Universidad Técnica Federico Santa María

Departamento de Informática

Valparaíso - Chile

“Arquitectura IoT de Bajo Costo para Redes de Sensores”

JAVIER EDUARDO CAMPOS HARO

Memoria para optar al Título de
Ingeniero Civil en Informática

Profesor Guía: Javier Cañas
Profesor Correferente: Leonardo Pizarro

Marzo - 2020

DEDICATORIA

Quiero dedicar este trabajo a mi familia, quienes ayudaron a que esto fuera posible, también quiero dedicarlo a mis amigos, compañeros y profesores de la universidad que me apoyaron.

AGRADECIMIENTOS

Quiero agradecer principalmente a mi familia, principalmente a mis padres Héctor Campos y Patricia Haro, es gracias a ellos que he tenido la oportunidad de estudiar esta carrera. También me han brindado todo el apoyo y facilidades que he necesitado para terminar mis estudios.

Quisiera agradecer también a mi profesor guía Javier Cañas, quien me ha enseñado muchas cosas en mi carrera y también me ha dado muchas oportunidades laborales como ayudante de cátedra. Además junto con Leonardo Pizarro, ambos demostraron un gran apoyo y preocupación por el desarrollo de esta memoria.

Agradezco también a Mayumi Kato, Chizuko Vera, Felipe López y Carolina Vivanco, quienes me han brindado un apoyo muy grande emocionalmente y me han brindado apoyo múltiples veces.

A mis compañeros más cercanos Diego Villegas, Martín Ulloa y Andrea Rodríguez, quisiera agradecerles por el apoyo en las horas de estudio y en los proyectos realizados, también por su humor y paciencia, la universidad hubiera sido más difícil y cansina sin su apoyo.

A mi círculo de amigos más cercano, Sergio Mora, Jorge Sepúlveda, Pablo Valenzuela y Alexis Bustos les agradezco por permanecer a mi lado y por darme un escape de relajación en momentos de mucho estrés en la universidad.

Quisiera también agradecer al Club de Rol Sansano, al grupo de Calistenia USM y a mi antiguo Sensei Sergio Nuñez por todas las instancias que pude compartir en la universidad, ya que me han enseñado muchas cosas y he podido compartir experiencias muy gratas e importantes en mi vida.

Por último, quisiera agradecer a mi antiguo profesor Lautaro Guerra, para mí fue un profesor ejemplar que me ayudó a tener una visión diferente de la vida y el mundo laboral.

RESUMEN

Resumen—El objetivo de esta memoria es definir una arquitectura de software y hardware escalable y modular, la cual será usada por distintas redes de sensores. La finalidad de esta arquitectura es proveer funciones de mantención, calibración y procesamiento de datos a la red de sensores, de modo que los sensores puedan ser configurados de forma remota y que estos entreguen datos estructurados. También debe tener algoritmos que permitan detectar y corregir el ruido en las mediciones.
Palabras Clave—IoT; Arquitectura; Sensores

ABSTRACT

Abstract—The objective of this thesis is to define a modular and scalable software and hardware architecture, which will be used by different sensor networks. The purpose of this architecture is to provide functions for maintenance, calibration and data processing to the sensor network, so that sensors may be remotely configured and it may return structured data. Also they must have algorithms that detects and proofread noise in the measurement.
Keywords— IoT; Architecture; Sensors

GLOSARIO

IoT (*Internet of things*): Es un término para englobar conceptos como redes de sensores inalámbricos, dispositivos "wearables" conectados a internet, sistemas embebidos de bajo poder, tráfico RFID, uso de teléfono para interactuar con el mundo real, dispositivos conectados a internet vía bluetooth por celulares, smart homes, autos conectados, y mucho más.[3]
WSN (*Wireless Sensor Networks*): Sigla para referirse a las redes de sensores inalámbricos, estas redes suelen usar dispositivos económicos, de bajo consumo energético, y suelen tener una amplia disponibilidad.[3]
SP (*Smartphone*): Es un tipo de teléfono móvil, con mayor capacidad de almacenar datos y de realizar actividades, semejante a la de una minicomputadora.[15]
NTP (*Network Time Protocol*): Protocolo de sincronización de tiempo en una red.[16]
STA/LTA (*Short Term Averaging / Long Term Averaging*): Algoritmo de *triggering* que ayuda a diferenciar entre una señal sísmica real y ruido. Realiza una comparación entre el promedio de ondas durante un corto periodo de tiempo y el promedio de ondas de un periodo largo.[18]
BUS: Es un sistema digital que transfiere datos entre los componentes de una computadora.
QoS (Quality of service): Representan el nivel de rendimiento que puede presentar un protocolo para el envío y recepción de paquetes.[7]
PWM: (*Pulse Width Modulation*): Es una técnica usada para convertir una señal digital a una analógica.
GPIO (*General Purpose Input/Output*): Es un sistema de entrada y salida de propósito general, es decir, consta de una serie de pines o conexiones que se pueden usar como entradas o salidas para múltiples usos.[17]
ADC/DAC (*Analogue to Digital Converter/Digital to Analogue Converter*): DAC es un dispositivo para convertir señales digitales con datos binarios en señales de corriente o de tensión analógica. Un dispositivo ADC realiza el procedimiento inverso.
I2C (Inter-Integrated Circuit): Es un protocolo de comunicación serial sincrónico, tiene la característica de conectar uno o múltiples esclavos a uno o múltiples maestros, también se caracteriza por usar solo dos cables para transmitir datos entre dispositivos.[19]
UART (Universal Asynchronous Receiver-Transmitter): Es un protocolo de comunicación asincrónico lo que significa que no hay una señal de *clock* para sincronizar, sino que utiliza una secuencia de bits para definir inicio y término de los paquetes de datos transferidos.[20]
SPI (Serial Peripheral Interface): Es un protocolo de comunicación síncrono que trabaja en modo full duplex para recibir y transmitir información, permitiendo que dos dispositivos pueden comunicarse entre sí al mismo tiempo utilizando canales diferentes o líneas diferentes en el mismo cable. [21]
JAX-RS: (Java API for RESTful Web Services) Interfaz de aplicación que provee soporte a la creación de sitios web.

ÍNDICE DE CONTENIDOS

ABSTRACT 3
ÍNDICE DE CONTENIDOS 5
ÍNDICE DE FIGURAS 7
ÍNDICE DE TABLAS 8
INTRODUCCIÓN 9
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA 10
1.1 OBJETIVO GENERAL 10
1.2 OBJETIVOS ESPECÍFICOS 11
CAPÍTULO 2: MARCO CONCEPTUAL 12
2.1 VISIÓN DE IOT 12
2.2 REQUERIMIENTOS Y BENEFICIOS DE UNA ARQUITECTURA 13
2.3 ARQUITECTURAS IOT Y WSN INVESTIGADAS 14
2.3.1 MODELO GENÉRICO DE ARQUITECTURA PROPUESTO PARA IOT 15
2.3.2 SISTEMA DISTRIBUIDO DE DETECCIÓN DE SISMOS USANDO UNA RED DE SENSORES INALÁMBRICA PARA ALERTA TEMPRANA 18
2.3.3 CONSTITUCIÓN DE LAS WSN 21
2.4 COMPARACIÓN DE PROTOCOLOS DE COMUNICACIÓN 23
2.5 COMPARACIÓN DE PLATAFORMAS DE HARDWARE EN UNA RED IOT 26
2.6 SEGURIDAD EN IOT 31
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN 35
3.1 DESCRIPCIÓN DE LAS CAPAS DE LA ARQUITECTURA 36
3.1.1 ADQUISICIÓN DE DATOS 36
3.1.2 PROCESAMIENTO DE DATOS Y ANÁLISIS 36
3.1.3 ENVÍO DE DATOS 36
3.1.4 PROCESAMIENTO DE EVENTOS 37
3.1.5 SEGURIDAD 37
3.1.6 GESTIÓN DE DISPOSITIVOS 38
3.2 ELECCIÓN DEL DISPOSITIVO 38
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN 41
4.1 ESPECIFICACIONES DE LA ARQUITECTURA 41

4.2 ESPECIFICACIONES DE LOS SENSORES Y ALGORITMOS ASOCIADOS.....41
4.3 ANÁLISIS DE LOS RESULTADOS 45
CAPÍTULO 5: CONCLUSIONES 54
5.1 CONCLUSIONES GENERALES.....54
5.2 TRABAJO FUTURO.....54
REFERENCIAS BIBLIOGRÁFICAS.....55
REFERENCIAS BIBLIOGRÁFICAS EN NORMA APA:.....55
ANEXOS 58

ÍNDICE DE FIGURAS

Figura 2.1: Diagrama de capas para un modelo de arquitectura IoT. 16
Figura 2.2: Arquitectura en tres niveles. 19
Figura 2.3: Proceso de detección de picos en un smartphone. 20
Figura 2.4: Aceleración neta obtenida del sismo Colombia 09/02/2013....20
Figura 2.5: Anatomía de un Mote. 22
Figura 2.6: Constitución de una WSN. 23
Figura 2.7: Throughput achieved by the protocols in the three reference scenarios. 25
Figura 3.8 Diagrama de la arquitectura de software....35
Figura 4.9: HC-SR04 ultrasonic sensor. 42
Figura 4.10: Sensor ultrasónico: Gráfico porcentaje de CPU v/s Tiempo. 45
Figura 4.11: Sensor ultrasónico: Gráfico de Memoria Virtual v/s Tiempo. 46
Figura 4.12: Sensor ultrasónico: Gráfico Flujo de Datos v/s Tiempo. 46
Figura 4.13: Acelerómetro: Gráfico porcentaje de CPU v/s Tiempo. 47
Figura 4.14: Acelerómetro: Gráfico de Memoria Virtual v/s Tiempo....48
Figura 4.15: Acelerómetro: Gráfico Flujo de Datos v/s Tiempo. 48
Figura 4.16: Cámara: Gráfico Porcentaje de CPU v/s Tiempo. 49
Figura 4.17: Cámara: Gráfico de Memoria Virtual v/s Tiempo. 50
Figura 4.18: Cámara: Gráfico Flujo de Datos v/s Tiempo. 50
Figura 4.19: Datos del acelerómetro sin filtrar....51
Figura 4.20: Datos del acelerómetro usando filtro de mediana. 52
Figura 4.21: Datos del acelerómetro usando filtro pasa bajos y de mediana. 52
Figura 4.22: Gráfico de volumen de datos generado. 53

ÍNDICE DE TABLAS

Tabla 2.1: Clasificación de protocolos middleware Pub/Sub. 24
Tabla 2.2: Visión general de los sistemas Pub/Sub. 25
Tabla 2.3: Una comparación de tarjetas y plataformas en términos de computación. 29
Tabla 2.4: Comparación de tarjetas y plataformas en términos de ambientes de desarrollo y estándares de comunicación. 30
Tabla 2.5: Una comparación de tarjetas y plataformas en términos de conectividad. 31
Tabla 3.6: Tabla comparativa de dispositivos. 39

INTRODUCCIÓN

A lo largo del tiempo en la informática se han creado diversos tipos de redes, éstas han evolucionado debido a las múltiples necesidades que han surgido. Entre ellas tenemos un tipo de red consistente en múltiples sensores interconectados, estas son las llamadas *Wireless Sensor Network* (WSN), las cuales son difíciles de mantener debido a su rápido crecimiento de dispositivos y volumen de datos generado. Ante los problemas que surgen para la mantención de estas redes, se ha propuesto la creación de una arquitectura de software y hardware que permita suplir las falencias presentes. Esta arquitectura deberá asegurar la correcta mantención de sensores, también debe mejorar el procesamiento de datos y la comunicación entre los dispositivos que conforman las WSN. Al ser esta arquitectura IoT un sistema distribuido, se planea cumplir algunos requerimientos importantes para su correcto funcionamiento. Dichos requerimientos están relacionados con la eficiencia de la red, ya que al ser usada por dispositivos IoT se debe asegurar un bajo consumo eléctrico, además debe garantizar eficiencia en términos de tiempo, de forma que las funcionalidades no interfieran ni generen mayor trabajo al sistema, además de estas exigencias hay que tener en cuenta otros aspectos importantes como lo son la conectividad, escalabilidad, análisis de datos y seguridad, éstos deberán ser considerados en la arquitectura.

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA

En un sistema de monitoreo es importante contar con diversos sensores que permitan percibir el entorno en tiempo real, pudiendo de esta forma conocer su estado actual, y así poder informar y tomar acciones pertinentes cuando sea necesario. El problema que tienen los sensores en general es que tienen deficiencias que impiden su uso óptimo, entre éstas tenemos las siguientes:

- Comunicación simplex (one way): Algunos de los sensores que usan en las WSN son netamente emisores, no pueden recibir información ni instrucciones.
- Falta de preprocesamiento de los datos: Muchas veces es necesario dar algún formato específico a la información generada por los sensores o modificar estos datos para tener mediciones calibradas, actualmente los sensores no pueden realizar preprocesamiento de datos.
- Funcionalidades limitadas: Hay ciertas funcionalidades que es deseable tenerlas en la red de sensores y que no han podido ser implementadas.
- Mantención en el sitio: Todo tipo de mantención que se realice a la red de sensores debe ser realizada en el lugar en donde ésta se encuentre, no hay automatización.

En una WSN estas características son muy importantes, al contar con una arquitectura que permita solucionar estas falencias llevará al sistema a dar una mayor escalabilidad, es decir, podrá crecer de forma más controlada, ya que la mantención de los dispositivos podrá realizarse de forma automática, sin tener que llegar a mayores interrupciones en el sistema. También hará que el sistema sea más eficiente, ya que la información cumplirá con una estandarización que disminuirá problemas de datos y facilitará su uso.[1]

Cabe destacar de que a pesar de que en esta memoria se usarán acelerómetros como caso de uso para medir y comprobar el funcionamiento de la arquitectura a crear, ésta no se limitará al uso exclusivo de éstos, sino que buscará una integración general para pueda dar soporte a varios tipo de herramientas o sensores, como los mareógrafos, boyas o barómetros.

1.1 OBJETIVO GENERAL

El objetivo es definir una arquitectura de Software y Hardware para IoT que sea escalable, modular y de bajo costo para una red genérica de sensores, la cual debe permitir la realización de preprocesamiento de datos extraídos de los sensores.

1.2 OBJETIVOS ESPECÍFICOS

- Proponer un Middleware para la integración de datos recibidos.
- Proponer canales de comunicación óptimos bajo restricciones del sistema (tiempo, ancho de banda).
- Proponer un software que permita la gestión remota de los sensores.

CAPÍTULO 2: MARCO CONCEPTUAL

Como se indica a lo largo de este documento, se busca realizar una arquitectura de software y hardware IoT para una red de sensores. En esta sección se presenta una conceptualización e investigación teórica para la construcción de una arquitectura para IoT.

2.1 VISIÓN DE IOT

IoT es un término para englobar conceptos como redes de sensores inalámbricos, dispositivos "*wearables*" conectados a internet, sistemas embebidos de bajo poder, tráfico RFID, uso de teléfono para interactuar con el mundo real, dispositivos conectados a internet vía bluetooth por celulares, smart homes, autos conectados, y mucho más [3].

Ya que se ocupan muchos tipos de dispositivos diferentes y varios de estos tienen funcionalidades muy distintas entre sí, no existe una única arquitectura que pueda agrupar a todos los dispositivos existentes, sin embargo se puede crear algún tipo de arquitectura que permite añadir o sustraer capacidades a medida que sean necesarias [3].

Dentro de una red IoT podemos distinguir tres tipos de elementos distintos que la conforman, lo primero es el hardware, por lo general en una red de este tipo se encuentran sensores, actuadores y hardware de comunicación. Lo segundo es el middleware, encargado del almacenamiento y el análisis de datos. El tercer elemento es *presentation,* consiste en todas aquellas herramientas que ayuden a la interpretación y a la visualización de los datos [2].

Algunas de las tecnologías que apoyan el desarrollo de una red IoT son las siguientes:

- Radio Frequency Identification (RFID): Es la tecnología que permite el diseño de microchips para comunicación inalámbrica, además sirve en el reconocimiento de cualquier dispositivo con algún código de barras [2].
- Esquema de direccionamiento: Habilidad de identificar cada dispositivo de forma única en la red, permite que en la red haya unicidad, confiabilidad, persistencia y escalabilidad [2].
- Actualmente los algoritmos asociados al almacenamiento de datos ocupan mucha memoria, se busca mejorar el análisis de datos para que su almacenamiento ocupe menos memoria [2].

- Wireless Sensor Network (WSN): Es una red de sensores, este tipo de redes suelen estar compuestas de dispositivos de bajo costo monetario y de bajo consumo energético, a pesar de esto, estas redes tienen una buena disponibilidad, es decir, sus dispositivos están la mayor parte del tiempo activos en caso de ser necesario su uso. En una WSN se debe cumplir que cada elemento de la red sea identificado de forma única, para que los dispositivos puedan ser controlados remotamente por internet. En estas redes los sensores mandan información a un lugar centralizado, el cual se encarga del análisis de la información recibida, y también se debe encargar del almacenamiento de datos en caso de ser necesario [2].

  Los componentes de una WSN son [2]:

  a) WSN hardware: Por lo general contienen interfaz de sensores, unidades de procesamiento, unidades transceptoras y fuentes de poder.
  b) WSN communication stack: Pila que permite la comunicación entre la subred WSN y el internet.
  c) Middleware: Mecanismo que combina la ciber infraestructura con una arquitectura orientada al servicio (SOA) y una red de sensores para proveer acceso heterogéneo a los recursos de los sensores.

- *Secure Data aggregation*: Se requiere de métodos de agregación segura de datos para extender el tiempo de vida de la red (evitar pérdidas de paquetes o errores en la red), ya que las fallas en los nodos debido a dichos errores son muy comunes. Estos métodos también aseguran datos confiables recolectados de los sensores.

2.2 REQUERIMIENTOS Y BENEFICIOS DE UNA ARQUITECTURA

Para entender por qué es necesario una arquitectura, explicaremos los beneficios que esta trae a una red de sensores. Algunos de los valores que agrega una arquitectura son [3]:

1. Los dispositivos IoT están inherentemente conectados, una arquitectura facilita la implementación y configuración de Firewall, NAT y otros servicios de internet.
2. Garantizará una alta disponibilidad y permitirá la recuperación en caso de algún problema.
3. Muchos de los dispositivos no tienen incorporada UI y están diseñados para ser de uso diario, una arquitectura puede dar soporte automático y gestionado de actualizaciones, al igual que permite gestionar remotamente estos dispositivos
4. Gestiona la identidad y el control de acceso a los dispositivos IoT.

Dicho lo anterior, para que la arquitectura funcione correctamente debe de cumplir ciertos requerimientos de distintas categorías, a pesar de que en general depende del tipo de red IoT

que se esté implementando los requisitos que ésta debe cumplir, se pueden agrupar en las siguientes categorías [3]:

1. Conectividad y comunicación: La arquitectura debe poder realizar consultas básicas, como por ejemplo si es un protocolo HTTP debe poder hacer consultas GET, POST. Se debe tener cuidado con el overhead del protocolo a utilizar, ya que los dispositivos IoT trabajan con poca capacidad de almacenamiento y bajo consumo energético, por lo que la mejor opción es trabajar con protocolos binarios.
2. Gestión de los dispositivos: La arquitectura debe poder asegurar funciones como desconexión de dispositivos maliciosos o robados, actualización de dispositivos/software, actualización de credenciales de seguridad, habilitar/deshabilitar capacidades de hardware, re configurar remotamente WiFi, GPRS o parámetros de red.
3. Recolección y análisis de datos escalable: Por lo general los sistema de redes de sensores empiezan trabajando con una pequeña cantidad de sensores, los cuales captan la información del medio en el que están y la envían a centros de control para su almacenamiento y análisis. A medida que pasa el tiempo, el sistema va adquiriendo una mayor cantidad de componentes, lo que provoca una mayor complejidad para gestionar la red ya que cada vez el volumen de datos es mayor, y por lo tanto es más difícil almacenarla y analizarla. En una arquitectura IoT es muy importante que el sistema sea escalable, vale decir que pueda adaptarse y dar soporte al crecimiento de la red y al volumen de datos generado, debe analizar en tiempo real la información para el procesamiento de eventos en el sistema, y el sistema debe ser capaz de almacenar la información en la nube o en un datacenter propio.
4. Alta disponibilidad: Asegurar que el sistema se mantenga en todo momento funcionando.
5. Seguridad: Se debe ver la seguridad de la arquitectura en tres aspectos, seguridad de red, seguridad de los dispositivos, y seguridad de los datos. [3]

Los aspectos más importantes a considerar en esta arquitectura son la conectividad, gestión de dispositivos, escalabilidad y alta disponibilidad, por lo que otros aspectos como la seguridad y la recolección y análisis de datos quedará en segundo plano o serán trabajos futuros para la arquitectura.

2.3 ARQUITECTURAS IOT Y WSN INVESTIGADAS

Para tener una mejor noción del modelo a utilizar para nuestro sistema se ha investigado acerca de arquitecturas IoT y WSN que hayan sido propuestas o utilizadas, entre ellas se distinguen tres modelos, el primero es una propuesta de arquitectura IoT general, es decir, una arquitectura que funcione para distintos tipos de dispositivos, no solo sensores [3]. El segundo modelo consiste en una WSN utilizada para la detección de sismos, la cual opera con Smartphone [11]. La tercera es

una propuesta de arquitectura de un paper que estudia el estado del arte para distintos tipos de WSN [10].
En esta sección analizaremos distintos modelos de arquitectura propuestos a modo de contrastarlas con las necesidades de nuestro sistema. [3, 10, 11]

2.3.1 MODELO GENÉRICO DE ARQUITECTURA PROPUESTO PARA IOT

Este es un modelo de arquitectura IoT genérico el cual intenta agrupar distintas necesidades que poseen las arquitecturas IoT de modo que puedan dar con un modelo que permita agregar o sustraer características para obtener una base inicial para el desarrollo de una arquitectura específica. [3]

Las capas que usa la arquitectura sugerida en el texto las cuales son:

- *Client/external communications - Web/Portal, Dashboard, APIs*
- *Event processing and analytics (including data storage)*
- *Aggregation/bus layer – ESB and message broker*
- *Relevant transports - MQTT/HTTP/XMPP/CoAP/AMQP, etc.*
- *Devices*

Las capas cruzadas son:

- *Device manager*
- *Identity and access management*

Podemos apreciar en la figura 2.1 un diagrama de las capas de la propuesta de arquitectura mencionada.

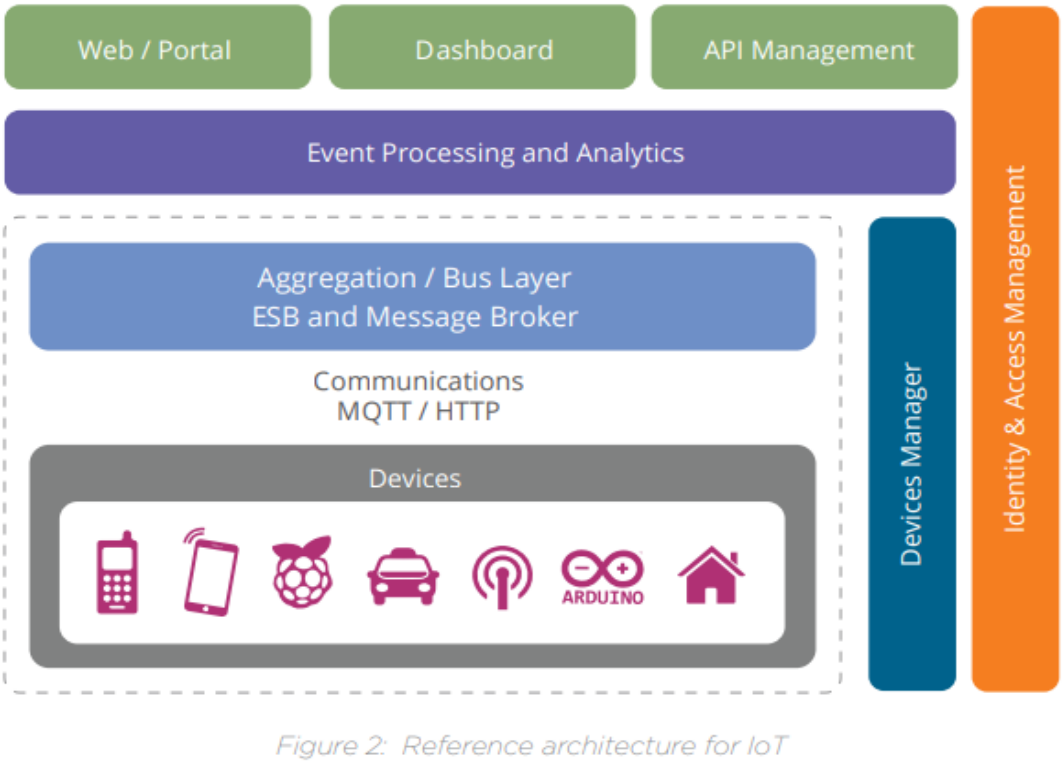


Figura 2.1: Diagrama de capas para un modelo de arquitectura IoT.
Fuente: A Reference Architecture for the Internet of Things. [3]

- *Device Layer*
  Es la capa de la arquitectura a más bajo nivel, los dispositivos pueden ser de distintos tipos, pero lo más importante es que todos puedan tener una conexión a internet directa o indirecta.
  Cada dispositivo debe tener un identificador único, este identificador puede ser un UUID (*Unique Identifier*) grabado en el dispositivo ya sea en el chip del dispositivo o en el subsistema de radio (ej. Identificador Bluetooth, WiFi MAC address). Adicionalmente a estos identificadores se puede agregar el protocolo de autenticación OAuth2 Refresh/Bearer Token [3].

- *Communication Layer*
  Es la capa encargada de soportar la conexión de los dispositivos, para esto se utilizan protocolos como HTTP, MQTT y/o CoAP.

  MQTT es un protocolo que tiene un *overhead* muy pequeño (2 bytes por mensaje), diseñado para conexiones intermitentes y para soportar pérdida de conexión, está diseñado para usar TCP, pero tiene una especificación para redes de sensores (MQTT-SN).

CoAP es un protocolo que provee un *RESTful application protocol*, modelado en semántica HTTP pero con *footprint* pequeños y basado en binario en vez de texto, diseñado para ser usado en UDP.[3]

La comunicación realizada es recomendable que se realice con el protocolo MQTT por los motivos anteriormente mencionados, (tener *overhead* muy pequeño, estar diseñado para conexiones intermitentes y tener implementación para redes de sensores).
También se recomienda este protocolo por su mejor adaptación y apoyo de mejores bibliotecas sobre otros protocolos como CoAP.[3]

- *Aggregation/Bus layer*
  Es la capa que se encarga de agregar o romper comunicaciones, es importante por 3 razones:
  1. La habilidad para soportar un servidor http y/o un MQTT broker para comunicarse con los dispositivos.
  2. La habilidad de agregar y combinar comunicaciones de diferentes dispositivos.
  3. La habilidad de conectar y transformar la comunicación entre distintos protocolos.

- *Device Manager*
  Capa encargada de comunicarse con los demás dispositivos, y así proveer control de éstos, debe mantener una lista de identificación de los dispositivos para manejar el control de acceso de cada uno.

  Un sistema completamente manejado soporta:
  - Gestionar el software del dispositivo.
  - Habilitar/deshabilitar características del dispositivo.
  - Gestionar el control de seguridad e identificación.
  - Monitorear la disponibilidad del dispositivo.
  - Mantener un registro de su posición.
  - Bloquear o limpiar el dispositivo remotamente en caso de que se vea comprometido.

- *Event Processing and Analytics Layer*
  Toma los eventos (mediciones) desde el bus y provee la habilidad de procesar y actuar en estos eventos tomados.
  La capacidad principal es el requerimiento de almacenar data en la base de datos.

El enfoque tradicional sería el de escribir una aplicación desde el lado del servidor, por ejemplo una aplicación **Jax-RS**[1] o **Django**[2] que esté respaldada por la base de datos, pero de todas formas se puede dar una perspectiva diferente, como una orientación de plataforma de análisis en Big Data, o también dar un enfoque en el soporte de procesamiento para eventos complejos.

Algunas recomendaciones para los enfoques:

- Altamente escalable, almacenamiento de datos en columnas para guardar eventos.
- Reducción mediante mapeo para procesamiento de datos de larga ejecución orientado a lotes.
- Procesamiento de datos complejos para mejorar la rapidez de procesamiento en memoria y alcanzar reacciones casi en tiempo real.

Adicionalmente esta capa puede soportar plataformas de procesamiento de aplicación tradicionales. (Java Beans, PHP, node.js, Python, etc.)[3]

2.3.2 SISTEMA DISTRIBUIDO DE DETECCIÓN DE SISMOS USANDO UNA RED DE SENSORES INALÁMBRICA PARA ALERTA TEMPRANA

Esta es una arquitectura de software creada para una red de sensores usada en un sistema de alerta sísmico.

En este sistema compuesto por smartphones (SPs) se hizo una arquitectura que consta de tres niveles. El primero consiste en la red de sensores de SPs, encargados de obtener los datos a procesar. El segundo nivel está compuesto de ordenadores con una mayor capacidad de procesamiento, su función es enviar y recibir la información de los sensores, a este nivel se le conoce como Intermediate Server (IS). Finalmente tenemos el centro de control (CC) que es el nivel encargado de mantener una visión global de la situación, también es el encargado de gestionar la arquitectura. [11]

1 https://jax-rs.github.io/apidocs/2.0/
2 https://www.djangoproject.com/

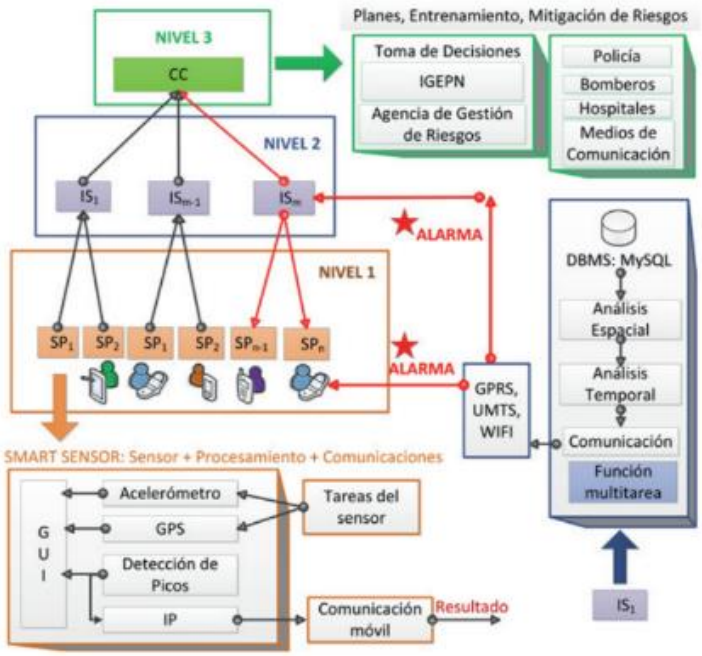


Figura 2.2: Arquitectura en tres niveles.
Fuente: Sistema Distribuido de Detección de Sismos Usando una Red de Sensores Inalámbrica para Alerta Temprana.[11]

Algunas características importantes de esta arquitectura:

1. Escalable: Aunque el número de dispositivos aumente, la arquitectura está diseñada para dar abasto a todos los sensores, ya que siempre se puede variar el número de IS para distribuir la carga.
2. Mantenimiento sencillo: Los usuarios son los encargados de actualizar la aplicación, estas actualizaciones toman pocos segundos
3. Confiable: En el caso de que uno de los dispositivos falle, la red se mantiene activa
4. Seguro: Usa MQTT sobre SSL junto con un mecanismo de autentificación de prefijos.
5. Sincronización: Usa NTP para la sincronización entre niveles.
6. Comunicaciones: Soporte de distintas tecnologías de comunicación (WiFi, GPRS, WIMAX, 3G o 4G).
7. Bajo costo energético.
8. Cobertura: Puede instalarse en cualquier espacio físico
9. Precisión: El sistema logra un 90% de precisión por el manejo de éste.

El software utilizado se enfocó en el bajo consumo, y en la facilidad de uso, se desarrolló en el sistema operativo Android, en la figura 2.3 se muestra un diagrama de los procesos de la aplicación para detectar picos de aceleración.[11]

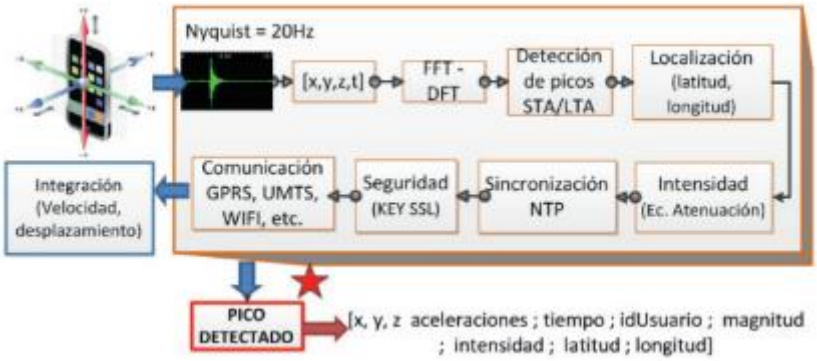


Figura 2.3: Proceso de detección de picos en un smartphone.
Fuente: Sistema Distribuido de Detección de Sismos Usando una Red de Sensores Inalámbrica para Alerta Temprana.[11]

Las señales sísmicas por lo general presentan componentes de frecuencia que van desde 1 hasta 10 Hz, por lo que el programa crea un proceso en *background* el cual recopila muestras siguiendo el teorema de muestreo de Nyquist, el cual enuncia que "la frecuencia de muestreo debe ser el doble que la máxima frecuencia contenida en la señal", por lo que la tasa de muestras debe ser de 20 muestras por segundo como mínimo. [11]
Los acelerómetros permiten medir los cambios de aceleración en cada eje (x, y , z), para obtener la aceleración neta se usa norma euclidiana y se remueve la aceleración de gravedad en el resultado obtenido. [11]

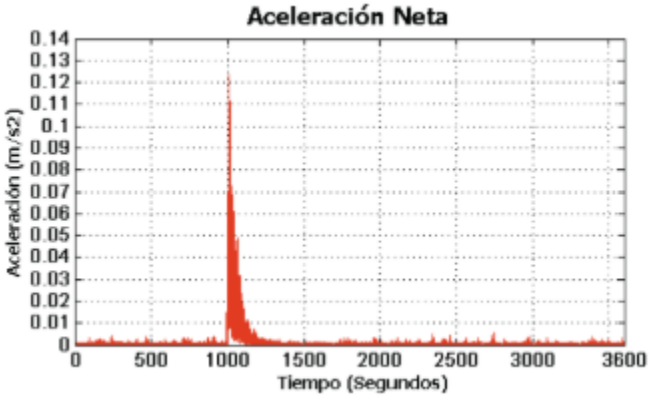


Figura 2.4: Aceleración neta obtenida del sismo Colombia 09/02/2013.
Fuente: Sistema Distribuido de Detección de Sismos Usando una Red de Sensores Inalámbrica para Alerta Temprana.[11]

$$a = \sqrt{x^2 + y^2 + z^2} - g \qquad \text{g = 980 gals}$$

Una vez tenemos la medición debemos considerar también el ruido en las mediciones, por lo que definimos un acelerograma como la unión de la señal sísmica y el ruido en función del tiempo, por lo que se usa la transformada discreta de Fourier (DFT) para cambiar del dominio del tiempo

a la frecuencia, el cual permite aplicar filtros pasa-bajos para eliminar altas frecuencias correspondientes al ruido. [11]
Posteriormente se usa *Short Term Averaging / Long Term Averaging* (STA/LTA), este algoritmo toma los valores de las últimas muestras entre todos los valores observados, STA como su nombre lo indica, toma un espectro corto de muestras, mientras que LTA toma las de un espectro más grande, dada las características de las muestras, podemos decir que en STA están contenidas las muestras de LTA. [11]
Se considera que se ha encontrado un pico sísmico si es que la relación entre el promedio de muestras obtenidas por STA/LTA supera cierto umbral, en el caso del proyecto desarrollado se hizo un umbral dinámico, el cual permite reconocer de mejor forma un sismo y diferenciarlo de cualquier ruido que pueda ocasionar el usuario con sus actividades diarias (caminar, correr, saltar, etc.). [11]
Si el algoritmo detecta un pico sísmico, accede al GPS para obtener la ubicación actual del usuario y mandar a un IS la información. Para mantener una correcta sincronización en la arquitectura se utiliza el protocolo NTP con un servidor situado en el CC, por último si los SP y el IS detectan un sismo, se enviarán alertas usando *Message Queue Telemetry Transport* (MQTT), protocolo de mensajería para notificaciones en tiempo real, se caracteriza por tener baja carga, uso mínimo de batería, envío mínimo de paquetes, distribución eficiente de la información, y facilidad para la conexión con otros dispositivos remotos. [11]

2.3.3 CONSTITUCIÓN DE LAS WSN

En este estudio se analizaron diversas arquitecturas de software en las cuales agruparon las principales características de las redes de sensores.

Las WSN son redes de características auto-configurables, compuesta por un pequeño número de nodos sensores, también llamados motes[3], distribuidos espacialmente y comunicados entre sí para lo que se usan señales de radio. Su finalidad es monitorizar y entender el mundo físico "para controlar diversas condiciones en distintos puntos, entre ellas la temperatura, el sonido, la vibración, la presión y movimiento o los contaminantes". [10]

El nombre "mote" asignado a los dispositivos se debe a que cumplen en una sola palabra dos conceptos principales, ser dispositivos de tamaño pequeño, y poder estar dispersos en cualquier parte que requiera la red. Esto es posible ya que los dispositivos son autónomos, funcionan con baterías similares a las de teléfonos celulares y que permiten ser cargadas por paneles solares en caso de ser necesario, además de esto, los dispositivos ocupan protocolos de bajo consumo, como lo es el caso de ZigBee, gracias al cual pueden pasar de un estado inactivo a un estado de transmisión en cualquier momento que sea requerido, evitando así desgaste de energía. Además

3 (Conjunto de comunicadores-Mote)

de esto, los motes pueden comunicarse entre sí gracias a la creación de *mesh networks*, usando protocolos como Zigbee y retransmisión de datos hasta un punto de control. [10]

Para que un mote pueda funcionar de forma correcta en una WSN debe tener ciertas características: [10]

- Contar con un procesador o microcontrolador que pueda alternar entre modos distintos (como mínimo dormido/ocioso).
- Tener una o varias fuentes de alimentación, ya sean baterías o paneles solares.
- Poseer una memoria para el almacenamiento de datos y almacenamiento de software.
- Establecer un radio utilizado para la comunicación y sensores que capten información del entorno.

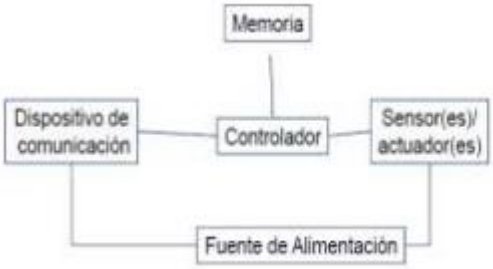


Figura 2.5: Anatomía de un Mote.
Fuente: Estado del arte de las redes de sensores inalámbricos.[10]

Una WSN está conformada por los siguientes elementos: [10]

- Sensores: Pueden ser de distintos tipos y tecnologías y su función es tomar la información presente en el medio y luego la convierten en señales eléctricas.
- Nodos de sensor: toman la información del sensor y la envían a la estación base.
- Gateway: Elementos para la interconexión entre la red de sensores y una red TCP/IP.
- Estación base: recolector de datos.
- Red inalámbrica: Recomendada que sea basada en ZigBee (estándar 802.15.4)

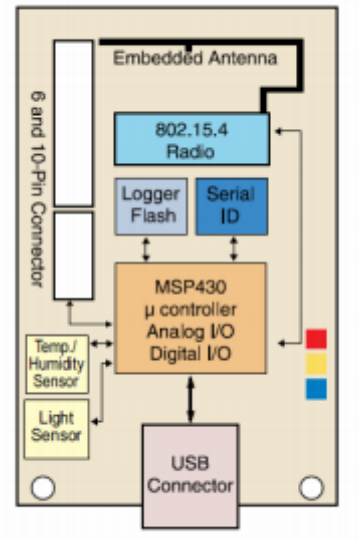


Figura 2.6: Constitución de una WSN.
Fuente: Estado del arte de las redes de sensores inalámbricos. [10]

2.4 COMPARACIÓN DE PROTOCOLOS DE COMUNICACIÓN

Los protocolos de comunicación son aquellos que pertenecen a la capa de aplicación del modelo TCP/IP el cual permite la gestión y transferencia de datos entre dos o más equipos en una red. A continuación se listan algunos de ellos: [4,5]

HTTP: Hypertext Transfer Protocol, es uno de los protocolos más usados en internet para la consulta de sitios web. Permite la transferencia de documentos de hipertexto (HTML) usando una arquitectura cliente-servidor. Este protocolo está basado en mensajes de texto plano, lo cual permite que sea legible e interpretable al momento de depurar, pero a la vez hace que sus mensajes sean ineficientes en tamaño. HTTP se basa en operaciones del cliente hacia el servidor, estableciendo ocho comandos, de los cuales destacan los más conocidos GET y POST, una vez respondida la consulta, el servidor HTTP cierra la conexión, HTTP no guarda sesiones ni puede mantener una comunicación bidireccional, ya que siempre es el cliente quien inicia el proceso.[5]
AMQP: Advanced Message Queuing Protocol es un protocolo de comunicación diseñado para proveer fiabilidad e interoperabilidad, se creó originalmente como un protocolo de comunicación para el sector bancario. Ofrece una amplia gama de funciones relacionadas con la integridad en la gestión de los mensajes, provee funciones como uso de colas, enrutamiento flexible y seguridad entre otras opciones. Cabe destacar que AMQP es un protocolo binario, usado principalmente cuando se busca estabilidad para el intercambio de información y que usa una arquitectura suscriptor-receptor.[4,5]

MQTT: Message Queue Telemetry Transport protocolo basado en el manejo sencillo y eficiente de mensajes, utiliza una arquitectura de suscriptor/receptor siendo pensado su uso para dispositivos con limitaciones de recursos o bajo condiciones de conectividad mínimas, se ha realizado un esfuerzo para que este protocolo tenga una implementación especial para redes de sensores (MQTT-S).[4,5]

XMPP: Extensible Messaging and Presence Protocol, protocolo de comunicación ampliamente utilizado, busca extensibilidad y adaptabilidad, se basa en el uso de XML lenguaje que permite almacenar información de manera legible y estructurada.[4,5]

ZeroMQ: Es una biblioteca que ofrece socket API con patrones de mensajes más avanzados que los berkeley sockets. Puede dar soporte a patrones de arquitectura request/response y pub/sub, el protocolo wire-level usado es ZeroMQ Message Transport Protocol (ZMTP) el cual está disponible bajo GNU General Public License.[4]

		AMQP	MQTT	XMPP	ZeroMQ
Messaging pattern	Pub/Sub	✓	✓	✓[1]	✓
	Point-to-point	✓		✓	✓
Filtering	Topic-based	✓	✓	✓[1]	✓
	Content-based				
QoS semantics	At-most-once	✓	✓	✓	✓
	At-least-once	✓	✓		
	Exactly-once	✓	✓		
	Last value caching	✓[2]	✓	✓	✓
Topology	Decentralized				✓
	Centralized	✓	✓	✓	✓
	Hybrid			✓[1]	
Message format	Payload agnostic	✓	✓	✓	✓
	Binary encoding	✓	✓		✓

[1]using XMPP Extension Protocols (XEPs)
[2]not required by standard, but mostly available via plugin

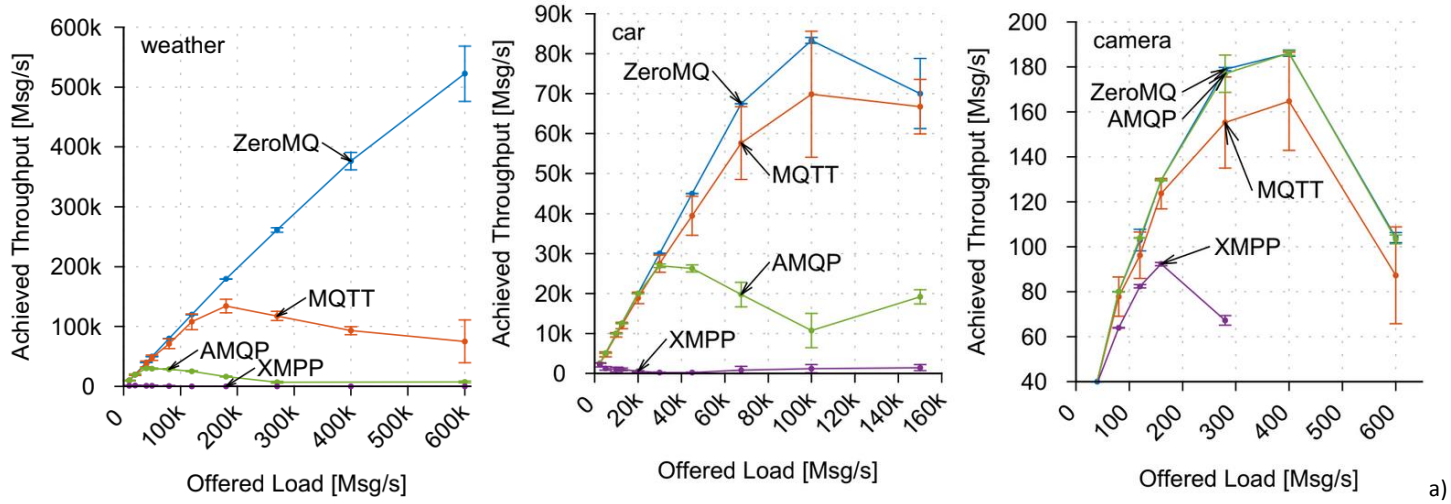
Tabla 2.1: Clasificación de protocolos middleware Pub/Sub.
Fuente: Meeting IoT platform requirements with open pub/sub solutions.[4]

Se realizaron mediciones de desempeño a los protocolos AMQP, MQTT, XMPP y ZeroMQ, estas mediciones se realizaron en un entorno de trabajo real en el cual pusieron a prueba los protocolos en un broker desarrollado en Amazon's Elastic Compute Cloud (EC2). El detalle de los brokers se puede apreciar en la tabla 2.2 [4].

Protocol	Broker	Version	Release date	Language
AMQP	RabbitMQ	3.4.4	2015-02-11	Erlang
MQTT	mosquitto	1.4	2015-02-18	C
XMPP	ejabberd	15.02	2015-02-17	Erlang
ZeroMQ	XPUB/XSUB	–	–	C

Tabla 2.2: Visión general de los sistemas Pub/Sub.
Fuente: Meeting IoT platform requirements with open pub/sub solutions.[4]

Las pruebas se realizaron en tres tipos de sensores distintos, los cuales son sensores simples, sensores complejos y sensores multimedia.
Los resultados revelaron que en mediciones de desempeño ZeroMQ mostraba una ligera ventaja en comparación a AMQP y MQTT, ventaja que se ve mayor en sensores simples. En cuanto a retrasos, AMQP, MQTT y ZeroMQ mostraron un desempeño similar, por lo que no se puede concluir que alguno de los protocolos es superior en este aspecto, sin embargo podemos concluir que XMPP en general es inferior a los demás protocolos tanto en desempeño como en retrasos, esto es debido a que XMPP es un protocolo basado en texto, lo que hace que sus mensajes sean menos eficientes que los protocolos binarios [4].



Desempeño vs carga del sensor meteorológico b) Desempeño vs carga compartida del coche c) Desempeño vs carga de la cámara

*En el eje de abscisas tenemos la carga de mensajes por unidad de tiempo, y en el eje de ordenadas tenemos el desempeño, es decir la carga útil por unidad de tiempo.

Figura 2.7: Throughput achieved by the protocols in the three reference scenarios.
Fuente: Meeting IoT platform requirements with open pub/sub solutions[4]

Además de este estudio tenemos otro que analizó los protocolos MQTT y XMPP desde los puntos de vista de la eficiencia del protocolo (bytes totales v/s nº de mensajes), consumo energético (Joule v/s nº de mensajes) y consistencia de la información (% mensajes perdidos v/s % de nivel de pérdida de datos interfaz red de servidor). Los resultados revelaron que MQTT es un protocolo más eficiente de transmisión de datos, independiente del QoS que se ocupe en MQTT, los resultados siempre favorecieron a este protocolo. En cuanto a la eficiencia energética los resultados indicaron que no se puede inferir una superioridad clara entre alguno de los protocolos, sin embargo, bajo ciertas circunstancias se mostraron ciertas tendencias, XMPP mostró ser más eficiente para dispositivos recibiendo información constantemente, mientras que MQTT QoS 1 mostró ser más eficiente en los envíos constantes de datos, y en general MQTT es mejor cuando hay un flujo de información pequeño, ya sea enviando o recibiendo información. Por último, los resultados revelaron que XMPP es un protocolo con menos tasas de fallos que MQTT, por lo que concluimos que en caso de requerir una red con una mejor consistencia de datos, el mejor protocolo a usar es XMPP[5].
Para ver de manera más gráfica estos resultados es recomendable ver el anexo 1, en donde se encuentran gráficos generados por este estudio.

2.5 COMPARACIÓN DE PLATAFORMAS DE HARDWARE EN UNA RED IOT

Muchas veces al pensar en hardware para una red IoT viene a la mente dispositivos como *smartphones*, ya que éstos poseen sensores, monitores y direcciones únicas al conectarse a internet, pero los dispositivos que pueden pertenecer a una red IoT son muchos y muy variados. Con el paso del tiempo lo dispositivos *"smart"* han ido en aumento y se cree que a futuro casi todas las cosas estarán conectadas a internet. Los dispositivos inteligentes en una red IoT pueden ser descritos como la parte "things" o "cosas" dentro de la red IoT, y esta parte es la encargada de proveer información en la interacción de la red.
Las formas de desarrollar software y hardware para este tipo de redes es casi infinita, a pesar de esto podemos agrupar los dispositivos en dos grandes categorías, de esta forma será más fácil establecer un contexto de desarrollo. Las dos grandes categorías son *"Wearables devices and Gadgets"* y *"Embedded Systems and Boards",* de las cuales nos concentraremos en la segunda. Esta se categoriza en que tanto el software como el hardware están abiertos para los desarrolladores, por lo general estos sistemas se usan para realizar control de dispositivos, adquisición de datos o desarrollo de aplicación[7].

Raspberry Pi 2: Es un computador de placa simple el cual usa un núcleo quad-core ARM7 800 MHz, un Videocore IV 250MHz como unidad de procesamiento gráfico, 40 pines GPIO, 4 puertos USB 2.0, un puerto Ethernet, un conector HDMI y una ranura para tarjeta micro-SD, opera de forma similar a un computador promedio, requiere de un teclado, monitor y fuente de poder. Puede correr diversos sistemas operativos, incluyendo Raspbian Linux, Ubuntu Mate y

Windows 10 IoT Core, siendo el primero el más usado al ser *open source* y estar diseñado principalmente para ser usado en Raspberry. También da soporte a varios lenguajes de programación como C/C++, Python y JavaScript. Algo importante a destacar es que la comunidad que da soporte a Raspberry es actualmente muy grande, se han creado múltiples proyectos gracias al amplio rango de uso que tiene[6,7].

Arduino: Es un vasto rango de tarjetas open-source, capaces de realizar tareas que van de hacer parpadear LED hasta publicar material online. Esto es posible gracias a ambiente de desarrollo de Arduino (IDE).
Arduino tiene un amplio número de tarjetas, desde el simple microcontrolador 8-b hasta tarjetas usadas en productos usables (*wearables*), IoT, impresiones 3D, y mucho más. Para IoT, la compañía ofrece Arduino Yun, la cual tiene integrado Wi-Fi y Ethernet. Tiene un ATmega32u4 con una velocidad de 16 MHz y un Atheros AR9331 (MIPS @ 400MHz), el cual respalda una distribución de Linux llamada OpenWrt-Yun. Además de esto la tarjeta contiene una ranura para tarjetas microSD, un puerto USB, 7 pines PWM, 12 entradas análogas y una conexión micro USB. Los pines GPIO de la Atheros AR9331 no son accesibles como lo son los que están conectados a la ATmega32u4 [7].

BeagleBone Black: Es una plataforma que recibe soporte de la comunidad de las plataformas BegleBoard. Tiene un procesador TI Sitara AM3358 ARM Cortex-A8 que corre a 1 GHz, además tiene 4 GB de memoria flash, 512 MB de DDR3L DRAM, un acelerador gráfico 3D, dos cabeceras de 46 pines y un puerto ethernet. Puede dar soporte a los sistemas operativos Debian, Android y Ubuntu [7].

Pinoccio: Es una placa open source *full-packed* para IoT, viene con un procesador Atmel ATmega256RFR2, el cual tiene una velocidad de 2.4 GHz, en cuanto a conectividad, tiene 17 pines para I/O digital, Wi-Fi y 802.15.4, también soporta redes de área personal inalámbricas, como lo son ZigBee, Bluetooth, ISA100.1 Ia. También da soporte a redes mesh sin hacer uso de internet [7].
Usa una batería recargable Li-Po (550 mAh) la cual permite trabajar de forma continua por 27 horas. Pinoccio es totalmente compatible con Arduino, pudiendo ocupar sus bibliotecas en el desarrollo de proyectos IoT.

Udoo Quad: Es un computador de placa simple, basado en open hardware, posee un prosesador ARM i.MX6 Freescale y un ATMEL SAM3X8E ARM (Compatible con Arduino), este dispositivo se hizo para combinar las capacidades de Raspberry y Arduino. Udoo Ofrece 76 pines GPIO totalmente disponibles, un módulo Wi-Fi integrado, un módulo Ethernet, LVDS + Touch y 1 GB de memoria RAM DDR3[7].

Cloudbits/Littlebits: Es una plataforma que viene con un sistema basado en Linux, con un procesador i.MX23 ARM926EJ-S y tiene 64 MB de RAM. Hace uso de un adaptador USB para 802.11 b/g/n.
Cuando el dispositivo se conecta a WiFi comienza a enviar datos desde otros módulos littleBits a la nube sin necesidad de programar. Además de esto cuenta con soporte de muchas API [7].

Particle Photon: Este dispositivo cuenta con un núcleo ARM Cortex M3 de 120-MHz, con 128 KB de RAM, un chip Wi-Fi Broadcom BCM43362, cuenta además con 18 GPIO. Localmente puede ser programado por ARM development environment, y conectado a internet puede ser programado por Particle's Cloud IDE. Además, está diseñado para que pueda cambiar automáticamente entre antenas externas e internas para seleccionar la mejor señal [7].

Samsung's Artik 10: Viene con los procesadores ARM A15x4 de 1.3 GHz, y A7x4 de 1.0 GHz, además cuenta con 2 GB de memoria RAM y 16 GB de memoria estática. Viene con Yocto 1.6 (Fedora) OS y hardware de audio y video codec, tiene además 51 pines GPIO con SPI, inter-IC sound (I2S), I2C, UART, USB, Bluetooth de baja energía y más.

Teniendo en cuenta los dispositivos anteriormente nombrados, debemos hacer una comparación entre ellos para determinar las ventajas que cada uno tiene, los puntos a evaluar para determinar sus ventajas serán los siguientes:

1. Especificaciones:
   Las especificaciones de un proyecto juegan un rol vital en la selección de la plataforma de hardware para una aplicación IoT en particular, se deben considerar aspectos fundamentales como procesador/microcontrolador, clock speed, GPIO, ADC/DAC, conectividad (WiFi, Bluetooth o Ethernet), comunicación (ej. I2C, UART y SPI), y el precio del hardware.

Características	Procesador/Microcontrolador	Unidad de procesamiento gráfico	Clock Speed	Tamaño	Memoria	RAM	Voltaje	Precio
SparkFun Blynk Board	Tensilica L106 32-b	No	26 MHz	51 mm x 42 mm	4 MB	128 KB	5 V via micro-USB/ Li-Po connector and charging circuit	US$29.95
Arduino Yun	ATmega32u4 and Atheros AR9331 (for Linux)	No	16 MHz and 400 MHz	73 mm x 53 mm	32 KB and 16 MB + micro-SD	64 MB DDR2	5 V via micro-USB	US$58
Raspberry Pi 3	Broadcom BCM2837 and ARM Cortex-A53 64-b Quad Core	VideoCore IV @ 300/400 MHz	1.2 GHz	85 mm x 56 mm	Micro-SD	1 GB LPDDR2	5 V via micro-USB	US$35
cloudBit	Freescale i.MX233 (ARM926EJ-S core)	No	454 MHz	55 mm x 19 mm	Micro-SD slot with 4-GB micro-SD	64 MB	5 V via micro-USB	US$59.95
Photon	STM32F205 120Mhz ARM Cortex M3	No	120 MHz	36.5 mm x 20.3 mm	1 MB	128 KB	5 V via micro-USB	US$19
BeagleBone Black	AM335x ARM Cortex-A8	PowerVR SGX530	1 GHz	86 mm x 56 mm	4 GB 8-b eMMC, micro-SD	512 MB DDR3	5 V via mini-USB	US$49
Pinoccio	ATmega256RFR2	No	16 MHz	70 mm x 25 mm	256 KB	32 KB	5 V via micro-USB/ Li-Po connector and charging circuit	US$109
UDOO	Freescale i.MX 6 ARM Cortex-A9 and Atmel SAM3X8E ARM Cortex-M3	Vivante GC 2000 for 3-D + GC 355 for 2-D (vector graphics) + GC 320 for 2-D	1 GHz	110 mm x 85 mm	Micro-SD	1 GB DDR3	12 V	US$135
Samsung Artik 10	ARM A15x4 and A7x4	Mali-T628 MP6 core	1.3 GHz and 1.0 GHz	39 mm x 29 mm	16 GB	2 GB LPDDR3	3.4–5 V	US$100

Tabla 2.3: Una comparación de tarjetas y plataformas en términos de computación.
Fuente: Create Your Own Internet of Things: A survey of IoT platforms. [7]

2. Open API:

Es un aspecto esencial a analizar para evitar dificultades en el desarrollo de software, sobre todo para desarrolladores independientes, por lo que es importante que los dispositivos cuenten con una comunidad que de soporte y también que cuente con bibliotecas abiertas, entre más grande sea la comunidad del dispositivo y entre más bibliotecas abiertas tenga esta hará que los desarrolladores independientes realicen aplicaciones en menos tiempo y tengan un uso de recursos más eficiente.

Características	Sistemas Operativos Soportados	IDE	Estandar de Video	Puertos de Video/Audio	Puertos USB	Comunicación
SparkFun Blynk Board	No	Arduino, Blynk	No	No	No	I2C, SPI, UART
Arduino Yun	OpenWrt-Yun (based on GNU/Linux)	Arduino	No	No	1x USB 2.0	I2C, UART
Raspberry Pi 3	Raspbian, Windows 10 IoT Core, OpenELEC, OSMC, Pidora, Arch Linux, RISC OS, Ubuntu	C#, Python, Java, Scratch and many more	MPEG-2,VC-1,H. 264 AVC (1080p @ 30 fps)	HDMI 1.4 with CEC, 4-pole 3.5-mm connector, Raw LCD (DSI)	4x USB 2.0	1x SPI, 2x I2C, PCM/I2S, 1x UART
cloudBit	Customized Arch Linux ARM distribution	Cloud API, Arduino	No	No	No	UART
Photon	FreeRTOS	Particle Build (Online), Particle Dev (Local)	No	No	No	2x SPI, 1x I2S, 1x I2C, 1x CAN, 1x UART
BeagleBone Black	Debian, Android, Ubuntu	C++, Perl, Python, Cloud9 IDE	NEON software decoding support	Micro-HDMI	1x USB	4xUART, 2x SPI, 2x I2C, 2x CAN BUS
Pinoccio	No	Arduino, Pinoccio HQ (Online/ Off-line)	No	No	No	I2C, SPI, 2x UART
UDOO	UDOObuntu, Android, XMBC, Yocto, Arch Linux, OMV	Arduino	MPEG-2, H.264 (1080p60)	HDMI, Analog audio and mic jacks, LVDS + Touch	1x USB OTG, 2x USB 2.0, 1x USB to Serial	SPI, I2C, UART, CAN BUS
Artik 10 Samsung	Yocto 1.6 (Fedora)	Arduino IDE, Samsung SDK, C/C++, Java, Groovy	1080p @ 120 fps H.263/H.264/ MPEG-4/VP8 + MPEG-2/VC1 decoding	Four-lane MIPI DSI up to WUXGA, HDMI, one-channel PCM and two-channel I2S audio interface	1x USB2.0, 1x USB3.0	1x SPI, 6x I2C, 1x I2S, 3x UART

GNU: GNU's not Unix; OSMC: Open-Source Media Center; RISC OS: reduced-instruction-set computer operating system; CEC: consumer electronics controller; LCD: liquid crystal display; DSI: digital serial interface; PCM: pulse code modulation; CAN BUS: controller area network; NEON: Nonprofit Enterprise Online Network; XMBC: Xbox Media Center; OMV: Open-Media Vault; LVDS: low-voltage differential signaling; OTG: on-the-go; MIPI: mobile industry processor interface; WUXGA: wide ultra-extended graphics array.

Tabla 2.4: Comparación de tarjetas y plataformas en términos de ambientes de desarrollo y estándares de comunicación.
Fuente: Create Your Own Internet of Things: A survey of IoT platforms. [7]

3. Open Hardware:

Este punto es importante para llevar lo que es un prototipo a un proyecto que lo convierta en un producto. Una plataforma con soporte de hardware será mucho más manejable en el desarrollo y aplicación del producto.

Características	GPIO	PWM Pins	Analog Pins	ADC/DAC	Ethernet	Wi-Fi	Bluetooth	Interfaz de Cámara	Sensores y Display integrados
SparkFun Blynk Board	9	3	1	10 b	No	IEEE 802.11 b/g/n	No	No	Temperature and humid-ity sensor, RGB LED
Arduino Yun	20	7	12	10 b	IEEE 802.3 10/100 Mb/s	IEEE 802.11 b/g/n	No	No	No
Raspberry Pi 3	26	2	No	No	IEEE 802.3 10/100 Mb/s	IEEE 802.11 b/g/n	Bluetooth 4.1 LE	CSI	No
cloudBit	No	No	No	No	No	IEEE 802.11 b/g/n	No	No	No
Photon	18	9	8	12-b ADC, 12-b DAC	No	IEEE 802.11 b/g/n	Yes	No	Real-time clock
BeagleBone Black	69	8	7	12-b ADC	IEEE 802.3 10/100 Mb/s	No	No	No	No
Pinoccio	17	4	8	10-b ADC	No	Wi-Fi Backpack IEEE 802.11 b/g/n	No	No	Temperature sensor, RGB LED
UDOO	76	13	12-ADC, 2-DAC	12-b ADC, 12-b DAC	IEEE 802.3 10/100 Mb/s	IEEE 802.11 b/g/n	No	CSI	No
Artik 10 Samsung	51	2	6	Six-channel	No	IEEE 802.11 b/g/n	Bluetooth 4.1 LE	1x two-lane MIPI CSI, 1x four-lane MIPI CSI up to 23 MP still, 8 MP @ 30 fps	No

fps: frames per second.

Tabla 2.5: Una comparación de tarjetas y plataformas en términos de conectividad.
Fuente: Create Your Own Internet of Things: A survey of IoT platforms. [7]

2.6 SEGURIDAD EN IOT

A pesar de que el crecimiento de los dispositivos IoT ha sido un gran avance tecnológico que ha revolucionado la infraestructura del internet, es también una gran vulnerabilidad, ya que la gran cantidad de dispositivos conectados podrían ser dispositivos vulnerables que eventualmente serán una amenaza contra una red IoT ya que los hackers pueden usar estas vulnerabilidades para hacer un mal uso del sistema. IoT tiene un gran potencial por flexibilidad, lo que promete un mejor futuro, pero también puede prometer un gran desastre si las vulnerabilidades no son tratadas [12].

Para poder categorizar y reconocer mejor las amenazas, se propone una arquitectura genérica la cual tiene 4 capas:

Perception Layer: Es la capa consistente en la obtención de datos por parte de sensores, algunos de los ataques que se pueden realizar a esta capa son [12]:

- *Unauthorized Access to the Tags:* Un usuario sin autorización logra acceder a los datos pudiendo leerlos, modificarlos o borrarlos.
- *Tag Cloning:* Un tag en un sistema RFID es un sistema de seguimiento utilizado para identificar artículos, *Tag Cloning* Consiste en usar una réplica de un tag para que el lector no pueda diferenciar entre la data real y la réplica.
- *Eavesdropping:* Consiste en capturar el tráfico de una red IoT para obtener datos sensibles como contraseñas.
- *Spoofing:* Consiste en enviar información falsa haciendo pasar ésta como si viniera de una fuente original.
- *RF Jamming:* Consiste en crear ruido en la red para que los dispositivos sean interrumpidos por exceso de ruido.

Network Layer: Es la capa encargada de transmitir los datos obtenidos a cualquier sistema de procesamiento de información. Algunos ataques que pueden afectar esta capa son [12]:

- *Sybil Attack:* Consiste en manipular un nodo de forma que este presente múltiples identidades y así generar fallas por información redundante.
- *Sinkhole Attack:* Consiste en convencer a los demás nodos para mandar sus datos a otro nodo específico, lo que resulta en una pérdida de datos.
- *Sleep Deprivation Attack:* Consiste en mantener un nodo despierto para que su batería se gaste más rápidamente y termine apagándose.
- *Denial of Service (DoS) Attack:* Consiste en saturar la red de peticiones para que deje de estar disponibles para los usuarios.
- *Malicious code injection:* Consiste en inyectar código en alguna entrada del sistema, de forma que un intérprete sin darse cuenta puede ejecutar una instrucción del atacante.
- *Man-in-the-Middle Attack:* Es un ataque que consiste en interceptar el tráfico de la red, en este caso el atacante no solo puede leer los mensajes, sino que también puede hacerse pasar por la víctima.

Middleware Layer: Es la capa encargada del procesamiento de los datos y de tomar decisiones automáticamente a partir de éstos. También se encarga de enlazar el sistema con la base de datos. Algunos ataques que la pueden afectar son [12]:

- *Unauthorized Access:* Un fallo en la seguridad puede desconectar a los nodos o borrar la información del sistema.
- *DoS Attack:* Consiste en saturar la red para que el middleware deje de funcionar.
- *Malicious Insider:* Consiste en que alguno de los nodos manipula los datos para beneficiarse o beneficiar a un tercero.

Application Layer: Es la capa encargada de realizar un servicio al usuario. Algunos ataques que se pueden realizar son [12]:

- *Malicious Code Injection*: Es el mismo ataque que se puede realizar en la capa *Perception Layer.*
- *Denial-of-Service (DoS) Attack*: Es el mismo ataque realizado en las capas anteriores.
- *Spear-Phishing Attack*: Consiste en el envío de correo fraudulento para hacer creer a la víctima que lo están contactando de la aplicación para poder robarle información como sus credenciales.
- *Sniffing Attack*: Consiste en introducir una aplicación que capture el tráfico de la red para obtener información.

Las sugerencias planteadas para solucionar estos ataques son las siguientes:

1. Autenticación: Usando algoritmos de hash se puede proveer a los dispositivos de firmas digitales que evitan los ataques de fuerza bruta y suplantación de identidad.
2. Encriptación: Al usar encriptación simétrica y asimétrica aseguramos la privacidad de los datos y evitamos que puedan ser interceptados.
3. Evaluación de riesgos: Es bueno buscar amenazas en el sistema para saber qué estrategia de seguridad se puede tomar.
4. Algoritmos de ruteo: Se debe asegurar unos buenos algoritmos de ruteo que permitan llevar los datos de manera segura hasta su destino, también se debe asegurar de que haya múltiples formas de que los datos puedan llegar al destinatario.
5. Mecanismos de control: Es importante que haya mecanismos que monitoreen las actividades de los usuarios, para que al detectarse una anomalía se pueda generar una alarma, también es importante que estos mecanismos puedan detectar intrusos [12].

Una posibilidad para implementar seguridad en dispositivos IoT es utilizar la tecnología Blockchain. Ésta consiste en un registro digital contable descentralizado en el cual, cada peer en la red posee una copia del ledger. [32]

Blockchain posee una serie de cualidades que lo hace ser un sistema escalable y seguro, sus principales características son:

- Inmutabilidad: No se pueden alterar transacciones pasadas ya que las cadenas de bloques se administran por consenso.
- Seguridad: Cada bloque en Blockchain esta encriptado con el hash anterior (SHA256) y variables del propio bloque, por lo que la encriptación realizada es muy fuerte.
- Verificabilidad: Al ser transparentes e inmutables, cualquier persona en el mundo puede comprobar que se cumplan las reglas del sistema.

- Transparencia: Ya que todas las transacciones se transmiten a toda la red, hacen que el ledger sea transparente, sin embargo al estar las transacciones encriptadas aseguran también la privacidad.

En la arquitectura propuesta e implementada en [32] se ha demostrado la eficacia de Blockchain para dar seguridad y privacidad de datos, además de agregar un nivel extra de seguridad que da una confiabilidad mayor a las transacciones realizadas en dicha arquitectura. Lo cual demuestra que Blockchain puede ser usada como una tecnología para proteger la información.

CAPÍTULO 3: PROPUESTA DE SOLUCIÓN

Basándose en los modelos de arquitectura IoT investigados al principio de esta memoria junto con algunas otras referencias[29, 30] se ha llegado a una decisión de un modelo de 6 capas, de las cuales 5 son capas principales y 1 es una capa intermedia.

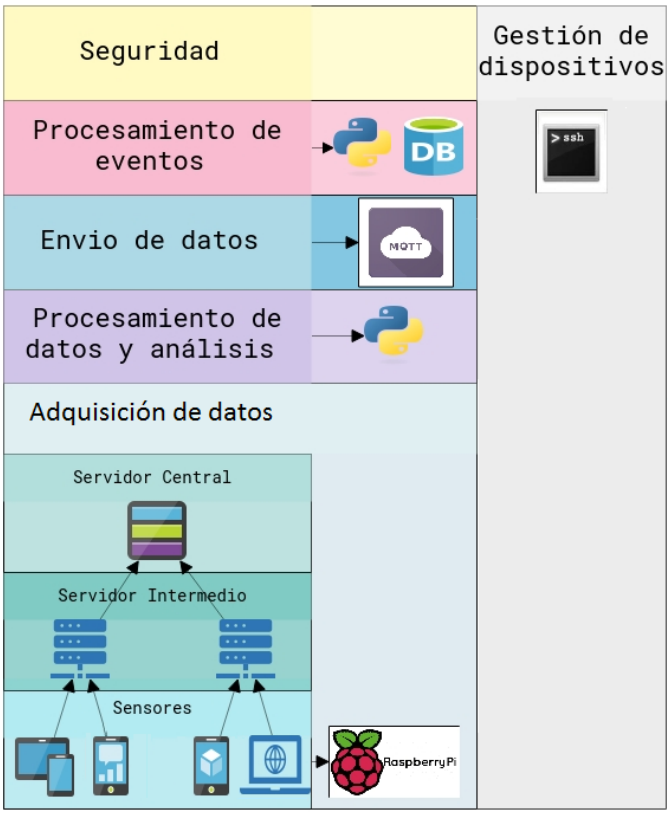


Figura 3.8 Diagrama de la arquitectura de software.
Fuente: Elaboración propia.

Las capas de la arquitectura son:

- Adquisición de datos
- Procesamiento de datos y análisis
- Envío de datos
- Procesamiento de eventos
- Seguridad
- Gestión de dispositivos

De las cuales ésta última es la capa intermedia.

3.1 DESCRIPCIÓN DE LAS CAPAS DE LA ARQUITECTURA

3.1.1 ADQUISICIÓN DE DATOS

En esta capa se toman las decisiones de diseño referente a los sensores y dispositivos, entre ellos la topología de la red y la selección de sensores.
Para nuestra arquitectura IoT se propone tener una topología similar a la de la red de sismógrafos presentada en el capítulo 2.3.2, la cual propone una arquitectura de tres niveles, un nivel para los acelerómetros, otro para los servidores intermedios, y finalmente el nivel en el que está el servidor central. Se ha decidido tomar este modelo ya que mejora la escalabilidad, al asignar servidores intermedios no solo ayuda a la distribución de carga del servidor central, sino que también facilita la gestión de dispositivos, ya que cada servidor intermedio está a cargo de distintos dispositivos, los cuales pueden ser distribuidos estratégicamente según el criterio que se desee establecer.
Se ha decidido ocupar Raspberry Pi 3 como dispositivo principal para la arquitectura, los detalles de la elección del dispositivo se explicarán más adelante.
Para la selección de sensores se recomienda usar dispositivos digitales, ya que Raspberry Pi 3 tiene un buen soporte para este tipo de dispositivos, pero no para los dispositivos analógicos.

3.1.2 PROCESAMIENTO DE DATOS Y ANÁLISIS

En esta capa se toman las decisiones referentes al análisis de datos. Se decidió ocupar Raspbian como sistema operativo, ya que está diseñado para operar con los dispositivos Raspberry, por lo que sus capacidades se aprovecharán al máximo usando este sistema operativo.
Para programar se ha decidido usar Python 3, Al ser un lenguaje que permite scripting facilita mucho la programación, además tiene bibliotecas específicas para el manejo de sensores y envío de datos usando distintos protocolos.

3.1.3 ENVÍO DE DATOS

Aquí se toman las decisiones de protocolos de comunicación con el bróker. Se ha decidido usar MQTT para establecer la conexión de los dispositivos, la razón de esta elección es porque MQTT en general mostró mejores resultados en desempeño y tiempos de respuesta que los demás protocolos, esto es debido a que MQTT es un protocolo binario y no basado en texto, se escogió por sobre ZeroMQ porque este último funciona mejor en arquitecturas descentralizadas [4], contrario a nuestra arquitectura que es centralizada. También se eligió porque la comunidad en la red tiene mejor soporte para este protocolo que otros similares.

3.1.4 PROCESAMIENTO DE EVENTOS

En esta capa es donde los servidores intermedios y central de nuestra arquitectura toman las decisiones referentes a los datos recibidos por los sensores y su posterior respuesta o reacción. También se encarga de ingresar a la base de datos la información recibida de los sensores.
El objetivo principal del servidor central es que éste sea el que tome las decisiones o reacciones ante algún evento reportado por los servidores intermedios, mientras que los servidores intermedios se encargan del análisis de los datos reportados por los sensores y de su almacenamiento en bases de datos. En caso de tener un proyecto acotado o un procesamiento de eventos poco complejo es posible omitir el servidor central en la arquitectura y delegar estas tareas a los servidores intermedios.

3.1.5 SEGURIDAD

Esta es la capa en la que se toman las decisiones relacionadas con la seguridad general de la red. Para mantener la seguridad en la arquitectura se sugiere tomar las medidas de prevención mencionadas en [12].
La implementación de autenticación evitará principalmente problemas referentes a la suplantación de identidad, la encriptación garantizará una mejor privacidad de los datos, y evitará la intercepción de éstos. La evaluación de riesgos permitirá decidir una buena estrategia para proteger la red de distintas amenazas, como lo son malware, ataques de denegación de servicios, etc. Los mecanismos de control nos permitirán mantener la red de forma segura en caso de que alguno de los nodos se vea comprometido. [12]
Una alternativa para dar seguridad, transparencia y privacidad a los datos es implementar las medidas de seguridad que provee Blockchain, sin embargo hay que tener en cuenta que esta tecnología funciona mejor en arquitecturas no centralizadas [32], por lo que es recomendable modificar la topología de la red de forma que de abasto a esta tecnología en caso de que se quiera implementar. También se debe tener en cuenta el costo computacional que implica el uso de esta tecnología, ya que unas de las mayores desventajas que tiene es su ineficiencia, las cadenas de bloques, sobre todo las que usan "*Proof of work*" son muy ineficientes, ya que su minería es muy competitiva, solo se aprovecha el trabajo realizado por un dispositivo mientras que el de los demás se desperdicia. El almacenamiento también es un problema ya que los *ledger* pueden crecer mucho a lo largo del tiempo, como ejemplo, el Blockchain de Bitcoin requiere alrededor de 200 GB. [33]
Debido a esto no se recomienda su implementación a no ser que la red a implementar sea pequeña y que se cuente con el soporte de hardware necesario.

3.1.6 GESTIÓN DE DISPOSITIVOS

Ésta es una capa intermedia porque implica la participación tanto del bróker como de los dispositivos, esta sección se encarga de la gestión remota de los dispositivos, se realiza por medio de una conexión SSH desde el bróker hasta los dispositivos, de esta forma permite habilitar o deshabilitar características del dispositivo, gestionar el software, monitorear la disponibilidad de éste, limpiar o bloquear dispositivos en caso de que se encuentren comprometidos.

3.2 ELECCIÓN DEL DISPOSITIVO

Según lo investigado en distintos sondeos, y estudios comparativos de dispositivos IoT [24, 25, 26, 27] se mencionaron múltiples características relevantes de los dispositivos, dependiendo de las necesidades de un proyecto, estos aspectos podrían ser o no relevantes para escoger un dispositivo.

Se revisaron los criterios de selección más frecuentes para una arquitectura genérica, se concluyó que los factores más relevantes para evaluar los dispositivos en una red de sensores son el voltaje, cantidad de memoria RAM, velocidad del procesador, costo, soporte de lenguajes de programación, soporte de sistemas operativos e interfaces de comunicación.[24 26 27]

Dispositivo	Voltaje	Memoria RAM	*Clock Speed*	Costo	IDE	Sistemas Operativos	Interfaces de comunicación
SparkFun Blynk Board	5V	128 KB	26 MHz	30 USD	Arduino, Blynk	No	I2C, SPI, UART
Arduino Yun	5V	64 MB	16 MHz	58 USD	Arduino	OpenWrt-Yun	I2C, UART
Raspberry Pi 3	5V	1 GB	1.2 GHz	35 USD	C#, Python, Java, Scratch, etc.	Raspbian, Windows 10 IoT Core, OpenELEC, OSMC, Pidora, Arch Linux, RISC OS, Ubuntu	SPI, 2x I2C, PCM/I2C, UART
cloudBit	5V	64 MB	454 MHz	60 USD	Cloud API, Arduino	Customized Arch Linux ARM distribution	UART
Photon	5V	128 KB	120 MHz	19 USD	Particle Build, Particle Dev	FreeRTOS	2x SPI, I2S, I2C, CAN, UART
BeagleBone Black	5V	512 MB	1 GHz	49 USD	C++, Perl, Python, Cloud9 IDE	Debian, Android, Ubuntu	4x UART, 2x SPI, 2x I2C, CAN, UART
Pinoccio	5V	32 KB	16 MHz	109 USD	Arduino, Pinoccio HQ	No	I2C, SPI, 2x UART

UDOO	12 V	1 GB	1 GHz	135 USD	Arduino	UDOObuntu, Android, XMBC, Yocto, Arch Linux, OMV	SPI, I2C, UART, CAN BUS
Samsung Artik 10	3.4-5 V	2 GB	1.3 GHz	100 USD	Arduino IDE, Samsung SDK, C/C++, Java, Groovy	Yocto 1.6 (Fedora)	SPI, 6x I2C, I2S, 3x UART

Tabla 3.6: Tabla comparativa de dispositivos.
Fuente: Elaboración Propia.

Según un estudio realizado en dispositivos IoT [26], podemos categorizar los dispositivos en 3 clases distintas:

- Low-end IoT: Dispositivos muy restringidos en términos de recursos, no pueden correr sistemas operativos tradicionales, y por lo general son de arquitecturas de 8 o 16 bits. [26]
- Middle-end IoT: Son dispositivos menos restringidos en términos de recursos, proveen más características con su poder de procesamiento en contraste con los dispositivos Low-end, como reconocimiento de imágenes. Adicionalmente pueden tener más de una tecnología de comunicación. [26]
- High-end IoT: Son por lo general computadoras de placa, tienen muy buenos recursos como poderosas unidades de procesamiento, gran cantidad de RAM y gran volumen de almacenamiento, algunas incluyen unidades de procesamiento gráfico. Pueden dar soporte a sistemas operativos clásicos como Linux o Windows, estas máquinas pueden incluso ejecutar algoritmos pesados de machine learning, estos dispositivos también suelen contar con características muy útiles como interfaces Fast/Ethernet chips, Wifi/BT, interfaz HDMI, varios puertos USB. [26]

Para que un dispositivo pueda categorizarse como clase High-end, debe poseer más de 50kB de RAM, más de 250 KB de memoria Flash, debe ser capaz de soportar un RTOS (sistemas operativos en tiempo real) y tener la capacidad de usar múltiples protocolos de comunicación. [26]
Dadas las características de nuestra arquitectura, lo más recomendable es que el dispositivo a elegir pueda realizar diversas funcionalidades, ya sean de seguridad, pre procesamientos de datos, etc. Por lo que lo más adecuado es elegir un dispositivo High-end, por esta razón descartamos el dispositivo Pinoccio el cual tiene menos de 50 KB de memoria. También descartamos el dispositivo SparkFun Blynk Board por no poseer sistema operativo.

Entre todos los dispositivos restantes se determinó que el más óptimo para nuestra arquitectura es Raspberry Pi 3. Esto es debido a que sus características son por lo general superiores a las de los demás dispositivos, siendo éste el dispositivo con mayor cantidad de entornos de

programación y sistemas operativos compatibles, también es el dispositivo más económico sin contar los dispositivos descartados. Además es el segundo mejor dispositivo tanto en velocidad de procesador como en cantidad de RAM. [7]

Además de lo anterior mencionado, se ha demostrado que Raspberry Pi tiene un gran potencial en muchos proyectos IoT, entre ellos se pueden mencionar proyectos como una implementación de sistema biométrico, [28] sistemas de monitoreo de pacientes [26] y sistemas de automatización de hogares. [26] También ha sido usado en conjunto con otras placas para compensar ciertas falencias que tiene Raspberry, entre ellos la fabricación de una plataforma para un vehículo autónomo de bajo costo [25] y un sistema de alarma contra incendios. [25]

A pesar de haber dispositivos con mejor desempeño computacional, se escoge por encima de éstos por ser mucho más económico que otras plataformas como UDOO o Samsung Artik 10. [27] Aunque exista esta diferencia, Raspberry Pi puede proveer un gran desempeño cuando se usa un sistema operativo Linux, es también fácil de programar a las necesidades de un proyecto y puede proveer una plataforma para conectar muchos dispositivos periféricos. [27]

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN

En los capítulos anteriores, hemos investigado cómo hacer esta arquitectura lo más eficiente posible, en este capítulo se pondrá en práctica lo investigado anteriormente para hacer una arquitectura IoT para una red de sensores y se estudiará la viabilidad de ésta midiendo los recursos computacionales que ocupa. En el desarrollo de la arquitectura se omitió la implementación de la capa de seguridad, sin embargo en un ambiente de producción es una capa muy importante que no puede faltar.

Los códigos de los programas usados junto con sus mediciones se encuentran en el repositorio https://gitlab.labcomp.cl/jcampos/memoria.

4.1 ESPECIFICACIONES DE LA ARQUITECTURA

El dispositivo utilizado para probar nuestra arquitectura es una Raspberry Pi 2 model B, se escogió este dispositivo por su disponibilidad, ya que es el dispositivo más versátil con el que contaba el Departamento de Informática de la universidad.
Se escogió el sistema operativo Raspbian ya que éste se desarrolló específicamente para el dispositivo, además de esto es muy fácil de instalar y utilizar.
Para la creación de los programas se ocupó Python 3, ya que es el lenguaje de programación más usado en Raspberry, al ser de alto nivel, facilita mucho la programación en el dispositivo, además de esto, cuenta con muchas bibliotecas específicas para Raspberry y el manejo de sensores, cabe destacar además que los programas envían datos por el protocolo MQTT, utilizando la biblioteca paho-mqtt.

4.2 ESPECIFICACIONES DE LOS SENSORES Y ALGORITMOS ASOCIADOS

Para la realización de las mediciones se ocuparon tres tipos de sensores distintos, el sensor ultrasónico HC-SR04, el acelerómetro GY-521 y la cámara Raspberry Pi Camera Board.

Sensor ultrasónico HC-SR04:
Este es un sensor utilizado para detectar distancias, cuenta con dos transmisores que sirven como transmisor y receptor. Como salida, este sensor cuenta con cuatro pines, los cuales son Vcc, Trig, Echo y Gnd como son mostrados en la siguiente imagen. [13]

Figura 4.9: HC-SR04 ultrasonic sensor.
Fuente: Kalman Filter Algorithm Design for HC-SR04 Ultrasonic Sensor Data Acquisition System.[13]

Este sensor funciona emitiendo una señal ultrasónica para luego captar la onda reflejada, la onda ultrasónica es emitida por el transmisor a través del pin Trigger, cuya señal es dada por el microcontrolador, y luego la onda reflejada es captada por el receptor y enviada al microcontrolador por el pin Echo. [13]
El retardo que hay entre la transmisión y la reflexión de la onda es usada como referencia para calcular la distancia entre el sensor y el objeto, para calcular la distancia se usa la fórmula $s = kT$, donde s es la distancia, T es el periodo de la onda en microsegundos y k es una constante, su valor depende de la unidad de medida de s (58 para conversión a cm, 148 para pulgadas).

El sensor tiene las siguientes especificaciones:

- Trabaja con un voltaje de 5 VDC y 15 mA de corriente
- Opera a una frecuencia de 40 kHz
- Funciona a una distancia máxima de 400 cm y una mínima de 2 cm

El programa que envía la información del sensor ultrasónico hace uso de la función del filtro de Kalman, este filtro es usado cuando la información obtenida de un ambiente es imprecisa. El algoritmo se usa en sistemas dinámicos lineales, puede o no ser estacionario, funciona en ambos casos. Cada actualización de estimación es computada desde la estimación anterior y lo nuevos datos recibidos, por lo que solo requiere que el estado anterior sea almacenado, lo que hace que Kalman Filter sea más eficiente que otros filtros, ya que éstos requieren analizar todos los datos anteriores en cada paso para filtrar. [13]

Kalman Filter está diseñado para trabajar con sistemas que funcionen acorde a las siguientes ecuaciones: [13]

$$X_k = FX_{k-1} + Bu_{k-1} + w_{k-1}$$
$$Z_k = HX_k + v_k$$

Donde:
$X_k \in R^n$, $u_k \in R^t$, $Z_k \in R^m$ Son respectivamente el vector estado, feedback y medición.
F $\in R^{nxn}$, B $\in R^{nxl}$, H $\in R^{mxn}$ Son la matriz constante de estado, feedback y medición.
w_ky Z_k es un vector de ruido blanco y de proceso de medición de ruido

El algoritmo consta de dos fases llamadas fase de predicción y fase de corrección, en la fase de predicción el algoritmo ocupa las siguientes ecuaciones: [13]

1. Proyección del estado hacia adelante: $\hat{X}_k$
2. Proyección de la covarianza del error hacia adelante: P_k
   - Donde $\hat{X}_{k-1}$ es un estado a priori.

Mientras que en la fase de corrección el algoritmo ocupa las siguientes ecuaciones:

3. Cómputo de la ganancia de Kalman: $K_k = P_k$
4. Actualización del estado con la medida (Z_k): $\hat{x}_k = \hat{x}_k$
5. Actualización de la covarianza del error: $P_k = (I - K_k H)P_k$
6. Devuelve $\hat{x}_k$y P_k

- $K_k$es un ponderador de Kalman, el cual sirve para minimizar la matriz de covarianza de error a posteriori P_ky R es la matriz de covarianza de la medición de ruido.
- F: Matriz nxn que relaciona el estado en el instante $t-1$ con el estado en el instante t, en ausencia de señales de control.
- B: Matriz nxl que relaciona las señales de control, opcionales, con el estado actual.
- Q: Matriz nxn que representa a la covarianza del ruido del proceso.
- H: Matriz mxn que relaciona el estado actual con las observaciones del entorno.
- R: Matriz nxn que representa a la covarianza del ruido de las observaciones.
- K_k: Matriz nxm que representa a la ganancia de Kalman. La ganancia de Kalman indica la confianza en las características observadas, empleando la incertidumbre de las mismas junto con una medida de la calidad de los datos proporcionados por el sensor.

Acelerómetro GY-521:

El sensor completo contiene un acelerómetro y un giroscopio en su chip, es un sensor muy preciso, y tiene un convertidor analógico a digital de 16 bits por cada canal, por lo que captura el eje X, Y y Z al mismo tiempo. El sensor usa la interfaz I2C-bus. [22]

Algunas de sus características son:

- Usa el chip: MPU-6050
- Usa un voltaje de 3 a 5 Volts
- Usa el standard IIC communications protocol

- El giroscopio tiene una sensibilidad de hasta 131 LSBs/dps (Least Significant Bit/degree per second) y una escala que va desde el rango de ±250, ±500, ±1000 y ±2000 dps
- El giroscopio tiene una rango de escala que va de ±2g, ±4g, ±8g y ±16g

Para manejar los datos de este acelerómetro se hizo uso de filtro de mediana y un filtro pasa bajos, el primero es un filtro digital no lineal usado para remover el ruido en las imágenes o señales, la idea principal de este filtro es recorrer una señal entrada por entrada reemplazando cada una por la media de las entradas vecinas, el patrón de vecindad es llamado "ventana", la cual se mueve entrada por entrada, para los valores en los bordes, se repiten los primeros valores de la señal para que haya suficientes valores a evaluar en la ventana.
En cuanto al filtro pasa bajos es un tipo de filtro que separa una señal de cualquier tipo de ruido o interferencia que haya en el ambiente, este filtro usa un convolucionador de tiempo discreto y amplitud discreta. Según la transformación de Fourier, la convolución lineal de dos secuencias es equivalente a la multiplicación de dos secuencias espectrales correspondientes en el dominio de la frecuencia. La multiplicación del espectro de la señal por la respuesta al impulso en el dominio de la frecuencia del filtro es el método básico para implementar un filtro digital. [14]

Adicionalmente, se utiliza otro algoritmo de forma paralela para realizar agregación de datos, de forma de reducir la cantidad de datos generado por el sensor, estos algoritmos son ENMO (Norma euclidiana menos uno) y MAD (Desviación de amplitud media). [31]
El primer algoritmo consiste en sacar la norma euclidiana de los datos generados y restarle 1 unidad gravitacional (9.8 m/s²), de esta forma se reduce la cantidad de datos enviada, ya que el sensor por defecto entrega la aceleración en sus 3 ejes, después de ejecutar el programa, este solo entrega una aceleración neta.
El segundo algoritmo se calcula con la siguiente fórmula:

$$MAD = \frac{1}{n}\sum_{i=1}^{n} r_i - \acute{r} \vee$$

$$r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$$

$$\acute{r} = mediadelvectormagnitud$$

De esta forma, durante cada periodo de tiempo, el algoritmo entrega un único valor correspondiente al entregado por la fórmula. [31]

Raspberry Pi Camera Board:
Es una cámara oficial de la fundación de Raspberry Pi, usa el sensor OmniVision OV5647 con una resolución de 2592 × 1944 pixels. [23]

Para este sensor se decidió ocupar PCA (Principal Component Analysis) el cual es un procedimiento que usa una transformación ortogonal para convertir un conjunto de observaciones de posible variables correlacionadas en un conjunto de valores lineales variables no correlacionados llamados componentes principales. Para efectos prácticos, el programa obtiene los contornos que puede reconocer de la imagen, a partir de cada contorno se hace un análisis en el cual se extraen los valores propios, vectores propios y ángulo de inclinación.
El conjunto de datos obtenidos de este análisis es lo que envía nuestro programa al bróker.

4.3 ANÁLISIS DE LOS RESULTADOS

Primero analizaremos el desempeño del sensor ultrasónico.
Lo primero que podemos ver es que el porcentaje de CPU usado por el proceso oscila entre el 50% y 32.5%, es un comportamiento esperado en este tipo de programas, ya que el programa es enviado a dormir constantemente configurando así la frecuencia del sensor (100 Hz), posterior a la obtención de datos se procede a los cálculos del filtro de Kalman para dar mayor precisión a los datos, hay que destacar también que éste es el único proceso ejecutándose excluyendo los procesos propios del sistema operativo y el proceso que se usó para medir el desempeño.

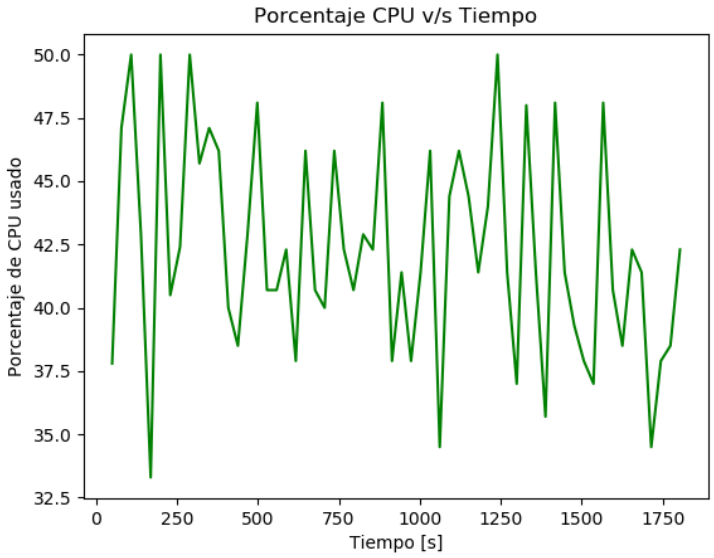


Figura 4.10: Sensor ultrasónico: Gráfico porcentaje de CPU v/s Tiempo.
Fuente: Elaboración propia.

También se puede apreciar que el programa constantemente usa 30452 Kb de memoria dinámica, este es un comportamiento esperado considerando que el proceso siempre ocupa la misma cantidad de memoria dinámica, esto es porque no hay creación ni destrucción constante de arreglos, listas o estructuras de datos que requieran uso de memoria dinámica, solo se usa

una determinada cantidad de variables las cuales después del primer ciclo de ejecución se mantienen las mismas variables pero con distintos valores.

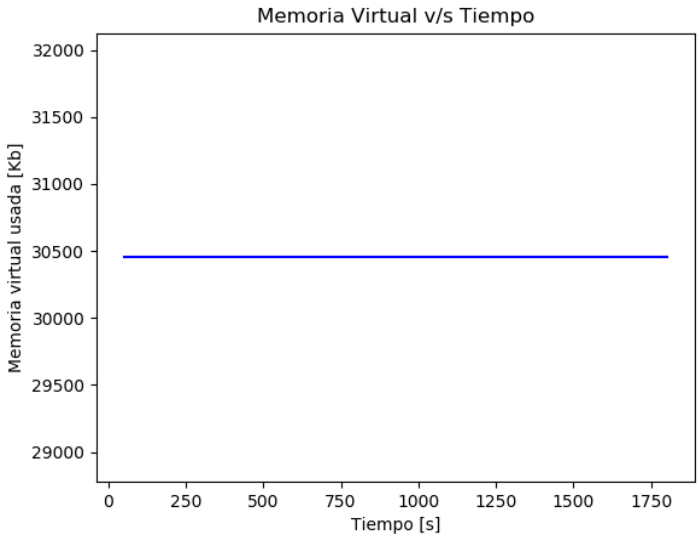


Figura 4.11: Sensor ultrasónico: Gráfico de Memoria Virtual v/s Tiempo.
Fuente: Elaboración propia.

En cuanto al flujo de datos, en el gráfico se puede apreciar la cantidad de KB por minuto que son enviados, se puede apreciar que por lo general va a una tasa de entre 200 KB/min y 215 KB/min, lo que también es un comportamiento esperado, ya que se están enviando datos cada 0.01 segundos, que es la frecuencia del sensor.

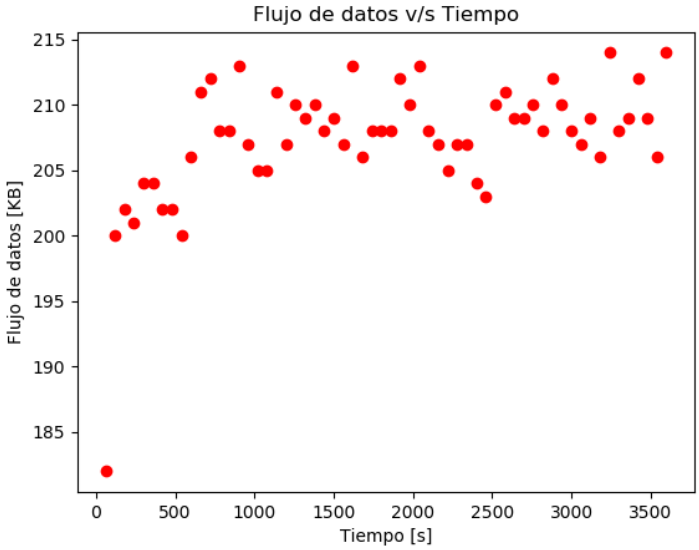


Figura 4.12: Sensor ultrasónico: Gráfico Flujo de Datos v/s Tiempo.
Fuente: Elaboración propia.

Ahora analizaremos los datos del acelerómetro, este programa tiene similitudes con el que fue usado para el sensor ultrasónico, ya que en ambos programas no se requiere de un gran uso de memoria dinámica ni tampoco tienen una gran cantidad de instrucciones que ejecutar por cada ciclo, pero aquí hay que hacer una distinción importante, ya que los filtros usados en este programa son el filtro de mediana y el filtro pasa bajos, ambos filtros requieren que el programa capte una gran cantidad de datos antes de poder realizar el filtrado de datos.

Teniendo esto en cuenta pasaremos a analizar los gráficos correspondientes, notamos que en el gráfico de uso de CPU el porcentaje tiene una mayor variación de valores que en el caso anterior, tomado valores de entre el 19% y el 67%, esto es debido a lo anterior mencionado, hasta no alcanzar los 1000 datos, el programa está tomando muestras o descansando, luego de alcanzar la cantidad de datos pone en marcha el algoritmo de filtrado.

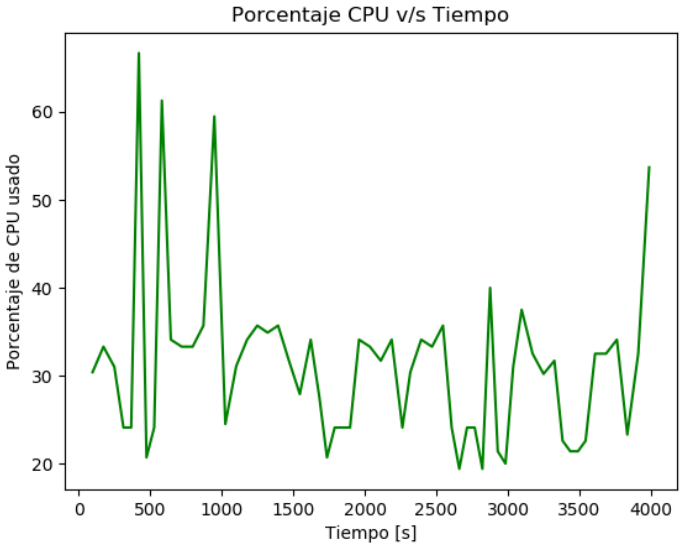


Figura 4.13: Acelerómetro: Gráfico porcentaje de CPU v/s Tiempo.
Fuente: Elaboración propia.

Notamos que al igual que en el primer programa, la memoria usada aquí permanece constante por la misma justificación anterior, no hay creación ni destrucción de arreglos, colas u otras estructuras de datos, la memoria virtual que se usa es pedida al inicio de la ejecución del programa y de aquí en adelante no necesita ocupar ni liberar memoria.

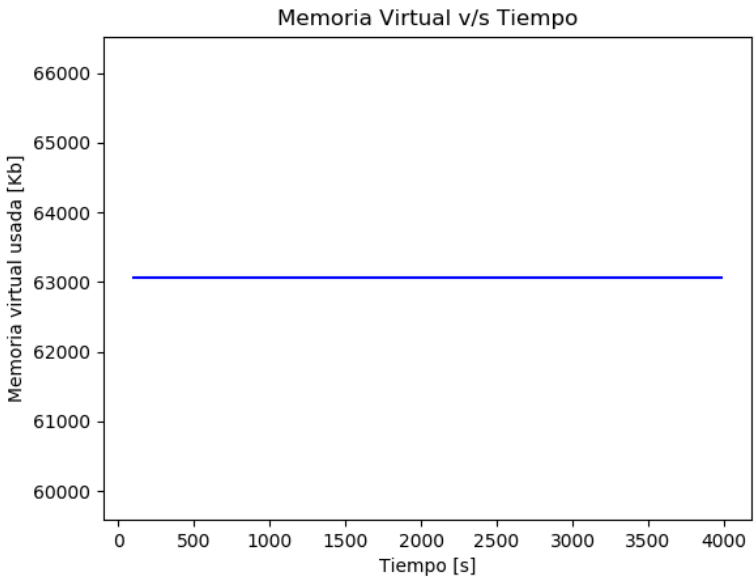


Figura 4.14: Acelerómetro: Gráfico de Memoria Virtual v/s Tiempo.
Fuente: Elaboración propia.

Por último podemos ver que el flujo de datos oscila entre los 235 KB y los 285 KB, en este caso los paquetes de datos son más pesados que en el programa del sensor ultrasónico, pero también los envía de forma menos frecuente, ya que este programa envía 1000 datos cada 10 segundos, y el programa del sensor ultrasónico envía 1 dato cada 0.01 segundo, por lo que no es de extrañar que los datos aquí se comporten de esta forma.

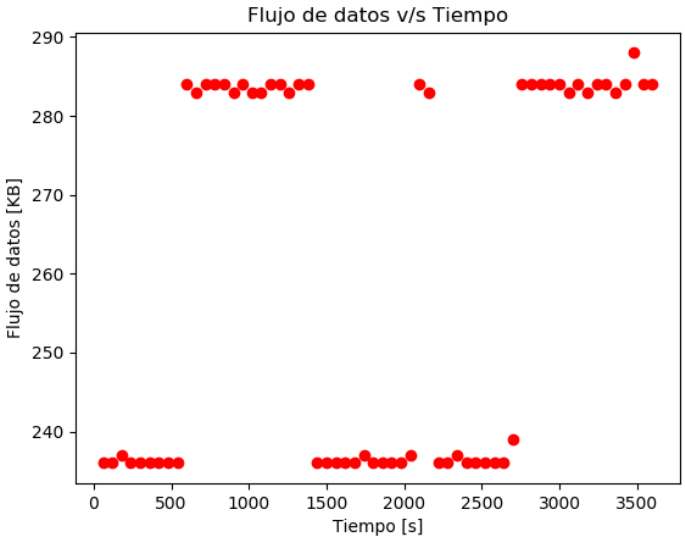


Figura 4.15: Acelerómetro: Gráfico Flujo de Datos v/s Tiempo.
Fuente: Elaboración propia.

Finalmente analizaremos los datos entregados por la cámara, como es de esperarse este sensor ocupa mucho más memoria que el resto de los sensores, esto es debido a que almacenar una imagen requiere de mucha más memoria que almacenar números y por lo general, al realizar preprocesamiento de imágenes requiere de un instrucciones que conllevan a un número muy superior de instrucciones de máquina. Como podemos ver en los gráficos siguientes el porcentaje de uso de CPU varía principalmente entre en 75% y el 50%, esto es debido a que el programa está siempre trabajando, a diferencia del resto de programas que cada cierto instante de tiempo se manda a dormir para determinar la frecuencia del sensor. En cada ciclo, el programa realiza un análisis para obtener contornos en la imagen, y a partir de los contornos encontrados realiza un análisis PCA a cada objeto encontrado, de esta forma obtenemos los vectores propios, valores propios y otros valores asociados al análisis.

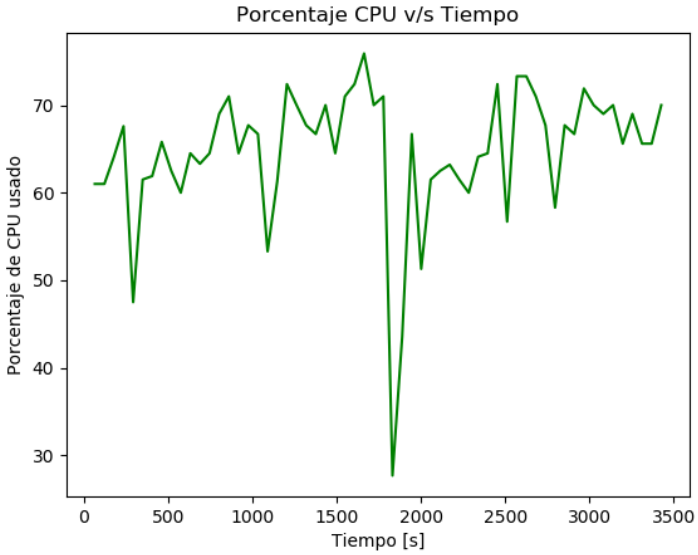


Figura 4.16: Cámara: Gráfico Porcentaje de CPU v/s Tiempo.
Fuente: Elaboración propia.

Notamos que al igual que en los demás programas el uso de memoria también es constante y toma un valor de 153828 [Kb], esto se debe a que trabajar con imágenes, por lo general, requiere de más memoria. Podemos decir que el programa puede correr de manera estable en el dispositivo, sin embargo, el programa realiza dicho análisis a una frecuencia de alrededor de 10 fotogramas por segundo, en caso de requerir una mayor frecuencia en el dispositivo, o de requerir un preprocesamiento más elaborado a las imágenes, las características del dispositivo podrían no ser suficientes para establecer una red de sensores, por lo que requeriría de un dispositivo con mayor poder de procesamiento y con mayor cantidad de memoria.

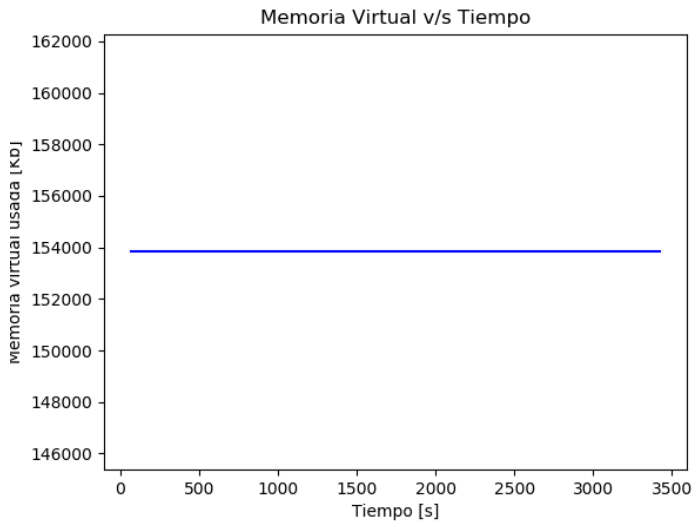


Figura 4.17: Cámara: Gráfico de Memoria Virtual v/s Tiempo.
Fuente: Elaboración propia.

Por último notamos que el envío de datos de la cámara es mínimo, esto es debido a que el programa envía poca información al broker, ya que en esencia el emisor hace todo el trabajo de reconocimiento de imágenes y luego envía los datos, además de esto hay que considerar que la frecuencia en que la cámara toma una imagen no es muy alta, esto es debido a que el preprocesamiento de este programa es mucho más largo que el de los demás programas, por lo que toma más tiempo en realizarlo.

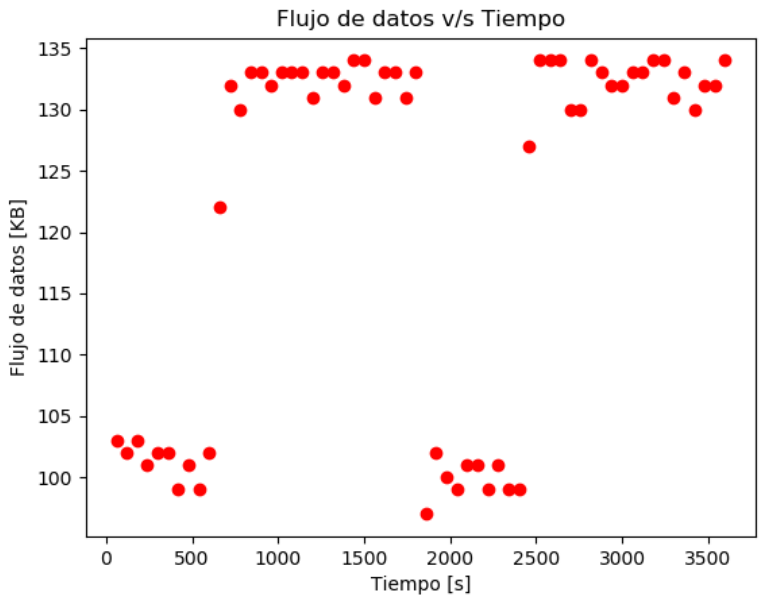


Figura 4.18: Cámara: Gráfico Flujo de Datos v/s Tiempo.
Fuente: Elaboración propia.

Se realizaron algunos análisis adicionales a la arquitectura, más específicamente a los algoritmos del acelerómetro, estos consisten en contrastar la diferencia entre el volumen de datos generado, y en mostrar la diferencia entre los datos antes y después de usar los filtros.

El filtro de mediana ayuda a remover las púas generadas por el ruido, el filtro pasa bajos ayuda a reducir el ruido provocado por altas frecuencias de baja amplitud. El resultado se puede ver en los gráficos siguientes:

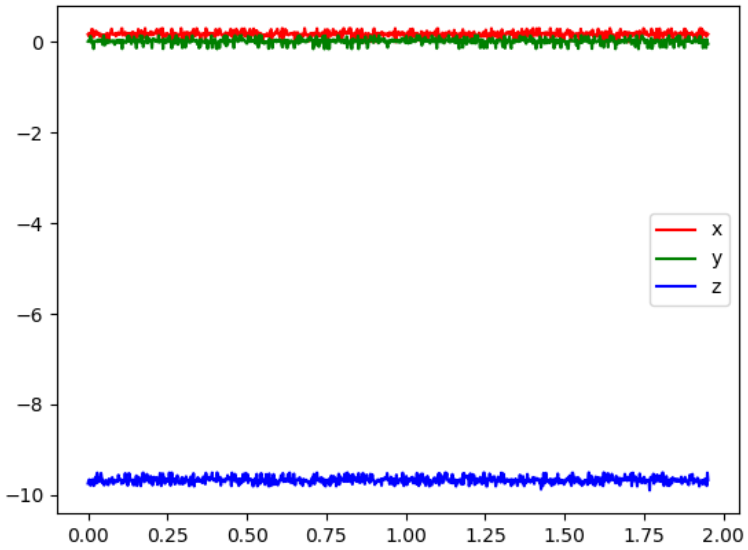


Figura 4.19: Datos del acelerómetro sin filtrar.
Fuente: Elaboración propia

Como se puede apreciar en el gráfico 4.11, los datos tienen una dispersión considerable debido al ruido captado por el sensor.

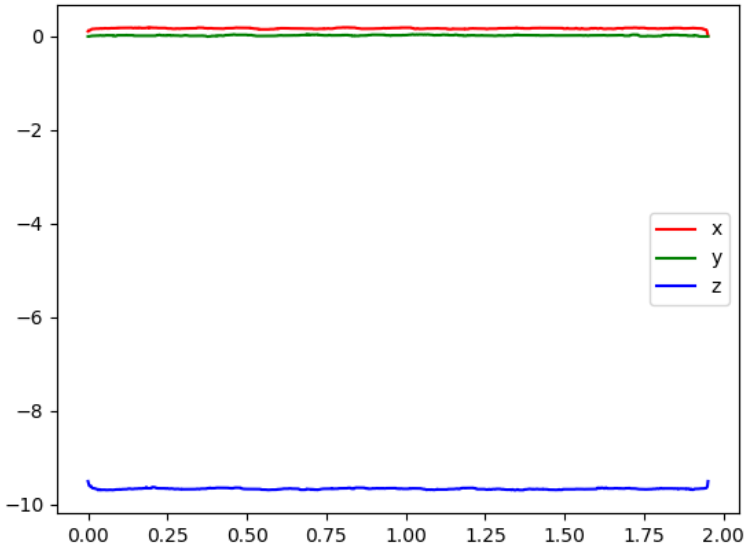


Figura 4.20: Datos del acelerómetro usando filtro de mediana.
Fuente: Elaboración propia

En el gráfico 4.12 se puede apreciar los datos ya no se encuentran tan dispersos como en el gráfico anterior, ahora la curva tiene un comportamiento más definido y constante, sin embargo aún queda ruido generado por altas frecuencias de baja amplitud.

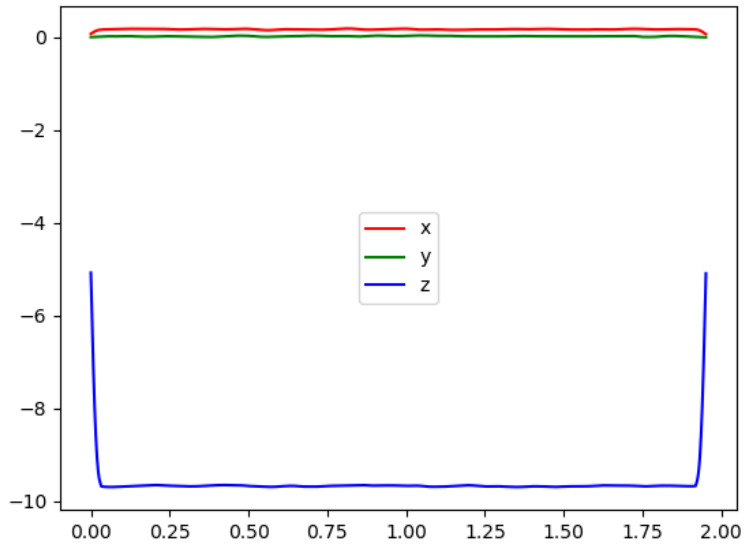


Figura 4.21: Datos del acelerómetro usando filtro pasa bajos y de mediana.
Fuente: Elaboración propia

En el gráfico 4.13 se puede ver que la línea se encuentra mejor definida, eliminando el ruido de mejor forma, los primeros y últimos datos del gráfico se encuentran alterados por las

características del filtro pasa bajos, sin embargo todo el resto de la data representa de forma más fiel el comportamiento del medio.

En la figura 4.14 se puede apreciar tres líneas, la roja corresponde al volumen de datos generado sin realizar procesamiento alguno, la línea verde corresponde al volumen generado al aplicar ENMO y la línea azul corresponde al volumen generado al aplicar MAD.

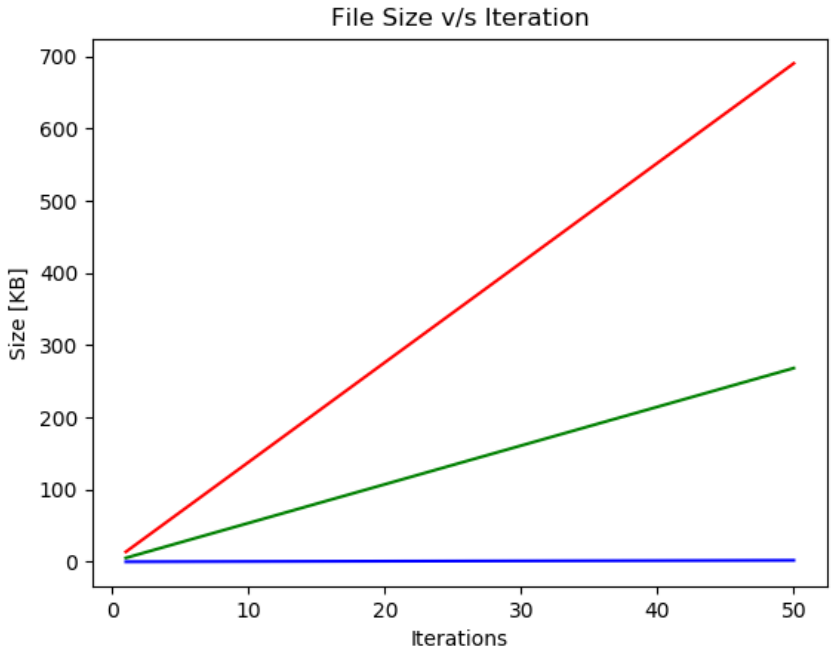


Figura 4.22: Gráfico de volumen de datos generado.
Fuente: Elaboración propia

Notamos que el volumen generado por la línea roja es de alrededor de 3 veces el generado por la línea azul, justo como se esperaba de la parte teórica. Notamos también que el volumen azul se encuentra muy por debajo de los demás algoritmos, esto es porque MAD entrega solamente un dato después de muchas iteraciones, revisando en la literatura [31] se indica que cada 5 segundos de periodo puede ser considerado como una cantidad adecuada de tiempo para reportar actividad. El sensor en todos los análisis realizados se mantuvo en 100 Hz tal y como se sugería en la literatura, [31] por lo que se estima que el volumen de datos generado sin ningún algoritmo es 500 veces más que el volumen generado por MAD.

CAPÍTULO 5: CONCLUSIONES

5.1 CONCLUSIONES GENERALES

En esta investigación hemos presentado una arquitectura genérica de software que fue probada con 3 tipos distintos de sensores, para todos los casos nuestra arquitectura ha realizado una implementación exitosa. Desde el punto de vista de Device Layer, Raspberry Pi ha mostrado un gran potencial para manejar sensores, ya que es capaz de realizar algoritmos muy complejos en un tiempo reducido, ajustándose a las necesidades de la arquitectura. Los protocolos de comunicación y la topología escogida también ayudan a cumplir las restricciones del sistema y a mejorar la escalabilidad de la red.

Gracias al soporte que tiene Raspberry Pi a distintos sistemas operativos se puede realizar una conexión remota segura al dispositivo, Raspbian permite configurar esta conexión de forma sencilla.

A pesar del gran potencial de Raspberry Pi, este dispositivo tiene falencias para trabajar con sensores analógicos, por lo que para esta arquitectura se recomienda trabajar con sensores digitales, en caso de requerir trabajar con sensores digitales se recomienda hacer un trabajo conjunto con algún otro dispositivo como Arduino, o reemplazar Raspberry por otro dispositivo con mejor soporte de hardware.

Tal y como se propuso, la creación de un middleware que permita la integración de datos no solo ayuda a la escalabilidad de la arquitectura, sino que además reduce el gasto energético del dispositivo. Por lo que es esencial en una red de sensores un middleware.

5.2 TRABAJO FUTURO

- Implementación de mecanismos de seguridad en IoT como lo son la encriptación, y firmas digitales.
- Estudio del impacto que las medidas de seguridad estudiadas en [12] tendrá en relación al desempeño de la red.
- Revisar placas o tarjetas nuevas sacadas al mercado, como Raspberry Pi 4.
- Utilizar containers para facilitar la implementación de la arquitectura en múltiples dispositivos.

REFERENCIAS BIBLIOGRÁFICAS

REFERENCIAS BIBLIOGRÁFICAS EN NORMA APA:

1. Datta, S. K., Bonnet, C., & Nikaein, N. (2014, March). An IoT gateway centric architecture to provide novel M2M services. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on* (pp. 514-519). IEEE.
2. Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, *29*(7), 1645-1660
3. Fremantle, P. (2014). A reference architecture for the internet of things. *WSO2 White paper*.
4. Happ, D., Karowski, N., Menzel, T., Handziski, V., & Wolisz, A. (2017). Meeting IoT platform requirements with open pub/sub solutions. Annals of Telecommunications, 72(1-2), 41-52.
5. Clavel, G., Marcelo, (2014), Protocolos de Comunicación Para Aplicaciones Móviles. Valparaíso, Chile: Universidad Técnica Federico Santa María.
6. Maksimović, M., Vujović, V., Davidović, N., Milošević, V., & Perišić, B. (2014). Raspberry Pi as Internet of things hardware: performances and constraints. design issues, 3, 8.
7. Singh, K. J., & Kapoor, D. S. (2017). Create Your Own Internet of Things: A survey of IoT platforms. IEEE Consumer Electronics Magazine, 6(2), 57-68.
8. Casco, S. (2017). Raspberry Pi, Arduino y Beaglebone Black Comparación y Aplicaciones. Universidad Católica Nuestra Señora de la Asunción, Facultad de Ciencias y Tecnología.
9. Hernandez, M. D. F., & de los Reyes Quiroz, F. (2017). MAPEO DE LA SEÑAL WIFI MEDIANTE UN DISPOSITIVO IMPLEMENTADO EN ARDUINO. JÓVENES EN LA CIENCIA, 3(1), 494-497.
10. Córdoba, D. M. A., & Buitrago, F. A. S. (2013). Estado del arte de las redes de sensores inalámbricos. *Tecnología Investigación y Academia*, *1*(2).
11. ZAMBRANO VIZUETE, A. N. A., Pérez Llopis, I., Palau Salvador, C. E., & Esteve Domingo, M. (2015). Sistema Distribuido de Detección de Sismos Usando una Red de Sensores Inalámbrica para Alerta Temprana. *Revista Iberoamericana de Automática e Informática Industrial (RIAI)*, *12*(3), 260-269.
12. Farooq, M. U., Waseem, M., Khairi, A., & Mazhar, S. (2015). A critical analysis on the security concerns of internet of things (IoT). *International Journal of Computer Applications*, *111*(7).
13. Al Tahtawi, A. R. Kalman Filter Algorithm Design for HC-SR04 Ultrasonic Sensor Data Acquisition System. IJITEE (International Journal of Information Technology and Electrical Engineering), 2(1), 15-19.

14. Ahmed, S., Bashir, M., & Suri, A. Low Pass FIR Filter Design and Analysis Using Hamming, Blackman and Kaiser Windows.
15. Cascón, J. (Diciembre de 2019). frikipandi. Obtenido de https://www.frikipandi.com/etiquetas/smartphone/.
16. N. T. Foundation, (Diciembre de 2019). *NTP*. Obtenido de http://ntp.org/.
17. Asociación, Programo Ergo Sum (Diciembre de 2019). Programoergosum. Obtenido de https://www.programoergosum.com/cursos-online/raspberry-pi/238-control-de-gpio-con-python-en-raspberry-pi/que-es-gpio.
18. ESG Solutions, (Diciembre de 2019). Esgsolutions. Obtenido de https://www.esgsolutions.com/es/recursos-tecnicos/base-de-conocimientos-microsismicos/deteccion-de-eventos-y-desencadenantes.
19. Circuit Basics, (Diciembre de 2019). Circuitbasics. Obtenido de http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/.
20. Circuit Basics, (Diciembre de 2019). Circuitbasics. Obtenido de http://www.circuitbasics.com/basics-uart-communication/.
21. Panama Hitek, (Diciembre de 2019). Panamahitek. Obtenido de http://panamahitek.com/como-funciona-el-protocolo-spi/.
22. Scribd, (Noviembre de 2019). Scribd. Obtenido de https://www.scribd.com/document/365515770/MPU-6050-GY-521-Datasheet/.
23. Raspberry Pi Foundation (Noviembre de 2019). raspberrypi. Obtenido de https://www.raspberrypi.org/documentation/hardware/camera/.
24. Patnaik Patnaikuni, D. R. (2017). A Comparative Study of Arduino, Raspberry Pi and ESP8266 as IoT Development Board. International Journal of Advanced Research in Computer Science, 8(5).
25. Noor, N. Q. M., Jamaludin, A. A., Azizan, A., Abas, H., Kamardin, K., & Yakub, M. F. M. (2018). Arduino vs Raspberry Pi vs Micro Bit: Platforms for Fast IoT Systems Prototyping. Open International Journal of Informatics (OIJI), 96-107.
26. Ojo, M. O., Giordano, S., Procissi, G., & Seitanidis, I. N. (2018). A Review of Low-End, Middle-End, and High-End Iot Devices. IEEE Access, 6, 70528-70554.
27. Patil, P. N. (2016). A comparative analysis of Raspberry pi Hardware with Ar-duino Phidgets Beaglebone black and Udoo. International Research Journal of Engineering and Technology (IRJET), 1595-1600.
28. Shah, D., & Haradi, V. (2016). IoT based biometrics implementation on Raspberry Pi. Procedia Computer Science, 79, 328-336.
29. Krčo, S., Pokrić, B., & Carrez, F. (2014, March). Designing IoT architecture (s): A European perspective. In 2014 IEEE World Forum on Internet of Things (WF-IoT) (pp. 79-84). IEEE.

30. Li, W., & Kara, S. (2017). Methodology for monitoring manufacturing environment by using wireless sensor networks (WSN) and the internet of things (IoT). Procedia CIRP, 61(Supplement C), 323-328.
31. Bakrania, K., Yates, T., Rowlands, A. V., Esliger, D. W., Bunnewell, S., Sanders, J., ... & Edwardson, C. L. (2016). Intensity thresholds on raw acceleration data: Euclidean norm minus one (ENMO) and mean amplitude deviation (MAD) approaches. PloS one, 11(10).
32. Reyes, D., Diego, (2018), APLICACIÓN DE BLOCKCHAIN PARA LA SEGURIDAD DE LOS DATOS DEL INTERNET DE LAS COSAS. Valparaíso, Chile: Universidad Técnica Federico Santa María.
33. Zhao, C. (n.d.). Blockchain Advantages and Disadvantages. Retrieved March 13, 2020, from https://www.binance.vision/blockchain/positives-and-negatives-of-blockchain

ANEXOS

Anexo 1: Gráficos comparativos entre protocolos XMPP Y MQTT

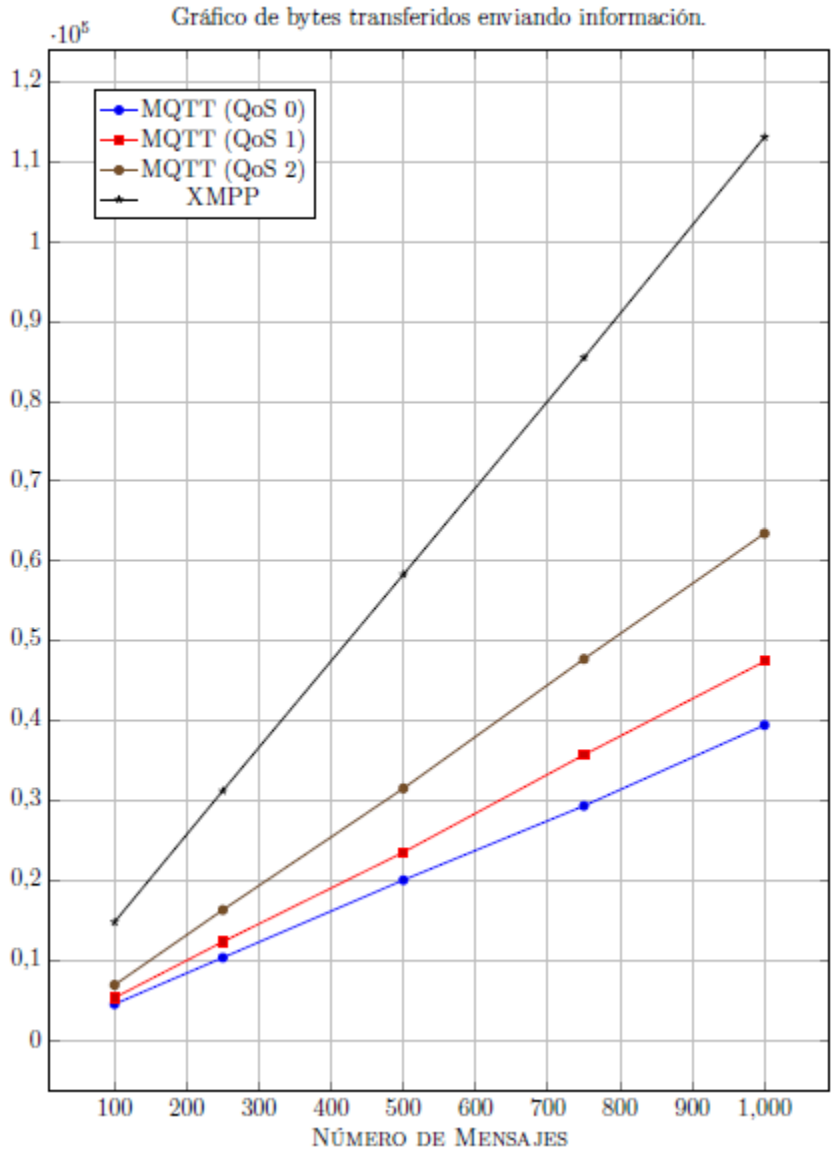


Figura 6.1: Gráfico de bytes transferidos enviando información.
Fuente: Protocolos de Comunicación Para Aplicaciones Móviles.[5]

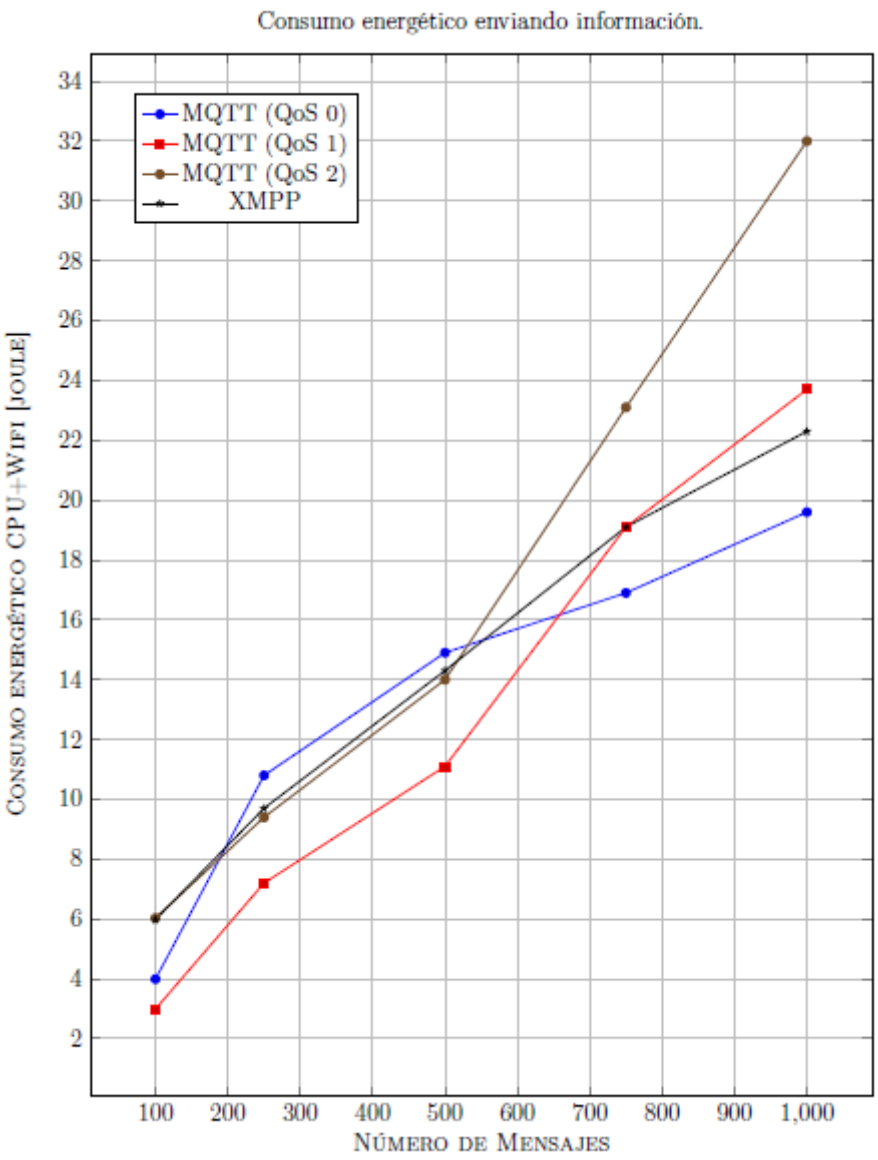


Figura 6.2: Consumo energético enviando información.
Fuente: Protocolos de Comunicación Para Aplicaciones Móviles.[5]

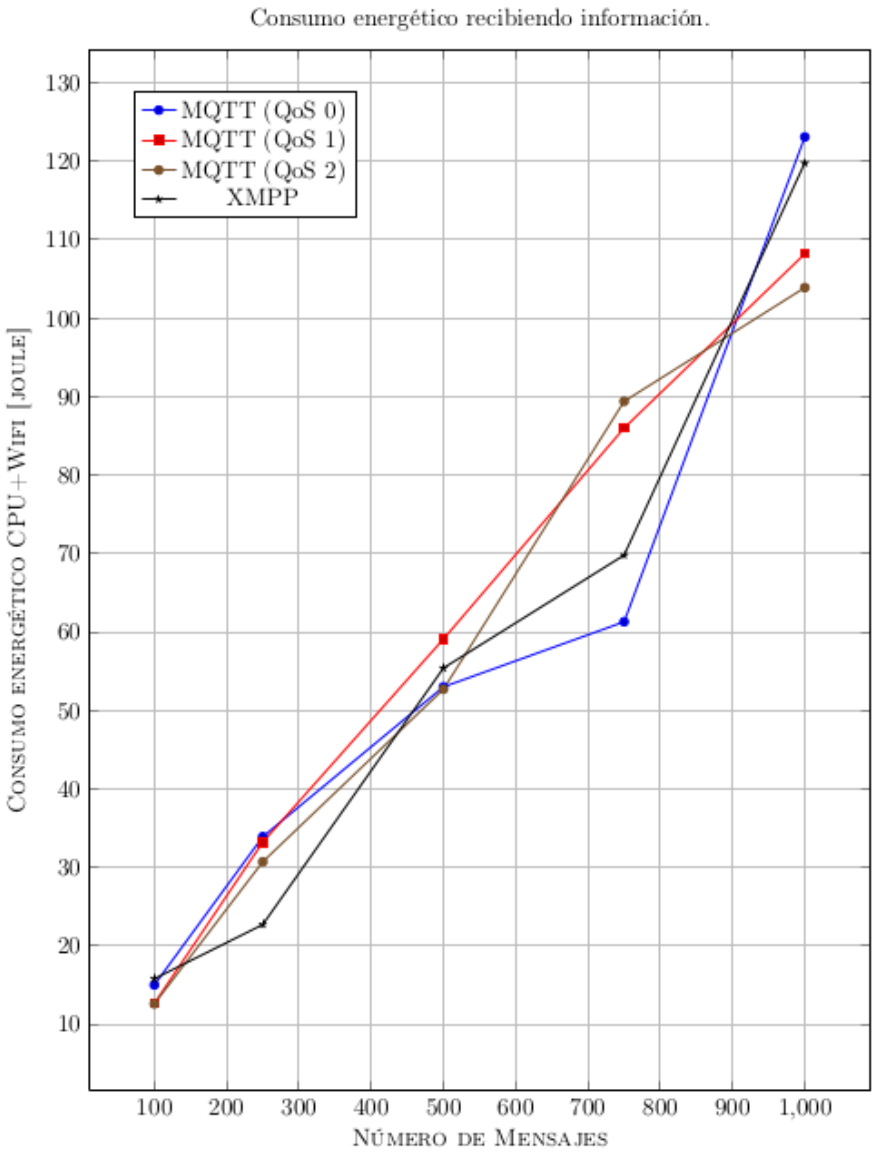


Figura 6.3: Consumo energético recibiendo información.
Fuente: Protocolos de Comunicación Para Aplicaciones Móviles.[5]

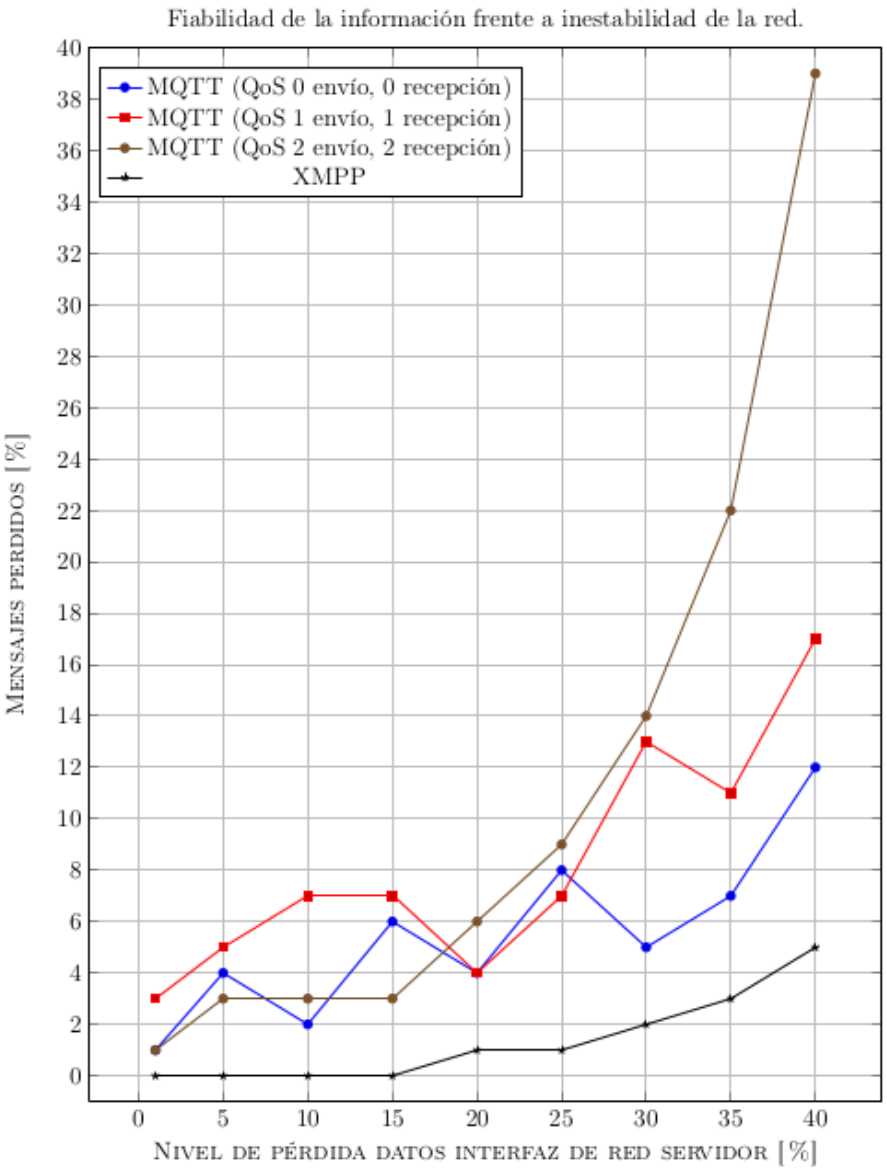


Figura 6.4: Fiabilidad de la información frente a inestabilidad de la red.
Fuente: Protocolos de Comunicación Para Aplicaciones Móviles.[5]