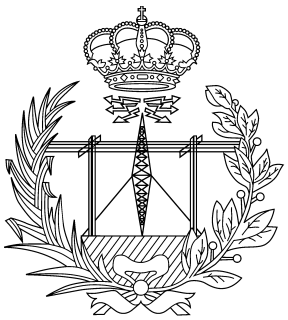
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Trabajo Fin de Grado

Estabilidad y retardo en sistemas de computación distribuida fog/cloud

(Stability and delay for fog/cloud distributed computing systems)

Para acceder al Título de

Graduado en
Ingeniería de Tecnologías de Telecomunicación

Autor: Carolina Teixeira Alberi

Septiembre -2023

UNIVERSIDAD DE CANTABRIA

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Carolina Teixeira Alberi
Director del TFG: Luis Francisco Diez Fernández, Ramón Agüero Calvo
Título: "Estabilidad y retardo en sistemas de computación distribuida fog/cloud"
Title: "Stability and delay for fog/cloud distributed computing systems"

Presentado a examen el día: 07 de Septiembre de 2023

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del tribunal:
Presidente (Apellidos, Nombre): Marta Domingo Gracia
Secretario (Apellidos, Nombre): José Ángel Miguel Díaz
Vocal (Apellidos, Nombre): María del Carmen Martínez Fernández

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

Fdo.: El Presidente	Fdo.: El Secretario
Fdo.: El Vocal	Fdo.: El Director del TFG (solo si es distinto del Secretario)
V^{o} B^{o} del Subdirector	Trabajo Fin de Grado N^{o} (a asignar por Secretaría)

Agradecimientos

Quiero agradecer sinceramente a mis directores de TFG, Luis Francisco Diez Fernández y Ramón Agüero Calvo, por su valiosa orientación y apoyo a lo largo de este proyecto. También quiero expresar mi profundo agradecimiento a mi familia y amigos por su constante aliento y respaldo en cada etapa de este viaje académico. Su apoyo ha sido fundamental en mi crecimiento y éxito.

Resumen

Este proyecto tiene como objetivo principal el diseño e implementación de algoritmos para distribuir servicios de cómputo de manera óptima en una arquitectura de tres niveles: IoT-Fog-Cloud. Tanto la plataforma como los algoritmos han sido desarrollados utilizando el lenguaje Python.

Los algoritmos propuestos en este trabajo buscan alcanzar un equilibrio en el sistema, considerando parámetros de mérito clave, como la ocupación de las colas y el retardo acumulado, con el fin de garantizar la estabilidad y un rendimiento óptimo. Las soluciones propuestas toman decisiones basadas en dichos parámetros, haciendo uso de la Teoría de Lyapunov, lo que contribuye a la mejora del comportamiento de la plataforma IoT-Fog-Cloud.

Este documento incluye una descripción detallada de la plataforma y de los algoritmos desarrollados, además de una serie de resultados, que permiten validar el correcto funcionamiento de las soluciones propuestas. Asimismo, se realiza una evaluación exhaustiva del rendimiento de los algoritmos propuestos, bajo diferentes configuraciones, demostrando su eficacia y versatilidad en diversos escenarios y casos de uso.

Palabras clave: IoT-Fog-Cloud; Teoría de Lyapunov; computación distribuida.

Abstract

This project aims to design and implement algorithms to optimally distribute computing services over a three-tier architecture: IoT-Fog-Cloud. Both the platform and the proposed algorithms have been developed using the Python programming language.

The proposed algorithms in this work seek to achieve a balance in the system, considering key performance indicators such as queue occupancy and delay, in order to ensure system stability and an optimal performance. The proposed solutions take decisions based on the aforementioned parameters, exploiting the Lyapunov Theory, thus contributing to the overall improvement of the IoT-Fog-Cloud platform.

This document includes a detailed description of the platform and the developed algorithms, followed by a number of results that allow validating the correct operation of the proposed solutions. Furthermore, a comprehensive evaluation of the algorithms' performance, under different configurations, is carried out, showing their effectiveness and versatility under various use cases and scenarios.

Keywords: IoT-Fog-Cloud; Lyapunov Theory; Distributed computing.

Índice general

Índice de figuras

Índice de tablas

Índice de acrónimos

AWS	Amazon Web Services.
CPUs	Central Processing Units.
FIFO	First In First Out.
IoT	Internet of Things.
QoS	Quality of Service.

Capítulo 1

Introducción

1.1. Motivación

El vertiginoso avance del Internet of Things (IoT) ha generado un aumento significativo en los servicios que hacen uso del mismo, así como de la generación de datos, lo que ha impulsado el uso creciente de la tecnología Cloud como solución para abordar la cada vez mayor demanda de recursos de procesamiento. Sin embargo, esta alternativa presenta una serie de desafíos, ya que las soluciones Cloud suelen ser sistemas centralizados, lo que puede resultar en recursos de procesamiento alejados de los servicios que los necesitan. Por otro lado, se ha presenciado, y de manera más evidente en los últimos años, un notable aumento en la capacidad de cómputo de dispositivos y redes. Este avance ha puesto en valor la importancia de enfoques alternativos, como el "Fog computing", que ofrecen soluciones para contrarrestar las desventajas de depender únicamente de la infraestructura en la nube de proveedores externos. Este nuevo enfoque, conocido como Fog computing, permite acercar ciertas capacidades de los servicios en la nube hacia los dispositivos.

Actualmente, los servicios en la nube ofrecidos por los principales proveedores (Microsoft Azure, Amazon Web Services (AWS) y Google Cloud) son ampliamente adoptados por diversos sectores empresariales, así como por particulares e instituciones, a nivel mundial. El despliegue progresivo de servicios IoT e Industrial IoT (IIoT), impulsado por el avance de las redes 5G, ha contribuido asimismo al incremento de la demanda de estos servicios, y se espera que esta tendencia continúe en los próximos años, dada la posibilidad que se abre, con la computación en la nube, para aprovechar varias de sus características principales [1].

Aunque ambas opciones, Cloud y Fog, pueden utilizarse de forma independiente, no es obligatorio optar exclusivamente por una de ellas, ya que ambas soluciones se complementan mutuamente. Esto ha llevado al desarrollo de arquitecturas de tres niveles (IoT-Fog-Cloud), que combinan las ventajas de ambos modelos y permiten reducir la latencia del servicio y mejorar además otros parámetros, como el consumo de energía. Estas arquitecturas también reducen la dependencia de los proveedores de servicios Cloud, lo que podría aportar una reducción de costes. Para maximizar estas ventajas, se requiere la implementación de un esquema de asignación de carga de trabajo en un sistema de cooperación IoT-Fog-Cloud, que garantice un desempeño óptimo, adaptado a una variedad de situaciones y requisitos diversos.

A pesar de las ventajas que presentan las arquitecturas IoT-Fog-Cloud, su gestión también presenta

desafíos. Aspectos como la cantidad de carga de trabajo a trasladar desde el Fog hasta las instancias en el Cloud, y la ubicación donde se toma esta decisión son cuestiones cruciales que deben ser abordadas para lograr un rendimiento óptimo y una colaboración efectiva en el sistema IoT-Fog-Cloud [2].

En este trabajo, se abordarán estos desafíos, presentando soluciones para optimizar la asignación de los trabajos de cómputo, y mejorar así el rendimiento general de las arquitecturas IoT-Fog-Cloud. Uno de los pilares fundamentales de este proceso radica en el desarrollo de algoritmos inteligentes y eficientes. Estos algoritmos, diseñados en base a un conocimiento completo de las características de la carga y las dinámicas de la red, juegan un papel crucial en la gestión dinámica de los recursos disponibles en la infraestructura IoT-Fog-Cloud.

El desarrollo de estos algoritmos implica un enfoque multidisciplinar, que combina conceptos de optimización, gestión de tráfico y sistemas de computación. Los algoritmos se plantean con el objetivo de lograr un equilibrio óptimo entre la asignación de tareas y los recursos disponibles en cada nivel de la arquitectura. Además, tienen en cuenta las limitaciones y restricciones específicas, la capacidad de procesamiento de los nodos Fog y la elasticidad de los recursos en la nube.

1.2. Objetivos

Considerando lo indicado anteriormente, se han identificado los siguientes objetivos específicos para este trabajo:

- Análisis detallado de las tecnologías de Cloud computing y Fog computing, con el propósito de comprender sus arquitecturas y aplicaciones en diferentes escenarios.
- Análisis de la arquitectura de tres niveles IoT-Fog-Cloud.
- Comprensión de la plataforma de emulación que se utiliza para llevar a cabo la evaluación de las soluciones planteadas.
- Se proponen dos algoritmos para la asignación de los servicios de cómputo, que tienen en cuenta la ocupación de las colas de los nodos, así como el retardo acumulado, en un entorno aleatorio y no controlado.
- Análisis del comportamiento de los algoritmos propuestos bajo diferentes situaciones y con requisitos diversos.
- Presentación de una serie de conclusiones finales, derivadas de los resultados obtenidos, y que se relacionan con los aspectos discutidos en el proyecto.

1.3. Estructura

Para abordar los objetivos planteados anteriormente, el documento se ha estructurado de la siguiente manera:

Capítulo 1: Introducción El primer capítulo establece el marco en el que se lleva a cabo este trabajo, delineando sus objetivos y presentando su estructura.

Capítulo 2: Contexto tecnológico Se lleva a cabo un análisis exhaustivo de la literatura relacionada con trabajo. Se presentan, de manera introductoria, los conceptos más relevantes de Fog y Cloud computing, así como la convergencia de estos dos paradigmas para la formulación y desarrollo de arquitecturas Fog-Cloud destinadas a entornos del Internet de las Cosas (IoT).

Capítulo 3: Plataforma de Emulación Se da paso a la presentación de la plataforma en la que se desarrollarán los experimentos para evaluar el comportamiento de las soluciones propuestas. Inicialmente, se destacan las funciones clave que incorpora, y se ofrece una visión global de la plataforma. Además, se describen los diferentes nodos que pueden desplegarse, así como sus respectivas funciones en el contexto de la plataforma.

Capítulo 4: Propuesta, Desarrollo e Implementación de Algoritmos Se describen los algoritmos propuestos, así como el modelo del sistema que se asume. Inicialmente, se desarrolla una solución que utiliza la ocupación de las colas como métrica. Posteriormente, se introduce un segundo algoritmo que considera el retardo acumulado en las colas como indicador de rendimiento.

Capítulo 5: Resultados Se lleva a cabo una extensa campaña de medidas en diversos escenarios, con el propósito de evaluar el rendimiento de los algoritmos desarrollados, primero en entornos Fog y, posteriormente, en escenarios Fog-Cloud.

Capítulo 6: Conclusiones Se presentan finalmente las conclusiones derivadas de la ejecución de este trabajo. Asimismo, se proponen potenciales líneas a abordar en investigaciones futuras.

Capítulo 2

Contexto Tecnológico

En general, el concepto IoT se refiere a un sistema interconectado de objetos inteligentes que cuentan con soporte computacional y se encuentran distribuidos en nuestro entorno. Estos dispositivos autónomos y adaptables permiten la creación de un sistema de computación ubicua y omnipresente, donde los objetos conectados interactúan sin necesidad de intervención humana directa.

El IoT facilita la recopilación de datos en tiempo real, lo que habilita la toma de decisiones fundamentada en información actualizada y precisa. Asimismo, permite la optimización de procesos, al identificar patrones y tendencias, lo que contribuye a mejorar la eficiencia y reducir costes en diversos ámbitos. En su conjunto, el IoT ha impulsado significativamente la transformación digital en diversas industrias, proporcionando soluciones innovadoras que mejoran la calidad de vida y abordan desafíos de manera más inteligente y efectiva [3].

En los siguientes apartados de este capítulo, se presentan los conceptos fundamentales de la arquitectura utilizada en este proyecto, haciendo especial énfasis en dos tecnologías clave: Cloud computing y Fog computing. Ambas son esenciales en el contexto del entorno IoT.

2.1. Cloud Computing

El Cloud computing hace referencia a la prestación de servicios a través de Internet, incluyendo servidores, bases de datos, redes, elementos software, análisis, almacenamiento e inteligencia, con el objetivo de ofrecer mayor rapidez, innovación y recursos flexibles. Este enfoque implica la virtualización de los recursos computacionales, permitiendo que los usuarios accedan a capacidades de almacenamiento y cómputo sin la necesidad de contar con una infraestructura propia.

El Cloud computing ha revolucionado la forma en que las empresas y los usuarios perciben y utilizan los recursos de las tecnologías de la información y comunicación, y su adopción ha aumentado significativamente en los últimos años. Entre las razones para recurrir a estos servicios se encuentran factores como su coste, rendimiento, velocidad y seguridad.

En el contexto del IoT, el Cloud computing ha demostrado aportar una mejora significativa, al proporcionar características de las que se carecía anteriormente. Entre estas ventajas destacan la escalabilidad, interoperabilidad, confiabilidad y flexibilidad. La combinación de IoT con Cloud computing ha permitido el flujo eficiente de datos, su procesamiento y recopilación, manteniendo el coste bajo control. Esto, a su

vez, ha facilitado la toma de decisiones basadas en datos, y en algoritmos para realizar predicciones [1].

Los principales elementos tecnológicos del Cloud computing son la comunicación, el almacenamiento y el cálculo. Así, la capacidad de la nube para ofrecer una infraestructura robusta y escalable ha sido fundamental para mejorar la funcionalidad del IoT, y hacer posible su implementación en diversos sectores y aplicaciones [4].

- **Comunicación**
  Los datos generados por dispositivos IoT son prácticamente ilimitados. Por ello, se requiere una conectividad a Internet de gran capacidad para transferir dichos datos a la nube. El Cloud supone una solución económica y eficiente para recopilar y gestionar la información. Facilita la monitorización y el control de dispositivos remotos, así como la coordinación de sus comunicaciones. Además, permite el acceso en tiempo real a los datos generados por estos dispositivos.

- **Almacenamiento**
  El IoT es una fuente ilimitada de datos que brinda oportunidades para agregar, compartir y procesarlos. Por lo tanto, necesita un almacenamiento adecuada. En este contexto, el almacenamiento en nube ofrece capacidades de almacenamiento virtuales ilimitadas, económicas y bajo demanda. La generación de grandes cantidades de datos requiere contar con una infraestructura que permita almacenar y gestionar de manera efectiva toda esta información. Al proporcionar recursos de almacenamiento virtual, el almacenamiento en nube se presenta como la solución ideal, lo que significa que las empresas pueden disponer de la cantidad de espacio que necesiten sin preocuparse por comprar hardware físico adicional. Dado que solo se paga por el almacenamiento utilizado, se trata de una opción más económica y escalable. Además, la seguridad es crucial para el manejo de datos del IoT. Para proteger los datos almacenados, la nube ofrece una amplia gama de técnicas de seguridad avanzadas, como encriptación, autenticación y acceso controlado. Estas características brindan un entorno seguro para almacenar datos sensibles y críticos, lo que tranquiliza tanto a las empresas como a los usuarios que confían en plataformas de computación en la nube para gestionar sus datos de Internet de las cosas.

- **Computación**
  Los dispositivos IoT tienen altas capacidades de procesamiento. Esto permite el análisis de datos en tiempo real, así como la implementación de aplicaciones escalables y colaborativas, lo que a su vez habilita la gestión eficaz de eventos complejos. En una variedad de industrias y campos de aplicación, esta combinación de tecnologías ha impulsado la innovación, y ha facilitado la aparición de soluciones avanzadas.

Otras razones por las cuales se puede considerar el Cloud computing como un elemento fundamental en el éxito del IoT se indican a continuación [5]:

- **Procesamiento**
  Los dispositivos IoT suelen disponer de pocas capacidades de cómputo. Sin embargo, Cloud computing ofrece un modelo bajo demanda y una gran capacidad de procesamiento. Esto permite satisfacer adecuadamente las necesidades del IoT, porque la computación pesada y compleja puede ser transferida a la nube, liberando a los dispositivos para que se concentren en tareas más específicas y ligeras.

El Cloud computing también permite una distribución equitativa de la carga de trabajo, y puede adaptarse dinámicamente a las demandas cambiantes del sistema, lo que garantiza un rendimiento y eficiencia óptimos.

- **Escalabilidad**
  La escalabilidad en la nube permite aumentar fácilmente la capacidad de procesamiento y almacenamiento bajo demanda, pero los altos costes y el retardo en el procesamiento pueden hacerla inviable para ciertos casos de uso sensibles al tiempo de respuesta, lo que impulsa la búsqueda de soluciones alternativas para el procesamiento de datos en IoT.

En conclusión, la combinación de IoT y Cloud computing ha impulsado un cambio significativo en el ámbito tecnológico, permitiendo a las empresas y usuarios aprovechar al máximo los recursos de comunicación y cómputo, mejorar la toma de decisiones basadas en datos y adaptarse a las crecientes demandas de la era digital de manera más eficiente y efectiva.

2.2. Fog computing

El Fog computing es un enfoque novedoso en la computación distribuida que extiende las capacidades del Cloud computing hacia el borde de la red para permitir un acceso eficiente a datos, computación, redes y almacenamiento. Esta tecnología permite el desarrollo, en el borde de la red, de nuevos servicios que incluyen heterogeneidad, movilidad, conciencia de ubicación, alta escalabilidad, baja latencia y distribución geográfica. En términos generales, los objetivos fundamentales del paradigma del Fog computing son reducir la carga de datos y tráfico hacia los servidores en la nube, disminuir la latencia y mejorar la Quality of Service (QoS). Para lograrlo, el Fog computing descentraliza el procesamiento y almacenamiento de datos, lo que alivia la sobrecarga en los centros de datos remotos y permite una respuesta más rápida y eficiente en aplicaciones críticas en tiempo real.

Este enfoque resulta especialmente útil en el contexto del IoT, donde se manejan grandes cantidades de datos generados por dispositivos dispersos y conectados. Al mejorar la eficiencia y optimización del procesamiento de datos en el borde de la red, el Fog computing contribuye al éxito y la escalabilidad del ecosistema IoT [6].

Las redes inteligentes, los sistemas de atención médica, las redes inalámbricas de sensores y los hogares inteligentes son algunos de los lugares donde el IoT ha aumentado rápidamente la cantidad de dispositivos conectados. En 2016, se utilizaron 2.6 millones de dispositivos IoT, un aumento del 30 % con respecto al año anterior [7]. Como se hizo referencia previamente, el aumento de los dispositivos de Internet de las cosas provoca una explosión en la generación de datos. Las aplicaciones convencionales de IoT requieren respuestas en tiempo real, lo que no siempre es factible con la computación en la nube tradicional. Sin embargo, debido a sus limitaciones en la capacidad de cálculo, el ancho de banda, la batería y la capacidad de almacenamiento, los dispositivos IoT tienen problemas inherentes. Estas restricciones pueden tener un impacto en la experiencia del usuario y la calidad del servicio. El Fog computing se presenta como una solución prometedora en este contexto. El Fog computing permite procesar datos más cerca de su origen, lo que reduce la latencia y mejora la eficiencia en aplicaciones críticas en tiempo real, al extender la capacidad de computación y almacenamiento hacia el borde de la red. Los recursos pueden ajustarse

dinámicamente según las necesidades de cada momento, lo que aumenta la escalabilidad y la flexibilidad del Fog computing.

En conclusión, este método ha demostrado ser particularmente ventajoso en aplicaciones IoT, como redes de vehículos y sistemas de monitorización en el ámbito sanitario, donde la ubicación de los dispositivos y la necesidad de una respuesta rápida son esenciales.

En la Tabla 2.1, se enumeran algunos aspectos de los servicios del IoT y cómo pueden ser solventados, parcial o totalmente, por soluciones Fog computing [7].

Tabla 2.1: Soluciones que ofrece Fog computing para aspectos IoT.

Aspectos servicios IoT	**Soluciones Fog computing**
Ancho de banda	El Fog computing reduce el tráfico dirigido a la nube, optimizando el uso del ancho de banda y mejorando la eficiencia de transmisión de los datos.
Latencia	Al procesar los datos cerca de la red, la latencia se reduce, lo que es especialmente importante en aplicaciones en tiempo real.
Eficiencia energética	Los dispositivos IoT pueden ahorrar energía al no depender totalmente de la nube.
Escalabilidad	Las soluciones Fog computing permiten una mayor escalabilidad, pues descentralizan el procesamiento y almacenamiento de los datos.
Procesamiento tiempo real	Al procesar localmente, se satisfacen los requisitos de aplicaciones en tiempo real.
Seguridad	Se disminuye el riesgo de vulnerabilidades, al no enviar todos los datos a la nube.

2.3. Arquitectura Fog-Cloud

Las limitaciones de recursos en los dispositivos IoT hacen necesario que las aplicaciones se adapten a los enfoques de Fog computing y Cloud Computing. Como se ha mencionado previamente, el Fog computing mejora significativamente la latencia del servicio, al acercar los recursos de procesamiento y almacenamiento a los propios dispositivos. Por otro lado, el Cloud Computing proporciona una capacidad de procesamiento y almacenamiento considerablemente superior, lo que permite gestionar la enorme cantidad de datos generados por los dispositivos IoT. Aunque ambas opciones pueden utilizarse de manera independiente, su uso en conjunto no solo es viable, sino altamente beneficioso, ya que se complementan mutuamente, y su coordinación conlleva mejoras en las capacidades del sistema en términos de eficiencia energética, procesamiento y almacenamiento de datos. Esto lleva a un uso óptimo de los recursos, aprovechando las ventajas de ambas soluciones según resulte conveniente en cada situación [8].

Por ejemplo, en escenarios donde se requiere una baja latencia o una alta seguridad en el procesamiento de la información, se podría potenciar el procesado en los nodos Fog. De esta manera, los datos no necesitan atravesar extensas redes, evitando posibles vulnerabilidades en la seguridad y minimizando los retardos. Por otro lado, cuando existen flujos de información voluminosos que demandan una capacidad

de procesamiento superior, la opción adecuada sería enviarlos a los nodos Cloud, ya que en los nodos Fog esta capacidad podría estar limitada. La utilización estratégica de cada enfoque permite aprovechar al máximo las capacidades disponibles y optimizar el rendimiento general del sistema IoT.

En capítulos posteriores de este proyecto, se discutirán los diferentes algoritmos implementados para determinar qué nodo será el responsable de procesar la información enviada por los dispositivos IoT. La Figura 2.1 muestra un esquema de la arquitectura de tres capas Fog-Cloud para IoT que se toma como referencia. Se puede ver que la capa inferior está compuesta por los dispositivos IoT, encargados de generar la información a ser procesada. En la segunda capa, o capa intermedia, se sitúan los nodos Fog, ubicados cerca o incluso integrados en los propios elementos que componen la red, recibiendo el tráfico generado por los dispositivos IoT. Por último, en la tercera capa o capa superior, se sitúan los nodos Cloud, a mayor distancia de la red IoT.

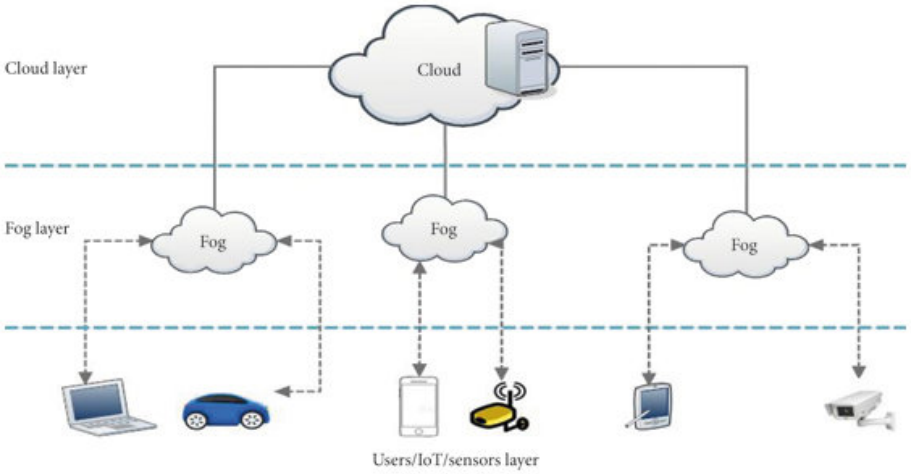


Figura 2.1: Arquitectura Fog-Cloud

En conclusión, la combinación de Fog computing y Cloud Computing en la arquitectura IoT ofrece una solución integral que aborda las limitaciones de recursos de los dispositivos, y proporciona una infraestructura eficiente, segura y ágil para procesar y gestionar los datos generados por los dispositivos IoT en un entorno cada vez más conectado y con más necesidades.

Capítulo 3

Plataforma de emulación

Una vez puesto en contexto este proyecto, explicados los principales términos y herramientas, en este capítulo se va a llevar a cabo la presentación de la plataforma de pruebas que se utilizará para desarrollar las soluciones propuestas en este trabajo. En [9] se puede encontrar la descripción de la implementación en detalle, en este capítulo se describirán las funcionalidades más relevantes de la plataforma que permitan explicar cómo se han desarrollado y validado los algoritmos propuestos en este trabajo. Se presenta la lógica de los diferentes nodos que participan en el sistema, así como su despliegue y funcionalidad. Esta plataforma de emulación desempeña un papel esencial en el marco de este proyecto, ya que posibilita la realización de verificaciones, desarrollos y evaluaciones necesarias de los algoritmos concebidos en un entorno realista y meticulosamente controlado.

Como se describe en [9], se han explorado diversas alternativas para la implementación y gestión de instancias Fog y Cloud, tanto en soluciones comerciales como de código abierto (por ejemplo, VmWare, OpenStack y otras). No obstante, estas tecnologías no fueron concebidas con el propósito de probar o evaluar soluciones de orquestación, sino más bien para administrar servicios en ambientes de producción. Esta característica puede dar lugar a limitaciones que no se alinean con los objetivos buscados en el diseño de la plataforma, o en este proyecto. Por esta razón, se desarrolló un entorno propio que satisface de manera más efectiva necesidades específicas. En la Figura 3.1 se observa un esquema general de la arquitectura del sistema.

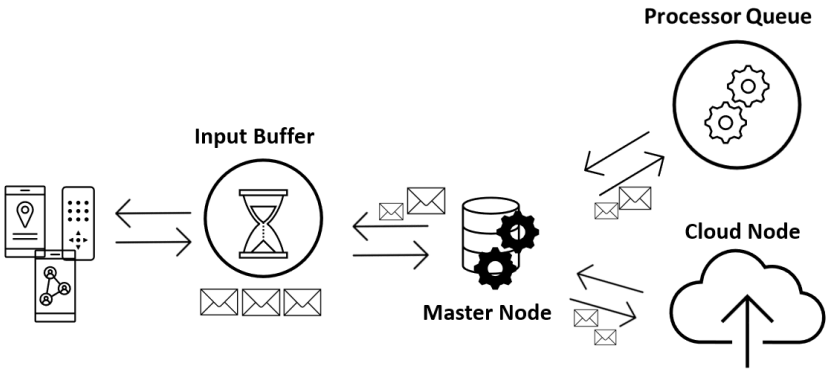


Figura 3.1: Arquitectura del sistema

La lógica de la plataforma está implementada en el lenguaje de programación Python e incluye tres tipos de elementos que emulan los nodos Fog y Cloud, además de un nodo Máster. Cada nodo está implementado como un programa Python que, a su vez, está dividido en hilos, los cuales se ejecutan de forma independiente y se encargan de emular cada una de las diferentes funciones del nodo al que pertenecen.

El enfoque modular de la plataforma hace que sea fácil personalizar y adaptarse a una variedad de escenarios de análisis. La configuración se simplifica mediante archivos JSON que contienen los parámetros necesarios para cada escenario. Para una representación más realista, los nodos Fog admiten configuraciones multiprocesador. Además, se incluye un módulo Python para la gestión de variables compartidas, y se permite el procesamiento paralelo de servicios mediante la fragmentación de estos. La plataforma también permite la definición de varias aplicaciones IoT, cada una con sus propias características, y es especialmente relevante, en el contexto de este trabajo, el hecho de que el algoritmo del nodo Máster sea intercambiable. Esta característica resulta crucial ya que facilita la comparación sistemática de diversas soluciones, lo que contribuye significativamente a alcanzar los objetivos planteados.

En la Figura 3.2 se representa la lógica de la plataforma. Ésta opera, de manera resumida, de la siguiente manera: En los nodos Fog se generan flujos de tráfico correspondientes a diferentes aplicaciones emuladas, con distribuciones aleatorias configurables. Estos flujos de tráfico son almacenados en un buffer (cola de la aplicación). Asimismo, en los nodos Fog, se generan servicios a partir de dicho buffer, con distribuciones aleatorias configurables de manera análoga a las aplicaciones. Una vez que los servicios han sido creados, entra en acción el nodo Máster, al cual los nodos Fog consultan respecto al procesamiento de la información. El nodo Máster toma una decisión fundamentada según el algoritmo correspondiente. Una vez que la decisión es recibida por el nodo Fog, este último procede a enviar los servicios a los procesadores locales o a la nube, donde serán finalmente procesados, de acuerdo con la decisión tomada por el nodo Máster.

A continuación se describe la funcionalidad de cada uno de los nodos que integran la plataforma.

3.1. Nodo Fog

El primer nodo a examinar es el Nodo Fog, el cual se encarga de recibir y procesar el tráfico proveniente de la red IoT a la que pertenece y que requiere un procesamiento. La lógica implementada en el Nodo Fog tiene como objetivo emular un servidor local. Para lograr esto, se emplean diversas funcionalidades de manera simultánea, utilizando el enfoque de multi-threading. Gracias a esto, todas las funcionalidades pueden operar en paralelo e independientemente, sin interferir entre sí. Estas funcionalidades pueden resumirse en tres aspectos principales: (1) generador de paquetes, (2) generador de servicios, y (3) procesadores. Cada uno de estos aspectos será abordado y explicado en detalle más adelante.

Cuando se inicializa la ejecución de este nodo, se leen los parámetros del archivo de configuración, y se establecen las conexiones a los nodos Cloud y Máster de la plataforma. Estas conexiones son utilizadas para el envío y recepción de las decisiones, parámetros necesarios y tráfico de datos. En el caso del nodo Cloud, la conexión se utilizará para el envío de la información de los nodos Cloud y la recepción de los servicios a procesar. En el caso del nodo Máster, la conexión se utilizará para el envío de consultas de procesado y la recepción de la decisión del lugar de procesamiento conforme al algoritmo utilizado.

En el nodo Fog (visible en la Figura 3.2), para cada aplicación existe un buffer con el tráfico procedente

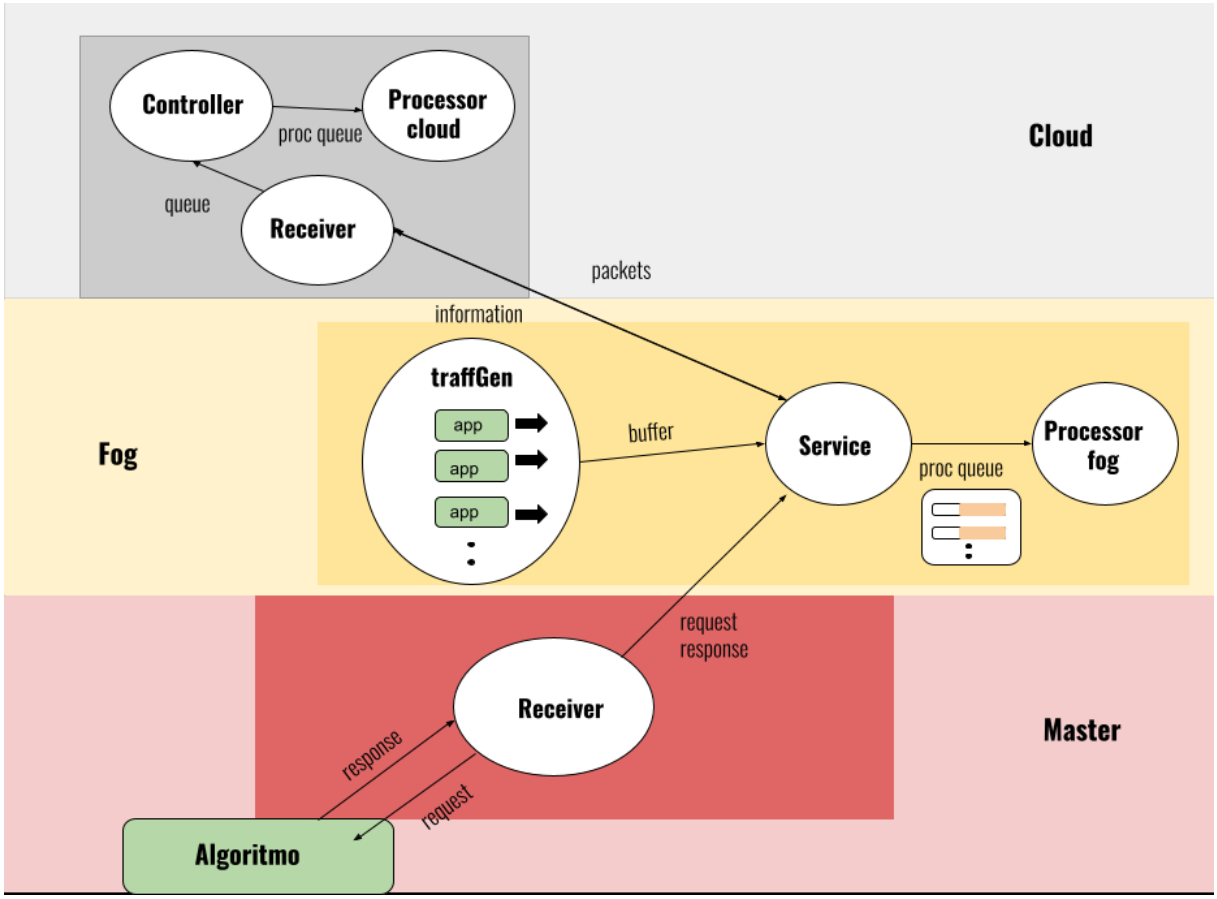


Figura 3.2: Plataforma de emulación

de las aplicaciones emuladas, una cola de servicios y una serie de procesadores, teniendo cada procesador una cola de procesamiento, que, junto al buffer de tráfico, son de longitud infinita para asegurar que no se produzcan pérdidas de paquetes. En esta plataforma, se asume que las colas siguen una política First In First Out (FIFO).

- **Generación de paquetes**
  Como se mencionó anteriormente, se emplea multi-threading, asignando un hilo de ejecución independiente a cada funcionalidad. En relación a esta funcionalidad, la plataforma emula la recepción de paquetes provenientes de los dispositivos IoT, categorizándolos en aplicaciones. En el nodo Fog, además, se implementa un thread por aplicación (TrafficGen). Asimismo, existe un buffer de entrada para cada aplicación, conforme se ha mencionado anteriormente. La recepción de paquetes se rige por una distribución aleatoria, configurable (al igual que su longitud) a través del archivo de configuración. En la Figura 3.3 se ilustra la estructura de un paquete generado en el nodo Fog.

  A continuación, se realiza una breve descripción de los elementos que componen un paquete.

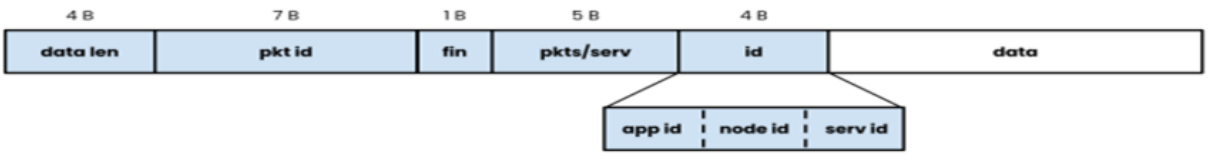


Figura 3.3: Paquete generado por TraffGen

- **data len** Es la longitud total en bytes de los datos del paquete
- **pkt id** Es el identificador único del paquete. Comienza en 1 y se incrementa paulatinamente, tras la generación de cada nuevo paquete.
- **fin** Flag que sirve para identificar el último paquete enviado a los nodos Cloud. Siempre vale 0, y se cambia a 1 para indicar el fin de la emulación.
- **pkts/serv** Hace referencia al número de paquetes totales pertenecientes al servicio. Solo se indica en el primer paquete de los servicios enviados.
- **serv id** Es el identificador único del servicio. Comienza en 1 y se incrementa con la generación de cada nuevo servicio.
- **app id** Identificador que indica la aplicación a la que pertenece el paquete.
- **node id** En el caso de disponer de diversos nodos Fog, indica el nodo al que pertenece el paquete.

Los valores de pkts/serv y serv id de establecen durante la generación de los servicios.

- **Generación de servicios**
  La generación de servicios para todas las aplicaciones se realiza en un único thread del nodo Fog, asumiendo que el tiempo de llegada de los servicios está ranurado en slots. Una vez que los paquetes se han generado, se almacenan en el buffer, para posteriormente definir un servicio como un conjunto de paquetes relacionados entre sí. En el archivo de configuración, explicado posteriormente, se fija una tasa de generación de servicios, siguiendo la distribución establecida.

  Al recibir un nuevo servicio, todos los paquetes actualmente almacenados en el buffer de la aplicación se asignan al mismo. Para consolidar los datos del nodo Fog, se genera un diccionario que contiene detalles sobre el estado de sus colas y procesadores. Este diccionario se combina con otros que contienen información de los nodos restantes. Posteriormente, este conjunto de datos se serializa y se envía al nodo máster. Una vez que el máster recibe los detalles de los nodos, procede a analizar la información disponible y a tomar una decisión con respecto a si los paquetes deben transferirse desde el buffer a la cola de procesamiento en el nodo Fog, o enviarse al nodo Cloud correspondiente, para su procesamiento remoto.

- **Procesador**
  La última funcionalidad en el nodo Fog es el procesador (Processor), responsable de llevar a cabo el procesamiento de los servicios que hayan sido enviados a la cola de los procesadores. En caso de que un servicio haya sido designado para ser procesado en el mismo nodo Fog, el hilo encargado de la generación de servicios dirigirá dicho servicio hacia la cola de procesamiento del procesador correspondiente.

  En el archivo de configuración, se fija la capacidad de procesamiento, que determinará cuántos bytes podrán ser procesados por slot.

  Para emular todos los tiempos de procesado y espera, se usan funciones de espera (*sleep*), con el tiempo requerido en cada caso, emulando el funcionamiento configurado en tiempo real. No se realiza ninguna operación de procesado real con los datos, que son aleatorios y sintéticos.

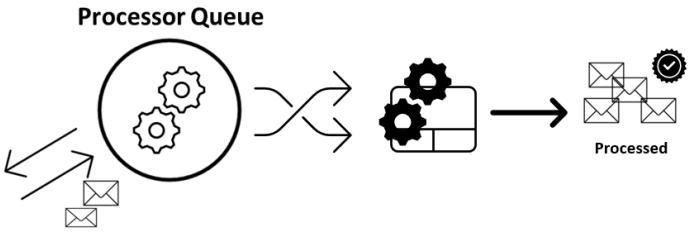


Figura 3.4: Procesamiento de servicios nodo Fog

3.2. Nodo Cloud

La lógica de este nodo es más simple que la de los nodos Fog y Máster, pues tiene una única funcionalidad, procesar los servicios cuyo procesado le hayan sido asignados.

En cada nodo Cloud, se incorpora una función de recepción implementada como un hilo independiente, que se mantiene en ejecución durante toda la emulación. Adicionalmente, hay un controlador encargado de gestionar el tráfico recibido y, por último, un procesador con una capacidad significativamente mayor que la de los procesadores presentes en los nodos Fog.

- **Receptor**
  Cuando un servicio es enviado al nodo Cloud, este llega al receptor, que comprueba la cabecera del primer paquete del servicio, analizando el campo pkts/serv explicado en **Generación de paquetes** para saber cuántos paquetes restantes pertenecen a dicho servicio. Para saber si el tiempo de emulación ha finalizado, también se chequea el campo de cabecera fin, pues si su valor es 1 significa que la emulación ha terminado.

- **Controlador y procesador**
  Posteriormente, el controlador se encargará de gestionar los servicios que se encuentren a la espera en la cola de entrada. Si se ha realizado una transferencia de datos completa, el hilo espera la llegada de más paquetes, mientras la emulación esté activa. En el caso indicado, el controlador enviará los paquetes a la cola de procesado, llegando posteriormente al procesador: primero se monitoriza continuamente el estado de la cola y, en caso de no estar vacía, se comprueba la cabecera del primer paquete, para conocer la longitud del servicio. Si se encuentra completo, se procesará. Se emularán los tiempos de procesado y espera de los paquetes de manera similar a como se comentó en el apartado anterior. A diferencia de los otros elementos, el nodo Cloud no genera tráfico, pero en su lugar, envía información a los nodos Fog acerca del estado de las colas, capacidad de procesamiento, etc. Posteriormente, éstos la envían junto con la suya propia cuando sea necesitada por el nodo máster.

3.3. Nodo Master

En todo el despliegue de la plataforma desarrollada, además de los nodos Fog y Cloud, se incorpora un nodo adicional denominado Master, que tiene como objetivo decidir en qué nodos y procesadores se realiza el procesado de los servicios. A continuación se describen las funcionalidades de este nodo.

- **Comunicación del Nodo Master con el Nodo Fog**
  Cada vez que un nodo Fog recibe un nuevo servicio para su procesamiento, notifica al nodo Master, el cual le proporcionará la información necesaria para determinar dónde llevar a cabo dicho procesamiento. Existe la posibilidad de dividirlo en varios slots, lo que permite al nodo Fog ser conocedor de que el Master no haya asignado todos los paquetes del servicio. En este caso, el nodo Fog conserva los paquetes restantes en un buffer, y solicita al Master procesarlos en un próximo slot.

  El Master recibe un Request por parte del Generador de Servicios del nodo Fog en cada slot. El Request contiene tanto los servicios generados en el slot actual como los servicios pendientes de procesar de slots anteriores. Además, incluye información sobre los nodos Cloud, los procesadores del nodo Fog y ciertos detalles del servicio. También se incluye la información de estado de los nodos Fog, la cual se corresponde, básicamente, con la ocupación de las colas de las aplicaciones y de los procesadores. Esta información puede ser o no utilizada en su totalidad, dependiendo del algoritmo implementado. Por ejemplo, un algoritmo basado en las colas de los procesadores no utilizará los mismos criterios (datos) que uno basado en costes. Se puede observar un ejemplo de Request en el Anexo A.

- **Comunicación Nodo Master y Algoritmo**
  Como se puede observar en el diagrama de la Figura 3.5, una vez que el nodo Master recibe el Request, lo procesa y, mediante la funcionalidad de recepción (ver Figura 3.2), envía el Request al Algoritmo especificado en el archivo de configuración (se integra como un complemento, lo que posibilita la comparación de diversas soluciones en los mismos escenarios). El Algoritmo devuelve al nodo Master una respuesta con la decisión de asignación, como se muestra en el Anexo A. Como en el Request, los servicios se organizan en diccionarios de dos niveles, donde se indica primero el identificador de la aplicación, seguido del correspondiente a los servicios asociados a la misma, y el lugar asignado para procesar esos servicios (en el caso de procesarse localmente, se indicaría el procesador asignado), así como el número de paquetes a procesar (en el caso de no ser indicado, se supone que se procesan todos los paquetes del servicio). Si el servicio se procesa en varios sitios, las asignaciones se separan por comas.

  Como previamente se ha indicado, una vez que el nodo Master recibe la solicitud, esta es reenviada al algoritmo elegido entre los dos implementados, cuyos detalles serán abordados en capítulos posteriores. Dependiendo del algoritmo escogido, se llevarán a cabo una serie de procesos. Una vez obtenida la decisión, esta se presenta en forma de diccionario (*dic*), y es enviada de vuelta al nodo Master, encargado de enviarlo finalmente al nodo Fog.

3.4. Configuración

Una de las principales ventajas de esta plataforma es su alta flexibilidad, gracias a su sencilla configuración, lo que permite realizar, de manera sencilla, pruebas sobre diferentes escenarios. La configuración del sistema se realiza a través de un archivo JSON. A continuación, se describe uno de los ejemplos utilizados en el Anexo A.

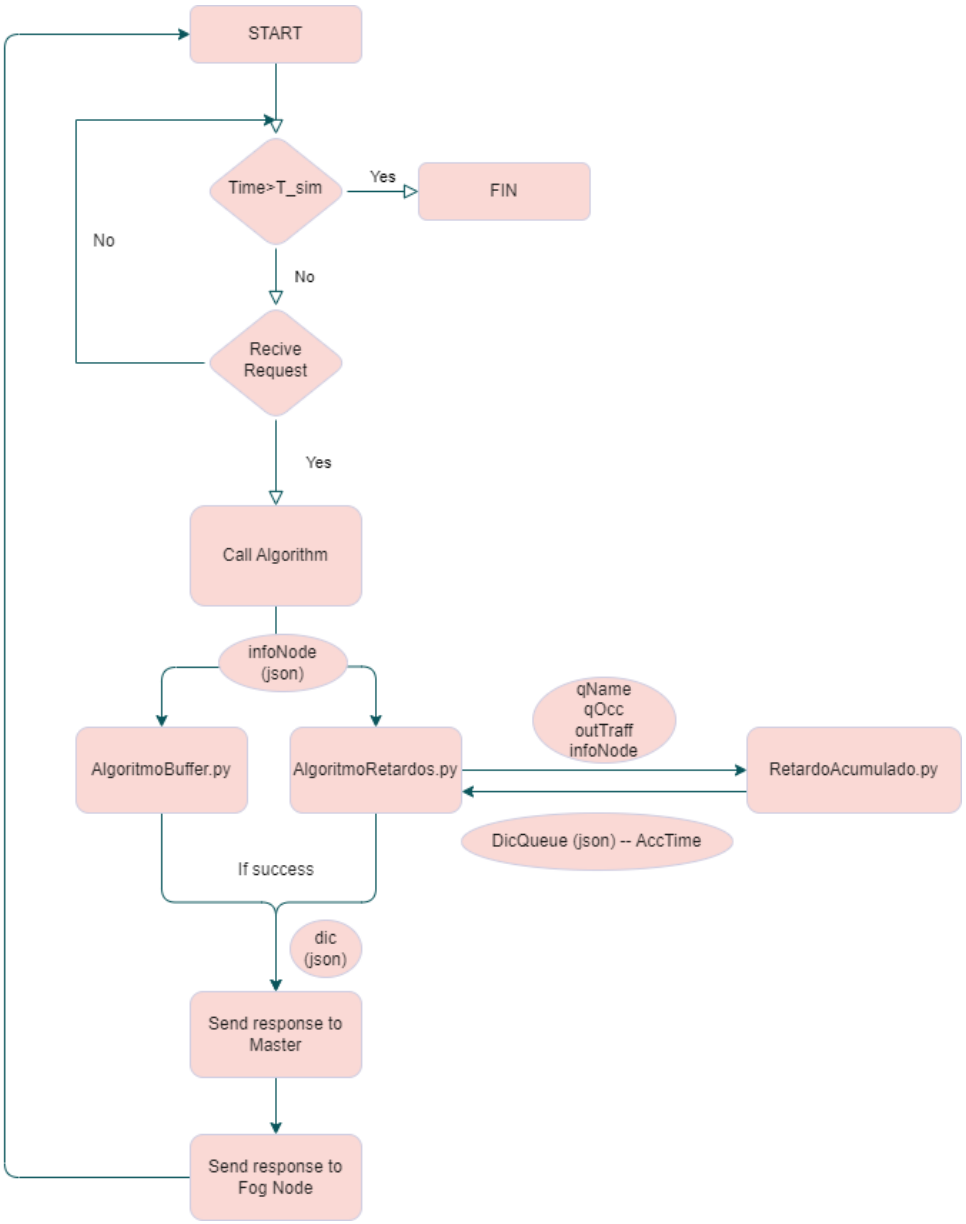


Figura 3.5: Diagrama de flujo comunicaciones nodo Master y Algoritmo

El archivo JSON cumple la función de configurar múltiples parámetros en la emulación. Estos parámetros se organizan en tres categorías principales: Nodo Fog, Nodo Máster y Nodo Cloud, junto con otros aspectos generales de la ejecución.

Para el Nodo Fog, se tiene la capacidad de controlar la tasa de llegadas de paquetes y servicios, definir el número de aplicaciones activas, ajustar la capacidad de los procesadores en el nodo, limitar la cantidad de paquetes que las aplicaciones pueden enviar a un procesador, y definir la distribución para la generación de paquetes y servicios.

En cuanto al Nodo Máster, se permite seleccionar el algoritmo para la toma de decisiones que, como se mencionó previamente, se integra como un complemento, permitiendo la comparación de diversas soluciones en escenarios idénticos.

Dentro del Nodo Cloud, es posible establecer la capacidad de procesamiento, y especificar la cantidad de

nodos Cloud disponibles.

Además, entre los parámetros globales, se pueden ajustar las longitudes de las cabeceras de las aplicaciones, definir la duración de la emulación, y determinar la distribución del tráfico, entre otros parámetros esenciales. En la Tabla 3.1 se presentan todos los parámetros que pueden ser configurados en la plataforma.

Esta capacidad de configuración permite adaptar la herramienta de análisis a diferentes escenarios y necesidades, lo que contribuye a un entorno de emulación altamente versátil y eficiente. El uso del archivo JSON como mecanismo de configuración proporciona una estructura clara y bien organizada para ajustar los parámetros y características de la emulación, facilitando así la experimentación y evaluación de diversos casos de estudio.

Tabla 3.1: Parámetros configurables de la plataforma

Categoría	**Parámetro configurable**	**Descripción**
Fog	$traf - rate$	Tasa de llegadas
	$serv - rate$	Tasa de llegadas
	$num - app$	Número de aplicaciones
	$num - Fog$	Número de nodos Fog
	$capacity$	Capacidad de procesado del procesador en pkts/s
	$limit$	Limitación en pkts/s que es capaz de recibir cada procesador por parte de las aplicaciones.
	K	Coste Cloud de procesar datos de las aplicaciones.
	$traf - dist$	Distribución del tráfico de una aplicación. Poisson o Cont*
	$serv - dist$	Distribución para la generación de servicios. Poisson o Cont*(slots)
Master	$algorithm$	Nombre del algoritmo en el nodo Master
Cloud	$cl - proc$	Capacidad Cloud medida en pkts/s
	$Cloud - nodes$	Número de nodos Cloud
General	$sim - time$	Tiempo de emulación en slots
	$IDSIZE$	Longitud en Bytes de la cabecera que contiene el identificador de pkt.
	$HEADERLENSIZE$	Longitud en Bytes de la cabecera que contiene la longitud de los datos en bytes.
	$PKTSERLEN$	Longitud en Bytes de la cabecera que contiene el número de paquetes por servicio.
	$LENSEV$	Longitud en Bytes de la cabecera que contiene el identificador del servicio
	$logger$	Niveles de prioridad de logging

3.5. Docker y LocalHost

Al explorar la mejor manera para llevar a cabo la ejecución del sistema, se presentan dos enfoques que requieren un análisis más detallado: la utilización de contenedores Docker o la alternativa de ejecución en entorno local (localhost). Ambas opciones conllevan ventajas e inconvenientes que deben valorarse. Si la

decisión se inclina hacia un método más próximo y controlado, la alternativa de localhost surge como una elección adecuada y de particular relevancia.

- **Despliegue en Docker: Modularidad y Escalabilidad**
  El empleo de contenedores Docker presenta la principal ventaja de aportar una gran modularidad y escalabilidad. Mediante la encapsulación de componentes en contenedores, se obtiene un entorno aislado y coherente, facilitando además su portabilidad. Docker se convierte en una herramienta esencial para la creación, distribución y administración de aplicaciones, reduciendo los posibles conflictos entre dependencias y versiones. Esta opción destaca en entornos distribuidos, permitiendo replicar el despliegue de manera precisa en distintos sistemas, y proporcionando una solución uniforme para diversos escenarios. La utilización de contenedores Docker también simplifica el proceso de implementación al permitir definir recursos, configuraciones y redes de manera sencilla [10].

- **Despliegue en Localhost: Control y Personalización**
  La elección de ejecutar las emulaciones en localhost se traduce en un mayor nivel de control sobre la configuración y el entorno de ejecución. Al optar por esta estrategia, aparece la capacidad de ajustar parámetros y configuraciones específicas de manera directa en la máquina local. Esto ofrece un despliegue más inmediato y un proceso de configuración rápido y eficiente.

  Muestra su máximo potencial en las etapas iniciales de desarrollo, pruebas y evaluación. En estos momentos, la simplicidad y la agilidad del proceso adquieren una gran relevancia, permitiendo iteraciones ágiles y ajustes constantes. La cercanía al entorno local también brinda una ventaja significativa al facilitar la depuración y la detección de posibles problemas, contribuyendo a la creación de un sistema sólido y confiable.

  Además, la ejecución en localhost ofrece la oportunidad de aprovechar plenamente los recursos del equipo local, lo cual resulta altamente beneficioso en términos de rendimiento y capacidad de procesamiento. Esta opción se presenta como la elección óptima para llevar a cabo una evaluación del sistema en la fase de desarrollo, ya que permite permite un mayor control y un análisis exhaustivo del comportamiento.

  Es importante destacar que los análisis de este trabajo se han obtenido mediante ejecuciones en localhost. Esta elección responde a la búsqueda de un mayor control y capacidad de depuración. En cualquier caso, los algoritmos implementados podrían ser utilizados igualmente en ejecuciones mediante contenedores sin requerir ninguna modificación.

Capítulo 4

Propuesta, Desarollo e Implementación de Algoritmos

A lo largo de este trabajo se ha asumido que el Nodo Master ostenta la responsabilidad de ejercer la toma de decisiones sobre qué elemento (Fog o Cloud) será el encargado de procesar la información. En ese sentido, la tarea de decidir se lleva a cabo por un algoritmo decisor, que se encargará de seleccionar el lugar más adecuado para realizar el procesamiento. En este capítulo se presentarán los algoritmos desarrollados en el marco de este trabajo.

4.1. Modelado del Sistema

En esta sección, se presenta el modelo del sistema utilizado, así como el diseño de la política de control implementada, que tiene como base la teoría de control de Lyapunov a través del marco teórico desarrollado por Neely [11].

Tabla 4.1: Definiciones y símbolos

Símbolo	Definición
	Variables aleatorias
$a_m(t)$	Llegadas a la cola de la aplicación m o puntos de procesamiento n en el slot t (bytes)
$b_m(t)$	Salidas de la cola de la aplicación m en el slot t (bytes)
$w_n(t)$	Capacidad de procesado en cada CPU en el slot t (bytes/slot)
$Q_m(t)$	Valor de la cola de la aplicación o punto de procesamiento m o n en el slot t
N	Número de puntos de procesamiento (CPUs) en el Fog e instancias Cloud
M	Número de aplicaciones proveedoras de servicios.
$g_i(t)$	Complejidad de procesado de los servicios de la aplicación i (m) en el slot t.
$\alpha_{i,j}(t)$	Decisión para la aplicación i (m) y punto de procesamiento j (n)(pkts).
$W_{i,j}$	Diferencia entre la métrica de la cola de la aplicación i (m) y el punto de procesamiento j (n).

En el contexto previamente descrito se ha establecido un sistema que consta de tres clases de nodos: Fog, Cloud y Master. El nodo Master asume el rol de "decisor", dependiendo del algoritmo implementado, mientras que los nodos Fog y Cloud actúan como nodos encargados de procesar la información. En este sistema, se contemplan aplicaciones emuladas que generan paquetes, que conformarán servicios, que serán finalmente enviados a los nodos Fog o Cloud para ser procesados.

La Tabla 4.1 proporciona un resumen de los símbolos empleados en el modelo propuesto, junto con sus respectivas definiciones.

En este escenario se contempla un sistema con M aplicaciones que son responsables de generar los servicios a procesar. Cada aplicación genera un servicio en cada intervalo de tiempo (slot) con una cantidad de tráfico (bytes o paquetes) que sigue una distribución aleatoria arbitraria. En este escenario inicial, no se incluye un nodo en la nube para el procesamiento. En cambio, se cuenta con un conjunto de N Central Processing Units (CPUs), que se corresponden con los procesadores de un nodo Fog. Como se ha mencionado anteriormente, los nodos Fog poseen colas de tráfico y colas de procesado (estas están tanto en los Fog como en los Cloud). En cada intervalo de tiempo (slot), el nodo Master determina, a través del algoritmo elaborado y seleccionado, la cantidad de tráfico que se traslada de cada cola de las aplicaciones a cada cola de procesado. Esto se hace con el propósito de asegurar la estabilidad de las colas, tanto para las aplicaciones como para los procesadores.

La Figura 4.1 ilustra un ejemplo del modelo de este sistema con tres aplicaciones ($M = 3$) y dos procesadores (Fog) ($N = 2$).

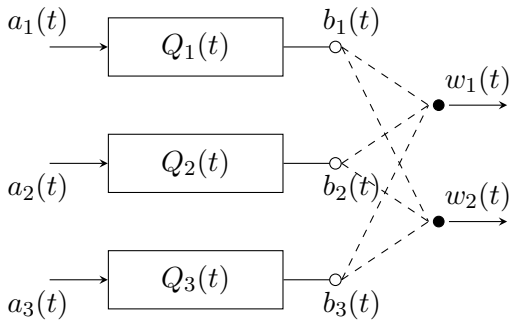


Figura 4.1: Modelo del sistema con 3 aplicaciones y 2 CPUs

Los paquetes de los servicios generados por las aplicaciones, $a_k(t)$, pasan a ser almacenados en el buffer/cola de las aplicaciones, localmente, $Q_k(t)$ en el slot t. Además, $b_k(t)$ corresponde al tráfico que sale desde el buffer en el slot t. En cada slot, las colas se actualizan según la Ec. (4.1), siendo $Q_k(t)$ el tamaño de la cola, es decir su ocupación, en cada slot t:

$$Q_k(t+1) = \text{máx}[Q_k(t) - b_k(t), 0] + a_k(t) \tag{4.1}$$

El objetivo principal es asegurar la estabilidad promedio, para lo que se hace uso de la métrica *Mean Rate Stability* que se define como sigue:

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

siendo $Q_k(t)$ el tamaño de la cola en el slot t, y $\mathbb{E}$ la esperanza matemática. La Ec. (4.2) indica que, a medida que observamos la variable aleatoria $Q_k(t)$ durante un período de tiempo cada vez más largo (T tiende a infinito), el promedio temporal de $Q_k(t)$ disminuirá hasta acercarse a cero. En otras palabras, esto sugiere que la media de $Q_k(t)$ se hace más pequeña a medida que consideramos intervalos de tiempo más largos.

El marco teórico que se utiliza en la elaboración de los algoritmos asegura la *Mean Rate stability* en cualquier caso. Como métrica más estricta de estabilidad se define en la Eq. (4.3) la *Strong Stability*. Si bien el entorno teórico adoptado en este trabajo no asegura la *Strong Stability*, en la práctica se suele conseguir.

$$\lim_{T\to\infty} \frac{1}{T} \sum_{\tau=1}^{\tau=T} \mathbb{E}\{Q_k(\tau)\} \leq \infty \tag{4.3}$$

Como se puede observar en la Eq. (4.3), esta nueva métrica de estabilidad se cumple si la media temporal de la suma acumulada de la ocupación de las colas se encuentra por debajo debajo de un umbral finito. En otras palabras se podría decir que el acumulado de todo el tráfico almacenado en las colas no crece "mucho más rápido" que el tiempo.

En cada slot, se toma una decisión $\alpha(t)$ dentro de un conjunto de posibilidades. La decisión $\alpha(t)$ tiene estructura de matriz $M \times N$, de modo que el elemento $\alpha_{m,n}(t)$ se corresponde con la decisión tomada para la aplicación m, que se asignaría al procesador n en el slot t. Se puede definir entonces $b_m(t)$ como se muestra en la Eq.(4.4), siendo $b_m(t)$ la salida de la cola de la aplicación m en el slot t y $\hat{b}$ identifica una función cualquiera. Se define $w(t)$ como la tasa de máxima de salida en bytes por slot t. Se asume que las tasas de salida siguen ditribuciones aleatorias arbitrarias sobre las que no se tiene ningún control. Como se muestra en la Eq.(4.4), la cantidad de datos enviados desde la aplicación m es función de la decisión $\alpha(t)$ y de la tasa de transferencia de los procesadores $w(t)$. En el caso de procesamiento local en el nodo Fog, $w(t)$ puede verse influenciada por la capacidad de lectura/escritura (I/O) en el propio nodo debido a la interacción de otros procesos, y por la capacidad de comunicación en el caso de procesarse remotamente por el nodo Cloud.

$$b_m(t) = \hat{b}(\alpha(t), w_1(t), w_2(t), \dots) = \hat{b}(\alpha(t), \omega(t)) \tag{4.4}$$

Por simplicidad en la implementación, para evitar que se asigne más tráfico del existente en una cola, se fija una restricción, de tal manera que la decisión no supere la cantidad de bytes disponibles en la cola de la aplicación, Ec. (4.5).

$$\sum_{j=1}^{N} \alpha_{ij}(t) \leq Q_i(t) \quad \forall i \in \{1, \dots, M\} \tag{4.5}$$

También se añade una restricción para no exceder la capacidad de envío o de transferencia, donde g_i es un factor de escala de la complejidad computacional de la aplicación (4.6).

$$\sum_{i=1}^{M} g_i \cdot \alpha_{ij}(t) \leq w_j(t) \quad \forall t, \forall j \tag{4.6}$$

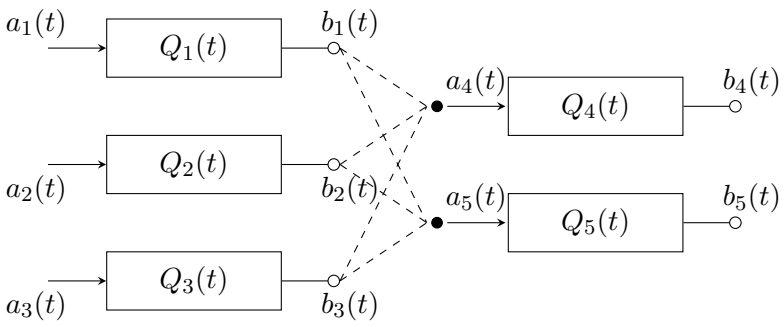


Figura 4.2: Ejemplo modelo del sistema con 3 aplicaciones y 2 CPUs $Q(t)$

De esta manera, las salidas de la cola $b_m(t)$ de cada aplicación se podrían expresar:

$$b_m(t) = \hat{b}(\alpha(t), \omega(t)) = \sum_{j=1}^{N} \alpha_{m,j}(t) \tag{4.7}$$

De (4.7) resulta $b_m(t)$, que se corresponde con la salida de la cola de la aplicación m, como el sumatorio de todas las decisiones que van de la aplicación m a todo el rango de procesadores $\forall n \in \{1, \ldots, N\}$.

4.2. Algoritmo 1: backpressure

El primer algoritmo implementado tiene como objetivo estabilizar de forma conjunta las colas de las aplicaciones y los procesadores, y hace uso directo de desarrollo en [11]. En concreto se trata de un algoritmo de *backpressure* sobre un sistema de colas conectadas.

Se considera un modelo del sistema con M aplicaciones y N procesadores, como se muestra en la Figura 4.2, donde se considera un sistema con colas infinitas en las CPU y un único nodo Fog.

En cada slot el algoritmo observa las colas de las aplicaciones y los procesadores y selecciona los valores de $\alpha(t)$ que maximice la Ec. (4.8).

$$\max_{\alpha(t)} \quad \sum_{i,j} b_{i,j}(t) \cdot W_{ij}(t) \tag{4.8}$$

En este primer enfoque se utiliza la ocupación de los buffers como métrica de la cola. De esta manera, $Q_i(t)$ es la ocupación de la cola de la aplicación i en el slot t, y $Q_j(t)$ la ocupación de la cola del procesador j en el slot t. Además se definen las variables $W_{ij}(t)$ como sigue:

$$W_{ij}(t) = Q_i(t) - Q_j(t) \tag{4.9}$$

De forma intuitiva, el algoritmo dará prioridad a aquellas asignaciones que con mayor diferencia entre las ocupaciones de la aplicación y el procesador. Si la aplicación está menos cargada que el procesador esa asignación contribuye de forma negativa a la función objetivo (Ec. 4.8), por lo que no se asignará. Por el contrario, si la aplicación está mucho más cargada que el procesador, esa asignación contribuirá mucho a la función objetivo.

Como se indicó en anteriores capítulos el nodo Fog envía, en cada slot, un Request al Master, con información relativa al estado de los diferentes elementos del sistema. En este caso, por cada slot t, el Nodo Master proporciona al Algoritmo la información recibida por el nodo Fog en un diccionario de dos niveles, haciendo uso el algoritmo de la información relativa tanto a los servicios como a la ocupación de las colas de las aplicaciones (`buf_len`) y de los procesadores (`q_len`) en ese intervalo temporal. En el Anexo A se muestra un ejemplo de la estructura del Request.

4.3. Algoritmo 2: delay-based backpressure

En este segundo enfoque también se considera un sistema con M aplicaciones y N procesadores, como se muestra en la Figura 4.2, en el que se asume además colas infinitas en las CPU y un único nodo Fog.

A diferencia del primer algoritmo, la decisión se basa en este caso en el retardo acumulado en los buffers como métrica de las colas. Este enfoque se basa en los trabajos [12, 13] en los que se demuestra que es posible mantener la estabilidad de las colas utilizando el tiempo que lleva esperando el tráfico en una cola (retardo acumulado) como métrica de ocupación. Además el uso de esta métrica tiene ventajas respecto a la ocupación.

En el caso de usar la ocupación, si una aplicación genera poco tráfico en comparación con otras la ocupación de su cola correspondiente crece muy lentamente. En este caso, se dejará crecer la cola de aplicación lenta antes asignar un procesador, ya que las colas de las otras aplicaciones obtendrán prioridad. Este comportamiento provoca que el retardo inducido en el tráfico de la aplicación lenta sea mucho mayor, lo que no sería justo. En cambio, haciendo uso del retardo acumulado como métrica de la cola este efecto se reduce y el algoritmo tenderá a igualar los retardos inducidos a todas las aplicaciones.

Cabe destacar que el problema a optimizar sigue siendo el planteado en la Ec. (4.8), pero el significado de las colas cambia. Así, $Q_i(t)$ se define como el retardo acumulado de la cola de la aplicación i en el slot t, y $Q_j(t)$ como el retardo acumulado de la cola del procesador i en el slot t.

El cálculo del retardo acumulado se lleva a cabo en un script de Python separado del resto de la lógica del algoritmo. Se invoca la función designada para el cálculo del retardo, suministrándole información extraída del Request, así como otros valores relevantes, como la decisión previa ($\alpha_{ij}(t)$) concerniente a la aplicación o al procesador respectivo, y un nombre que identifica el elemento para el cual se está efectuando dicho cálculo (ya sea una aplicación o procesador). Después, en la función, se consulta un diccionario, utilizando el nombre que le fue proporcionado, con el fin de determinar la cantidad de paquetes que aún no han sido enviados para procesar en cada intervalo de tiempo previo al actual. Con esta información, y la proporcionada por el algoritmo, se procede a la actualización de los valores en cada slot, incluyendo los valores del slot actual S. Por último, se calcula el retardo acumulado, como se puede ver a continuación Ec.(4.10):

$$\sum_{i=0}^{S} S - i \tag{4.10}$$

En cualquiera de los dos enfoques, las decisiones se toman únicamente entre las colas de las aplicaciones y las de los procesadores. Por otro lado, los procesadores drenan todos los paquetes posibles en ese intervalo de tiempo.

4.4. Modelado de Sistema con Nodo Cloud

En esta sección, se introduce el nodo Cloud al modelo de sistema previamente presentado.

La formulación es la misma, siendo la única diferencia que la cola asociada el nodo Cloud es lo que se ha enviado menos lo que se ha procesado. Esto incluye tanto el almacenamiento en las comunicaciones (en la red intermedia) como en el propio nodo Cloud (el cual se puede asumir que es 0). Además, el marco teórico permite añadir funciones de coste sobre las decisiones, lo que nos permite definir políticas de decisión. En este caso esta función representará el coste de uso del nodo Cloud y será proporcional a los datos enviados al Cloud.

Se recuerda que M es el número de aplicaciones y N el de procesadores. El procesador con índice $j = N$ corresponde con un nodo Cloud, mientras que el resto de procesadores $j \in \{1, \ldots, N-1\}$ son locales al nodo fog. Además, se define una función de coste C de uso del nodo Cloud proporcional a la cantidad de datos, tal que:

$$C(t) = K \sum_{i=1}^{M} b_{i,N}(t) \tag{4.11}$$

De tal manera que K es el coste de procesar datos de las aplicaciones. Finalmente, el problema modifica sería:

$$\max_{\alpha(t)} \quad \sum_{i,j} b_{i,j}(t) \cdot W_{ij}(t) - K \sum_{i=1}^{M} b_{i,N}(t)$$

4.5. Implementación en Clases

Para finalizar este capítulo, se presenta un diagrama de clases compuesto por tres elementos que representan los algoritmos desarrollados. Como se observa en la Figura 4.3, se parte de una interfaz común llamada 'Algoritmo', que define una serie de propiedades y métodos que se esperan en cualquier clase implementada a partir de esta interfaz. Las operaciones 'call()' como función principal de llamada al algoritmo, 'CalCap()' como función adicional para el calculo de bounds, 'init()' para inicialización de variables y 'objetive()' como función objetivo del algoritmo, son funciones que estarán presentes en todas las clases implementadas a partir de la interfaz Algoritmos:

- **Clase AlgoritmoBuffer**

  Esta clase implementa la interfaz Algoritmo, lo que significa que proporciona implementaciones concretas de todos los métodos definidos en la interfaz. Además, esta clase también presenta sus propios atributos, que hacen referencia a las ocupaciones de las colas, tanto de las aplicaciones como de los procesadores.

- **Clase AlgoritmoRetardos**

  Al igual que la clase anterior, proporciona implementaciones concretas para las operaciones definidas en la interfaz. Además, presenta atributos propios que representan los retardos acumulados en las

colas de las aplicaciones y los procesadores. Así, como un método adicional encargado de calcular dichos retardos.

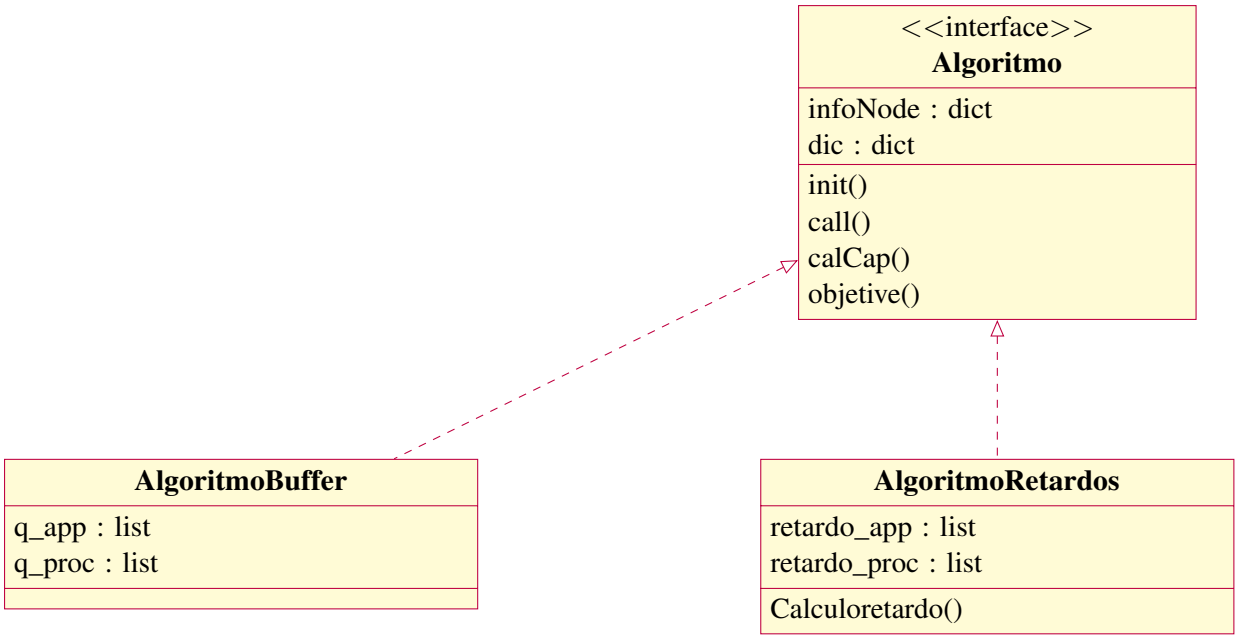


Figura 4.3: Diagrama de Clases de los Algoritmos

Capítulo 5

Resultados

En este capítulo, se lleva a cabo un conjunto de experimentos sobre la plataforma desarrollada, con el objetivo de analizar el rendimiento de los algoritmos propuestos. Se presentan cuatro escenarios específicos:

- En primer lugar, en la Sección 5.1 se estudia el correcto funcionamiento del algoritmo basado en la ocupación de las colas analizando su rendimiento ante diferentes configuraciones.
- A continuación, en la Sección 5.2 se comparan los dos algoritmos desarrollados en el marco de este trabajo.
- Como extensión de lo anterior, en la Sección 5.3 se examina el comportamiento de los algoritmos ante diferentes restricciones en la capacidad de transmisión entre aplicaciones y procesadores.
- Por último, en el cuarto escenario que se presenta en la Sección 5.3, se evalúa un esquema que ofrece la opción de procesamiento en el nodo Cloud.

Estos cuatro escenarios de experimentación permiten una evaluación exhaustiva del desempeño de los algoritmos en diversas situaciones, proporcionando una visión completa y detallada de su eficacia y aplicabilidad.

5.1. Validación

En todas las configuraciones que se utilizan en esta sección, se parte de un sistema con $M = 2$ aplicaciones y $N = 2$ procesadores, todos ubicados en un único nodo Fog, tal como se ilustra en la Figura 5.1. Además, se cuenta con un nodo Master y, aunque no se emplea en los primeros escenarios, se reserva un nodo Cloud que será empleando en experimentos futuros. Los detalles específicos de la configuración se presentan en la Tabla 5.1. En el Anexo A se muestra un ejemplo del archivo de configuración JSON.

Es importante destacar que, en todos los casos, las ejecuciones abarcan un periodo de 500 intervalos de tiempo (slots), cada uno de un segundo. En cada intervalo, las aplicaciones generan un servicio que consta de un número aleatorio de paquetes. Además, se especificará la tasa particular de generación de servicios de las aplicaciones en todos los escenarios.

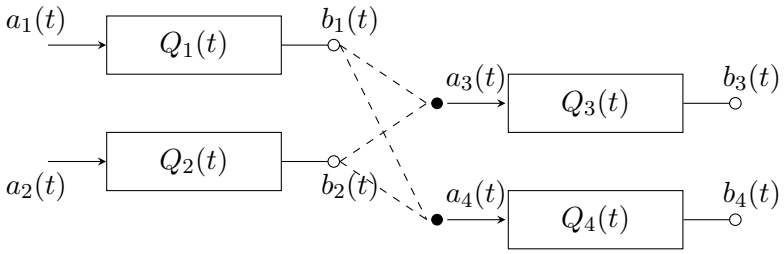


Figura 5.1: Modelo del sistema con 2 aplicaciones y 2 CPUs.

Tabla 5.1: Configuración base para la muestra de resultados.

Parámetro	Valor configurado
Tiempo de simulación (slots)	500
Slot	1 s
Número de aplicaciones	2
Número de CPUs en el nodo Fog	2
Tasa de generación de servicios	Continua 0.5 $serv/s$
Tasa de generación de paquetes	Poisson [6,9,10,12] $pkts/s$
Capacidad de procesamiento de una CPU	2 kB/s
Capacidad procesamiento Cloud	[500,100,10] $Mbps$
Longitud de un paquete	200 $Bytes$
Longitud cabeceras de un paquete	20 $Bytes$
Límites para las restricciones	[8,....,12] $pkts/s$

Este primer escenario tiene como objetivo verificar el comportamiento de los algoritmos de acuerdo al esperado. Con este propósito, se realizaron pruebas relacionadas con el primer algoritmo desarrollado (ver Sección 4.2). El siguiente conjunto de resultados se centrará en analizar el impacto de varias configuraciones en el comportamiento del algoritmo, centrándose en la estabilidad de las colas. Para ello, se considera el modelo mencionado previamente, excluyendo el nodo Cloud. Esta particularidad permite una observación más precisa de la estabilidad de las colas de aplicaciones y procesadores, y también facilita una evaluación crítica en términos de capacidad de procesamiento. En todos los escenarios presentados en este trabajo, salvo que sea indicado, el algoritmo aplica las restricciones definidas en las Ecuaciones (4.5) y (4.6).

Inicialmente, se establece una tasa de 9 $pkts/s$ en ambas aplicaciones, lo que resulta en una tasa agregada de 18 $pkts/s$. Esta configuración implica operar al 90 % de la capacidad total de procesamiento, que se fija en 2000 $Bytes/s$ en cada procesador. Esto permite una observación óptima y clara de los resultados.

La Figura 5.2 muestra la estabilidad de la tasa promedio de las colas de aplicaciones a la derecha y de las colas de procesadores a la izquierda, utilizando la Ec. (4.2) en el algoritmo descrito en la Sección 4.2, utilizando la ocupación de los buffer como métrica de las colas.

En términos generales, como se observa en la Figura 5.2, se puede constatar que el algoritmo propuesto logra mantener la estabilidad de las colas en ambos contextos. Es importante notar que, conforme a lo previsto en la Ec. (4.2), a medida que se considera la variable aleatoria $Q_k(t)$, que representa la ocupación de la cola (aplicación o procesador) durante intervalos de tiempo cada vez más extensos, el promedio de $Q_k(t)$ disminuye gradualmente, hasta aproximarse a cero. En otras palabras, esto sugiere que el valor medio de $Q_k(t)$ se reduce a medida que se analizan intervalos de tiempo más prolongados, lo que es un

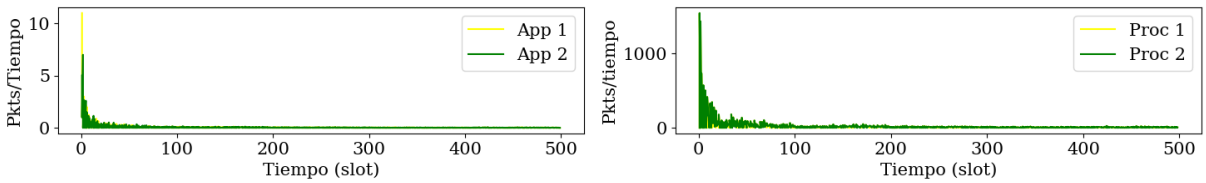


Figura 5.2: Mean Rate Stability del algoritmo basado en ocupación del buffer con tasas iguales. A la izquierda colas de las aplicaciones, a la derecha de los procesadores.

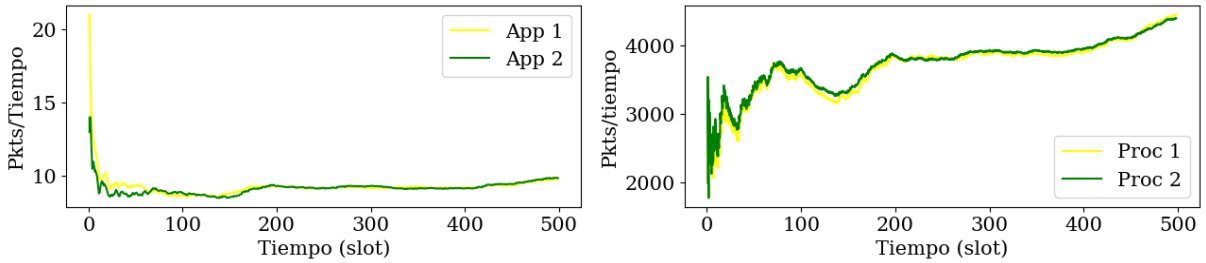


Figura 5.3: Strong Stability del algoritmo basado en ocupación del buffer con tasas iguales. A la izquierda colas de las aplicaciones, a la derecha de los procesadores.

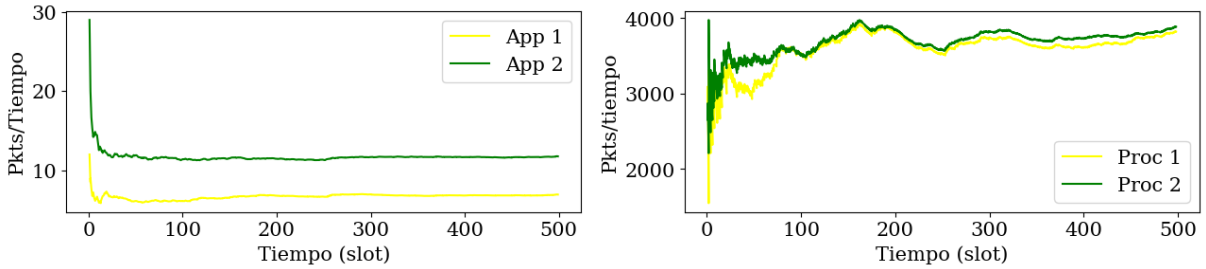


Figura 5.4: Strong Stability del algoritmo basado en ocupación del buffer con tasas distintas. A la izquierda colas de las aplicaciones, a la derecha de los procesadores.

indicativo de la estabilidad de las colas.

Manteniendo la misma tasa en ambas aplicaciones, en la Figura 5.3 se presenta la métrica de Strong Stability, donde se aprecia, de acuerdo con Eq. (4.3), que el sistema de colas es capaz de controlarse, manteniéndose estable a lo largo del tiempo. En otras palabras, el promedio temporal de la variable $Q_k(t)$ se mantiene acotado, sin aumentar de manera indefinida a medida que el tiempo progresa.

Se estudia ahora el impacto de la tasa de tráfico sobre la ocupación de las colas. Para verificar que el algoritmo garantiza la estabilidad de las colas en escenarios menos equilibrados, se establecen tasas diferentes para las aplicaciones. Se fijaron tasas de 6 $pkts/s$ para la primera aplicación y 12 $pkts/s$ para la segunda aplicación, mientras que las capacidades de procesamiento de los procesadores se mantuvieron constantes. Como se puede apreciar en la Figura 5.4, la métrica de Strong stability pone de manifiesto que el algoritmo garantiza efectivamente la estabilidad de las colas.

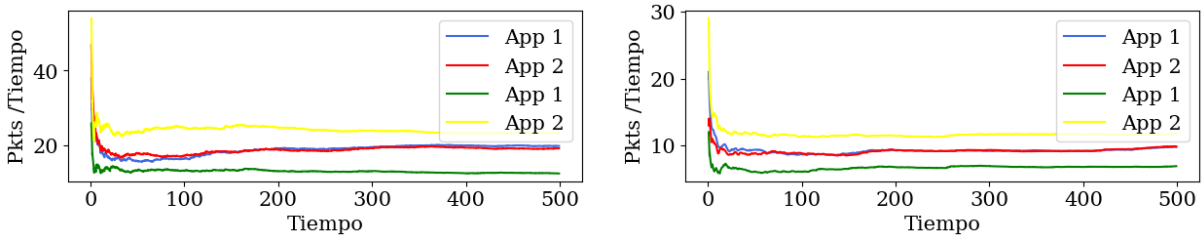
5.2. Enfoque Comparativo de Algoritmos

El objetivo de esta sección es realizar una comparativa entre los dos algoritmos desarrollados. Con este propósito, se han explorado diferentes escenarios para cada uno de los algoritmos. Para simplificar, se

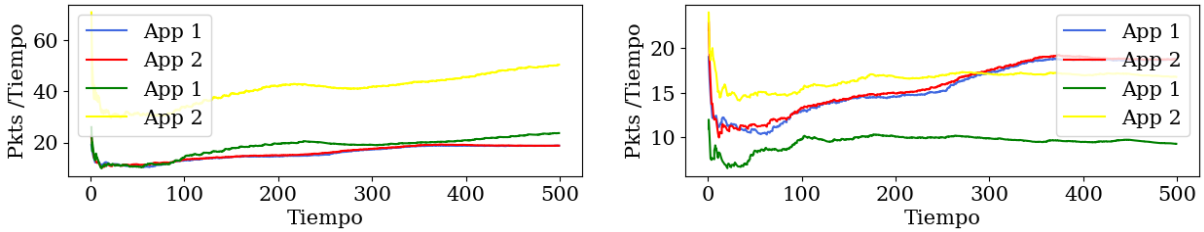
emplea el mismo sistema y las configuraciones que se detallaron en la primera sección de este capítulo y las ejecuciones tienen duraciones de 500 slots.

En la Figura 5.5, se presenta una comparativa del Strong Stability para ambos algoritmos. Los resultados abarcan las dos configuraciones previamente mencionadas: una con tasas idénticas en ambas aplicaciones (representadas en rojo y azul), y otra con diferentes tasas (representadas en verde y amarillo). En el primer conjunto de resultados (gráficas superiores), se puede destacar que el algoritmo basado en retardos permite mantener la estabilidad en las colas, cuya ocupación se sitúa dentro de un rango limitado, que varía según la tasa y la aplicación, como se observó el algoritmo analizado en la sección anterior.

En todos los escenarios evaluados hasta el momento, se puede ver que, cuando las tasas son equivalentes, la estabilidad se mantiene prácticamente simétrica para ambas aplicaciones. Al contrario, cuando las aplicaciones generan diferentes volúmenes de tráfico, se producen ligeras variaciones en su estabilidad promedio a lo largo del tiempo.



(a) Strong Stability con suma de tasas menor que los procesadores. A la izquierda algoritmo basado en retardos y a la derecha algoritmo basado en ocupación buffer.



(b) Strong Stability con suma de tasas mayor que los procesadores, al borde de la capacidad del sistema . A la izquierda algoritmo basado en retardos y a la derecha algoritmo basado en ocupación buffer.

Figura 5.5: Strong Stability basado en ocupación de las colas en ambos algoritmos en dos diferentes escenarios.

Para ilustrar un caso de menor estabilidad (Strong Stability), se contemplan dos configuraciones en las que se opere al borde de la capacidad del sistema: un escenario con aplicaciones homogéneas, en el que se establece una tasa de 10 $pkts/s$ para ambas; y un caso con tasas dispares, 5 $pkts/s$ para la primera aplicación, y 15 $pkts/s$ para la segunda. En ambos casos se alcanza un tráfico total de entrada de 20 $pkts/s$, que coincide con la capacidad máxima de procesamiento de 2000 $Bytes/s$ de los procesadores. Los resultados se muestran en la parte inferior de la Figura 5.5.

Como se puede observar, al comparar los resultados entre las gráficas superiores e inferiores que se muestran en la Figura 5.5, se ve que, a medida que aumenta la tasa de generación de servicios, la estabilidad de las colas varía. Así, se vuelve más complicado mantener acotado un promedio temporal de

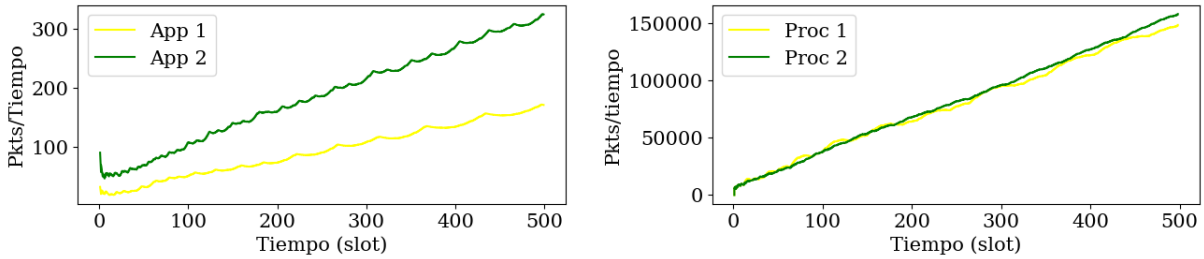


Figura 5.6: Strong Stability de las colas basadas en ocupación de las colas del algoritmo basado en retardos en condiciones de saturación.

$Q_k(t)$. Para analizar este comportamiento, en la Figura 5.6 se incremente la tasa de entrada hasta los 25 $pkts/s$ (5 $pkts/s$ y 20 $pkts/s$) superando la capacidad y se representa la estabilidad de las colas basado en la ocupación de las colas. Como se puede ver, no se logra garantizar la estabilidad en las colas. En este escenario, el promedio temporal de $Q_k(t)$ no se encuentra limitado y crece de manera continua.

Con el propósito de examinar en detalle el rendimiento de ambos algoritmos, se procedió a calcular el retardo medio en ambos casos. Este parámetro refleja el tiempo promedio transcurrido desde la generación de los servicios en el nodo Fog hasta su procesamiento completo.

Tal como se puede apreciar en la Figura 5.7, cuando ambas aplicaciones presentan la misma tasa (rojo y azul), se observa una ligera disparidad en los retardos medios entre ambas, para los dos algoritmos analizados. Este comportamiento se puede atribuir a diversos factores, como la naturaleza aleatoria en la generación de servicios y la complejidad inherente al proceso de procesamiento. No obstante, es interesante notar que, en ambos escenarios, la segunda aplicación tiende a manifestar un retardo medio superior en comparación con la primera. Esto se puede deber a que la solución del problema de optimización resultante en los algoritmos no sea única, y que los solver utilizados tiendan a devolver una solución óptima que seleccione la primera aplicación con más probabilidad que la segunda.

Por otro lado, cuando las aplicaciones presentan tasas distintas (verde y amarillo), se observa un patrón predecible: la aplicación con una tasa más alta exhibe un retardo ligeramente superior, consecuencia directa del mayor volumen de tráfico. Este fenómeno se refleja claramente en la Figura 5.7, donde se evidencia que la aplicación 2, con una tasa de 12 $pkts/s$, presenta un retardo mayor.

En el escenario de aplicaciones homogéneas, las diferencias entre los algoritmos son relativamente insignificantes. Sin embargo, al examinar situaciones con tasas distintas, se aprecia que el algoritmo que utiliza el retardo acumulado en las colas como métrica logra obtener un mayor equilibrio en los retardos (gráfica superior). Este resultado era de esperar, dado que dicho algoritmo centra sus esfuerzos en proporcionar el menor retardo acumulado en las colas.

5.3. Implementación de Nuevas Restricciones en los Algoritmos

La presente sección tiene como objetivo mostrar cómo se ve afectada la estabilidad de las colas mediante la implementación de restricciones adicionales en los algoritmos. Para ambos algoritmos se establece la siguiente restricción:

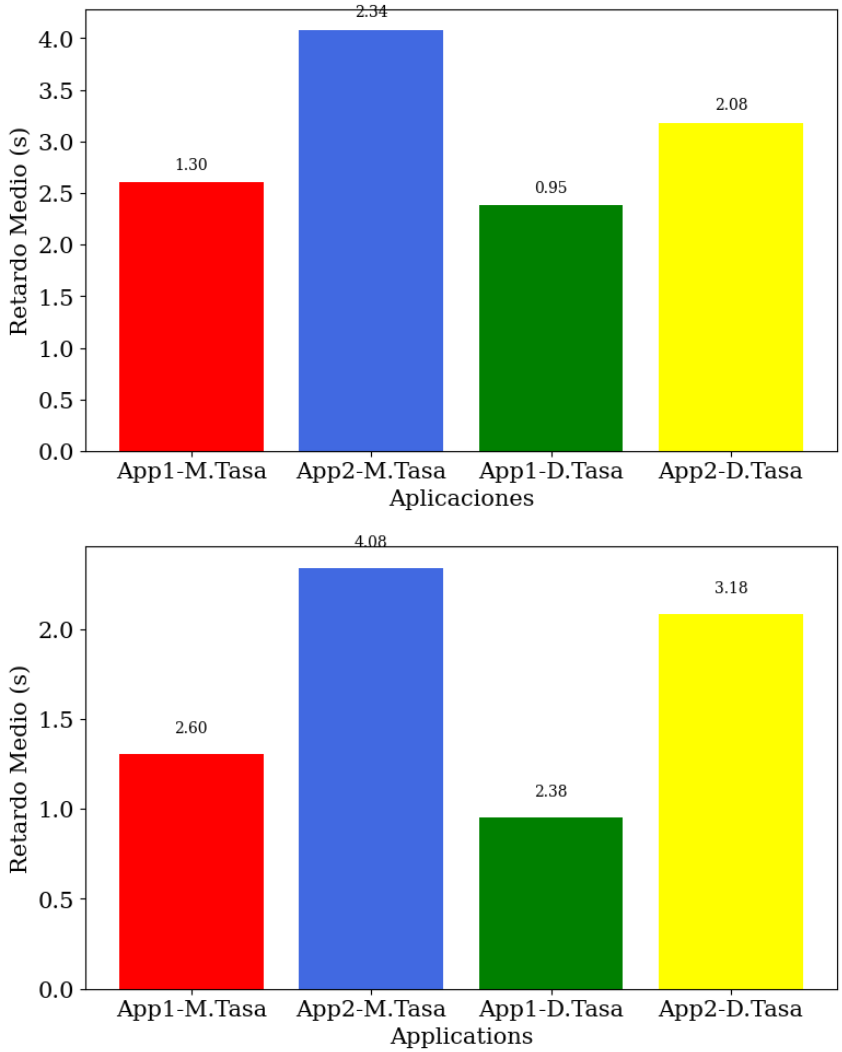


Figura 5.7: Gráfica Comparativa Retardo Medio Algoritmos con M.Tasa como misma tasa y D.Tasa como distintas tasas. Gráfica superior algoritmo basado en retardos,gráfica inferior algoritmo basado en la ocupación buffer.

$$\sum_{i=1}^{M} \alpha_{ij}(t) \leq L_j(t) \quad \forall j \in \{1, \ldots, N\} \tag{5.1}$$

De este modo, se pretende asegurar el tráfico que se transfiere a cada procesador j, suma de $\alpha_{ij}(t)$, se encuentre por debajo de un límite $L_j(t)$. En otras palabras, la Ec. (5.1) establece un límite en la cantidad de paquetes que puede recibir simultáneamente el punto de procesamiento j en el slot t. Cada punto de procesamiento tiene su propio límite $L_j(t)$, y la suma de las $\alpha_{ij}(t)$ asignadas a ese procesador no debe exceder ese límite en ese momento t.

En primer lugar, se procede a evaluar el impacto del parámetro $L_j(t)$ en los algoritmos. En esta sección, se mantiene el modelo de sistema con dos aplicaciones y dos procesadores, y se evalúan 500 intervalos de tiempo. En la primera prueba, se establece un límite de 11 $pkts/s$ en ambos procesadores, lo cual se sitúa por encima de la tasa de generación de las aplicaciones, que es de 10 $pkts/s$ para ambas aplicaciones. La Figura 5.8 ilustra una comparativa de la estabilidad (Strong stability) en el modelo de sistema con

restricciones base (también se establecen tasas de 10 $pkts/s$ en ambas aplicaciones) (Ecuación 4.5) en los colores rojo y azul, y el modelo de sistema con la nueva restricción (Ecuación 5.1) en los colores verde y amarillo. Esto se aplica al algoritmo basado en retardos (4.3).

Cabe destacar que, si bien la estabilidad de las colas de las aplicaciones se mantiene similar en ambos casos, se observa una diferencia significativa en el comportamiento de los procesadores. En el escenario con el modelo de restricción base, se logra estabilizar las colas de los procesadores en los últimos 100 intervalos de tiempo de la simulación. Esto es comprensible, ya que se opera en el límite con una tasa total de 20 $pkts/s$. Sin embargo, al introducir la nueva restricción, se logra alcanzar el promedio temporal deseado en un período más corto.

Esta tendencia se mantiene constante en ambos algoritmos. En otras palabras, cuando el límite establecido supera la tasa de generación de las aplicaciones (incluso si se sitúa cerca del límite), se observa una mejora notable en la estabilidad por parte de los procesadores.

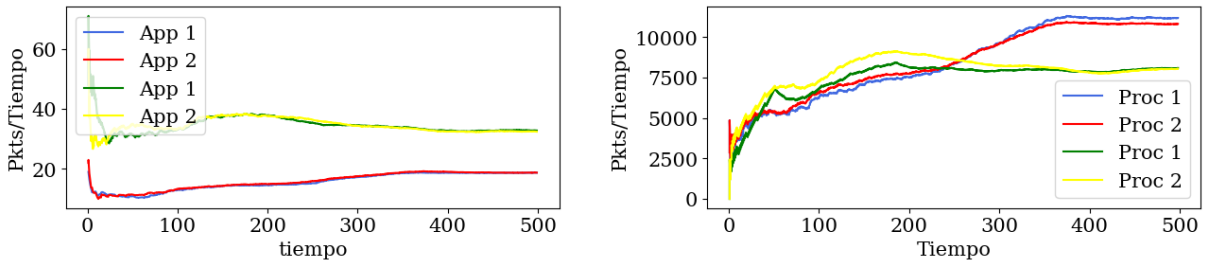


Figura 5.8: Strong Stability del algoritmo basado en retardos con la nueva restricción y con límite de escritura en procesadores por encima de tasa de aplicaciones.

En la segunda prueba, sin embargo, se observa un escenario diferente. Como se representa en la Figura 5.9, para el mismo Algoritmo 4.3, cuando se establece un límite inferior a la tasa de de las aplicaciones (tasa de 9 $pkts/s$ y un límite de 8 $pkts/s$), el comportamiento de las colas comienza a fluctuar. La estabilidad que antes existía en las colas con las restricciones base (tasa de 9 $pkts/s$), descritas en la Ec. 4.5, desaparece tanto en las colas de las aplicaciones como en las de los procesadores. En esta situación, el algoritmo responde produciendo estabilidad en uno de los procesadores, pero como consecuencia, desestabiliza el otro. Este comportamiento puede deberse a diversos factores. Por ejemplo, establecer un límite inferior muy cercano a la tasa de procesamiento puede provocar que las pequeñas fluctuaciones en la llegada de paquetes o en el tiempo de procesamiento tengan un impacto desproporcionado en la estabilidad. Se observa el mismo comportamiento en ambos algoritmos.

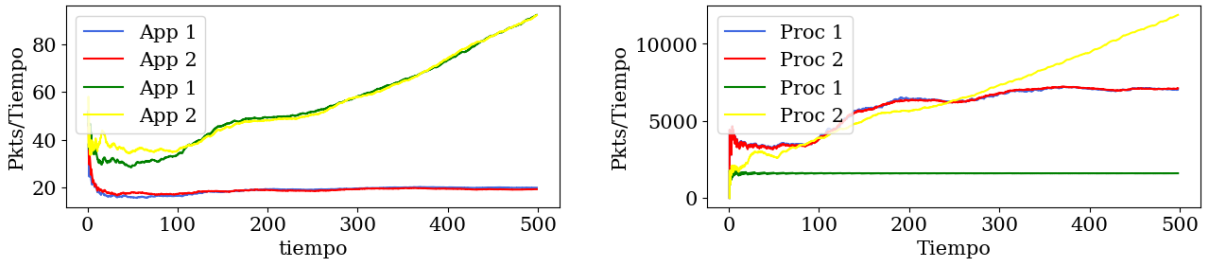


Figura 5.9: Strong Stability del algoritmo basado en retardos con la nueva restricción y con límite de escritura en procesadores por debajo de tasa de aplicaciones.

La Figura 5.10 ilustra la comparación entre dos variantes del algoritmo basado en retardos: una con las restricciones base (tasas de 6 y 12 $pkts/s$) y otra con las nuevas restricciones (tasas de 6 y 12 $pkts/s$, junto con un límite de 8 $pkts/s$ en ambos procesadores). Se observa que, incluso cuando las tasas difieren entre sí, el algoritmo opera de manera similar cuando el límite establecido es inferior a al menos una de las tasas establecidas.

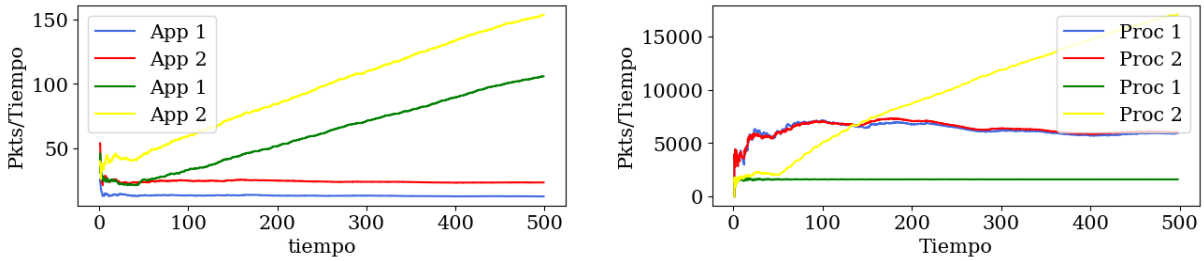


Figura 5.10: Gráfica Strong Stability Algoritmo Retardos nueva restricción diferentes tasas, límite inferior.

Para concluir esta sección y manteniendo las mismas tasas de aplicaciones que en la configuración anterior, se han impuesto límites de 12 $pkts/s$ para el primer procesador y 8 $pkts/s$ para el segundo, y los resultados se muestran en la Figura 5.11. Es fundamental destacar que ambos límites superan la tasa mínima requerida por ambas aplicaciones. Con esta disposición, el algoritmo logra proporcionar cierta estabilidad en las colas, tal como se observó bajo las restricciones base.

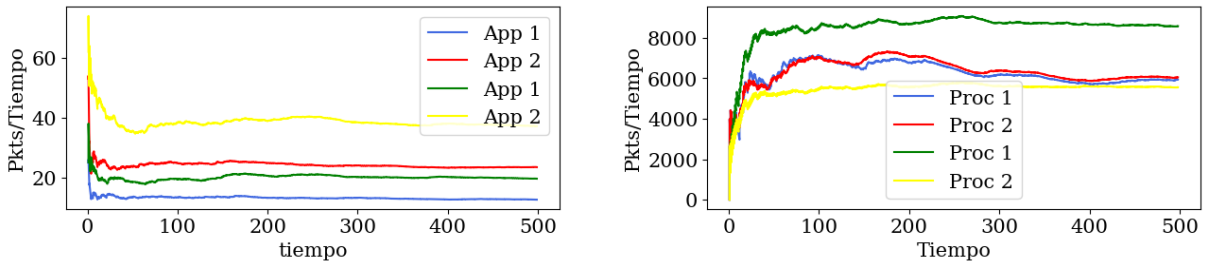


Figura 5.11: Gráfica Strong Stability Algoritmo Retardos nueva restricción diferentes tasas, diferentes límites superiores.

En contraste, aunque ambos límites superen la tasa mínima, si el primer procesador tiene el límite inferior (8 $pkts/s$), las colas de las aplicaciones no logran establecer un promedio temporal limitado, es decir, no llegan a estabilizarse, como se puede observar en la Figura 5.12. Además, las colas de los procesadores exhiben un patrón similar al caso donde se establece el mismo límite para ambos procesadores, pero este límite es inferior a las tasas de las aplicaciones, donde el promedio temporal del primer procesador se limita al valor de la capacidad configurada, mientras que el segundo procesador no logra alcanzar un promedio temporal limitado, y tiende a aumentar a medida que pasa el tiempo.

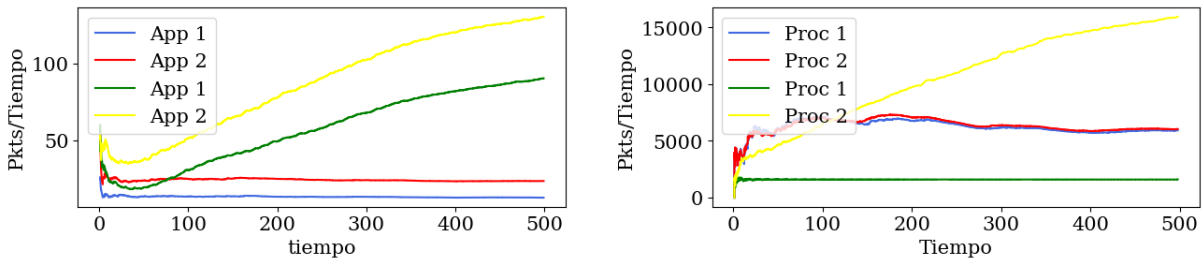
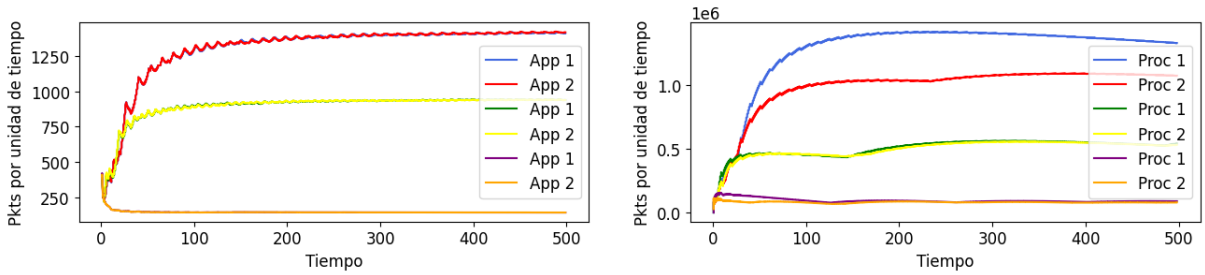


Figura 5.12: Gráfica Strong Stability Algoritmo Retardos, nueva restricción diferentes tasas, diferentes límites, Proc1 menor límite.

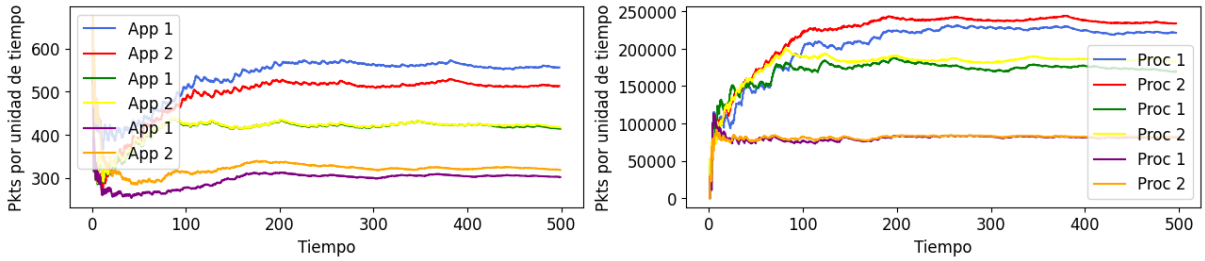
5.4. Sistema con nodo Cloud

Para concluir este capítulo, en esta sección se introduce el nodo Cloud al sistema previamente presentado, como se comentó en la Sección 4.4.

Para ambos algoritmos, se ha configurado el sistema con una capacidad de 100 $Mbps$ para el nodo Cloud, así como una capacidad de 625 KB/s en los procesadores, y tasas de 2051 $pkts/s$ para ambas aplicaciones. En la Figura 5.13 se ilustra la medida de Strong Stability de la ocupación de las colas para ambos algoritmos. Como se ha observado en secciones anteriores, cuando la tasa total de las aplicaciones supera la capacidad de los procesadores, las colas de ambos experimentan cierta inestabilidad.



(a) Algoritmo basado en la ocupación de las colas.



(b) Algoritmo basado en retardos.

Figura 5.13: Strong Stability ocupación colas con nodo Cloud.

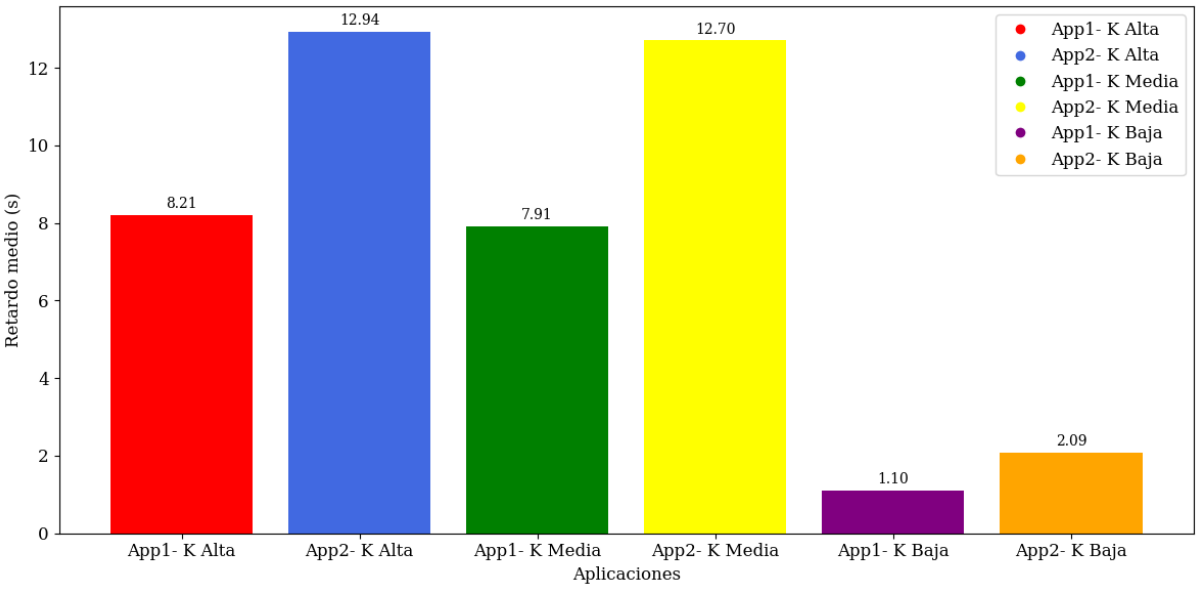
La introducción del nodo Cloud tiene un impacto significativo: el algoritmo logra estabilizar las colas de las aplicaciones. Sin embargo, se nota una variación que depende del valor asignado a K. Si K tiene un valor bajo ($K < 10$), prácticamente todos los servicios se envían al nodo Cloud, ya que su uso no conlleva un coste significativo, y su capacidad de uso no está restringida de manera severa (naranja y morado). Por

otro lado, si K toma un valor muy alto ($K > 1500$), el coste de utilizar el nodo Cloud será muy elevado, lo que resultará en un uso puntual, y únicamente en casos extremadamente necesarios en los que no se pueda realizar el procesamiento local (rojo y azul).

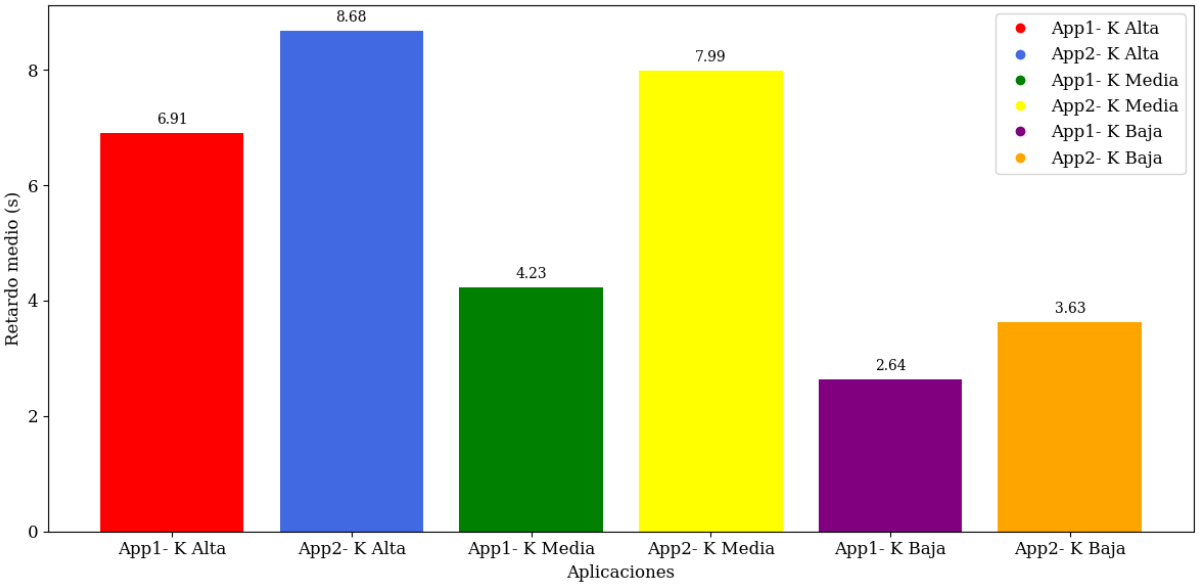
El escenario óptimo se encuentra en un valor intermedio de K ($K = 1000$). Esta configuración proporciona un equilibrio en el uso del nodo Cloud. Como se puede observar en las gráficas representadas en colores verde y amarillo, se consigue la mayor estabilidad en las colas de las aplicaciones. Además, esta configuración también favorece un comportamiento homogéneo, en la estabilidad entre ambas aplicaciones.

En resumen, la presencia del nodo en la nube bajo diferentes valores de K tiene un impacto directo en la estabilidad del sistema, con un valor óptimo de K que logra un equilibrio en el uso del nodo Cloud, y maximiza la estabilidad de las colas de aplicaciones. Se llevaron a cabo pruebas en tres configuraciones distintas en el nodo Cloud y el resto de elementos: 500 $Mbps$, 100 $Mbps$ y 10 $Mbps$. En cada uno de estos tres escenarios, se pudo observar un comportamiento similar y consistente, en el que simplemente variaba el valor en el que se estabiliza. Para concluir, se muestra el retardo promedio en ambos escenarios. Es evidente que a medida que el valor de K aumenta, también lo hace el retardo promedio de los servicios de las aplicaciones. Este comportamiento se podría haber esperado, ya que a medida que K crece, el uso del nodo Cloud se vuelve más limitado. Como resultado, el procesamiento de todos los servicios debe distribuirse entre los procesadores locales, lo que conlleva un aumento en el tiempo que un servicio requiere para ser procesado.

En la Figura 5.14 se ilustra claramente lo descrito anteriormente. En ella se puede apreciar que el retardo promedio de los servicios, es decir, el tiempo que tarda un servicio desde su creación hasta su procesamiento, aumenta conforme incrementa el valor de K, lo que refleja la restricción en el uso del nodo en la nube, y la consecuente extensión en el tiempo de procesamiento de los servicios. Además, se puede apreciar que el algoritmo basado en retardos logra, en las mismas condiciones, valores de retardos significativamente menores en las configuraciones con valores de K elevados.



(a) Algoritmo basado en la ocupación de las colas.



(b) Algoritmo basado en retardos.

Figura 5.14: Retardo medio aplicaciones con nodo Cloud.

Capítulo 6

Conclusiones

Utilizando como punto de partida una plataforma de emulación que contempla arquitecturas IoT-Fog-Cloud, se ha establecido como objetivo principal de este trabajo el desarrollo de dos algoritmos en el lenguaje de programación Python para distribuir trabajos de computación en dichos escenarios. Ambos consideran métricas clave como la ocupación de colas y el retardo acumulado, al mismo tiempo que buscan asegurar un control sobre la estabilidad del sistema. Para lograr esto, se ha planteado un modelo detallado del sistema y se ha concebido la implementación de una política de control, basada en la teoría de control de Lyapunov, apoyándose en el marco teórico propuesto por Neely [11].

Este enfoque se adapta a patrones de tráfico que son variables y, en ocasiones, impredecibles. Además, se tiene en cuenta la diversidad en las capacidades de cálculo de los nodos y los costes asociados con los recursos en la nube. Se comprobó el correcto comportamiento del modelo, a través de una serie de experimentos sobre escenarios representativos. Los resultados obtenidos, tal como se han detallado en el documento, ilustran que las soluciones desarrolladas logran un equilibrio entre la utilización de instancias Fog, y la combinación de instancias Fog con recursos en la nube. Asimismo, se ha demostrado cómo determinadas configuraciones pueden tener un impacto relevante en el comportamiento del sistema, y cómo los algoritmos propuestos son capaces de adaptarse ante variadas situaciones y restricciones, incluyendo condiciones cambiantes de flujos de tráfico y demandas.

Un enfoque futuro de investigación prometedor sería incorporar funciones logarítmicas al problema de optimización, para analizar su impacto en el rendimiento del sistema. Esta modificación podría ofrecer una perspectiva más completa sobre cómo ciertas métricas, como la ocupación de colas y el retardo acumulado, responden a cambios específicos en el sistema.

Además, se podría examinar un modelo en el que se permita ajustar el valor del coste de procesamiento en los nodos Fog, con la flexibilidad de considerar valores mayores a cero. Esta característica permitiría tener en cuenta los costes asociados al funcionamiento de los servidores locales, alineándose con los límites previamente definidos. Incorporar este criterio podría resultar particularmente relevante, ya que brindaría una visión más completa del efecto del consumo en el rendimiento general del sistema, enriqueciendo la evaluación de las soluciones propuestas.

Adicionalmente, cabe destacar la posibilidad de introducir variaciones en forma de diferentes tipos de servicios con asignaciones de prioridad. Este enfoque permitiría explorar cómo las distintas prioridades asignadas a los servicios influirían en el comportamiento del sistema, dependiendo del escenario bajo

consideración. Este análisis puede arrojar información crucial sobre cómo las prioridades y los tipos de servicios interactúan con las arquitecturas IoT-Fog-Cloud, ofreciendo una perspectiva integral sobre la adaptabilidad y eficiencia del sistema en diferentes contextos.

En conjunto, estos escenarios de estudio brindan una oportunidad para analizar con mayor profundidad los factores energéticos, las prioridades de los servicios y su interacción en la dinámica de las arquitecturas IoT-Fog-Cloud, contribuyendo a una comprensión más completa y a la toma de decisiones informadas en la operación de este tipo de sistemas, lo que redundaría en una mejora de sus prestaciones.

Para finalizar me gustaría mencionar que este proyecto ha sido el logro más significativo de mi carrera académica hasta la fecha. Ha representado un desafío excepcional tanto a nivel académico como personal. Enfrentar y resolver las complejidades de desarrollar algoritmos en el contexto de arquitecturas IoT-Fog-Cloud me proporcionó un profundo entendimiento de las dinámicas involucradas.

Además, este proyecto marcó un hito personal al realizarse en su mayoría de manera remota. A pesar de las dificultades que la distancia y las circunstancias pudieron haber impuesto, la determinación y la pasión por el tema me llevaron a superar las barreras y lograr resultados sólidos.

A lo largo de este proceso, he aprendido no sólo sobre la materia en sí, sino también sobre mi propia capacidad para gestionar proyectos complejos, adaptarme a situaciones cambiantes y perseverar en la búsqueda del conocimiento. Estoy segura de que las habilidades y la experiencia adquiridas en este proyecto me servirán en mi futuro académico y profesional.

Bibliografía

[1] H. Tyagi y R. Kumar. "Cloud Computing for IoT". En: *Internet of Things (IoT)*. Ed. por M. M. Alam, K. M. Shakil y S. Khan. Springer, 2020. Cap. 2, págs. 21-40. URL: https://doi.org/10.1007/978-3-030-37468-6_2.

[2] D. C. Trueba. "Arquitectura fog/cloud en el ámbito del IIoT: modelado, despliegue y análisis". En: *Repositorio abierto de la Universidad de Cantabria* (2021), págs. 1-47.

[3] M. A. López Peña e I. Muñoz Fernández. "SAT-IoT: An Architectural Model for a High-Performance Fog/Edge/Cloud IoT Platform". En: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. 2019, págs. 633-638. DOI: 10.1109/WF-IoT.2019.8767282.

[4] J. Guth y et al. "Comparison of IoT platform architectures: A field study based on a reference architecture". En: *Cloudification of the Internet of Things (CIoT)*. IEEE. 2016, págs. 1-6. DOI: 10.1109/CIoT.2016.015.

[5] L. Bagherzadeh, H. Shahinzadeh, H. Shayeghi, A. Dejamkhooy, R. Bayindir y M. Iranpour. "Integration of Cloud Computing and IoT (CloudIoT) in Smart Grids: Benefits, Challenges, and Solutions". En: *2020 International Conference on Computational Intelligence for Smart Power System and Sustainable Energy (CISPSSE)*. 2020, págs. 1-8. DOI: 10.1109/CISPSSE49931.2020.9212195.

[6] P. Habibi, M. Farhoudi, S. Kazemian, S. Khorsandi y A. Leon-Garcia. "Fog Computing: A Comprehensive Architectural Survey". En: *IEEE Access* 8 (2020), págs. 69105-69133. DOI: 10.1109/ACCESS.2020.2983253.

[7] A. Alrawais, A. Alhothaily, C. Hu y X. Cheng. "Fog Computing for the Internet of Things: Security and Privacy Issues". En: *IEEE Internet Computing* 21.2 (2017), págs. 34-42. DOI: 10.1109/MIC.2017.37.

[8] B. R. Prasanthi, D. Veeraswamy, S. Abhilash y K. Ganesh. "Cloud-Fog Trustworthy Computing for Information Sharing in Dynamic IoT System". En: *2022 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE)*. Naya Raipur, India, 2022, págs. 198-202. DOI: 10.1109/WIECON-ECE57977.2022.10150596.

[9] N. Villegas. "Diseño, desarollo y evaluación de algoritmos de computación distribuida en una plataforma Fog-Cloud". En: *Repositorio abierto de la Universidad de Cantabria* (2023), págs. 1-62.

[10] C. Anderson. “Docker [Software engineering]”. En: *IEEE Software* 32.3 (2015), págs. 102-c3. DOI: 10.1109/MS.2015.62.

[11] M. J. Neely. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Synthesis Lectures on Communication Networks. Morgan & Claypool Publishers, 2010. URL: http://dx.doi.org/10.2200/S00271ED1V01Y201006CNT007.

[12] B. Ji, C. Joo y N. B. Shroff. “Delay-Based Back-Pressure Scheduling in Multihop Wireless Networks”. En: *IEEE/ACM Transactions on Networking* 21.5 (2013), págs. 1539-1552. DOI: 10.1109/TNET.2012.2227790.

[13] L. Hai, Q. Gao, J. Wang, H. Zhuang y P. Wang. “Delay-Optimal Back-Pressure Routing Algorithm for Multihop Wireless Networks”. En: *IEEE Transactions on Vehicular Technology* 67.3 (2018), págs. 2617-2630. DOI: 10.1109/TVT.2017.2770183.

Apéndice A

Ejemplo de petición y respuesta de decisión

```
{
   "service": {
      "dic_serv": {
         "1": {
            "1000": 9,
            "detailed": "none"
         },
         "2": {
            "2000": 13,
            "detailed": "none"
         }
      },
      "num_slot": 0,
      "num_app": 2,
      "p_len": 200,
      "t_slot": 1
   },
   "clouds": [],
   "fog": {
      "num_proc": 2,
      "buf_len": [
         9,
         13
      ],
      "id": 1,
      "proc1": {
         "q_len": 0.0,
         "cola": 10000,
         "proc_cap": 2000,
         "processing": false
      },
      "proc2": {
```

```
          "q_len": 0.0,
          "cola": 10000,
          "proc_cap": 2000,
          "processing": false
       }
    }
}
```

Listing A.1: Ejemplo de request

```
{
   "1": {},
   "2": {}
}
 ****************************************************************
{
   "1": {
      "1000": "local::1::9",
      "1001": "local::1::1,local::2::10"
   },
   "2": {
      "2000": "local::1::10,local::2::3",
      "2001": "local::2::7"
   }
}
 ****************************************************************
{
   "1": {},
   "2": {}
}
```

Listing A.2: Ejemplo de response

```
{
    "simulation": {
      "sim_time": 500,
      "traf_dist": "POISSON",
      "serv_dist": "CONT",
      "slot_time": 1,
      "pkt_len_dist": "CONT",
      "traf_rate": [
        6,
        12
      ],
      "pkt_len": 200,
      "HEADERLENSIZE": 4,
      "IDSIZE": 7,
      "PKTSERLEN": 5,
      "LENSEV": 4,
```

```
      "serv_rate": 0.5,
      "peso_coste": 9,
      "peso_tiempo": 1,
      "path0": "./Resultados",
      "num_app": 2,
      "v": 1.0,
      "eth": 2.0

    },
    "master": {
      "host": "localhost",
      "port": 8082,
      "algorithm": "AlgoritmoRetardos",
      "pesoCoste": 9,
      "pesoTiempo": 1
    },
    "fog_nodes": 1,
    "fog1": {
      "fog_cost": 0,
      "num_proc": 2,
      "cola": 10000,
      "processor1": {
        "capacity": 2000,
        "limit": 8

      },
      "processor2": {
        "capacity": 2000,
        "limit": 12
      }
    },
    "cloud_nodes": 0,

    "logger": {
      "nodoFog": "CRITICAL",
      "trafficGen": "CRITICAL",
      "processor": "INFO",
      "service": "INFO",
      "cloud": "CRITICAL",
      "master": "DEBUG",
      "AlgoritmoRetardos": "INFO"
    }
  }
```

Listing A.3: Ejemplo archivo de configuración