**ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN**

**UNIVERSIDAD DE CANTABRIA**

*Trabajo Fin de Grado*

# Desarrollo de una arquitectura de gestión SNMP para monitorización de sensores sobre plataforma Arduino y entorno de visualización Grafana
# (Development of an SNMP Management Architecture for Sensor Monitoring on Arduino Platform with Visualization in Grafana)

**Para acceder al Título de**

*Graduado en*
*Ingeniería de Tecnologías de Telecomunicación*

**Autor: Luis Carlos Deza Monge**
**Septiembre - 2025**

# GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

**Realizado por:** Luis Carlos Deza Monge
**Director del TFG:** Jose Ángel Irastorza Teja
**Título:** "Desarrollo de una arquitectura de gestión SNMP para monitorización de sensores sobre plataforma Arduino y entorno de visualización Grafana"
**Title:** "Development of an SNMP Management Architecture for Sensor Monitoring on Arduino Platform with Visualization in Grafana"

**Presentado a examen el día:** 18 de septiembre de 2025

para acceder al Título de

**GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN**

Composición del Tribunal:
Presidente (Apellidos, Nombre): Garcia Arranz, Marta
Secretario (Apellidos, Nombre): Conde Portilla, Olga Maria
Vocal (Apellidos, Nombre): Crespo Fidalgo, Jose Luis

Este Tribunal ha resuelto otorgar la calificación de: .....

**Fdo: El Presidente** **Fdo: El Secretario**

**Fdo: El Vocal** **Fdo: El Director del TFG**
(sólo si es distinto del Secretario)

Vº Bº del Subdirector Trabajo Fin de Grado Nº
(a asignar por Secretaría)

# *Agradecimientos*

En primer lugar, quiero agradecer a mi familia por enseñarme los valores que me han convertido en la persona que soy hoy, por acompañarme en este camino y por mostrarme siempre su apoyo incondicional, tanto en los buenos como en los malos momentos. A mis abuelos, que, por más lejos que estén, siempre me llamaban para animarme a seguir luchando por mis objetivos.

También quisiera tener en cuenta a aquellos compañeros de clase que hicieron que los años de la carrera fueran más amenos.

Finalmente, deseo expresar mi agradecimiento a mi director del TFG, Jose Angel Irastorza, por su amabilidad, sus consejos y el tiempo dedicado durante el desarrollo de este proyecto.

# *Resumen*

En una primera fase, se desarrolló un agente SNMP en una placa Arduino equipada con un módulo Ethernet Shield, utilizando la librería *Agentuino*. Dado que esta presentaba diversas limitaciones y errores de funcionamiento, fue necesario realizar correcciones y adaptaciones que permitieron dotar al agente de la estabilidad y capacidades necesarias para su uso en este proyecto.

Posteriormente, se validó su correcto funcionamiento mediante la comprobación de las operaciones básicas del protocolo (GET, GET-NEXT y SET) así como la generación y envio de traps. Para ello se utilizaron distintos gestores SNMP como Manage MIB Browser y Power SNMP, junto con el analizador de protocolos de redes WireShark.

Para la representación de los datos, se diseñó un dashboard en Grafana que muestra en tiempo real los valores de temperatura y humedad obtenidos por los sensores, e incorpora la capacidad de interacción remota mediante operaciones SNMP SET para el control del encendido y apagado de un actuador LED conectado a la placa.

Finalmente, se configuró el gestor PRTG Network Monitor con el fin de generar alertas automáticas por correo electrónico, desencadenadas a partir de la recepción de traps emitidas por el agente al superarse umbrales definidos para las variables monitorizadas.

Cabe destacar que todas las herramientas empleadas en el desarrollo de este proyecto son de uso libre, lo que refuerza su carácter accesible y replicable. Los resultados obtenidos demuestran la viabilidad de integrar dispositivos de bajo coste con protocolos de gestión de redes estándar, contribuyendo al desarrollo de soluciones ligeras, escalables y aplicables a escenarios de monitorización en entornos de Internet de las Cosas (IoT).

# *Abstract*

In an initial phase, an SNMP agent was developed on an Arduino board equipped with an Ethernet Shield module, using the Agentuino library. Since this library presented several limitations and malfunctions, it was necessary to apply corrections and adaptations that provided the agent with the stability and capabilities required for its use in this project.

Subsequently, its correct operation was validated by verifying the basic protocol operations (GET, GET-NEXT, and SET) as well as the generation and transmission of traps. For this purpose, different SNMP managers such as Manage MIB Browser and Power SNMP were used, together with the network protocol analyzer WireShark.

For data visualization, a Grafana dashboard was designed to display in real time the temperature and humidity values obtained by the sensors. It also incorporates remote interaction capabilities through SNMP SET operations to control the switching on and off of an LED actuator connected to the board.

Finally, the PRTG Network Monitor manager was configured to generate automatic email alerts, triggered by the reception of traps emitted by the agent when predefined thresholds for the monitored variables were exceeded.

It is worth noting that all tools used in the development of this project are open source or freely available, which reinforces its accessible and replicable nature. The results obtained demonstrate the feasibility of integrating low-cost devices with standard network management protocols, contributing to the development of lightweight, scalable solutions applicable to monitoring scenarios in Internet of Things (IoT) environments.

# Índice general

# Índice de Figuras

# Índice de Tablas

# Lista de códigos

# Índice de acrónimos

| | |
|---|---|
| API | Application Programming Interface. |
| ASN.1 | Abstract Syntax Notation One |
| HTML | HyperText Markup Language. |
| ICSP | In-Circuit Serial Programming. |
| IDE | Integrated Development Environment. |
| IP | Internet Protocol. |
| ISO | International Organization for Standardization. |
| LED | Light Emitting Diode. |
| MAC | Media Access Control. |
| MIB | Management Information Base. |
| OID | Object Identifier. |
| PC | Personal Computer. |
| TCP | Transmission Control Protocol. |
| UDP | User Datagram Protocol. |
| USB | Universal Serial Bus. |

# Capítulo 1. Introducción

## 1.1. Objetivos del proyecto

Como se ha mencionado antes, el principal objetivo de este Trabajo Fin de Grado es el desarrollo de una arquitectura de gestión SNMP para la monitorización de sensores sobre plataforma Arduino y entorno de visualización Grafana. El Arduino UNO con Ethernet Shield actúa como agente SNMP, integrando un sensor DHT11, encargado de medir temperatura y humedad, así como un LED cuyo encendido/apagado es controlado de forma remota.

Para alcanzar este objetivo, es trabajo se ha dividido en varias etapas. En una primera fase, se desarrolló el agente SNMP en el Arduino haciendo uso de la librería Agentuino, implementando la MIB que gestiona las variables de temperatura, humedad y estado del LED. Posteriormente, se realizaron pruebas de validación con distintos gestores SNMP: con ManageEngine MibBrowser se comprobó el correcto funcionamiento de las operaciones básicas de lectura y escritura; con PRTG Network Monitor se verificó la recepción de traps y la emisión de un correo electrónico notificando que se ha superado un umbral establecido. También, se utilizó Wireshark para analizar los paquetes SNMP intercambiados en la red.

En una segunda fase, el sistema se integró en un entorno de monitorización compuesto por Telegraf, InfluxDB y Grafana. Telegraf se configuró como colector SNMP para extraer periódicamente los valores de temperatura y humedad del Arduino, InfluxDB permitió almacenarlos en una base de datos de series temporales, y Grafana se empleó para diseñar paneles de visualización que muestran el histórico de los valores de temperatura y humedad.

Por último, se desarrolló una API en Python con Flask y la librería pysnmp, que actúa como puente entre Grafana y el agente SNMP, permitiendo el control del estado del LED desde el propio panel de visualización.

## 1.2. Motivación

Decidí escoger este tema para el Trabajo Fin de Grado porque, aunque ya había trabajado anteriormente con Arduino, me pareció interesante descubrir que podía configurarse como un Agente SNMP. Otro aspecto que me impulso a escoger este tema es que durante la asignatura de 2 curso “ Gestión y Operación de Redes” me gusto el tema del protocolo SNMP y me resulto atractivo poder aplicarlo en un entorno real, desarrollando un agente desde cero y validando su funcionamiento mediante distintos gestores.

También me motivo el hecho de que nunca había utilizado Grafana como herramienta de visualización. Diseñar paneles gráficos interactivos que no solo muestran la evolución de los datos, sino que además permitieran encender o apagar un dispositivo, representaba un reto practico y atractivo.

Por último, este proyecto me ofreció la oportunidad de desarrollar por primera vez una API en Python, implementada en Flask y conectada con el agente SNMP. Este trabajo me permitió adentrarme en el desarrollo de servicios web y comprender mejor cómo integrar distintas tecnologías en un sistema funcional.

## 1.3. Estructura del documento

Este documento se organiza en seis capítulos que abordan de forma progresiva los distintos aspectos del trabajo. A continuación, se presenta la estructura del documento:

En el Capítulo 1. Introducción, se presenta brevemente el proyecto, proporcionando información sobre la motivación del Trabajo Fin de Grado y los objetivos que guían su desarrollo.

En el Capítulo 2. Fundamentos Teóricos y Tecnológicos, se presentan los fundamentos teóricos y tecnológicos necesarios para comprender el desarrollo del sistema.

En el Capítulo 3. Descripción del Agente SNMP, se describe el proceso de implementación del agente SNMP en el entorno Arduino. Se especifican tanto los aspectos de hardware como las librerías utilizadas y la lógica de gestión de variables.

En el Capítulo 4. Gestor SNMP y Plataforma de Monitorización, se describen las herramientas utilizadas como gestores SNMP durante el desarrollo y validación del sistema, así como la integración de Telegraf, InfluxDB y Grafana en el sistema de monitorización.

El Capítulo 5. Integración y Validación del Sistema, se detalla el funcionamiento del sistema completo, la configuración de red, la validación de consultas SNMP, la integración con la plataforma de monitorización y el control remoto del LED mediante una API en Python.

Por último, el Capítulo 6. Conclusiones y desarrollos futuros, se recoge las conclusiones del proyecto y posibles mejoras.

# Capítulo 2: Fundamentos Teóricos y Tecnológicos

Este capítulo tiene como objetivo sentar las bases teóricas y tecnológicas necesarias para comprender el funcionamiento del sistema desarrollado. Se han descrito los principios del protocolo SNMP, las herramientas de gestión empleadas, así como los componentes hardware y software que integran la arquitectura propuesta.

## 2.1. Protocolo de Gestión SNMP

El protocolo SNMP (Simple Network Management Protocol) es un protocolo de capa de aplicación definido por la placa de arquitectura de Internet en RFC1157. SNMP se utiliza para intercambiar información de administración entre dispositivos de red. Es uno de los protocolos más comunes utilizados para la administración de red. SNMP forma parte del conjunto de protocolos de control de transmisión/protocolos de Internet (TCP/IP) según lo define la Comisión de Ingeniería de Internet. [1]

SNMP se basa en una arquitectura cliente-servidor, donde los elementos de red actúan como agentes SNMP, y una o varias estaciones centrales asumen el rol de gestores SNMP.

- El agente es el componente implementado en cada dispositivo, responsable de recopilar información sobre su estado y exponerla mediante una interfaz estándar.

- El gestor es el sistema que realiza consultas a los agentes, procesa los datos obtenidos y permite su visualización o análisis.

El gestor envía las peticiones transmitidas sobre UDP/TCP con puertos 161 para los comandos generales, y 162 para las traps. Los principales comandos del protocolo SNMP son: [1]

- GET: Solicita el valor actual de una variable concreta definida en el dispositivo.

- GETNEXT: Solicita la siguiente variable dentro de la jerarquía de datos.

- SET: Permite modificar el valor de una variable, si el agente lo permite.

- GETBULK: Introducido en SNMPv2, Se utiliza para recuperar datos masivos de una tabla MIB grande.

- TRAP: Permite a un agente enviar una notificación al gestor cuando ocurre un evento relevante, sin necesidad de una consulta previa.

Para organizar y estructurar toda la información accesible en un dispositivo, SNMP hace uso de una **MIB,** que es una base de datos jerárquica que define todos los objetos administrables mediante identificadores únicos (OIDs).

Existen tres versiones del protocolo SNMP:

- **SNMPv1:** Año 1988. RFC 1065, 1066, 1067. Define la arquitectura física (gestor-agente), el modelo de información de gestión (SMIv1) y las operaciones básicas del protocolo. Es muy inseguro. Se transmiten las contraseñas en texto claro. [2]

- **SNMPv2:** Introducido en 1993 a través de las especificaciones RFC 1441 a 1452, trajo mejoras significativas en prácticamente todos los aspectos del modelo de gestión SNMP. Amplía el modelo de información (SMIv2), añade algunas operaciones (Trap), reduce la carga de tráfico adicional para la monitorización (con los GetBulk e Informs) y soluciona los problemas de monitorización remota. Mejora la seguridad, pero resulta muy complicado y apenas se utiliza. Las versiones de SNMP son compatibles, en el sentido que SNMPv2 puede leer SNMPv1. [2]

- **SNMPv3**: Año 2002. Mejora los aspectos de seguridad. Incluye autenticación previa a efectuar operaciones de lectura o escritura, confidencialidad e integridad. Gracias a estas mejoras, las operaciones entre gestor y agente pueden realizarse de forma segura, garantizando que solo usuarios autorizados puedan consultar o modificar datos sensibles. En el agente SNMP se implementa un subsistema denominado Access Control Subsystem, encargado de verificar qué partes de la MIB pueden ser accedidas por un determinado gestor, lo que permite una gestión más precisa y segura de la red. Además, SNMPv3 mantiene la compatibilidad con versiones anteriores. [2]

## 2.2. Herramientas de Gestión SNMP

Durante la validación del sistema implementado en Arduino, se emplearon gestores SNMP (ManageEngine MIB Browser y SNMPSoft) para verificar la correcta comunicación del agente, comprobando que las operaciones de lectura (GET) y escritura (SET) sobre los sensores y el LED funcionaran adecuadamente.

### 2.2.1. ManageEngine MIB Browser

Es una herramienta gratuita que permite cargar MIBs, consultar agentes SNMP y enviar comandos como GET, GETNEXT o SET de forma manual. Soporta SNMPv1, v2c y v3, permite graficar OIDs en tiempo real, visualizar tablas SNMP, etc. [3]

### 2.2.2. PRTG Network Monitor

Es una herramienta de monitorización de redes desarrollada por Paessler AG que integra la gestión basada en SNMP. PRTG actúa como gestor SNMP, permitiendo la recepción de traps SNMP enviadas por los agentes. El sistema escucha dichas traps en el puerto UDP 162 y procesa las notificaciones generadas por switches, routers u otros equipos gestionados. Una de sus funcionalidades más destacadas es la posibilidad de asociar estas traps a mecanismos de notificación, como el envío de correos electrónicos al administrador. [4]

## 2.3. MIBs (Management Information Base)

La **Management Information Base** (MIB) es un componente fundamental del modelo de gestión SNMP. Se trata de una base de datos virtual que describe, organiza y categoriza los objetos que pueden ser gestionados mediante SNMP en un dispositivo o agente.
Cuando los datos se envían desde el dispositivo a un gestor de SNMP, el compilador del gestor utiliza la MIB para convertir los datos a un formato legible por el usuario.[5]

La MIB se organiza como un árbol jerárquico en el que cada nodo está identificado mediante un OID. Este árbol se inicia desde una raíz común definida por la ISO, y bajo ella se encuentran la rama **org(3).dod(6).internet(1)** qué es el punto de entrada de la mayoría de las MIBs empleadas en redes IP.

La Figura 2.1 muestra la estructura jerárquica de este árbol de OIDs, en el que se distinguen las principales ramas utilizadas en el ámbito de las telecomunicaciones y redes.

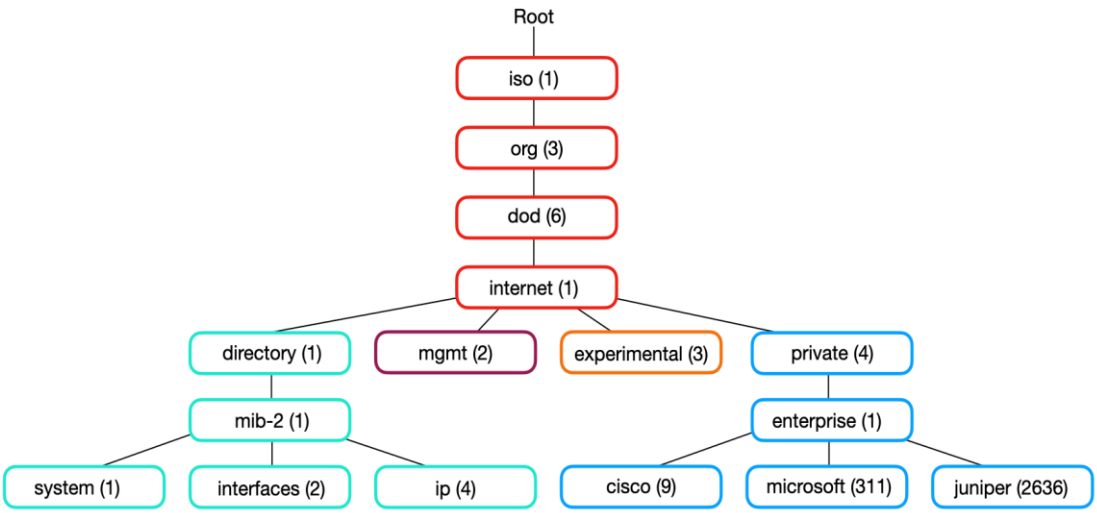


*Figura 2.1: Estructura jerárquica de los OIDs en SNMP. [6]*

A partir de este punto se ramifican diferentes tipos de MIBs, clasificadas en tres categorías principales: [7]

- **MIBs públicas**: Están definidas mediante estándares y proporcionan información general del sistema.

- **MIBs experimentales**: Se encuentran en desarrollo y están bajo revisión por parte de grupos de trabajo, usualmente con fines de investigación.

- **MIBs privadas:** Definidas por los distintos fabricantes o desarrolladores, y ofrecen información más detallada y concreta de sus dispositivos o sistemas. Todas las MIBs privadas se ubican dentro del nodo private(4).enterprises(1) del árbol jerárquico (OID: 1.3.6.1.4.1).

En este proyecto se hace uso de una **MIB privada personalizada** para representar los datos proporcionados por el sistema Arduino, incluyendo variables como temperatura, humedad y control del estado de un LED.
Las MIBs se definen utilizando el **SMI,** un estándar que especifica cómo describir los objetos que un agente SNMP puede gestionar.
Este lenguaje se basa en **ASN.1**, una notación utilizada para representar estructuras de datos de forma estandarizada e independiente del sistema o lenguaje de programación.[8]

El **SMI** define aspectos clave de cada objeto, como su tipo de dato, permisos de acceso, descripción y ubicación dentro del árbol de OIDs. Gracias a este formato, las herramientas SNMP (como los MIB Browsers) pueden interpretar correctamente los objetos definidos, mostrar sus nombres simbólicos y garantizar que las consultas respeten las reglas establecidas.

## 2.4. Dispositivo Sensor y LED

### 2.4.1. Sensor de temperatura y humedad DHT11 Módulo KY-015

El sensor DHT11 (Figura 2.2), es un sensor digital económico que permite medir tanto la temperatura como la humedad relativa del ambiente. Este dispositivo integra internamente un sensor de humedad de tipo capacitivo y un termistor NTC para la medición de temperatura, ofreciendo una solución sencilla y compacta para aplicaciones básicas de monitoreo ambiental. [9]

El sensor entrega los datos en forma digital a través de un solo pin, sin necesidad de convertir señales analógicas, lo que lo hace especialmente útil en sistemas embebidos como Arduino. Sin embargo, es importante tener en cuenta que solo proporciona nuevas lecturas cada dos segundos, por lo que no es adecuado para aplicaciones que requieren actualizaciones de alta frecuencia. [9]

Para medir la humedad, el DHT11 emplea un sensor capacitivo que detecta la presencia de vapor de agua midiendo la variación de resistencia eléctrica entre dos electrodos. A medida que la humedad relativa aumenta, el sustrato del sensor absorbe más vapor, liberando iones que aumentan la conductividad eléctrica. De este modo, una mayor humedad implica una resistencia más baja y viceversa [9].

Por su parte, la medición de temperatura se realiza mediante un termistor NTC (Coeficiente de Temperatura Negativo), que reduce su resistencia a medida que la temperatura aumenta, permitiendo así una lectura precisa del entorno térmico [9].

Gracias a su bajo coste y facilidad de uso, el DHT11 es una opción popular en proyectos educativos y de prototipado rápido que no requieren gran precisión ni velocidad de lectura.

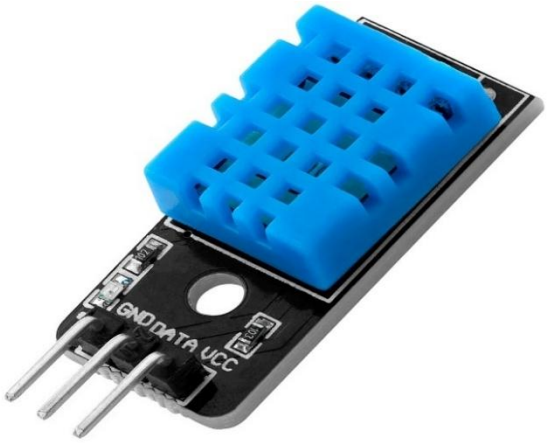


*Figura 2.2: Sensor Temperatura y Humedad DHT11. [9]*

En el Anexo I se presenta la tabla A.1 que presenta las características técnicas del sensor DHT11.

### 2.4.2. Diodo LED (Light Emitting Diode)

El LED (Light Emitting Diode) es un componente electrónico que emite luz al ser polarizado en sentido directo. En este proyecto se utilizó un LED verde de 5 mm como actuador visual, con el objetivo de indicar, de forma local, el estado de una variable SNMP gestionada remotamente.

El uso del LED (Figura 2.3) en este contexto permite demostrar que el sistema no sólo realiza funciones de monitorización, sino que también es capaz de actuar sobre el entorno físico desde un gestor SNMP externo, cumpliendo el modelo de gestión remota definido por SNMP.

En el Anexo I en la tabla A.2 se observa las características técnicas del Diodo LED empleado en el sistema.

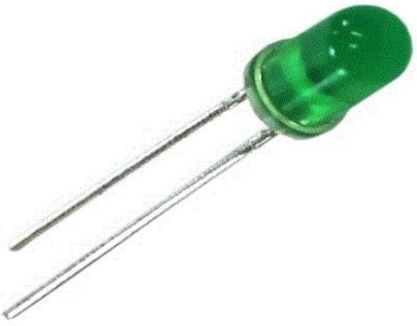

*Figura 2.3: Diodo LED. [10]*

### 2.4.3. Resistencia limitadora de 220 ohmios

Para asegurar el funcionamiento seguro del LED, se añadió en serie una resistencia de 220 Ω, cuyo propósito es limitar la corriente que fluye a través del componente.

Dado que los LEDs no tienen una forma interna de limitar la corriente, el uso de una resistencia es imprescindible para evitar el daño por sobrecorriente, tanto en el LED como en el pin del microcontrolador que lo controla. El valor de 220 ohmios fue seleccionado para mantener la corriente por debajo de los 20 mA, considerando una tensión de alimentación de 5 V y una caída de tensión de aproximadamente 2 V en el LED.

*Figura 2.4: : Resistencia 220 ohmios. [11]*

En el Anexo I se muestra una tabla A.3 con las características que presenta esta resistencia de 220 ohmios.

## 2.5. Plataforma Arduino

En este proyecto se ha utilizado la plataforma Arduino como base para la implementación del Agente SNMO. Esta compuestas por dos elementos principales:

- **Arduino UNO R3**, es una placa de desarrollo basada en el microcontrolador Atmega328P. Cuenta con 14 pines de entrada/salida digital, 6 entradas analógicas, una conexión USB, un conector de alimentación, un conector ICSP y un botón de reinicio [13]. Es una de las placas más utilizadas en proyectos de prototipado y automatización debido a su simplicidad, versatilidad y compatibilidad con múltiples módulos.

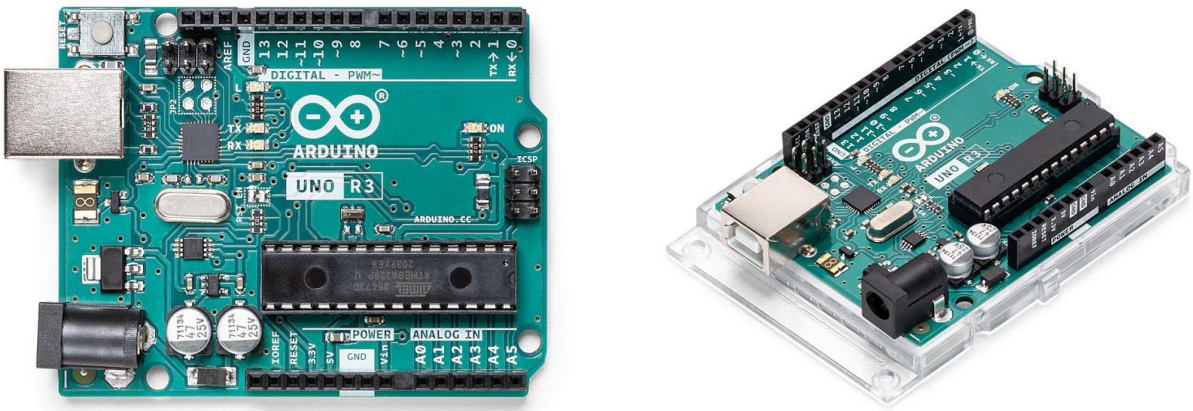


*Figura 2.5: Arduino UNO R3. [13]*

- **Ethernet Shield 2** (Figura 2.6), se trata de un shield compatible con Arduino UNO. Está basado en el chip Wiznet W5500 [14], soporta protocolos TCP/IP y permite que el Arduino se integre en una red local o en Internet mediante un cable Ethernet, habilitando la implementación de servicios como SNMP. Este shield es compatible directamente con Arduino UNO mediante la librería Ethernet oficial de Arduino.

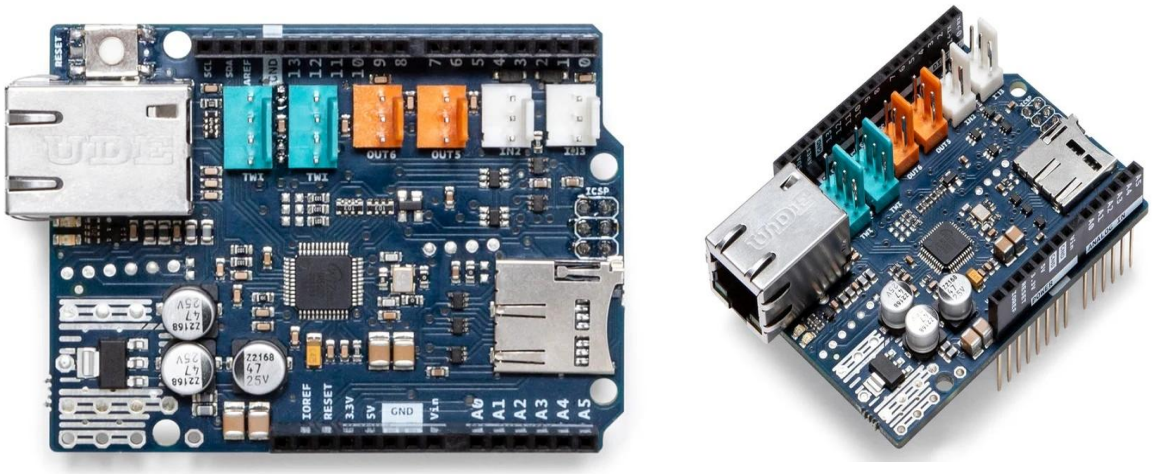

*Figura 2.6: Ethernet Shield 2. [14]*

En el Anexo I se muestra una tabla A.4 con la comparativa entre el Arduino UNO R3 y el Ethernet Shield 2 [13] [14], destacando la complementariedad entre ambos dispositivos: mientras que el primero aporta la capacidad de cómputo y control, el segundo proporciona la interfaz de red Ethernet.

## 2.6. Herramientas y Dispositivos Auxiliares.

Además de la plataforma Arduino, el desarrollo del sistema requirió la incorporación de diversas herramientas software y hardware que desempeñan un papel fundamental en la recolección, almacenamiento, visualización de los datos obtenidos. A continuación, se describen los principales componentes auxiliares empleados.

**InfluxDB**

InfluxDB es una base de datos especializada en el almacenamiento y gestión de series temporales. Estas series consisten en secuencias de valores asociados a marcas de tiempo, siendo ideal para aplicaciones como la monitorización de sensores, métricas de sistemas o análisis en tiempo real. Su estructura se organiza en cuatro componentes: measurements (conjunto de datos equivalente a una tabla), fields (valores numéricos o de texto a almacenar), tags (pares clave-valor usados para etiquetar y filtrar series) y timestamps (registran el momento exacto en que se capturo el dato). En este proyecto, InfluxDB almacena los valores obtenidos por Telegraf, permitiendo una consulta eficiente y organizada por tiempo, tipo de medición u otros objetos definidos. [15]

### Telegraf

Telegraf es un agente de recopilación de métricas desarrollado por InfluxData, diseñado para recolectar, procesar y enviar datos hacia una base de datos de series temporales como InfluxDB. Presenta una arquitectura basada en plugins, lo que permite integrar múltiples fuentes y destinos de datos. Para este proyecto, se utiliza el plugin de entrada (input) SNMP. Este plugin de entrada permite especificar direcciones IP de dispositivos SNMP, comunidades y OIDs. [16]

Telegraf actúa como intermediario entre el agente SNMP implementado en Arduino y la base de datos InfluxDB, encargándose de recolectar y transferir las métricas de temperatura y humedad.

### Grafana

Grafana es una plataforma de código abierto para la visualización de datos en tiempo real. Se caracteriza por su flexibilidad, facilidad de uso y amplio conjunto de funcionalidades. Permite crear paneles interactivos (dashboards) mediante gráficos, indicadores, tablas y otros elementos visuales, adaptados a las necesidades del usuario. [17]

Una de las características principales que presenta Grafana es que se puede conectar con todas las fuentes de datos posibles, conocidas también como base de datos como Prometheus, MySQL, PostgreSQL, InfluxDB, y otros. [17]

En este proyecto, Grafana se utiliza para representar los datos obtenidos de los sensores de mi Agente SNMP Arduino de forma gráfica, permitiendo tanto la supervisión en tiempo real como el análisis histórico.

### Switch D-Link DGS-1100-08PV2

El switch que se emplea en este proyecto es el D-Link DGS-1100-08PV2, un modelo Smart Managed (semi-gestionable) que combina la simplicidad de los switches no gestionables con funciones de administración básicas, adecuadas para entornos de pequeña y mediana escala.

Este equipo cumple un doble rol en el proyecto: por un lado, proporciona la conectividad Ethernet entre el agente SNMP (Arduino) y el gestor SNMP (PC); y por otro, puede ser monitorizado mediante SNMP, lo que permite ampliar el alcance de la arquitectura propuesta incluyendo métricas de red (uso de puertos, estado de enlace, etc.)

En el Anexo 1 se presenta la Tabla 2.5 que resume las principales características del D-Link DGS-1100-08PV2.

### Wireshark

Wireshark es una herramienta multiplataforma de software de código abierto utilizada para el análisis de tráfico de red. [18]

Permite capturar y examinar los paquetes de datos que se envían y reciben a través de una red, además de opciones de filtrado para visualizar solo los paquetes de interés.

Proporciona una interfaz gráfica que muestra una representación detallada de cada paquete capturado, incluyendo la información sobre la fuente y el destino, protocolos involucrados, campos de cabecera y datos transportados. [18]

En el proyecto se usa para capturar y validar los mensajes SNMP GET, SET, GET-NEXT y TRAP)

## 2.7. Visión General de la Arquitectura Propuesta

La arquitectura propuesta para este trabajo se basa en la interacción entre un Agente SNMP, un Gestor SNMP y una plataforma de monitorización, todos ellos interconectados a través de la red.

A continuación, en la Figura 2.7, se muestra de forma esquemática el sistema desarrollado. El Arduino UNO con su Ethernet Shield actúa como agente SNMP integrando la sensórica (Sensor DHT11) junto con el LED con su respectiva resistencia. Este dispositivo implementa la MIB y es capaz de enviar traps hacia el gestor. La red de comunicación, basada en el protocolo SNMP, asegura la transmisión de operaciones (GET, GET-NEXT, SET) y traps utilizando los puertos estándar UDP/161 y UDP/162. Finalmente, en el equipo gestor (PC), se concentra las herramientas de exploración de la MIB y el entorno de monitorización basado en Telegraf, Influx DB y Grafana, la cual permite almacenar y visualizar las métricas obtenidas desde el agente.

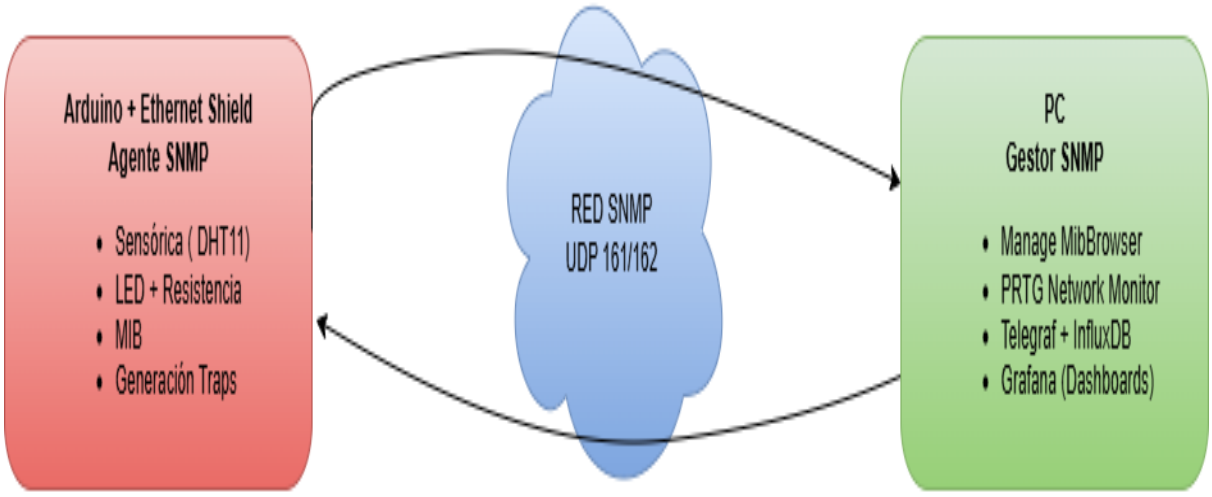


*Figura 2.7: Diagrama esquemático de la arquitectura propuesta.*

# Capítulo 3. Descripción del Agente SNMP

En este capítulo se describe el proceso de diseño e implementación del Agente SNMP en Arduino, tanto en lo relativo al hardware como el software empleado. El agente se encarga de exponer, mediante el protocolo SNMP, los datos obtenidos por el sensor de temperatura DHT11, así como permitir el control del LED. Para ello, se utilizan librerías específicas compatibles con Arduino y se implementa una lógica que gestiona las solicitudes del tipo GET y SET provenientes de gestores SNMP.

## 3.1. Hardware Agente SNMP

El agente SNMP se implementó sobre una placa Arduino UNO R3 con un Ethernet Shield W5100, que permite al Arduino UNO actuar como nodo SNMP dentro de la red TCP/IP a través del puerto estándar UDP/161.

En cuanto a los componentes conectados son los siguientes:

- Se emplea un **sensor DHT11**, que permite medir tanto la temperatura como la humedad. La elección de este sensor se debe a que responde con la simplicidad del montaje y a su compatibilidad con las librerías de Arduino.

- Como actuador se integró un **LED de color verde**, que puede encenderse o apagarse mediante operaciones SET desde el gestor SNMP. Este componente no solo permite comprobar que el sistema no solo monitoriza variables, sino que también admite control remoto. Para limitar la corriente de funcionamiento y proteger el LED se añadió **resistencia de 220 Ω**.

La Figura 3.1 representa el esquema de conexiones eléctricas para el montaje del Agente SNMP Arduino. El sensor DHT11 se conecta al pin digital D2 del Arduino para la transmisión de datos, mientras que el LED se conecta al pin digital D7 con la resistencia de 220 Ω.

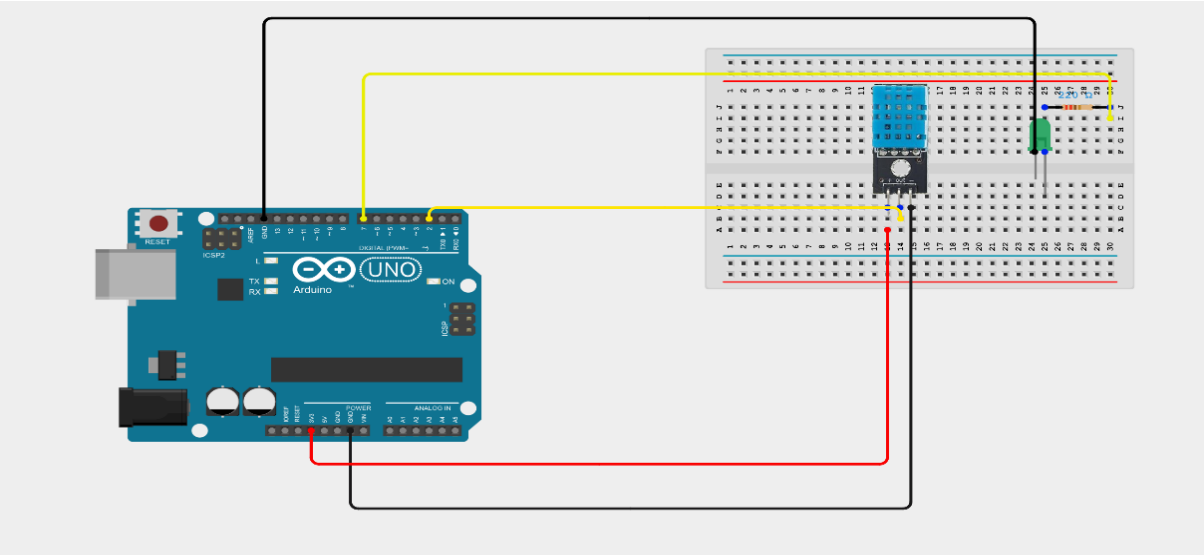


*Figura 3.1: Esquema de conexión del sensor DHT11 y LED con Arduino UNO.*

## 3.2. Librerías utilizadas en el desarrollo del agente SNMP

Para implementar el protocolo SNMP al Arduino para que actúe como Agente SNMP, se ha hecho uso de librerías, tanto estándar como personalizadas. Cada una de ellas cumple una función específica en la gestión de la red, el protocolo SNMP y la lectura de sensores.

*Tabla 3.1: Librerías utilizadas en el desarrollo del agente SNMP en Arduino.*

| Librería | Función Principal |
|---|---|
| Ethernet.h | Configuración de red y gestión de paquetes UDP. |
| SPI.h | Comunicación SPI con el Ethernet Shield. Necesaria para la transmisión de datos entre Arduino y el módulo Ethernet. |
| DHT.h | Controla el sensor de temperatura y humedad DHT11 |
| Agentuino.h | Implementa las funciones básicas de un agente SNMP (SNMPv1). |
| Variables.h | Declaración de variables globales. |
| MIB.h | Define y registra los OIDs utilizados en la MIB personalizada. |

## 3.3. Estructura del Agente SNMP

El agente SNMP Arduino se basa en la inicialización del entorno de red, registro de los OIDs definidos en la MIB y un bucle principal de atención a solicitudes.

1. **Inicialización de Red y agente:** Al inicio del programa, se configura la interfaz de red Ethernet del Arduino utilizando la librería Ethernet.h. Se asigna una dirección IP que va a hacer la de mi Arduino, junto con la dirección MAC del Arduino, para poder así permitir su identificación en la red local. También se inicializa el agente SNMP mediante la función Agentuino.begin(), que lo prepara para procesar y enviar paquetes UDP en el puerto 161.

2. **Manejo de solicitudes SNMP:** Se utiliza la función pduReceived() para poder gestionar todos los OIDs. Se encarga de que cada vez que llegue una solicitud SNMP (por ejemplo, para leer la temperatura o encender un LED), el agente no busca en una lista larga de objetos registrados, sino que envía esa petición directamente a la función pduReceived(), se revisa que dato se está pidiendo (según el OID) y se responde con el valor pedido, o bien se cambiar un valor si es una orden SET (por ejemplo, encender el LED).

3. **Bucle escucha y respuesta:** En la función loop(), el agente SNMP entra en un ciclo de escucha de peticiones. La función Agentuino.listen() se encarga de detectar paquetes SNMP entrantes y en caso de existir, ejecuta la función pduReceived() para procesarlos.

En el Anexo III, se presenta el Código que se implementa para que mi Arduino actúe como un Agente SNMP.

## 3.4. Gestión de Variables

Para que el agente SNMP pueda responder correctamente a las peticiones del cliente (como mostrar la temperatura o encender un LED), necesita acceder a datos internos del sistema. Estos datos se guardan en variables globales que están organizadas en dos archivos:

- Variable.h: Se declaran las variables con extern, ya que se van a utilizar en otros archivos. En el Codigo 3.1, se muestra un fragmento del archivo Variable.h

```
#ifndef _VARIABLE_H_
#define _VARIABLE_H_

#include <Arduino.h>

// GRUPO DE INFORMACION


extern uint32_t locUpTime;
extern char locContact[20];
extern char locName[20];
extern char locLocation[20];
extern char email[30];
extern char ip[15];
extern byte my_IP_address[4];// arduino IP address


// GRUPO SENSORES LEIDOS

extern int32_t index1;
extern int32_t index2;
extern int32_t index3;


extern char descr1[25];
extern char descr2[25];
extern char descr3[25];

extern int32_t sensortemp;
extern int32_t sensorhum;
extern int32_t valorled;

extern int32_t tiempoMuestreo1;
extern int32_t tiempoMuestreo2;
extern int32_t tiempoMuestreo3;
```

*Código 3.1: Ejemplo de gestión de variables en Variable.h.*

- Variable.cpp: Se definen y se inicializan las variables creadas en el archivo Variable.h. En el Codigo 3.2, se muestra un fragmento del archivo Variable.cpp

```
#include "Variable.h"

uint32_t locUpTime = 0;
char locContact[20] = "Luis Carlos Deza";
char locName[20] = "TestTFG";
char locLocation[20] = "Habitacion";
char email[30] = "luiscardm8@gmail.com";
char ip[15] = "10.90.90.20"; //IP ARDUINO
byte my_IP_address[4] = {10, 90, 90, 20};


int32_t index1 = 1;
int32_t index2 = 2;
int32_t index3 = 2;

char descr1[25] = "Sensor de Temperatura";
char descr2[25] = "Sensor de Humedad";
char descr3[25] = "Valor Led";

int32_t sensortemp = 7;
int32_t sensorhum  = 0;
int32_t valorled   = 0;

int32_t tiempoMuestreo1 = 10;
int32_t tiempoMuestreo2 = 10;
int32_t tiempoMuestreo3 = 10;


uint32_t instanteLectura1 = 0;
uint32_t instanteLectura2 = 0;
uint32_t instanteLectura3 = 0;
```

*Código 3.2: Ejemplo de gestión de variables en Variable.cpp.*

Este mecanismo garantiza que los datos ofrecidos al gestor SNMP estén siempre actualizados y accesibles desde cualquier parte del programa.

## 3.5. Creación MIB personalizada

Con el fin de que los gestores puedan interpretar las variables con nombres simbólicos, se diseñó el archivo MIBSENSORES.mib en formato ASN.1.

La MIB personalizada se estructura bajo el nodo enterprises 1.3.6.1.4.1.36582 y se divide en tres bloques principales:

1. **informacionDeGestion (1):** Datos del dispositivo. Entre ellos destacan:

    - Identificador: Número de identificación del dispositivo.
    - localización: Ubicación del dispositivo.
    - direccionIP: IP asignada al Arduino.
    - encargadoDeGestion: Persona responsable de la gestión.
    - correoContacto: Correo electrónico del responsable.

**2. valoresLeidosSensores (2):** Contiene la table arduinoTable, que organiza los valores obtenidos de los sensores. Cada fila de esta tabla (arduinoEntry) incluye:

- indice: Indice del sensor.
- descrepSensor: Nombre del sensor.
- valorLeido: Lectura actual del sensor.
- tiempoMuestreo: Intervalo de muestreo en segundos.
- instanteLectura: Marca de tiempo SNMP (TimeTicks) que corresponde al último valor leido.

Esta estructura en forma de tabla permite recorrer secuencialmente todos los sensores mediante operaciones GET-NEXT.

**3. eventosTraps (3):** Define la tabla eventosTable, orientada a la gestión de eventos y envió de traps. Cada fila de esta tabla (eventosEntry) describe un evento.

- indiceEv: Identificador de evento.
- ifIndexSensor: Referencia al sensor sobre el que aplica el evento.
- descrEvento: Descripción del evento.
- umbralSup y umbralInf: Umbrales superior e inferior configurables.
- ipReceptorTrap: Dirección IP de destino para los traps generados.

En la Figura 3.2 se observa el árbol jerárquico del archivo MIBSENSORES.mib. En el que se aprecian las tres ramas principales: informacionDeGestion, valoresLeidosSensores y eventosTraps.

Cabe señalar que el nodo *enterprises* está situado en el OID 1.3.6.1.4.1 del árbol de registro.

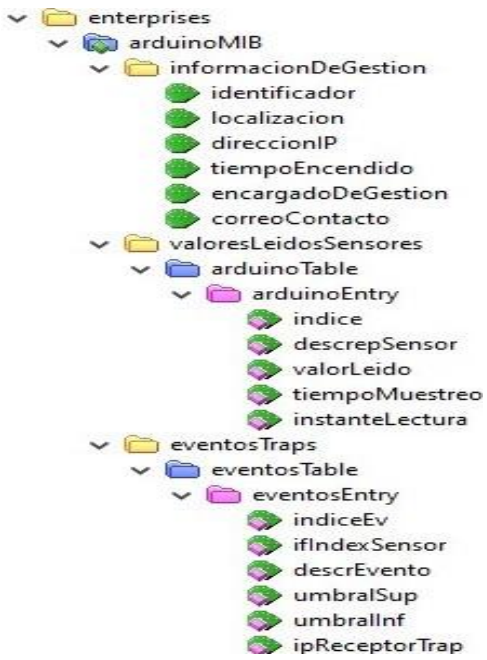


*Figura 3.2: Árbol Jerárquico "MIBSENSORES.mib".*

En este apartado se ha descrito de manera resumida la función de cada objeto de la MIB. La definición completa en formato ASN.1 puede consultarse en el Anexo II.

## 3.6. Implementación de la MIB en Arduino (MIB.h)

En el archivo MIB.h se implementa la lógica de los OIDs definidos en la MIB, registrando su asociación con las variables globales. Esto permite que cuando un gestor realice una operación GET o SET, el Arduino acceda directamente a la variable correspondiente.

```
#ifndef _MIB_H_
#define _MIB_H_

const char sysName[] PROGMEM = "1.3.6.1.4.1.36582.1.1.0";

const char sysLocation[] PROGMEM = "1.3.6.1.4.1.36582.1.2.0";

const char ipAddress[] PROGMEM = "1.3.6.1.4.1.36582.1.3.0";

const char sysUpTime[] PROGMEM = "1.3.6.1.4.1.36582.1.4.0";

const char sysContact[] PROGMEM = "1.3.6.1.4.1.36582.1.5.0";

const char mail[] PROGMEM = "1.3.6.1.4.1.36582.1.6.0";

// VALORES LEIDOS
const char ifIndex1[] PROGMEM = "1.3.6.1.4.1.36582.2.1.1.1.1";
const char ifIndex2[] PROGMEM = "1.3.6.1.4.1.36582.2.1.1.1.2";
const char ifIndex3[] PROGMEM = "1.3.6.1.4.1.36582.2.1.1.1.3";

const char des1[] PROGMEM = "1.3.6.1.4.1.36582.2.1.1.2.1"; //
const char des2[] PROGMEM = "1.3.6.1.4.1.36582.2.1.1.2.2"; //
const char des3[] PROGMEM = "1.3.6.1.4.1.36582.2.1.1.2.3"; //


const char sensorTemp[] PROGMEM = "1.3.6.1.4.1.36582.2.1.1.3.1
const char sensorHum[] PROGMEM = "1.3.6.1.4.1.36582.2.1.1.3.2"
const char valorLed1[] PROGMEM = "1.3.6.1.4.1.36582.2.1.1.3.3"

const char tMuestreo1[] PROGMEM = "1.3.6.1.4.1.36582.2.1.1.4.1
const char tMuestreo2[] PROGMEM = "1.3.6.1.4.1.36582.2.1.1.4.2
const char tMuestreo3[] PROGMEM = "1.3.6.1.4.1.36582.2.1.1.4.3
```

*Código 3.3: Fragmento del archivo MIB.h con definición de OIDs.*

## 3.7. Funcionamiento de la librería Agentuino

La librería Agentuino implementa el protocolo SNMP en Arduino, permitiendo que actúe como un agente capaz de recibir y responder a solicitudes SNMP.

Está compuesta por dos archivos:

- **Agentuino.h:** Define estructuras y funciones como encode() y decode() para gestionar distintos tipos de datos. [21]

- **Agentuino.cpp:** Implementa la lógica del protocolo, incluyendo la recepción de paquetes, la comparación de OIDs y la generación de respuestas.

La relación entre los OIDs definidos en MIB.h y las variables internas del sistema se realizan en el Agentuino.cpp, utilizando bloques else if que comparan el OID recibido con los definidos en la MIB. [21]

Cuando llega una solicitud SNMP (GET, SET, GET-NEXT):

1. Se compara el OID recibido con los OIDs registrados.

2. Si coincide, se ejecuta la operación:

    - **SET**, se invoca la función decode () para actualizar la variable con el valor recibido.
    - **GET**, se utiliza encode () para codificar y devolver el valor de la variable al gestor.

```
else if (strcmp_P(oid, des1) == 0) {

 if (pdu.type == SNMP_PDU_SET) {
    // response packet from set-request - object is read/write
    status = pdu.VALUE.decode(descr1, strlen(descr1));
    pdu.type = SNMP_PDU_RESPONSE;
    pdu.error = status;
 } else {

    status = pdu.VALUE.encode(SNMP_SYNTAX_OCTETS, descr1);
    pdu.type = SNMP_PDU_RESPONSE;
    pdu.error = status;
 }
```

*Código 3.4: Gestión de OID tipo DisplayString en pduReceived() (archivo Agentuino.cpp).[21]*

Las funciones decode() [21] solo se aplican a variables de escritura (ej. el LED). En cambio, para sensores de solo lectura (read-only) como temperatura y humedad, cualquier intento de SET es rechazado con el error SNMP_ERR_READ_ONLY, tal y como se ve en el Código 3.5.

```
else if (strcmp_P(oid,sensorTemp ) == 0) {
  // handle sysServices (set/get) requests
  if (pdu.type == SNMP_PDU_SET) {
      // response packet from set-request - object is read-only
      pdu.type = SNMP_PDU_RESPONSE;
      pdu.error = SNMP_ERR_READ_ONLY;
  } else {
      status = pdu.VALUE.encode(SNMP_SYNTAX_INT, sensortemp);
      pdu.type = SNMP_PDU_RESPONSE;
      pdu.error = status;
  }
  //
UG


else if (strcmp_P(oid,sensorHum ) == 0) {
  // handle sysServices (set/get) requests
  if (pdu.type == SNMP_PDU_SET) {
      // response packet from set-request - object is read-only
      pdu.type = SNMP_PDU_RESPONSE;
      pdu.error = SNMP_ERR_READ_ONLY;
  } else {
      status = pdu.VALUE.encode(SNMP_SYNTAX_INT, sensorhum);
      pdu.type = SNMP_PDU_RESPONSE;
      pdu.error = status;
  }
```

*Código 3.5: Gestión de OIDs de temperatura y humedad en pduReceived() (archivo Agentuino.cpp) [21].*

## 3.8.Implementación del programa en Arduino

En este apartado se describe el código desarrollado en Arduino, cuyo objetivo es que la placa Arduino actúe como un agente SNMP. El código se estructura en distintas funciones que cubren inicialización, atención de peticiones y generación de traps.

### 3.8.1 Función setup()

La función setup() prepara el sistema para actuar como agente SNMP, responsable de la inicialización del sistema.

Su funcionamiento se divide en las siguientes etapas:

- **Inicialización de comunicación:** se activa la comunicación serie, se inicia el sensor DHT11 y se configura el LED, garantizando que arranca en un estado seguro (apagado).

- Estructura de sensores y variables: Cada objeto gestionado se almacena en una estructura de punteros (sensores []), lo que facilita un acceso uniforme desde otras rutinas del programa.
- **Configuración de red:** Se inicia el servidor local ( server.begin() ) y se configura la interfaz Ethernet con dirección MAC, IP y mascara de red ( Ethernet.begin() ); se obtiene y se muestra la dirección IP asignada al Arduino confirmando su correcta conexión a la red.

- **Inicialización de contadores de muestreo.**

- **Arranque del agente SNMP:** Se invoca Agentuino.begin(). En caso de inicialización correcta, se registra la función pduReceived() como manejador de PDUs entrantes (GET/SET/GET-NEXT).

```
void setup() {
    Serial.begin(9600);
    HT.begin();
    pinMode(ledPin, OUTPUT);
    digitalWrite(ledPin, LOW); // LED comienza apagado

    Serial.println("Arduino Iniciado, logs preparados para ser servidos :)");


    addString("Trap generado por: Sensor de Temperatura");
    addString("Trap generado por: Sensor de Humedad");
    addString("Trap generado por: Led");


    sensores[0] = &sensortemp;
    sensores[1] = &sensorhum;
    sensores[2] = &valorled;

    server.begin();

    Ethernet.begin(mac,ipArduino,gateway,subnet);

    IPAddress address = Ethernet.localIP();// PARA OBTENER MI DIRECCION IP
    for (uint8_t i=0;i<=4;i++) {
        my_IP_address[i] = address[i];
        if(i!=4){
        Serial.print(my_IP_address[i]); //ME PINTA MI DIRECCION IP
        }
        Serial.print(".");
    }
    Serial.println("");


    cont1=1000; //SON 1000 MS QUE SERAN 10 SEG
    delay(100);
```

*Código 3.6: Fragmento de la función setup().*

Tras esta función setup(), el Arduino queda conectado a la red, con el sensor y el led inicializados, umbrales preparados para la generación de traps y el agente SNMP listo para interactuar con los gestores SNMP.

### 3.8.2. Función loop()

La función loop() constituye el bucle principal de ejecución del Arduino. Se repite de forma indefinida y permite mantener al Agente SNMP en funcionamiento continuo, asegurando tanto la atención de peticiones externas como la actualización de variables y la generación de traps.
Sus etapas principales son:

- **Atención de solicitudes SNMP:** La instrucción Agentuino.listen() mantiene al agente a la escucha en el puerto UDP/161, de modo que cualquier gestor puede realizar operación GET,SET o GET-NEXT.

- **Actualización de variables:** Las funciones actualiza1(), actualiza2() y actualiza3() refrescan periódicamente las lecturas del sensor DHT11 y el estado del LED.

- **Comprobación de eventos y traps:** Las funciones evento1(), evento2() y evento3() comparan los valores actuales con los umbrales definidos. Si se detecta que un valor supera dichos límites, se construye y envía una trap SNMP al gestor configurado.

- **Gestión del tiempo de actividad:** La variable locUptime se incrementa cada segundo mediante la comparación con millis(). De este modo, el sistema mantiene un contador de tiempo de encendido conforme al estándar SNMP (sysUptime).

```
void loop() {
    // LECTURA DE PARÁMETROS
    Agentuino.listen();

   // FUNCIONES DE MUESTREO/LECTURA DE VARIABLES

    actualiza1();
    actualiza2();
    actualiza3();

    //COMPROBAMOS LOS POSIBLES TRAPS
    evento1();
    evento2();

 // VARIABLE SYSUPTIME, LA ACTUALIZAMOS!
    if (millis() - prevMillis > 1000) {

        prevMillis += 1000;

        locUpTime += 100 ;

       // CONTADOR DE LOS SENSORES!
        cont1=cont1-100;
        cont2=cont2-100;
        cont3=cont3-100;
    }
}
```

*Código 3.7: Bucle principal de ejecución del agente SNMP función loop().*

- En paralelo, **se decrementan los contadores** (cont1, cont2, cont3), que regulan los periodos de muestreo del sensor y del LED, permitiendo las lecturas de forma controlada.

En resumen, la función loop() garantiza que el Arduino actúe de manera continua como agente SNMP, ofreciendo datos actualizados y enviando notificaciones cuando se superan los umbrales definidos.

### 3.8.3. Funciones de actualización de variables

El sistema implementa tres funciones principales de actualización, que se invocan desde la función loop(). El objetivo de estas funciones es leer periódicamente los valores de los sensores y actualizar las variables asociadas en la MIB, de forma que los gestores SNMP puedan obtener siempre datos actualizados.

- **actualiza1():** Esta función es para la actualización de la temperatura. Comprueba si ha cambiado el tiempo de muestreo configurado para el sensor (tiempoMuestreo1). Cuando el contador (cont1) llega a cero, se realiza la lectura mediante (HT.readTemperature()), se muestra el valor por consola para ver si es correcto y se actualiza la variable sensortemp. Además, se registra el instante de la lectura (instanteLectura1) tomando como referencia el contador de tiempo de actividad (locUpTime).

```
void actualiza1(void)
{
    if(tiempoMuestreo1!=muestreo1Ant){
      muestreo1Ant=tiempoMuestreo1;
      cont1 = tiempoMuestreo1*100;
    }
    if(cont1==0){
      //SE LEEE EL VALOR Y SE ACTUALIZA EL TIEMPO
      sensortemp = HT.readTemperature();
      Serial.print("Valor Sensor Temperatura: ");
      Serial.println(sensortemp);
      if (isnan(sensortemp)) {
        Serial.println("Error al leer el sensor DHT11");
        //return;
      }
      instanteLectura1 = locUpTime;
      cont1=tiempoMuestreo1*100;

    }
}
```

*Código 3.8: Función actualiza1() - actualización de temperatura.*

- **actualiza2():** Esta función es para la actualización de la humedad. Es similar a la función actualiza1() pero accediendo al valor de la humedad (HT.readHumidity()).

Cada lectura se registra en la variable global sensorhum y se marca el instante de lectura (instanteLectura2).

```
void actualiza2(void)
{
    if(tiempoMuestreo2!=muestreo2Ant){
      muestreo2Ant=tiempoMuestreo2;
      cont2 = tiempoMuestreo2*100;
    }
    if(cont2==0){
      sensorhum = HT.readHumidity();
      Serial.print("Valor Sensor Humedad: ");
      Serial.println(sensorhum);
      instanteLectura2 = locUpTime;
      cont2=tiempoMuestreo2*100;
    }
}
```

*Código 3.9: Función actualiza2() - actualización de humedad.*

- **actualiza3():** Esta función es para la actualización del estado del LED. A diferencia del sensor DHT11, el LED no se mide físicamente, si no que se controla mediante operaciones SNMP SET. Su valor gestionado puede ser 0 (apagado) o 1 (encendido). Esta función imprime periódicamente el estado de la variable valorled y actualiza la marca temporal (instanteLectura3).

Todas utilizan contadores (cont1, cont2, cont3) para regular el intervalo de muestreo.

```
void actualiza3(void)
{
    if(tiempoMuestreo3!=muestreo3Ant){
      muestreo3Ant=tiempoMuestreo3;
      cont3 = tiempoMuestreo3*100;
    }
    if (cont3 == 0) {

      Serial.print("Estado actual del LED: ");
      Serial.println(valorled);
      instanteLectura3 = locUpTime;
      cont3 = tiempoMuestreo3 * 100;
    }
}
```

*Código 3.10: Función actualiza3() - actualización de estado del LED.*

### 3.8.4. Funciones de eventos y generación de traps

El sistema incorpora funciones específicas para detectar condiciones de alarma sobre las variables gestionadas y generar notificación SNMP (traps) hacia el gestor configurado. A continuación, se describen las funciones evento1() y evento2()

**Función evento1() – alarma por umbral superior**

La función evento1() evalúa la condición de que un sensor supere un umbral máximo configurado. Esta función, se utiliza principalmente con el sensor de temperatura. Cuando se supera el umbral, el Arduino envía una notificación SNMP (trap) al gestor configurado.

Su lógica se resume en los siguientes pasos:

1. Selección del objeto.

   Se valida que Sind1 ≠ 0. Si vale cero, significa que no hay un evento configurado en la tabla de la MIB para esta variable. En caso contrario, Sind1 indica el índice del objeto gestionado: 1 = temperatura, 2 = humedad, 3 = LED.

2. Comparación con umbral.

   Se obtiene el valor actual mediante *sensores[Sind1-1] y se compara con el umbral superior (uS1). Si el valor lo supera, la condición de alarma se considera activa.

3. Antirebote.

La variable booleana activada evita el envío de traps repetidas mientras se mantenga la condición de alarma. La primera vez que se dispara el evento, se envía la trap y activada pasa a 1. Cuando el valor vuelve a la normalidad, el sistema se rearma (activada = 0), lo que permite futuras notificaciones si el umbral vuelve a superarse.

4. Construcción y envío de la trap.

Para el envío de la trap se invoca Agentuino.Trap() a la que se le pasa como parámetros: mensaje (strcad), la dirección IP del gestor (ip1), el tiempo de actividad (locUpTime), el enterprise OID de tu organización (1.3.6.1.4.1.36582...), y el OID especifico de evento. Además, se imprime un registro en la consola con el instante de activación, calculado en horas, minutos y segundos a partir de locUpTime.

```
void evento1(void)
{
    if(Sind1!=0){

      if(Sind1==1 || Sind1==2 || Sind1==3){
      //Serial.print(*sensores[Sind1-1]);
      if((*sensores[Sind1-1]>uS1)){//*sensores[Sind1-1] es una lista dinamica que me va racorriendo los indices
        if(activada==0){

          char str1[10]= "";
          char strcad[25]= "";
          sprintf(str1, "%d", *sensores[Sind1-1]);

          strcat(strcad, d1); // Concatenar cadena2 a cadena1
          strcat(strcad, " Valor: "); // Concatenar cadena2 a cadena1
          strcat(strcad, str1); // Concatenar cadena2 a cadena1

          Agentuino.Trap(strcad , ip1, locUpTime, "1.3.6.1.4.1.36582.1.1.1.4.1","1.3.6.1.4.1.36582.1.1.1.4.1");
          activada=1;
```

*Código 3.11: Fragmento de la función evento1() - generación de trap por umbral superior (temperatura).*

Relación con la MIB: Sind1 corresponde al índice de sensor seleccionado (equivalente a ifIndexSensor en la tabla de eventos), uS1 es el umbral superior (umbralSup), d1 la descripción del evento (descrEvento) y ip1 se corresponde con el campo ipReceptorTrap.

**Función evento2() – Alarma por banda de umbrales**

La función evento2() comprueba una banda de valores umbrales aceptables para la humedad, delimitada por un umbral superior (uS2) y un umbral inferior (uI2).

En este caso, la alarma se activa cuando el valor de la humedad (sensorhum) queda fuera del rango definido por los umbrales superior (uS2) e inferior (uI2).

El resto de la lógica es equivalente a la función evento1():

- La variable Sind2 selecciona el sensor a vigilar (en este caso humedad).
- La variable de control activada2 evita el envío repetido de traps.
- La notificación se construye concatenando la descripción del evento (d2) y el valor leído, y se envía al gestor mediante Agentuino.Trap(), incluyendo como parámetros el mensaje, la dirección IP configurada (ip2), el tiempo de actividad (locUpTime) y los OIDs correspondientes.

```
void evento2(void)
{
  if(Sind2!=0){

   if(Sind2==1 || Sind2==2){

   if((*sensores[Sind2-1]>uS2)||(*sensores[Sind2-1]<uI2)){ //MAYOR QUE EL UMBRAL SUPERIOR O MENOR QUE EL UM
    if(activada2==0){

     char str2[10]= "";
     char strcad2[25]= "";
     sprintf(str2, "%d", *sensores[Sind2-1]);

     strcat(strcad2, d2); // Concatenar cadena2 a cadena1
     strcat(strcad2, " Valor: "); // Concatenar cadena2 a cadena1
     strcat(strcad2, str2); // Concatenar cadena2 a cadena1

     Agentuino.Trap(strcad2, ip2, locUpTime, "1.3.6.1.4.1.36582.1.1.1.4.1","1.3.6.1.4.1.36582.1.1.1.4.1");
     activada2=1;
```

*Código 3.12: Fragmento de la función evento2() - generación de trap por banda de umbrales (humedad).*

Relación con la MIB: Sind2 corresponde al índice de sensor,uS2 es el umbral superior (umbralSup), uI2 es el umbral inferior (umbralInf), d2 la descripción del evento (descrEvento) y ip2 que es la dirección IP receptora (ipReceptorTrap).

# Capítulo 4. Gestor SNMP y Plataforma de Monitorización

## 4.1. Introducción al Gestor SNMP

Mientras que el Arduino implementa el rol de Agente, exponiendo variables y generando traps, el gestor es el encargado de realizar consultas (GET/ GET-NEXT), modificar parámetros mediante ordenes (SET) y recibir notificaciones (TRAPs) ante eventos relevantes.

En este proyecto se utilizaron diferentes gestores con el fin de validar la correcta implementación del agente en Arduino y su integración en una plataforma de monitorización.

## 4.2. Herramientas de gestión utilizadas en el proyecto

Con el objetivo de cubrir tanto la exploración de la MIB como la validación de traps se utilizaron los siguientes gestores SNMP: ManageEngine MIB Browser y PRTG Network Monitor.

### 4.2.1. Pruebas con ManageEngine MIB Browser

ManageEngine MIB Browser es una herramienta orientada a la verificación de la MIB personalizada [3.x], exploración de OIDs y variables SNMP de forma gráfica.

En el marco de este proyecto, se utilizó para comprobar el correcto funcionamiento del agente SNMP implementado en el Arduino. Concretamente, permitió verificar la respuesta del dispositivo ante operaciones GET sobre las variables de temperatura, humedad y estado del LED; confirmar la actualización adecuada de los valores definidos en la tabla SNMP; y validar que los objetos de la MIB personalizada [3.x] se interpretaban de manera correcta a través de sus nombres simbólicos.

### 4.2.2. Recepción de traps con PRTG Network Monitor

Para validar la capacidad del agente de generar notificaciones SNMP, se empleó PRTG Network Monitor, un gestor especializado en la recepción de traps. En el sistema desarrollado, cuando se supera un umbral de temperatura o humedad, el Arduino que actúa como Agente SNMP envía una trap, que luego PRTG Network Monitor es el encargado de recibir esta trap. Cuando se recibe esta trap, se configuró el PRTG para emitir un correo electrónico al administrador como alerta que sea superado un umbral.

## 4.3. Integración con Telegraf e InfluxDB

Una vez validadas las operaciones básicas, el sistema se integró en un entorno de monitorización más avanzado utilizando Telegraf e InfluxDB.

- **Telegraf:** Se configuro como colector SNMP mediante su plugin nativo. Telegraf interroga periódicamente al Arduino, recopilando datos de temperatura y humedad.

- **InfluxDB:** Almacena los datos recopilados de Telegraf en formato de series temporales, lo que permite consultas históricas y un análisis detallado de la evolución de las variables.

## 4.4. Visualización con Grafana

Finalmente, los datos almacenados en InfluxDB se visualizaron mediante Grafana, que permitió la creación de paneles personalizados (Dashboards) para representar en tiempo real las variables gestionadas. En este entorno se implementaron paneles gráficos que facilitan el seguimiento continuo de la temperatura y la humedad, proporcionando al usuario una interpretación clara de la información registrada.

Además, se desarrolló la integración con una API externa en Python (Flask y pysnmp), lo que hizo posible incorporar en los paneles de Grafana botones interactivos para el control del estado del LED (Encendido / Apagado).

## 4.5. Esquema Global del sistema

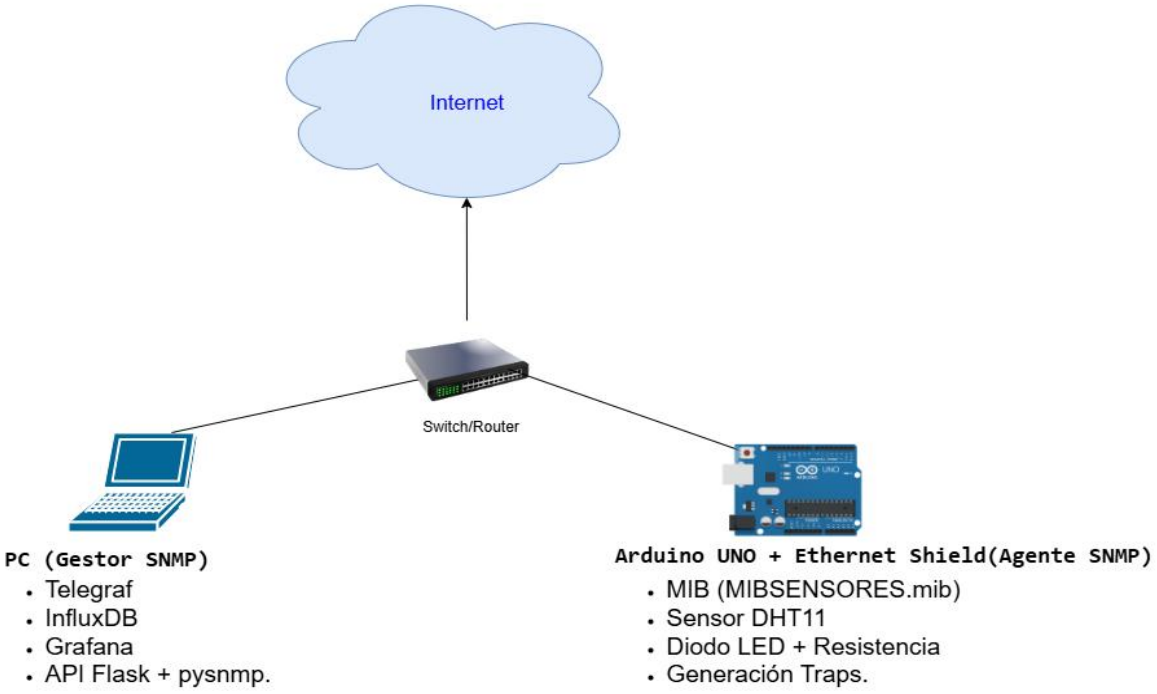


*Figura 4.1: Esquema global del sistema de gestión y monitorización SNMP.*

Una vez descritos los distintos componentes software y hardware que intervienen en la gestión SNMP. La Figura 4.1 representa el esquema global del sistema de gestión y monitorización SNMP desarrollado y las funciones específicas que cada uno desempeña dentro del sistema.

- **Equipo Gestor (PC):** Integra las funcionalidades de los gestores SNMP(ManageEngine MibBrowser, PorwerSNMP), y la plataforma de monitorización (Telegraf+InfluxDB y Grafana). Adicionalmente, aloja la API en Python (Flask + pysnmp), que actúa como puente entre Grafana y el Agente SNMP para el control del estado del LED.

- **Switch/Router:** Proporciona la conectividad entre gestor y agente, además de facilitar el acceso hacia Internet. Este elemento asegura la correcta transmisión de peticiones SNMP y Traps dentro de la red.

- **Agente SNMP (Arduino UNO + Ethernet Shield):** Interconecta el sensor DHT11 + LED, implementa la MIB y expone las variables definidas en la MIB. Además, genera las Traps cuando se superan ciertos umbrales.

# Capítulo 5. Integración y validación del sistema

## 5.1. Configuración de red y puertos SNMP

La primera etapa de validación del sistema consistió en definir la topología de red de pruebas y comprobar la conectividad entre los distintos elementos que lo componen. La red se implementó bajo la subred 10.90.90.0/24, con la siguiente asignación de direcciones IP: el equipo gestor SNMP (PC) con dirección IP 10.90.90.10, el Arduino UNO con Ethernet Shield actuando como agente SNMP configurado con la dirección IP 10.90.90.20 y el Switch con dirección IP 10.90.90.90.

En el gestor SNMP (PC) se habilitaron los puertos estándar asociados al protocolo SNMP: UDP/161, destinado a las operaciones GET, GET-NEXT y SET, y UDP/162 reservado para la recepción de TRAPs. Para garantizar la comunicación, fue necesario configurar reglas específicas en el cortafuegos del equipo gestor (PC). En particular, se habilitaron reglas de entrada y salida para los puertos UDP/161 y UDP/162, lo que permitió que las peticiones SNMP circularan sin restricciones y que las TRAPs fueran recibidas correctamente.

La Figura 5.1 muestra la topología de red utilizada en las pruebas, donde se observa la interconexión de los tres dispositivos principales bajo la misma subred.

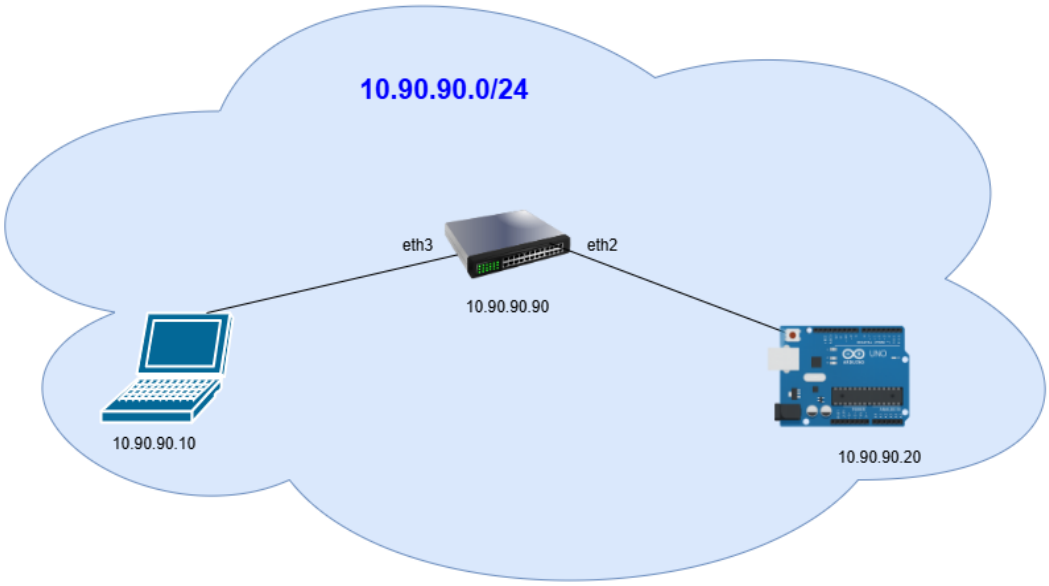


*Figura 5.1: Topología de red con asignación de direcciones IP y puertos SNMP.*

## 5.2. Validación de consultas SNMP

La primera fase de validación se centró en comprobar que el agente SNMP implementado en el Arduino respondía correctamente a operaciones básicas de gestión.

Para ello, se empleó el gestor SNMP ManageEngine MIB Browser, que permite cargar la MIB (MIBSENSORES.mib) y consultar de forma gráfica las variables definidas, así como la herramienta Wireshark, utilizada para verificar a nivel de red la correcta emisión y recepción de los mensajes SNMP.

En la Figura 5.2 se muestra la carga de la MIB personalizada en el gestor ManageEngine MibBrowser. Dentro del árbol de objetos se observa la tabla valoresLeidosSensores, que incluye las variables correspondientes a la temperatura, humedad y el estado del LED. Los valores recuperados coinciden con los reportados por el Arduino a través del Serial Monitor, lo que permitió confirmar la coherencia de la información gestionada por el agente.

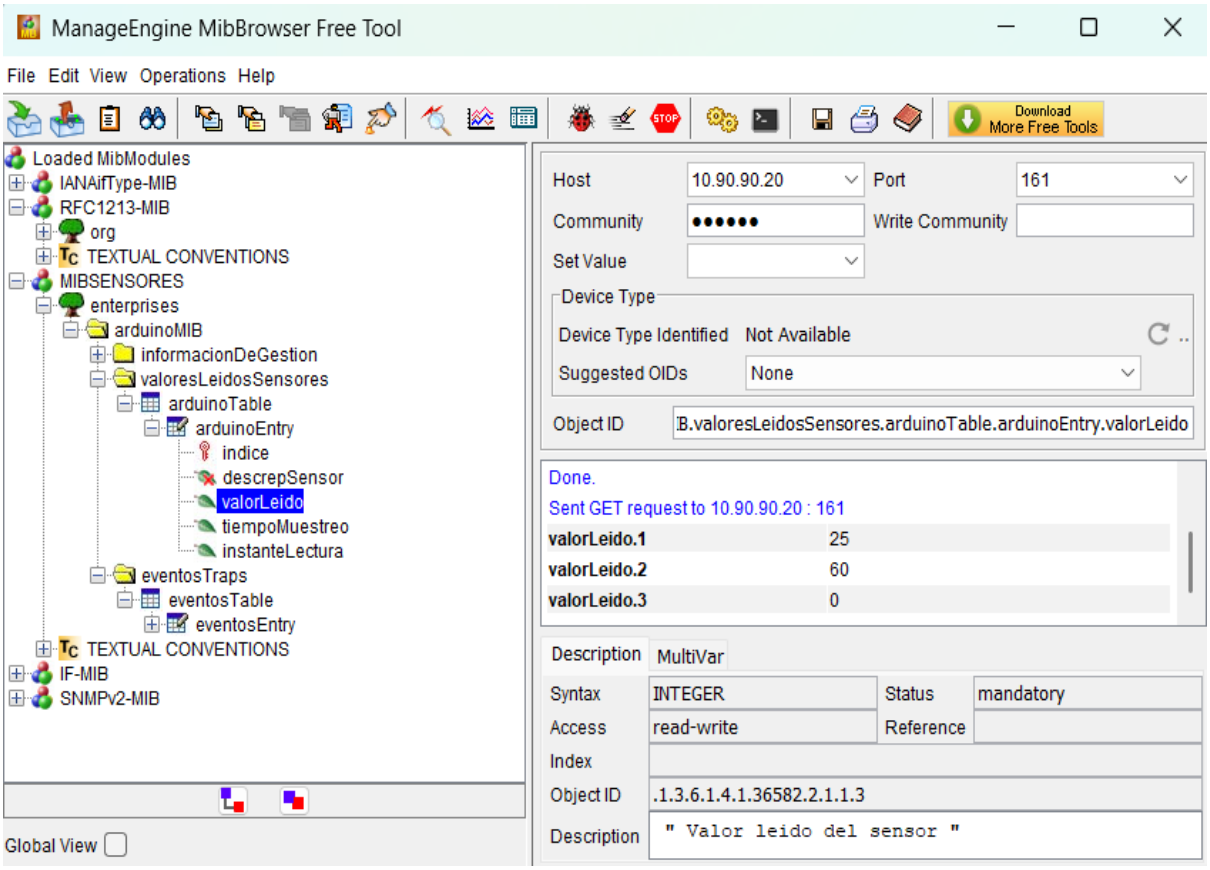


*Figura 5.2: Carga de la MIB personalizada en ManageEngine MIB Browser.*

Posteriormente, para visualizar de manera estructurada el contenido de la tabla, se utilizó la opción SNMP Table disponible en MIbBrowser. La Figura 5.3 muestra la tabla valoresLeidosSensores, donde se registran en cada fila el índice del sensor, la descripción, el valor leído, el tiempo de muestreo y el instante de lectura. En este caso, se aprecia cómo se reportan simultáneamente las mediciones de temperatura, humedad y el estado del LED.

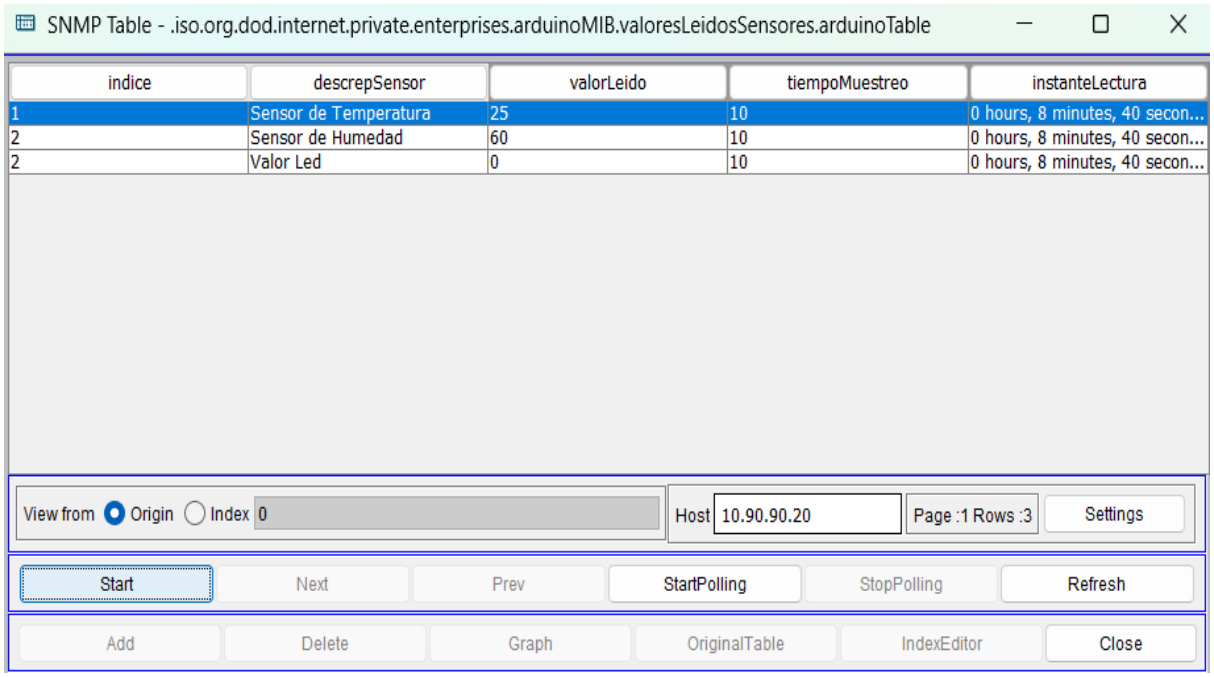


*Figura 5.3: Tabla de valoresLeidosSensores mostrada en ManageEngine MIB Browser.*

La Figura 5.4 muestra una operación **GET** realizada desde MibBrowser sobre el objeto encargadoDeGestion. En esta prueba, el gestor envía una petición de lectura al Arduino, que responde con el valor almacenado en la MIB.

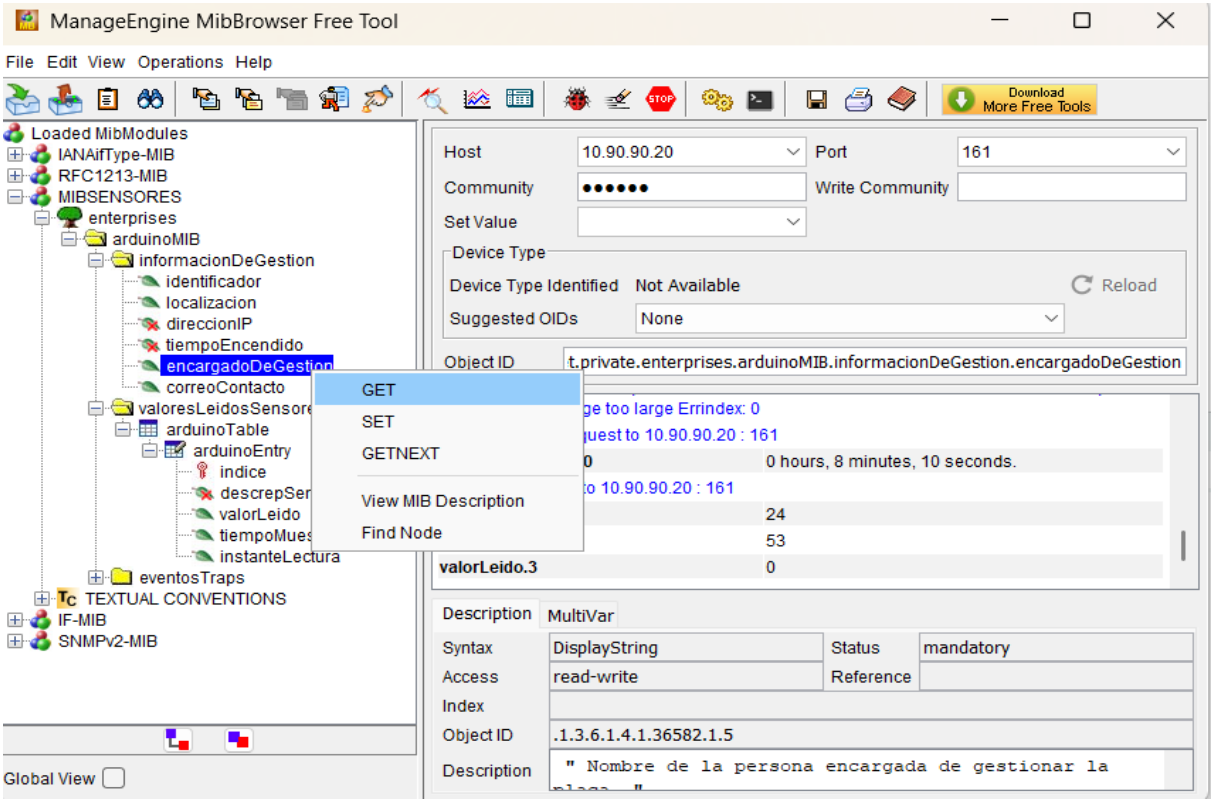


*Figura 5.4: Operación GET sobre la variable encargadoDeGestion.*

La validez de esta operación se confirmó con Wireshark, donde se capturo el intercambio de paquetes entre el PC (gestor) y el Arduino (agente). En la Figura 5.5 se aprecia la petición **GET-REQUEST** enviada desde el gestor para consultar el valor del objeto encargadoDeGestion (OID 1.3.6.1.4.1.36582.1.5). La Figura 5.6 muestra la GET-RESPONSE correspondiente que contiene como valor la cadena “Luis Carlos Deza”.

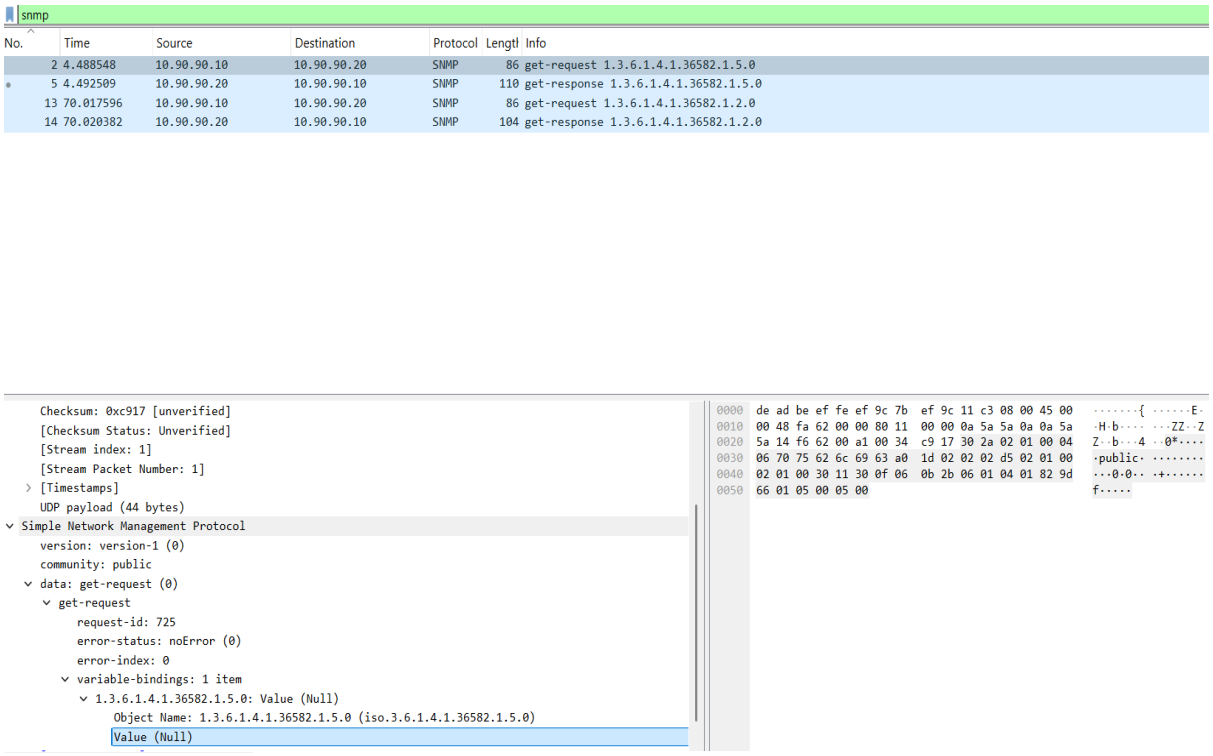


*Figura 5.5: Captura en Wireshark de la petición GET-REQUEST.*

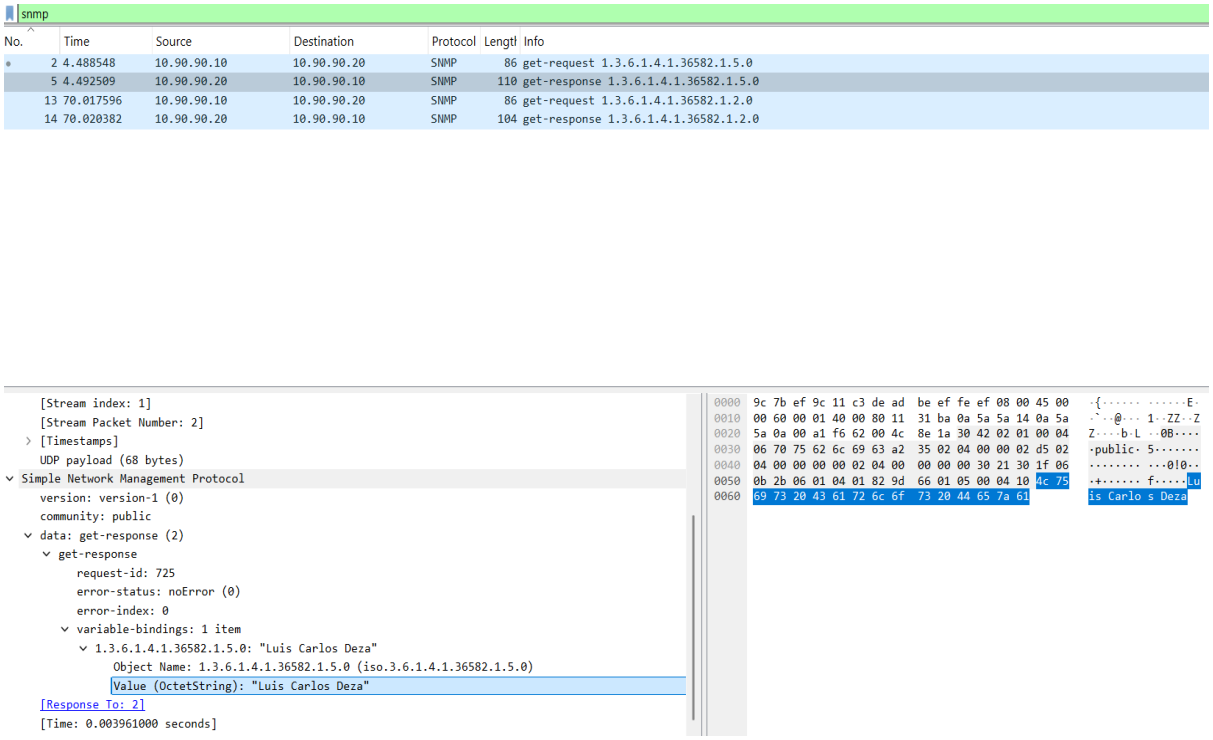


*Figura 5.6: Captura en Wireshark de la respuesta GET-RESPONSE.*

Posteriormente se validaron las operaciones **SET**, que permiten modificar valores en el agente desde el gestor MIBbrowser. En la Figura 5.7 se observa cómo se modifica el valor del campo tiempoMuestreo en la tabla valoresLeidosSensores, asignándole el valor 5 para el sensor de temperatura. Como consecuencia, el instante de lectura asociado pasa de 5 a 10.

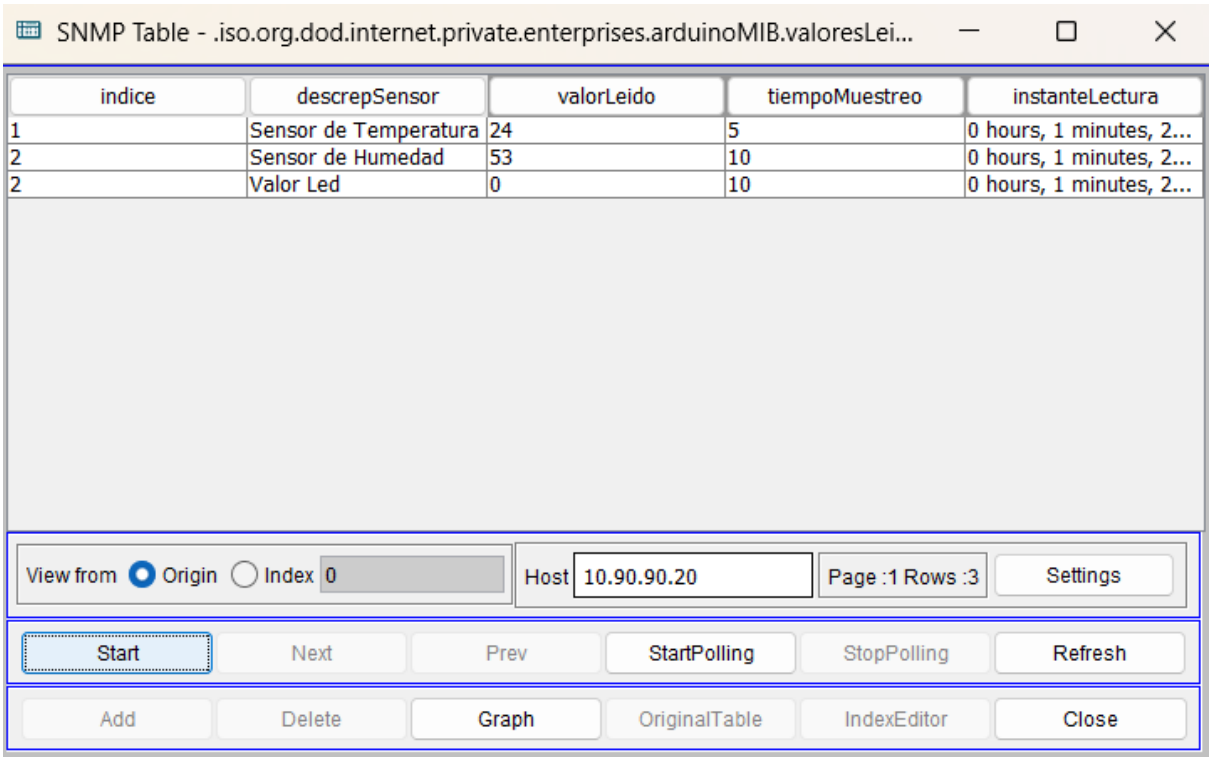


*Figura 5.7: Cambio tiempoMuestreo a 5 segundos.*

En la Figura 5.8 se muestra una captura en Wireshark donde el gestor envia un SET-EQUEST sobre un OID correspondiente al tiempoMustreo (1.3.6.1.4.1.36582.2.1.1.4.1). Finalmente, la Figura 5.9 muestra la confirmación con el valor actualizado recibido en la respuesta.

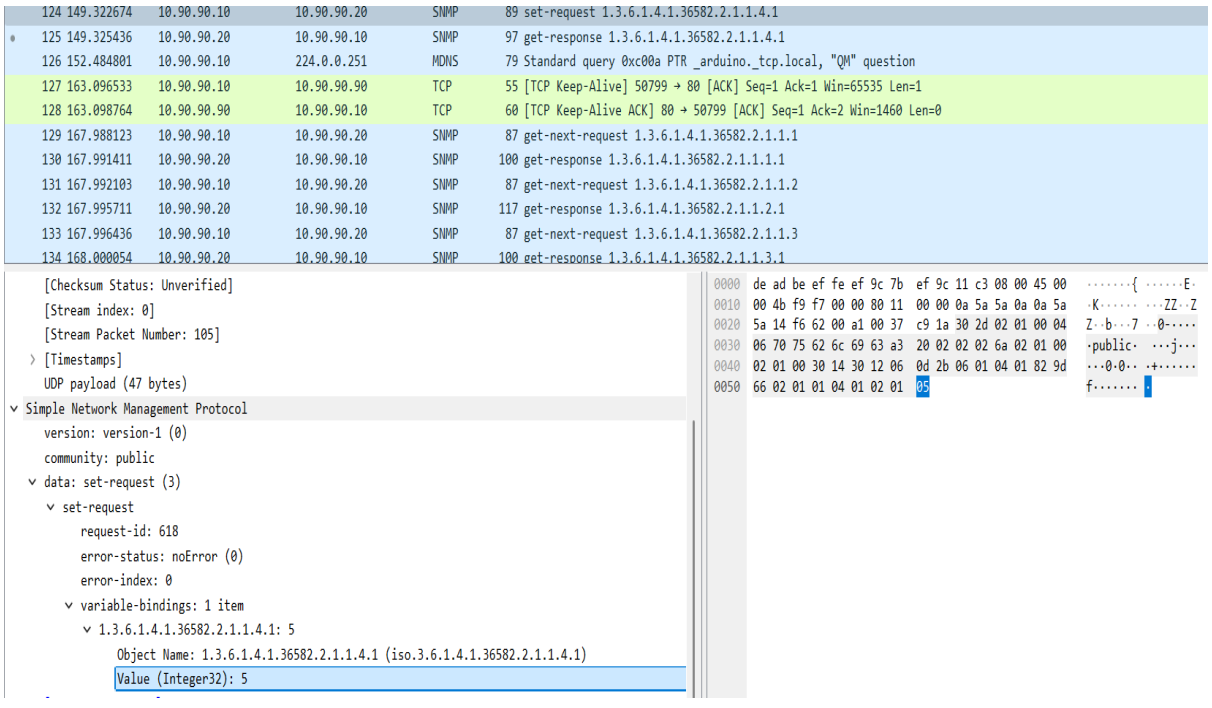


*Figura 5.8: Captura en Wireshark de la petición SNMP SET enviada desde el gestor hacia el Arduino.*

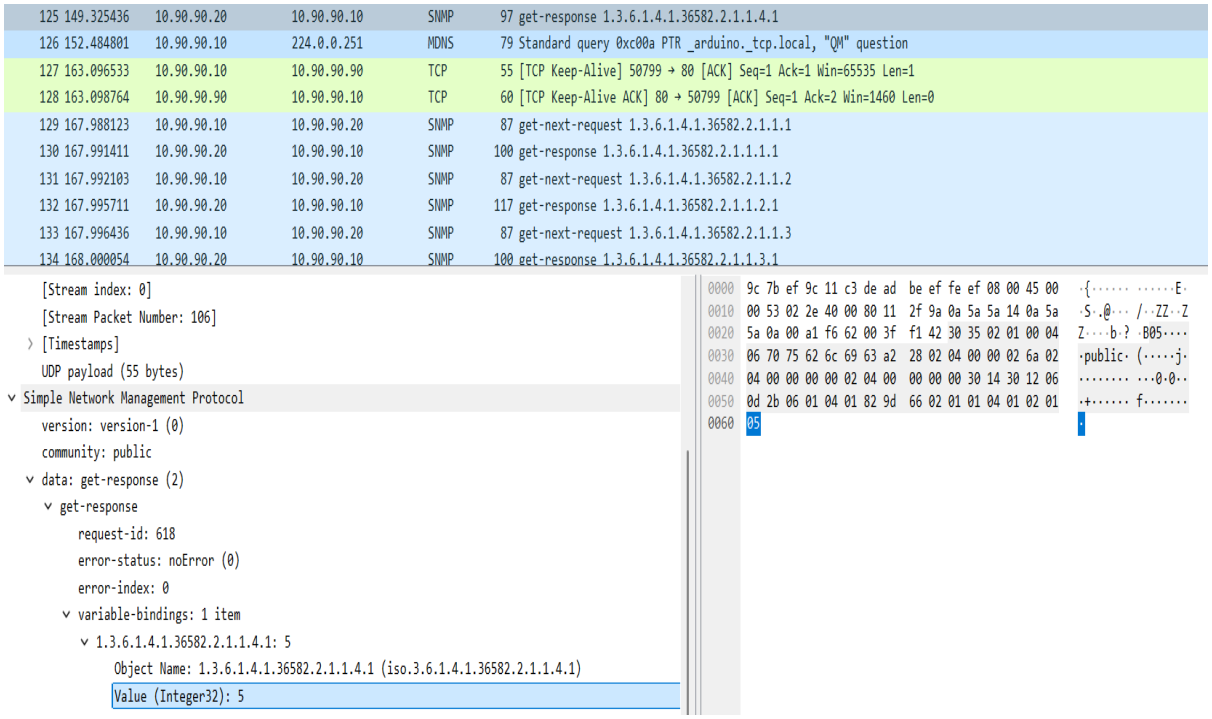


*Figura 5.9: Captura en Wireshark de la respuesta a la operación SET confirmando la actualización del valor.*

Finalmente, se verifico el recorrido secuencial de la tabla definida en la MIB mediante operaciones GET-NEXT. La Figura 5.10 muestra en MibBrowser la selección de la operación GET-NEXT sobre el objeto que vamos a hacer un GET-NEXT al objeto descrepSensor (OID 1.3.6.1.4.1.36582.2.1.1.2), lo que permite solicitar el siguiente elemento en la jerarquía de la MIB que será el OID 1.3.6.1.4.1.36582.2.1.1.2.1.

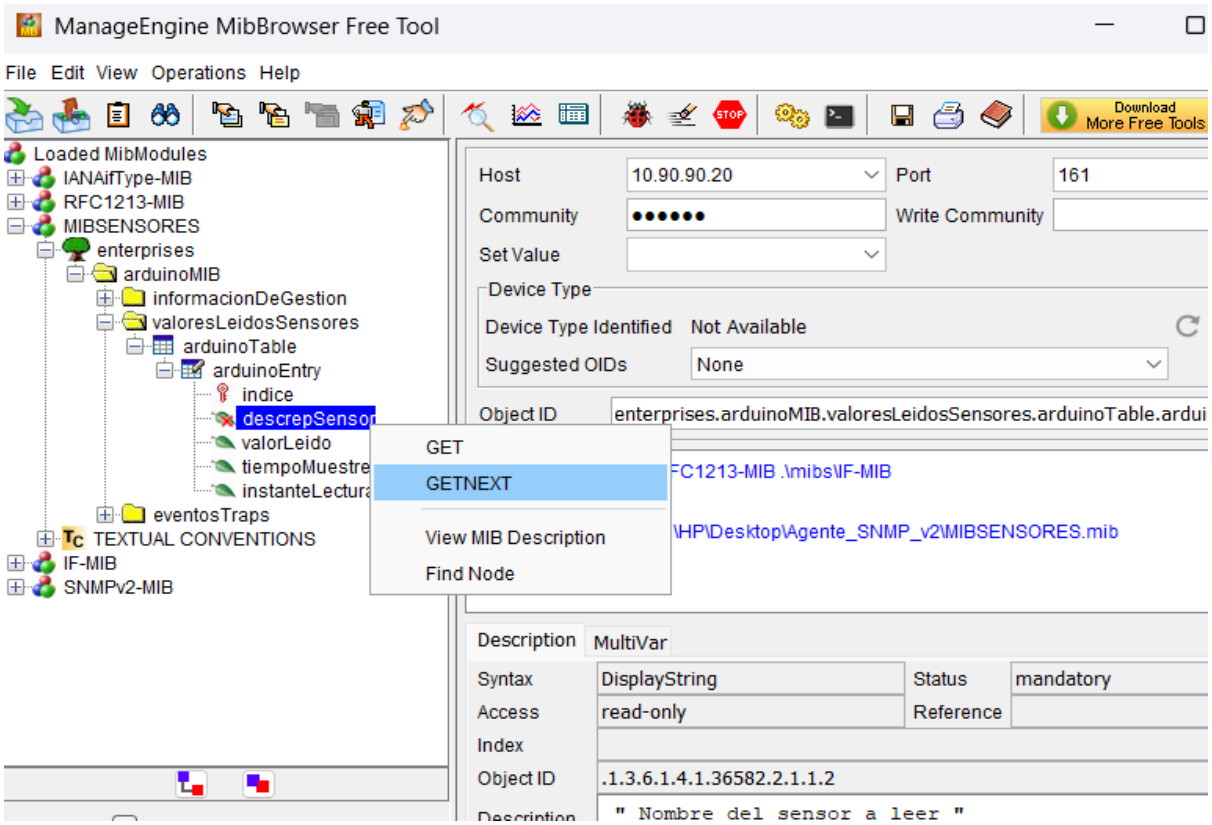


*Figura 5.10: Operación GETNEXT sobre la variable encargadoDeGestion.*

La Figura 5.11 se muestra una captura en Wireshark donde se observa el paquete GET-NEXT-REQUEST enviado por el gestor al agente Arduino, mientras que en la Figura 5.12 se muestra la GET-RESPONSE correspondiente, donde se devuelve como resultado el siguiente objeto disponible, en este caso el identificador del sensor de temperatura.

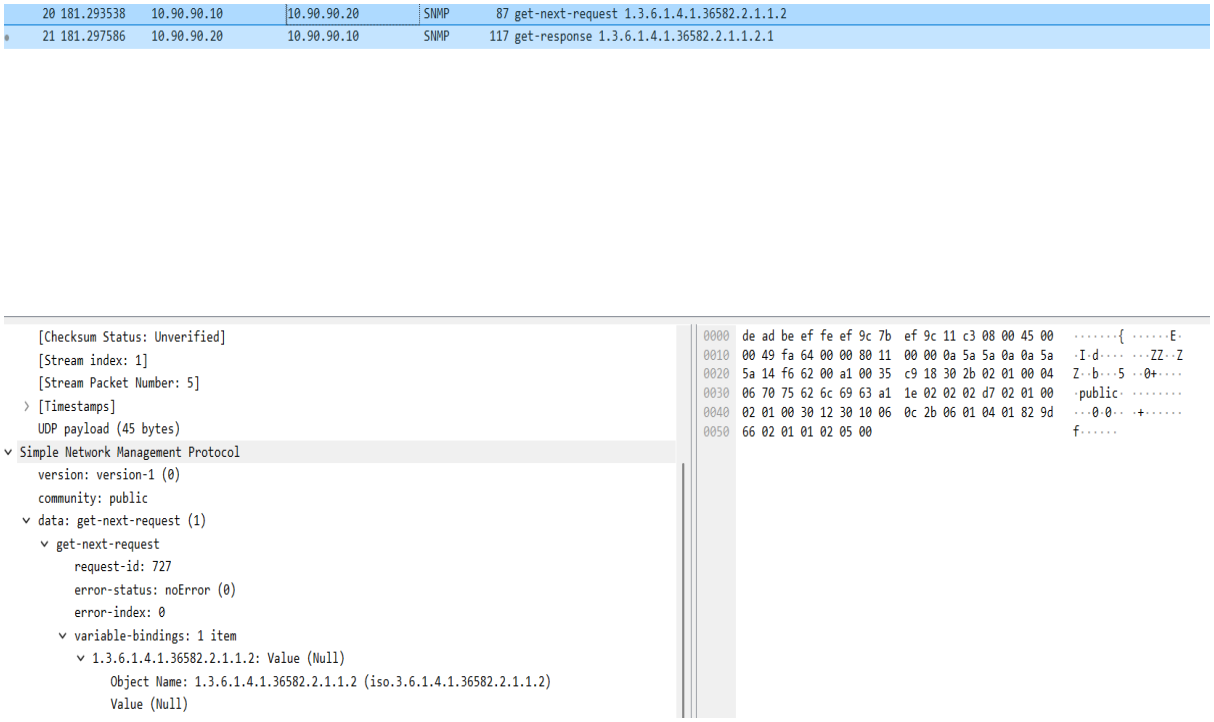


*Figura 5.11: Captura en Wireshark de la petición GET- NEXT.*

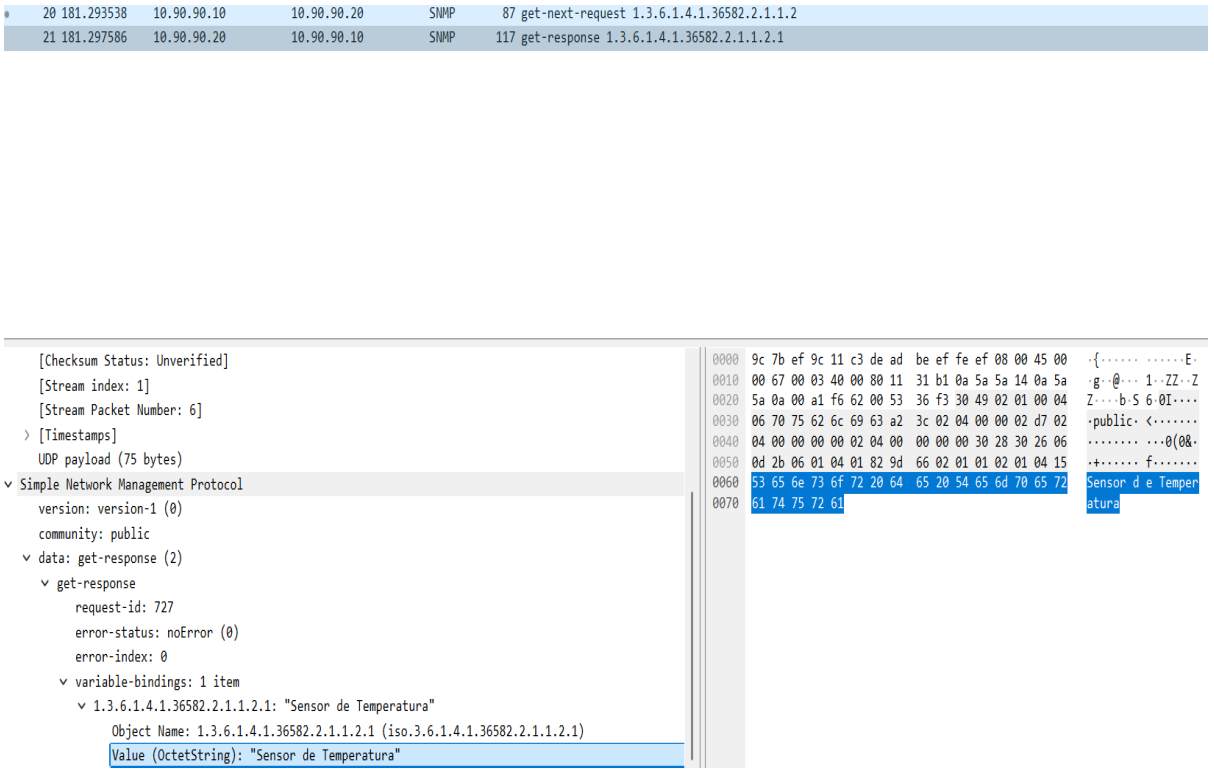


*Figura 5.12: Captura en Wireshark de la respuesta GET-NEXT con el valor del objeto leído.*

## 5.3. Integración con Telegraf, InfluxDB y Grafana

Una vez comprobado que las operaciones básicas de consultas SNMP sobre el Arduino funcionan correctamente, se procedió a integrar un sistema de visualización basado en Telegraf + InfluxDB + Grafana con el objetivo de representar de una forma más visual los valores de temperatura y humedad del sensor DHT11 conectado al Arduino. Este sistema permitió automatizar la recolección de métricas de los sensores de temperatura y humedad, almacenarlas en una base de datos de series temporales y representarlas gráficamente en paneles dinámicos y en tiempo real.

El proceso se inicia en Telegraf, donde se configuró un archivo denominado “arduino_snmp.conf” para definir al Arduino como agente SNMP. En este archivo se especifica parámetros como la dirección IP del Arduino (10.90.90.20), la comunidad pública y los OIDs correspondientes a la temperatura y la humedad definidas en la MIB [MIB.h].

La Figura 5.13 muestra el archivo de configuración de Telegraf para la captura de OIDs de temperatura y humedad del Arduino.

arduino_snmp.conf

C: > Program Files > telegraf > telegraf-1.34.4 > arduino_snmp.conf

```
[[inputs.snmp]]
  agents = [ "10.90.90.20:161" ]  # Cambia esta IP si tu Arduino usa otra
  version = 2
  community = "public"
  name = "arduino_snmp"

  [[inputs.snmp.field]]
    name = "temperatura"
    oid = "1.3.6.1.4.1.36582.2.1.1.3.1"

  [[inputs.snmp.field]]
    name = "humedad"
    oid = "1.3.6.1.4.1.36582.2.1.1.3.2"

[[outputs.influxdb_v2]]
  urls = ["http://localhost:8086"]
  token = "OEC0jTRmt1TTckjk_OlAJ1PIQuIL-zVJKOs703D1LDk-6NwRP69Wd8wHp93wtltI73lWHpsNBp3UIIXORo8fqA=="
  organization = "Logos"
  bucket = "valores_sensorDHT11"
```

*Figura 5.13: Archivo configuración Telegraf arduino_snmp.conf.*

Los datos recogidos por Telegraf son enviados a InfluxDB, una base de datos de series temporales que permite almacenar mediciones con marca de tiempo. Para ello, se creó un bucket denominado valores_sensorDHT11, donde quedaran registradas todas las lecturas.

La Figura 5.14 muestra la creación de una bucket en InfluxDB.

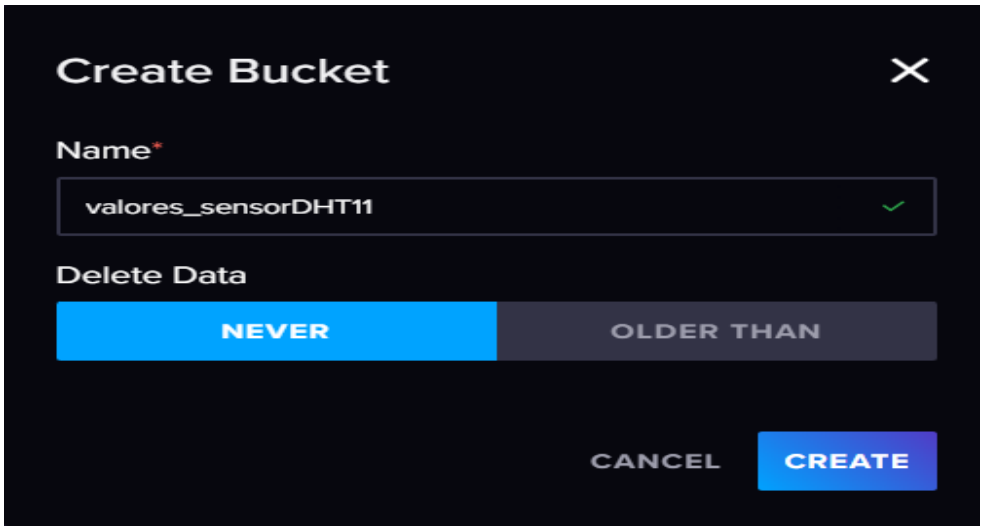


*Figura 5.14: Creación bucket valores_sensorDHT11.*

Posteriormente, se estableció la conexión entre InfluxDB y Grafana, vinculando el bucket creado como fuente de datos. Una vez realizada esta integración, Grafana quedo habilitado para realizar consultas a la base de datos en InfluxDB.

La Figura 5.15 muestra la configuración de la fuente de datos en Grafana, en la que se define la conexión con InfluxDB.

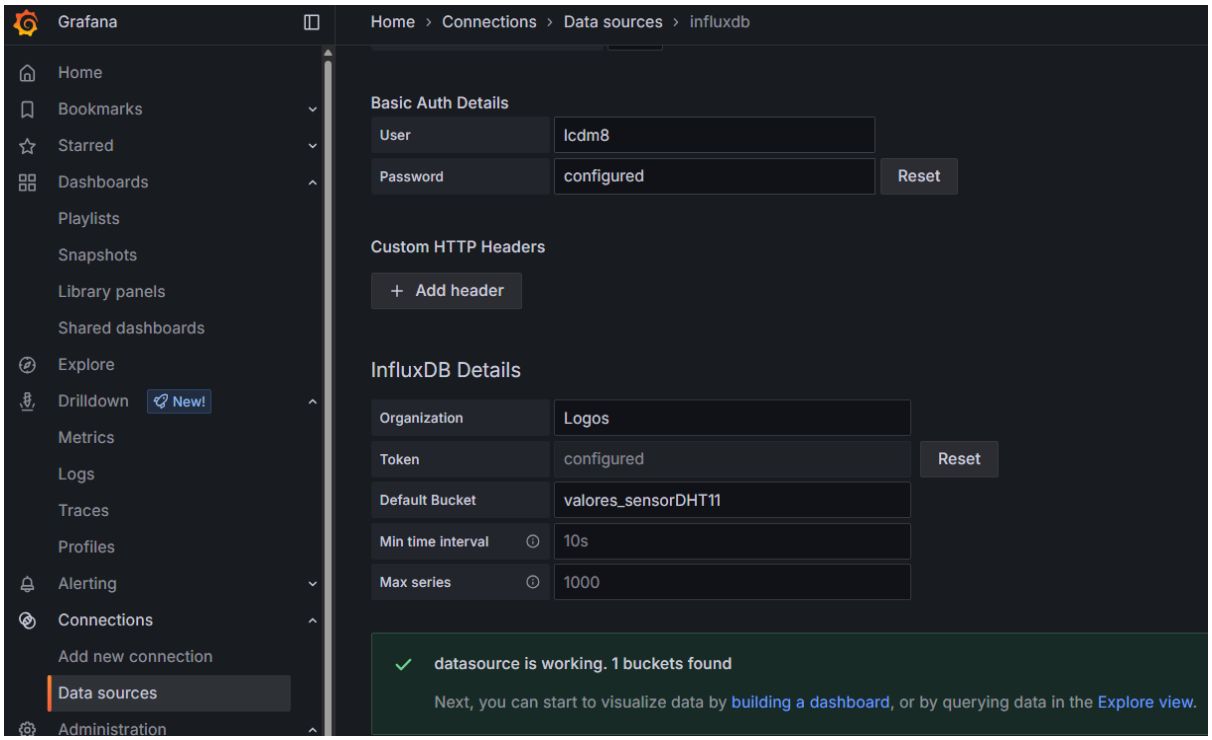


*Figura 5.15: Configuración de la fuente de datos InfluxDB en Grafana vinculada al bucket valores_sensorDHT11.*

Con la conexión activa, fue posible definir consultas que permiten extraer y filtrar los valores registrados de cada variable gestionada.

En la Figura 5.16 se muestra un ejemplo de consulta realizada en Grafana para obtener los valores de temperatura desde el bucket en InfluxDB, mientras que en la Figura 5.17 se observa la consulta realizada para la variable de humedad.

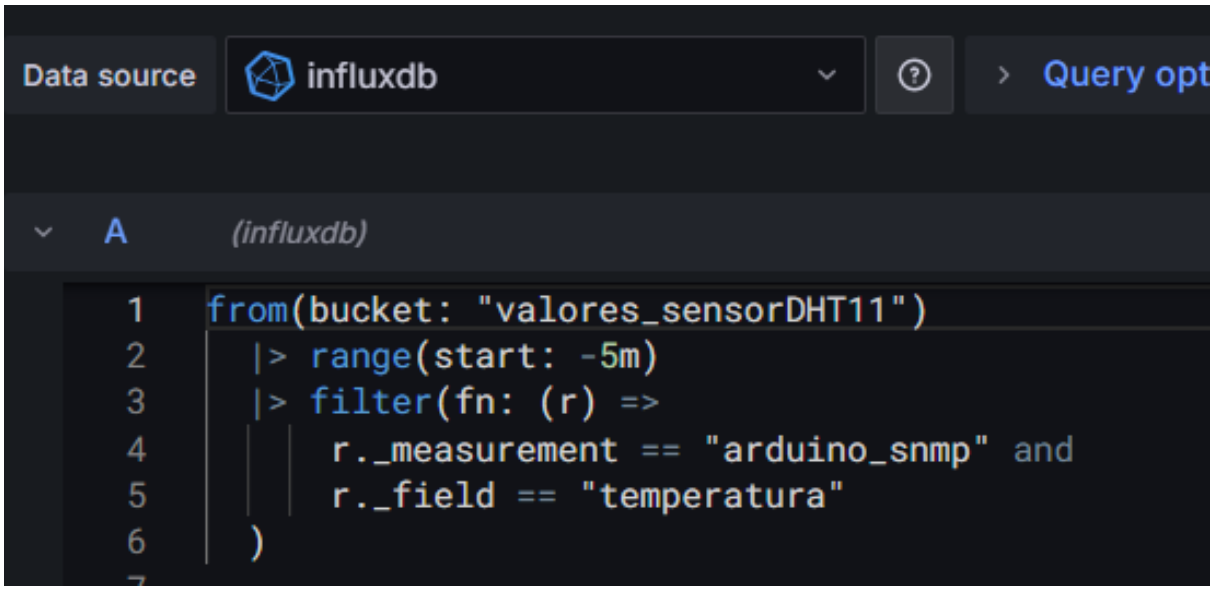


*Figura 5.16: Consulta Grafana para valores de Temperatura.*

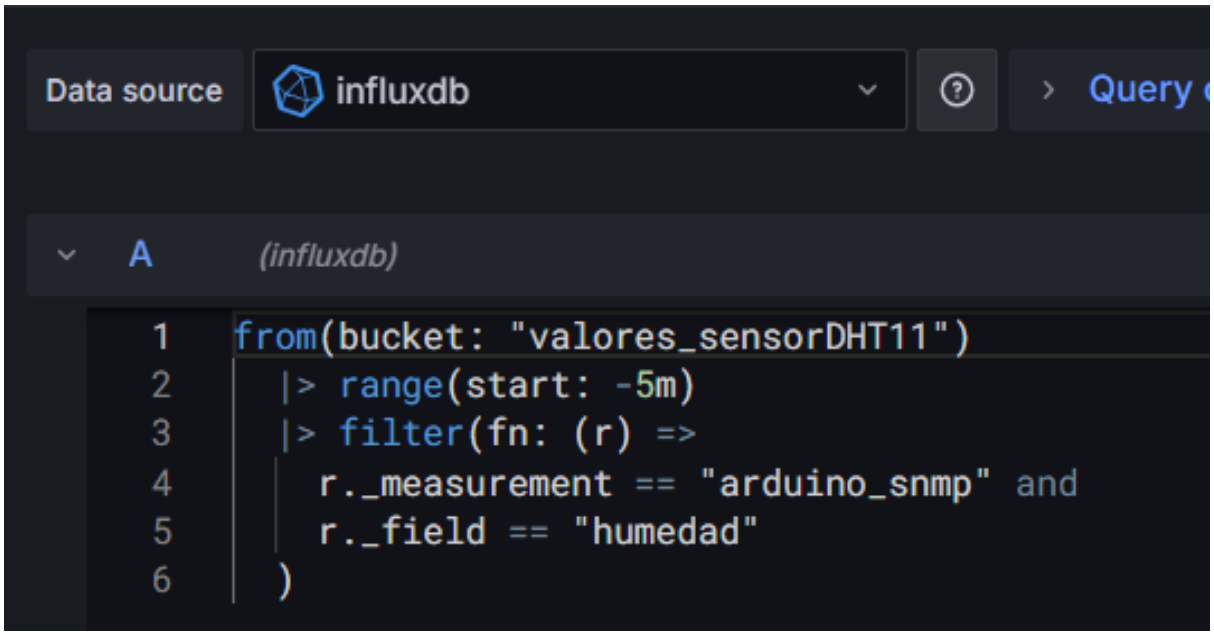


*Figura 5.17: Consulta Grafana para valores de Humedad.*

Finalmente, se diseñó un dashboard en Grafana que incluye dos paneles: uno para la temperatura y otro para la humedad. En cada grafico se representa gráficamente la evolución de las mediciones en un intervalo de tiempo configurable.

La Figura 5.18 muestra el panel resultante, en el que se representa las curvas de temperatura y humedad recolectadas desde el Arduino a través de SNMP.


*Figura 5.18: Curvas temperatura y humedad en Grafana.*

De esta manera, la integración de Telegraf, InfluxDB y Grafana no solo permite observar los valores instantáneos del sensor DHT11, sino también contar con un histórico de datos y también con un análisis de las tendencias.

## 5.4. Control remoto del LED mediante API Flask + Grafana

### 5.4.1. Gestión LED mediante API en Python (Flask + pysnmp)

Para el control remoto del LED desde Grafana, fue necesario un puente entre la interfaz gráfica y el protocolo SNMP. Para esto se desarrolló una API en Python utilizando el microframework Flask que traduce las órdenes enviadas desde Grafana en operaciones SET hacia el Arduino.

En el Código 5.1 se muestra la funcion_snmp_set(), encargada de enviar una instrucción SNMP SET al agente Arduino. Esta función recibe como argumento un valor número (0 o 1) que representa el estado del LED. La librería pysnmp se encarga de construir el paquete SNMP, especificando la comunidad (public), la dirección IP del Agente que es el Arduino ( 10.90.90.20), el puerto estándar UDP (161) y el OID del LED definido en la MIB (1.3.6.1.4.1.36582.2.1.1.3.3). La función envía la petición y devuelve un resultado indicando si la operación fue correcta o si hubo algún error de red.

```python
def enviar_snmp_set(valor):
    print(f"\n[INFO] Enviando SNMP SET: valor = {valor}")
    errorIndication, errorStatus, errorIndex, varBinds = next(
        setCmd(
            SnmpEngine(),
            CommunityData(COMMUNITY, mpModel=0),  # SNMPv1
            UdpTransportTarget((ARDUINO_IP, 161)),
            ContextData(),
            ObjectType(ObjectIdentity(OID_LED), Integer(valor))
        )
    )

    if errorIndication:
        print("[ERROR] SNMP Error:", errorIndication)
        return False
    elif errorStatus:
        print(f"[ERROR] SNMP Set fallido: {errorStatus.prettyPrint()} at index {errorIndex}")
        return False
    else:
        print("[SUCCESS] SNMP SET enviado correctamente. Respuesta del agente:")
        for varBind in varBinds:
            print("   →", varBind)
        return True
```

*Código 5.1: Función principal de envío de un SNMP SET desde Python.*

En el Código 5.2 se presenta la definición de una ruta web (/led) dentro de Flask, que actúa como un punto de entrada (endpoint) para recibir peticiones externas. Un endpoint es una dirección expuesta por la API que permite a otros sistemas enviar ordenes o consultar información.
En este caso, cuando Grafana envía una petición HTTP POST a la ruta /led, el servidor Flask recibe el mensaje, extrae el valor transmitido (0 o 1) y lo reenvía al agente mediante la función enviar_snmp_set().

De esta forma, la API actúa como “traductor” entre el mundo HTTP (Grafana) y el protocolo SNMP (Arduino), permitiendo que el LED pueda encenderse o apagarse desde un panel interactivo en Grafana.

```python
@app.route('/led', methods=['POST']) #ruta /led que acepta el método POST
def controlar_led(): #actúa como puerta de entrada (API)
    data = request.get_json()
    valor = int(data.get("valor", 0))  # 0 o 1
    print(f"[POST] Petición recibida desde frontend: valor = {valor}")
    exito = enviar_snmp_set(valor)#lanza el set al arduino
    return jsonify({"ok": exito, "estado": valor})
```

*Código 5.2: Definición del endpoint en Flask.*

En la Figura 5.19, se observa el flujo de comunicación implementado para el control remoto del LED. Los botones Encendido/Apagado configurados en Grafana envían una petición HTTP POST al endpoint /led de la API Flask.

Esta API, desarrollada en Python, recibe la petición POST y traduce la orden recibida en una operación SET utilizando la librería pysnmp y la envía al Arduino UNO, a través del puerto UDP/161. El Arduino UNO, actuando como agente SNMP mediante su Ethernet Shield, procesa el mensaje SET recibido, actualiza la variable asociada al LED y ejecuta la acción física de encender o apagar el LED.

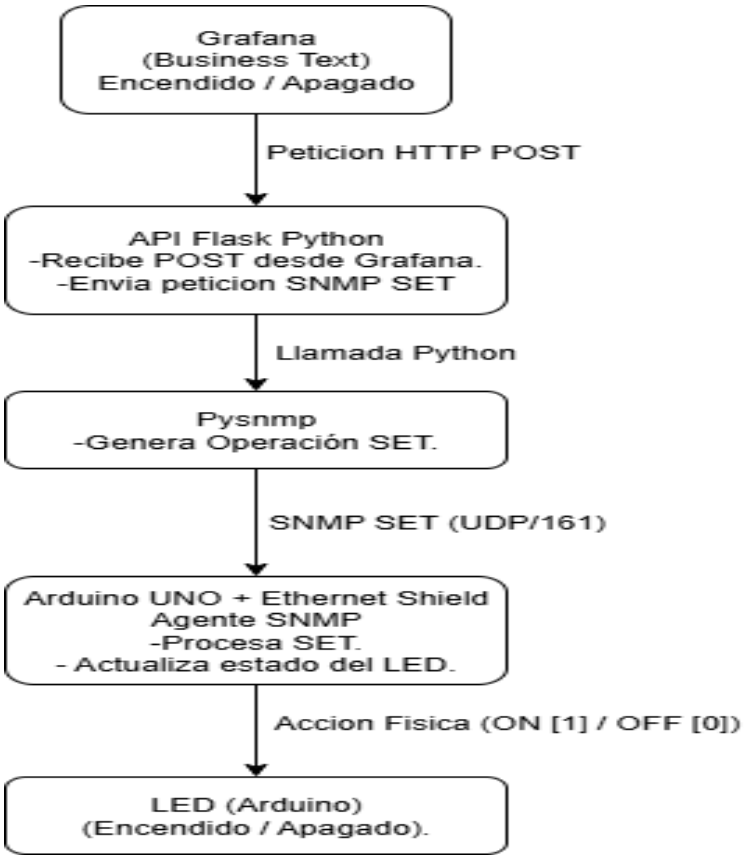


*Figura 5.19: Diagrama de flujo de la integración Grafana–Flask–Arduino para el control del LED.*

En el Anexo IV se presenta el archivo set_led.py, que contiene la API Flask desarrollada en Python.

### 5.4.2. Integración en Grafana con Business Text.

En Grafana, la integración se realizó mediante el plugin Business Text, que permite incluir contenido dinámico mediante HTML y JavaScript. Se configuró con botones interactivos: Encender y Apagar que envían peticiones POST a la API desarrollada en Flask.

En el Código 5.3 se muestra el fragmento de HTML definido en el campo Default Content. Allí se declaran dos botones (“Encender” y “Apagar”), junto con un elemento de texto que refleja el estado actual del LED.

```html
<h2>Control de LED Arduino</h2>

<button id="btn-on">🔆 Encender</button>
<button id="btn-off">💡 Apagar</button>

<p id="respuesta">Estado: esperando…</p>
```

*Código 5.3: Diagrama de flujo de la integración Grafana–Flask–Arduino para el control del LED.*

En el Código 5.4 se presenta el bloque de JavaScript que se incluye en el campo After Content Ready. Este script implementa la función enviar(), que recibe como parámetro el estado del LED (0 OFF y 1 ON). La función construye una peticion HTTP POST y la envía al endpoint /led expuesto en la API Flask.

```javascript
function enviar(valor) {
  document.getElementById("respuesta").innerText = "Enviando comando...";
  fetch("http://localhost:5000/led", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ valor: valor })
  })
    .then(res => res.json())
    .then(data => {
      document.getElementById("respuesta").innerText =
        "LED " + (data.estado === 1 ? "ENCENDIDO" : "APAGADO");
    })
    .catch(err => {
      document.getElementById("respuesta").innerText =
        "❌ Error al enviar comando";
    });
}

document.getElementById("btn-on").addEventListener("click", () => enviar(1));
document.getElementById("btn-off").addEventListener("click", () => enviar(0));
```

*Código 5.4: Código JavaScript para la comunicación con la API Flask en el control del LED.*

De esta forma, al pulsar Encender o Apagar en Grafana se envía una petición HTTP POST a la API en Python, que traduce el comando en un SNMP SET hacia el Arduino, consiguiendo el control remoto en tiempo real del LED desde el dashboard.

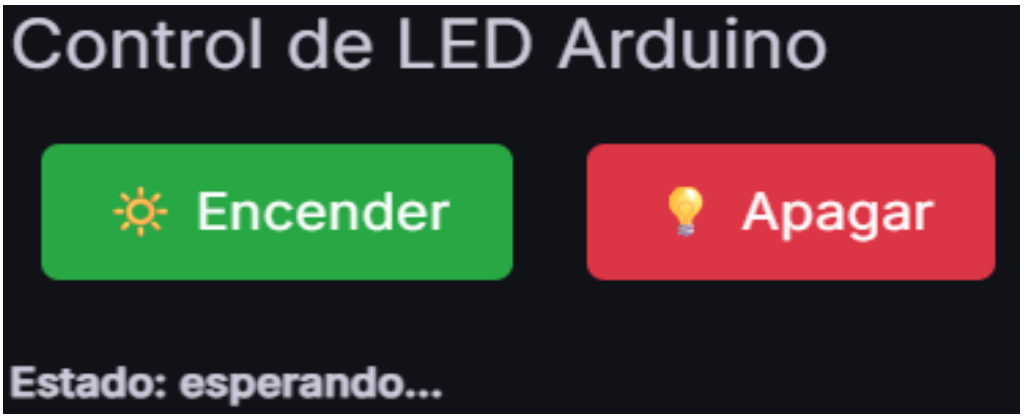


*Figura 5.20: Panel de Grafana con botones para el control remoto del LED.*

## 5.5. Validación del envió de correo electrónico.

Para comprobar la capacidad del sistema de generar de forma automática un correo electrónico, se configuró el gestor PRTG Network Monitor como receptor de traps SNMP en el puerto estándar UDP/162. El Arduino, actuando como agente SNMP, genera una trap cuando se superan determinados umbrales de temperatura o humedad, siendo el gestor el encargado de recibirla y procesarla.

### 5.5.1. Validación de traps con Wireshark

Antes de realizar la integración con PRTG, se verificó el correcto funcionamiento del envío de traps SNMP mediante el uso de Wireshark.

En primer lugar, tal y como se aprecia en la Figura 5.21, se configuró en la tabla eventosTable del gestor Manage MIB Browser el objeto ifIndexSensor, asignando el valor 1 al indiceEv. Este índice representa el valor asociado al sensor de temperatura. Para este caso de prueba, se estableció un umbral superior en 26, de modo que cuando ese valor se supere, el agente debía generar y enviar una trap al gestor configurado.

La Figura 5.22 muestra una captura de Wireshark, donde se aprecia un paquete SNMP trap enviado desde el agente Arduino con dirección IP 10.90.90.20 hacia el gestor PC con dirección IP 10.90.90.10, utilizando el puerto UDP 162. El paquete contiene el OID 1.3.6.1.4.1.36582 y el valor "mensaje evento1 Valor: 27", confirmando que el umbral establecido fue superado.

Esta comprobación garantizó que los traps estaban siendo generados y transmitidos correctamente antes de configurar la notificación en PRTG.

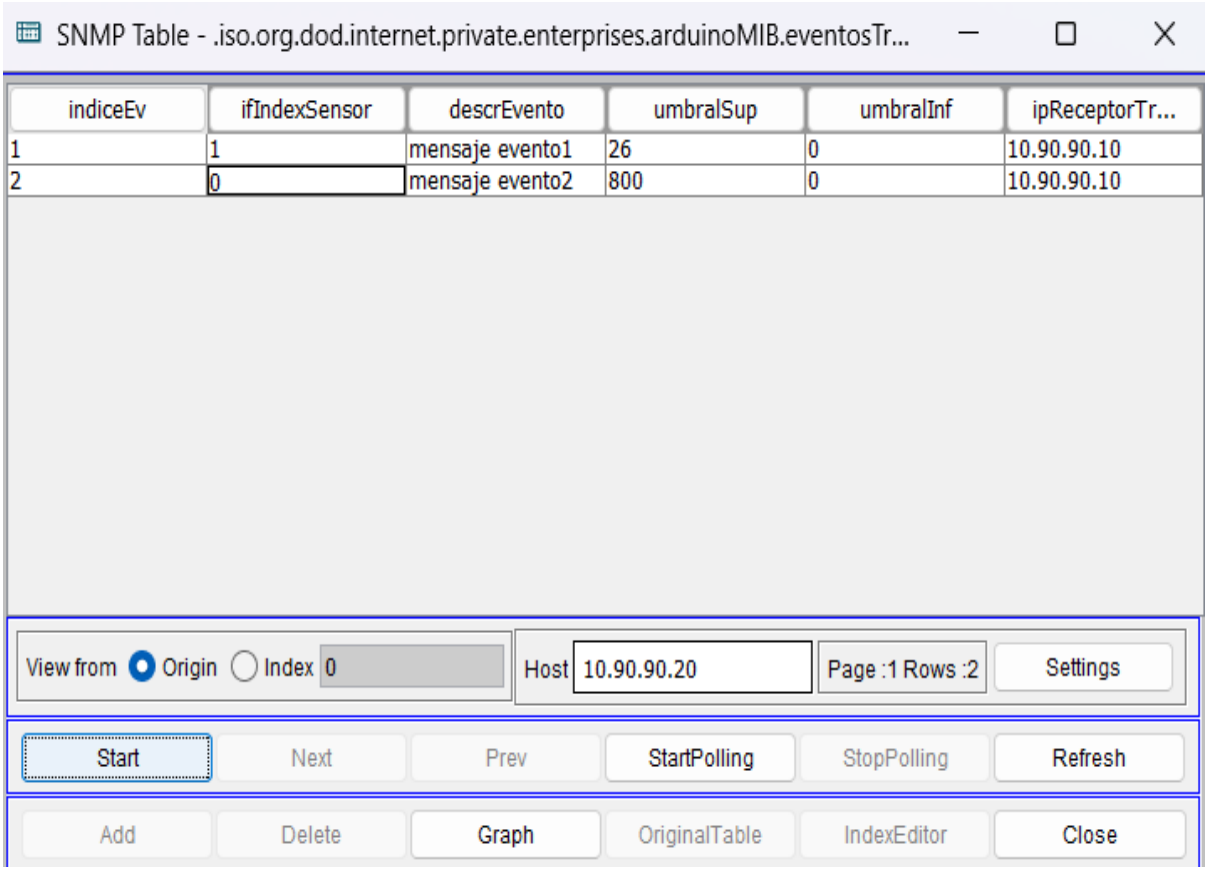


*Figura 5.21: Activar evento 1 en Tabla eventosTable.*

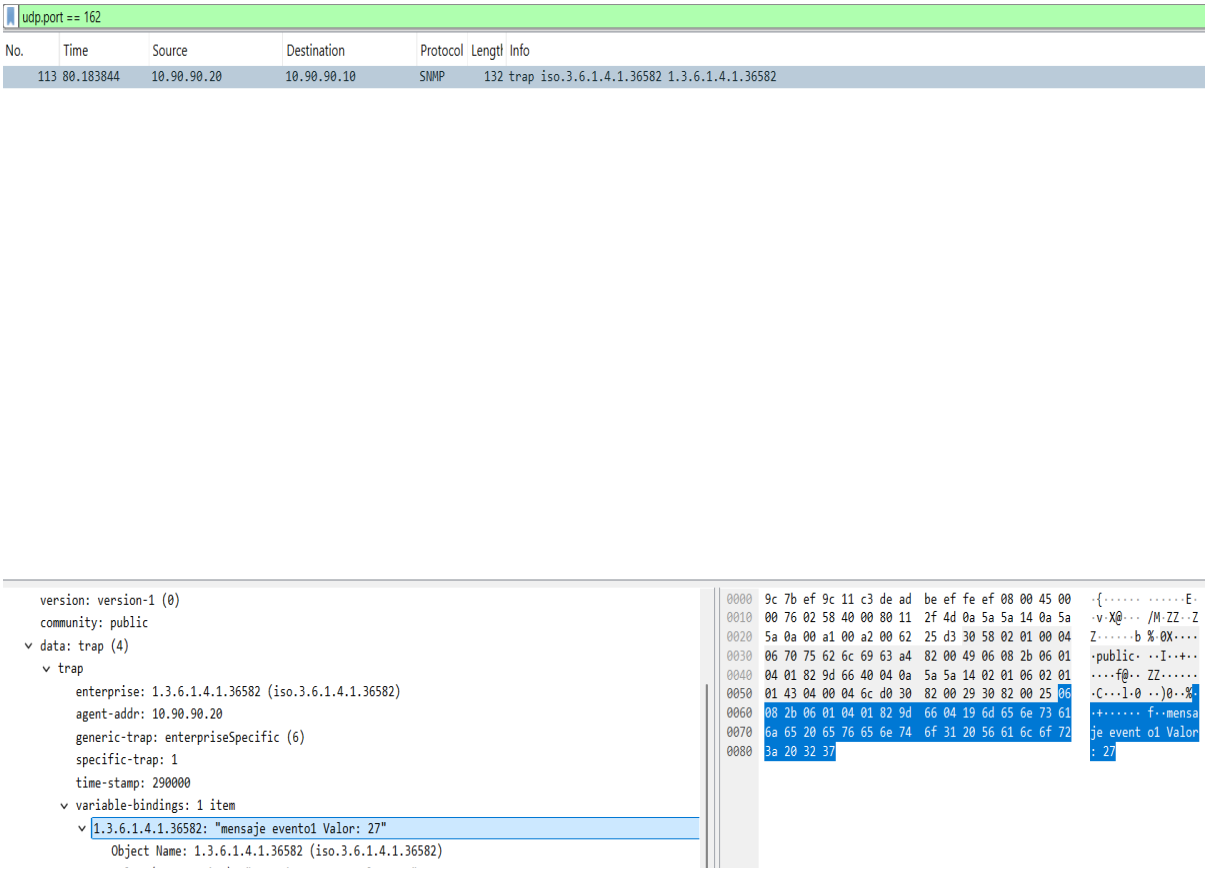


*Figura 5.22: Captura Wireshark Trap.*

### 5.5.2. Configuración en PRTG Network Monitor

Una vez comprobado con Wireshark que el agente enviaba correctamente las traps, se configuró PRTG Network Monitor como gestor SNMP. Para ello, se añadió a la subred 10.90.90.0/24 el dispositivo correspondiente al agente SNMP con dirección IP 10.90.90.20. Sobre este dispositivo se habilitó el sensor Receptor de trap SNMP, encargado de recibir las traps generadas por el Arduino.

En la Figura 5.23 se muestra la estructura de la subred 10.90.90.0/24 dentro de PRTG y la inclusión del dispositivo 10.90.90.20 (agente SNMP). Sobre este dispositivo se añadió el sensor Receptor de trap SNMP. El cual actúa receptor de las traps enviadas por del agente SNMP.[19]

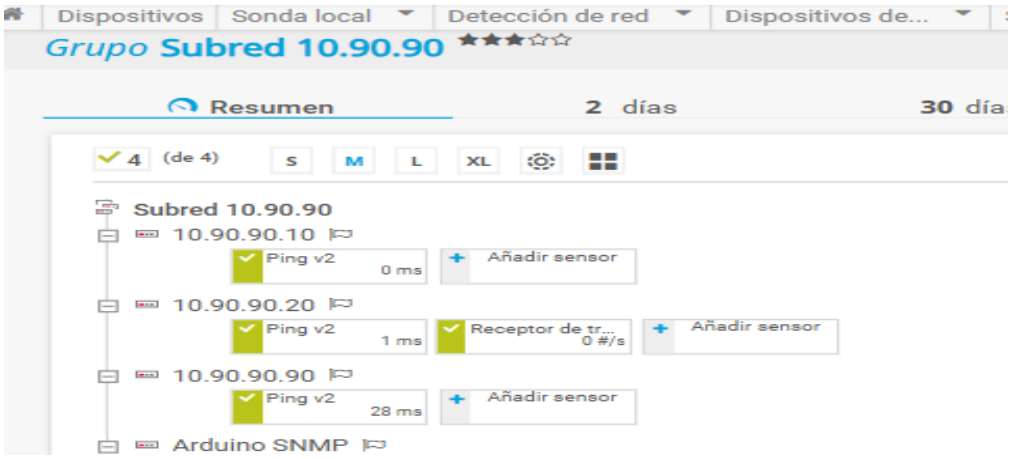


*Figura 5.23: Dispositivos dentro de PRTG.*

En la Figura 5.24 se muestra cuando entramos al sensor Receptor de trap SNMP. En el apartado Trap messages se visualizan las traps recibidas que se han generado; en este caso aparece el texto "mensaje evento1 Valor: 27", lo que confirma que el agente ha superado el umbral establecido y ha generado la notificación correspondiente. [19]

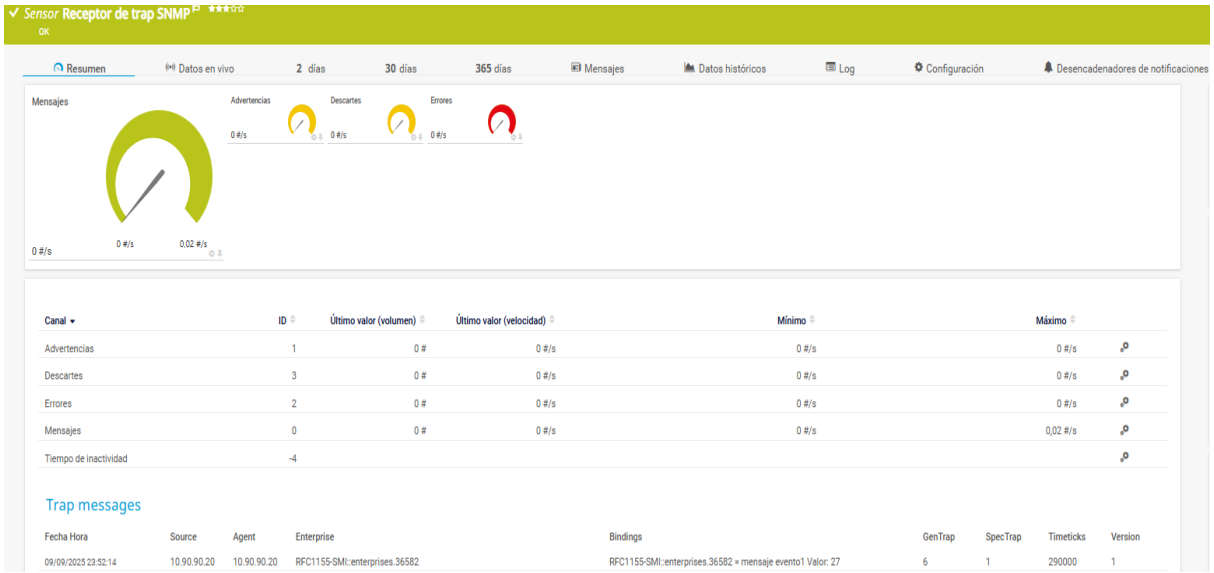


*Figura 5.24: Receptor de trap SNMP en PRTG.*

Con esta configuración, cada vez que el agente genera un trap por superación de umbral, PRTG envía un correo electrónico al administrador.

### 5.5.3. Envió de correo electrónico

Finalmente, se verificó la capacidad de PRTG para notificar al administrador mediante correo electrónico cada vez que se recibe una trap del agente SNMP.

En la Figura 5.25 se muestra un ejemplo del mensaje recibido en la cuenta configurada, donde se indica que el sensor Receptor de trap SNMP ha detectado la llegada del evento “mensaje evento1 Valor: 27”, tal y como se ha demostrado en la Figura 5.24.

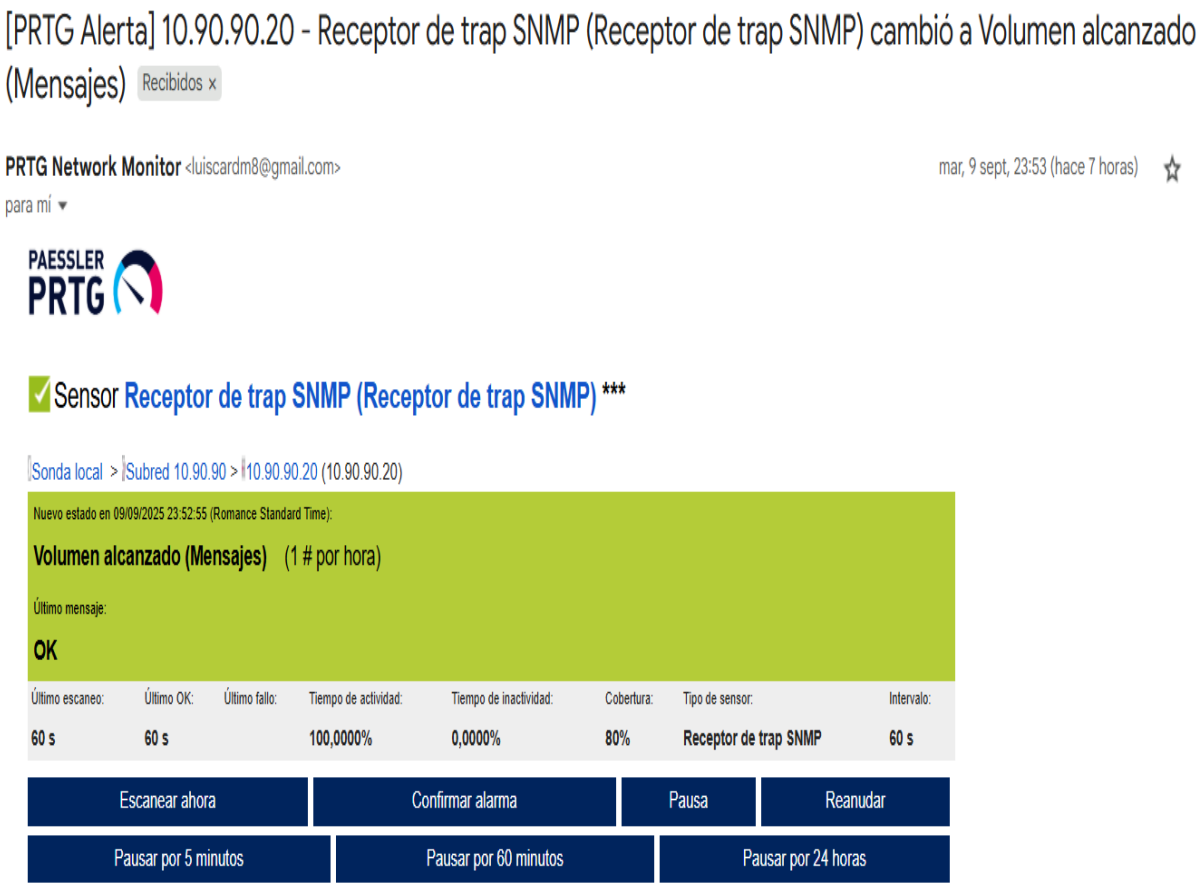


*Figura 5.25: Correo electrónico notificando de la Trap.*

Es importante señalar que la configuración detallada del servidor SMTP y de los desencadenadores de notificación se ha documentado en el Anexo V, ambos necesarios para que el PRTG pueda mandar un correo electrónico cuando se reciba una trap.

# Capítulo 6. Conclusión y líneas futuras

## 6.1. Conclusiones

Como se comentó al principio de esta memoria, el objetivo del proyecto era diseñar, implementar y validar una arquitectura basada en el protocolo SNMP sobre una plataforma Arduino, el cual tendrá un sistema de monitorización y visualización de datos en Grafana.

En primer lugar, se ha demostrado que un dispositivo como Arduino UNO, junto con un Ethernet Shield, puede configurarse como un agente SNMP plenamente funcional, capaz de gestionar variables, responder a operaciones SNMP como GET y SET validado mediante el gestor SNMP ManageEngine MibBrowser, así como generador de traps . El uso de Wireshark resultó fundamental, al permitir observar el intercambio de mensajes a nivel de red y confirmar el correcto funcionamiento del protocolo.

Durante el desarrollo del proyecto se ha utilizado la librería *Agentuino* como base para la implementación del agente SNMP en la plataforma Arduino. No obstante, esta librería presentaba importantes limitaciones derivadas de su falta de mantenimiento y actualización, entre ellas respuestas incompletas en determinadas operaciones y restricciones en la definición de OIDs personalizados. Con el fin de garantizar la estabilidad y el correcto funcionamiento del sistema, fue necesario llevar a cabo un proceso de depuración y mejora del código fuente, corrigiendo dichos errores e introduciendo adaptaciones específicas al hardware empleado. Estas modificaciones constituyen una de las principales contribuciones del trabajo, al permitir disponer de un agente SNMP plenamente operativo y reutilizable en proyectos similares que requieran integrar dispositivos de bajo coste en arquitecturas de monitorización basadas en protocolos de gestión de red estándar.

Adicionalmente, Para poder tener un sistema de monitorización más avanzado, se integró Telegraf, InfluxDB y Grafana, el cual recopilaba datos de forma periódica, los almacenaba en una base datos de series temporales y los representaba en dashboard dinámicos. Grafana resulto ser una herramienta versátil, no solo para la visualización de métricas, sino también para la interacción directa con el sistema. Prueba de ello fue la integración de una API Flask desarrollada en Python, que habilitó el control remoto del LED desde el propio panel de monitorización.

Finalmente, se validó la integración con el gestor PRTG Network Monitor, configurado como receptor de traps. Cada vez que el agente superaba un umbral establecido y generaba una trap, el sistema fue capaz de enviar un correo electrónico al administrador, extendiendo así las funcionalidades hacia un entorno de gestión más completo y cercano a posibles escenarios reales.

Este proyecto me ha permitido profundizar en tecnologías con las que apenas había trabajado, como el desarrollo de APIs en Python o la visualización con Grafana. La progresiva evolución, desde la programación del Agente SNMP en Arduino hasta la configuración del gestor PRTG Network Monitor para enviar un correo electrónico cuando recibe una trap, ha sido una experiencia especialmente gratificante.

## 6.2. Líneas Futuras

En este apartado se van a mostrar las posibles mejoras que se pueden realizar en un futuro.

En primer lugar, sería interesante ampliar el número de sensores y actuadores gestionados por el agente SNMP incorporando así nuevos dispositivos que aporten nuevas métricas ambientales o permitan un control más avanzado, y que no sea un simple sensor DHT11 y un diodo LED.

Adicionalmente, se podría diseñar una aplicación web o móvil que actúe como gestor SNMP, centralizando en una misma plataforma todas las funciones de monitorización: realización de consultas, modificación de parámetros, visualización y gestión de traps. De este modo, se eliminaría la dependencia de gestores externos y se ofrecería al usuario final un entorno unificado que, además, podría incorporar el envío de correos electrónicos.

Finalmente, otra opción a considerar sería la migración a una versión más avanzada del protocolo SNMP, concretamente SNMPv3, lo que permitiría dotar al sistema de mayor seguridad, incorporando autenticación y cifrado en las comunicaciones.

# BIBLIOGRAFÍA


[1] Site24x7, "¿Qué es SNMP?", site24x7.com, [Online]. Available: https://www.site24x7.com/es/network/what-is-snmp.html [Accessed: 9-sep-2025].

[2] "Simple Network Management Protocol". Departamento de Sistemas Telemáticos y Computación (GSyC), Universidad Rey Juan Carlos, diciembre 2013. [Online]. Available: https://gsyc.urjc.es/~mortuno/lagrs/07-snmp.pdf

[3] ManageEngine, "Free SNMP MIB-Browser", manageengine.com, [Online]. Available: https://www.manageengine.com/products/mibbrowser-free-tool/ [Accessed: 9-sep-2025].

[4] Paessler, "Monitoring via SNMP", paessler.com, [Online]. Available: https://www.paessler.com/manuals/prtg/monitoring_via_snmp [Accessed: 9-sep-2025].

[5] IBM, "Management information base (MIB) files", ibm.com, [Online]. Available: https://www.ibm.com/docs/es/ts4500-tape-library?topic=library-management-information-base-mib-files [Accessed: 9-sep-2025].

[6] Tecnocrática, "SNMP, MIBS y OID", tecnocratica.net, [Online]. Available: https://tecnocratica.net/wikicratica/books/monitorizacion-y-snmp/page/snmp-mibs-y-oid [Accessed: 9-sep-2025].

[7] J. Á. Irastorza Teja. "Gestión SNMP v1, v2, v3". Apuntes de la Asignatura de Gestión y Operación de Redes, Universidad de Cantabria, 2020.

[8] J. Á. Irastorza Teja. "Abstract Syntax Notation: ASN.1.". Apuntes de la Asignatura de Gestión y Operación de Redes, Universidad de Cantabria, 2020.

[9] AZ-Delivery, "Sensor de temperatura y humedad DHT11 KY-015", az-delivery.de, [Online]. Available: https://www.az-delivery.de/es/products/dht-11-temperatursensor-modul [Accessed: 9-sep-2025].

[10] OcioDual, "Kit 50 Diodos LED 5mm Rojo Azul Verde Amarillo Blanco para Robótica Electrónica", ociodual.com, [Online]. Available: https://ociodual.com/products/kit-diodos-azul-verde-amarilo-rojo-blanco-robotica-electronica [Accessed: 9-sep-2025].

[11] VelascoStore, "Resistencia 220 Ω", velascostore.com, [Online]. Available: https://velascostore.com/resistencias/3002-resistencia-carbon.html [Accessed: 9-sep-2025].

[12] Newark – Multicomp PRO, "Carbon Film Resistor 220 Ohm 250 mW (MCF-0.25W-220R)", newark.com, [Online]. Available: https://www.newark.com/multicomp-pro/mcf-0-25w-220r/carbon-film-resistor-220-ohm-250mw/dp/38K0351 [Accessed: 9-sep-2025].

[13] Arduino, “Arduino Uno Rev3”, arduino.cc, [Online]. Available: https://store.arduino.cc/products/arduino-uno-rev3/ [Accessed: 9-sep-2025].

[14] Arduino, “Arduino Ethernet Shield 2”, arduino.cc, [Online]. Available: https://store.arduino.cc/products/arduino-ethernet-shield-2?srsltid=AfmBOor3Btt3Gcj6sBhXYvXiUoJ1bAu8STuR5BTI4CzINaQIS7vt9VRZ [Accessed: 9-sep-2025].

[15] InfluxData, “Key Concepts”, influxdata.com, [Online]. Available: https://docs.influxdata.com/influxdb/v1/concepts/key_concepts/ [Accessed: 9-sep-2025].

[16] InfluxData, “Telegraf Best Practices: SNMP Plugin”, influxdata.com, [Online]. Available: https://www.influxdata.com/blog/telegraf-best-practices-snmp-plugin/ [Accessed: 9-sep-2025].

[17] Grafana Labs, “Configure InfluxDB Data Source”, grafana.com, [Online]. Available: https://grafana.com/docs/grafana/latest/datasources/influxdb/configure-influxdb-data-source/ [Accessed: 9-sep-2025].

[18] Wireshark, “About Wireshark”, wireshark.org, [Online]. Available: https://www.wireshark.org/about.html [Accessed: 9-sep-2025].

[19] Paessler, “SNMP Trap Receiver Sensor”, paessler.com, [Online]. Available: https://www.paessler.com/manuals/prtg/snmp_trap_receiver_sensor [Accessed: 9-sep-2025].

[20] Paessler, “Notification Templates”, paessler.com, [Online]. Available: https://www.paessler.com/manuals/prtg/notification_templates#email [Accessed: 9-sep-2025].

[21] V1cenZ, “Arduino-Advanced-SNMP-Project-Agentuino-”, GitHub repository, [Online]. Available: https://github.com/V1cenZ/Arduino-Advanced-SNMP-Project-Agentuino- [Accessed: 9-sep-2025].

# ANEXO I. Características técnicas de los componentes

En este anexo se presentan las especificaciones técnicas de los componentes utilizados en el desarrollo del sistema.

A continuación, se muestran en detalle las características del sensor, LED, resistencia y módulos electrónicos empleados, organizados en tablas para facilitar su consulta.

Tabla A.1:Características técnicas del sensor DHT11. [9]

| Parámetro | Valor |
|---|---|
| Tensión de alimentación | 3–5 V |
| Corriente máxima de alimentación | 2.5 mA |
| Rango de medición de humedad | 20 % – 80 % (±5 %) |
| Rango de medición de temperatura | 0 °C – 50 °C (±2 °C) |
| Frecuencia de muestreo | < 1 Hz (una lectura por segundo) |
| Dimensiones del sensor | 15.5 mm x 12 mm x 5.5 mm |

Tabla A.2: Características técnicas del Diodo LED. [10]

| Parámetro | Valor |
|---|---|
| Tipo | LED verde 5 mm |
| Tensión directa (Vf) | ~3.0 V |
| Corriente típica | 20 mA |
| Polaridad | Ánodo al pin de control, cátodo a GND |
| Función en el sistema | Actuador visual SNMP (ON/OFF) |

Tabla A.3: Características técnicas de la resistencia limitadora de 220 Ω. [12]

| Parámetro | Valor |
|---|---|
| Tipo | Resistencia fija |
| Valor nominal | 220 Ω |
| Tolerancia | ±5 % |
| Función en el sistema | Limitador de corriente para el LED |

Tabla A.4: Comparativa Arduino UNO vs Ethernet Shield [1] [2]

| Característica | Arduino Uno R3 | Arduino Ethernet Shield 2 |
|---|---|---|
| Microcontrolador | ATmega328P | Usa el Arduino base |
| Chip de red | - | Wiznet W5500 |
| Voltaje de funcionamiento | 5V | 5V (a través del Arduino) |
| Voltaje de entrada recomendado | 7–12V | — (alimentado por Arduino) |
| Pines digitales de E/S | 14 (6 PWM) | — (usa los pines del Arduino) |
| Entradas analógicas | 6 | - |
| Memoria Flash | 32 KB | - |
| Frecuencia de reloj | 16 MHz | - |
| Comunicación Ethernet | No | Sí (10/100 Mbps, puerto RJ45) |
| Almacenamiento externo | No | Sí (ranura para tarjeta microSD) |
| Interfaz de comunicación | USB (tipo B) | SPI (comunicación con el Arduino) |
| Compatibilidad | - | Compatible con Arduino Uno, Mega, etc |

Tabla A.5: Características del D-Link DGS-1100-08PV2.

| **Característica** | **Switch DGS-1100-08PV2** |
|---|---|
| Puertos | 8x10/100/1000 Mbps |
| Tipo | Smart Managed (semi-gestionable) |
| Alimentación PoE | 802.3af (hasta 30W por puerto, 64W total) |
| Gestión | Interfaz web, D-Link Network Assistant, SNMP v1/v2c |
| Funciones Avanzadas | VLAN 802.1Q, QoS, diagnóstico de cableado, control de ancho de banda |
| Seguridad | Loopback detection, control de tormentas de broadcast |
| Dimensiones | 208 x 126 x 44 mm |

# ANEXO II: Código ASN.1 de la MIBSENSORES.mib

A continuación, se muestra el código completo de la MIB “MIBSENSORES.mib” definida en ASN.1, que describe la jerarquía, tablas y objetos de gestión diseñados para el prototipo Arduino. Este archivo MIB es el que cargamos también en los distintos gestores SNMP.

```
MIBSENSORES DEFINITIONS ::= BEGIN

IMPORTS
   MODULE-IDENTITY, enterprises
   FROM RFC1155-SMI
   OBJECT-TYPE
   FROM RFC-1212
   IpAddress
   FROM SNMPv2-SMI;

arduinoMIB MODULE-IDENTITY
   LAST-UPDATED "202502090000Z"
   ORGANIZATION "Gestion de Redes"
   CONTACT-INFO "luiscardm8@gmail.com"
   DESCRIPTION
         " Proyecto Trabajo Fin de Grado
         "
   ::= { enterprises 36582 }



informacionDeGestion  OBJECT IDENTIFIER ::= { arduinoMIB 1 }
valoresLeidosSensores OBJECT IDENTIFIER ::= { arduinoMIB 2 }
eventosTraps        OBJECT IDENTIFIER ::= { arduinoMIB 3 }


-- Definicion del string utilizado en toda la MIB
DisplayString ::=
            OCTET STRING (SIZE (0..255))

identificador OBJECT-TYPE
   SYNTAX  DisplayString
   ACCESS  read-write
   STATUS  mandatory
   DESCRIPTION
         "
           Numero identificador del prototipo arduino gestionado
         "
   ::= { informacionDeGestion 1 }
```

```
localizacion OBJECT-TYPE
   SYNTAX  DisplayString
   ACCESS  read-write
   STATUS  mandatory
   DESCRIPTION
         "
           Localizacion del prototipo arduino
         "
   ::= { informacionDeGestion 2 }

direccionIP OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
         "
           Direccion IP asignada al arduino
         "
   ::= { informacionDeGestion 3 }

tiempoEncendido OBJECT-TYPE
   SYNTAX  TimeTicks
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
         "
           Tiempo encendido del arduino
         "
   ::= { informacionDeGestion 4 }
encargadoDeGestion OBJECT-TYPE
   SYNTAX  DisplayString
   ACCESS  read-write
   STATUS  mandatory
   DESCRIPTION
         "
           Nombre de la persona encargada de gestionar la placa
         "
   ::= { informacionDeGestion 5 }

correoContacto OBJECT-TYPE
   SYNTAX  DisplayString
   ACCESS  read-write
   STATUS  mandatory
   DESCRIPTION
         "
           Correo electrónico del encargado del proyecto
         "
```

```
    ::= { informacionDeGestion 6 }

--Comenzamos creando la tabla para leer o modicar la inforamcion que obtenemos del
sensor DHT11 que actua como sensor de temperatura y de humedad

 arduinoTable OBJECT-TYPE
    SYNTAX  SEQUENCE OF Filas
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
            " Tabla que contiene la información de gestión que implementa nuestro dispositivo
Arduino "
    ::= { valoresLeidosSensores 1 }

 arduinoEntry OBJECT-TYPE
    SYNTAX  Filas
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
            " Filas de la tabla"

    INDEX  { indice }
    ::= { arduinoTable 1 }

 Filas::=
    SEQUENCE {
       indice
          INTEGER,
       descrepSensor
          DisplayString,
       valorLeido
          INTEGER,
       tiempoMuestreo
         INTEGER,
          instanteLectura
         TimeTicks
    }

 indice OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            " Indice de la fila "
    ::= { arduinoEntry 1 }

 descrepSensor OBJECT-TYPE
    SYNTAX  DisplayString
```

```
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
          " Nombre del sensor a leer "
    ::= { arduinoEntry 2 }

 valorLeido OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
          " Valor leido del sensor "
    ::= { arduinoEntry 3 }

 tiempoMuestreo OBJECT-TYPE
   SYNTAX  INTEGER
   ACCESS  read-write
   STATUS  mandatory
   DESCRIPTION
         "
           Valor en segundos de el tiempo en muestrear los parámetros obtenidos por
arduino.
         "
   ::= { arduinoEntry 4 }

 instanteLectura OBJECT-TYPE
    SYNTAX  TimeTicks
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
          "
            Valor de la variable sysUpTime en el instante de lectura del ultimo valor leido
          "
    ::= { arduinoEntry 5 }



-- A partir de aquí comienza el grupo de eventos, que proporciona una tabla para poder
establecer hasta 5 eventos distintos


   eventosTable OBJECT-TYPE
    SYNTAX  SEQUENCE OF FilasEventos
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
          " Tabla que contiene la eventos que implementa nuestro dispositivo Arduino "
    ::= { eventosTraps 1 }
```

```
eventosEntry OBJECT-TYPE
    SYNTAX  FilasEventos
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
            " Filas de la tabla"

    INDEX  { indiceEv }
    ::= { eventosTable 1 }

FilasEventos::=
    SEQUENCE {
        indiceEv
            INTEGER,
        ifIndexSensor
            INTEGER,
        descrEvento
            DisplayString,
        umbralSup
            INTEGER,
        umbralInf
            INTEGER,
        ipReceptorTrap
            IpAddress

    }

indiceEv OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            " Número de la variable arduino, esta es la fila indice "
    ::= { eventosEntry 1 }

ifIndexSensor OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            " Coincide con la columna indice de la tabla de valoresLeidos, para establecer que
sensor será al que se le aplique el evento "
    ::= { eventosEntry 2 }


descrEvento OBJECT-TYPE
    SYNTAX  DisplayString
```

```
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            " Descripcion personalizada del evento  "
    ::= { eventosEntry 3 }

 umbralSup OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            " Valor leido del sensor "
    ::= { eventosEntry 4 }

 umbralInf OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "
              Valor de la variable sysUpTime en el instante de lectura del ultimo valor leido
            "
    ::= { eventosEntry 5 }
 ipReceptorTrap OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-write
   STATUS  mandatory
   DESCRIPTION
           "
             Valor en segundos de el tiempo en muestrear los parámetros obtenidos por
arduino.
           "
   ::= { eventosEntry 6 }

END
```

# ANEXO III. Código Agente_SNMP.ino

Este código se ejecuta en la placa Arduino para que actúe como un agente SNMP, capaz de escuchar y recibir comandos SNMP gracias al uso de la librería **Agentuino**.

```
#include <DHT.h>
#include <DHT_U.h>
#include "Streaming.h"
#include <Ethernet.h>
#include "Agentuino.h"
#include "MIB.h"
#include "Variable.h"
//#include <CapacitiveSensor.h>
#include <string.h>
//#define DHTPIN 2        // Pin donde está conectado el DHT11
#define Type DHT11   // Tipo de sensor (DHT11 o DHT22)
int dhtPin=2;

int ledPin=7;

DHT HT(dhtPin, Type);

int dt (500);

int numAccesos;

int numAccesosAnteriorMuestreo;


char* strings[4]; // Array de punteros a caracteres (strings)
int num_strings = 0; // Variable para mantener el número actual de strings en la lista
String readString;
// Crear un servidor en el puerto 80
EthernetServer server(80);
IPAddress address;
//VARIABLES PARA MEDIR LOS RETRASOS
static int cont1;
static int cont2;
static int cont3;

int x=100;

static int muestreo1Ant=10;// es decir 10 segundos
static int muestreo2Ant=10;
static int muestreo3Ant=10;
```

```
static int activada=0;
static int activada2=0;

//VARIABLE PARA LA FUNCION DE EVENTOS
static int32_t* sensores[2];//PUNTERO PARA LOS SENSORES

//CONFIGURACIÓN IP Y MAC DEL ARDUINO
static byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xEF};
//static byte ipArduino[] = {192, 168, 1, 89}; //MI IP DE ARDUINO
static byte ipArduino[] = {10, 90, 90, 20}; //MI IP DE ARDUINO
static byte gateway[] = {10, 90, 90, 1};
static byte subnet[] = {255, 0, 0, 0};

void setup() {
   Serial.begin(9600);
   HT.begin();
   pinMode(ledPin, OUTPUT);
   digitalWrite(ledPin, LOW); // LED comienza apagado

   Serial.println("Arduino Iniciado, logs preparados para ser servidos :)");


   addString("Trap generado por: Sensor de Temperatura");
   addString("Trap generado por: Sensor de Humedad");
   addString("Trap generado por: Led");

   sensores[0] = &sensortemp;
   sensores[1] = &sensorhum;
   sensores[2] = &valorled;

   server.begin();

   Ethernet.begin(mac,ipArduino,gateway,subnet);

   IPAddress address = Ethernet.localIP();// PARA OBTENER MI DIRECCION IP
   for (uint8_t i=0;i<=4;i++) {
      my_IP_address[i] = address[i];
      if(i!=4){
      Serial.print(my_IP_address[i]); //ME PINTA MI DIRECCION IP
      }
      Serial.print(".");
   }
   Serial.println("");


   cont1=1000; //SON 1000 MS QUE SERAN 10 SEG
   delay(100);
   cont2=1000;
```

```
    delay(100);
    cont3=1000;
    delay(100);

    delay(100);

    api_status = Agentuino.begin();

    if (api_status == SNMP_API_STAT_SUCCESS) {

        Agentuino.onPduReceive(pduReceived);

        delay(10);

        return;
    }

    delay(10);

}

void loop() {
    // LECTURA DE PARÁMETROS
    Agentuino.listen();

   // FUNCIONES DE MUESTREO/LECTURA DE VARIABLES

    actualiza1();
    actualiza2();
    actualiza3();

    //COMPROBAMOS LOS POSIBLES TRAPS
    evento1();
    evento2();

 // VARIABLE SYSUPTIME, LA ACTUALIZAMOS!
    if (millis() - prevMillis > 1000) {

        prevMillis += 1000;

        locUpTime += 100 ;

       // CONTADOR DE LOS SENSORES!
        cont1=cont1-100;
        cont2=cont2-100;
        cont3=cont3-100;
    }
}
```

```
void actualiza1(void)
{
   if(tiempoMuestreo1!=muestreo1Ant){
     muestreo1Ant=tiempoMuestreo1;
     cont1 = tiempoMuestreo1*100;
   }
   if(cont1==0){
     //SE LEEE EL VALOR Y SE ACTUALIZA EL TIEMPO
     sensortemp = HT.readTemperature();
     Serial.print("Valor Sensor Temperatura: ");
     Serial.println(sensortemp);
     if (isnan(sensortemp)) {
       Serial.println("Error al leer el sensor DHT11");
       //return;
     }
     instanteLectura1 = locUpTime;
     cont1=tiempoMuestreo1*100;

   }
}

void actualiza2(void)
{
   if(tiempoMuestreo2!=muestreo2Ant){
     muestreo2Ant=tiempoMuestreo2;
     cont2 = tiempoMuestreo2*100;
   }
   if(cont2==0){
     sensorhum = HT.readHumidity();
     Serial.print("Valor Sensor Humedad: ");
     Serial.println(sensorhum);
     instanteLectura2 = locUpTime;
     cont2=tiempoMuestreo2*100;
   }
}

void actualiza3(void)
{
   if(tiempoMuestreo3!=muestreo3Ant){
     muestreo3Ant=tiempoMuestreo3;
     cont3 = tiempoMuestreo3*100;
   }
   if (cont3 == 0) {

       Serial.print("Estado actual del LED: ");
       Serial.println(valorled);
       instanteLectura3 = locUpTime;
```

```
        cont3 = tiempoMuestreo3 * 100;
    }
}

void evento1(void)
{
    if(Sind1!=0){

       if(Sind1==1 || Sind1==2){
       if((*sensores[Sind1-1]>uS1)){//*sensores[Sind1-1] es una lista dinamica que me va
racorriendo los indices de mi tabla eventoTraps
         if(activada==0){

            char str1[10]= "";
            char strcad[25]= "";
            sprintf(str1, "%d", *sensores[Sind1-1]);

            strcat(strcad, d1); // Concatenar cadena2 a cadena1
            strcat(strcad, " Valor: "); // Concatenar cadena2 a cadena1
            strcat(strcad, str1); // Concatenar cadena2 a cadena1

            Agentuino.Trap(strcad , ip1, locUpTime,
"1.3.6.1.4.1.36582.1.1.1.4.1","1.3.6.1.4.1.36582.1.1.1.4.1");
            activada=1;

          // GENERADOR DE LOGS
            unsigned long seconds = locUpTime / 100;
            unsigned long minutes = seconds / 60;
            unsigned long hours = minutes / 60;
            seconds %= 60;
            minutes %= 60;
            Serial.println(strings[Sind1-1]);
            Serial.print("En el instante: ");
            Serial.print(hours);
            Serial.print("horas, ");
            Serial.print(minutes);
            Serial.print("minutos, ");
            Serial.print(seconds);
            Serial.println("segundos");
            Serial.println(" ");


            //delay(1000);
          // locUpTime = locUpTime + 100;
         }


       }else{
```

```
        activada=0;
        }
       }
    }
}


void evento2(void)
{
   if(Sind2!=0){

     if(Sind2==1 || Sind2==2){

      if((*sensores[Sind2-1]>uS2)||(*sensores[Sind2-1]<uI2)){ //MAYOR QUE EL UMBRAL
SUPERIOR O MENOR QUE EL UMBRAL INFERIOR
        if(activada2==0){

         char str2[10]= "";
         char strcad2[25]= "";
         sprintf(str2, "%d", *sensores[Sind2-1]);

         strcat(strcad2, d2); // Concatenar cadena2 a cadena1
         strcat(strcad2, " Valor: "); // Concatenar cadena2 a cadena1
         strcat(strcad2, str2); // Concatenar cadena2 a cadena1

         Agentuino.Trap(strcad2, ip2, locUpTime,
"1.3.6.1.4.1.36582.1.1.1.4.1","1.3.6.1.4.1.36582.1.1.1.4.1");
         activada2=1;


         // GENERADOR DE LOGS
         unsigned long seconds = locUpTime / 100;
         unsigned long minutes = seconds / 60;
         unsigned long hours = minutes / 60;
         seconds %= 60;
         minutes %= 60;
         Serial.println(strings[Sind2-1]);
         Serial.print("En el instante: ");
         Serial.print(hours);
         Serial.print("horas, ");
         Serial.print(minutes);
         Serial.print("minutos, ");
         Serial.print(seconds);
         Serial.print("segundos");
         Serial.println(" ");


          //delay(1000);
```

```
          // locUpTime = locUpTime + 100;
        }


       }else{
        activada2=0;
        }

    }
  }
}

void addString(const char* str) {
    if (num_strings < 10) {
        // Asignar memoria para el nuevo string y copiar el contenido
        strings[num_strings++] = strdup(str);
    }
}
```

# ANEXO IV. set_led.py

Esta API, desarrollada en Python, recibe la petición POST y traduce la orden en una operación SET utilizando la librería pysnmp, enviándola posteriormente al Arduino UNO para poder tener el control del estado del LED (ENCENDIDO “1” / APAGADO “0”).

```
from flask import Flask, request, jsonify
from flask_cors import CORS
from pysnmp.hlapi import (
    SnmpEngine,
    CommunityData,
    UdpTransportTarget,
    ContextData,
    ObjectType,
    ObjectIdentity,
    Integer,
    setCmd
)

app = Flask(__name__)
CORS(app)  # Permite peticiones desde Grafana u otros orígenes

# Configuración del agente SNMP (tu Arduino)
#
ARDUINO_IP = "10.90.90.20" # <-- IP del Arduino
OID_LED = "1.3.6.1.4.1.36582.2.1.1.3.3" # <-- Debe coincidir con tu OID en Arduino
COMMUNITY = "public"

def enviar_snmp_set(valor):
    print(f"\n[INFO] Enviando SNMP SET: valor = {valor}")
    errorIndication, errorStatus, errorIndex, varBinds = next(
        setCmd(
            SnmpEngine(),
            CommunityData(COMMUNITY, mpModel=0),  # SNMPv1
            UdpTransportTarget((ARDUINO_IP, 161)),
            ContextData(),
            ObjectType(ObjectIdentity(OID_LED), Integer(valor))
        )
    )

    if errorIndication:
        print("[ERROR] SNMP Error:", errorIndication)
        return False
    elif errorStatus:
        print(f"[ERROR] SNMP Set fallido: {errorStatus.prettyPrint()} at index {errorIndex}")
        return False
```

```
    else:
        print("[SUCCESS] SNMP SET enviado correctamente. Respuesta del agente:")
        for varBind in varBinds:
            print("   →", varBind)
        return True

@app.route('/led', methods=['POST']) #ruta /led que acepta el método POST
def controlar_led(): #actúa como puerta de entrada (API)
    data = request.get_json()
    valor = int(data.get("valor", 0))  # 0 o 1
    print(f"[POST] Petición recibida desde frontend: valor = {valor}")
    exito = enviar_snmp_set(valor)#lanza el set al arduino
    return jsonify({"ok": exito, "estado": valor})

if __name__ == '__main__':
    print("[FLASK] Servidor iniciado en http://0.0.0.0:5000")
    app.run(host="0.0.0.0", port=5000)
```

# ANEXO V. Configuración en PRTG Network Monitor para envío correo electrónico

Para que PRTG pueda enviar correos electrónicos cuando se recibe una trap, es necesario configurar dos aspectos clave:

**El envío SMTP** (Figura A.1): en este apartado se especifica el servidor de correo saliente, en este caso smtp.gmail.com sobre el puerto 587 con autenticación estándar. Asimismo, se introduce la cuenta de correo electrónico del remitente junto con sus credenciales y se habilita la seguridad TLS. De esta forma, PRTG tiene la capacidad de conectarse al servicio de correo y remitir notificaciones.

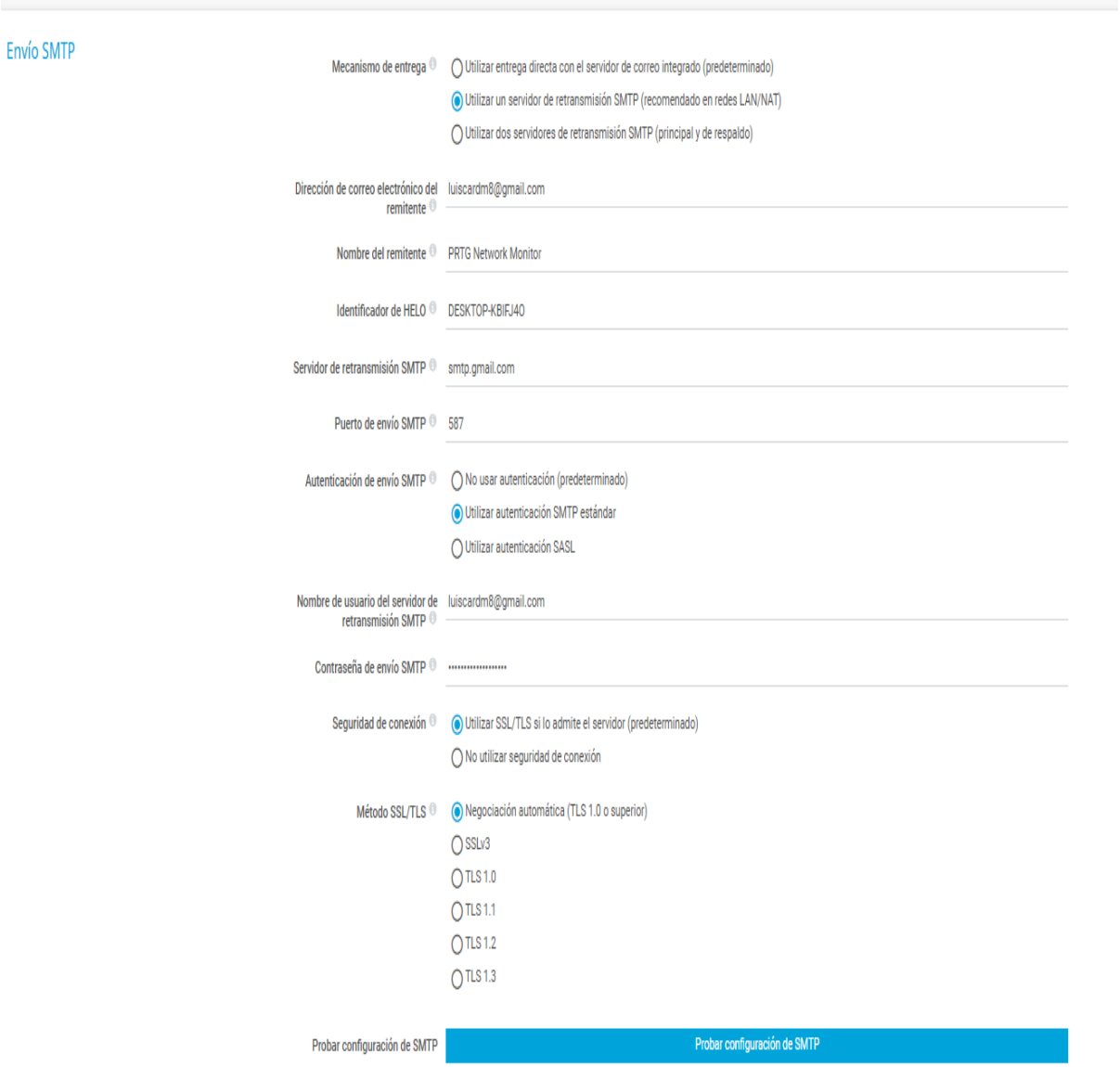


*Figura A.1: Servidor de correo saliente.*

**Los desencadenadores de notificaciones** (Figura A.2): una vez que el sensor Receptor de trap SNMP recibe un mensaje, los desencadenadores definen la acción que debe ejecutarse. En este caso, se configuró que cuando el canal Mensajes alcanza al menos un valor, se envíe un correo electrónico al administrador.

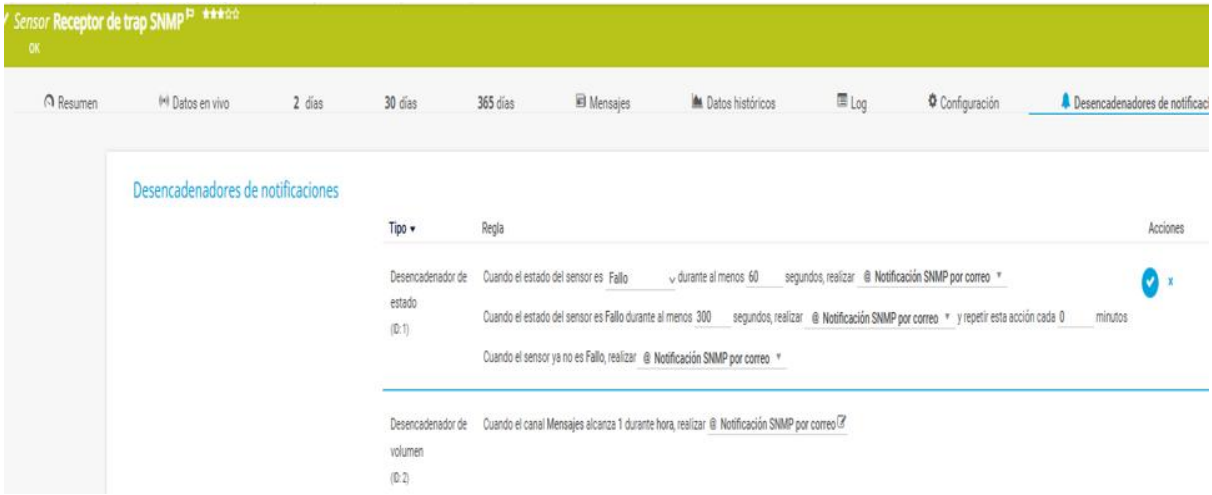


*Figura A.2: Desencadenadores de notificaciones.*

De esta manera, cuando el gestor PRTG recibe una trap por parte del agente, este envía un correo electrónico al administrador.