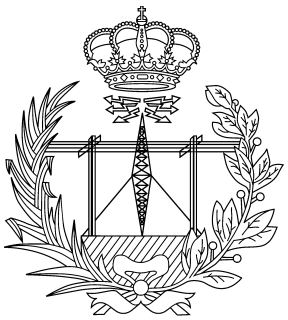
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Trabajo Fin de Máster

DISEÑO, DESARROLLO Y EVALUACIÓN DE ALGORITMOS DE COMPUTACIÓN DISTRIBUIDA EN UNA PLATAFORMA FOG-CLOUD

(Design, development and evaluation of distributed computing algorithms over a Fog-Cloud platform)

Para acceder al Título de

Máster Universitario en Ingeniería de Telecomunicación

Autor: Neco Villegas Saiz

Julio- 2023

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE MÁSTER

Realizado por: Neco Villegas Saiz
Directores del TFM: Luis Francisco Diez Fernández, Ramón Agüero Calvo
Título: "DISEÑO, DESARROLLO Y EVALUACIÓN DE ALGORITMOS DE COMPUTACIÓN DISTRIBUIDA EN UNA PLATAFORMA FOG-CLOUD"
Title: "Design, development and evaluation of distributed computing algorithms over a Fog-Cloud platform"

Presentado a examen el día:

para acceder al Título de

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Composición del tribunal:
Presidenta (Apellidos, Nombre): Mercedes Granda
Secretario (Apellidos, Nombre): Amparo Herrera
Vocal (Apellidos, Nombre): Alberto Eloy García

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFM
(solo si es distinto del Secretario)

V^{o} B^{o} del Subdirector

Trabajo Fin de Máster N^{o}
(a asignar por Secretaría)

Índice general

Índice de figuras **4**

Índice de tablas **5**

Acrónimos **6**

1 Introducción **9**
1.1. Motivación 9
1.2. Objetivos 10
1.3. Estructura 11

2 Estado del arte **12**
2.1. Computación distribuida aplicada a IoT 12
2.1.1. Cloud Computing 12
2.1.2. Fog Computing 13
2.1.3. Arquitecturas IoT-Fog-Cloud 14
2.1.4. Algoritmos de offloading 17
2.2. Trabajos relacionados 18

3 Plataforma de simulación **20**
3.1. Descripción general 20
3.2. Nodo Fog 22
3.3. Nodo Cloud 26
3.4. Nodo Master 26
3.5. Dockerización 29
3.6. Configuración 31

4 Desarrollo e implementación de algoritmos de offloading **33**
4.1. Solución de referencia 33
4.2. Propuesta de algoritmo 34

5 Resultados **39**

5.1. Colaboración Fog y Cloud 40
5.2. Análisis de rendimiento en Fog 42
5.3. Entorno realista de generación de tráfico 43
5.4. Comparativa de rendimiento con otras soluciones 45

6 Conclusiones y líneas futuras **48**

Bibliografía **50**

A Anexo **55**

Índice de figuras

2.1. Arquitectura IoT-Fog-Cloud. 15
2.2. Requisitos de latencia de servicios IoT en un escenario Fog-Cloud. 16

3.1. Vista general de la plataforma de Fog-Cloud. 21
3.2. Implementación de la plataforma de Fog-Cloud. 23
3.3. Estructura de los paquetes. 24
3.4. Diagramas de flujo del nodo Fog. 25
3.5. Diagramas de flujo del nodo Cloud. 27
3.6. Diagrama de flujo del nodo Master. 29
3.7. Despliegue de los contenedores Docker. 30

4.1. Esquema funcional del algoritmo Round Robin. 33
4.2. Sistema Fog-Cloud con las colas de aplicaciones (Q_m), las colas virtuales de energía (G_n) y las colas de los procesadores. 34

5.1. Uso del Cloud frente a la variación de la tasa de tráfico agregada. 40
5.2. Coste de energía promedio frente a la variación de la tasa de tráfico agregada. 41
5.3. Evolución de las colas debido a la variación del umbral de energía. 43
5.4. Evolución de las colas debido a la variación de la tasa de tráfico de una aplicación. 44
5.5. Coste del Cloud y consumo de energía en función de V y E_{th}. 45
5.6. Diagrama comparativo entre algoritmos y configuraciones. 46

Índice de tablas

2.1. Parámetros y métricas de rendimiento considerados en algoritmos de la literatura reciente que suponen escenarios similares al algoritmo propuesto. 19

3.1. Parámetros configurables en la plataforma. 32

4.1. Símbolos y variables del modelo del sistema. 35

5.1. Configuración común de la plataforma para la campaña de simulaciones. 39
5.2. Configuración de la simulación del entorno realista. 44
5.3. Puntos de intersección en función del parámetro V. 45

Acrónimos

ADMM Alternating Direction Method of Multipliers.

AWS Amazon Web Services.

Deep RL Deep Reinforcement Learning.

IIoT Industrial Internet of Things.

ILP Programación Lineal Entera.

IoT Internet of Things.

M2M Machine to Machine.

MEC Mobile Edge Computing.

ML Machine Learning.

NFV Network Functions Virtualization.

OPC Consorcio OpenFog.

QoE Quality of Experience.

RFID Identificación por Radio Frecuencia.

SDN Software-Defined Networking.

tc Traffic Control.

VN Vehicular Network.

Resumen Ejecutivo

En el presente trabajo se diseña y desarrolla una plataforma de tres niveles (IoT-Fog-Cloud) utilizando técnicas de virtualización, que se utiliza posteriormente para implementar varios escenarios, con nodos que tienen diferentes características, emulando el comportamiento de dispositivos Fog y Cloud. A continuación, se presentan políticas de offloading para arquitecturas Fog-Cloud que consideran diferentes parámetros de rendimiento. Se emplea la Teoría de Lyapunov para introducir un algoritmo de offloading que equilibra el consumo de energía en los nodos Fog y el coste monetario de usar el Cloud. El algoritmo propuesto es capaz de encontrar un equilibrio entre estos dos parámetros, al tiempo que garantiza la estabilidad del sistema y los requisitos de retraso. Se realiza una campaña de simulaciones en la plataforma desarrollada que permite mostrar el correcto funcionamiento de la solución. Además, al ajustar los parámetros operativos del algoritmo, se pone de manifiesto que la solución propuesta es capaz de adaptar su comportamiento a diferentes objetivos, permitiendo evaluar su rendimiento bajo configuraciones realistas.

Executive Abstract

In this work, we design and develop a three-tier platform (IoT-Fog-Cloud) using virtualization techniques, which can be used to implement scenarios with nodes having different characteristics, mimicking the features of Fog and Cloud. Subsequently, we present offloading policies for Fog-Cloud architectures that consider various performance parameters. We employ Lyapunov Theory to introduce an offloading algorithm that balances energy consumption at Fog nodes and the monetary cost of using the Cloud. The proposed algorithm can find a trade-off between these two parameters while ensuring system stability and meeting delay requirements. We conduct a thorough simulation campaign on the developed platform to assess the feasibility of the appropriate operation point of the solution. Furthermore, by adjusting the operational parameters of the algorithm, we show that it can adapt its behavior to different targets, and we evaluate its performance under realistic configurations.

1 Introducción

1.1. Motivación

Hoy en día, el número de servicios en la nube está aumentando continuamente, especialmente aquellos ofrecidos por los principales proveedores como Microsoft Azure, Amazon Web Services (AWS) y Google Cloud. Este incremento en la demanda viene motivado, entre otros aspectos, por el aumento de servicios de Internet of Things (IoT) e Industrial Internet of Things (IIoT). Al mismo tiempo, existe una creciente implantación de redes 5G, cuyas tecnologías subyacentes ofrecen varias ventajas, por ejemplo, en términos de latencia, disponibilidad o fiabilidad. Por todo ello, son muchos los sectores que ven ahora una oportunidad para implementar diferentes servicios de IoT e IIoT. De hecho, junto con la masiva implementación del 5G, el número de conexiones IoT ya ha alcanzado los 14.6 mil millones, y se espera que supere los 30 mil millones en 2027[1].

Esto ha generado un interés creciente en la integración de IoT con la computación en la nube o Cloud Computing, puesto que la combinación de estas dos tecnologías presenta un gran potencial. Sin embargo, el previsible fuerte aumento de servicios de IoT e IIoT, y su implementación en nuevos sectores con requisitos más estrictos, pueden llevar a situaciones en las que el Cloud Computing no sea del todo rentable en términos de latencia o precio, entre otros indicadores. Como alternativa, la computación en la niebla o Fog Computing ha surgido como una extensión del Cloud Computing, proporcionando servicios de cómputo de alto rendimiento para aplicaciones de IoT. Este modelo se basa en transferir tareas a un nodo cercano geográficamente, en lugar de a una nube remota, lo que podría aportar diferentes ventajas.

Una de las principales diferencias entre Cloud y Fog Computing es la escala de los componentes de hardware. El Cloud Computing proporciona alta disponibilidad de recursos de procesado con un consumo de energía relativamente alto (centros de datos), mientras que Fog Computing proporciona una disponibilidad moderada de recursos, con un consumo de energía más bajo (servidores pequeños, routers, switches, gateways, etc.). Aunque ambas opciones, Cloud y Fog, se pueden usar de manera independiente, no es necesario elegir únicamente una de ellas, ya que las dos soluciones se complementan entre sí, y la cooperación entre ellas podría generar, de hecho, un uso óptimo de los recursos, mejorando las capacidades del sistema en términos de eficiencia energética, reducción de costes, y procesamiento, agregación y almacenamiento de datos. Un elemento orquestador podría gestionar esta cooperación entre Cloud y Fog.

Este enfoque conduce a arquitecturas de tres niveles (IoT-Fog-Cloud), que aprovechan las ventajas de ambos modelos, reduciendo el retardo en el servicio de tareas y el consumo de energía. Además, se alivia la fuerte dependencia a los proveedores de servicios en el Cloud, lo que lleva a una reducción de los

[1] https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/iot-connections-outlook

costes económicos. Para consolidar estas posibles ventajas, es necesario utilizar un esquema de asignación de carga de trabajo en un sistema de cooperación IoT-Fog-Cloud que produzca un rendimiento óptimo para diferentes escenarios y requisitos heterogéneos.

La mayoría de los conceptos discutidos anteriormente también son aplicables a la computación en el borde o Edge Computing. Así, Fog y Edge Computing mueven las tareas de cómputo más cerca de los nodos finales, aunque estos paradigmas no son idénticos. El Consorcio OpenFog (OPC, por sus siglas en inglés) establece una clara distinción. Fog Computing sigue un paradigma jerárquico, proporcionando cómputo, redes, almacenamiento, control y aceleración en cualquier lugar, desde la nube hasta las "cosas", mientras que Edge Computing tiende a estar limitada al cómputo en el borde de la red. En el presente trabajo se emplea el término Fog Computing, pero todo lo expuesto también es aplicable a Edge Computing.

Frente a todas las ventajas que aportan las arquitecturas IoT-Fog-Cloud existen algunos desafíos que deben abordarse en relación a los mecanismos de offloading de tareas de cómputo. El primero es la cantidad de carga de trabajo a desplazar desde el Fog hasta las instancias en el Cloud, considerando una amplia gama de parámetros de rendimiento. Otra cuestión a abordar se refiere al lugar donde se toma esta decisión.

1.2. Objetivos

En el presente trabajo se propone un algoritmo multiobjetivo de offloading en escenarios de IoT o IIoT donde se dispone de una arquitectura IoT-Fog-Cloud. El algoritmo propuesto es adaptativo y considera conjuntamente el consumo de energía en el Fog y el coste monetario del Cloud. Además, se plantea el desarrollo de una plataforma que permita la simulación de este y otros algoritmos en un entorno controlado y altamente configurable. Se pretende, por tanto, disponer de un marco de experimentación que siga la arquitectura IoT-Fog-Cloud, donde poder llevar a cabo campañas de medidas sistemáticas, para afrontar el análisis y la comparación de diferentes soluciones de offloading en función de diversos parámetros.

En definitiva, se plantean los siguientes objetivos a lo largo del proyecto:

- Basándose en una arquitectura IoT-Fog-Cloud, se desarrolla una plataforma que utiliza técnicas de virtualización.

- Se propone un esquema de offloading o asignación dinámica de la carga de trabajo que tiene en cuenta el consumo de energía y el coste monetario como principales parámetros, en un entorno aleatorio y no controlado.

- Se aborda el problema de optimización estocástica resultante, aplicando la Teoría de Lyapunov, de modo que el problema se reduce a uno de estabilización de un sistema de colas. El problema de optimización resultante puede resolverse con el algoritmo de drift-plus-penalty, que corresponde a una secuencia de problemas de Programación Lineal Entera (ILP, por sus siglas en inglés).

- Se lleva a cabo un análisis exhaustivo del algoritmo propuesto en diferentes escenarios y bajo condiciones heterogéneas. El desempeño del algoritmo también se compara con otras soluciones.

1.3. Estructura

En esta sección se detalla la estructura del documento, el cual consta de 6 capítulos que se describen a continuación.

Capítulo 2. Estado del arte

Se realiza una revisión de la literatura existente relacionada con la temática del trabajo. En primer lugar, se introduce el concepto de Cloud Computing y, posteriormente, de Fog Computing. Se abordan las ventajas y desventajas de ambos paradigmas de computación distribuida, centrándose en aquellas que resultan de interés en entornos de IoT e IIoT. Finalmente, se presentan las arquitecturas IoT-Fog-Cloud, las cuales tratan de aunar lo mejor de ambos paradigmas. De estas arquitecturas subyace la necesidad de un uso eficiente de los recursos.

Capítulo 3. Plataforma de simulación

A continuación, se introduce la plataforma desarrollada. Para ello, en primer lugar, se presenta una descripción general, donde se muestran las principales funcionalidades implementadas, así como un esquema general de la plataforma. Se describen los tipos de nodos presentes en la plataforma y su papel dentro de la misma. Finalmente, se hace hincapié en las posibles configuraciones y en la facilidad a la hora de crear un gran abanico de escenarios de simulación.

Capítulo 4. Propuesta de algoritmo de offloading

Una vez que se dispone de una visión general de la plataforma de simulación, se presentan los algoritmos de offloading que se han propuesto. En primer lugar, se implementa un algoritmo sencillo, como solución inicial de referencia. A continuación, se plantea un algoritmo adaptativo de mayor complejidad, basado en la Teoría de Lyapunov. En esta segunda solución se tienen en cuenta métricas de rendimiento, en concreto se lleva a cabo un trade-off entre el consumo de energía y el coste monetario debido al uso del Cloud. En todos los casos se busca asegurar la estabilidad del sistema.

Capítulo 5. Resultados

Se completa una amplia campaña de medidas en diferentes escenarios. En el primero, se busca ver el desempeño de los algoritmos propuestos en un entorno de colaboración Fog-Cloud. A continuación, se elimina la posibilidad de acceder a la elevada capacidad de cómputo de las instancias Cloud, con el objetivo de analizar la estabilidad del sistema en condiciones cada vez más exigentes y restrictivas. En el tercer escenario, se busca replicar un caso más práctico y próximo a la realidad. Finalmente, se realiza un último análisis en términos de consumo de energía, coste monetario, latencia y ocupación de las colas.

Capítulo 6. Conclusiones y líneas futuras

Por último, se exponen las conclusiones extraídas de la realización de este trabajo. Además, se plantean posibles líneas futuras que aborden aspectos adicionales que puedan aportar mayor riqueza a las simulaciones realizadas en la plataforma, así como ampliaciones en el modelo que desemboquen en nuevas estrategias de offloading.

2 Estado del arte

2.1. Computación distribuida aplicada a IoT

Se puede ver la IoT como una red dinámica y global de objetos inteligentes, autoconfigurables y heterogéneos interconectados, tales como sensores, actuadores, sistemas de Identificación por Radio Frecuencia (RFID, por sus siglas en inglés) o computadoras integradas. Desde su definición en 1999, se ha convertido en una de las tecnologías más disruptivas del siglo XXI. Permite la generación de escenarios de computación ubicua y omnipresente, donde los dispositivos conectados se integran de manera transparente en cualquier entorno, creando una infraestructura interconectada y en constante comunicación. Esto posibilita la recopilación de datos en tiempo real, la automatización de tareas, la optimización de procesos y la toma de decisiones basada en información actualizada y precisa. La IoT abre nuevas oportunidades en sectores como la salud, la industria, el transporte, la agricultura, el hogar inteligente y muchas otras áreas [1-5].

En el presente capítulo, se presentan los conceptos de Cloud Computing y Fog Computing, centrándose en su aplicabilidad en entornos IoT e IIoT. Cloud e IoT han tenido un origen y evolución independiente, sin embargo, en los últimos años, se han identificado, en la literatura relacionada, varias ventajas mutuas derivadas de su integración. En este contexto, Fog Computing nace como una extensión de Cloud Computing, aportando soluciones a las limitaciones en esta integración IoT-Cloud. Esto desemboca en el desarrollo de arquitecturas IoT-Fog-Cloud.

2.1.1. Cloud Computing

Cloud Computing proporciona recursos escalables, bajo demanda y virtualizados para los usuarios. Ofrece acceso a un conjunto compartido de recursos de cómputo provistos con un esfuerzo mínimo de gestión. Además, es común que los recursos se proporcionen mediante un modelo de "*pay as you go*" en el que los usuarios sólo pagan por los recursos que utilizan. Evitar la sobreasignación y la subasignación de recursos y reducir el coste de implementar hardware son algunas motivaciones para que las empresas lleven su negocio a la nube [6]. Se pueden considerar como modelos de implementación las nubes privadas, comunitarias, públicas e híbridas. Los usuarios pueden elegir uno de estos modelos en función de sus necesidades.

El Cloud Computing tiene capacidades prácticamente ilimitadas en términos de almacenamiento y potencia de procesamiento. Los centros de datos son componentes fundamentales del Cloud Computing. Estos centros son instalaciones físicas diseñadas para albergar servidores, almacenamiento y otros equipos necesarios para proporcionar servicios en la nube, conformando una infraestructura robusta y escalable. Por todo ello, el Cloud Computing es una tecnología madura que permite dar solución a la mayoría

de los problemas de IoT, al menos parcialmente [7]. La IoT puede beneficiarse de las capacidades y recursos virtualmente ilimitados de la nube para compensar sus limitaciones tecnológicas (por ejemplo almacenamiento, procesamiento o energía). Fundamentalmente, el Cloud actúa como una capa intermedia entre los dispositivos IoT y las aplicaciones, ocultando toda la complejidad y brindando las funcionalidades necesarias para implementar estas últimas.

En resumen, los principales aspectos de los que IoT se beneficia del paradigma IoT-Cloud son los que se indican a continuación.

- **Almacenamiento.** La IoT genera una gran cantidad de datos no estructurados o semiestructurados. Estos datos poseen las característica típicas del Big Data [8]: volumen, variedad y velocidad (i.e. frecuencia de generación). Por lo tanto, se requiere afrontar su recolección, acceso, procesamiento, visualización, archivado, compartición y búsqueda. El Cloud se presenta como la solución conveniente y rentable para manejar tal cantidad de información.

- **Procesamiento.** Los dispositivos IoT tienen capacidades de cómputo limitadas. Las capacidades de procesamiento tan elevadas del Cloud, junto con su modelo bajo demanda, permiten satisfacer adecuadamente las necesidades de procesamiento de la IoT.

- **Comunicación.** Uno de los requisitos de la IoT es permitir que los dispositivos habilitados para IP se comuniquen a través de hardware dedicado, y el soporte para dicha comunicación puede resultar muy costoso. El Cloud ofrece una solución efectiva y económica para conectar, rastrear y gestionar cualquier cosa desde cualquier lugar y en cualquier momento, utilizando portales personalizados y aplicaciones built-in [9]. Gracias a la disponibilidad de redes de alta velocidad, permite la monitorización y el control de dispositivos remotos, su coordinación, sus comunicaciones y el acceso en tiempo real a los datos que generan [9-11].

- **Otras capacidades.** La IoT se caracteriza por una alta heterogeneidad de dispositivos, tecnologías y protocolos. Por lo tanto, la escalabilidad, interoperabilidad, confiabilidad, eficiencia, disponibilidad y seguridad pueden resultar muy difíciles de garantizar. La integración con el Cloud resuelve la mayoría de estos problemas y también proporciona ventajas adicionales como facilidad de acceso y de uso, y reducción de los costes de implementación [5, 12, 13].

2.1.2. Fog Computing

El crecimiento exponencial del número de dispositivos IoT requiere encontrar una arquitectura adecuada, capaz de procesar y almacenar todos los datos generados. Si bien las arquitecturas basadas en el Cloud se utilizan actualmente con ese propósito, se vislumbra un nuevo paradigma de computación que busca escalar y optimizar las infraestructuras de la IoT. El concepto de Fog Computing fue introducido por Cisco en [14]. Este término se utiliza tanto por la analogía con la niebla y la nube [15], como por la expresión “From cOre to edGe” [16]. La niebla representa una nube más cercana al dispositivo (o al borde de la red), lo que implica que los recursos de cómputo y almacenamiento estén más cerca. Por otro lado, “From cOre to edGe” enfatiza la idea de extender la capacidad de procesamiento y almacenamiento desde los centros de datos (core) que conforman el Cloud hasta los dispositivos próximos al borde de la red (Edge). Ambas explicaciones se utilizan para ilustrar el concepto de Fog Computing y su objetivo principal

de acercar los recursos. En definitiva, Fog Computing se define como un paradigma de computación distribuida que extiende los recursos tradicionales del Cloud Computing [17].

Similar al Cloud, el Fog Computing proporciona computación, almacenamiento, redes y servicios de aplicaciones ubicuos en una plataforma situada entre los dispositivos finales y los centros de datos. Esto es, mientras que el Cloud Computing se basa en centros de datos remotos, el Fog Computing se despliega en infraestructuras distribuidas más cercanas, como routers, gateways, elementos de acceso o los propios dispositivos IoT [18]. Esta ubicación geográfica cercana reduce la latencia y permite una respuesta más rápida a las necesidades de los dispositivos [19]. Además, el Fog Computing reduce el uso del ancho de banda, al evitar el envío de los datos IoT a través de la red hasta llegar a las instancias Cloud. La virtualización es una tecnología fundamental para el Fog Computing, ya que separa las infraestructuras físicas para crear diversos recursos dedicados que pueden ejecutar múltiples sistemas operativos y múltiples aplicaciones al mismo tiempo y en el mismo recurso. En [20] se propone una definición formal: "Fog Computing es un escenario en el que un gran número de dispositivos heterogéneos (inalámbricos y a veces autónomos), ubicuos y descentralizados se comunican y potencialmente cooperan entre sí y con la red para realizar tareas de almacenamiento y procesamiento sin la intervención de terceros. Estas tareas pueden ser para admitir funciones básicas de red (encaminamiento y conmutación) o nuevos servicios y aplicaciones que se ejecutan en un entorno aislado y restringido. Los usuarios que alquilan parte de sus dispositivos para alojar estos servicios reciben incentivos por hacerlo". El enfoque descentralizado del Fog Computing fomenta la colaboración entre los dispositivos finales y los recursos de red, lo que resulta en una mayor autonomía y capacidad de toma de decisiones a nivel local. Esto tiene ventajas significativas, especialmente en aplicaciones que requieren una baja latencia. Por ello, el Fog Computing está pensado para IoT y redes Machine to Machine (M2M), donde en muchas ocasiones se requiere de un procesamiento y gestión de los datos en tiempo real o, por lo menos, donde el tiempo resulta un parámetro crítico; por ejemplo, en aplicaciones que requieren procesamiento en tiempo real, como la monitorización de sistemas críticos, el análisis de datos en el borde de la red (Edge Analytics) o la automatización de procesos industriales. Sin embargo, también presenta algunas desventajas, como son el consumo de energía de los dispositivos y la escasa capacidad de cómputo, la cual puede ser en ocasiones insuficiente.

2.1.3. Arquitecturas IoT-Fog-Cloud

El crecimiento del número de objetos conectados en escenarios IoT e IIoT desemboca en un problema de escalabilidad que encuentra una primera solución en la integración de IoT con Cloud Computing. Por otra parte, gestionar servicios para aplicaciones IoT e IIoT que requieren una latencia en tiempo real y predecible es un desafío. Son varias las aplicaciones que presentan requisitos estrictos de retardo, debido a la interacción entre los dispositivos IoT y el entorno físico a través de la detección y la actuación. Además, los datos generados por algunos dispositivos IoT pueden dejar de ser válidos en cuestión de segundos. Dada la naturaleza de estas aplicaciones, el estricto requisito de retardo hace que el procesamiento de estos datos en el Cloud sea un cuello de botella en términos de rendimiento.

Las capacidades limitadas de los dispositivos requieren que las aplicaciones se integren en los paradigmas de Cloud y Fog Computing. El primero presenta una capacidad de procesamiento y almacenamiento notablemente superior, a partir de los que se pretende disponer de recursos suficientes para procesar la inmensa cantidad de datos generados por los dispositivos IoT. Estos dispositivos generarán e intercambiarán

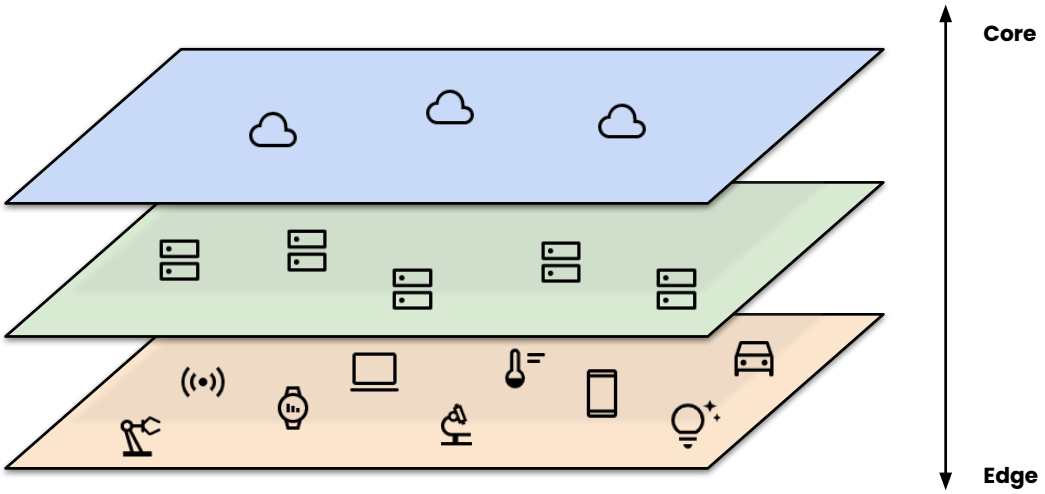


Figura 2.1: Arquitectura IoT-Fog-Cloud.

datos en gran volumen, a alta velocidad y con notable heterogeneidad, lo que se conoce como Big Data. Por su parte, el Fog Computing mejora significativamente la latencia del servicio al acercar los recursos a los dispositivos. Las características tanto de uno como de otro paradigma permiten la integración y la interoperación de una gran cantidad de dispositivos y servicios de IoT en diferentes dominios.

Aunque ambas opciones, Cloud y Fog, se pueden usar de manera independiente, no es necesario elegir únicamente una de ellas, ya que las dos soluciones se complementan entre sí, y la cooperación entre ellas conduce a un uso óptimo de los recursos, mejorando las capacidades del sistema en términos de eficiencia energética, reducción de costes, y procesamiento, agregación y almacenamiento de datos. En este sentido, un orquestador podría manejar esta cooperación entre Cloud y Fog.

Los servicios IoT abarcan una amplia gama de aplicaciones en las que los dispositivos conectados recopilan, transmiten y procesan datos de diferente índole. Cada servicio concreto demanda un tratamiento particular. Un ejemplo de ello se ilustra en la Figura 2.2, donde se pueden ver ejemplos de servicio IoT con diferentes requisitos de latencia. Un ejemplo de servicio con altos requisitos de latencia podría ser un sistema de control de tráfico inteligente. En este escenario, se utilizan sensores IoT distribuidos en diferentes puntos de una ciudad para recopilar información en tiempo real sobre el flujo de vehículos, la congestión y otros datos relacionados con el tráfico. El sistema requiere una baja latencia para poder procesar rápidamente la información recopilada y tomar decisiones en tiempo real, como ajustar los tiempos de los semáforos o redirigir el tráfico. En esta situación, el tratamiento de los datos en los propios dispositivos IoT o en servidores Fog cercanos resulta plausible. Por el contrario, un servicio que no requiere una latencia tan baja es, por ejemplo, un sistema de monitorización de consumo de energía en el hogar. Los medidores IoT ubicados en el hogar se utilizan para recopilar datos sobre el consumo de energía en diferentes dispositivos y zonas de la casa. La recopilación y análisis de datos sobre el consumo de energía puede realizarse a intervalos regulares (por ejemplo cada hora o cada día), y no es necesario que los datos se procesen o se presenten de inmediato. Para este tipo de servicios, el uso de Cloud Computing puede aportar beneficios significativos. El uso conjunto de ambos paradigmas permite aprovechar las fortalezas de cada uno y maximizar los beneficios en diversos servicios IoT, liberando carga de trabajo y tratando de satisfacer los requisitos de cada servicio.

Esta interdependencia Fog-Cloud se destaca, por ejemplo, en [17], donde los autores desarrollaron una

definición de Fog Computing que asume ya en sí misma la colaboración con el Cloud: “Fog Computing es una arquitectura de computación distribuida geográficamente con un grupo de recursos que consiste en uno o más dispositivos heterogéneos conectados de forma ubicua (incluidos dispositivos Edge) ... respaldados de manera integral por servicios de Cloud Computing, para proporcionar de manera colaborativa computación flexible, almacenamiento y comunicación (y muchos otros nuevos servicios y tareas) en entornos aislados (*sandboxed*) a una gran cantidad de clientes en proximidad”. Fog y Cloud crean así una arquitectura cooperativa en la que cada solución complementa lo que le falta a la otra. Los nodos Fog podrían atender tareas que en su mayoría sean de alta prioridad y que necesitan ser procesadas en tiempo real o casi real [21]. Hay tareas menos exigentes en términos de latencia y/o prioridad, como ya se ha comentado, que pueden ejecutarse en el Cloud [22]. Sin embargo, también hay que tener en cuenta que el Fog Computing presenta limitaciones para trabajar con grandes volúmenes de datos [23]. Un ejemplo de estas limitaciones sería el consumo de energía, lo que resultaría en el agotamiento de las baterías [24]. Además, un gran volumen y una elevada tasa de llegada de los datos puede incrementar la ocupación de las colas de espera de las aplicaciones, y aumentar con ello la latencia [25]. La interacción entre Fog y Cloud resulta, por tanto, en un nuevo paradigma prometedor que permite atender a miles de millones de dispositivos y aplicaciones de IoT. Ejemplos de estas aplicaciones incluyen automatización industrial, transporte, transmisión en vivo, juegos en tiempo real y en línea, realidad aumentada, vehículos conectados, monitorización remota de la salud de pacientes, entre otros.

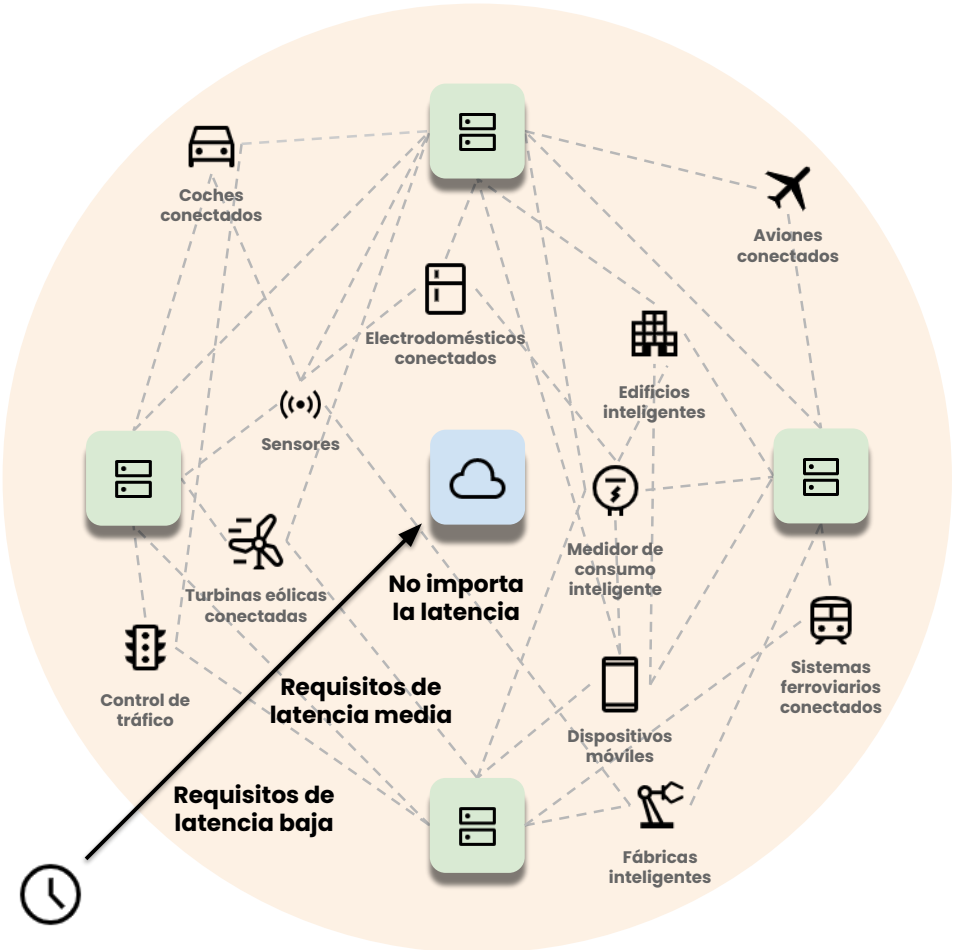


Figura 2.2: Requisitos de latencia de servicios IoT en un escenario Fog-Cloud.

Esta cooperación conduce a arquitecturas de tres niveles (IoT-Fog-Cloud) [26]. La Figura 2.1 muestra una vista conceptual de alto nivel de esta integración de Fog y Cloud. La capa inferior representa la capa IoT, que cuenta con una serie de dispositivos IoT distribuidos geográficamente. Estos dispositivos generan datos y los transfieren a las capas superiores para su procesamiento y almacenamiento. La segunda capa, formada por un conjunto de nodos Fog, brinda servicios a los dispositivos IoT. Finalmente, la capa superior está formada por los centros de datos que constituyen el Cloud. Los dispositivos IoT pueden, por tanto, transferir datos al Fog o al Cloud según sus necesidades.

2.1.4. Algoritmos de offloading

Las arquitecturas IoT-Fog-Cloud permiten mejorar la eficiencia en entornos IoT al disponer de alternativas para el procesado y almacenamiento de los servicios IoT, las cuales se complementan entre sí y facilitan abordar las estrictas necesidades de este tipo de servicios. Sin embargo, el paradigma Fog-Cloud también presenta muchos obstáculos y entre ellos, en particular, se encuentra la falta de acuerdo sobre las mejores prácticas o la existencia de un modelo de referencia que defina cómo se debe utilizar el Cloud y Fog Computing en entornos IoT. A medida que crece la cantidad de dispositivos IoT, la complejidad de las tareas y las restricciones en términos de consumo de energía, coste económico y latencia tienden a aumentar en un entorno de aplicación no controlado y, en general, muy variable. Además, la naturaleza heterogénea de la arquitectura hace que los recursos disponibles también pueden variar. Los problemas de recursos también pueden ser amplificados por los dispositivos móviles y el tráfico de la red. Por último, la ubicación y la transferencia dinámica de tareas han surgido como un problema grave en el modelo IoT-Fog-Cloud. Nace, por tanto, la necesidad de aliviar la carga de trabajo y aprovechar los recursos de manera más eficiente y adaptativa. Para lograr un menor retraso de procesamiento y utilizar mejor los recursos disponibles, Fog y Cloud dependen de un mecanismo de offloading adecuado. El desarrollo de algoritmos que tengan en cuenta diversos parámetros y casuísticas resulta interesante para abordar el problema.

El algoritmo de offloading desplegado en la arquitectura IoT-Fog-Cloud se encarga de tomar decisiones respecto a la transferencia selectiva de tareas y cargas de trabajo desde el Fog hacia los recursos remotos en el Cloud. Esta transferencia tiene como objetivo liberar recursos locales y aprovechar toda la potencia de cómputo y almacenamiento disponible. Esto permite obtener varios beneficios. En primer lugar, se reduce la carga de trabajo en el Fog, lo que permite a los dispositivos operar de manera más eficiente y utilizar recursos limitados de manera óptima, reduciendo el consumo de energía. Además, el offloading permite aprovechar de forma óptima los recursos contratados a los proveedores de servicios Cloud, lo que facilita el procesamiento intensivo de datos y la ejecución de aplicaciones complejas mientras se reduce el coste, en este caso, monetario, debido a la contratación de los servicios Cloud. Otra ventaja radica en que al utilizar los servicios Cloud para tareas exigentes en recursos, se evita la necesidad de adquirir, alimentar y mantener más hardware adicional. En definitiva, al liberar recursos locales y aprovechar la potencia del Cloud, se logra un equilibrio entre capacidad de cómputo, eficiencia energética y costes. Sin embargo, es importante tener en cuenta que no existe una solución única y que el proceso de offloading depende de las políticas y estrategias establecidas. Cada aplicación y escenario puede requerir enfoques específicos para lograr el máximo rendimiento y optimización de recursos.

2.2. Trabajos relacionados

La combinación de IoT e IIoT con Fog y Cloud Computing ha atraído recientemente la atención de la comunidad investigadora desde diferentes ángulos. Con una perspectiva global, algunos trabajos han propuesto diferentes arquitecturas. Por ejemplo, los autores de [27] se centran en una visión general de una solución para la Industria 4.0, basada en IoT-Fog-Cloud, identificando una serie de casos de uso y desafíos emergentes. Del mismo modo, Mouradian et al. presentan en [28] un estudio sobre Fog Computing, estableciendo un conjunto común y conciso de criterios de evaluación que abarca tanto arquitecturas como algoritmos. Los autores de [29] muestran un modelo de arquitectura integrada para combinar Mobile Edge Computing (MEC) y Fog Computing para redes 5G. Proponen orquestar dinámicamente todas las funciones y recursos necesarios en los nodos 5G, sin asumir ninguna configuración predefinida. Aunque estos trabajos comparten el mismo escenario de aplicación que el presentado en este documento, su enfoque se sitúa más a nivel de arquitectura, mientras que el principal interés aquí se centra en la lógica de offloading del procesamiento y las soluciones algorítmicas para obtener comportamientos óptimos.

Otros trabajos existentes se han centrado más específicamente en enfoques de offloading. Sengupta et al. proponen en [30] una solución para asegurar una arquitectura de tres niveles, así como una técnica de offloading para hacer cumplir los requisitos de seguridad. Una solución arquitectónica alternativa se presenta en [31], donde los autores analizan un despliegue jerárquico de Fog, comparándolo con topologías planas, es decir, configuraciones de red en las que todos los dispositivos están conectados directamente entre sí, sin ninguna jerarquía o estructura adicional. Los autores utilizan la Teoría de Colas para analizar el rendimiento del sistema, y consideran diferentes tipos de peticiones, proponiendo soluciones de offloading para hacer frente a cargas elevadas. Los autores de [32] discuten una novedosa arquitectura Software-Defined Networking (SDN)/Network Functions Virtualization (NFV) que abarca el Edge y el Cloud, para optimizar las tareas de computación. SDN también se utiliza en [33] para introducir un algoritmo que selecciona los puntos óptimos de acceso y computación.

Otro grupo interesante de trabajos se centra en el consumo de energía. Por ejemplo, el offloading de computación en un entorno Fog-Cloud se aborda, desde una perspectiva analítica, en [34], donde los autores tratan de optimizar el consumo de energía utilizando el algoritmo de gradiente acelerado, técnica comúnmente utilizada en el campo del Machine Learning (ML) y la optimización. Del mismo modo, el Deep Reinforcement Learning (Deep RL) se explota en [35] para optimizar la energía en arquitecturas Fog, e Iqbal y Buhnova analizan, en [36], las ventajas en términos de eficiencia energética que aporta Fog Computing en escenarios de edificios inteligentes.

Además, existe un número de trabajos que describen la aplicación de varias técnicas a la computación compartida en Fog y Cloud. Por ejemplo, en [37] se utiliza la Teoría de Juegos para implementar la asignación de recursos en una arquitectura IoT-Cloud de tres niveles, mientras que en [38] se adopta un novedoso esquema de subasta. Peralta et al. utilizan la codificación de red en [39] para minimizar el tiempo de offloading de una arquitectura Fog-Cloud, mientras que la optimización meta-heurística se analiza en [40] para mejorar la programación de tareas en estos escenarios. En [41], los autores trabajan con un Alternating Direction Method of Multipliers (ADMM) distribuido para resolver un problema de offloading que considera la Quality of Experience (QoE) y la eficiencia energética como parámetros de rendimiento. En [42] Xiaoting *et al.* proponen una "drift plus computing cost" basado en la optimización de Lyapunov para el offloading de las aplicaciones, y lograr el equilibrio entre el coste de offloading y el rendimiento del sistema en un entorno IoT-Fog. En [43], se propone una estrategia de offloading

computacional offline, utilizando un proceso de decisión de Markov. Los autores utilizaron el ancho de banda disponible como restricción para MEC en Vehicular Network (VN). En [44] se desarrolla otro algoritmo basado en la optimización de Lyapunov. Funciona online, sin requerir información futura, y reduce la latencia de cálculo, manteniendo además un bajo consumo energético. En la misma línea, en [45] se propone un esquema predictivo de offloading y asignación de recursos para sistemas de Fog Computing multinivel.

Tabla 2.1: Parámetros y métricas de rendimiento considerados en algoritmos de la literatura reciente que suponen escenarios similares al algoritmo propuesto.

Algoritmo		Retardo	Energía	Coste	Estabilidad
[41]	Optimización distribuida basada en ADMM.	✓	✓	✗	✗
[42]	Programación estocástica basada en Lyapunov.	✓	✗	✗	✓
[44]	Programación estocástica basada en Lyapunov.	✓	✓	✗	✓
[45]	Offloading predictivo.	✓	✓	✗	✓
Propuesto	Programación estocástica basada en Lyapunov.	✓	✓	✓	✓

A diferencia de estos trabajos, la propuesta descrita en este documento se centra no sólo en el consumo de energía, sino también en el coste monetario, manteniendo la estabilidad de las colas y reduciendo así el retardo. Además, al igual que otras soluciones que utilizan la Teoría de Lyapunov, también tiene en cuenta la evolución temporal del escenario, en el que se producen eventos aleatorios. En la Tabla 2.1 se compara el algoritmo propuesto con enfoques similares de la literatura, al menos en sus objetivos. Se seleccionan aquellos trabajos que asumen entornos aleatorios (no controlados) y que proponen técnicas para proporcionar una adaptación instantánea. La comparación se realiza en términos del algoritmo de decisión y de los parámetros y métricas de rendimiento que considera. Se incluyen los que se enumeran seguidamente:

- **Retardo** Se refiere al retardo que sufren las tareas o servicios de computación, desde que se generan hasta que se procesan completamente.
- **Consumo de energía.** Es debido principalmente al procesamiento. Se suele tener en cuenta para los nodos Fog, que tienen capacidades más limitadas.
- **Coste monetario.** Corresponde al coste de utilizar la capacidad de procesamiento del Cloud. Se considera un modelo de "*pay as you go*", puesto que es lo que ofrecen la mayoría de los proveedores (Microsoft, Amazon, Google, etc.).
- **Estabilidad.** Hace referencia a la estabilidad de las colas de memoria en el sistema global.

Como puede observarse, la solución propuesta es la única que considera conjuntamente la energía (en los nodos Fog) y el coste monetario (en los nodos Cloud), manteniendo la estabilidad del sistema. Por ello, es posible concluir que complementa y amplía el estado del arte disponible relacionado con la distribución de tareas de computación en despliegues de Fog-Cloud.

3 Plataforma de simulación

Existen varias alternativas para implementar y gestionar instancias de Fog y Cloud. La mayoría de las grandes empresas tecnológicas ofrecen servicios en la nube, como AWS, Azure, Google Cloud, Linode, entre otras. Además, hay alternativas que permiten implementar instancias de Fog-Cloud propietarias y autoadministradas, tanto comerciales (por ejemplo, VmWare) como de código abierto (por ejemplo, OpenStack, Apache CloudStack, Proxmox). Sin embargo, estas tecnologías no están diseñadas para probar o evaluar el rendimiento de soluciones de orquestación, sino para gestionar servicios en producción. En este sentido, hay una necesidad clara de desarrollar frameworks que cubran la brecha entre la planificación analítica y la evaluación del rendimiento esperado en entornos controlados. Existen algunos trabajos relacionados en los que se han desarrollado plataformas con una arquitectura de tres niveles [46-48], pero que presentaban limitaciones para los propósitos aquí buscados. Por lo tanto, se opta por desarrollar un entorno propio, basado en [49], y diseñado específicamente para el análisis de offloading de cómputo en entornos de Fog-Cloud.

3.1. Descripción general

La plataforma desarrollada se ilustra en la Figura 3.1. Incluye tres tipos de elementos que simulan Fog, Cloud y un nodo maestro (Master). Los tres roles se muestran bien diferenciados en la figura. Los nodos Fog generan flujos de tráfico sintético independientes, pertenecientes a diferentes aplicaciones, utilizando distribuciones aleatorias configurables (por ejemplo Poisson, uniforme, lognormal, etc.). El tráfico generado de cada aplicación se almacena en un buffer de entrada, representado en la figura como paso 1. A partir del tráfico generado, los nodos Fog definen servicios (tareas de procesamiento) que agrupan un número aleatorio de paquetes. La generación de servicios, representada en la figura como paso 2, también es configurable, utilizando distintas distribuciones aleatorias. Cuando se definen los servicios, el nodo Fog consulta al nodo Master si es el momento de procesar los datos y a qué puntos de procesamiento se deben enviar para ello. El nodo Master lo decide según el algoritmo implementado en ese instante (paso 3). A continuación, el nodo Fog envía los datos del servicio ya sea a procesadores locales o a instancias remotas en la nube (paso 4). Finalmente, se procesan los servicios en los puntos de procesamiento (pasos 5 y 6). Durante todo el proceso cada componente de la plataforma genera *logs* (trazas) que permiten realizar un seguimiento de cada servicio que pasa por el sistema, así como conocer la evolución temporal y el estado de todos los dispositivos implicados. Toda la lógica anteriormente descrita se ha implementado empleando el lenguaje de programación Python, con un diseño que permite cumplir los aspectos que aparecen a continuación.

- **Modularidad.** Todas las tareas que constituyen la funcionalidad de la plataforma se conciben como

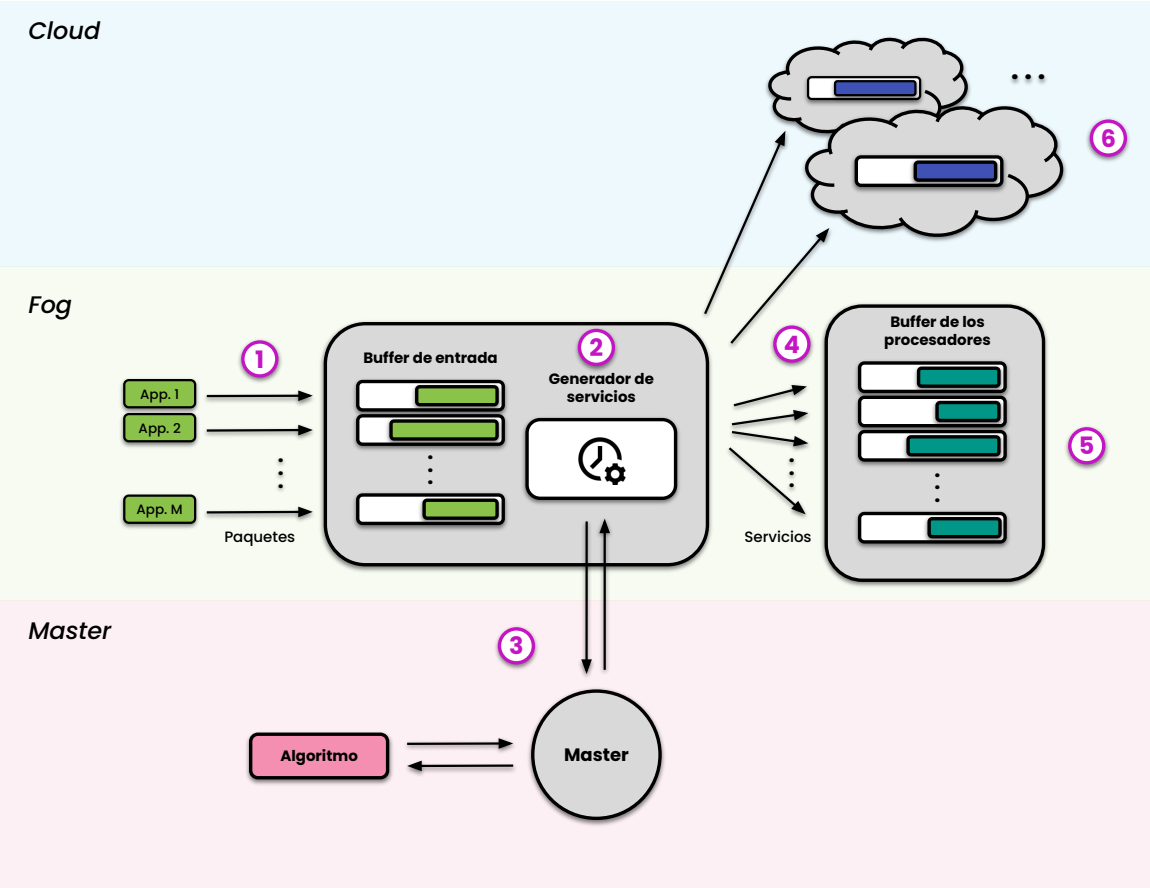


Figura 3.1: Vista general de la plataforma de Fog-Cloud.

módulos, algunos de ellos fácilmente intercambiables, de tal forma que se pueda personalizar de forma sencilla, adaptándose a las necesidades de cada escenario a analizar. En este sentido, el generador de tráfico, el generador de servicios y los diferentes procesadores se ejecutan de forma independiente y son fácilmente intercambiables.

- **Configuración sencilla y rápida.** Para facilitar la configuración previa a cada simulación se generan ficheros *json* que contienen todos los parámetros necesarios. De esta forma, se puede definir cada escenario de manera sencilla, generando un fichero para cada escenario específico, y permitiendo, en definitiva, simplificar el proceso de configuración y cambiar rápidamente entre escenarios.

- **Multiprocesador.** Para una mejor representación de la realidad, los nodos Fog admiten una configuración multiprocesador, donde se establece el número de procesadores de cada nodo, así como sus características, en el fichero de configuración.

- **Variables compartidas entre funciones.** Se establece un módulo Python para una mejor gestión de las variables compartidas. Esto resulta de especial interés en el caso de la gestión de las colas y la lectura de los parámetros de configuración.

- **Procesado de servicios en paralelo.** Mediante la fragmentación de los servicios, se puede procesar un mismo servicio en varios puntos de procesamiento o en diferentes instantes de tiempo.

- **Aplicaciones IoT.** Se implementa la posibilidad de definir varias aplicaciones que generen el tráfico. Así cada aplicación puede tener unas características propias, por ejemplo en la distribución de los datos o en la complejidad que conlleva su posterior procesado.

- **Algoritmo de offloading intercambiable.** El algoritmo de offloading alojado en el nodo Master se implementa como un *plugin*, fácilmente intercambiable, permitiendo así la comparación de diferentes soluciones de decisión bajo las mismas circunstancias.

En la Figura 3.2 aparecen las diferentes funcionalidades que componen los roles Fog, Master y Cloud en colores verde, rosa y azul, respectivamente. Cada funcionalidad se implementa como un hilo de ejecución (threads) independiente. En el caso de las funcionalidades *trafficGen* (generador de tráfico que emula aplicaciones) y *processor* (procesador de Fog o Cloud), se despliegan varios threads, uno por cada generador o procesador configurado. Además, cabe mencionar que tanto *trafficGen* como el algoritmo ubicado en el nodo Master se implementan como soluciones independientes en diferentes archivos Python. Estos archivos se intercambian según los objetivos específicos de cada simulación, como se presenta en los test-bench del Capítulo 5. Esta modularidad permite adaptar y seleccionar la solución más adecuada para cada caso de estudio, y simplifica la realización de las pruebas y análisis en el contexto de las simulaciones. En la figura se incluyen, además, las principales variables compartidas que relacionan las diferentes funciones: las colas de las aplicaciones (*buffer*), de los procesadores (*q_proc* y *q*), otras colas intermedias (*q_serv* y *buffer*), los paquetes enviados al Cloud (*pkt*), la información de las instancias Cloud disponibles (*info*), y los diccionarios intercambiados en las comunicaciones Fog-Master (*dic_request* y *dic_response*) e internas del Master (*dic_services* y *dic_assign*). También se indican los puntos donde se recogen *logs* de las ejecuciones, tales como la situación de las colas, tiempos de generación de servicios, decisiones de los algoritmos implementados o información relativa al procesado.

Finalmente, en relación al despliegue de la plataforma, este puede llevarse a cabo en contenedores mediante la herramienta Docker o en local. Las ventajas de emplear contenedores sobre el despliegue en local, especialmente a medida que aumenta la complejidad o el número de nodos, se comentan posteriormente, en la Sección 3.5. A continuación se describen en mayor detalle los tres tipos de roles o nodos que contempla la plataforma.

3.2. Nodo Fog

La lógica implementada para un nodo de la capa Fog pretende imitar el comportamiento de un nodo real, y comprende para ello un conjunto de funcionalidades concurrentes que utilizan multi-threading. Se pueden resumir en tres: generador de tráfico, generador de servicios y procesador. En la Figura 3.4 se muestra un diagrama de flujo que recoge, de forma resumida, la lógica que sigue el nodo Fog. Puesto que las funciones del nodo Fog se lanzan mediante diferentes threads, todas pueden trabajar y ejecutar su lógica en paralelo, de forma independiente, sin interferir la una con la otra.

La primera funcionalidad se ocupa de la emulación de la creación de servicios y la generación de los datos pertenecientes a dichos servicios. En este sentido, en lugar de recibir tráfico de dispositivos IoT reales, el programa emula dicha generación de tráfico (paquetes). Se implementa en el nodo Fog un thread por aplicación, y se asigna un identificador a cada una. Asimismo, existe un buffer de entrada para cada aplicación. La recepción de paquetes sigue una distribución y una longitud de los mismos, configurables

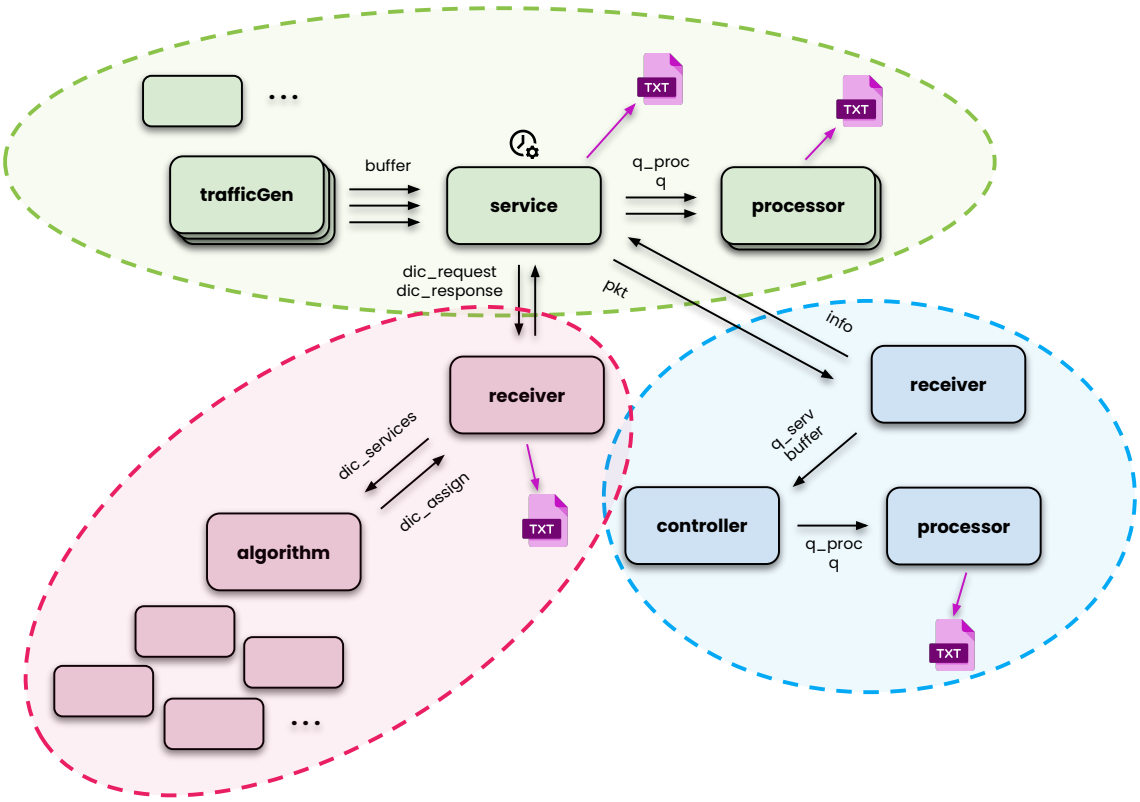


Figura 3.2: Implementación de la plataforma de Fog-Cloud. Se muestra el conjunto de funcionalidades que conforman Fog (verde), Cloud (azul) y Master (rosa).

a través de un fichero. En la Figura 3.4, en el punto 1, se muestran dos posibles implementaciones. El diagrama de la izquierda muestra el esquema seguido para generar tráfico de Poisson. Como es bien sabido, en un proceso de Poisson los tiempos entre llegadas siguen una distribución exponencial negativa. Así, estos se consideran variables aleatorias independientes e idénticamente distribuidas (i.i.d.), por lo que el thread pausa su ejecución utilizando una variable aleatoria exponencial y, al despertar, genera un paquete que almacena en la cola de la aplicación correspondiente. A continuación, el programa comprueba que la simulación continúa, es decir, que todavía no se ha alcanzado el tiempo de simulación, situación que sería notificada por el generador de servicios, mediante la variable compartida *fin_event*. La generación de paquetes de forma uniforme (por ejemplo un paquete cada $100\,ms$) también se puede implementar siguiendo esta forma de actuar. Por otra parte, el diagrama de la derecha ilustra la implementación de otras distribuciones de tráfico. En este caso se hace uso de dos eventos incluidos en el generador de servicios, los cuales permiten la correcta sincronización entre los threads. Cuando el generador de servicios lo requiera, el generador de tráfico despierta y genera un número aleatorio de paquetes (por ejemplo según una variable lognormal), introduciéndolos en la cola correspondiente. Esta operación se repite a lo largo de toda la simulación.

Los paquetes generados se almacenan en la cola de aplicación que corresponda y, posteriormente, en lugar de tratar cada paquete individualmente, se implementa un modelo de servicio. Así, se define un servicio como un conjunto de paquetes relacionados entre sí. La generación de servicios se establece mediante un fichero de configuración, al igual que en el caso de la generación de paquetes. Se aportan más detalles de dicho fichero en la Sección 3.6. A continuación, en función de la decisión tomada por el nodo Master, los servicios se transfieren del buffer de llegada a la cola de procesamiento del nodo Fog, en

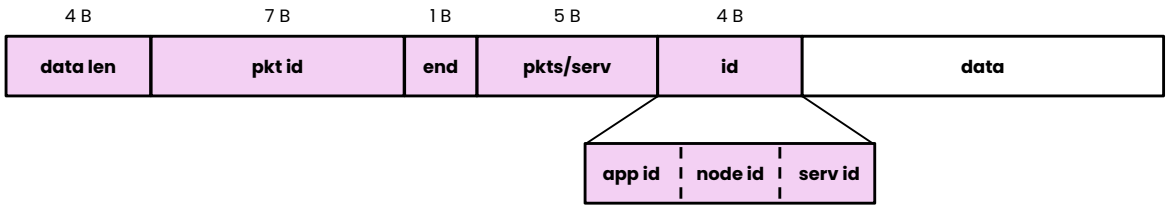


Figura 3.3: Estructura de los paquetes.

el caso de procesamiento local, o al nodo Cloud, para ser procesados remotamente.

Antes de almacenar los paquetes generados por las aplicaciones en el buffer de entrada y previo a su envío al punto de procesamiento, se añaden una serie de cabeceras que permiten rastrear y procesar adecuadamente todos los paquetes (ya sea de forma local o remota). En la Figura 3.3 se muestra el encapsulado de los datos. Como se puede observar, a los datos generados por las aplicaciones se les añade una serie de cabeceras, las cuales se desglosan a continuación.

- ***data len*.** Indica el tamaño de los datos de aplicación.
- ***pkt id*.** Asigna un identificador al paquete. Es único en cada aplicación.
- ***end.*** Bit empleado para indicar el fin de la simulación y. por tanto, el cese de la actividad de los nodos.
- ***pkts/serv*.** Indica el número de paquetes que componen el servicio. Solo se indica en el primer paquete.
- ***id*.** Identificador que permite diferenciar los servicios.
  - ***app id*.** Indica la aplicación a la que pertenece el servicio.
  - ***node id*.** Nodo al que pertenece el servicio. Útil en el caso de despliegues con varios nodos Fog en la red.
  - ***serv id*.** Identificador único del servicio. Comienza en 0 y se incrementa con la generación de cada nuevo servicio.

Los valores de las cabeceras *pkts/serv* y *serv id* se establecen durante la generación de los servicios.

El modelo de generación de servicios se implementa en un thread independiente dentro del nodo Fog, encargándose de varias tareas. En el punto 2 de la Figura 3.4 se ilustra parte de su lógica. En primer lugar, se agrupan los paquetes generados por una aplicación en un servicio, asignándole un identificador. Esta funcionalidad permite a la plataforma emular un procesado a nivel de servicio, consiguiendo un comportamiento más realista. A continuación, cuando corresponda, se solicita al nodo Master que indique qué hacer con los servicios, es decir, si procesarlos en ese momento o mantenerlos en cola. El thread se encarga también del envío de los paquetes a través de la red al nodo Cloud correspondiente cuando sea necesario, así como de transferirlos desde el buffer hacia la cola de procesado (en el propio nodo Fog), cuando el Master opta por la opción de procesado local. En el generador de servicios se implementa además la capacidad de solicitar al Cloud información relativa a su configuración y su estado actual.

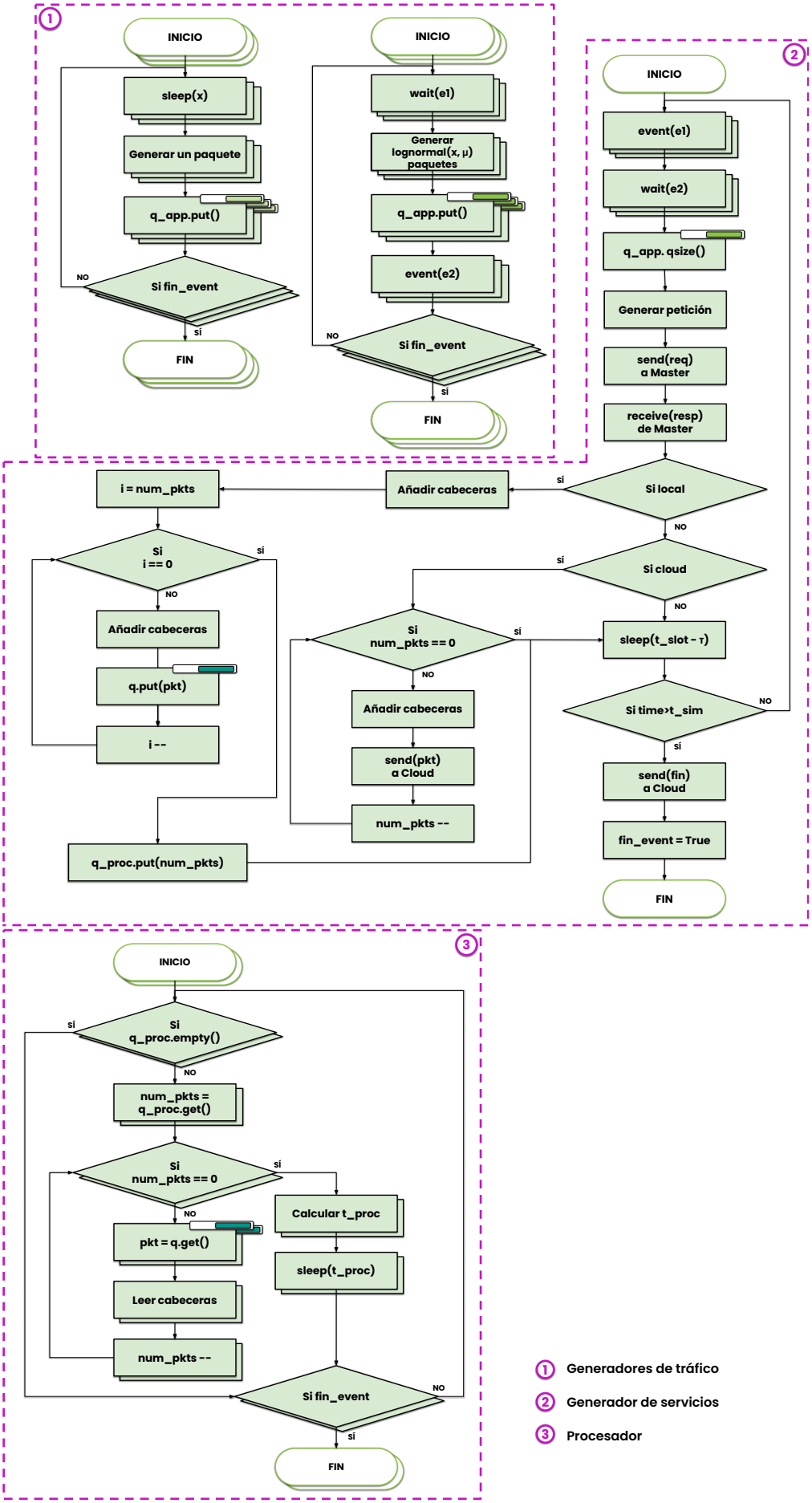


Figura 3.4: Diagramas de flujo del nodo Fog.

Finalmente, la última funcionalidad de un nodo Fog se encarga de procesar localmente el tráfico. El procesado se implementa como otro thread independiente que sigue los pasos mostrados en el punto 3 de la Figura 3.4. Se crea un thread para cada procesador cuando se opta por una configuración multi-procesador, y se asigna un identificador a cada uno de los procesadores desplegados. Un procesador dispone de dos colas FIFO. La primera se puede entender como una caché del procesador, y en ella se almacena información relativa al número de paquetes que forma el servicio (o servicio parcial si existe fragmentación) que se debe procesar. La segunda es la propia cola de procesado, donde se guardan los paquetes. El procesador saca los paquetes que correspondan de esta cola y, tras estimar el tiempo de procesado en función de su tamaño, ejecuta un *sleep* durante ese tiempo. No se realiza ninguna operación de procesado real con los datos, que son aleatorios.

3.3. Nodo Cloud

Dentro de estos nodos se ejecuta un único script Python con una funcionalidad sencilla: procesar los servicios entrantes. La Figura 3.5 muestra un diagrama de flujo que ilustra el comportamiento principal de las tres tareas que ejecuta un nodo Cloud. De izquierda a derecha en la figura se muestran: un receptor del tráfico entrante, un "controlador" encargado de la gestión de este tráfico, y un procesador.

Cada nodo Cloud tiene una función de recepción que almacena en una cola todos los paquetes de los servicios asignados a ese nodo. Este receptor se implementa como un thread independiente que se ejecuta continuamente durante toda la simulación. Este thread permanece a la espera de recibir paquetes que procesar. Los paquetes entrantes se almacenan en una cola. A su llegada se comprueba la cabecera *pkts/serv* del primer paquete de servicio, para conocer los que restan de ese servicio. Esta información se almacena en otra cola compartida. También se comprueba la cabecera *fin*, puesto que cuando su valor es 1 indicaría que el tiempo de simulación ha concluido, comenzando el cierre de todos los threads que componen la implementación del nodo Cloud. Finalmente, una última funcionalidad sería el envío de información del Cloud (por ejemplo estado de las colas o la capacidad de cómputo) al Fog, cuando este lo solicite.

La siguiente funcionalidad se encarga de gestionar el tráfico que se almacena en el buffer de llegada, usando la información del tamaño de los servicios. El objetivo principal es la transferencia de los paquetes correspondientes al procesador. Una vez se completa la transferencia, el thread comprueba que la simulación continúa, y espera hasta que haya más paquetes pendientes.

Por último, siguiendo la misma lógica que los nodos Fog, se implementa otro thread que desempeña el papel de procesador. El procesador de un nodo Cloud implementa un par de colas (caché y buffer del procesador), y emula los tiempos de procesado de los paquetes.

3.4. Nodo Master

Además de los nodos Fog y Cloud, cualquier despliegue de la plataforma desarrollada cuenta con un nodo adicional llamado Master. Este nodo aplica las políticas de offloading preconfiguradas, utilizando la información de estado recopilada de los nodos Fog y Cloud, así como otros parámetros del sistema. El algoritmo alojado en el nodo Master establece la política concreta de offloading a evaluar. Este algoritmo se implementa como un plugin, lo que permite la comparación de diferentes soluciones en los mismos

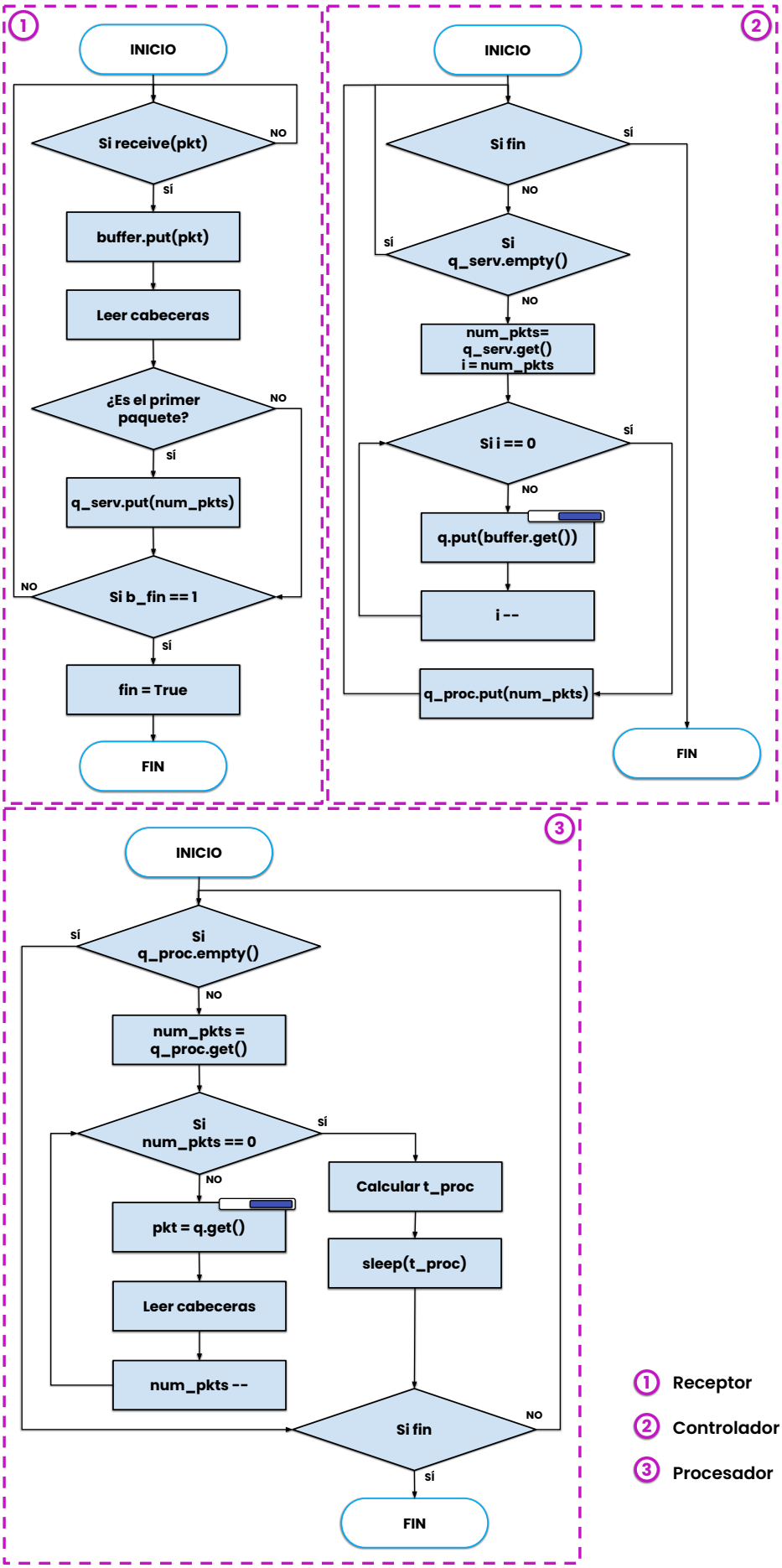


Figura 3.5: Diagramas de flujo del nodo Cloud.

escenarios. Cabe destacar que la plataforma implementa una interfaz genérica para comunicarse con el nodo Master, que es independiente del algoritmo de decisión adoptado.

La plataforma implementa un conjunto mínimo de paquetes de control que se intercambian entre nodo Master y Fog. Por un lado, cuando es necesario decidir dónde ejecutar un servicio, se envía al nodo Master la información de interés. Esta información intercambiada contempla aquella relativa a la configuración estática de los nodos Fog y Cloud, así como a los propios servicios. El Fog se encarga de recopilar toda la información, entre la que se encuentra, por ejemplo, el número de procesadores, la tasa de procesamiento, el tamaño de los servicios, etc. Además, se incluye la información de estado (dinámica) de los nodos, la cual se corresponde, básicamente, a la ocupación de las colas de las aplicaciones y de los procesadores. Toda esta información puede ser o no utilizada por el algoritmo implementado en ese momento en función de las prioridades y, en definitiva, del funcionamiento que establece la estrategia de orquestación. Así, no usa el mismo tipo de información un algoritmo basado en costes de energía que uno basado en retardo. El intercambio de información resulta fácilmente personalizable, pudiendo incluirse o eliminarse información en función de las necesidades. Además, se contempla la posibilidad de fragmentar los servicios, es decir, de "repartir" el procesado de un servicio en el tiempo, de tal forma que el nodo Fog tiene la capacidad de darse cuenta de que la respuesta del Master no asigna la totalidad de los paquetes que componen un servicio. En este supuesto, el Fog es capaz de mantener en el buffer los paquetes restantes del servicio, y volver a solicitar al Master su procesado en una próxima ocasión. Esto permite también procesar un servicio en varios nodos o puntos de procesamiento.

En el Anexo A se incluye un ejemplo de petición y respuesta. La información se estructura en un diccionario de Python. En el caso de las peticiones, en primer lugar, se encuentra la información vista anteriormente y, a continuación, la correspondiente a los servicios. En ambos casos, petición y respuesta, los servicios se organizan en un diccionario con dos niveles. El primero se corresponde con el identificador de la aplicación, y el segundo con el identificador del servicio. El valor presenta un formato en el que se indica, separado por *::*, el lugar donde procesar (local o Cloud), un número que establece el procesador en el caso de que se vaya a ejecutar localmente, o el identificador del nodo en el caso de moverlo al Cloud, y, finalmente, un número que indica el número de paquetes a procesar (si se omite, se asume que se procesan todos los paquetes del servicio). Si el servicio se procesa en varios sitios, las asignaciones se separan por comas. Se puede incluir fácilmente información extra sobre los servicios para, por ejemplo, establecer diferentes niveles de prioridad o evitar la fragmentación.

La Figura 3.6 incluye un diagrama de flujo que refleja el comportamiento del nodo. Al inicio de la simulación se crea, para cada nodo Fog, una conexión y un thread. A continuación, el Master se mantiene a la espera de peticiones procedentes del Fog. Llegado el momento, recibe la petición y, tras procesarla, hace llegar la información al algoritmo implementado en el nodo. El algoritmo se describe en una clase presente en un fichero independiente, cuya ruta se especifica en el fichero de configuración. Utilizando la información proporcionada por el resto de nodos, y las decisiones tomadas por el algoritmo implementado, el nodo Master responde a la petición. La respuesta dicta al nodo Fog cómo proceder, es decir, indica qué servicios (o fragmentos del servicio) procesar localmente en el mismo Fog, cuáles remotamente en un Cloud o cuáles mantener en la cola de la aplicación. En este sentido, el despliegue de diferentes políticas de procesamiento se reduciría a la selección de un algoritmo u otro, lo que da lugar a una configuración muy flexible del experimento.

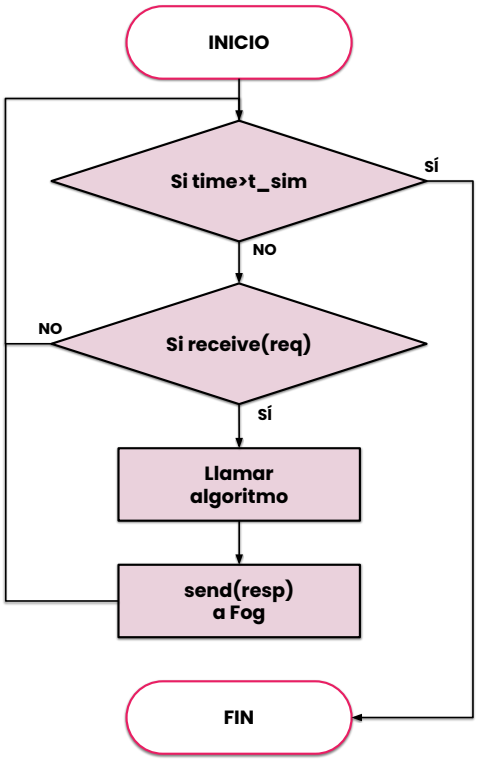


Figura 3.6: Diagrama de flujo del nodo Master.

3.5. Dockerización

Para garantizar una plataforma escalable y ligera, en la que puedan desplegarse múltiples nodos sin sobrecargar la máquina anfitriona, todos los nodos se han desplegado en contenedores utilizando Docker[1]. Esta herramienta permite a los usuarios desplegar rápidamente múltiples contenedores personalizados en el mismo host. Cada contenedor es una unidad de software aislada que empaqueta código y sus dependencias, lo que le permite ejecutarse, independientemente del host subyacente. Docker ejecuta imágenes de contenedores que son paquetes de software ligeros, e independientes, y que incluyen todo lo necesario para ejecutar una aplicación (i.e. código, librerías del sistema, configuraciones...). Las imágenes pueden configurarse mediante un *Dockerfile* para ejecutar aplicaciones personalizadas en contenedores aislados. Docker se ejecuta sobre el sistema operativo anfitrión y actúa como una capa de abstracción para las aplicaciones. En este caso, los tres tipos de nodos disponibles en la plataforma se crean a partir de imágenes personalizadas basadas en Ubuntu 22.04. Los tres Dockerfiles creados se recogen en el Anexo A.

En el caso de la plataforma de evaluación aquí descrita, cada uno de los contenedores funciona como un nodo Fog, Cloud o Master, que se construyen a partir de diferentes imágenes Docker personalizadas. Estas se adjuntan en el Anexo A. En todos los casos, las imágenes utilizan Ubuntu 22.04 como sistema operativo base. La decisión de implementar las funcionalidades de cada uno de los tres tipos de nodos en contenedores independientes, en lugar de llevar a cabo el despliegue de varios nodos o incluso la plataforma al completo en un mismo contenedor, aporta varias ventajas. En primer lugar, la flexibilidad, puesto que es posible realizar distintos despliegues al configurar el número de nodos Fog, Cloud y Master

[1] `https://www.docker.com`

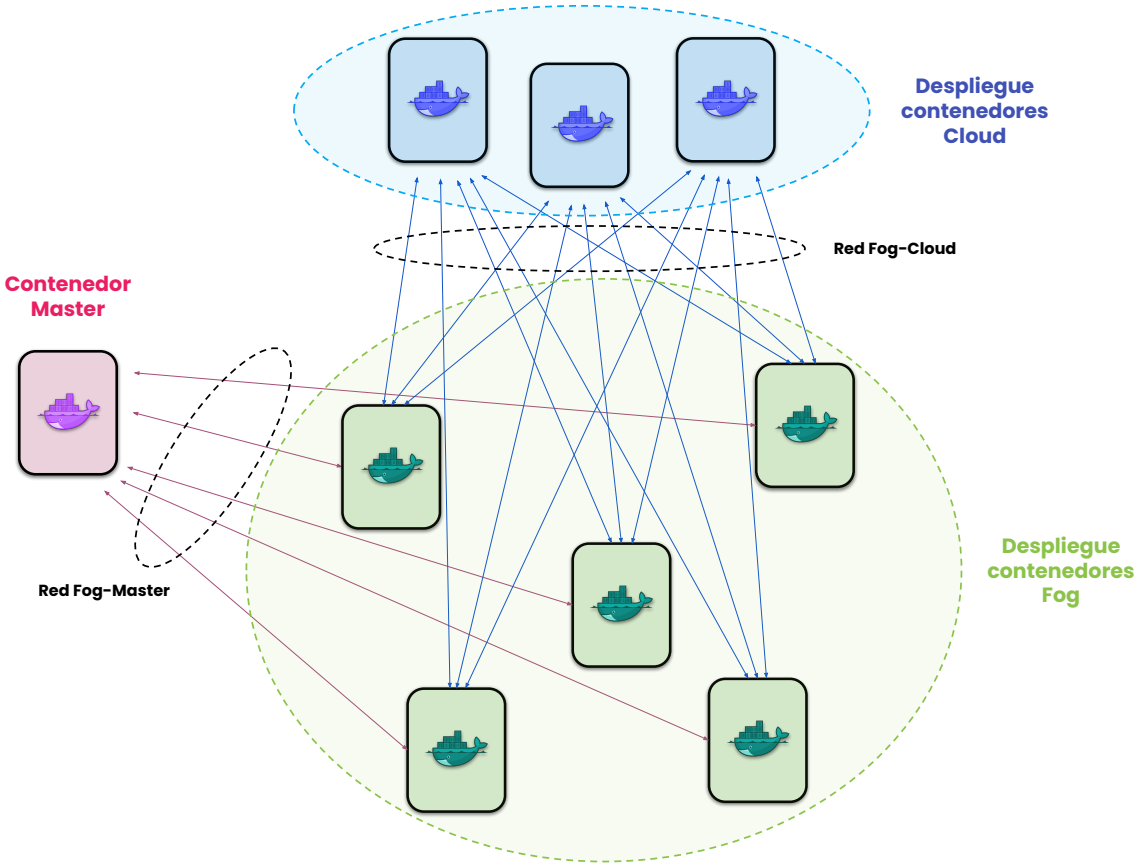


Figura 3.7: Despliegue de los contenedores Docker.

de forma sencilla. En este sentido, se emplea la herramienta Docker *Compose*[2] para automatizar la creación y el despliegue de los nodos. En el Anexo A se muestra un ejemplo de archivo *YAML*, que especifica los detalles de los contenedores, volúmenes, redes y otras configuraciones necesarias para ejecutar la aplicación en Docker.

En la Figura 3.7 se representa el despliegue de los diferentes contenedores con sus respectivas relaciones. La figura muestra también la existencia de dos redes, las cuales permiten el intercambio de mensajes entre el contenedor Master y los contenedores Fog, y el envío de información de cada contenedor Cloud a los contenedores Fog y de los servicios desde el Fog hacia el Cloud, respectivamente. Las redes en Docker son una potente herramienta a través de la que es posible definir y controlar la comunicación entre contenedores. Docker ofrece varios tipos de redes que se adaptan a diferentes necesidades y objetivos, y además brinda la flexibilidad de interconectarlas utilizando los denominados controladores de red (*bridge*, *host* y *none*, entre otros). Una de las ventajas fundamentales de las redes en Docker es su capacidad para crear entornos aislados y seguros, donde los contenedores pueden comunicarse entre sí sin interferencias externas. Por defecto se emplea el controlador de red "*bridge*", que crea una red virtual interna y asigna una interfaz de red a cada contenedor conectado a ella. De esta manera, los contenedores pueden intercambiar datos a través de la red bridge de Docker, al tiempo que se mantienen aislados del host y de otros contenedores que no estén en la misma red. Sin embargo, Docker también permite la creación de redes personalizadas, opción por la que se ha optado en la plataforma Fog-Cloud. Las redes personalizadas brindan una mayor flexibilidad y control sobre la configuración de red, permitiendo

[2] `https://docs.docker.com/compose/`

crear sus propias reglas y configuraciones específicas, lo que facilita la creación de entornos de red más complejos y adaptados a las necesidades de cada aplicación.

Se configura, por tanto, una red personalizada que permite la conexión entre Fog y Master, y otra que haga lo propio con Fog y Cloud. La creación de estas redes abre la posibilidad de usar Traffic Control (tc)[3] en cada una de las redes definidas: Fog-Master y Fog-Cloud. Esta herramienta de gestión de tráfico, propia de Linux, permite configurar y ajustar diversos parámetros relacionados con el ancho de banda, la latencia y otras características de la red. La funcionalidad principal de tc se basa en la utilización de *qdiscs* (disciplinas de colas) y clases de tráfico. Los *qdiscs* se encargan de manejar la distribución de los paquetes en las colas de salida y de aplicar políticas específicas a cada una de ellas. Las clases, por otro lado, permiten clasificar los paquetes en diferentes categorías según criterios como, por ejemplo, direcciones IP. En concreto las características que resultan de mayor interés en el contexto de este trabajo son, básicamente, las que se describen a continuación.

- **Limitar el ancho de banda.** Permite establecer límites de ancho de banda tanto para la entrada como para la salida de paquetes en una interfaz de red. Esto es útil para evitar la congestión de la red y garantizar una distribución equitativa del ancho de banda entre diferentes aplicaciones o usuarios.

- **Controlar la latencia.** Se puede introducir cierta latencia en la red, para probar el comportamiento de aplicaciones bajo diferentes condiciones de retardo.

3.6. Configuración

Uno de los principales objetivos fijados a la hora de desarrollar la plataforma fue hacerla altamente configurable. La idea es poder realizar diversas pruebas, cambiando entre diferentes escenarios de una forma sencilla y rápida. Para ello, se emplea un fichero *json* de configuración. En el Anexo A se muestra un ejemplo. A continuación, se pretende dar una idea de cómo se lleva a cabo esta configuración previa a la simulación, y qué opciones permite la plataforma en este sentido.

Los parámetros que se pueden configurar en cada uno de los nodos desplegados se recogen en la Tabla 3.1. Se observa que la configuración incluye rutas a ficheros que implementan determinadas funcionalidades de la plataforma, lo que permite una alto grado de personalización. Es posible cambiar el generador de tráfico según las necesidades específicas de cada simulación, con tan solo generar un script de Python que respete una cierta estructura: usar las colas de aplicaciones ya definidas y sincronizarse adecuadamente con el generador de servicios si fuera necesario. El otro fichero intercambiable es el algoritmo de offloading. Cualquier solución implementada debe mantener el formato petición-respuesta establecido.

Además, en el fichero *json* se configuran varios parámetros de la simulación, el número de nodos a desplegar y, para cada uno de ellos, sus características. Todos estos parámetros son accesibles por los programas Python mediante un módulo encargado explícitamente de la lectura del fichero *json* de configuración. Además, se encarga de almacenar las variables compartidas, tales como las colas de las aplicaciones y de los procesadores, así como los eventos que permiten la sincronización entre el modelo

[3] `https://tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.qdisc.html`

de generación de paquetes y el de generación de servicios, y aquellos relativos al proceso de finalización de la simulación.

Se incluye en la plataforma un sistema de *log* que permite registrar eventos y mensajes. Esto ayuda a entender el flujo de trabajo y depurar el código. En definitiva, promueve la programación modular y escalable al generar *logs* (trazas) para cada componente de la plataforma, haciéndola más sencilla de mantener y depurar a medida que se implementan nuevas funcionalidades o se extienden las ya existentes. Para implementar este sistema, se usa la librería logging de Python y se personaliza cada mensaje para diferenciar de qué componente de la plataforma proviene: generadores de tráfico, procesadores, generador de servicios, Cloud o Master. Los niveles de prioridad (*Debug*, *Info*, *Warning*, *Error* y *Critical*) se definen al inicio de la simulación a través del fichero de configuración para cada uno de estos componentes.

Tabla 3.1: Parámetros configurables en la plataforma.

Parámetro	Descripción
$sim\_time/slot\_number$	Tiempo de simulación mediante el número de slots o de segundos totales.
$traf\_gen$	Ruta al fichero que aloja el generador de tráfico.
$traf\_dist$	Distribución de tráfico en función de las opciones disponibles en el generador seleccionado.
$serv\_dist$	Distribución de generación de servicios. Indica si uno por slot o siguiendo una distribución aleatoria (por ejemplo Poisson).
$slot\_time$	Duración de un slot en segundos.
pkt_len_dist	Distribución de la longitud de los paquetes. Puede ser fija o seguir una distribución exponencial.
$traf\_rate$	Media de tráfico en $pkts/s$ para cada una de las aplicaciones.
pkt_len	Media en $bytes$ de la longitud de los datos generados por las aplicaciones.
$HEADERLENSIZE$	Longitud en $bytes$ de la cabecera que contiene la longitud de los datos.
$IDSIZE$	Longitud en $bytes$ de la cabecera con el identificador del paquete.
$PKTSERLEN$	Longitud en $bytes$ de la cabecera que indica el número de paquetes que componen el servicio.
$LENSEV$	Longitud en $bytes$ de la cabecera con el identificador del servicio.
$serv\_rate$	Media de servicios en $serv/s$.
num_app	Número de aplicaciones IoT.
v	Valor del parámetro de configuración del algoritmo propuesto.
eth	Umbral de energía que limita el consumo.
$host$	Dirección IP del nodo.
$port$	Puerto del nodo.
$algorithm$	Algoritmo alojado en el nodo Master.
fog_nodes	Número de nodos Fog.
num_proc	Número de procesadores de un nodo Fog.
$capacity$	Capacidad de un procesador medida en $pkts/s$.
cl_proc	Capacidad de una instancia Cloud medida en $pkts/s$.
$algorithm$	Ruta al fichero que aloja el algoritmo de offloading.
$logger$	Niveles de prioridad de logging.

4 Desarrollo e implementación de algoritmos de offloading

En la presente sección se plantean algoritmos de offloading para arquitecturas Fog-Cloud para ser integrados en la plataforma de simulación. En primer lugar, se busca implementar una primera solución sencilla, que permita llevar a cabo las primeras pruebas en la plataforma, validando su correcto funcionamiento. A continuación, se emplea la Teoría de Lyapunov para introducir una política que equilibre el consumo de energía en los nodos de Fog y el coste monetario de usar el Cloud.

4.1. Solución de referencia

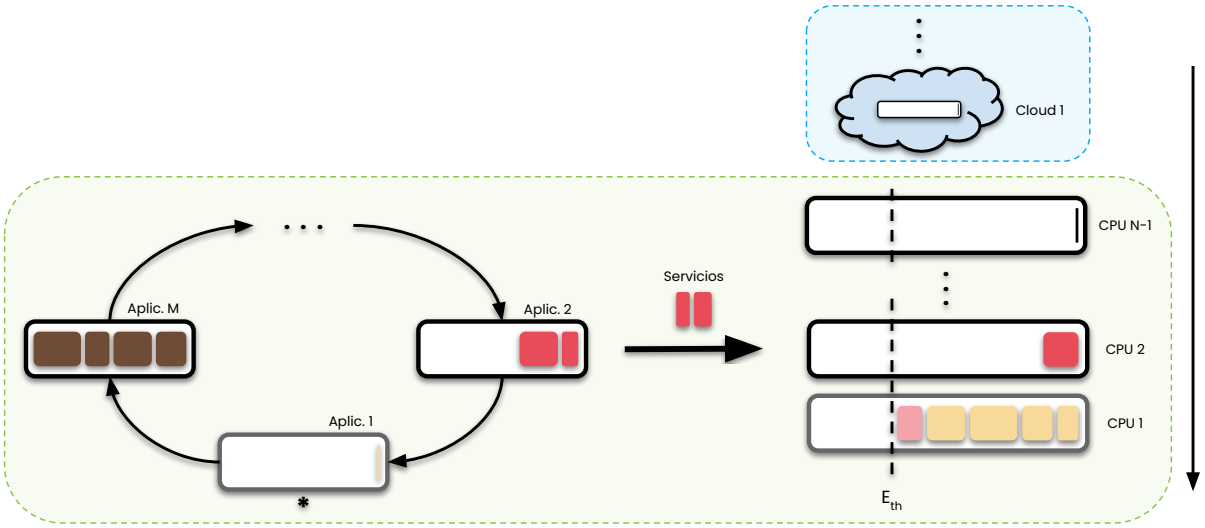


Figura 4.1: Esquema funcional del algoritmo Round Robin.

El objetivo es disponer de una solución que permita probar el correcto funcionamiento de la plataforma y llevar a cabo las primeras simulaciones. Además, se busca que sea un algoritmo sencillo y fácil de implementar, que se pueda usar a modo de referencia para comparar los resultados obtenidos con otras soluciones. En este sentido, se opta por desarrollar un algoritmo basado en Round Robin.

El algoritmo Round Robin es ampliamente conocido y utilizado en diversos campos de la informática y las redes. Su popularidad se debe a su simplicidad y eficacia para distribuir equitativamente la carga de trabajo entre los recursos disponibles. Su funcionamiento se basa en asignar paquetes en porciones iguales y en orden circular, tratándolos todos sin ninguna prioridad. Una vez agotada toda la capacidad de un procesador que puede utilizarse en un intervalo temporal, se pasaría al siguiente procesador. Por

tanto, no tiene en cuenta el consumo de energía ni el coste monetario. Además, se implementa de forma que cambia la aplicación inicial en cada slot, para evitar dar prioridad a una aplicación sobre otra. Los paquetes que no pueden procesarse en la niebla por falta de capacidad se envían a la nube.

Aunque el algoritmo Round Robin no considera aspectos más complejos de optimización, como retardos, consumos de energía o costes, su simplicidad lo convierte en una opción adecuada para comparar y evaluar el rendimiento de otros algoritmos más sofisticados. En definitiva, el desarrollo de un algoritmo basado en Round Robin ofrece una solución inicial y de referencia para la distribución equitativa de cargas de trabajo. Esto permitirá evaluar el rendimiento y sentar las bases para futuras optimizaciones y comparaciones con otros algoritmos más complejos.

4.2. Propuesta de algoritmo

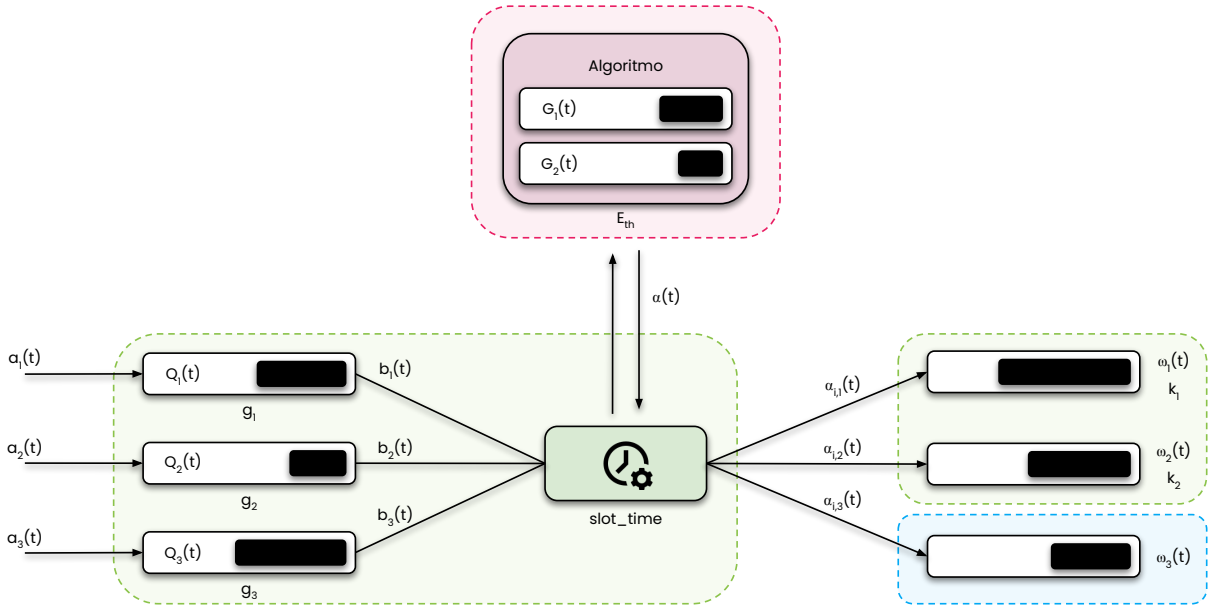


Figura 4.2: Sistema Fog-Cloud con las colas de aplicaciones (Q_m), las colas virtuales de energía (G_n) y las colas de los procesadores.

Esta sección describe el modelo del sistema estocástico y el diseño de la política de control utilizando la Teoría de Lyapunov [50]. En la Tabla 4.1 se resumen los símbolos utilizados en el modelo propuesto y su significado. Se utiliza el término "servicio" para referirse a conjuntos de paquetes sobre los que se necesita aplicar cierta computación.

Como se ha descrito con detalle en el Capítulo 3 se considera un sistema compuesto por nodos Fog y Cloud con diferentes capacidades de procesamiento. En este escenario, múltiples aplicaciones de usuario generan servicios, compuestos por paquetes, que se envían a los nodos Fog. A continuación, los servicios pueden procesarse localmente (en los mismos nodos Fog) o ser enviados al Cloud. El sistema incluye también un orquestador, o nodo Master, que toma las decisiones de offloading, dependiendo de la política particular implementada.

Sea M el número de aplicaciones que generan servicios. Se supone que el tiempo transcurre en slots, y que cada aplicación genera un servicio en cada slot. A su vez, el número de paquetes por servicio sigue una cierta distribución aleatoria. Los paquetes de los servicios generados se almacenan localmente en las

Tabla 4.1: Símbolos y variables del modelo del sistema.

Notación	Descripción
N	número de puntos de procesamiento, tales como CPUs en el Fog e instancias Cloud.
M	número de aplicaciones independientes generando servicios para ser procesados.
$a_m(t)$	llegadas a la cola de la aplicación m en el slot t, medido en paquetes.
$b_m(t)$	salidas desde la cola de la aplicación m en el slot t, medido en paquetes.
$Q_m(t)$	tamaño de la cola de la aplicación m en el slot t, medido en paquetes.
$\alpha_{m,n}(t)$	decisión para la aplicación m y la CPU n en el slot t, medido en número de paquetes.
$\alpha(\mathbf{t})$	$M \times N$ matriz de las variables de decisión.
$\mathcal{A}(t)$	conjunto de decisiones admisibles en el slot t.
$\omega_n(t)$	tasa de transferencia de la opción de procesamiento n en el slot t. Refleja la variación de la capacidad de procesamiento disponible.
$g_m(t)$	complejidad de procesado de los servicios de la aplicación n en el slot t.
$C(t)$	coste monetario por el uso del Cloud en el slot t.
$k_n(t)$	coste genérico de usar el punto de procesamiento en el slot t.
$E_n(t)$	coste energético del punto de procesamiento n en el slot t.
E_nth	umbral de energía del punto de procesamiento n.
$G_n(t)$	cola virtual relativa al consumo de energía en el punto de procesamiento n en el slot t.
$\hat{\bullet}$	se utiliza para indicar una función arbitraria que produce una variable $\bullet$.

colas de las aplicaciones. Se supone que el escenario tiene N alternativas de procesamiento, incluyendo procesadores locales (alternativas $1, \ldots, N-1$) y una nube (alternativa N). En cada slot el nodo Master establece la cantidad de datos de cada aplicación a procesar en cada alternativa: o procesamiento local o la nube, satisfaciendo algunas restricciones. La política de offloading debe garantizar que las colas de aplicaciones permanezcan estables, con el fin de evitar que se produzcan retrasos no aceptables. En este escenario, se aplica la Teoría de Lyapunov, ampliamente utilizada en optimización estocástica, para garantizar la estabilidad del sistema.

Sea $a_m(t)$ la cantidad de paquetes que llegan a la cola de aplicación m, $m \in \{1, \ldots, M\}$, en la ranura t, y $b_m(t)$ el número de paquetes drenados de esa cola como consecuencia de la política aplicada. La dinámica de colas viene dada por la Ecuación (4.1), donde $Q_m(t+1)$ es la cola de espera de la aplicación m en el slot t.

$$Q_m(t+1) = \text{máx}[Q_m(t) - b_m(t), 0] + a_m(t) \tag{4.1}$$

El objetivo es garantizar la estabilidad de la tasa media de las colas de las aplicaciones, que viene dad por la Definición 1.

Definición 1 (Estabilidad de la tasa media) *Una cola es estable en tasa media si:*

$$\lim_{T\to\infty} \frac{1}{T}\, \mathbb{E}\{Q(t)\} = 0 \tag{4.2}$$

donde $Q(t)$ es el tamaño de la cola en el slot t, y $\mathbb{E}$ es la esperanza matemática.

Sea $\alpha(t)$ una matriz MxN tal que el elemento $\alpha_{m,n}(t)$ corresponde a la cantidad de datos de la aplicación m que se asigna al procesador n en el slot t. En cada slot se toma una decisión $\alpha(t)$ dentro de un conjunto $\mathcal{A}(t)$ de posibles elecciones. Además, se supone que existe una tasa de transferencia de datos de cada opción de procesamiento, que dicta cuántos bytes se pueden aceptar en un slot determinado.

Entonces, es posible definir $b_m(t)$ según la Ecuación (4.3), donde $\omega(t)$ es la tasa de transferencia de cada procesador en el slot t, en bytes por slot. En general, se supone que la tasa de transferencia varía con el tiempo, siguiendo una distribución arbitraria. Como puede observarse, la cantidad de datos drenados por cada aplicación es función de la decisión y de la tasa de transferencia de los procesadores. La Figura 4.2 muestra un ejemplo del modelo del sistema con 3 aplicaciones y 2 procesadores. Cabe señalar que la tasa de transferencia puede verse influida tanto por la capacidad de cálculo de la CPU como por la capacidad de comunicación entre la cola de aplicaciones y el procesador. Por ejemplo, en el procesamiento local la tasa de transferencia estaría dominada por la capacidad de cálculo, mientras que en el caso del procesamiento remoto (datos que deben enviarse al Cloud) estaría limitada por la capacidad de comunicación.

$$b_m(t) = \hat{b}(\alpha(t), w_1(t), w_2(t), \dots) = \hat{b}(\alpha(t), \omega(t)) \tag{4.3}$$

Para evitar la asignación de paquetes inexistentes, en cada slot se impone que la cantidad total de bytes asignados de una cola de aplicación i en la ranura t no supere los bytes de la cola correspondiente, como se indica en la Ecuación (4.4).

Además, se asegura que la asignación no supere la capacidad de transferencia, con la restricción definida en la Ecuación (4.5), donde $g_i(t)$ denota un factor de escala genérico. Por ejemplo, en el caso de una tasa de transferencia limitada por la capacidad de cálculo, este parámetro estaría relacionado con la complejidad de cálculo del servicio. De este modo, los servicios más complejos producirían tiempos de cálculo más lentos para el mismo número de bytes, lo que se representa escalando el número de bytes, mientras se mantiene constante la capacidad de cálculo.

$$\sum_{j=1}^{N} \alpha_{ij}(t) \leq Q_i(t) \quad \forall i \in \{1, \dots, M\}, \forall t \tag{4.4}$$

$$\sum_{i=1}^{M} g_i(t) \cdot \alpha_{ij}(t) \leq w_j(t) \quad \forall t, \forall j \tag{4.5}$$

Ahora se puede reescribir la salida $b_m(t)$, como se muestra en la Ecuación (4.6).

$$b_m(t) = \hat{b}(\alpha(t), \omega(t)) = \sum_{j=1}^{N} \alpha_{m,j}(t) \tag{4.6}$$

También se consideran las penalizaciones por utilizar las distintas alternativas de procesamiento. Se utiliza el símbolo $k_i(t)$ para denotar el coste de utilizar la alternativa de procesamiento i en el slot t. Las penalizaciones se definen para las CPU del Cloud y del Fog de manera diferenciada. Para el procesamiento en el Cloud el objetivo es minimizar el coste monetario, que está relacionado con la potencia de cálculo necesaria, tal y como se define en la Ecuación (4.7).

$$C(t) = \hat{C}\left(\sum_{i=1}^{M} g_i(t) \cdot k_N(t) \cdot \alpha_{i,N}(t)\right) \tag{4.7}$$

donde $g_i(t)$ se corresponde, como ya se ha mencionado, a la complejidad de cálculo de los servicios generados por la aplicación i, la alternativa de procesamiento N corresponde al Cloud, y así $k_N(t)$ es el coste monetario de utilizar el Cloud en el slot t. Como puede verse, el coste es proporcional a la cantidad de tráfico enviado a la instancia del Cloud, escalado por la complejidad computacional. En general, se

supone que tanto la complejidad computacional de los servicios de cada aplicación como la tarifa del Cloud pueden variar con el tiempo, siguiendo distribuciones aleatorias arbitrarias.

En cambio, el procesamiento local en el Fog podría estar limitado por el consumo energético. En este caso, no se busca minimizar la energía, sino garantizar que, en promedio, se mantenga por debajo de un determinado valor. Esto sería necesario, por ejemplo, para los dispositivos alimentados por baterías que pueden recargarse periódicamente. Se define la restricción energética de la Ecuación (4.8).

$$E_j(t) = \sum_{i=1}^{M} g_i(t) \cdot k_j \cdot \alpha_{i,j}(t) \quad \forall j \neq N \tag{4.8}$$

donde k_j representa un mapeo general entre el número de bytes que hay que procesar, escalado por la complejidad del procesamiento, y la energía necesaria. A diferencia del Cloud, el coste asociado a la energía (k_j) dependería sobre todo de las características del hardware del procesador, por lo que se supone que no varía: $k_j(t) = k_j \ \ \forall t \, \forall j \neq N$. En ambos casos, se tienen en cuenta las penalizaciones a lo largo del tiempo, por lo que se utilizan sus expectativas medias temporales $\overline{C}$ y $\overline{E}$, que se definen en las Ecuaciones (4.9) y (4.10) respectivamente.

$$\overline{C} = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} \mathbb{E}\{C(t)\} \tag{4.9}$$

$$\overline{E} = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} \mathbb{E}\{E(t)\} \tag{4.10}$$

En conjunto, se establece una política de control a partir del problema de optimización enunciado en el Problema 1

Problema 1

$$\min_{\alpha(t)} \quad \overline{C} \tag{4.11}$$

$$\textit{s.t.} \quad \overline{E}_j \leq E_j^{Th} \quad \forall j \in \{1, \ldots, N-1\} \tag{4.12}$$

$$\alpha(t) \in \mathcal{A}(t) \tag{4.13}$$

donde E_j^{Th} es el umbral de energía definido para cada procesador de un nodo Fog, y $\mathcal{A}(t)$ se cumple para el conjunto de restricciones definidas en las Ecuaciones (4.4) y (4.5) en cada slot. Utilizando el framework de optimización estocástica desarrollado en [50], las desigualdades relacionadas con la limitación del consumo de energía pueden convertirse en colas virtuales, al igual que las colas de aplicaciones definidas anteriormente. Así, la actualización de la cola virtual asociada a la energía del procesador de Fog j, G_j, se define en la Ecuación (4.14).

$$G_j(t+1) = max\{G_j(t) + (E_j(t) - E_j^{Th}), 0\} \tag{4.14}$$

La cola virtual se introduce como un método fuerte para garantizar que se satisface la restricción media de consumo de energía requerida. Así, se puede definir el conjunto de colas (de aplicaciones y la cola virtual) como $\Theta(t)$. La función de Lyapunov $L(\Theta(t))$ y la deriva (drift) $\Delta(\Theta(t))$ se definen como se muestra en (4.15) y (4.16).

$$L(\Theta(t)) = \frac{1}{2}\left(\sum_{j=1}^{N} G_j(t) + \sum_{i=1}^{M} Q_i(t)\right) \tag{4.15}$$

$$\Delta(\Theta(t)) = \mathbb{E}\{L(\Theta(t+1)) - L(\Theta(t))|\Theta(t)\} \tag{4.16}$$

La solución al Problema 1 es el algoritmo drift-plus-penalty. En cada slot t, se observa el estado de las colas, y se toma una decisión que resuelve el Problema 2, donde V es un factor de ponderación positivo que establece el compromiso entre la deriva y la penalización. Se trata de un problema de programación lineal entera (ILP), que puede resolverse con herramientas existentes.

Problema 2

$$\min_{\alpha(t)} \quad V \cdot C(t) + \sum_{i=1}^{M} Q_i(t)[a_i(t) - b_i(t)] + \tag{4.17}$$

$$\sum_{j=1}^{N} G_j(t)(E_j(t) - E_j^{Th}) \tag{4.18}$$

$$\textbf{\textit{s.t.}} \quad \sum_{j=1}^{N} \alpha_{ij}(t) \leq Q_i(t) \quad \forall i \in \{1, \ldots, M\}, \forall t \tag{4.19}$$

$$\sum_{i=1}^{M} g_i(t) \cdot \alpha_{ij}(t) \leq w_j(t) \quad \forall t, \forall j \tag{4.20}$$

5 Resultados

En esta sección se lleva a cabo una campaña de experimentos sobre la plataforma desarrollada, que busca analizar el rendimiento de la propuesta de algoritmo de offloading. En primer lugar, se estudian las propiedades de la solución propuesta en dos escenarios sintéticos, con y sin opción de procesamiento en el Cloud. De esta forma, el primer escenario se centra en el equilibrio entre las limitaciones en el consumo de energía y el coste monetario, mientras que la última configuración presta especial atención al impacto que la limitación de energía puede tener sobre el tráfico entrante de la aplicación. A continuación, se evalúa el esquema propuesto en un tercer escenario más realista, en términos de capacidad de procesamiento y generación de tráfico.

En estas tres configuraciones, el rendimiento de la solución propuesta se compara con el observado con un algoritmo simple basado en Round Robin, el cual se utiliza, por tanto, como referencia. Además, se incluye un cuarto apartado donde se lleva a cabo una comparativa con otras soluciones. Para ello, se tienen en cuenta distintas métricas de rendimiento. En todas las pruebas se opta por configurar un único nodo Fog con tres aplicaciones y dos procesadores, un nodo Cloud y un nodo Master, que ejecuta los algoritmos. Los detalles de la configuración base utilizada se muestran en la Tabla 5.1. Como puede observarse, en todos los casos se realizan ejecuciones que transcurren en 1000 slots de de 1 segundo cada uno. En cada slot, las aplicaciones generan un servicio consistente en un número aleatorio de paquetes. En la Tabla 5.1 se muestra la tasa media agregada de las aplicaciones, y en cada escenario se especificará la tasa particular de cada aplicación. Como puede observarse en esa misma tabla algunas variables aleatorias arbitrarias se definen como constantes, para simplificar la interpretación de los resultados.

Tabla 5.1: Configuración común de la plataforma para la campaña de simulaciones.

Parámetro	Valor
Slots simulados	1000
Duración del slot	1 `s`
Capacidad de procesamiento de una CPU en el Fog	1 `kB/s`
Capacidad de procesamiento en el Cloud	100 `kB/s`
Longitud de un paquete	$200 + 12$ `bytes`
Tasa de tráfico media agregada	Poisson $[6, \ldots, 30]$ `pkt/s`
Tasa de generación de servicios	1 `serv/s`
Factor de energía del Fog (k_j)	1
Coste del Cloud (k_N)	1
Complejidad de la aplicación (g_m)	1
Umbral de energía	$[1, \ldots, 5]$
Número de aplicaciones	3
Número de CPUs en un nodo Fog	2

5.1. Colaboración Fog y Cloud

Con la primera configuración se pretende analizar el equilibrio de procesamiento entre los nodos Fog y Cloud en diferentes configuraciones. Cabe destacar que la capacidad de procesamiento del nodo Cloud, teniendo en cuenta el tráfico generado por las aplicaciones y la alta capacidad que típicamente tienen los centros de datos, puede considerarse infinita.

En primer lugar, se procede a evaluar el impacto del parámetro V sobre el offloading de computación. Cabe recordar que dicho parámetro ajusta el compromiso entre consumo energético y coste monetario. La Figura 5.1 muestra la proporción de tráfico enviado al nodo Cloud en función de la tasa de tráfico media agregada. En esta configuración, se fija el valor umbral de energía, E_{th}, en 2. Además, se realizan simulaciones para distintos valores de V. La figura también muestra, con líneas discontinuas, los resultados obtenidos al utilizar el algoritmo Round Robin (RR) y una variante de Round Robin que tiene en cuenta el umbral de energía prefijado (RRe). La primera consume toda la capacidad de procesamiento en el Fog, enviando el excedente a la nube. La segunda realiza una selección round-robin entre los procesadores Fog, sin superar el umbral de energía, enviando entonces el resto de paquetes a la nube. Para cada configuración, la Figura 5.1 muestra el uso de la nube para experimentos que duran 1000 slots.

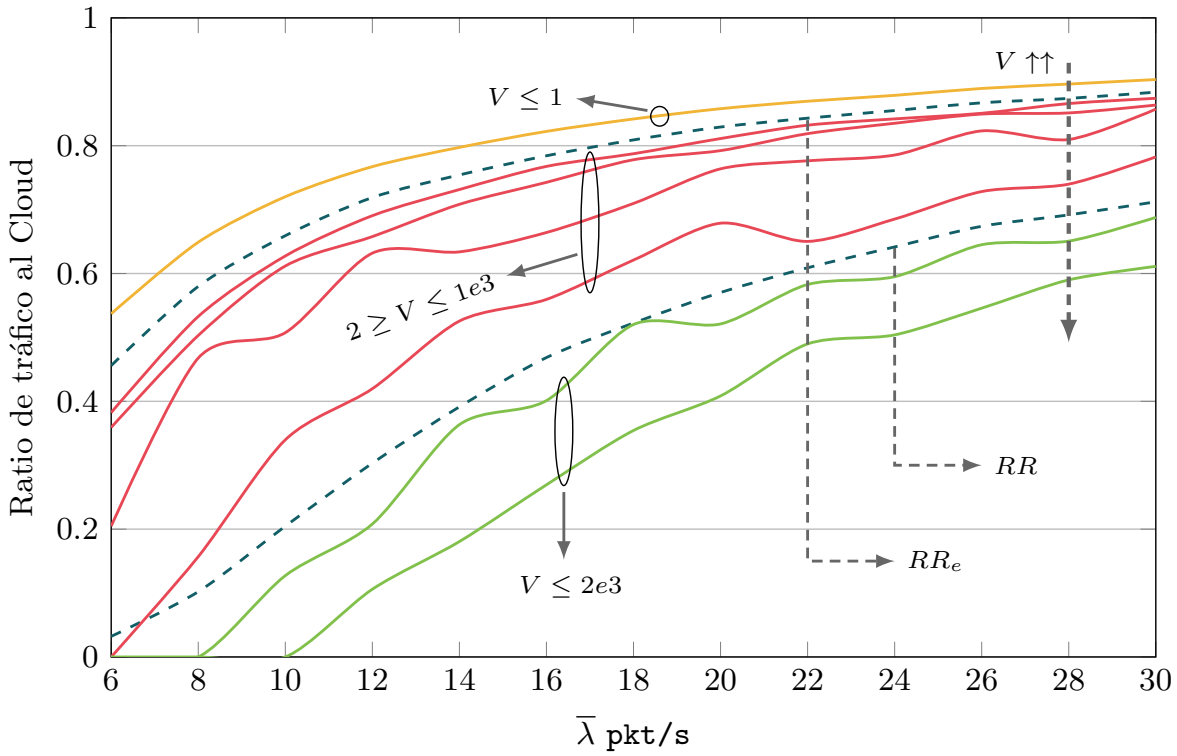


Figura 5.1: Uso del Cloud frente a la variación de la tasa de tráfico agregada.

Como era de esperar, se puede observar que una mayor tasa de tráfico conlleva más carga de procesado en la nube. Además, a medida que aumenta el valor de V, se ve una tendencia decreciente en el uso de la nube, como era de esperar. También se pueden identificar tres regiones de operación diferentes para la solución propuesta, a medida que se modifica V, delimitadas por los algoritmos Round Robin.

En la primera región, se pone de manifiesto que se envía más tráfico al Cloud que con el algoritmo RRe. Esto ocurre con valores de V inferiores a 1, donde se da muy poca importancia al coste monetario. El resultado es que no se utiliza la máxima capacidad de procesamiento disponible en el Fog, aunque no

se alcance el umbral de energía. La segunda región corresponde a valores de V comprendidos entre 2 y 1000, y el rendimiento observado se sitúa entre las dos versiones Round Robin. En esta región, los valores más altos de V reducen significativamente el uso del Cloud. A su vez, conduce a una eventual saturación de los procesadores del Fog. Para cumplir el umbral de energía, las soluciones propuestas mantienen el tráfico en las colas de aplicaciones, y equilibran las decisiones para garantizar la estabilidad del sistema, teniendo en cuenta las colas de aplicaciones, la energía y el coste.

Se analiza además este efecto estudiando cómo afectan las distintas configuraciones al indicador de rendimiento energético. En la Figura 5.2 se representa, con un diagrama de barras, el consumo medio de energía observado con las soluciones propuestas para distintas configuraciones de la tasa de tráfico agregada, y para diversos valores de V. Cabe destacar que en todos los casos el umbral de energía se fija en 2, y que la capacidad de procesamiento agregada en el nodo Fog es de 4 `pkt/s`.

Como puede observarse, cuando V se encuentra dentro de la primera región identificada en la Figura 5.1 ($V = 1$), el consumo medio de energía está muy por debajo del umbral, independientemente de la tasa de tráfico. A medida que se incrementa el valor de V se observa que el esquema propuesto no es capaz de mantener la energía por debajo del umbral, debido al alto coste de uso de la instancia Cloud. Además, los resultados muestran que el impacto de V también varía con la tasa de tráfico. En este sentido, el consumo de energía se satura con $V = 1e3$ para la tasa de tráfico más baja (6 `pkt/s`), mientras que este valor de saturación aumenta para tasas más altas. La razón es que los valores moderados de V, con tráfico bajo, obligan a procesar todos los servicios en el Fog, ya que la cola de energía virtual no crece mucho. En cambio, con tasas más altas, la cola virtual aumenta, por lo que algunos servicios acaban siendo enviados al Cloud, a menos que el valor de V también aumente.

Este primer conjunto de resultados valida el comportamiento del esquema propuesto, mostrando que es capaz de equilibrar la carga computacional, considerando diferentes parámetros (energía, coste y colas de aplicaciones). Además, puede ser configurado para fomentar diferentes comportamientos, gracias al parámetro de configuración V.

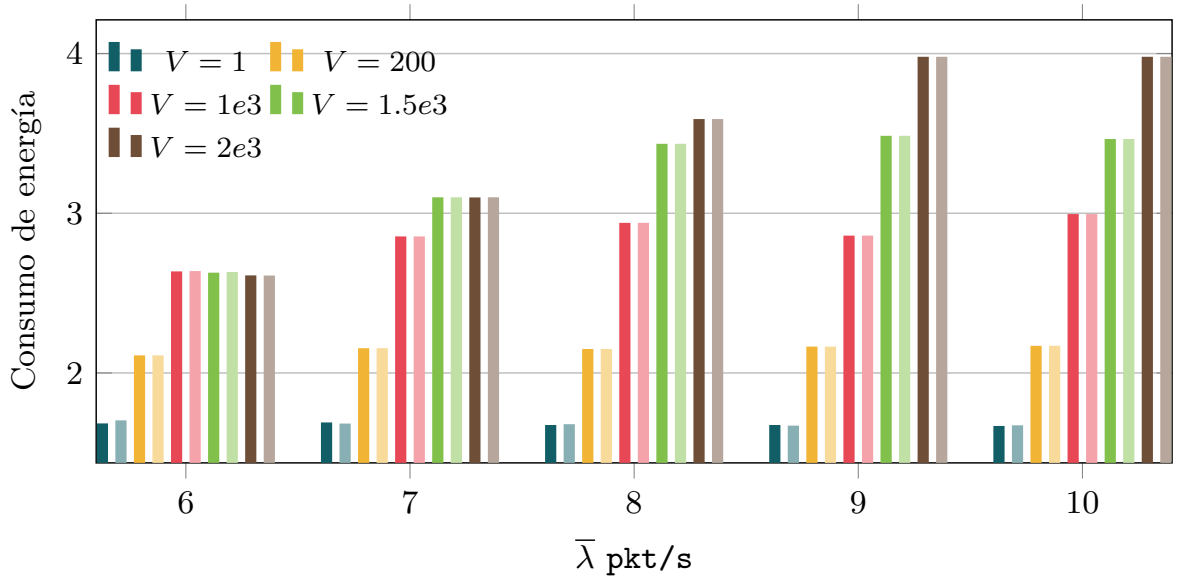


Figura 5.2: Coste de energía promedio frente a la variación de la tasa de tráfico agregada.

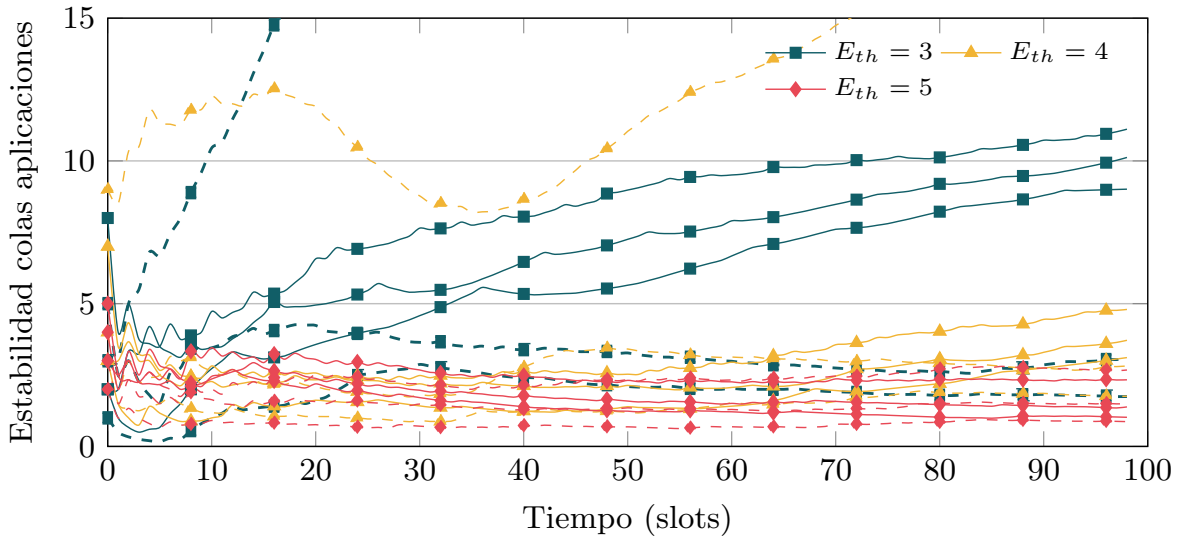
5.2. Análisis de rendimiento en Fog

El presente conjunto de resultados se centra, con mayor detalle, en el impacto que tienen las distintas configuraciones sobre la energía y las colas de aplicaciones en el Fog. Como se ha visto en la sección anterior, valores bajos de V evitarían la sobrecarga de los nodos Fog, puesto que obligan a enviar muchos servicios al Cloud. Teniendo esto en cuenta, se considera una situación en la que el Cloud y su alta capacidad de procesamiento no están disponibles, lo que de hecho sería equivalente a tener un V alto. Sin el Cloud, es posible evaluar configuraciones más críticas en términos de capacidad de procesamiento, lo que permite observar más de cerca la estabilidad de las colas de aplicaciones.

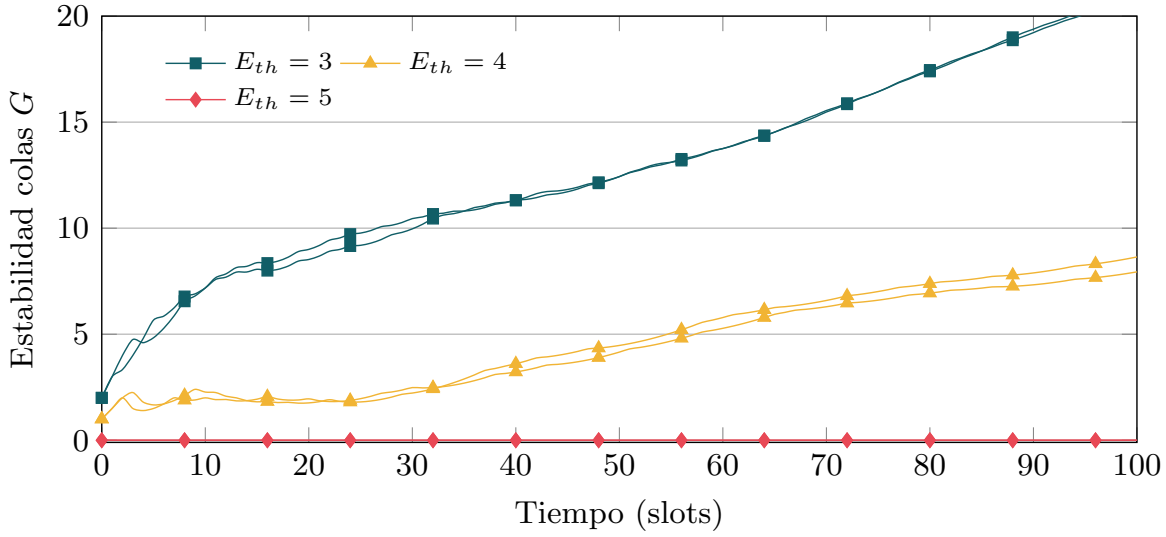
En este caso, la tasa media agregada de tráfico se fija en 7 `pkt/s`. En concreto, las aplicaciones, primera, segunda y tercera, generan 1, 2 y 4 `pkt/s`, respectivamente. La Figura 5.3 muestra la estabilidad temporal de las colas de aplicaciones y energía, utilizando la Ecuación 4.2, para distintos valores del umbral de energía. Cabe señalar que los umbrales grandes conducen a no imponer ningún límite al consumo de energía. En aras a conseguir una mayor claridad en la representación de los resultados, se muestran únicamente los correspondientes a los 100 primeros slots. La figura ilustra la estabilidad de la cola de las aplicaciones, obtenida con el algoritmo propuesto y la correspondiente a RRe (líneas discontinuas). Los distintos colores reflejan la estabilidad de las distintas colas de las aplicaciones.

En general, se puede observar que el algoritmo propuesto consigue mantener la estabilidad de todas las colas de las aplicaciones, independientemente de los límites establecidos en el consumo de energía. Como puede observarse, cuando se utiliza Round Robin con el umbral de energía fijado en 3 y 4, hay una cola muy inestable, que corresponde a la aplicación con mayor tasa, mientras que las demás muestran valores bastante bajos. Por otra parte, la solución propuesta es capaz de adaptarse a las tasas de tráfico, como demuestra el hecho de que todas las colas presenten valores similares. Cuando se utiliza un umbral de 3, no se puede garantizar la estabilidad, pero con restricciones energéticas más suaves, ésta se alcanza rápidamente. La Figura 5.3b muestra la estabilidad de la cola virtual de energía (G_j) de cada CPU del nodo Fog. Los resultados reflejan una tendencia similar a la observada para las colas de las aplicaciones. A medida que se relaja la restricción energética, el esquema propuesto es capaz de estabilizar ambos tipos de colas, mientras que penaliza aquellas con requisitos más estrictos.

Se estudia ahora el impacto de la tasa de tráfico sobre las colas. La Figura 5.4 utiliza una representación de su estabilidad, similar a las anteriores. En este caso se fija el umbral de energía en 3,5, y se realizan experimentos para distintos valores de la tasa de tráfico de la tercera aplicación. Las demás tasas no cambian (1 y 2 `pkt/s`, respectivamente). Los resultados muestran que la estrategia Round Robin no es capaz de estabilizar la cola correspondiente a la tercera aplicación a medida que su tasa aumenta. Como puede observarse, cuando la tercera aplicación se configura con 4 `pkt/s`, la estabilidad de la cola correspondiente supera los límites del gráfico en el slot 15, y muestra una tendencia creciente incluso cuando la tasa de la aplicación se fija en 3 `pkt/s`. Por el contrario, el algoritmo propuesto consigue mantener estables todas las colas, incluso con la tasa más alta. Además, cuando la tasa de la tercera aplicación se fija en 4 `pkt/s`, la cola tiende a un valor estable dentro del intervalo de tiempo mostrado. Al mismo tiempo, los resultados ponen de manifiesto que el esquema propuesto también es capaz de estabilizar las colas de energía mostradas en la Figura 5.4b.



(a) Colas de las aplicaciones. Round Robin está representado con líneas discontinuas.



(b) Colas virtuales de energía.

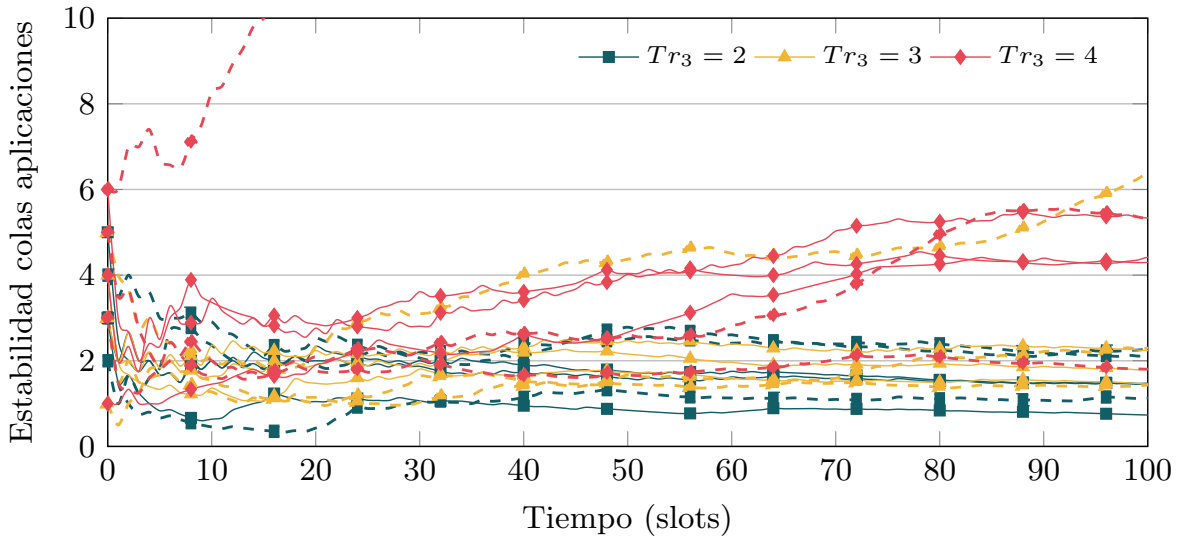
Figura 5.3: Evolución de las colas debido a la variación del umbral de energía.

5.3. Entorno realista de generación de tráfico

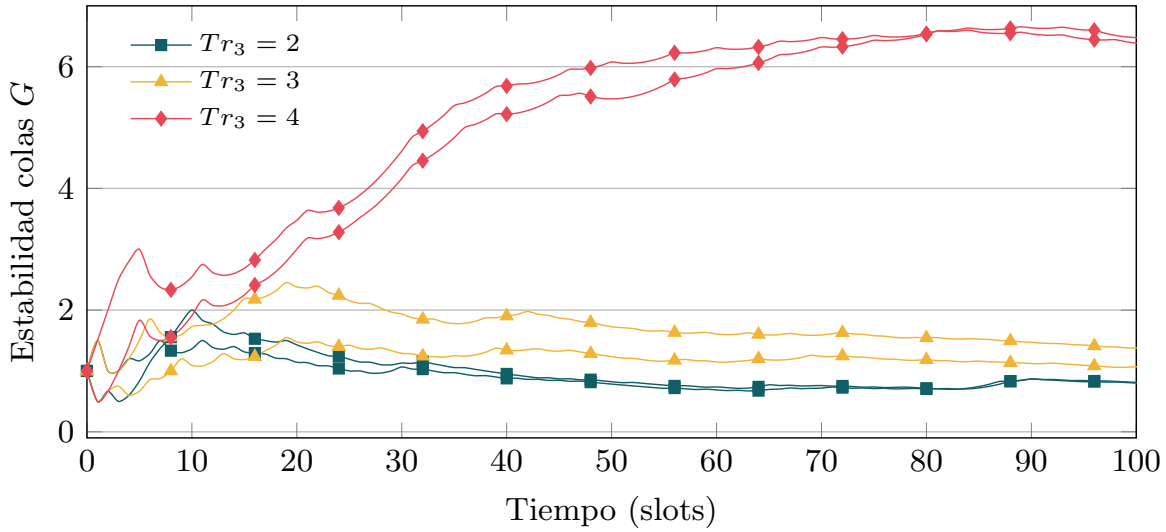
Para validar el rendimiento del esquema de offloading propuesto se amplía la evaluación, utilizando un entorno más realista. En concreto, se ajustan las distribuciones de tráfico de las aplicaciones, así como las capacidades de los dispositivos. Esto se refleja en la Tabla 5.2.

Los valores se obtienen de [51], donde los autores recogieron un rastro de carga de trabajo de E/S en un periodo de dos semanas de AliCloud, uno de los mayores proveedores de nube de Asia. Utilizando la traza de AliCloud, caracterizaron la carga de trabajo de E/S y la distribución de datos. Como se muestra en la Tabla 5.2, el tráfico sigue una distribución lognormal, que fue la opción que mejor se ajustó en [51]. En concreto, el valor esperado y la varianza de la distribución normal subyacente se fijan en $4{,}5$ y $0{,}8$, respectivamente. A su vez, se obtiene una tasa media de tráfico de 125 `pkt/s`[1] Además, se escala

[1]El valor esperado de la distribución lognormal $\mathcal{X}$ viene dado por $\mathbb{E}\{\mathcal{X}\} = \exp(\mu + \frac{\sigma^2}{2})$, donde μ y σ^2 son el valor esperado y la varianza de la distribución normal $\mathcal{N}$, de modo que $ln(\mathcal{X}) \sim \mathcal{N}(\mu, \sigma^2)$.



(a) Colas de las aplicaciones. Round Robin está representado con líneas discontinuas.



(b) Colas virtuales de energía.

Figura 5.4: Evolución de las colas debido a la variación de la tasa de tráfico de una aplicación.

la capacidad del nodo Fog en función de las nuevas tasas de tráfico, de forma que pueda hacer frente cómodamente, en media, al tráfico generado por las tres aplicaciones. Así, el escenario configurado permite que el umbral de energía desempeñe un papel importante en los resultados de la simulación.

Tabla 5.2: Configuración de la simulación del entorno realista.

Parámetro	Valor
Slots simulados	1000
Duración del slot	1 s
Capacidad de procesamiento de una CPU en el Fog	100 kB/s
Capacidad de procesamiento en el Cloud	10^6 kB/s
Umbral de energía	$[75, \ldots, 120]$
Distribución de tráfico para cada aplicación	*lognormal*$(4{,}5; 0{,}8)$

Bajo esta configuración se pretende representar configuraciones óptimas del algoritmo según ciertas

limitaciones de coste monetario y con un consumo energético adecuado. En este sentido, la Figura 5.5 muestra una representación en dos ejes del coste monetario (eje izquierdo) y el consumo de energía (eje derecho) con líneas sólidas y discontinuas, respectivamente. Las líneas representan el valor medio obtenido a partir de 30 experimentos independientes, cada uno de los cuales con una duración de 1000 slots. Junto con los valores medios, también se representan el máximo y el mínimo obtenidos durante las simulaciones con el fondo sombreado. Se muestran los resultados a medida que se incrementa el valor del umbral de energía E_{th}, y para distintos valores del parámetro V.

Como era de esperar, al relajar el umbral de energía, el coste disminuye, ya que se procesan más datos en el Fog, mientras que el consumo de energía crece. Las intersecciones corresponden a los puntos (configuraciones) en los que ambos costes se igualan. En este sentido, el esquema propuesto permite establecer configuraciones (V y E_{th}) que equilibran los costes energéticos y monetarios. Como puede observarse, para el escenario considerado, E_{th} debe fijarse entre 90 y 115 para todo el rango de valores de V. En la Tabla 5.3 se indican los puntos de intersección de más valores de V que no se incluyen en la Figura 5.5, para simplificar la representación. Se puede realizar un análisis similar para diferentes relaciones entre el consumo de energía y el coste del Cloud, o fijando el umbral de energía en lugar del coste del Cloud.

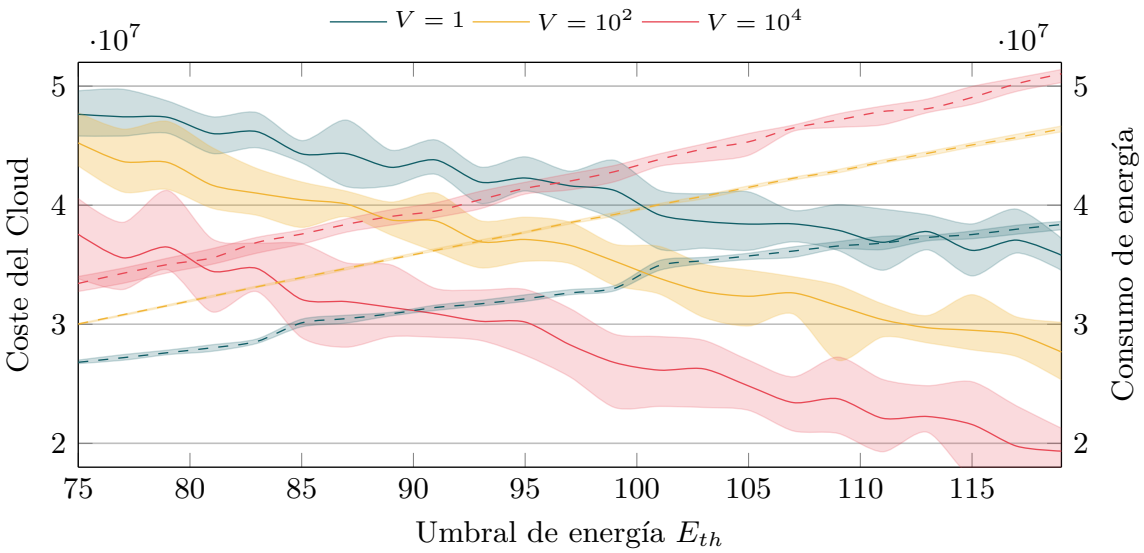


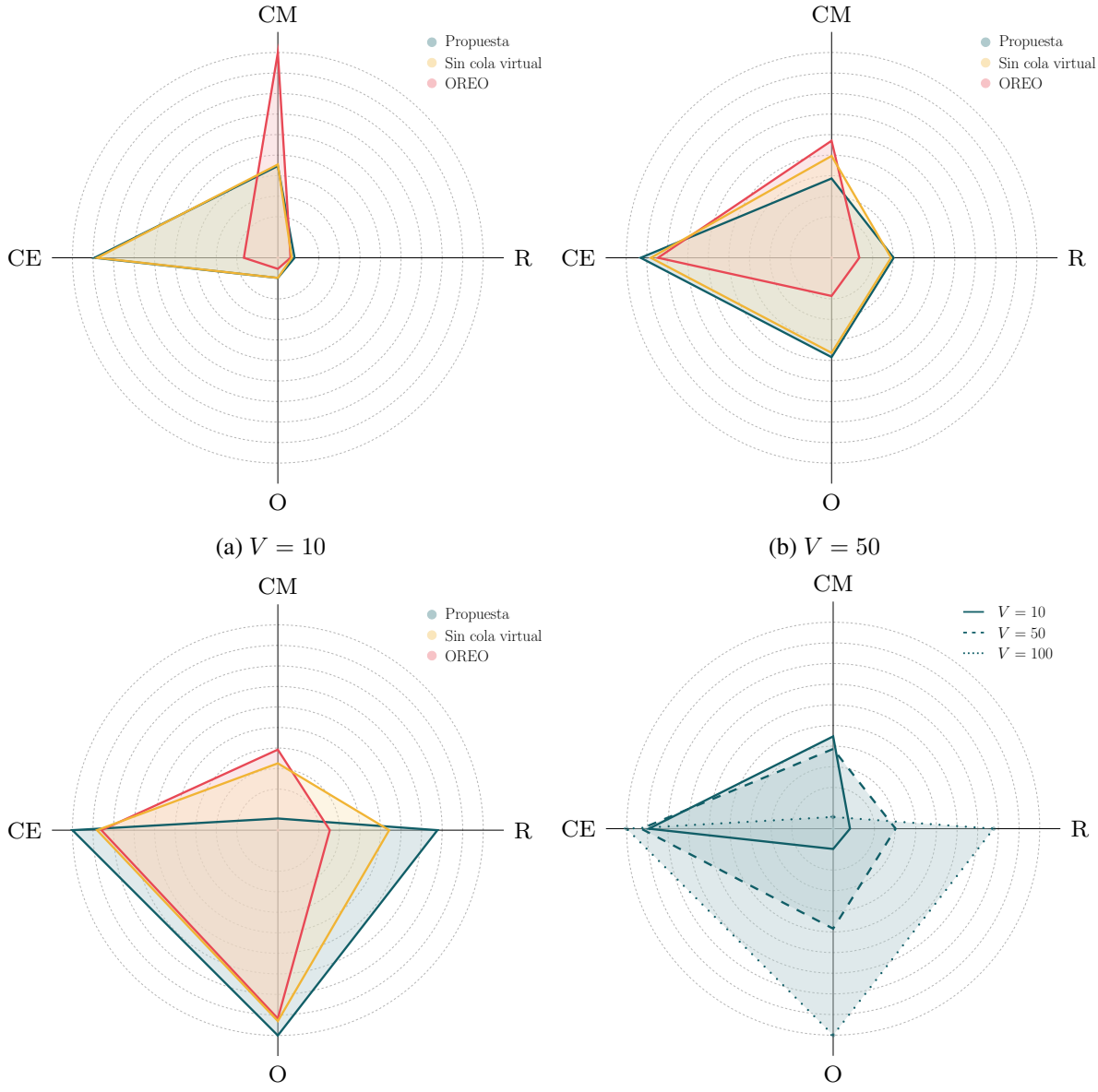
Figura 5.5: Coste del Cloud y consumo de energía en función de V y E_{th}.

Tabla 5.3: Puntos de intersección en función del parámetro V.

V	1	10	10^2	10^3	10^4
E_{th}	80,13	91,61	92,96	108,61	113,54
Coste ($\cdot 10^6$)	35,33	37,05	36,96	37,07	37,36

5.4. Comparativa de rendimiento con otras soluciones

La presente sección tiene como objetivo realizar una comparativa de rendimiento entre diferentes soluciones de offloading. Para ello, se consideran diferentes métricas de rendimiento: consumo de energía, coste monetario, ocupación media de las colas de las aplicaciones y retardo medio de cada servicio. Los



(a) $V = 10$

(b) $V = 50$

(c) $V = 100$

(d) Algoritmo propuesto para $V = 10, 50$ y 100.

Figura 5.6: Diagrama comparativo entre algoritmos y configuraciones. Se compara en términos de coste monetario (CM), retardo medio por servicio (R), ocupación media de las colas (O) y coste de energía (CE).

algoritmos a estudiar son el algoritmo propuesto, una modificación sin cola virtual de energía y una adaptación de OREO [44], algoritmo basado también en la Teoría de Lyapunov, aunque con objetivos diferentes en el problema de optimización que plantea. Se pretende de esta forma ver el impacto de la cola de energía en el algoritmo y, por otra parte, ver los resultados que se podrían obtener con algoritmos que se enfocan en otros objetivos.

Para llevar a cabo la comparativa se configura un escenario similar al de la Sección 5.1, donde se realizan varias simulaciones de 1000 slots, con diferentes valores de V. Como se ha visto anteriormente, este parámetro de configuración permite, para el algoritmo propuesto, regular el compromiso entre el consumo de energía y el coste monetario debido al uso del Cloud. Por su parte, OREO cuenta también con un parámetro V que permite adaptar su funcionamiento, pero que, sin embargo, regula el consumo

de energía y el peso que se le da al retardo en el problema de optimización. El retardo inducido por el procesado en el nodo Fog es provocado, en los experimentos realizados, por el tiempo de procesado en dicho nodo. Por su parte, en el caso del procesamiento remoto en el nodo Cloud, y para simplificar la implementación de OREO, se opta por asumir que el retardo es el doble que el que se obtendría al procesar en el Fog.

La Figura 5.6 muestra los resultados obtenidos para los tres algoritmos y para tres valores de V. Como se puede observar en la Figura 5.6a, para un valor de V bajo, los retardos para todas las soluciones son relativamente pequeños. Sin embargo, el comportamiento mostrado es diferente. En el caso de OREO, el coste monetario resulta notablemente más elevado, debido a que una V tan baja implica que el algoritmo centre sus esfuerzos en reducir el consumo de energía. Al aumentar V, se observa como los algoritmos reducen el uso del Cloud. En el caso de OREO al darle menos importancia al consumo de energía y en el caso del algoritmo propuesto por penalizar más el procesamiento en el Cloud. En ambos casos, sube la ocupación en las colas de las aplicaciones, es decir, los algoritmos optan por no procesar los servicios y mantenerlos en cola, eso sí, vigilando que se mantenga la estabilidad de las colas. Este aumento en la ocupación se traduce también en un incremento en los retardos. Los servicios permanecen en la cola y tardan varios slots en ser completamente procesados. En este sentido, resulta fácil notar que esta fragmentación de los servicios afecta negativamente al retardo.

En cuanto a la cola virtual de energía, su uso permite sobrepasar el umbral de energía configurado. Esto se observa en la Figura 5.6c, donde tanto la variante sin cola virtual como OREO, respetan de forma más estricta el umbral de energía. Como contrapartida, no logran reducir en igual medida el coste monetario. En el caso de la variante sin cola virtual, la limitación en el uso de energía es estricta, al incluirse como restricción en el propio problema de optimización. Por su parte, OREO, al ser configurado con una V elevada, actúa de una forma similar. Dispone de una cola virtual de energía, pero puesto que no tiene en cuenta el coste monetario, una vez alcanzado el umbral de energía, resulta muy complicado que lo supere, puesto que solo un aumento drástico en el retardo de los servicios podría producir que el consumo energético se dispare.

Finalmente, la Figura 5.6d recoge el comportamiento del algoritmo propuesto para los tres valores de V anteriores. Esta figura permite ver de forma más clara cómo afecta la configuración a la ocupación de las colas y al retardo, complementando los resultados mostrados en las secciones anteriores, los cuales se enfocaban únicamente en los costes. Así, se observa cómo la reducción en el coste monetario trae consigo inevitablemente una penalización, al no disponer de recursos suficientes para procesar la gran cantidad de datos que generan las aplicaciones. El uso de una cola virtual y el objetivo de reducir el coste monetario, en especial para valores de V elevados, otorga cierta libertad al algoritmo para poder sobrepasar los límites de energía configurados. Con ello logra suavizar los efectos de aumentos puntuales de tráfico, donde el algoritmo puede decidir emplear más recursos del Fog, superando el consumo de energía objetivo. A continuación, para compensar este exceso de consumo, durante períodos con menor demanda, el uso del Fog se reduce, del tal forma que, en media, el consumo de energía se corresponde, idealmente, con el que se había configurado previamente.

6 Conclusiones y líneas futuras

El enorme volumen de datos generados por los dispositivos IoT precisa encontrar una arquitectura adecuada, capaz de procesar y almacenar todos estos servicios. Si bien las alternativas basadas en la nube se utilizan actualmente con ese propósito, se vislumbra que el nuevo paradigma de Fog Computing permitirá escalar y optimizar las infraestructuras de IoT. Mientras que Cloud Computing tiene el potencial de satisfacer muchos de los requisitos de IoT, como la monitorización de servicios, el procesamiento de grandes flujos de datos y las tareas de visualización; las soluciones basadas en Fog Computing están diseñadas para abordar el procesamiento en tiempo real, la respuesta rápida a eventos y peticiones, y los problemas de latencia, ampliando así las capacidades del Cloud más cerca de los dispositivos IoT.

En este contexto, a lo largo del presente proyecto se ha desarrollado, en primer lugar, una plataforma que permite la simulación de arquitecturas IoT-Fog-Cloud, centrándose en la flexibilidad a la hora de evaluar diversos escenarios. Además, se ha prestado especial atención a facilitar la configuración de diferentes soluciones de offloading en la plataforma. Otro de los objetivos principales que se han planteado ha sido desarrollar un algoritmo que tenga en cuenta ciertas métricas de interés y que, además, garantice la estabilidad del sistema. Para ello, se ha propuesto un modelo de sistema genérico, que asume patrones de tráfico variables y arbitrarios, capacidades de cálculo diferenciadas por nodo, y costes en la nube. Posteriormente, se ha formulado un problema de optimización estocástica, y se he hecho uso de la Teoría de Lyapunov para convertirlo en una secuencia temporal de problemas de Programación Lineal Entera, que pueden resolverse fácilmente con herramientas existentes.

El esquema propuesto se ha aplicado a un conjunto de escenarios de Fog-Cloud para validar su rendimiento bajo diferentes configuraciones. Los resultados demuestran que la solución propuesta es capaz de equilibrar el uso de instancias de Fog y Cloud para controlar el consumo de energía de acuerdo a un umbral configurable. También se pone de manifiesto que es posible ajustar fácilmente su comportamiento, para dar más o menos relevancia al coste monetario o a la limitación en el consumo de energía. Así, se ha analizado la respuesta de esta solución para equilibrar las colas de tráfico y el propio consumo energético. Los resultados obtenidos muestran que el algoritmo propuesto es capaz de adaptarse a flujos de tráfico no balanceados, garantizando la estabilidad del sistema, incluso bajo cargas altas de tráfico o restricciones de energía más estrictas. Posteriormente, se ha ampliado el análisis con una configuración más realista, poniendo de manifiesto que se puede configurar para operar con una respuesta deseada específica, por ejemplo, garantizando que el consumo de energía y el coste monetario en la nube sean similares.

Basado en los resultados y el análisis presentados en este TFM, se han generado dos artículos de investigación:

- N. Villegas et al. “Energy-aware optimum offloading strategies in fog-cloud architectures: A Lyapunov based scheme”. En: *IEEE Access* (2023). Pendiente de publicación

- N. Villegas et al. “Estrategias de offloading en arquitecturas Fog-Cloud: Un esquema basado en Lyapunov”. En: *Jornadas de Ingeniería Telemática (JITEL)* (2023). Pendiente de revisión

El artículo [52], enviado a IEEE Access, ha sido aceptado y está pendiente de publicación. Además, también se ha enviado una contribución [53] a las Jornadas de Ingeniería Telemática (JITEL), organizadas por la Sociedad Científica de Ingeniería Telemática (SCITEL), actualmente en proceso de revisión.

Tras la realización del proyecto surgen una serie de líneas futuras, tanto en relación a la plataforma Fog-Cloud, como a la ampliación del modelo de diferentes maneras.

- **Extensiones de la plataforma.** La plataforma se plantea desde un inicio con la idea de facilitar futuras ampliaciones en sus funcionalidades. En este sentido, algunas líneas de trabajo que se proponen afrontar son la mejora de los modelos de consumo de energía y del coste monetario, y en las comunicaciones entre los diferentes nodos. En la plataforma actual se lleva a cabo el cálculo del consumo únicamente a partir de la cantidad de bytes procesados en cada punto de procesado, junto a un factor de escala de la complejidad que supone el análisis de cada servicio concreto. Se asume, por tanto, que la transmisión de servicios a las instancias Cloud no conlleva un gasto de energía, lo cual podría no ser del todo preciso, especialmente con servicios de gran tamaño. Por otra parte, en relación al cálculo del coste monetario por el uso del Cloud, este se realiza mediante la definición de una variable que puede ser constante o aleatoria. Resultaría interesante la inclusión de un modelo de precios basado en cadenas de Markov y/o en el histórico de precios de servicios de Cloud comerciales. Finalmente, en cuanto a las comunicaciones, se contempla actualmente la posibilidad de usar *Traffic Control* para emular una red real, pero, sin embargo, este es un aspecto de la plataforma que puede ampliarse. Resultaría de interés, por ejemplo, el despliegue de una red de conmutación de paquetes entre las capas Fog y Cloud, o el uso de protocolos alternativos a TCP (por ejemplo QUIC[1]) que permitan abordar otros aspectos en el análisis de soluciones basadas en arquitecturas IoT-Fog-Cloud, haciendo hincapié en las penalizaciones que la transmisión de la información a través de la red podrían inducir a la hora de tomar decisiones de offloading.

- **Extensiones del modelo.** En primer lugar, se propone analizar el rendimiento al agregar funciones más complejas (por ejemplo, logarítmicas) al problema de optimización, para fomentar diferentes compensaciones entre las limitaciones de coste monetario y energía. Además, existe la posibilidad de incluir la ocupación de las colas de los procesadores dentro del modelo de sistema, de modo que evolucione hacia una red de colas interconectadas, en donde se podrían aplicar algoritmos de *back-pressure*. En este sentido, otra línea futura que subyace es la aplicabilidad del *back-pressure* basada en retardo, para fomentar una respuesta del sistema con una latencia muy baja. Se podría asimismo utilizar esta plataforma para comparar el rendimiento de este tipo de soluciones frente a estrategias que hagan uso de otras alternativas, como técnicas de Inteligencia Artificial o de Deep RL.

[1] `https://datatracker.ietf.org/doc/html/rfc9000`

Bibliografía

[1] A. K. Shaw, A. Chakraborty, D. Mohapatra y S. Dutta. "Scalable IoT Solution using Cloud Services – An Automobile Industry Use Case". En: *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. 2020, págs. 326-331. DOI: 10.1109/I-SMAC49090.2020.9243544.

[2] P. P. Jayaraman, D. Palmer, A. Zaslavsky y D. Georgakopoulos. "Do-it-Yourself Digital Agriculture applications with semantically enhanced IoT platform". En: *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. 2015, págs. 1-6. DOI: 10.1109/ISSNIP.2015.7106951.

[3] S. Xiao, H. Yu, Y. Wu, Z. Peng e Y. Zhang. "Self-Evolving Trading Strategy Integrating Internet of Things and Big Data". En: *IEEE Internet of Things Journal* 5.4 (2018), págs. 2518-2525. DOI: 10.1109/JIOT.2017.2764957.

[4] S. Kakati, K. Ray y R. Deka. "Cloud and Fog Computing based Industrial IoT Production Management". En: *2022 2nd International Conference on Intelligent Technologies (CONIT)*. 2022, págs. 1-5. DOI: 10.1109/CONIT55038.2022.9847879.

[5] B. Rana, Y. Singh y P. K. Singh. "A systematic survey on internet of things: Energy efficiency and interoperability perspective". En: *Transactions on Emerging Telecommunications Technologies* 32.8 (2021), e4166. DOI: 10.1002/ett.4166.

[6] K. Sumalatha y M. S. Anbarasi. "A review on various optimization techniques of resource provisioning in cloud computing". En: *International Journal of Electrical and Computer Engineering (IJECE)* 9.1 (2019), pág. 629. DOI: 10.11591/ijece.v9i1.pp629-634.

[7] A. Botta, W. de Donato, V. Persico y A. Pescapé. "On the Integration of Cloud Computing and Internet of Things". En: *2014 International Conference on Future Internet of Things and Cloud*. 2014, págs. 23-30. DOI: 10.1109/FiCloud.2014.14.

[8] P. Zikopoulos y C. Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. Google-Books-ID: 0sJqV1t4UVsC. McGraw Hill Professional, 2011. 176 págs.

[9] B. B. P. Rao, P. Saluia, N. Sharma, A. Mittal y S. V. Sharma. "Cloud computing for Internet of Things & sensing based applications". En: *2012 Sixth International Conference on Sensing Technology (ICST)*. ISSN: 2156-8073. 2012, págs. 374-380. DOI: 10.1109/ICSensT.2012.6461705.

[10] G. C. Fox, S. Kamburugamuve y R. D. Hartman. "Architecture and measured characteristics of a cloud based internet of things". En: *2012 International Conference on Collaboration Technologies and Systems (CTS)*. 2012, págs. 6-12. DOI: 10.1109/CTS.2012.6261020.

[11] P. Parwekar. “From Internet of Things towards cloud of things”. En: *2011 2nd International Conference on Computer and Communication Technology (ICCCT-2011)*. 2011, págs. 329-333. DOI: 10.1109/ICCCT.2011.6075156.

[12] M. Ishaq, M. H. Afzal, S. Tahir y K. Ullah. “A Compact Study of Recent Trends of Challenges and Opportunities in Integrating Internet of Things (IoT) and Cloud Computing”. En: *2021 International Conference on Computing, Electronic and Electrical Engineering (ICE Cube)*. 2021, págs. 1-4. DOI: 10.1109/ICECube53880.2021.9628191.

[13] A. Bonkra y P. Dhiman. “IoT Security Challenges in Cloud Environment”. En: *2021 2nd International Conference on Computational Methods in Science & Technology (ICCMST)*. 2021, págs. 30-34. DOI: 10.1109/ICCMST54943.2021.00018.

[14] Cisco, The Newsroom. *Cisco Delivers Vision of Fog Computing to Accelerate Value from Billions of Connected Devices*. URL: https://newsroom.cisco.com/c/r/newsroom/en/us/a/y2014/m01/cisco-delivers-vision-of-fog-computing-to-accelerate-value-from-billions-of-connected-devices.html (visitado 29-05-2023).

[15] F. Bonomi, R. Milito, J. Zhu y S. Addepalli. “Fog computing and its role in the internet of things”. En: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. SIGCOMM ’12: ACM SIGCOMM 2012 Conference. Helsinki Finland: ACM, 2012, págs. 13-16. DOI: 10.1145/2342509.2342513.

[16] S. Sarkar, S. Chatterjee y S. Misra. “Assessment of the Suitability of Fog Computing in the Context of Internet of Things”. En: *IEEE Transactions on Cloud Computing* 6.1 (2018), págs. 46-59. DOI: 10.1109/TCC.2015.2485206.

[17] L. M. Vaquero y L. Rodero-Merino. “Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing”. En: *ACM SIGCOMM Computer Communication Review* 44.5 (2014), págs. 27-32. DOI: 10.1145/2677046.2677052.

[18] M. Khaleel Hamadani y E. Borcoci. “Fog Nodes Placement for D2D Networks”. En: *2021 44th International Conference on Telecommunications and Signal Processing (TSP)*. 2021, págs. 152-156. DOI: 10.1109/TSP52935.2021.9522637.

[19] K. Zen, S. Mohanan, S. Tarmizi, N. Annuar y N. U. Sama. “Latency Analysis of Cloud Infrastructure for Time-Critical IoT Use Cases”. En: *2022 Applied Informatics International Conference (AiIC)*. 2022, págs. 111-116. DOI: 10.1109/AiIC54368.2022.9914601.

[20] S. Yi, Z. Hao, Z. Qin y Q. Li. “Fog Computing: Platform and Applications”. En: *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*. 2015, págs. 73-78. DOI: 10.1109/HotWeb.2015.22.

[21] E. Gomes, F. Costa, C. De Rolt, P. Plentz y M. Dantas. “A Survey from Real-Time to Near Real-Time Applications in Fog Computing Environments”. En: *Telecom* 2.4 (2021), págs. 489-517. DOI: 10.3390/telecom2040028.

[22] M. Malawski, G. Juve, E. Deelman y J. Nabrzyski. “Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds”. En: *Future Generation Computer Systems* 48 (2015). Special Section: Business and Industry Specific Cloud, págs. 1-18. DOI: https://doi.org/10.1016/j.future.2015.01.004.

[23] R. K. Naha, S. Garg y A. Chan. “Fog-computing architecture: survey and challenges”. En: *Big Data-Enabled Internet of Things*. Institution of Engineering y Technology, 2019, págs. 199-223. DOI: 10.1049/pbpc025e_ch10.

[24] S. S. Hajam y S. A. Sofi. “IoT-Fog architectures in smart city applications: A survey”. En: *China Communications* 18.11 (2021), págs. 117-140. DOI: 10.23919/JCC.2021.11.009.

[25] J. Yang. “Low-latency cloud-fog network architecture and its load balancing strategy for medical big data”. En: *Journal of Ambient Intelligence and Humanized Computing* (2020), págs. 1-10.

[26] K. Lone y S. A. Sofi. “A review on offloading in fog-based Internet of Things: Architecture, machine learning approaches, and open issues”. En: *High-Confidence Computing* 3.2 (2023), pág. 100124. DOI: 10.1016/j.hcc.2023.100124.

[27] M. Aazam, S. Zeadally y K. A. Harras. “Deploying Fog Computing in Industrial Internet of Things and Industry 4.0”. En: *IEEE Transactions on Industrial Informatics* 14.10 (2018), págs. 4674-4682. DOI: 10.1109/TII.2018.2855198.

[28] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow y P. A. Polakos. “A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges”. En: *IEEE Communications Surveys & Tutorials* 20.1 (2018), págs. 416-464. DOI: 10.1109/COMST.2017.2771153.

[29] P. Bellavista, L. Foschini y D. Scotece. “Converging Mobile Edge Computing, Fog Computing, and IoT Quality Requirements”. En: *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. 2017, págs. 313-320. DOI: 10.1109/FiCloud.2017.55.

[30] J. Sengupta, S. Ruj y S. D. Bit. “A Secure Fog-Based Architecture for Industrial Internet of Things and Industry 4.0”. En: *IEEE Transactions on Industrial Informatics* 17.4 (2021), págs. 2316-2324. DOI: 10.1109/TII.2020.2998105.

[31] D. A. Chekired, L. Khoukhi y H. T. Mouftah. “Industrial IoT Data Scheduling Based on Hierarchical Fog Computing: A Key for Enabling Smart Factory”. En: *IEEE Transactions on Industrial Informatics* 14.10 (2018), págs. 4590-4602. DOI: 10.1109/TII.2018.2843802.

[32] S. Garg, K. Kaur, G. Kaddoum y S. Guo. “SDN-NFV-Aided Edge-Cloud Interplay for 5G-Envisioned Energy Internet Ecosystem”. En: *IEEE Network* 35 (2021), págs. 356-364. DOI: 10.1109/MNET.011.1900602.

[33] I. Ahammad, M. A. R. Khan y Z. U. Salehin. “QoS Performance Enhancement Policy through Combining Fog and SDN”. En: *Simulation Modelling Practice and Theory* 109 (2021), pág. 102292. DOI: https://doi.org/10.1016/j.simpat.2021.102292.

[34] S. Chen, Y. Zheng, W. Lu, V. Varadarajan y K. Wang. “Energy-Optimal Dynamic Computation Offloading for Industrial IoT in Fog Computing”. En: *IEEE Transactions on Green Communications and Networking* 4.2 (2020), págs. 566-576. DOI: 10.1109/TGCN.2019.2960767.

[35] Y. Ren, Y. Sun y M. Peng. “Deep Reinforcement Learning Based Computation Offloading in Fog Enabled Industrial Internet of Things”. En: *IEEE Transactions on Industrial Informatics* 17.7 (2021), págs. 4978-4987. DOI: 10.1109/TII.2020.3021024.

[36] D. Iqbal y B. Buhnova. “Fog Based Energy Efficient Process Framework for Smart Building”. En: *Evaluation and Assessment in Software Engineering*. EASE 2021. Trondheim, Norway: Association for Computing Machinery, 2021, págs. 387-393. DOI: 10.1145/3463274.3463364.

[37] Y. Jie, C. Guo, K.-K. R. Choo, C. Z. Liu y M. Li. “Game-Theoretic Resource Allocation for Fog-Based Industrial Internet of Things Environment”. En: *IEEE Internet of Things Journal* 7.4 (2020), págs. 3041-3052. DOI: 10.1109/JIOT.2020.2964590.

[38] A. Aggarwal, N. Kumar, D. P. Vidyarthi y R. Buyya. “Fog-Integrated Cloud Architecture enabled multi-attribute combinatorial reverse auctioning framework”. En: *Simulation Modelling Practice and Theory* 109 (2021), pág. 102307. DOI: https://doi.org/10.1016/j.simpat.2021.102307.

[39] G. Peralta, P. Garrido, J. Bilbao, R. Agüero y P. M. Crespo. “Fog to cloud and network coded based architecture: Minimizing data download time for smart mobility”. En: *Simulation Modelling Practice and Theory* 101 (2020). Modeling and Simulation of Fog Computing, pág. 102034. DOI: https://doi.org/10.1016/j.simpat.2019.102034.

[40] H. Singh, S. Tyagi, P. Kumar, S. S. Gill y R. Buyya. “Metaheuristics for scheduling of heterogeneous tasks in cloud computing environments: Analysis, performance evaluation, and future directions”. En: *Simulation Modelling Practice and Theory* 111 (2021), pág. 102353. DOI: https://doi.org/10.1016/j.simpat.2021.102353.

[41] Y. Xiao y M. Krunz. “QoE and power efficiency tradeoff for fog computing networks with fog node cooperation”. En: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. 2017, págs. 1-9. DOI: 10.1109/INFOCOM.2017.8057196.

[42] X. Duan, F. Xu e Y. Sun. “Research on Offloading Strategy in Edge Computing of Internet of Things”. En: *2020 International Conference on Computer Network, Electronic and Automation (ICCNEA)*. 2020, págs. 206-210. DOI: 10.1109/ICCNEA50255.2020.00050.

[43] Y. Qi, L. Tian, Y. Zhou y J. Yuan. “Mobile Edge Computing-Assisted Admission Control in Vehicular Networks: The Convergence of Communication and Computation”. En: *IEEE Vehicular Technology Magazine* 14.1 (2019), págs. 37-44. DOI: 10.1109/MVT.2018.2883336.

[44] J. Xu, L. Chen y P. Zhou. “Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks”. En: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, págs. 207-215. DOI: 10.1109/INFOCOM.2018.8485977.

[45] X. Gao, X. Huang, S. Bian, Z. Shao e Y. Yang. “PORA: Predictive Offloading and Resource Allocation in Dynamic Fog Computing Systems”. En: *IEEE Internet of Things Journal* 7.1 (2020), págs. 72-87. DOI: 10.1109/JIOT.2019.2945066.

[46] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh y R. Buyya. “iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments”. En: *Software: Practice and Experience* 47.9 (2017), págs. 1275-1296. DOI: https://doi.org/10.1002/spe.2509.

[47] A. Kertész, T. Pflanzner y T. Gyimothy. “A Mobile IoT Device Simulator for IoT-Fog-Cloud Systems”. En: *Journal of Grid Computing* 17 (2019). DOI: 10.1007/s10723-018-9468-9.

[48] I. Lera, C. Guerrero y C. Juiz. “YAFS: A Simulator for IoT Scenarios in Fog Computing”. En: *IEEE Access* 7 (2019), págs. 91745-91758. DOI: 10.1109/ACCESS.2019.2927895.

[49] D. Cruz Trueba. “Arquitectura fog/cloud en el ámbito del IIot: modelado, despliegue y análisis”. Tesis de maestría. 2021.

[50] M. J. Neely. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Synthesis Lectures on Learning, Networks, and Algorithms. Cham: Springer International Publishing, 2010. DOI: 10.1007/978-3-031-79995-2.

[51] Z. Ren, W. Shi, J. Wan, F. Cao y J. Lin. “Realistic and Scalable Benchmarking Cloud File Systems: Practices and Lessons from AliCloud”. En: *IEEE Transactions on Parallel and Distributed Systems* 28.11 (2017), págs. 3272-3285. DOI: 10.1109/TPDS.2017.2715327.

[52] N. Villegas, L. Diez, I. de la Iglesia, M. González-Hierro y R. Agüero. “Energy-aware optimum offloading strategies in fog-cloud architectures: A Lyapunov based scheme”. En: *IEEE Access* (2023). Pendiente de publicación.

[53] N. Villegas, L. Diez, I. de la Iglesia, M. González-Hierro y R. Agüero. “Estrategias de offloading en arquitecturas Fog-Cloud: Un esquema basado en Lyapunov”. En: *Jornadas de Ingeniería Telemática (JITEL)* (2023). Pendiente de revisión.

A Anexo

```
FROM ubuntu:22.04
LABEL maintainer='Neco'

RUN apt-get update \
    && apt-get install -y python3 python3-pip \
    && rm -rf /var/lib/apt/lists/*

RUN pip install numpy

WORKDIR /home
COPY . .

ARG CONFIG
ARG NUM

CMD python3 nodoFog.py ${CONFIG} ${NUM}
```

Código A.1: Dockerfile del nodo Fog.

```
FROM ubuntu:22.04
LABEL maintainer='Neco'

RUN apt-get update \
    && apt-get install -y python3 python3-pip \
    && rm -rf /var/lib/apt/lists/*

RUN pip install numpy

WORKDIR /home
COPY . .

ARG CONFIG
ARG NUM

CMD python3 cloudRec.py ${CONFIG} ${NUM}
```

Código A.2: Dockerfile del nodo Cloud.

```
FROM ubuntu:22.04
```

```
LABEL maintainer='Neco'

RUN apt-get update \
    && apt-get install -y python3 python3-pip \
    && rm -rf /var/lib/apt/lists/*

RUN pip install numpy scipy==1.9.3

WORKDIR /home
COPY . .

ARG CONFIG

CMD python3 master.py ${CONFIG}
```

Código A.3: Dockerfile del nodo Master.

```
version: "3"
services:
  ########################## Nodo Master ##########################
  master_node:
    image: master_nodes
    privileged: true
    environment:
      - CONFIG=/home/config.json
    volumes:
      - ./:/home/
    networks:
      - master
  ########################## Nodos Fog ##########################
  fog_node_1:
    image: fog_nodes
    privileged: true
    environment:
      - CONFIG=/home/config.json
      - NUM=1
    volumes:
      - ./:/home/
    networks:
      - master
      - cloud
  ########################## Nodos Cloud ##########################
  cloud_node_1:
    image: cloud_nodes
    privileged: true
    environment:
      - CONFIG=/home/config.json
      - NUM=1
    volumes:
      - ./:/home/
    networks:
      - cloud
```

```
36
37   cloud_node_2:
38     image: cloud_nodes
39     privileged: true
40     environment:
41       - CONFIG=/home/config.json
42       - NUM=2
43     volumes:
44       - ./:/home/
45     networks:
46       - cloud
47   ################################################
48 networks:
49   master:
50   cloud:
```

Código A.4: Ejemplo de fichero YAML.

```
1  {
2  "simulation": {
3      "sim_time": 1500,
4      "slot_number": 1000,
5      "traf_gen": "trafficGen",
6      "traf_dist": "POISSON",
7      "serv_dist": "CONT",
8      "slot_time": 1,
9      "pkt_len_dist": "CONT",
10     "traf_rate": [3, 4, 5],
11     "pkt_len": 200,
12     "HEADERLENSIZE": 4,
13     "IDSIZE": 7,
14     "PKTSERLEN": 5,
15     "LENSEV": 4,
16     "serv_rate": 1,
17     "num_app": 3,
18     "v": 100,
19     "eth": 3.0
20 },
21 "master": {
22     "host": "master_node",
23     "port": 8080,
24 },
25 "fog_nodes": 1,
26 "fog1": {
27     "num_proc": 2,
28     "processor1": {
29         "capacity": 1000
30     },
31     "processor2": {
32         "capacity": 1000
33     },
34     "processor3": {
35         "capacity": 1000
```

```
    },
    "processor4": {
        "capacity": 1000
    }
},
"fog2": {
    "num_proc": 4,
    "processor1": {
        "capacity": 1000
    },
    "processor2": {
        "capacity": 1000
    },
    "processor3": {
        "capacity": 1000},
    "processor4": {
        "capacity": 1000
    }
},
"cloud_nodes": 1,
"cloud1": {
    "host": "cloud_node_1",
    "port": 8081,
    "cl_proc": 100000.0
},
"cloud2": {
    "host": "cloud_node_2",
    "port": 8082,
    "cl_proc": 5000
},
"logger": {
    "nodoFog": "ERROR",
    "trafficGen": "INFO",
    "processor": "ERROR",
    "service": "ERROR",
    "cloud": "ERROR",
    "master": "INFO"
}
}
```

Código A.5: Ejemplo de fichero de configuración.

```
{
   "service": {
    "dic_serv": {
      "1": {
          "1008": 4,
          "1010": 1,
          "1011": 3,
          "1012": 1,
          "1014": 2,
          "detailed": "none"
      },
```

```
      "2": {
          "2012": 3,
          "2013": 2,
          "2014": 2,
          "detailed": "none"
      },
      "3": {
          "3009": 4,
          "3010": 2,
          "3011": 3,
          "3014": 3,
          "detailed": "none"
      }
    },
    "num_slot": 14,
    "num_app": 3,
    "p_len": 200,
    "t_slot": 1
  },
  "clouds": [
    {
      "cloud1": {
          "proc_cap": 100000.0,
          "fog": false,
          "id": 1
      }
    }
  ],
  "fog": {
    "num_proc": 2,
    "buf_len": [11,7,12],
    "id": 1,
    "proc1": {
      "q_len": 0.0,
      "cola": 100000,
      "proc_cap": 1000,
      "processing": false
    },
    "proc2": {
      "q_len": 0.0,
      "cola": 100000,
      "proc_cap": 1000,
      "processing": false
    }
  }
}
```

Código A.6: Ejemplo de petición al nodo Master.

```
{
  "1": {
    "1008": "cloud::1::4",
    "1010": "cloud::1::1",
```

```
    "1011": "cloud::1::3",
    "1012": "cloud::1::1",
    "1014": "cloud::1::2"
   },
   "2": {},
   "3": {
    "3009": "local::1::4",
    "3010": "local::1::1,local::2::1",
    "3011": "local::2::3",
    "3014": "local::2::1,cloud::1::2"
   }
}
```

Código A.7: Ejemplo de respuesta del nodo Master.