Prototipo de Servicio IoT con SRD y Recursos de Google

Wilder Arley Carrillo Pinilla y Diego Andrés Campo Saavedra

Trabajo de Grado para Optar al Título de Ingeniero Electrónico

Director

Homero Ortega Boada

Doctor en Ciencias de la Ingeniería, Radiocomunicaciones

Universidad Industrial de Santander

Facultad de Ingeniería Físicomecánicas

Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones (E3T)

Ingeniería Electrónica

Bucaramanga

2025

Dedicatoria

Wilder Arley Carrillo Pinilla

A Dios, por la vida, la fortaleza y la guía en cada decisión. A mi madre Yaneth Azucena Pinilla Sánchez y a mi padre Ovidio Carrillo González, por su apoyo económico y emocional, por sus ánimos constantes y sus oraciones. “Encomienda a Jehová tus obras, y tus pensamientos serán afirmados.” (Proverbios 16:3, RVR1960).

Diego Andrés Campo Saavedra

A mi madre Nubia Saavedra Rangel, mi padre Cayetano Campo Silva, a mi hermano Juan Camilo Campo Saavedra y familia, por creer en mí, acompañarme y sostenerme hasta convertir este esfuerzo en un logro. Sin ellos no habría sido posible.

Agradecimientos

Gracias a Dios por darme calma cuando fue difícil y la fuerza para seguir adelante. Gracias al Dr. Homero Ortega Boada (Doctor en Ciencias de la Ingeniería, radiocomunicaciones), director de este trabajo, por su guía rigurosa, sus preguntas oportunas y el ejemplo profesional que elevó la calidad del proyecto. Gracias al profesor Efrén Darío Acevedo Cárdenas por sus aportes importantes que fortalecieron el desarrollo de este proyecto. Gracias a mi familia y amistades por su apoyo constante y su confianza.

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Lista de Apéndices

Los apéndices están adjuntos y puede visualizarlos en la base de datos de la biblioteca UIS

Apéndice A. Conceptos Previos.

Apéndice B. Descripción de bloques en GNU Radio de la arquitectura STERT1.

Apéndice C. WEB Service.

Apéndice D. Web App.

Apéndice E. Descripción de bloques en GNU Radio de la arquitectura STERT2.

Apéndice F. Creación de un contenedor en Microsoft Azure para Node-RED.

Apéndice G. Procesamiento de datos en Node-RED.

Apéndice H. Configuración de nodos Tiempo UDP y Tiempo TCP.

Apéndice I. Configuración para generar gráficas.

Apéndice J. Panel de navegación.

Apéndice K. Análisis de resultados.

Resumen

Título: Prototipo de Servicio IoT con SRD y Recursos de Google [*]

Autor: Wilder Arley Carrillo Pinilla, Diego Andrés Campo Saavedra [**]

Palabras Clave: SDR, Espectro Radioeléctrico, Tiempo Real, GNU Radio, Node-RED, Microsoft Azure, Web Service, IoT.

Descripción: Este proyecto presenta una solución innovadora para el monitoreo remoto del espectro en tiempo real, abordando un desafío clave en el laboratorio de radioastronomía de la Universidad Industrial de Santander (UIS), ubicado en un entorno remoto. A diferencia de las soluciones tradicionales basadas en capa de aplicación, se implementa una solución a nivel de capa de transporte, utilizando los protocolos UDP y TCP, lo que permite una visualización en tiempo real del espectro durante el control remoto de antenas.

Se implementaron los elementos clave para ajustar la velocidad de transmisión de manera similar a cómo se gestionan los flujos de datos en videoconferencias. De manera que el proyecto sienta las bases para futuras soluciones capaces de regular de manera dinámica la velocidad de transmisión de las ventanas espectrales en función del nivel de congestión del enlace de comunicación. Además, se destaca el uso de Node-RED como herramienta de visualización en línea, que facilita la integración y análisis de los datos en tiempo real.

Finalmente, se compara esta solución con una basada en HTTP, demostrando ventajas en términos de latencia y capacidad de transmisión, y destacando su aplicabilidad en laboratorios remotos y otros entornos educativos. Esta investigación contribuye significativamente al avance de la educación remota y la investigación en radioastronomía.

---

[*] Trabajo de Grado

[**] Facultad de Ingenierías Físico-mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: Homero Ortega Boada. Doctor en ciencias de la ingeniería, radiocomunicaciones

Abstract

Title: IoT Service Prototype with SRD and Google Resources[*]

Author: Wilder Arley Carrillo Pinilla, Diego Andrés Campo Saavedra[**]

Key Words: SDR, Radio Spectrum, Real Time, GNU Radio, Node-RED, Microsoft Azure, Web Service, IoT.

Description: This project presents an innovative solution for real-time remote spectrum monitoring, addressing a key challenge in the radio astronomy laboratory at the Universidad Industrial de Santander (UIS), located in a remote environment. Unlike traditional application-layer approaches, a transport-layer solution is implemented using UDP and TCP protocols, enabling real-time spectrum visualization during remote antenna control.

The essential components were implemented to adjust transmission speed in a manner similar to how data streams are managed in video conferencing. Thus, this project lays the groundwork for future solutions capable of dynamically regulating spectral window transmission rates based on communication-link congestion. Additionally, Node-RED is highlighted as an online visualization tool that facilitates real-time data integration and analysis.

Finally, this solution is compared with an HTTP-based approach, demonstrating advantages in terms of latency and transmission capacity, and underscoring its applicability in remote laboratories and other educational settings. This research makes a significant contribution to the advancement of remote education and radio astronomy research.

[*] Degree Work

[**]School of Physicomechanical Engineering. School of Electrical, Electronic and Telecommunications Engineering. Electronic Engineering. Director: Homero Ortega Boada. PhD in Engineering Sciences, Radio Comumunications.

Introducción

En un mundo donde la conectividad es el motor de la innovación, el espectro radioeléctrico se ha convertido en uno de los recursos más valiosos para las comunicaciones. Desde la transmisión de datos en redes móviles hasta el acceso a Internet en zonas remotas, el espectro es la infraestructura invisible que sustenta nuestras interacciones diarias. Sin embargo, el monitoreo del espectro, especialmente en tiempo real y a distancia, sigue siendo un desafío tecnológico importante, especialmente en entornos como el laboratorio de radioastronomía de la Universidad Industrial de Santander (UIS), donde se realizan investigaciones en ubicaciones geográficas de difícil acceso. Tradicionalmente, este tipo de análisis requería presencia física, equipos especializados y conocimientos avanzados, lo que limitaba su uso a entornos muy específicos. Sin embargo, con el avance de tecnologías como la radio definida por software (SDR) y las plataformas de computación en la nube, se abre la posibilidad de construir sistemas más accesibles, flexibles y escalables. Este trabajo de grado nace de esa posibilidad y de una necesidad concreta: dotar al Laboratorio de Comunicaciones de la Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones (E3T) de la Universidad Industrial de Santander, y al grupo de investigación RadioGIS, de una herramienta que permita visualizar y analizar el espectro radioeléctrico en tiempo real, desde cualquier lugar y sin depender de infraestructura física compleja. A lo largo de los años, con la alianza de los grupos CEMOS y RadioGIS se ha implementado un sistema de estudio de radio interferencias para la radioastronomía llamado CASIRI, el cual presenta importantes avances en la integración de sensores, conectividad y procesamiento en la nube, pero hasta ahora poco se ha avanzado en técnicas de monitoreo remoto del espectro. Este proyecto busca llenar ese vacío pero va más lejos ya que le apuesta al monitoreo remoto del espectro en tiempo real, proponiendo y comparando dos métodos para la transmisión remota de datos espectrales: uno

basado en la capa de aplicación del modelo TCP/IP, usando el protocolo HTTP y otra basada en la capa de transporte mediante el uso de los protocolos TCP y UDP. La solución también se integra con Node-RED desplegado en un contenedor en la nube. El objetivo general que guía esta investigación es el de desarrollar y evaluar comparativamente estos dos tipos de solución (la basada en la capa de aplicación y la basada en la capa de transporte), con el fin de identificar cuál de ellas representa mayores ventajas en función de las necesidades de la radioastronomía y de los laboratorios remotos. Este enfoque comparativo es clave porque permite no solo medir la eficiencia técnica de cada método, sino también considerar aspectos como la facilidad de implementación, la escalabilidad, el consumo de recursos y la experiencia de usuario. Para ello, se establecieron objetivos específicos como estudiar las capacidades del protocolo HTTP para conectar equipos SDR remotamente, para lo cual se implementó una solución de conectividad basada en Google Apps Script, pero también se enfocó en el protocolo TCP y UDP para lo cual se diseñó un sistema de transmisión de datos a nivel de la capa de transporte aprovechando recursos de GNU Radio y de Node-Red para luego proceder a estudios para identificar el campo de aplicación más adecuado para cada alternativa. La combinación de estos objetivos es garantizar una evaluación completa y equilibrada de las dos arquitecturas propuestas. Se logró implementar exitosamente ambas soluciones y se comprobó que, mientras el método basado en HTTP es ideal para registrar grandes volúmenes de datos con simplicidad y costos reducidos, la arquitectura basada en UDP o TCP junto con Node-RED permite una visualización más fluida y en tiempo real gracias a la baja latencia de los protocolos UDP y TCP. Se destaca además el desarrollo de bloques personalizados en GNU Radio para el tratamiento de las ventanas, el diseño de un sistema de banderas para evitar la distorsión de datos en la transmisión en ambos protocolos, y una interfaz visual interactiva construida con Plotly, accesible desde cualquier navegador. Estas contribuciones

no solo resuelven el problema planteado, sino que sientan las bases para el desarrollo de sistemas de monitoreo espectral más robustos, personalizables y orientados a la nube.

En el ámbito académico, facilita nuevas formas de enseñanza remota y prácticas pedagógicas con acceso a señales reales. En términos tecnológicos, demuestra que es posible construir soluciones IoT avanzadas con recursos accesibles y herramientas de código abierto. Desde una perspectiva social y científica, se abren las puertas a un monitoreo continuo del espectro en ubicaciones críticas o remotas, con implicaciones en áreas como la gestión del espectro, la radioastronomía, la vigilancia ambiental, o incluso la investigación geográfica. Esta propuesta, por tanto, no solo responde a un reto puntual, sino que plantea un modelo de solución replicable, escalable y adaptable a múltiples escenarios futuros.

1. Objetivos

1.1 Objetivo General

Desarrollar y evaluar comparativamente dos métodos para la transmisión remota del espectro en tiempo casi real, uno basado en el uso de UDP y TCP a través de Node-RED y otro mediante Web Service implementados con Google Apps Script (con servicios para el registro y consulta de datos en Google Sheets), con el fin de identificar el enfoque que proporcione una visualización representativa y de baja latencia, satisfaciendo las necesidades del laboratorio de comunicaciones de la E3T y del grupo RadioGIS.

1.2 Objetivos Específicos

El cumplimiento del objetivo general del trabajo de grado comprende:

Realizar un estudio de las capacidades de Google para conectar remotamente equipos SDR.

Implementar una solución de conectividad basada en Apps Script de Google.

Implementar una solución de conectividad a nivel de la capa de Transporte del modelo TCP/IP para un sistema SCADA o similar, utilizando los protocolos UDP y TCP a través de Node-RED.

Identificar para cada uno de los casos el campo de aplicación que pueden tener.

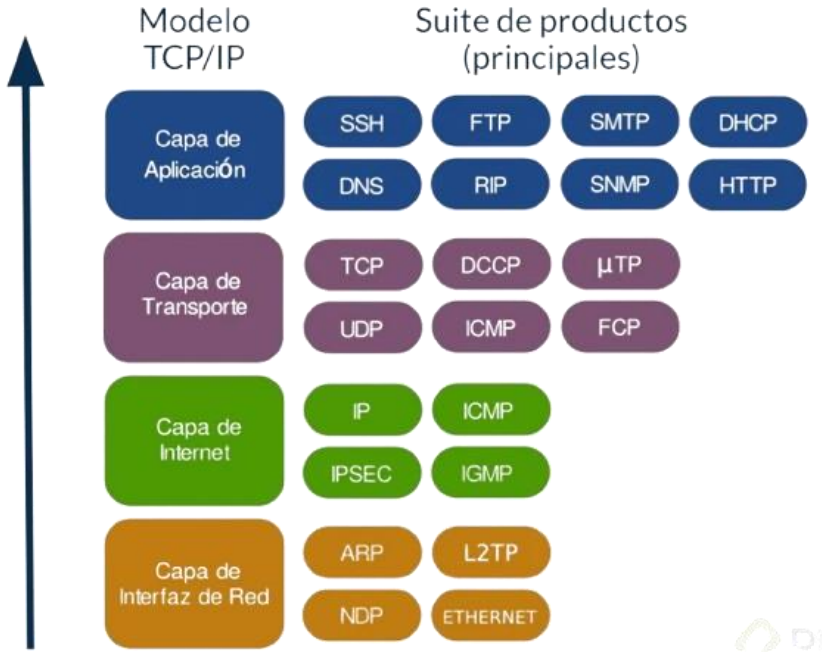
2. Conceptos Previos

El propósito de este capítulo es describir los conceptos básicos que ayudan al lector a entender en detalle los aspectos técnicos y tecnológicos de este trabajo; para comprender las comunicaciones digitales actuales es importante introducir el Modelo de Referencia TCP/IP, el cual estructura la transmisión de datos en cuatro niveles funcionales, Capa de Aplicación, Capa de

Transporte, Capa de Internet y Capa de Interfaz de Red (Bodnar, 2021) y asigna a cada capa una parte específica del proceso de comunicación, permitiendo que distintas tecnologías y protocolos operen de forma coordinada.

Figura 1.

Representa las capas del modelo TCP/IP junto a los protocolos más representativos de cada capa.



Nota. Este esquema representa las capas del modelo TCP/IP junto a los protocolos más comunes en cada una. Se destaca HTTP en la capa de aplicación y TCP/UDP en la capa de transporte, claves para este proyecto (Sacco, 2024).

Este proyecto se centra particularmente en las dos capas superiores:

2.1 La Capa de Transporte.

Donde se sitúan los protocolos TCP **(**Transmission Control Protocol**)** y UDP (User Datagram Protocol).

2.1.1 TCP

Es un protocolo orientado a conexión: establece una relación confiable entre emisor y receptor, garantizando la entrega ordenada y completa de los datos. Sin embargo, esto introduce algo de latencia debido a sus mecanismos de verificación y retransmisión.

2.1.2 UDP

Es un protocolo sin conexión: no realiza comprobaciones ni garantiza el orden de llegada de los datos, lo que lo hace mucho más rápido, pero menos confiable. Es útil en aplicaciones donde la inmediatez es prioritaria, como videoconferencias o sistemas de transmisión en tiempo real.

2.2 La Capa de Aplicación

Permite la interacción directa con el usuario final o con otros servicios. Aquí se encuentra el protocolo HTTP (Hypertext Transfer Protocol), base del tráfico web, que funciona bajo un modelo de solicitud-respuesta. HTTP utiliza TCP como transporte subyacente, lo que garantiza la entrega correcta de los datos, aunque con mayor latencia que UDP.

La radio definida por software (SDR), es un sistema de comunicación de radio que utiliza software para procesar diferentes señales. Tradicionalmente, las radios estaban compuestas por circuitos y componentes electrónicos específicos para cada función, lo que limitaba su flexibilidad y capacidad de adaptación. La SDR, en contraste, reemplaza muchas de estas funciones físicas por procesamiento digital realizado mediante software (programas), permitiendo configurar un único dispositivo para distintas aplicaciones con solo cambiar el software que ejecuta (de Villegas, 2024). La SDR se materializa mediante el hardware (circuitería) del HackRF One, un dispositivo de bajo costo que permite captar señales en un amplio rango de frecuencias, desde 1 MHz hasta 6 GHz. Este rango hace posible que se utilice para diferentes aplicaciones, desde radioaficionados hasta monitoreo de espectro profesional. Para procesar las señales capturadas por el HackRF One

(HackRF One, 2009-2023), se utiliza GNU Radio, una plataforma de código abierto basada en bloques de Python para diseñar y ejecutar en tiempo real el procesamiento de señales (What Is GNU Radio - GNU Radio, 2022), desde filtrado y transformadas de Fourier hasta modulaciones, permitiendo la conexión directa con el HackRF One para visualizar y analizar el espectro radioeléctrico de forma rápida (Características Del HackRF One, n.d.). Este trabajo forma parte del IoT **(**Internet de las Cosas**)**, donde dispositivos y sensores comparten datos en la nube para automatizar tareas y reducir la necesidad de supervisión humana. Gracias al IoT, tareas que antes exigían presencia física o revisiones constantes ahora se pueden automatizar y controlar a distancia (¿Qué Es El Internet De Las Cosas (IoT)?, n.d.). Un ejemplo de un sistema IoT es el sistema SCADA, el cual recoge y muestra en tiempo real variables de campo, almacena datos para análisis de tendencias, genera alarmas ante desviaciones y permite el control remoto de equipos desde una interfaz central (Qué Es Un Sistema SCADA, Cómo Funciona Y Para Qué Sirve, 2021).

Un ejemplo de implementación avanzada de SDR es la Estación CASIRI, desarrollada por los grupos CEMOS y RadioGIS de la Universidad Industrial de Santander (UIS) para el estudio de sitios con niveles de radiointerferencias adecuados para la radioastronomía. Inicialmente, CASIRI se implementó como un sistema completo, pero sin integrar aún sistemas SCADA. Con el tiempo, se desarrolló una solución basada en Ignition, centrada únicamente en la componente meteorológica. Más tarde, esta estación fue adaptada para utilizar Node-RED, lo que permitió su integración con plataformas de monitoreo remoto, mejorando su capacidad de análisis espectral y la visualización en tiempo real de los datos obtenidos (OpenJS, 2024). La Estación CASIRI, al igual que otros sistemas SDR, aprovecha las ventajas del procesamiento en la nube y la flexibilidad de la tecnología SDR para ofrecer una herramienta robusta en la radioastronomía.

Hoy en día existen múltiples plataformas de computación en la nube, cada una con un catálogo propio de servicios para almacenamiento, procesamiento y análisis de datos. Por ejemplo, Microsoft Azure ofrece Blob Storage para almacenamiento de objetos, Virtual Machines para instancias de cómputo y Azure Functions para funciones sin servidor; IBM Cloud incorpora Cloud Object Storage, Virtual Servers y Cloud Functions, entre otros (Rojas, 2024). En este caso, por motivos de interés del grupo RadioGIS, este proyecto integra capacidades de Google Apps Script y Google Sheets. En este contexto se introduce el concepto de Web Service, una pieza fundamental para la comunicación entre aplicaciones distribuidas. Un Web Service es una interfaz que permite a dos sistemas intercambiar datos a través de la red usando estándares como HTTP (Ballejos, 2024). En este proyecto, se desarrollaron Web Services para enviar y recibir datos espectrales utilizando Google Apps Script. Estos servicios pueden utilizar métodos HTTP, siendo los más comunes GET y POST. El método GET se usa para solicitar datos (por ejemplo, consultar una hoja de cálculo), mientras que el método POST se utiliza para enviar datos al servidor (por ejemplo, registrar una nueva medición del espectro) (Descripción General De Google Apps Script, 2024). La elección entre ambos depende del tipo de operación que se desea realizar y de la necesidad de mantener la seguridad e integridad de los datos transmitidos. Sobre este Web Services se construyó una WebApp, es decir, una aplicación web accesible desde cualquier navegador, que permite al usuario visualizar de forma dinámica y actualizada el espectro radioeléctrico. A diferencia de las aplicaciones de escritorio tradicionales, una WebApp se ejecuta completamente en el navegador del usuario y suele estar desarrollada con tecnologías como HTML, JavaScript y servicios de Backend como Google Apps Script (¿Qué Es Una WebApp?, n.d.).

Mientras los Webs Services operan en la capa de aplicación, los sockets actúan como el canal subyacente en la Capa de Transporte, encargado de establecer y gestionar las conexiones de

datos entre los nodos. Un socket representa una combinación de dirección IP y número de puerto, y permite establecer canales de comunicación entre dos procesos (por ejemplo, GNU Radio y Node-RED) (Sockets, n.d.).

2.3 Diferencias Clave

2.3.1 Web Service

Requiere que el cliente inicie cada comunicación, y normalmente utiliza el protocolo HTTP.

2.3.2 Socket

Un socket, en comparación con un Web Service, puede mantenerse abierto y permitir un flujo constante de datos entre dos extremos, lo cual es ideal para transmisión continua, como ocurre con el monitoreo espectral en este proyecto, en las videoconferencias, en las llamadas IP, donde la inmediatez y la continuidad del intercambio de información son importantes. A diferencia de los Web Services, los sockets requieren que ambos extremos se sincronicen previamente y se mantengan conectados durante la sesión de comunicación.

2.3.3 Node-RED

Es una herramienta de programación visual que puede correr en prácticamente cualquier equipo que funcione como servidor, una manera de usar Node-RED es desplegarlo dentro de un contenedor en Microsoft Azure. Un contenedor es como una “cajita” ligera que agrupa la aplicación (en este caso, Node-RED) junto con todo lo que necesita para funcionar (librerías, configuraciones, etc.). A diferencia de una máquina virtual completa, el contenedor comparte el sistema operativo del servidor y ocupa menos recursos. El contenedor dispone de una dirección IP (un identificador único en la red) a través de la cual puede enviar y recibir datos. Para distinguir múltiples puntos de comunicación sobre esa misma dirección, se emplean los sockets: cada socket

integra la dirección IP con un número de puerto específico, funcionando como canal de entrada y salida de datos para un servicio o flujo concreto en Node-RED. Gracias a los sockets, es posible asignar puertos independientes dentro de una misma IP, lo que garantiza el aislamiento y la correcta dirección de cada flujo de datos según su propósito.

Desde ese contenedor en Azure, Node-RED coordina la recepción, el procesamiento y la entrega de los datos espectrales. Su interfaz visual permite integrar dispositivos, APIs y servicios sin necesidad de escribir mucho código, lo que lo hace ideal para prototipos IoT y soluciones basadas en la nube. Para facilitar la comprensión del sistema, es importante distinguir entre dos componentes clave: el Backend y el Frontend. El Backend es la parte del sistema que se encarga de recibir, procesar y gestionar los datos, generalmente funcionando en servidores o contenedores en la nube. En este caso, Node-RED actúa como el Backend, controlando la comunicación con los dispositivos y procesando la información. Por otro lado, el Frontend es la interfaz gráfica con la que interactúa el usuario final, a través de la cual puede visualizar y analizar los datos procesados (Frisoli, 2025). Para que esta interacción entre el Backend y el Frontend sea posible, se utiliza los protocolos de comunicación TCP, UDP y HTTP, cada uno con sus ventajas y aplicaciones. TCP está orientado a conexión, lo que significa que primero establece un canal entre emisor y receptor antes de enviar datos. Esto garantiza que las ventanas espectrales lleguen en orden y sin pérdidas, aunque introduce algo de latencia. Un ejemplo típico son los sistemas de telefonía IP (VoIP), como Cisco Unified Communications o Asterisk, donde cada fragmento de voz debe recibirse completo y en secuencia para mantener la claridad de la llamada; de igual modo, protocolos como SSH o la capa de transporte de HTTPS se apoyan en TCP para asegurar que ninguna parte de la información crítica se pierda o desordene (Bodnar, 2021). UDP, en cambio, es sin conexión, y envía las ventanas espectrales directamente, sin confirmar su llegada, lo que reduce la demora y acelera la

transmisión, aunque puede haber pérdida o desorden de las ventanas espectrales. Por esta razón se emplea en sistemas de videoconferencia como Zoom o Google Meet, donde los participantes se conectan a una hora acordada y el flujo continuo de audio y vídeo importa más que la entrega perfecta de cada ventana espectral; pequeñas pérdidas puntuales apenas se notan en la conversación, pero la baja latencia es fundamental para evitar retardos y ecos (UDP: Conozca La Revolución Silenciosa En La Transmisión De Datos, 2025). HTTP**,** que funciona sobre TCP, es el protocolo de la web que sigue un modelo de petición y respuesta cliente-servidor que garantiza la entrega íntegra de cada recurso. Así, miles o incluso millones de usuarios pueden acceder a un mismo servicio (por ejemplo, descargar un documento o navegar una página) en cualquier momento, sin necesidad de coordinación previa entre ellos, asegurando siempre que los datos lleguen completos y correctamente (Cloudflare, 2025). Finalmente, existen dos tipos de sistemas Tiempo Real, Tiempo Real Duro (Hard Real-Time) que requiere que los plazos de tiempo sean cumplidos de manera rigurosa y Tiempo Real Blando (Soft Real-Time) donde los plazos son importantes, pero pequeños retrasos son tolerables sin afectar el funcionamiento general (Jain, 2025). En este trabajo, cuando se dice Tiempo Real, en realidad se está hablando de Tiempo Real Blando. La transmisión y visualización en tiempo real implica retos de latencia, estabilidad y pérdida de datos, por lo que se aplican técnicas de control y sincronización para mitigarlos y garantizar una representación coherente del espectro; en este contexto, el reloj Unix, es una medida de tiempo que contabiliza los segundos desde el 1 de enero de 1970 a las 00:00:00 UTC, se emplea para marcar eventos en sistemas operativos y redes aunque su formato de 32 bits llegará al límite en 2038, situación conocida como el "problema del año 2038", hecho ilustrado con una visualización circular en notación hexadecimal y una herramienta web que convierte fechas a timestamps Epoch, (S32 De Unix, 2025) (Unix / Epoch Timestamp Conversion Tools, 2025); de

forma complementaria, el protocolo NTP sincroniza los relojes de estos sistemas mediante mensajes UDP por el puerto 123 en una jerarquía de estratos, calcula el desfase a partir del retardo de ida y vuelta y el jitter, y ajusta gradualmente el reloj local para lograr precisiones de milisegundos en Internet y microsegundos en redes locales (Servidor NTP O Servidor De Hora, 2025). Los conceptos previos descritos en este capítulo se encuentran desarrollados con mayor detalle en el Apéndice A, al cual se puede acudir para profundizar en definiciones, ejemplos y referencias adicionales.

3. Desarrollo de la Solución

En este capítulo se describe el desarrollo de la solución propuesta para llevar a cabo las pruebas comparativas orientadas a identificar la opción más adecuada para el monitoreo remoto del espectro en tiempo real, una necesidad planteada por el laboratorio de comunicaciones de la E3T en colaboración con el grupo RadioGIS. La visualización del espectro debe ser lo suficientemente representativa como para que el ojo humano perciba la información como “en vivo”, sin necesidad de transmitir toda la señal de manera continua.

El enfoque de desarrollo se organiza en dos soluciones complementarias, representadas en la Figura 2, donde los aspectos comunes son:

Equipos en la Antártida, aunque los equipos que miden el espectro pudiesen estar en cualquier parte del mundo, aquí se resaltan aquellos que se poseen actualmente en la UIS, específicamente la alianza de grupos de investigación de la E3T que trabaja en temas de

radioastronomía. El caso de la Antártida es especial debido a lo remoto del lugar y las difíciles condiciones de trabajo.

Equipos en Parque Guatiguará, de manera similar, en el Parque Guatiguará se viene construyendo un laboratorio de radioastronomía que, si bien tiene muy buenas capacidades de comunicación, reúne una amplia gama de tecnologías que requieren ser operadas muchas veces a distancia.

Equipos en Páramo de Berlín, eventualmente se realizan operaciones de medición del espectro en el Páramo de Berlín. Si bien es cierto que allí las condiciones de conectividad son muy difíciles, en ciertas circunstancias se da la necesidad de obtener señales del espectro en línea.

El personal en la UIS, Aunque en principio la solución permite la visualización desde cualquier lugar del mundo, aquí se usa el ejemplo concreto de los mayores interesados que son los investigadores que se encuentran en la UIS.

En la Figura 2, también se pueden ver los dos prototipos de solución así:

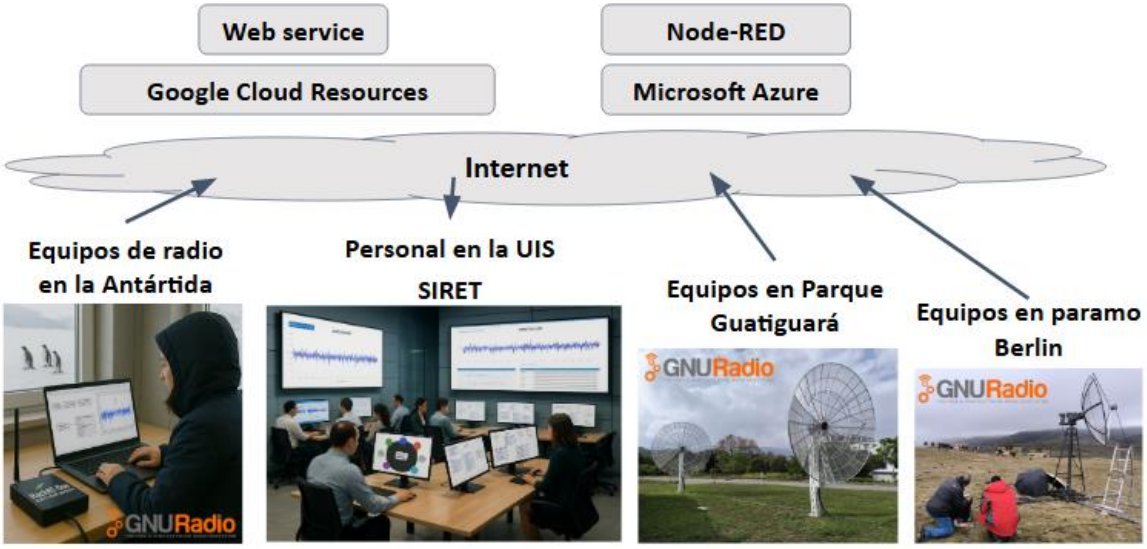
Prototipo I corresponde a una arquitectura basada en la capa de aplicación, que utiliza el protocolo HTTP mediante un Web Service construido con Google Apps Script, apoyado en recursos de Google Cloud como Google Sheets para el almacenamiento y la consulta de datos espectrales. Ver parte izquierda de la Figura 2.

Prototipo II Implementa una arquitectura basada en la capa de transporte, utilizando los protocolos UDP y TCP para la transmisión directa de datos desde GNU Radio. En este caso, los datos son recibidos por Node-RED, ejecutado dentro de un contenedor Docker desplegado sobre Microsoft Azure, y visualizados en tiempo real a través de una interfaz web accesible públicamente. Ver parte derecha de la Figura 2.

Ambas arquitecturas están diseñadas para ofrecer una visualización del espectro con baja latencia, útil tanto para tareas de sintonización remota y detección de interferencias, como para aplicaciones más exigentes, como el monitoreo espectral en sistemas montados sobre drones.

Figura 2.

Arquitectura general donde toda señal radioeléctrica que un SDR logra captar es transformada en datos útiles por el sistema.



Nota. La imagen ilustra cómo el espectro radioeléctrico es recibido por un dispositivo SDR (Equipos de radio en la Antártida, Equipos en Parque Guatiguará, Equipos en Páramo Berlín), para luego ser enviado a internet para ser vista por el usuario, que en este caso es el Personal en la UIS (RadioGis, 2025).

3.1 Prototipo I: Arquitectura Basada en la Capa de Aplicación

La Figura 3 muestra con mayores detalles el esquema general de esta arquitectura, organizada en dos fases principales. En la parte inferior derecha, se identifica el punto de origen de la señal: el dispositivo SDR, conectado a un computador con un módulo de software llamado STERT1, responsable de procesar las señales captadas por el SDR y transformarlas en ventanas

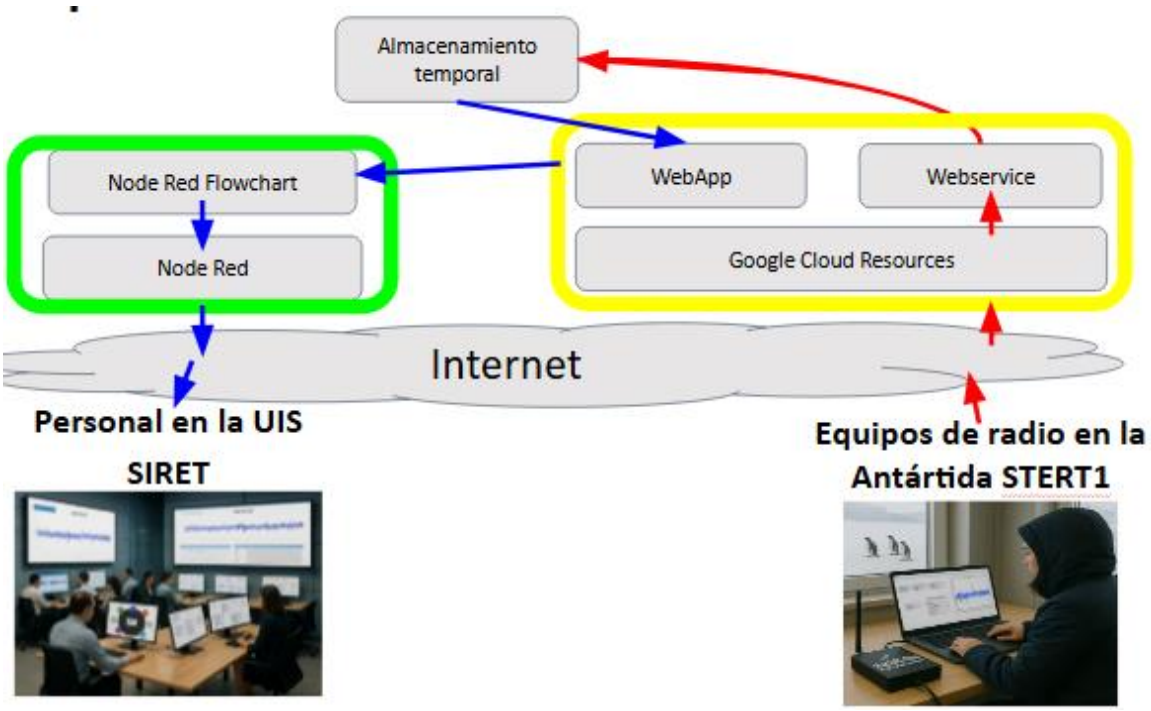
espectrales con formato digital. Estas ventanas son luego transmitidas a través del protocolo HTTP, utilizando el método POST hacia un Web Service desplegado en los recursos de Google Cloud.

En este esquema, la Fase de medición, representada en el recuadro amarillo, incluye todo el procesamiento realizado localmente con GNU Radio y el envío del dato a la nube. La Fase de visualización, identificada con el recuadro verde, corresponde al punto de interacción con el usuario final, quien accede a una WebApp desarrollada también sobre Google Cloud. Esta WebApp, que cumple el papel de Web Service de consulta, está equipada además con un frontend de visualización del espectro. De esta manera, el usuario puede ver de forma gráfica y dinámica la ventana espectral más reciente, con opciones para ajustar unidades y escalas, lo que convierte a esta WebApp en una solución funcional de monitoreo visual.

Para lograr que todo el desarrollo esté integrado en Node-RED junto con otros desarrollos, la WebApp ha sido embebida en Node-RED de manera similar a como se podría embeber en una página web. De manera que, en este prototipo, Node-RED no procesa los datos espectrales, solo sirve como pasarela de visualización. De este modo, los usuarios remotos se conectan a Node-RED desde cualquier navegador, y a través de su panel acceden a la instancia de la WebApp que les presenta la información espectral de forma continua y actualizada, en una experiencia cercana al tiempo real.

Figura 3.

Comunicación entre estaciones remotas usando HTTP.



Nota. En esta imagen se ve el diseño global del prototipo basado en HTTP, donde se integran tres componentes principales distribuidos en la nube**.**

3.1.1 Fase de Envío de Datos Usando Web Service

La información procesada y enviada desde GNU Radio es recibida por un Web Service desarrollado con Google Apps Script, que actúa como intermediario para almacenar temporalmente los datos espectrales en una hoja de cálculo de Google Sheets. El Web Service es capaz de recibir las mediciones espectrales, pero también se sabe que cuando un usuario invoca un Web Service, se crea una instancia en la nube, que luego desaparece cuando el usuario pierde la conexión con el Web Service. Entonces surge la pregunta ¿cómo pueden uno o varios usuarios acceder a esos datos? Para resolver este desafío, se implementa un mecanismo de almacenamiento temporal en la nube. Cada vez que el Web Service recibe una nueva ventana espectral, la registra inmediatamente en una hoja de cálculo de Google Sheets, actuando como la “portería de un conjunto residencial” que recibe y guarda un paquete para que otro componente lo entregue al

usuario final. Pero a diferencia de la “portería de un conjunto residencial”. La memoria en la nube solo mantiene el último dato recibido, lo cual es suficiente ya que esta solución se trata de un sistema de tiempo real, donde el interés está en observar el estado actual del espectro, no su histórico. La guía de funcionamiento del Web Service, que conecta GNU Radio con Google Apps Script, está disponible para profundizar en definiciones adicionales se encuentra en el Apéndice C.

Figura 4.

Registro de Datos en la Hoja de Cálculo.

A	B	C	D	E	F	G	H	I	J	K
Fecha	latitud	longitud	Azimut	Elevacion	Altitud	Descripcion	finicial	fpaso	N	Datos
2025-08-06 14:22:18	7,141126	-73,12289	0	90	959	Sensado HackrfOne	1420000000	488,28125	2048	4.16868019,1.20891356,0.33172965,0.1655

Nota. En la hoja de cálculo se registran los datos enviados desde GNU Radio, donde el usuario puede ver la fecha en la que se registran los datos.

El Web Service está configurado para recibir los datos mediante solicitudes HTTP POST en formato JSON. Al recibir una petición, el script realiza las siguientes acciones:

3.1.1.1 Parseo de Datos. Extrae los valores de los datos enviados desde GNU Radio, que incluyen información como la fecha y hora, ubicación (latitud y longitud), parámetros técnicos de la medición, y la matriz de datos del espectro.

3.1.1.2 Almacenamiento Temporal. La hoja de cálculo está identificada por un ID único y contiene una sola hoja donde se almacenan los datos como se muestra en la Figura 4. Para facilitar la visualización en tiempo real y evitar un crecimiento ilimitado del archivo, el script escribe siempre los datos en la fila 2 de la hoja de cálculo, esto crea la ilusión para el usuario de

que los datos se actualizan en tiempo real sin necesidad de almacenar un registro histórico extenso, optimizando así el uso de recursos.

3.1.1.3 Confirmación y Manejo de Errores. El script responde con un mensaje JSON que confirma el éxito o el fallo de la operación, lo que permite a GNU Radio o a cualquier cliente saber si la transmisión fue exitosa. Este esquema garantiza una comunicación eficiente entre el sistema local de adquisición y procesamiento de señales y la infraestructura en la nube, permitiendo la actualización continua y controlada de los datos para su posterior visualización o análisis.

3.1.2 Fase de Medición STERT1

La fase de medición abarca todo el procesamiento de la señal espectral, desde su captura y tratamiento en GNU Radio hasta un bloque Python encargado de transmitirla a Internet. Es realizada por el Sistema de Transmisión de Espectro Radioeléctrico en Tiempo Real (STERT1) y está representada en el flujograma de GNU Radio mostrado en la Figura 5. Este flujograma toma la señal captada por un equipo SDR y la transforma en información espectral (o espectro) representada como un conjunto de N valores numéricos que indican la intensidad de la señal en cada componente de frecuencia.

El desafío que surge es: ¿cómo lograr que estos datos lleguen al Web Service en la nube? La solución adoptada en el marco de este proyecto conste en el desarrollo de un bloque personalizado en GNU Radio (un Python Block), denominado “Enviar a Google Sheet”, que se encarga de interpretar el espectro, añadirle información adicional (como ubicación geográfica y parámetros de medición) y preparar el paquete de datos para ser enviado por internet, para lo cual este programa para consumir el Web Service previamente implementado.

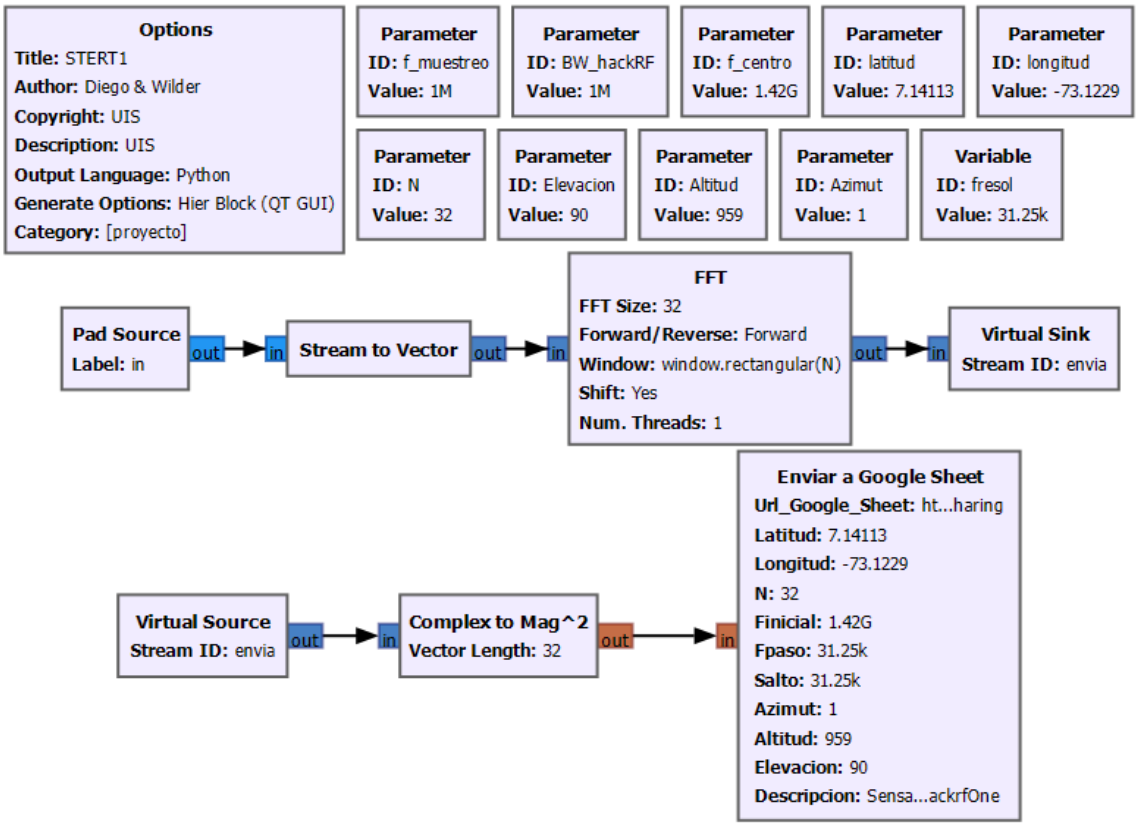
El Apéndice B documenta en detalle el flujograma STERT1, así como el código fuente y el uso del bloque “Enviar a Google Sheet”.

El módulo STERT1 opera a partir de la señal de radio captada por el equipo SDR y convertida a Envolvente Compleja. Como primer paso la señal se convierte a tipo vectorial mediante el bloque “Stream to vector” para que sea compatible con los demás bloques que manejan señales tipo vector, como es el caso del bloque “FFT” encargado de obtener el espectro, el bloque “Complex to Mag2” que se encarga de convertir el espectro en valores reales de magnitud al cuadrado y finalmente nuestro bloque “Enviar a Google Sheet” que se encarga de enviar a la nube esos valores. El bloque “Enviar a Google Sheet” deber ser previamente configurado con: la URL del servicio web al que se enviarán los resultados, la latitud y longitud de la estación, los valores técnicos de frecuencia inicial, paso de frecuencia y número de muestras, así como metadatos adicionales —azimut, elevación, altitud y una breve descripción del sitio— definidos por los grupos de investigación RadioGIS y CEMOS.

Para evitar congestionar la red, el bloque incluye el parámetro “salto” y un módulo que permite saltarse un número de ventanas espectrales antes de invocar el Web Service. Para el envío, el vector es convertido en una cadena de datos, se construye un objeto JSON con fecha y hora, los metadatos configurados y los valores espectrales, y lo remite mediante una petición HTTP POST al Web Service implementado en Google Apps Script. Finalmente, es posible verificar la respuesta del servidor así: si la operación resulta exitosa confirma el registro y es posible ver que cambian los valores en la Google Sheet; en caso contrario, informa el error en la consola para diagnóstico y depuración.

Figura 5.

Flujograma STERT1.

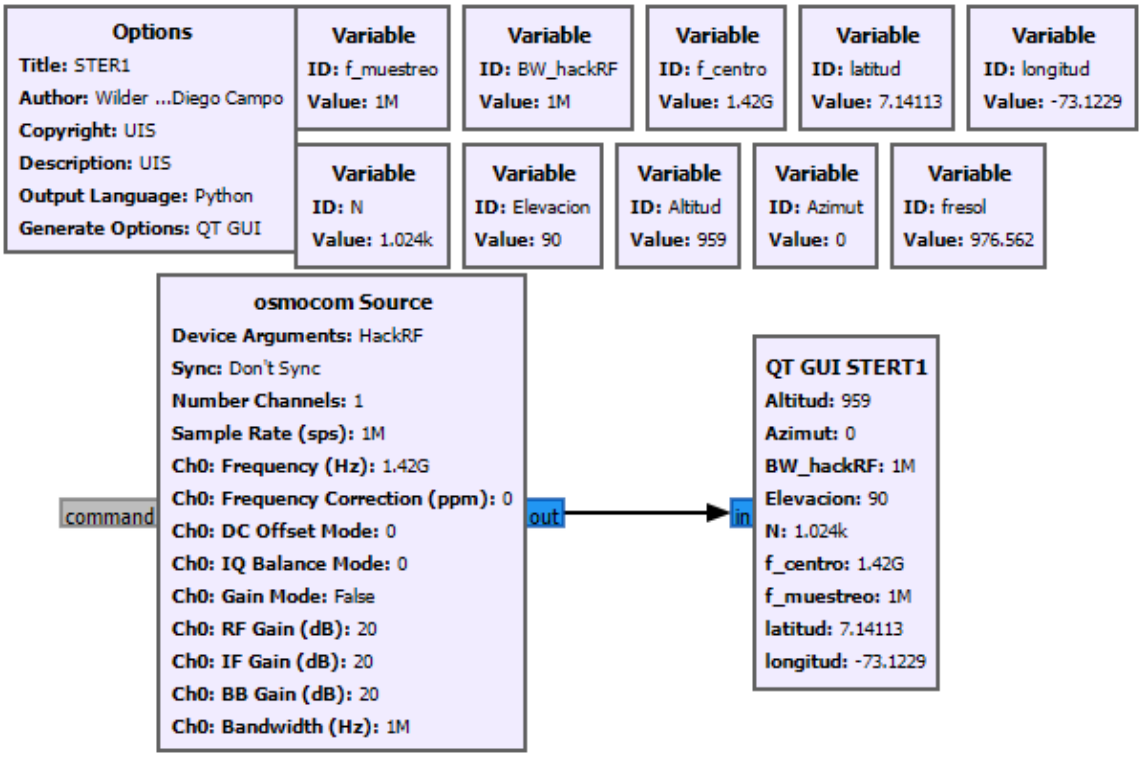


Nota. Esta es la configuración en GNU Radio para el envío de datos a internet por el bloque de Python ENVIAR a Google Sheet.

Para simplificar el sistema, se encapsuló todo el contenido de STERT1 en un único bloque jerárquico (Hier Block), de modo que el diagrama principal muestra únicamente ese bloque sin exponer sus detalles internos.

Figura 6.

Configuración del flujograma general para la transmisión por HTTP.



Nota. El bloque QT GUI ENVIAR A GOOGLE SHEET contiene por dentro la configuración del flujograma STERT1.

3.1.3 Fase de Visualización para el Usuario Final Usando WebApp

En esta etapa, se aprovecha el trabajo que hace el Web Service guardando en la nube, de manera temporal, la última ventana espectral. Así, la WebApp solo necesita leerla y mostrarla de manera gráfica a cualquier persona desde cualquier computadora o dispositivo móvil. La información sobre el funcionamiento de la WebApp, incluyendo detalles sobre la integración con Google Sheets y la actualización dinámica de gráficos, se encuentra en el Apéndice D.

3.1.3.1 Obtención y Gestión de los Datos. La WebApp cuenta con un script Backend que lee la hoja de cálculo de Google Sheets donde se almacenan los datos. Este script es capaz de recuperar la URL y el ID de la hoja de cálculo para acceder a ella, extraer los nombres de las hojas disponibles, permitiendo al usuario seleccionar cuál desea visualizar, leer los datos espectrales organizados, incluyendo la frecuencia inicial, el paso de frecuencia, el número de muestras y los

valores de intensidad, adaptándose para su presentación gráfica, gestionar parámetros como la cantidad de datos a mostrar y la posición temporal para simular una visualización en tiempo real.

3.1.3.2 Interfaz de Usuario para Visualización. La página HTML sirve como Frontend de la aplicación, ofreciendo una interfaz amigable que incluye controles para seleccionar la unidad de frecuencia (Hz, kHz, MHz, GHz) y la escala de magnitud (mag^2 o dB), un selector para elegir la hoja de cálculo activa para la visualización, una gráfica dinámica realizada con la biblioteca Plotly que muestra el espectro de frecuencias en dos dimensiones (Frecuencia y Magnitud), actualización automática de la gráfica cada segundo para reflejar cambios en los datos almacenados, simulando tiempo real.

3.1.3.3 Flujo de Funcionamiento. Al cargar la WebApp, el script Backend recupera los nombres de las hojas y carga los datos iniciales para la visualización. Cuando el usuario cambia cualquiera de las opciones (unidad, escala o hoja), la gráfica se actualiza para reflejar esa configuración. La gráfica permite visualizar la intensidad de la señal a lo largo del espectro de frecuencias, facilitando el análisis inmediato de las condiciones de la señal recibida.

Este conjunto de recursos en la nube garantiza que, desde la captura en GNU Radio hasta la visualización en el navegador, el sistema opere de manera integrada, eficiente y accesible para el usuario final.

3.2 Prototipo II: Arquitectura Basada en la Capa de Transporte

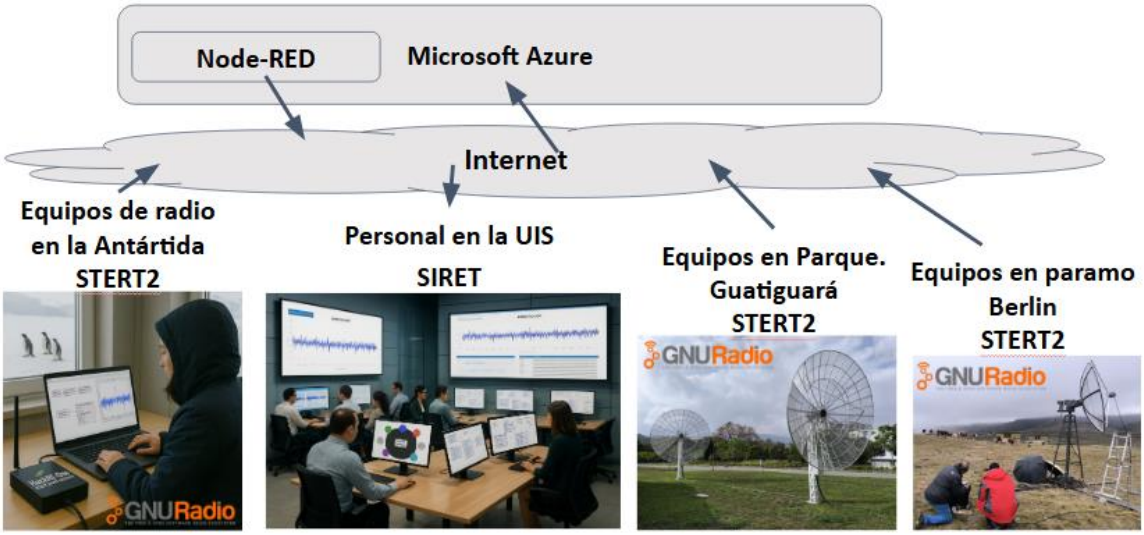
Tal como muestra la Figura 7, la arquitectura pone en marcha un proceso completamente remoto como el caso anterior, pero ahora ya no se basa en la capa de aplicación sino en la capa de transporte.

En cada estación SDR (por ejemplo, con un HackRF One operando en el Páramo de Berlín, en el Parque Guatiguará o desde un equipo portátil en la Antártida) se captura el espectro y se

procesa localmente mediante el flujograma STERT2 de GNU Radio. Esta configuración permite agrupar los datos en ventanas espectrales para la transmisión mediante UDP o TCP, a través de internet hacia un contenedor de Node-RED alojado en Microsoft Azure (ver sección 5.3).

Figura 7.

Comunicación a través de Internet entre estaciones SDR remotas, la plataforma Node-RED en Azure y los usuarios de la UIS.



Nota. Esquema de la comunicación bidireccional de datos espectrales: las estaciones SDR en la Antártida, la UIS, el Parque Guatiguará y el Páramo de Berlín pueden capturar y procesar el espectro localmente y enviarlo a través de Internet a la plataforma Node-RED desplegada en Microsoft Azure; las flechas indican el sentido de envío y recepción de la información.

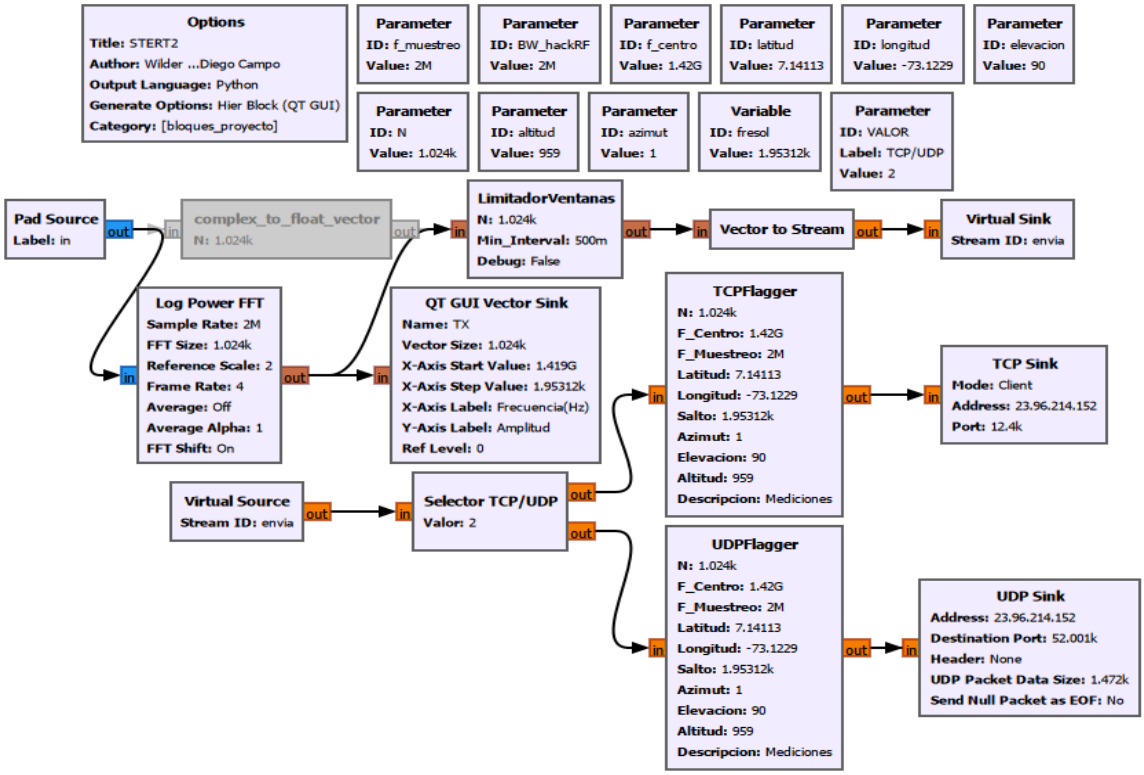
3.2.1 Fase de Emisión del Espectro Hacia Internet STERT2

A continuación, se muestra el flujograma STERT2, en la Figura 8, donde cada bloque representa una etapa independiente del procesamiento espectral. El bloque Pad Source se utiliza para definir los puertos de entrada y salida de un bloque jerárquico en GNU Radio. No es un bloque de procesamiento de datos en sí mismo, sino que actúa como un contenedor para otros bloques dentro de un bloque jerárquico. A continuación, el bloque complex to float vector puede ser

utilizado para transformar la señal compleja en un vector de valores en formato de punto flotante. En el proceso, se toma la magnitud de cada muestra compleja utilizando la función np.abs, lo que permite trabajar solo con los valores absolutos de la señal compleja. Es importante notar que este bloque solo se puede usar si se decide ver la señal de esta manera, ya que no puede pasar por el bloque Log Power FFT al mismo tiempo debido a la diferencia en los tipos de datos de salida. En caso de optar por el bloque Log Power FFT, este calcula la densidad espectral de potencia de la señal, aplicando una función de ventana a las muestras de la envolvente compleja. A continuación, ejecuta la transformada rápida de Fourier (FFT), obteniendo los coeficientes de la señal en el dominio de la frecuencia. Cada coeficiente complejo es convertido en su magnitud al cuadrado, y luego se escala logarítmicamente. Esta operación hace que las variaciones de nivel de potencia en el espectro sean más evidentes, lo cual facilita la visualización y análisis de las características espectrales de la señal (Log Power FFT, 2025).

Figura 8.

Bloques en GNU Radio llamado STERT2 para la transmisión del espectro.



Nota. Esta imagen muestra que el Pad Source encapsula varios bloques, agrupándolos en un solo elemento visual en GNU Radio.

El siguiente bloque se llama LimitadorVentanas y regula la frecuencia de emisión según el parámetro Min_Interval, es decir que solo permite la salida de una ventana cada Min_Interval segundos, esto con el fin de poder regular el tráfico en función de las condiciones de congestión que presente la comunicación. Luego viene el bloque Vector to Stream que traduce esos vectores en un flujo continuo de datos; Los bloques virtual sink y virtual source están conectados internamente, sin la necesidad de mostrar la línea de conexión, el Selector TCP/UDP dirige la señal hacia TCPFlagger o UDPFlagger, donde se insertan unas banderas y metadatos que se explican más adelante. Finalmente, la información sale por el TCP Sink o el UDP Sink hacia el destino remoto.

El flujograma STERT2 implementa, de forma modular, la captura, el procesado y el envío remoto del espectro radioeléctrico. Su estructura principal es la siguiente:

3.2.2 La Necesidad de Regular la Tasa de Envío de Ventanas Espectrales

Cada ventana espectral que entra a STERT2 tiene N muestras y en la radioastronomía se viene usando el valor de N = 4096. Vamos a suponer una frecuencia de muestreo de $f_s = 5MHz$ y se obtiene que:

(Ecuación 1) $Ventanas\ por\ segundo = \frac{f_s}{N} = \frac{5MHz}{4096} \approx 1220\ ventanas$

Esta tasa supera con creces las ≈ 24 fotogramas por segundo (FPS) que utiliza el cine y el video para que el ojo humano perciba movimiento continuo sin parpadeo. Para evitar que la interfaz se sature sin perder detalle útil, se creó el bloque Python Block personalizado, llamado LimitadorVentanas, que reduce la salida a 2 ventanas/s, un valor para percibir cada cambio. Este bloque mide el tiempo transcurrido entre una ventana y la siguiente y solo deja pasar la siguiente cuando se cumple el intervalo mínimo definido por el parámetro Min_Interval dado segundos. Con el valor predeterminado de 0.5 segundos, las ventanas bajan a:

(Ecuación 2) $Ventana\ de\ salida = \frac{1}{min\_interval} = \frac{1}{0.5\ s} = 2\ ventanas\ por\ segundo$

frecuencia que el usuario percibe como una actualización clara pero no demasiado rápida. De esta forma, se evita que las ventanas lleguen demasiado rápido y se sobrecargue la etapa de envío o la interfaz de visualización, con un tráfico innecesario.

El ajuste de 0.5 segundos, equilibra ambos factores y hace imperceptibles las ventanas descartadas, cumpliendo el objetivo de una visualización clara sin saturar el sistema. A continuación, el bloque Vector to Stream transforma cada vector de N muestras en un flujo continuo de datos, preparándolo para su envío. Este flujo continuo de datos, entra luego al bloque personalizado Selector TCP/UDP, que dispone de un parámetro de modo (valor = 0, 1 ó 2) para

decidir si envía las muestras únicamente por TCP, únicamente por UDP o por ambas rutas simultáneamente, simplemente reenvía lo que llega a la entrada, a las salidas indicadas.

3.2.3 El Sistema de Banderas

Durante el desarrollo e implementación de los protocolos UDP y TCP, se observó que ambos afectan el espectro radioeléctrico de manera distinta. UDP, al no garantizar la llegada completa ni el orden correcto de la información, provoca que el espectro recibido se distorsione considerablemente, perdiendo similitud con el espectro originalmente enviado. Esta situación dificulta mantener la calidad y precisión en la transmisión del espectro radioeléctrico. Por otro lado, TCP, al garantizar la entrega completa y ordenada de los datos mediante su mecanismo de retransmisión, introducía otro tipo de problema, debido a los reenvíos de segmentos cuando se detectaban pérdidas o retrasos, el sistema podía recibir ciertos fragmentos de la señal más de una vez. Esto generaba que el espectro se muestre deformado, con datos duplicados o superpuestos, afectando la representación fiel de la señal y dificultando su análisis correcto. Para resolver estos problemas en la transmisión mediante UDP y TCP, se diseñan dos bloques específicos: TCPFlagger y UDPFlagger, desarrollados en Python block y empleados según el protocolo elegido. Ambos bloques tienen la función de estructurar los datos espectrales con el fin de evitar las distorsiones provocadas por las características propias de cada protocolo. Para lograrlo, se incorpora un sistema de banderas numéricas que permiten delimitar claramente la ventana espectral y diferenciar los datos útiles de la información.

En cada ventana espectral, se inserta la bandera -1.0 para indicar el inicio y -2.0 para marcar el final del bloque de muestras. Justo antes y después de estos bloques, se agregan metadatos de frecuencia: frecuencia inicial (f_inicial), paso de frecuencia (f_paso), frecuencia central (f_central)

y frecuencia final (f_final), que proporcionan el contexto necesario para interpretar correctamente las muestras recibidas.

Además, para distinguir claramente la información espectral de los datos adicionales como latitud, longitud, salto, azimut, elevación, altitud y una descripción (en este caso el lugar donde se está tomando la muestra) se implementa banderas especiales: -3.0 marca el inicio y -4.0 el final de este conjunto de metadatos complementarios.

Este sistema de banderas y metadatos cumplen dos funciones:

3.2.3.1 Control de Ventanas Espectrales. Permite al receptor identificar el comienzo y fin de cada ventana espectral, mitigando los efectos de reenvíos o pérdidas que generan deformaciones o duplicaciones en la señal, especialmente en TCP.

3.2.3.2 Separación Clara de Datos. Facilita la distinción entre las muestras espectrales y la información contextual, evitando confusiones en el procesamiento posterior.

Gracias a esta estructura, se logra estabilizar la representación del espectro radioeléctrico, eliminando distorsiones y duplicaciones en la visualización. Así, se optimiza la calidad y precisión en la recepción y análisis de las señales, independientemente del protocolo utilizado. Finalmente, los datos regulados se transmiten a Internet mediante un TCP Sink o UDP Sink, apuntando a la dirección IP y puerto previamente definidos, garantizando una comunicación estable y eficiente. El Apéndice E ofrece una explicación detallada de cada uno de los bloques que conforman el flujograma STERT2, explicando su función y cómo contribuyen al proceso general.

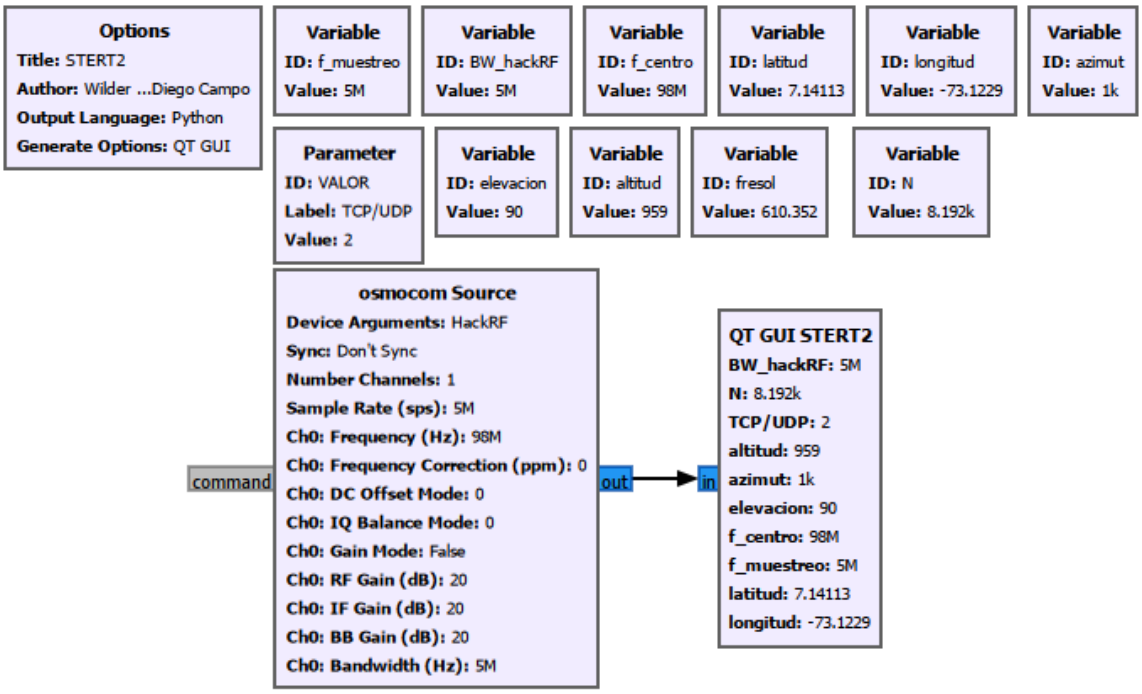
3.2.4 Fase de Medición del Espectro Acoplada a STERT2

El bloque Osmocom Source, mostrado en la Figura 9, recibe las muestras de la envolvente compleja del espectro radioeléctrico desde dispositivos SDR compatibles, como HackRF One. La

fase de medición del espectro se acopla al bloque STERT2, que completa el procesamiento y funcionalidad del sistema tras recibir los datos espectrales.

Figura 9.

Flujograma STERT2 en un bloque.



Nota. El bloque STERT2 envía cualquier señal x(t) que se le conecte, enviando esa señal a la nube. Así, puedes cambiar la señal fácilmente sin tocar nada más.

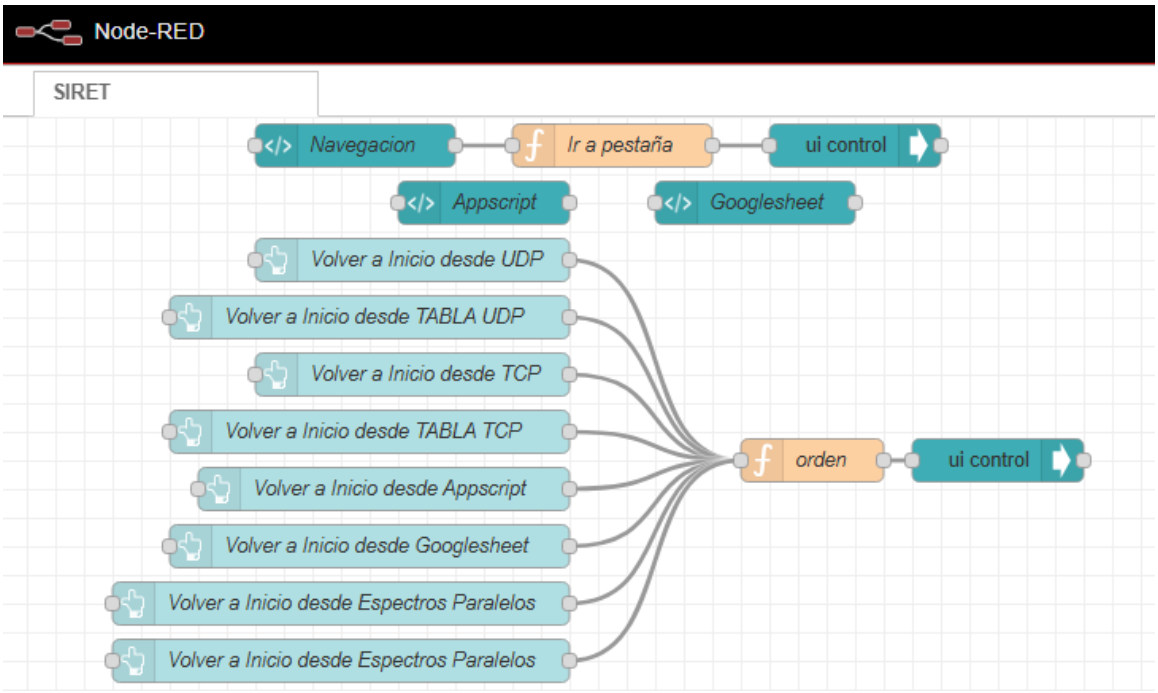
3.3 Fase de Visualización Integrada para el Servicio de Usuario Final

Para unificar la visualización de ambos prototipos y facilitar el establecimiento de la comunicación, se utilizó Node-RED, destacando su versatilidad el cual basta con sustituir la señal de entrada al bloque STERT2 (audio, portadora de prueba o espectro SDR) para evaluar distintos escenarios sin necesidad de reescribir código ni modificar la configuración de red, su aislamiento ya que Node-RED opera de forma independiente de la fuente, por lo que no requiere ajustes internos al cambiar la señal, su escalabilidad la cual permite que nuevas funciones (como

metadatos o modulaciones adicionales) pueden integrarse fácilmente añadiendo bloques al diseño existente.

Figura 10.

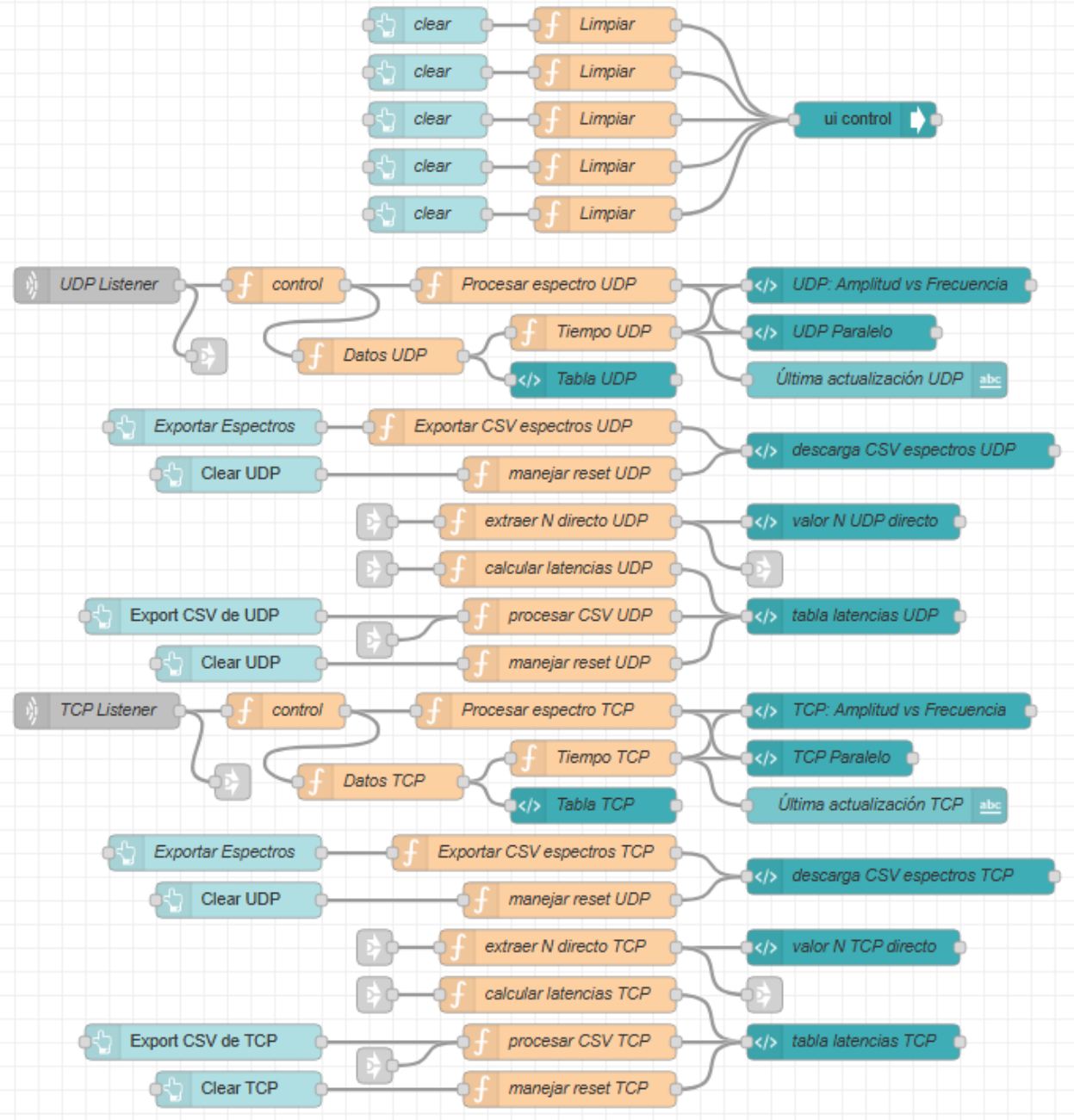
Flujograma SIRET para la recepción y visualización del espectro radioeléctrico (UDP/TCP) Parte A.



Nota. Esta figura la parte A del Backend del sistema que muestra la configuración de los nodos. En Node-RED para recibir, procesar y visualizar los datos espectrales en tiempo real, utilizando los protocolos UDP y TCP.

Figura 11.

Flujograma SIRET para la recepción y visualización del espectro radioeléctrico (UDP/TCP) Parte B



Nota. Esta figura la parte B del Backend del sistema que muestra la configuración de los nodos. En Node-RED para recibir, procesar y visualizar los datos espectrales en tiempo real, utilizando los protocolos UDP y TCP.

Con Node-RED, los datos se visualizan mediante gráficos interactivos que simplifican su interpretación y apoyan la toma rápida de decisiones. La aplicación ha sido desplegada en un

contenedor Linux sobre Microsoft Azure, configurado con tres puertos: 1880 para la interfaz web, 12400 (TCP) y 52001 (UDP) para la recepción de espectros. Cuando el contenedor en la nube se encuentra en ejecución, el sistema es accesible desde cualquier navegador mediante la IP pública o el FQDN (Fully Qualified Domain Name) asignado, seguido del puerto 1880. Por ejemplo:

IP:http://4.157.187.135:1880/ui

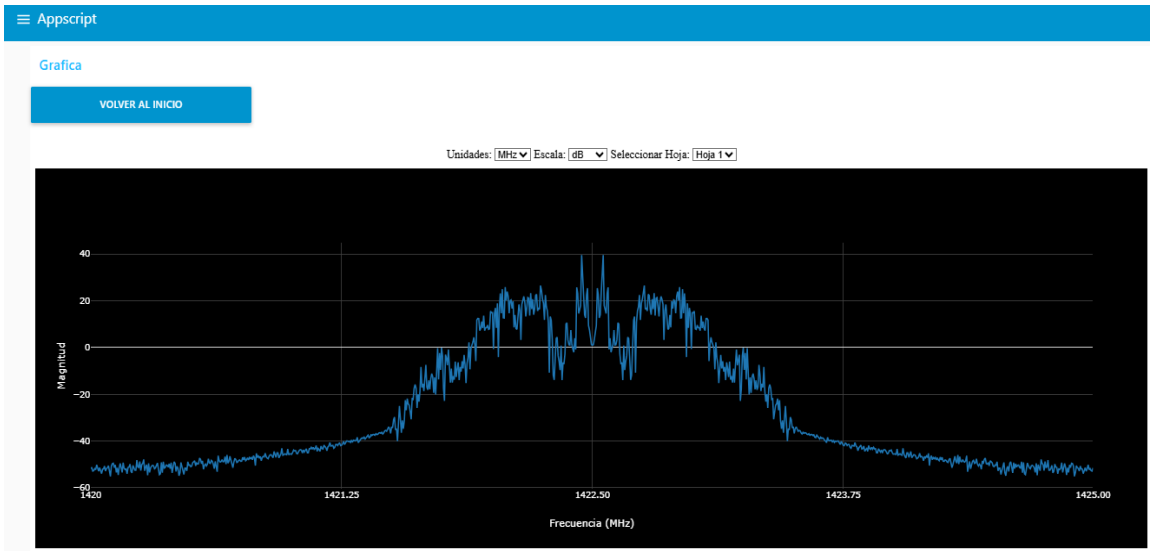
FQDN: http://scada-iot.e7ebafcnc8d2eeez.eastus.azurecontainer.io:1880/ui.

Una vez conectado, el usuario puede visualizar y monitorear en tiempo real los datos generados por el SDR. El proceso completo de creación y configuración de dicho contenedor se detalla en el Apéndice F.

3.3.1 Integración de la WebApp y Google Sheets en Node-RED para la Visualización de Datos.

Figura 12.

Gráfica vista desde la interfaz en Node-RED.



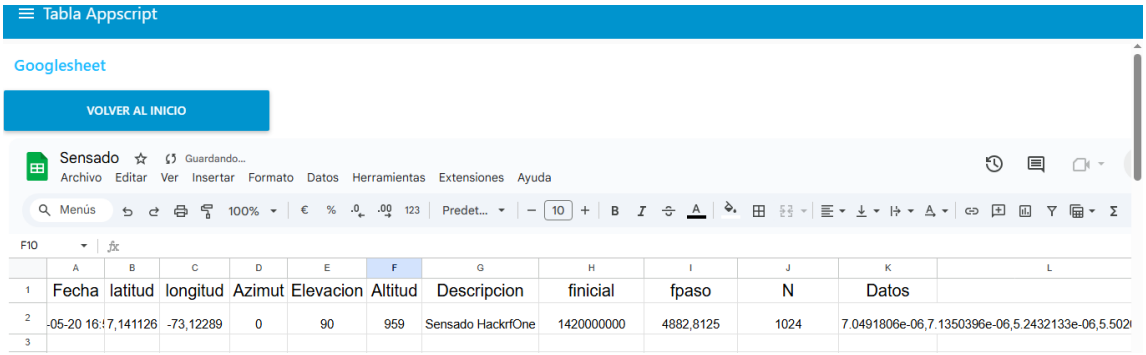
Nota. Esta imagen corresponde a la información que ve el usuario en la interfaz interactiva final.

Para integrar tanto la **WebApp** como la **hoja de cálculo de Google** en Node-RED, se utilizó un enfoque similar para ambos y este fue de insertarlos en el dashboard de Node-RED mediante etiquetas iframe (un elemento HTML que permite introducir contenido externo dentro

de una página web), lo que permite al usuario visualizar directamente dentro de la plataforma sin necesidad de abrir nuevas ventanas o pestañas. En el caso de la WebApp, se inserta la URL del Web Service activo generado por Google Apps Script, mientras que para la hoja de cálculo se utiliza el enlace directo de Google Sheets. La visualización se adapta para que ambos elementos ocupen el 100% del ancho disponible, ajustando la altura de cada uno según lo necesario para asegurar una presentación limpia y accesible.

Figura 13.

Información de la hoja de cálculo en Node-RED.



Nota. Esta imagen la información en la que el usuario puede ver el tiempo que tarda en llegar la información de los datos a la hoja de cálculo.

3.3.2 Captura de la Ventana Espectral con los Nodos Listener (UDP Y TCP)

Una vez activo el contenedor en la nube, cada listener recibe los datagramas entrantes y los envía al bloque correspondiente Procesar espectro UDP/TCP para su posterior procesamiento y visualización.

3.3.2.1 Decodificación y Reconstrucción del Eje de Frecuencias. Los nodos Procesar espectro UDP y Procesar espectro TCP convierten los datos recibidos a través de sockets (medios para la comunicación en tiempo real mediante UDP/TCP), preparándolos para su visualización. El proceso se organiza en los siguientes pasos:

3.3.2.1.1 Acumulación Segura del Búfer. El nodo Function almacena los fragmentos de datos hasta completar una ventana espectral. Una vez alcanzado el tamaño necesario, la ventana completa se envía y el resto se guarda para la siguiente iteración.

3.3.2.1.2 Localización de Banderas. El algoritmo identifica las marcas de inicio (-1.0), fin (-2.0) y bloques de constantes (-3.0 a -4.0, como latitud y longitud, etc).

3.3.2.1.3 Reconstrucción del Eje de Frecuencias. En el eje X se representa las frecuencias cubiertas por la FFT en una escala lineal, generando N puntos equidistantes, en el eje Y representa la potencia en decibeles (dB), etiquetada como “Potencia (dB)” gracias al bloque Log Power FFT, y se grafican las muestras obtenidas, Eje Y: Representa la potencia en decibeles (dB), etiquetada como “Potencia (dB)” gracias al bloque Log Power FFT, y se grafican las muestras obtenidas, en la salida para Plotly los datos recibidos se guardan en el almacenamiento local del navegador para recuperar la gráfica tras recargar la página. La configuración de los nodos en los protocolos UDP y TCP se detalla en el Apéndice G, en la marca temporal los nodos Tiempo UDP y Tiempo TCP actualizan la etiqueta Última actualización. En el Apéndice H, está todo el detalle de la configuración de estos tiempos.

3.3.3 Visualización Gráfica Interactiva

Una vez que los nodos Procesar espectro UDP/TCP entregan el objeto {x, y}, los bloques UDP: Amplitud vs Frecuencia (traza azul) y TCP: Amplitud vs Frecuencia (traza roja) lo dibujan con Plotly JS directamente en el dashboard de Node-RED. Cada gráfica muestra el espectro en tiempo real (frecuencia vs amplitud en dB) y se actualiza a medida que llegan nuevas ventanas espectrales, la plantilla de Plotly es idéntica para ambos protocolos, de manera que la escala, las etiquetas y el aspecto visual sean coherentes, al pulsar el botón Dual, el usuario visualiza ambas

gráficas superpuestas en un único panel, lo que facilita su comparación, al ser interactiva, la gráfica permite acercar, desplazar y exportar la vista sin interrumpir la recepción de datos.

Para el detalle completo del código que genera las gráficas incluida la gestión de redimensionado, la persistencia de la hora y los estilos de Plotly, consúltese el Apéndice I.

3.3.3.1 Control de la Interfaz Sin Recarga de Página. En el tablero de Node-RED existe un componente llamado ui_control que se encarga de mostrar diferentes pantallas en el mismo espacio. Cuando el usuario pulsa un botón, el sistema envía un mensaje interno con el nombre de la pantalla que desea ver. El componente ui_control toma ese mensaje y muestra inmediatamente la nueva pantalla, sin necesidad de volver a cargar todo el sitio. De esta forma, el cambio es instantáneo y la navegación resulta fluida, incluso para operadores que acceden desde la red de internet. Para ver detalles del panel de navegación consulte Apéndice J.

3.3.3.2 Panel de Navegación Circular para Usuarios a Distancia. El panel de navegación (Figura 14) se diseña para optimizar la interacción con el sistema de monitoreo. Con una disposición circular y una paleta cromática coherente, con los gráficos (azul para UDP y rojo para TCP), sus botones permiten alternar entre las distintas vistas sin recargar la página.

Figura 14.

Panel circular de navegación que permite al usuario cambiar de vista con un solo clic.

Nota. El panel es el Frontend del diseño, que se diseña para usuarios que operan a distancia, permite un control y visualización de datos en tiempo real, sin importar la ubicación geográfica. A través de este panel, los usuarios pueden navegar entre vistas, realizar ajustes, y visualizar los datos procesados de manera interactiva.

4. Resultados

4.1 Métricas Globales de Desempeño Según el Tamaño de la Ventana(N)

En general, el protocolo HTTP, utilizado con Google Apps Script como intermediario, demostró ser una opción funcional para la medición y el registro de datos con una latencia aceptable y un jitter bajo, incluso con tamaños de muestra más grandes. La estabilidad de la conexión es relativamente buena, con la congestión de la red como el principal factor que afecta la latencia. Si bien HTTP no es el protocolo más adecuado para aplicaciones en tiempo real, en este contexto específico mostró un desempeño confiable. La evidencia obtenida sugiere que, para proyectos con requerimientos de latencia y estabilidad en condiciones controladas, HTTP puede ser una solución válida, aunque siempre será recomendable alternativas como UDP o TCP para aplicaciones donde latencias bajas sean muy importantes. Los resultados descritos en este capítulo se encuentran desarrollados con mayor detalle en el Apéndice K, al cual se puede acudir para profundizar en definiciones, cálculos y referencias adicionales.

Tabla 1.

Promedio de Mediciones de Latencia TCP por Diferentes Valores de N y Media General.

Tamaño de N	**Min (ms)**	**Max (ms)**	**Avg (ms)**	**P50 (ms)**	**P95 (ms)**	**Jitter (ms)**	Loss(%)
256	172.80	189.39	178.22	176.35	179.82	3.07	0.00%
512	188.25	739.21	202.05	194.80	233.62	14.71	0.00%
1024	625.26	1180.93	726.56	709.33	774.53	54.36	0.00%
2048	697.31	117229.34	3756.50	724.34	1234.79	3134.82	0.00%
4096	1216.25	139222.77	11649.86	1268.28	138750.8	3612.12	0.00%
8192	No datos	No datos	No datos	No datos	No datos	No datos	0.00%

Media General	593.77	29219.01	5402.03	715.02	1567.51	1176.42	0.00%

Nota. La tabla muestra los valores promedio de latencia para el protocolo TCP con diferentes tamaños de ventana (N). También incluye la media general calculada de todas las mediciones de N, excluyendo el caso de N = 8192, ya que no hay datos disponibles.

Tabla 2.

Promedio de Mediciones de Latencia UDP por Diferentes Valores de N y Media General.

Tamaño de N	**Min (ms)**	**Max (ms)**	**Avg (ms)**	**P50 (ms)**	**P95 (ms)**	**Jitter (ms)**	**Loss(%)**
256	4649.42	52966.65	28110.67	6528.13	102420.0	3013.81	0.00
512	1304.03	127939.01	126322.9	6528.13	102420.0	3013.81	17.43
1024	663.37	1192.05	716.08	698.57	721.15	33.47	64.29
2048	197.27	1199.15	698.57	698.02	722.01	129.22	81.08
4096	205.26	1231.54	705.53	699.69	718.98	252.83	89.47
8192	No datos	No datos	No datos	No datos	No datos	No datos	No datos
Media General	2011.67	28390.88	12362.58	6747.29	7474.45	1665.43	38.53

Nota. La tabla muestra los valores promedio de latencia (min, max, avg, jitter, loss) para los diferentes tamaños de ventana N. Se incluyen los promedios de cada grupo (N) y una media general de todos los tamaños de N.

Tabla 3.

Comparación Global de Latencias para HTTP por Diferentes Valores de N y Media General.

Tamaño de N	Min (ms)	Max (ms)	Avg (ms)	P50 (ms)	P95 (ms)	Jitter (ms)	Loss(%)
256	2565	4105.33	3366.65	3294.33	4105.33	454.216	0.00%
512	2405	4501.66	3313.606	3269.33	4501.66	531.33	0.00%
1024	2439.33	4441.33	3291.056	3315	4441.33	566.053	0.00%
2048	1915.66	3528.66	2761.396	2802.66	3525.33	459.79	0.00%
4096	2436.66	3482.66	2920.606	2905	3482.66	294.616	0.00%
8192	No datos	No datos	No datos	No datos	No datos	No datos	0.00%
Media General	2352.33	4011.928	3130.6628	3117.264	4011.262	461.201	0.00%

Nota. La tabla muestra los valores promedio de latencia para el protocolo HTTP con diferentes tamaños de ventana (N). También incluye la media general calculada de todas las mediciones de N, excluyendo el caso de N = 8192, ya que para esta cantidad de muestras no registra datos.

5. Conclusiones

Los resultados obtenidos confirman que se cumplieron todos los objetivos propuestos: tanto la solución basada en Google Apps Script como la implementada con UDP/TCP en Node-RED permiten una visualización adecuada y con baja latencia del espectro. La alternativa con Apps Script facilita el registro automático en Google Sheets, aunque con una latencia mayor, mientras que el enfoque con UDP/TCP en Node-RED logra tiempos de respuesta mínimos a costa de una mayor complejidad de configuración. Ambas estrategias cubren las necesidades del

laboratorio de comunicaciones de la E3T y del grupo RadioGIS en sus respectivos contextos de uso. En consecuencia, se identificó claramente el entorno óptimo para cada método, cumpliendo así plenamente los requerimientos iniciales.

Los resultados obtenidos para el protocolo HTTP + Apps Script indican que, aunque este protocolo no está pensado para transmitir datos en tiempo real, su desempeño en términos de latencia es aceptable en comparación con las otras opciones. Aunque la latencia aumenta considerablemente al usar ventanas de mayor tamaño, HTTP logró mantener un bajo nivel de variabilidad (jitter), incluso con tamaños grandes de ventana (N = 4096). Esto hace que sea adecuado para situaciones donde la fiabilidad es más importante que la rapidez en la transmisión de datos. La confiabilidad de HTTP se demuestra por la ausencia de pérdida de datos durante las pruebas. No obstante, su rendimiento en aplicaciones en tiempo real es limitado debido a la latencia más alta, especialmente cuando se compara con los otros protocolos, como UDP y TCP, que tienen menores retrasos.

Tanto UDP como TCP funcionaron bien dentro de Node-RED, pero cada uno tiene ventajas dependiendo de lo que se necesite. UDP fue el más rápido, especialmente cuando se usaron ventanas más pequeñas, lo que lo hace ideal para situaciones donde la velocidad de transmisión es lo más importante, como en el monitoreo en tiempo real o videollamadas. Sin embargo, cuando se usaron ventanas más grandes, la variabilidad en los tiempos de llegada de los datos aumentó, lo que afectó su estabilidad. Por otro lado, TCP fue más estable y se aseguró de que los datos llegaran completos y en el orden correcto, aunque era más lento que UDP, especialmente con ventanas más grandes. Aunque TCP tuvo una mayor latencia, su capacidad de ser fiable lo hizo más adecuado para situaciones donde es importante que los datos lleguen sin errores, aunque tome un poco más de tiempo. En resumen, UDP es mejor cuando se necesita rapidez, mientras que TCP es la opción

ideal cuando lo más importante es asegurar que los datos lleguen completos y en orden, aunque con un poco más de demora.

La experiencia adquirida durante los periodos de confinamiento evidencia la necesidad de contar con alternativas de formación asíncrona, virtual o con acceso a prácticas a larga distancia sobre laboratorios reales. En este contexto, para el laboratorio de la E3T resulta especialmente valiosa, esta solución, permite diseñar prácticas educativas adaptadas a modalidades a distancia, ampliando así las posibilidades de enseñanza y aprendizaje. Además, este prototipo establece una base sólida para futuros proyectos de transmisión y análisis de datos en tiempo real.

Aunque Node-RED no se diseñó para análisis espectral en tiempo real, el prototipo confirma que puede operar como un sistema ligero de monitoreo de señales de radio a través de internet, con muy poco hardware especializado. El ajuste del bloque LimitadorVentanas permitió adaptar la tasa de actualización a la percepción humana sin sobrecargar la interfaz, y la experiencia obtenida define usos concretos para cada protocolo: UDP para supervisión rápida, TCP cuando se requiere exactitud en los datos y HTTP/Google Apps Script para registro histórico. Estas pautas orientan futuras implementaciones según las necesidades de cada escenario.

6. Recomendaciones

Diseño del Panel de Navegación para Ubicaciones Específicas Para facilitar la selección de origen de datos, se propone un panel de navegación interactivo que muestre las ubicaciones disponibles y sus protocolos de transmisión asociados. Cada ubicación como la Antártida, Parque

Guatiguará y Páramo Berlín, se presenta como un bloque con tres opciones de envío de datos: TCP, UDP, Dual (TCP + UDP).

Al elegir una ubicación, Node-RED despliega dinámicamente el panel correspondiente y enruta la transmisión según el protocolo seleccionado. De esta forma, el usuario puede monitorear y comparar fácilmente el comportamiento de cada protocolo en función del entorno de operación. Este diseño busca optimizar la experiencia de usuario y la claridad en la gestión de transmisión de datos, al asociar cada ubicación con rutas específicas en Node-RED que garantizan una transmisión eficiente y organizada.

En futuros proyectos, se recomienda aprovechar los prototipos de mediciones meteorológicas desarrollados en proyectos de pregrado y maestría (Ignition, Node-RED con Davis), agregando el sistema de monitoreo del espectro radioeléctrico de este proyecto, de modo que ambos formen un único sistema IoT mucho más completo.

Implementar una estrategia que permita apagar y reiniciar el contenedor en Microsoft Azure sin perder su dirección IP, de modo que no sea necesario volver a cargar el archivo JSON.

Agregar un mecanismo de seguridad basado en roles que impida a usuarios no autorizados visualizar o modificar el Backend, restringiendo su acceso exclusivamente al Frontend de la interfaz de monitoreo. Este mecanismo protegerá la información frente a posibles ciberataques, reduciendo la probabilidad de que personas malintencionadas vean información importante que se pueda manejar.

Se sugiere crear una aplicación que sea compatible con el celular para que la persona pueda dar la orden de iniciar el sensando de cualquiera de los dos sistemas a larga distancia y visualizar de forma inmediata los resultados espectrales en tiempo real, de ser posible en esa misma aplicación podría tener un botón diseñado a la impresión de los últimos datos recibidos en un

tiempo determinado, por ejemplo, imprimir los resultados de los últimos tres días y que sean guardados en un archivo en la nube para luego ser consultado.

Referencias Bibliográficas

Advantages Of Cloud Computing. (n.d.). Google Cloud. Retrieved April 5, 2025, from https://cloud.google.com/learn/advantages-of-cloud-computing

Antoniou, Andreas. (2018). "Definition." Chap. 7.2 in Digital Filters: Analysis, Design, and Signal Processing Applications. 1st ed. New York: McGraw-Hill Education. https://www-accessengineeringlibrary-com.bibliotecavirtual.uis.edu.co/content/book/9780071846035/toc-chapter/chapter7/section/section3

Ballejos, L. (2024, June 10). What Are Web Services? Ninja One. Retrieved April 7, 2025, from https://www.ninjaone.com/it-hub/it-service-management/what-are-web-services/

Barrera, E. (n.d.). Azure - Creación contenedor Node-RED. YouTube. Retrieved April 29, 2025, from https://www.youtube.com/watch?v=EQ1v2kY2ggU

Bodnar, D. (2021, June 4). TCP/IP: ¿qué es el modelo TCP/IP y cómo funciona? AVG AntiVirus. Retrieved April 10, 2025, from https://www.avg.com/es/signal/what-is-tcp-ip

Características del HackRF One. (n.d.). https://docs.google.com/document/d/1qMSkixjg_aI10PJ-KaLGmGS7csVNCXQ3/edit

Cisco Networking Basics: IP Addressing - Cisco Networking Basics: IP Addressing. (2018, April 17). Network Computing. Retrieved May 22, 2025, from https://www.networkcomputing.com/ip-subnetting/cisco-networking-basics-ip-addressing

Cloudflare. (2025). ¿Qué es HTTP? Cloudflare. Retrieved April 10, 2025, from https://www.cloudflare.com/es-es/learning/ddos/glossary/hypertext-transfer-protocol-http/

Descripción general de Google Apps Script. (2024, December 21). Google for Developers. Retrieved April 5, 2025, from https://developers.google.com/apps-script/overview?hl=es-419

de Villegas, F. F. (2024, June). EMISORAS DE RADIO DEFINIDAS POR SOFTWARE-SDR-Software defined radio. EA1URO.COM. Retrieved April 5, 2025, from https://www.ea1uro.com/sdr.html

Embedded Python Block - GNU Radio. (2019, October 17). GNU Radio Wiki. Retrieved June 9, 2025, from https://wiki.gnuradio.org/index.php/Embedded_Python_Block

Float To Complex. (2025, April 10). GNURadio. Retrieved June 9, 2025, from https://wiki.gnuradio.org/index.php?title=Float_To_Complex

Frequency Bin. (2025). ANALOG DEVICES. Retrieved June 18, 2025, from https://www.analog.com/en/resources/glossary/frequency_bin.html

Frisoli, C. (2025, April 1). Qué es Frontend y Backend: guía con diferencias, ejemplos y herramientas. Blog de HubSpot. Retrieved May 22, 2025, from https://blog.hubspot.es/website/frontend-y-backend#que-es

HackRF One. (2009-2023). Great Scott Gadgets. Retrieved April 5, 2025, from https://greatscottgadgets.com/hackrf/one/

Hsu, Hwei P. 2020. “Correlations and Power Spectral Densities.” Chap. 9.2 in Schaum's Outline of Signals and Systems. 4th ed. New York: McGraw-Hill Education. https://www-accessengineeringlibrary-com.bibliotecavirtual.uis.edu.co/content/book/9781260454246/toc-chapter/chapter9/section/section3

Jain, S. (2025, July 12). Difference Between Hard Real Time and Soft Real Time System. GeeksforGeeks. Retrieved July 21, 2025, from https://www.geeksforgeeks.org/operating-systems/difference-between-hard-real-time-and-soft-real-time-system/

Joey. (2024, July 11). Industrial IoT Systems: Benefits, Essential Capabilities, and Best Practices. MQ. Retrieved April 5, 2025, from https://www.emqx.com/en/blog/industrial-iot-systems

Log Power FFT. (2025, April 14). GNU Radio. Retrieved June 9, 2025, from https://wiki.gnuradio.org/index.php/Log_Power_FFT

Los beneficios de IoT: Ejemplos del mundo real. (n.d.). Digi International. Retrieved April 5, 2025, from https://es.digi.com/blog/post/the-benefits-of-iot-real-world-examples

Metadatos, definición y características. (n.d.). PowerData. Retrieved May 25, 2025, from https://www.powerdata.es/metadatos

Muthuswamy, Jit. 2021. "SPECTRAL ANALYSIS OF DETERMINISTIC AND STATIONARY RANDOM SIGNALS." Chap. 23.3 in Biomedical Engineering Fundamentals. 3rd ed., edited by Myer Kutz. New York: McGraw Hill. https://www-accessengineeringlibrary-com.bibliotecavirtual.uis.edu.co/content/book/9781260136265/toc-chapter/chapter23/section/section10

Network Jitter - Common Causes and Best Solutions | IR. (n.d.). Integrated Research. Retrieved July 20, 2025, from https://www.ir.com/guides/what-is-network-jitter

Nodered/Node-RED - Docker Image. (n.d.). Docker Hub. Retrieved April 29, 2025, from https://hub.docker.com/r/nodered/node-red

Null Source. (2022, July 19). GNURadio. Retrieved June 9, 2025, from https://wiki.gnuradio.org/index.php?title=Null_Source

OpenJS, F. (2024, June 20). Node-RED. Node-RED: Low-code programming for event-driven applications. Retrieved April 5, 2025, from https://nodered.org

Options - GNU Radio. (2025, April 1). GNU Radio Wiki. Retrieved June 15, 2025, from https://wiki.gnuradio.org/index.php/Options

Ortega, H. (2024, May 19). Trabajo en Azure. Parte 2. Suscribirse a Azure como estudiante. Retrieved May 05, 2025, from https://www.youtube.com/watch?v=9w0Em-E7uUA

Pad Source. (2022, June 21). GNU Radio. Retrieved June 9, 2025, from https://wiki.gnuradio.org/index.php/Pad_Source

QT GUI Vector Sink - GNU Radio. (2020, April 30). GNU Radio Wiki. Retrieved June 9, 2025, from https://wiki.gnuradio.org/index.php/QT_GUI_Vector_Sink

¿Qué es el Espectro Radioeléctrico? | Generalidades. (n.d.). Portal Espectro Visible. Retrieved May 25, 2025, from https://portalespectro.ane.gov.co/Style%20Library/ane_master/que-es-el-espectro-radioelectrico.aspx

¿Qué es el Internet de las cosas (IoT)? (n.d.). IBM. Retrieved April 5, 2025, from https://www.ibm.com/mx-es/topics/internet-of-things

¿Qué es una WebApp? (n.d.). Imascono. Retrieved July 21, 2025, from https://imascono.com/que-es-una-webapp/

Qué es un sistema SCADA, cómo funciona y para qué sirve. (2021, May 28). SICMA21. Retrieved May 22, 2025, from https://www.sicma21.com/scada-que-es-y-como-funciona/

RadioGis, G. (2025, April 10). Generador de imágenes de OpenAI (ChatGPT) [Generación de imagen a partir de texto y modificada con Paint]. OpenAI. Retrieved April 10, 2025, from https://openai.com/chatgpt

Rojas, J. E. (2024, May 21). Microsoft Azure: Qué es, Funcionamiento y Para Qué Sirve. Raona. Retrieved April 10, 2025, from https://raona.com/que-es-para-que-sirve-microsoft-azure/

Sacco, I. L. (2024, 02 21). Arquitectura TCP/IP: La Espina Dorsal de la Conectividad en la Era Digital. PERITOS INFORMÁTICOS. Retrieved June 17, 2025, from https://peritosinformaticos.ar/arquitectura-tcp-ip/

Servidor NTP (Network Time Protocol) Servidor de tiempo - 2025. (2025). tecnozero. Retrieved July 20, 2025, from https://www.tecnozero.com/servidor/ntp/

Sockets. (n.d.). IBM. Retrieved May 22, 2025, from https://www.ibm.com/docs/es/aix/7.2.0?topic=concepts-sockets

TCP Sink. (2020, December 18). GNU Radio. Retrieved June 9, 2025, from https://wiki.gnuradio.org/index.php?title=TCP_Sink

Throttle. (2023, November 22). GNURadio. Retrieved June 9, 2025, from https://wiki.gnuradio.org/index.php?title=Throttle

UDP: Conozca la revolución silenciosa en la transmisión de datos. (2025, November 27). ITMASTERS MAG. Retrieved April 10, 2025, from https://www.itmastersmag.com/transformacion-digital/que-es-el-udp-y-para-que-sirve/

UDP Sink. (2023, February 23). GNU Radio. Retrieved June 9, 2025, from https://wiki.gnuradio.org/index.php?title=UDP_Sink

Una visualización del reloj Unix en tiempo real. (2025, January 10). Microsiervos. Retrieved June 27, 2025, from https://www.microsiervos.com/archivo/ordenadores/visualizacion-reloj-unix-tiempo-real-.html

Unix / Epoch Timestamp Conversion Tools. (2025). Epoch Converter - Unix Timestamp Converter. Retrieved June 27, 2025, from https://www.epochconverter.io/

Vector to Stream - GNU Radio. (2020, November 30). GNU Radio Wiki. Retrieved June 9, 2025, from https://wiki.gnuradio.org/index.php?title=Vector_to_Stream

Virtual Sink. (2022, June 20). GNU Radio. Retrieved June 9, 2025, from https://wiki.gnuradio.org/index.php?title=Virtual_Sink

Virtual Source. (2022, June 20). GNU Radio. Retrieved June 9, 2025, from https://wiki.gnuradio.org/index.php?title=Virtual_Source

Warnicke, E., & Lamping, U. (n.d.). Wireshark User's Guide. Wireshark. Retrieved April 7, 2025, from https://www.wireshark.org/docs/wsug_html_chunked/

Wav File Source. (2022, June 26). GNURadio. Retrieved Junio 9, 2025, from https://wiki.gnuradio.org/index.php?title=Wav_File_Source

What Is GNU Radio - GNU Radio. (2022, December 4). GNU Radio Wiki. Retrieved April 10, 2025, from https://wiki.gnuradio.org/index.php?title=What_Is_GNU_Radio