UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

CONFIGURACIÓN REMOTA DE COLECTORES DE DATA CON TECNOLOGÍA ZIGBEE DESDE UNA INTERFAZ GRÁFICA WEB

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

FELIPE ALFREDO RAQUIL PIZARRO

PROFESOR GUÍA:
LEONARDO CAMENT RIVEROS

MIEMBROS DE LA COMISIÓN:
SANDRA CÉSPEDES UMAÑA
ANDRÉS CABA RUTTE

SANTIAGO DE CHILE
2020

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: FELIPE ALFREDO RAQUIL PIZARRO
FECHA: 2020
PROF. GUÍA: LEONARDO CAMENT RIVEROS

CONFIGURACIÓN REMOTA DE COLECTORES DE DATA CON TECNOLOGÍA ZIGBEE DESDE UNA INTERFAZ GRÁFICA WEB

En una planta industrial, el proceso de recolección de datos posee diversas dificultades debido a que es manual, haciéndolo susceptible a errores humanos y a demoras en tiempo considerables, por lo que es necesaria una alternativa a este proceso de recolección que elimine el factor humano de error, monitoree los colectores de datos de la planta industrial a tiempo real y ejecute rápidamente los cambios en caso que éstos sean necesarios. Así, la información capturada puede ser utilizada con mayor confianza para mejorar el desempeño del proceso que se esté midiendo.

El presente trabajo muestra la implementación de un sistema de configuración remota para el monitoreo de estado de colectores de data, dentro de una red inalámbrica con tecnología Zigbee. El sistema de configuración consiste en una interfaz web a través de la cual se realizan operaciones con parámetros propios de los colectores de data.

La metodología utilizada consiste, por un lado, en la propuesta e implementación de una arquitectura de *software* que permita la configuración y monitoreo de los distintos dispositivos y por otro lado, en la definición del contexto del sistema desarrollado, a nivel del estado del sistema y de las situaciones en que éste opere, llamadas casos de uso. La arquitectura de *software* es validada a través de una versión a escala de una planta industrial consistente en tres colectores de data, obteniendo los tiempos en que se ejecutan las peticiones desde la interfaz. Los tiempos de operación medidos poseen un costo económico asociado según el caso de uso, como también los tiempos estimados en la situación base, en que las tareas automatizadas en el sistema de configuración son realizadas por un ser humano. Para cada caso de uso, se comparan los costos económicos entre las dos situaciones, analizando así las mejoras dadas por el sistema desarrollado.

Los resultados obtenidos indican que la reducción de costos dada por la existencia de este sistema de configuración para contextos de cambios masivos de parámetros se encuentra entre un 75 % y un 94 %, dependiendo del caso de uso que se está considerando, es decir, se obtiene una mejora significativa.

Agradecimientos

Primero, quiero agradecer a quienes conforman la comisión de este trabajo: Leonardo Cament, por despertar mi interés y aceptarme como parte de este proyecto, considerando que no tuve experiencia con telecomunicaciones ni automatización. A Andrés Caba, por mostrar disposición a dar aportes respecto a cómo presentar la información. Finalmente, a Sandra Céspedes, por el entusiasmo y rigor brindados al desarrollo de esta memoria.

Deseo también agradecer a mi familia, porque pese a todas las diferencias en cuanto a visión de vida y en muchas ocasiones, la falta de entendimiento mutuo, supieron ser un soporte dentro de mis años de formación y búsqueda. Sinceramente, espero comprenderlos más a lo largo del avance de mi existencia.

Al difunto y mítico 1212, por ser el lugar donde he encontrado a personas con las que puedo recorrer mi camino y que ha sido escenario de algunas de las experiencias más intensas estos últimos años. A quienes se han vuelto mis cercanos, admito que no puedo pagar toda la ayuda recibida por ustedes, pero el futuro me hará encontrar una forma.

A la importancia que ha tenido la música a lo largo de estos años, siempre sonando en algún lado mientras trabajo y siempre acompañando desde los momentos de mayor alegría hasta los de mayor melancolía, ocupando un lugar que ningún otro ser humano ni pasatiempo podrá ocupar jamás.

Finalmente a nadie en especial, quiero decir, a todo ese set de casualidades y de causalidades llamado realidad. Pese a que la incesante búsqueda por lo no-obvio, que incluye a Ingeniería Eléctrica me ha dejado un montón de frustraciones, he entendido que ellas se desvanecen y lo que queda finalmente es una sensación agradable de acierto y con esto, mayores ganas de seguir buscando lo no-obvio.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto

La industria 4.0 permite la existencia de fábricas inteligentes con el uso de Internet de las cosas (IoT), que envía datos por medio de sensores a una nube en que se realiza un procesamiento con tal de ejecutar las acciones requeridas del sistema. Se diagnostica en [9] que alrededor de 75 billones de dispositivos estarán conectados a IoT en 2025.

La eficiencia productiva en plantas industriales, una de las ramas de la industria 4.0, es de gran importancia para sus gestores, ya que una monitorización inadecuada implica un perjuicio a nivel económico y/o de recursos, generando así pérdidas que pueden ser considerables para la empresa.

Neykos es una empresa encargada de brindar servicios a otras empresas en entornos industriales por medio de la integración de tecnologías en diversas tareas como la transformación digital, automatización de procesos, visualización de datos y la que más concierne a este trabajo: maximización de eficiencia. La herramienta utilizada para la captura de la información que permita este manejo de eficiencia es llamada PMSystem, que permite que las empresas lleven un control de la producción en tiempo real por medio de medición y seguimiento de los procesos productivos.

PMSystem (*Plant Monitoring System*) se encarga de la captura automatizada de la información y la monitorización remota de las líneas de producción, pudiendo capturar datos generados por instrumentos que permiten el control de calidad de los productos y también enviando datos a la nube para análisis que permitan la mejora de la productividad, reducción de costos de mantenimiento y optimización de las operaciones de la fábrica. Los dispositivos encargados de recopilar la información dentro de PMSystem se denominan *Data Collectors*, que tal como puede verse en la figura 1.1, envían la información a concentradores denominados *Gateways*, que a su vez la envían a un servidor donde se procesan y arman reportes, análisis estadísticos, visualización de estados de funcionamiento, etc.

Resumiendo, los *Data Collectors* con su rol de generadores de la información, son cruciales para asegurar el buen funcionamiento de los procesos, a tal punto que si alguno no funciona

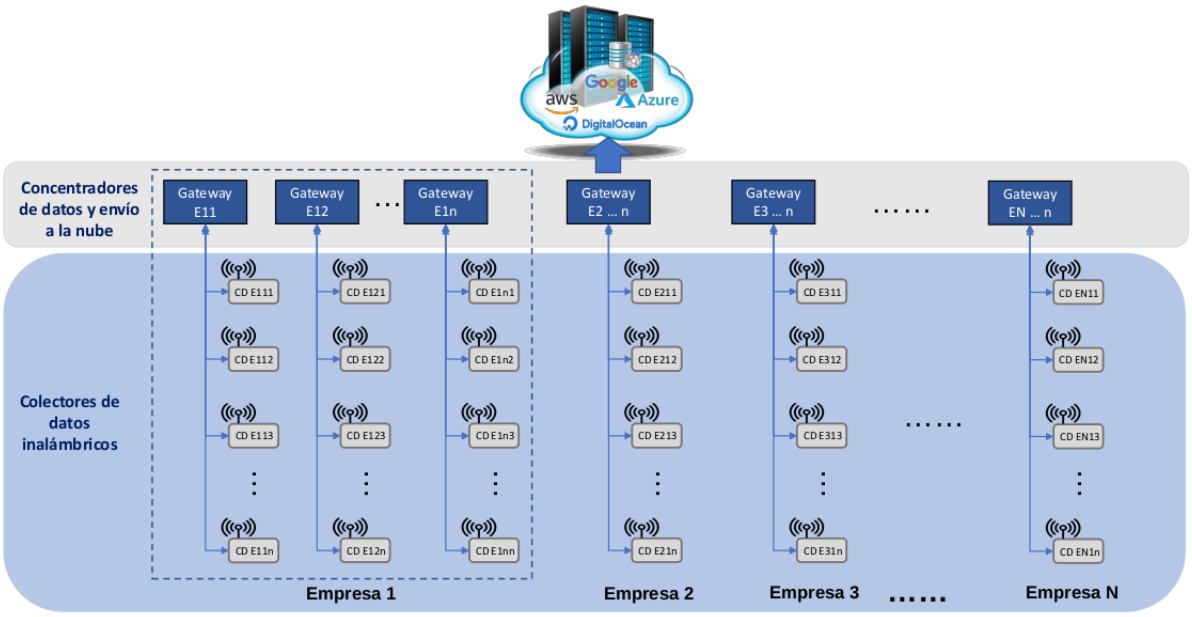


Figura 1.1: Diagrama básico de Dispositivos de Monitorización, usando la herramienta PMSystem

de forma correcta, es posible que la producción planificada no pueda cumplirse porque los datos a través de los cuales se juzga no son confiables. Al ser los *Data Collector* altamente configurables, un funcionamiento incorrecto no significa que el dispositivo no esté enviando mediciones a la nube o que se encuentre apagado, basta que sus parámetros de configuración se encuentren mal establecidos.

1.2. Descripción general del problema

Una planta industrial suele seguir la lógica de la figura 1.2: en una planta, hay información recolectable, dada por los 3 insumos (trabajo, materias primas y energía), generándose productos y desechos. Se busca controlar eficientemente la información, ya que mucho es actualmente recopilado a mano (no automatizado) y esto implica datos imprecisos, desfasados, ineficientes, etc. La recopilación no automatizada significa incrustaciones de datos directas en el código fuente de los archivos de configuración para los dispositivos de medición. Esta solución es ineficiente en tiempo y alta en errores de tipo humano (ineficaz), por lo que se desea una nueva alternativa que permita que cualquier técnico o miembro del equipo de Neykos sea capaz de configurar los *Data Collectors* de forma intuitiva, simple y masiva, semi-automatizando de esta manera el proceso de configuración y aumentando la eficacia y eficiencia de implementación. El proceso de automatización implica que la planta ya no debe parar para realizar los cambios, por lo que elimina la invasividad de PMSystem sobre los clientes a los que presta servicios.

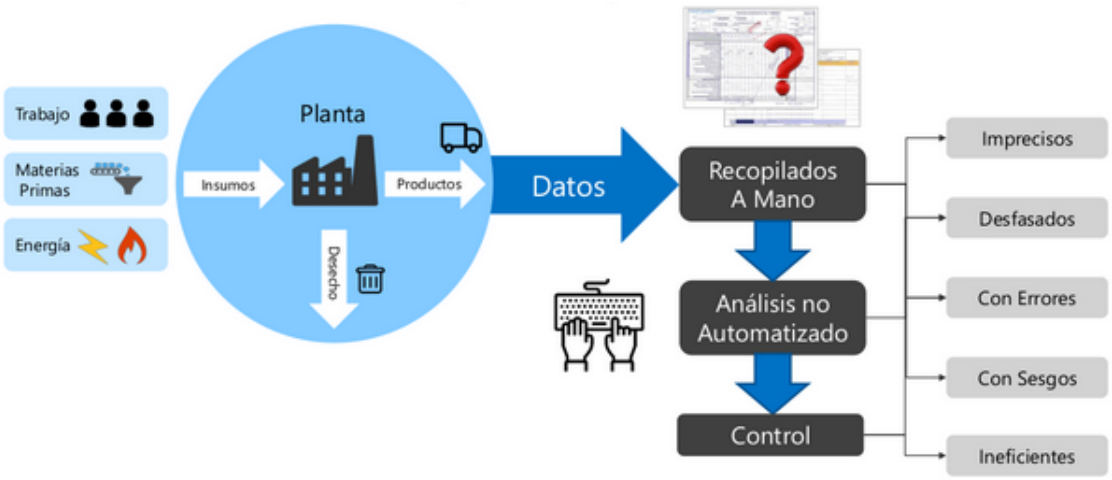


Figura 1.2: Deficiencias en la recopilación actual de los datos por PMSystem

1.3. Objetivos

El objetivo general dentro del marco académico es analizar las mejoras de implementación dadas por el monitoreo de estado de colectores de data en una planta, mientras que en lo que se refiere a la empresa Neykos, el objetivo es el desarrollo del sistema de monitoreo en sí. El propósito desde ambos marcos es el de mostrar la pertinencia del sistema de configuración desarrollado por medio de una interfaz, derivada de la actual incapacidad por parte de un técnico para observar el estado de dispositivos en tiempo real y ejecutar los cambios correspondientes en caso que sea necesario. La evaluación se realizará a través de un análisis económico relacionado a la disminución de los errores por tres variables:

- La forma actual de trabajo, que contempla la ejecución de comandos de unix y el cambio de parámetros en archivos de configuración.
- Los tiempos de reacción menores debido al monitoreo in-situ.
- La capacidad de personal no entrenado en programación para ejecutar las labores de configuración y monitoreo correctamente.

Los objetivos secundarios de este trabajo son:

- Definir la relación entre costos y las mejoras de eficiencia/eficacia del monitoreo dados por esta implementación, por medio de búsqueda bibliográfica y conocimiento de aspectos más específicos del contexto de funcionamiento de la planta, con el propósito de establecer de forma clara los parámetros que permitan medir la mejora económica.
- Definir claramente la lógica de utilización de la implementación completa, con el objeto de dar un contexto acerca de cuándo debiese haber monitoreo y cuándo configuración de uno o más *Data Collectors*, por medio de la definición del estado del sistema y los rangos correctos e incorrectos de sus variables, junto a las acciones a realizar en cada evento de estado incorrecto, todo basado en información de fábrica de los dispositivos involucrados.
- Proponer e implementar una arquitectura de software integrada en los dispositivos utilizados, con tal de permitir las funcionalidades deseadas (configuración y monitoreo), lo

que se hará por medio de programación remota, aprovechando las facilidades asociadas a los dispositivos utilizados y haciendo uso de un servidor a modo de usuario.

- Validar y analizar la implementación en una versión a escala de la planta, con el propósito de obtener las variables de tiempo en forma experimental, a través de la disposición de nodos con su arquitectura junto a la programación de las unidades colectoras de data, el *router* y servidor.

1.4. Estructura del Documento

La estructura de este documento es la siguiente: En el capítulo 2 se introducen conceptos importantes en el desarrollo de este trabajo y se muestran avances respecto a la estandarización de colectores de data y estado del arte de uso de Zigbee en entornos industriales. En el capítulo 3 se describe la metodología de desarrollo, destacando la arquitectura de la solución presentada y el modo de operación de los dispositivos utilizados. En el capítulo 4, se reportan y analizan los resultados obtenidos. Finalmente, el capítulo 5 presenta las conclusiones y las posibles proyecciones de este trabajo de memoria.

Capítulo 2

Marco teórico y estado del arte

2.1. Marco teórico

En este apartado, se describe la industria 4.0 en la que este proyecto está inserto para entrar a describir los dos componentes importantes del sistema analizado: *Data Collector* y *Gateway*, para luego profundizar en la tecnología Zigbee que da nombre a la red en que operan los dispositivos y dos aspectos principales: Por el lado del *Gateway*, Formas de comunicación intraprocesos, es decir, entre procesos corriendo en un mismo procesador y por el lado del *Data Collector*, el Firmware, que permite la operación del microcontrolador.

2.1.1. Industria 4.0

La industria 4.0 se define según [10] como una nueva fase en la revolución industrial que se enfoca fuertemente en la interconectividad, automatización, *Machine Learning*, datos en tiempo real, entre otras, abarcando todas las aplicaciones enunciadas en 2.1. La industria 4.0 encapsula IIoT (internet de las cosas de tipo industrial) y manufactura inteligente, creando de esta forma un ecosistema más holístico y mejor conectado para empresas enfocadas en manufactura y manejo de cadenas de producción.

Como concepto distinto al de industria 4.0, vale destacar el de Internet de las cosas, que se refiere a las conexiones entre objetos físicos como sensores y máquinas e Internet. Más específicamente hablando, en el ámbito industrial se refiere a conectar objetos físicos, internet y personas dentro de un contexto de producción industrial.

2.1.2. Componentes importantes: *Data Collector* y *Gateway*

El concepto de *Data Collector* es de tipo comercial para la empresa Neykos y se basa en el de *Data Logger*, un dispositivo electrónico que se utiliza para adquirir y guardar datos a través de sensores conectados físicamente a él o se trata de sistemas de comunicación serial que utilizan algún computador para realizar las mediciones [11]. La diferencia se encuentra en que un *Data Collector* tiene baja capacidad de cómputo y de almacenamiento, por lo cual sólo puede recolectar datos, enviándolos a algún servidor que los guarde, analice, muestre o

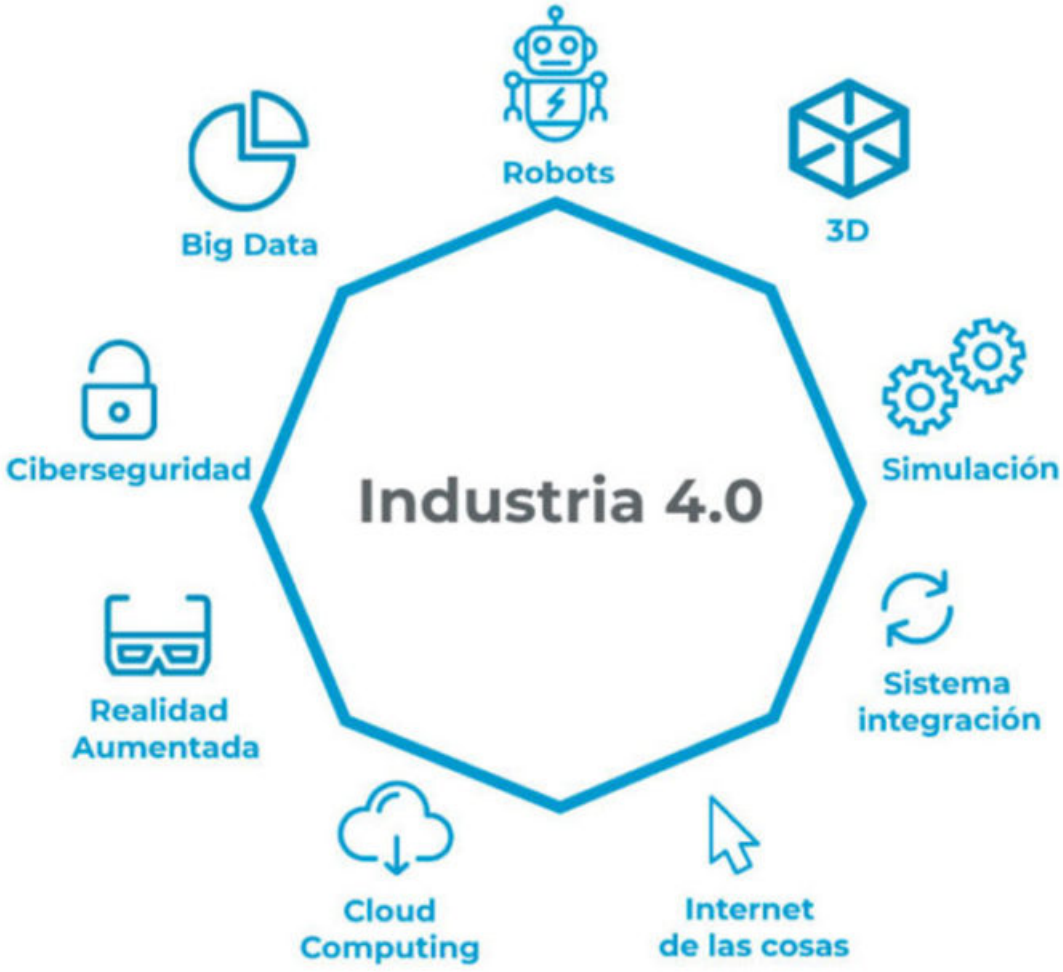


Figura 2.1: Aplicaciones de la Industria 4.0. Tomado de [1].

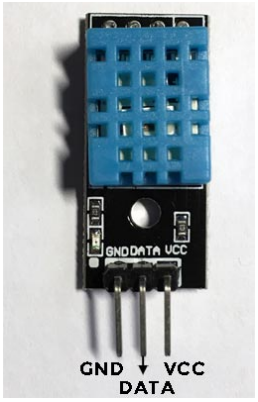
genere acciones en otro objeto físico, dentro de la lógica de IoT. La figura 2.2 muestra un ejemplo de *Data Logger* y otro de *Data Collector*, para ilustrar la diferencia.

Un *Gateway* en comunicaciones, se define como un dispositivo que actua como puerta entre dos redes [13], razón por la que en español se le conoce como puerta de enlace. Permite que datos (de cualquier tipo) viajen hacia y desde una red hacia otra, razón por la que es considerado como un nodo de la red a la que pertenece.

En el contexto de este trabajo, un *Data Collector* envía información a algún servidor pasando por un *Gateway*, y este servidor puede ejecutar acciones dentro de la misma red del *Data Collector* con datos que contienen la instrucción a través del mismo *Gateway*.

2.1.3. Zigbee

Se trata, según [2], de una especificación para redes de bajo costo y baja potencia de consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal o WPAN. Se utiliza en aplicaciones con envío de datos de baja tasa y que requieran conservación



(a) Lector de humedad y temperatura con arduino. Tomado de [12]

(b) Medidor de temperatura con pantalla. Tomado de [11]

Figura 2.2: Diferencia entre un *Data Collector*, sin procesamiento ni guardado de datos (a la izquierda) y un *Data Logger*, mucho más elaborado (a la derecha)

de energía en las baterías. Opera en frecuencias de 868 MHz, 902-928 MHz y 2.4 GHz, mientras que la velocidad de envío de datos puede llegar a 250 Kbps (a 2.4GHz).

Dependiendo de los requerimientos, existen tecnologías de transmisión de información mucho más idóneas para resolver un problema, descritas en [14]. Primero, las tecnologias *Peer-to-peer* (P2P) en que solamente dos unidades pueden participar. Luego, van las de tipo LAN que poseen un área de cobertura moderada y son rápidas, baratas, fáciles de implementar, pero con un consumo de potencia muy alto (Wi-Fi). También están las redes de área amplia y de baja potencia (LPWAN), utilizadas cuando se requieren comunicaciones a larga distancia pero con un envío de datos bajo. Finalmente, están las redes WPAN (Wireless personal area network) que son ideales para productos con batería, que necesiten enviar bajas cantidades de datos con bajo rango de operación. Algunas de ellas utilizan red malla (many-to-many) y la idea de esto es que para enviar información de un lugar a otro con distancia considerable, se haga por medio de unidades intermedias, requiriendo baterias de menor consumo en el proceso. Estas últimas son en las que se enmarca Zigbee, tecnología utilizada en este trabajo.

Entre las tecnologías WPAN, también descritas en [14], se encuentra también BLE (Bluetooth Low-Energy), más utilizada que Zigbee y que también opera a 2.4 GHz, soportando hasta 33000 dispositivos aproximadamente (Zigbee soporta el doble, 65000), Z-wave que compite en domótica con Zigbee y BLE pero usa una banda sub-1GHz que es variable según el lugar, haciendo difícil la venta de productos con esta tecnología de manera global, tiene rango mayor, interferencia menor pero 10 veces menos velocidad aproximadamente que Zigbee y BLE, finalmente 6LOWPAN es una red basada en IP como Wi-Fi, su nombre se basa en el protocolo IP-6 y Low-Power Wireless Personal Area Network.

La arquitectura de sistemas Zigbee tiene la forma de la imagen 2.3, en que IEEE 802.15.4 comprende las dos capas inferiores. Basándose en [2]:

- La capa física comprende las operaciones encargadas de envío y recepción de señales.
- La capa MAC o control de acceso al medio se encarga de la sincronización de comunicación, con el objetivo de transmitir la información de manera confiable.

- La capa de red se encarga de todo lo relacionado a sus operaciones: Establecimiento, desconexión, ruteo, etc.
- La capa de soporte de aplicación activa los servicios necesarios para que el objeto dispositivo Zigbee (ZD0) y los objetos de aplicación interactuen con las capas de red para el manejo de datos, es decir, conecta dos dispositivos de acuerdo a sus servicios y necesidades.
- Finalmente, la capa de aplicación se divide en los objetos de aplicación y ZDO (Objeto Dispositivo Zigbee). Un objeto de aplicación se encarga de implementar la aplicación (por ejemplo, un LED o un set de entrada/salida) y como tal está al tope de la capa de aplicación, mientras que un ZDO es una interfaz que se encarga de la gestión de los objetos de aplicación.

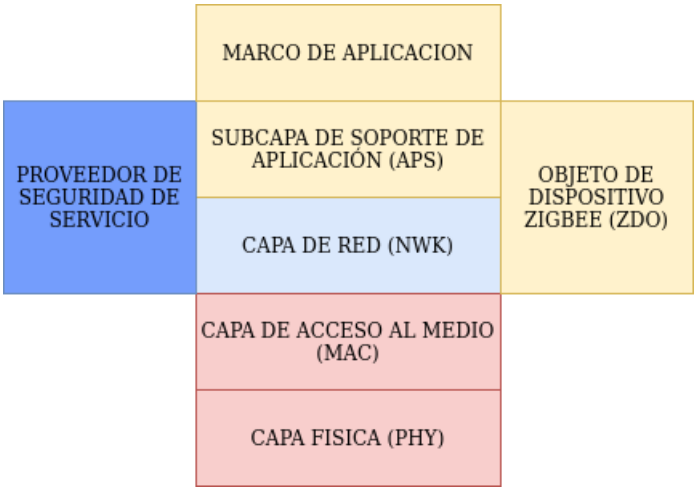


Figura 2.3: Arquitectura de Zigbee. Adaptado de [2]

Los dispositivos Zigbee pueden ser utilizados en tres modalidades, dependiendo del uso que se les pretenda dar. Los coordinadores se encargan de la gestión de información, a la vez que se ejecutan las operaciones de transmisión de datos. Los *router* son centros intermedios que permiten el traspaso de información. Finalmente, los dispositivos de destino tienen funcionalidad limitada, mantienen bajo su consumo y son así controlados por nodos padre. La cantidad de unidades en una red pertenecientes a cada modalidad depende de las distintas topologías posibles, presentadas en la figura 2.4.

La forma estrella implica que la información viaja de una unidad a otra a través del mismo camino, existiendo un coordinador y dispositivos de destino. La topología árbol (cluster en la figura 2.4) contiene *router* que actuan como nodos centrales y operan para extender el dominio de la red, comunicándose con el coordinador y los dispositivos de destino. La más utilizada es la topología malla (mesh en la figura 2.4), por permitir redundancia en el envío de información por medio de conectar todos los *router* entre sí, lo que entrega como consecuencia una red menos jerárquica.

La cantidad de dispositivos supuesta se basa en la creciente demanda en servicios de automatización, considerando la pujante industria 4.0. Sin embargo, este crecimiento debe venir de la mano de una mayor mano de obra especializada en el despliegue y mantenimiento de este tipo de tecnologias. Ejemplos de productos Zigbee son ampolletas LED, termostatos,

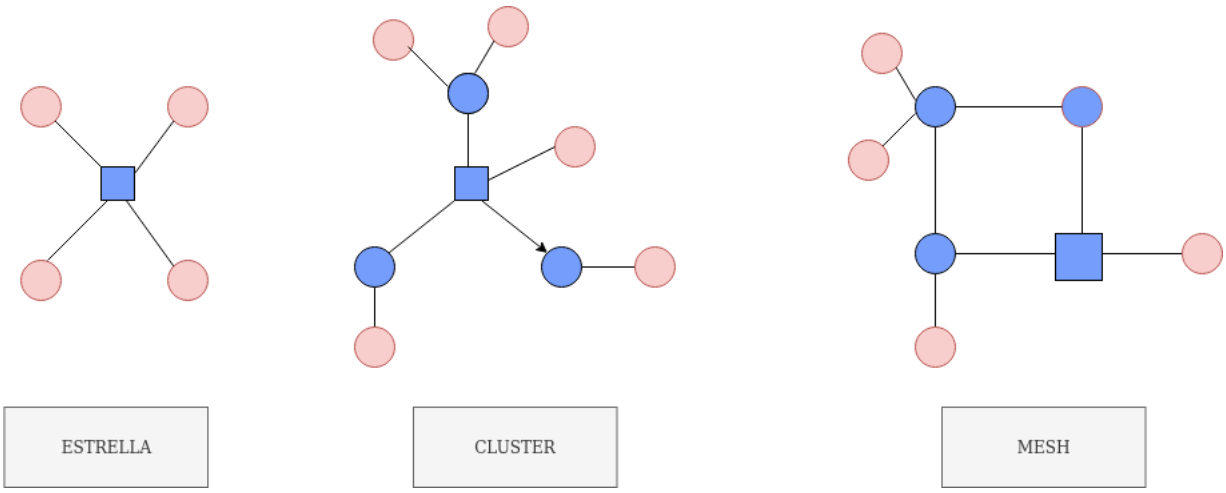


Figura 2.4: Topologías de Zigbee. Adaptado de [2]

medidores de gas y candados [15]. Empresas involucradas a la fecha son multinacionales como Samsung, Philips y Xiaomi.

2.1.4. Firmware

Vale primero definir dos conceptos cuyo límite se confunde con firmware y que son parte de un (micro)computador: Software, que es un conjunto de programas corriendo en algún sistema que han sido diseñados para ser manipulados por un usuario y hardware, que son los componentes físicos sobre los cuales corre el sistema [16].

El firmware es un programa de software o set de instrucciones programadas en un hardware, esto es, en un dispositivo. Proveee la lógica de comunicación con el hardware de algún otro dispositivo con el cual pueda conectarse y se puede programar guardando este set en memoria flash, que se caracteriza por su posibilidad de ser eliminada o reescrita [17].

Las tres partes señaladas por lo general se diferencian en cuanto a su actualización: El software se actualiza todo el tiempo, el hardware casi nunca y el firmware cada cierto tiempo tiene cambios que requieren ser hechos [16]. En un contexto IoT, muchos dispositivos físicos requieren actualización y suelen surgir problemas de seguridad e invasividad, mejor detallados en la sección 2.2.2.

2.1.5. Comunicación Intraprocesos: Sockets vs RPC

La comunicación entre procesos [18] consiste en distintos procesos de un sistema operativo que se comunican y sicronizan a través de compartimiento de espacios de memoria o herramientas provistas por rutinas específicas de comunicación, la que se establece por medio de protocolos de comunicación desarrollados en cada capa. Los procesos pueden ejecutarse en más de un computador conectado a la red y las técnicas de comunicación están divididas para paso de mensajes, sincronización, memoria compartida y llamadas de procediemientos remotos (RPC). La técnica utilizada depende de los requerimientos de los procesos, entre ellos: ancho de banda, latencia, tipo de datos y simultaneidad de las peticiones.

Si bien existen otras estrategias eficientes para establecer comunicaciones entre procesos dentro de una misma máquina a distintas capas de protocolo, como el patrón de comunicación Pub/Sub (*Publisher/Subscriber*) [19] y Redis, estructura de almacenamiento de datos en memoria utilizada como base de datos, caché e intermediario para la compatibilidad entre dos lenguajes de programación dentro de un sistema [20], se analizan las dos formas con las que se realizan pruebas, RPC mediante gRPC y Sockets mediante 0mq, aunque cabe mencionar que gRPC también utiliza Sockets.

El sistema operativo posee dos operaciones primarias, enviar (*send*) y recibir (*receive*) y se debe agregar un enlace de comunicación entre procesos, el que puede ser unidireccional o multidireccional, en uno o en varios sentidos. Algunas características de dos tipos de comunicación entre procesos considerados en el desarrollo de esta memoria, RPC y sockets, son las siguientes:

- **RPC:** En español denominado Llamada a Procedimiento Remoto, permite que uno o más programas realicen llamadas a funciones localizadas en otras máquinas dentro de una red, operando estas funciones tal como si fuesen llamadas localmente, lo que se denomina transparencia. Cada función pasa a tener una parte cliente, en la cual se implementa el prototipo de una función para la invocación remota y una parte servidor, en que se implementa la función base. Es necesario tener cuidado en el traspaso de parámetros, no suponiendo desde el cliente aspectos como el lenguaje de programación o el tipo de codificación por parte del servidor. Un ejemplo de este tipo de comunicación es gRPC, protocolo de código abierto desarrollado por Google y que tiene su implementación en Python, C++, Java, Ruby, entre otros lenguajes de programación [21].
- **Sockets:** Se basa en la existencia de un receptor que no siempre está activo y dispuesto a comunicarse, no soportando el envío de mensajes de forma persistente. Establecen un enlace de comunicación bidireccional [22] entre un cliente y un servidor, operando en base a funciones o peticiones de programación a modo de comando específico, por ejemplo: *Bind* añade la dirección local al socket, *Close* desvincula la dirección local del socket y *Listen* permite que quede en espera de nuevas conexiones.

  Ambos protocolos pueden operar sobre TCP/IP, en que TCP (Protocolo de Control de Transmisión) asegura un transporte fiable de los datos e IP (Protocolo de Internet) lleva los datos a distintas máquinas dentro de la red [23].

2.2. Estado del arte

2.2.1. Dispositivos XBee

¿Qué son los XBee?

Se trata de soluciones empaquetadas e implementadas por la empresa de manufactura Digi, que ofrecen conectividad inalámbrica por medio del uso de distintos estándares, entre ellos 802.15.4, DigiMesh, WiFi y Zigbee. Existen diversas series, variables en cuanto a la funcionalidad, tipo de antena y frecuencia de funcionamiento, descritas en [3] y señaladas en 2.5.

- **XBee Series 1:** No requieren configuración, fáciles de utilizar.
- **XBee Series 2:** Por un lado, se encuentran los de tipo Znet que ya se encuentran obsoletos y requieren ser configurados, ya sea en modo transparente o por comandos API. Un modelo más reciente es el ZB, cuya única diferencia es una actualización de firmware y el más actual como también el utilizado es llamado 2C, con mejoras en cuanto al uso de la potencia y que incorpora comunicación SPI (Serial peripheral interface).
- **XBee Series 3:** Poseen principalmente la capacidad de procesamiento de información, por medio de la inclusión de micropython, volviéndose además programables los distintos nodos.

Adicionalmente, existe la familia de 900 MHz (la ya descrita utiliza 2.4 GHz), en que se incluyen los módulos X2C que poseen un mayor alcance, pero menor velocidad de transmisión de datos y la consecuente mejora X2C S3B, con disminución en el consumo de energía.

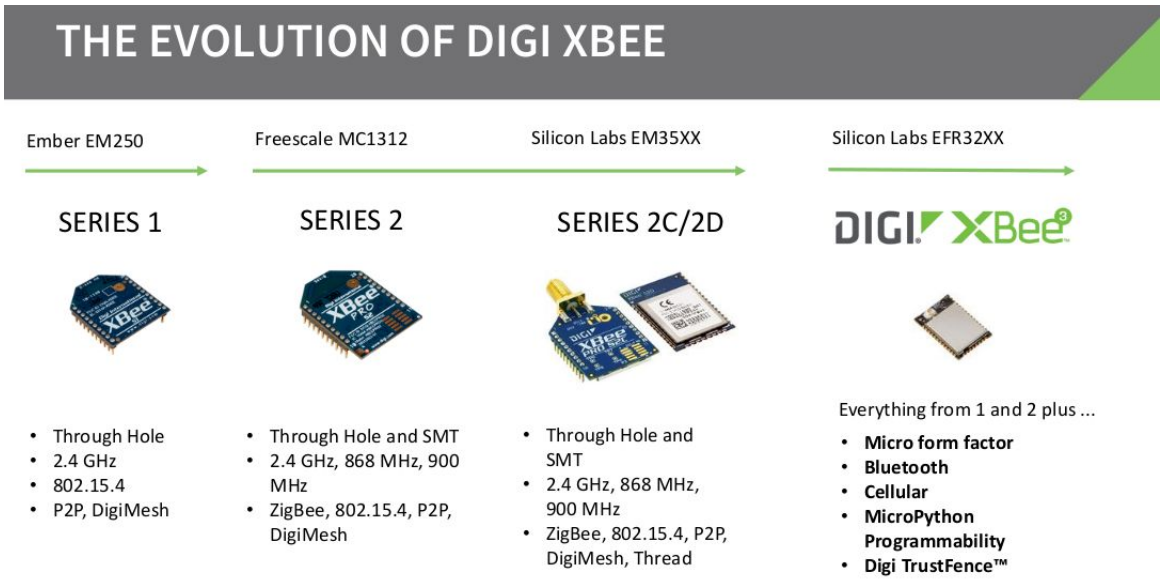


Figura 2.5: Evolución de prototipos Zigbee por Digi [3]

Importantes en términos de aplicación son las antenas a utilizar, algunas de ellas visibles en la figura 2.6. El tipo whip es un cable saliente, u.FL es un conector para agregar la antena propia y chip es una tarjeta pequeña a modo de transmisor.

XBee: Modos y Comandos

Las unidades X2C pueden ser utilizadas en modo AT y API, cuyas características son las siguientes:

- **AT (Transparente):** Consiste en que cada dato enviado a un dispositivo XBee es inmediatamente enviado al módulo remoto que se encuentre identificado como dirección de destino. No son necesarios paquetes y por ende, es la forma más simple de transmitir y conviene cuando se trata de comunicación con un solo dispositivo.

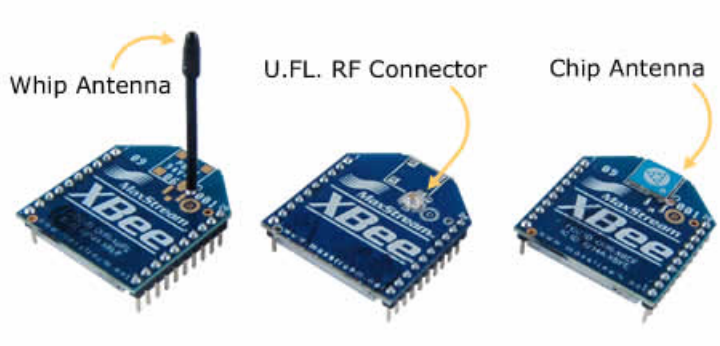


Figura 2.6: Tipos de antenas disponibles [3]

- **API (Interfaz de Programación de Aplicación):** El mensaje es armado en un formato dentro de un paquete de datos, siendo especialmente útil en redes grandes en que se quiere transmitir información a varios dispositivos a la vez, por ejemplo, configurar nodos de forma remota.

En el sistema PMSystem tratado en esta memoria, se utilizan ambos modos: API para los *Gateway*, ya que ellos configuran múltiples dispositivos y AT para los *Data Collector*, ya que estos necesitan enviar información de manera rápida y no requieren establecer comunicación con un nodo distinto al *Gateway*.

En la página oficial [24] se encuentra el manual de la guía de usuario de XBee, en que se cubren los comandos principales. Entre las funcionalidades se encuentra la posibilidad de crear una red con un PAN ID (id única de una red de área personal) asociado y un identificador único para cada nodo, lectura/escritura de entrada/salida digital y análoga, manejo del modo *sleep*, definición del coordinador, lectura y escritura de parámetros en los *Data Collectors* desde el *Gateway*, entre muchas otras. Tres ejemplos de comandos útiles para el desarrollo de este trabajo, de un total en la referencia de 97, son:

- **SC:** Lista de canales a buscar para incluir a la red (por parte del *Data Collector*) y para involucrarse con un *Data Collector* (por parte del *Gateway*).

- **AP:** Cambio del modo API (transparente, *API mode* y *API escaped operating mode*).

- **IR:** Modificar o leer el periodo de las entradas seleccionadas.

En cuanto a los mensajes, se encapsulan de forma que abarquen la información necesaria en el menor espacio posible, tomando en cuenta los campos utilizados por la librería XBee. El mensaje de recepción optimizado tiene la forma de un diccionario del tipo:

["parameter":D7, ïd":3, "value":03, "status":00]

En que el status es de la recepción (código 00 si todo sale bien, distinto si no), value es el valor del parámetro parameter e id es la identificación del *Data Collector*. Siendo dos entradas medidas (sensores), el mensaje es una lista de dos diccionarios.

XBee: Disponibilidad de soporte

El software libre utilizado es XCTU (tutorial en [4]), aplicación diseñada para permitir a los desarrolladores interactuar con los módulos Digi a través de una interfaz gráfica simple (figura 2.7). Las funcionalidades incluyen de procesos propios de la capa red hacia arriba: Detección de los módulos conectados a la red a través del coordinador, asignación de parámetros individuales a las unidades Zigbee, envío de mensajes y comandos, entre otras.

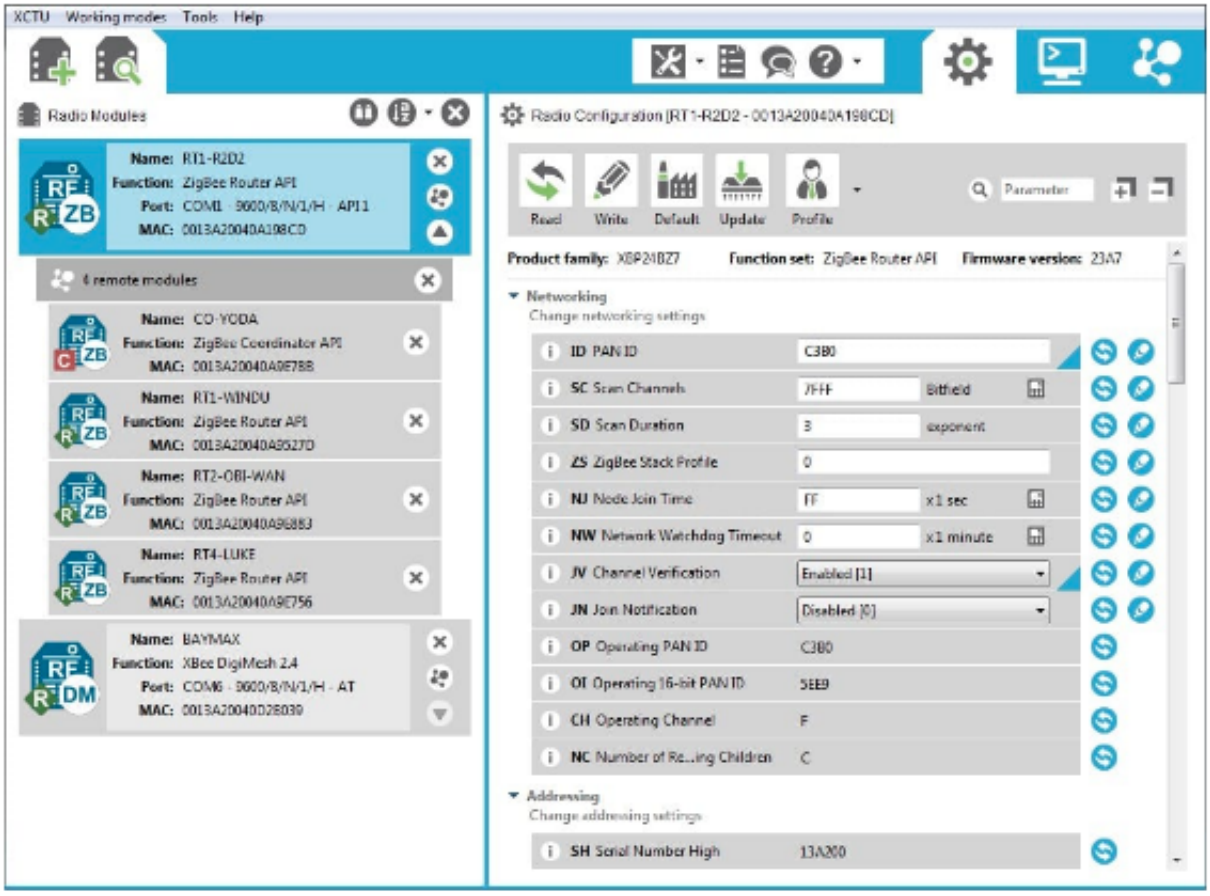


Figura 2.7: Interfaz XCTU [4]

Los comandos son enviados por comunicación serial al dispositivo XBee *Gateway*, que a su vez los envía a los XBee remotos. Son universales y se utilizan en los distintos lenguajes que tienen soporte para aplicaciones reales de Zigbee, como Java, C++ y Python, en librerías ya sea creadas por la empresa manufacturera de los dispositivos XBee (Digi) o por otros grupos de desarrolladores, que permiten la construcción de los mensajes. Ejemplos de repositorios en distintos lenguajes para estas librerías son: Xbee-python y xbee_ansic_library, oficiales de Digi en Python y C [25], [26], libxbee3 en C++ [27] y XBee en python como opción alternativa [28] [25].

2.2.2. Avances sobre el monitoreo de colectores de data

Trabajos académicos

Se proponen maneras en que se puede proteger distintos sistemas IoT de diversos ataques malintencionados en aplicaciones como vehículos autónomos, casas inteligentes, entre otros.

IoT-based Software Update Proposal for Next Generation Automotive Middleware Stacks En [29] se resuelve el problema de actualización en vehículos autónomos por medio de una arquitectura que integra el sistema OBLO y la plataforma AUTOSAR adaptativa (OTA), ambas de tipo IoT, tomando como contexto la necesidad de rapidez para actuar frente a potenciales bugs debido a la conexión de elementos del vehículo con el mundo exterior. EL sistema OBLO es originalmente para casas inteligentes y se trata de un portal de usuario (back-end) que notifica la existencia de actualizaciones y da la opción de elegir actualizar. OTA es un puente plataforma-servicios IoT que se abstrae de los protocolos de estos últimos. La plataforma ARA::UCM es un servicio que se encarga de las actualizaciones de todo tipo, recibiendo como entrada un paquete con archivos binarios que es creado y testeado para ser subido a OBLO. Las estrategias de seguridad utilizadas consisten en validación de paquetes, chequeos de memoria/versiones correctas y la ejecucion de la actualizacion mientras se encuentre el vehículo en modo estacionado o de conducción, siendo puesto previamente en modo actualización.

Decentralized coordination of dynamic software updates in the Internet of Things El problema a tratar en [30] es la inconsistencia derivada de la actualización automática en componentes de nodos distribuidos, que se realiza de forma independiente, perdiéndose información en el proceso. Se propone una solución compuesta por dos aspectos y probada en Java 1.8: Un mecanismo de adaptación no anticipado basada en LyRT, arquitectura que envía las actualizaciones a un *manager* dentro de un *host* que redirige a su vez estos al resto de los dispositivos que deban ejecutar la actualización y que tiene el propósito de asegurar una disrupción mínima al sistema de medición con actualizaciones seguras en los nodos y una capa *middleware* (*software* que se sitúa entre un sistema operativo y las aplicaciones que se ejecutan en él) descentralizada que se encarga de gestionar las actualizaciones con tal de asegurar una transición consistente de la aplicación, esto para extender la solución hacia nodos distribuidos. Aún con esta estrategia, los datos en puntos comunicados pueden perderse debido a inconsistencias de encriptaciones hechas por distintos softwares (en que un emisor se actualiza y envía, pero el receptor al momento de recibir aún no ha actualizado), además que estos nodos deben tener un poder de cómputo y memoria fuertes, distinto al caso tratado en esta memoria.

ASSURED: Architecture for Secure Software Update of Realistic Embedded Devices El problema tratado en [31] es la protección contra malware, debido a exposición remota de dispositivos conectados a internet mayor versus otras tecnologías, actualización complicada de sistemas embebidos y su funcionamiento desatendido, en general. Se desarrolla ASSURED, un framework de actualización seguro y escalable para IoT en dispositivos limitados en recursos. Este extiende algún framework existente y un mecanismo de autorización (ASSURED) que permite la especificación de restricciones de actualización de partes interesadas, que verifican el despliegue exitoso de ésta, considerando un contexto de lanzamientos

IoT de larga escala, siendo la idea permitir a los creadores de controladores y manufactura de dispositivos el imponer sus propias restricciones respecto a firmware de una forma establecida. Esta organización entre las distintas partes interesadas en una tecnologia determinada no existe aún para XBee, no pudiendo contar ingenieros a cargo de proyectos implementados con la posibilidad de manejar este tipo de configuraciones, aunque como los trabajos siguientes de IETF indican, esto cambiará en un futuro cercano.

Remote Software Update in Trusted Connection of Long Range IoT Networking Integrated With Mobile Edge Cloud El problema tratado en [32] se basa en la utilización de dispositivos con menor capacidad de almacenamiento para la implementación de IoT, en que se hace importante mantener conexiones de confianza durante la entrega de actualizaciones. La propuesta emplea una LPWAN para un largo alcance y usa computación perimetral de acceso múltiple en que el procesamiento se realiza más cerca de la fuente de datos para mejorar la eficiencia de cómputo, que es integrada en un *Gateway* y puede procesar actualizaciones remotas de software LPWAN, al igual que un análisis estadístico de conexiones, determinando las de confianza, aumentando la proporción de entrega y con consumo bajo en la transmisión. El sistema es útil para servicios remotos IoT en que el *delay* sea una limitante importante y al ser Zigbee una tecnología de tipo LPWAN, la solución propuesta es interesante, aunque no se conoce como funcionaría con el poder de cómputo de una Raspberry u otro controlador de similares características.

Avances en el establecimiento de estándares

Software Updates for Internet of Things (suit) es un grupo de trabajo perteneciente a IETF (Grupo de Trabajo de Ingeniería de Internet, organización abierta de estandarización) que se encarga de establecer un mecanismo seguro de actualización de firmware por vulnerabilidades en IoT usable en clase 1 (10 kiB RAM, 100 KiB flash) [33]. Intenta publicar una Arquitectura de actualización de firmware en IoT y una o más especificaciones con formato de manifiesto (conjunto de metadatos sobre el firmware de un dispositivo IoT, dónde encontrar el firmware, los dispositivos a los que se aplica y la información criptográfica que le protege). El grupo ya señalado tiene un segundo documento llamado “Un modelo de información para actualizaciones de firmware en dispositivos IoT [34]", documento de boceto valido por seis meses y que puede ser acutualizado, reemplazado u obsoleto por otros documentos en cualquier momento, por lo que se trata de un trabajo en progreso. Se mencionan elementos de seguridad que este modelo debería tener, consideraciones de seguridad (ejemplificando con casos prácticos/reales de vulnerabilidades existentes) y formas de resolución de estos problemas.

Otro grupo de trabajo en IETF es el de procedimientos de testeo AT (rats en inglés) [35], basándose en que en protocolos de intercambio de red, a menudo una entidad requiere evidencia acerca del par remoto (y sus componentes de sistema) con tal de ver la confianza del par. Mientras que existen mecanismos de atestación de dominio específico, no hay forma interoperable de crear y procesar evidencia de atestación para hacer determinaciones acerca de componentes de sistema entre las partes confiantes de distintos origenes y distinta manufactura. El grupo desarrollará estándares que soporten atestamiento remoto interoperable para componentes de sistema y sus principales entregables son la especificación de los casos de uso para atestamiento remoto, de terminología y arquitectura que permite técnicas de

atestamiento, junto a la estandarización de aspectos como el modelo de información, de datos que permitan su implementación y de protocolos interoperables para transferir de forma segura aserciones.

Existe también un documento llamado terminología para redes de nodo restringido [36], publicado con el propósito de informar de la terminología utilizada en otros documentos. Define, entre otras cosas, lo que es un nodo restringido, una red restringida, entre las que se encuentra 6LoWPAN (mencionada en este documento), limitaciones energéticas y lo más importante, estrategias para utilizar eficientemente dispositivos inalámbricos con estas limitaciones energéticas. Este documento es importante, ya que en el documento y en la informacion de los grupos de trabajo mencionados se utilizan terminos como las clases de dispositivos y nodos/redes restringidas.

2.2.3. Trabajos realizados con uso de tecnología Zigbee

Se ejemplifican a continuación desarrollos realizados con tecnología Zigbee con el propósito de monitorear algún entorno industrial a pequeña escala, tal como es el caso del presente trabajo.

En Mane et al. [5] se implementa un sistema de monitorización de amoniaco en una planta industrial utilizando una PIC18F4550, que es conectada a un dispositivo XBee. Es un sistema con cuatro nodos, tres de ellos son lectores de datos (porcentaje de amoniaco) y el cuarto es un nodo a la vez coordinador y actuador. Los tres primeros nodos son los marcados como A en la figura 2.8 y la lectura de dato es enviada a través de un dispositivo XBee, mientras que el nodo coordinador en B mueve una válvula cuando obtiene una concentración de gas peligrosa. A ambos lados hay procesamiento por medio de una PIC18, un LED y una alarma que avisan de niveles nocivos de amoniaco. No existe un enlace a internet, por lo que no es necesario el uso de un *Gateway*.

La implementación realizada tiene éxito y las mediciones son precisas como requerimiento de seguridad, aunque no se presentan mayores detalles acerca de los tiempos de reacción del sistema ni la pérdida de datos, por lo cual no permite entender si la PIC18 representa una alternativa válida como microcontrolador para preprocesamiento.

En Nikhade et al. [6] utilizan un sistema con estándar Zigbee con aplicaciones en agricultura, en que se utiliza una Raspberry como unidad coordinadora de Zigbee y la unidad que recibe el dato como *router*, generando la conexión con Wi-Fi a través de una interfaz web HTML, tal como puede verse en la figura 2.9. Esta unidad base puede manejar varios sensores-nodos, cada uno de ellos consistentes en una combinación de microcontrolador, sensor y módulo XBee.

Existe un coordinador conectado por medio de protocolo UART a la Raspberry y aunque en el experimento descrito se habla de una topología malla, se utilizan solamente dos nodos y un mayor escalamiento queda propuesto, razón por la que no se vislumbra el tipo de problemas existentes en el *software* del *gateway* al tener una red más compleja, como la existente en este problema.

En Masetti et al. [7] se diseña un sistema de monitorización de fermentación de vino

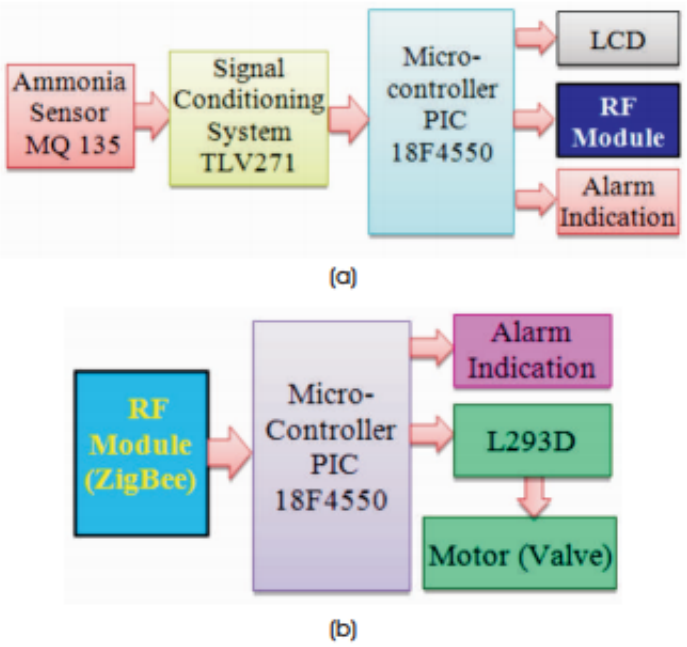


Figura 2.8: Diagrama de bloques del sistema realizado por Mane et al. [5]

durante su proceso de síntesis. El sensor de medición consiste en una boya hecha en impresora 3D, que contiene el microcontrolador ATMega328p, XBee y los sensores con su electrónica asociada. Se mide pH, nivel de líquido y temperatura, a ser enviadas hacia un punto de acceso, que consiste en una Raspberry, cada una hora, que luego envía la información hacia un servidor web. El sistema físico puede verse en la figura 2.10.

El sistema implementado genera un consumo de 55.111 mA en modo activo para el dispositivo XBee, de los cuales 45.000 mA provienen de él. Este valor para modo hibernación/sleep es de 0.456 mA, dando así a entender de forma más clara la importancia de esta funcionalidad en el ahorro de energía. Con un modo sleep de 30 minutos por ciclo, la durabilidad de las baterías reportada es de 16 días, insuficiente para satisfacer los requerimientos, por lo cual se propone como trabajo futuro por parte de los mismos autores un aumento del tiempo de modo hibernación. A diferencia del trabajo a desarrollar, estas lecturas no son constantes y se hace uso de baterías propias de cada nodo.

En Xiang et al. [8] se describe el monitorización de control de calidad del motor dentro de un barco mediante Zigbee, pudiendo los usuarios autorizados acceder al estado en tiempo real del sistema, analizar el historial de datos y enviar comandos de control de ambiente a través de una aplicación en Android o una página web en un computador.

Se utiliza SQL para la base de datos asociada, junto a la herramienta Hibernate en Java. Para el desarrollo web se utiliza Eclipse y se integra posteriormente en Android, a diferencia del uso de Ruby para una interfaz web dentro del trabajo a realizar (no de aplicación web).

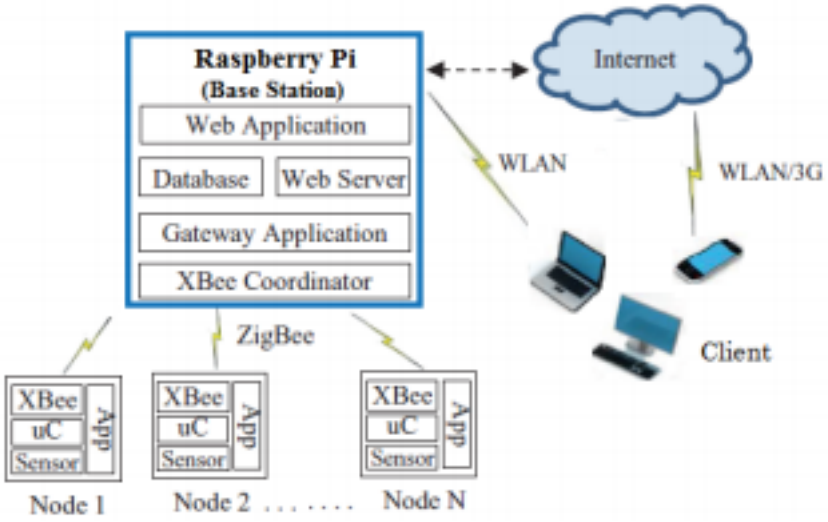


Figura 2.9: Diagrama de bloques del sistema realizado por Nikhade et al. [6]

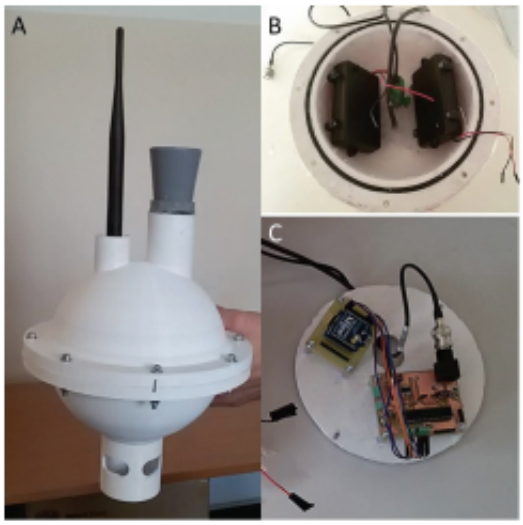


Figura 2.10: Boya desarrollada en Masetti et al. [7]

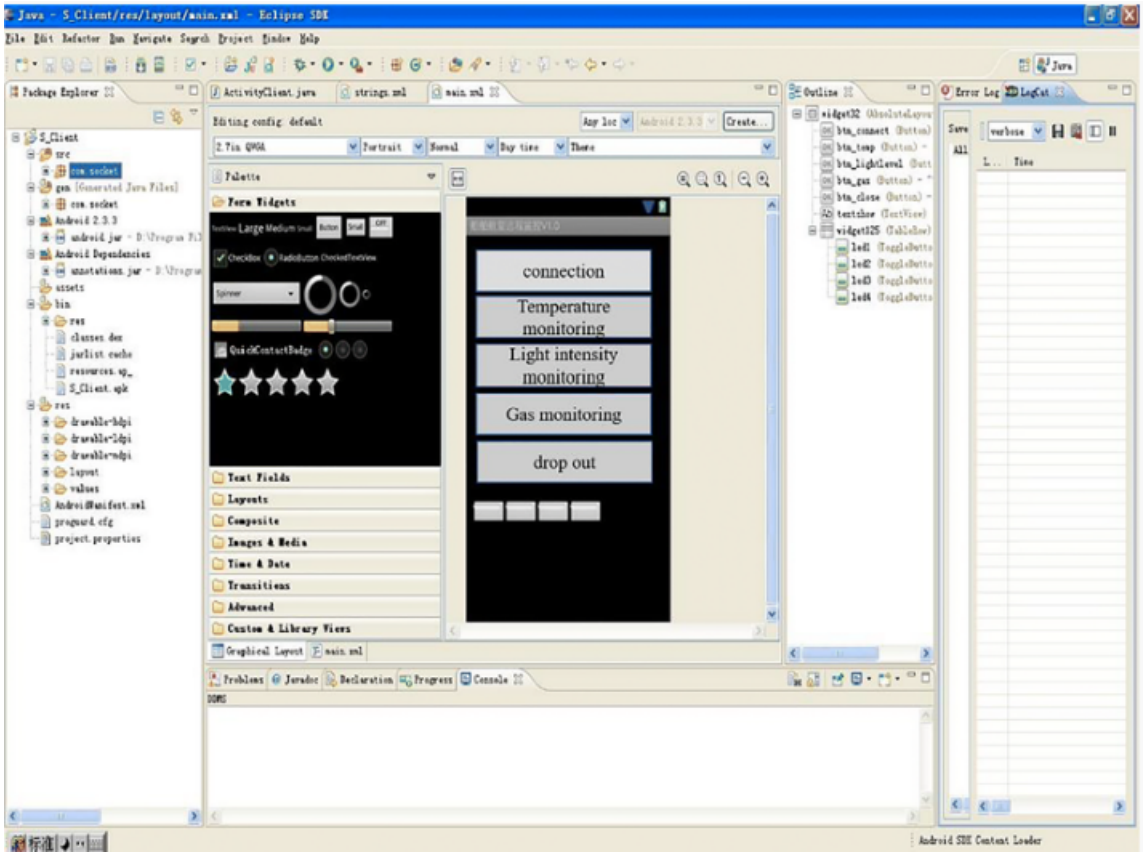


Figura 2.11: Aplicación Android a partir del trabajo de Xiang et al. [8]

Capítulo 3

Metodología

En esta sección se presenta primero el estado, luego la arquitectura de sistema actual y propuesta en que se incluyen dos versiones de implementación de sistema.

Primero, es necesario definir el estado a medir a través del desarrollo de este trabajo, para describir por medio de los dispositivos XBee la forma en que pueden medir este estado.

La segunda parte de esta sección está constituida por la arquitectura del sistema antes del desarrollo del trabajo, las mejoras propuestas a esta arquitectura, tareas a realizar y finalmente la implementación, a través de un sistema inicial con el propósito de probar comunicaciones interprocesos y un sistema final, en que se desarrolla por medio de Django un servidor al que un cliente desde un navegador le realiza las peticiones pertinentes.

3.1. Definición del Estado de los dispositivos

El estado, es decir, el conjunto de variables que se desean medir, es el siguiente:

- **Monitoreo de *Data Collectors*:** Los sensores utilizados, ya sea de pulso o de estado encendido/apagado se encuentran conectados a los pines de un ATMega328 que a su vez se conecta con el XBee, estando el ATMega328 aislado eléctricamente del sensor por medio de un optoacoplador. El *gateway* recibe mediciones, que tienen la opción de filtrado para eliminar el ruido. Además de la información de las mediciones en si, los parámetros a verificar están ligados a que los pines (físicamente conectados) estén activados para ser leídos de la manera en que se desea, por ejemplo, una tasa de muestreo correcta o que no estén configurados como entradas análogas o como salidas cuando son entradas digitales. Para el ATMega328, además de las dos mediciones de los sensores señaladas en el párrafo anterior, está el ver si el modo de operación (ligado al procesamiento a realizar, específico de la aplicación) es el correcto, si el voltaje de alimentación es adecuado, si el ATMega está encendido o si la temperatura está en el rango de operación del hardware, todo esto permite ver si se puede confiar en lo que se está midiendo.
- **Configurar *Data Collectors*** : En caso que el monitoreo entregue un funcionamiento

incorrecto por parte de algún *Data Collector*, es necesario entrar a medir parámetros más específicos para determinar qué está produciendo el error visible, los cuales constituyen el estado en este caso. Una vez medidos, estos parámetros se deben modificar correctamente. Por ejemplo, si el monitoreo entrega un error en la medición recibida al no reportarla, el estado específico a verificar y configurar podría ser las entradas físicas de XBee a reportar o el periodo con el cual lo hacen.

- **Monitoreo de la Red**: Se desea poder monitorear los dispositivos, por lo que el estado en este caso es la existencia o no de conexión entre el Raspberry y el *Data Collector* o sensor (puede estar conectado el XBee, pero con sensores a él que no están siendo leidos). A la vez, se tiene un archivo de configuración con sensores existentes y *Data Collectors* existentes, lo que permite adicionalmente detectar dispositivos XBee conectados a un determinado *Gateway*, pero que no se encuentran incluidos en la red deseada de estos archivos de configuración.
- **Gestión de la Red**: En ciertas aplicaciones se requiere la deshabilitación temporal de un *Data Collector* en específico, por lo que la existencia o no de conexión vista en el caso anterior no es suficiente, requiriendo de esta manera un estado adicional que indique si el dispositivo se encuentra deshabilitado o no. Sin embargo, hay necesidad de desconexión permanente en otras situaciones (por ejemplo, la falla de un sensor), por lo que en lo que se refiere a la red, la existencia o no de conexión es suficiente.
- **Envío de Firmware**: La programación del ATMega es motivada por la versión de firmware y configuración de parámetros inconsistente, nuevamente por ejemplo, el modo de aplicación, los periodos asociados a él o más específico aún, que el pin del ATMega no se corresponda al pin del XBee, por lo cual se lee basura.

Hay más estados posibles que no van a ser considerados: parámetros de los XBee que permiten ver el estado de conexión de manera más específica (ejemplos: canal de conexión o canal de escaneo), características del *router* (ejemplo: cuántos nodos se pueden agregar), características de firmware más allá de la versión, entre muchos otros parámetros más profundos y propios de los dispositivos que no vale la pena considerar por simplicidad y por complejidad innecesaria añadida al cometido.

3.2. Tareas a realizar

El set de tareas principales a llevar a cabo para cumplir el objetivo de evaluación de una interfaz web que monitoree el estado de *Data Collectors* en una planta es el siguiente:

1. Comunicación entre *Gateway* y unidades Zigbee (API)
   - Configuración del XBee
   - Configuración del microcontrolador
   - Programación del microcontrolador, que permite la carga de su código
2. Comunicación entre servidor y *Gateway*
3. Implementación de interfaz web para configuración remota de los dispositivos

La comunicación entre *Gateway* y *Data Collectors* considera la librería a utilizar para comunicar con el microcontrolador y el XBee, junto a los mensajes de envío y sus respectivos

experimentos. Los mensajes de envío son de dos tipos: Datos de sensores, mediciones propias de la planta que van desde el *Data Collector* hacia el servidor en la nube pasando por el *Gateway* y Datos de configuración, que van en ambos sentidos para obtener la configuración del *Data Collector* o para modificar parámetros.

Para la comunicación entre el servidor y el *Gateway*, se encuentra nuevamente la lógica de envío de mensajes, ya que no es claro si se puede enviar todo a una velocidad como la que el proceso opera y además, si existen pérdidas de los mensajes en esta parte. Por ahora, la conexión es posible en dirección *Gateway* y servidor, pero no la inversa servidor a *Gateway*, que permite la configuración remota de los dispositivos.

Cabe mencionar que ahora la conexión se realiza correctamente en dirección *Gateway* a servidor, pero falta la conexión inversa servidor a *Gateway*, que es la que permite configurar los dispositivos.

Finalmente, vale considerar la programación del servidor, debido a la gran cantidad de trabajo que esto requiere y la conexión entre Python, que es el lenguaje utilizado y Ruby on Rails, en que se trabaja el desarrollo web.

3.3. Arquitectura Propuesta

3.3.1. Arquitectura del Sistema Actual

El diagrama de bloques del sistema completo es visible en la figura 3.1. Puede verse que el proceso está constituido de la entrada sensada por unidades llamadas *Data Collectors* que se comunican vía estándar Zigbee, en cuyo interior utilizan una unidad XBee, que implementa la tecnología señalada, enviando toda esta información a un *Gateway* que recibe las mediciones y parámetros solicitados desde cada *Data Collector* y lleva hacia ellos las solicitudes. Finalmente, un *Gateway* o unidad coordinadora de los *Data Collectors* se comunica con el servidor para el posterior acceso por medio de un computador o una aplicación web.

Comunicación entre el *Data Collector* y el *Gateway*

Las peticiones son realizadas desde el servidor en sentido hacia los *Data Collector* en la figura 3.1, mientras que las respuestas a estas peticiones son realizadas en el sentido contrario, es decir, desde los *Data Collector* hacia el servidor, además de ir también en este sentido los datos reportados de medición.

El *Data Collector*, que considera el dispositivo XBee junto al microcontrolador y cuya data de los sensores se encuentra separada por un optoacoplador (señalado como aislador de señal). Ambos poseen alimentación desde la energía de planta, pero en caso de fallas hay baterías que permiten que los *Data Collectors* sigan funcionando, enviando en este caso alertas de corte energético del proceso.

Comunicación entre el *Gateway* y el servidor

En el *Gateway* existen dos motores: El *Engine XBee*, que establece la comunicación con el *Data Collector* con la ayuda de la librería [28] y el *Engine Processor*, que se comunica con el

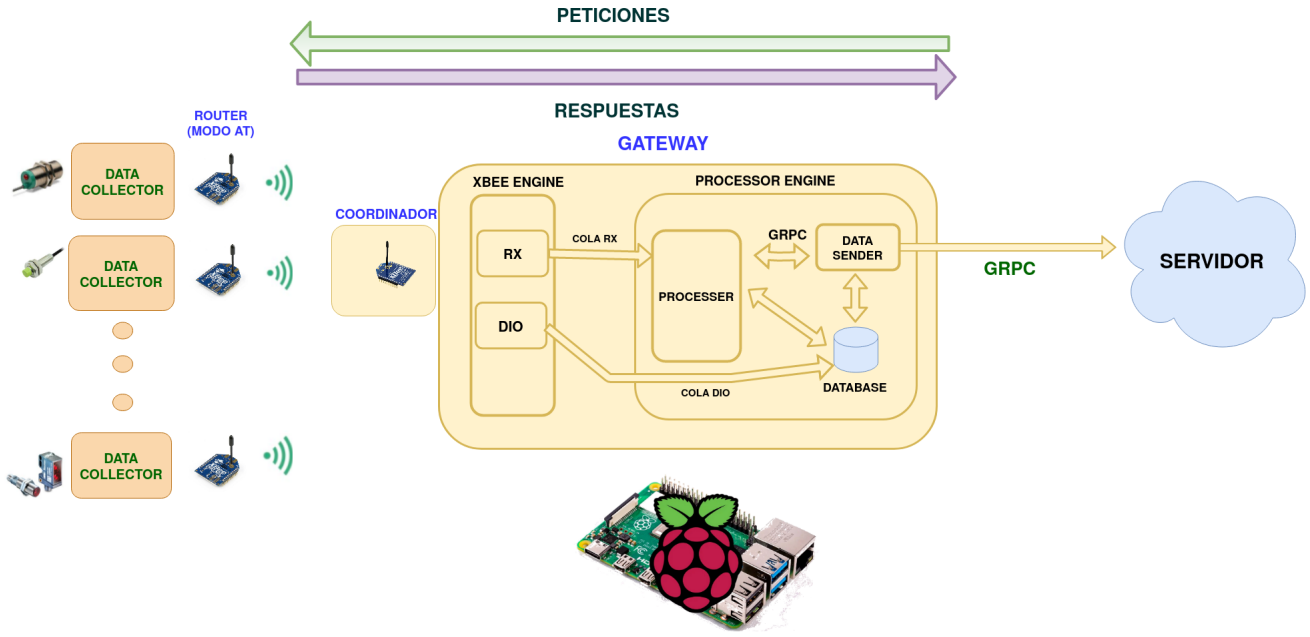


Figura 3.1: Diagrama de bloques del sistema

servidor a través de gRPC. Existen dos canales utilizados por *Engine XBee*: DIO, que envía las mediciones que los *Data Collector* estén realizando y RX, que envía cualquier información de configuración.

Se realiza un procesamiento de sensores en el submódulo *Processer* para transformar la información al modelo relacional de la base de datos SQL implementada, mientras que por medio de un request se envía este mismo valor guardado hacia la nube usando conexión Wi-Fi. Existe en esta parte un detector de falla, tal que se pide a la base de datos que vuelva a enviar el paquete en caso de error o se espera hasta que se reestablezca la conexión y luego se envían los datos almacenados, en caso de fallas de conexión de la red.

Se utiliza el protocolo de comunicaciones Google Remote Procedure Calls (gRPC) entre el *Gateway* y servidor web. La ventaja es su flexible estructura de datos que permite compatibilidad entre distintos lenguajes, para el caso de este trabajo, Python en el *Gateway* y *Ruby* en el servidor. Se utiliza para mensajes pequeños, que es el caso de este problema.

3.3.2. Arquitectura del Sistema Mejorado

El microcontrolador al que se hace referencia en esta subsección es el que forma parte del *Data Collector*, cuya estructura corresponde a la figura 3.2. Existen diversos aspectos en el funcionamiento a tomar en cuenta:

Configuración y lecturas del microcontrolador El microcontrolador posee un protocolo de comunicaciones. Sus funcionalidades dependen de lo que sea necesario preprocesar, por esto tiene una serie de parámetros configurables remotamente y variables que entrega, en caso de ser requeridas, por medio de una función GET o SET en el *Gateway*, dependiendo de si lo que se quiere es preguntar o modificar el valor de un parámetro.

Programación del microcontrolador Se utiliza para reprogramar el microcontrolador, esto es, actualizar su *firmware*.

Lecturas de pulsos Se leen pulsos y estados encendido/apagado. Estos pulsos corresponden a un producto, que pasando a través de una correa transportadora, es detectado por un sensor, habiendo varios de estos sensores a través de esta correa.

Configuración de los XBee Se utilizan los comandos ya descritos en [24] a través de modo API, desde el XBee del *Gateway* hacia los XBee de los *Data Collectors*. Con esto, se pueden configurar parámetros, leer valores relevantes como la alimentación del XBee, invocar el modo *sleep*, entre otros.

Para la comunicación entre *Gateway* y *Data Collectors*, se analiza entre las distintas librerías en cuanto a su índice de aciertos tanto como envío correcto de datos como de ejecución sin retorno de la librería, junto a su tiempo de latencia, seleccionando alguna para la implementación: xbee-python y xbee_ansic_library, oficiales de Digi en Python y C [25], [26], libxbee3 en C++ [27] y XBee en Python como opción. En cuanto a la comunicación entre servidor y *Gateway*, hay complicaciones en cuanto al envío de datos. Desde el *Gateway* al servidor el funcionamiento es usando GRPC, ya que la IP del servidor es única y visible, pero este no es el caso con la IP del *Gateway*, ya que tener la IP del gateway depende de que el cliente habilite esta IP para ser vista desde el exterior, lo que alternativa [28], siendo esta última la que se ha utilizando en la planta de manera provisional.

En lo que a cambios se refiere dentro de la red XBee, cabe mencionar por el lado del *Gateway* la estructura de los mensajes a muy bajo nivel, que a su vez compatibilizan con el microcontrolador cuando se hace necesario. Por el lado del *Data Collector* es donde están los mayores cambios, añadiendo en el Motor XBee (*XBee Engine* en la figura 3.3) dos nuevos canales: TX que permite transmisión serial y AT, bidireccional y que ayuda al envío/recepción de comandos AT de configuración. Además, éste se reformula completamente, debido al análisis realizado acerca de las librerías en Python utilizadas.

Dentro del *Gateway* se añaden dos motores: *Data Collector Engine*, que se encarga de gestionar la información desde y hacia un *Data Collector* en específico, esto porque se hace necesaria la separación de tráfico de datos debido a la cantidad de *Data Collector* existentes y *Configuration Engine*, que maneja los datos de configuración en ambos sentidos entre el *Data Collector Engine* y el servidor, estableciendo la conversión entre el *Data Collector* manejado y su identificador en el mensaje del servidor. *Data Collector Engine* posee adicionalmente un submódulo llamado *Data Collector Communicator* que maneja exclusivamente los datos por canal AT, que a su vez son los únicos que siguen la vía del *Configuration Engine*.

En cuanto a la comunicación entre servidor y *Gateway*, hay complicaciones en cuanto al envío de datos. Desde el *Gateway* al servidor el funcionamiento es usando GRPC, ya que la IP del servidor es única y visible, pero este no es el caso con la IP del *Gateway*, ya que tener la IP del gateway depende de que el cliente habilite esta IP para ser vista desde el exterior, lo que no necesariamente va a poder hacerse porque puede involucrar medidas de seguridad tomadas por la empresa a la que Neykos presta servicio. La solución es buscar alternativas de conexión: intervenir o configurar el router de la planta para mapear sus puertos a la IP/puerto

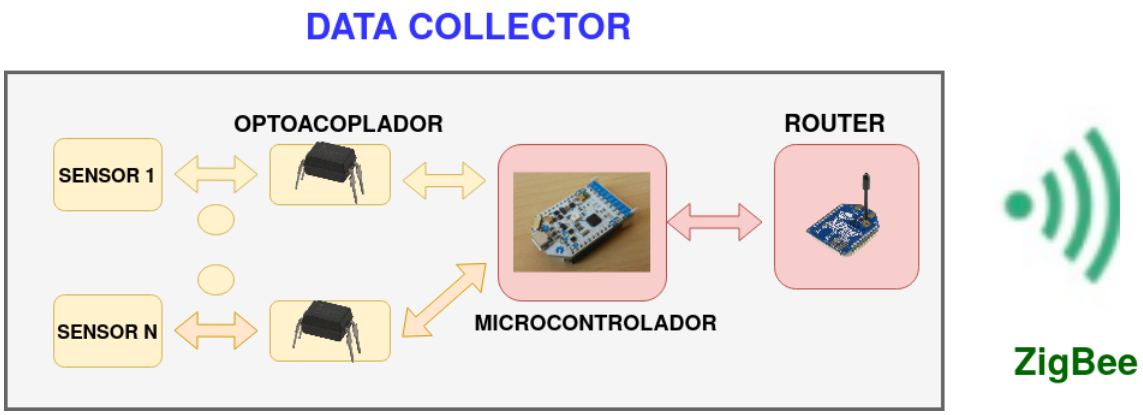


Figura 3.2: Diagrama de los cambios a implementar en el *Data Collector* (en rojo)

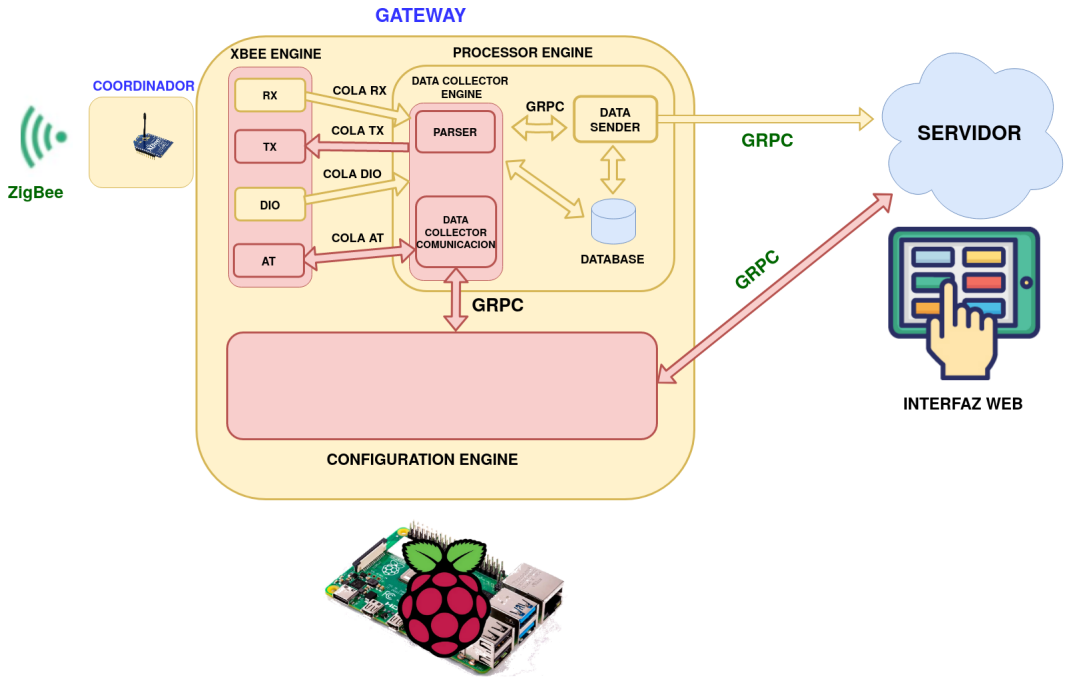


Figura 3.3: Diagrama de los cambios a implementar en el *Gateway* (en rojo)

del *Gateway* es una opción que se ha utilizado, buscar una conexión bidireccional desde el *Gateway* con GRPC es una alternativa utilizada y probada en las siguientes secciones. Este cambio en el diagrama 3.3 se traduce en otra conexión gRPC existente entre el servidor y el *Gateway*, más específicamente en este último, para el *Configuration Engine*.

Para la programación del servidor web desde el lado del sistema de medición, se ejecutarán peticiones dadas por el *Gateway* mediante el uso de código (*backend*), mientras que para la programación del servidor web (*frontend*), se implementará una interfaz web de forma que actue coherentemente con la operación que cada botón señale realizar. Notar que esto no es lo mismo que la compatibilidad con el servidor web, ya que uno opera a nivel de código y el otro a nivel de interfaz gráfica.

3.3.3. Estructura de la Clase DataCollector y de mensajes

Los bloques del *Gateway* en la figura 3.3, son los que se describen a continuación, estando marcados en rojo los que se añaden para constituir el sistema mejorado.

XBee Engine: Está encargado de manejar las comunicaciones con los dispositivos XBee vía coordinador. Solamente gestiona las peticiones del servidor y respuestas de los *Data Collectors* que sean vía XBee y no las que son vía serial.

Data Collector Engine: Desde la información de archivos de configuración, tiene objetos *Data Collector* que manejan las respuestas desde el *Data Collector* en específico hacia el servidor desde alguna petición realizada por este último o los reportes de pulsos que van llegando. La comunicación intraprocesos se hace por medio de colas, con cola RX como la cola que transmite la respuesta de un comando solicitado por el servidor, cola DIO como la cola que se encarga del reporte de pulsos y Cola RX como la cola que envía los comandos recibidos desde *Configuration Manager*. La cola AT es a través de la cual circulan los mensajes de configuración (con comandos AT de XBee), pasando por un bloque de comunicación con *Configuration Engine* denominado *Data Collector Communicator*. El bloque *Parser* parsea los mensajes seriales (UART Rx y Tx), dándoles el formato necesario para continuar avanzando.

Processor Engine: Se encarga de manera general del procesamiento en los sensores e incluye al bloque *Data Collector*. La información de recepción se transforma a una estructura conocida para el resto de los bloques, con tal de asegurar consistencia en la comunicación intraprocesos.

DB Manager: Maneja la comunicación con la base de datos, tal que si la conexión falla o un dato se pierde, lo guarda en la base de datos y envía nuevamente.

Data Sender: Funciona en la vía servidor - *Data Collectors* enviando las peticiones de cualquier parámetro y en la vía *Data Collectors* - servidor enviando el reporte de pulsos o la respuesta de la petición de parámetro, por medio de un cable de par trenzado (UTP) a internet.

Configuration Engine: Recibe desde el servidor los parámetros a consultar, ya sea del microcontrolador o de XBee. Estos parámetros se transmiten a *Data Collector* (el bloque) a través de gRPC, ya sea para la información relacionada con el microcontrolador o con el dispositivo XBee. Los parámetros van por esta misma vía hacia el servidor.

Cada *Data Collector* debe guardar en alguna estructura dentro del procesador, que como ya se ha señalado, es la Raspberry. Adicionalmente, los dispositivos XBee a los que se encuentran conectados los sensores dentro de los *Data Collector* envían las mediciones de las entradas que tengan activadas y configuradas al *Gateway*, por lo que se necesita este colector, además de la redirección de mensajes desde el servidor (como la ejecución de comandos para cambiar una configuración).

Dentro de la información de este *Data Collector* se encuentran todos los sensores asociados a él y un direccionamiento a ellos, que poseen una estructura Sensor según cuál tipo sean: Los hay digitales, analógicos y serial, que funciona en base al protocolo RS-485 (lectores de

energía).

El sistema en esta parte está interconectado, una parte es *DataProcessor* en el servidor, que gestiona y procesa la información guardándola en las clases o enviándola a alguna parte y la otra es el programa *DataCollector*, que utiliza alguna de las librerías anteriores para recibir y enviar los mensajes por medio del protocolo ZigBee. En el proceso de recepción de mensajes por parte del servidor, existe un procesamiento del data para hacerlo más eficiente, considerando exclusivamente lo necesario y acortando la cantidad de bits.

Finalmente, las recepciones y eventos incorrectos se capturan por medio de archivos .log a través del uso de la librería *logging* en *Python*. La estructura desarrollada permite una gestión de mensajes más eficaz y eficiente, a ser utilizada/integrada en el desarrollo final de este trabajo.

3.4. Sistema de Configuración Mejorado

Se programa un sistema básico que considera las interacciones entre los procesos dentro de la Raspberry. A continuación se describe la arquitectura del sistema, cuyo servidor no se considera, herramientas para la comunicación intra-procesos que han sido utilizadas y la interfaz en sí.

Este sistema escalará hacia más ventanas integradas en el sistema general, que operarán de forma paralela con el resto de las funcionalidades, pero la lógica de envío se encuentra establecida. La comunicación con los microcontroladores para envío de configuraciones es otro aspecto a agregar, ya que ella opera de forma serial.

3.4.1. Funcionamiento del sistema de configuración

La interfaz básica en la que se basa esta aplicación tiene la arquitectura de la figura 3.4, consistente en tres motores: XBee (main), *Data Collector* (datacollector) y Configuración (config). Las peticiones, consistentes en comandos XBee y no aún en comandos del firmware subido a los dispositivos asociados, se hacen desde una interfaz gráfica llamada *config* hacia un programa intermedio llamado *DataCollector* que corre en un *thread* las operaciones de cada uno de los *Data Collector*. Luego, la petición viaja a un motor de comunicación con cada dispositivo XBee, llamado *main*. Son dos conexiones distintas: Una es XBee, en que el servidor es *main* y el cliente es *DataCollector*, quien le hace las peticiones y la otra es interfaz, en que *config* es el servidor y *DataCollector* es el cliente, ya que aunque es el servidor quien hace las peticiones al cliente en este caso, el sistema escalado consistiría en n *Data Collectors* y bajo esta lógica, son n clientes y 1 servidor.

El sistema funciona con dos dispositivos XBee que no tienen conectadas entradas a leer, aunque estas se harán en paralelo dentro de la implementación final. El address de cada *Data Collector* no es un dato que se lee desde el archivo de configuración, aunque cada uno se encuentra identificado dentro del motor datacollector a través de un archivo de configuración .yaml que los relaciona, ya que la idea es que el motor de interfaz no tenga datos que el sistema maneje internamente.

La conexión entre los *threads* y la recepción de los mensajes desde la interfaz es dada por una cola asignada a cada *thread* como variable global, de tal forma que el procesamiento avance (en el código, secuencialmente) si la cola tiene algún valor o que quede en *stand-by* si no lo hace. Está la opción de dejar parámetros visibles a través de un motor de base de datos Redis no relacional.

En el motor XBee el mensaje se recibe y se arma de forma que corresponda con una petición que opere bajo la librería en python utilizada, de la misma forma, el mensaje a colocar en una cola que espera a que llegue para avanzar en el procesamiento, es posteriormente transformado a un mensaje que pueda manejar el motor *Data Collector* y de configuración, este último haciendo las operaciones necesarias sobre los botones o informando que la información pedida no se pudo obtener.

- SET envía los cinco parámetros para ser guardados en el *Data Collector* seleccionado, comprobando internamente que ellos se ajusten al formato que deben y pregunta antes qué parámetros debería cambiar comparando los valores de los parámetros a modificar versus los que actualmente son. El sistema luego avisa que la operación está hecha o que los parámetros no cumplen con el formato.
- GET obtiene los cinco parámetros señalados y los guarda en las cajas (cuyos parámetros son modificables). Cada comando sigue el formato del manual de Digi, no el significado de este formato.

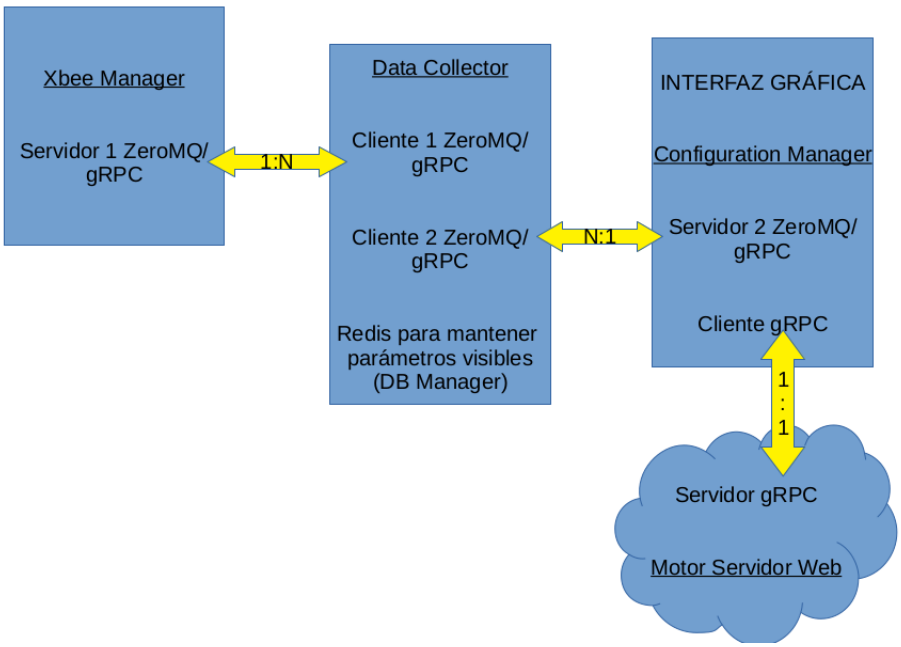


Figura 3.4: Diagrama de la Interfaz a implementar

3.4.2. Herramientas para comunicación de procesos

Se requieren formas de comunicar los procesos entre sí, de forma que la información viaje a través de las distintas partes de la arquitectura desarrollada. A continuación se muestran dos librerías útiles y probadas para este desarrollo.

gRPC consiste en llamadas a procedimiento remoto en código abierto (citar) desarrollado inicialmente para Google, en que este procedimiento es llamado Stub y consiste en el cliente copiando la funcionalidad del servidor, para luego añadir la funcionalidad que desee, por ejemplo, imprimir los campos de una estructura. En la práctica, utiliza un archivo .proto que recibe los servicios, funciones y estructuras, permitiendo generar los códigos necesarios para hacer operar el sistema bajo el sistema de compresión gRPC, que tiene la eficiencia y rapidez de envío como característica principal a su favor.

Por linea de comando, se generan dos archivos desde el .proto, uno de ellos estableciendo la forma de los mensajes y otro, una interfaz entre el servicio de gRPC y las partes involucradas servidor y cliente. Las conexiones también son generadas con la librería gRPC en Python. La etapa servidor-interfaz será desarrollada utilizando gRPC.

ZeroMQ es una librería para la construcción de aplicaciones distribuidas, que no necesita de un *broker* intermedio que gestione los mensajes, haciendo el proceso más rápido. Se utiliza la librería zmq para Python. Además del tipo socket (llamado *pair*), hay más patrones de mensajes, como *publisher-subscriber*, *pipeline* y *request-reply*, este último utilizado y que consiste en varios clientes que hacen request a servidores, una vez el mensaje llega al servidor, este establece una comunicación única con el cliente y le responde sólo a él, siendo necesario que otros clientes esperen a obtener una respuesta del servidor si le han enviado un mensaje de forma simultanea. En otras palabras, este patrón de mensajes tiene un "estado.ªsociado dependiendo de la etapa de comunicación en que se encuentre.

3.4.3. Interfaz web

Se desarrolla una versión de interfaz web utilizando 0mq (tipo PAIR para comunicacion interfaz-*Data Collector* y tipo REQ-REP para comunicación *Data Collector*-motor de XBee) y otra utilizando gRPC, que entre *Data Collector*-motor de XBee debe tener un puente que haga la conexión servidor-cliente, lo que se hace por el uso de colas y una función *stream*, ya que gRPC hace originalmente peticiones desde cliente a servidor. La figura 3.5 consiste en la ventana finalmente desarrollada que tiene las siguientes funcionalidades:

- Elección entre dos dispositivos XBee en la ventana verde superior. Cualquier operación hecha tiene su dependencia al dispositivo específico en base a este botón.
- ADC1 y ADC2 indican las direcciones en hexadecimal del dispositivo indicado. No son leídas del archivo de configuración, sino que son una variable definida en la interfaz.
- Las cajas de llenado son los valores de cada uno de los cinco parámetros presentados. Pueden cambiarse para ser modificadas o leerse desde los dispositivos XBee para ser impresas en cada una de ellas.
- GET y SET son botones pulsados para obtener o modificar los parámetros indicados, respectivamente.
- El texto inferior indica el estado de la operación: Si es exitosa, si falla, el valor de parámetros, etc.

Finalmente, se encuentra implementada la posibilidad de matar los procesos. Ya sea en gRPC o en 0mq, los dos servidores inician una conexión determinada que maneja los procesos y la idea es que al menos en esta implementación, ellos terminen en el momento que el

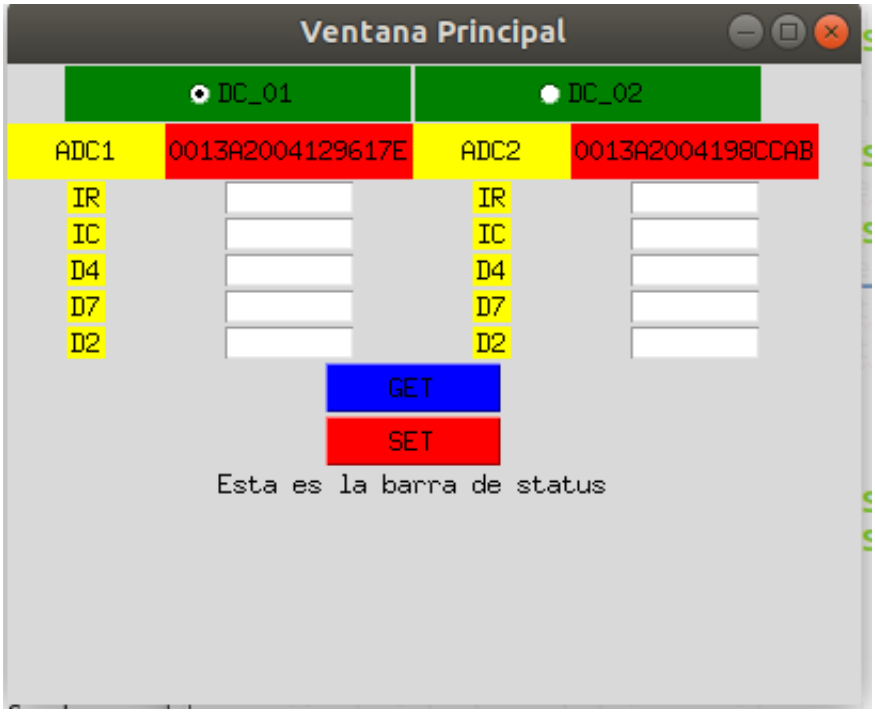


Figura 3.5: Interfaz implementada con tkinter

programa termine. Se medirán los tiempos de respuesta de ambos sistemas desarrollados, con el doble propósito de conocer la importancia del tiempo de esta parte de la arquitectura total y comparar la rapidez en el envío de mensajes para ambas soluciones, esto a través de archivos .pickle que recopilan la información.

3.4.4. Servidor

La conexión entre nube y cliente se hará utilizando *Python* desde el cliente (que es la parte ya desarrollada) y *Ruby* en el servidor. El puente inverso entre servidor y cliente para gRPC entre *DataCollector* y *config* (la interfaz) es el mismo que se va a usar para comunicar con el servidor.

Con este lado del diagrama presentado se completa el ciclo y opera de la siguiente manera:

- Un cliente (desde otra IP) envía a través de la misma interfaz programada los mensajes al servidor.
- El servidor redirige el mensaje al motor *Data Collector* para que procese y gestione el mensaje en el *thread* asignado al dispositivo XBee.
- El mensaje se envía al motor XBee, que realiza la petición al dispositivo físico a través de la librería de comunicación.
- La respuesta vuelve al motor XBee, identificada por un frame_id.
- El motor *Data Collector* procesa el mensaje de respuesta en la estructura de diccionario establecida y lo envía hacia la IP de cliente.
- El mensaje se guarda en las ventanas de parámetros (caso GET) o se muestra un mensaje que indica los parámetros modificados (caso SET).

3.5. Sistema Final

El sistema final se basa en los casos de utilización de esta aplicación, los cuales se incluirán de alguna forma, ya sea por medio de una sola gran ventana o de varias de ellas que respondan a cada una de las rutinas. De los tiempos medidos y complicaciones posibles a partir de las pruebas para cada caso de uso es que se realiza la evaluación a nivel de ahorro económico de esta aplicación.

3.5.1. Casos de uso

Ellos se basan en los estados a medir 3.1. Se debe tomar en cuenta que las demoras en los procesos a realizar son fundamentales, debido a las siguientes razones:

- Hay periodos diferidos, principalmente en la madrugada, considerando la experiencia in situ, en que la planta no estará operativa y en ellos, las configuraciones se harían. En otras palabras, el tiempo disponible es acotado y de no alcanzar, no queda más que esperar hasta el otro día.
- Se deben considerar las horas hombre de un técnico utilizando esta ayuda versus las que gasta un ingeniero actualmente haciendo estos procesos sin la existencia de la aplicación.
- En la situación actual, la planta debe parar su producción para poder generar los cambios, por lo que este tipo de pérdidas debe ser contabilizada, notando que esto sólo puede pasar durante la instalación de dispositivos en la planta y no por alguna configuración distinta. De igual manera, hay un tiempo considerable gastado en detectar el error, ya que se debe ir probando el cambio de parámetros y comparación con los que efectivamente se setearon o analizar estos últimos con tal de entender qué puede estar pasando y todo esto, dentro del periodo en que la planta esta parada, ya que en el caso actual, cualquier tipo de cambio aunque sea minimo se hace con este procedimiento que incluye parar la produccion.
- Es posible programar rutinas que permitan visibilizar a los dispositivos y de esa manera, acortar los tiempos de reacción ante problemas, ya que hoy en día son mediciones de proceso malas las que motivan el cambio en las configuraciones, no dando siquiera a entender de manera correcta qué es exactamente lo que está fallando.
- La no existencia de la aplicación implica la realización de estos procesos de manera desordenada, con un aumento de probabilidad de errores, provenientes de digitar mal un parámetro en un archivo de configuración o en linea de comandos. Es este aspecto, sobre todo, lo que motiva la adición de la interfaz intuitiva y rutinas bien definidas.

Hay distintos casos o escenarios de uso que esta aplicación tiene, considerando que ella funcionará en paralelo al resto de los servicios del sistema completo. Basándose en la experiencia en planta, estos son:

- Agregar nodos para una red en Construcción
- Modificación Masiva
- Corrección Masiva
- Carga de Firmware

Agregar Nodos para una Red en Construcción: Consiste en un sistema ya existente al que se le agrega cada vez un *Data Collector* representado por una dirección XBee con dos o cuatro sensores correctamente seteados. Luego de las conexiones físicas (no tomadas en cuenta en este análisis), se conecta el dispositivo XBee a la red, se setean los parámetros XBee para determinar la forma de los mensajes y las entradas físicas leídas (analógicas o digitales), el firmware del microcontrolador que contiene los campos parámetros que indican el modo de operación y preprocesamiento de los datos leidos antes de ser enviados al gateway y finalmente, el seteo de estos parámetros. Cada uno de los cuatro procesos descritos tiene como paso intermedio su comprobación. Las razones de este procedimiento pueden ser el armado de una nueva red desde cero, el reemplazo de un *Data Collector*, la hipotetica desconfiguracion detras del cambio de un *Gateway* o algun cambio en la linea de produccion/mayor abarque del estado a medir de la planta que implique disponer de un *Data Collector* nuevo para hacer mediciones.

Modificación Masiva: Consiste en un cambio de parámetros ejecutado sobre muchos *Data Collector*, motivado por una necesidad diferente a la falla del sistema de configuración. Se puede subdividir en las siguientes modificaciones:

- **Modificación de microcontrolador:** Se considera una situación en que, luego de una verificación rutinaria, hay una falla que implique una corrección en parámetros del microcontrolador y no del XBee. Ejemplos de esto son los cambios del modo de uso/preprocesamiento, cambios de constantes de tiempo relevantes y uso del sensor de test virtual. Es posible que un cambio anterior de configuración haya alterado uno o mas parámetros por accidente, el dispositivo esta defectuoso y hace cosas que no debe o la configuración seteada tenga otras consecuencias no previstas que provoquen un mal funcionamiento del nodo y por tanto, del sistema. En la interfaz se puede seleccionar el parámetro a cambiar de una lista de opciones o se debería poder enviar la configuración por defecto.
- **Modificación de XBee:** Nuevamente, la situación a considerar es una verificacion rutinaria que detecta una falla que implique realizar correcciones en el dispositivo XBee. Ejemplos de esto pueden ser: El modo API/Transparente, existencia de autoreporte de pulsos y su frecuencia, entradas medidas no correspondientes a las entradas físicas de los dos o cuatro sensores que componen un *Data Collector*. Al estar incluídas estos datos en el archivo .yaml por defecto, estas configuraciones a diferencia del caso de las del microcontrolador son un estandar del sistema y como tal, se planea que la corrección sea automática en un futuro, en lugar de modificable por parte de un usuario. Sin embargo, esta automatización escapa de los alcances del trabajo de tesis aquí presentado y por tanto, la interfaz desarrollada sólo considera la posibilidad de cambio por parte de usuario, ya sea por defecto o cambiando un comando individual.

Corrección Masiva: En este caso, se buscará evidencia tanto en los parámetros del microcontrolador como en el XBee ante evidentes errores de sistema, por lo que las solicitudes de tipo GET se harán simultáneamente a través de la comunicación serial y la que utiliza gRPC, entregada por XBee. Los parámetros obtenidos de esta forma se encuentran relacionados por ejemplo con la temperatura de los dispositivos, voltaje de entrada (y estado de energía) y la señal de keep-alive del microcontrolador, es decir, parámetros enfocados en la visualizacion

de la salud de cada data collector y que en sí no son cambiables, sino que denotan la necesidad de alterar algún parámetro ya sea del microcontrolador o del XBee para mantener la estabilidad del nodo, medida a traves de estos parámetros. En un futuro, la idea es que el sistema corrija por si solos estos errores, pero dentro del alcance del trabajo no es posible debido a que no se tiene aun el conocimiento sobre que acción tomar para corregir, por lo tanto la implementación permite el envío de parametros ya sea en grupos predeterminados, tal como en los casos de corrección ya vistos o de forma individual, eligiendo entre las dos listas de opciones (XBee y microcontrolador).

Carga de firmware Existe la posibilidad de cambio de versión de firmware en el microcontrolador, ya sea porque hay una nueva versión de la placa en sí que funciona mejor (rapidez, menor tasa de error, etc) o porque se desea cambiar la funcionalidad del *Data Collector*, por ejemplo de dos a cuatro entradas o añadir un nuevo modo de uso. Para realizar este cambio, se deshabilita todo el resto de los nodos debido al alto tráfico que este proceso implica, se envía el firmware y tras unos minutos, se pueden volver a habilitar los nodos de dispositivo. Finalmente, es necesario enviar las configuraciones deseadas hacia el microcontrolador, como el modo de uso, tiempos, etc. Esta es la funcionalidad que demoraría mas tiempo en ser ejecutada, debido al peso del archivo firmware. En la interfaz, la versión del firmware es un dato disponible a ser obtenido desde el envío de comandos via serial.

Sobre todo en la Corrección Masiva, La verificacion general de un nodo y red, es la materialización de una ventaja principal de este sistema, que es la capacidad de monitorear el estado de los *Data Collectors*, incluyendo su conexión a la red (al gateway). Considera tres formas de verificación: Parámetros del microcontrolador (comunicación serial), parámetros de comunicación de XBee y parámetros que se encuentran esparcidos en el microcontrolador y en XBee, añadiendo un nivel de dificultad al tener que combinar los dos tipos de comunicación. La idea es que este proceso en una iteración futura del sistema se automatice y obtenga todos los valores cada un cierto periodo de tiempo, comparando con los parámetros realmente seteados en el agregado del nodo e incluso generando un aprendizaje por medio de big data o machine learning de las configuraciones que dan problemas para no establecerlas. Esta accion de monitoreo implica ahorro en tiempo de detección de fallas, ya que en la situacion base de no existencia de este sistema la información se resume a la llegada de pulsos incorrectos o en peor caso, a la no llegada.

La tabla 3.1, a modo de resumen, presenta los casos de uso a analizar en este trabajo de título.

Caso de uso	Descripción
Agregar nodos para una red en Construcción	Agregar varios *Data Collector*, por armado de red nueva o falla/recambio de dispositivo
Modificación masiva	Cambio de algún parámetro relacionado al microcontrolador o XBee, no debido a fallas del sistema de configuración, sino que por tratarse de una alternativa mejor para realizar las mediciones
Correccion Masiva	Falla del sistema de medición, que amerite el cambio de un conjunto de parámetros en muchos *Data Collectors* en que se presente
Carga de Firmware	Nueva versión de la placa, nueva funcionalidad requerida o intercambio de disposición física (pasar de 2 a 4 dispositivos conectados en un *Data Collector*, por ejemplo)

Tabla 3.1: Casos de uso

3.5.2. Relación económica

Tal como se menciona antes, los aspectos a considerar son el horario diferido y acotado en que se pueden hacer cambios importantes, horas hombre y rutinas automáticas de chequeo. El modelo para la situación con sistema es:

$$Costos = C_T + HC_H + PC_P \tag{3.1}$$

En que:

- C_T, Costo de transporte a la planta por parte de los ingenieros de Neykos. Éste se encuentra solamente en el sistema actual, ya que el sistema mejorado implica que los problemas de planta son arreglados por técnicos.
- H, Horas hombre dedicadas a la resolución de la tarea que el caso de uso analizado implica, que se reducen con el sistema mejorado, al no requerir de procesos a mano (en código duro). C_H es el costo de horas hombre, que para el sistema actual es para ingenieros y para el sistema mejorado es para técnicos.
- P, Horas de producción perdidas debido a la intervención de la planta por corrección insitu de ella a la hora de la resolución de la tarea. Ellas se encuentran inevitablemente en

el sistema actual, mientras que se definen como un rango para el sistema mejorado, ya que hay una posibilidad que los técnicos deban, por ejemplo, reemplazar dispositivos e intervenir igualmente la planta. Este rango se encuentra entre 0 (cuando no se interviene la planta) y un costo máximo, en que la intervención tiene probabilidad 1.

3.5.3. Interfaz

El sistema final se desarrolla en el entorno de desarrollo Django para lenguaje Python. Se utilizan los mismos motores de configuración en el sistema preliminar, pero en lugar de existir una interfaz gráfica desde Python, se desarrollan ventanas de usuario operables desde cualquier browser. La figura 3.6 establece la arquitectura del sistema definitivo, cuyos cambios principales son la comunicación por medio de colas simples entre el motor XBee y el motor *Data Collector* y la elección definitiva de gRPC por sobre 0mq.

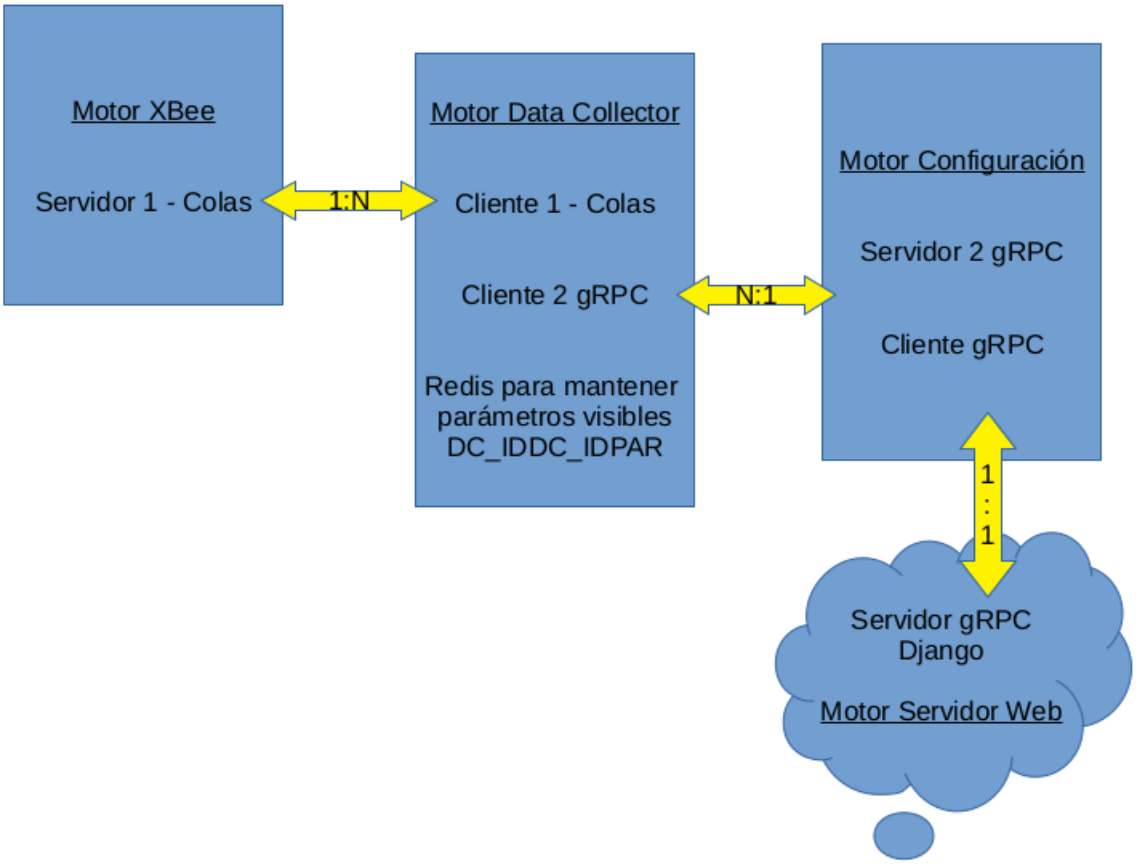


Figura 3.6: Modelo de sistema final

La interfaz consiste en un menú inicial que lleva hacia las distintas funcionalidades del sistema, a través de la cual se desarrollan las pruebas que permiten caracterizar y validar el monitoreo de los colectores de estado. Las funcionalidades de esta interfaz son las visibles en la figura 3.7 y descritas a continuación.

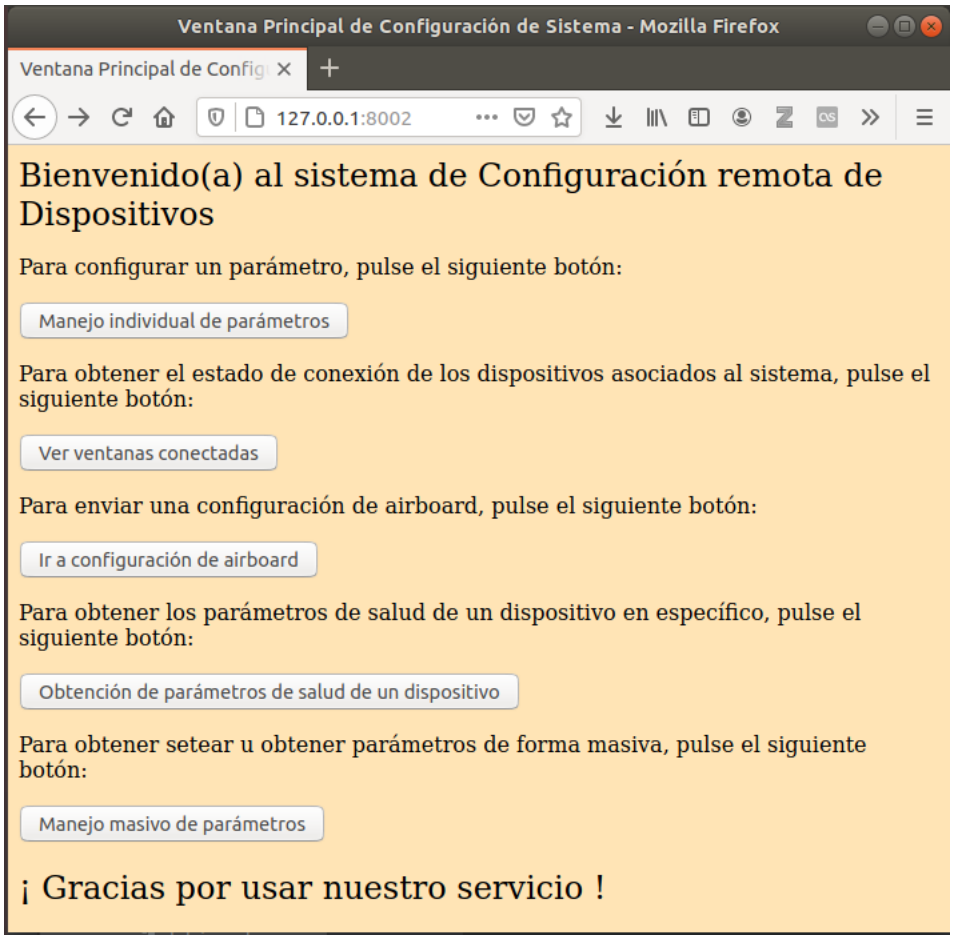


Figura 3.7: Menú inicial de Interfaz web

Operación individual de parámetros: A partir de un *Gateway* y un *Data Collector* manejado por él, se obtiene o se configura el valor de un solo parámetro para XBee o para el microcontrolador. Existen validaciones dentro del formulario con tal de impedir que se envíe un mensaje incoherente a través de los distintos motores. Una de las opciones desplegables es visible en la figura 3.8, en que se ha seleccionado un parámetro XBee y si se desea hacer una configuración del parámetro, se debe llenar el campo "Elija su valor a setearz seguir la instrucción bajo la caja, de otra manera el mensaje no se envía a través de los motores y se muestra una alerta en el browser.

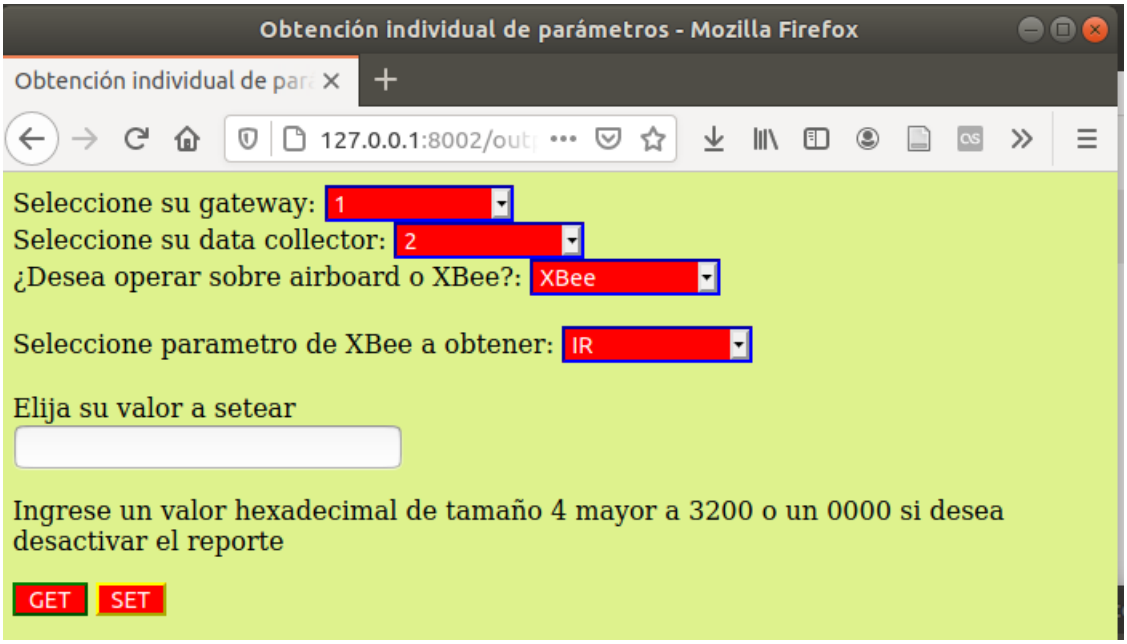


Figura 3.8: Operación individual de parámetros

El resultado de la consulta se muestra como un detalle del mensaje recibido por el motor de configuración. En la figura 3.9 se muestran parámetros como el *Data Collector*, tipo de parámetro y nombre, valor de lectura y tiempo de ejecución del comando, desde el pulsado del botón hasta la aparición de la ventana HTML de respuesta, lo que considera el proceso ida-vuelta del mensaje. Para este ejemplo, se lee un valor 0000 para el comando tasa de muestreo de entradas/salidas (IR) y demora 0.08 segundos.

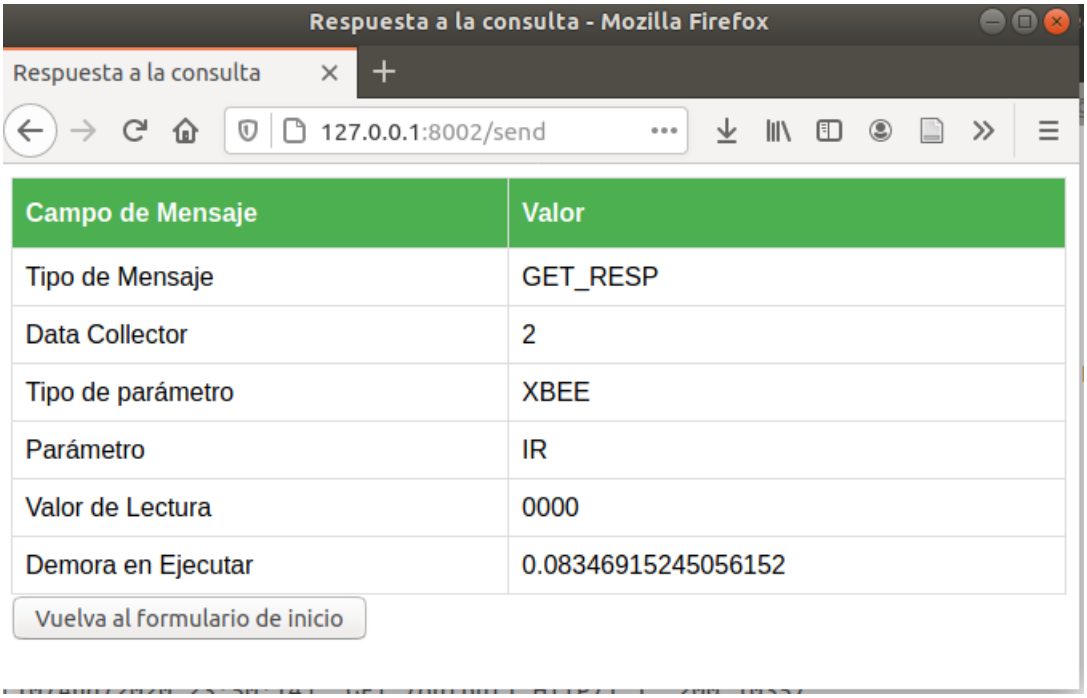


Figura 3.9: Operación individual de parámetros - respuesta

Visualización de dispositivos: Por medio de un botón, el *Gateway* obtiene todos los data collectors conectados a él y los compara con el archivo de configuración asociado, imprimiendo tres posibles estados en una tabla:

- Presente: Un dispositivo operativo en la red y existente en la configuración.
- Ausente: Un dispositivo no operativo en la red y existente en la configuración.
- Fantasma: Un dispositivo operativo en la red y no existente en la configuración.

Se obtiene, tal como señala la figura 3.10, el tiempo de ejecución del proceso desde que se pulsa el botón en la pantalla inicial y es considerable, debido a un *timeout* establecido en el motor de configuración para que todos los dispositivos conectados alcancen a reportarse. En este ejemplo, existen dos *Data Collector* dentro de la red y ambos son los que se encuentran en los archivos de configuración del servidor, por lo que el *Gateway* está conectado exactamente a los dispositivos que debería.

Envío de firmware remoto: Esta funcionalidad no se encuentra completamente implementada, y consiste en la conexión vía serial con un *Data Collector* específico dentro de un *Gateway*, junto a un archivo de configuración a ser enviado de forma remota, adjunto desde el cuadro Examinar de la imagen 3.11. El servidor comprueba que el tamaño de archivo no sea mayor que un límite y que la extensión sea la requerida para un microcontrolador arduino, que es el utilizado en los *Data Collectors* actuales, de otra forma, se envía al browser una ventana de alarma.

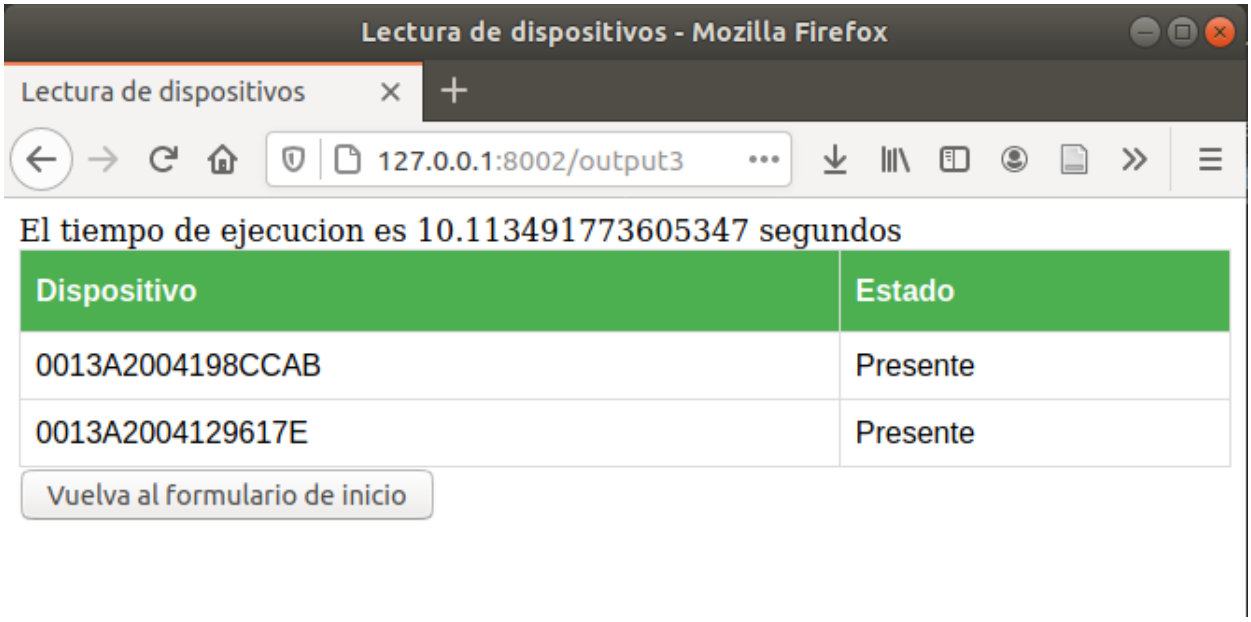


Figura 3.10: Ventana de Visualización de Dispositivos

Obtención de estado de salud: Existen ciertos parámetros que permiten establecer qué tan bien están funcionando los *Data Collectors* y sensores asociados. Es necesario seleccionar un *Data Collector*, tal que al pulsando un botón, se obtienen todos estos valores de forma masiva en una tabla en la misma pantalla en que se enviaron. Un ejemplo de formulario inicial es el de la figura 3.12, mientras que 3.13 muestra el resultado y su tiempo de ejecución asociado, de 0.15 segundos.

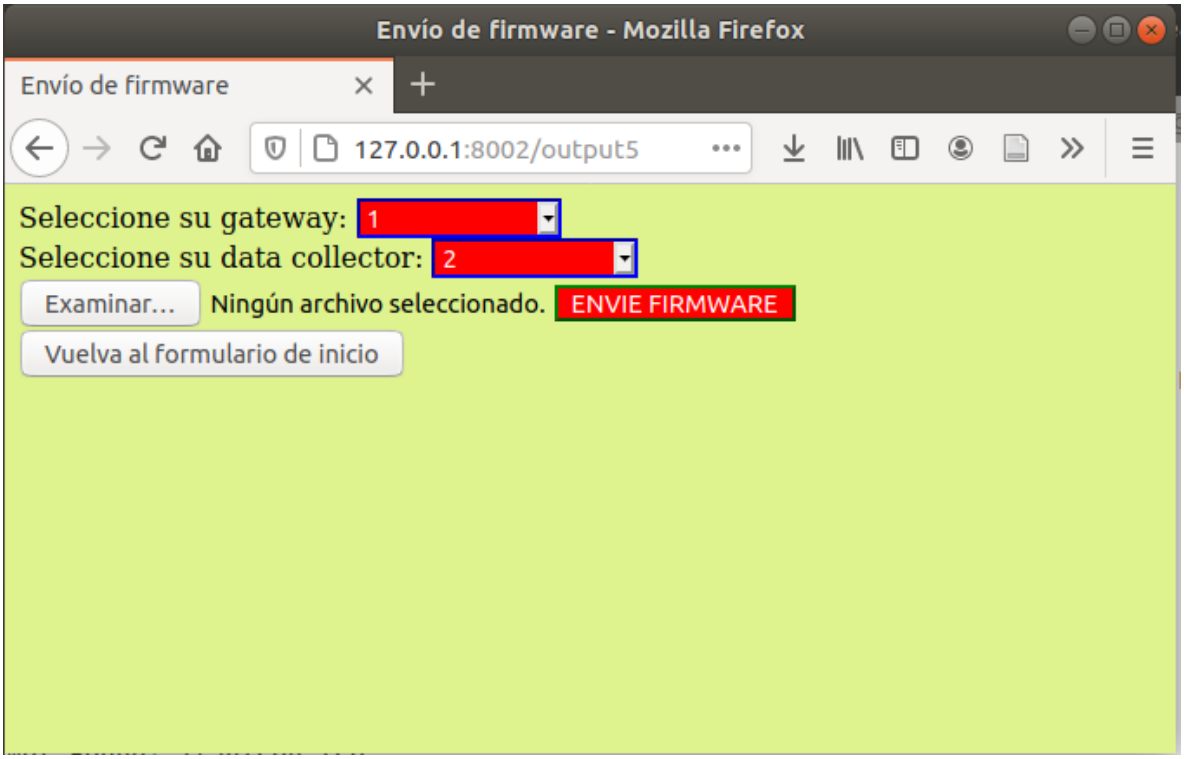


Figura 3.11: Envío de firmware remoto

Operación masiva de parámetros: En esta ventana, se puede elegir configurar u obtener parámetros masivamente y a selección del usuario a través de tickets, seleccionando el *Data Collector* para el caso de GET (en la figura 3.14) o a través de cajas de texto llenables a través de SET (en la figura 3.15) con la descripción, de tal forma que los parámetros que no se desee cambiar se dejen en blanco para SET y los que no se quiera leer, se dejen sin ticket para GET. Es necesario notar que no todos los parámetros escogidos se pueden configurar, por ejemplo, el de voltaje de alimentación del dispositivo XBee es de sólo lectura.

En comparación con la solicitud de parámetros individuales, en este caso se está pidiendo mucha más información de una vez, por lo que la respuesta a modo de tabla no se compone de gran parte de las variables del mensaje completo, sino que del tipo de parámetro, parámetro y valor. En el ejemplo visible en la figura 3.16, se tickean tres parámetros tipo XBee para el *Data Collector* con ID 2.

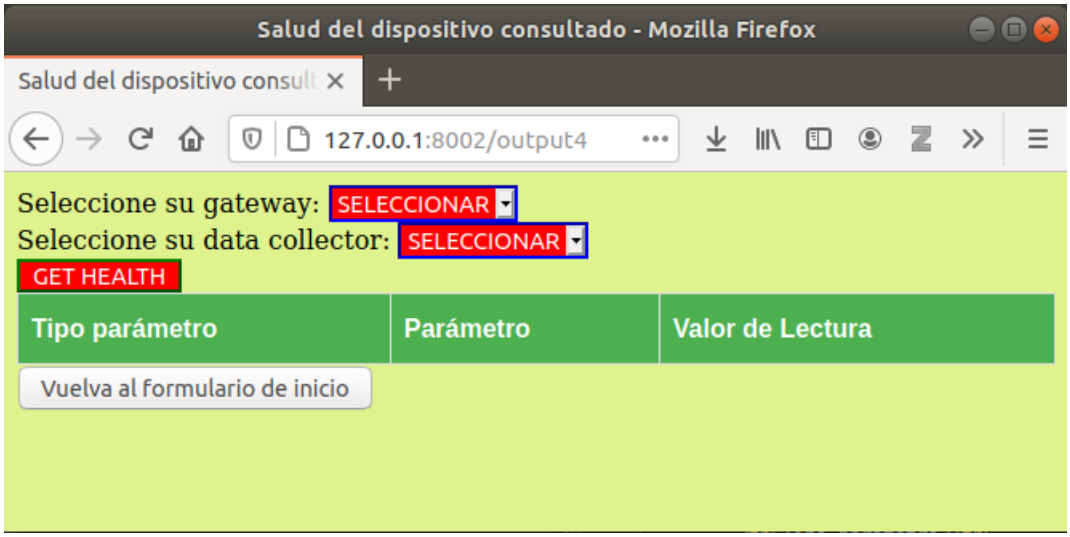


Figura 3.12: Obtención de estado de salud: Pre-consulta

El tiempo de ejecución fue de 0.17 segundos, aproximadamente 3 veces el tiempo de obtención de un sólo parámetro. Se podría pensar que la ventaja de esta forma de operación va en minimizar los tiempos de consulta por parámetro en relación a establecer una comunicación paralela, sin embargo, este no es el caso y el ahorro de tiempo proviene de la conveniencia de hacer varias solicitudes en un sólo formulario.

Los ejemplos mencionados, además de permitir ilustrar el funcionamiento de esta interfaz web, entrega indicios de los tiempos de ejecución para cada una de las formas de uso que la componen, adelantando en cierta forma los resultados del desarrollo de esta memoria.

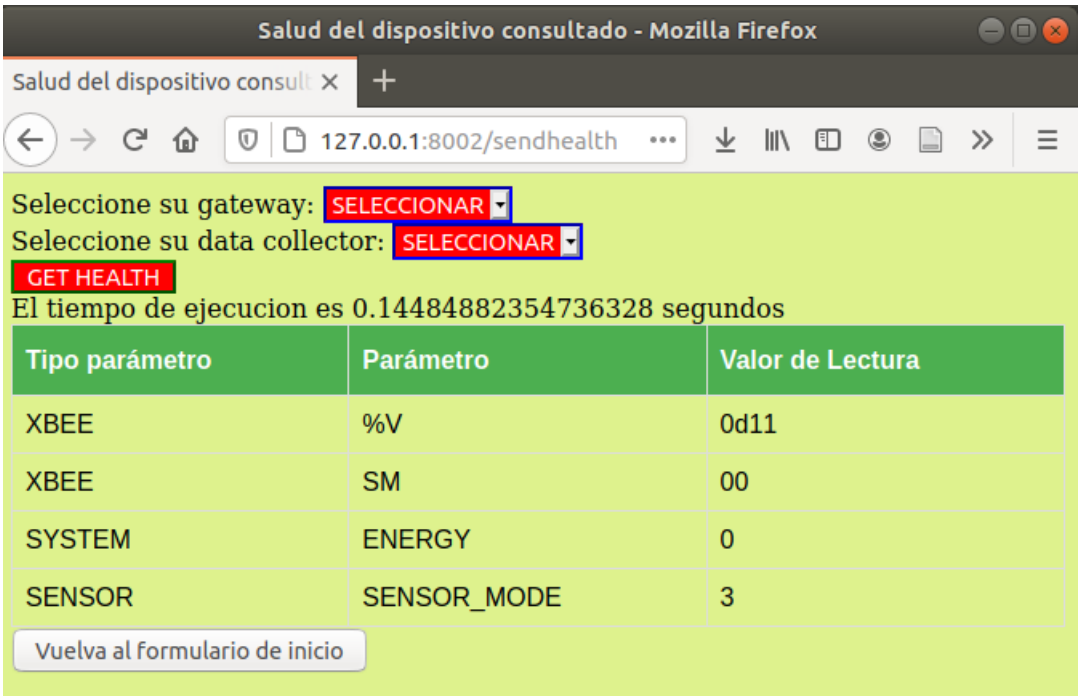


Figura 3.13: Obtención de estado de salud: Post-consulta

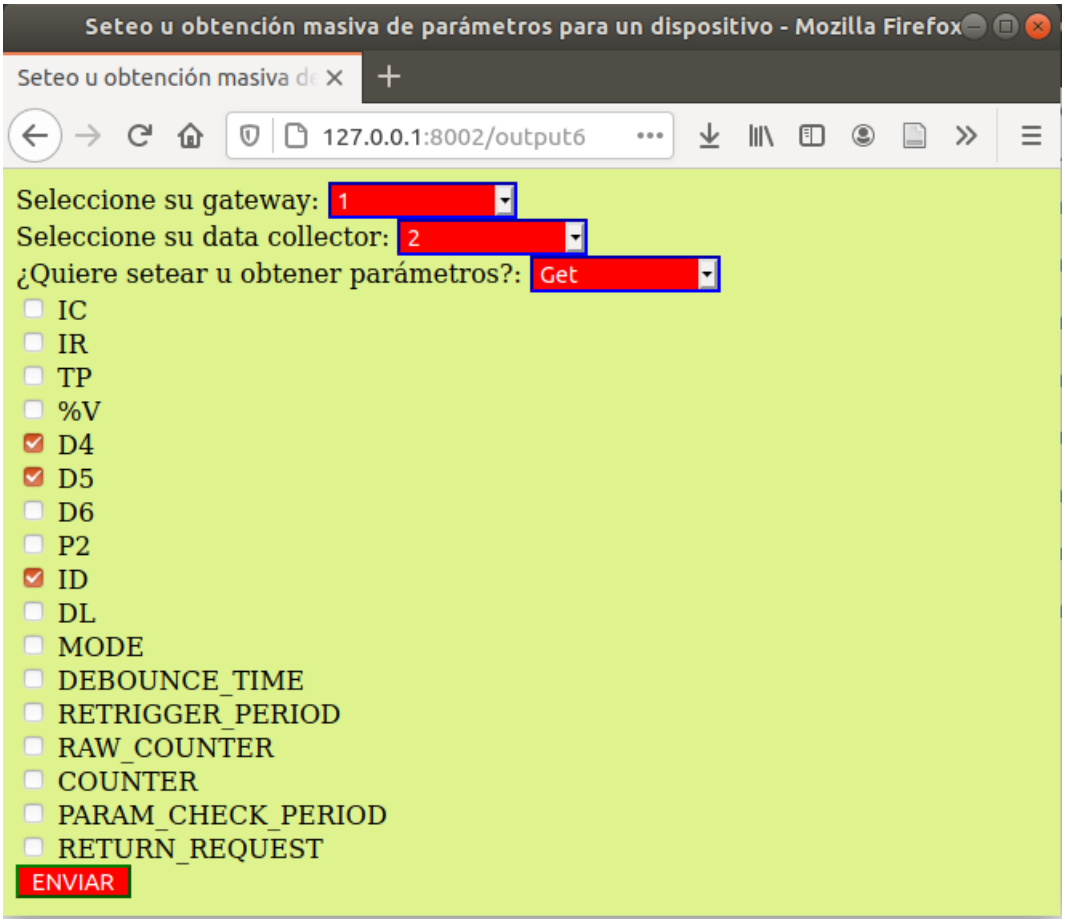


Figura 3.14: Operación masiva de parámetros: Submenú GET

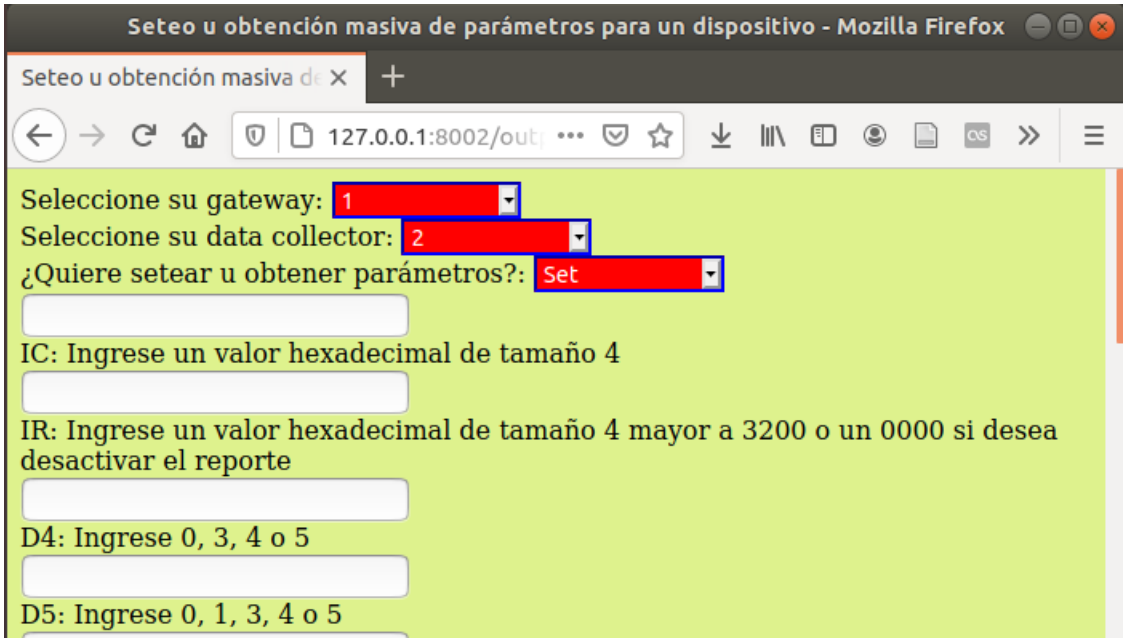


Figura 3.15: Operación masiva de parámetros: Submenú SET

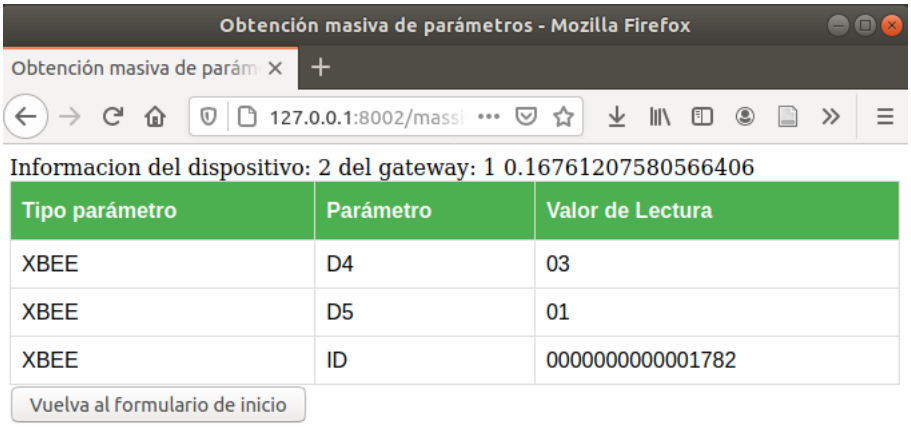


Figura 3.16: Operación masiva de parámetros

Capítulo 4

Resultados

4.1. Librería de uso

La dificultad de la compilación de las librerías en lenguaje C provoca que se hagan las pruebas solamente con las que utilizan lenguaje Python. Resumiendo:

- La librería oficial de Digi [25] tiene mayor soporte en Internet, en cuanto a su utilización, documentación y control de errores. Sin embargo, no funciona correctamente a priori y no es la que se está utilizando actualmente en la empresa para captar información de los sensores de forma remota hacia la Raspberry PI.
- La librería no oficial/alternativa [28] es desarrollada por aficionados y no tiene mucho soporte, sin embargo, permite una operación más específica (funciona en un lenguaje más bajo que la anterior), lo que la hace más simple de manejar y actualmente, la que se encuentra funcionando.

Las librerías deben realizar peticiones lo más rápido posible y con una baja tasa de errores, dado que el desarrollo de la herramienta de configuración tiene como foco la realización de peticiones masivas. Se realizan pruebas con las dos librerías de Python, con el doble objetivo de identificar cuál funciona mejor en base a su latencia y menor cantidad de mensajes perdidos y familiarizarse con el sistema en general para entender la lógica de uso de los distintos elementos, ahorrando una cantidad de tiempo considerable al momento de la implementación final. Las pruebas realizadas son las siguientes:

- Prueba 1: Se envían comandos de lectura a un dispositivo XBee. Se utilizan 20 comandos y 50 iteraciones.
- Prueba 2: Se envían comandos de escritura a un dispositivo XBee. Se utilizan 5 comandos y 50 iteraciones.
- Prueba 3: Se envian comandos microcontrolador a través de un dispositivo microcontrolador. Se utilizan 10 de ellos en 200 iteraciones.

Cada comando se identificó con un frame_id (letra) determinado, de manera que en la función de *callback* se pudiese captar el tiempo de demora y el *status* del mismo frame_id, a modo de rastreo. Los comandos utilizados son: RP, D2, D3, D4, D5, IR, ZS, CH, NC, SH,

SL, CI, ID, SD, II, AP, NJ, IC, D1, VR. Ellos, identificados de manera respectiva con un frame_id de la A a la T. Ambas librerías tienen errores: La no oficial tiene tiempos negativos, mientras que la oficial tiene *status* distinto de cero. La no oficial tiene 0 a 2 de estos errores entre 50 intentos, para cada comando, mientras que la oficial tiene 1 error en general de *status*. La tabla 4.1 indica una comparación entre velocidades (tiempo de recepción menos tiempo de envío) y errores para ambas librerías. Los errores en la librería oficial suelen ser menos que en la librería no oficial y su velocidad es mayor.

-	Oficial	No oficial	Indiferente
Velocidad	17	3	0
Errores	5	8	7

Tabla 4.1: Resultados prueba de Lectura en XBee

Bajo la perspectiva de lectura, la librería oficial es preferente por sobre la no oficial.

Cada comando se identificó con un frame_id (letra) determinado, de manera que en la función de *callback* se pudiese captar el tiempo de demora y el *status* del mismo frame_id, a modo de rastreo. Los comandos utilizados son: NH, D3, D5, D2, IR. Ellos, identificados de manera respectiva con un frame_id de la A a la E. A diferencia del caso anterior, no hay tiempos negativos, pero si *status* distintos de cero en la librería oficial.

La tabla 4.2 indica una comparación entre velocidades (tiempo de recepcion menos tiempo de envio) y errores para ambas librerías, en escritura. Puede verse que la velocidad es mejor para la librería no oficial y los errores no existen. D

-	Oficial	No oficial	Indiferente
Velocidad	0	5	0
Errores	0	0	5

Tabla 4.2: Resultados prueba de Escritura en XBee

Bajo la perspectiva de escritura, la librería no oficial es preferente por sobre la oficial, contrario a la perspectiva de lectura.

Se enumeran de la A a la J los 10 comandos utilizados en un frame_id, que son: GET0, GET1, GET2, GET3, GET4, GET5, GET6, GET7, GETTM, VERSION. Los resultados se enuncian en la tabla 4.3, en que la librería no oficial recibe respuestas con más rapidez y tiene menos errores. De hecho, no se reportan errores en la librería no oficial, mientras que en la oficial se obtienen 2 status distintos de cero entre 100 iteraciones.

-	Oficial	No oficial	Indiferente
Velocidad	0	10	0
Errores	0	10	0

Tabla 4.3: Resultados prueba de lectura en microcontrolador

Bajo esta perspectiva, la prueba de envío de mensajes al microcontrolador preferencia el uso de la librería no oficial sobre la oficial.

Como conclusión, se opta por utilizar ambas librerías como motor de comunicación dando esto como entrada en la ejecución del programa principal, ya que se reportan mejores resultados en una librería u otra según la funcionalidad que se esté utilizando, siendo todas estas funcionalidades parte del programa a desarrollar. Se procurará elaborar todas las partes siguientes con las dos librerías, a menos que se indique lo contrario.

4.2. Interfaz 0mq versus gRPC

En base a los dos sistemas desarrollados, se prueba midiendo tiempos de las peticiones con tal de vislumbrar cuál es más rápido y elegir cuál utilizar, aunque este no es el único parámetro a ver, ya que el uso de 0mq es mucho más restrictivo que el de gRPC (la capacidad de lidiar con varias peticiones hechas simultaneamente al servidor de forma paralela es inexistente en 0mq, a diferencia de gRPC en que el cliente copia una versión del servidor y ejecuta). Los resultados luego de 25 intentos pueden verse en la tabla 4.4.

Mediciones		
Parámetro	Promedio (s)	Desviacion Estandar (s)
GET - 0mq	0.3267	0.0569
GET - gRPC	0.3105	0.0527
SET - 0mq	0.3546	0.0889
SET - gRPC	0.5067	0.0667

Tabla 4.4: gRPC vs 0mq en la interfaz

En base a estos resultados, no se ve diferencia significativa entre los tiempos dados por el uso de 0mq con los de gRPC, considerando que la configuración no tiene por qué ser tan rápida y que la complejidad de las rutinas es similar a la prueba realizada. Sin embargo, gRPC tiene a favor la ejecución remota, que permite consultas simultáneas de muchos clientes a un mismo servidor de forma paralela, por lo que se opta por usar gRPC para comunicación intraprocesos.

Por otro lado, el tiempo extra en la comunicación de procesos no es significativo como para entorpecer los tiempos de viaje ni la configuración en sí. El tiempo es de aproximadamente 0.05 en el motor XBee, por lo que este motor es un 15 a 20 por ciento del total de tiempo de ejecución. La influencia del servidor, es decir, el uso de otra IP para hacer las peticiones desde esta misma interfaz, debiese tener una injerencia también en los tiempos de ejecucion, sin embargo, ésto ya puede dar una idea del rango entre envío y recepción de varios comandos de configuración y obtención/monitoreo.

4.3. Sistema General

Se realizan pruebas con tres dispositivos conectados a un *Gateway*, con el sistema de configuración corriendo en un servidor de test de la empresa Neykos. De los tres dispositivos,

uno se encuentra conectado a un microcontrolador y los otros dos no, por lo cual uno de ellos puede recibir comandos del microcontrolador y XBee y los otros dos pueden recibir comandos sólo de XBee.

4.3.1. Primera Prueba: Comandos Individuales

La primera prueba consiste en enviar de manera aleatoria 20 comandos GET y 20 comandos SET desde la ventana de Configuraciones Individuales, para medir el tiempo que el sistema demora en obtener o configurar un sólo comando. Los mensajes GET y SET de XBee envían una respuesta de vuelta al usuario, por lo que este tiempo se puede medir, mientras que en el caso de los comandos de microcontrolador llega una respuesta sólo para los mensajes GET porque esto no se encuentra configurado para los mensajes SET. La tabla 4.5 denota los tiempos máximo y medio obtenidos para cada caso, junto a su desviación estándar.

Mediciones			
Parámetro	Promedio (s)	Valor Máximo (s)	Desviacion Estandar (s)
GET - XBee	0.4628	1.1185	0.2654
GET - Microcontrolador	0.4900	0.7416	0.2280
SET - XBee	0.5815	1.1137	0.3235

Tabla 4.5: Tiempo de Respuesta por tipo de Comando

Estos resultados pueden compararse con los obtenidos para GET y SET en la interfaz inicial para comandos XBee, considerando las mediciones para el método gRPC que como ya se ha señalado es el utilizado para este desarrollo final. Puede afirmarse que:

- Los tiempos para GET y SET en comandos XBee son mayores con la interfaz en el servidor. GET para la interfaz inicial muestra un tiempo de 0.3105 [s], registrando un aumento en un 49 % y SET muestra un tiempo de 0.5067 [s], registrando un aumento en un 15 %. El aumento es esperable, ya que la interfaz inicial consiste en un sistema local, mientras que la interfaz final es un sistema montado en un servidor.
- La desviación estándar es mucho mayor para la interfaz final, lo que sumado a tiempos promedio mayores, implica que la red servidor-*Gateway* puede presentar atrasos que lleven a los tiempos máximos de respuesta registrados de 1.1 [s].
- Los tiempos de respuesta para XBee, tanto para la interfaz inicial como para la interfaz definitiva, son mayores para los comandos SET que para los comandos GET. Un 63 % más para la interfaz inicial y un 26 % más para la interfaz definitiva, reduciéndose el cambio para esta última ya que la red Servidor-*Gateway* suma los mismos tiempos para ambos comandos.
- Los tiempos de respuesta para comandos GET de microcontrolador tienen un rango similar que el de los comandos GET *XBee*. A su vez, el efecto de esta desviación estándar para todos los casos no permite intentar probar si los tiempos de comandos GET y SET son suficientemente distintos para cada uno de los tres dispositivos XBee utilizados, por lo cual se asumirá que ellos son similares.

4.3.2. Segunda Prueba: Lectura de Dispositivos en la Red

La segunda prueba es breve y consiste en ejecutar el Lector de Red de la interfaz actual 10 veces, midiendo los tiempos. Esta funcionalidad tiene en el motor XBee un *timeout* de 10 [s] y corresponde a un GET de tipo XBee que recibe más de una respuesta, por lo que 10 [s] es el tiempo mínimo que la lectura podría medir. La figura 4.6 presenta los resultados obtenidos luego de la ejecución.

Mediciones	
Medida	Valor (s)
Tiempo Promedio	10.4158
Tiempo Máximo	10.8663
Tiempo Mínimo	10.2315
Desviación Estándar	0.2188

Tabla 4.6: Tiempo de Respuesta para Lectura de Dispositivos

De los resultados de la tabla 4.6 puede señalarse que:

- Tanto el valor promedio como la desviación estándar descontando los tiempos de *timeout* son similares a una instrucción GET, que registra valores respectivos de 0.4628 [s] y 0.2654 [s]. Esto se encuentra dentro de lo esperable, ya que (reiterando) se trata de una instrucción GET.
- Se registra una diferencia de hasta casi 4 veces entre el tiempo mínimo y el tiempo máximo, descontando los 10 segundos de *timeout*.

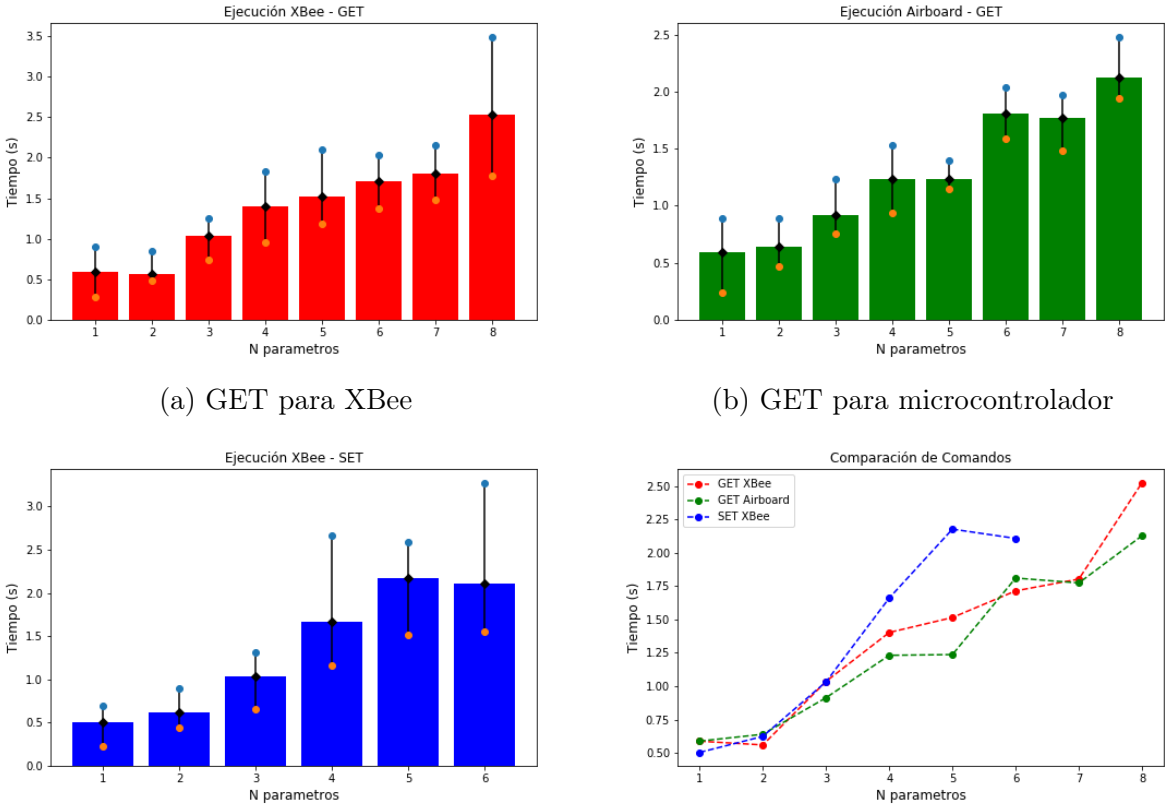
4.3.3. Tercera Prueba: Comandos Masivos

La tercera prueba consiste en un análisis de sensibilidad sobre cada uno de los tipos de comando (GET para XBee, GET para microcontrolador y SET para XBee). Se va aumentando de uno en uno los comandos enviados de manera simultanea para cada tipo y se mide la respuesta del servidor, considerando que en la interfaz definitiva hay 6 parámetros que pueden ser introducidos con SET, 8 parámetros de microcontrolador con GET y 8 parámetros XBee con GET. Se intenta 5 veces por cada valor, a través de una petición realizada desde la función Comandos Masivos de la interfaz definitiva.

Los resultados pueden verse en la figura 4.1, en que las barras sobre las figuras superiores e inferior izquierda representan el tiempo promedio y las líneas negras sobre ella, el tiempo mayor entre 5 (en azul) y menor entre 5 (amarillo). La cuarta figura representa los valores medios para cada tipo de comando. De estos resultados puede verse que:

- Nuevamente, los comandos SET demoran más comparativamente en ejecutarse que los comandos GET, ya sea de XBee o de microcontrolador.
- La existencia de tiempos individuales con un valor de hasta 2 veces más que los tiempos promedio para cada tipo de Comando puede tener un efecto que sumado entrega una dispersión mayor en una consulta con muchos parámetros simultaneos, como puede verse en las figuras 4.1a y 4.1c para 8 y 6 parámetros, respectivamente.

- Tal como lo que ocurre para el caso individual, no hay evidencia en la diferencia de tiempos entre GET de tipo XBee y GET DE tipo microcontrolador, visibles de forma respectiva como la linea roja y verde en la figura 4.1d.
- Aunque existe una tendencia creciente en el tiempo versus la cantidad de parámetros en los tres tipos de comandos, no puede hablarse de linealidad creciente. Esta linealidad se vería mejor representada una vez la cantidad de pruebas aumente, ya que de esta manera las mediciones extremas pesarían menos en el valor promedio representado por las barras.



(a) GET para XBee

(b) GET para microcontrolador

(c) SET para XBee

(d) Mediciones Comparadas

Figura 4.1: Resultados de Tercera Prueba. El gráfico 4.1a conforma los tiempos para Comandos GET de tipo XBee, 4.1b conforma los tiempos para Comandos GET de tipo microcontrolador, 4.1c conforma los tiempos para Comandos SET de tipo XBee y 4.1d la comparación de tiempos medios entre todos los anteriores

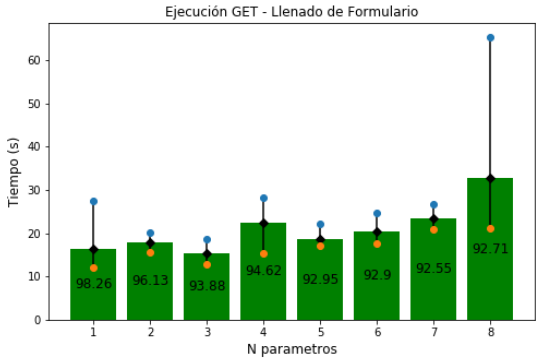
4.3.4. Cuarta Prueba: Llenado de Formulario

La cuarta prueba mide el tiempo en que un potencial usuario configure varios parámetros de una vez, es decir, el tiempo en llenar el formulario para la función Comandos Masivos de la interfaz definitiva. Es cierto que los tiempos de ejecución para cada tipo de comando tienen gran dispersión, pero ¿Son relevantes estos tiempos comparados con lo que demora un usuario en hacer su petición?.

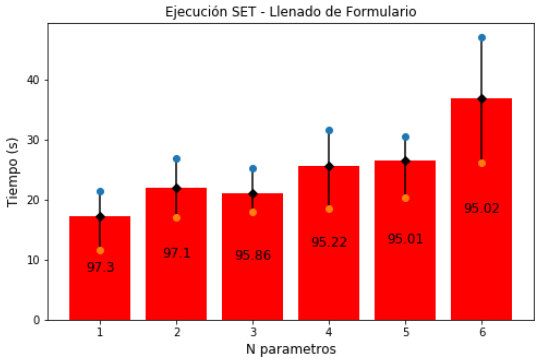
Tal como en la tercera prueba, el experimento consiste en un análisis de sensibilidad con 5 intentos por cada número de parámetros, esta vez no haciendo distinción entre comandos GET de XBee y comandos GET de microcontrolador, ya que se ha visto que estos tiempos no tienen suficiente diferencia. Se denominan entonces comandos GET y comandos SET, simplemente.

Los resultados para SET y GET pueden verse en la figura 4.2. Puede observarse de ellos que:

- El tiempo es levemente mayor (22[s] versus 18[s] aproximadamente, viendo las figuras



(a) Comandos GET



(b) Comandos SET

Figura 4.2: Resultados de Llenado de Formulario. Las barras denotan el tiempo de ejecución, con valores máximos y mínimos señalados entre 5 intentos. El número es el porcentaje del proceso de Llenado con respecto al tiempo total (Llenado + Ejecución del Sistema, ya reportado)

4.2a y 4.2b) para los comandos SET que para los comandos GET. Este resultado se esperaba, aunque más pronunciado, debido a la naturaleza del formulario: Para GET se tickean los valores deseados, mientras que para SET se deben ingresar en cajas de texto.

- Hay una desviación estándar importante debido a la distracción o equivocación del usuario a la hora de llenar el formulario, que puede llevar a tiempos de hasta 1 minuto (en una prueba GET en 4.2a, para 8 parámetros).
- Para cualquier cantidad de parámetros y para ambos tipos de comandos GET y SET, el porcentaje de tiempo dedicado al llenado del formulario está en promedio sobre el 90 %, es decir, este tiempo es de un orden de magnitud mayor al tiempo de respuesta.
- Al realizar este test muchas veces, hay una familiarización con las posiciones de los comandos y una memorización de las rutinas a realizar con teclado y mouse, por lo que los tiempos reportados están por debajo de los que un técnico reportaría, tomando en cuenta la diversidad de las tareas realizadas por él durante una jornada de trabajo.

Los resultados presentados son válidos considerando un estado de concentración ideal por parte del usuario, por lo que los porcentajes reportados suben aún más y la importancia relativa de los tiempos de ejecución del sistema y sus fluctuaciones con respecto al tiempo de demora del usuario es aún más baja de lo reportado. Es la suma de los dos tiempos reportados y principalmente, este tiempo de Usuario, la justificación para los tiempos supuestos en la sección siguiente: Evaluación de los casos de uso.

4.4. Evaluación de los casos de uso

La comparación se realiza entre la situación actual, en que no existe el sistema de configuración dentro de PMSystem y la situación mejorada, en que este sistema existe.

Como supuestos, se toma en cuenta como bien se ha mencionado (3.5.1) que hay tres aspectos principales para la evaluación de los casos de uso: Costos de transporte por acudir a

Parámetro	Valor
Litro de Bencina (CLP)	750
Rendimiento (Km/Litro)	12
Ganancia de planta (CLP/Hora)	45.000
Sueldo de Ingeniero (CLP)	2.000.000
Sueldo de Técnico (CLP)	800.000

Tabla 4.7: Parámetros transversales a todos los ejemplos de uso

la planta, costo de mano de obra respecto a la corrección de estos errores (tiempo que les toma a los trabajadores) y pérdidas de producción, relacionadas con el parado de la planta para corregir los errores. Los costos de transporte en la situación mejorada no corren, mientras que la planta idealmente con este sistema no debiese parar, excepto ocurran externalidades fuera del alcance del sistema mismo, razón por la cual la situación mejorada toma un rango de costos, entre un valor optimista y uno pesimista, considerando que para un valor realista o probable es necesaria más información a modo de historial del comportamiento de los *Data Collector* que hoy no se tiene.

Para el costo de transporte, se toma el valor de bencina como 750 pesos chilenos [37] y el ingeniero que debe asistir a la fábrica lo hace con vehículo, cuyo costo de bencina es pagado por la empresa. Un automóvil comun rinde 12 Km/Litro [38] y la distancia varía en cada caso, pudiendo ser de 500 Km a una fábrica fuera de Santiago o 50 Km para una que se encuentre fuera del área metropolitana.

Para simplificar y estandarizar la situación, en todos los casos se trata de plantas de yogurt similares a las descritas en [39], en que se producen 468 bolsas de yogurt por hora y por cada una, la ganancia es de 100 CLP por cada uno, esta ganancia por hora es de 46.800 CLP. Para aproximar, se supondrá 45.000 CLP por hora.

Se toma el sueldo de un técnico especializado como 800.000 de pesos chilenos y el de un ingeniero como 2.000.000 de pesos chilenos [40], ambos valores aproximados. Es importante esta distinción en el aspecto de evaluación relacionado al costo de la mano de obra, ya que la situación mejorada implica el paso de la resolución de problemas desde los ingenieros de la empresa hacia los técnicos de planta.

En resumen, la tabla 4.7 permite ver los parámetros considerados para todos los casos de uso.

- **Agregado de nodos al sistema:** Se trata de 10 dispositivos en otra región a 500 Km y se deben configurar al inicio dentro de una red, pero los dispositivos se encuentran en planta. Con el sistema actual, un ingeniero demora alrededor de 2.5 horas en configurar un dispositivo y con el sistema mejorado, 10 minutos para un técnico, incluyendo la comprobación de todos los aspectos necesarios, que son: Inclusión en la red ZigBee, envio de *firmware*, configuraciones de parámetros de XBee y microcontrolador y el funcionamiento completo.

  Para el sistema mejorado, nunca habría necesidad por parte de un técnico de acudir a la planta a configurar dispositivos, por lo tanto, el costo en el mejorado actual de

Parámetro	Situación Actual	Situación Mejorada
Tiempo de Corrección (Horas)	25	5/3
Costo por Transporte (CLP)	60.000	0
Costo por Mano de Obra (CLP)	312.500	8.300
Costo por Producción (CLP)	1.125.000	75.000
Costo Total (CLP)	1.497.500	83.300
Reducción (%)	-	94.4

Tabla 4.8: Ejemplo de uso: Agregado de nodos al sistema

transporte es 0 CLP. En cuanto al sistema actual, el cálculo es el siguiente:

$$Litros_{bencina} = \frac{500[Km]}{12[Km/Litro]} \approx 42[Litros] \tag{4.1}$$

$$Costo_{transporte} = 42[Litros] * 750[CLP/Litro] = 31{,}500[CLP] \tag{4.2}$$

En cuanto al costo por mano de obra, el tiempo total de configuración para el sistema actual es de 25 horas, mientras que en el sistema mejorado, de 1 hora y 40 minutos. Si el sueldo de un ingeniero y de un técnico por 40 horas semanales son de 2.000.000 CLP (12500 [CLP/Hora]) y 800.000 CLP (5000 [CLP/Hora]), respectivamente. El cálculo de costos es el siguiente, con CostoA como el actual y CostoM como el mejorado:

$$CostoA_{manoobra} = 12{,}500[CLP/Hora] * 25[Horas] = 312{,}500[CLP] \tag{4.3}$$

$$CostoM_{manoobra} = 5{,}000[CLP/Hora] * 5/3[Horas] \approx= 8{,}300[CLP] \tag{4.4}$$

La producción en el sistema mejorado puede parar debido a dispositivos que se detectan como defectuosos y deben cambiarse por otros, por falla incorregible de cualquiera de las etapas. No se sabe cuál es la probabilidad que ocurra esto, por lo tanto, se toma una pérdida de producción por intervención (física) del sistema entre 0 y 100 por ciento del tiempo, con este último peor caso, el costo es:

$$CostoM_{produccion} = 5/3[Horas] * 45{,}000[CLP/Horas] \approx 75{,}000[CLP] \tag{4.5}$$

Por otro lado, el sistema actual siempre tiene estas pérdidas, porque el sistema debe intervenirse para hacer cambios. Es necesario notar que las pérdidas por producción no consideran tiempos muertos adicionales como el asociado a cumplimiento de normas de seguridad y de permisos administrativos.

$$CostoA_{produccion} = 25[Horas] * 45{,}000[CLP/Horas] \approx 1{,}125{,}000[CLP] \tag{4.6}$$

La tabla 4.8 describe un detalle de este ejemplo de uso, en que la mejora con el sistema de configuración es de un 94.4 %.

- **Configuración Masiva de un parámetro:** Este ejemplo se trata de configurar de forma masiva un par de parámetros en una planta de alrededor de 100 dispositivos, en que ella se encuentra dentro de la región metropolitana (a 50 [Km] del lugar de los

ingenieros). Como referencia, por cada dispositivo la demora es de 30 minutos en el sistema actual para configurar y comprobar un par de parámetros, mientras que son 2 minutos por cada dispositivo en el sistema mejorado, principalmente por la digitación y comparación visual de los parámetros del dispositivo por parte del usuario dentro de la interfaz.

En cuanto a los costos de transporte, son 0 para la situación de sistema mejorado, ya que aunque fallen los dispositivos, basta con dar instrucciones a los técnicos, mientras que para el sistema actual el costo es:

$$Litros_{bencina} = \frac{50[Km]}{12[Km/Litro]} \approx 4(5)[Litros] \tag{4.7}$$

$$Costo_{transporte} = 5[Litros] * 750[CLP/Litro] = 3{,}750[CLP] \tag{4.8}$$

En que se definen 5 litros porque con 4 no se alcanza a llegar al lugar. Para los costos de mano de obra, el tiempo dedicado por ella sería para el sistema actual de 50 horas (3000 minutos, 30 minutos para cada uno de los 100 dispositivos) y de 3 horas y 20 minutos para el sistema mejorado (200 minutos, 10 minutos para cada uno de los 100 dispositivos). Los costos para ambos casos, con CostoA para sistema actual y costoM para sistema mejorado son los siguientes:

$$CostoA_{manoobra} = 12{,}500[CLP/Hora] * 50[Horas] = 625{,}000[CLP] \tag{4.9}$$

$$CostoM_{manoobra} = 5{,}000[CLP/Hora] * 10/3[Horas] \approx= 16{,}600[CLP] \tag{4.10}$$

En relación a los costos de producción, nuevamente existe la posibilidad de dispositivos defectuosos que deben cambiarse por otros y no se sabe la probabilidad que esto ocurra en el caso de sistema mejorado, mientras que en el sistema actual, obligatoriamente se debe asistir a planta a intentar arreglar el problema. Del sistema mejorado, en el peor caso en que siempre se deba arreglar la situación, el costo es:

$$CostoM_{produccion} = 45{,}000[CLP/Hora] * 10/3[Horas] \approx 150{,}000[CLP] \tag{4.11}$$

Mientras que obligatoriamente, para el sistema actual los costos son:

$$CostoA_{produccion} = 45{,}000[CLP/Hora] * 50[Horas] = 2{,}250{,}000[CLP] \tag{4.12}$$

La tabla 4.9 describe un detalle de este ejemplo de uso, , en que la mejora con el sistema de configuración es de un 94.2 %.

- **Corrección de errores por no envío de datos:** Este ejemplo consiste en una planta a la cual no llegan datos a 10 de sus *Data Collector*, la planta está fuera de Santiago, nuevamente a 500 Km. Adicionalmente, no se sabe qué está generando estos problemas en cada uno de los dispositivos, así que se toma en cuenta la notificación de no-medición, la búsqueda del error a través de los parámetros de salud del sistema y la corrección, con su posterior verificación.

  El costo por transporte no existe para el sistema mejorado y es el mismo que en la primera situación analizada, 31.500 CLP aproximadamente.

Parámetro	Situación Actual	Situación Mejorada
Tiempo de Corrección (Horas)	50	10/3
Costo por Transporte (CLP)	6.000	0
Costo por Mano de Obra (CLP)	550.000	16.600
Costo por Producción (CLP)	2.250.000	150.000
Costo Total (CLP)	2.860.000	166.600
Reducción (%)	-	94.2

Tabla 4.9: Ejemplo de uso: Configuración Masiva de un parámetro

El costo por mano de obra considera 2 horas por el proceso completo de corrección, pero un día completo (que puede ser 8 horas extra) por costos de producción. Por mano de obra, 10 dispositivos equivalen a 20 horas totales para el sistema actual y 3 horas y 20 minutos para el sistema mejorado (el tiempo por corrección con la interfaz web baja a 10 minutos por dispositivo), esto implica los costos siguientes, donde CostoA es costo del sistema actual y costoM es el costo del sistema mejorado:

$$CostoA_{manoobra} = 12{,}500[CLP/Hora] * 25[Horas] = 312{,}500[CLP] \tag{4.13}$$

$$CostoM_{manoobra} = 5{,}000[CLP/Hora] * 5/3[Horas] \approx= 83{,}000[CLP] \tag{4.14}$$

En cuanto a los costos producción, estos para sistema actual y mejorado se consideran como variables, ya que puede haber una falla de producción que no se está contabilizando y detectando por *Data Collectors* defectuosos. Este error se contabiliza tomando en cuenta dos factores para el sistema actual: Una pérdida de producción por intervención del sistema y una posible pérdida proveniente de un *Data Collector* que no la está detectando. Para el sistema mejorado, se realiza la suposición de un sistema de medición de parámetros que opera de manera automática cada 30 minutos, por lo tanto, el factor de falla no contabilizada no se presenta.

Los costos para el sistema actual son los siguientes:

$$CostoA_{produccion} = 45{,}000[CLP/Hora] * 20[Horas] = 900{,}000[CLP] \tag{4.15}$$

$$CostoA_{factorfalla} = 45{,}000[CLP/Hora] * 80[Horas] = 2{,}700{,}000[CLP] \tag{4.16}$$

Donde el factor $CostoA_{factorfalla}$ puede o no presentarse, dando un Costo de pérdida por producción entre 900.000 [CLP] y 3.600.000 [CLP].

Para el sistema mejorado los costos toman en cuenta dos factores, el tiempo de corrección de 10 minutos por cada dispositivo y el tiempo de detección máximo, dado por el muestreo de 30 minutos (5 horas entre 10 dispositivos), dando un total de 6 horas y 40 minutos. El costo de producción es entonces:

$$CostoM_{produccion} = 45{,}000[CLP/Hora] * 20/3[Horas] \approx 300{,}000[CLP] \tag{4.17}$$

La tabla 4.10 describe un detalle de este ejemplo de uso, en que la mejora con el sistema de configuración es de un 75.8 %.

Parámetro	Situación Actual	Situación Mejorada
Tiempo de Corrección (Horas)	25	5/3
Costo por Transporte (CLP)	60.000	0
Costo por Mano de Obra (CLP)	312.500	8.300
Costo por Producción (CLP)	900.000	300.000
Costo Total (CLP)	1.272.500	308.300
Reducción (%)	-	75.8

Tabla 4.10: Ejemplo de uso: Corrección de errores por no envío de datos

Parámetro	Situación Actual	Situación Mejorada
Tiempo de Corrección (Horas)	50	5
Costo por Transporte (CLP)	60.000	0
Costo por Mano de Obra (CLP)	625.000	25.000
Costo por Producción (CLP)	2.250.000	225.000
Costo Total (CLP)	2.935.000	250.000
Reducción (%)	-	91.5

Tabla 4.11: Ejemplo de uso: Envío Masivo de Firmware

- **Envío de Firmware Masivo:** Para el caso en que hay una nueva funcionalidad en el preprocesamiento de datos que el microcontrolador realiza, se cambian los dispositivos. Se estima para el sistema actual que en enviar y comprobar que los cambios queden bien, dado que este proceso en particular suele fallar, demora 1 hora por dispositivo, considerándose 50 dispositivos a los que se envía el mismo firmware de forma masiva. El sistema mejorado disminuye este tiempo a 6 minutos por dispositivo.

  En cuanto a los costos de transporte, considerándose una ruta de 500 Km, no existen para el sistema mejorado, mientras que 60.000 CLP para el sistema actual.

  Como son 50 dispositivos, el tiempo es de 50 horas para el sistema actual y de 5 horas para el sistema mejorado. Los costos para cada caso son los siguientes:

$$CostoA_{manoobra} = 12{,}500[CLP/Hora] * 50[Horas] = 625{,}000[CLP] \tag{4.18}$$

$$CostoM_{manoobra} = 5{,}000[CLP/Hora] * 5[Horas] = 25{,}000[CLP] \tag{4.19}$$

  En el caso de pérdida de producción, se presenta nuevamente un costo fijo para el sistema actual y un costo máximo para el sistema mejorado, esto es:

$$CostoM_{produccion} = 45{,}000[CLP/Hora] * 50[Horas] \approx 2{,}250{,}000[CLP] \tag{4.20}$$

$$CostoA_{produccion} = 45{,}000[CLP/Hora] * 5[Horas] = 225{,}000[CLP] \tag{4.21}$$

  Un resumen de estos costos se presenta en la tabla 4.11, , en que la mejora con el sistema de configuración es de un 91.5 %.

  Para todos los casos analizados, la reducción de costos mínima es significativa. Tal como lo señala la tabla 4.12, la reducción mínima se encuentra entre 75.8 % para corrección

Caso de Uso	Costo Actual (CLP)	Costo Mejorado (CLP)	Reducción Porcentual (%)
Agregar Nodos	1497500	83.300	94.4
Configuración Masiva	2.860.000	166.600	94.2
Corrección de Errores	1.272.500	308.300	75.8
Envío Firmware	2.935.000	250.000	91.5

Tabla 4.12: Reducción de Costos por Caso de Uso

de Errores y 94.4 % para el agregado de Nodos, ambos de forma masiva. Notar que este 75.8 % posiblemente sea mayor, ya que no está considerando la pérdida de producción por la no resolución inmediata del problema ($Costo_{factorfalla}$) y el monitoreo para identificación de error se considera cada 30 minutos, periodo que perfectamente puede reducirse.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

Se logra establecer una relación entre los costos y las variables del sistema, entendidas como los tiempos relevantes dentro de la interfaz: Los de ejecución, entendidos como los periodos comprendidos entre el instante en que el usuario envía la solicitud y el que recibe la respuesta y mucho más importantes, los de usuario o también llamados De Llenado de Formulario, periodo de tiempo entre la apertura del formulario y su envío, que tal como se ha reportado, tiene un orden de magnitud mayor a los tiempos de ejecución. Estos tiempos producen una reducción en los costos versus el sistema PMSystem actual que no posee el sistema de configuración de hasta 94 %, lo que implica una mejora significativa.

El análisis de la importancia de los tiempos anteriores no hubiese sido posible sin el establecimiento de una lógica de utilización de la implementación completa, en que se definen los casos de uso y los estados relevantes (que motivan la configuración) a partir de ellos. De la información bibliográfica existente sobre los dispositivos XBee/microcontrolador y la lógica de utilización establecida dentro de la empresa Neykos, se establecen rangos correctos y los comandos con los parámetros configurables.

Algunas de las implementaciones realizadas para la arquitectura de software integrada en los dispositivos, las más pertinentes, son presentadas: Uso de gRPC en la comunicación Intraprocesos, selección de la librería para el Motor XBee, funcionamiento de los motores implementados, de los canales de comunicación y de la lógica de mensajes utilizados. Todo esto, sumado a la validación del servidor en la etapa final, permitió que el sistema general funcionara suficientemente bien para obtener resultados de él.

Finalmente, la validación y análisis de una versión a escala en base a la obtención de tiempos relevantes por medio de tres dispositivos XBee permite la posibilidad de una puesta en marcha de la interfaz de configuración desarrollada, lo que junto al análisis económico de los casos de uso, permite el análisis de las mejoras dadas por esta implementación y a definir si es o no pertinente este sistema de configuración y monitoreo, objetivo principal del presente trabajo.

5.2. Trabajo Futuro

Al inicio del proyecto, se considera probar la recepción de XBee (XBee *engine* en la figura 5.1) con las librerías de C y C++, permitiendo así la creación y recepción de mensajes con mayor rapidez en comparación con Python, en caso de obtener tiempos mejores a las librerías en Python. Esta idea tomaría pruebas similares a las realizadas en la sección 4.1 y las compararía con los resultados obtenidos en esta memoria, de resultar más rápida la recepción de mensajes, se implementaría alguna de las librerías en C o C++ en el motor de *Gateway* XBee Engine, descrito en la sección 3.6.2.

Las tareas programadas pueden ser automatizadas por medio de una reorganización de la interfaz web, de manera que todos los casos de uso presentado sean evaluados y en caso que se requiera, corregidos. Por ejemplo, puede haber un proceso sin necesidad de usuario, de tal forma que sitúe los parámetros de salud de un *Data Collector* en un rango y una vez que un parámetro específico se salga de ese rango, se detone alguna notificación que avise al cliente de la interfaz web que un *Data Collector* específico se encuentra en problemas y está haciendo lo posible por detectar y corregir el problema.

El sistema de planta puede escalarse en tamaño, es decir, aumentar la cantidad de *Gateways* y *Data Collectors* en la parte izquierda de la figura 5.1, de forma que se puedan realizar nuevas pruebas de concepto y detectar nuevos problemas relacionados a redes más grandes, con tal de robustecer PMSystem y tener la capacidad de prestar servicios a entornos industriales con mayor variedad de parámetros a medir, requiriendo de esta forma más *Data Collectors* y/o mayor cantidad de lineas de producción, que se traduce en una mayor cantidad de *Gateways*. A su vez, las pruebas permitirían entender el efecto económico de configurar parámetros en forma masiva, algo no abordado en el desarrollo de esta memoria.

Se pueden integrar medidores inteligentes que utilicen comunicación serial de tipo RS-485, utilizados en entornos industriales como medidores de energía, temperatura, etc. Para ello, se tendrían que integrar como sensores dentro del motor de *Gateway* Data Collector en 5.1, descrito en la sección 3.6.2, para luego crear una nueva estructura de mensajes que sea compatible con la de los medidores, medir los tiempos de configuración y de lectura de parámetros de un medidor en específico e integrar estos mensajes con peticiones a realizar desde el cliente por medio de nuevas ventanas en la interfaz creada.

Una garantía de seguridad mayor puede establecerse al incluir encriptaciones y aspectos de seguridad como los descritos por IETF en la sección 2.2.1 dentro de la programación tanto del *Gateway* como de los *Data Collectors*, lo que se puede extender incluso a la utilización de redes de prueba que utilicen 6LOWPAN u otros estándares, en contextos de prueba consistentes en ataques de seguridad también documentados en IETF.

La interfaz web desarrollada puede ser transformada en una aplicación web para algún teléfono móvil, de tal forma que pueda ser operada por técnicos de planta en un dispositivo de bolsillo, conectándose a una red de internet. Para ello, se utilizaría un software de desarrollo de aplicaciones web, como AppSheet, Jira o Caspio [41].

Finalmente, el modelo de estimación de costos se puede complejizar más, haciendo estudios de variables estocásticas como la probabilidad de fallas de cada uno de los casos de uso,

posterior a su correcta configuración. Para esto, se necesita un registro anterior de las fallas que una planta tenga, su orígen dentro de los casos de uso y la última fecha y hora en que fue configurado el *Data Collector* que falla. Con lo anterior, pueden establecerse rangos de costo para la empresa Neykos y abarcar un espectro de escenarios mayor.

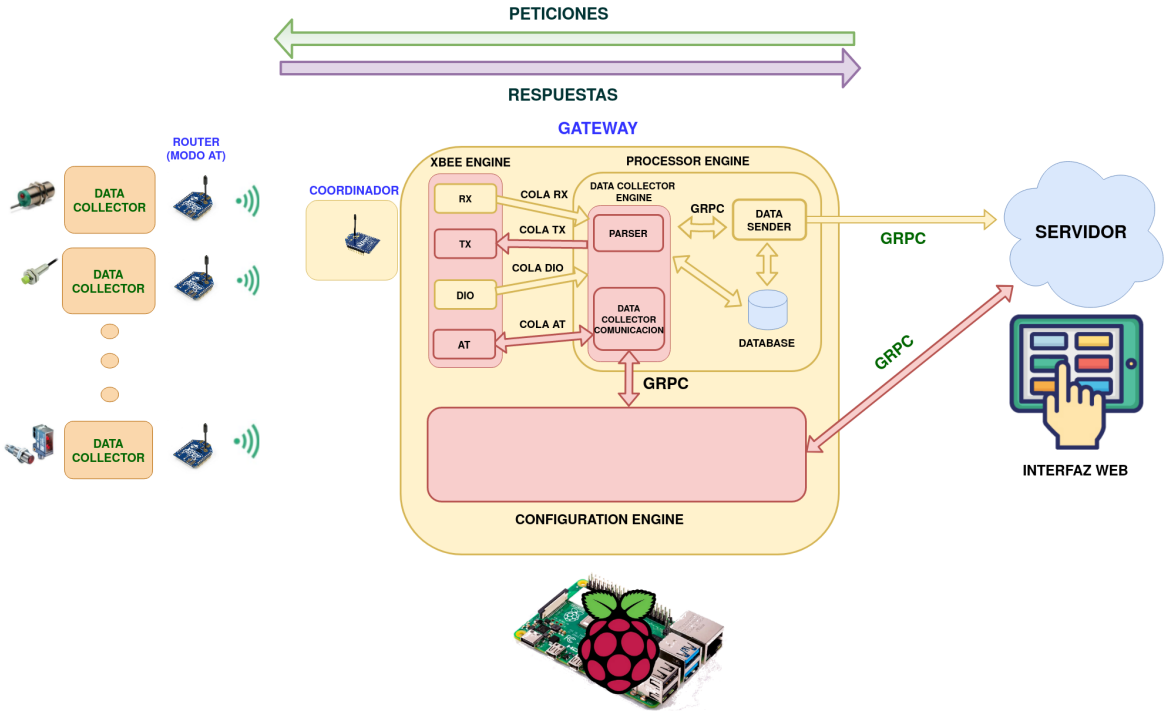


Figura 5.1: Diagrama del Sistema Completo Desarrollado

Bibliografía

[1] "¡Bienvenido a la Industria 4.0, la Cuarta Revolución Industrial! – Eviciti Technologies," visited on 2020-10-06. [Online]. Available: https://www.eviciti.com.mx/blog-post/bienvenido-a-la-industria-4-0-la-cuarta-revolucion-industrial/

[2] "What is ZigBee Technology, Architecture and its Applications?" visited on 2019-12-27. [Online]. Available: https://www.elprocus.com/what-is-zigbee-technology-architecture-and-its-applications/

[3] "¿Qué es XBee? XBee.cl - Comunicación Inalámbrica para Tus Proyectos," visited on 2019-12-27. [Online]. Available: https://xbee.cl/que-es-xbee/

[4] "XBee X-CTU tutorial | Libelium," visited on 2019-12-27. [Online]. Available: http://www.libelium.com/development/waspmote/documentation/x-ctu-tutorial/

[5] P. V. M. Deshmukh, "Monitoring and control of gas leakages of industrial sector using pic 18f4550, zigbee and wireless sensor actuator network," *i-Manager's Journal on Electronics Engineering*, vol. 8, no. 3, p. 5, 2018.

[6] S. G. Nikhade, "Wireless sensor network system using raspberry pi and zigbee for environmental monitoring applications," in *2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*. IEEE, 2015, pp. 376–381.

[7] G. Masetti, F. Marazzi, L. Di Cecilia, and L. Rovati, "Iot-based measurement system for wine industry," in *2018 Workshop on Metrology for Industry 4.0 and IoT*. IEEE, 2018, pp. 163–168.

[8] C. Xiang and B. Li, "Research on ship intelligent manufacturing data monitoring and quality control system based on industrial internet of things," *The International Journal of Advanced Manufacturing Technology*, pp. 1–10, 2019.

[9] "The future of iot: 4 predictions about the internet of things," visited on 2020-01-02. [Online]. Available: https://thriveglobal.com/stories/the-future-of-iot-4-predictions-about-the-internet-of-things/

[10] "What is Industry 4.0—the Industrial Internet of Things (IIoT)?" visited on 2020-08-13. [Online]. Available: https://www.epicor.com/en-us/resource-center/articles/

what-is-industry-4-0/

[11] "What is a data logger? - Advantages of data logging," visited on 2020-08-13. [Online]. Available: https://www.omega.co.uk/prodinfo/dataloggers.html

[12] "Cómo utilizar el DHT11 para medir la temperatura y humedad con Arduino," Mar. 2017, section: Arduino. Visited on 2020-08-13. [Online]. Available: https://programarfacil.com/blog/arduino-blog/sensor-dht11-temperatura-humedad-arduino/

[13] "Gateway Definition," visited on 2020-08-13. [Online]. Available: https://techterms.com/definition/gateway

[14] "Comparison of Wireless Technologies (Bluetooth, WiFi, BLE, Zigbee, Z-Wave, 6lowpan, NFC, WiFi Direct, GSM, LTE, LoRa, NB-IoT, and LTE-M) | PREDICTABLE DESIGNS," visited on 2019-12-27. [Online]. Available: https://predictabledesigns.com/wireless_technologies_bluetooth_wifi_zigbee_gsm_lte_lora_nb-iot_lte-m/

[15] "Zigbee Certified Products Archives - Zigbee Alliance," visited on 2019-12-27. [Online]. Available: https://zigbeealliance.org/product_type/certified_product/

[16] "The Difference Between Software, Firmware, and Hardware," Feb. 2015, visited on 2020-08-13. [Online]. Available: https://danielmiessler.com/blog/software-firmware-hardware/

[17] "Firmware Definition," visited on 2020-08-13. [Online]. Available: https://techterms.com/definition/firmware

[18] "4.1.2 COMUNICACIÓN ENTRE PROCESOS (SOCKETS, RPC). - Taller de sistema Operativos ll," visited on 2020-08-17. [Online]. Available: https://sites.google.com/site/tallerdesistemaoperativosll/unidad-iv/4-1-interoperabilidad-entre-sistemas-operativos/4-1-1-sistemas-de-archivos-y-recursos-nfs-impresoras/4-1-2-comunicacion-entre-procesos-sockets-rpc

[19] H.-A. Jacobsen, *Publish/Subscribe*. Boston, MA: Springer US, 2009, pp. 2208–2211. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_1181

[20] "Redis," visited on 2020-10-06. [Online]. Available: https://redis.io/

[21] "gRPC," visited on 2020-08-17. [Online]. Available: https://grpc.io/

[22] "¿Qué es un socket?" visited on 2020-08-17. [Online]. Available: https://www.speedcheck.org/es/wiki/socket/

[23] "Qué es TCP/IP y características principales," Jun. 2019, visited on 2020-08-17. [Online]. Available: https://openwebinars.net/blog/que-es-tcpip/

[24] "Zigbee rf modules," visited on 2019-12-27. [Online]. Available: https://www.digi.com/resources/documentation/digidocs/pdfs/90000976.pdf

[25] "Github - digidotcom/xbee-python: Python library to interact with digi internationals

xbee radio frequency modules." visited on 2020-01-02. [Online]. Available: https://github.com/digidotcom/xbee-python

[26] "Github - digidotcom/xbee_ansic_library: A collection of portable ansi c code for communicating with digi internationals xbee wireless radio modules in api mode." visited on 2020-01-02. [Online]. Available: https://github.com/digidotcom/xbee_ansic_library

[27] "Github - attie/libxbee3: A c/c++ library to aid the use of digi xbee radios in api mode," visited on 2020-01-02. [Online]. Available: https://github.com/attie/libxbee3

[28] "Xbee - pypi," visited on 2020-01-02. [Online]. Available: https://pypi.org/project/XBee/

[29] S. Stević, V. Lazić, M. Z. Bjelica, and N. Lukić, "Iot-based software update proposal for next generation automotive middleware stacks," in *2018 IEEE 8th International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*. IEEE, 2018, pp. 1–4.

[30] M. Weißbach, N. Taing, M. Wutzler, T. Springer, A. Schill, and S. Clarke, "Decentralized coordination of dynamic software updates in the internet of things," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. IEEE, 2016, pp. 171–176.

[31] N. Asokan, T. Nyman, N. Rattanavipanon, A.-R. Sadeghi, and G. Tsudik, "Assured: Architecture for secure software update of realistic embedded devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2290–2300, 2018.

[32] D.-Y. Kim, S. Kim, and J. H. Park, "Remote software update in trusted connection of long range iot networking integrated with mobile edge cloud," *IEEE Access*, vol. 6, pp. 66 831–66 840, 2017.

[33] "Software Updates for Internet of Things (suit) -," visited on 2020-05-26. [Online]. Available: https://datatracker.ietf.org/wg/suit/about/

[34] H. B. B. Moran, H. Tschofenig, "An information model for firmware updates in iot devices (work in progress)," Internet Requests for Comments, RFC Editor, draft-ietf-suit-information-model 05, May 2020. [Online]. Available: https://www.ietf.org/id/draft-ietf-suit-information-model-05.txt

[35] "Remote ATtestation ProcedureS (rats) -," visited on 2020-05-26. [Online]. Available: https://datatracker.ietf.org/wg/rats/about/

[36] E. M. Bormann, C. and A. Keranen, "Terminology for constrained-node networks," Internet Requests for Comments, RFC Editor, Informational 05, May 2014. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7228.txt

[37] "Buscador de servicentros CNE," visited on 2020-09-30. [Online]. Available: http://www.bencinaenlinea.cl/web2/

[38] "Rendimiento de gasolina - ¿Cómo se calcula? - Rastreator.mx®," visited on 2020-09-30. [Online]. Available: https://www.rastreator.mx/seguros-de-auto/guias/

como-calcular-el-rendimiento-de-gasolina-por-km

[39] G. Grimaldo, D. Moreno, and M. Salamanca, "Medición del trabajo de una línea de producción de yogurt-empresa la hacienda," *Revista I3+, II (2)*, pp. 62–81, 2015.

[40] C. l. v. c. t. c. m. s. p. e. C. y. s. e. VCM, "Conoce las veinte carreras técnicas con mejor sueldo promedio en Chile y su empleabilidad," Dec. 2018, section: Noticias. Visited on 2020-09-30. [Online]. Available: https://vcm.emol.com/3774/noticias/conoce-las-veinte-carreras-tecnicas-con-mejor-sueldo-promedio-en-chile-y-su-empleabilidad/

[41] "[Top 10] Best App Development Software Platforms of 2020," visited on 2020-08-17. [Online]. Available: https://www.softwaretestinghelp.com/best-app-development-software/