UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

TESIS

Evaluación y Optimización de Arquitecturas Distribuidas tipo Fog Computing para Internet of Things

PARA OBTENER EL TÍTULO PROFESIONAL DE:

LICENCIADO EN CIENCIA DE LA COMPUTACIÓN

Elaborado por:

VICTOR GIOVANNY MONDRAGÓN RUIZ

Asesor:

Dr. JOSÉ MANUEL CASTILLO CARA

LIMA - PERÚ

2019

Este trabajo está dedicado a mi querida madre Fidelia Ruiz Rojas,
por su esfuerzo y trabajo para que pueda ser un profesional,
por su ejemplo como persona para ayudar a los demás,
por su continuo e inmensurable amor,
y por un millón de razones más.

Agradecimientos

Agradezco a toda mi familia, en especial a mi madre, por todo la paciencia y confianza que tiene en mí, por su continuo ejemplo como madre y persona para que pueda cumplir mis metas como profesional.

Agradezco también al profesor Dr. Manuel Castillo Cara por asesorarme y darme la oportunidad de pertenecer al laboratorio Intelligent Ubiquitous Technologies - Smart Cities (IUT-SCi) de CTIC, donde gracias a los distintos proyectos en que fui parte ayudaron a mi formación profesional en el área de la investigación. Y sobre todo por su amistad.

Así también a mis profesores no solo de la Universidad Nacional de Ingeniería (UNI) sino también de la academia y el colegio, por sus enseñanzas y breves consejos para motivarme y así seguir esforzándome para cumplir mis metas.

Finalmente, agradezco a todas las personas y amigos que de alguna forma u otra han apoyado en el desarrollo de mi carrera como profesional. Somos la suma de nuestras experiencias.

Resumen

En vista de que en el laboratorio IUT-SCi de CTIC se viene creando módulos de sensores medioambientales; en la presente Tesis se realiza la evaluación y optimización de una plataforma con arquitectura Fog Computing enfocándose en el Nodo Edge y el Usuario Final.

El Fog Computing surge como un modelo complementario del Cloud Computing que busca descentralizar el trabajo en el servidor de la Nube. El empleo de técnicas de Fog Computing ofrece un mejor aprovechamiento de los recursos del sistema en general, lo que implica ahorros en costos de alquileres y mantenimiento, además de una reducción en la latencia de respuesta.

A lo largo de esta obra se detalla el caso de uso utilizando un Raspberry Pi 3 modelo B+ como Nodo Edge, un servidor físico como Nube y teniendo una aplicación en Android como Usuario Final; además de utilizar un motor de eventos complejos (Complex Event Processing-CEP) para el análisis de alarmas. Luego se presentan propuestas para la reducción de la latencia y la descentralización del trabajo de la Nube; posteriormente se presentan los detalles de la implementación.

Finalmente se realiza a extenso el estudio y resultados entre la arquitectura tradicional de Cloud Computing vs Fog Computing, en los que primero se muestran la reducción de la latencia para un Usuario Final conectado por Wifi al Nodo Edge frente Usuarios Finales conectados por 3G y 4G a la Nube. Para después, a partir de una simulación de alarmas, realizar comparaciones y discusiones de la reducción performance del CPU y el espacio de memoria RAM, al mover el módulo CEP de la Nube al Nodo Edge.

Índice general

Índice de figuras

Índice de Tablas

Índice de Códigos

Índice de Acrónimos

IoT Internet of Things

M2M Machine to Machine

SBC Single Board Computer

BLE Bluetooth Low Energy

CoAP Constrained Application Protocol

MQTT Message Queuing Telemetry Transport

PMU Performance Monitoring Unit

MAC Media Access Control

CEP Complex Event Proccesing

PAN Personal Area Network

LAN Local Area Network

WAN Wide Area Network

IP Internet Protocol

API Application Programming Interface

JSON JavaScript Object Notation

CPU Central Processing Unit

RAM Random Access Memory

DBMS DataBase Management System

GHz Gigahertz

PMU Performance Monitoring Unit

BSSID Basic Service Set Identifie

Kbp Kilo bit por segundo

Mbps Mega bit por segundo

SO Sistema Operativo

NE Nodo Edge

UF Usuario Final

Capítulo 1

Introducción

En este capítulo introductorio se hace mención sobre las motivaciones que han llevado y dieron razón al desarrollo de esta tesis de titulación para la creación de una plataforma Fog Computing. Posteriormente, se presenta brevemente los objetivos del trabajo, para finalizar con la estructura de la memoria que ofrece al lector una mejor idea del esquema trabajado.

1.1. Motivación

Hoy en día las aplicaciones del Internet de la cosas, o como sus siglas en inglés IoT, forman parte del día a día de las personas y su crecimiento en los últimos años es cada vez mayor. Así el modelo del ya conocido Cloud Computing ejecutor de la conectividad y ejecución en IoT afronta nuevos retos y límites al paso de su expansión. Todo esto sugiere nuevos estudios e investigaciones con el objetivo de lograr un desarrollo eficiente, escalable y fiable con miras a las denominadas Smart Cities.

El Fog Computing surge como un modelo complementario del Cloud Computing que busca descentralizar el trabajo en el servidor de la Nube creando una jerarquía de capas entre esta y los dispositivos finales como también entre los dispositivos y el gateway. El empleo de técnicas de Fog Computing ofrece un mejor aprovechamiento de los recursos del sistema en general lo que implica ahorros en costos de mantenimiento y alquileres, además de una reducción en la latencia. Sin embargo, en múltiples trabajos sugieren

tomar en cuenta las condiciones del lugar donde se implementan ya que los resultados pueden variar.

En el Perú los puntos a tomar en cuenta para IoT van más allá de tener sensores especiales para cada zona sino también las condiciones de conexión a red. La red de conexión a internet en el país sufre eventuales cortes y variaciones en su ancho de banda en distintos lugares, en especial los alejados de la urbes y la reconexión toma un lapso de tiempo que podría ser decisivo en el análisis y notificación de alarmas o eventos cruciales. En este contexto, una plataforma distribuida como Fog Computing se perfila como solución perfectamente viable frente a este problema.

Un aplicación de las IoT frecuentada en el país es sobre el monitoreo de la contaminación ambiental. Es por ello que en el laboratorio Intelligent Ubiquitous Technologies - Smart Cities (IUT-SCi) se viene creando módulos de sensores medioambiental con herramientas open-hardware y open-source en busca del desarrollo de una plataforma económica y adaptable que brinde este trabajo de forma eficiente.

En este sentido, este trabajo de Tesis se realiza la implementación y desarrollo de una plataforma con arquitectura Fog Computing enfocándose en el Nodo Edge para al que además se busca presentar una comparativa de la reducción de consumo y performance promedio frente al modelo de Cloud Computing.

1.2. Objetivos

A partir del contexto explicado en la motivación, el objetivo principal de esta Tesis es proponer los fundamentos para el diseño e implementación de una plataforma Fog Computing con herramientas open-source para el caso de una red de sensores con un Raspberry Pi como Nodo Edge y que tiene usuarios móviles.

En específico, el presente trabajo tiene como hitos sentar la labor sobre los siguientes puntos en cuestión:

- Implementar en el Raspberry Pi un Nodo Edge adecuado para el caso de uso y que descentralice el trabajo en la Nube.
- Estudiar técnicas y herramientas open-source para el trabajo de validación, análisis y envío de la información medida que llega al Nodo Edge.
- Disponer de un servicio de envío de la información por parte del Nodo Edge en condiciones de fallo de la red.
- Desarrollar una aplicación móvil con la cual se pueda apreciar los valores enviados por el Nodo Edge en tiempo real con la menor latencia posible, además de un servicio para notificación de alarmas.
- Realizar pruebas de latencia en la obtención de valores medidos por los módulos para las distintas conexiones que puede tener el usuario final como son por 3G, 4G y Wifi.
- Realizar pruebas de rendimiento entre el Cloud Computing y Fog Computing para comprobar la mejora del performance en el caso de uso.

1.3. Estructura de la Memoria

Con el afán de ofrecer al lector una mejor idea del presente trabajo, a continuación se detalla la estructura del mismo:

- **Introducción.** Se hace mención sobre las motivaciones que han llevado y dieron razón al desarrollo de este trabajo de Tesis. Posteriormente se presenta brevemente los objetivos del trabajo, para finalizar con la estructura de la memoria que ofrece al lector una mejor idea del esquema trabajado.
- **Estado del Arte.** Este capítulo comienza definiendo brevemente los conceptos bases en el que se desarrolla esta Tesis. Luego se menciona

algunos trabajos actuales relacionados que sirvieron de apoyo y contraste a los resultados obtenidos.

- **Caso de Uso y Tecnologías a Usar.** Se presenta el esquema general de trabajo para el caso de uso de un Raspberry Pi como Nodo Edge y la ubicación topológica de los usuarios móviles. Prosiguiendo a explicar todas las tecnologías y herramientas a usar en el desarrollo del trabajo.

- **Arquitectura y Diseño de la Plataforma.** El objetivo de este capítulo es ofrecer un marco del esquema y del diseño que se realiza para el caso de uso escogido, mostrando además una comparativa con el flujo de la información con Fog Computing frente a Cloud Computing. Terminando con el procedimiento de cómo se utilizaron las tecnologías mencionadas en el capítulo anterior.

- **Implementación de la Plataforma y Discusiones.** En este capítulo como previo se explica la diferencia de un usuario de la plataforma con un usuario del Broker, para luego especificar los detalles en la configuración del módulo CEP. Culminando con los tipos de publicaciones utilizadas y las vistas resultantes en la aplicación Android.

- **Resultados y Discusiones de Latencia y Performance.** Este capítulo se enfoca en los resultados obtenidos de la implementación descrita anteriormente, haciendo una comparativa entre las arquitecturas Cloud Computing y Fog Computing. Primero centrándose en los resultados de la latencia y su forma de obtención para 1 mensaje publicado. Finalmente, se presenta el estudio de los resultados de performance y latencia para la simulación descrita en el capítulo anterior.

- **Conclusiones y Trabajo a Futuro.** En este último capítulo se presenta las conclusiones del trabajo desarrollado e implementado, fundamentándose en los resultados del capítulo predecesor. Adicionalmente, se mencionan el trabajo a futuro para continuar con la implementación, mejora y posterior expansión de la plataforma.

Capítulo 2

Estado del Arte

Este capítulo es un previo para poner en claro los fundamentos y palabras base mencionados a lo largos de este trabajo. Primero se comenzará explicando los conceptos relacionados directamente con la plataforma a desarrollar, seguido de algunos trabajos actuales relacionados, que sirvieron de apoyo y contraste a los resultados obtenidos.

2.1. Conceptos Previos

En esta sección de conceptos previos se expondrá los principales fundamentos teóricos relacionados al proyecto en orden de tener claro cuando se mencionen en los posteriores capítulos.

2.1.1. El Internet de las Cosas

El Internet de las Cosas (IoT) es un sistema de dispositivos de computación relacionados entre sí, máquinas mecánicas y digitales, objetos, animales o personas que están provistos de identificadores únicos y poseen la capacidad de transferir datos a través de una red sin necesidad de interacción humano a humano o humano con la computadora [2].

El principio funcional del Internet de las Cosas son las tecnologías máquina a máquina (M2M), que permiten la comunicación entre aparatos, captando información y convirtiéndola en acciones puntuales. Esto se refiere a las tecnologías que permiten que tanto los sistemas inalámbricos como por cable

puedan comunicarse con otros dispositivos de la misma capacidad. M2M utiliza un dispositivo (como un sensor o medidor) para capturar un evento (como la temperatura, nivel de agua, etc.), que se retransmite a través de una red hacia una aplicación (software), que traduce el evento capturado en información significativa [3].

La evolución que han tenido las IoT es a partir de la convergencia de las tecnologías inalámbricas, sistemas micro-electromecánicos, micro-servicios e internet. Generando una nueva vía que completa la evolución de las comunicaciones y la informática, aplicándola a los objetos, lo que facilita una mejor interacción con ellos. Esto se refiere a una red de objetos interconectados a través de internet [2].

Hoy en día con el crecimiento y mejora de la tecnología el IoT junto a los sistemas ciber-físicos, que son sistemas hardware-software controlados y monitoreados por algoritmos, forman parte de las bases para la llamada Industria 4.0. Revolución técnica-económica que se espera aumente su impacto en la siguiente década [4].

2.1.2. Fog Computing

La arquitectura de Fog Computing deriva de la arquitectura Cloud Computing como una extensión en que ciertas aplicaciones y procesamiento de los datos se realizan por parte de un dispositivo inteligente antes de ser enviados a la Nube. Gracias a esto se busca que los servicios de análisis, computación y procesamiento de los datos estén más cerca de su fuente de origen y también del usuario final disminuyendo así la latencia de envío y respuesta [5].

Esta arquitectura tiene como enfoque principal IoT funcionando como una capa entre la fuente de origen de los datos y la Nube que es el servidor final, y entre dispositivos y el gateway, teniendo como límite routers o los Edge Nodes (Nodos Edge) que son dispositivos ubicados en esta capa; por este motivo al Fog Computing también se le conoce como computación al borde (Edge Computing). Dicha jerarquía de elementos mencionada esta compuesta por

mínimo un dispositivo inteligente al que se le llama servidor fog o Nodo Edge capaz de realizar distintas tareas para enfrentar las tareas de manera abierta interoperable y con alto rendimiento [6, 7].

Fog Computing tiene una arquitectura horizontal que puede ser replicada en diferentes niveles de red y cuya aplicación termina ofreciendo los siguientes beneficios: [8, 9]

- **Cognición:** Conocimiento más profundo del flujo y análisis de los datos.
- **Adición de seguridad:** Nuevas políticas de ciber-seguridad, seguridad física y control específicas en cada nodo.
- **Análisis en tiempo real:** Cómputo en local de los datos.
- **Reducción de costos:** Al tener desplegado varios Nodos Edge hace un mejor aprovechamiento de los recursos físicos que se utilizan además de un menor consumo energético del servidor de la Nube.
- **Mejora de la latencia y disminución del ancho de banda:** Teniendo el cómputo cerca del origen la velocidad de respuesta se ve reducida. Eso permite enviar sólo un resumen o información relevante reduciendo el tráfico al servidor de la Nube.
- **Disponibilidad offline:** En los casos de cortes de la conexión a internet la arquitectura Fog Computing mantiene sus servicios a disposición ya que mucho de su trabajo es en la red interna.

Las arquitecturas que incluyen Fog Computing aceleran el procesamiento de datos y la respuesta a los eventos al eliminar un viaje de ida y vuelta a la Nube para su análisis. Además, evita la necesidad de costosas ampliaciones de ancho de banda al descargar grandes cantidades de giga bytes de tráfico de la red central. También protege data sensible analizándola dentro de la red local. En última instancia, las organizaciones que adoptan el Fog Computing obtienen información más profunda y rápida, lo que aumenta la agilidad del negocio, aumenta los niveles de servicio y mejora la seguridad [7].

2.1.3. Publicación-Suscripción

Es un patrón de mensajería en donde los proveedores de información (publicadores) no tienen conexión directa a los consumidores específicos de esa información (suscriptores) pero las interacciones entre publicadores y suscriptores son controladas por un intermediario de la publicación y suscripción llamado Broker. En un sistema de publicación/suscripción, el publicador no precisa de saber quién recibe la información (publicación) que proporciona, y a su vez un suscriptor no necesita saber quién proporciona la información que recibe de su suscripción [10].

El flujo de la información puede ser basado en dos aspectos. Uno es basado según el tópico o tema en este los mensajes son publicados con un tópico y todos los que estén suscritos a dicho tópico recibirán la información del Broker. El segundo es basado según el contenido del mensaje, así los mensajes que son enviados al suscriptor son según ciertas condiciones en el contenido o en los atributos del mensaje [11].

Este patrón proporciona una mayor escalabilidad de la red y una topología de red más dinámica, con una resultante flexibilidad para modificar al publicar y la estructura de la información que se publica [10]. Este patrón ha sido tomado en cuenta como la base para el tipo de mensajería asíncrona por grandes empresas como IBM, Microsoft y Facebook.

2.1.4. Procesamiento Complejo de Eventos

El procesamiento de eventos complejos o por sus siglas en inglés CEP (Complex Event Processing) es un modelo que analiza grandes flujos de datos que pueden ser de múltiples orígenes según el tiempo en que suceden permitiendo responder y tomar acciones rápidamente a un bajo costo computacional [12]. Este modelo es utilizado en múltiples campos como monitoreo de la actividad comercial, redes de sensores, datos de mercado, etc [13].

El modelo CEP gira en torno al concepto clave de ver los eventos que ocurren en el mundo externo como notificaciones de flujo de información interminables. Estos eventos pasan a través de un proceso de filtrado que luego tienen que ser combinados entre sí para entender lo que está sucediendo; estas combinaciones se denominan eventos de nivel superior. El propósito de este modelo es notificar a los interesados los patrones especiales que ocurren en estos eventos de alto nivel, que se derivan de eventos de nivel inferior [14].

A diferencia de los tradicionales Sistemas de Gestión de Bases de Datos, por sus siglas en inglés DBMS, en los que se ejecuta una consulta en datos almacenados, CEP ejecuta datos en una consulta almacenada. Con esto todos los datos que no son relevantes para la consulta pueden descartarse inmediatamente. Este enfoque muestra varias ventajas, dado que las consultas CEP se aplican en un flujo de datos potencialmente infinito con un consumo mínimo de recursos. Una vez que el sistema ha visto todos los eventos para una secuencia coincidente, los resultados se emiten inmediatamente. Este aspecto conduce de manera efectiva la capacidad analítica en tiempo real del CEP [15].

2.2. Trabajos Relacionados

Los trabajos relacionados al Fog Computing son diversos y cada día salen más ya que todos están ligados al IoT. En esta sección se presenta tres trabajos con casos de uso actuales y un trabajo de investigación centrado en la energía consumida, todos exponiendo lo beneficioso de las plataformas Fog Computing bajo sus diferentes puntos de vista y condiciones.

2.2.1. Juego Online

En [16] los autores trabajan sobre una versión online del popular juego Pokemon Go llamado iPokeMon. En este artículo se particionó el trabajo teniendo como Data Center una máquina virtual de Amazon situada en Dublin y como Nodo Edge a un Odroid XU+E, tarjeta de desarrollo, en su ciudad local de Irlanda.

La partición de tareas que menciona se da de manera que el servidor en la Nube mantiene una vista global de los Pokemons, mientras que el Nodo Edge tiene una vista local de los usuarios que estaban conectados a este. El Nodo Edge actualizaba periódicamente la vista global del servidor de la Nube. Teniendo como resultados una disminución del 20 % en el tiempo promedio de respuesta y un reducción del 90 % en el ancho de datos enviados al servidor. Dichas reducciones de la frecuencia de comunicación en un caso como este dan lugar a una mejora en su calidad de servicio y experiencia del juego con el usuario.

2.2.2. Monitoreo de la Salud

El documento [17] es un trabajo doctoral que describe una arquitectura Mobile Computing, un derivado del Fog Computing en el que el Nodo Edge es un smartphone, junto con un prototipo para la monitorización remota de pacientes en el que existe un sistema con motor de procesamiento de eventos complejos (CEP); es decir, el sistema realiza el análisis y la detección de eventos complejos en el dispositivo móvil y envía los resultados a un servidor de back-end del hospital para su posterior procesamiento.

Al aprovechar la gran capacidad de cómputo de los celulares inteligentes de hoy en día, el autor demuestra la viabilidad de todo su sistema y aplicación móvil al reducir la carga de trabajo en los servidores del hospital, además de la reducción de latencia para un patrón de prueba. Con lo que finalmente termina proponiendo unos puntos que podrían servir de guía para trabajos similares con el motor CEP en Mobile Computing.

2.2.3. Fabricación de Tuberías

Esta aplicación es una de las tantas en la emergente Industria 4.0 que aprovecha la tecnología para ofrecer mejoras en las áreas de producción gracias a indicadores en tiempo real que sirven para crear mejores planes administrativos y logísticos. El trabajo realizado en [18] toma el caso en una

fábrica constructora de tuberías teniendo la aplicación del Fog Computing en un sistema cyber-físico capaz de detectar eventos relacionados al movimiento y fabricación de las tuberías y así decidir si enviar notificaciones al sistema o a los clientes.

Como es típico utilizan una arquitectura de capas, una en donde se ubican ciertos sensores y actuadores con emisores de radiofrecuencia, la capa fog es la intermedia se encuentra basada en tarjetas Orange Pi y se divide en sub módulos siendo uno de estos el de Business Intelligence que detecta los eventos y envía notificaciones.

Finalmente, la implementación del Fog Computing le ofrece respuestas en promedio más rápidas por la reducción de la latencia para con los eventos detectados ofreciendo además la capacidad de analizar más datos que en este caso aumentaría la producción de tuberías. No obstante hacen mención que su trabajo es bajo las condiciones del lugar donde se desarrolló las pruebas por lo que no se puede generalizar todos sus resultados.

2.2.4. Ahorro de energía al Cloud Computing

Para terminar, en el trabajo de [19] los autores se enfocan en el consumo energético de Data Centers con arquitectura Cloud Computing VS nano Data Center con Fog Computing, siendo este último implementado con Raspberry Pis. A lo largo del documento toma en cuenta varios factores en son de plantear las condiciones en que es favorable una arquitectura sobre otra. Para eso se realizan varios test como cargas de páginas web estáticas, aplicaciones con contenido dinámico y videovigilancia, y de carga de multimedia estática para videos en demanda. Algunas de las condiciones sobre las que trabajó fue variantes en el tipo de la red de acceso, el tiempo inactivo-activo de los nodos, número de descargas por usuario, etc.

En sus conclusiones termina resumiendo que bajo la mayoría de condiciones las plataforma con Fog Computing muestran indicadores favorables en reducción de la energía consumida, sin embargo en unos cuantos casos se ve lo contrario. Debido a eso ultima que para lograr aprovechar las ventajas del

Fog Computing se realiza identificando las aplicaciones cuya ejecución en dicha plataforma realizan un consumo eficiente de la energía en todo el sistema.

Capítulo 3

Caso de Uso y Tecnologías a Usar

En este capítulo se presenta el esquema general para el caso de uso del proyecto de una red de sensores con Nodos Edge, seguido de la topología de red de conexión de los usuarios móviles según su ubicación. Prosiguiendo finalmente a explicar todas las tecnologías y herramientas a usar en el desarrollo del trabajo.

3.1. Descripción y Diseño del Caso de Uso

El caso en el que se trabajó fue un proyecto en el que se crearon múltiples módulos de sensores medioambientales embebidos, cada uno de estos módulos llega a medir en promedio hasta 25 parámetros distintos y su comunicación es bajo el protocolo XBee en una semi red mesh hasta que llega al microcontrolador coordinador. Hasta este punto queda claro que dicho microcontrolador es el Nodo Edge encargado de validar, analizar y enviar la información de estos end-point a la Nube. Para concluir los usuarios de la plataforma son aplicativos móviles Android.

3.1.1. Topología de Redes en el caso de Uso

Como se explicó en la sección 2.1.1 la capa donde se ubica el Nodo Edge trabaja como una intermedia entre los end-points y la Nube. En este trabajo y según el caso de uso descrito el Nodo Edge está ubicado en la capa Fog y sus límites son el alcance de su LAN teniendo al gateway como el puente para su

envío a la Nube como se muestra en la Figura 3.1. Para este trabajo se especificó que limitar con el gateway es limitar con el Access Point pues el Nodo Edge tendrá acceso a Internet por red Wifi. Punto crucial pues varias tareas que este realice son en base a estas consideraciones, una de ellas será cuando el usuario final se encuentre en la misma LAN. El PAN es la red mesh creada por los módulos de sensores bajo el protocolo XBee perteneciendo a la capa inferior de Origen de Datos.

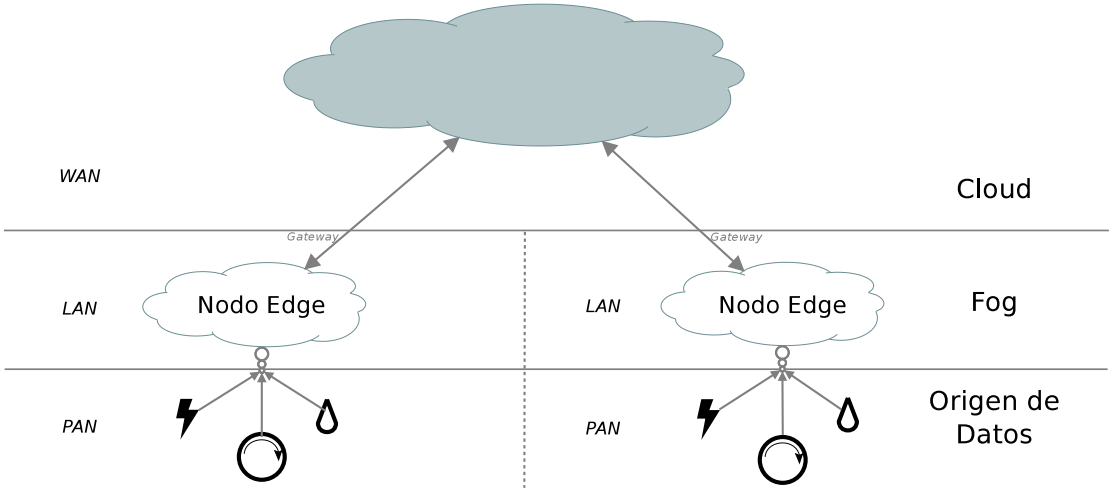


FIGURA 3.1: Topologías del caso de Uso. Fuente: Elaboración propia

Los usuarios móviles de la plataforma podrán obtener la información en tiempo real de los sensores dependiendo de la ubicación en la que se encuentren, pudiendo estar en la misma LAN del Nodo Edge o por defecto en la WAN.

3.2. Tecnologías a usar

Antes de proceder a la implementación de la plataforma se mencionarán las características y beneficios de las tecnologías utilizadas, las cuales siguen la línea de ser open-source y open-hardware lo que les da una gran comunidad de usuarios que la respaldan con bastante documentación.

3.2.1. Raspberry Pi 3 Model B+

El Raspberry Pi 3 Model B +, el último producto de la gama Raspberry Pi 3, es un computador de placa simple (SBC) que cuenta con un procesador de cuatro núcleos de 64 bits que funciona a 1,4 GHz, doble banda 2.4 GHz y 5 GHz LAN inalámbrica, Bluetooth 4.2 / BLE, Ethernet más rápido además de poseer un disipador de calor [20].

Esta placa es un mini computador silencioso (no produce ruido) y al que se le puede instalar distintas distribuciones de GNU/Linux como sistema operativo (SO). Toda la información que se almacenada se hace en una tarjeta Micro SD, en esta se incluye el SO. Esta versión de Raspberry utiliza un bajo consumo de energía de entre 5 y 7 Watts y además posee un comunidad activa que apoya y ofrece información del hardware y software [21].

En adición, las pruebas en [22] demuestran que las tarjetas Raspberry Pi son altamente eficientes cuando manejan bajos volúmenes de tráfico de red. Sus resultados avalan lo útil que son en la ejecución de aplicaciones ligeras orientados al IoT como son el protocolo de aplicación restringida (CoAP) y los protocolos de transporte de telemetría de MQTT.

Con lo anterior expuesto, el Raspberry Pi 3 B+ posee las características necesarias y suficientes para ser el Nodo Edge de nuestra plataforma en el que se enfocará este trabajo y al que se le instalará las siguientes herramientas.

3.2.2. Mosquitto MQTT

MQTT (Message Queue Telemetry Transport) es un protocolo usado para la comunicación M2M en el Internet of Things. Este protocolo está orientado a la comunicación de sensores, debido a que consume muy poco ancho de banda, se adapta fácil a diferentes niveles de latencia y puede ser utilizado en la mayoría de los dispositivos embebidos con pocos recursos. Un ejemplo de uso de este protocolo es la aplicación de Facebook Messenger tanto para Android y Iphone [23].

La arquitectura de MQTT sigue una topología de estrella basada en el paradigma de mensajería publicación-suscripción, en el que un nodo central hace de servidor o Broker como se muestra en la Figura 3.2. El Broker es el encargado de gestionar la red y de transmitir los mensajes, para mantener activo el canal, los clientes mandan periódicamente un paquete (PINGREQ) y esperan la respuesta del Broker (PINGRESP) [1].

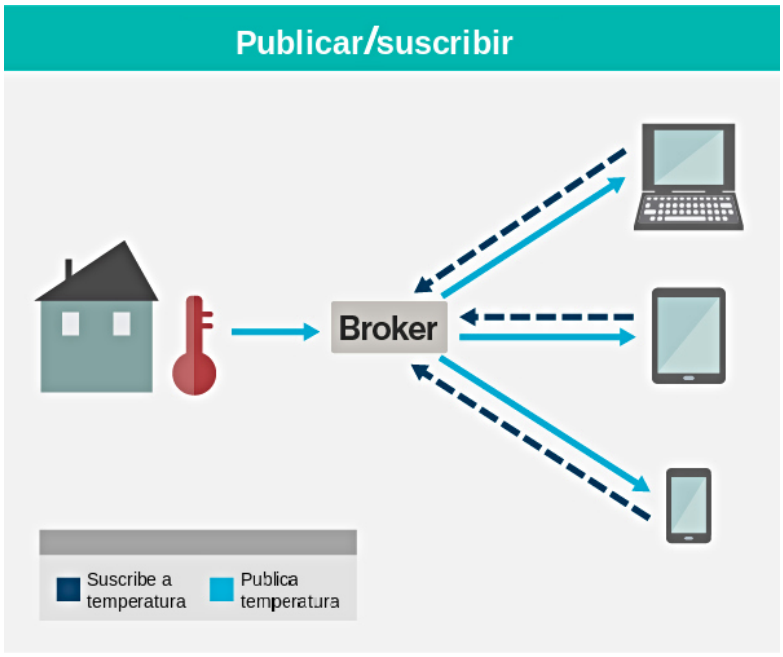


FIGURA 3.2: Publicar y Suscribir. Fuente: Traducido de internetofthingsagenda.techtarget.com [1]

Una de las opciones de seguridad que utiliza MQTT es la autenticación que se presenta como un nombre de usuario y contraseña de texto plano que es enviado por el cliente al servidor como parte de la secuencia de paquetes CONNECT/CONNACK [1]. La otra opción es a partir de llaves pública y privada trabajando de la misma manera.

Finalmente, para este trabajo se utilizó Eclipse Mosquitto que es un proyecto open-source (con licencia EPL/EDL) que implementa la versión 3.1 y 3.1.1 del protocolo MQTT [24]. Durante las pruebas se utilizó la seguridad de usuario y contraseña en texto plano.

3.2.3. Apache Flink-CEP

Apache Flink es un framework open-souce para cálculos de estado sobre flujos de datos ilimitados y limitados. Flink esta diseñado como sistema distribuido que puede ejecutarse en todos los entornos de clúster comunes, lo que facilita desplegar aplicaciones a gran escala aprovechando el mejor rendimiento de memoria. Además a estas ventajas Flink, que actualmente está en su versión 1.8.0, posee una comunidad activa que da soporte a sus múltiples librerías entre ellas la que permite trabajar con cargas CEP [15].

Durante el entorno de ejecución en Apache Flink se crean dos tipos de procesos en la JVM. Uno es el *Jobmanager*, también llamado el máster, encargado de coordinar la ejecución distribuida, asignación de tareas, manejo de fallas, etc. El otro es el *Taskmanager*, también llamado trabajador, encargado de ejecutar las tareas asignadas por el Jobmanager sobre el flujo de datos. Durante la ejecución tiene que haber al menos un JobManager y un TaskManager. Adicionalmente cada TaskManager tiene al menos un *task slot* encargado de la realizar las subtareas, cada uno de estos hilos tiene una parte de la memoria del TaskManager [25].

FlinkCEP es la biblioteca de procesamiento de eventos complejos de Flink. Permite detectar fácilmente ciertos patrones entre distintos eventos complejos de un flujo de datos interminables llamado DataStream, dichos eventos pueden construirse a partir de secuencias coincidentes. Esto le da la oportunidad de obtener rápidamente lo que es realmente importante en sus datos [26].

3.2.4. Perf

Perf, también llamado *perf_events*, es una herramienta de análisis de rendimiento en Linux, disponible desde la versión del kernel 2.6.31 siendo en el 2012 reconocida por trabajadores de IBM como una de las herramienta de perfilado más utilizadas en distribuciones Linux [27].

El acceso a la herramienta Perf es desde la línea de comandos y proporciona varios sub-comandos, siendo capaz de realizar un perfil estadístico de todo el

sistema. El perfilado entre el Perf y el kernel consta de un solo syscall y se realiza a través de un descriptor de archivos y una región de memoria con formato mm. A diferencia de otros perfiladores este no necesita demonios de servicio, ya que la mayoría de las funcionalidades están integradas en el kernel [27].

Perf trabaja midiendo distintos eventos, como por ejemplo el counter del sistema al que se le llaman eventos de software, otros son los eventos del hardware basados en el propio procesador y su Unidad de Monitoreo de Rendimiento, que por sus siglas en inglés PMU, que proporcionan una lista para medir eventos de micro-arquitectura como son los ciclos, instrucciones, branches, etc. Estos varían con cada tipo de procesador y modelo. Existen otros dos tipos de eventos que son los de eventos de caché de hardware y eventos de rastreo que están disponibles según la versión del kernel [28].

Para este trabajo los sistemas operativos que se van a usar son basados en Linux por lo que esta herramienta se usará para exponer el performance según condiciones explicadas más adelante.

3.2.5. Top

Top es un programa de monitorización, administración y visor de procesos que se encuentra en muchos sistemas operativos de tipo Unix [29]. Proporciona una vista dinámica en tiempo real del sistema en ejecución mostrando información resumida del CPU así como una lista de procesos actualmente administrados por el kernel de Linux [30].

En la sección resumida del CPU se encuentra información general del uso de espacio de memoria RAM y swap, hora del sistema y cantidad de sesiones usuario abierta, estados y cantidad de procesos, porcentajes del uso del CPU entre varios procesos y finalmente el promedio de carga [31].

Top al ser más popular hoy en día y al ofrecernos información resumida del estado del sistema también se utilizará para medir el performance en la plataforma.

3.2.6. Android

El sistema operativo Android actualmente es supervisado por Google bajo la licencia de open-source. Android posee una basta documentación y librerías que apoyan el desarrollo de nuevos aplicativos con nuevas y mejores capacidades a favor de los usuarios.

Android basado en el sistema operativo Linux y otros software de open-source [32] posee mayor facilidad para poder crear aplicaciones móviles en comparación con el sistema operativo iOS el cual se necesita una computadora con MAC para desarrollar sus aplicativos. Esa es la principal razón por la cual se escogió empezar con aplicativos Android para las primeras versiones de la plataforma.

Si bien es cierto en Mayo del 2017 Google anunció soporte para la utilización de Kotline como su lenguaje de programación oficial para Android [33] eso no restringe ni prohíbe el desarrollo aplicaciones en el lenguaje Java además que la documentación de librerías externas están desarrolladas en el lenguaje Java y para su posterior migraciones tomará varios años.

Capítulo 4

Arquitectura y Diseño de la Plataforma

El objetivo de este capítulo es ofrecer un marco del esquema y del diseño que se implementó en el Nodo Edge con la arquitectura Fog Computing para el caso de uso, además de mostrar una comparativa con el flujo de la información con Cloud Computing. Terminando con el procedimiento de cómo se utilizaron las tecnologías que se mencionaron en el capítulo anterior.

4.1. Mosquitto como pilar de Comunicación

Teniendo en cuenta que la plataforma a desarrollar es parte del IoT se utilizó Mosquitto en su versión 1.5.5. Antes de explicar quienes son los Brokers, publicadores y suscriptores se hablará de la seguridad implementada.

4.1.1. Usuarios y permisos

Como se mencionó en la Sección 3.2.2 el Broker es el encargado de gestionar el flujo en la red y de transmitir los mensajes. Este puede ser configurado para admitir conexiones específicas y además otorgar permisos de publicación o suscripción.

La Tabla 4.1 especifica los usuarios configurados en el Broker Mosquitto y los permisos otorgados. Para la tercera fila XBEE_MAC como su nombre lo indica es el MAC del XBEE único en cada módulo de sensores y XBEE_MAC^{-1}

es el mismo texto sólo que con los caracteres al revés así como en la primera y segunda fila se expone el caso de CTIC-SMARTCITY y Raspberry. La X en la cuarta fila para AlarmX y XAlarm respresenta un número (no tiene ninguna especificación con las opciones de configuración de Mosquitto).

Usuario	**Contraseña**	**Suscribir**	**Publicar**	**Tópico**
CTIC-SMARTCITY	YTICTRAMS-CITC	SÍ	SÍ	#
Raspberry	yrrebpsaR	NO	SÍ	+/+/+
XBEE_MAC	XBEE_MAC^{-1}	SÍ	NO	%u/+/+
AlarmX	XAlarm	SÍ	NO	%u/ %u/ %u

TABLA 4.1: Usuarios y permisos en el Broker.

En Mosquitto, para un usuario suscrito al tópico # significa que no tiene ninguna restricción para publicar o suscribirse a cualquier tópico en el Broker; el símbolo +/+/+ significa que el tópico consta de tres niveles divididos por barras, como por ejemplo *0013A200716C0E50/BME280/temperature* así cada + representa un texto plano. El *%u*/+/+ quiere decir que el primer nivel coincide en caracteres con el nombre del usuario. Mosquitto posee más opciones para configurar respecto a la seguridad como la encriptación por llaves privaba-pública sin embargo se optó sólo por utilizar las expuestas en el cuadro.

A continuación se procede a explicar a más detalle cada *Usuario* de la tabla con su función y motivo según el caso de uso.

- El usuario CTIC-SMARTCITY no tiene ninguna restricción para suscribirse y/o publicar en el Broker. Este súper usuario será utilizado en la Nube tanto para almacenar la información que llegue como para publicar una información relevante a futuro como alarmas.

- El usuario Raspberry sólo puede publicar un mensaje cuyo tópico sea de tres niveles. Dichos niveles definen el origen de la información que se explicará a más detalle en los siguientes puntos. Este usuario se usará exclusivamente en el Nodo Edge.

- Los usuarios XBEE_MAC sólo pueden suscribirse a tópicos de tres niveles teniendo como primer nivel el mismo *XBEE MAC* que es el identificador del módulo de sensores end-point, el segundo nivel es el *identificador del sensor* y el último es el tipo de parámetro que mide dicho sensor, la razón de estos dos últimos niveles es debido a que existen casos en que un sensor mide varios parámetros y estos parámetros pueden coincidir con otro sensor en el mismo módulo. Por ejemplo, para el módulo cuyo MAC XBEE es *0013A200716C0E50* su contraseña es *05E0C617002A310*, un ejemplo de tópico que podrá utilizar es *0013A200716C0E50/BME280/temperature* donde *BME280* es el identificador del sensor y temperature es el parámetro medido, así se busca priorizar el orden y coherencia de la información. El objetivo de los usuarios XBEE_MAC es el de sólo poder obtener la información de los sensores de dicho módulo, y será utilizado como suscriptor en el usuario móvil final para el aplicativo Android.

- Similar caso para los usuarios AlarmX donde X es un número entero que comienza desde el número 0. Pueden suscribirse a tópicos cuyos tres niveles son el mismo identificador de la Alarma. Ejemplo, para el usuario *Alarm1* su contraseña es *1Alarm* y podrá suscribirse al tópico *Alarm1/Alarm1/Alarm1*. Y serán utilizados por el usuario móvil en el aplicativo Android como suscriptor.

Finalmente Mosquitto maneja las conexiones a partir de un ID; es decir que si alguien está conectado, ya sea para publicar o estar suscrito, con un ID otro nuevo que se conecte con ese ID no podrá ni suscribirse ni publicar a pesar de tener los permisos de usuario para el Broker.

4.1.2. Broker, Publicador y Suscriptor

Esta sección queda descrita en dos casos según la red de conexión en la que se encuentre el suscriptor que es el usuario móvil final, el cual recibe los datos enviados del Broker.

Suscriptor en la WAN

Este es el caso obvio y base. En la Figura 4.1 queda descrito como el Suscriptor A*, este usuario final sólo puede recibir información a través de la WAN ya que su único contacto con el origen de los datos es a través de la Nube. Por lo tanto, la Nube es el Broker (Broker global) y el Nodo Edge es el publicador. Así, el flujo de información parte desde la publicación en el Nodo Edge hacia el Broker global y de ahí es enviado hacia el suscriptor.

Suscriptor en la LAN

En este caso especial el Nodo Edge es el publicador y a la vez es el Broker (Broker local); esto quiere decir que el Nodo Edge se publica a sí mismo y el usuario final, previamente suscrito, recibe la información de él, así como se muestra en la Figura 4.1 para el caso del Suscriptor B*. De esta forma se aprovecha que el usuario final puede recibir la información directa desde el nodo que se encuentra más cerca del origen de los datos evitando así que la información viaje hasta la Nube y luego regrese a otro punto de la LAN inicial.

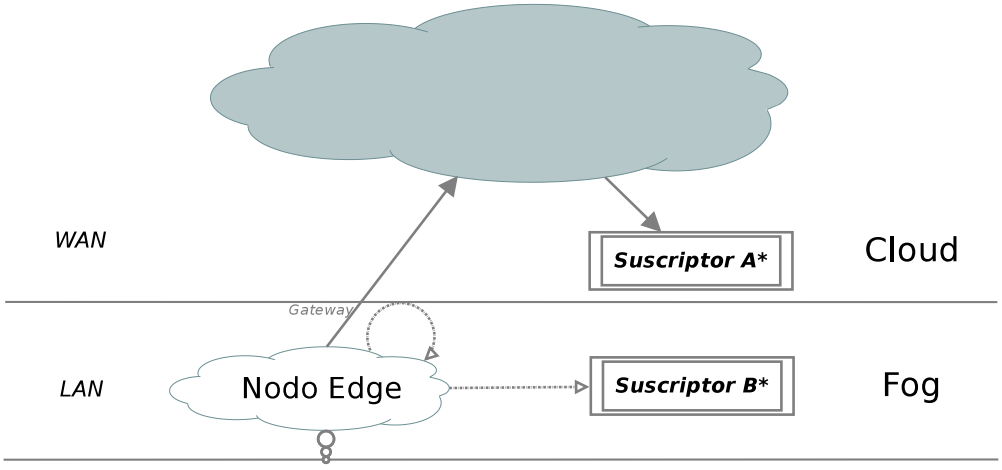


FIGURA 4.1: Suscriptores MQTT. Fuente: Elaboración propia.

Para ambos casos descritos anteriormente el Nodo Edge publicará con el usuario *Raspberry* en un tópico cuyo formato de tres niveles pueda ser suscrito por los usuarios XBEE_MAC o AlarmX como se mencionó a detalle en el punto

anterior. Cabe mencionar, que tanto el Nodo Edge como el servidor de la Nube tienen la misma configuración de seguridad en su Broker.

4.2. Detección de Alarmas con FlinkCEP

El framework Apache Flink junto a su librería *FlinkCEP* permite en tiempo real analizar y detectar de un flujo de datos, patrones o eventos de interés que para nuestro caso generarán una alarma. Flink maneja dicho flujo en una sucesión de 3 fases:

- **Lectura del Flujo:** Esta primera fase, también llamada Input, es la responsable de brindar la fuente de datos a analizar. Esta fuente puede establecerse de distintos orígenes como un socket u otro API, por lo que para este trabajo se decidió utilizar el API de Mosquitto con el que se utiliza un cliente MQTT para aprovechar la publicación en local de todos los valores medidos por los sensores.

- **Análisis del Flujo:** Fase también conocida como Task, es la que analiza todo el flujo de datos entrante dividiendo y realizando la lógica a los eventos (valores enviados desde los módulos). En esta fase se utiliza la librería *FlinkCEP* para el análisis de los patrones que generan la alarma en un tiempo determinado, es decir, se analiza a partir de condiciones de valores y tiempo de llegada para comprobar que se ha generado una alarma.

- **Acción:** Esta fase es la salida, una vez que la fase anterior se ha cumplido con éxito Flink da la libertad de ejecutar una acción (función o clase) para así crear un nuevo flujo de datos con valores de salida. Para nuestro caso una vez que se ha cumplido el patrón se realizará una nueva publicación con su tópico respectivo notificando la alarma.

Un punto a resaltar en esta sección es que se pretende derivar el motor CEP ya sea en el Nodo Edge o en la Nube. Sin embargo tampoco se descarta

que exista un motor CEP en ambos ya que cada uno podría analizar patrones distintos según lo que se pretenda detectar.

4.3. Nodo Edge

Como ya se tiene mencionado el candidato a ser el Nodo Edge es el Raspberry Pi 3 modelo B+, dicha tarjeta posee una gran capacidad de cómputo que se desea aprovechar al máximo para no sobrecargar al servidor de la Nube, además de disminuir la latencia de análisis y respuesta. En vista de esto y del caso de uso este Nodo Edge se encarga de realizar las siguientes tareas:

- **Validación de la información:** Los valores medidos por los módulos están en un formato de texto plano divido por el caracter # así como se muestra a continuación:

  0013A200716C0E50#BME280#temperature#27.51

  El primer campo es el identificador del módulo, seguido del identificador del sensor, el parámetro y finalmente su valor medido. Una condición más que se hizo al momento de verificar fue que el valor medido tiene que ser mayor igual a cero (≥ 0) , esto por documentación y característica de ciertos sensores. Finalmente ni bien validado los campos se procede a crear el mensaje JSON a publicar que tiene en su contenido el valor medido junto con su correspondiente *timestamp* que es el tiempo actual del Raspberry, la demás información será utilizada para el tópico.

- **Envío MQTT a los dos Brokers:** Una vez que se tiene listo el JSON y el tópico que se va publicar, el Nodo Edge subsigue a publicar el valor medido tanto hacia el Broker global como al Broker local para los dos casos de la ubicación del suscriptor como se indicó en la Sección 4.1.2. Finalmente, del ejemplo de mensaje del punto anterior el tópico de la publicación es el siguiente:

  0013A200716C0E50/BME280/temperature

- **Análisis para Alarmas - CEP:** Aprovechando la publicación en local esta se toma como fuente de origen de eventos para analizar en el motor CEP y cuando se tenga que publicar la alarma detectada, al igual que los valores medidos, se hará en el Broker global y local. La información a detalle del funcionamiento del motor ya fue explicado en la sección anterior.

La Figura 4.2(a) es el diagrama de flujo que detalla las actividades que se realizan en el Nodo Edge para la arquitectura Fog Computing mencionadas en esta y la anterior sección. La Figura 4.2(b) es un contraste de como sería en una arquitectura normal Cloud Computing.

Las diferencias en ambos flujos radica primero en la ubicación del módulo CEP para detección de alarmas ya que como se aprecia en la Figura 4.2(a) esta es parte del Nodo Edge y en la Figura 4.2(b) es parte de la Nube. La segunda diferencia es en el tipo de publicación pues para el caso de Fog Computing se hace una doble publicación tanto al Broker local como al global, mientras en la figura del flujo en Cloud Computing es una publicación solo al Broker global.

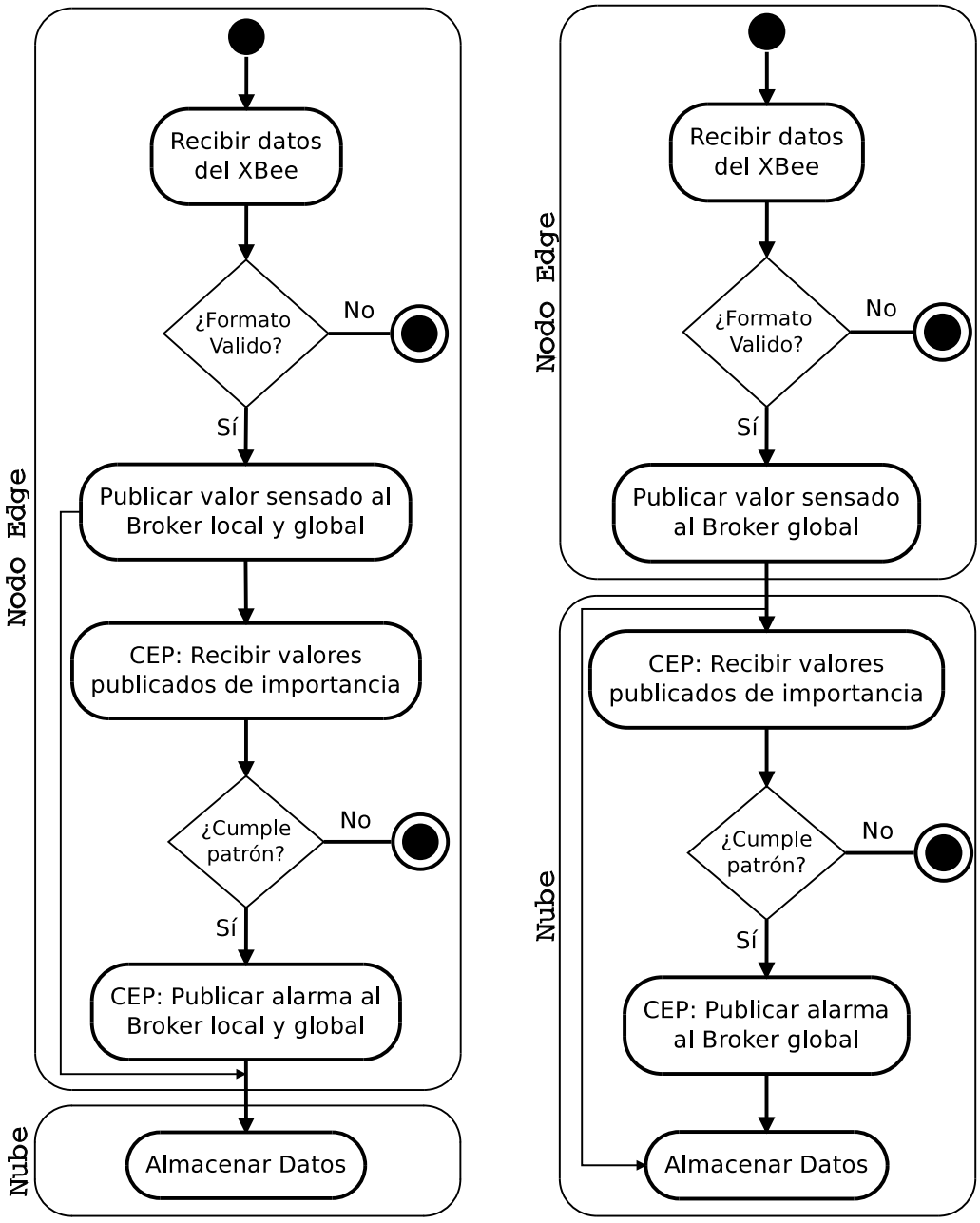
4.4. Usuario dentro de la LAN/WAN

El objetivo principal de este punto es aprovechar una de las ventajas de Fog Computing que es reducir la latencia de los mensaje publicados que son los valores medidos por los módulos y alarmas detectadas hacia los usuarios que estén suscritos en la misma red (LAN).

Este caso ya fue descrito en gran parte por la Figura 4.1 y su respectiva sección, sin embargo, se mencionará como discernir entre un usuario que se encuentre en la misma LAN del Nodo Edge a uno que no se encuentra en ella, definiendo a que Broker se suscribirá.

Lo primero es tener claro qué condición se tiene que cumplir para asegurar que un usuario móvil se encuentre en la misma LAN que un Nodo Edge. Para este caso de uso el Nodo Edge tiene conexión a Internet a través de una red Wifi provista por un modem-router. Según esto último para que un usuario móvil

se encuentre en la misma LAN del Nodo edge tiene que cumplir la condición de tener el mismo Access Point.



(a) Fog Computing (b) Cloud Computing

FIGURA 4.2: Diagramas de Flujo Fog Computing vs Cloud Computing para un valor medido. Fuente: Elaboración propia.

Si dos dispositivos con conexión Wifi poseen el mismo Access Point entonces pueden alcanzarse (ping) a través de su IP local. Por lo tanto cuando el suscriptor tenga el mismo Access Point deberá suscribirse al IP local del Broker

local que es el Nodo Edge. En caso contrario se suscribirá al IP de la Nube, Broker global.

4.5. Aplicación en Android

Como ya se vino exponiendo a lo largo de este capítulo el usuario final es un Aplicativo en Android, esta app tiene como objetivos los siguientes puntos:

- Graficar en tiempo real la llegada de los nuevos valores medidos por un sensor.
- Recibir una notificación cuando se haya producido una alarma.
- Suscribirse al IP adecuado según la red de conexión en la que se encuentre.

El primer punto se resolverá gracias a una librería llamada *MPAndroidChart* la cual permite añadir nuevos elementos a los puntos ya graficados.

Para cubrir el segundo punto se utilizó una clase que extiende de la clase *Service* que permite ejecutar servicios en segundo plano la cual iniciará la suscripción a la alarma y cuya acción frente a la llegada de un mensaje será la de mostrar una notificación.

Para el último punto influye tanto en el primer y el segundo ya que según se encuentre la red de conexión del usuario final este tiene que suscribirse al IP correspondiente. Con esto se refiere tanto a la suscripción de la alarmas como de los valores medidos. Para realizar dicho trabajo se hace en 4 pasos:

1. Verificar que está conectado por Wifi.
2. Obtener la MAC del Access Point.
3. Asignar el IP correspondiente a los módulos y alarmas a suscribirse. Tener en cuenta que el hecho de que el usuario final esté en la LAN de un Nodo Edge no quiere decir que en ese estén todos los módulos y alarmas.
4. Reiniciar las suscripciones.

Capítulo 5

Implementación de la Plataforma y Configuraciones

En este capítulo se explica la diferencia de un usuario de la plataforma con un usuario del Broker como un previo para luego especificar los detalles del módulo CEP. Estos detalles van desde las configuraciones realizadas hasta la implementaciones utilizadas en el siguiente capítulo. Este capítulo culminará con los tipos de publicaciones utilizadas y las vistas resultantes en la aplicación Android.

5.1. Usuario Plataforma vs Usuario Broker

Lo primero a tratar es respecto a que los usuarios finales de la plataforma no son los mismos que los usuarios del Broker explicados en la Sección 4.1.1.

Un usuario final de la plataforma puede recibir los valores medidos en tiempo real de varios sensores además de distintas notificaciones de alarmas generadas. Por lo tanto tiene la información para poder suscribirse a distintos usuarios MQTT del Broker como los de los XBEE_MAC y las AlarmX expuestos en la Tabla 4.1.

En la implementación de la plataforma luego de haberse logueado el usuario de la plataforma recibe JSONs similares a la Figura 5.1. El JSON de la izquierda contiene toda la información relevante de los módulos de sensores en la que están los identificadores de los módulos, el IP de su Nodo Edge, su respectivo Access Point y la información de los tipos de sensores. A su vez el JSON de la

derecha contiene la información relevante para las alarmas en la que además está presente su respectivo tópico.


```
{ "module_id":"0013A200416C0E50",
  "fogip": "192.168.20.75",
  "accessPoint": "42:df:bf:8a:c7:15",
  "sensors":[
      {"id_sensor":"BME280",
      "parametro":"Temperatura",
      "unit":"°C"
      },
      {"id_sensor":"HIH6130",
      "parametro":"Humedad",
      "unit":"%"
      },
      {"id_sensor":"VEML6075",
      "parametro":"UV_Index",
      "unit":"Indice"
      },
      {"id_sensor":"SPEC",
      "parametro":"H2S",
      "unit":"ppb"
      },
      {"id_sensor":"SPEC",
      "parametro":"CO",
      "unit":"ppb"
      },
      {"id_sensor":"SPEC",
      "parametro":"O3",
      "unit":"ppb"
      }
   ]
}
```

```
[
  { "id_Alarma": "Alarm1",
    "topic": 0013A200416C0E50/Alarm1/Alarm1",
    "fogip": "192.168.20.75",
    "accessPoint": "42:df:bf:8a:c7:15"
  },
  { "id_Alarma": "Alarm2",
    "topic": "0013A20012345678/Alarm2/Alarm2",
    "fogip": "192.168.1.105",
    "accessPoint": "84:C9:B2:A6:76:48"
  }
]
```


FIGURA 5.1: Ejemplos JSONs obtenido por Usuario de la Plataforma. Fuente: Elaboración propia.

5.2. CEP implementado

La implementación del framework Apache-Flink junto con su API *FlinkCEP* es una sección que tomó mucho estudio para poder configurar y desplegar un servicio que aproveche al máximo los recursos, sea ligero y realice los objetivos marcados en los capítulos anteriores.

5.2.1. Configuración

Previo a la programación del servicio se configuró el archivo *taskmanager.sh* ubicado en la carpeta bin de flink, recordando que el *Taskmanager* es el

encargado de realizar la tarea lógica a analizar en dicho archivo este tiene por defecto un tamaño máximo de pila de 8388607 TB. Este excesivo tamaño ocasiona un bajo performance en una computadora debido a que Apache-Flink se carga en mayoría sobre la memoria RAM debido a que está perfilado para clústers, sin embargo el caso de implementación busca ser ligero así que se asignó un tamaño de 512 MB.

Posteriormente a lo modificado, también se editó el archivo *flink-conf.yaml* ubicado en la carpeta conf para que los *taskslot*, hilos del *Taskmanager*, tengan un paralelismo igual a 50 en el Nodo Edge y 175 en el servidor de la Nube a fin de aprovechar los recursos de cada uno.

Dentro de la programación se estableció que el análisis de los eventos sea según su llegada al motor CEP que es el tiempo de procesamiento (*ProcessingTime* como se ve en el Código 5.1) de esta forma se tiene el beneficio de menor latencia y menor consumo de recursos para con la detección de las alarmas. Así mismo, también se puede analizar los eventos añadiendo un tiempo adicional a la espera de los eventos que lleguen con retraso (delay) o según su hora en el parámetro *timestamp* del JSON, aunque esta configuración no se realizó porque retrasaría el tiempo de análisis.

```
1 final StreamExecutionEnvironment env  = StreamExecutionEnvironment
2                 .getExecutionEnvironment();
3 env.setStreamTimeCharacteristic(TimeCharacteristic.ProcessingTime);
4 //Asignacion de la fuente de datos
5 DataStream<EventSensed> InputSensed = env.addSource(new MyMQTTSource());
```

CÓDIGO 5.1: Asignación de la forma de análisis en el CEP.

Luego de haber configurado el entorno de análisis y como se ve al final del Código 5.1 se asigna la fuente de origen del flujo de datos gracias a la clase que implementa la función *run* de *SourceFunction*. En esta función expuesta en el Código 5.2 se realiza la suscripción del *MqttClient* para la recolección de los eventos que sean relevantes para el patrón que se espera. Antes del fin de la función *run* se añadió un loop infinito para que quede suscrito durante el tiempo de vida del servicio CEP.

```
public static class MyMQTTSource implements SourceFunction<EventSensed> {
    public void run(SourceContext<EventSensed> ctx) throws Exception {
        MqttClient client
            = new MqttClient("tcp://localhost", MqttClient.generateClientId());

        client.setCallback(new MqttCallback() {
            public void messageArrived(String topic, MqttMessage mqttMessage) {
                JSONObject jsonObject
                        = new JSONObject(new String(mqttMessage.getPayload()));
                String timestamp = jsonObject.getString("timestamp");
                Double value= jsonObject.getDouble("value");
                //Recoleccion de eventos
                ctx.collect(new EventoSensed(value,timestamp,topic));
            }
        });
        client.connect(getOptionsToConnect());
        client.subscribe(getTopicToSuscribe());

        InfiniteLoop();
    }
}
```

CÓDIGO 5.2: Recolección de Eventos relevantes en el CEP.

5.2.2. Patrón CEP y Simulación

A continuación se presenta el patrón CEP implementado que se utilizará para analizar los eventos y notificar una alarma. Además de la simulación de eventos que servirá para el estudio de la latencia y performance expuestos en el siguiente capítulo.

La Figura 5.2 expone el patrón que se desea detectar. Este patrón consta de dos eventos cuyos valores medidos sean mayor al número 40 ($V > 40$) y sucedan en un lapso de 30 segundos. El key utilizado sirve como un filtro para distinguir entre los diferentes tipos de eventos que llegaron, este valor único es el tópico del evento ya que en este se encuentra la información del origen del mensaje. Es decir, por cada tópico analizado primero tiene que cumplir que llega un evento con valor medido mayor a 40 y si en el lapso menor a 30 segundos llega otro evento también con un valor medido mayor a 40 entonces se ha generado una alarma.

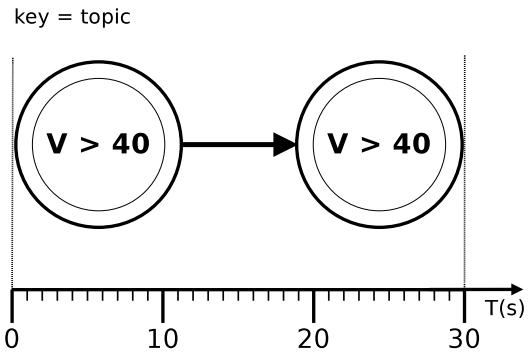


FIGURA 5.2: Patrón de Alarma. Fuente: Elaboración propia.

Teniendo definido el patrón a detectar se expone que la simulación utilizada para el siguiente capítulo consiste que para cada parámetro de sensor simulado, en su respectivo tópico, se realizará publicaciones con valores de 40, 41 y 42 con un lapso de 20 segundos entre cada uno. De esta manera se espera que por cada tópico que se publique se detecte una alarma cada minuto. Las pruebas siguientes serán de un total de 10 minutos de este loop.

La Figura 5.3 muestra un ejemplo de simulación hasta el segundo 130. Al inicio cuando llega al CEP el evento $V = 40$ este es descartado por no cumplir la condición. A la llegada del evento $V = 41$ este se almacena por cumplir el primer caso del patrón y los 20 segundos siguientes cuando llega el evento $V = 42$ se ha cumplido el patrón por lo que se publica la alarma. Para el segundo 50 el primer evento $V = 41$ queda descartado sin embargo el evento $V = 42$ queda almacenado hasta el segundo 70 ya que también cumple el primer caso del patrón. De esta forma se realiza las acciones de descarte de evento, retención de evento y detección de patrón.

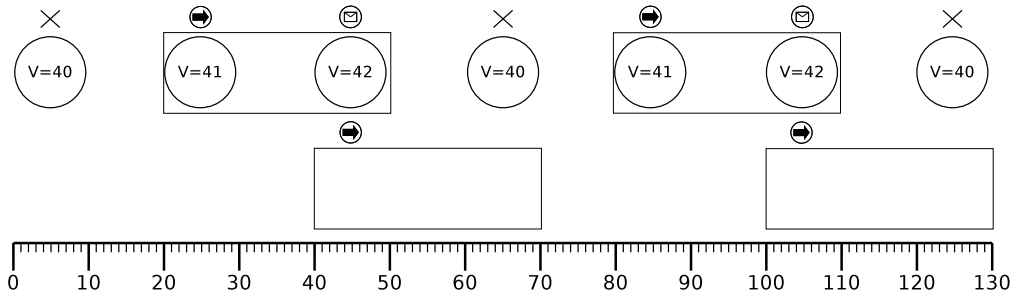


FIGURA 5.3: Simulación de Eventos. Fuente: Elaboración propia.

5.3. Paralelismo en la Publicaciones

El paralelismo mencionado en las publicaciones no sólo se refiere al momento de publicar tanto al Broker local como al Broker global, sino que también hace alusión al paralelismo entre nuevas publicaciones ya que estas tiene que ser mutuamente excluyentes. Todas las publicaciones se hace con un Quality of Service (QoS) de 0 de Mosquitto que es el más ligero. Así en este proyecto se tiene dos focos de publicaciones los cuales se trabajaron de distinta manera.

El primer foco es en la publicación de los valores medidos por los módulos. Como se aprecia en el Código 5.3 se trabajó en el lenguaje Python y se utilizó la función *publish.single* de la librería *paho-mqtt*. Dicha función se conecta al Broker, publica e inmediatamente se desconecta.

```
1  import paho.mqtt.publish as publish
2  import _thread
3  import json
4
5  def publicar(topico,mensaje,ip_server):
6      sem.acquire()
7      publish.single(topico, mensaje, hostname=ip_server,auth = myauth)
8      sem.release()
9
10 def FocoPublicar(topico,json_data):
11     #publico al Broker local!
12     _thread.start_new_thread( publicar, (topico, json_data, "localhost",) )
13     #publico al Broker global!
14     _thread.start_new_thread( publicar, (topico, json_data, IP_Nube,) )
```

CÓDIGO 5.3: Paralelismo de valores medidos desde el Nodo Edge.

El segundo foco se refiere a la publicación de las alarmas desde el motor CEP, este es un caso similar al anterior sólo que se utiliza un array de *MqttAsyncClient* tanto para las publicaciones en local como hacia la Nube. Dicho array primero realiza una conexión a su Broker respectivo y luego hace la publicación con esto queda a la espera de la llegada de otra alarma para su directa publicación con este cliente ya conectado. El Código 5.4 ejemplifica este foco para el mencionado cliente que publica en local, las funciones para el

cliente de la Nube se dejaron en puntos suspensivos (...) ya que tienen la misma estructura de publicar en local.

```
1  private static MqttAsyncClient clientLocal[] = new MqttAsyncClient[50];
2  private static MqttAsyncClient clientNube[] = new MqttAsyncClient[50];
3
4  public static void ConnectClient_Local(int hilo,String serverURI){
5      clientLocal[hilo] = new MqttAsyncClient(serverURI, MqttClient.generateClientId());
6      clientLocal[hilo].connect(getOptionsToConnect);
7  }
8
9  public static void PublicarLocal(int hilo, String nameAlerta, String mensaje){
10     MqttMessage messageMQTT = new MqttMessage();
11     messageMQTT.setPayload(mensaje.getBytes());
12     if(clientLocal[hilo]!=null){
13         semaphore.acquire();
14         clientLocal[hilo].publish(nameAlerta, messageMQTT);
15         semaphore.release();
16     }else {
17         semaphore.acquire();
18         ConnectClient_Local(hilo,"tcp://localhost");
19         semaphore.release();
20         PublicarLocal(hilo, nameAlerta, mensaje);
21     }
22
23 }
24
25 public static void ConnectClient_Nube(int hilo,String serverURI){ ... }
26
27 public static void PublicarNube(int hilo, String nameAlerta, String mensaje){ ... }
```

CÓDIGO 5.4: Paralelismo en publicaciones de Alarmas desde el CEP.

En ambos focos se utilizaron semáforos de tamaño 230 en la sección crítica de publicar, de esta manera se obtuvo mejor performance en cantidad publicada y menor latencia además que se evitaron errores de sobrecarga no poder conectarse al respectivo Broker o de publicar.

5.4. Aplicación Android

La aplicación fue realizada en Android-Studio con la versión de compilación 28 y se decidió que tenga un *NavigationView* para que el usuario tenga una mejor

experiencia por lo que las vistas para los gráficos se trabajó con *Fragments*.

5.4.1. MQTTClient y librería de gráficos

Para el servicio MQTT se utilizó las librerías de *org.eclipse.paho* en su versión 1.1.1 esta no da fallos al momento de cortar una suscripción y realizar una nueva, acción necesaria para cuando se necesite cambiar la IP suscrita cuando el usuario entre a la LAN del Nodo Edge o salga. Y en conjunto con la librería gráfica *PhilJay/MPAndroidChart* en su versión 3.0.3 dieron las siguientes vistas.

La Figura 5.4 muestra la vista del cambio en la gráfica cuando llega un nuevo valor medido. Nótose que en la tabla superior de la gráfica detalla la información del sensor.

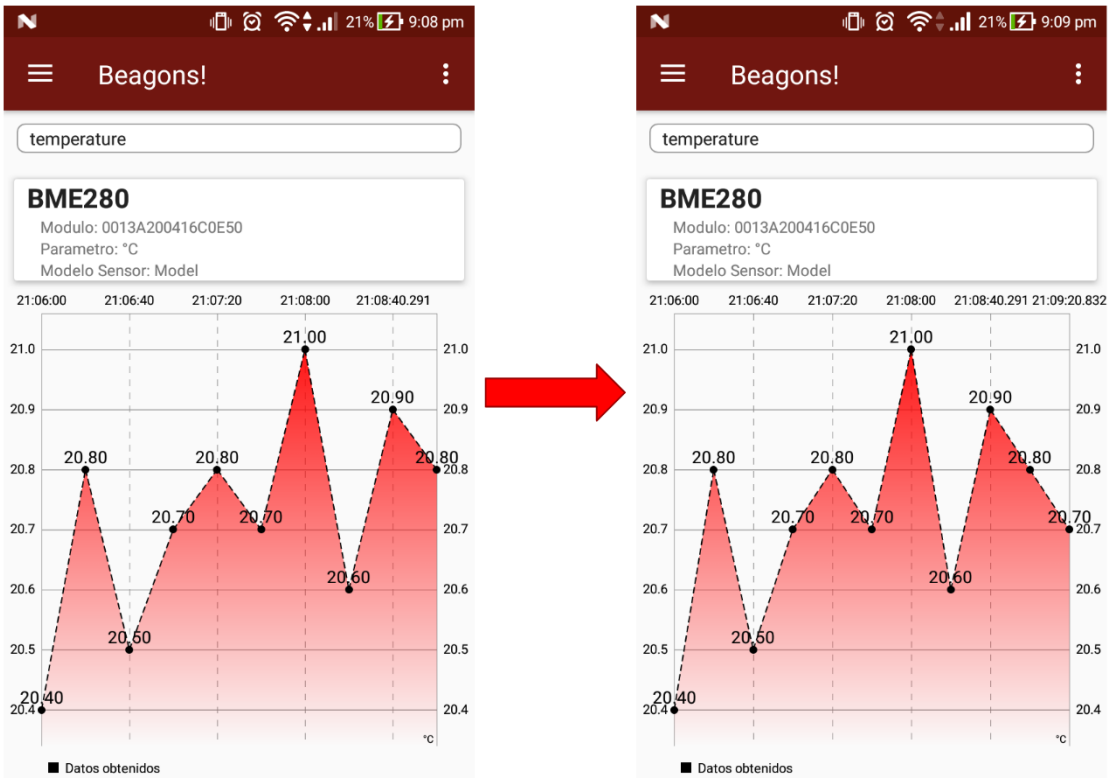


FIGURA 5.4: Vistas de la gráfica de valores en tiempo real. Fuente: Elaboración propia.

Para esta parte de actualización de la grafica de valores en tiempo real no se muestra su Código ya que es muy similar a los anteriores Java que se conectan y cuando llega un valor de la suscripción refresca la gráfica.

5.4.2. Servicios de Wifi y Notificaciones

Para la notificación de alarmas se creó un servicio al inicio de la actividad que crea el *NavigationView*. Este servicio crea todas las suscripciones correspondientes para cada alarma y cuando llega una, se crea una notificación como se ve en la Figura 5.5.

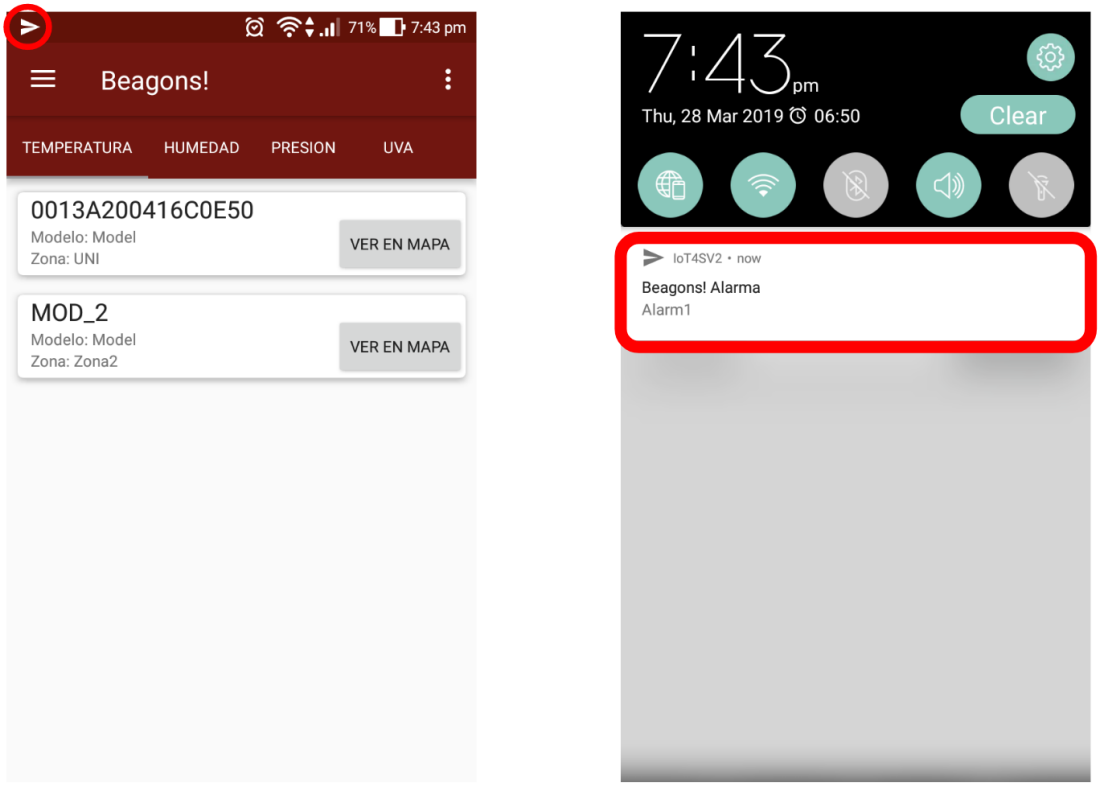


FIGURA 5.5: Vistas de la Notificación del llegado de una Alarma.
Fuente: Elaboración propia.

Un servicio adicional que se maneja es el de detección de cambio de conexión de Internet, esto se logra gracias a un objeto de la clase *BroadcastReceiver*. Este objeto al detectar un cambio de señal lanza un aviso, el cual permite obtener la información de si está conectado por Wifi o no, y si lo está obtener el BSSID (*Basic Service Set Identifier*) que es la MAC del Access Point. Para la obtención de dicha información se hace una espera prudente de 5 segundos debido a que la asignación de IP por parte del router, para con los dispositivos que se conectan, no es instantánea.

Bajo este contexto, el servicio original para la notificación de alarmas maneja la clase *BroadcastReceiver* como se ve en el Código 5.5 que al detectar que está conectado por Wifi actualiza los IPs a suscribir. Esta actualización se realiza a todos los módulos y alarmas que pertenecen a ese Nodo Edge por su correspondiente IP local o en su defecto el IP del Broker global (Nube). Terminando la ejecución anterior reinicia las suscripciones de las alarmas ya que estando en la LAN de un Nodo Edge o no, con el cambio de conexión a Internet, se tiene que realizar nuevamente la suscripción porque ya se perdió conexión de origen de sesión con el Broker.

```
1
2  public class WifiChangeReceiver extends BroadcastReceiver {
3      // Esta funcion verifica si esta conectado por Wifi
4      public synchronized void CheckAccessPoint(WifiManager wifi){
5          if(wifi!=null && wifi.isWifiEnabled()){ /// Wifi activado
6              if(wifi.getConnectionInfo().getBSSID()!=null){
7                  String MAC = wifi.getConnectionInfo().getBSSID().toUpperCase();
8                  setCurrentWIFIMAC(MAC);
9              }else{
10                 setDefaultWIFIMAC();
11             }
12         }else{
13             setDefaultWIFIMAC();
14         }
15         refreshIPinModulesbyMAC(getCurrentWIFIMAC());
16         RestartSubcriptionforAlarms(UtilsConexion.getCtxServiceAlarma());
17     }
18     //Cuando ocurre un cambio de conexion a Internet se activa esta funcion
19     public void onReceive(Context context, Intent intent) {
20         new HiloEspera5seg(
21             (WifiManager)context.getSystemService(Context.WIFI_SERVICE)).execute();
22     }
23 }
```

CÓDIGO 5.5: Actualización de los Ips a conectar por parte del Usuario Final.

En este sentido, en el punto anterior que menciona la visualización del gráfico con los valores medidos, también tiene implementada la clase *BroadcastReceiver* que le permite detectar si hay cambio de conexión. Sin embargo, en este caso espera 5,5 segundos para volver a realizar la suscripción; tiempo suficiente para tener actualizado el IP respectivo.

Capítulo 6

Resultados y Discusiones de Latencia y Performance

Este capítulo comienza con una sección introductoria que menciona las características de hardware y red, para luego enfocarse en los resultados obtenidos de la implementación descrita a lo largo de los anteriores capítulos de esta Tesis, haciendo una comparativa entre las arquitecturas Cloud Computing y Fog Computing. Los primeros resultados se centran en las pruebas de la latencia y su forma de obtención para 1 mensaje publicado. Finalmente, se muestra el estudio de los resultados de performance y latencia para la simulación descrita en el capítulo anterior.

6.1. Introducción

Antes de proceder a las secciones de pruebas se hace mención que el Nodo Edge es un Raspberry Pi 3 modelo B+ con SO Raspbian Lite y el servidor que hace de cloud es un servidor físico-remoto con Ubuntu Server 18.04 y 3 Gigabytes de RAM. La velocidad de bajada y subida del proveedor de internet para estas pruebas son de valores que oscilan entre 11 Mbps y 850 Kbps, respectivamente.

Para los siguientes resultados se realizaron múltiples tests para asegurar su veracidad; sin embargo, para cada sección se escogieron gráficos con los resultados más representativos. Adicionalmente, en el Apéndice A de este trabajo se tiene los *Boxplot* de los gráficos de este capítulo.

6.1.1. Latencia de 1 Publicación

La razón de esta sección es comprobar que se cumpla uno de los principios del Fog Computing que es la reducción de la latencia gracias a que el usuario final recibe la información medida directa de la fuente de origen.

Apoyados de la Figura 6.1 se define la latencia L de la ecuación (6.1) como la suma del tiempo l_1 que tomó la publicación del Nodo Edge (NE) hasta el Broker, con el tiempo l_2 que tomó en llegar del Broker al Usuario Final (UF). El símbolo t_0 es el tópico y m_0 es el mensaje que contiene la hora de salida T_0.

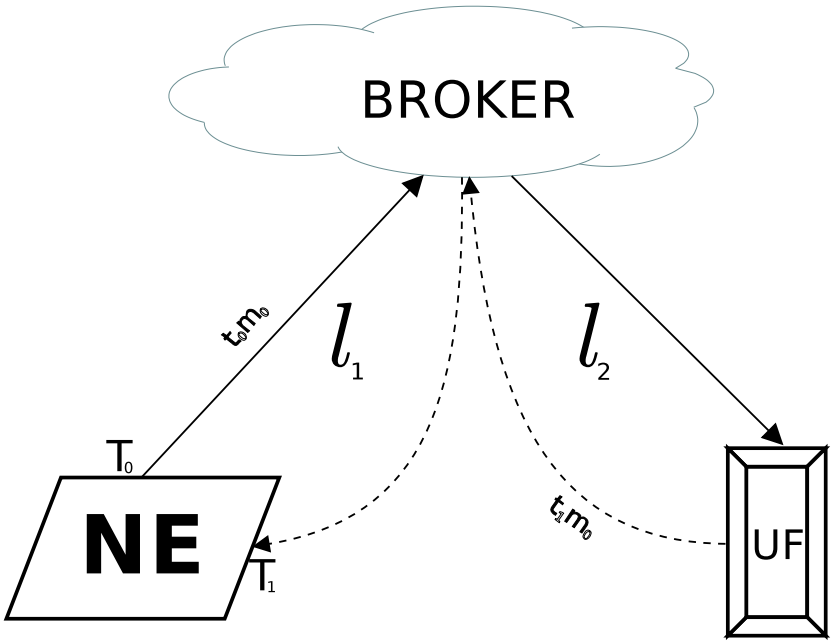


FIGURA 6.1: Latencia 1 envío. Fuente: Elaboración propia.

$$L = l_1 + l_2 \tag{6.1}$$

Sin embargo, para la obtención de la latencia de 1 sola publicación se hizo que el UF *re-publique* con otro tópico t_1 el mismo mensaje m_0 hacia el broker y el NE, previamente suscrito. Por tanto, el nodo Edge será el que reciba el mensaje, así como se muestra en la Figura 6.1. De esta forma se tiene la hora de llegada de la re-publicación (T_1) y, de dicho mensaje re-publicado, se obtiene la hora de salida (T_0), ambas marcadas por el reloj del Nodo Edge.

El motivo de hacer la re-publicación es que los relojes del Nodo Edge (Raspberry) y del Usuario Final (Aplicación Android) tienen un desfase entre ellos por lo que si se toma el tiempo marcado por el reloj del Usuario Final va ser distinto al tiempo marcado por el Nodo Edge. Si se pretende obtener la latencia como la hora que llegó al UF (obtenido del reloj del móvil) menos la hora que salió del NE (obtenido reloj del Raspberry) ambos deberían tener sincronizado relojes, por lo que se necesitaría equipos de alta precisión como relojes atómicos [34].

De la Figura 6.1 se tiene claro que para la hora T_1 se ha realizado un viaje de ida y vuelta del mensaje m_0 por lo que la diferencia con la hora T_0 es igual a dos veces el tiempo t_1 del NE - Broker, sumado con dos veces el Broker - UF como se aprecia en la siguiente ecuación.

$$T_1 - T_0 = 2l_1 + 2l_2 \tag{6.2}$$

Por lo tanto de la ecuación (6.1) y de (6.2) se tiene que la latencia queda definida como:

$$L = \frac{T_1 - T_0}{2} \tag{6.3}$$

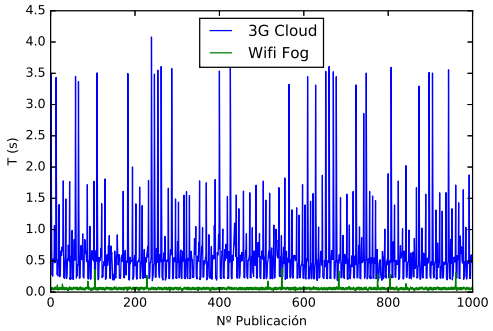
6.1.2. 3G vs 4G vs Wifi

Ahora que ya está definida la latencia en (6.3), en esta sub-sección se procede a mostrar los resultados de la fórmula para con las 3 redes de conexión más comunes actualmente. Esta prueba consistió de 1 Nodo Edge que está haciendo la doble publicación y 3 usuarios finales uno suscrito por Wifi al Nodo Edge que viene a representar Fog Computing y los otros dos suscritos al Broker de la Nube por la redes 3G y 4G representando a Cloud Computing. El Nodo Edge hizo publicaciones de un mismo tópico a una tasa de cada 4 segundos y los resultados son los siguientes:

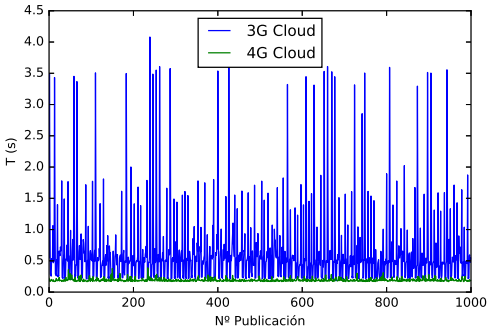
Las gráficas expuestas en las siguientes Figuras 6.2(a), 6.2(b) y 6.2(c) muestran resultados favorables con el usuario final suscrito por Wifi al Nodo Edge frente a las otras conexiones, aunque su ventaja es mínima haciendo una

comparación con la suscripción por 4G. En distintas pruebas se obtuvo una latencia de hasta 20 segundos en 3G tiempo que puede ser crucial pues si el mensaje sería una alarma de incendio para alguien conectado en 3G perdería 20 segundos decisivos para reaccionar frente al problema.

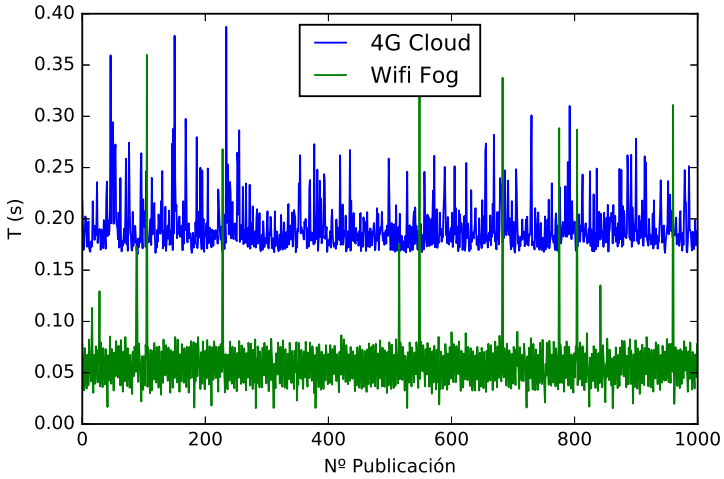
De la gráfica de Wifi Fog vs 4G Cloud de la Figura 6.2(c) se logra apreciar picos en donde la tecnología 4G tiene una menor latencia, dicho detalle es debido al desempeño del router. Sin embargo, al ser fracciones de segundos, la diferencia puede terminar siendo imperceptible.



(a) Latencia 3G vs Wifi.



(b) Latencia 3G vs 4G.



(c) Latencia 4G vs Wifi.

FIGURA 6.2: Latencia de 1 publicación entre conexiones 3G, 4G y Wifi. Fuente: Elaboración propia.

El cuadro 6.1 presenta los detalles de la Figura 6.2(c), en este se infiere que en promedio las conexiones Wifi Fog y 4G Cloud son más estables con

desviaciones estándar similares aunque ambas son fracciones de segundo. Esto motiva a que las siguientes pruebas sólo se hagan entre estas dos tecnologías para apreciar mejor los resultados.

Conexión	Latencia Promedio (s)	Desviación Estándar (s)
Wifi	0.058585	0.027956
4G	0.191857	0.024981
3G	0.612475	0.617738

TABLA 6.1: Latencia promedio para 1 publicación.

6.2. CEP en el Nodo Edge vs CEP en la Nube

Esta sección se realizó de la misma forma como se aprecia en los diagrama de flujos de las Figuras 4.2(a) y 4.2(b) ubicando el motor CEP en el Nodo Edge y realizando la doble publicación para el caso de Fog Computing. Para el caso de Cloud Computing el Nodo Edge sólo publicaba hacia la Nube donde este último tiene el motor CEP. En las posteriores pruebas no se tomó en cuenta el almacenamiento de los datos debido a que en ambos casos está en la Nube y este estudio recae sobre la ubicación del mòdulo CEP.

Dada la capacidad computacional de cada uno en el Raspberry se trabajó con un nivel de paralelismo en los taskslot de 50 y en la Nube de 175, ambos con un tamaño de pila del Taskmanager de 512 MB.

Como ya se mencionó en la Sección 5.2.2 del capítulo de implementación, para todas las pruebas se hicieron simulaciones de 10 minutos de dicho loop, obteniendo en todas un 100 % datos analizados y alarmas publicadas a excepción del caso 800 alarmas en el caso de Fog Computing-4G en el que se llega a publicar a la Nube un 99 % de las alarmas detectas.

La razón de dicha falla se debe tanto al ancho de banda de subida de las condiciones de las pruebas explicadas al inicio de esta sección como también a la falta de recursos del Nodo Edge, teniendo así dichas publicaciones acumuladas en el buffer. Estos problemas son más notorios para las posteriores

escalas en los que se apreciaban fallos similares durante la simulación en los que incluso el Nodo Edge empieza a presentar errores en la creación de nuevos hilos para publicar. Por este motivo se tomó 800 alarmas como límite de las posteriores pruebas que según este caso de uso vendría a hacer un aproximado de más de 30 módulos de sensores.

El cuadro 6.2 resume que hasta 700 es el límite en el que se obtiene un 100 % de alarmas publicadas a la Nube, para 800 empieza a disminuir el porcentaje total de alarmas que debió publicar. Finalizando con los casos de 900 y 1000 alarmas/min en los que a veces cumplió en analizar el 100 % de datos analizados aunque al igual que para 800 no logró hacer todas las publicaciones de alarmas.

Nº Alarmas	**% Publicado**	**Errores**
700	100 %	No
800	99 %	Sí
900	96 %	Sí
1000	72 %	Sí

TABLA 6.2: Porcentaje de mensaje publicados a la Nube con el modelo Fog Computing.

6.2.1. Latencia

El objetivo de esta sección no es obtener el tiempo exacto en ser detectada las alarmas, sino mostrar qué arquitectura y conexión mostró un mejor perfil. Debido a esto, en esta sección se utilizará la misma fórmula final que la Sección 6.1.1 y se tomará como un valor referencial ya que, durante las pruebas, las condiciones y estados son diferentes tanto en el flujo de datos como en el análisis del mismo.

Como se ve en la Figura 6.3 hay varios puntos diferentes que se detallan a continuación:

- Como primer punto, el flujo es distinto ya que ahora se está analizando la información con el motor CEP (recordemos que anteriormente solamente

se encontraba el Broker) y al momento de la re-publicación el mensaje m_1 no vuelve a ser analizado por CEP; del mismo modo este mensaje no es el original m_0 durante todo el flujo.

- Segundo punto, las condiciones de estrés sobre el Broker y sobre el UF son variantes según la cantidad de trabajo; es por eso que se detalla con un reloj al costado de cada momento de publicación. En este punto entra en juego el ancho de subida y del ancho de bajada.

- El tercer y último punto está relacionado a las publicaciones, ya que originalmente del Nodo Edge son sesiones que publican y se desconectan; y en el motor CEP hay un array ya conectado a la espera de un nuevo mensaje para publicar.

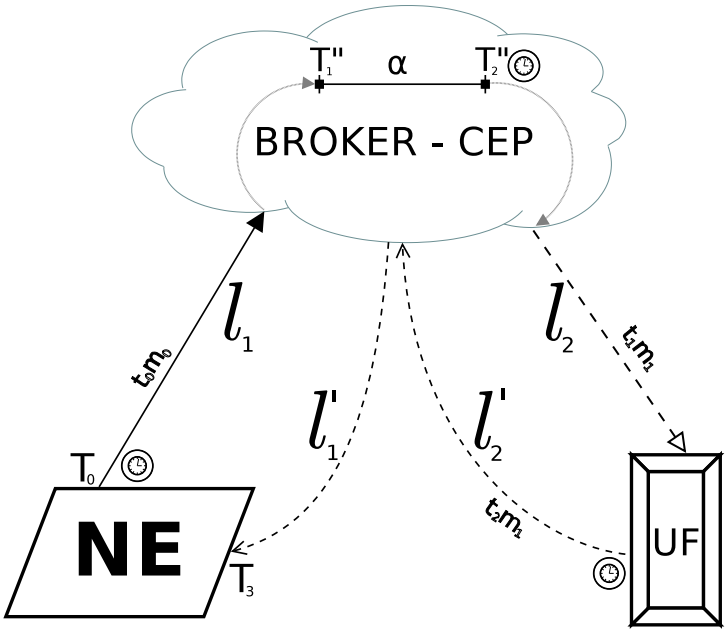


FIGURA 6.3: Latencia con motor CEP. Fuente: Elaboración propia.

Continuando con la explicación del párrafo anterior, de la Figura 6.3 se tiene a α como el tiempo de análisis del módulo CEP con $T_1^{"}$ como la hora en que ingresó al módulo CEP la publicación que genera la alarma y $T_2^{"}$ como la hora en que se intentó publicar la alarma. Nuevamente, los relojes que están al costado de T_0, $T_2^{"}$ y del UF indican que la hora en que se intentó publicar no es la misma

hora en que se llegó a publicar; es decir, en estos existe un pequeño delay y es variante según lo que tome la conexión y publicación acorde a la carga que esté sobre el Broker en ese momento.

Por motivo de este delay y de la variante en el mensaje m_0 y m_1 es que no se puede asumir que las latencias l_1 con $l_1^{'}$ y l_2 con $l_2^{'}$ son las mismas, además que las sesiones de publicación son distintas como se explicó en la Sección 5.3. Las líneas dentro de la Nube del BROKER-CEP significa el tiempo en la obtención y publicación del módulo CEP con su Broker.

Debido a todos esos motivos sólo se tomará un referencial en todos los casos, como el tiempo en que se tomó el mensaje en regresar al Nodo Edge dividido por dos ya que como se mencionó al inicio de la sección el objetivo no es dar el tiempo exacto sino mostrar que arquitectura muestra un mejor perfil. Sin embargo, lo que sí nos interesa es el tiempo α que es el tiempo que tomó el módulo CEP para detectar la alarma.

Por lo tanto a continuación se presenta la Figura 6.4 como el gráfico de los resultados de latencia promedio aproximados utilizando como referencia la fórmula (6.3) para las arquitecturas Cloud y Fog Computing con las conexiones Wifi y 4G, asimismo más adelante se presenta una tabla con el tiempo α de análisis en el módulo CEP.

En la Figura 6.4 se aprecia que para 1 alarma cada minuto la diferencia no es mucha siendo la respuesta casi inmediata similar a lo visto anteriormente en la Sección 6.1.2, no obstante a medida que se aumenta la cantidad de alarmas por minuto se ve que los resultados de Wifi - Fog Computing son los que muestran menores valores de latencia aproximada frente a los otros, y que 4G - Cloud Computing muestra los resultados más altos.

Otro punto resaltante en estos resultados es que la pendiente de subida para las pruebas con Fog Computing es más pequeña a comparación de los de Cloud Computing esto debido a que no es lo mismo publicar en un Broker local a hacerlo a uno remoto, estos resultados son similares a los que obtuvieron los autores en [35] en el que afirman que utilizar el ancho de banda local da mejores resultados. Finalmente entre mismas arquitecturas las conexiones por

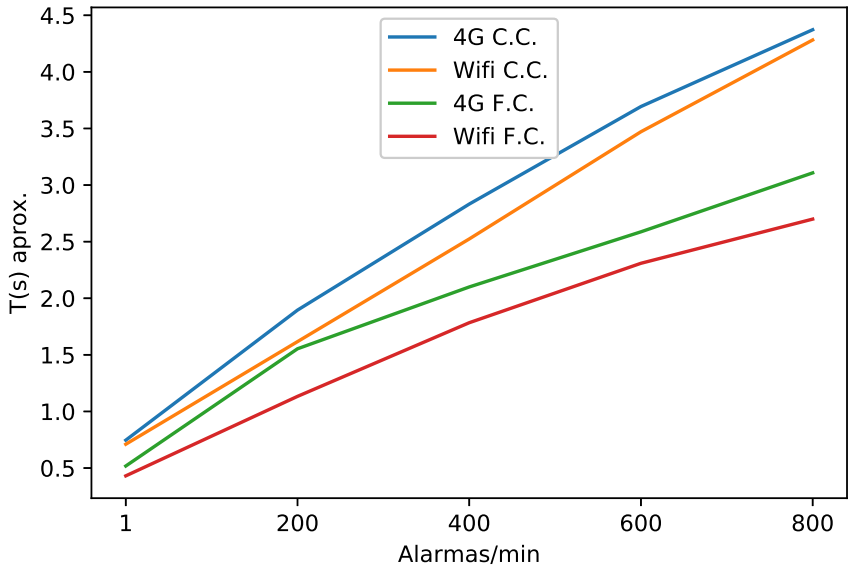


FIGURA 6.4: Latencia general aproximada. Fuente: Elaboración propia.

Wifi dieron menores valores de latencia que las de 4G.

La Tabla 6.3 muestra el tiempo de análisis α que tomó el CEP en cada arquitectura según la cantidad de alarmas por minuto. Como era de esperarse la Nube al tener mayor capacidad de cómputo da menores resultados en el tiempo de análisis a comparación del Nodo Edge, que mínimo toma 6 veces más. Sin embargo, en general los tiempos son pequeños a comparación de lo mostrado en la Figura 6.4 por lo que se puede concluir que el desempeño del Broker juega un papel importante en la latencia, siendo este el responsable de que el tiempo aproximado sea tan alto.

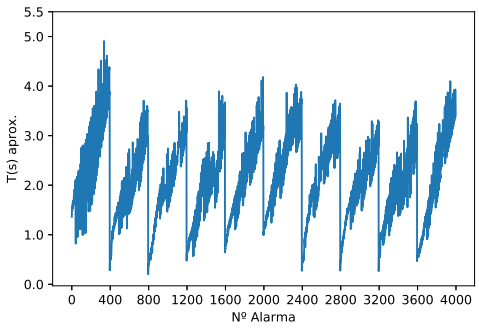
Nº Alarmas	**Cloud Computing (s)**	**Fog Computing (s)**
1	0	0
200	0.046	0.325
400	0.048	0.380
600	0.066	0.432
800	0.072	0.461

TABLA 6.3: Tiempo α de Análisis por el motor CEP en segundos.

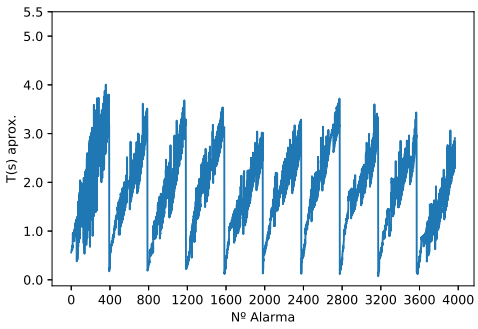
A continuación se muestran los gráficos lineales de la latencia promedio en

el caso de 400 alarmas cada minuto como representativo de las otras pruebas. El objetivo de estos gráficos es mostrar el comportamiento del Broker con las Alarmas ya que, siguiendo los resultados de la Tabla 6.3, la mayor parte del tiempo de latencia (aproximada) es del envío y recepción en la publicación y suscripción.

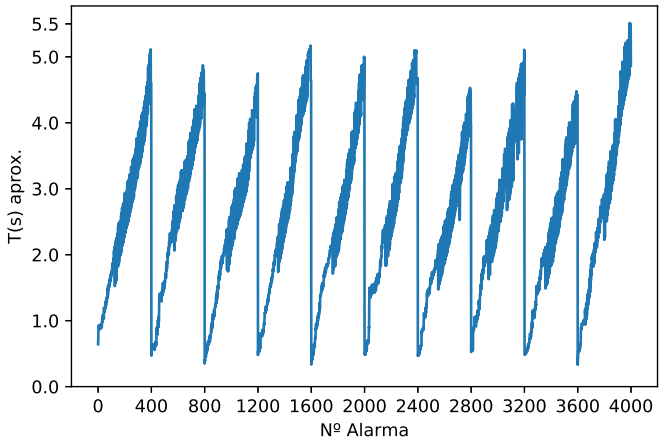
En las Figuras 6.5(a) y 6.5(b) se pueden visualizar la latencia en Wifi y 4G para Fog Computing. Por otro lado, la Figura 6.5(c) muestra la latencia en Wifi para Cloud Computing. En todas las figuras el eje X es el número de alarma y cada bloque de 400 alarmas ocurre en un minuto.



(a) 4G - Fog Computing.



(b) Wifi - Fog Computing



(c) Wifi - Cloud Computing.

FIGURA 6.5: Latencia de 400 Alarmas/min para 4G y Wifi en Fog y Cloud Computing. Fuente: Elaboración propia.

La razón de no poner a 4G - Cloud Computing es porque el flujo de envío y análisis es prácticamente el mismo que Wifi - Cloud Computing teniendo

al Broker y al módulo CEP ubicados en la Nube, siendo la única diferencia el tipo de conexión. En cambio, para Wifi - Fog Computing se tiene al Broker y al módulo CEP en el Nodo Edge ya que todo el flujo es en la misma LAN; sin embargo, para 4G - Fog Computing también se utiliza el Broker global ya que de ahí el UF recibe la publicación de la alarma.

De las tres figuras anteriores, se nota que se cumple aproximadamente un patrón de repetición por cada tanda de publicaciones (bloque de cada 400 alarmas). Teniendo en ambos casos de Fog Computing para la alarma número 400, un pico de latencia más alto que los posteriores en 800, 1200, etc que son cuando ya se detectó la última alarma simulada. La razón de este alto valor de latencia es debido a la implementación mencionada en 5.3, ya que al inicio se realiza la conexión y publicación de las alarmas pero para las siguientes publica directamente. No obstante, como la Nube posee más recursos para Cloud Computing es indistinto.

Finalmente de las Figuras 6.5 y en conjunto con el tiempo de análisis α de la Tabla 6.3, que es pequeño en comparación a estos resultados, nos da a entender que este comportamiento es debido al manejo del Broker para con las publicaciones y suscripciones. Este comportamiento tiene aspecto de encolar las publicaciones y se aprecia tanto para Cloud como Fog Computing, por lo que es independiente de la red.

6.2.2. Performance

Prosiguiendo con las pruebas sobre la simulación definida en la Sección 5.2.2, este punto se enfoca en la comparación de consumo de recursos por parte del servidor de la Nube y también del Raspberry, cada uno con las dos arquitecturas.

Los gráficos que se muestran a continuación tanto para el servidor de la Nube y el Raspberry son los porcentajes promedio de CPU y RAM utilizados, ambos obtenidos por el comando *top*. Y finalmente aprovechando que la Nube es un servidor-físico se obtuvo los valores de consumo de energía en Joules (J) de los cores y de la RAM gracias a los eventos estadísticos *power/energy-cores/* y

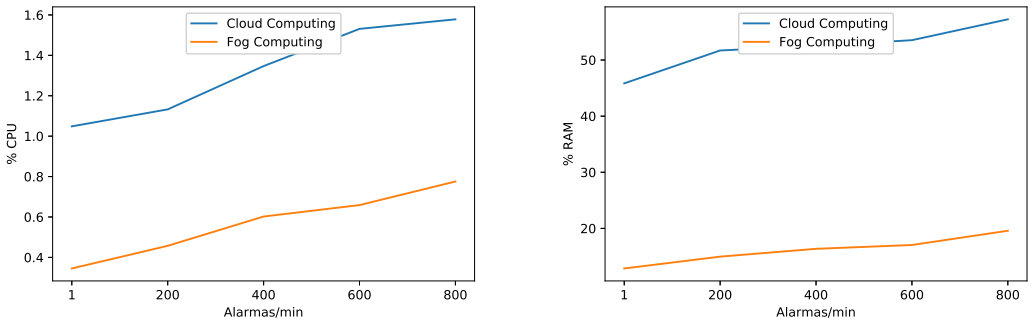
power/energy-ram/ del comando perf, estos eventos de kernel no se encuentran en el Raspberry.

Para aclarar, la obtención de un único valor en los test es la medición en un lapso de 0.33 segundos por lo que si se quiere obtener Watts que es J/s, aproximadamente tendría que multiplicarse por 3 y así obtener el consumo total promedio.

Nube

La Figura 6.6(a) muestra que el consumo promedio de % CPU no ha sido excesivo si bien es cierto cuando se utiliza el motor CEP (Apache Flink) para el caso de Cloud Computing da un aumento en el consumo, este sólo bordea el 1 % a lo largo de las pruebas. Este gráfico, además, es una pequeña muestra de como serán los demás resultados para el servidor de la Nube, el cual puede observarse un consumo de Cloud Computing por encima de Fog Computing.

Los resultados se vuelven más interesantes para la Figura 6.6(b) que es el porcentaje de espacio RAM utilizado. En esta figura para el caso de 1 alarma por minuto se logra apreciar que la sola utilización del motor CEP da un salto de más del 30 % en el consumo de memoria. Esto es debido a los procesos del Jobmanager y el Taskmanagers y a la carga de inicializar los 175 taskslots. En contraste, para el caso de Fog Computing, la carga es baja teniendo como único



(a) % de CPU utilizado.

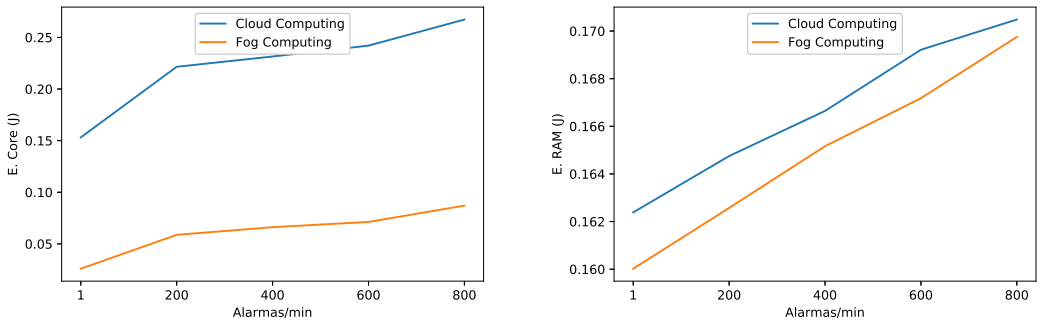
(b) % de espacio de memoria RAM utilizado

FIGURA 6.6: Porcentaje de CPU y de espacio RAM utilizados en la Nube con Cloud y Fog Computing. Fuente: Elaboración propia.

consumo el del Broker MQTT. Estos valores, para cada arquitectura, se ven con poca variación a lo largo de las pruebas demostrando que durante ejecución, tanto las tareas del motor CEP como del Broker MQTT, son administrados de forma eficiente por el SO, es decir, no hay tanta varianza.

En cuestiones de energía, como se mencionó al inicio de esta subsección, el comando *perf* ofrece resultados del consumo energético en Joules. La Figura 6.7(a) muestra una reducción promedio de más del 0.10 J que viene a ser una reducción de más del 50 % del consumo en los cores del sistema en cada prueba. Esta reducción tiene relación con la Figura 6.6(a) pues si se utiliza menos del CPU de seguro va ser menor su consumo energético.

A su par en el Figura 6.7(b) se observa que el consumo para Fog y Cloud Computing son similares, si bien es cierto Fog Computing consume menos la diferencia no es tan notoria siendo 0,001 J para el caso de 800 alarmas. Esta tendencia es en todos los test además queda a resaltar que en líneas general entre el mínimo de Fog Computing para 1 alarma cada minuto contra el mayor de Cloud Computing con 800 alarmas la diferencia no es mayor a un 0,012 J.



(a) Consumo de los Cores.

(b) Consumo de la RAM

FIGURA 6.7: Energía consumida en Joules (J) de los Core y la RAM en la Nube con Cloud y Fog Computing. Fuente: Elaboración propia.

Contrastando las Figuras 6.7(b) y 6.6(b) a simple vista los resultados promedio del espacio de memoria RAM utilizado contra su energía consumida promedio no sugiera relación directa pero no es para asombrase puesto que la memoria RAM, que de sus siglas dicen memoria de acceso aleatorio o en inglés

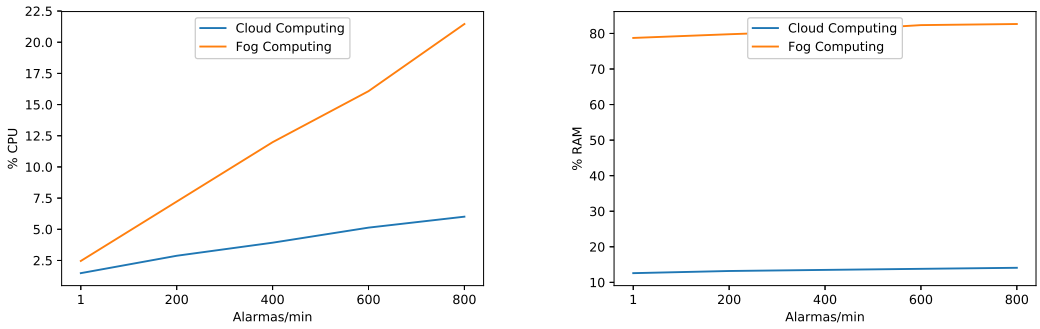
Random Access Memory, de por sí es un área solicitada frecuentemente para la lectura y escritura de múltiples fuentes y utilizada según vea conveniente el SO. Lo que se quiere explicar es que el espacio de memoria utilizado no demuestra el trabajo que se realiza sobre el mismo. De esto para cada caso la pequeña diferencia en Joules es a causa del consumo energético que toma el trabajo realizado por los procesos e hilos del CEP sobre el espacio de memoria.

Nodo Edge

El Raspberry es un dispositivo que no tiene un hardware que permita al comando *perf* medir el evento de consumo energético por lo que se procede a exponer sólo el consumo de CPU y RAM.

Continuando con los resultados ya discutidos en las Figuras 6.6(a) y 6.6(b) se espera que los resultados de porcentaje de CPU y de RAM utilizados en las Figuras 6.8(a) y 6.8(b) tengan valores promedio más en general teniendo al caso de Fog Computing como el que consume mayor cantidad.

En la Figura 6.8(a) se presenta que para el caso de Fog Computing se consume cada vez más % CPU promedio y esto es debido a que la carga de trabajo no es sólo la del motor CEP sino, también del trabajo que toma realizar una doble publicación, mostrando así que la pendiente para esta arquitectura es mucho mayor que para el caso de Cloud Computing.



(a) % de CPU utilizado.

(b) % de espacio RAM utilizado

FIGURA 6.8: Porcentaje de CPU y de espacio RAM utilizados en el Nodo Edge con Cloud y Fog Computing. Fuente: Elaboración propia.

Finalizando en la Figura 6.8(b) se ve que la carga y trabajo de los procesos del motor CEP y Broker le consume gran parte del espacio de la RAM al Raspberry Pi siendo más de un 60 %. Esto da razón al escalamiento del % CPU ya que como la RAM está siendo consumida en gran tamaño recurre a los recursos del CPU para su trabajo en ejecución. Finalmente, ya ejecutados tanto el módulo para Fog Computing como para Cloud Computing el aumento de carga de trabajo sobre la RAM tiende a aumentar.

6.3. Discusiones Finales

Como sección final se presentan unos puntos como discusión y resumen de los resultados expuestos a lo largo de este capítulo:

- Tanto para el caso de 1 publicación como durante la simulación de alarmas se obtuvieron menores resultados de latencia para cuando el usuario final está conectado por Wifi al Nodo Edge.
- En las pruebas de simulación de alarmas se obtuvo que el valor de la latencia es en gran parte debido al desempeño del Broker de Mosquitto.
- El módulo CEP (Apache-Flink) consume mayormente el espacio de memoria RAM a diferencia del porcentaje de CPU. Esto se hizo muy notorio en Nodo Edge (Raspberry).
- El consumo energético de RAM medido por el comando *perf* para la Nube, mostró resultados que no guardan relación con el espacio de memoria RAM utilizado. El espacio de memoria RAM utilizado no demuestra el trabajo que se realiza sobre el mismo.
- Utilizar el módulo CEP en el Nodo Edge demuestra un gran aprovechamiento de los recursos computacionales de este, además de ofrecer menores tiempos de latencia y reducirle gran parte de trabajo computacional a la Nube.

Capítulo 7

Conclusiones y Trabajo a futuro

En este último capítulo se presenta las conclusiones del trabajo desarrollado e implementado, fundamentándose en los resultados del capítulo predecesor. Adicionalmente, se mencionan el trabajo a futuro para continuar con la implementación, mejora y posterior expansión de la plataforma.

7.1. Conclusiones

Luego de todo el estudio realizado y explicado a lo largo de este trabajo de Tesis se evidencia las siguientes conclusiones:

- Ha sido comprobado que la utilización del módulo de análisis CEP en el Nodo Edge ha dado buenos resultados por lo que la descentralización del mismo es altamente recomendable. El estudio de los resultados realizado en el capítulo anterior se pretende que sirva de fundamento o pasos para posteriores tareas a fin de continuar con la descentralización de la Nube.
- La utilización de herramientas open-source como Mosquitto y Apache Flink resultaron ser eficientes tanto en el Nodo Edge como en la Nube, sólo teniendo que configurar este último según las características y capacidad de cada uno.
- La implementación de la doble publicación en LAN y WAN ofrece el continuo servicio de envío de información en online y offline lo que es beneficioso para zonas alejadas de las urbes en la que la red de Internet presente inestabilidad y o cortes de conexión.

- La aplicación Android del usuario final mostró una interfaz simple y amigable. La suscripción al IP correspondiente según la red en la que se encuentre no presentó inconveniente y para casos de cambio de red la obtención de la MAC de Access Point se hizo gracias al delay de espera de 5 segundos. Los gráficos de la aplicación cumplieron con mostrar los últimos valores y actualizarse con la llegada de nuevos medidos.

- Se comprobó que la latencia para cuando el usuario final está conectado por Wifi al Nodo Edge es menor a cuando está conectado por 4G o 3G a la Nube. Esto es beneficioso en general en las aplicaciones IoT en las que se precise de notificaciones rápidas o incluso en las que sólo se tenga la conexión 3G.

- Las pruebas de performance muestran la reducción de trabajo en la Nube, lo que si fuese el caso de un servidor de alquiler como los de Amazon reduciría sus costos de mantenimiento. Además, que permitiría dejar disponible recursos para otro tipo de cómputo.

7.2. Trabajo a Futuro

A continuación se presentan las propuestas para el trabajo a futuro para continuar con este presente trabajo en la optimización de la descentralización de la Nube con la arquitectura Fog Computing.

7.2.1. Nuevas tareas en el Nodo Edge

Siguiendo con la descentralización de la Nube, se pretende continuar con el estudio de más tareas que puedan ser realizadas por el Nodo Edge como son las de almacenamiento de últimos valores según sensor para envío de valores promedios o resumidos. Adicionalmente el almacenamiento provisorio de información que no fue publicada a la Nube por distintos motivos hasta que se pueda publicar con éxito junto con su informe de errores, trabajo que complementa al actual.

7.2.2. Benchmarks y nuevas métricas

En la literatura para clasificar software y hardware existen otras métricas además de las utilizadas en este trabajo como son los distintos benchmark y otras métricas que involucran acceso y fallos de lectura en la memoria cache del CPU por lo que sería interesante hacer un estudio de dichas métricas para comparar con el trabajo de otros autores.

7.2.3. Implementación con otro Hardware

A pesar de que los resultados con el Raspberry Pi 3 B+ y el servidor físico fueron satisfactorios en el mercado existen más tarjetas y servidores con capacidades y precios similares como son las Odroid Xu4 y las UDOO X86. Así sería preciso hacer un estudio similar en las que se indique cuales son las ventajas de estas nuevas con las anteriores según este mismo caso de uso.

7.2.4. Mobile Computing y versión iOS

Si bien es cierto la aplicación en Android cumple con la labor para este trabajo de Tesis, se ve con buenos ojos la utilización del paradigma de Mobile Computing en el que el Nodo Edge sea un smartphone ya que cada año el precio de los celulares y tablets de gran capacidad disminuye. Esto conlleva también desarrollar una versión en iOS que cumpla con la misma labor ya sea para la versión actual y también para una versión como Nodo Edge.

7.2.5. Microclouds en el Nodo Edge y la Nube

En la actualidad se está trabajando la optimización de recursos sin perjudicar la calidad de servicio. En arquitecturas centralizadas como Cloud Computing puede verse que el uso de contenedores mejora la funcionalidad en este tipo de arquitecturas, llamando a esta técnica, microclouds. Un aspecto interesante es ver el impacto de estas herramientas en Fog Computing

analizando el consumo computacional a imagen y semejanza de como se ha presentado en esta tesis.

7.3. Competencias Adquiridas

Finalmente, como cierre de este trabajo de Tesis se pone en escrito las competencias adquiridas y demostradas a lo largo de este documento. Los siguientes puntos guardan relación con los objetivos y conclusiones, y con la currícula actual de la Escuela.

- Se adquirió la capacidad para el desarrollo de aplicaciones Android. En la que se utilizó el modelo de vista controlador para el desarrollo del aplicativo en la gráfica y notificación de valores publicados. Además del desarrollo de una aplicación adicional que republicaba la información para obtener la data necesaria en la fórmula de la latencia.

- Con el manejo del Raspberry Pi 3 B+ y el servidor de la Nube, se adquirió capacidad de manejo de Sistemas Operativos como Raspbian Lite y Ubuntu Server para la configuración y aprovechamiento de las capacidades del hardware.

- A lo largo de este proyecto se adquirieron conocimientos sobre el paradigma de publicador/suscriptor para proyectos IoT y del análisis de patrones en flujo de datos con el motor CEP.

- Para la obtención de gráficos y tratamiento de la data en el post procesamiento se adquirieron conocimientos de Python con Jupyter-notebook.

- Se adquirió conocimiento de scripts en bash para la ejecución sincronizada de los distintos servicios y también en las pruebas de rendimiento ya que en este se necesitaba recolectar la data relevante entre todo el output que ofrecían los comandos *perf* y *top*.

Bibliografía

[1] M. Rouse, "What is mqtt (message quering telemetry transport)?." http://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport, 2018. [Online; Accedido: 2019-Marzo-15].

[2] M. Rouse, "What is internet of things." http://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT, 2019. [Online; Accedido: 2019-Marzo-25.

[3] BankInter, "Internet de las cosas." https://swimming652.wordpress.com/2016/09/22/93/, 2016. [Online; Accedido: 2019-Marzo-25].

[4] C. Klingenberg, "Industry 4.0: what makes it a revolution?," *Conference:EurOMA 2017*, 7 2017.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, 8 2012.

[6] M. Castillo-Cara, E. Huaranga-Junco, M. Quispe-Montesinos, L. Orozco-Barbosa, and E. A. Antúnez, "Frog: a robust and green wireless sensor node for fog computing platforms," *Journal of Sensors*, vol. 2018, 2018.

[7] O. Consortium, "Welcome to the fog computing era." https://www.openfogconsortium.org/#fog-computing, 2019. [Online; Accedido: 2019-Marzo-15].

[8] P. P, D. K.G, Y. P, M. V. Ganesh, and V. B, "Fog computing: Issues, challenges and future directions," *International Journal of Electrical and Computer Engineering (IJECE)*, p. 3669 3673, 12 2017.

[9] O. Consortium, "What we do." https://www.openfogconsortium.org/what-we-do/, 2019. [Online; Accedido: 2019-Marzo-25].

[10] Wikipedia, "Publish–subscribe pattern." https://en.wikipedia.org/wiki/Publish\T1\textendashsubscribe_pattern, 2019. [Online; accedido 07-Abril-2019].

[11] IBM, "Publish/subscribe." https://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.mft.doc/aq01120_.htm, 2019. [Online; accedido 07-Abril-2019].

[12] D. C. Luckham, "Complex event processing." https://en.wikipedia.org/wiki/Complex_event_processing, 2019. [Online; accedido 29-Marzo-2019].

[13] M. Eckert and F. Bry, "Aktuelles schlagwort: Complex event processing (cep)," *Informatik-Spektrum, Vol. 32*, pp. 163–167, 4 2009.

[14] M. P. Madumal, D. Atukorale, and T. Usoof, "Adaptive event tree-based hybrid cep computational model for fog computing architecture," *IEEE Advances in ICT for Emerging Regions (ICTer), 2016 Sixteenth International Conference on*, pp. Electronic ISSN: 2472–7598, 1 2017.

[15] T. Rohrmann, "Introducing complex event processing (cep)." https://flink.apache.org/news/2016/04/06/cep-monitoring.html, 2016. [Online; Accedido: 2019-Marzo-15.

[16] B. Varghese, N. Wang, D. S. Nikolopoulos, and R. Buyya, "Feasibility of fog computing," *arXiv*, 1 2017.

[17] A. S. Dhillon, S. Majumdar, M. St-Hilaire, and A. El-Haraki, "Mcep: A mobile device based complex event processing system for remote healthcare," *IEEE International Conference on Internet of Things*, 6 2018.

[18] T. M. Fernández-Caramés, P. Fraga-Lamas, M. Suárez-Albela, and M. A. Díaz-Bouza, "A fog computing based cyber-physical system for the automation of pipe-related tasks in the industry 4.0 shipyard," *Sensors*, 6 2018.

[19] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, and R. S. Tucker, "Fog computing may help to save energy in cloud computing," *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, pp. 1728 – 1739, 3 2016.

[20] R. ORG, "Raspberry pi model b plus product brief." `https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf`, 2018. [Online; Accedido: 2019-Marzo-29].

[21] C. C. Clay., "Raspberry pi: 11 reasons why it's the perfect small server." `http://www.zdnet.com/article/raspberry-pi-11-reasons-why-its-the-perfect-small-server/`, 2016. [Online; Accedido: 2019-Marzo-29].

[22] R. MORABITO, "Virtualization on internet of things edge devices with container technologies: A performance evaluation," *IEEE Access*, pp. 8835 – 8850, 5 2017.

[23] F. S. E. L. Zhang., "Building facebook messenger." `https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920/`, 2011. [Online; Accedido: 2019-Marzo-15].

[24] M. ORG., "Eclipse mosquitto." `https://mosquitto.org/`. [Online; Accedido: 2019-Marzo-15].

[25] A. Flink, "Documentation: Distributed runtime environment." `https://ci.apache.org/projects/flink/flink-docs-stable/concepts/runtime.html`. [Online; Accedido: 2019-Marzo-26].

[26] A. Flink, "Documentation: Flinkcep - complex event processing for flink." https://ci.apache.org/projects/flink/flink-docs-stable/dev/libs/cep.html#flinkcep-complex-event-processing-for-flink. [Online;Accedido: 2019-Marzo-26].

[27] Wikipedia, "perf (linux)." https://en.wikipedia.org/wiki/Perf_(Linux), 2019. [Online; accedido 29-Marzo-2019].

[28] MediaWiki, "Tutorial - perf wiki." https://perf.wiki.kernel.org/index.php/Tutorial, 2015. [Online; accedido 29-Marzo-2019].

[29] Wikipedia, "top (software)." https://en.wikipedia.org/wiki/Top_(software), 2019. [Online; accedido 29-Marzo-2019].

[30] J. . J. C. Warner, "top - unix, linux command." http://www.tutorialspoint.com/unix_commands/top.htm, 2014. [Online; Accedido: 2019-Marzo-29].

[31] S. Biswas, "A guide to the linux "top" command." https://www.booleanworld.com/guide-linux-top-command/, 2018. [Online; Accedido: 2019-Marzo-29].

[32] Wikipedia, "Android." https://es.wikipedia.org/wiki/Android, 2019. [Online; accedido 29-Marzo-2019].

[33] P. Miller, "Google is adding kotlin as an official programming language for android development." https://www.theverge.com/2017/5/17/15654988/google-jet-brains-kotlin-programming-language-android-development-io-2017, 2017. [Online; Accedido: 2019-Marzo-25].

[34] T. Cozzens, "Atomic clock." https://www.gpsworld.com/orolia-technology-synchronizes-black-hole-photo-telescopes/, 2019. [Online; accedido 29-Mayo-2019].

[35] C. Shi, Z. Ren, K. Yang, C. Chen, H. Zhang, Y. Xiao, and X. Hou, "Ultra-low latency cloud-fog computing for industrial internet of things," *2018 IEEE*

Wireless Communications and Networking Conference (WCNC), pp. Electronic ISSN: 1558–2612, 6 2018.

Apéndice A

Resultados en BoxPlots

Los Boxplot o también llamados diagrama de caja son gráficos que muestran los datos en lo que parece ser una caja con límite en su primer y tercer cuartil, y la mediana se marca como una linea que parte dicha caja. Además permite mostrar los valores atípicos también llamados outliers.

A continuación se presentan los Boxplot que complementan lo escrito en las Secciones 6.1.2, 6.2.1 y 6.2.2 .

A.1. Latencia de 1 publicación

Estos primeros Boxplot de la Figura A.1 muestran el comportamiento general de la latencia en los tipos de conexión para 1 sola publicación. Donde se claramente que el para la conexión Wifi es mucho menor que los anteriores.

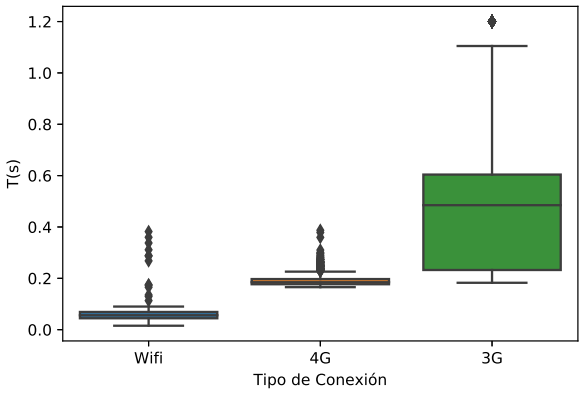
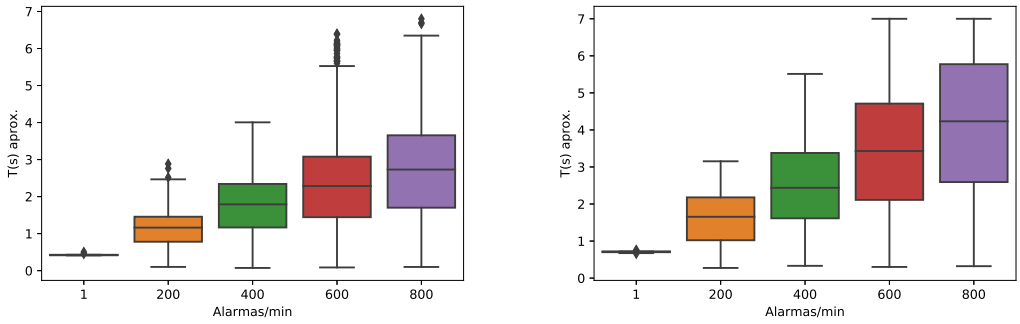


FIGURA A.1: Boxplot de latencia 1 publicación. Fuente: Elaboración propia.

Estos Boxplots además son un ejemplo de los siguientes que se mostraran, teniendo cortada las gráficas hasta el límite del tercer cuartil del Boxplot más grande, ignorando de esta forma los outliners para que no se desvíe la atención de los menores.

A.2. Latencia de las alarmas

Como se explicó en la Sección 6.2.1 los gráficos lineales para Cloud y Fog Computing son muy similares, por lo que sólo se mostrarán para los resultados obtenidos por wifi.



(a) Cloud Computing con Wifi.

(b) Fog Computing con Wifi.

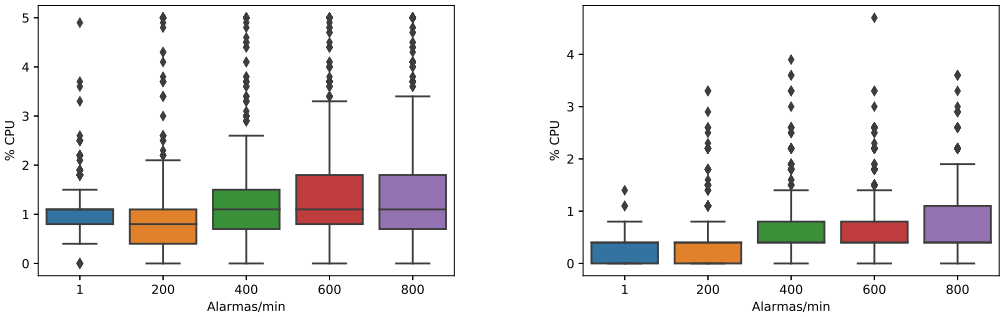
FIGURA A.2: Boxplots de latencia de las alarmas con Wifi en Cloud y Fog Computing. Fuente: Elaboración propia.

Como era de esperarse para la Figura de Fog Computing existen varios outliers en su cota superior y es debido a la primera conexión ya en general después es sólo publicación lo que hace que los valores sean menores y se acumulen por debajo de los iniciales, debido a esto para el test de 1 alarma por minuto sus valores quedan un poco arriba a comparación de los otros test ya que en todos se conecta y luego publica.

A.3. Performance

A.4. Nube

Los boxplot para el caso de % CPU este da la referencia de como va aumentando su tercer cuartil además de que existen picos en donde se va consumiendo más del CPU. Así se ve en las Figuras A.3(a) y A.3(b).

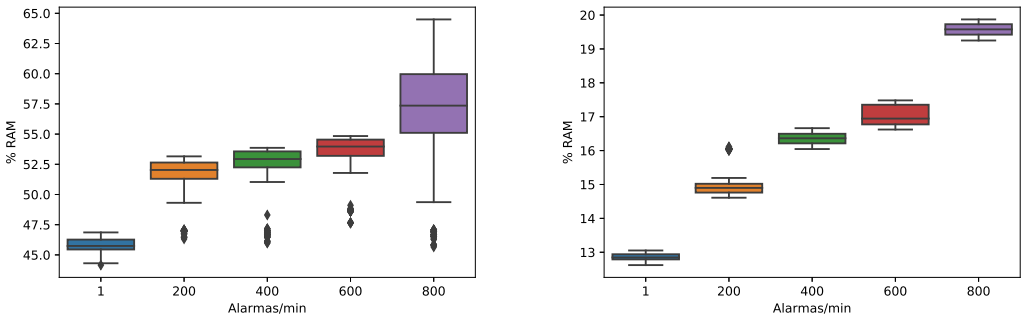


(a) CPU en Cloud Computing.

(b) CPU en Fog Computing.

FIGURA A.3: Boxplots del consumo de CPU de la Nube en Cloud y Fog Computing. Fuente: Elaboración propia.

Similares resultados son para el porcentaje de espacio de memoria RAM utilizada en las Figuras A.4(a) y A.4(b), sólo que para el caso de Cloud Computing existe outliers por debajo lo que nos informa que el SO intenta liberar espacio en la memoria RAM.

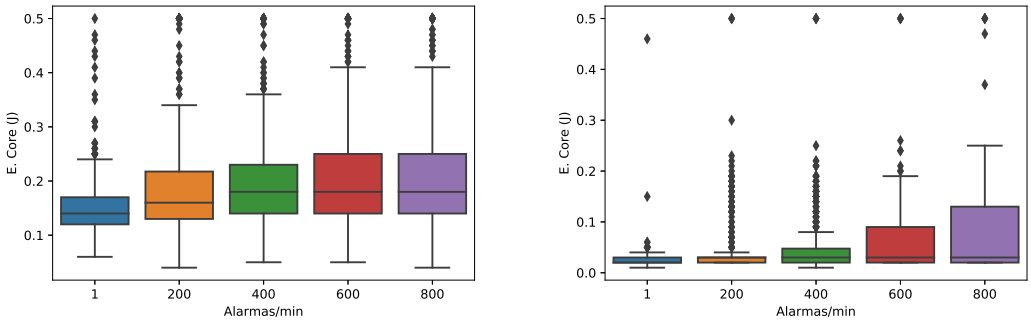


(a) RAM en Cloud Computing.

(b) RAM en Fog Computing.

FIGURA A.4: Boxplots del consumo de RAM de la Nube en Cloud y Fog Computing. Fuente: Elaboración propia.

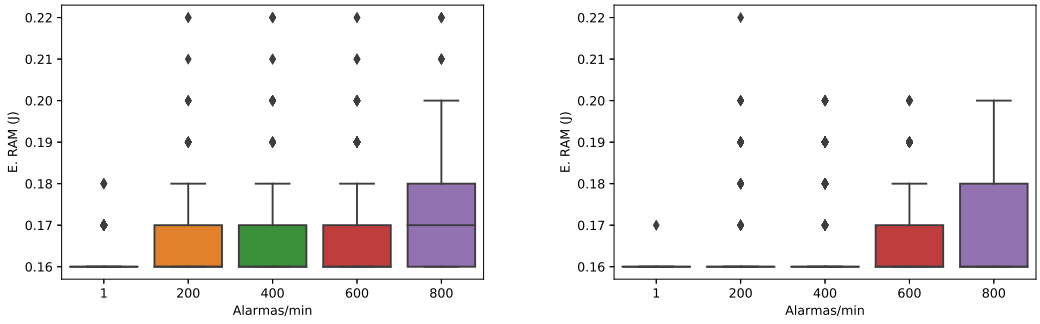
Para el caso de la energía consumida por los cores del CPU. Para Cloud Computing de la Figura A.5(a) se aprecia un aumento constante y similar de los valores teniendo varios picos en todos los tests. Del otro lado en la Figura A.5(b) en Fog Computing tiende a aumentar paulatinamente el consumo en su último cuartil y esto es al breve tiempo en que llegan los valores publicados y alarmas al broker ya luego no hay más trabajo.



(a) E. Cores en Cloud Computing. (b) E. Cores en Fog Computing.

FIGURA A.5: Boxplots del consumo en Joules (J) de los Cores de la Nube en Cloud y Fog Computing. Fuente: Elaboración propia.

Las Figuras A.6(a) y A.6(b) son del consumo de las RAM en estas se logra apreciar mejor que la RAM como mínimo consume 0.16J y según el trabajo sobre esta va aumentando. En este par de gráficos se aprecia mejor la acumulación y aumento de la energía en Cloud vs Fog Computing.

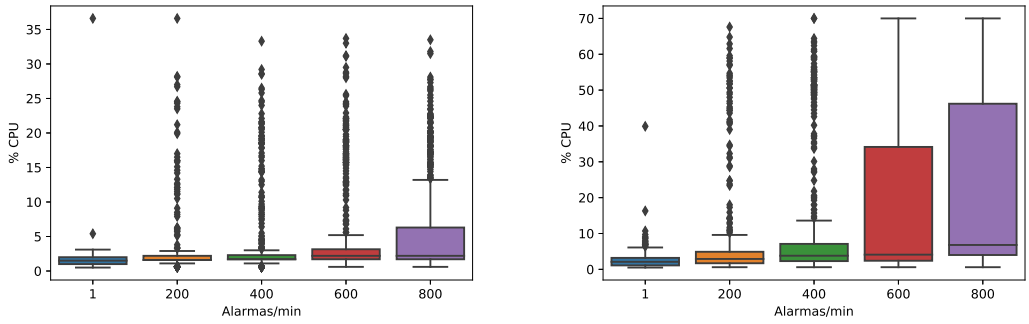


(a) E. RAM en Cloud Computing. (b) E. RAM en Fog Computing.

FIGURA A.6: Boxplots del consumo en Joules (J) de la RAM de la Nube en Cloud y Fog Computing. Fuente: Elaboración propia.

A.5. Nodo Edge

Ahora en esta última sección se comienza con el % CPU. En las Figuras A.7(a) y A.7(b) se logra ver que el nodo edge, al igual que el servidor, tiende a no consumir mucho de este reduciendo constantemente a razón de esto la mediana está muy cerca el primer cuartil.

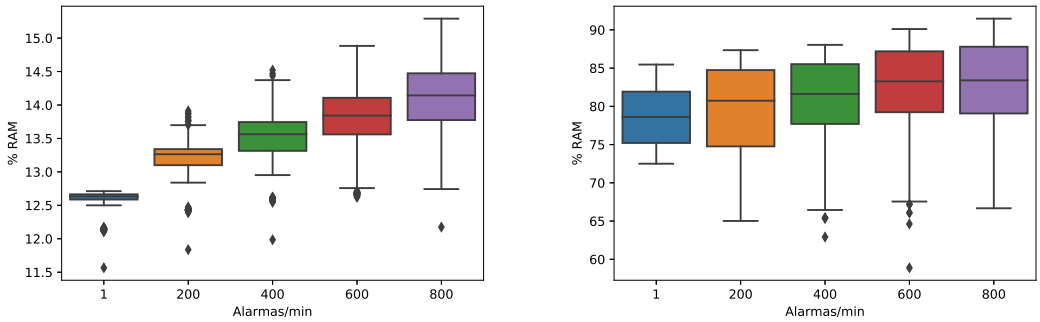


(a) % CPU en Cloud Computing.

(b) % CPU en Fog Computing.

FIGURA A.7: Boxplots del consumo de CPU del Nodo Edge en Cloud y Fog Computing. Fuente: Elaboración propia.

Finalmente en las Figuras A.8(a) y A.8(b) el consumo de espacio de la RAM es un caso peculiar al generalmente tener la mediana casi a la mitad de la caja, estos nos dice que su trabajo es más enfocado sobre ciertas áreas ya utilizadas.



(a) % RAM en Cloud Computing.

(b) % RAM en Fog Computing.

FIGURA A.8: Boxplots del consumo de espacio RAM del Nodo Edge en Cloud y Fog Computing. Fuente: Elaboración propia.