UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DESARROLLO DE BACKEND PARA SISTEMA DE MEDICIÓN AUTOMÁTICA DE CALIDAD DE AGUA, ADECUADO A SENSORES DE BAJO COSTO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

GABRIELA ELISA MENDOZA MUÑOZ

PROFESOR GUÍA:
JAVIER BUSTOS JIMÉNEZ

MIEMBROS DE LA COMISIÓN:
SANDRA CÉSPEDES UMAÑA
LUIS MATEU BRULE
BÁRBARA POBLETE LABRA

SANTIAGO DE CHILE
2021

Resumen

En un contexto de escasez hídrica en Chile, surge el proyecto FONDEF ID19 I10363[1] que propone monitorear la calidad de diversas fuentes de agua, utilizando sensores de bajo costo. Para esto, es necesario disponer de un sistema de procesamiento y almacenamiento de datos adecuado a registros de calidad de agua y provenientes de estos dispositivos de bajo costo, cuya implementación es el objetivo principal de este trabajo de memoria.

En el siguiente trabajo se diseña e implementa un *backend* de adquisición de datos para el sistema de monitoreo de calidad de agua propuesto por el proyecto FONDEF. La solución integra las aplicaciones de dos redes LPWAN[2] con una API que ofrece alternativas para mitigar el ruido que pueden generar los sensores que toman registros, mediante distintos algoritmos de filtrado de ruido. El trabajo presenta un análisis de estos algoritmos sobre datos históricos de calidad de agua en Chile, y un *benchmark* comparativo de filtros de ruido sobre datos obtenidos con el sensor de temperatura DS18B20.

Los registros de calidad de agua son almacenados en una base de datos de series temporales integrada a un tablero de control. Permitiendo así ofrecer una alternativa de visualización, recopilamiento y almacenamiento de datos adecuado a un sistema de bajo costo.

Para validar la solución, se realizan pruebas de conexión con el servidor y escrituras a la base de datos, obteniendo como límite un máximo de 100 clientes concurrentes, utilizando las configuraciones por defecto del servidor. También, se realiza un análisis comparativo de algoritmos de filtrado de ruido en línea y fuera de línea, obteniendo mejores métricas utilizando el algoritmo descompositivo clásico, en comparación a las métricas de los datos sin filtrar.

[1] Sistema abierto experto para apoyar la gestión de recursos hídricos, mediante monitoreo de bajo costo en tiempo real de aguas superficiales y subterráneas.

[2] Low-power wide-area network.

A mi familia, por ser tan especiales,

y a unas buenas horas de sueño.

Agradecimientos

Nada de esto sería posible sin el apoyo incondicional de mi mamá y papá; muchas gracias por ser tan dedicados a su familia; sus dos hijas e hijo, gracias por llenarme de amor, por tenerme paciencia, por enseñarme cosas diferentes y por criarme sin sesgos de género.

Quiero agradecer a mis amigos y amigas; a Victor y Pedro, mis primeros amigos del DCC; a la Javi, por acercarse a hablarme ese día en el cerro, que inició una larga amistad; gracias por presentarme a los chiquillos. A Esteban por compartir este viaje conmigo y por el apañe.

A mis profesores Javier y Sandra; por darme la oportunidad de trabajar en este proyecto y enseñarme a ser mejor profesional. Al Nic Labs y todas sus generaciones, por ser una gran escuela, y grandes personas con un gran corazón.

Quiero agradecer a FONDEF, por financiar parcialmente este trabajo.

Tabla de Contenido

1. Introducción **1**
- 1.1. Contexto . . . 1
- 1.2. Problema . . . 1
- 1.3. Alternativas analizadas . . . 2
- 1.4. Objetivos . . . 2
  - 1.4.1. Objetivos Específicos . . . 2
- 1.5. Solución . . . 2
- 1.6. Resumen de los resultados . . . 3
- 1.7. Estructura . . . 3

2. Estado del Arte **4**
- 2.1. Terminología y Conceptos Principales . . . 4
- 2.2. Parámetros monitoreados . . . 5
- 2.3. Redes de bajo consumo . . . 6
- 2.4. Filtrado de ruido . . . 7
  - 2.4.1. Algoritmos de reducción de ruido en línea: . . . 7
  - 2.4.2. Algoritmos de reducción de ruido fuera de línea . . . 9

3. Problema **11**
- 3.1. Replicabilidad . . . 11
- 3.2. Escalabilidad . . . 12
- 3.3. Ruido y Fallas . . . 12

4. Solución Propuesta **14**
- 4.1. Arquitectura Propuesta . . . 14
  - 4.1.1. Tecnología Propuesta . . . 15

5. Implementación de la Solución **17**
- 5.1. Servidor y protocolo de comunicación . . . 17
  - 5.1.1. Levantamiento de servidor MQTT . . . 17
  - 5.1.2. Levantamiento de servidor HTTP . . . 17
  - 5.1.3. Conexión con la nube de la red . . . 18
- 5.2. API . . . 20
  - 5.2.1. Backend . . . 20
  - 5.2.2. Endpoints y esquema de datos . . . 21
  - 5.2.3. Esquemas JSON de paquetes provenientes de redes LPWAN . . . 21

5.2.4. Externalización de información 22
5.3. Diseño de base de datos 23
5.3.1. Caracterización de los datos 23
5.3.2. Elección de base de datos 24
5.3.3. Diseño y limitantes 25
5.3.4. Implementación 26
5.4. Filtrado de datos 28
5.4.1. Algoritmos 28
5.4.2. Experimentos 30
5.5. Interfaz de Usuario 33
5.5.1. Plataforma web 33
5.5.2. Tablero de control 34
5.5.3. Chatbot 35

6. Validación **37**
6.1. Conexión 37
6.2. Escalabilidad 39
6.2.1. Pruebas de estrés utilizando Jmeter 39
6.2.2. Configuraciones de Jmeter 39
6.2.3. Configuraciones de Apache: 40
6.2.4. Configuraciones de WSGI: 40
6.2.5. Experimentos: 40
6.2.6. Experimento 1 40
6.2.7. Experimento 2 40
6.3. Filtrado: 42
6.3.1. Medidas de performance 42
6.3.2. Resultados 42

7. Conclusión **45**
7.0.1. Trabajo a futuro 45

Glosario **47**

Bibliografía **48**

Índice de Tablas

5.1. Información del sensor. 22
5.2. Información de la estación: 22

Índice de Ilustraciones

2.1. Pozo de agua, estación en sector Cariño Botado, en San Esteban. Imagen gentileza de Matías Taucare PhD en Ciencias mención Geología. 4
2.2. a) Agujero para ingresar cilindro de sensor (en rojo). b) Cilindro con ranura para portar sensor. Imágenes gentileza de Matías Taucare PhD en Ciencias mención Geología. 5
2.3. Dispositivos IoT, *gateways* de red LPWAN y nube de la red. 6

3.1. Niveles de temperatura en estaciones con distinta ubicación geográfica. Utilizando datos entregados por un equipo de hidrogeología del Departamento de Geología. 12
3.2. Niveles de conductividad eléctrica en estaciones con distinta ubicación geográfica. Utilizando datos entregados por un equipo de hidrogeología del Departamento de Geología. 12

4.1. Arquitectura de la solución. 14
4.2. *Diseño de conexión a través de un Broker MQTT*: Broker Mosquitto conectado a una API en Python utlizando el cliente MQTT Pahoo [3]. Conexión a base de datos InfluxDB que alimenta tablero de control Grafana. 15
4.3. *Diseño de conexión a través de servidor web HTTP*: Servidor HTTP Apache conectado a un *backend* en Flask a través de WSGI. Conexión a base de datos InfluxDB que alimenta tablero de control Grafana 15

5.1. Consola de TTN con la opción de acceder a las aplicaciones y *gateways* del sistema. 19
5.2. Aplicación de testeo del proyecto FONDEF. 19
5.3. Dispositivo de la aplicación del proyecto FONDEF. 19
5.4. Integración HTTP para la aplicación del proyecto FONDEF. 20
5.5. Correlación entre los parámetros de 18 estaciones de medición de agua potable rural, utilizando datos provistos por un equipo de hidrogeología del departamento de Geología. Experimentos realizados por Miguel Solís Dr. en Ingeniería Informática. 23
5.6. Data Point influxDB, imagen obtenida de la documentación de Influx[4] . . . 24
5.7. Almacenes de datos. 27
5.8. Suavizado de datos de pozo de minera (anonimizados), facilitados por Arcadis 28
5.9. Suavizado con descomposición aditiva, de datos de pozo de minera (anonimizados), facilitados por Arcadis. 29
5.10. Subsample seleccionado, datos de 1 día. 30

5.11. Datos reales y predichos por la red. 31
5.12. Filtrado y predicción con distintas configuraciones. 32
5.13. Landing page con drop down para registrar Dispositivo. 33
5.14. Modal para registrar nuevo nodo. 33
5.15. Tablero de control con datos históricos de institución anonimizada. 34
5.16. Conexión con canal de Telegram. 35
5.17. Mensaje alerta de prueba. 35
5.18. Integración a Grafana. 36
5.19. Integración con Chatbot de Telegram. 36

6.1. Bases de datos conectadas con Grafana. 37
6.2. Escrituras exitosas a la base de datos. 38
6.3. Configuraciones de jmeter. 39
6.4. Señal original ensuciada con ruido. 43
6.5. Filtrado de ruido a señal ensuciada. 44

Capítulo 1

Introducción

1.1. Contexto

Los recursos hídricos de la zona norte y centro de Chile son cada vez más escasos, de los cuales el porcentaje asignado para consumo doméstico es menor al 10 % del total, según informes de la Mesa Nacional del Agua [10]. La mega sequía que vive el país es tan sólo uno de los desafíos a los que se enfrentan los distintos proveedores de agua potable del país, si consideramos los altos índices de contaminación por residuos industriales, deslaves y sedimentación de cauces o pozos subterráneos.

Actualmente, existen mediciones automatizadas de contaminación hídrica a partir de sensores conectados vía satelital, pero que se encuentran limitadas en la cantidad de puntos de medición debido a los altos costos de conexión. Otras soluciones contemplan procesos manuales de recolección de medidas, lo cual es muy costoso, tanto en horas como en los riesgos que se puede incurrir al no tener inmediatez en el acceso a los datos de monitoreo. Por lo cual, se hace urgente implementar otras medidas de manejo y monitoreo de calidad de agua. Es en este contexto que surge el proyecto FONDEF ID19 I10363[1], al cual se encuentra asociado esta memoria.

1.2. Problema

Con el fin de adelantarse a posibles riesgos de contaminación, el proyecto contempla el desarrollo de software y hardware que monitorean parámetros de interés y alertan sobre factores de contaminación a tiempo, para lo cual se requiere el uso de dispositivos capaces de conectarse y transferir datos a través de una red, bajo el paradigma de Internet de las cosas (IoT, por su sigla en inglés). En la mayoría de los despliegues industriales, se dispone de tecnología de punta y sensores cuyo valor de mercado es muy alto, sin embargo, en este proyecto FONDEF se desea lograr una solución de monitoreo de bajo costo, para cubrir sectores de menos recursos que hoy no cuentan con seguimiento de sus fuentes hídricas.

[1]Sistema abierto experto para apoyar la gestión de recursos hídricos, mediante monitoreo de bajo costo en tiempo real de aguas superficiales y subterráneas.

Por lo que se vuelve necesario utilizar sensores de menor calidad, posiblemente con menor resolución, mayor ruido y susceptibilidad a fallas.

Se identifican dos tareas fundamentales en esta memoria como aporte al proyecto. Primero, la implementación de una arquitectura estable que soporte la comunicación con distintas aplicaciones de redes de sensores y el almacenamiento de un alto volumen de datos. Y como segundo, intentar mitigar el ruido de los datos, para poder alimentar un sistema de alerta temprana y detección de eventos, minimizando el impacto del ruido en los datos.

1.3. Alternativas analizadas

Para poder resolver las tareas descritas se evalúa el uso de distintas tecnologías, en particular se detallan y comparan:

- Los protocolos de comunicación HTTP y MQTT, en las secciones 2.3 y 4.1.1.
- Dos servidores web HTTP: Apache y Nginx en la sección 5.1.2.
- Los brokers MQTT Mosquitto y EMQx en la sección 5.1.1.
- Distintas bases de datos de series de tiempo (TSDB) en la sección 5.3.2.
- Algoritmos de filtrado de ruido en línea y fuera de línea, en las secciones 2.4, 5.4 y 6.3.
- Los tableros de control Kibana, Grafana y Chronograf en la sección 5.5.2.

1.4. Objetivos

Esta memoria tiene por objetivo diseñar y construir una arquitectura de procesamiento de datos, apropiada para registros de calidad de agua, mediante el uso de tecnologías de almacenamiento modernas. Sujeto al hecho de que se desea lograr replicabilidad con un presupuesto limitado, superando los desafíos que esto implica en la escalabilidad del diseño y el procesamiento de los datos provenientes de sensores de bajo costo.

1.4.1. Objetivos Específicos

- Lograr modularidad en la conexión, permitiendo la integración con distintas tecnologías.
- Diseñar e implementar un *backend* robusto, modular y escalable que permita la comunicación con el módulo de sensores.
- Diseñar y construir una base de datos adecuada al sistema.
- Construir un módulo de filtrado que permita manejar distintos escenarios de falla.

1.5. Solución

Como solución, se implementa la arquitectura necesaria para el procesamiento y almacenamiento de datos, que incluye:

- El protocolo de comunicación con el *backend* de la red de sensores, utilizando como servidor web HTTP Apache.

- Un almacén de datos acorde a los registros entregados por la red, utilizando la base de datos de series de tiempo InfluxDB.
- La elaboración de funciones que permiten el uso de este almacén, denominadas en su conjunto como API , utilizando el *framework* Flask para el lenguaje de programación Python.
- El uso de algoritmos y modelos matemáticos que permiten filtrar el ruido de los datos asociado a las mediciones de los sensores de bajo costo y fallas en los mismos.
- Un tablero de control utilizando Grafana, para realizar visualizaciones.

1.6. Resumen de los resultados

El trabajo logra implementar de forma satisfactoria una solución para procesar y almacenar datos de calidad de agua acorde a los requisitos del proyecto, obteniendo así, el siguiente resumen de resultados:

- La base de datos InfluxDB permite almacenar un gran volumen de datos históricos, indexados temporalmente e identificables por cliente, sector y estación.
- La API soporta hasta 100 nodos concurrentes, con las configuraciones por defecto de Apache.
- El algoritmo descompositivo entrega mejores resultados en el *benchmark* comparativo, comprobando su utilidad.

1.7. Estructura

El siguiente trabajo se divide en cinco capítulos:

- Capítulo 2: En este capítulo se explica la terminología utilizada en las redes de abastecimiento de agua, y la tecnología que actualmente se usa en soluciones existentes. Además, se presentan algoritmos y modelos de filtrado de ruido en series temporales.
- Capítulo 3: Se presentan los principales problemas a los que se enfrenta este trabajo de memoria.
- Capítulo 4: Se presentan la arquitectura y tecnologías de la solución.
- Capítulo 5: Se discute el diseño e implementación de la solución.
- Capítulo 6: Se valida la solución, realizando pruebas de conectividad y escalabilidad de la API y filtrado del ruido de los datos.

Capítulo 2

Estado del Arte

2.1. Terminología y Conceptos Principales

Las redes de abastecimiento de agua, a las cuales se desea integrar el sistema experto desarrollado en el proyecto FONDEF, pueden proceder de diferentes fuentes. En particular, se cuenta con datos de **aguas subterráneas**, y **aguas superficiales** de distintas estaciones, provistas por instituciones del rubro de manejo de aguas. Una estación, como la que se observa en la Figura 2.1, es un punto de medición donde se puede colocar el sensor para tomar los registros de calidad de agua; un acuífero en una determinada ubicación geográfica puede contar con múltiples estaciones. Además, en una misma estación se pueden medir varios parámetros de calidad de agua a la vez.

Figura 2.1: Pozo de agua, estación en sector Cariño Botado, en San Esteban. Imagen gentileza de Matías Taucare PhD en Ciencias mención Geología.

Para colocar un sensor en una estación de pozo subterráneo es necesario introducirlo en un porta sensor, con forma de cilindro. De forma tal, que quede asegurado herméticamente. En la Figura 2.2 se aprecia el punto de inserción y el cilindro portador del sensor.

(a)

(b)

Figura 2.2: a) Agujero para ingresar cilindro de sensor (en rojo). b) Cilindro con ranura para portar sensor. Imágenes gentileza de Matías Taucare PhD en Ciencias mención Geología.

2.2. Parámetros monitoreados

Trabajos anteriores han mostrado que independiente a las condiciones registradas, ciertos parámetros medidos en una estación son más relevantes para indicar cuándo el agua ha sido contaminada o no. Por ejemplo, estudios realizados por la USEPA[1] han demostrado que los contaminantes químicos y biológicos tienen un efecto notorio en ciertos parámetros como la **turbiedad**, el **pH** y la **conductividad eléctrica** [5].

En 2016, Geetha et al. [5] desarrollan un software de monitoreo de calidad de agua en la India, utilizando tecnología IoT y realizando seguimiento a cinco parámetros, escogidos en base a los estudios realizados por la USEPA, estos son: **temperatura, conductividad, pH, turbidez y presión.**[2]. Son estos mismos parámetros, los que se pretende monitorear con el sistema desarrollado en el proyecto FONDEF. La solución presentada en [5] dista de los requerimientos de este trabajo, ya que no ahonda en el filtrado de ruido proveniente de sensores de bajo costo.

[1] US Environmental Protection Agency.
[2] Se puede reportar tanto nivel, o caudal como alternativa a la presión.

2.3. Redes de bajo consumo

Recientemente, han surgido las redes de bajo consumo y amplio alcance (LPWAN, por su sigla en inglés)[3] que reducen el uso de batería de los sensores inalámbricos, ofreciendo la infraestructura necesaria para conectarse a una red de largo alcance. Para poder conectar diferentes dispositivos a Internet, la arquitectura de red LPWAN cuenta con *gateways*, que tienen la capacidad de conectarse a una nube que se aprovecha de servidores (*network server*) y aplicaciones internas (*application server*), como se aprecia en la Figura 2.3.

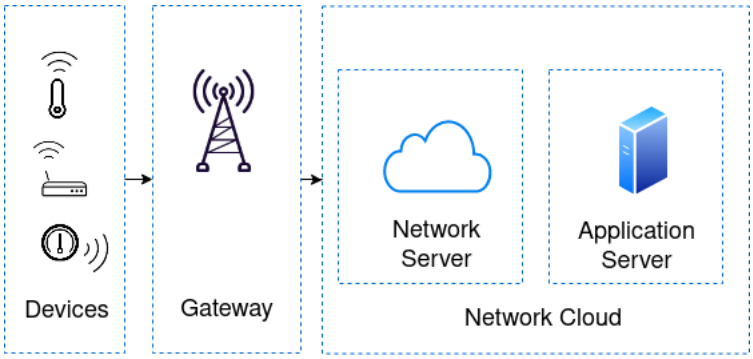


Figura 2.3: Dispositivos IoT, *gateways* de red LPWAN y nube de la red.

Dos de las redes LPWAN más utilizadas son LoRa[4] y Sigfox[5], ambas cumplen con el propósito de conectar dispositivos de bajo consumo energético, además de ofrecer servicios y contar con presencia en varios continentes. Por otro lado, ambas tecnologías tienen varias diferencias, las más relevantes para el proyecto y para esta memoria son los protocolos internos y externos de comunicación. En particular, el protocolo externo de comunicación es aquel que conecta el *application server* de la red con la API desarrollada en esta memoria. LoRa se puede conectar a una nube de acceso abierto denominada The Things Network (TTN), la cual soporta integraciones a APIs externas a través de los protocolos HTTP[6] y MQTT[7](un protocolo de comunicación que funciona en base a suscripciones y publicaciones a tópicos). Mientras que la nube de Sigfox soporta HTTP, pero no cuenta con una implementación de MQTT en la capa de aplicación. HTTP es un protocolo muy confiable, sin embargo, genera mucho gasto ya que debe enviar muchos paquetes pequeños al establecer cada conexión, consumiendo recursos de la red [12]. De acuerdo a Wukkadada et al. [1] MQTT sería un protocolo de menor latencia gracias a su intercambio de datos mediante subscripciones a tópicos, lo cual reduciría el uso de ancho de banda.

Las aplicaciones de estas redes son varias: desde sistemas de detección temprana de incendios, como el propuesto por Sendra et al. [11] utilizando la red de bajo consumo y amplio alcance *The Things Network*, para cubrir un área circular de 4 km de radio en el bosque español; hasta sistemas de monitoreo de irrigación para la agricultura, como el sistema propuesto por Fernández et al. [4], utilizando una arquitectura híbrida con las redes *Sigfox* y *LoRa*.

[3]Low-power wide-area network.
[4]https://lora-alliance.org/.
[5]https://www.sigfox.com/en.
[6]Hypertext Transfer Protocol.
[7]Message Queue Telemetry Transport

2.4. Filtrado de ruido

Los sensores de bajo costo pueden generar mayor ruido en la medición, debido a que son de una calidad más baja. Esto puede ser un problema al momento de intentar detectar eventos de contaminación, ya que el ruido puede afectar severamente el porcentaje de acierto de cualquier modelo basado en datos [8]. Karunasingha et al. indican que entradas de datos con reducción de ruido producen una mayor porcentaje de acierto incluso en modelos entrenados con datos ruidosos, de acuerdo a estudios anteriores donde se utilizaron series de tiempo caóticas. [8].

Reducir ruido en series de tiempo, es un problema que se puede definir como sigue:

Suponemos que y_{i} es una serie de tiempo ruidosa, y x_{i} es la misma serie de tiempo, pero sin ruido, cuya señal real es en realidad desconocida. Se pretende entonces minimizar $\hat{\varepsilon} = |\hat{y}_{\mathrm{i}} - x_{\mathrm{i}}|$ preferentemente para todo $\mathrm{i} = 1, 2, 3...N$, Tal que la nueva serie filtrada $\hat{y}_{\mathrm{i}}$ esté más cerca de x_{i}, que la serie ruidosa y_{i}.

La reducción de ruido, en general puede ser aplicada en línea, o fuera de línea [8]. En aplicaciones fuera de línea tanto los valores pasados como futuros de un registro son utilizados para reducir ruido. En contraste, en aplicaciones de tiempo real, los valores futuros no son utilizados.

2.4.1. Algoritmos de reducción de ruido en línea:

Filtro de Kalman

El filtro de Kalman es un tipo de reducción de ruido en línea ampliamente usado en estadísticas y control. En trabajos anteriores, se ha utilizado en sistemas industriales ciber-físicos, que se definen como sistemas geográficamente dispersos y críticos para la vida, donde muchos sensores y actores forman parte de una red para monitoreo en tiempo real [2]. Ding et al. resumen varios algoritmos distribuidos de filtros de Kalman, y su desempeño en sistemas como éste.

En su forma clásica, el filtro de Kalman describe el estado k de un proceso lineal dinámico a partir de $k-1$ según:

$$x_k = A_k x_{k-1} + B_k u_k + w_{k-1} \tag{2.1}$$

- A y x_{k-1} son la matriz y vector de estado.
- B y u son la matriz y vector de control.
- w es el ruido del proceso, que se asume como Gausiano $N(0, Q)$.

Se define el valor medido z, como:

$$z_k = H_k x_k + v_k \tag{2.2}$$

- donde H es la matriz de medición.
- v es el ruido medido, se asume Gausiano $N(0, R)$.

Luego, se definen los estados a priori y posteriori del paso z_k como $\hat{x}^-$ y $\hat{x}$.

El filtro estimado en el paso k se define como:

$$\hat{x}_k = K_k z_k + (1 - K_k)\hat{x}_{k-1} \tag{2.3}$$

Donde K es la matriz de ganancia a determinar por el sistema. Esto se realiza en dos pasos iterativos:

1. Actualización Temporal:

$$\hat{x}^- = A\hat{x}_{k-1} + Bu_k \tag{2.4}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{2.5}$$

2. Actualización de Medida:

$$K_k = P_k^- H_k^T [H_k P_k^- H_k^T + R_k]^{-1} \tag{2.6}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{2.7}$$

$$P_k = (I - K_k H_k)P_k^- \tag{2.8}$$

El filtro de Kalman fue originalmente pensado para sistemas lineales, pero la mayoría de los sistemas del mundo real son no lineales.[8] La extensión del filtro de Kalman para sistemas no lineales es el filtro de Kalman Extendido.

Filtro de Kalman Extendido

El sistema está gobernado por una ecuación diferencial estocástica no lineal:

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}) \tag{2.9}$$

y la medida correspondiente :

$$z_k = h(x_k, v_k) \tag{2.10}$$

Se usan Series de Taylor para linealizar y luego de aproximar el sistema se obtiene las actualizaciones temporales y de medida según 1 y 2 :

1. Actualización Temporal:

$$\hat{x}^- = f(\hat{x}_{k-1}, u_{k-1}, 0) \tag{2.11}$$

$$P_k^- = AP_{k-1}A^T + WQW^T \tag{2.12}$$

2. Actualización de Medida:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + VR_k V)^{-1} \tag{2.13}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0)) \tag{2.14}$$

$$P_k = (I - K_k H_k)P_k^- \tag{2.15}$$

Donde A y W son las matrices de derivadas jacobianas parciales de f respecto a x y w respectivamente. Mientras que H y V son las matrices de derivadas jacobianas parciales de h respecto a x y v.

2.4.2. Algoritmos de reducción de ruido fuera de línea

En el análisis de series financieras se suelen usar métodos de filtrado fuera de línea, siendo dos de los más típicos los métodos: SMA (Simple moving average, por sus siglas en inglés); y EMA (Exponential moving average). Audrius Dzikevičius y Svetlana Šaranda comparan estos dos métodos aplicados en dos índices de la bolsa Estadounidense, para predecir la bolsa de valores [3], indicando en el estudio que EMA entrega mejores resultados, al utilizar una evaluación de error sistemática.

Simple moving average

SMA toma el promedio de un número de muestra sobre un número de periodos.

$$SMA = \frac{V_1 + V_2 + ...V_n}{n} \tag{2.16}$$

Donde n es el largo de la ventana o número de periodos.

Exponential moving average

A diferencia de SMA que asigna igual peso a cada valor, EMA le asigna mayor ponderación a aquellos valores que son más recientes.

$$EMA_t = V_t * (\frac{\alpha}{1+n}) + EMA_{t-1} * (1 - \frac{\alpha}{1+n}) \tag{2.17}$$

Donde α es el factor de suavizado, a determinar.

En [3] los autores indican que un valor de α más alto es relevante para obtener un sesgo más bajo y para que la predicción sea más certera.

Descomposición Aditiva

Otro método fuera de línea para suavizar series temporales es la descomposición. Este modelo asume que la serie de tiempo se forma a partir de hasta 4 componentes aditivas, estas son:

- T_t : Variación de Tendencia: corresponde al comportamiento a largo plazo, de la serie temporal.
- S_t : Variación Estacional: movimientos periódicos (amplitud menor a un año), en torno a la tendencia que se atribuye, por lo general, a efectos de las estaciones del año.
- C_t : Variación Cíclica: fluctuaciones en torno a la tendencia de amplitud mayor a un año.
- R_t: Variación Aleatoria: comportamiento errático, no atribuible a ninguna de las 2 causas anteriores.

Un modelo aditivo sugiere que la serie es la suma de estas componentes:

$$y_t = T_t + S_t + C_t + R_t \tag{2.18}$$

Un modelo aditivo relativamente simple es el modelo clásico, que supone que la serie está compuesta de tres componentes: una cíclico-tendencial T_t, una estacional S_t y un residuo R_t. En una descomposición clásica, se asume que la estacionalidad se repite año tras año. [7]

Las componentes se obtienen a partir de las siguientes ecuaciones:

$$T_t = \begin{cases} 2m - MA & \text{si m es par} \\ m - MA & \text{si m es impar} \end{cases} \tag{2.19}$$

Donde m es el periodo estacional y MA es el promedio movible de la serie (*Moving Average*, por sus siglas en inglés).

$$S_t = f(y_t - T_t) \tag{2.20}$$

Donde S_t, para cada estación, se obtiene promediando la serie *detrended*[8] y ajustando los datos para que sumen cero. [7]

$$R_t = y_t - T_t - S_t \tag{2.21}$$

Donde R_t es el residuo.

[8]Serie original menos la tendencia.

Capítulo 3

Problema

El proyecto FONDEF define tres problemáticas o dificultades que se deben abordar en este trabajo de memoria; la primera corresponde a la dificultad de replicar soluciones existentes, ya que cada software es único para el sistema de aguas de su localidad, además de que soluciones existentes pueden contar con una calidad superior de sensores. Por lo que surge la segunda dificultad de este trabajo, que corresponde a plantear formas de abordar el ruido de la señal, asociado a los sensores de baja calidad, cuya relación en señales se conoce como SNR O Signal/Noise Ratio, por sus siglas en Inglés. Por último, se define la necesidad de una solución que escale a un despliegue masivo de sensores, ya que se requiere almacenar y monitorear un gran volumen de datos.

3.1. Replicabilidad

La dificultad de replicar soluciones existentes se debe a que cada sistema considera variables particulares como la **geografía**, la **cobertura** de red del lugar, y una **regulación** sanitaria completamente distintas. Sumado a lo anterior, se debe considerar que soluciones existentes pueden contar con **calidad** diferente de sensores.

Para ejemplificar este problemas se cuenta con datos de mediciones de temperatura y conductividad eléctrica de distintas geografías del país, graficadas en las Figuras 3.1 y 3.2. Las muestras se toman utilizando equipos de calidad similar para cada estación, sin embargo, se puede observar que ambos parámetros varían considerablemente en cada estación de medición, lo cual puede obedecer a condiciones geográficas y meteorológicas de la localidad, como por ejemplo una relación entre la altitud y la temperatura, y no necesariamente a un evento de contaminación. Por lo que una solución existente, ajustada a una localidad específica, podría arrojar falsas alarmas al aplicarla en una localidad diferente.

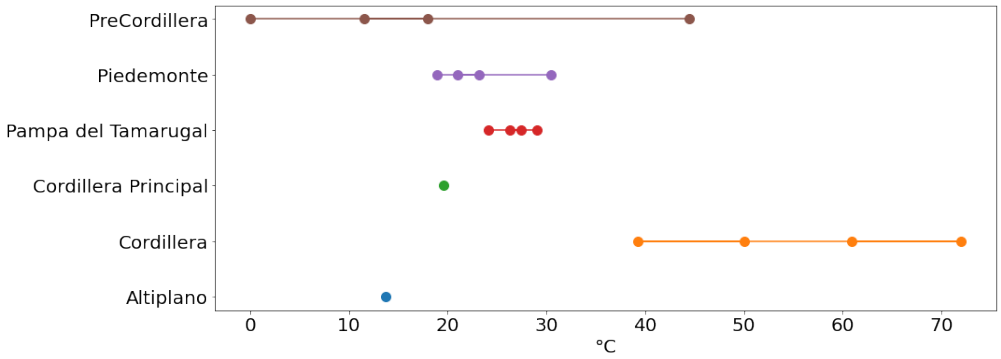


Figura 3.1: Niveles de temperatura en estaciones con distinta ubicación geográfica. Utilizando datos entregados por un equipo de hidrogeología del Departamento de Geología.

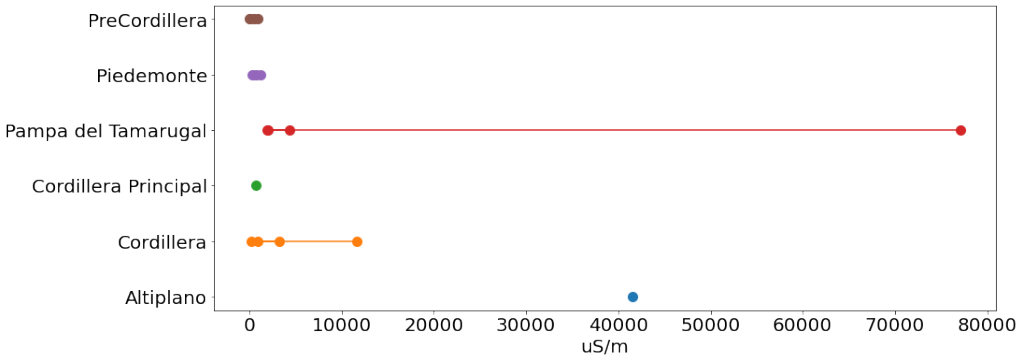


Figura 3.2: Niveles de conductividad eléctrica en estaciones con distinta ubicación geográfica. Utilizando datos entregados por un equipo de hidrogeología del Departamento de Geología.

3.2. Escalabilidad

Es necesario considerar el despliegue masivo de sensores en distintas geografías, al momento de diseñar un sistema de recopilación y procesamiento de datos. Teniendo en cuenta la cobertura de red, de la localidad misma donde se instalan los sensores. Por lo tanto, el trabajo de esta memoria debe tener en consideración la integración con dos redes LPWAN, **Sigfox** y **LoRaWAN**, para poder acceder a una mejor cobertura.

3.3. Ruido y Fallas

Por último, otro gran desafío de este trabajo, es el manejo de ruido y errores asociado al uso de sensores de baja calidad. A nivel de *backend*, se manejan dos tipos de errores: ruido en la **medición**, y una **falla** en el dispositivo. Mientras que los errores en la **comunicación** se manejan en una capa inferior (capa de red).

El primer error, se refiere al ruido que distingue el valor real del valor medido por el sensor. Para abordar este problema es necesario implementar algoritmos de filtrado de ruido como los descritos en la sección anterior. Mientras que una falla en el dispositivo corresponde a un comportamiento anómalo en el sensor, en particular se abordan medidas fuera del rango indicado por el sensor.

En cuanto se refiere a evaluar la efectividad de estos filtros de ruido, sobre datos donde no se cuenta con la señal real, formas completamente confiables de verificar el filtrado de ruido no están disponibles [6]. Una forma indirecta de evaluar su efectividad, es utilizando un modelo predictivo entrenado sobre los datos ruidosos y comparar el valor predicho con el valor original, como se proponen Karunasingha et al. [8].

Capítulo 4

Solución Propuesta

4.1. Arquitectura Propuesta

Para poder monitorear la calidad de agua en fuentes subterráneas y superficiales, el proyecto FONDEF propone la construcción de una arquitectura con **tres módulos**:

1. Una **red de sensores IoT** de bajo costo (configurables al sitio de instalación y los distintos puntos de medición).
2. Un **backend de adquisición** (compatible con distintas configuraciones en la red de sensores).
3. Una **sistema experto** que cuente con algoritmos capaces de predecir alarmas.

Esta memoria se enfoca en el **backend de adquisición**, fundamental para el éxito del proyecto, cuya solución propuesta contempla cinco bloques relevantes: un **protocolo** de comunicación en un servidor, una **API**, una **base de datos**, algoritmos de **filtrado** y una **interfaz** gráfica. Estos cinco bloques conforman la arquitectura propuesta para la solución.

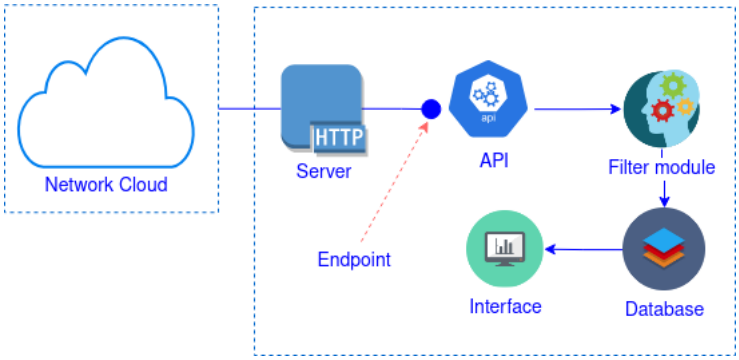


Figura 4.1: Arquitectura de la solución.

En la Figura 4.1 se observa el *backend* de adquisición conectado con la arquitectura de la red de sensores mediante un protocolo de comunicación para la transmisión de datos en Internet. Esta red se suscribe a la API publicando, cada cierto tiempo, los valores medidos en terreno, que son recibidos por un *endpoint*. Luego, la API dirige estos registros al módulo

de filtrado que se encarga de eliminar el ruido e imperfecciones de éste y escribir el nuevo valor en una base de datos. Asimismo, estos datos son consumidos por un tablero de control, que permite realizar análisis, obtener tendencias y crear gráficos.

4.1.1. Tecnología Propuesta

El *backend* desarrollado debe conectarse a la aplicación de dos redes LPWAN: la nube de **Sigfox** y **TTN**(conectada a **LoRaWAN**). Para cumplir este objetivo se proponen las tecnologías de las Figuras 4.2 y 4.3 para implementar la arquitectura descrita. La Figura 4.2 presenta la tecnología utilizada para una solución integrada al protocolo MQTT, mientras que la Figura 4.3 muestra una solución para HTTP[1].

Se cuenta con un servidor dedicado exclusivamente al proyecto, en el cual se levanta la API para recolectar los datos procedentes de la red. Por lo tanto, en esta memoria se diseñan e implementan las soluciones locales con la tecnología propuesta en las Figuras 4.2 y 4.3. Mientras que alternativas que contemplan tecnología en la nube sólo son evaluadas y propuestas para trabajos futuros.

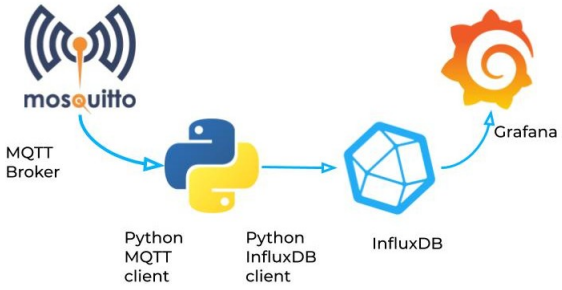


Figura 4.2: *Diseño de conexión a través de un Broker MQTT*: Broker Mosquitto conectado a una API en Python utlizando el cliente MQTT Pahoo [2]. Conexión a base de datos InfluxDB que alimenta tablero de control Grafana.

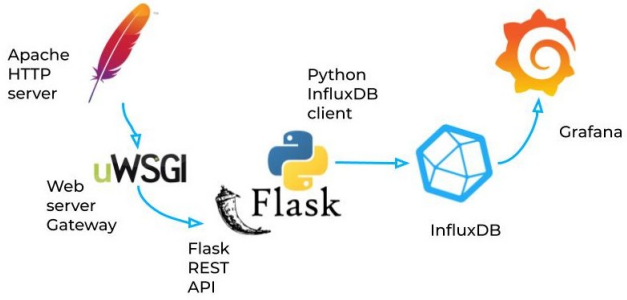


Figura 4.3: *Diseño de conexión a través de servidor web HTTP*: Servidor HTTP Apache conectado a un *backend* en Flask a través de WSGI. Conexión a base de datos InfluxDB que alimenta tablero de control Grafana

La principal diferencia entre ambos servidores es que MQTT funciona con el principio de publicación y suscripción, donde los clientes no cuentan con un enlace directo sino que se

[1]Ver sección 2.3
[2]https://flask.palletsprojects.com/en/1.1.x/

suscriben a tópicos. Mientras que HTTP si requiere definir un enlace o *endpoint* donde son dirigidos los mensajes. El principio de MQTT es muy útil ya que permite que cada nodo de la red se comporte como un publicador, distribuyendo la información de forma más eficiente.

Capítulo 5

Implementación de la Solución

En esta sección se describen las decisiones de diseño sobre cada componente de la arquitectura propuesta en la sección 4.1 e ilustrada en la Figura 4.1, mostrando todas las alternativas que se prueban, y también aquellas que se descartan.

5.1. Servidor y protocolo de comunicación

5.1.1. Levantamiento de servidor MQTT

Se levanta un servidor de mensajería para el protocolo MQTT utilizando Mosquitto, sin embargo, por razones de diseño[1], se decide no utilizar este servicio para el proyecto FONDEF, por lo que **no se ahonda en esta solución más allá de esta sección**.

Mosquitto es un proyecto *open source*, desarrollado en el lenguaje C por Eclipse Foundation, regularmente mantenido y versionado. Es descargable en muchos sistemas operativos y compatible con muchas librerías y *pluggins*, además, cuenta con su propia implementación de clientes. EMQx también es un *broker open source*, programado en Erlang por EMQ Project, es mantenido de forma frecuente y descargable en MacOS, Windows y algunas distribuciones de Linux. Además, es compatible con los clientes de Mosquitto y el cliente eMQTT[2], que debe ser descargado por separado. Se distingue del anterior al ofrecer un servicio pagado en la nube, llamado EMQx Cloud, que indica ser masivamente escalable. Al observar ambos repositorios en Github, Mosquitto es el más popular de los dos.

5.1.2. Levantamiento de servidor HTTP

Esta solución contempla la utilización del protocolo HTTP, usando como servidor web Apache, para comunicar con el *backend* de adquisición.

Dos servidores web que predominan en la implementación local de HTTP son Apache y Nginx. Apache es una plataforma gratuita y *open-source* que lleva más de 20 años en uso,

[1] Revisar sección 5.1.3 .
[2] https://github.com/emqx/emqtt .

algunas ventajas de Apache es que viene por defecto en algunos sistemas operativos, cuenta con amplia documentación y es modificable (recibe nuevas features). Por otro lado Nginx ha ganado terreno no sólo en el desarrollo web, sino que también puede ser usado como *proxy* reversible y servidor de correo.

Para poder comunicar un *backend* programado en Python con Apache, es necesario utilizar un *web server gateway interface* (WSGI, por sus siglas en inglés). WSGI reenvía los mensajes del servidor web, hacia aplicaciones o *frameworks* escritos en Python.

Procesadores, threads y escalabilidad

Al configurar WSGI en el servidor es necesario especificar el número de **procesadores** y **threads** asignados a la aplicación. Ambos parámetros ayudan a aumentar la concurrencia y se asignan tomando en consideración cuánto tráfico tiene la aplicación. Por ejemplo, si se usa un sólo thread, el número de procesadores asignado debe aguantar la suma de *requests* por segundo que el sitio debe manejar.

El número máximo de procesadores que se puede asignar está condicionado a cuánta memoria usa cada procesador. Si el número de procesadores que se requiere es mayor a la memoria disponible en el servidor, entonces es necesario utilizar más máquinas.

En el caso de necesitar escalar la API más allá de las capacidades de la máquina, es conveniente usar un Servicio de Infraestructura (IaaS, *Infrastracture as a Service*, por sus siglas en Inglés) de un tercero (Amazon, Google Microsoft cloud services). Por lo general se suele arrendar una Máquina Virtual, con un espacio asignado, y cuando se desea escalar, se puede contratar otra. Alternativamente, se puede externalizar la base de datos a una nube y así contar con más memoria disponible para servir la app.

5.1.3. Conexión con la nube de la red

Consola de TTN para LoRaWAN

A través de la consola de TTN se puede registrar y monitorear las *gateways* y aplicaciones de la red. El proyecto FONDEF cuenta con una aplicación en la nube de TTN, como se puede observan en las Figuras 5.1 y 5.2.

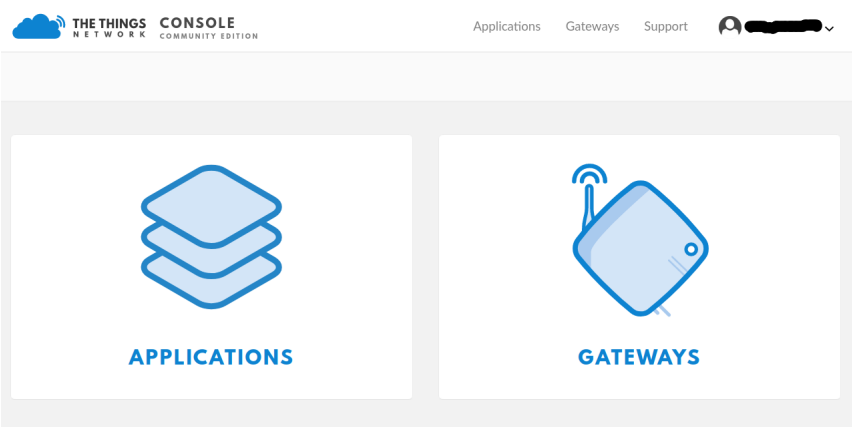


Figura 5.1: Consola de TTN con la opción de acceder a las aplicaciones y *gateways* del sistema.

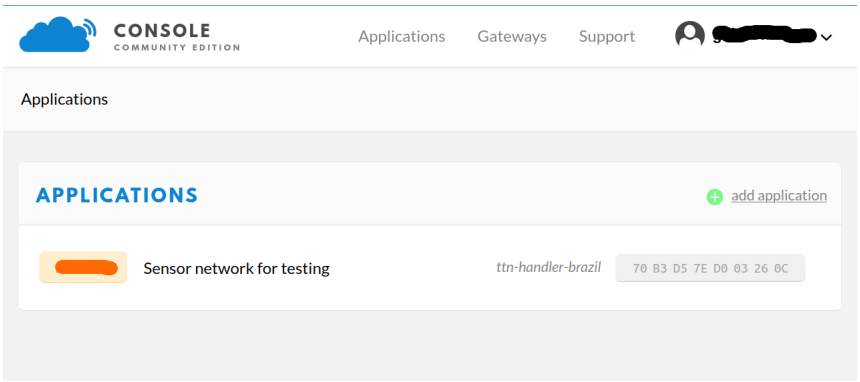


Figura 5.2: Aplicación de testeo del proyecto FONDEF.

Una aplicación cuenta con múltiples **dispositivos**, un dispositivo es un **nodo** al cual llega la información de una **estación**, donde puede haber uno o más sensores. En la Figura 5.3 se observa que la aplicación del proyecto FONDEF cuenta hasta el momento con un dispositivos de prueba.

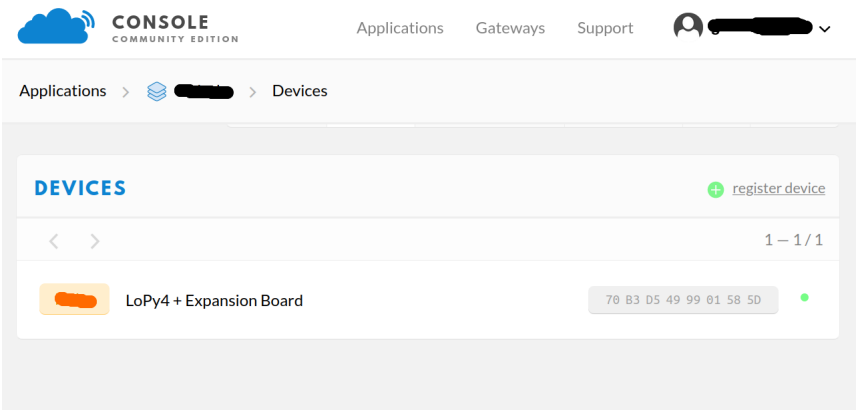


Figura 5.3: Dispositivo de la aplicación del proyecto FONDEF.

Cada aplicación puede integrarse a otras aplicaciones a través de un protocolo de comunicación. En la Figura 5.4 se observa una integración vía HTTP instanciada.

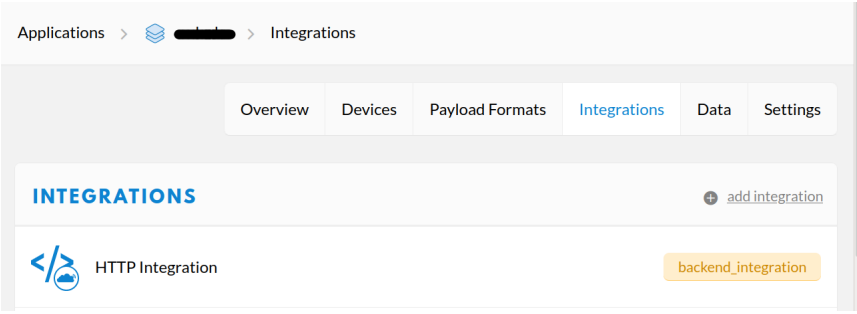


Figura 5.4: Integración HTTP para la aplicación del proyecto FONDEF.

Es importante notar que TTN sólo permite agregar integraciones a la aplicación y no a los dispositivos, lo que quiere decir que todos los mensajes de tal aplicación son dirigidos al mismo *endpoint*. Para lograr asignar un *endpoint* a cada dispositivo, TTN requiere una integración a Node_Red [3]: una herramienta de control de flujos que permite conectar dispositivos de hardware, APIs y servicios en línea. Sin embargo, esta integración ha sido descontinuada de la consola de TTN [4], siendo difícil implementar una integración en el marco de esta memoria.

En un protocolo basado en el principio de publicación y subscripción como MQTT, el balance de cargas depende del manejo inteligente de los tópicos. Es decir, donde publicadores y subscriptores balancean la carga de los mensajes. Sin una herramienta como Node_Red no es posible balancear estas cargas, por lo que se decide no utilizar el protocolo MQTT, y dejar esta integración para un trabajo futuro.

5.2. API

5.2.1. Backend

Flask es un micro *framework* web para Python, que provee todas las herramientas necesarias para realizar desarrollo web. Se diferencia de otros *frameworks* populares para Python (por ejemplo Django) al ser liviano, logrando acelerar el proceso de desarrollo.

Es importante notar que se escoge el lenguaje Python para programar la API. Existen dos razones para tomar esta decisión:

1. El agradable manejo de ambientes virtuales de Python, gracias a la existencia de varios administradores de paquetes para este lenguaje. Un mejor manejo de ambientes permite una mejor escalabilidad de la solución, ya que **encapsula** la API y reduce los conflictos con otras aplicaciones existentes.
2. El extenso número de librerías para Python, dedicadas al procesamiento de datos, como pandas[5] o numpy[6], las cuales serán fundamentales al momento de implementar algoritmos de filtrado, y preprocesar datos.

[3]https://www.thethingsnetwork.org/forum/t/mqtt-in-node-red-howto/39909 .
[4]https://www.thethingsnetwork.org/forum/t/why-is-node-red-integration-no-longer-supported/39390/7 .
[5]https://pandas.pydata.org/docs/.
[6]https://numpy.org/.

5.2.2. Endpoints y esquema de datos

El *backend* puede recibir paquetes procedentes de la aplicación de LoRaWAN, o Sigfox, a través del protocolo HTTP. Cada red tiene un esquema diferente para transmitir datos, por lo que es necesario definir dos *endpoints* HTTP diferentes para procesar los mensajes POST:

1. {url}/data/sigfox para paquetes procedentes de la aplicación de Sigfox (donde **url es el dominio de la aplicación**).
2. {url}/data/lorawan para los paquetes procedentes de LoRaWAN.

Al recibir un paquete procedente de la red, el *backend* verifica que presente el esquema correspondiente a tal aplicación, como se muestra en los Códigos 5.1 y 5.2. El esquema valida dos cosas: que los campos correspondan a los tipos señalados, y que el paquete contenga los requisitos indicados.

5.2.3. Esquemas JSON de paquetes provenientes de redes LPWAN

Código 5.1: Esquema JSON de paquete proveniente de una red Sigfox

```
{
  "required": ["deviceType", "device", "data"],
  "properties": {
    "deviceType": { "type": "string" },
    "device": { "type": "string" },
    "time": { "type": "string" },
    "data": { "type": "string" },
    "seqNumber": { "type": "number" },
    "ack": { "type": "number" }
  }
}
```

En el esquema del Código 5.1 los campos requisitos son:

- deviceType: tipo de dispositivo que transmite la información.
- device: id del dispositivo que transmite el paquete.
- data: datos recopilados por los sensores de la estación y enviados al dispositivo.

Código 5.2: Esquema JSON de paquete proveniente de una red LoRaWAN

```
{
  "required": ["app_id", "dev_id", "payload_raw"],
  "properties": {
    "app_id": { "type": "string" },
    "dev_id": { "type": "string" },
    "hardware_serial": { "type": "string" },
    "port":{ "type": "number" },
    "counter": { "type": "number" },
    "payload_raw": { "type": "string" },
    "metadata": { "type": "object" },
    "downlink_url": { "type": "string" }
  }
}
```

En el esquema del Código 5.2 los campos requisitos son:

- app_id : id de la aplicación de la consola de LoRaWAN.
- dev_id: id del dispositivo que realiza la transmisión.

- payload_raw: datos recopilados por los sensores de la estación y enviados al dispositivo.

5.2.4. Externalización de información

Los datos recopilados por la estación se encuentran en los campos **data** y **payload_raw** del Código 5.1 y 5.2 respectivamente. Como ambas redes tienen *payloads* muy limitados por paquete, los campos mencionados contienen información codificada, que puede ser asociada con cierta información estática del sensor, que no es transmitida con cada paquete.

Los datos codificados son procesados por un módulo de decodificación que entrega una lista de tuplas que contienen: el ID del sensor, el *timestamp* de la medida y el valor correspondiente. Con este ID se puede acceder a información del sensor que se encuentra almacenada en una tabla de una base de datos Myslq, que incluye los siguientes campos: el nombre del sensor, su rango de medición, unidad de medición, unidad del rango, el parámetro que mide, el error indicado, y el tipo de error (porcentual, intervalo).

id	nombre	rango_inf	rango_sup	unidad	unidad_rango	param	error	tipo_error

Tabla 5.1: Información del sensor.

En otra tabla se guarda la información del cliente, sector, estación y tipo de estación, indexado por el código del dispositivo, como se observa en la Tabla 5.2. Esta información puede ser accedida con el id del dispositivo, que corresponde a los campos **device** y **dev_id** de los paquetes Sigfox y LoRa respectivamente.

id	cliente	sector	estación	tipo_estación

Tabla 5.2: Información de la estación:

5.3. Diseño de base de datos

5.3.1. Caracterización de los datos

Al diseñar la base de datos del sistema que almacena los valores medidos en terreno, se toman en consideración diferentes aspectos de los registros, considerando que los parámetros a medir son al menos cinco: pH, turbidez, presión (nivel o caudal), temperatura y conductividad eléctrica.

Un aspecto relevante para definir la estructura de la base de datos, es el comportamiento de los parámetros. En particular, se evalúa si existe o no, correlación entre las variables, mediante el análisis de mapas de calor, cómo se muestra en la Figura 5.5:

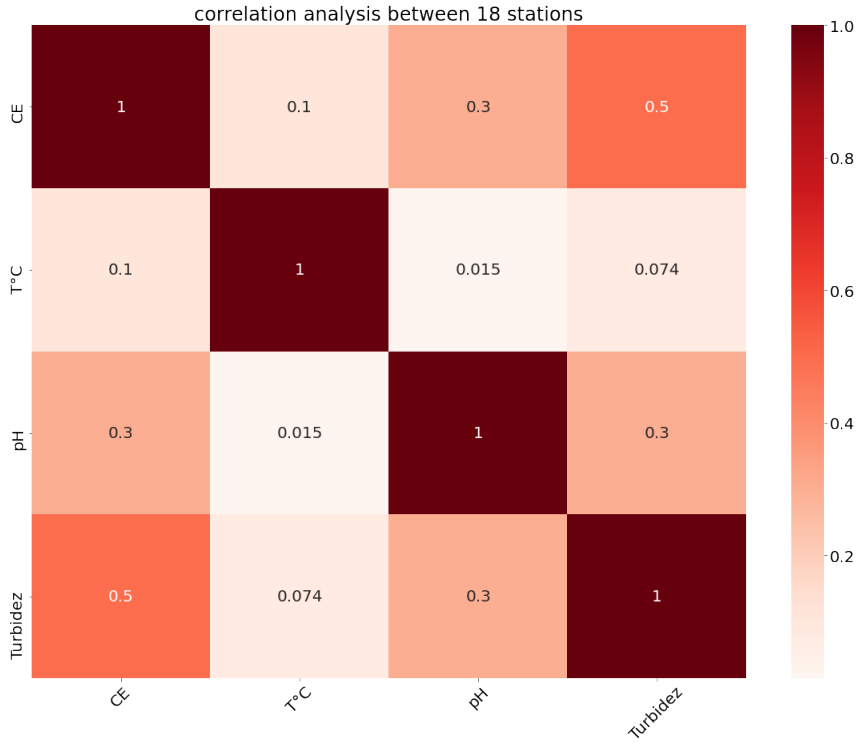


Figura 5.5: Correlación entre los parámetros de 18 estaciones de medición de agua potable rural, utilizando datos provistos por un equipo de hidrogeología del departamento de Geología. Experimentos realizados por Miguel Solís Dr. en Ingeniería Informática.

De la Figura 5.5, se deduce que la temperatura no presenta una correlación significativa con otros parámetros, por otro lado, el pH presenta una correlación de 0.3 con la turbidez y la conductividad, lo que se considera moderada o baja. Por último, la turbidez tiene una correlación de 0.5 con la conductividad, lo cual no es suficiente como para indicar una relación relevante.

Los resultados evidencian que no existe suficiente relación entre los parámetros, al menos en las estaciones analizadas. La correlación entre variables podría fluctuar al utilizar otras estaciones, pero esto no indica una dependencia ya que los resultados pueden verse afectados por la ubicación geográfica de la estación.

Otro aspecto importante, al momento de diseñar la base de datos, es que no todas las variables de una estación son monitoreadas con la misma frecuencia. Por lo tanto no es posible indexar más de una variable bajo el mismo tiempo, sin perder precisión en esta llave. Dado lo anterior se trabaja cada parámetro en una tabla individual, ya que cada uno define un modelo diferente.

5.3.2. Elección de base de datos

Debido a que los registros a trabajar son series de tiempo, se propone utilizar una base de datos de series de tiempo (TSDB, por sus siglas en inglés). Este tipo de base de datos tiene mejor compresión que una relacional con registros temporales, además de transmitir y guardar datos en concordancia con el ciclo de recolección [9], lo cual trae un aumento en el desempeño significativo.

Fadhel et al. [9] compara 8 bases de datos de series de tiempo: OpenTSDB[7], InfluxDB[8], ElasticSearch[9], Kdb+[10], RRD-tool [11], Graphite [12], Prometheus[13] y DalmatinerDB[14], para almacenar datos de calidad de agua. En el trabajo se concluye que la más apropiada para almacenar este tipo de datos es InfluxDB, en base a las múltiples facilidades listadas: es fácil de instalar, fácil de mantener y provee múltiples formatos de interfaz, además de tolerar grandes números de consultas debido a su bajo consumo de recursos (ram y cpu). Dado esto, parece una buena alternativa utilizar **InfluxDB** como base de datos en esta memoria.

Las series de tiempo en InfluxDB se conforman por *data points*, que corresponden a una unidad de información discreta. Un punto se identifica como **único** por su medida o *measurement* (considerada como el nombre de la serie de tiempo almacenada), set de etiquetas (*tag set*) y *timestamp*. Por lo tanto, una serie de tiempo estaría indexada por su set de etiquetas y timestamp.

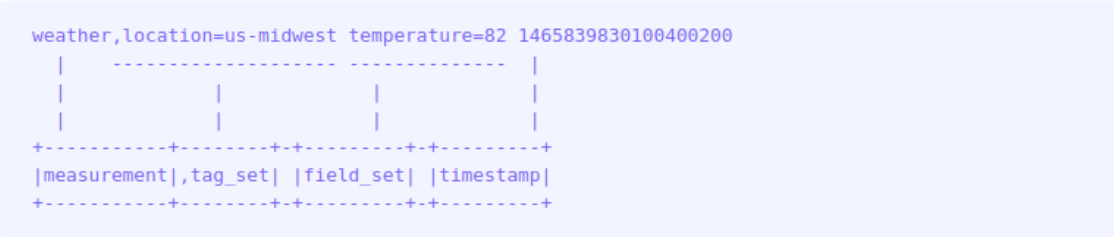


Figura 5.6: Data Point influxDB, imagen obtenida de la documentación de Influx[15]

.

[7] http://opentsdb.net/
[8] https://www.influxdata.com/
[9] https://www.elastic.co/
[10] https://kx.com/
[11] https://oss.oetiker.ch/rrdtool/
[12] https://graphiteapp.org/
[13] https://prometheus.io/
[14] https://dalmatiner.io/
[15] https://docs.influxdata.com/influxdb/v1.8/write_protocols/line_protocol_tutorial/

En la Figura 5.6, se puede observar un *data point* de InfluxDB, con sus respectivos campos: *measurement*, *tag_set*, *filed_set* y *timestamp*.

5.3.3. Diseño y limitantes

Es importante que la base de datos no sólo logre almacenar datos etiquetados por estación, sino que también entregue la opción de realizar consultas que revelen información importante de los datos. A diferencia de una base de datos relacional, en InfluxDB sólo se puede realizar consultas condicionales sobre aquellos campos que forman parte del set de etiqueta. Al formar parte del índice de un registro, no es recomendable incluir campos de alta cardinalidad en este grupo, ya que podrían llegar a consumir mucha memoria.

Para poder identificar la procedencia de un registro sólo se necesita el código de la estación, esta es la etiqueta de mayor cardinalidad. Sin embargo, es conveniente una mayor granularidad en la identificación de los datos, por lo que se agregan los campos cliente y sector. Por último se agregan los campos tipo_estación, y unidad para identificar el tipo de acuífero (subterráneo, superficial), y la unidad del valor muestreado, respectivamente.

La cardinalidad de los datos es entonces:

$C(cliente, sector, estacion, tipo\_estacion, unidad)$ con $C =$ combinaciones.

Como el código de la estación es único y no se puede repetir para clientes, sectores o tipos de estaciones diferentes, no se tiene más de |estacion| combinaciones. Por lo tanto no existe un problema de cardinalidad.

Por lo que, la base de datos almacena la siguiente información:

measurement	tag-set					field-set	timestamp
parámetro	cliente	sector	estación	tipo_estación	unidad	valor	tiempo

Donde:

- **parámetro**: *string*, corresponde al nombre del parámetro medido que puede tomar los valores [TU, pH, T, CE, P, N, C,][16].
- **cliente:** *string*, nombre de la institución sobre la cual se realiza el monitoreo.
- **sector**:*string*, corresponde al nombre de la fuente de agua monitoreada.
- **estación**: *string*, código del punto en la fuente de agua, en la cual se realiza la medición.
- **tipo_estación** : *string*, corresponde al tipo de punto de medición en el cual se obtuvo la muestra. Puede corresponder a fuente subterránea (pozo), o superficial (río).
- **unidad**: *string*, unidad en la que se mide la muestra.
- **valor**: *float* con 3 posiciones decimales, corresponde al valor asociado al parámetro medido.
- **tiempo**: *timestamp*, corresponde a la fecha y hora de la muestra, en nanosegundos.

[16]Turbiedad, pH , Temperatura, Conductividad Eléctrica, Presión, Nivel, y Caudal

5.3.4. Implementación

Creación de bases de datos

La forma más fácil de crear una base de datos en InfluxDB es a través de la consola o **influx CLI**, ingresando en la terminal el comando **influx**.

Para crear una base de datos se utiliza el comando CREATE DATABASE que puede recibir varios campos opcionales, que definen la **política de retención**:

```
1 CREATE DATABASE <database_name> [WITH [DURATION <duration>] [REPLICATION <n>] [SHARD ↩
    DURATION <duration>] [NAME <retention-policy-name>]]
```

donde:

- **Duration:** corresponde al tiempo en segundos, en que se guarda la data.
- **Replication**: en el caso de usar un InfluxDB cluster, corresponde a la replicación de la base de datos entre múltiples *shards*.
- **Shard duration**: define la duración mínima para que InfluxDB empiece a utilizar un nuevo *shard*.

Por defecto InfluxDB usa una política de retención llamada autogen, que mantiene los datos por siempre (política de retención infinita), con un *shard duration* de 7 días y factor de replicación 1. Para efectos de esta memoria se utiliza la política de replicación autogen de InfluxDB.

Se crea una instancia de InfluxDB en el servidor, con dos bases de datos: una para almacenar los datos sin filtrar llamada **monitor_agua**, y otra para guardar los registros tras aplicar un filtrado llamada **monitor_agua_filtrado**. La primera base de datos es útil al momento de entrenar modelos de detección de anomalías y diagnóstico de fallas. Mientras que la segunda sirve para alimentar visualizaciones y como entrada de modelos predictivos.

Cada base de datos guarda el registro de al menos cinco tablas : pH, T , CE, TU, P, y dos tablas opcionales : N y C que guardan registro del Nivel y Caudal, como se observa en la Figura 5.7.

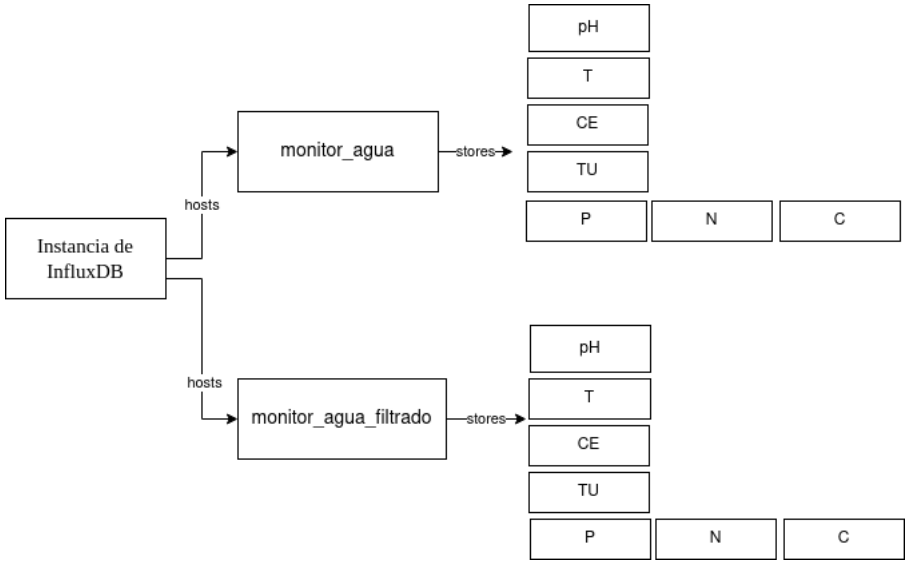


Figura 5.7: Almacenes de datos.

Por defecto InfluxDB genera métricas internas y las guarda en una base de datos llamada _internal. Se utiliza esta base de datos para monitorear el flujo de escrituras durante la validación en la sección 6, y detectar problemas. Sin embargo, InfluxDB no recomienda usar esta base de datos en producción, ya que genera un *overhead* innecesario.

Escritura y conexión

Para escribir a la base de datos se utiliza el cliente InfluxDBClient de la librería influxdb [17].

```
1 client = InfluxDBClient(host=_host, port=_port, database=_database)
```

Este cliente recibe la IP y el puerto donde corre la instancia de Influx, además del nombre de la base de datos. Se instancian dos clientes, uno para cada base de datos creada, y con el método client.write() se realizan las escrituras. Es importante mencionar, que el método write() no es *thread safe*, por lo tanto se instancian dos colas de prioridad, para realizar las escrituras de forma secuencial a cada base de datos. Dos *background jobs* desencolan los elementos de su cola respectiva, y llaman a la función write(), existiendo paralelismo entre las escrituras y otros trabajos del *backend*.

En cada llamado a write() se escriben uno o más *data points* recibidos en un paquete, aprovechando la concurrencia que ofrece InfluxDB.

[17] https://github.com/influxdata/influxdb-python

5.4. Filtrado de datos

Cada vez que llega un dato se realizan dos cosas:

La primera es revisar que el dato esté dentro del rango en el cual trabaja el sensor que reportó tal medida. Para esto se consulta la tabla de sensores, usando el id del sensor, si el dato se encuentra fuera de rango existen tres formas de proceder:

- **Eliminar**: el dato no se escribe a la base de datos
- **Duplicar**: se escribe la medida anterior reportada por el sensor, y guardada en un caché.
- **Truncar**: se escribe el límite inferior o superior del sensor.

Como segundo, se intenta filtrar el ruido que puede ocasionar la calidad de los sensores, con un módulo de filtrado que implementa algunos de los algoritmos mencionados en la sección 2.4. Este módulo recibe los datos procedentes de la aplicación de la red de sensores y los filtra de acuerdo a algún algoritmo especificado en el *backend*, para luego escribirlos a la base de datos **monitor_agua_filtrado**.

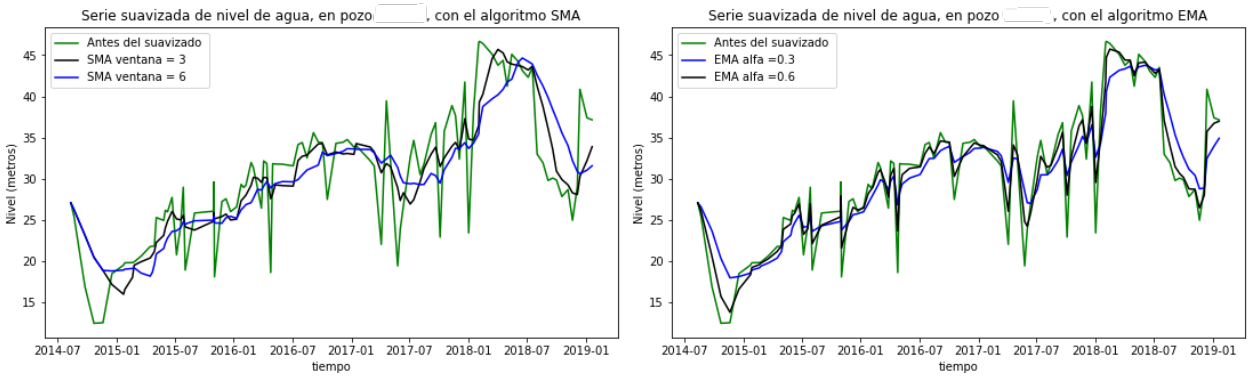
En la siguiente sección, se prueban algunos de los algoritmos mencionados en la sección 2.4.

5.4.1. Algoritmos

Como se menciona en la sección 2.4, existen implementaciones en línea y fuera de línea para filtrar ruido. Estas últimas a veces son llamadas "suavizado" ya que tienen el efecto de aplanar la serie utilizando ventanas de tiempo.

Ejemplos de suavizado

Como una primera aproximación se utilizan datos históricos facilitados por Arcadis, para observar el resultado de aplicar cada algoritmo. En la Figura 5.8 se observa el suavizado de registros de Nivel de un pozo, con los algoritmos SMA y EMA respectivamente.



(a) Suavizado con algoritmo SMA.

(b) Suavizado con algoritmo EMA.

Figura 5.8: Suavizado de datos de pozo de minera (anonimizados), facilitados por Arcadis

De la Figura 5.8a se observa que el algoritmo SMA filtra imperfecciones de forma agresiva, sin embargo, no es tan preciso y se aleja de la serie original. Por otro lado EMA logra aproximarse a la serie original, y suavizar las cumbres y valles.

Otro algoritmo de suavizado es la descomposición aditiva utilizando ventanas deslizables. En el siguiente ejemplo se utiliza una implementación de descomposición clásica de tres componentes utilizando la librería stastmodels [18], pasando como parámetro 5 periodos, uno por cada año observado.

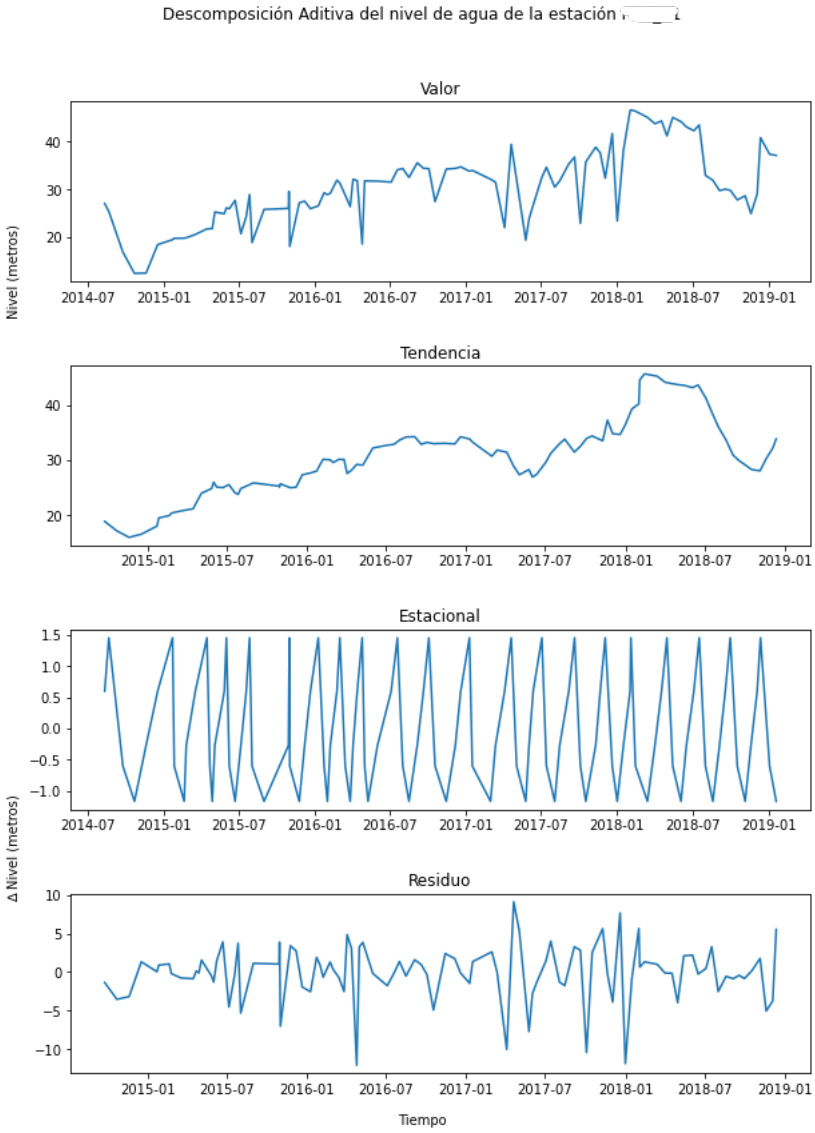


Figura 5.9: Suavizado con descomposición aditiva, de datos de pozo de minera (anonimizados), facilitados por Arcadis.

En la Figura 5.9 se observa cómo el modelo descompositivo clásico separa la serie original en una componente tendencial, una estacional y un residuo. El algoritmo parece ser mucho más agresivo que los dos anteriores, eliminando todas las fluctuaciones pronunciadas. La

[18]https://www.statsmodels.org/stable/index.html

componente estacional refleja que a lo largo de un año el nivel del agua varía entre -1 y 1.5 metros de la tendencia para ese año, mientras que el residuo refleja eventos puntuales donde se alcanzó una variación de hasta 10 metros. A largo plazo este método podría estar filtrando acontecimientos importantes para detectar eventos de contaminación, por lo que se debe evaluar su efectividad en intervalos temporales más acotados.

5.4.2. Experimentos

Selección de datos

Los datos históricos provistos por Arcadis no sirven para un análisis comparativo de desempeño de los algoritmos de filtrado ya que poseen una frecuencia de muestreo demasiado baja.

Por lo que, para realizar pruebas, se utiliza un *dataset* de muestras tomadas por un sensor de temperatura DS18B20, con una frecuencia de muestreo de 1 minuto. Se realiza un *subsampling* obteniendo los datos de 24 horas aproximadamente, como se observa en la Figura 5.10

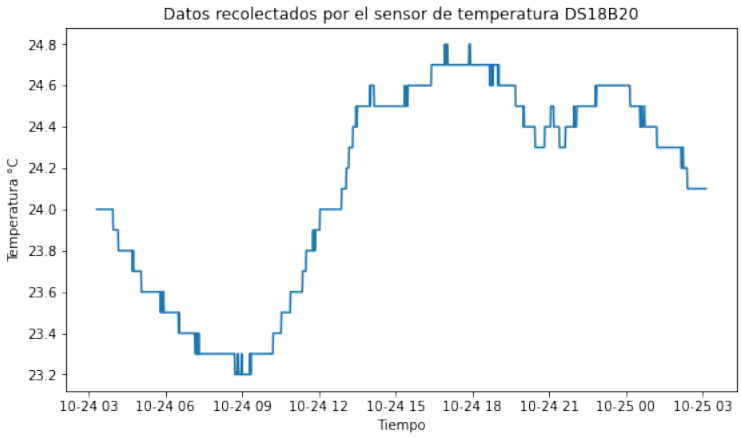


Figura 5.10: Subsample seleccionado, datos de 1 día.

El sensor DS18B20 tiene un rango de -10 a 80 °C y un error de hasta +/-0.5 °C sobre la señal real, por lo tanto pueden existir fluctuaciones en la medición que no se corresponden con la señal real, que es desconocida. En la Figura 5.10 se observa que la señal en todo el día varía en un rango de 1.6 grados celsius, el cambio se observa como una señal escalonada, debido a la precisión del sensor.

Definición de parámetros

Sobre esta señal se aplican 4 algoritmos con parámetros customizables:

- SMA: ventana tamaño n con $n \in [5, 10, 60, 120]$.
- EMA: con $\alpha \in [0{,}1, 0{,}3, 0{,}6, 0{,}8]$.
- Descomposición Aditiva: con periodo $p \in [2, 4, 6, 12]$.

- Filtro de Kalman: con covarianza de proceso $Q \in [0{,}5, 0{,}05, 0{,}005, 0{,}001]$[19].

Predicción con Feed Forward multiperceptron network

Se instancia una red neuronal de entrada 10 utilizando la librería sklearn[20], y se entrena con el 70 % del *dataset* seleccionado. Luego se prueba la red con un set de testeo que corresponde al 30 % de los datos, como se observa en la Figura 5.11.

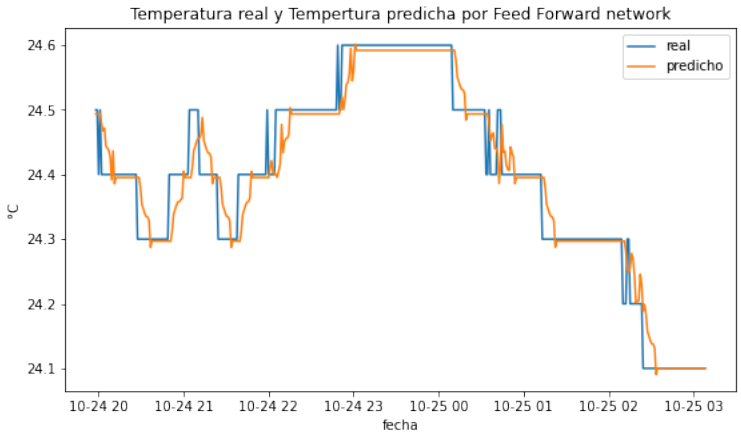


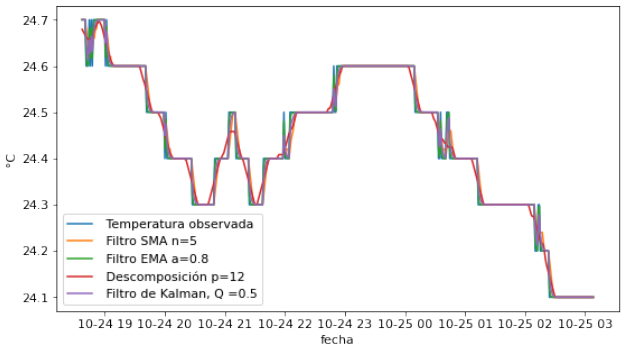
Figura 5.11: Datos reales y predichos por la red.

Se prueba la competitividad de cada algoritmo pasando como input de la red el 30 % del *dataset* filtrado por cada algoritmo, como se propone en la sección 3.3. En el caso del algoritmo descompositivo, se rellenaron los campos NA con los valores adyacentes, ya que la red debe recibir puntos equidistantes. Para cada algoritmo de filtrado se prueban varias configuraciones y se presentan los mejores resultados en la sección 6.3.

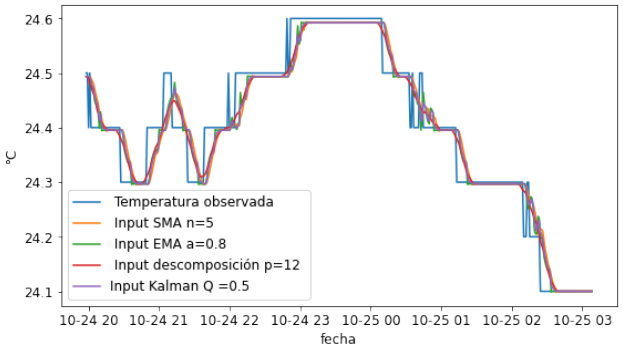
En la Figura 5.12 se observa 3 resultados, en la primera columna se presenta el *dataset* filtrado por cada algoritmo, para cierta configuración de parámetro, mientras que la segunda columna corresponde a la predicción que genera la red al recibir cada *dataset* como Input. La primera fila presenta el experimento que obtuvo los mejores resultados, mientras que la última presenta aquellas configuraciones con los peores resultados.

[19]Por simplicidad el resto de los parámetros se dejan fijos.
[20]https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

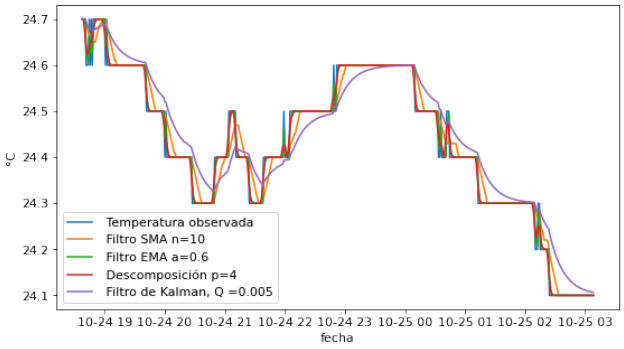
Temperatura observada por sensor DS18D20 y filtrada con distintas configuraciones para cada algoritmo (columna izquierda). Temperatura predicha por red neuronal *Feed Forward*, sobre los datos filtrados (columna derecha).



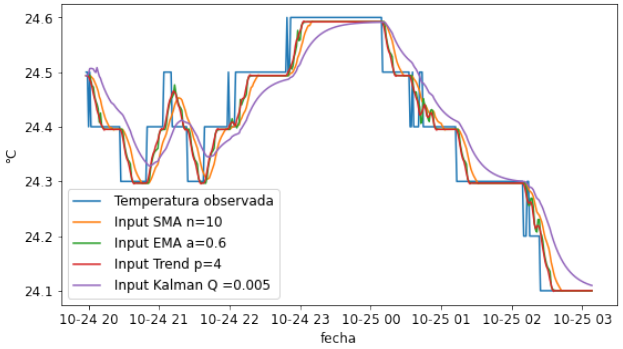
(a) Mejor configuración (Filtrado)



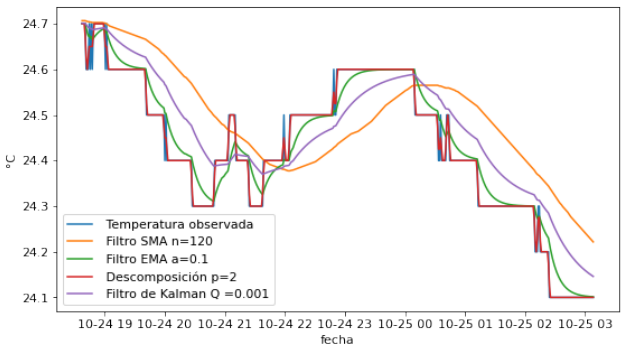
(b) Mejor configuración (Predicción)



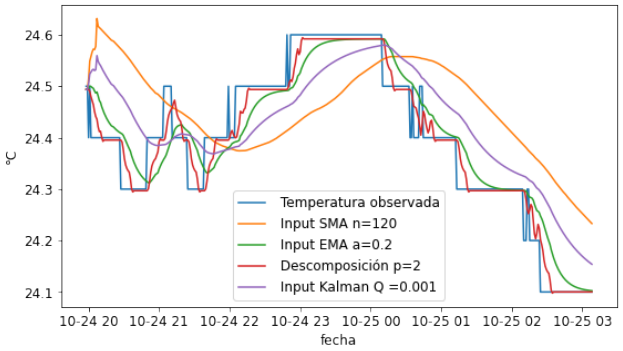
(c) Configuración Intermedia (Filtrado)



(d) Configuración Intermedia (Predicción)



(e) Peor configuración (Filtrado)



(f) Peor configuración (Predicción)

Figura 5.12: Filtrado y predicción con distintas configuraciones.

Las configuraciones que entregan mejores resultados (Figura 5.12a) son aquellas que logran preservar la forma de la señal, y al mismo tiempo suavizar y eliminar ruido e imperfecciones. Mientras que aquellas que dan peores resultados (Figura 5.12e) generan una señal sobre suavizada, alejándose de la señal original que tiene una forma escalonada.

5.5. Interfaz de Usuario

5.5.1. Plataforma web

Se crea una plataforma web utilizando el motor de plantilla Jinja [21], la biblioteca de css Bootstrap [22] y el lenguaje de programación Javascript. Con el propósito de permitir que el equipo de trabajo, luego de una autenticación, pueda registrar dispositivos que se hayan instalado en estaciones y también nuevos sensores adquiridos.

En las figuras 5.13 y 5.14 se observa un botón *drop down* con la opción de registrar nodos y sensores, y un *modal* con el formulario de inscripción de un nodo, respectivamente.

Figura 5.13: Landing page con drop down para registrar Dispositivo.

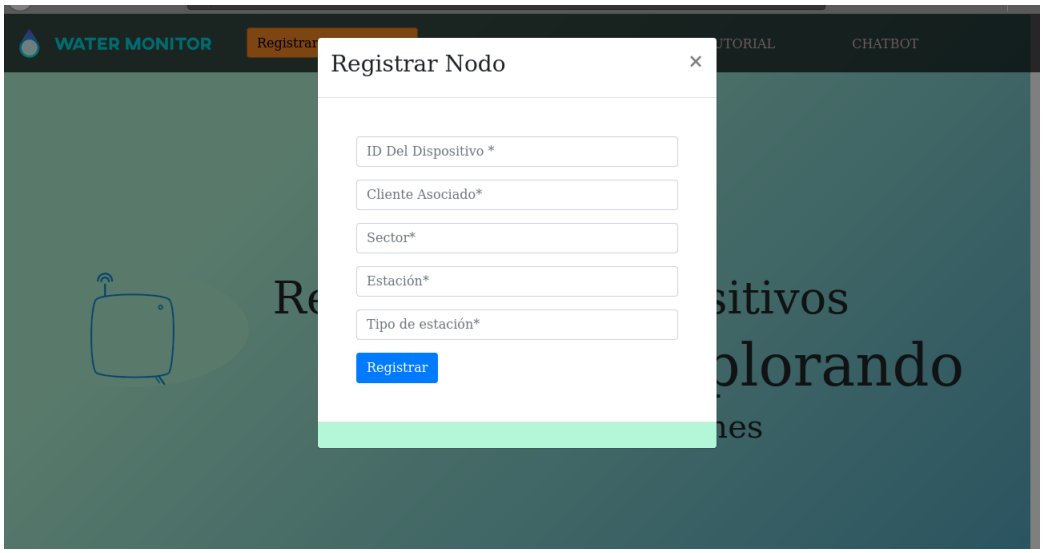


Figura 5.14: Modal para registrar nuevo nodo.

[21] https://jinja.palletsprojects.com/en/2.11.x/
[22] https://getbootstrap.com/

5.5.2. Tablero de control

La solución dispone de herramientas gráficas, de uso interno, apropiadas para la toma de decisiones sencillas. Es por esto que se escoge un visualizador de datos *open source*, que soporta series de tiempo, de uso amigable y que provee múltiples visualizaciones. Dos visualizadores populares son Kibana [23] y Grafana[24].

Kibana es una solución que corre sobre Elasticsearch cuyo principal uso es el monitoreo de mensajes de registro, pero también soporta funciones de reporte, como por ejemplo visualización de series de tiempo. Por otro lado, Grafana es una aplicación multiplataforma, que permite realizar numerosas visualizaciones (enfocada en series temporales), analítica y alarmas sobre los datos provistos. Se distingue de la anterior, ya que puede conectarse con múltiples almacenes de datos incluido InfluxDB. Por otro lado, Influx también posee su propio visualizador de datos, llamado Chronograph[25], el cual a pesar de ser una herramienta pensada para el uso exclusivo de aplicaciones de Influx, tiene ciertas ventajas como mejorar el desempeño de las consultas a la base de datos. Sin embargo, para aprovechar sus ventajas requiere la instalación de muchos *pluggins* y librerías.

Dado lo anterior, se decide utilizar Grafana ya que soporta datos InfluxDB, se enfoca en la analítica de series de tiempo y requiere menos dependencias que Chronograf, lo que facilita su uso. En la Figura 5.15 se observan algunas visualizaciones en un Tablero de Grafana.

Figura 5.15: Tablero de control con datos históricos de institución anonimizada.

[23] https://www.elastic.co/kibana
[24] https://grafana.com/
[25] https://www.influxdata.com/time-series-platform/chronograf/

5.5.3. Chatbot

Grafana permite conectar distintos canales a su aplicación de alerta, uno de éstos es Telegram. Este canal se conecta a Grafana a través de un Chat Bot, introduciendo el *API token* del *bot* y el ID del chat donde se incluye al asistente virtual. En la Figura 5.16 se observa la conexión activa entre el *bot* y las alarmas de Grafana.

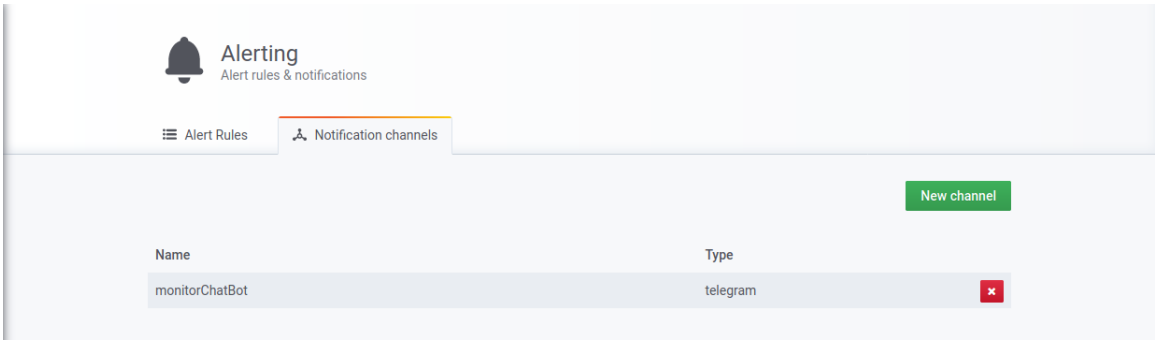


Figura 5.16: Conexión con canal de Telegram.

Cada vez que ocurre una alarma de Grafana, el *bot* alerta a todos los subscriptores del canal de Telegram. En la Figura 5.17 se observa una alerta de prueba, que es comunicada a través del canal MonitorAgua.

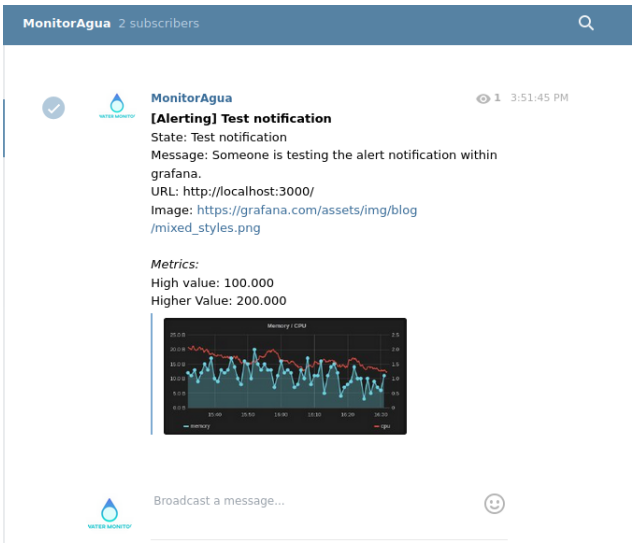


Figura 5.17: Mensaje alerta de prueba.

En las figuras 5.18 y 5.19 se observan *shortcuts* para acceder al tablero de control Grafana y al *Chatbot* de Telegram, respectivamente.

Figura 5.18: Integración a Grafana.

Figura 5.19: Integración con Chatbot de Telegram.

Capítulo 6

Validación

Para validar la solución implementada se prueban tres cosas: que la conexión entre la API y la aplicación de la red de sensores sea exitosa, que la solución escale a las necesidades del proyecto y por último que los algoritmos de filtrado propuestos logren suavizar el ruido de la señal.

6.1. Conexión

Para verificar que la conexión sea exitosa, se realiza el siguiente experimento:

- Se conecta un sensor de temperatura DS18B20 a un nodo de una red LoRaWAN. El sensor recolecta y envía muestras a la aplicación de la red con una frecuencia de 1 minuto.
- A través de Grafana se agrega como *data source* la tabla _internal generada automáticamente por Influx, como se observa en la Figura 6.1.
- Se verifica que no existan fallas en las escrituras, como se observa en la Figura 6.2a, mientras que se monitorea la recepción de datos (Figura 6.2b).

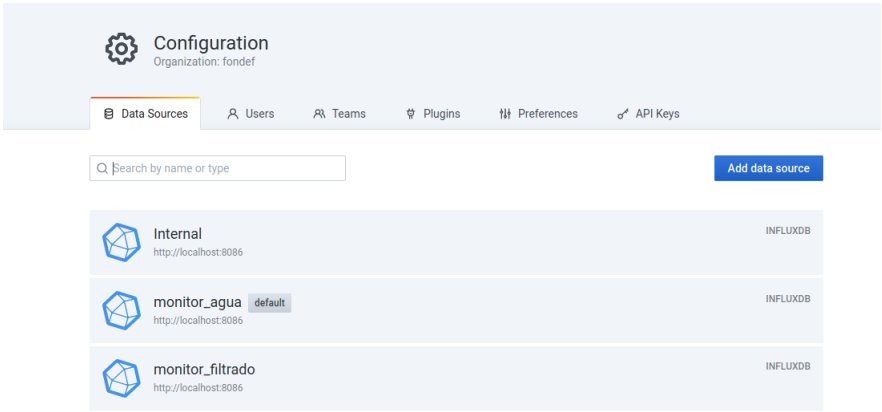
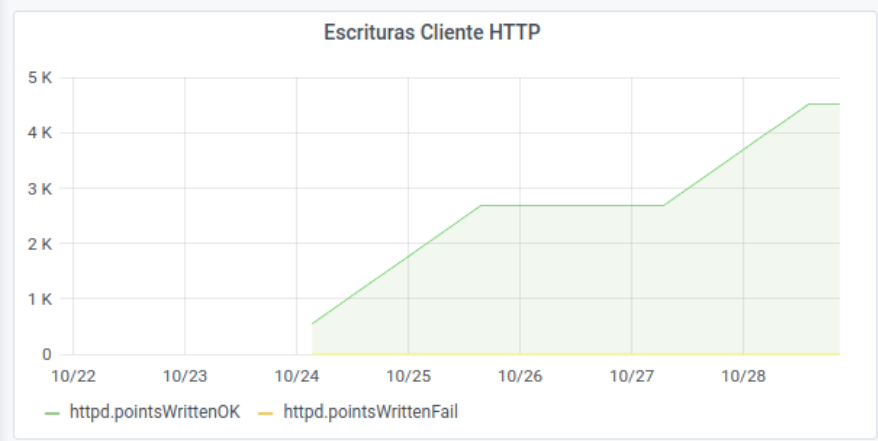
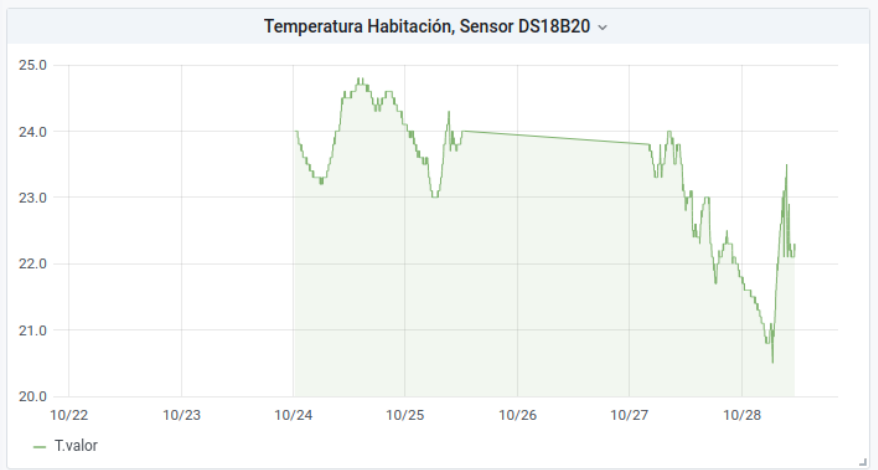


Figura 6.1: Bases de datos conectadas con Grafana.



(a) Escrituras exitosas y fallidas.



(b) Registros escritos a la base de datos.

Figura 6.2: Escrituras exitosas a la base de datos.

De la Figura 6.2 es posible observar que el sensor transmite datos durante 4 días, presentando una desconexión entre los días 25 y 27 de octubre. En la Figura 6.2a se observa que todas las *request* de escritura a Influx a través de httpd (Apache) fueren exitosas, por lo que los días que se dejó de recibir datos corresponden a desconexiones en una capa inferior, lo cual se verifica revisando la consola de TTN.

6.2. Escalabilidad

Para validar que la aplicación escala, se realizan pruebas de estrés aumentando el número de clientes que solicitan escribir a la base de datos, utilizando la conexión con Apache de por medio. Para esto se generan clientes *mock-up* con la aplicación Jmeter, donde cada cliente representa un **nodo**[1] de una estación.

6.2.1. Pruebas de estrés utilizando Jmeter

Apache Jmeter[2] es una aplicación escrita en Java, diseñada para testear aplicaciones web, permitiendo generar múltiples clientes con acciones automatizadas, que estresan la app que se desea poner a prueba.

Jmeter permite correr un Test Plan customizando:

- Número de threads: corresponde a la cantidad de clientes que acceden al sitio.
- Ramp up : tiempo que transcurre desde el inicio de la prueba hasta que se levantan todos los threads.
- Loop count: cantidad de veces que se repite la prueba.

Figura 6.3: Configuraciones de jmeter.

En la Figura 6.3 se observa las propiedades de los threads que se pueden customizar a través de la interfaz de Jmeter.

6.2.2. Configuraciones de Jmeter

En una situación real cada sensor tiene una frecuencia de muestreo que puede no ser idéntica, por lo tanto la probabilidad de que transmitan mensajes al mismo tiempo es baja. Se puede simular este desfase de transmisiones utilizando un *ramp-up* alto, por ejemplo 30 segundos, por lo que para simular esta situación se evalúa un número de clientes y *ramp-up* incremental.

[1]Un nodo es un dispositivo que recibe los datos de varios sensores en una estación, y los transmite a través de la aplicación de la red al *endpoint* del *backend*.

[2]https://jmeter.apache.org/

6.2.3. Configuraciones de Apache:

Para cada prueba se utilizan las siguientes configuraciones del servidor web Apache:

- MaxKeepAliveRequests: 100 [3]
- KeepAliveTimeout: 5 [4].
- MaxRequestWorkers/MaxClients: 150 [5]

6.2.4. Configuraciones de WSGI:

Además de especificar las configuraciones de Apache, se debe escoger el número de procesadores y threads para WSGI. Se realizan dos configuraciones diferentes:

- Experimento 1: 1 procesador y 5 threads.
- Experimento 2: 2 procesadores y 10 threads.

6.2.5. Experimentos:

En los siguientes experimentos se evalúa el porcentaje de error promedio en 10 *loops*, con 3 segundos de desfase entre el inicio de cada ronda.

6.2.6. Experimento 1

Porcentaje de errores				
	Ramp up (segundos)			
Número de clientes	30	10	1	0.1
50	0 %	0 %	0 %	0 %
100	0 %	0 %	0 %	0 %
150	0 %	0 %	4.93 %	8.13 %
200	3.75 %	8.77 %	25.85 %	20.05 %

6.2.7. Experimento 2

Porcentaje de errores				
	Ramp up (segundos)			
Número de clientes	30	10	1	0.1
50	0 %	0 %	0 %	0 %
100	0 %	0 %	0 %	0 %
150	0 %	0 %	0 %	5 %
200	3.75 %	14 %	24.05 %	28.04 %

La aplicación empieza a fallar con 150 clientes y un *ramp up* de 1 segundo. En particular se cae por errores de Apache, y no por problemas con la base de datos. Por lo que un

[3]Número máximo de requests permitidas durante una conexión persistente (*requests* simultáneas).
[4]Segundos que se espera la siguiente request del mismo cliente en la misma conexión.
[5]Número máximo de procesos de servidor permitidos a iniciar (clientes simultáneos).

número seguro para correr la aplicación es 100 clientes (nodos o dispositivos de la aplicación) simultáneos, bajo las configuraciones actuales.

Esta situación puede ocurrir cuando el número de nodos que transmiten datos de forma simultánea excede el número máximo de clientes concurrentes (MaxRequestWorkers) de Apache. También puede suceder cuando la API responde a una velocidad menor al *ramp up* de los cliente, provocando que se exceda el número máximo de conexiones activas.

Para conseguir un número mayor de clientes simultáneos se puede aumentar el parámetro MaxRequestWorkers de Apache, y la cantidad de threads y procesadores a WSGI, sin embargo, estos números están limitados por los recursos de la máquina. También, para evitar que la aplicación se caiga por un límite de conexiones activas se puede disminuir el KeepAliveTimeout, o simplemente quitar la opción de mantener conexiones persistentes ya que los nodos se deben conectar para transmitir datos y no para realizar múltiples acciones. De todas formas, 100 nodos simultáneos es más que suficiente para los objetivos de la primera etapa del proyecto FONDEF.

6.3. Filtrado:

En esta sección se presentan los resultados de los experimentos de la sección 5.4.2, utilizando dos métricas que comparan la señal real con aquella predicha por la red *Feed Forward*.

6.3.1. Medidas de performance

Error cuadrático medio normalizado:

El error cuadrático medio es una medida de distancia entre dos series de tiempo. Al normalizar utilizando $\bar{x}$ el promedio de las muestras, obtenemos que una medida 0 indica una predicción perfecta, mientras que un valor mayor a 1 indica que las predicciones no son mejores que utilizar el promedio de la serie de tiempo $\bar{x}$.

$$NRMSE = \sqrt{\frac{\sum_{\mathrm{i}=1}^{N}(x_{\mathrm{i}} - \hat{x}_{\mathrm{i}})^2}{\sum_{\mathrm{i}=1}^{N}(x_{\mathrm{i}} - \bar{x}_{\mathrm{i}})^2}} \tag{6.1}$$

Medida de error absoluto:

Corresponde al error promedio entre la serie y la predicción, indica que tanto se desvía la predicción de la serie real.

$$MAE = \frac{\sum_{\mathrm{i}=1}^{N} \|x_{\mathrm{i}} - \hat{x}_{\mathrm{i}}\|}{N} \tag{6.2}$$

6.3.2. Resultados

Feedforward Network

Configuración con mejores resultados

Algoritmo	NRMSE	MAE
Ninguno	0.250	0.022
SMA	0.285	0.027
EMA	0.253	0.023
desc	0.226	0.021
Kalman	0.265	0.024

Configuración con peores resultados

Algoritmo	NRMSE	MAE
Ninguno	0.250	0.022
SMA	0.959	0.128
EMA	0.37	0.041
desc	0.244	0.022
Kalman	0.599	0.077

A partir de la configuración que obtuvo mejores métricas se puede afirmar que la predicción mejora con una ventana más pequeña, un *alpha* más grande, un mayor número de periodos y una mayor covarianza, para SMA, EMA, descomposición aditiva y filtro de Kalman respectivamente.

El método indirecto de validación sugiere que el algoritmo que mejor filtra y preserva las características generales de la serie es la descomposición aditiva clásica (con las modificaciones realizadas), superando incluso a los datos de *testing*. Lo que indica que la red entrenada sobre datos con ruido, logra predecir mejor la serie original, si recibe como input un *dataset* filtrado con el algoritmo descompositivo.

En general, los algoritmos fuera de línea muestran un mejor performance que el filtro de Kalman. Sin embargo, Kalman sigue siendo bastante competitivo frente a los otros algoritmos, y si se considera que entrega datos en tiempo real, es una muy buena alternativa. Es necesario mencionar que este filtro requiere un *tunning* de las variables iniciales, para cada modelo, lo cual podría mejorar considerablemente el desempeño del algoritmo.

Caso práctico: ensuciar la señal

Se ensucia la señal original con ruido $r \in [-0{,}5, 0{,}5]$ °C como se observa en la Figura 6.4. Luego se prueba cada uno de los algoritmos, como se observa en la Figura 6.5, y se discuten los resultados.

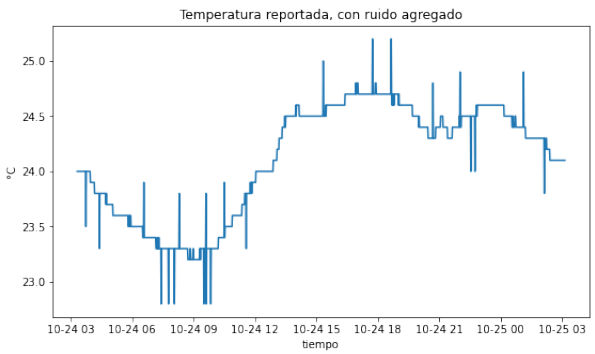
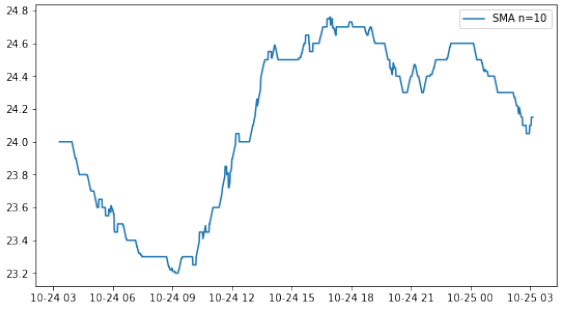
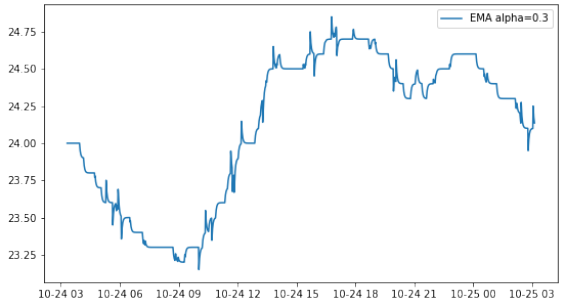


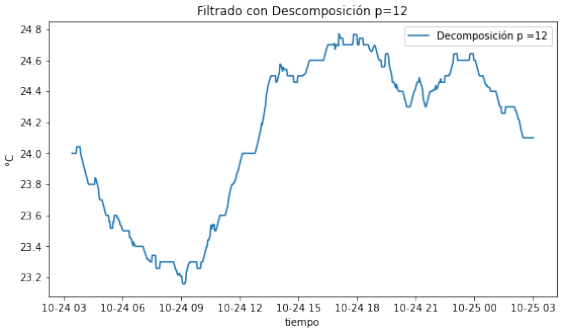
Figura 6.4: Señal original ensuciada con ruido.



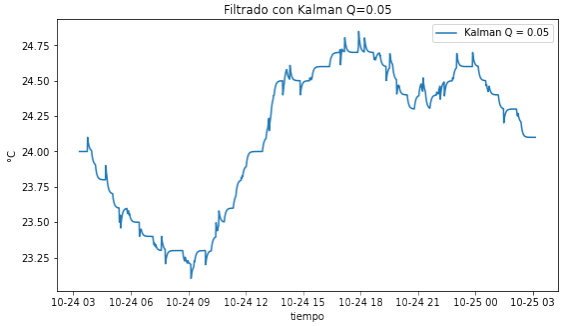
(a) SMA



(b) EMA



(c) Descomposición Aditiva



(d) Filtro de Kalman.

Figura 6.5: Filtrado de ruido a señal ensuciada.

Todos los algoritmos parecen actuar de forma bastante similar, reduciendo el ruido considerablemente pero sin eliminarlo completamente, además de suavizar la serie original. Se debe tomar en cuenta que la serie original ya contiene ruido, y se añade aún más ruido por motivos de experimentación.

Capítulo 7

Conclusión

Tras finalizar este trabajo de memoria, se logra integrar satisfactoriamente una red de sensores con un *backend* de adquisición, que incluye un servidor web Apache, conectado con una API en Flask, un módulo de filtrado y un tablero de control. Se logra validar la conexión con un número máximo de 100 clientes y se realiza un *benchmark* entre 4 algoritmos de filtrado, utilizando un método indirecto de validación que involucra el uso de una red neuronal. Cumpliendo de esta forma los objetivos planteados al inicio de este trabajo de memoria.

Existen múltiples factores que se toman en consideración al momento de realizar una aplicación funcional y escalable, por lo que gran parte de este trabajo ha contemplado una investigación. Pudiendo decir que una de las mayores dificultades de este trabajo ha sido el levantamiento de requisitos en un proyecto con equipos multidisciplinarios, y sobre un área poco explorada desde el punto de vista de la Ingeniería en Ciencias de la Computación. Luego de terminar este trabajo, se toma en cuenta lo complejas que son las redes de agua en Chile, y lo mucho que falta avanzar en la modernización de sistemas antiguos, así como también en la democratización de sistemas expertos con mejor cobertura y bajo presupuesto.

Muchos aspectos de este trabajo no han sido realizados como se planeó inicialmente, debido a la lamentable situación de pandemia que ha afectado a todos los países. En particular, no se ha podido instalar dispositivos en terreno, debido a las restricciones sanitarias, lo que ha dificultado contar con una variedad de datos, más fieles a la realidad.

7.0.1. Trabajo a futuro

Existen muchas formas de mejorar y extender este trabajo, pero algunos tareas que se pueden realizar en el futuro son:

- Implementar o encontrar una herramienta como Node-RED que pueda integrar una aplicación en TTN con un broker MQTT.
- Optimizar la velocidad de la API desarrollada, con un mayor paralelismo en la escritura de datos.

- Utilizar y evaluar otro tipo de algoritmos de filtrado que involucren el uso de varias señales, una vez que se cuente con suficientes datos de diferentes estaciones. Por ejemplo el algoritmo ICA, Independent Component Analysis, por sus siglas en inglés. ICA es una técnica de *machine learning* para separar componentes independientes, a partir de una señal compuesta, lo cual permite cancelar el ruido de ambiente.
- Crear un módulo de notificaciones, que envíe mensajes a los usuarios, cuando se detecten fallas en los sensores.

Glosario

API: *Application Programming Interface.*

EMA: *Exponential Moving Average.*

FONDEF: Fondo de Fomento al Desarrollo Científico y Tecnológico.

HTTP: *Hypertext Transfer Protocol.*

IaaS: *Infrastructure as a Service.*

ICA: *Independent Component Analysis.*

IoT: *Internet of Things.*

LPWAN: *Low Power Wide Area Network.*

MA: *Moving Average.*

MQTT: *Message Queuing Telemetry Transport.*

SMA: *Simple Moving Average .*

TSDB: *Time Series Data Base.*

WSGI: *Web Server Gateway Interface.*

TTN: *The Things Network.*

Bibliografía

[1] et al. B. Wukkadada. Comparison with http and mqtt in internet of things (iot). In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore*, pages 249–253, 2018.

[2] et al. D. Ding. A survey on model-based distributed control and filtering for industrial cyber-physical systems. In *in IEEE Transactions on Industrial Informatics, vol. 15*, pages 2483–2499, 2019.

[3] A. Dzikevičius and S. Šaranda. Ema versus sma usage to forecast stock markets: The case of sp 500 and omx baltic benchmark. *Business: Theory and Practice*, 11(3):248–255, 2010.

[4] et al. Fernández-Ahumada, L. Proposal for the design of monitoring and operating irrigation networks based on iot, cloud computing and free hardware technologies. *Sensors*, 19(10):2318, 2019.

[5] S. Geetha and S. Gouthami. Internet of things enabled real time water quality monitoring system. *Smart Waters*, 2(1):1, 2016.

[6] et al. Grassberger P. On noise reduction methods for chaotic data. *nary Journal of Nonlinear Science*, 3(2):127–141, 1993.

[7] Athanasopoulos G. Hyndman, R.J. *Forecasting: principles and practice*. OTexts, Melbourne, Australia, 2018.

[8] D. Karunasingha and S-Y. Liong. Enhancement of chaotic hydrological time series prediction with real-time noise reduction using extended kalman filter. volume 565, pages 737–746, 2018.

[9] et al. M. Fadhel. A comparison of time series databases for storing water quality data. *Auer M., Tsiatsos T. Mobile Technologies and Applications for the Internet of Things. IMCL 2018. Advances in Intelligent Systems and. Springer*, 909:418–429, 2019.

[10] Ministerio de Obras Públicas. Mesa nacional del agua – primer informe. Technical report, 2020.

[11] et al. Sendra, S. Lorawan network for fire monitoring in rural environments. *Electronics*, 9(3):531, 2020.

[12] T. Yokotani and Y. Sasaki. Comparison with http and mqtt on required network resources for iot. pages 1–6, 2016.