa el *Cloud* para el procesado. En cambio, cuando se tienen en consideración los dispositivos *Fog* para el procesamiento, este consumo decrece significativamente.

En la tabla 10 se pueden distinguir los valores simulados para ambos auriculares, notándose una mejora cuando se utiliza el headset A en ambas estrategias de planificación.

Configuración	***Edgeward* Headset A (Kbytes)**	***Edgeward* Headset B (Kbytes)**	***Cloud* Headset A (Kbytes)**	***Cloud* Headset B (Kbytes)**	**Mejora (%)**
Config 1	3533.33	6733.33	39906.60	77300.30	91.29
Config 2	6400	12853.33	82923.90	161716.20	92.05
Config 3	12133.33	24476.67	182261.60	357285.30	93.15
Config 4	23600	49040	412076.80	695120.60	92.95
Config 5	46533.33	96780	706903.10	1274013.20	92.40

Tabla 10: Uso de la red. Caso base.

En la figura 9 se puede apreciar mejor visualmente las diferencias entre ambas estrategias:

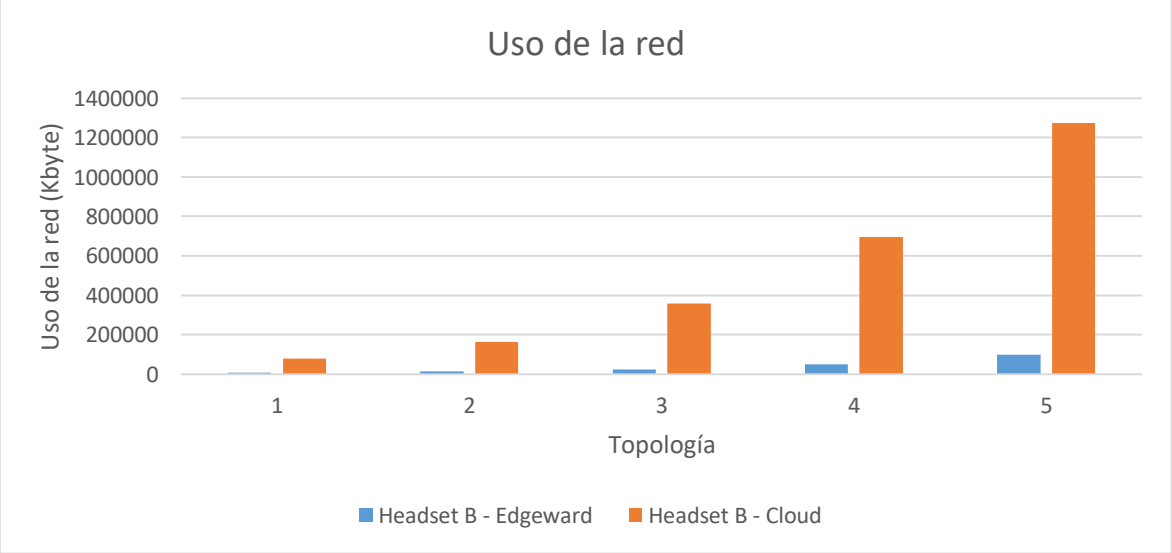


Figura 10: Uso de la red. Caso base.

Este resultado también puede ser interpretado como la escalabilidad de las aplicaciones basadas en dispositivos *Fog*. Un crecimiento incontrolado del consumo de la red en estrategias *Cloud* pueden llevar a situaciones de cuello de botella, en las que la red se satura. En cambio, se puede evitar llegar a esta situación si se utiliza una estrategia basada en *Fog Computing*.

5.1.1.3 Tiempo de ejecución de la simulación

El gráfico de la figura 10 muestra cómo se incrementa el tiempo de ejecución de la simulación cuando el número de dispositivos y la tasa de transmisión aumentan. Para la planificación *Edgeward* aun cuando el tiempo de ejecución alcanza su valor máximo sigue siendo un valor asumible.

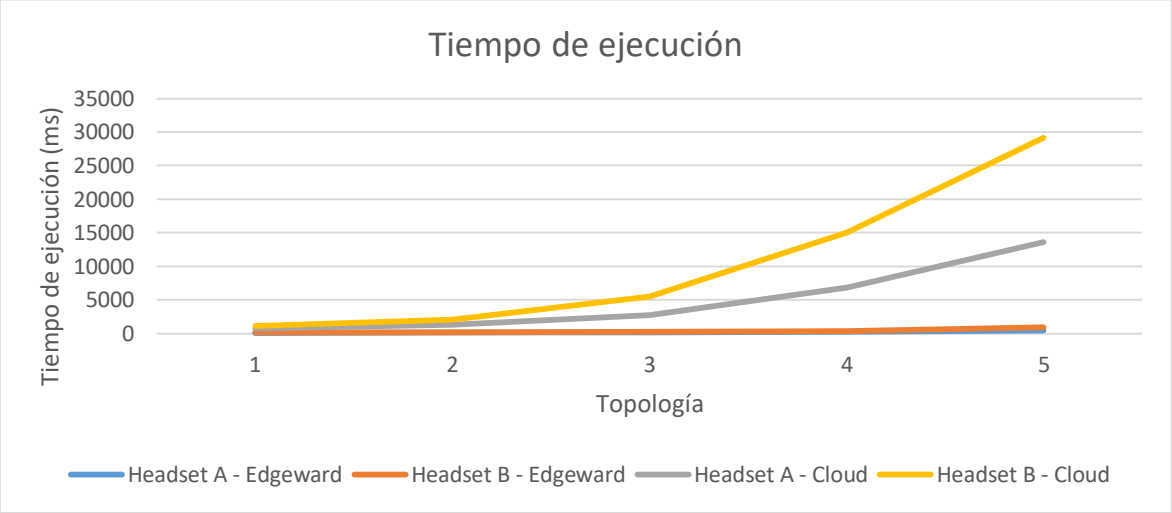


Figura 11: Tiempo de ejecución. Caso base.

5.1.1.4 Coste de ejecución en el Cloud

El consumo de recursos *Cloud* en el caso de planificación en la nube es claramente la totalidad de los recursos, por lo que alcanzará el coste máximo entre los dos tipos de planificación a comparar. Sin embargo, en el caso *Edgewards*, informa de cuántos recursos se necesitan utilizar debido a que el resto de dispositivos *Fog* no tienen suficientes para la computación de las tareas, mandando entonces estas a la nube para su ejecución.

Configuración	*Edgeward* Headset B	*Cloud* Headset B
Config 1	6976.80	541599.26
Config 2	7086.40	1119157.87
Config 3	7305.60	1947776.11
Config 4	7744.00	3062835.24
Config 5	8620.80	4436831.90

Tabla 11: Coste de ejecución en la nube. Caso base.

En la tabla 11 se observa que el aumento de jugadores conlleva un mayor uso de los recursos *Cloud*, pues hay más que procesar.

5.1.2 Mejoras de hardware

Se estudiará la influencia individual tras mejorar cada componente de la red definida para nuestro caso de uso. Para ello se mantendrá invariante el número de jugadores por routers (4 en este caso) y el número de routers totales (4 también).

5.1.2.1 Mejora de hardware en los smartphones

Como se ha detallado anteriormente, se dejarán invariantes el número de jugadores por cada router y el número de routers totales, haciendo un total de 16 jugadores simultáneamente.

Para estudiar como una mejora de las características de los smartphones sobre la latencia final y otros parámetros se hace de la forma especificada en capítulos anteriores; en cinco configuraciones. En este caso de uso se hará el estudio con ambos auriculares. A y B denota de cuál de ellos se trata. Solo se detallarán los estudios en los que se vea algo que destacar, pues puede darse el caso que una mejora del hardware no suponga ningún cambio en el coste de ejecución en la nube, por ejemplo.

- Estudio de la latencia

Configuración	**Headset A- *Edgeward* (ms)**	**Headset B- *Edgeward* *(ms)***	**Headset A- *Cloud* *(ms)***	**Headset B- *Cloud* *(ms)***	**Mejora (%)**
Config 1	18.56	30.17	230.60	225.56	86.62
Config 2	17.43	36.17	230.61	225.30	83.95
Config 3	13.50	12.67	230.52	225.11	94.37
Config 4	13.02	12.06	229.96	224.90	94.64
Config 5	12.93	12.36	230.03	224.86	94.50
Comparación (config 1 vs config 5) %	30.36	59.02	0.25	0.31	

Tabla 12: Latencia media en el bucle de control. Mejora de hardware smartphones.

En la tabla 12 se muestra el porcentaje de mejora entre tecnologías (*Edgeward* y *Cloud*) siempre para los auriculares B, los cuales son el caso de mayor latencia.

Además, en la fila “Comparación” se muestra el porcentaje de mejora entre las peores características de hardware (configuración 1) y las mejores (configuración 5). De esta forma se aprecia la reducción de la latencia que supone aumentar las capacidades de los terminales.

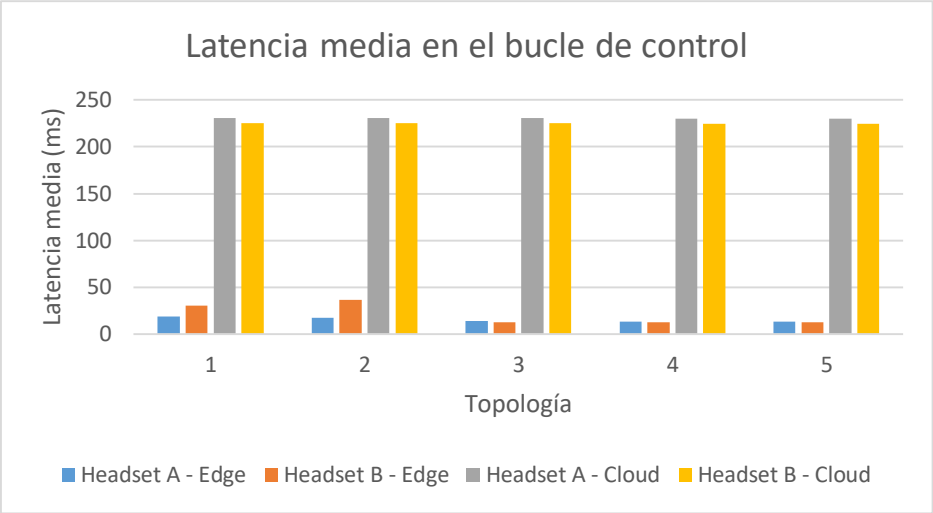


Figura 12: Latencia media en el bucle de control. Mejora de hardware smartphones.

En el caso de *Cloud Computing*, una mejora de las capacidades de estos terminales no resulta en una mejora apreciable en la latencia final.

Sin embargo, en el caso de *Edge Computing* se puede ver hasta una mejora del 60%.

- Estudio del uso de la red

Configuración	**Headset A- *Edgeward* (Kbytes)**	**Headset B- *Edgeward (Kbytes)***	**Headset A- *Cloud (Kbytes)***	**Headset B- *Cloud (Kbytes)***	**Mejora (%)**
Config 1	12133.33	24826.67	183409.90	356107.90	93.03
Config 2	12133.33	12133.33	182872	356071.90	96.60
Config 3	12133.33	12133.33	182602.50	356012.10	96.60
Config 4	12133.33	12133.33	182832.80	356815.80	96.60
Config 5	12133.33	12133.33	182797.40	357676	96.61
Comparación (config 1 vs config 5) %	0	51.13	0.33	-0.44	

Tabla 13: Uso de la red. Mejora de hardware smartphones.

En la tabla se puede apreciar qué en cuanto a consumo de la red, la mejora en el caso de los auriculares A es inexistente, pues, ya estaba al valor mínimo alcanzable (para esta afirmación basta con mirar el resto de simulaciones, este valor es el mínimo alcanzable). En el caso de los auriculares B, una mínima mejora en el hardware (configuración 2) basta para llegar al mínimo consumo de red. En el caso *Cloud* no es apreciable.

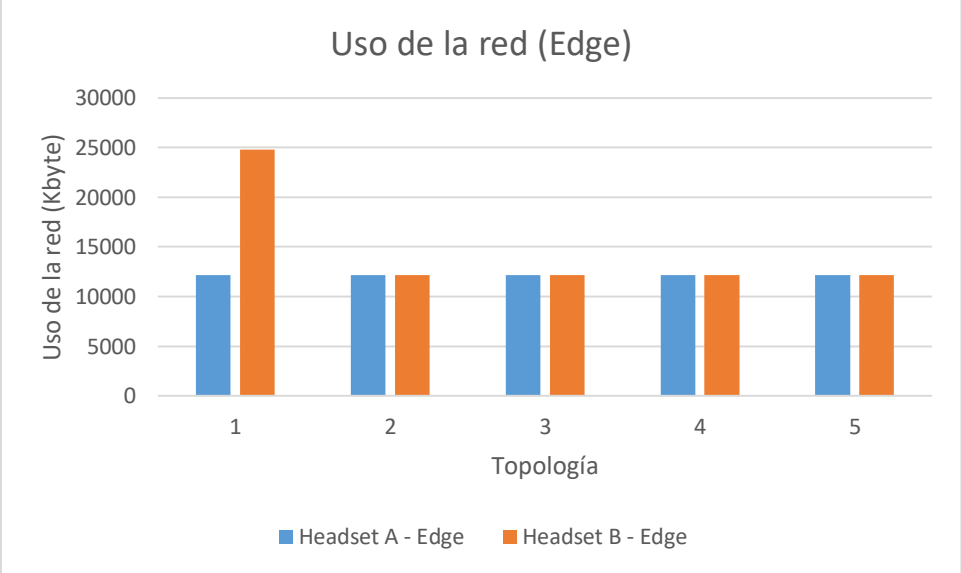


Figura 13: Uso de la red. Mejora hardware smartphones.

5.1.2.2 Mejora de hardware en los routers

Para este caso de uso, de igual forma se dejará invariante el número de routers y el número de jugadores por cada uno de ellos. Se irán variando las mejoras en 5 configuraciones diferentes.

- Estudio de la latencia

Configuración	**Headset A-**	**Headset B-**	**Headset A-**	**Headset B-**	**Mejora (%)**

	***Edgeward* (ms)**	***Edgeward (ms)***	***Cloud (ms)***	***Cloud (ms)***	
Config 1	19.45	29.98	230.62	225.55	86.71
Config 2	19.22	22.38	230.63	225.54	90.08
Config 3	19.59	20.09	230.66	225.56	91.09
Config 4	18.11	18.89	230.60	225.55	91.63
Config 5	18.69	18.87	230.63	225.55	91.64
Comparación (config 1 vs config 5) %	3.91	37.07	0.00	0.00	

Tabla 14: Latencia media en el bucle de control. Mejora hardware routers.

A la vista de los resultados de la tabla de este nuevo caso de uso se puede concluir que unas mejores características de hardware en los routers producen una menor latencia, pero no tanto menor como la de mejorar los terminales de los usuarios, como se demostró anteriormente. Para el caso *Cloud* la diferencia continúa siendo inapreciable, mientras para el caso *Edgeward* sí que conduce a una mejoría.

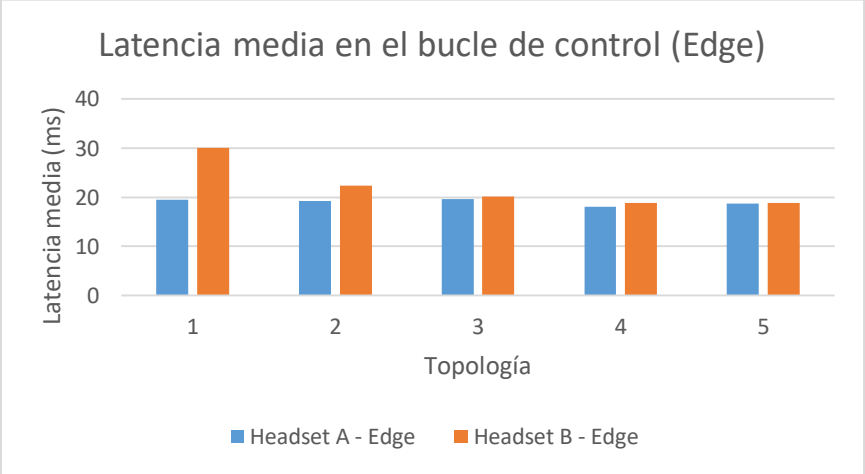


Figura 14: Latencia media en el bucle de control. Mejora routers WiFi.

- Estudio de uso de la red

Mejorar la característica de los routers, según las simulaciones, no influye en un mejor consumo de los recursos de la red, como se muestra en el gráfico siguiente.

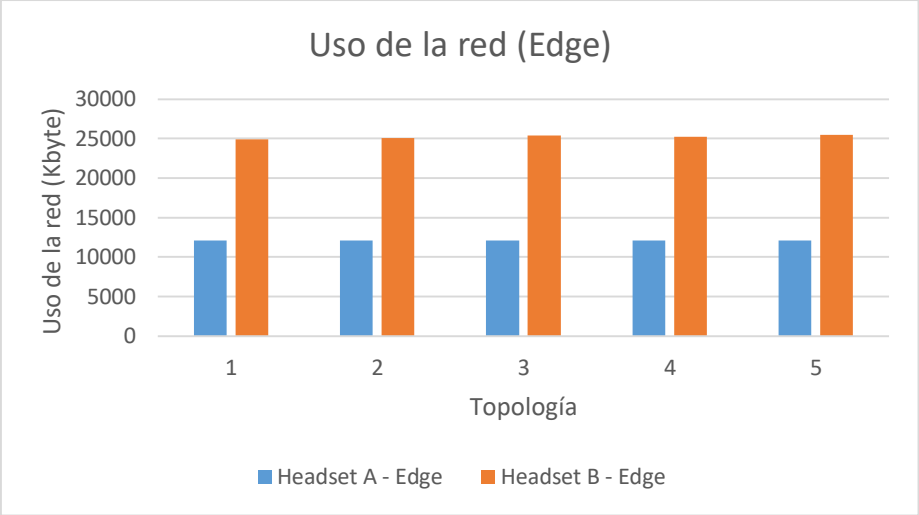


Figura 15: Uso de la red. Mejora hardware routers WiFi.

5.1.2.3 Mejora de hardware en los ISP Gateway

En este caso de uso se estudiará los efectos que suponen mejorar las características de hardware de un ISP Gateway.

Para ello, nuevamente se mantendrán 4 jugadores por router, y 4 routers en total.

Sin embargo, para este caso de uso no se ha encontrado ninguna mejora (o empeora) significativa en términos de latencia, uso de red o coste en la nube.

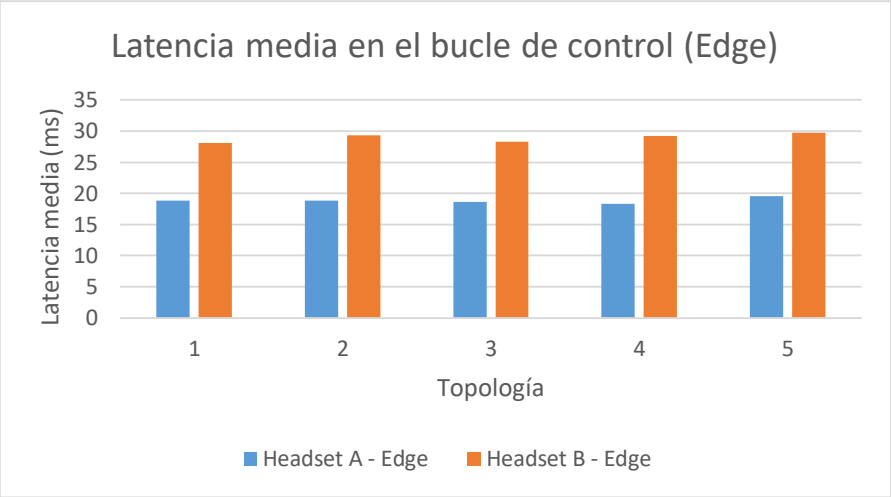


Figura 16: Latencia media en el bucle de control. Mejora ISP.

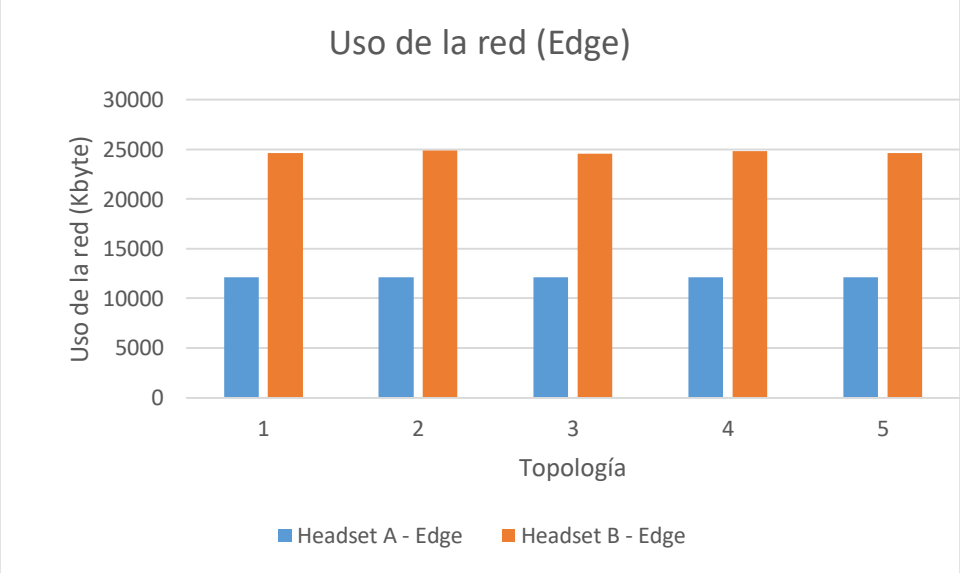


Figura 17: Uso de la red. Mejora de hardware ISP.

Por lo que se concluye que no hay necesidad de mejorar estos elementos de red para el fin de obtener una mejor administración de los recursos de red o una latencia menor.

5.1.2.4 Mejora de hardware del Cloud

El último elemento en la topología definida para el juego objeto de estudio es el *Cloud*. El cual dará soporte cuando los demás dispositivos no puedan hacerse cargo de todas las tareas de computación, y que, además, almacenará por defecto uno de los módulos del juego.

Tras analizar los resultados de las simulaciones, se observa que cambiar las características del *Cloud* no influye sobre ninguno de nuestros objetos de estudio. Para estudiar este caso de uso se ha procedido de la misma forma que en los anteriores; se ha mantenido invariante los routers y jugadores, y aumentado las características en cinco configuraciones diferentes para observar la evolución.

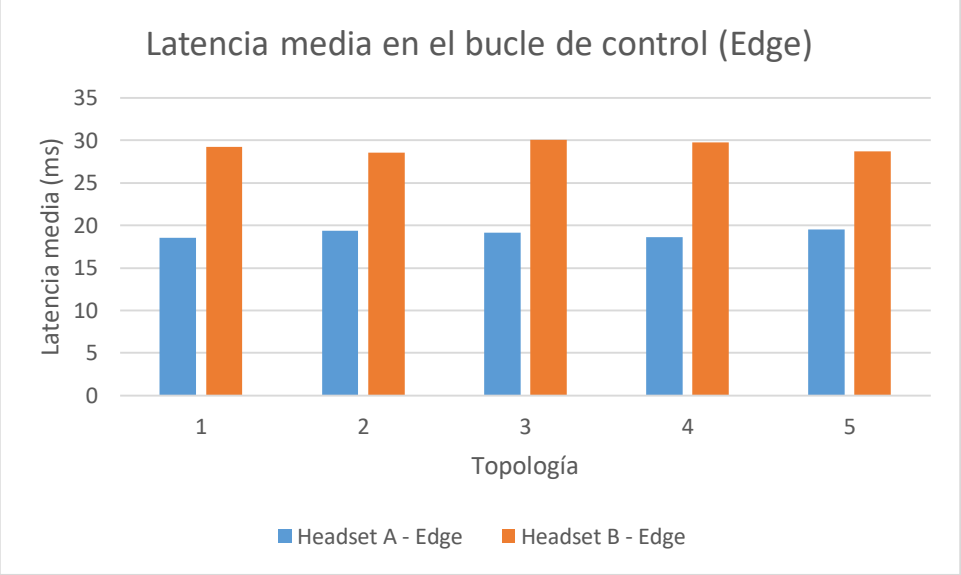


Figura 18: Latencia media en el bucle de control. Mejora de hardware del Cloud.

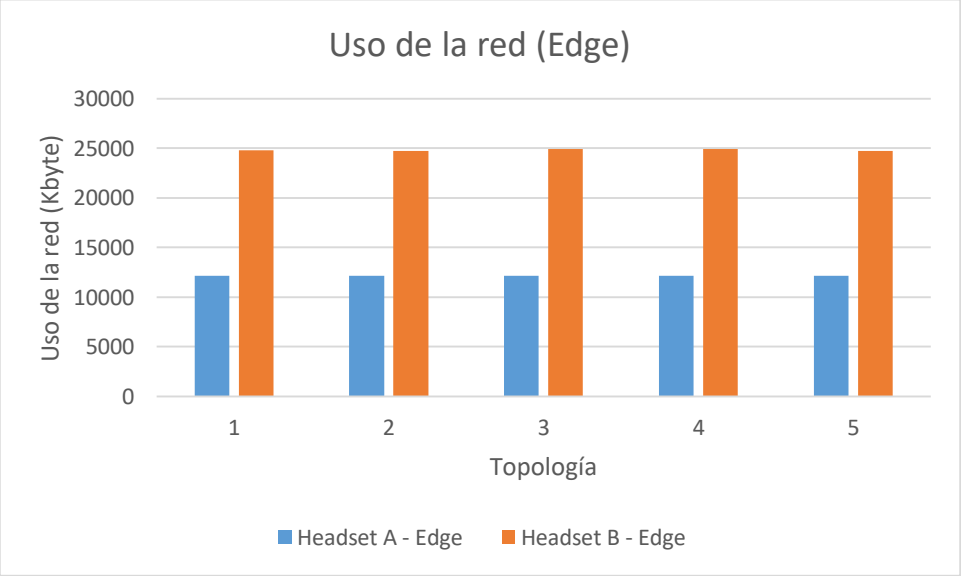


Figura 19: Uso de la red. Mejora de hardware del Cloud.

5.1.3 Ampliación de topologías, resultados de interés

En este apartado se estudiarán algunos de los resultados obtenidos en las simulaciones tras la ampliación de topologías.

5.1.3.1 Variación número de jugadores

Para este caso, se ha dejado constante el número de routers en cuatro, y se ha ido variando el número de jugadores por cada router. En realidad, al igual que el caso base acabará con el mismo número de jugadores, 64. De esta forma, compararemos los resultados del caso base con estos. En esta memoria solo se presentarán los casos con los auriculares B, ya que son el peor escenario posible. Sin embargo, si se quiere hacer un estudio más exhaustivo sobre estas simulaciones quedan a disposición el resto de resultados con los otros auriculares en los anexos.

- Estudio de la latencia en este nuevo caso

Configuración	***Edgeward* (caso base, ms)**	***Edgeward* (nuevo caso, ms)**	***Cloud* (caso base, ms)**	***Cloud* (nuevo caso, ms)**	**Mejora *Cloud* (%)**	**Mejora *Edgeward* (%)**	**Comparación (%)**
Config 1	28.18	16.58	225.16	225.1663473	0.00	41.14	92.63
Config 2	28.11	17.98	225.28	225.2755258	0.00	36.06	92.02
Config 3	29.55	29.44	225.56	225.5456219	0.01	0.38	86.95
Config 4	29.40	245.64	2954.07	2957.186407	-0.11	-735.43	91.69
Config 5	28.75	257.57	4569.87	4570.865148	-0.02	-795.78	94.37

Tabla 15: Latencia media en el bucle de control. Ampliación. Caso base.

En la tabla 15 se muestra el porcentaje de mejora (o empeora, si el número es negativo) de la estrategia de planificación con el nuevo caso de uso respecto al caso de

uso base. La comparación se hace sobre la estrategia *Edgewards* contra la estrategia *Cloud*.

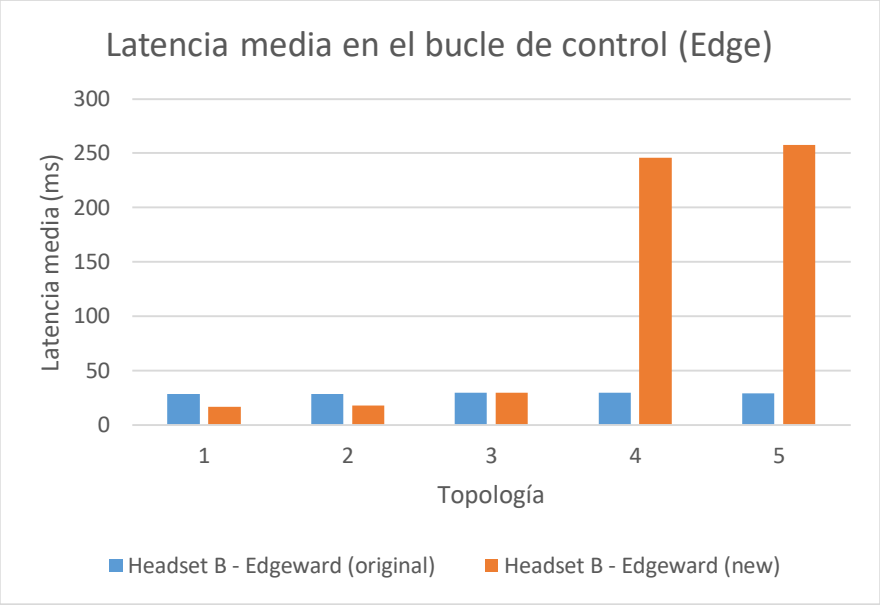


Figura 20: Latencia media en el bucle de control. Ampliación. Caso base.

En el caso de haber más de 32 jugadores (configuración 4), la latencia se dispara tanto en *Cloud* como en la estrategia *Edgeward*. Un dato que resalta es el aumento de latencia en el nuevo caso de uso a partir de esta configuración, cuando en el caso base no es apreciable este aumento. Esto se debe a que en la estrategia *Edgeward*, si se mantiene constante el número de routers (los cuales ayudan a la computación debido a que son dispositivos *Fog*) y se aumenta el número de jugadores (los cuales no ayudan tanto en la computación al ser dispositivos finales) no habrá tantos dispositivos *Fog* para la ejecución de tareas al contrario que en el caso base.

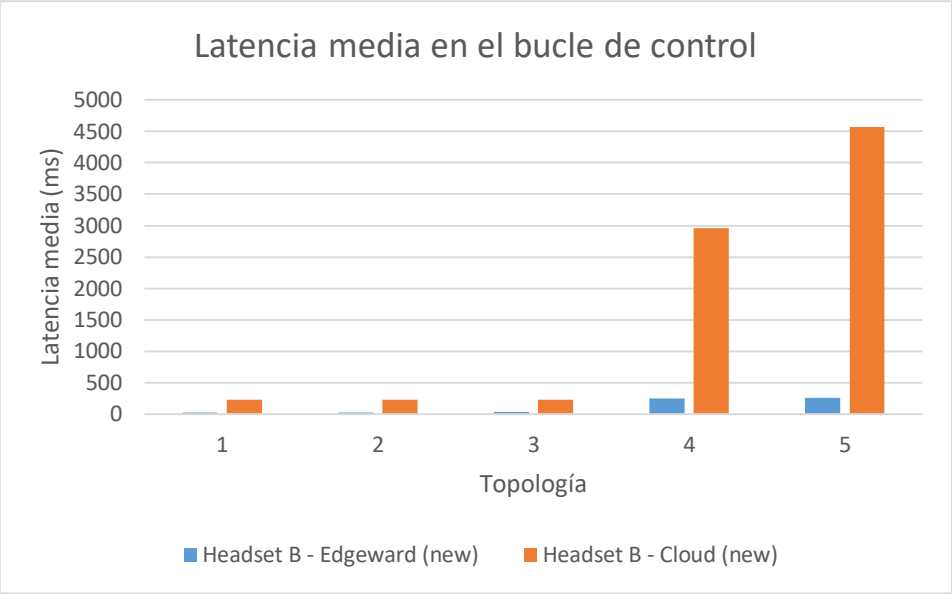


Figura 21: Latencia media en el bucle control. Ampliación. Caso base.

Aun habiendo un notable aumento de la latencia es un valor insignificante contra el emplazamiento *Cloud*.

- Estudio del uso de la red

Configuración	***Edgeward* (caso base, Kbytes)**	***Edgeward* (nuevo caso, Kbytes)**	***Cloud* (caso base, Kbytes)**	***Cloud* (nuevo caso, Kbytes)**	**Mejora *Cloud* (%)**	**Mejora *Edgeward* (%)**	**Comparación (%)**
Config 1	6733.33	4880	77300.30	81839.90	-5.87	27.52	93.69
Config 2	12853.33	10243.33	161716.20	167661.10	-3.68	20.31	93.67
Config 3	24476.67	25046.67	357285.30	357471.70	-0.05	-2.33	92.99
Config 4	49040	482570	695120.60	694719.10	0.06	-884.03	30.58
Config 5	96780	923120	1274013.20	1273137.50	0.07	-853.83	27.54

Tabla 16: Uso de la red. Ampliación. Caso base.

El uso de la red en el caso *Edgeward*s a partir de la configuración número cuatro se hace más notorio en el caso de dejar constantes el número de routers. Como se especuló anteriormente, debe ser porque al saturar los routers cuando hay tantos jugadores, el tráfico se tiene que desviar hacia otros dispositivos o la nube.

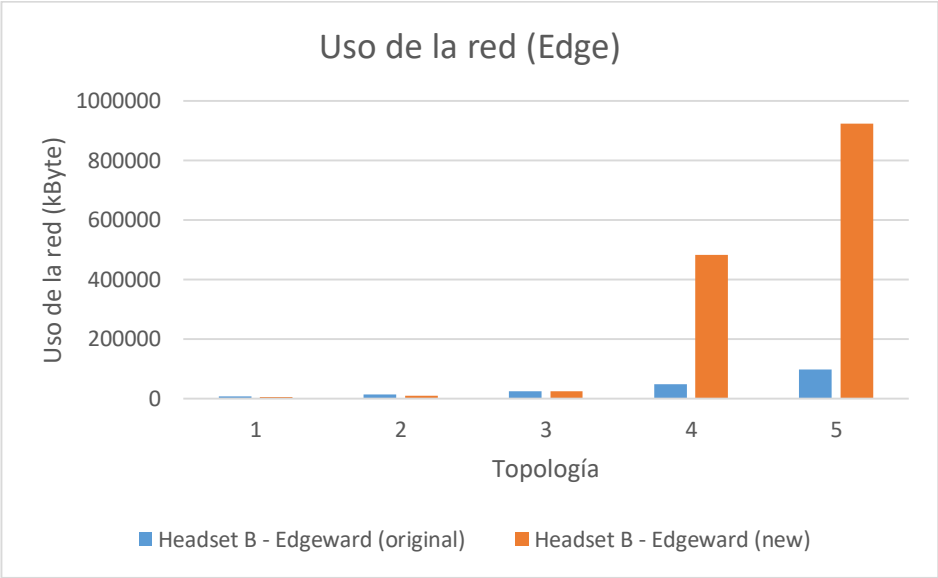


Figura 22: Uso de la red. Ampliación. Caso base.

Aún con este aumento del uso de los recursos de red, la estrategia *Edge* sigue siendo más eficiente que la planificación *Cloud*.

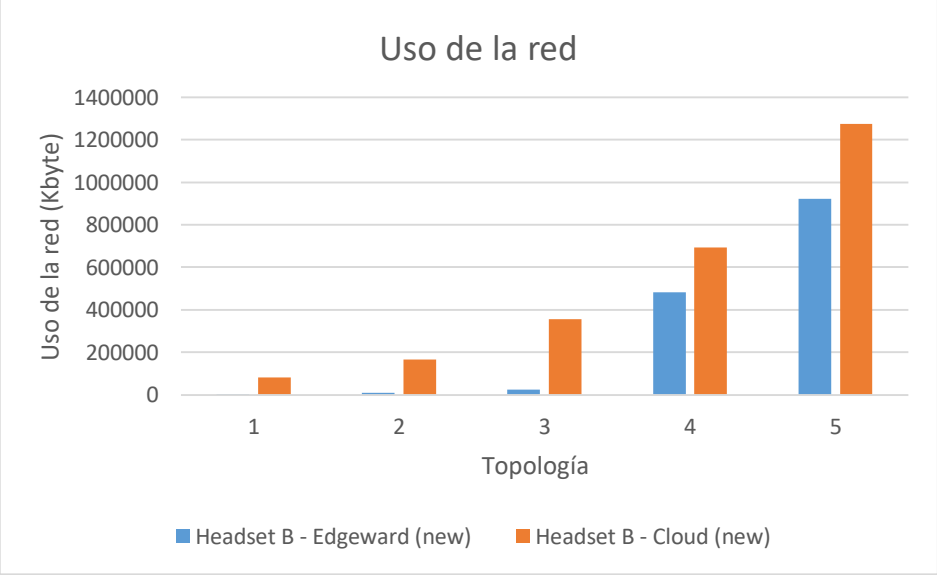


Figura 23: Uso de la red. Ampliación (comparativa). Caso base.

- Estudio de uso del coste de ejecución en la nube

Configuración	*Edgeward* (caso base)	*Edgeward* (nuevo caso)	*Cloud* (caso base)	*Cloud* (nuevo caso)	Mejora *Cloud* (%)	Mejora *Edgeward* (%)	Comparación (%)
Config 1	*6036*	*6036*	*541698.98*	*541431.69*	*0.05*	*0*	*98.89*
Config 2	*6145.60*	*6145.60*	*1144509.42*	*1106976.26*	*3.28*	*0*	*99.44*
Config 3	*6364.80*	*6364.80*	*1952168.32*	*1949659.90*	*0.13*	*0*	*99.67*
Config 4	*6803.20*	*83184.60*	*3062326.46*	*3071815.10*	*-0.31*	*-1122.73*	*97.29*
Config 5	*7680*	*85256.40*	*4436633.60*	*4437358.02*	*-0.02*	*-1010.11*	*98.08*

Tabla 17: Coste de ejecución en la nube. Ampliación. Caso base.

El coste de ejecución en la nube cuando se colapsan los routers se incrementa muchísimo, pues estos mandan a procesar los datos con los que no pueden a la nube, incrementándose este coste.

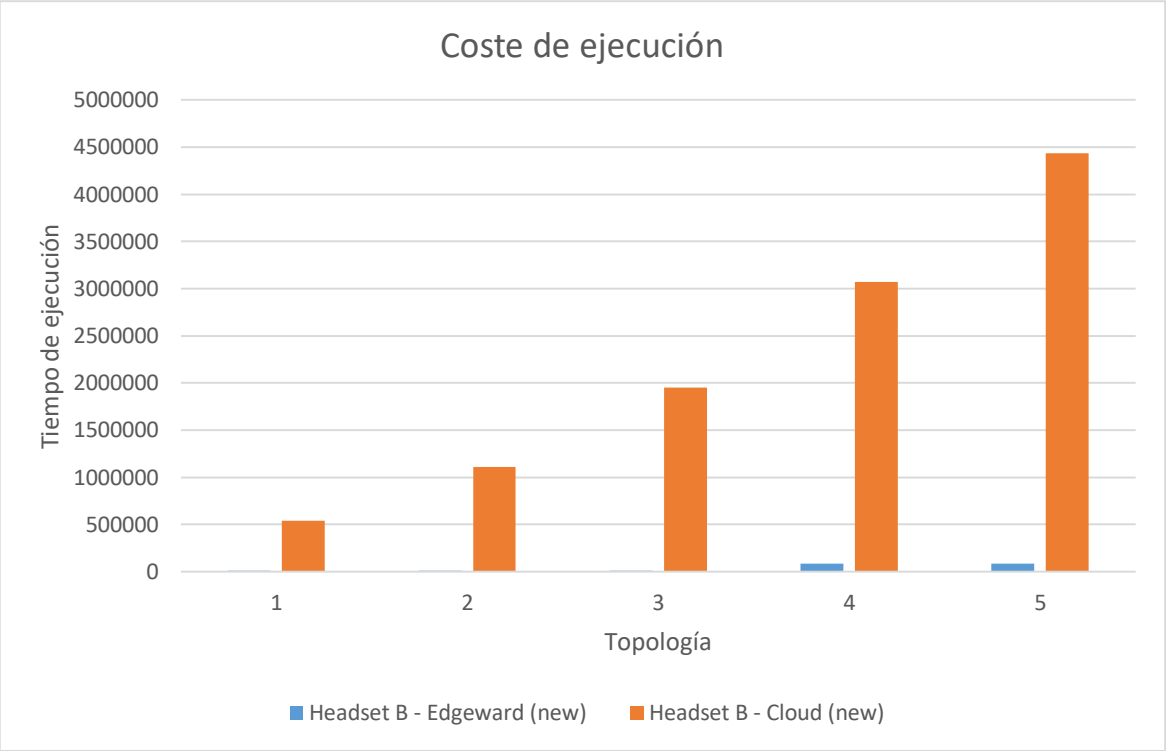


Figura 24: Coste de ejecución en la nube. Ampliación (comparativa). Caso base.

Comparando ambas estrategias, aunque se colapsen los routers WiFi y manden a ejecutar el resto de tareas en la nube sigue siendo un coste bastante menor, pues sí que se pueden hacer cargo de algunas de las tareas y quitarle carga a la nube de esta forma.

5.1.3.2 Mejora de hardware de los smartphones

En el siguiente caso de uso se plantea la cuestión de que pasaría si las características del hardware en los smartphones utilizados para ejecutar la aplicación *EEG Tractor Beam* fueran mejores, ya que, las antiguas características están desfasadas para la gama media de los smartphones actuales.

De la forma en la que se explicó en el capítulo anterior, se establecieron un máximo en las capacidades de los terminales que serían usados para la simulación, y, desde las mínimas (para los resultados base se utilizan estas mínimas características) aumentarlas gradualmente en 5 configuraciones diferentes.

Para los resultados que se analizaran a continuación se ha dejado constante el número de jugadores por router y el número de estos irá variando.

- Estudio de la latencia

Configuración	*Edgeward* (caso base, ms)	*Edgeward* (nuevo caso, ms)	*Cloud* (caso base, ms)	*Cloud* (nuevo caso, ms)	Mejora *Cloud* (%)	Mejora *Edgeward* (%)	Comparación (%)
Config 1	28.18	11.74	225.16	224.31	0.38	58.34	94.77
Config 2	28.11	12.00	225.28	224.60	0.30	57.33	94.66
Config 3	29.55	12.39	225.56	224.87	0.31	58.06	94.49
Config 4	29.40	12.50	2954.07	2958.41	-0.15	57.50	99.58
Config 5	28.75	12.21	4569.87	4570.34	-0.01	57.52	99.73

Tabla 18: Latencia media en el bucle de control. Ampliación. Mejora smartphones.

La latencia cuando las capacidades de los smartphones son superiores se reduce a más de la mitad en el caso de planificación *Edgeward*. En el caso *Cloud*, al no ser estos responsables de la computación, la latencia permanece prácticamente constante.

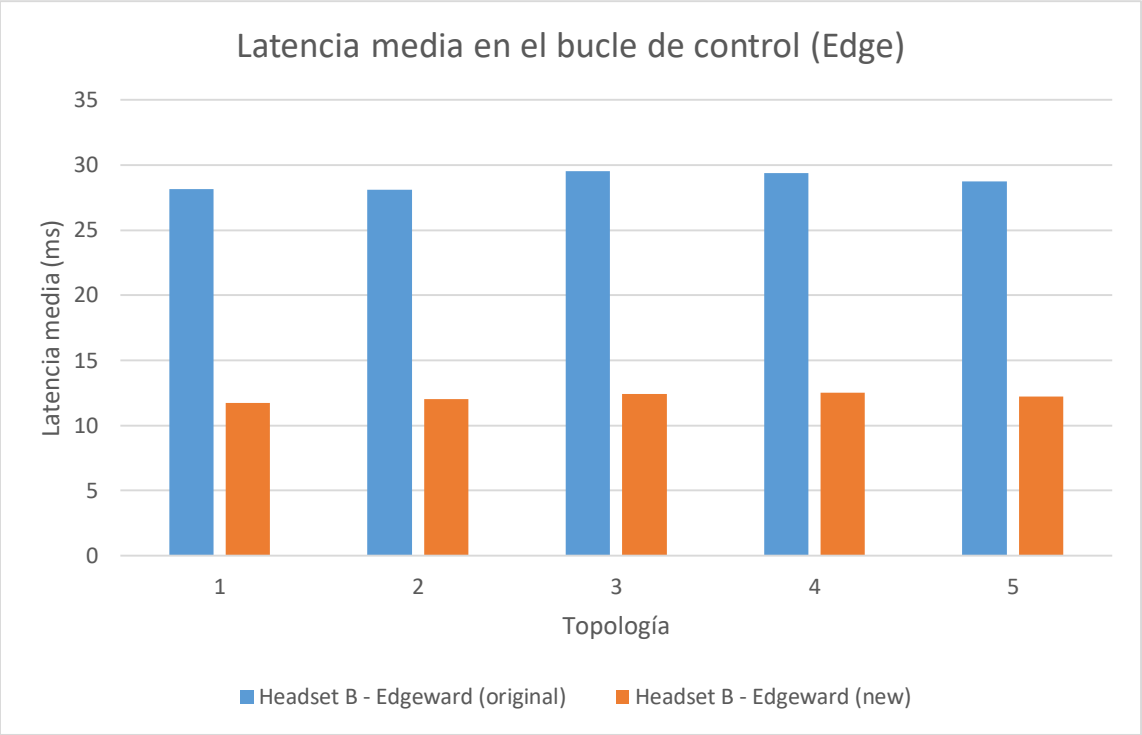


Figura 25: Latencia media en el bucle de control. Ampliación. Mejora smartphones.

Si los dispositivos finales tienen recursos suficientes para realizar la ejecución de tareas, quitan carga de computación a los otros dispositivos *Fog*. Efectuándose estos procesamientos in situ se consigue menor latencia.

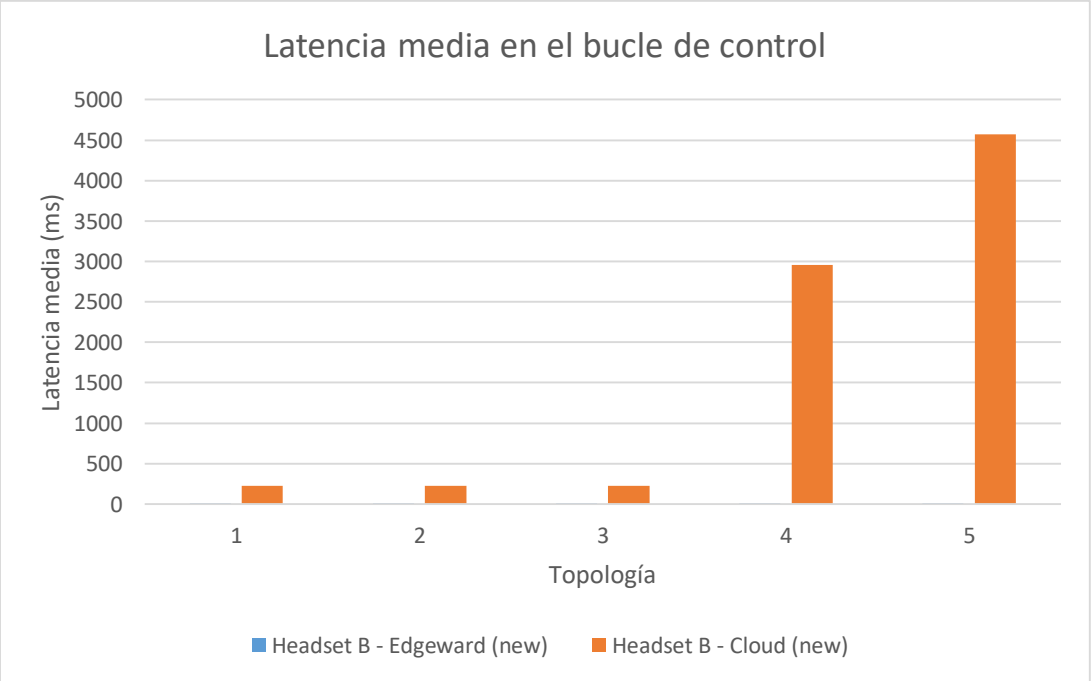


Figura 26: Latencia media de ambas tecnologías. Ampliación (comparación). Mejora smartphones.

- Uso de la red

Configuración	*Edgeward* (caso base, Kbytes)	*Edgeward* (nuevo caso, Kbytes)	*Cloud* (caso base, Kbytes)	*Cloud* (nuevo caso, Kbytes)	Mejora *Cloud* (%)	Mejora *Edgeward* (%)	Comparación (%)
Config 1	6733.33	3533.33	77300.30	77714.60	-0.54	47.52	95.45
Config 2	12853.33	6400	161716.20	161814.20	-0.06	50.21	96.04
Config 3	24476.67	12133.33	357285.30	356728.90	0.16	50.43	96.60
Config 4	49040	23600	695120.60	695902.90	-0.11	51.88	96.61
Config 5	96780	46533.33333	1274013.2	1271116.70	0.23	51.92	96.34

Tabla 19: Uso de la red. Ampliación. Mejora smartphones.

En cuanto al uso de la red, en los datos simulados se puede apreciar una mejora del 50% aproximadamente en la estrategia *Edgeward*, pues no se tiene que mandar tantas tareas a otros dispositivos para su ejecución.

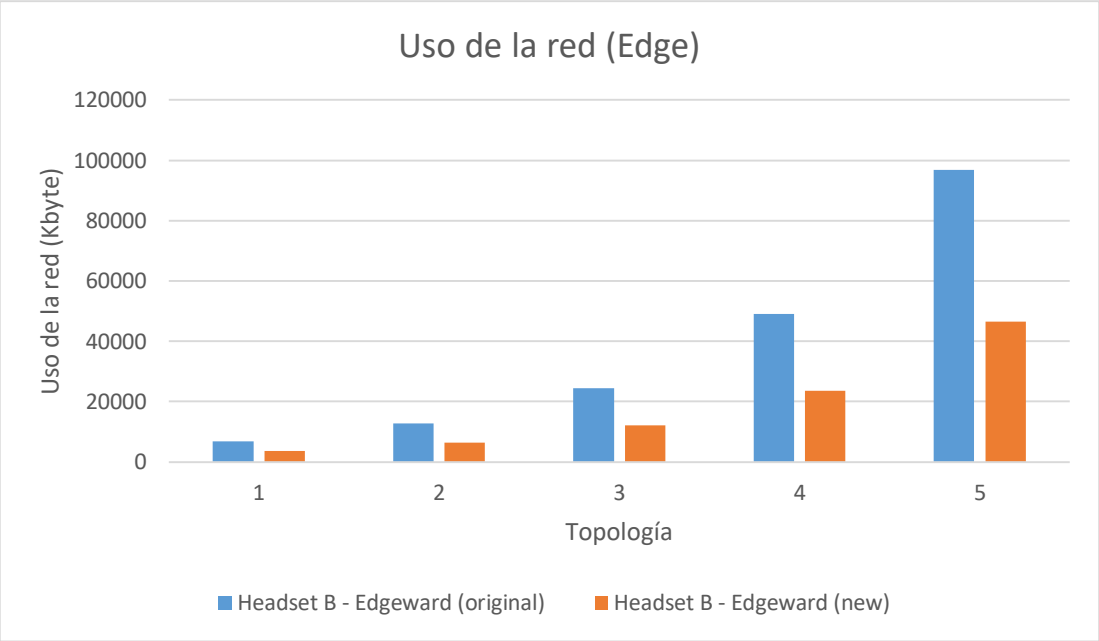


Figura 27: Uso de la red. Ampliación. Mejora smartphones.

En el caso *Cloud* sigue siendo un consumo de los recursos de red bastante alto. La mejora de hardware en los dispositivos finales no influye en el caso *Cloud*.

El hecho de que estas mejoras influyan en el caso *Edge* puede suponer una ventaja, ya que los usuarios finales suelen que, como en este caso de uso, tengan videojuegos en sus smartphones, suelen tener uno con buen hardware para ello.

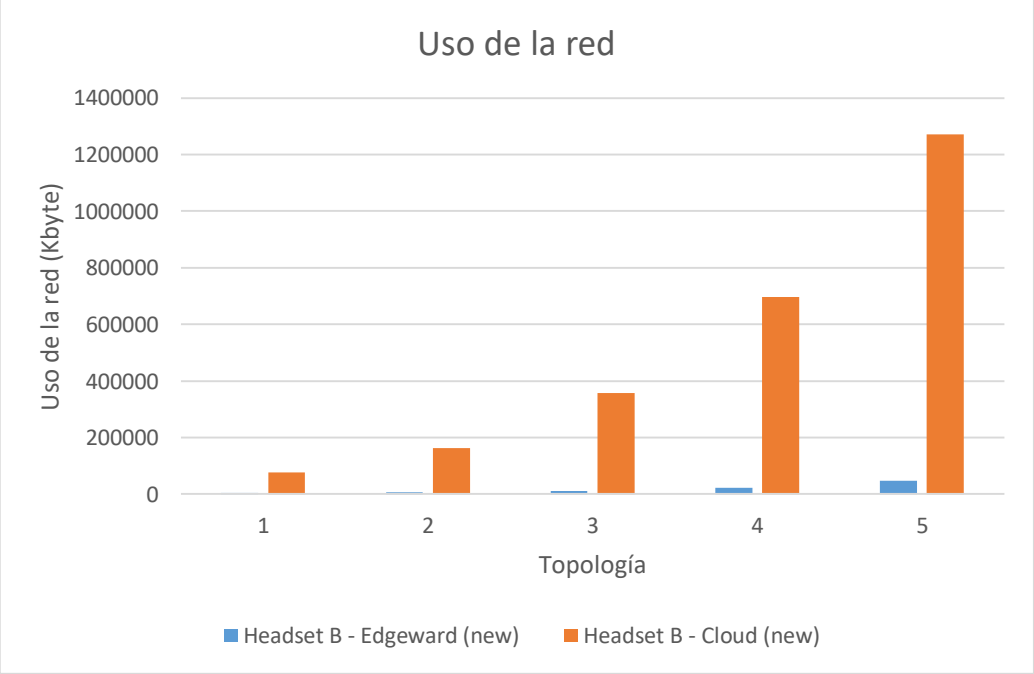


Figura 28: Uso de la red de ambas tecnologías. Ampliación (comparación). Mejora smartphones.

Comparando ambas tecnologías se ve la gran diferencia entre el consumo de recursos de red.

5.1.3.3 Mejora de las características del hardware de los routers WiFi

En este caso de uso se mejorarán las características de hardware de los routers tal y como se detalló en el capítulo anterior.

Se mantendrá intacto el número de jugadores por router y los routers se irán variando.

- Estudio de la latencia

Configuración	***Edgeward* (caso base, ms)**	***Edgeward* (nuevo caso, ms)**	***Cloud* (caso base, ms)**	***Cloud* (nuevo caso, ms)**	**Mejora *Cloud* (%)**	**Mejora *Edgeward* (%)**	**Comparación (%)**
Config 1	28.18	18.87	225.16	225.15	0.00	33.05	91.62
Config 2	28.11	18.65	225.28	225.28	0.00	33.66	91.72
Config 3	29.55	18.45	225.56	225.54	0.01	37.55	91.82
Config 4	29.40	18.78	2954.07	2962.46	-0.28	36.14	99.37
Config 5	28.75	18.80	4569.87	4574.75	-0.11	34.62	99.59

Tabla 20: Latencia media. Ampliación de topologías. Mejora routers WiFi.

Al mejorar las características de los routers WiFi, tabla 20, mejora la latencia en el caso *Edge*. Sin embargo, si se compara con el porcentaje de mejora con el anterior caso de uso (al mejorar las características de los smartphones, ver tabla 21), la latencia es mayor. Esto es debido a que los smartphones, al permanecer con características más mediocres en este caso de uso, la mayor parte de la ejecución de las tareas es realizada por los routers WiFi, por lo que se añade una latencia adicional.

Configuración	Mejora hardware routers (%)	Mejora hardware smartphones (%)
Config 1	33.05	58.34
Config 2	33.66	57.33

Tabla 21: Muestra tabla 18.

Aun cuando la latencia es sensiblemente mayor que en el anterior caso de uso, sigue siendo menor que en el caso base que se está comparando.

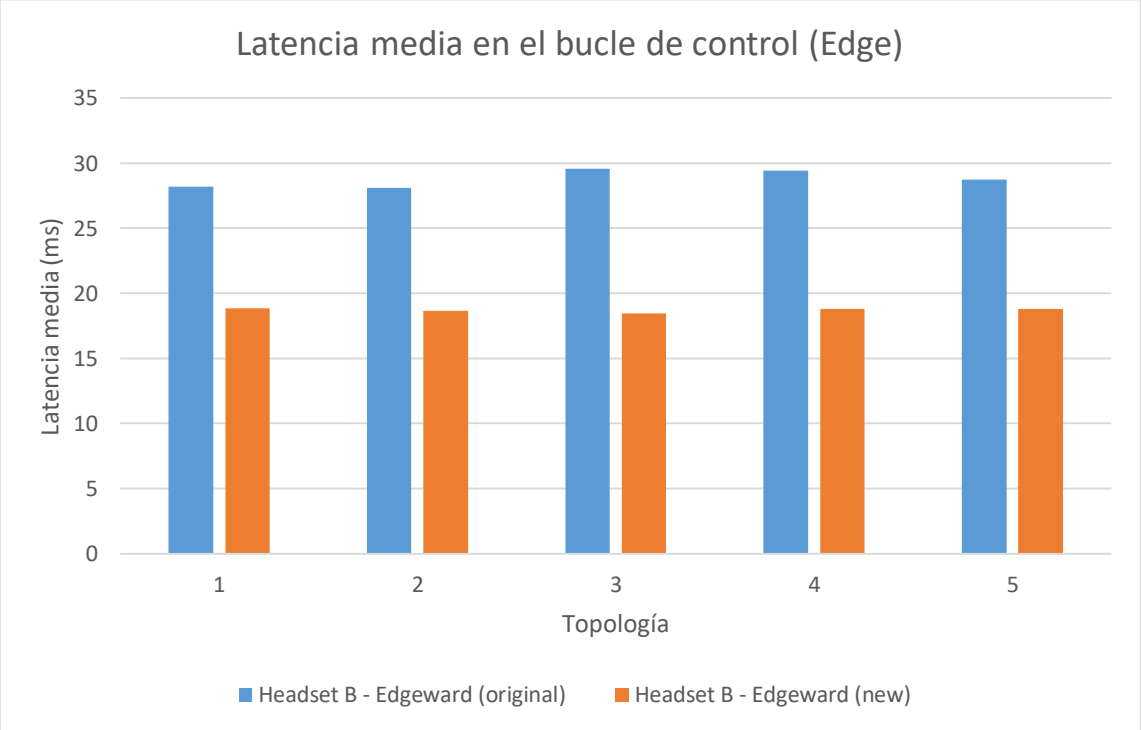


Figura 29: Latencia bucle de control. Ampliación. Mejora routers WiFi.

A pesar de ser ligeramente superior que el anterior caso de uso, sigue siendo inapreciable al compararla con la estrategia de *Cloud Computing*.

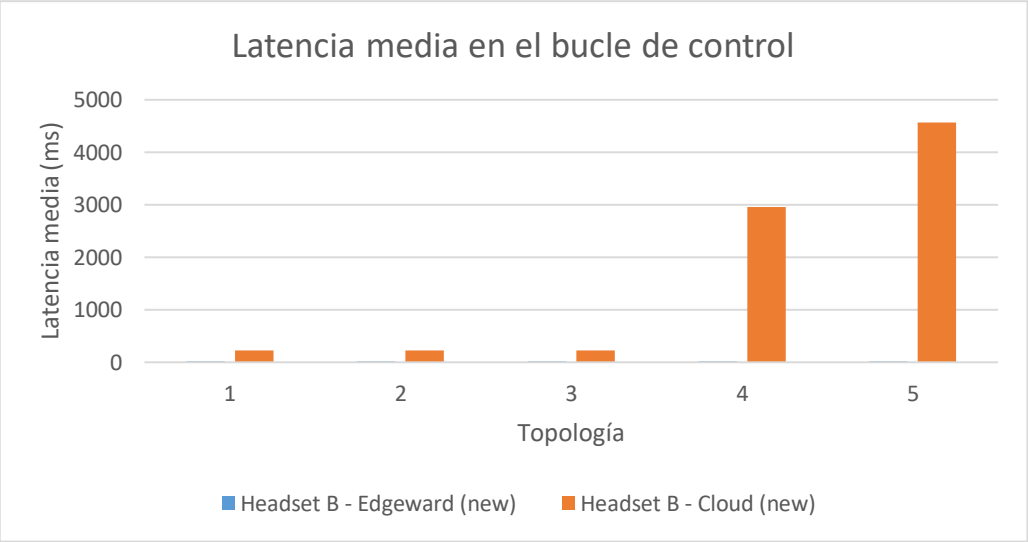


Figura 30: Latencia media. Ampliación (comparación). Mejora routers WiFi.

- Uso de la red

Configuración	*Edgeward* (caso base, Kbytes)	*Edgeward* (nuevo caso, Kbytes)	*Cloud* (caso base, Kbytes)	*Cloud* (nuevo caso, Kbytes)	Mejora *Cloud* (%)	Mejora *Edgeward* (%)	Comparación (%)
Config 1	6733.33	6873.33	77300.30	77367.70	-0.09	-2.08	95.43
Config 2	12853.33	12930	161716.20	162192	-0.29	-0.60	96.05
Config 3	24476.67	24836.67	357285.30	357730.20	-0.12	-1.47	96.61
Config 4	49040	49506.67	695120.60	696639.20	-0.22	-0.95	96.61
Config 5	96780	99390	1274013.20	1274173.90	-0.01	-2.70	96.35

Tabla 22: Uso de la red. Ampliación. Mejora routers WiFi.

En este caso, el consumo de recursos de red en el caso *Edge* es ligeramente superior con respecto al caso base.

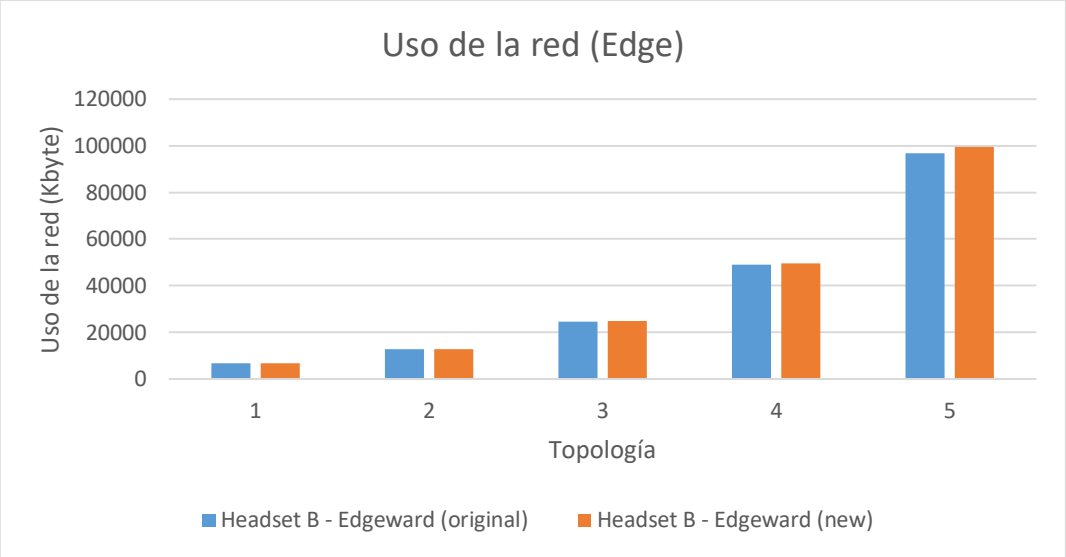


Figura 31: Uso de la red. Ampliación. Mejora routers WiFi.

En todo caso, sigue siendo mucho más eficiente que el caso de *Cloud Computing*.

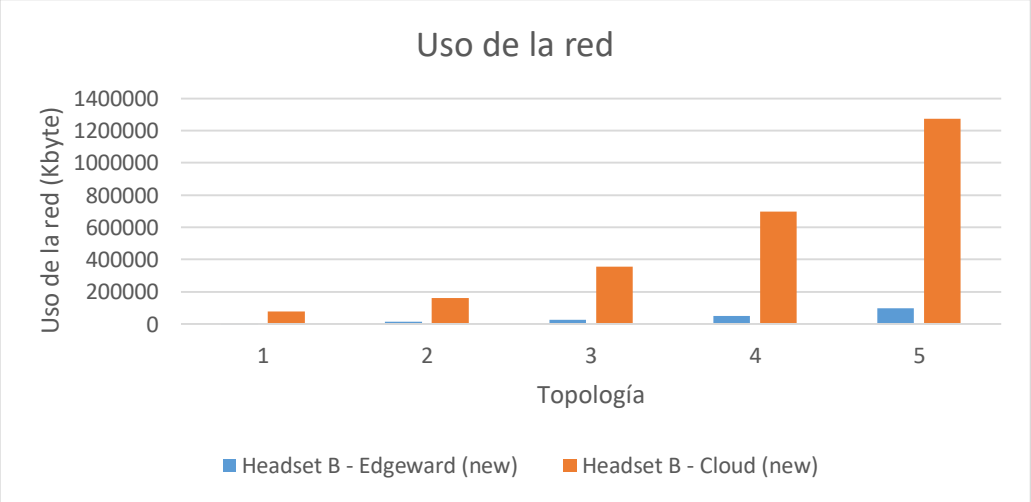


Figura 32: Uso de la red. Ampliación de topologías (comparación). Mejora routers WiFi.

5.1.3.4 Mejora de las características del ISP Gateway

En este caso de uso se mejorarán las características del Gateway proveedor de servicios considerado para la simulación. Para ello, se variarán las características como se explicó en el capítulo anterior.

Se vuelven a dejar constantes el número de jugadores por router WiFi y se variarán estos según la configuración a considerar.

- Estudio de la latencia

Configuración	***Edgeward* (caso base, ms)**	***Edgeward* (nuevo caso, ms)**	***Cloud* (caso base, ms)**	***Cloud* (nuevo caso, ms)**	**Mejora *Cloud* (%)**	**Mejora *Edgeward* (%)**	**Comparación (%)**
Config 1	28.18	30.18	225.16	225.15	0.00	-7.09	86.60
Config 2	28.11	30	225.28	225.27	0.00	-6.72	86.68
Config 3	29.55	28.25	225.56	225.53	0.01	4.40	87.47
Config 4	29.40	29.02	2954.07	2958	-0.13	1.29	99.02
Config 5	28.75	29.13	4569.87	4570.44	-0.01	-1.32	99.36

Tabla 23: Latencia media. Ampliación. Mejora ISP.

Tras examinar la tabla 23, se concluye con que dejar las características intactas del resto de dispositivos mientras se mejoran las de los ISP no supone una mejora en la latencia.

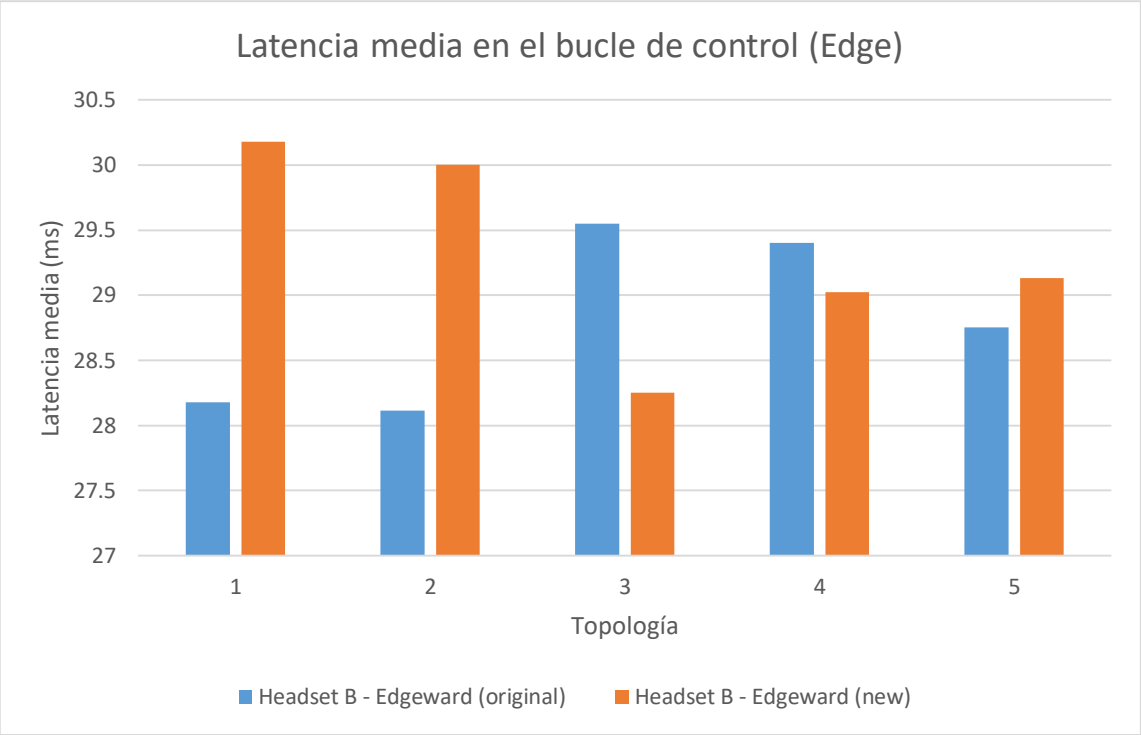


Figura 33: Latencia media. Ampliación. Mejora ISP.

De hecho, es dificil especular sobre lo que significan estos datos pues para algunas configuraciones consigue mejorarse la latencia (configuraciones tres y cuatro) pero para el resto es superior.

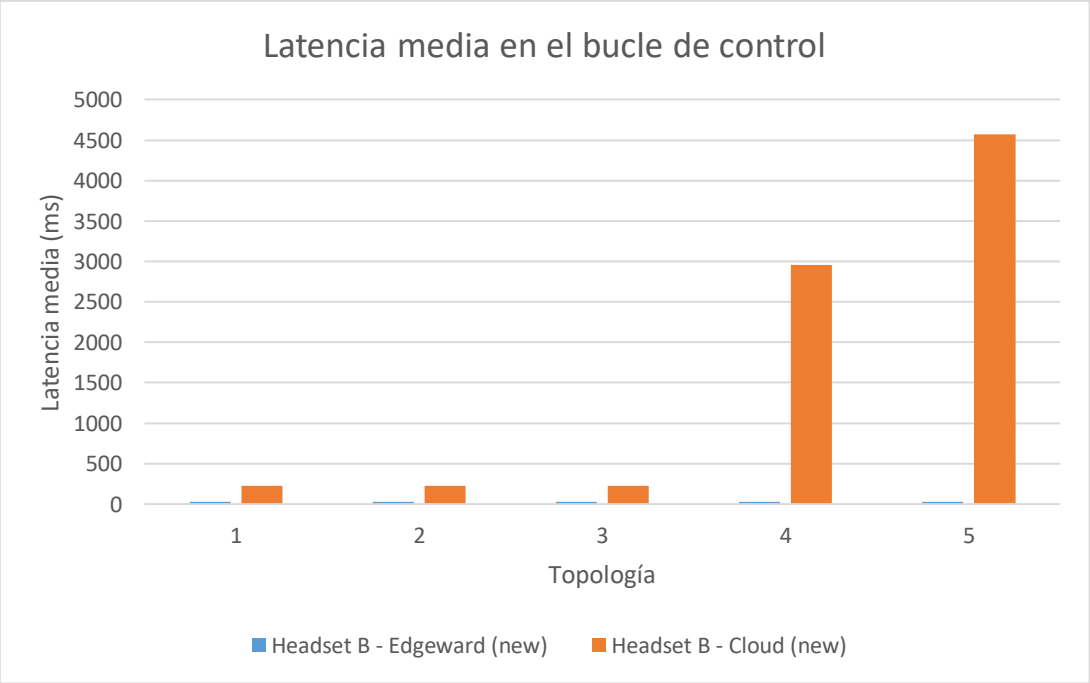


Figura 34: Latencia media. Ampliación (comparación). Mejora ISP.

Aun así, a la vista del gráfico de la figura 33, se puede observar en que sigue resultando en una menor latencia que en el caso *Cloud*.

- Estudio del uso de la red

Configuración	***Edgeward* (caso base, Kbytes)**	***Edgeward* (nuevo caso, Kbytes)**	***Cloud* (caso base, Kbytes)**	***Cloud* (nuevo caso, Kbytes)**	**Mejora *Cloud* (%)**	**Mejora *Edgeward* (%)**	**Comparación (%)**
Config 1	6733.33	6716.67	77300.30	77775.70	-0.62	0.25	95.46
Config 2	12853.33	12613.33	161716.20	162069.10	-0.22	1.87	96.05
Config 3	24476.67	24836.67	357285.30	356898.20	0.11	-1.47	96.60
Config 4	49040	49133.33	695120.60	696579.70	-0.21	-0.19	96.61
Config 5	96780	97440	1274013.20	1272949.70	0.08	-0.68	96.34

Tabla 24: Uso de la red. Ampliación. Mejora ISP.

A la vista de los datos de la tabla 24, se puede decir que tampoco influye en gran medida en el consumo de los recursos de red, ya que las variaciones para ambos casos oscilan como mucho en el 1.8%.

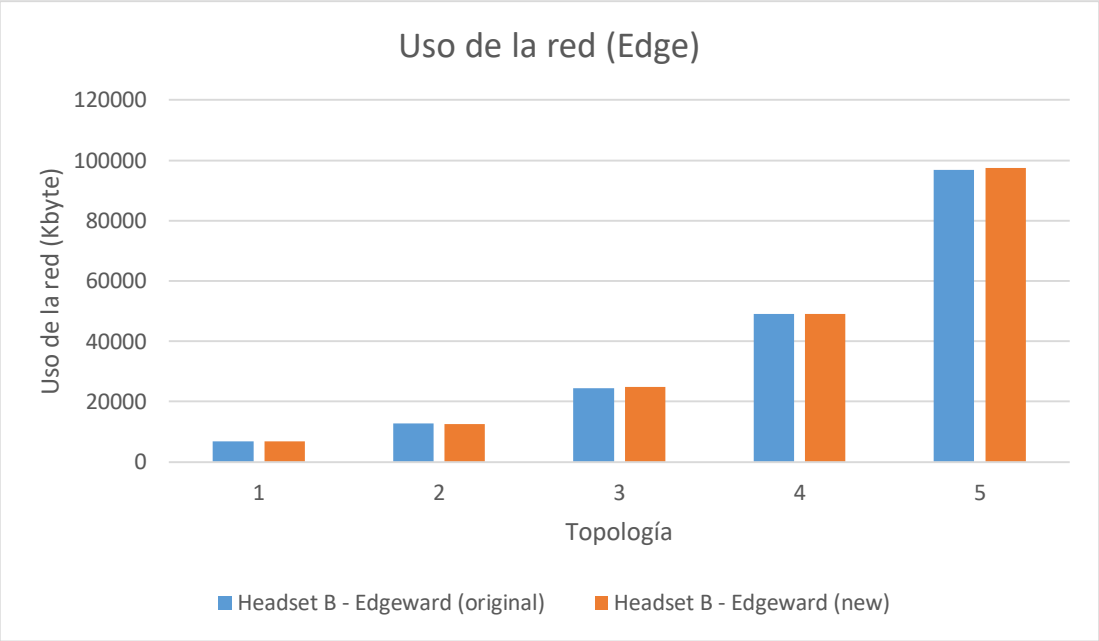


Figura 35: Uso de la red. Ampliación. Mejora ISP.

La comparativa entre el caso base y este nuevo resulta fundamentalmente la misma, como se aprecia en el gráfico de la figura 34. El cual, parece no cambiar con el del caso base.

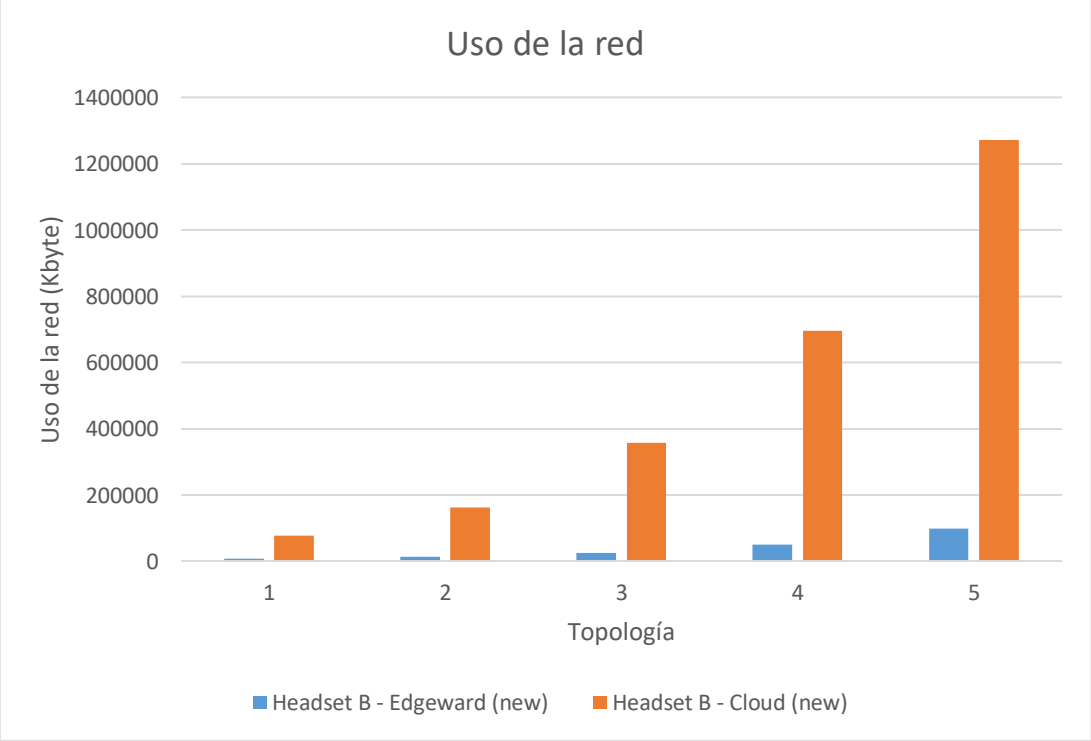


Figura 36: Uso de la red. Ampliación (comparativa). Mejora ISP.

5.1.3.5 Mejora de las características del Cloud

Para el último análisis de la ampliación de topologías que se añadirá en el presente documento, se decide mejorar las características del *Cloud*. Al igual que en los demás casos de uso, se dejará intacto el número de jugadores por router WiFi mientras se va variando el número de routers.

- Estudio de la latencia

Configuración	***Edgeward* (caso base, ms)**	***Edgeward* (nuevo caso, ms)**	***Cloud* (caso base, ms)**	***Cloud* (nuevo caso, ms)**	**Mejora *Cloud* (%)**	**Mejora *Edgeward* (%)**	**Comparación (%)**
Config 1	28.18	11.72	225.16	224.37	0.35	58.40	94.78
Config 2	28.11	12.19	225.28	224.57	0.31	56.64	94.57
Config 3	29.55	12.36	225.56	224.90	0.29	58.16	94.50
Config 4	29.40	12.46	2954.07	2949.70	0.15	57.61	99.58
Config 5	28.75	12.44	4569.87	4568.45	0.03	56.73	99.73

Tabla 25: Latencia media. Ampliación de topologías. Mejora Cloud.

En el caso de aumentar las características del *Cloud* se observa una reducción de latencia en el caso de planificación *Edgeward* alrededor de un 50%. Esto es debido a que la aplicación *EEG Tractor Beam* ubica el módulo encargado de almacenar la información de los estados cerebrales medios de los jugadores en el *Cloud*. Con un aumento de las capacidades de este, se puede disponer de esta información más rápidamente y con ello, disminuir la latencia.

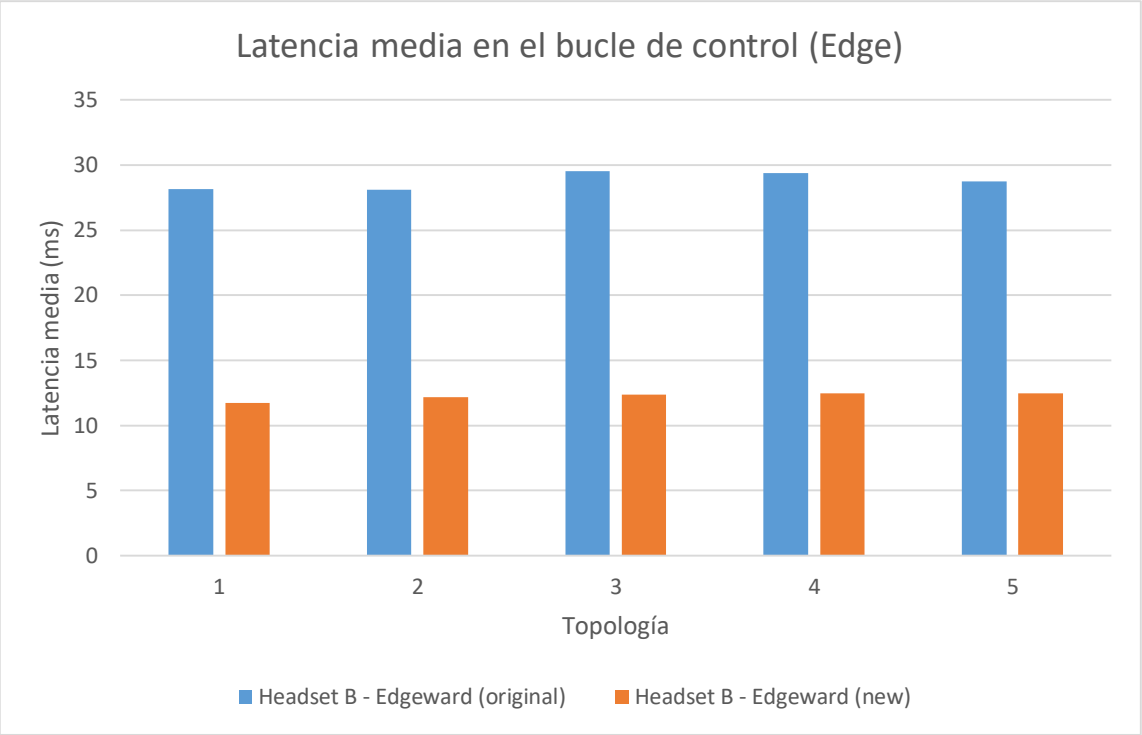


Figura 37: Latencia media. Ampliación. Mejora Cloud.

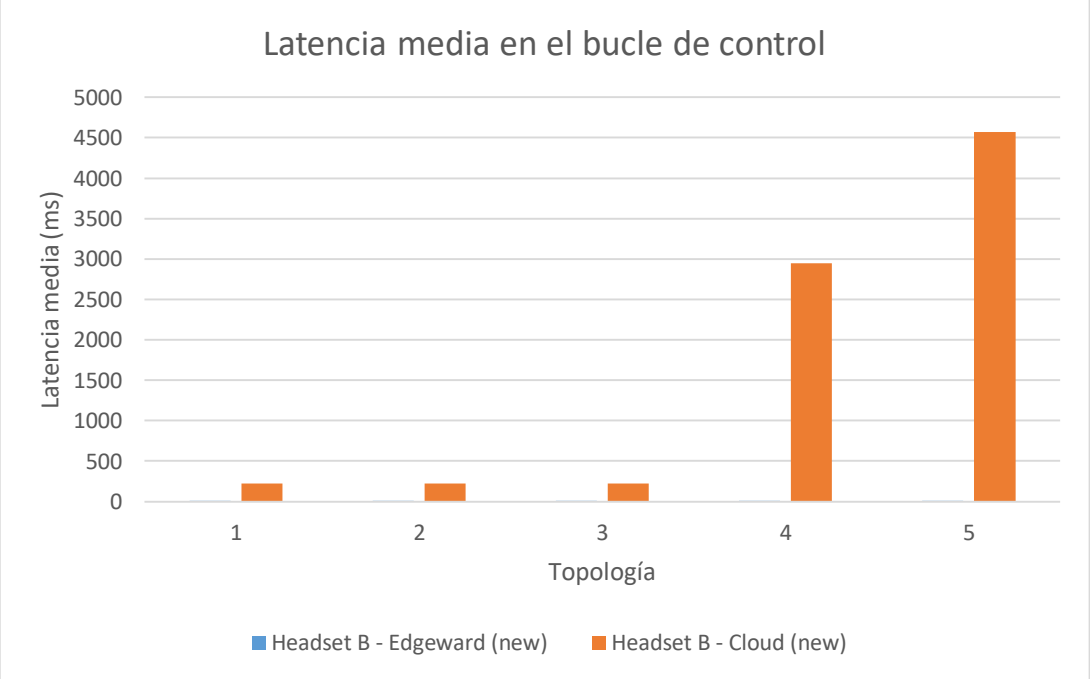


Figura 38: Latencia media. Ampliación (comparativa). Mejora Cloud.

En el caso *Cloud* no hace tanta diferencia cambiar sus características porque ya disponía de recursos suficientes para todo el procesamiento sin llegar a saturarse, como podría pasar en el caso de muchas más aplicaciones usando este mismo *Cloud*.

- Estudio del uso de la red

Configuración	*Edgeward* (caso base, Kbytes)	*Edgeward* (nuevo caso, Kbytes)	*Cloud* (caso base, Kbytes)	*Cloud* (nuevo caso, Kbytes)	Mejora *Cloud* (%)	Mejora *Edgeward* (%)	Comparación (%)
Config 1	6733.33	3533.33	77300.30	77420.30	-0.15	47.52	95.44
Config 2	12853.33	6400	161716.20	162351.40	-0.39	50.21	96.06
Config 3	24476.67	12133.33	357285.30	357331.20	-0.013	50.43	96.60
Config 4	49040	23600	695120.60	695916.60	-0.11	51.88	96.61
Config 5	96780	46533.33	1274013.20	1274242.30	-0.02	51.92	96.35

Tabla 26: Uso de la red. Ampliación. Mejora Cloud.

En el caso *Cloud* no se aprecia variación ya que es inferior al 1%, pero, en el caso *Edge* se aprecia una mejora de hasta un 50% respecto al caso base de estudio.

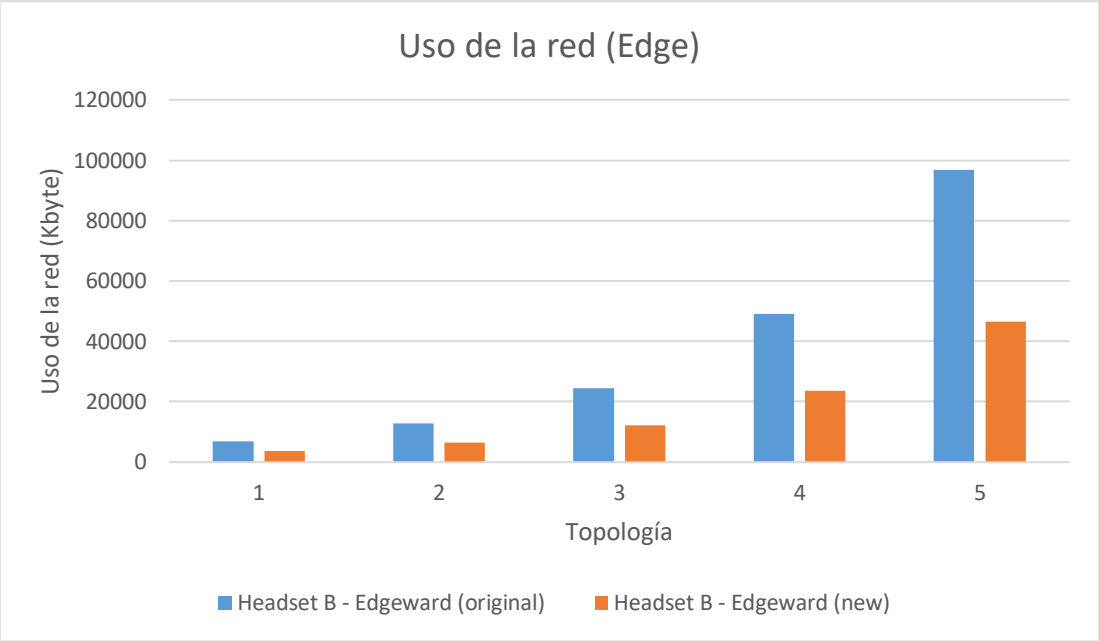


Figura 39: Uso de la red. Ampliación. Mejora Cloud.

Por lo que la mejora es aún más pronunciada al comparar ambas estrategias, como se aprecia en el gráfico de la figura 39.

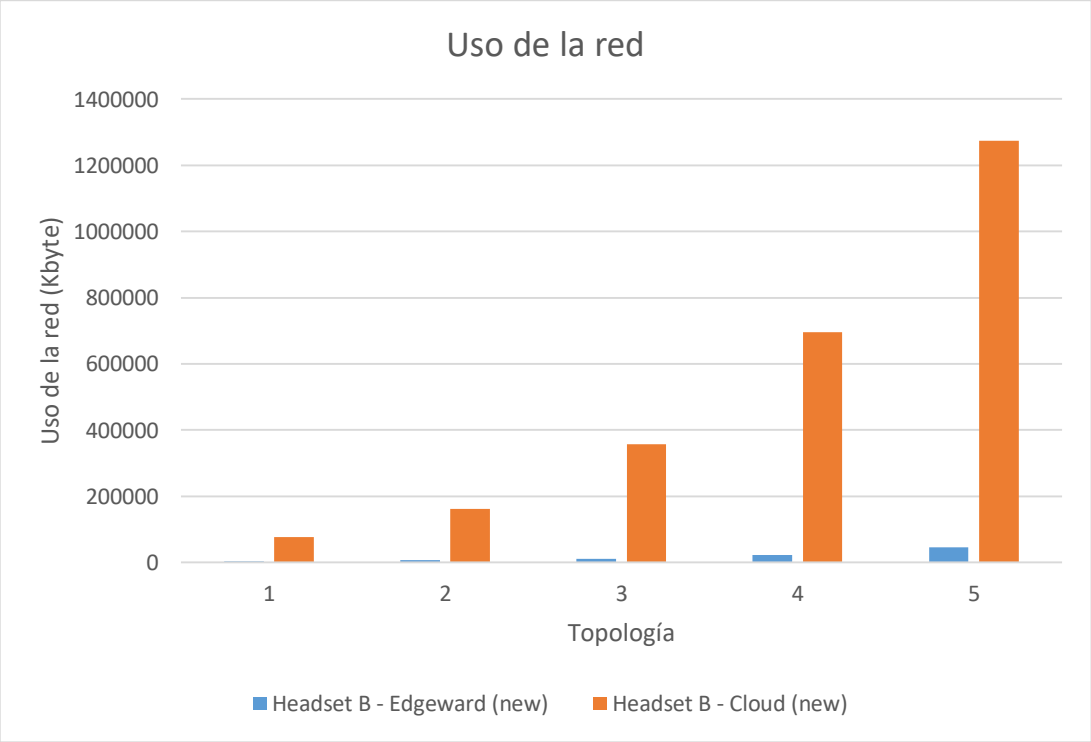


Figura 40: Uso de la red. Ampliación (comparación). Mejora Cloud.

5.2 Resultados obtenidos con las técnicas de planificación implementadas

En este apartado se analizarán conjuntamente los resultados de las simulaciones con ambas estrategias de planificación implementadas; *Priority Scheduling* y *Min Min Scheduling*. Para ello, las compararemos con la planificación implementada por defecto; *Edgeward*s.

Como se indicó anteriormente, ambas nuevas implementaciones se centran en planificar tareas entre módulos. Anteriormente la planificación consistía en un FCFS (*First Come First Serve*) el cual funcionaba bastante bien, ya que las tareas se van generando conforme se van necesitando [1].

Para presentar los resultados, se hará una comparación con el anterior caso base y además, con la ampliación de características de hardware de cada uno de los elementos de la red, para entender como mejora el rendimiento con ellos.

5.2.1 Caso base

Para el caso base tendremos en consideración solo los auriculares B, ya que son los que resultan en más latencia.

Se irán el número de routers WiFi siguiendo la secuencia 1, 2, 4, 8 y 16.

5.2.1.1 Estudio de la latencia en el bucle de control

Configuración	***Edgeward (ms)***	***Priority Scheduling (ms)***	***Min Min Scheduling (ms)***	**Mejora Priority (%)**	**Mejora Min Min (%)**
Config 1	28.18	121.23	26.35	-330.22	6.50
Config 2	28.11	122.34	28.85	-335.16	-2.62
Config 3	29.55	120.54	27.97	-307.94	5.33
Config 4	29.40	126.15	29.51	-329.04	-0.35
Config 5	28.75	121.07	29.32	-321.05	-1.98

Tabla 27: Latencia media. Caso base. Algoritmos implementados.

En la tabla 27 se muestran los resultados de la simulación para la planificación *Edgeward*s, con y sin planificación de tareas. Sin duda la planificación de las tareas no conlleva a una mejora en la latencia final, y, además, en el caso de añadirles prioridad, las empeora.

5.2.1.2 Estudio del uso de la red

Configuración	***Edgeward* (Kbytes)**	***Priority Scheduling* (Kbytes)**	***Min Min Scheduling* (Kbytes)**	**Mejora priority (%)**	**Mejora Min Min (%)**
Config 1	6733.33	7566.67	6686.67	-12.38	0.69
Config 2	12853.33	14736.67	12710	-14.65	1.12
Config 3	24476.67	28713.33	24670	-17.31	-0.79
Config 4	49040	56983.33	49070	-16.20	-0.06
Config 5	96780	113256.67	97146.67	-17.02	-0.38

Tabla 28: Uso de la red. Caso base. Algoritmos implementados.

En la presente tabla se presentan los resultados de la simulación para el consumo de red. Nuevamente, para el caso del *Priority Scheduling*, se consumen bastantes más recursos que sin ningún tipo de planificación. En el caso del *Min Min*, la diferencia de consumo de recursos podría considerarse inapreciable con respecto a la de *Edgeward*s.

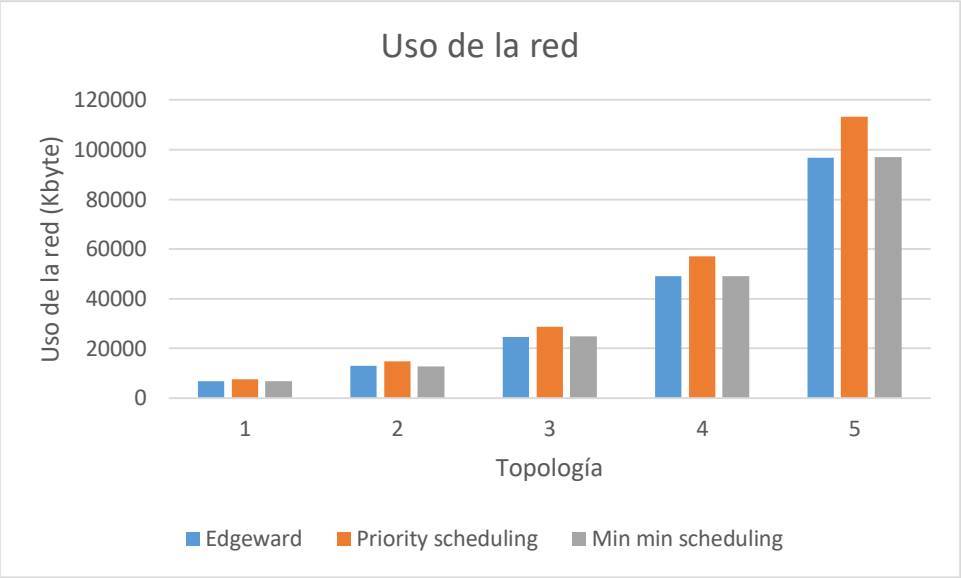


Figura 41: Uso de la red. Caso base. Algoritmos implementados.

En la figura 40 se puede ver como ambas nuevas implementaciones siguen la tendencia a aumentar como lo hace la estrategia implementada por defecto.

5.2.1.3 Estudio del consumo de los recursos del Cloud

Configuración	***Edgeward***	***Priority Scheduling***	***Min Min Scheduling***	**Mejora priority (%)**	**Mejora Min Min (%)**
Config 1	6976.80	6036	6036	13.48	13.48
Config 2	7086.40	6145.60	6145.60	13.28	13.28
Config 3	7305.60	6364.80	6364.80	12.88	12.88
Config 4	7744	6803.20	6803.20	12.15	12.15
Config 5	8620.80	7680	7680	10.91	10.91

Tabla 29: Coste de ejecución en la nube. Caso base. Algoritmos implementados.

En la tabla 29 se ve el coste de ejecución de la aplicación en la nube. En el caso de planificar las tareas, menos son ejecutadas en la nube por lo que resultan en un menor coste (hasta del 13% en el mejor de los casos).

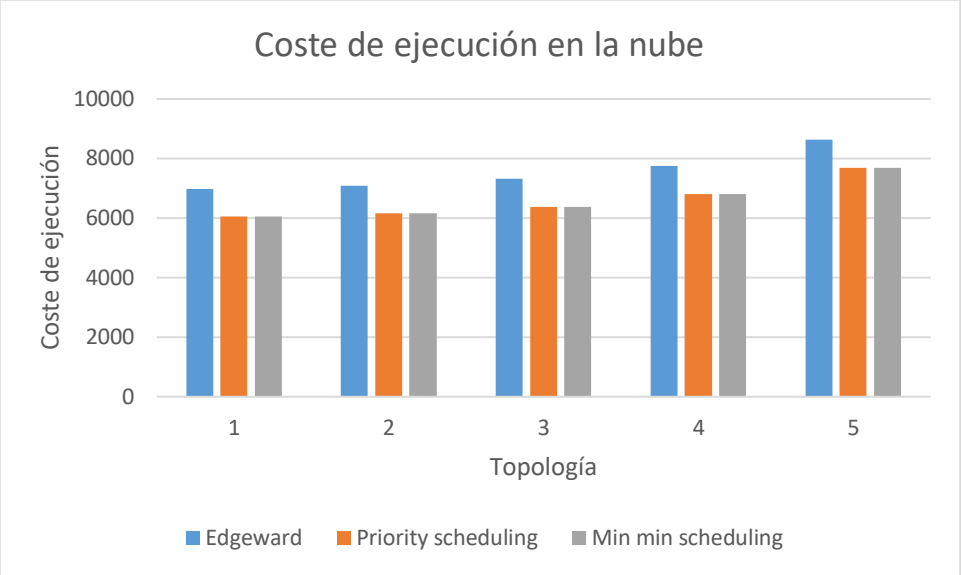


Figura 42: Coste de ejecución en la nube. Caso base. Algoritmos implementados.

5.2.2 Mejoras de hardware

Aunque se ha visto que planificar las tareas no resulta en una menor latencia, el cual es el objetivo que se busca, se completa el estudio mostrando los resultados de estas planificaciones cuando se mejora el hardware de los componentes de red.

5.2.2.1 Mejora del hardware de los smartphones

Para ello se procederá como en los apartados anteriores; solo se escogerá el caso de los auriculares B y se dejarán fijos los números de jugadores por router WiFi y el número de estos en 4. Se irán variando las características en cinco configuraciones diferentes de forma que se vea la evolución progresiva de los objetos de estudio.

- Estudio de la latencia

Configuración	***Edgeward (ms)***	***Priority Scheduling (ms)***	***Min Min Scheduling (ms)***	**Mejora Priority (%)**	**Mejora Min Min (%)**
Config 1	30.17	131.72	29.88	-336.59	0.97
Config 2	36.17	128.11	42.41	-254.23	-17.28

Config 3	12.67	109.10	13.34	-761.36	-5.28
Config 4	12.06	90.15	12.48	-647.78	-3.59
Config 5	12.93	75.19	12.44	-481.67	3.80
Comparación (config 1 vs config 5) %	60.04	42.91	58.37		

Tabla 30: Latencia media. Mejoras smartphones. Implementación algoritmos.

En la tabla se puede ver como una mejora sucesiva de las características de hardware de los terminales de los usuarios acaba resultando en una mejora final de la latencia. Sin embargo, los nuevos tipos de planificación no mejoran los resultados sobre el algoritmo original.

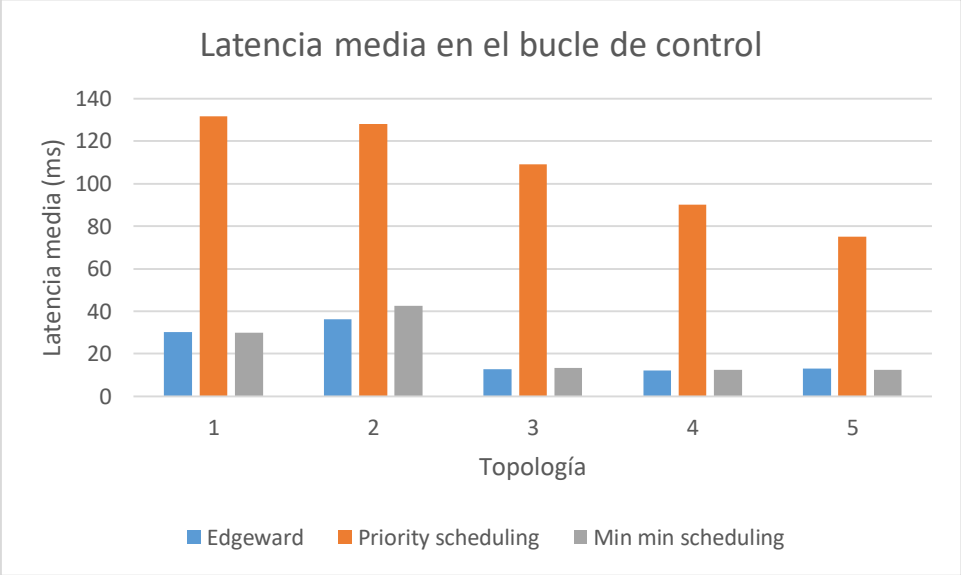


Figura 43: Latencia media. Mejoras smartphones. Implementación algoritmos.

En el gráfico de la figura 42 se demuestra como las nuevas estrategias siguen la tendencia de la original. Conforme se aumentan estas características, la latencia final disminuye pues los terminales tienen más fuerza de computación y pueden hacerse cargo de más tareas.

- Estudio del uso de la red

Mejorando las características de los dispositivos se tiene que finalmente se usan la misma cantidad de recursos de red, como se puede apreciar en el gráfico de la figura 43.

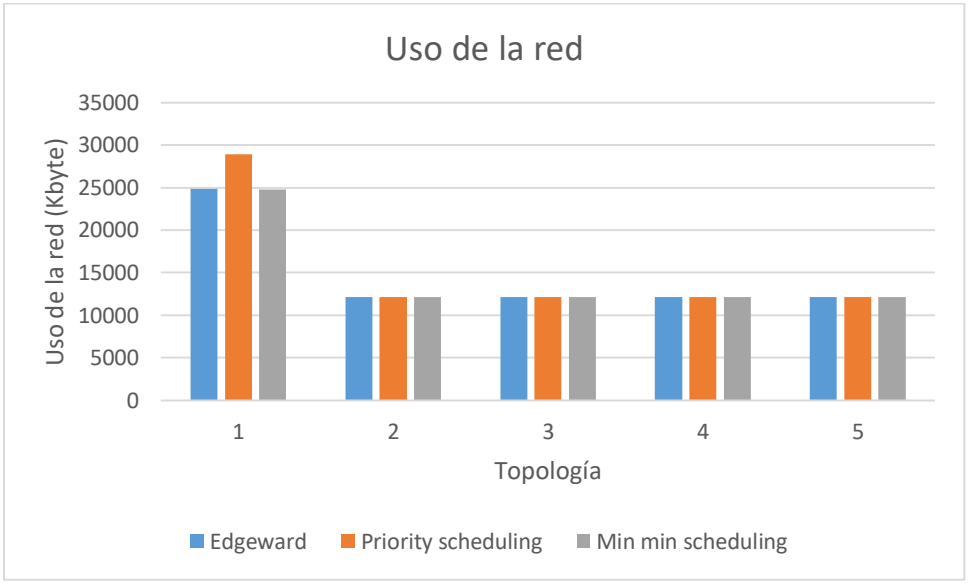


Figura 44: Uso de la red. Mejoras smartphones. Implementación algoritmos.

- Coste de ejecución en la nube

Al igual que con los recursos de red, se tiene que cuando se suben las características de los terminales estos gastan la misma cantidad de recursos.

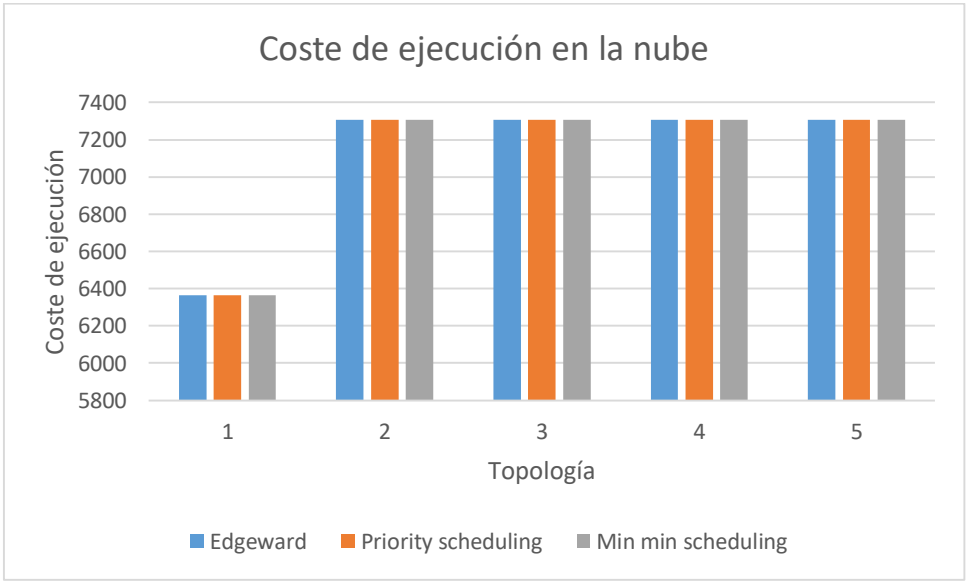


Figura 45: Coste de ejecución en la nube. Mejoras smartphones. Implementación de algoritmos.

5.2.2.2 Mejora del hardware de los routers WiFi

Para el estudio de este caso de uso se propone aumentar las características de los routers WiFi en cinco configuraciones para observar la evolución de los resultados simulados más lentamente.

Nuevamente, se dejan fijos tanto el número de jugadores como el número de routers WiFi.

- Estudio de la latencia

Configuración	***Edgeward (ms)***	***Priority Scheduling (ms)***	***Min Min Scheduling (ms)***	**Mejora Priority (%)**	**Mejora Min Min (%)**
Config 1	29.98	126.03	28.92	-320.37	3.54
Config 2	22.38	101.90	22.86	-355.38	-2.14
Config 3	20.09	85.44	20.52	-325.25	-2.15
Config 4	18.89	67.82	19.65	-259.10	-4.03
Config 5	18.69	49.94	18.79	-167.13	-0.52
Comparación (config 1 vs config 5) %	37.65	60.38	35.03		

Tabla 31: Latencia media. Mejoras routers WiFi. Implementación algoritmos.

Se observa en la tabla 31 que cambiar el hardware de estos dispositivos termina en una menor latencia. En el caso del *Priority Scheduling* puede resultar de hasta un 60% la diferencia entre tener unas características u otras.

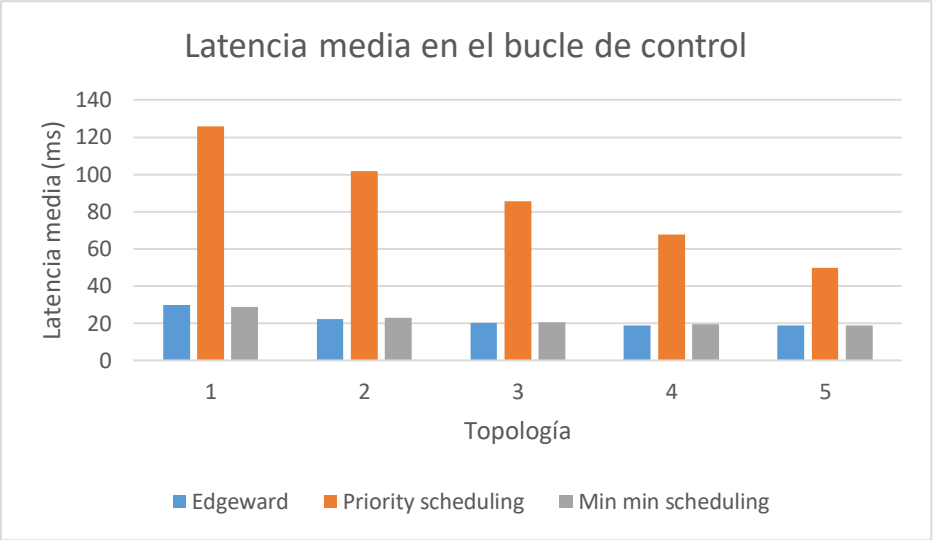


Figura 46: Latencia media. Mejora routers WiFi. Implementación algoritmos.

El gráfico de la figura 46 demuestra que ambas planificaciones siguen la tendencia de disminuir la latencia conforme se van mejorando las características de los dispositivos.

- Estudio sobre el uso de la red

Configuración	***Edgeward* (Kbytes)**	***Priority Scheduling* (Kbytes)**	***Min Min Scheduling* (Kbytes)**	**Mejora Priority (%)**	**Mejora Min Min (%)**
Config 1	24920	28766.67	24796.67	-15.44	0.49
Config 2	25066.67	34566.67	24996.67	-37.89	0.28
Config 3	25416.67	40743.33	25046.67	-60.30	1.46
Config 4	25216.67	46136.67	25283.33	-82.96	-0.26
Confign 5	25450	52743.33	25136.67	-107.24	1.23
Comparación (config 1 vs config 5) %	-2.13	-83.35	-1.37		

Tabla 32: Uso de la red. Mejora routers WiFi. Implementación algoritmos.

Mejorar las características de los routers no influye en un consumo más eficiente de los recursos de red.

- Coste de ejecución en la nube

No supone cambio alguno en los costes de ejecución en la nube.

5.2.2.3 Mejora del hardware de los ISP Gateway

En este caso de uso se variarán las características de los ISP Gateway con el fin de ver la variación sobre los parámetros objetos de estudio en este trabajo.

Para ello, se procederá como en los anteriores casos; número invariante de jugadores y routers WiFi.

Como ya se detalló en el anterior apartado, no se llegaban a resultados de interés al incrementar las prestaciones de estos, por lo que es de la misma forma con estas dos nuevas estrategias de planificación.

La latencia no presenta una variación significativa en este caso.

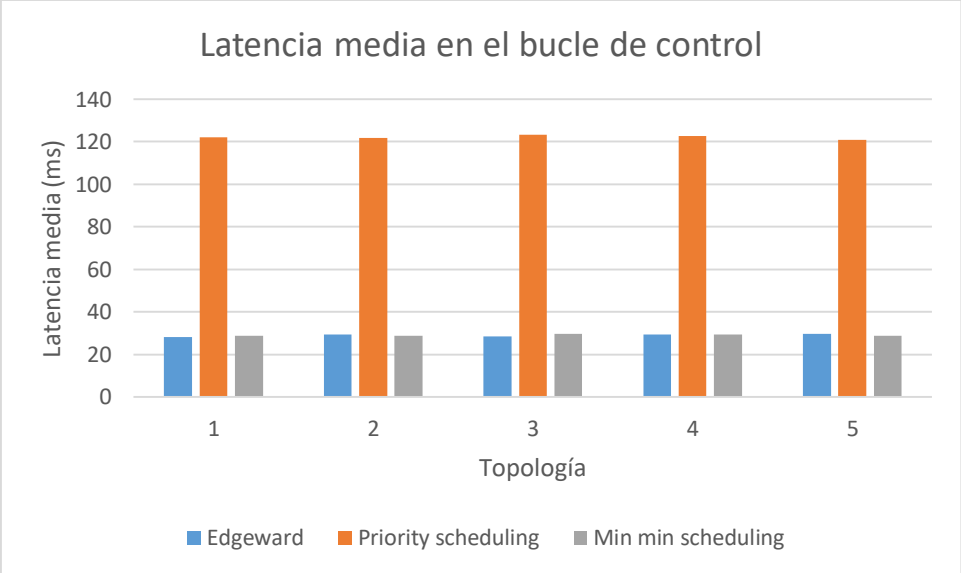


Figura 47: Latencia media. Mejora ISP. Implementación algoritmos.

De la misma forma, el uso de la red en este caso permanece constante.

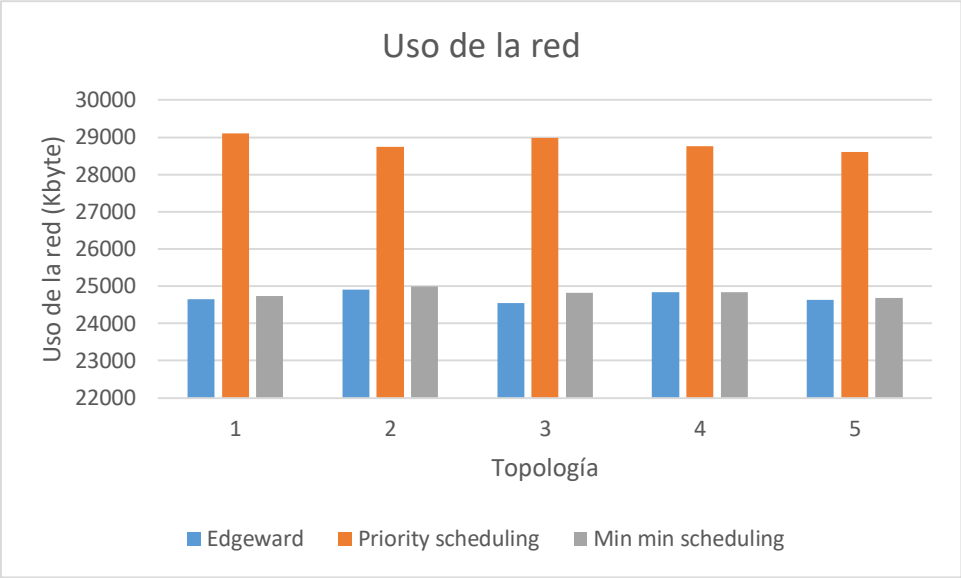


Figura 48: Uso de la red. Mejora ISP. Implementación algoritmos.

5.2.2.4 Mejora del hardware del Cloud

El último caso de uso, se aumentarán las prestaciones del *Cloud*. El estudio procederá de la misma forma que en todos los casos anteriores.

Igual que en el anterior capítulo aumentar las capacidades del *Cloud* no resultaba en un cambio significativo de los resultados de las simulaciones, en estas sucede de igual forma.

La latencia no se ve afectada por estos cambios;

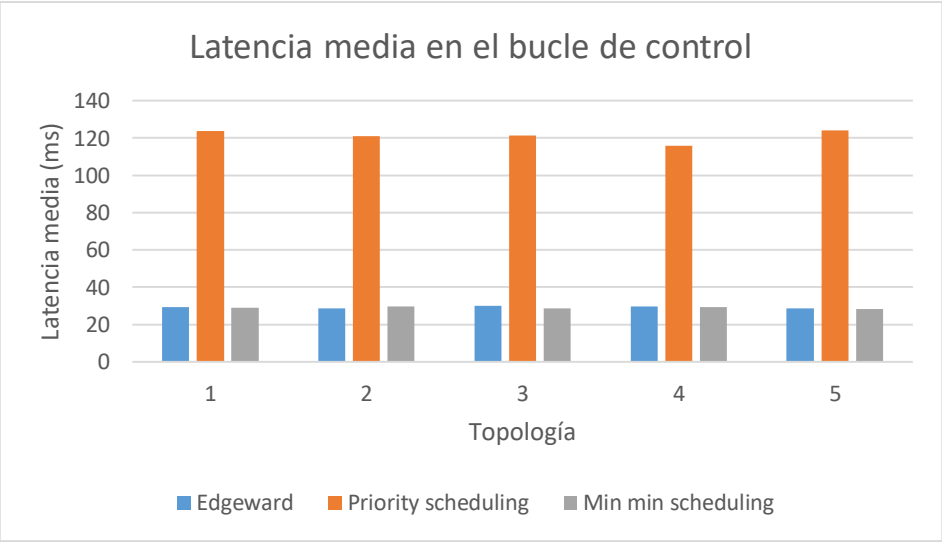


Figura 49: Latencia media. Uso de la red. Implementación algoritmos.

El uso de los recursos de red también permanece invariante

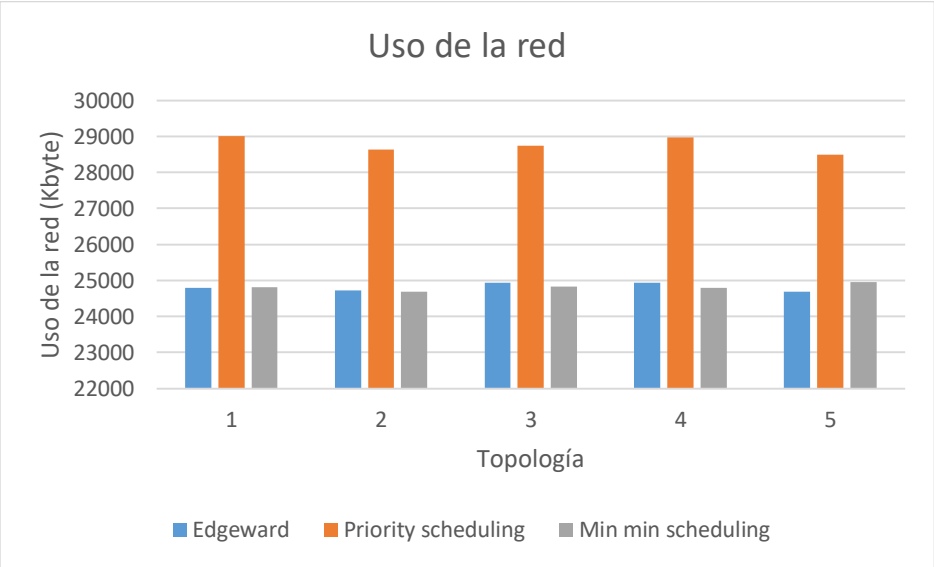


Figura 50: Latencia media. Uso de la red. Implementación algoritmos.

6 CONCLUSIONES

En este último capítulo se expondrán las conclusiones extraídas tras el desarrollo de este presente estudio.

Para empezar este proyecto se indagó sobre las bondades de *Fog Computing* y que ventajas podría tener sobre paradigmas de computación actuales como es el *Cloud* Computing, el cual es comparado con la tecnología a estudiar en numerosas ocasiones en el presente documento.

Además, se requirió de un exhaustivo estudio del entorno de simulación a emplear, *iFogSim*. Ya que la documentación en las redes no es abundante sobre este tema, se ha necesitado de mucho tiempo para la comprensión de su funcionamiento. Para un mejor entendimiento de cómo podría funcionar el nuevo paradigma de programación que se nos atañe, se propone la creación de una amplia base de topologías para ver la evolución del comportamiento de la red en las diferentes situaciones propuestas. Como se ha comprobado experimentalmente, el aumentar las características de hardware de los dispositivos finales resulta clave a la hora de un mejor rendimiento en cuanto a términos de latencia, llegándose a reducir hasta un 50%. Este hecho puede resultar ventajoso, ya que los usuarios tienden a poseer mejores tecnologías conforme avanza el tiempo, y supondría no invertir más esfuerzos en cambiar otros dispositivos de red. En cambio, el cambiar las características de otros terminales no acaba resultando en una mejoría notable a excepción de los auriculares que recogen los datos en crudo, los cuales también acaban dependiendo del usuario final en este caso de uso. Para otro tipo de aplicaciones, como podrían ser de emergencias médicas supondría que para obtener latencias mínimas habría que invertir en mejores sensores que permitan recoger y, si tienen fuerza de computación suficiente, procesar las muestras más rápidamente.

Una vez observado dicho comportamiento, se propone el estudio e implementación de estrategias de planificación adicionales. Tras estudiar la literatura científica disponible al respecto, se observó que se centraban en una planificación de tareas, las cuales las se disponen en módulos; estos se definen por cada caso de uso y quedan encargados de realizar distintos bloques de tareas, en *EEG TRACTOR BEAM* si se estudia el ejemplo del módulo *CONCENTRATION_CALCULATOR* es el encargado de calcular el nivel de concentración de cada jugador, por ejemplo. Por ser la naturaleza del presente trabajo de tipo experimental, se decide estudiar una implementación de las tareas que constituyen dichos módulos en lugar de la tradicional, es decir, de cómo se comunican entre ellos. A la vista de los resultados anteriormente expuestos, se puede concluir que con los tipos de planificación escogidos no se produce una disminución aparente de la latencia, llegando en el caso de *Priority Scheduling*, a aumentarla. También se observa en cuanto a uso de

la red un ligero aumento en el caso de la planificación por prioridad pero un mejor rendimiento en cuanto a costes de ejecución en *Cloud*, para ambas planificaciones. El ahorro en recursos *Cloud* podría indicar que menos tareas están siendo mandadas a la nube para su ejecución porque estas se están repartiendo entre los dispositivos de las capas inferiores.

Por último, hay que destacar la enorme ventaja que supone el uso de simuladores para estudios de esta clase ya que, aunque el adecuado manejo de este suponga muchas horas invertidas en el mismo, resulta en un ahorro económico por no realizar el despliegue de la infraestructura necesaria para la tecnología estudiada. También facilita las mediciones de las observaciones, puesto que son fácilmente programables y por esta misma razón, permite bastante grado de flexibilidad en las condiciones impuestas a las simulaciones.

6.1 Líneas de futuro

Durante el desarrollo de este proyecto ha sido necesaria la lectura de mucha literatura científica en lo que concierne a esta tecnología, pero ha quedado claro que *Edge Computing* es una tecnología sobre la que merece la pena poner esfuerzos.

A pesar de que el presente trabajo fin de grado se haya centrado a estudiar la latencia, hay que estudiar algunos otros temas:

- La seguridad. Se usarán dispositivos que tengan recursos disponibles para la computación de tareas. Es fundamental tener en cuenta estrategias de encriptado y seguridad adicionales para asegurar fuga de información sensible.

- La energía consumida. Al tratarse de un sistema de computación descentralizado es necesario tener algoritmos eficientes con la energía pues el total podría suponer más energía utilizada que en sistemas más centralizados.

Aún con el estudio de la latencia salen otras posibles líneas de estudio como otros casos de uso (aplicaciones sanitarias, de servicios de emergencia ante catástrofes) en donde tener una latencia mínima sea crucial para el correcto funcionamiento de las aplicaciones. También se puede considerar la inclusión de sistemas de aprendizaje automático, lo cual conseguiría una respuesta mucho más óptima, un ejemplo de uno de ellos lo podemos encontrar en [15], en el cual mejora la ubicación de recursos.

7 ANEXOS

Cómo se había mencionado previamente, no se van a detallar todos los resultados de la ampliación de topologías, ya que, al ser unos resultados tan extensos haría tediosa la lectura del presente documento. Pero ha sido necesario un estudio de todos los posibles casos de uso y por ello se quieren añadir como anexos.

Hay 64 casos de usos, distintos conjuntos de variaciones de las mejoras de hardware anteriormente detalladas. Para explorar las distintas combinaciones se crea una hoja de Excel llamada TRACKING CASES en la que se especifica qué parámetros se han modificado y cuáles no, y el correspondiente nombre la hoja de Excel que contiene las tablas y gráficos como resultados de las simulaciones con dichos valores.

En el documento TRACKING CASES, algunos casos de uso salen marcados en un color anaranjado, el cual significa que es un caso de uso al que se le ha dado importancia, y como tal, detallado en esta memoria.

Por cada caso de uso hay un documento de Excel de cuatro hojas en las que indican los resultados de la simulación para la latencia, el coste de ejecución en la nube, el tiempo de ejecución de la simulación y los recursos de red consumidos (A en la figura 50). En cada una de estas hojas, hay tres gráficos; uno es una comparación entre el caso base y el nuevo caso de uso a considerar (siempre con el headset B, que es el peor escenario) con la estrategia de planificación *Edgeward*, otro de la misma forma con la planificación *Cloud* y el último con una comparativa entre ambas planificaciones (B en la figura 50 También, a la hora de exponer los resultados de cada simulación hay tres columnas adicionales marcadas en verde con los porcentajes de mejora entre cada tecnología y con la comparación de la mejora entre ambas planificaciones (C en la figura 50).

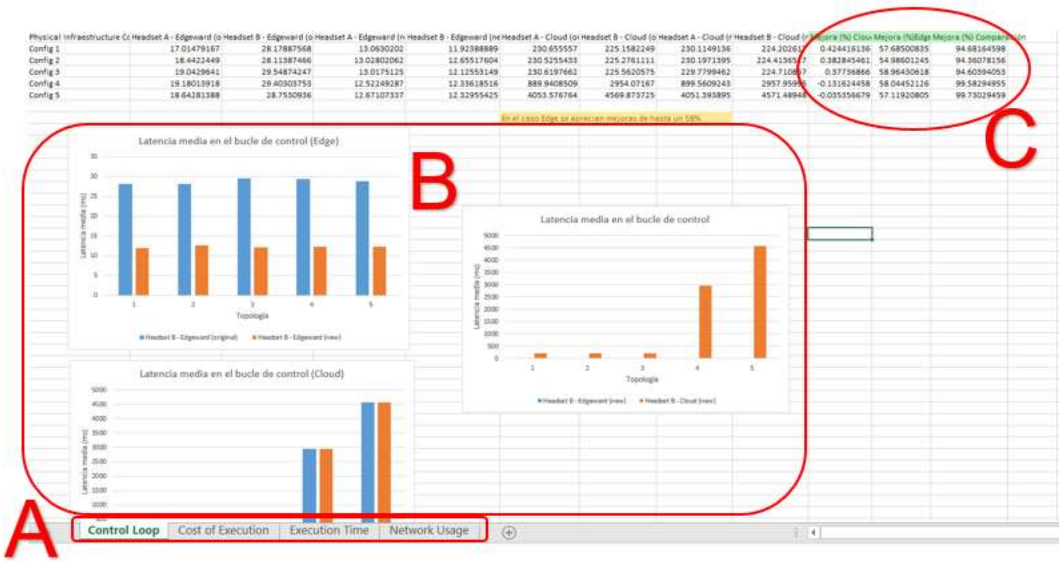


Figura 51: Vista hoja de análisis de resultados

8 REFERENCIAS BIBLIOGRÁFICAS

[1] Dave Evans, The Internet of Things: How the Next Evolution of the Internet Is changing Everything, 2011.

[2] Amir Vahid Dastjerdl and Rajkumar Buyya. Fog Computing Helping the Internet of Things Realize its Potential. *Cloud* Cover, 2016, 40-44.

[3] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, Rajkumar Buyya. CloudSim: a toolkit for modelling and simulation of cloud computing environments and evaluation of resource provisioning algorirthms.

[4] Rodrigo N. Calheiros, Rajiv Ranjan, César A. F. De Rose, and Rajkumar Buyya. CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services.

[5] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala et al. All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey.

[6] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch Glitho, et al. A Comprenhensive Survey on Fog Computing: State-of-the-art and Research Challenges.

[7] Cisco IOx: https://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html

[8] LocalGrid Fog Computing Platform Datasheet: http://www.localgridtech.com/wp-content/uploads/2015/02/LocalGrid-Fog-Computing-Platform-Datasheet.pdf

[9] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N. Calheiros, Soumya K. Ghosh and Rajkumar Buyya. Fog Computing: Principles, Architectures, and Applications, Chapter 4.

[10] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, RajkMumar Buyya. IFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Enviroments.

[11] John K. Zao, Tchin Tze Gan, Chun Kai You, Sergio José Rodríguez Méndez, et al. Augmented Brain Computer Interaction Based on Fog Computing and Linked Data, IEE, 2014.

[12] Redowan Mahmud, Rajkumar Buyya. Modelling and Simulation of Fog and Edge Computing Environments using iFogSim Toolkit. Introduction to Fog and Edge Computing, Chapter 17.

[13] Tejaswini Cloudhari, Melody Moh, Teng-Sheng Moh. Prioritized Task Scheduling in Fog Computing.

[figura 4: 2 de mi biografi]

[14] Sabihe Kabirzadeh, Dadmehr Rahbari, Mohsen Nickray. A hyper heuristic Algorithm for Scheduling of Fog Networks.

[15] Dadmehr Rahbari, Mohsen Nickray. Scheduling of fog networks with optimized knapsack by symbiotic organisms search, IEE.