NOMBRE DEL TRABAJO
Tesis de maestria M16490944.pdf

AUTOR
lee lee

RECUENTO DE PALABRAS
12295 Words

RECUENTO DE CARACTERES
66367 Characters

RECUENTO DE PÁGINAS
76 Pages

TAMAÑO DEL ARCHIVO
4.7MB

FECHA DE ENTREGA
Sep 29, 2023 5:20 PM PDT

FECHA DEL INFORME
Sep 29, 2023 5:21 PM PDT

20% de similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para cada base d

- 19% Base de datos de Internet
- 1% Base de datos de publicaciones
- Base de datos de Crossref
- Base de datos de contenido publicado de Crossr
- 12% Base de datos de trabajos entregados

Excluir del Reporte de Similitud

- Material bibliográfico
- Material citado
- Coincidencia baja (menos de 10 palabras)

INSTITUTO TECNOLÓGICO DE MEXICALI

DIVISIÓN DE ESTUDIOS DE POSGRADO

DEPARTAMENTO DE SISTEMAS COMPUTACIONALES

TESIS

ANÁLISIS DE EFICIENCIA ENERGÉTICA Y ALGORÍTMICA APLICADA EN LOS SISTEMAS DE COMUNICACIÓN DE LOS DISPOSITIVOS (IOT)

QUE PARA OBTENER EL TÍTULO DE:

MAESTRO(A)

MAESTRÍA EN SISTEMAS COMPUTACIONALES

PRESENTA:

GABRIEL LEE ALVAREZ ROSADO

ASESOR:

VERÓNICA QUINTERO ROSAS

CO-ASESOR:

MARIO ALBERTO CAMARILLO RAMOS

MEXICALI, BAJA CALIFORNIA, MÉXICO. SEPTIEMBRE 2023

México B.C., 14/Agosto/2023
Oficio DEPI-86/23
ASUNTO: Autorización de impresión

C. GABRIEL LEE ALVAREZ ROSADO
PRESENTE

El que suscribe, Jefe de la División de Estudios de Posgrado e Investigación, comunica a usted que para la obtención del grado de Maestría en Sistemas Computacionales, se ha autorizado la reproducción de su trabajo de tesis, cuyo título es:

"ANALISIS DE EFICIENCIA ENERGETICA Y ALGORITMICA APLICADA EN LOS SISTEMAS DE COMUNICACION DE LOS DISPOSITIVOS IOT "

La autorización se emite con base a la revisión y dictamen emitido favorablemente y avalado con su rúbrica por los integrantes del Comité Tutorial, integrado por:

VERONICA QUINTERO ROSAS CAMARILLO RAMOS
Presidente

EDUCACIÓN
INSTITUTO TECNOLÓGICO DE MEXICALI
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

MARIO ALBERTO
Secretario

ROBERTO LOPEZ AVITIA
Vocal

ATENTAMENTE
Excelencia en Educación Tecnológica®
La Tecnología para el Bien de la Humanidad

ARNOLDO DIAZ RAMIREZ
Jefe de la División de Estudios de Posgrado e Investigación

Agradecimientos

Quiero agradecer a mi familia que estuvo conmigo apoyándome en todo momento para seguir preparándome profesionalmente, desde mis padres Lirio rosado y Gabriel Alvarez, que me apoyaron incondicionalmente. A mis hermanos Lirio Jatziri y Kenji Jared que me ayudaron a escribir y concentrarme en la tesis. También agradecer a mi novia Karen olivares que me ayudo a mantenerme concentrado y ayudarme en los momentos difíciles. También agradecer a mis maestros de maestría, Veronica Quintero y Mario Camarillo, que siempre me apoyaron y ayudaron en salir adelante.

Resumen

***Palabras clave*: IoT, Internet de las cosas, Consumo energético, Corriente, MQTT, Arduino uno wifi rev 2, ESP32, ESP8266, Protocolo,CoAP**

Actualmente, tanto en la industria como en el hogar se han sufrido varios cambios con el desarrollo de las nuevas tecnologías, y con ellas ha surgido la cuarta revolución industrial.

La cuarta revolución industrial ha desarrollado muchas tecnologías, una de ellas es el internet de las cosas (IoT), esta nos permite estar mas conectados que nunca a los distintos equipos y hacerlo de manera mas accesible. Los podemos observar en dispositivos inteligentes para el hogar como refrigeradores inteligentes, focos inteligentes, entre otras cosas, como los dispositivos compatibles con Google home y Alexa. De igual manera, en la industria se han observado áreas de oportunidad en donde involucrar estas tecnologías.

La mayoría de estos dispositivos funcionan a base de redes inalámbricas (RF), como wifi, Zeegbe, LoRa, entre otros. Al ser estos inalámbricos, algunos dispositivos funcionan de manera portable, por lo que la energía que estos necesitan es suministrada por una fuente finita (baterías). Las baterías juegan un papel esencial en las aplicaciones, ya que sin estos, los dispositivos portátiles quedarían inútiles.

Al ser estos dispositivos portátiles, es necesario un protocolo de comunicación para que los dispositivos puedan comunicarse con otros dispositivos, para esto existen varios protocolos como MQTT, HTTP, CoAP. A base de una investigación que realizo eclipse, se observo que el protocolo mas utilizado por los desarrolladores es MQTT, por lo que esta investigación se enfoco en realizar análisis al transmitir información por MQTT.

En esta investigación se propuso la metodología para obtener el consumo energético al transmitir datos mediante WIFI utilizando el protocolo MQTT, esta metodología se aplico para obtener el consumo energético en tres placas de desarrollo IoT, ESP32, ESP8266 y Arduino uno Wifi rev 2. En la investigación se concluyo que la placa ESP8266 obtuvo el menor consumo energético, con un

promedio de corriente de 77.328 mA al transmitir un mensaje por MQTT. La placa que contempla un mayor consumo fue ESP32 con un consumo promedio de corriente de 140.22 mA. Esto se evaluó con tres diferencias de potencial para obtener un consumo estimado al utilizar baterías con distinto potencial.

Abstract

***Keywords*: IoT, Energy, MQTT, CoAP, HTTP, WIFI,**

Actually the industry and the homes have changed, This happens due to technological progress. This developed the fourth industrial revolution. The fourth industrial revolution has developed a lot of technology, one of these things is the internet of things (IoT). IoT allows us to stay more connected than ever to distinct devices. We can see this technology on devices like Google home, Alexa.

Most of these devices work wit radio frequency systems (RF), like WIFI, Zeegbe, LoRa, among others. Due this devices are wireless, some of them are portable, so the energy needed to work is supplied by a finite energy source (batteries). Batteries do fundamental work in this application, because those devices without a power supply would be useless.

Due this devices are wireless, this use protocols to communicate with other devices, like MQTT, HTTP, LoRa, to name a few. Based on research of Eclipse, the most used protocol by the developers is MQTT. For this research, we developed a methodology to measure the energy consumed transmitting data through WIFI using MQTT. We use three different iot boards, ESP32, Arduino uno wifi rev 2 and ESP8266.

We obtained that board ESP8266 got the lowest energy consumption with 77.328 mA transmitting data. The board with the bigger consume was ESP32 with 140.22 mA. This was evaluated with three different power supply (3.7v, 5v, and 9v) to obtain the estimated consumed using batteries.

Índice general

Índice de tablas

Índice de figuras

Nomenclatura

Glosario

IoT	internet of things - Internet de las cosas.
mA	mili amperes.
mC	mili coulomb.
mJ	mili joules.

CAPÍTULO 1

INTRODUCCIÓN

Capítulo 1

Introducción

Actualmente, tanto en la industria como en el hogar se han sufrido varios cambios con el desarrollo de las nuevas tecnologías, y con ellas ha surgido la cuarta revolución industrial. Esta revolución implemento nuevas tecnologías, como el internet de las cosas (IoT - Internet Of things), siendo esta la que nos permite compartir y controlar información mediante internet.

Hoy en día podemos observar dispositivos IoT al alcance de nuestras manos, siendo la mayoría de la linea blanca inteligente, como lo son los refrigeradores, lavadoras, hornos, entre otros, y el ser inteligentes nos permite monitorear sus parámetros o sus acciones sin la necesidad de estar presentes. Estas tecnologías cada ves se hacen mas accesibles y pueden ser implementadas en distintas áreas, por ejemplo, en el sector industrial, en el que se estima que apenas un 30 % de las compañías de México han observado las ventajas del IoT [1] .

1.1. Planteamiento del problema

Debido al auge del internet de las cosas (Internet of things - IoT) en la actualidad, estos tienden a ser inalámbricos o wireless, por lo que surge la necesidad de tener su propio suministro de energía. Estos dispositivos IoT requieren de estar alimentados con una batería para poder utilizarce de manera portable, pero al acabarse el suministro de energía el equipo queda inútil hasta que el usuario realice mantenimiento al dispositivo. Al ser estos dispositivos portables o inalámbricos, es necesario que cuenten con diferentes tecnologías de radio frecuencia (RF) y un protocolo de comunicación, este protocolo nos permite

transmitir la información de manera correcta entre dispositivos. El proceso de comunicación demanda energía, esta acción ejecuta tareas internas de las que no tenemos conocimiento, y en medida que aumenta la demanda de transmisión aumenta el consumo energético de estos, el análisis de este consumo energético es crucial en aplicaciones en donde el dispositivo IoT tiene que estar encendido por grandes tiempos sin intervención del usuario.
Algunos fabricantes de microcontroladores [2] han brindado herramientas en software y hardware a los desarrolladores, estas herramientas permiten comprender el consumo energético del dispositivo de la compañía, pero estos procesos no son combatibles entre otras compañías.
Si se pudiera evaluar la energía al transmitir información de cualquier dispositivo, seria posible determinar que parte del software o hardware es la que consume al transmitir una información, al igual que se podría determinar la cantidad aproximada de mensajes totales posibles a transmitir con un dispositivo IoT, o cuando es necesario hacer un mantenimiento. Para esto se plantea el desarrollo de una metodología que nos permite realizar la medición de la energía al transmitir un dato por MQTT aplicable en cualquier dispositivo enfocado al internet de las cosas.

1.2. Hipótesis

Medir el consumo de energía del dispositivo IoT al transmitir un dato por MQTT nos permite analizar el comportamiento del equipo, además, esto nos permitirá calcular la cantidad de mensajes a transmitir por MQTT y cuando sería necesario darle servicio o mantenimiento a los dispositivos IoT.

1.3. Objetivos

1.3.1 Objetivo general

- Diseñar un método de evaluación del consumo energético en tarjetas de desarrollo con tecnología IoT.
- Evaluar el algoritmo de MQTT proporcionados por los fabricantes.

1.3.2 Objetivos específicos

- Diseñar un método de evaluación del consumo energético al transmitir un dato en una tarjeta de desarrollo IoT.
- Evaluar dicha tarea utilizando la misma tarjeta, con distintas fuentes de energía.

1.4. Justificación

1.5. Estado del arte

Con el aumento de la tecnología se ha optado por utilizar dispositivos electrónicos en diferentes áreas de interés. Estos dispositivos que han evolucionado enormemente generaron la 4ta transformación industrial, creando junto con ella los dispositivos con el internet de las cosas (Internet of Things, IoT). IoT describe la red de objetos físicos (cosas) que incorporan sensores, software y otras tecnologías con el fin de conectar e intercambiar datos con otros dispositivos y sistemas a través de Internet [3].

Con el paso de los años la aplicación de los dispositivos IoT ha aumentado de manera exponencial gracias a todas sus funcionalidades y su gran facilidad de adaptabilidad con el entorno, ha crecido tanto que supera la cantidad de población de la tierra, y se estima que seguirá creciendo en los próximos años, como se puede observar en la figura 1.1, por lo que se estima que para 2027 existirán 27 billones de dispositivos IoT conectados. Esta imagen puede ser encontrada en el siguiente fuente [4] .

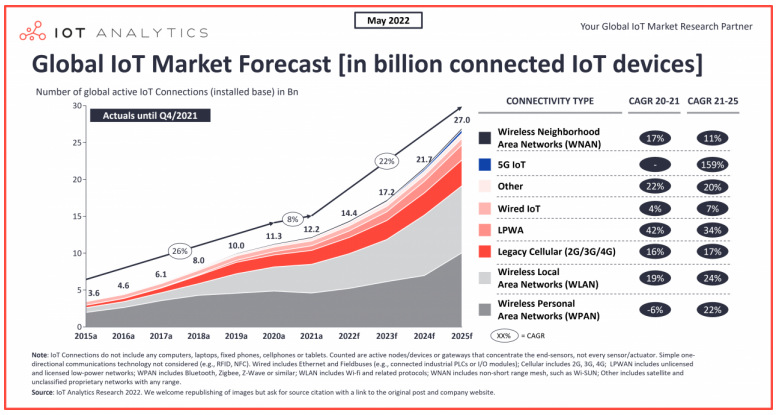


Figura 1.1: Cantidad de dispositivos IoT

El consumo energético se ha vuelto mas importante con el desarrollo de aplicaciones en la industria, creando consigo el nuevo termino Internet Industrial de las Cosas (Internet Industrial Of Things - IIoT).

Estos dispositivos IIoT han revolucionado la industria y se han aplicado en diferentes secciones, encontrando áreas de desarrollo [5] como la ganadera, en donde se implantan sensores en las orejas del ganado, esto funciona para conocer los signos vitales, sus movimientos, y mas parámetros que se quieran obtener. FoodLogiQ permite a los proveedores de alimentos etiquetar y rastrear sus productos a lo largo de una cadena de suministros, y a los consumidores verificar de donde provienen.

De igual manera, en la industria de la salud [6] se adquieren datos biomédicos con un dispositivo IoT, en el que busca registrar la temperatura y signos vitales del paciente, y estos datos obtenidos son almacenados en la nube.

También existen diferentes enfoques del internet de las cosas como Internet de las cosas verde (Green IoT), este está enfocado en la reducción del efecto invernadero en el mundo de los dispositivos IoT [7], el cual propone varias estrategias, como el de optimizar su consumo energético, mejorando el código utilizado buscando modos de ahorro de energía o de un manejo correcto de los mismos. En [8], proponen realizar operaciones en tres modelos distintos que aportan un consumo menor de 47.9 %, en donde el buen manejo del software puede optimizar el rendimiento de los dispositivos, esto reduciendo el mantenimiento junto con menos uso de baterías para realizar menor contaminación.

CAPÍTULO 2

MARCO TEÓRICO

Capítulo 2

Marco Teórico

2.1. Internet de las cosas (IoT)

El internet de las cosas (IoT) es la cuarta revolución industrial, la cual ha traído diferentes escenarios en el área de la manufactura, el hogar, medicina, entre otros, en donde se busca la presencia de un dispositivo inteligente, estos dispositivos buscan estar conectados entre si para así poder realizar metas en común.

2.2. Protocolos de comunicación

Los equipos IoT cuentan con la característica que pueden comunicarse con otros, para esto es necesario que puedan comprenderse entre ellos. Estos se comunican mediante reglas y formatos establecidos para una correcta comprensión entre el emisor y el receptor [9].
Estos protocolos de comunicación están regularmente en la séptima capa o nivel de aplicación del modelo OSI y generalmente en el cuarto nivel de la pila TCP [10]. El usuario normalmente no tiene acceso a esta capa de aplicación, solo interactúa con el software y este utiliza los protocolos de comunicación. En esta capa aparecen diferentes protocolos:

2.2.1 Protocolos

- **FTP** (File transfer protocol - Protocolo de transferencia de archivos) para transferencia de archivos.

- **DNS** (Domain Name System - Sistema de nombres de dominio).
- **DHCP** (Dynamic Host Configuration Protocol - Protocolo de configuración dinámica de anfitrión).
- **HTTP** (HyperText Transfer Protocol - Protocolo de configuración dinámica de anfitrión).
- **COAP** (Constrained Applications Protocol Secure - Protocolo seguro de transferencia de hipertexto).
- **MQTT** (Message Query Telemetry Transport - Protocolo de sistemas de mensajería asíncrona).
- **POP** (Post Office Protocol) - Para recuperación de correo electrónico.
- **SMTP** (Simple mail Transport Protocol).
- **SSH** (Secure SHell).
- **TELNET** (Teletype Network).
- **TFTP** (Trivial File Transfer Protocol).
- **LDAP** (Lightweight Directory Access Protocol).
- **XMPP** (Extensible Messaging and Presence Protocol - Protocolo estándar para mensajería instantánea)

2.2.2 MQTT

MQTT son las siglas de MQ Telemetry Transport, aunque en primer lugar fue conocido como Message Queing Telemetry Transport. Este es un protocolo de comunicación M2M (Machine to Machine) de tipos de mensajes queue [11]. Esta basado en la pila TCP/IP como fundamento para la comunicación. En el caso de MQTT cada conexión se mantiene abierta y se ”reutiliza” en cada ocasión.

MQTT fue creado originalmente por el Dr. Andy Stanford-Clark y Arlen Nipper en 1999. El propósito original del método de comunicación era permitir que los

dispositivos de monitoreo utilizados en la industria del petróleo y el gas enviaran sus datos a servidores remotos. MQTT se estandarizó como fuente abierta bajo la Organization for the Advancement of Structured Information Standards (OASIS) en 2013. OASIS todavía administra el estándar MQTT.

El funcionamiento de MQTT es un servicio de mensajería push con patron publicador/suscriptor (Pub-Sup). Como se describe anteriormente, el cliente se conecta a un servidor central que se denomina broker. Para diferenciar los mensajes enviados a diferentes clientes se utilizan los ”Topicos” o ”Topics” que permite organizar y poder comunicar entre distintos dispositivos. Un cliente puede suscribirse a un mismo tópico y el broker hará que los mensajes lleguen a todos los clientes suscritos. Se puede observar la arquitectura de comunicación del protocolo MQTT en la figura 2.1, en donde el broker administra la información recibida y transmitida.

Los clientes inician una conexión TCP/IP con el broker, el cual mantiene un registro de los clientes conectados, y esta conexión se mantiene abierta hasta que el cliente la finaliza. Por defecto, MQTT emplea el puerto 1883 y 8883 cuando funciona sobre TLS.

Para ello el cliente envía un mensaje CONNECT que contiene información necesaria (Nombre de usuario, contraseña, client-id...). El broker responde con un mensaje CONNACK, que contiene el resultado de la conexión (Aceptada, rechazada, etc). Para enviar los mensajes el cliente emplea mensajes PUBLISH, que contienen el topic y el payload. Para suscribirse y desuscribirse se emplean mensajes SUBSCRIBE y UNSUSCRIBE, que el servidor responde con SUBACJ y UNSUBACK. Por otro lado, para asegurar que la conexión esta activa los clientes mandan periódicamente un mensaje PINGREQ que es respondido por el servidor con PINGRESP. Finalmente, el cliente se desconecta enviando un mensaje DISCONNECT.

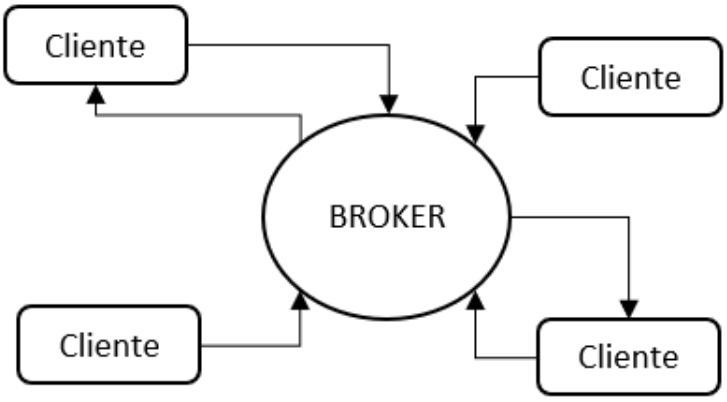


Figura 2.1: Arquitectura de comunicación MQTT

2.2.3 Estructura de mensaje MQTT

Uno de los componentes mas importantes del protocolo MQTT es la definición y la topología de los mensajes, ya que son una de las bases de la agilidad en la que radica su fortaleza. Cada mensaje consta de 3 partes:

- Cabecera fija:
  Ocupa de 2 a 5 bytes, obligatorio. Consta de un código de control, que identifica el tipo de mensaje enviado y de la longitud de este. La longitud se codifica en 1 a 4 bytes, de los cuales se emplean los 7 primeros bits, y el ultimo bit de continuidad.

- Cabecera variable:
  Opcional, contiene información adicional que es necesaria en ciertos mensajes o situaciones.

- Contenido (Payload):
  Es el contenido real del mensaje. Puede tener un máximo de 256 Mb aunque en implementaciones reales el máximo es de 2 a 4 KB.

Los tipos de mensaje y códigos de control que se envían en el protocolo MQTT se pueden observar en la figura 2.2.

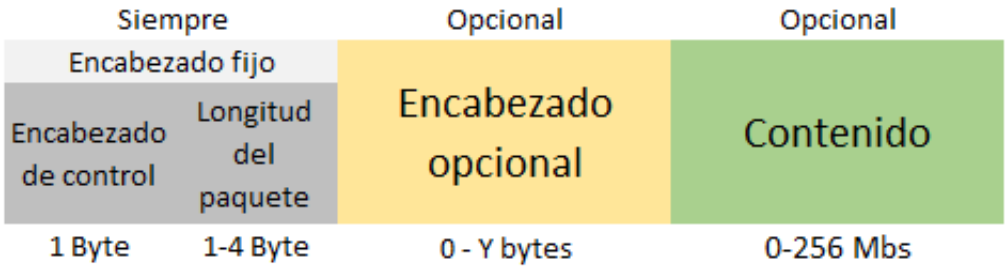


Figura 2.2: Estructura de mensaje MQTT

2.2.4 Seguridad MQTT

La seguridad siempre debe ser un factor importante a considerar en cualquier sistema de comunicación M2M. El protocolo MQTT dispone de distintas medidas de seguridad que podremos adoptar para proteger las comunicaciones.
Esto incluye transporte SSL/TLS y autenticación por usuario y contraseña mediante certificación. Sin embargo, hay que tener en cuenta que muchos disponen de escasa capacidad, por lo que el SSL/TLS puede suponer una carga del proceso importante. En muchos casos, la autenticación consiste en una contraseña y usuario que son enviados como texto plano. Por ultimo, también es posible configurar el broker para aceptar conexiones anónimas.
Todo esto debe ser tenido en cuenta a la hora de configurar un sistema MQTT, y entender los riesgos de cada uno de ellos, así como su impacto en la eficiencia del sistema.

2.3. COAP

Es un protocolo a nivel de aplicación pensado para ser usado en dispositivos electrónicos simples permitiendo que pueda comunicarse sobre internet, está especializado en la transferencia web para usar nodos restringidos y del mismo modo, redes restringidas [12]. Estos nodos usualmente tienen microcontroladores de 8-bits con una pequeña cantidad de ROM y RAM, mientras que las redes restringidas tal como IPv6 a través de Low-Power Wireless Personal Area Networks (LoWPANs) a menudo tienen una gran tasa de paquetes erróneos y su rendimiento

habitual es de 10x de kbits/s.

Esta pensado para sensores de baja potencia y para aplicaciones destinadas al intercambio Máquina a Máquina (M2M). Por otro lado este protocolo implementa el modelo REST de HTTP, el cual es un estilo de arquitectura para sistemas hipermedia (conjunto de métodos para escribir, diseñar o componer contenidos) distribuidos como la World Wide Web (Red informática mundial), por otro lado CoAP utiliza también cabeceras reducidas y limita el intercambio de mensajes, añadiendo soporte UDP.

Dicho protocolo también provee un modelo de interacción de solicitud/respuesta entre la aplicación y los puntos finales, esta diseñado con una interfaz HTTP simple para la integración web mientras cumple con los requisitos especializados de soporte de multicast, de muy bajo costo para la simplicidad de los entornos restringidos, una representación de como funciona el protocolo CoAp se puede ver en la figura 2.3.

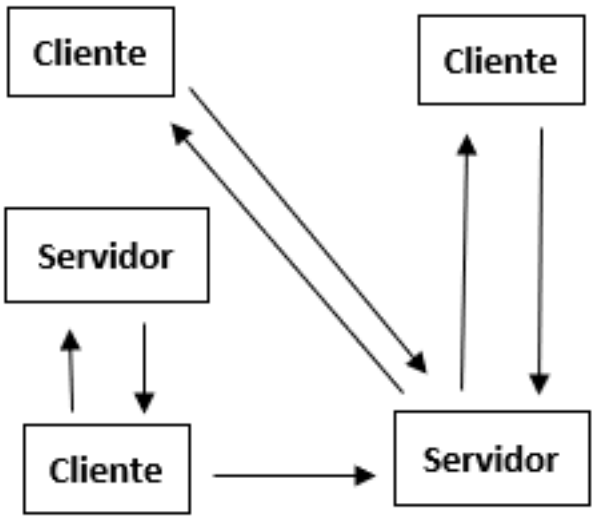


Figura 2.3: Arquitectura de comunicación CoAP

2.3.1 Características COAP

- Reducción de sobrecargas TCP por medio de una vinculación UDP en opciones de fiabilidad unicast y apoyo multicast de peticiones.
- El diseño del protocolo traduce fácilmente a HTTP la integración simplificada

con la web.

- Intercambio de mensajes asíncronos.
- Baja sobrecarga de cabeceras para reducir la complejidad al analizar el mensaje.
- Apoyo de URI y contenido (Content-Type).
- Búsqueda de recursos y mecanismos de suscripción opcional.
- Caching simple basado en MAX-AGE.

2.3.2 Estructura del mensaje CoAP

Los mensajes CoAP están divididos en tres partes:

- Cabecera CoAP: Contiene información básica que permite el reconocimiento de una versión CoAP, tipo, código y el ID del mensaje.
  - Versión (Ver): Indica el numero de versión CoAP.
  - Tipo (T): Si el mensaje es tipo CON, NON, ACK, RST.
  - Contador de opciones (OC): Número de opciones después de la cabecera, si este campo es rellenado con un ”0.es porque no hay opciones y se pasa directamente al campo de carga útil (Payload).
  - Código: Indica si el mensaje es una petición (con calores de 1 a 31) indicado en el Request method”, si es una respuesta (valores de 64 a 191) indicando el ”Response Code” o si esta vació (Valor 0).
- Opciones CoAP: Provee los parámetros para rellenar peticiones.
- Carga útil CoAP: Contiene el cuerpo del mensaje.

La estructura se puede ver organizada en la figura 2.4, esta sigue las indicaciones antes vistas.

Offset	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
4	32	VER		Type		Token Length				Request/Response Code								Message ID															
8	64	Token (0-8 Bytes)																															
12	96																																
16	128	Options (If available)																															
20	160	1	1	1	1	1	1	1	1	Payload (If available)																							

Figura 2.4: Estructura de mensaje CoAP

2.4. Consumo de energía

El uso de dispositivos inteligentes que manejen cuando se va a realizar una acción ha permitido que los dispositivos portátiles alimentados por baterías tengan una mayor duración en sus aplicaciones. Es posible utilizar metodologías en estos equipos para administrar los módulos internos, así como los aditamentos que tengan para realizar una mejora en el consumo energético.

Hay muchas maneras de llevar un seguimiento del consumo energético en baterías de uso casero [13] que permiten la correcta administración de la energía almacenada, y es posible aplicar estas técnicas a placas de desarrollo IoT. Estas placas pueden administrar sus periféricos internos, así como también los componentes qué conformen el equipo para ser mas inteligente el consumo de energía. Como podemos observar en [14], se hizo la demostración de que un equipo puede funcionar por tres meses aproximadamente, con un consumo promedio de $5mA$ alimentado por una batería alcalina estándar de $2A$. El mismo equipo administrado por metodologías de ahorro de energía, optimizándola y haciendo el algoritmo mas inteligente podría bajar el consumo de corriente a $500uA$, lo que puede extender la duración de la batería a casi el doble, es decir, seis meses.

Las metodologías a las que comúnmente se refieren para el ahorro de consumo se dividen en dos: hardware y software [15]. Para el hardware se aplican en la parte del semiconductor y entre las uniones de los módulos a utilizar. Estas implementaciones no son modificables. Diferentes fabricantes de microcontroladores ya cuentan con estas tecnologías de ahorro de energía [16] [17] (Energymicro, Renesas, Texas Instruments, Microchip, entre otros). Algunas empresas proporcionan herramientas para calcular el consumo energético de cada instrucción en un bloque de código ejecutado en el microcontrolador, con estas

herramientas es posible estimar el perfil de energía del equipo [18].
Las metodologías de optimización por software están divididas en análisis de manejo de algoritmos a nivel código fuente, a nivel algoritmo y arquitectura del software, y en esta parte el usuario o desarrollador tiene un mayor control del consumo energético que puede tener el equipo [15].

2.5. Perfil de energía

El concepto de bajo consumo de energía varía dependiendo el rubro. Para estas aplicaciones, reducir la energía consumida por la placa de desarrollo al máximo es lo esencial. En muchas aplicaciones, el suministro de energía no es ningún problema, aun así se puede optimizar el consumo para reducir costos o maximizar la eficiencia [19]. Cuando los equipos funcionan de manera portable, cuentan con sistemas finitos de alimentación (baterías), la autonomía de estos equipos esta ligada a la capacidad de la fuente, por lo que en estos momentos requieren tener un análisis del perfil energético para optimizar la autonomía del mismo.
Los dispositivos que utilizan tecnología CMOS dividen la energía en dos tipos: dinámica y estática. La energía estática es la que se consume simplemente al estar en reposo y energizado. La energía dinámica es la que se consume cuando el dispositivo esta ejecutando alguna rutina del proceso [20].

2.6. Técnicas de bajo consumo de energía

2.6.1 Hardware

Cuando los circuitos CMOS conmutan y polarizan los circuitos analógicos, es parte de la energía dinámica. Esta energía se calcula para cada uno de los circuitos utilizando la ecuación 2.1:

.

$$E_{Dinamica} = V^2 * f * c \tag{2.1}$$

En donde V es el voltaje de operación del sistema (VDD), f es la frecuencia de conmutación y C es la capacitancia de carga de cada uno de los elementos CMOS

[20] [21].

Los parámetros como la frecuencia y la capacitancia son independientes para cada dispositivo, ya que el fabricante selecciona sus frecuencias de conmutación, y la obtención de los parámetros de carga son complicados de conseguir. El único parámetro que tenemos acceso a modificar, es el voltaje de polarización VDD. Reducir el voltaje es una manera de disminuir la energía consumida por el microcontrolador.

Otras maneras de optimizar el consumo energético es realizar una acción en los puertos de los microcontroladores. La función de dichos puertos puede ser establecida ya sea como entrada o salida. Si se elige entrada, estos puertos deben ser conectados con resistencias a VDD o VSS. Si se elige configurar el puerto como salida es posible dejarla flotando, pero es preferible elegir un voltaje estático para que no active y/o desactive la región lineal de los circuitos CMOS internos [19] [20] [22].

Otra manera es utilizar las configuraciones de bajo consumo de energía que el fabricante proporciona. Estas configuraciones pueden ser activadas en etapas de procesamiento o ejecución del algoritmo, por lo que pueden llegar a confundirse con implementaciones de software, pero estos se consideran de hardware debido a que son especificaciones diseñadas por los fabricantes para los microcontroladores [23] [19] [20] [24].

2.6.2 Software

El impacto del algoritmo implementado en relación a la energía consumida fue estudiado por el autor Tiwari [25], el cual realizo experimentos ejecutando diferentes instrucciones y técnicas de programación para observar la energía consumida por el algoritmo. Obtuvo buenos resultados al reducir el consumo de energía en un 40 % y plantear nuevas técnicas de compilación [26].

En distintas investigaciones se ha demostrado que la manera de programar impacta directamente al consumo energético del dispositivo, otras investigaciones observaron que el nivel de optimización utilizado para generar el código maquina tiene repercusión en el uso de la energía [27] [28] [29]. Las técnicas que utilizan los compiladores pueden afectar al consumo energético. El compilador puede convertir un lenguaje de alto nivel a código maquina de una manera muy reducida para

poder implementarlo en dispositivos de menos recursos, pero la mayoría de los compiladores producen el código para una ejecución rápida, por lo que aumenta el consumo de energía [29]. El no utilizar técnicas de optimización para la generación de código maquina en los distintos compiladores podría ser una manera de optimizar el consumo de energía, pero esto puede conllevar a tener hasta un 250 % mas de instrucciones [29].

A medida que la frecuencia de operación aumenta, el consumo también lo hace, aunque de manera contraproducente, en casos en donde se utiliza una frecuencia baja puede llegar a tener un mayor consumo de energía [23] [29]. El tiempo que se invierte en ejecutar algoritmos complejos se extendería, por lo que la energía acumulada tendría a elevarse, el objetivo es realizar algoritmos inteligentes que busquen durar el menor periodo posible de ejecución aprovechando todos los recursos del equipo y regresar al estado mínimo o de reposo cuando no se este ejecutando una acción [23] [29].

El tiempo que se invierte en algoritmos complejos sería mayor, por lo que la energía acumulada tendería a elevarse. La tendencia es utilizar algoritmos eficientes durante periodos cortos al máximo de la capacidad que proporciona el dispositivo y regresar al estado de mínimo consumo de energía [23] [29]. Otros trabajos han demostrado que al momento de desarrollar algoritmos es necesario cuidar el número de variables a utilizar, el tipo de variables y como acceder a ellas. Al igual que como esta diseñado el dispositivo para acceder a la memoria afecta al consumo de energía [26] [27] [28] [29].

2.7. Medición de energía

La energía es un parámetro que no se puede obtener directamente, para conseguir este es necesario recabar la potencia que consume el dispositivo bajo prueba (DUT) durante el periodo en el que se interesa medir. Esta información es necesario integrarla. En la mayoría de las ocasiones es preciso adquirir el promedio de energía, y para esto se realiza una integral con la potencia obtenida.

Cuando tenemos un circuito ejecutando una aplicación especifica y de ahorro de energía [24], la energía total consumida por el dispositivo, es la integral de la potencia instantánea del periodo de tiempo de la señal recabada.

Si la corriente y el voltaje se obtienen de manera instantánea, los parámetros de

energía y potencia también serán instantáneos. Al ser instantáneo podemos determinar específicamente cual es el consumo en el área de interés. A continuación, se presentan algunas de las más utilizadas.

2.8. Bobina de Rogowski

La bobina de Rogowski es utilizada para medir la corriente alterna que consume un DUT. Es una bobina en forma circular que tiene en sus extremos un circuito integrador. Este tipo de método de adquisición de energía tiene la ventaja de que no es invasiva, pero esta implementación tiene los requisitos de ser solo utilizada con corriente alterna y generalmente corrientes mayores en comparación a los DUT[30]. Una implementación del circuito se observa en la figura 2.5, en donde se puede apreciar la bobina y un circuito sencillo integrados para obtener la corriente que pasa entre la bobina.

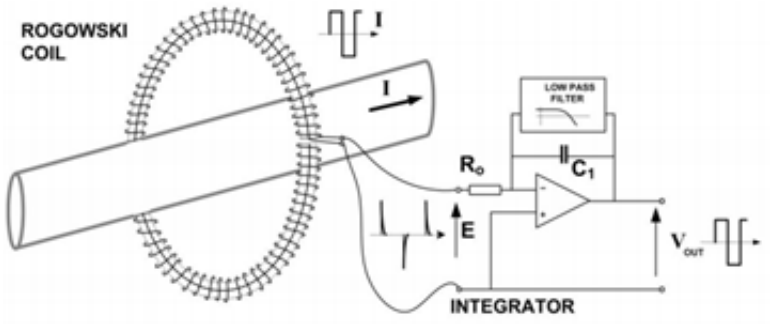


Figura 2.5: Bobina de Rogowski

2.9. Espejo de corriente

El espejo de corriente es un circuito o técnica para replicar la corriente que esta consumiendo un circuito en otro circuito. Las ventajas de esta metodología es la alta impedancia de salida, esto permite mantener la corriente independiente de la carga [31]. Se observa en la figura 2.6 la base del espejo de corriente [32] [33] [34] y [35].

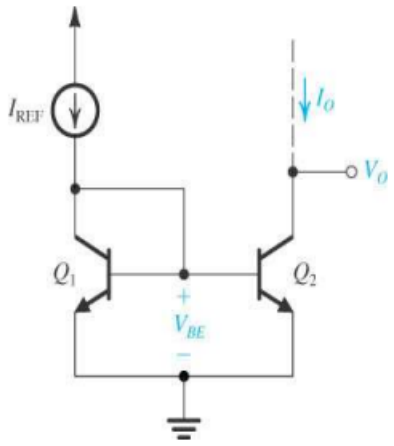


Figura 2.6: Espejo de corriente

2.10. Carga en capacitor

Otras metodologías para medir energía es medir la carga almacenada en un capacitor, al realizar pruebas a los DUT, estos se comportan como fuentes de corriente constante, esta metodología implica utilizar la ecuación 2.2 la que nos habla sobre como medir la descarga de corriente del capacitor [20], al utilizar un voltaje constante durante el periodo de prueba, nos permite realizar una relación para obtener la corriente consumida por el microcontrolador.

$$I = C\frac{dv}{dt} \tag{2.2}$$

Donde I es la corriente que consume el dispositivo en amperes, V es el voltaje generado en volts, tiempo es t en segundos y C es la carga almacenada en el capacitar en Faradios.

2.11. Efecto Hall

También es posible medir la corriente utilizando el efecto Hall. Este comportamiento se presenta a través de un conductor por el que fluye corriente. Este genera un campo magnético perpendicular en sentido de la corriente y produce un voltaje proporcional en sentido a la corriente que fluye por el conductor. La ecuación 2.3 presenta la expresión necesaria para la derivación de la corriente a través del efecto Hall [36].

$$V_{Hall} = \frac{IB}{qnd} \tag{2.3}$$

Donde V_{HALL} es el voltaje en volts del efecto Hall, I es la corriente que fluye a través de un conductor, B es el campo magnético, q es la carga del electrón, n es la densidad de cargas en movimiento, y d es el diámetro del conductor por donde circula la corriente. Esta metodología puede ser utilizada para capturar corrientes pequeñas a escala de microamperes [37].

2.12. Resistencia shunt

Una de las técnicas mas utilizadas es el de resistencia shunt, debido a lo sencillo y practico de su utilización. Esta técnica consta en colocar una resistencia en el paso de la corriente a medir, y realizar la medición entre la resistencia. Este tipo de técnica se rige con la ecuación de corriente de ohm, como se observa en la ecuación 2.4.

$$I = \frac{V}{R} \tag{2.4}$$

Las principales ventajas de utilizar esta técnica es su bajo costo, una alta precisión y su gran posibilidad de medir pequeños flujos de corriente. La desventaja es que es necesario agregar la resistencia en el lazo de polarización, a causa de esto, la disipación de energía debido a la potencia generada por la resistencia, puede llegar a modificar el comportamiento del dispositivo si la caída de tensión es significativa en la resistencia shunt [38]. La medición de energía a través de la resistencia es

de manera directa, pero existen dos variaciones al utilizarla, que son la medición de lado alto y la medición de lado bajo.

2.12.1 Medición de lado alto

Esta es cuando la resistencia se encuentra entre la fuente de alimentación y el dispositivo del que se quiere obtener la corriente. La ventaja de utilizar esta configuración es que el DUT esta conectado directamente a tierra. Sin embargo, para realizar este tipo de técnica, el dispositivo que se utilice para capturar el voltaje tiene que tener la capacidad de medir altos valores de voltaje, como se puede observar en [39], donde se realizan modelados para la medición del consumo de energía por instrucciones.

2.12.2 Medición de lado bajo

Cuando se quiere realizar esta configuración, la resistencia se coloca entre el sistema bajo prueba y tierra. Una de las ventajas de esta es la referencia a tierra para realizar las mediciones, y el dispositivo que se utilice para capturar el votaje no tiene que tener obligatoriamente la capacidad de medir altos voltajes.

2.13. Soluciones comerciales

Estas son soluciones que proporcionan las empresas utilizando distintas técnicas o metodologías para poder obtener la corriente. La empresa Tektronix ofrece una punta de prueba (TCP312) para osciloscopio que permite realizar mediciones del orden de miliamperes [40].
La empresa EEMBC ofrece un sistema de inyección de carga para el monitoreo del consumo de energía llamado EnergyBench [41]. El que plantea realizar modulación de la energía aplicada al sistema bajo prueba utilizado. También existen productos con una resistencia shunt, como el uCurrent probe de cmicrotek [42], el cual sera muy similar al que se usara en esta investigación.

CAPÍTULO 3

DESARROLLO

Capítulo 3

Desarrollo

Actualmente algunos desarrolladores de algoritmos para placas de desarrollo realizan aplicaciones sin tener en cuenta el consumo energético que su aplicación puede tener. Para esto se propone facilitar la obtención del perfil energético, esto ayudara al usuario desarrollador a optimizar su algoritmo. El consumo energético puede variar por muchos factores, como la arquitectura, la cantidad de núcleos, el compilador, hasta el software utilizado para realizar el algoritmo.
Generalmente los dispositivos IoT se alimentan con fuentes de energía finitas (Baterías), y estos se comunican mediante radiofrecuencia (WIFI, LoRa, Zigbee, bluetooth, Etc.) a otros dispositivos, lo que hace fundamental el empleo de técnicas de ahorro de energía para optimizar la eficiencia energética. Por ejemplo, un dispositivo IoT que es alimentado con una batería de 3.7v consume un promedio de 7 mC al transmitir un mensaje por MQTT, si se utiliza una batería de 500 mA (1800 coulomb), podemos estimar que puede transmitir un aproximado de 257000 mensajes por MQTT. Esta estimación de mensajes es fundamental al momento de diseñar, ya que puede que la transmisión de datos de manera constante no sea necesaria, y así poder prolongar en gran medida la duración de la fuente de energía.

3.1. Herramientas

- **Gpe 1.0**. Se desarrollo un software en Labview que permite ingresar una señal pre-procesada de corriente. Al ser ingresada, este software muestra los perfiles energéticos obtenidos de la señal.

- **Eclipse Mosquitto**. Es un software agente de mensajes de código abierto que implementa las versiones 5.0, 3.1.1 y 3.1 del protocolo MQTT, existen múltiples softwares que permiten administrar los mensajes por MQTT, pero Mosquitto es un software liviano y versátil, que puede ser utilizando en dispositivos de baja y alta potencia para realizar una comunicación con MQTT.
- **Diadem**. Es una herramienta que nos permite procesar señales obtenidas, este nos posibilita seleccionar el momento en el que se inicio la transmisión de información por medio de MQTT.
- **Analog Waveform**. Este software nos permite realizar la comunicación con el osciloscopio Analog Discovery 2, nos permitía guardar en la memoria los canales necesarios para realizar el perfil energético.
- **Analog Discovery 2**. Es un osciloscopio portátil de 2 canales, este nos proporciona la señal de corriente y disparo necesaria para sincronizar la lectura de transmisión de datos.
- **Current Ranger**. Este dispositivo transforma la corriente que pasa por la resistencia shunt interna a voltaje. Este admite corrientes desde los nA hasta los mA.

3.2. Metodología

En esta sección se planteará la metodología para realizar la medición de energía consumida por las placas de desarrollo IoT. Se establecen 4 secciones, la primera establece el algoritmo utilizado, la segunda sección establece el servidor Mosquitto, la tercera sección establece el equipo y aplicación del mismo y la cuarta sección refiere al post-procesamiento de la información obtenida.

3.2.1 Algoritmo

Se propuso una metodología para la medición de consumo energético al transmitir por MQTT, se realiza la transmisión del carácter ASCII ”A” y el tópico

"Test". El lenguaje usado para la implementación de las pruebas es C++. En la figura 3.1 se muestra el diagrama de flujo de la metodología. Este inicia despertando el microcontrolador. Después de realizar una configuración del microcontrolador, se establece la comunicación WIFI con la red. Posteriormente, se establece la conexión con el servidor Mosquitto. En este momento, se realiza un pulso de sincronía, dejando en estado alto un puerto de la placa de desarrollo, el cual nos permite empezar a hacer la medición de corriente. Luego del pulso de sincronía, se manda el carácter "A" con el tópico "Test". Al transmitir el carácter necesario, y para finalizar, se coloca en estado bajo el puerto de sincronía, al terminar los procesos se activa el modo ahorro de energía.

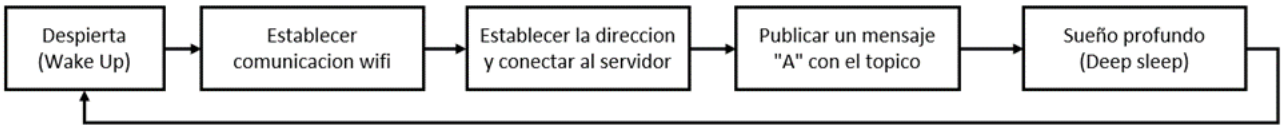


Figura 3.1: Diagrama de flujo algoritmo

Esta metodología es posible implementarla en distintas placas de desarrollo. En este caso se aplico en 3 placas de desarrollo distintas, siendo ESP32, ESP8266 y Arduino Yun.

3.2.2 Algoritmos de transmisión

El siguiente algoritmo fue el utilizado para realizar la comunicación MQTT con el servidor Mosquitto en la placa de desarrollo ESP32.

```
#include <WiFi.h>
#include <PubSubClient.h>
//Reemplazar las siguientes variables con su SSID y contraseña
char* ssid = Wifi_ssid;
char* password = Password;
const char* mqttServer = IP_Servidor;
const int mqttPort = 1883;
const char* mqttUser = MQTT_USER;
const char* mqttPassword = MQTT_PASSWORD;
WiFiClient espClient;
```

```
PubSubClient client(espClient);
const char* topicName = "Test"; // Topic con el que trabajamos
void setup()
{
  Serial.begin(115200);
  pinMode(13,OUTPUT);
  pinMode(14,OUTPUT);
  client.setServer(mqttServer, 1883);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(50);
  }
while (!client.connected()) {
    if (client.connect("ESP32Client", mqttUser, mqttPassword ))
  {
    } else {
      delay(20);
    }
}
}
void loop()
{
  digitalWrite(13,HIGH);
  client.publish(topicName, "A");
  digitalWrite(13,LOW);
  esp_sleep_enable_timer_wakeup(10000); // ESP32 wakes up every 10 mseg
  esp_deep_sleep_start();
}
```

El siguiente algoritmo fue el utilizado para realizar la comunicación MQTT con el servidor Mosquitto en la placa de desarrollo ESP8266.

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
//Reemplazar las siguientes variables con su SSID y contraseña
```

```
char* ssid = Wifi_ssid;
char* password = Password;
const char* mqttServer = IP_Servidor;
const int mqttPort = 1883;
const char* mqttUser = MQTT_USER;
const char* mqttPassword = MQTT_PASSWORD;
WiFiClient espClient;
PubSubClient client(espClient);
const char* topicName = "Test"; // Topic con el que trabajamos
void setup()
{
  Serial.begin(115200);
  pinMode(13,OUTPUT);
  pinMode(14,OUTPUT);
  client.setServer(mqttServer, 1883);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(50);
  }
while (!client.connected()) {
    if (client.connect("ESP32Client", mqttUser, mqttPassword ))
  {
    } else {
      delay(20);
    }
}
}
void loop()
{
  digitalWrite(13,HIGH);
  client.publish(topicName, "A");
  digitalWrite(13,LOW);
  ESP.deepSleep(1000);
}
```

El siguiente algoritmo fue el utilizado para realizar la comunicación MQTT con el servidor Mosquitto en la placa de desarrollo Arduino Uno Wifi rev2.

```
#include "LowPower.h"
#include <Bridge.h>
#include <BridgeClient.h>
#include <PubSubClient.h>
IPAddress mqtt_server(192, 168, 31, 192); //Ip de Servidor
BridgeClient briClient;
PubSubClient client(briClient);
unsigned char dat = 255;
// Topic con el que trabajamos
const char* topicName = "test";
void setup()
{
  Serial.begin(57600);
   Bridge.begin();
  client.setServer(mqtt_server, 1883);
}
void loop()
{
  if (!client.connected()) {
  //  Serial.print("Connecting ...\n");
    client.connect("Arduino Client");
  }
  else {
    // Envio
    client.publish(topicName, "A");
  }
  // Tiempo entre envios (en ms)
  client.loop();
  //delay(50);
LowPower.powerDown(SLEEP_15MS, ADC_OFF, BOD_OFF);
}
```

Estos tres algoritmos nos sirven para transmitir información a través de internet. El servidor fue creado en una computadora local. El proceso de instalación sera explicado en la siguiente sección.

3.2.3 Servidor Mosquitto

Para la realización de la metodología, es necesario contar con un servidor MQTT para la administración de los mensajes, se opto por utilizar el software Mosquitto, el cual nos permite crear un servidor en diferentes equipos. Para instalar Mosquitto es necesario recurrir a la pagina oficial [43].
En este caso seleccionamos la versión adecuada para nuestro dispositivo (Windows x64).

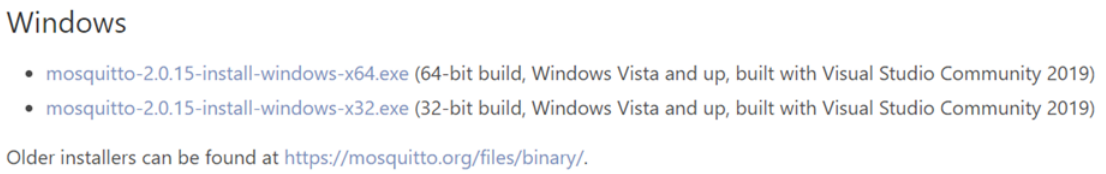


Figura 3.2: Opciones de instalación en Windows

Al descargar el software, es necesario seguir las instrucciones del instalador de Mosquitto, figura 3.3.

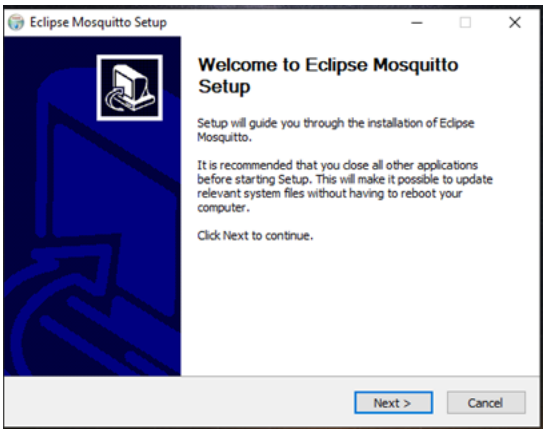


Figura 3.3: Panel de instalación Mosquitto

Al tener instalado el software MQTT es necesario abrir como administrador la aplicación ”Símbolo del sistema”, en esta pestaña se requiere colocar la ruta de

instalación del software Mosquito, para esto, colocamos la siguiente instrucción: ”cd $C : \ProgramFiles\mosquitto$”. En caso de que instalaran Mosquitto en otro directorio, es crucial especificarlo. Se puede ver la selección de la ruta de instalación en la figura 3.4.

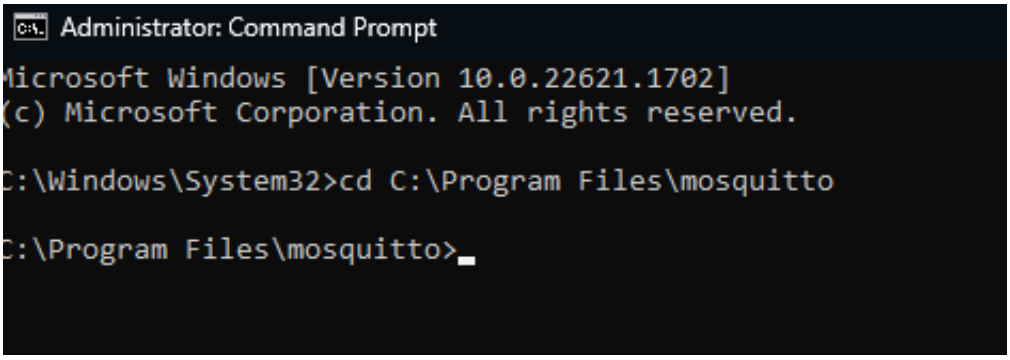


Figura 3.4: Ingreso de la ruta de instalación Mosquitto

Para iniciar el servidor MQTT, ejecutamos la instrucción ”net start mosquitto” en la pestaña ya mencionada.

3.2.4 Configuración de Firewall

Es fundamental configurar el puerto a utilizar por el servidor, para esto entramos al firewall de windows, y seleccionamos ”opciones avanzadas”, como se puede ver en la figura 3.5.

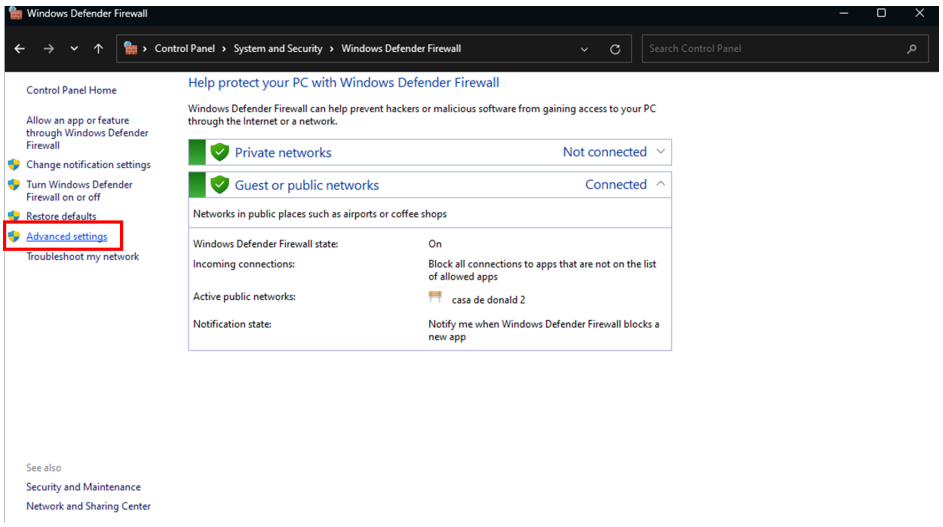


Figura 3.5: Firewall - Opciones avanzadas

Al entrar en modo de opciones avanzadas, creamos una nueva regla, como se puede ver en la figura 3.6

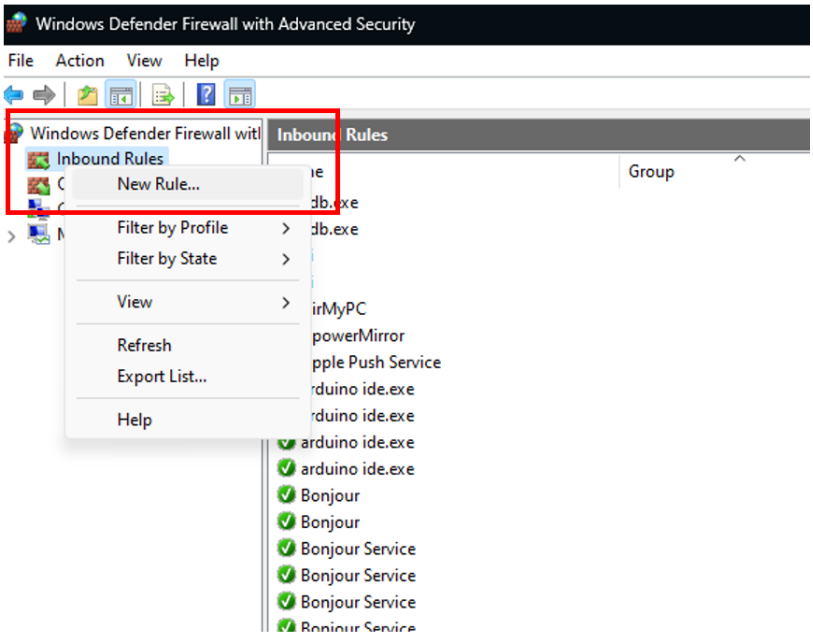


Figura 3.6: Agregar una nueva regla

Seleccionamos la opción ”puerto” (en este caso ”port”), como se puede ver en la figura 3.7.

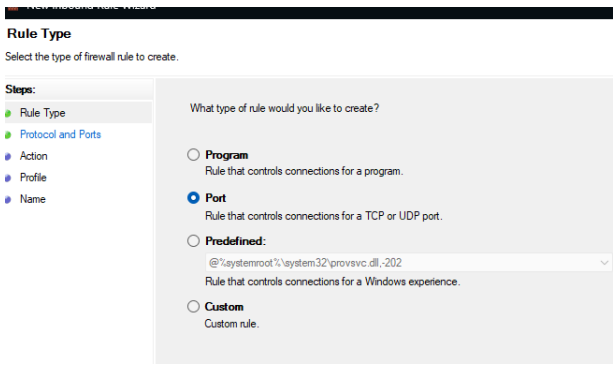


Figura 3.7: Nueva regla - Puerto

Después, seleccionar la opción ”TCP” y colocar el puerto ”1883”, como se puede ver en la figura 3.8.

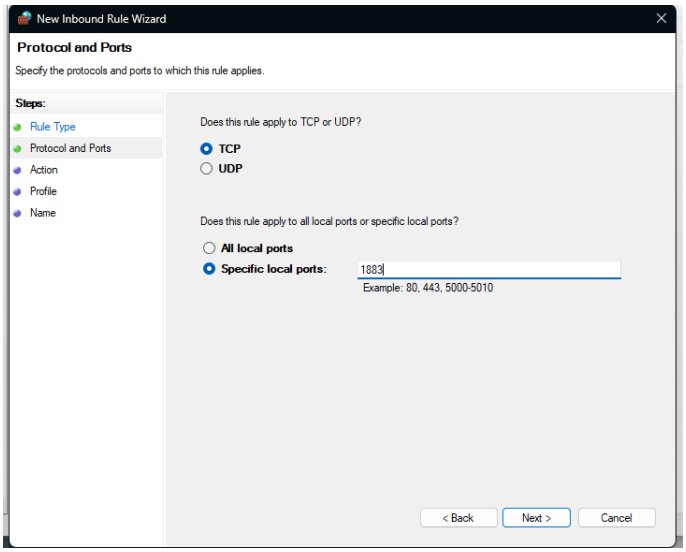


Figura 3.8: Puerto TCP

A continuación, se selecciona la opción ”Permitir la conexión” como se puede ver en la figura 3.9

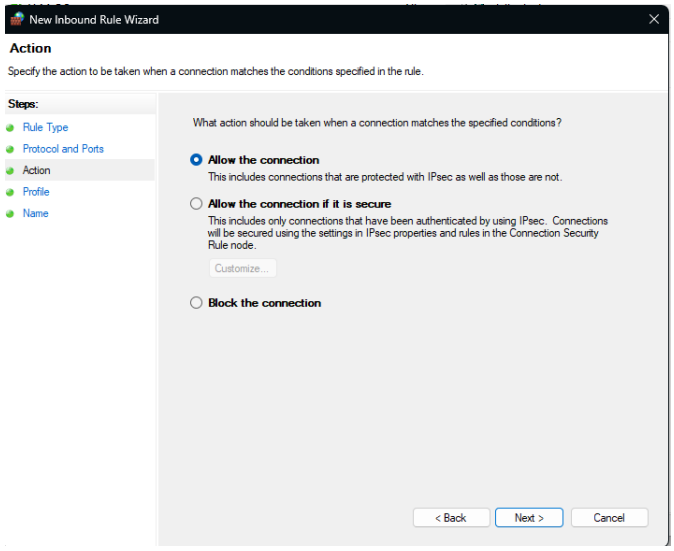


Figura 3.9: Permitir conexión

En el siguiente paso aparecen por defecto los tres recuadros marcados, en caso de no ser así, marcar los recuadros y dar ”siguiente”.

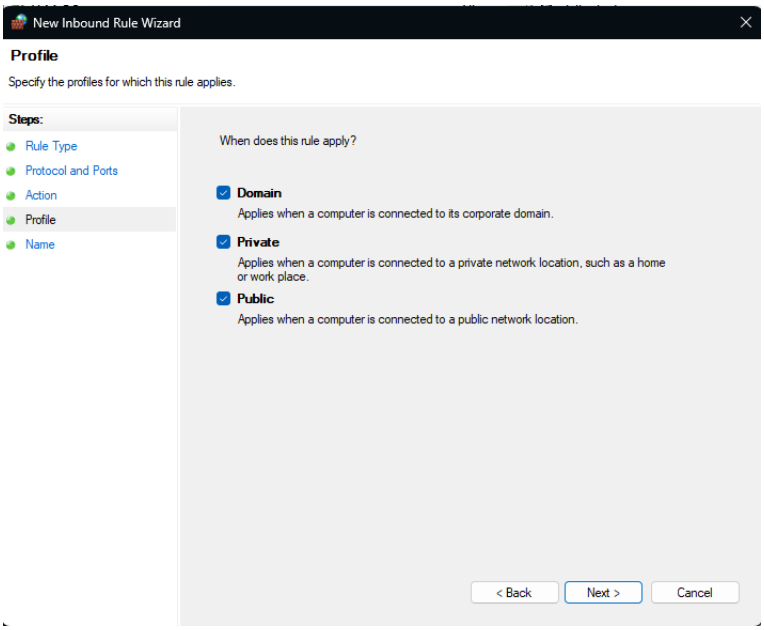


Figura 3.10: Aplicación de reglas

Se le asigna un nombre a la regla, esta puede ser llamada de cualquier modo, pero la llamaremos con el nombre del servidor "Mosquitto", como se puede ver en la figura 3.11.

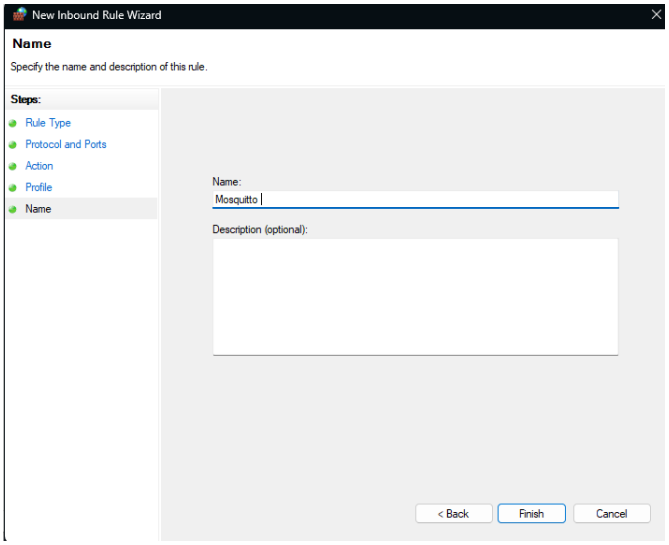


Figura 3.11: Asignación de nombre

Al tener esta regla de entrada asignada, también creamos una regla de salida, para esto seleccionamos nueva regla de salida, como se puede ver en la figura

3.12.

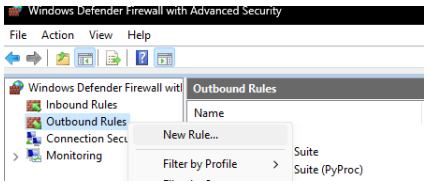


Figura 3.12: Nueva regla de salida

Es necesario especificar que en esta regla se utilizara un puerto TCP, como se puede ver en la figura 3.13.

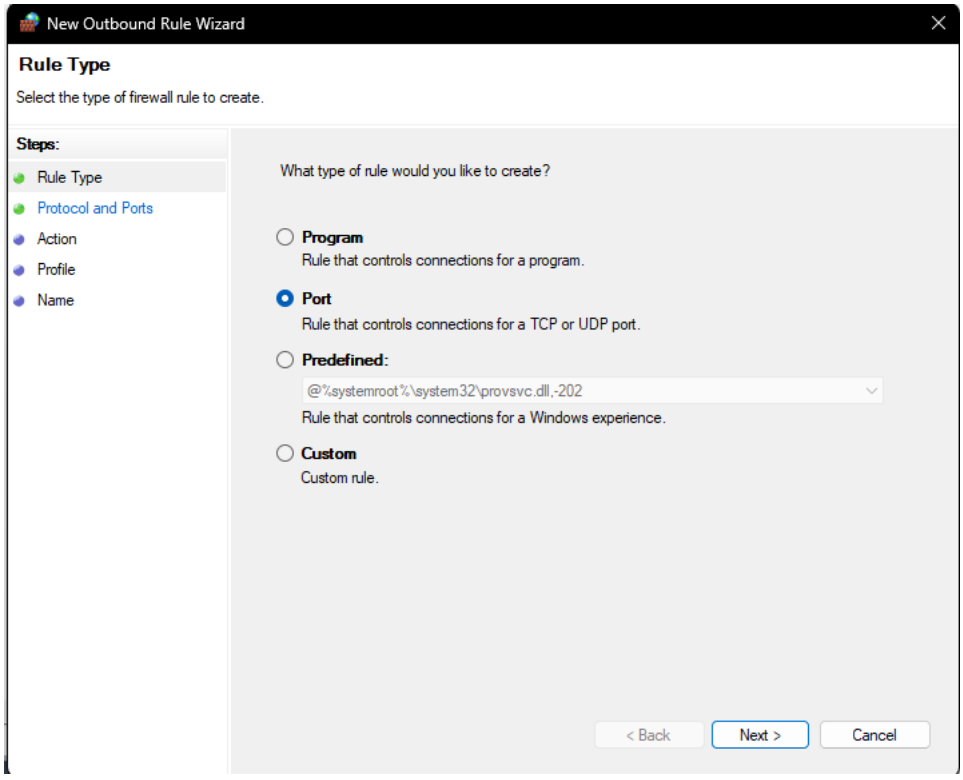


Figura 3.13: Opción de regla

Al avanzar en la configuración, seleccionamos ”TCP” y el puerto ”1883”, como se puede observar en la figura 3.14.

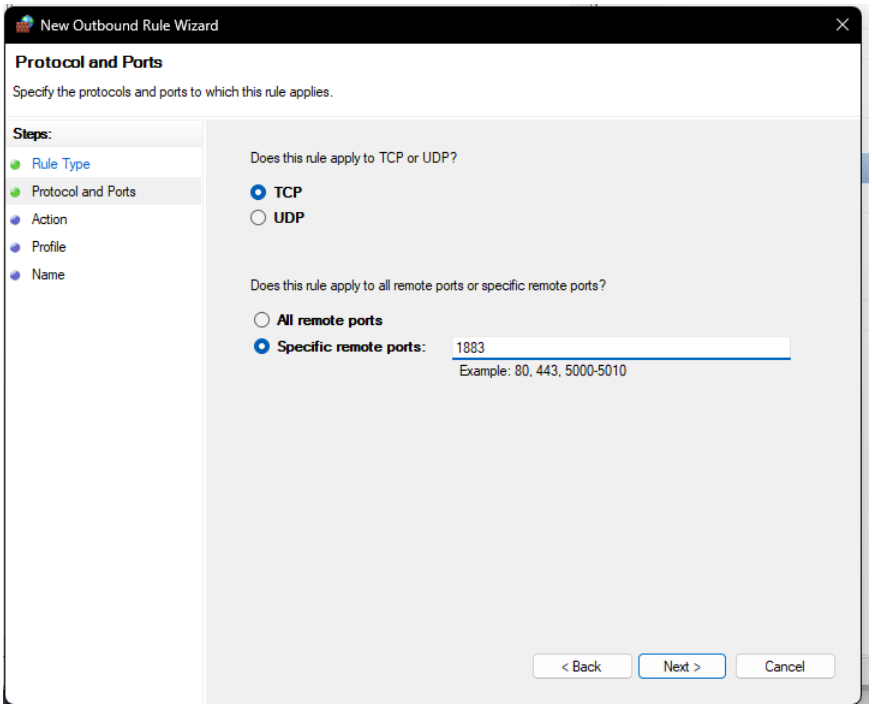


Figura 3.14: Selección de puerto

En la siguiente instrucción, elegimos la opcion ”Permitir la conexión”, como se puede observar en la figura 3.15

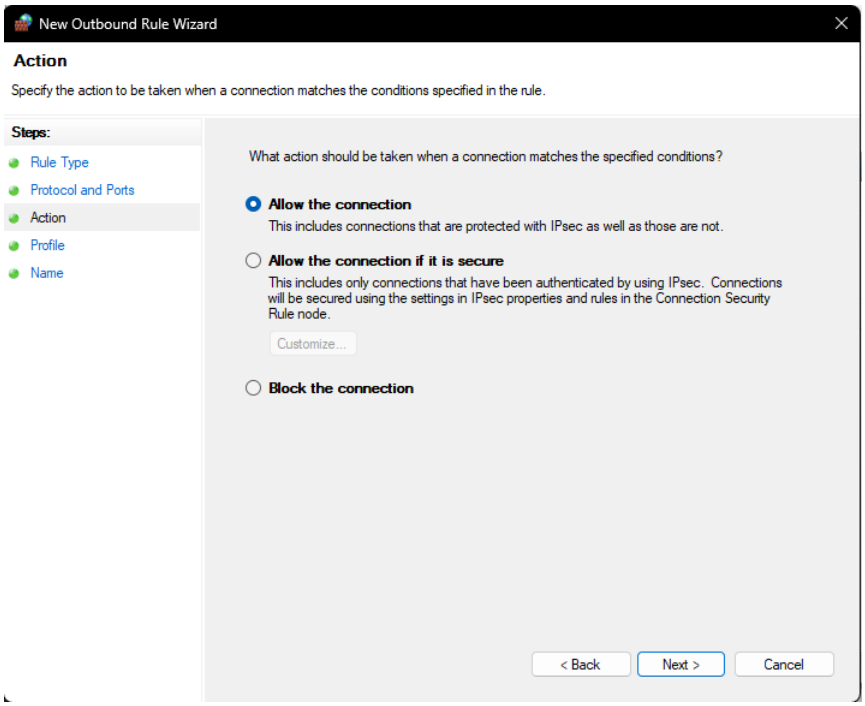


Figura 3.15: Permitir conexiones

En el siguiente paso, se marcan las 3 casillas, como se puede ver en la figura 3.16.

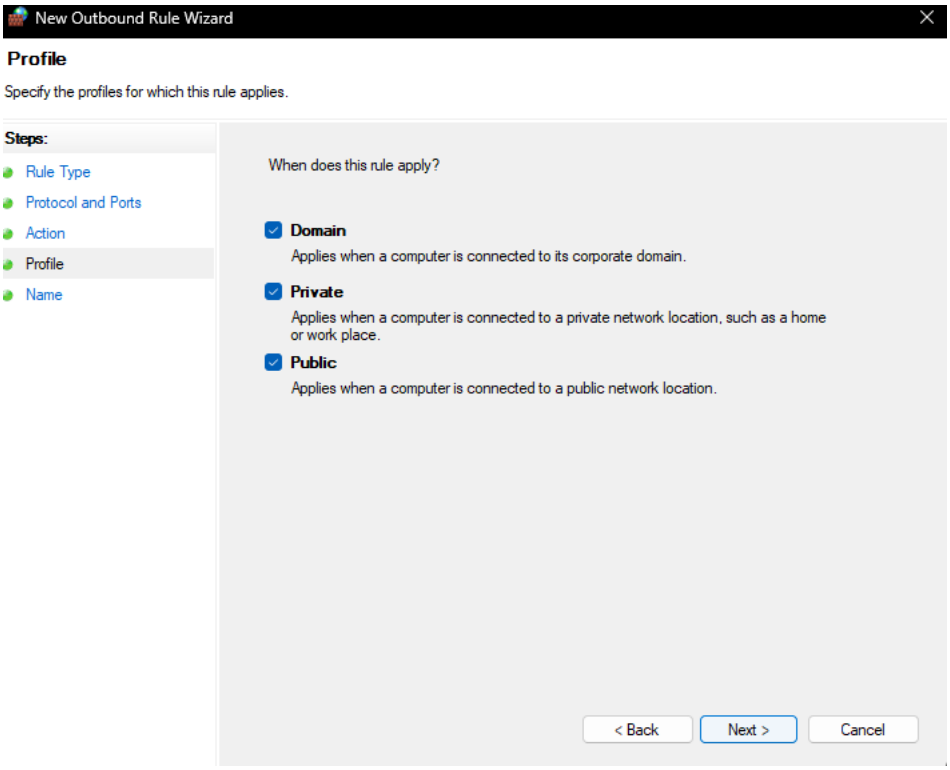


Figura 3.16: Aplicación de la regla

Para finalizar la regla, le asignamos un nombre, en este caso ”Mosquitto”. figura 3.17

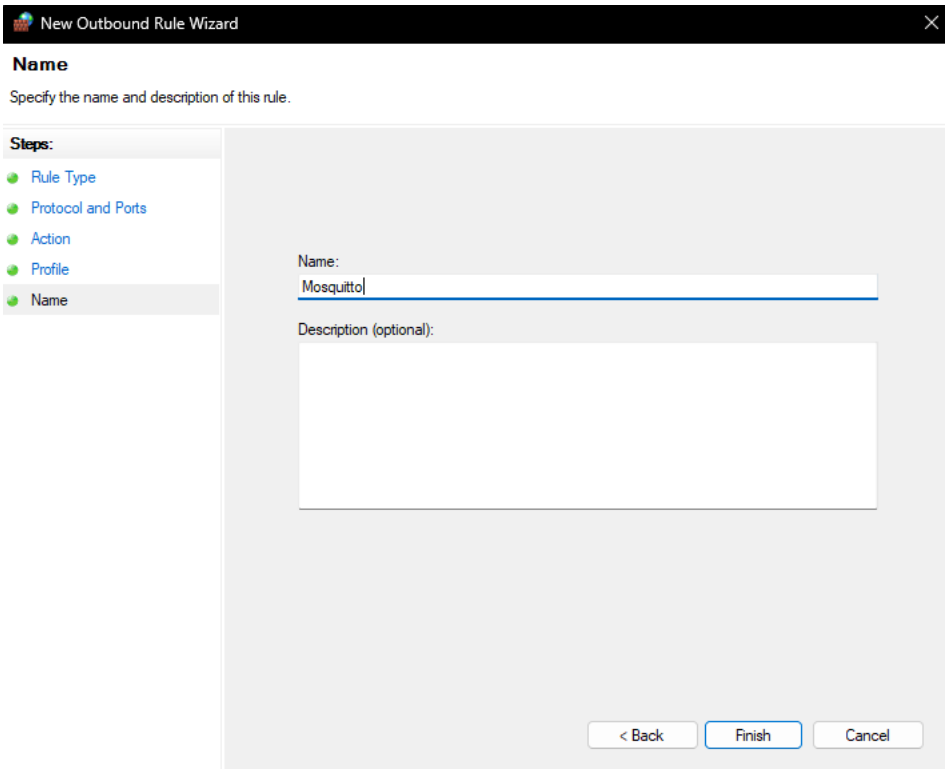


Figura 3.17: Asignación de nombre - Regla

Al estar configurado el firewall, realizamos configuraciones directas en el servidor Mosquitto, para permitir que dispositivos externos al servidor puedan interactuar.

3.2.5 Configuración de servidor Mosquitto

Para esto modificamos el archivo ”mosquitto.conf”, este archivo se encuentra ubicado en la ruta de instalación determinada por el usuario. A fin de poder modificar este documento, se requiere utilizar un editor de texto como ”sublime text” o ”Block de notas”.
Al abrir el documento, se busca la sección llamada ”Listeners” y se ingresan las siguientes instrucciones:

listener 1883 0.0.0.0
allow_anonymous true

La primera instrucción nos permite escuchar cualquier dispositivo desde cualquier

ubicación, y la segunda nos da la facultad de conectar dispositivos no autenticados al servidor.

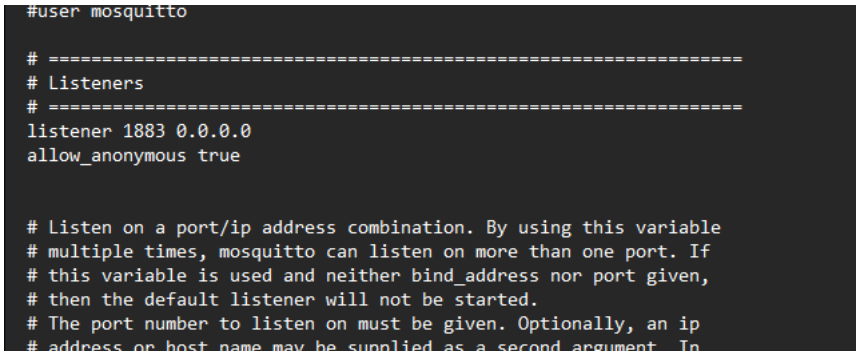


Figura 3.18: configuración del archivo mosquitto.conf

Al terminar de configurar el archivo conf, es preciso cargar las nuevas configuraciones, para esto se abre la aplicación ”Simbolo del sistema” y se selecciona la dirección de instalación, después se ejecuta la siguiente linea ”mosquitto -c mosquitto.conf”. Al ejecutar la instrucción, ya es posible utilizar el servidor Mosquitto con cualquier dispositivo IoT.

3.3. Equipo

En esta sección se observará la implementación de la metodología para poder obtener la corriente en un circuito bajo prueba, en este caso una placa de desarrollo IoT.

Para esto se propusieron las conexiones de la figura 3.19. Primero se realiza la conexión del dispositivo bajo prueba (DUT) a la fuente de alimentación a través del dispositivo Current ranger, este dispositivo cuenta con una resistencia shunt interna, que permite realizar la medición de lado bajo. El dispositivo Current ranger suministra de manera analógica la corriente obtenida en escala de mili Volts (mV), en el cual es posible seleccionar la escala necesaria o que funcione con un ajuste automático.

La señal analógica es capturada por el osciloscopio, en este caso se utilizó el osciloscopio Analog Discovery 2, el cual funciona con la aplicación Waveform, esta aplicación nos permite cambiar los parámetros del osciloscopio.

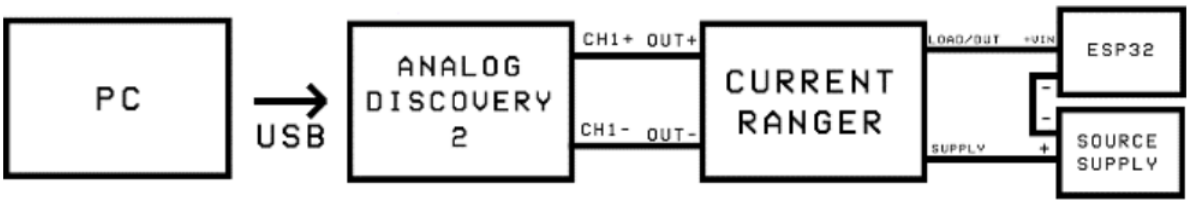


Figura 3.19: Metodología - Medición de corriente

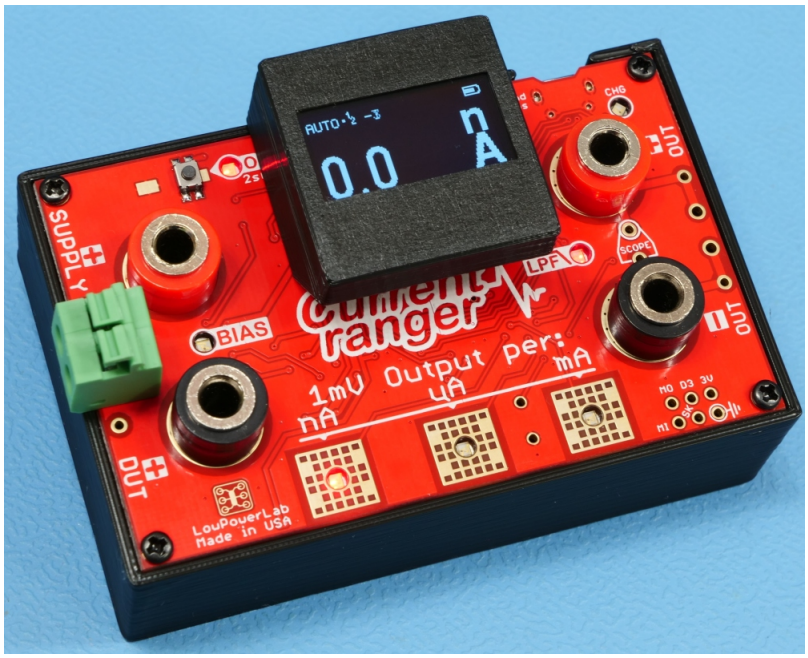


Figura 3.20: Dispositivo - Current ranger

Figura 3.21: Interface de aplicación waveform

Para obtener una sincronía, se utiliza una bandera o señal que nos permita saber en que parte se empezara a realizar la acción a medir. Este pulso de sincronía ayuda a nuestro sistema de adquisición de datos, utilizando los disparos de la aplicación waveform, estas opciones que se pueden modificar para realizar lecturas se observan en la figura 3.22.

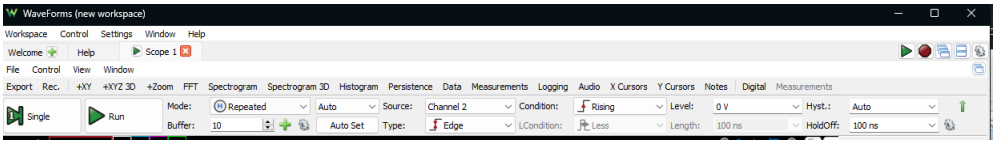


Figura 3.22: Parámetros de disparo

Esto nos permite establecer los parámetros de la bandera a utilizar y obtener la medición de corriente necesaria, una vez conseguida la corriente, se utilizo el software para crear una base de datos. Para obtener esta base de datos, exportamos los datos visualizados en el software con la función .Exportar"que se encuentra en la casilla "Files", como se puede ver en la figura 3.23.

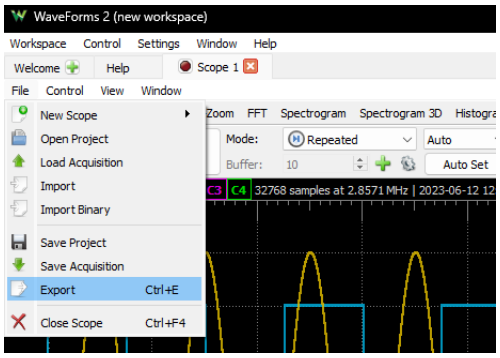


Figura 3.23: Opción - Exportar a base de datos

En esta sección, se podrán visualizar los datos y la imagen obtenida por el osciloscopio, como se puede ver en la figura 3.24.

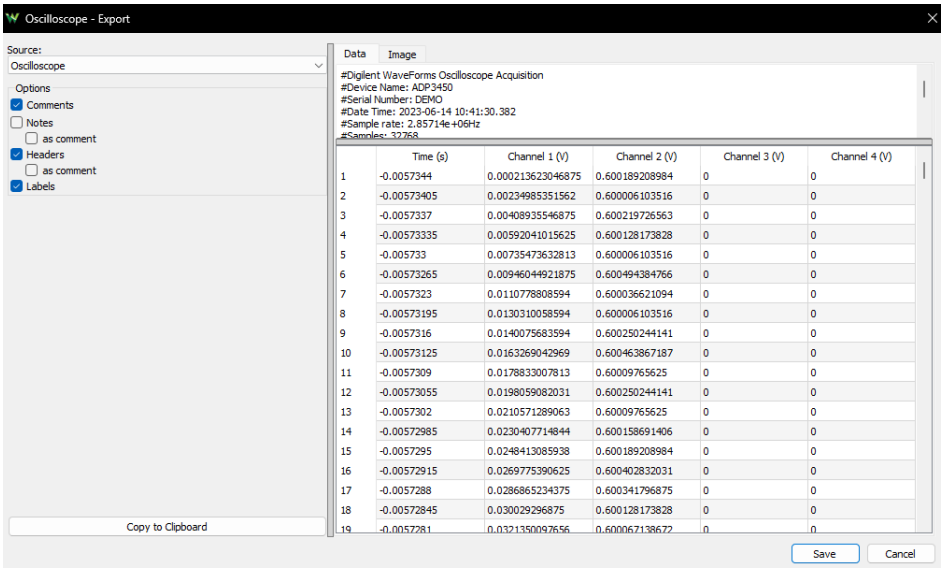


Figura 3.24: Sección - exportar

Esta base de datos sera guardada en formato TDMS (Technical data management system), el cual es un formato especializado en guardar datos en disco [44].

3.4. Post-Procesamiento

En la sección de post-procesamiento, se verá como se consiguió la información necesaria para obtener una conclusión de las señales, para esto se utilizaron la

aplicación de Diadem y una aplicación desarrollada en LabView llamada GPE1.0 (Graficador de perfil energético). Esta segunda herramienta nos permite insertar una base de datos y nos proporciona los perfiles energéticos de esta misma.

3.4.1 Diadem

Para iniciar el procesamiento, se ejecuta la aplicación ”Diadem”, la cual nos ayuda a realizar modificaciones en señales capturadas en distintos formatos. Al abrir el software, este inicia en la sección ”Navigator” en donde se debe de cargar la base de datos conseguida por el osciloscopio. Se puede ver en la figura 3.25 la ventana principal en donde se tendrá que buscar la base de datos requerida.

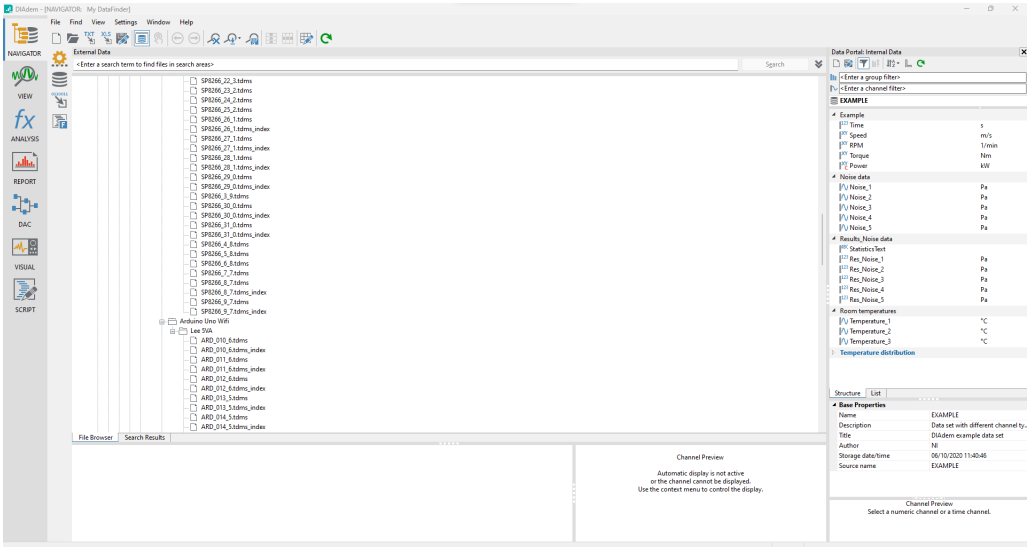


Figura 3.25: Aplicación Diadem

Como se puede observar en la figura 3.26, la página principal esta dividida en dos secciones, en la parte de búsqueda se ocupa encontrar la base de datos a procesar, luego, seleccionar esta base de datos con el botón derecho del ratón y elegir la opción ”Load Data”, al hacer esto, en la parte derecha se podrá observar el contenido de la base de datos.

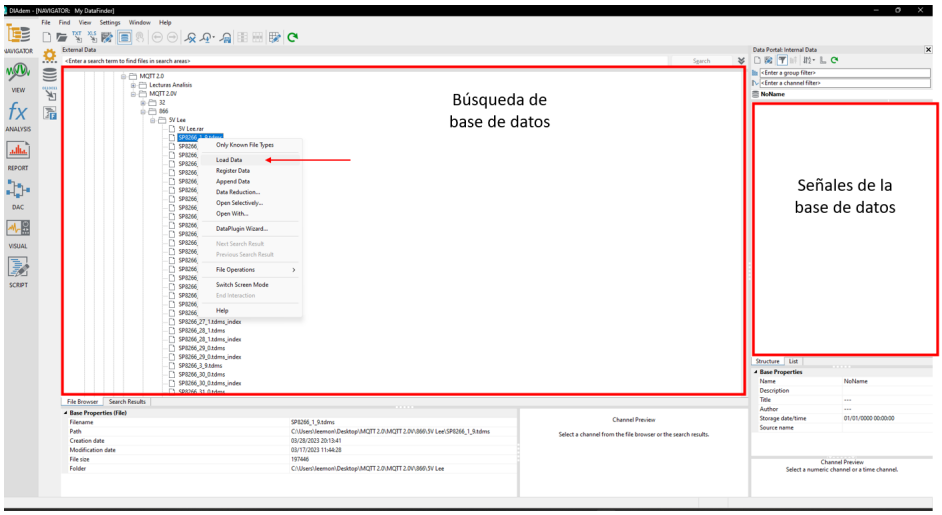


Figura 3.26: Características de página de inicio

Como se puede ver en la figura 3.27, se visualizan 3 señales distintas, una de tiempo, y una por cada canal del osciloscopio, en este caso 2.

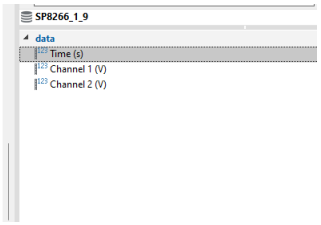


Figura 3.27: Señales obtenidas

En nuestro caso, el sistema waveform nos suministra las dos señales del osciloscopio sin una base de tiempo adecuada, mas bien de muestras. Para cambiar esto, se combinan las señales ”Channel 1” y ”2” con la señal ”Time”, este proceso es realizado en la pestaña de ”analysis”, seleccionar símbolo FX y usar ”Numeric channels $-$ $>$ waveforms channels”, esta selección se puede ver en la figura 3.28.

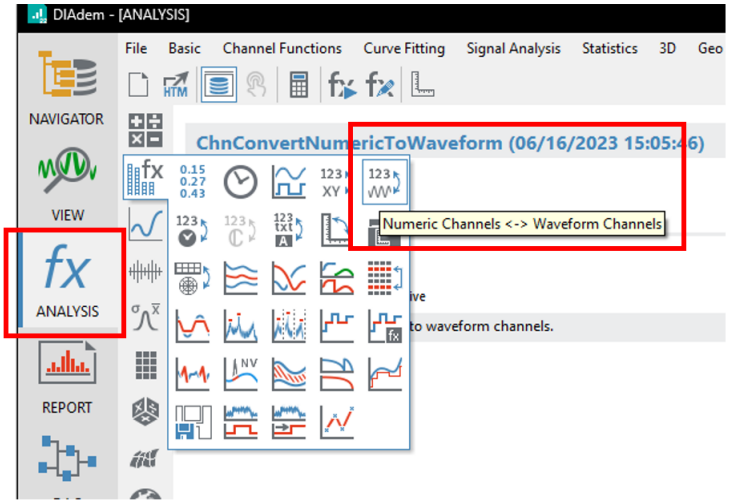


Figura 3.28: Conversión de señales

Al ingresar a esa opción, es posible seleccionar la información que se encontrara en los ejes ”X” y ”Y”. En el eje ”X” se seleccionara la señal ”Time”, la cual nos proporciona el tiempo en el que se obtuvo la señal, y en la señal ”Y” se ingresaran los datos de corriente y bandera obtenidos en la captura de información (se puede observar en la figura 3.29). Al seleccionar estos datos, es necesario colocar el botón ”OK”. Al realizar esta operación, las señales tendrán una base de tiempo en ves de muestras, esto nos ayuda a contemplar la duración de una señal.

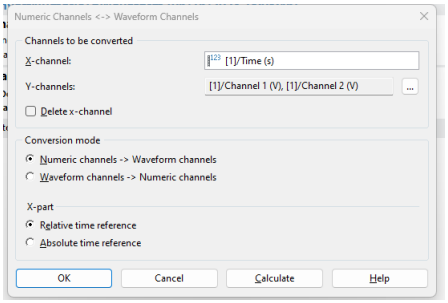


Figura 3.29: Selección de señales

Después nos pasamos a la pestaña ”View” , en esta pestaña es posible ver y manipular la base de datos obtenida, para esto es necesario seleccionar las señales convertidas con el click izquierdo, dejar presionado y arrastrarlo al centro de la ventana. Al dejarlo en el centro, se selecciona ”2D Axis System”, como se puede observar en la figura 3.30.

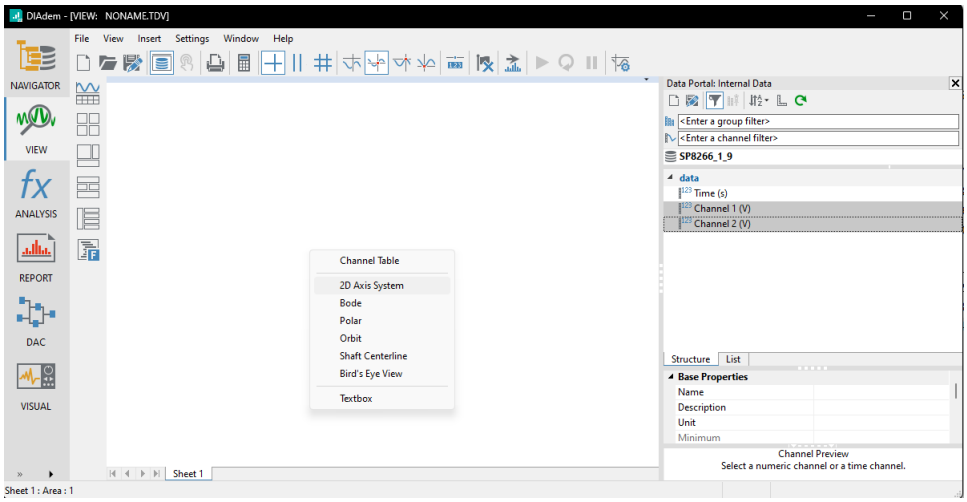


Figura 3.30: Sección - View

Las señales se pueden visualizar en la figura 3.31.

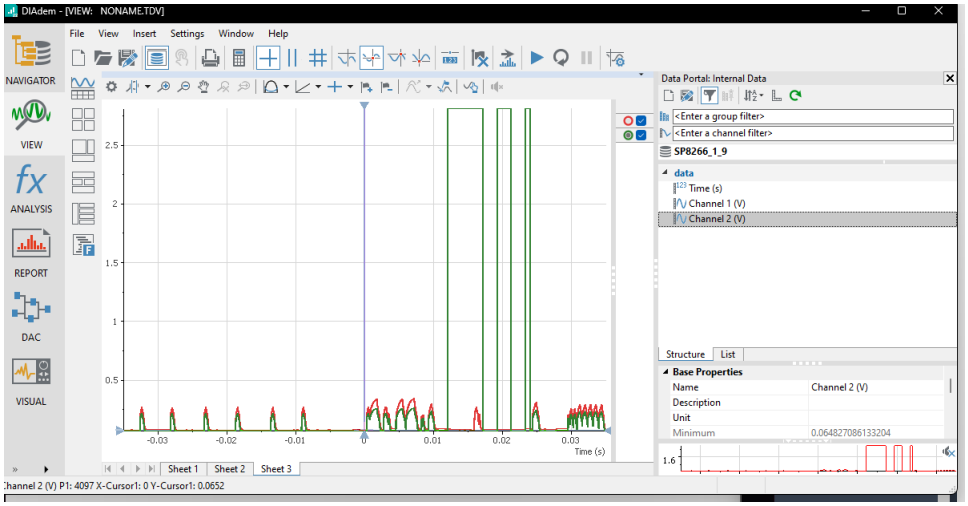


Figura 3.31: Señales graficadas

Observando estas gráficas, es necesario delimitar el intervalo de tiempo en el que se transmite un byte por medio de MQTT, para esto se marca el punto en donde inicia nuestra bandera, y se selecciona el cursor entre 2 intervalos (este cursor se puede ver en la figura 3.32) en el recuadro 1. El cuadro 2 nos permite seleccionar los puntos entre nuestros dos intervalos. El cuadro 3 traslada estos puntos a una señal aparte, en la cual podemos trabajar.

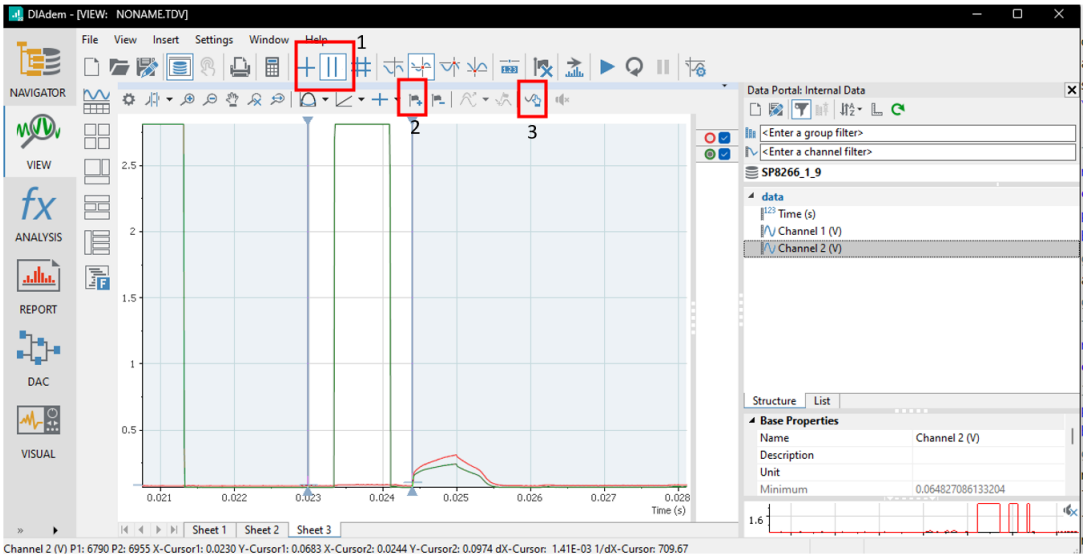


Figura 3.32: Filtrado de la señal

Ya que filtramos y seleccionamos los datos entre nuestra bandera, creamos nuestra base de datos en ”CSV”, para eso, se eliminan las señales ”Time”, ”Channel 1” y ”2”, dejando solo la señal recortada.
Después de eso, se separa esta señal en 2 canales, siendo en tiempo y muestra, en donde se requiere ir a la pestaña ”Analysis” y seleccionar la función ”Numeric channels $->$ waveforms channels” (se puede ver este proceso en la figura 3.28).
Ya entrando, se selecciona nuestro modo de conversión ”Waveform channels $->$ Numeric Channels”, y se presiona el botón ”OK”.
Después, nos dirigimos a la pestaña ”Navigator” y se selecciona la pestaña ”File” y se presiona la opción ”Save as”, en el cual es fundamental elegir ”Textfile - Export(*.csv)”.

3.4.2 GPE 1.0

Para la implementación de la metodología se desarrollo una aplicación en LabView llamada GPE 1.0, esta aplicación nos permite subir una señal de corriente con la información de voltaje utilizado, y obtener las características de esta señal, como: corriente, coulombs y Joules (energía consumida).
Al iniciar la aplicación se puede ver una pestaña igual que en la figura 3.33, en esta se requiere ingresar el CSV obtenido en los procesos anteriores.

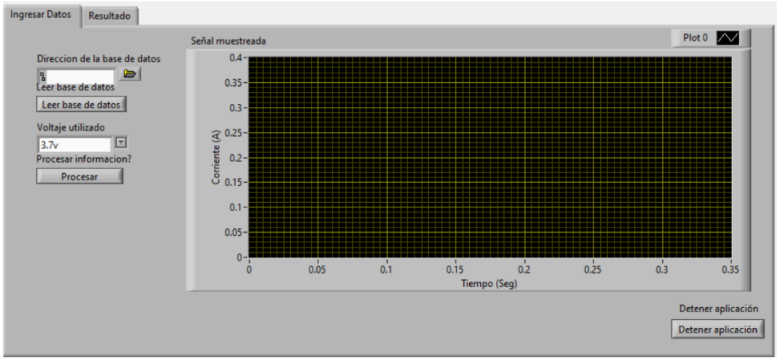


Figura 3.33: GPE 1.0

Para ingresar nuestro ÇSV”, se coloca su ubicación en la casilla de dirección mostrada en la figura 3.34

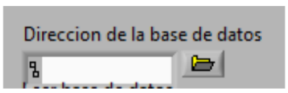


Figura 3.34: Cuadro de dirección

Al seleccionar nuestra base de datos adecuada, es necesario presionar el botón ”Leer base de datos”, al hacer esto se realiza lectura de nuestro archivo ”CSV”. Al presionar el botón aparecerá el mensaje ”¿Iniciar tiempo en cero?”, este mensaje nos permite seleccionar el tiempo en 0 segundos, en caso de no ser necesario, presionar ”No”, este mensaje puede ser visto en la figura 3.35.

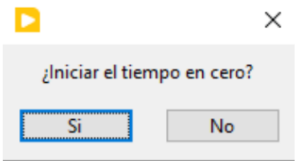


Figura 3.35: Iniciar el tiempo de la señal adquirida en ”0”

Al realizar el paso anterior, se puede observar el resultado en la figura 3.36, en donde se ve la corriente obtenida con una duración aproximada de 300 mSeg.
Al terminar el proceso de la lectura del ÇSV”se habilitará la opción de voltaje utilizado, en este apartado se seleccionará el voltaje usado. Existen tres opciones elegibles, 3.7v, 5v y 9v. Al seleccionarlo, se da clic en el botón ”Procesar” y sera redirigido a la pestaña de resultado.

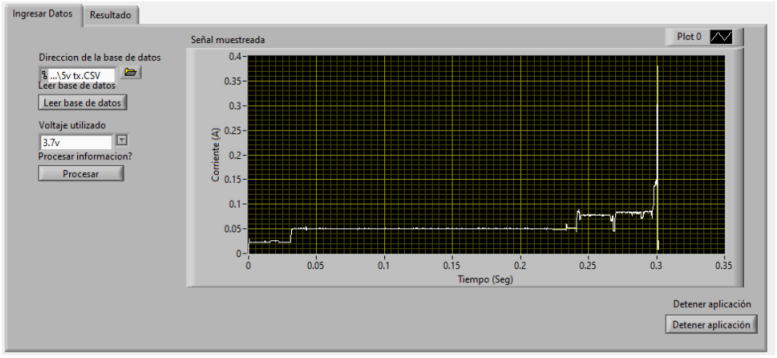


Figura 3.36: Señal adquirida iniciando en tiempo 0

En la pestaña que se puede apreciar en la figura 3.37, se obtienen los métricos de la señal que son el máximo, mínimo y promedio de la corriente, energía (Joules)

y de la carga (Coulombs), al igual que se puede observar el comportamiento de la carga con respecto al tiempo. Esta pestaña en la parte inferior nos permite generar un reporte en word con esta información, para esto primero se requiere ingresar el titulo del documento, nombre de la muestra, y la ubicación en donde sera guardado el archivo. Esto se puede observar en la figura 3.38.

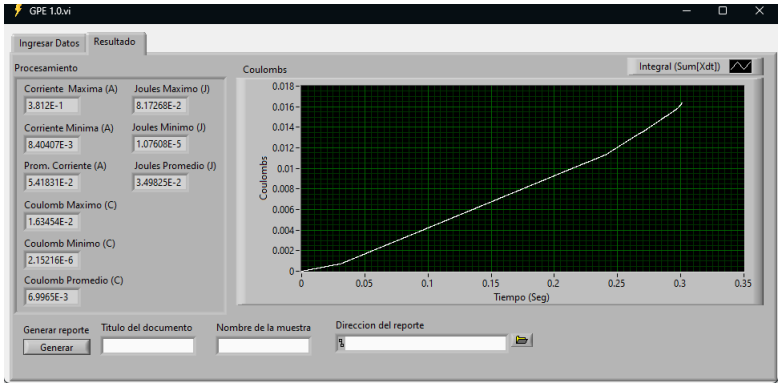


Figura 3.37: Pestaña ”Resultados”

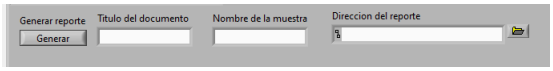
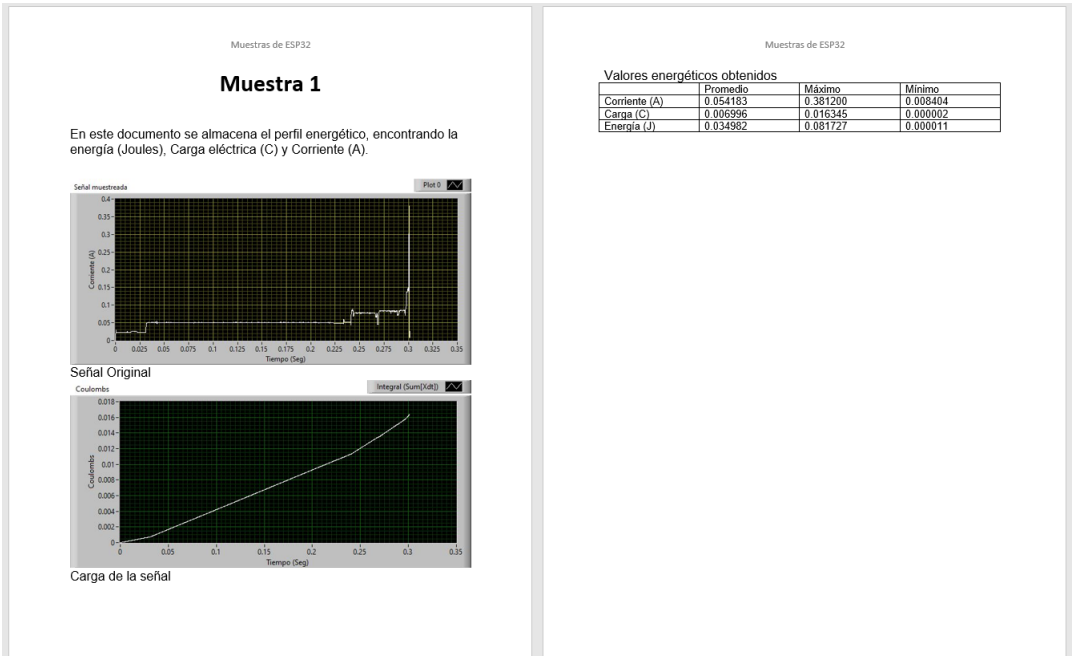


Figura 3.38: Cuadros de información del documento

Este documento puede ser convertido en PDF o manipulado para su externa aplicación o análisis, un ejemplo de este reporte puede verse en la figura 3.39, en el cual se grafican la señal original, la carga y tabla con valores energéticos obtenidos por la aplicación.



Muestras de ESP32

Muestra 1

En este documento se almacena el perfil energético, encontrando la energía (Joules), Carga eléctrica (C) y Corriente (A).

Señal Original

Carga de la señal

Muestras de ESP32

Valores energéticos obtenidos

	Promedio	Máximo	Mínimo
Corriente (A)	0.054183	0.381200	0.008404
Carga (C)	0.006996	0.016345	0.000002
Energía (J)	0.034982	0.081727	0.000011

Figura 3.39: Reporte generado

CAPÍTULO 4

RESULTADOS Y DISCUSIONES

Capítulo 4

Resultados y discusiones

En este capítulo se observarán los resultados obtenidos en las tres placas de desarrollo IoT utilizando las herramientas vistas en el capitulo anterior. Las 3 placas de desarrollo fueron medidas con 3 voltajes distintos, los cuales fueron 3.7v, 5, y 9v. Sus resultados fueron administrados y documentados.

4.1. Corriente

Se implemento la metodología vista en la sección 3 en la figura 3.19. En donde se obtuvieron las siguiente señales al transmitir un dato por MQTT.
Las placas de desarrollo ESP32 y ESP8266 entran a la metodología planteada, ya que estas permiten realizar un deep sleep por un tiempo de 10 mSeg para poder tomar muestra de la corriente consumida al transmitir por MQTT. Como se puede observar en la figura 4.1 se ve la corriente consumida al mandar un dato por MQTT con diferentes fuentes de polarización. La duración de un mensaje por MQTT es un aproximado de 1.25 mSeg, en donde la señal con menor duración fue de 0.75 mSeg, y esto fue con el microcontrolador ESP32 con una fuente de polarización de 9v.

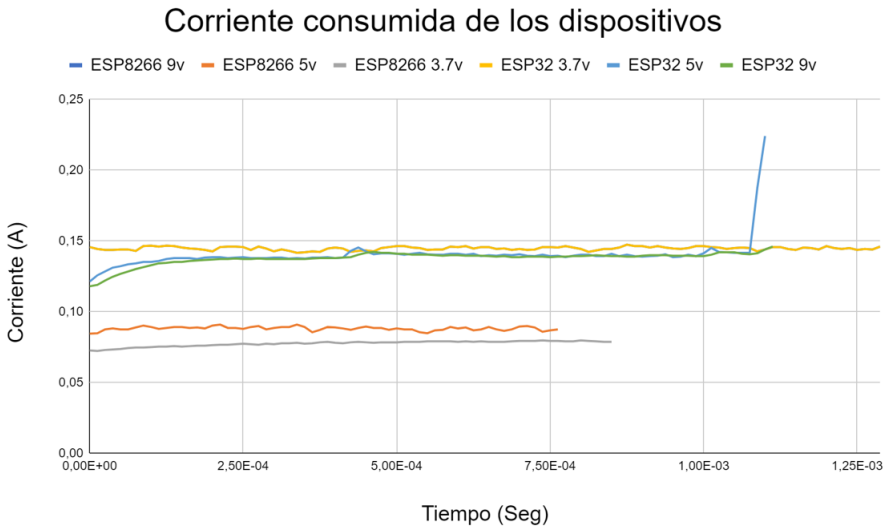


Figura 4.1: Corriente consumida

Se puede observar en la figura 4.2 la señal completa antes, durante y después de transmitir un dato por MQTT. Se realizaron 3 pulsos de disparo o de sincronización, esto se hizo para facilitar la captura en nuestro sistema de adquisición de datos. La duración del tercer pulso es el momento en el que se transmite un dato por MQTT, después de este pulso, se da un pulso bajo y entra en modo deep sleep, en el que el microcontrolador se reinicia y vuelve a iniciar la rutina, como se vio anteriormente en el capitulo 3.

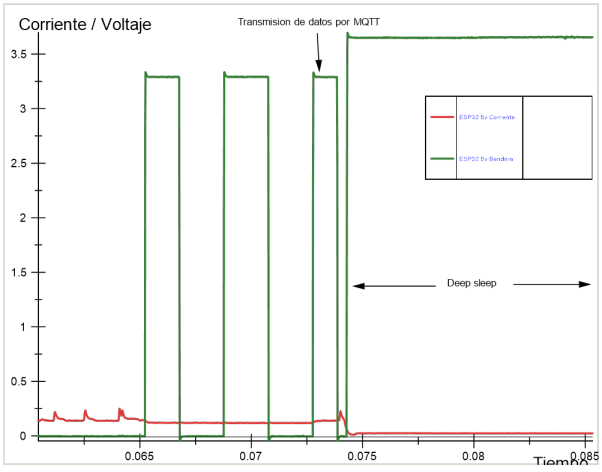


Figura 4.2: Corriente y banderas ESP32 alimentado a 5v

El Arduino uno fue evaluado un poco diferente, ya que este no cuenta con una

función desarrollada del deep sleep por el fabricante, por lo que se implemento un reinicio forzoso utilizando un pin externo para el reinicio del dispositivo, en la figura 4.3, se puede observar que el tiempo que necesita para transmitir un dato por MQTT es un aproximado de 2.5 mSeg, y este tiene una corriente promedio de 120 mA.

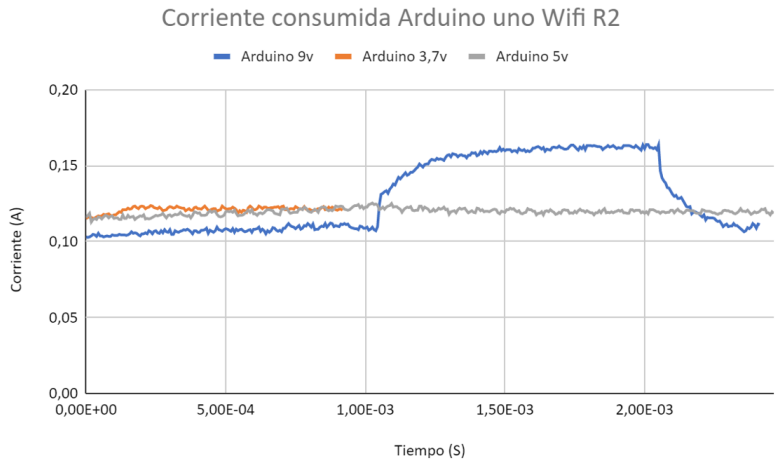


Figura 4.3: Corriente consumida por Arduino uno Wifi

Los resultados de todas las señales las podemos resumir en la tabla 4.1.

Tabla 4.1: Corrientes obtenidas por placa de desarrollo

Corriente				
Tarjeta	Voltaje	Corriente promedio	Máximo	Mínimo
ESP32	3.7v	138.272 mA	144.273 mA	122.98 mA
	5v	140.225 mA	224.037 mA	121.29 mA
	9v	137.716 mA	154.963 mA	117.911 mA
ESP8266	3.7v	77.505 mA	79.719 mA	72.283 mA
	5v	77.328 mA	79.719 mA	72.621 mA
	9v	144.683 mA	147.315 mA	141.569 mA
Arduino uno wifi	3.7v	119.585 mA	125.346 mA	112.841 mA
	5v	121.078 mA	123.656 mA	114.869 mA
	9v	129.613 mA	163.876 mA	102.701 mA

4.2. Carga (Coulomb)

Al obtener las señales de corriente, es posible calcular la carga eléctrica, este parámetro nos permite realizar una estimación de autonomía que puede tener el dispositivo. Para hacer esto es necesario realizar una integración de la señal de corriente, y con eso conseguimos la cantidad de carga.
Al tener la carga es posible estimar la cantidad de mensajes posibles a transmitir por medio de MQTT, para esto aplicamos la ecuación 4.1.

$$\frac{mAh_{fuente} * 3.6}{Coulomb_{Carga}} = \text{Cantidad de mensajes} \tag{4.1}$$

Para calcular la autonomía de la placa de desarrollo utilizamos la ecuación 4.2, que nos ayuda a darnos una estimación de cuanto tiempo puede esta transmitir información.

$$\frac{mAh}{mA} = hrs \tag{4.2}$$

Se realizó el calculo para cada señal adquirida de las placas de desarrollo, y se obtuvo la tabla 4.2.

Tabla 4.2: Carga obtenida por placa de desarrollo

Carga				
Tarjeta	Voltaje	Carga promedio	Carga maxima	Carga minima
ESP32	3.7v	4.304 mC	4.382 mC	4.23 mC
	5v	2.737 mC	2.816 mC	2.661 mC
	9v	3.838 mC	3.916 mC	3.762 mC
ESP8266	3.7v	0.033 mC	0.067 mC	0.001 mC
	5v	0.249 mC	0.275 mC	0.224 mC
	9v	0.478 mC	0.524 mC	0.431 mC
Arduino uno wifi	3.7v	2.887 mC	3.035 mC	2.74 mC
	5v	4.5 mC	4.611 mC	4.389 mC
	9v	3.627 mC	3.793 mC	3.48 mC

Con los datos de la tabla 4.2, podemos proponer una batería para realizar una estimación de la autonomía y de la cantidad de mensajes que son posibles transmitir. Se propuso una batería de 500 mAh utilizando la ecuación 4.1, la batería

tendría una carga de 1800 coulombs, por lo que se utilizó esa cantidad para realizar los cálculos.

Tabla 4.3: Autonomía y mensajes estimados

Tarjeta	Batería propuesta		Corriente promedio de carga (mA)	Carga eléctrica promedio (mC)	Duración (Hrs)	Mensajes estimados
	Voltaje	mAh				
ESP32	3.7v	500	138.272	4.304	3.62	418216
	5v	500	140.225	2.737	3.57	657654
	9v	500	137.716	3.838	3.63	468994
ESP8266	3.7v	500	77.505	0.033	6.45	54545455
	5v	500	77.328	0.249	6.47	7228916
	9v	500	144.683	0.478	3.46	3765690
Arduino uno Wifi	3.7v	500	119.585	2.887	4.18	623484
	5v	500	121.078	4.5	4.12	400000
	9v	500	129.613	3.627	3.85	496277

Como se puede observar en la tabla 4.3, la autonomía varía por cada tarjeta de desarrollo, en donde la placa ESP8266 tiene una mayor autonomía, con una cantidad aproximada de 7228916 mensajes transmitidos por MQTT y una duración de 6.47 horas de transmisión constante de los datos.

4.3. Energía (Joules)

Los joules forman una parte importante en nuestros parámetros, ya que estos nos permiten conocer cuanta energía a consumido nuestra placa de desarrollo al realizar una implementación. Para esto, multiplicamos nuestra señal anteriormente procesada para obtener la carga y multiplicarla por el voltaje de polarización aplicado. Como antes mencionado, esto nos ayuda a observar cual equipo o tarjeta de desarrollo esta consumiendo mas energía al realizar una comunicación.

Tabla 4.4: Energía consumida por las placas de desarrollo

Energía				
Tarjeta	Voltaje	promedio	Máxima	Mínima
ESP32	3.7v	15.926 mJ	16.213 mJ	15.649 mJ
	5v	13.686 mJ	14.079 mJ	13.307 mJ
	9v	34.544 mJ	35.243 mJ	33.861 mJ
ESP8266	3.7v	0.124 mJ	0.247 mJ	0.003 mJ
	5v	1.247 mJ	1.377 mJ	0.112 mJ
	9v	4.3 mJ	4.719 mJ	3.881 mJ
Arduino uno wifi	3.7v	14.436 mJ	15.175 mJ	13.702
	5v	16.648 mJ	17.062 mJ	16.238 mJ
	9v	32.641 mJ	34.135 mJ	31.32 mJ

Como se pudo observar en la tabla 4.4, las placas de desarrollo al ser polarizadas con una fuente de voltaje mayor, tienden a consumir mas energía promedio. Aunque las placas de desarrollo consuman una cantidad menor de corriente, puede verse reflejado en la energía. Por ejemplo, puede ser observado que el consumo promedio de corriente del ESP32 polarizado a 9v es el menor, y aun así es el que consume una mayor cantidad de energía.

CONCLUSIONES Y TRABAJO FUTURO

Conclusiones y trabajo futuro

Algo muy importante de los dispositivos IoT es el consumo energético que estos pueden tener. No se tiene mucho conocimiento de cuanto consumen al transmitir información, y esta investigación nos ofrece la metodología para realizar un análisis de eficiencia energética. Al igual que, gracias al software desarrollado (GPE 1.0) se puede implementar este proceso de manera mas eficiente en comparación de utilizar otros softwares como Diadem, en donde se tenia que realizar el procesamiento paso por paso de manera manual.

Observando los resultados del capitulo anterior, se puede concluir que la tarjeta de desarrollo ESP8266 obtuvo el menor consumo energético promedio al ser polarizada a 5v. Si esta placa de desarrollo es comparada con el ESP32 polarizada igualmente a 5v, podemos ver una diferencia del 55.14 %, si comparamos la autonomía, podemos observar una diferencia del 55.17 %, incrementando de 3.57 hrs hasta 6.47 hrs. Esto se ve muy reflejado en la cantidad de mensajes estimados, en donde el ESP8266 casi envía 11 veces la cantidad de mensajes posibles enviados por el ESP32.

Para el trabajo futuro se propone realizar el análisis energético al transmitir un dato por MQTT encapsulado en un paquete Json, ya que este es uno muy recurrente al momento de transmitir datos por este protocolo. Al transmitir estos datos también es importante observar la cantidad de dígitos que son transmitidos, así que como segunda propuesta, se propone realizar transmisión utilizando una variable float, en donde se colocaran decimales hasta superar la cantidad posible. Con esto se espera conocer como se comporta la placa de desarrollo al procesar la información, y si la cantidad de dígitos transmitidos afecta de manera significativa el consumo energético.

Bibliografía

[1] G. Ortiz, "México rezagado en internet de las cosas: D.noticias," https://www2.deloitte.com/mx/es/pages/dnoticias/articles/internet-de-las-cosas-en-mexico.html, 09 2017, accedio en julio 2023.

[2] MICROCHIP, "Power monitor," 2021. [Online]. Disponible en: https://microchipdeveloper.com/realice:power-monitor

[3] Oracle, *¿Que es IoT?* Oracle, 2016.

[4] M. Hasan. (2022) State of iot 2022: Number of connected iot devices growing 18 % to 14.4 billion globally. [Online]. Disponible en: https://iot-analytics.com/number-connected-iot-devices/

[5] T. E. grouo Limited. (2010) Augmented business. [Online]. Disponible en: https://www.economist.com/special-report/2010/11/06/augmented-business

[6] A. Jamin *et al.*, "An aggregation plateform for iot-based healthcare: illustration for bioimpedancemetry, temperature and fatigue level monitoring," in *Internet of Things Technologies for HealthCare: Third International Conference, HealthyIoT 2016, Västerås, Sweden, October 18-19, 2016, Revised Selected Papers 3*. Springer, 2016, pp. 125–130.

[7] A. S. M. W. A. Y. H. A. Rushan, Z. Saman, "Green iot: An investigation on energy saving practices for 2020 and beyond," 2020.

[8] R. G. Matti Eteläperä, Massino Vecchio, "Improving energy efficiency in iot with re-configurable virtual objects," 2014.

[9] G. Tolosa, "Protocolos y modelo osi," *Obtenido de http://www. tyr. unlu. edu. ar/pub/02-ProtocolosOSI. pdf*, 2014.

[10] S. Friedl *et al.*, "Transport layer security (tls) application-layer protocol negotiation extension," Tech. Rep., 2014.

[11] J. Roldán-Gómez *et al.*, "Security analysis of the mqtt-sn protocol for the internet of things," *Applied Sciences*, vol. 12, no. 21, 2022. [Online]. Disponible en: https://www.mdpi.com/2076-3417/12/21/10991

[12] D. Garcia-Carrillo y R. Marin-Lopez, "Multihop bootstrapping with EAP through CoAP intermediaries for IoT," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 4003–4017, oct 2018.

[13] B. Belvedere *et al.*, "A microcontroller-based power management system for standalone microgrids with hybrid power supply," *IEEE Trans. Sustain. Energy*, vol. 3, no. 3, pp. 422–431, jul 2012.

[14] http://www.designnews.com/lecture-calendar.asp?p_l_ed=CEC_Semester_Three_2013#lecture_track_cgid_168., accessed: 2023-4-21.

[15] G. Luo *et al.*, "Analysis and optimization of embedded software energy consumption on the source code and algorithm level," in *2009 Fourth International Conference on Embedded and Multimedia Computing*. IEEE, dec 2009.

[16] ——, "Analysis and optimization of embedded software energy consumption on the source code and algorithm level," in *2009 Fourth International Conference on Embedded and Multimedia Computing*. IEEE, dec 2009.

[17] http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=AC244008., accessed: 2023-4-21.

[18] http://www.ti.com/tool/energytrace., accessed: 2023-4-21.

[19] A. Holberg, *ATMEL*, 2006.

[20] B. Ivey, *Microchip*, 2011.

[21] I. Tsekoura *et al.*, "An evaluation of energy efficient microcontrollers," in *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. IEEE, 2014, pp. 1–5.

[22] R. Richey, "Low power design using picmicro™ microcontrollers," *AN606, M icrochip Technology Inc*, 1997.

[23] http://www.techonline.com/electrical-engineers/education-training/webinars/4420344/Implementing-Ultra-low-Power-Design-Techniques-for-High-performance-32-bit-MCU accessed: 2023-4-21.

[24] A. Medina-Duran *et al.*, “Medición de energía en sistemas embebidos para el internet de las cosas,” 2020.

[25] V. Tiwari, S. Malik, y A. Wolfe, “Power analysis of embedded software: A first step towards software power minimization,” in *IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2005.

[26] ——, “Compilation techniques for low energy: an overview,” in *Proceedings of 1994 IEEE Symposium on Low Power Electronics*. IEEE, 2002.

[27] C. Robertson y C. J. Martinez, “Analyzing the software aspect of an embedded system’s power consumption,” in *2011 24th Canadian Conference on Electrical and Computer Engineering(CCECE)*. IEEE, may 2011.

[28] D. A. Ortiz y N. G. Santiago, “Impact of source code optimizations on power consumption of embedded systems,” in *2008 Joint 6th International IEEE Northeast Workshop on Circuits and Systems and TAISA Conference*. IEEE, jun 2008.

[29] M. E. A. Ibrahim, M. Rupp, y S. E.-D. Habib, “Compiler-based optimizations impact on embedded software power consumption,” in *2009 Joint IEEE North-East Workshop on Circuits and Systems and TAISA Conference*. IEEE, jun 2009.

[30] J. Liao *et al.*, “Studies of rogowski coil current transducer for low amplitude current (100a) measurement,” *IEEE CCECE*, vol. I, pp. 463–466, 2003.

[31] “Chapter 11: The current mirror [analog devices wiki],” https://wiki.analog.com/university/courses/electronics/text/chapter-11., accessed: 2023-4-21.

[32] A. Borovyi *et al.*, “Analysis of circuits for measurement of energy of processing units,” in *2007 4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*. IEEE, sep 2007.

[33] V. Konstantakos *et al.*, “Energy consumption estimation in embedded systems,” *IEEE Trans. Instrum. Meas.*, vol. 57, no. 4, pp. 797–804, apr 2008.

[34] V. Konstantakos y T. Laopoulos, “A power measuring technique for built-in test purposes,” in *2006 IEEE Instrumentation and Measurement Technology Conference Proceedings*. IEEE, apr 2006.

[35] V. Konstantakos *et al.*, “Measurement of power consumption in digital systems,” *IEEE Trans. Instrum. Meas.*, vol. 55, no. 5, pp. 1662–1670, oct 2006.

[36] E. Ramsden, “Hall-effect sensors: theory and applications, newnes,” 2006.

[37] C. Cirstea, M. Cernaianu, y A. Gontean, “An inductive system for measuring microampere currents,” in *2012 IEEE 18th International Symposium for Design and Technology in Electronic Packaging (SIITME)*. IEEE, oct 2012.

[38] “Current sensing circuit concepts and fundamentals AN1332,” *Available*, 2011.

[39] M. Bazzaz, M. Salehi, y A. Ejlali, “An accurate instruction-level energy estimation model and tool for embedded systems,” *IEEE Trans. Instrum. Meas.*, vol. 62, no. 7, pp. 1927–1934, jul 2013.

[40] “No title,” http://www.tequipment.net/TektronixTCP312.html?gclid=Cj0KEQjwztG8BRCJgseTvZLctr8BEiQAA_kBD7Zf1VqE-Ms0oFcwfs5eEoKzdRe_F_BZVaW0T8agDbwaAiKM8P8HAQ, accessed: 2023-4-21.

[41] L. M. Surhone, M. T. Tennoe, y S. F. Henssonow, Eds., *Embc*. Betascript Publishing, aug 2010.

[42] http://www.cmicrotek.com/uCP.htm., accessed: 2023-4-21.

[43] R. A. Light, “Mosquitto: server and client implementation of the mqtt protocol,” *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017. [Online]. Disponible en: https://doi.org/10.21105/joss.00265

[44] N. Instrument”. (2023) ”ni tdms file format - what is a tdms file?”. [Online]. Disponible en: ”https://www.ni.com/es-mx/support/documentation/supplemental/06/the-ni-tdms-file-format.html”

20% de similitud general

Principales fuentes encontradas en las siguientes bases de datos:

- 19% Base de datos de Internet
- Base de datos de Crossref
- 12% Base de datos de trabajos entregados
- 1% Base de datos de publicaciones
- Base de datos de contenido publicado de Crossr

FUENTES PRINCIPALES

Las fuentes con el mayor número de coincidencias dentro de la entrega. Las fuentes superpuestas no se mostrarán.

1	repositorioinstitucional.uabc.mx Internet	5%
2	luisllamas.es Internet	2%
3	profesores.elo.utfsm.cl Internet	2%
4	repositorio.unp.edu.pe Internet	2%
5	repositorio.upct.es Internet	1%
6	Universidad Autónoma de Ciudad Juárez on 2023-07-27 Submitted works	1%
7	iydt.files.wordpress.com Internet	<1%
8	iot-lab.dk Internet	<1%

9	es.wikipedia.org Internet	<1%
10	'Shota Rustaveli' State University on 2021-07-03 Submitted works	<1%
11	todosloshechos.es Internet	<1%
12	repositorio.umsa.bo Internet	<1%
13	sistemasgeneral.blogspot.com Internet	<1%
14	hdl.handle.net Internet	<1%
15	clubensayos.com Internet	<1%
16	Universidad Carlos III de Madrid on 2020-09-07 Submitted works	<1%
17	Universidad Autónoma de Nuevo León on 2016-09-13 Submitted works	<1%
18	ricveal.com Internet	<1%
19	tesis.ipn.mx Internet	<1%
20	Universidad Politecnica Salesiana del Ecuado on 2021-09-05 Submitted works	<1%

21	Universidad Tecnológica Israel on 2023-06-19 Submitted works	<1%
22	Universidad de Oviedo on 2022-10-27 Submitted works	<1%
23	Escuela Superior Politécnica del Litoral on 2023-09-22 Submitted works	<1%
24	Universidad Autónoma de Ciudad Juárez on 2023-07-27 Submitted works	<1%
25	de.slideshare.net Internet	<1%
26	Pontificia Universidad Catolica del Peru on 2009-03-24 Submitted works	<1%
27	Systems Link on 2013-02-25 Submitted works	<1%
28	Universidad Politécnica de Madrid on 2023-09-15 Submitted works	<1%
29	Universidad de León on 2021-03-10 Submitted works	<1%
30	cdigital.uv.mx Internet	<1%
31	riunet.upv.es Internet	<1%
32	Glasgow Caledonian University on 2019-07-31 Submitted works	<1%

33	Stefan cel Mare University of Suceava on 2023-09-06 Submitted works	<1%
34	Universidad Carlos III de Madrid - EUR on 2023-09-17 Submitted works	<1%
35	gomix.homelinux.net Internet	<1%
36	repositorio.espe.edu.ec Internet	<1%
37	slovakiahoster.com Internet	<1%
38	tesis.ucsm.edu.pe Internet	<1%
39	Universidad Carlos III de Madrid - EUR on 2023-09-04 Submitted works	<1%
40	bleepcoder.com Internet	<1%
41	forum.arduino.cc Internet	<1%
42	moam.info Internet	<1%
43	coursehero.com Internet	<1%
44	dspace.espol.edu.ec Internet	<1%

45	labc.usb.ve Internet	<1%
46	revistamasseguridad.com.mx Internet	<1%
47	scribd.com Internet	<1%
48	slideshare.net Internet	<1%