ULPGC Universidad de Las Palmas de Gran Canaria | eite ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO DE FIN DE GRADO

WiWiewCast. Sistema de streaming interactivo multiusuario con WebRTC utilizando Raspberry Pi y dispositivos móviles.

Titulación:	Grado en Ingeniería en Tecnologías de la Telecomunicación
Mención:	Sistema de Telecomunicaciones
Autor:	Benito Santana Díaz
Tutor:	Miguel Ángel Quintana Suárez
Fecha:	Julio 2025

Resumen

Este trabajo presenta un estudio exploratorio y aplicado sobre el diseño, desarrollo y evaluación de WiWiCast, un sistema de streaming interactivo multiusuario orientado a entornos educativos con recursos limitados. La iniciativa surge ante la necesidad de mejorar la visibilidad y la participación en laboratorios universitarios, integrando tecnologías web modernas como WebRTC, Node-RED y Socket.IO sobre una arquitectura accesible basada en Raspberry Pi. El objetivo principal consiste en ofrecer una solución flexible y de bajo coste que permita la transmisión de vídeo en tiempo real y herramientas colaborativas, como la pizarra interactiva, favoreciendo la inclusión y la experimentación docente.

Durante el desarrollo, se examinaron diferentes arquitecturas de transmisión (P2P, SFU, MCU) optando finalmente por un enfoque híbrido con SFU, que equilibra la escalabilidad, el consumo de ancho de banda y la baja latencia. El sistema se compone de varias capas: hardware económico (Raspberry Pi 5 y router dedicado), un ecosistema de software open source (Node.js, Node-RED, módulos WebRTC y Socket.IO), y una interfaz web diferenciada según los roles de emisor, espectador y administrador. Esta estructura promueve la colaboración y facilita el aprendizaje activo, permitiendo la adaptación sencilla a diversos escenarios educativos.

La evaluación en entornos reales mostró que WiWiewCast es capaz de superar las barreras tradicionales de visibilidad e interacción en el aula, demostrando su robustez en laboratorios con numerosos alumnos y recursos limitados. Se validaron diferentes modos de uso y configuración, destacando la mejora significativa en la interacción grupal y en la gestión docente, así como la accesibilidad del sistema tanto para estudiantes como para profesores. El análisis identificó puntos de mejora, especialmente en la gestión de autenticación, la ampliación de funcionalidades colaborativas y la integración de nuevas herramientas multimedia.

Finalmente, el trabajo plantea diversas líneas futuras para la evolución del sistema, entre ellas: el refuerzo de la seguridad y autenticación, la incorporación de grabación multicámara, la expansión de las capacidades de anotación y la creación de diferentes niveles de permisos. Se discute también el impacto de WiWiewCast en el proceso educativo, subrayando cómo una solución tecnológica adaptada puede contribuir a la equidad digital, a la mejora de las competencias TIC y al fomento de un aprendizaje colaborativo y personalizado en la educación superior.

Abstract

This work presents an exploratory study on the design, implementation, and evaluation of WiWiCast, an interactive, multi-user streaming system developed for educational environments with constrained resources. The project is motivated by the need to enhance both visibility and student participation in university laboratories, leveraging modern web technologies such as WebRTC, Node-RED, and Socket.IO. These are integrated into a cost-effective architecture centered on a Raspberry Pi serving as the central server within a dedicated local network. The primary goal is to provide a flexible, low-cost solution that enables real-time video streaming and interactive digital tools—including a collaborative whiteboard—to foster inclusive and hands-on learning experiences.

The development process involved a technical analysis of different streaming architectures (P2P, SFU, MCU), with the final system adopting a hybrid SFU approach to balance scalability, bandwidth usage, and low latency. The architecture is structured in layers: affordable hardware (Raspberry Pi and dedicated router), a modular open-source software stack (Node.js, Node-RED, WebRTC, and Socket.IO modules), and a web interface tailored to the roles of broadcaster, viewer, and administrator. This design facilitates collaborative learning and allows for easy adaptation to diverse educational scenarios.

Real-world testing demonstrated that WiWiewCast effectively addresses traditional classroom visibility and interaction challenges, proving robust in environments with many students and limited resources. Different usage modes and configurations were validated, highlighting significant improvements in group interaction, classroom management, and accessibility for both students and instructors. The evaluation also identified areas for future enhancement, particularly in authentication management, expanded collaborative features, and the integration of additional multimedia tools.

The study concludes by outlining future directions for system evolution, including enhanced security and authentication, multicamera recording capabilities, advanced annotation features, and differentiated user permissions. It also discusses the educational impact of WiWiCast, emphasizing how tailored technological solutions can promote digital equity, improve ICT competencies, and encourage collaborative, personalized learning in higher education settings. The project serves as a practical example of how advanced, interactive technologies can be made accessible and effective in resource-limited educational contexts.

Tabla de contenido

Índice de Figuras

Índice de Tablas

Capítulo 1. Introducción y Objetivos

1. Introducción

El desarrollo de las tecnologías web estos últimos años ha facilitado, entre otros avances, el acceso al conocimiento en entornos educativos, eliminando barreras tradicionales de tiempo y espacio. Esta evolución tecnológica ha modificado significativamente los métodos educativos tradicionales, permitiendo que el aprendizaje supere las limitaciones físicas del aula convencional. Este trabajo de fin de grado tiene como objetivo desarrollar un ecosistema integral, basado en estas tecnologías, que permita la transmisión de video y la interacción en tiempo real a nivel local, optimizado para el uso de dispositivos móviles. La solución propuesta se fundamenta en la implementación de un sistema basado en WebRTC (Web Real-Time Communication) [1] y el entorno de programación visual Node-RED [2], tecnologías que posibilitan la transmisión de video de baja latencia junto con la capacidad de realizar anotaciones sincronizadas sobre el contenido transmitido, aprovechando las capacidades de Node-RED para gestionar los flujos de comunicación.

La arquitectura del sistema se implementará sobre una Raspberry Pi, utilizando una red local dedicada a través de un router WiFi en modo punto de acceso (AP). Esto permitirá garantizar un entorno controlado y optimizado para el uso en aulas, donde los docentes podrán transmitir contenido audiovisual mientras los estudiantes realizan anotaciones en tiempo real, creando así un espacio de aprendizaje interactivo y colaborativo.

La motivación principal de este proyecto surge de la necesidad de crear soluciones educativas accesibles y universales. En este sentido, la propuesta se fundamenta en el uso de tecnologías web modernas que, junto con la implementación de un portal para la gestión de usuarios, garantizan el acceso desde cualquier dispositivo con un navegador web. Esta aproximación tecnológica elimina la dependencia de software especializado y facilita su adopción inmediata en diferentes contextos educativos.

2. Antecedentes

La transmisión de video en tiempo real en tecnologías web ha experimentado un avance notable durante la última década, impulsado principalmente por el desarrollo de WebRTC,

una tecnología que permite la comunicación directa entre navegadores sin necesidad de plugins como Flash, Silverlight o Java, eliminando así los problemas de compatibilidad que caracterizaban las soluciones anteriores.

Entre los principales logros, destaca la estandarización de WebRTC por organismos como el W3C (World Wide Web Consortium) y la IETF (Internet Engineering Task Force), lo que ha permitido su adopción generalizada. Además, se han implementado tecnologías de optimización de red como STUN (Session Traversal Utilities for NAT), TURN (Traversal Using Relays around NAT) e ICE (Interactive Connectivity Establishment) [3] [4], facilitando la conectividad peer-to-peer (P2P) en entornos educativos y colaborativos. Paralelamente, el ecosistema Node-RED ha evolucionado, incorporando nodos especializados en video y WebRTC, lo que amplía las posibilidades de integración y control de flujos multimedia.

Sin embargo, a pesar de estos avances, la implementación de sistemas de streaming interactivo eficientes enfrenta diversos desafíos técnicos que requieren soluciones específicas. Las limitaciones en la escalabilidad del modelo P2P cuando el número de usuarios crece hacen necesario evaluar arquitecturas alternativas como SFU (Selective Forwarding Unit) o Multicast. Además, la optimización del ancho de banda y la reducción de la latencia continúan siendo aspectos críticos para garantizar una experiencia educativa fluida y atractiva. Por otro lado, la gestión y visualización de flujos de comunicación complejos en sistemas WebRTC puede resultar complicada, especialmente cuando se requiere un control centralizado y una monitorización sencilla. Node-RED aporta ventajas significativas en este ámbito, gracias a su interfaz de programación visual que facilita la visualización, control y automatización de los procesos de comunicación.

En este contexto, el presente trabajo explora las diferentes alternativas y se centra en un enfoque híbrido basado en SFU y WebRTC, combinando la eficiencia y escalabilidad de la distribución de video con la interactividad y la baja latencia propias de WebRTC. Esta estrategia, junto con la integración de Node-RED, tiene como objetivo mejorar la eficiencia y la escalabilidad de las transmisiones interactivas en tiempo real, optimizar el uso del ancho de banda y facilitar la gestión educativa en el aula digital.

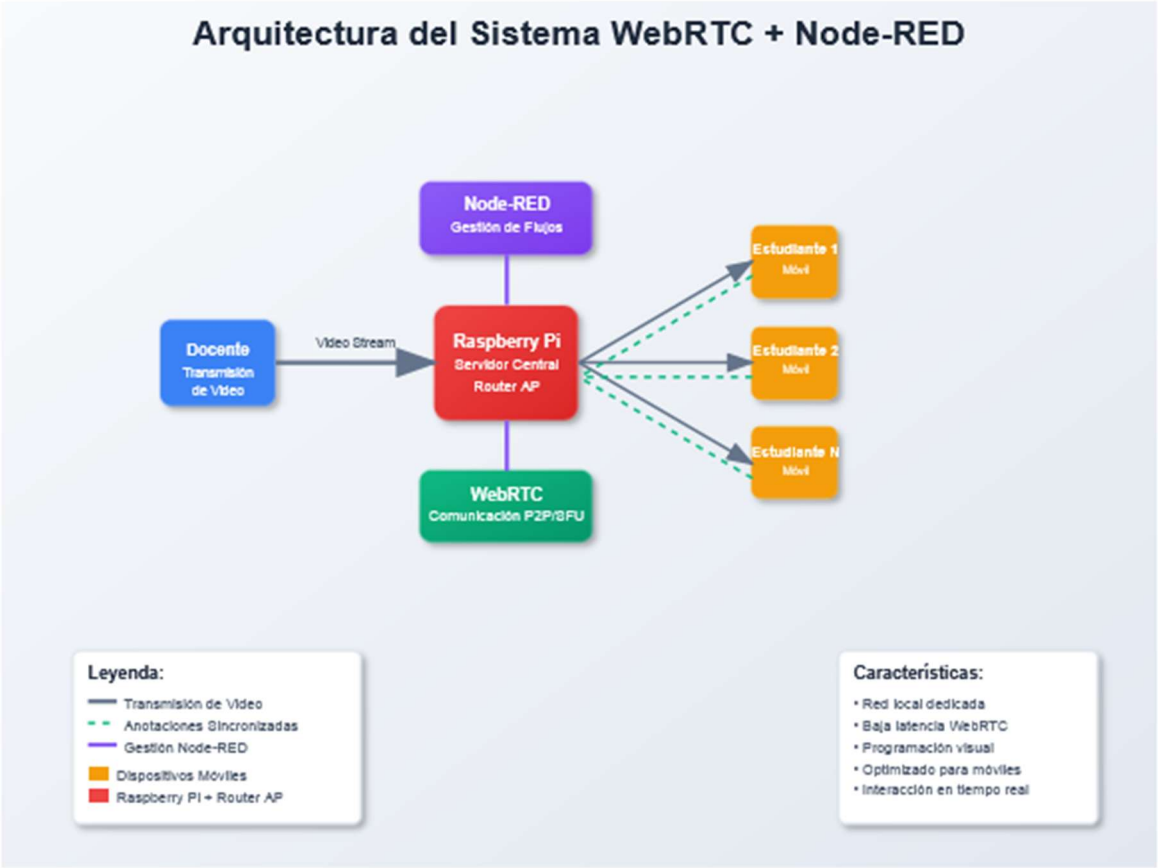


Figura 1. Arquitectura del Sistema WebRTC más Node-RED.

3. Objetivos

El análisis del estado del arte y el desarrollo de este trabajo persiguen una serie de objetivos fundamentales orientados a abordar tanto los desafíos técnicos como los pedagógicos que plantea la implementación de sistemas de streaming interactivo en entornos educativos. Estos objetivos buscan asegurar que la solución propuesta no solo sea técnicamente viable, sino que también responda a las necesidades reales de docentes y estudiantes en el contexto educativo actual.

Los objetivos específicos son:

- Comprender la arquitectura y el funcionamiento interno de WebRTC, con especial atención a la transmisión de vídeo en tiempo real y a los protocolos que garantizan la calidad, estabilidad y baja latencia de las conexiones. Este conocimiento es

esencial para ofrecer una experiencia educativa fluida y colaborativa, donde docentes y estudiantes puedan interactuar sin demoras ni interrupciones.

- Explorar la integración de WebRTC con Node-RED, analizando tanto las ventajas como los desafíos derivados de la combinación de ambas tecnologías. La programación visual que ofrece Node-RED facilita la gestión, visualización y control de los flujos multimedia, permitiendo que la administración de la información en el aula sea más intuitiva y accesible, fomentando la participación activa de los estudiantes.

- Evaluar diferentes arquitecturas de transmisión de vídeo, considerando aspectos clave como la latencia, la escalabilidad, el consumo de ancho de banda y la carga sobre servidores o clientes. Esta evaluación permitirá identificar la solución más adecuada a las necesidades educativas y a las limitaciones técnicas del entorno de implementación, asegurando que el sistema sea adaptable y escalable.

- Determinar la arquitectura óptima para un sistema de streaming interactivo multiusuario, adaptada a las características de hardware y red del entorno educativo, y que permita tanto la transmisión audiovisual como la interacción de datos en tiempo real. El objetivo es que los estudiantes puedan no solo visualizar contenido, sino también interactuar activamente mediante herramientas como el dibujo colaborativo, promoviendo el aprendizaje colaborativo y la participación directa en el proceso educativo.

4. Estructura del documento

Esta memoria se organiza en cinco capítulos, cada uno de los cuales aborda un aspecto esencial para el diseño, desarrollo e implementación del sistema de streaming interactivo basado en WebRTC y Node-RED, con énfasis en la interacción colaborativa y la gestión de datos en tiempo real.

En el capítulo 1, "Introducción y objetivos", se presenta el contexto tecnológico y la motivación del proyecto, exponiendo los antecedentes de WebRTC y Node-RED, así como los desafíos actuales en la transmisión de video interactiva y escalabilidad en entornos educativos. Se definen cuatro objetivos específicos que abarcan desde la comprensión de la arquitectura WebRTC hasta la determinación de una solución óptima para streaming

interactivo multiusuario. Este capítulo establece el marco conceptual y justifica la relevancia de la propuesta basada en un enfoque híbrido SFU-WebRTC con Node-RED.

El capítulo 2, "Fundamentos técnicos", desarrolla los principios y tecnologías clave del proyecto. Se analizan en profundidad WebRTC y Node-RED, explicando sus componentes, funcionamiento y ventajas para la transmisión de video y la gestión de flujos de datos interactivos. Además, se revisan las diferentes arquitecturas de transmisión (P2P, SFU, Multicast, híbridas) [5], evaluando su idoneidad para escenarios multiusuario y colaborativos. También se introducen los conceptos de interacción de datos y dibujo colaborativo como parte integral del sistema.

En el capítulo 3, "Instalación y configuración del sistema", se detalla el proceso de preparación del entorno de desarrollo, incluyendo la selección y configuración del hardware (como Raspberry Pi) y del software necesario. Se describen los pasos para instalar y configurar WebRTC, Node-RED y los módulos específicos para la transmisión de video y la interacción de datos. Se documentan las consideraciones de red y seguridad para garantizar un entorno estable y controlado en el aula.

El capítulo 4, "Especificaciones de la aplicación y arquitectura del sistema", define los requisitos funcionales y no funcionales del sistema, abarcando tanto la transmisión de video como las funcionalidades de colaboración (por ejemplo, la pizarra digital o el dibujo compartido en tiempo real). Se presenta el diseño detallado, los diagramas de flujo de datos y la estructura de los módulos, justificando las decisiones tomadas con base en los objetivos pedagógicos y técnicos del proyecto.

En el capítulo 5, "Evaluación y Conclusiones", presenta la metodología de pruebas, los escenarios de uso y los resultados obtenidos tras la implementación del sistema. Se analizan métricas clave como la latencia, la escalabilidad, y la robustez de la interacción colaborativa. Se exponen las conclusiones, el grado de cumplimiento de los objetivos y se proponen líneas de trabajo futuro para la mejora y ampliación del sistema.

5. Uso de IA Generativas

En cumplimiento de las "Recomendaciones sobre uso de la IAGen en la UPGC" ,aprobadas por el Consejo de Gobierno Extraordinario de la ULPGC el 6 de junio de 2024, este TFG ha empleado inteligencia artificial generativa como herramienta de apoyo,

usándose de manera equivalente a un navegador web para contrastar fuentes de información, generar ideas y proponer soluciones.

Capítulo 2. Fundamentos técnicos

1. Introducción

El desarrollo del sistema de *streaming* interactivo propuesto en este proyecto requiere la integración de múltiples tecnologías que trabajan de manera coordinada para proporcionar transmisión de video, interacción en tiempo real y gestión de usuarios en entornos educativos. La solución se implementa utilizando JavaScript como lenguaje de programación unificado, aprovechando su capacidad para ejecutarse tanto en el navegador (cliente) como en el servidor, lo que simplifica significativamente el desarrollo y mantenimiento del sistema.

Este capítulo examina los fundamentos técnicos de cada componente software del ecosistema:

- **WebRTC** para comunicación multimedia,
- **Node.js** [6] como entorno de ejecución,
- **Socket.io** [7] para comunicación bidireccional,
- **Axios** [8] para integraciones HTTP, y
- **Node-RED** para orquestación visual de flujos de datos.

También se recogen las características específicas del hardware seleccionado, incluyendo la Raspberry Pi [9] como servidor central y el router WiFi configurado en modo punto de acceso para crear la red local dedicada.

Entender estas tecnologías es fundamental para comprender las decisiones de arquitectura tomadas en el proyecto y para evaluar las diferentes alternativas de transmisión de video (P2P, SFU, MCU), que se analizarán posteriormente. El enfoque adoptado busca equilibrar el nivel técnico necesario y la aplicación práctica dentro del entorno específico de este trabajo.

2. WebRTC y sus componentes clave

WebRTC representa un conjunto de estándares y tecnologías que permite la comunicación en tiempo real directamente entre navegadores web, sin necesidad de *plugins* adicionales o software especializado. Esta tecnología, estandarizada por el W3C (*World Wide Web Consortium*) y la IETF (*Internet Engineering Task Force*), ha revolucionado la forma en la que se implementan las aplicaciones de comunicación multimedia en la web, proporcionando una base sólida para el desarrollo de sistemas interactivos y colaborativos.

La importancia de WebRTC en el contexto educativo radica en su capacidad para acercar las tecnologías de comunicación avanzadas a todo tipo de usuario, eliminando las barreras tradicionales de instalación de software y configuración compleja. Su implementación nativa en navegadores modernos garantiza una compatibilidad casi universal y permite su uso inmediato en cualquier dispositivo con acceso web, desde ordenadores hasta móviles con pocos recursos. El diagrama general de WebRTC puede verse en Figura 2 .

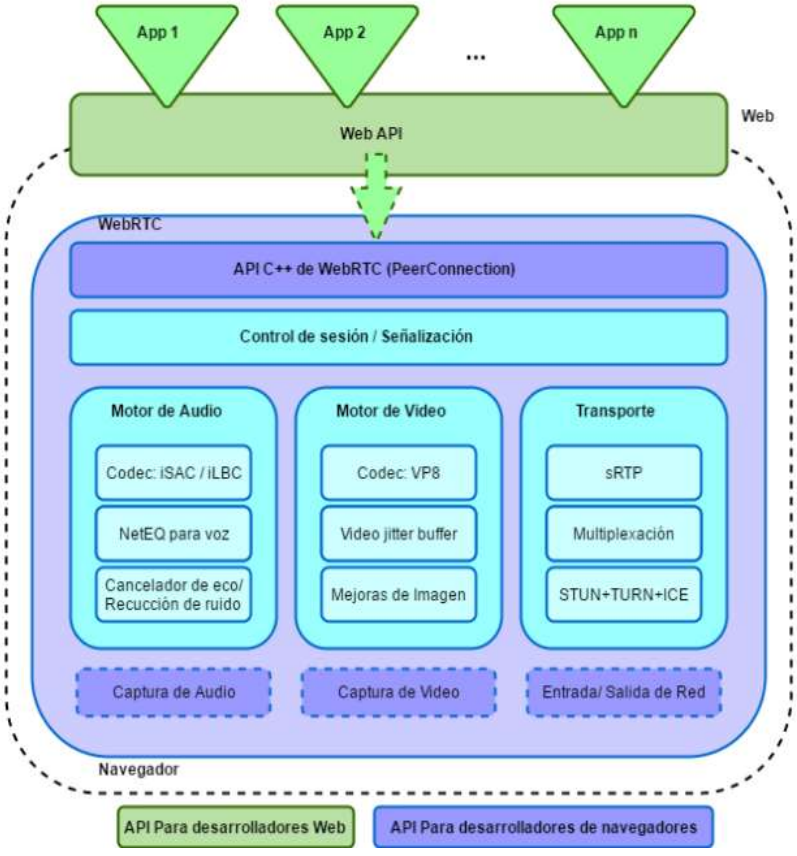


Figura 2. Diagrama general de WebRTC. Fuente [4].

2.1. Arquitectura y componentes principales

La arquitectura de WebRTC se fundamenta en tres APIs principales que trabajan de forma coordinada para facilitar la comunicación multimedia, cada una especializada en aspectos específicos del proceso de transmisión:

- **RTCPeerConnection**

Constituye el núcleo técnico de WebRTC, implementando la lógica compleja de establecimiento y gestión de conexiones *peer-to-peer*. Esta API coordina múltiples procesos simultáneamente, incluyendo la negociación inicial de capacidades entre pares mediante el intercambio de *Session Description Protocol* (SDP), el establecimiento de canales seguros de comunicación utilizando *Datagram Transport Layer Security* (DTLS), la adaptación dinámica de calidad según condiciones de red variables mediante técnicas como *Adaptive Bitrate* (ABR), y la implementación de mecanismos robustos de recuperación ante fallos de conectividad. La API gestiona automáticamente aspectos complejos como el reordenamiento de paquetes, la detección y corrección de errores, y la sincronización temporal entre streams de audio y vídeo.

- **MediaStream (getUserMedia API)**

Permite la captura y manipulación de contenido multimedia, como son el acceso a la cámara y el micrófono del dispositivo. Esta API proporciona control sobre los recursos multimedia del dispositivo, permitiendo configurar parámetros críticos como resolución de vídeo (desde 320x240 hasta 1920x1080 o superior), tasa de *frames* (15, 24, 30 fps), calidad de audio (8kHz a 48kHz), y selección específica del dispositivo (cuando múltiples fuentes están disponibles). Esta flexibilidad es fundamental para optimizar una adecuada experiencia de usuario, según las capacidades del hardware disponible, garantizando que dispositivos menos potentes puedan participar adecuadamente en sesiones interactivas mediante configuraciones adaptadas a sus limitaciones.

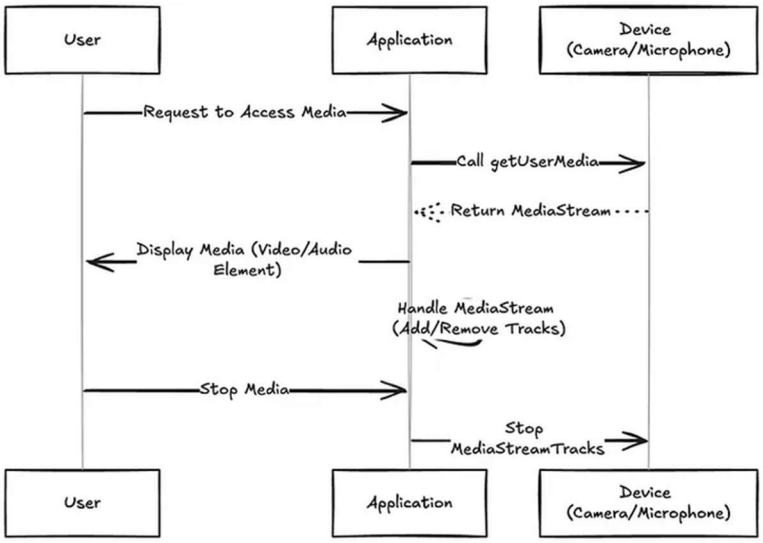


Figura 3. Proceso de captura y uso de transmisiones multimedia en una aplicación WebRTC. Fuente [10]

- **RTCDataChannel**

Extiende las capacidades de WebRTC más allá de la transmisión multimedia, proporcionando canales bidireccionales de baja latencia para el intercambio de otros datos entre pares conectados. Según las necesidades específicas de la aplicación, estos canales pueden configurarse como confiables (asegurando la entrega ordenada de los datos mediante mecanismos similares a los del protocolo TCP) o no confiables (priorizando velocidad mediante mecanismos similares a los utilizados en el protocolo UDP). RTCDataChannel facilita la implementación de características colaborativas avanzadas como anotaciones sincronizadas en tiempo real, pizarras digitales compartidas con resolución de conflictos, sistemas integrados de mensajería instantánea, transferencia de archivos entre participantes, y sincronización de estados de aplicación para mantener coherencia entre múltiples usuarios.

2.2. Protocolos de conectividad y optimización de red

WebRTC utiliza un conjunto de protocolos especializados para asegurar una conectividad robusta y eficiente, incluso en redes complejas y heterogéneas. Estos protocolos son esenciales para aplicaciones WebRTC que funcionan a través de Internet y deben superar barreras como NAT y firewalls. En la Figura 4 puede verse el esquema general que se utiliza para la localización de otros equipos. En este proyecto la comunicación se realiza sobre una red local dedicada, configurada mediante un router WiFi en modo punto de acceso. En este entorno controlado, muchos de los mecanismos destinadas a superar esos obstáculos no resultan necesarios. No obstante, es relevante comprender cómo funcionan

estos protocolos para valorar las ventajas y simplificaciones que aporta la arquitectura de red local implementada.

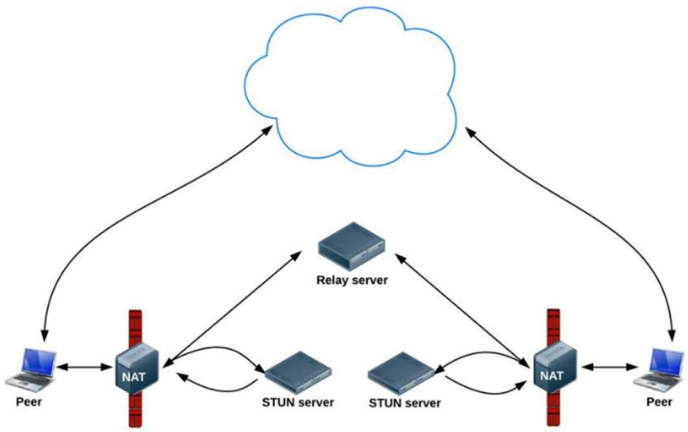


Figura 4. Localización de candidatos para una conexión WebRTC. Fuente [11].

- **STUN (Session Traversal Utilities for NAT)**

Este protocolo permite a los dispositivos descubrir automáticamente su dirección IP pública y caracterizar el comportamiento del NAT que está utilizando su router o firewall. STUN opera mediante un servidor externo que refleja la dirección y puerto desde los cuales recibe peticiones, proporcionando información crucial sobre la topología de red. Esta información incluye el tipo específico de NAT *(Full Cone, Restricted Cone, Port Restricted, o Symmetric*), la dirección IP pública asignada, y los puertos disponibles para comunicación externa. En nuestro sistema con una red local dedicada, los dispositivos pueden descubrir directamente sus direcciones IP locales sin necesidad de servidores STUN externos, simplificando significativamente el proceso de establecimiento de conexiones.

- **TURN (Traversal Using Relays around NAT)**

Es un mecanismo de respaldo robusto cuando las conexiones directas P2P no son viables debido a configuraciones restrictivas de seguridad de red. El servidor TURN actúa como un proxy multimedia inteligente, recibiendo y retransmitiendo todo el tráfico entre pares que no pueden conectarse directamente. TURN implementa mecanismos de autorización para prevenir uso no autorizado, gestión eficiente de recursos para minimizar latencia adicional, y balanceado de carga para distribuir el tráfico entre múltiples servidores cuando el volumen lo requiere. Al operar en una red local controlada, nuestro proyecto elimina la necesidad de servidores TURN, ya que todos los dispositivos están en el mismo segmento de red y pueden establecer conexiones directas, reduciendo latencia y eliminando dependencias de infraestructura externa.

- **ICE (Interactive Connectivity Establishment)**

Es un protocolo que coordina el uso de STUN y TURN para encontrar la mejor forma de conectar dos dispositivos en red. Su función principal es probar diferentes caminos de conexión y seleccionar el más eficiente. Para ello, ICE recopila varias combinaciones posibles de direcciones IP y puertos, llamadas *candidatos ICE*, y realiza pruebas simultáneas para ver cuál funciona mejor. Estos candidatos pueden ser locales, públicos (obtenidos mediante STUN) o de retransmisión (mediante TURN). Luego, el protocolo verifica la conectividad en ambos sentidos y elige el par de candidatos con mejor rendimiento. En nuestro caso, como el sistema opera dentro de una red local dedicada, ICE se simplifica considerablemente, ya que solo necesita usar candidatos locales. Esto reduce la complejidad y permite establecer la conexión de manera más rápida y predecible.

- **DTLS (Datagram Transport Layer Security)**

Es el protocolo de seguridad utilizado en WebRTC para cifrar toda la comunicación de extremo a extremo, asegurando la confidencialidad e integridad de los datos sin afectar el rendimiento en tiempo real. Está diseñado específicamente para entornos donde es crucial la baja latencia, e incluye procesos de negociación optimizados (*handshakes*) y una gestión eficiente de claves de cifrado. Aunque en este proyecto el sistema opera en una red local, DTLS sigue siendo necesario, ya que la protección de las comunicaciones es fundamental e independiente del tipo de red utilizada.

2.3. Códecs y optimización multimedia

WebRTC ofrece soporte nativo para una amplia gama de códecs de audio y video de última generación, integrando mecanismos de selección automática y adaptación dinámica según las capacidades del dispositivo, las condiciones de red y los requisitos específicos de la aplicación.

- **Vídeo**

En el caso del video, la suite incluye códecs como VP8, VP9, H.264 (en sus perfiles baseline, main y high), AV1 y H.265, cada uno optimizado para distintos escenarios. VP9 destaca por su alta eficiencia de compresión, especialmente útil para contenido estático como presentaciones o pantallas compartidas, mientras que H.264 garantiza una compatibilidad amplia y aceleración por hardware en la mayoría de los dispositivos móviles. Por su parte, AV1 representa el estado del arte en compresión eficiente, ideal para

conexiones con ancho de banda limitado. La selección del códec más adecuado se realiza automáticamente teniendo en cuenta factores como el hardware disponible, el ancho de banda detectado, el tipo de contenido (por ejemplo, video de cámara frente a pantalla compartida) y las prioridades definidas por la aplicación.

Para adaptarse a variaciones en la calidad de la red, WebRTC incorpora técnicas como *Simulcast* (transmisión simultánea de múltiples resoluciones) *y Scalable Video Coding* (SVC), que permiten ajustar dinámicamente la calidad del video transmitido. Estas estrategias permiten una degradación gradual de la calidad visual en situaciones adversas, manteniendo la continuidad de la transmisión y priorizando la comprensibilidad del contenido. En los casos donde los recursos son limitados, se da preferencia a preservar el audio frente al video.

- **Audio**

En cuanto al audio, WebRTC incluye códecs como Opus (altamente optimizado para voz y música), **G.711** (para compatibilidad con sistemas antiguos) y G.722 (que ofrece mayor calidad). Además, se emplean técnicas avanzadas de procesamiento digital de señales, como la cancelación automática de eco (AEC), la reducción de ruido (NS) y el control automático de ganancia (AGC). Estas funciones están orientadas a mejorar la inteligibilidad del audio, especialmente en entornos como aulas, donde pueden coexistir múltiples fuentes de sonido y distintos dispositivos de entrada. No obstante, en el sistema desarrollado para este proyecto, el componente de audio no es relevante, ya que la comunicación se limita a la transmisión de video y a la interacción a través del dibujo colaborativo.

3. Node.js

Node.js [6] es un entorno de ejecución (*runtime environment*) de JavaScript de código abierto y multiplataforma que permite ejecutar código JavaScript fuera del navegador, es decir, en el servidor. Esto ha revolucionado el desarrollo de aplicaciones web, ya que antes JavaScript solo podía usarse en el lado del cliente, pero con Node.js se pueden crear aplicaciones de red rápidas y escalables usando el mismo lenguaje tanto en el cliente como en el servidor. Su funcionamiento se basa en reaccionar a eventos (*event-driven*) y en no quedarse esperando a que una tarea termine para empezar otra (*non-blocking I/O*), lo que lo hace muy eficiente para aplicaciones que necesitan atender a muchos usuarios al mismo tiempo y responder rápidamente. Por eso, Node.js es especialmente útil para desarrollar plataformas interactivas en tiempo real, como chats, juegos online o sistemas de

colaboración, donde es fundamental manejar múltiples conexiones simultáneas sin perder rendimiento.

3.1. Arquitectura y características fundamentales

Node.js está construido sobre el motor V8 de Google Chrome, que es el encargado de ejecutar JavaScript de manera muy rápida. Una de las cosas más importantes de Node.js es cómo gestiona muchas tareas al mismo tiempo sin necesidad de crear un hilo (*thread*) para cada una. Esto lo consigue gracias a un sistema llamado bucle de eventos (*event loop*), que es un coordinador que va atendiendo las tareas una a una y decide cuándo puede pasar a la siguiente.

Cuando Node.js necesita hacer algo que puede tardar, como leer un archivo o consultar una base de datos, no se queda esperando a que termine. En vez de eso, deja esa tarea en segundo plano y sigue atendiendo otras cosas. Cuando la tarea larga termina, Node.js recibe un aviso y ejecuta la función que estaba esperando ese resultado. Esta asincronía es posible gracias al *event loop* y al uso de funciones especiales como *callbacks*, promesas (*promises*) y *async/await*

Esta forma de trabajar hace que Node.js sea ideal para aplicaciones que tienen que atender a muchos usuarios a la vez y responder rápido, como chats, juegos online o servicios de *streaming*, y en nuestro caso, peticiones de diferentes *viewers* para dibujar a la vez. Nuestro desarrollo debe poder manejar diferentes conexiones sin volverse lento ni bloquearse.

Además, Node.js tiene un sistema que gestiona la memoria de manera automática (recolector de basura) y herramientas para supervisar el rendimiento y detectar problemas, lo que es muy útil para aplicaciones que deben estar funcionando durante mucho tiempo sin interrupciones.

3.2. NPM y el ecosistema de paquetes

NPM (*Node Package Manager*) es una herramienta que viene con Node.js y sirve para instalar y gestionar librerías o “paquetes” que otros programadores han creado y compartido. Esto hace que no tengas que programar todo desde cero, sino que puedas aprovechar soluciones ya hechas para sumar funciones a tu proyecto, como por ejemplo WebRTC, Socket.io o Axios, todas utilizadas en este proyecto.

Con NPM puedes instalar, actualizar o eliminar fácilmente estas librerías, y también te ayuda a asegurarte de que todas las dependencias de tu proyecto (es decir, los paquetes que necesita para funcionar) estén bien organizadas y sean compatibles entre sí. Todo esto se controla a través de un archivo llamado `package.json`, donde se guarda la lista de dependencias y sus versiones. Además, NPM utiliza un sistema llamado "versionado semántico" (*semantic versioning*) para gestionar las versiones de los paquetes, asegurando así la actualización de las librerías de forma segura, sabiendo que los cambios importantes no romperán el proyecto, algo muy importante para mantener la estabilidad de cualquier aplicación.

4. Socket.io

Socket.io [7] es una librería de JavaScript que permite que el servidor y los usuarios (clientes) se comuniquen entre sí en tiempo real, es decir, que los mensajes y actualizaciones lleguen al instante, sin necesidad de recargar la página o estar haciendo peticiones constantes. Socket.io usa principalmente una tecnología llamada WebSocket, pero si por algún motivo no está disponible, puede cambiar automáticamente a otros métodos como "*polling*" HTTP para asegurar que la conexión siga funcionando.

Otra característica importante es la gestión de conexiones. Si un usuario pierde la conexión durante en algún instante, Socket.io puede reconectarlo automáticamente y mantener el estado de la sesión. De esta manera, las interrupciones temporales de la red no afectan a la experiencia de usuario y todo sigue sincronizado para los usuarios. Además, Socket.io permite organizar a los usuarios en "salas" o "espacios" separados, conocidos como *rooms* y *namespaces*. Esto es útil para dividir a los participantes en grupos, clases o equipos, y así mantener conversaciones o actividades independientes dentro de la misma aplicación.

En cuanto a los usos en aplicaciones, una de las funciones más destacadas es la sincronización en tiempo real. Por ejemplo, en nuestro caso, cada vez que alguien dibuja, todos los demás ven el cambio al instante en el panel de administración. También permite definir eventos personalizados para distintas acciones, como cuando un estudiante dibuja algo "políticamente incorrecto", el profesor puede borrar solo su dibujo.

Por último, Socket.io puede configurarse para funcionar en varios servidores al mismo tiempo (*clustering*), lo que permite que aplicaciones con muchos usuarios conectados simultáneamente sigan funcionando sin problemas. Es decir, facilita la creación de

aplicaciones donde la información debe viajar rápido y sin retrasos entre todos los usuarios, como chats, pizarras colaborativas o juegos multijugador.

5. Axios

Axios [8] es una librería que facilita el envío y recepción de solicitudes HTTP, es decir, la comunicación entre una aplicación y servicios web o APIs. Está basada en promesas (*promises*), lo que permite manejar de forma sencilla las respuestas asíncronas, tanto en navegadores como en aplicaciones Node.js.

Una de las ventajas principales de Axios es que permite configurar interceptores (*interceptors*), que son funciones que se ejecutan automáticamente antes de enviar una petición o después de recibir una respuesta. Esto ayuda a implementar funcionalidades comunes como la autenticación automática, el registro de operaciones (*logging*), el manejo centralizado de errores y la transformación de datos, todo de forma transparente para el desarrollador.

Axios también ofrece un sistema de manejo de errores más completo que el método nativo `fetch`. Puede distinguir entre diferentes tipos de errores, como problemas de red, errores del servidor, tiempos de espera (*timeouts*) o errores de validación. Esto es especialmente útil en aplicaciones donde se requiere una respuesta adecuada según el tipo de fallo.

Otra característica importante es la posibilidad de cancelar peticiones (*request cancellation*) usando tokens especiales. Esto evita problemas cuando, por ejemplo, un usuario realiza varias acciones rápidas que generan múltiples solicitudes, y algunas de ellas quedan obsoletas antes de completarse. Axios facilita la integración con sistemas externos como plataformas de gestión de aprendizaje (LMS), bases de datos de estudiantes o servicios de calificaciones, ofreciendo una forma consistente y segura de comunicarse con estas APIs.

Además, gracias a los interceptores, es posible implementar mecanismos de autenticación automática con tokens JWT, renovar estos tokens sin que el usuario lo note, y cerrar sesión automáticamente cuando las credenciales expiran, manteniendo la seguridad sin afectar la experiencia del usuario.

Esta librería permite configurar opciones para optimizar el rendimiento, como establecer tiempos máximos de espera (*timeouts*), reintentos automáticos en caso de fallos, y

reutilización de conexiones (*connection pooling*). Estas características son clave en entornos donde la calidad de la red puede variar.

Axios es una herramienta potente y sencilla que ayuda a manejar las comunicaciones HTTP de manera eficiente, segura y adaptable a las necesidades específicas de aplicaciones modernas, incluyendo las educativas. En este proyecto Axios se utiliza en los dispositivos clientes para hacer peticiones HTTP.

6. Node-RED

Node-RED [2] es una herramienta de programación visual que permite crear aplicaciones conectando bloques llamados nodos, sin necesidad de escribir mucho código. Estos nodos se pueden arrastrar y soltar desde un panel en el navegador, y luego se conectan entre sí para definir cómo fluye la información entre ellos, los servicios web y las APIs. Gracias a este enfoque, Node-RED hace que la programación sea más accesible y sencilla, incluso para quienes no son expertos en desarrollo de software, ver Figura 5.

La arquitectura de Node-RED se basa en el concepto de flujos de datos: cada nodo realiza una función específica, y las conexiones entre nodos marcan el camino que siguen los datos a lo largo del sistema. Aunque Node-RED está construido sobre Node.js y aprovecha todas sus ventajas, añade una capa visual que facilita mucho la comprensión y el mantenimiento de sistemas complejos. Además, permite integrar fácilmente cualquier paquete de Node.js como un nodo personalizado, lo que significa que librerías como Socket.io, Axios o WebRTC pueden usarse dentro de los flujos visuales para sumar funcionalidades avanzadas.

El ecosistema de Node-RED incluye nodos especializados para diferentes tareas. Por ejemplo, hay nodos para gestionar conexiones WebRTC, que simplifican la creación de videollamadas o la transmisión de datos en tiempo real, y pueden trabajar junto con servidores Socket.io para coordinar conexiones entre usuarios. También existen nodos HTTP y WebSocket para conectar con APIs y servicios web, y nodos de procesamiento de vídeo que permiten manipular imágenes, aplicar filtros o adaptar los vídeos a las capacidades de distintos dispositivos.

En la gestión de sistemas, Node-RED ofrece ventajas como la monitorización visual en tiempo real del estado del sistema y de las conexiones activas. Esto facilita detectar problemas rápidamente y entender cómo se comporta la aplicación. Además, permite

modificar los flujos de trabajo mientras el sistema está funcionando, sin necesidad de reiniciar servicios ni interrumpir la actividad de los usuarios.

Por último, Node-RED es muy flexible y escalable: facilita añadir nuevas funciones o integrar servicios externos mediante la creación de nodos personalizados, manteniendo siempre la simplicidad visual y aprovechando toda la potencia del ecosistema Node.js. Por todo esto, Node-RED se ha convertido en una herramienta clave tanto en la industria como en proyectos educativos, domótica e Internet de las Cosas.

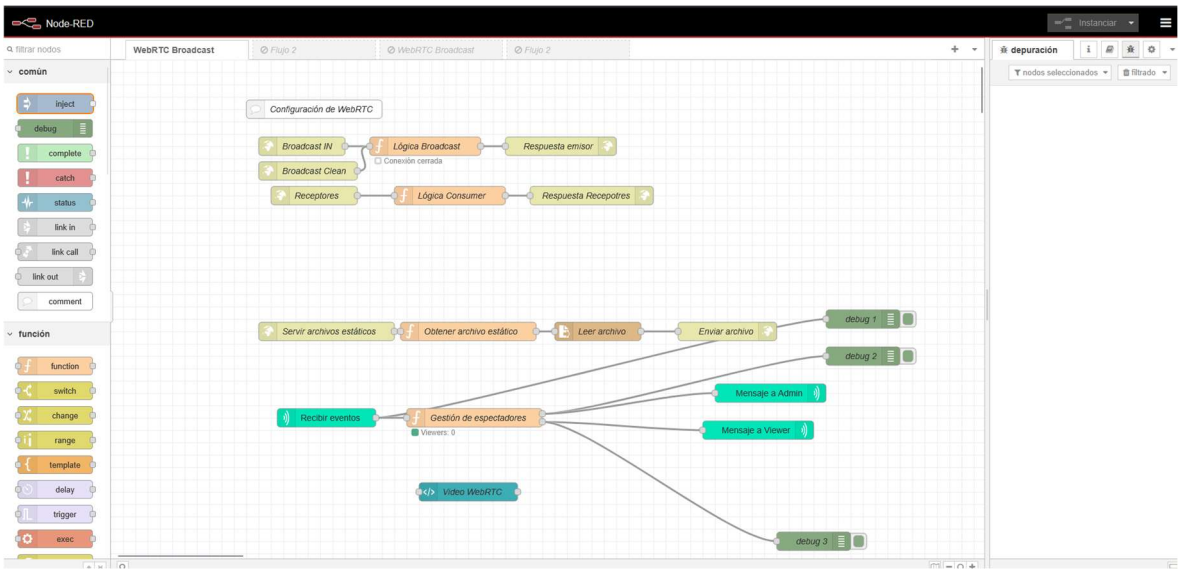


Figura 5. Ventana de ejemplo de Node-RED.

7. Arquitecturas de transmisión de vídeo

La arquitectura de transmisión de vídeo es clave para el rendimiento y la capacidad de crecimiento de cualquier sistema educativo basado en *streaming*. Cada tipo de arquitectura tiene sus propias ventajas y limitaciones, por lo que es importante analizar cuál se adapta mejor a cada situación.

En la **arquitectura P2P**, cada participante se conecta directamente con los demás, formando una red donde todos se comunican sin intermediarios. Esto permite una latencia muy baja y una alta calidad de transmisión, ya que no hay servidores centrales que recodifiquen el vídeo ni gestionen todo el tráfico. Sin embargo, este modelo tiene un gran inconveniente: a medida que aumenta el número de participantes, también crece de manera exponencial el número de conexiones y el consumo de ancho de banda, lo que limita seriamente su escalabilidad. Por eso, el P2P es ideal para grupos pequeños, como

tutorías individuales o equipos de trabajo reducidos, donde la interacción directa y la rapidez son más importantes que la cantidad de usuarios.

Por otro lado, la **arquitectura SFU** (*Selective Forwarding Unit*) utiliza un servidor central que recibe los vídeos de todos los participantes y los reenvía selectivamente a quienes los necesitan. El servidor SFU no modifica ni recodifica los vídeos, solo los distribuye, lo que reduce la carga de procesamiento y mantiene la latencia baja. Cada usuario envía su vídeo solo una vez al servidor, que luego se encarga de repartirlo a los demás. Esto mejora mucho la escalabilidad y permite adaptar la calidad de cada transmisión según la conexión y el dispositivo de cada usuario. Además, facilita funciones como la grabación de sesiones, la moderación y la obtención de estadísticas. En entornos educativos, la SFU permite, por ejemplo, que el profesor transmita en alta calidad mientras los estudiantes lo hacen en resoluciones más bajas, optimizando así los recursos disponibles.

La arquitectura MCU (*Multipoint Control Unit*) es diferente, ya que el servidor central recibe todos los vídeos, los decodifica, los combina en una sola imagen y luego envía ese resultado a todos los participantes. Este proceso requiere mucho más procesamiento y puede introducir algo de retraso, pero ofrece un control total sobre cómo se ve la imagen final. Es especialmente útil en situaciones donde se necesita una presentación unificada, como conferencias con varios ponentes, evaluaciones en las que se debe ver al estudiante y su trabajo al mismo tiempo, o grabaciones con un diseño visual profesional.

Finalmente, existen **arquitecturas híbridas** y adaptativas que combinan las ventajas de los modelos anteriores. Por ejemplo, se pueden usar conexiones P2P para subgrupos pequeños y SFU para la transmisión general, o emplear MCU solo para ciertos elementos visuales mientras el resto de los vídeos se distribuyen mediante SFU. Algunos sistemas pueden cambiar automáticamente de arquitectura según el número de usuarios, la calidad de la red o el tipo de actividad educativa que se esté realizando.

Además de las arquitecturas tradicionales como P2P, SFU y MCU, existe la posibilidad de implementar un sistema de transmisión *multicast* en colaboración con un router específico que soporte este tipo de tráfico. En una red *multicast*, el servidor envía un solo flujo de vídeo a una dirección IP especial, y solo los dispositivos que estén interesados en recibir ese contenido se suscriben a ese grupo *multicast*. Esto permite que, independientemente del número de usuarios conectados, el servidor mantenga la misma carga y el consumo de ancho de banda no aumente con cada nuevo receptor, a diferencia de lo que sucede en *unicast* o en arquitecturas P2P.

Para que el *multicast* funcione correctamente, es fundamental que tanto el router como los switches de la red sean compatibles y estén configurados para gestionar tráfico *multicast*, utilizando protocolos como IGMP *Snooping* en IPv4 o MLD en IPv6. Este enfoque es especialmente eficiente en entornos educativos o institucionales donde muchos usuarios necesitan recibir el mismo contenido en tiempo real, ya que reduce la saturación de la red y optimiza el uso de los recursos disponibles. Sin embargo, su implementación requiere una infraestructura de red adecuada y un control centralizado sobre los dispositivos conectados.

Elegir la arquitectura adecuada permite optimizar tanto la calidad de la experiencia como el uso de los recursos, asegurando que el sistema pueda adaptarse a diferentes tamaños de grupo, necesidades pedagógicas y condiciones técnicas.

8. Características del hardware: Raspberry Pi y red local

La implementación del sistema propuesto se apoya en una arquitectura de hardware accesible y potente, basada en una Raspberry Pi 5 [9] (8GB RAM) como servidor central y un Router WiFi: ASUS Wireless-AC2900 configurado en modo punto de acceso para crear una red local dedicada. Esta combinación resulta ideal para entornos educativos, ya que ofrece un equilibrio óptimo entre capacidad técnica, bajo costo, portabilidad y facilidad de gestión institucional.

8.1. Raspberry Pi: Servidor central del ecosistema

La Raspberry Pi 5 (8GB RAM) utilizada en este proyecto incorpora un procesador Broadcom BCM2712 de cuatro núcleos ARM Cortex-A76 a 2,4 GHz, junto con 8 GB de memoria LPDDR4X-4267 SDRAM. Para mejorar significativamente el rendimiento de almacenamiento y la velocidad de acceso a datos, hemos implementado una placa extensora que permite la conexión de un SSD NVMe de 256GB, proporcionando velocidades de lectura/escrituras muy superiores a las tarjetas microSD tradicionales, ver Figura 6.

Figura 6. Raspberry Pi utilizada.

En el apartado multimedia, la GPU VideoCore VII de 800 MHz soporta decodificación y codificación por hardware de vídeo H.265 (HEVC) en 4K, así como OpenGL ES 3.1 y Vulkan 1.2. Esto reduce la carga sobre la CPU principal al manejar varios *streams* de vídeo, manteniendo baja la latencia y permitiendo que la CPU se enfoque en la lógica de aplicación, como la gestión de usuarios y la sincronización de pizarras digitales.

En cuanto a conectividad, la Raspberry Pi 5 incorpora Wi-Fi 5 de doble banda (2,4 y 5 GHz), Bluetooth 5.0/BLE y un puerto Ethernet Gigabit para conexión cableada. Dispone de dos puertos USB 3.0 y dos USB 2.0 para almacenamiento externo, cámaras o dispositivos de entrada, así como de dos salidas micro-HDMI capaces de emitir vídeo en 4K a 60 Hz de forma simultánea, ideal para presentaciones o proyección en el aula.

8.2. Router WiFi: Infraestructura de red dedicada

El router WiFi se configura como punto de acceso (*Access Point*, AP) para crear una red local independiente de la infraestructura institucional. Esto elimina la dependencia de redes corporativas, que pueden tener restricciones o políticas no optimizadas para multimedia. El router Gateway local asignando direcciones IP privadas y gestionando el acceso de los dispositivos de estudiantes y docentes.

Para un rendimiento óptimo, se recomienda un router con soporte Wi-Fi 802.11ac (Wi-Fi 5) o superior, capacidad dual-band para separar el tráfico de gestión y multimedia, y puertos Gigabit Ethernet para la conexión cableada con la Raspberry Pi. El router debe ser capaz de gestionar al menos 30 dispositivos concurrentes sin degradación significativa del rendimiento. En nuestro caso el ASUS Wireless-AC2900 cumple sobradamente con las recomendaciones. En la Figura 7podemos ver un esquema de conexión de 3 dispositivos mediante wifi y dos cableados.

La gestión de calidad de servicio (QoS) es fundamental: permite priorizar el tráfico WebRTC (audio/vídeo), asignar ancho de banda garantizado al *stream* del docente y evitar que un solo dispositivo consuma todos los recursos de la red, asegurando así una experiencia educativa estable y de calidad.

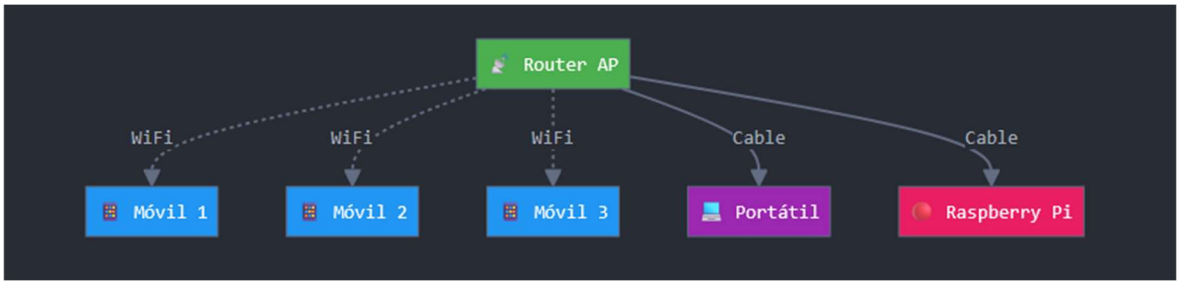


Figura 7. Ejemplo de conexiones al router WiFi. Elaboración propia.

Capítulo 3. Instalación y configuración del sistema

En este capítulo presento una guía detallada para la instalación y configuración completa del sistema WiWiewCast. La implementación del sistema se fundamenta en una arquitectura híbrida que combina hardware de bajo costo con software de código abierto, utilizando como elemento central una Raspberry Pi 5 equipada con 8GB de RAM y un almacenamiento rápido NVMe, que ofrece la potencia necesaria para manejar transmisiones de video en vivo, soportar hasta 20 usuarios conectados al mismo tiempo y ejecutar funciones colaborativas.

La arquitectura del sistema WiWiewCast se compone de tres partes principales que trabajan de manera integrada para ofrecer una plataforma de transmisión interactiva eficiente y accesible. En primer lugar, el hardware base está formado por una Raspberry Pi 5 con 8GB de RAM y una placa extensora MCUZone MPS2280, que permite conectar un disco SSD NVMe de 480 GB, mejorando significativamente la velocidad de almacenamiento en comparación con las tarjetas microSD tradicionales. Además, un router WiFi configurado como punto de acceso crea una red local dedicada que garantiza la conectividad estable de todos los dispositivos participantes.

En segundo lugar, el stack de software se basa en Raspberry Pi OS de 64 bits y utiliza tecnologías clave como Node.js para la ejecución de aplicaciones, Node-RED para la programación visual de flujos de datos, Socket.IO para la comunicación en tiempo real y Axios para las peticiones HTTP a nivel de clientes.

Finalmente, la capa de servicios de aplicación ofrece interfaces diferenciadas para los emisores de contenido, los espectadores y los administradores, permitiendo una experiencia colaborativa y un control efectivo del sistema.

1. Configuración de la Raspberry Pi

Para poner en marcha la Raspberry Pi 5, lo primero es preparar todo el hardware necesario: la propia Raspberry Pi 5, una fuente de alimentación adecuada, una tarjeta microSD, y un SSD NVMe con placa adaptadora. También se necesita acceso a internet para la descarga del software necesario.

El primer paso consiste en instalar el sistema operativo. Para ello, se debe descargar el programa Raspberry Pi Imager desde la página oficial (Figura 8) en un ordenador con acceso a internet.

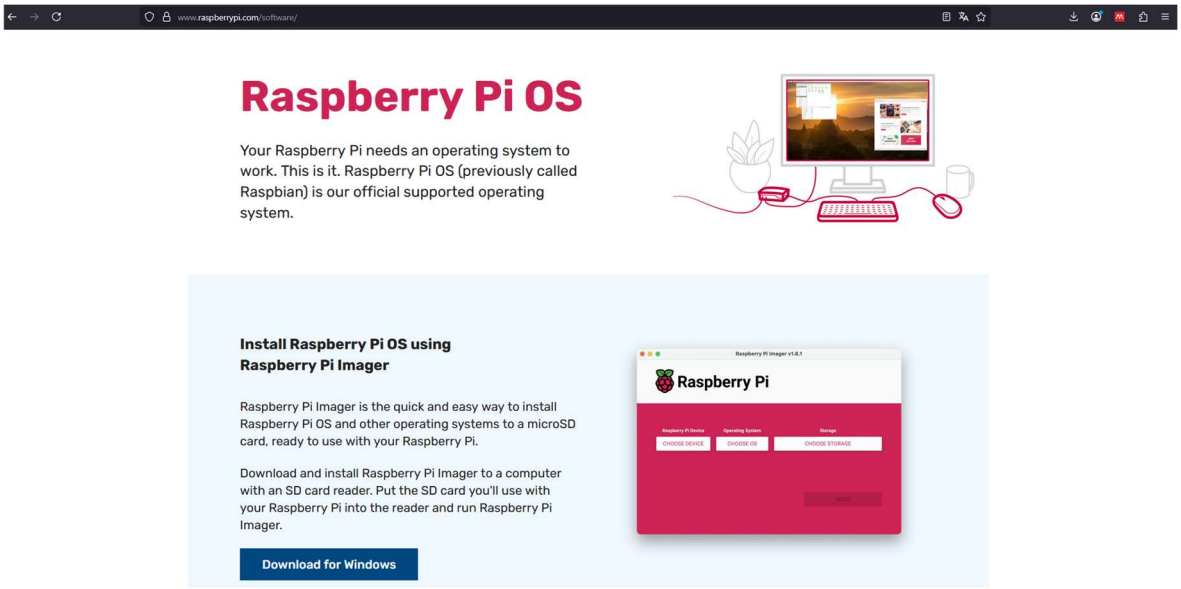


Figura 8. Página de descarga de Raspberry Pi Imager

Para grabar la tarjeta microSD, usando el Imager, selecciona el dispositivo Raspberry Pi, en este caso Raspberry Pi 5, luego la opción "Raspberry Pi OS (64-bit)" en el sistema operativo y elige la tarjeta como almacenamiento.

Antes de grabar la imagen del sistema operativo en la Raspberry Pi, es importante ajustar algunas configuraciones que harán mucho más fácil y seguro el uso posterior del dispositivo. Entre estas configuraciones clave están: cambiar el nombre del dispositivo (hostname), activar el acceso remoto seguro mediante el protocolo SSH, definir el usuario y la contraseña, y establecer los parámetros de localización como el idioma y la zona horaria. Activar el servicio SSH es especialmente útil porque permite administrar la Raspberry Pi a distancia, sin necesidad de conectar un monitor, teclado o ratón. Esto resulta ideal para el mantenimiento del sistema una vez que WiWiewCast esté funcionando en el entorno educativo, ya que se puede acceder a la Raspberry Pi desde otro ordenador usando la red local simplemente con el comando SSH y las credenciales configuradas.

Es fundamental cambiar el usuario y la contraseña por defecto, ya que mantener los valores estándar supone un riesgo de seguridad considerable, especialmente si el dispositivo se conecta a redes más amplias o tiene acceso a Internet. Para establecer la conexión remota, el dispositivo cliente debe tener instaladas aplicaciones específicas para la comunicación

SSH, como PuTTY o Terminal (para acceso a consola), VNC Viewer (para acceso gráfico remoto), o FileZilla/SCP (para transferencia segura de archivos entre la Raspberry Pi y el dispositivo local).

Estas opciones de gestión remota, combinadas con la posibilidad de acceso físico directo cuando sea necesario, simplifican la administración del sistema y reducen la necesidad de recursos técnicos especializados para las tareas rutinarias. Así se garantiza la estabilidad y el funcionamiento continuo del servicio, manteniendo una arquitectura sencilla y eficiente. A partir de aquí vamos a configurar la Raspberry Pi con PuTTY y desde consola.

Una vez grabada la imagen, coloca la tarjeta microSD en la Raspberry Pi y conéctala en la misma red que el dispositivo cliente. Desde el equipo cliente nos conectamos a la consola de la Raspberry Pi averiguando primero la IP que le ha sido asignada por el router de la red con el comando:

```
$> ping raspberrypi.local
```

Con dicha IP nos conectamos a la consola de la Raspberry Pi a través de SSH y a continuación actualizamos el sistema:

```
$> ssh pi@192.168.1.XX
# Dentro de la consola de Raspberry Pi actualizamos
pi@raspberrypi:~$> sudo apt update && sudo apt upgrade -y
```

Acabada la actualización, el siguiente paso sería configurar el equipo con una IP fija. La elección de la IP es importante porque tiene que estar fuera del rango de direcciones DHCP con las que configuraremos el router del sistema WiWiewCast. El comando que utilizamos para asignar la dirección IP estática es:

```
$> sudo nmtui
```

Como último paso, y antes de pasar a instalar el entorno de desarrollo, hay que instalar el disco NVMe SSD M.2. Apagamos la Raspberry Pi e instalamos la placa adaptadora (HAT) para después montar y conectar el disco al puerto PCIe de la placa con el cable flex. Después de colocar el hardware, volvemos a encender la Raspberry Pi y habilitamos el puerto PCIe, que viene deshabilitado por defecto. Dentro de la terminal hay que ejecutar un editor de texto para acceder al fichero que habilita el puerto, y luego introducir al final del archivo dos líneas de configuración:

```
$> sudo nano /boot/firmware/config.txt
# Al final del archivo insertar los siguientes comandos
dtparam=nmve #activa el puerto
dtparam=pciex1_gen=3 # activa máxima velocidad de PCIe 3.0
```

Después de guardar los cambios en el archivo, reiniciamos la Raspberry Pi para que los cambios surtan efecto. A continuación, configuraremos la Raspberry Pi para que arranque desde el puerto PCIe con el siguiente comando:

```
$> sudo rpi-eeprom-config --edit
# Una vez dentro del archivo de configuración
[all]
BOOT_UART=1
POWER_OFF_ON_HALT=0
BOOT_ORDER=0Xf416
PCIE_PROBE=1
```

La variable BOOT_ORDER hace referencia al orden de arranque de la Raspberry Pi, donde cada dígito representa un método de arranque diferente: el 4 equivale al USB, el 1 a la tarjeta microSD y el 6 al puerto PCIe NVMe. El valor 0xf416 indica que intentará arrancar primero desde USB, luego desde microSD, y finalmente desde NVMe. La línea PCIE_PROBE=1 habilita la detección automática de dispositivos PCIe.

Guardamos el archivo y volvemos a reiniciar el sistema. Solo nos queda hacer un clon de la tarjeta microSD al disco NVMe. Para esto instalamos en consola rpi-clone, que detecta y monta particiones automáticamente, evita copiar sectores vacíos y cambia UUIDs automáticamente

```
$>git clone https://github.com/billw2/rpi-clone.git
$>cd rpi-clone
$>sudo cp rpi-clone /usr/local/sbin
$>lsblk #verificamos que el M2 /dev/nvme0n
$>sudo rpi-clone /dev/nvme0n1 #Ejecutamos la clonación.
```

Una vez clonada la SD al M2, apagamos la Raspberry Pi, retiramos la tarjeta SD y volvemos a encender. El sistema operativo arrancará desde el disco M2

2. Instalación de Node.js y Node-RED

Una vez completadas las configuraciones anteriores de la Raspberry Pi y verificado el correcto funcionamiento del almacenamiento NVMe, procedemos a instalar el stack de software fundamental para el sistema WiWiewCast: Node.js como runtime de ejecución y Node-RED como entorno de desarrollo visual de flujos de datos.

- **Instalación de Node.js (versión 18 LTS)**

Node.js constituye el núcleo del sistema WiWiewCast, proporcionando el entorno de ejecución JavaScript necesario para todas las aplicaciones del servidor. La versión 18 LTS (Long Term Support) garantiza estabilidad y soporte a largo plazo. Para instalar Node.js desde el repositorio oficial de NodeSource, ejecutamos los siguientes comandos:

```
# Descargar e instalar script de configuración del repositorio
$> curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
# Instalar Node.js y npm
$> sudo apt-get install -y nodejs
# Verificar la instalación correcta
$> node --version # Debe mostrar v18.x.x
$> npm --version # Debe mostrar 9.x.x o superior
```

Es importante verificar que tanto Node.js como npm (Node Package Manager) se han instalado correctamente. La versión de Node.js debe comenzar con v18, mientras que npm debe estar en la versión 9 o superior para garantizar compatibilidad con todas las dependencias del proyecto.

- **Instalación de Node-RED**

Node-RED es una herramienta de programación visual basada en flujos que permite conectar dispositivos de hardware, APIs y servicios en línea de manera intuitiva. En el contexto de WiWiewCast, Node-RED actúa como el orquestador principal que gestiona las comunicaciones en tiempo real entre los diferentes componentes del sistema.

```
# Instalar Node-RED globalmente en el sistema
$> sudo npm install -g --unsafe-perm node-red
# Verificar la instalación
$> node-red -version
```

La opción --unsafe-perm es necesaria cuando se instala Node-RED como usuario root, ya que algunos módulos requieren permisos especiales durante la compilación de dependencias nativas.

El sistema WiWiewCast requiere módulos adicionales para manejar las comunicaciones en tiempo real y las conexiones WebRTC. Estos módulos extienden las capacidades básicas de Node-RED con funcionalidades específicas para streaming de video y comunicación bidireccional.

```
# Instalar módulo Socket.IO para Node-RED
$> sudo npm install -g node-red-contrib-socketio
# Instalar WebRTC nativo para Node.js
$> sudo npm install -g wrtc # Instalar módulos adicionales necesarios
$> sudo npm install -g node-red-contrib-axios
$> sudo npm install -g node-red-dashboard
```

Antes de continuar, es fundamental verificar que todas las dependencias del sistema necesarias para WebRTC estén instaladas:

```
# Instalar librerías del sistema requeridas por WebRTC
$> sudo apt-get install -y \
 libgtk-3-dev \
 libnotify-dev \
 libnss3-dev \
 libxss1 \
 libxtst6 \
 xvfb \
 libatspi2.0-0 \
```

```
 libdrm2 \
 libxcomposite1 \
 libxdamage1 \
 libxrandr2 \
 libgbm1 \
 libasound2-dev
# Verificar que WRTC se ha compilado correctamente
$> node -e "console.log(require('wrtc'))"
```

Una vez instalado Node-RED, es recomendable ejecutarlo por primera vez para generar la estructura de directorios y archivos de configuración:

```
$> node-red
```

Dado que la Raspberry Pi 5 tiene 8GB de RAM, podemos optimizar la configuración de memoria para Node.js y Node-RED:

```
# Añadir variables de entorno al perfil de usuario
$> echo 'export NODE_OPTIONS="--max-old-space-size=6144"' >> ~/.bashrc
$> echo 'export NODE_RED_OPTIONS="--max-old-space-size=6144"' >>
 ~/.bashrc
# Recargar el perfil
$> source ~/.bashrc
```

Esta configuración permite que Node.js utilice hasta 6GB de RAM, dejando 2GB para el sistema operativo y otros procesos, optimizando así el rendimiento del sistema WiWiewCast.

Con estos pasos completados, el sistema dispone ya del entorno de ejecución JavaScript y las herramientas de desarrollo visual necesarias para implementar las funcionalidades de transmisión en tiempo real de WiWiewCast. El siguiente paso será configurar los certificados de seguridad y establecer las conexiones HTTPS necesarias para que los navegadores web permitan acceso a los dispositivos multimedia.

3. Creación de certificados SSL autofirmados

Los navegadores web modernos implementan políticas de seguridad estrictas que requieren conexiones HTTPS para acceder a dispositivos multimedia como cámaras y micrófonos. Esto es fundamental para el funcionamiento de WiWiewCast, ya que el sistema depende del acceso a estos dispositivos para la transmisión de video en tiempo real. Por esta razón, es imprescindible configurar certificados SSL que permitan establecer conexiones seguras entre los dispositivos cliente y el servidor de la Raspberry Pi.

Antes de generar los certificados, necesitamos crear una estructura de directorios organizada y configurar el entorno adecuadamente:

```
# Crear directorio para certificados
$> sudo mkdir -p /home/pi/ssl/
$> cd /home/pi/ssl/
```

```
#Generar clave privada
$> sudo openssl genrsa -out server.key 2048
#Generar certificado autofirmado (válido por 365 días)
$> sudo openssl req -new -x509 -key server.key -out server.crt -days 365
#Configurar Permisos
$> sudo chown root:ssl-cert /home/pi/ssl/server.key
$> sudo chown root:ssl-cert /home/pi/ssl/server.crt
$> sudo chmod 640 /home/pi/ssl/server.key
$> sudo chmod 644 /home/pi/ssl/server.crt
#Añadir usuario pi al grupo ssl-cert
$> sudo usermod -a -G ssl-cert pi
```

Con estos certificados SSL correctamente instalados y configurados, el sistema WiWiewCast podrá establecer conexiones HTTPS seguras que permitan el acceso a dispositivos multimedia desde los navegadores web de los usuarios. Este paso es fundamental antes de proceder con la configuración final de Node-RED y el despliegue de la aplicación web.

4. Configuración y acceso a Node-RED

Una vez instalados los certificados SSL, procedemos a configurar Node-RED para que utilice conexiones seguras HTTPS, lo cual es fundamental para que WiWiewCast pueda acceder a los dispositivos multimedia de los usuarios y garantizar la seguridad de las comunicaciones.

Antes de configurar Node-RED, necesitamos asegurar que la estructura de directorios esté correctamente establecida:

```
#Crear directorio de trabajo y acceder
$> mkdir ~/.node-red | cd ~/.node-red
# Crear directorio para archivos estáticos de la aplicación web
$> mkdir -p /home/pi/nodered/public
```

El archivo settings.js es el núcleo de la configuración de Node-RED. Este archivo define cómo Node-RED manejará las conexiones, la seguridad y las funcionalidades específicas del sistema WiWiewCast:

```
#Configurar settings.js
$> nano settings.js
```

```
javascriptvar fs = require("fs");
module.exports = {
    // Configuración HTTPS
    https: {
        key: fs.readFileSync('/etc/ssl/wiwicast/server.key'),
        cert: fs.readFileSync('/etc/ssl/wiwicast/server.crt')
    },
    // Directorio para archivos estáticos
   httpStatic: '/home/pi/nodered/public',
    // Puerto para HTTPS
    uiPort: process.env.PORT || 1880,
    // Resto de configuración existente...
    functionGlobalContext: {
```

```
            wrtc: require('wrtc'),
            os: require('os'),
        },
        // Confiiguración de seguridad
         requireHttps: true
    }
```

- **Acceso a Node-RED**

Tras configurar Node-RED en tu Raspberry Pi, puedes acceder fácilmente a su interfaz gráfica desde cualquier ordenador conectado a la misma red. Solo tienes que abrir tu navegador web y escribir en la barra de direcciones la IP de tu Raspberry Pi seguida de :1880. Por ejemplo, si la IP de tu Raspberry Pi es 192.168.1.100, deberás ingresar en la barra de direccines https://192.168.1.100:1880, así aparecerá la interfaz web de Node-RED, donde podrás crear, editar y gestionar tus flujos de trabajo.

A continuación realizaremos un ejemplo de una página web que muestre “Hola Mundo”. Para ello creamos un flujo en Node-RED y añadimos los nodos necesarios. En este caso HTTP In (Recibir petición), Template (Generar HTML) y HTTP Response (Enviar respuesta).

El nodo HTTP In se configura con el endpoint /hola, el nodo HTTP Response no hace falta configurarlo y al nodo Template se le añade el siguiente código html:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta    name="viewport"    content="width=device-width,    initial-
scale=1.0">
    <title>Hola Mundo - Node-RED</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 100px;
            background-color: #f0f0f0;
        }
        h1 {
            color: #ff6b6b;
            font-size: 48px;
        }
        p {
            color: #666;
            font-size: 18px;
        }
```

```
    </style>
</head>
<body>
    <h1>¡Hola Mundo!</h1>
    <p>Esta página fue generada con Node-RED</p>
</body>
</html>
```

Antes de instanciar unimos los nodos HTTP In y HTTP Response al nodo Template quedando el flujo de la siguiente como se ve en laFigura 9 :

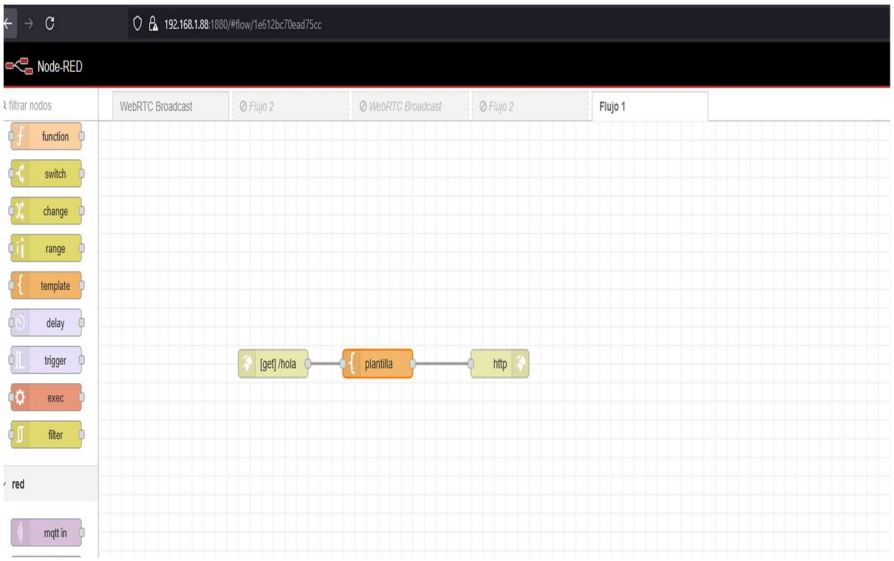


Figura 9. Flujo de "Hola Mundo"

Ahora solo con ingresar en el navegador de cualquier dispositivo que esté en la misma red la dirección https://ip_raspberry:1888/hola , podrá ver el mensaje “Hola Mundo” devuelto por Node-RED como aparece en la Figura 10.

Figura 10. Hola mundo desde Node-RED

- **Configuración del router**

La configuración del router WiFi en modo Access Point (AP) es una parte fundamental en la arquitectura de WiWiewCast, ya que es la encargada de crear una red local dedicada

que asegura una conexión estable entre todos los dispositivos que participan en el sistema. Al establecer el router como AP, se genera una red aislada y optimizada específicamente para la transmisión de video en tiempo real, lo que ayuda a evitar interferencias externas y garantiza un mejor ancho de banda.

Para implementar esta red dedicada, es necesario contar con un router que tenga la opción de funcionar en modo Access Point, una característica presente en la mayoría de los routers domésticos modernos, aunque puede aparecer con distintos nombres según el fabricante, como AP Mode, Bridge Mode o simplemente Access Point. Es recomendable que el router soporte Wi-Fi 802.11ac (Wi-Fi 5) o superior, tenga al menos cuatro puertos Ethernet, sea capaz de gestionar más de 20 dispositivos conectados simultáneamente y permita la configuración de red dual band (2.4GHz y 5GHz) para optimizar el rendimiento.

El primer paso para la configuración consiste en acceder a la interfaz de administración del router, lo cual se hace conectando un ordenador al router mediante un cable Ethernet y accediendo a la dirección IP de administración desde el navegador web. Un paso importante es asignar una IP estática a la Raspberry Pi, por ejemplo 192.168.100.5, y definir un rango DHCP específico para los demás dispositivos, como de 192.168.100.10 a 192.168.100.200, asegurando así una gestión ordenada de las direcciones IP en la red. El siguiente paso es configurar las opciones de Calidad de Servicio (QoS) para priorizar el tráfico de video y evitar saturaciones.

Tras realizar la configuración inicial, es recomendable verificar el funcionamiento de la red y realizar pruebas de diagnóstico para confirmar que todos los dispositivos pueden conectarse correctamente y que la transmisión de video es fluida.

- **Navegadores web**

Los navegadores web constituyen la interfaz principal entre los usuarios y el sistema WiWiewCast, para la comunicación bidireccional y colaboración. La elección del navegador y su correcta configuración son aspectos clave para que el sistema funcione bien, ya que tecnologías como WebRTC, Media Capture API y WebSocket necesitan soporte específico y optimizado que no todos los navegadores ofrecen de la misma manera.

El navegador no solo sirve para abrir la aplicación web, sino que también ejecuta el código JavaScript, gestiona el acceso a la cámara y el micrófono, procesa la transmisión de video en tiempo real y mantiene las conexiones necesarias para la colaboración instantánea. Si

el navegador no está bien configurado o no es compatible, la experiencia del usuario puede verse afectada y algunas funciones podrían no funcionar correctamente.

Para WiWiewCast, Google Chrome es el navegador recomendado porque ofrece el soporte más completo y robusto para todas las tecnologías necesarias, especialmente WebRTC y las APIs de acceso a medios. Chrome fue uno de los primeros en implementar WebRTC y sigue siendo el que mejor rendimiento ofrece en este tipo de aplicaciones. Se recomienda usar Chrome versión 90 o superior en ordenadores, y Chrome Mobile 90 o superior en dispositivos móviles, para asegurar la compatibilidad. Además, navegadores como Mozilla Firefox, Microsoft Edge, Safari y Opera también son compatibles con WebRTC y las tecnologías asociadas, aunque pueden presentar pequeñas diferencias en el comportamiento o en el soporte de algunas funciones avanzadas.

Por último, es importante asegurarse de que el navegador esté actualizado y que los permisos de acceso a la cámara y el micrófono estén correctamente configurados, ya que estos son necesarios para la transmisión de video y la interacción en tiempo real. También se recomienda cerrar otras pestañas o aplicaciones que puedan consumir recursos o ancho de banda para garantizar una experiencia fluida durante el uso de WiWiewCast.

Capítulo 4. Especificaciones de la aplicación y arquitectura del sistema

1. Introducción

La motivación principal de este trabajo surge de la necesidad de superar las barreras que todavía persisten en determinados entornos educativos, especialmente en los laboratorios tradicionales. En estos espacios, la elevada cantidad de estudiantes suele dificultar la visibilidad, impidiendo que todos puedan observar con claridad las demostraciones prácticas o las manipulaciones realizadas por el profesor. Esta limitación no solo afecta la comprensión de los procedimientos, sino que también restringe la interacción directa entre el alumnado y el docente, reduciendo las oportunidades de participación activa y dificultando la resolución inmediata de dudas.

El origen de estos obstáculos se encuentra, principalmente, en las restricciones físicas y espaciales propias de los laboratorios, que limitan el acceso visual y la movilidad dentro del aula. Como resultado, el potencial educativo de las actividades prácticas no se aprovecha plenamente, lo que puede impactar negativamente en la calidad del aprendizaje.

Cabe destacar que no todos los laboratorios cuentan con los mismos recursos: mientras algunos disponen de equipamiento avanzado, otros operan con medios mucho más limitados. Este trabajo se contextualiza dentro de Anatomía Veterinaria I y II, asignaturas impartidas en el primer y segundo curso del Grado en Veterinaria de la ULPGC. Las sesiones prácticas, impartidas en el laboratorio, abordarán el estudio directo de diferentes preparaciones anatómicas y la disección integral y reglada de diferentes especies de mamíferos domésticos. Resulta evidente que el material de prácticas empleado es bastante limitado, y requiere de condiciones especiales para su conservación, y tratamiento previo hasta ponerse en disposición del alumnado. Estas limitaciones hacen aún más patente la necesidad de soluciones innovadoras que permitan mejorar la experiencia educativa y garantizar el acceso equitativo al aprendizaje práctico.

A continuación, se presenta la Figura 11 de la que refleja de manera visual los problemas de visibilidad y participación que se derivan de las barreras mencionadas, sirviendo como punto de partida para el desarrollo de la solución propuesta en este proyecto.

Figura 11. Clase masificada

La aplicación desarrollada está específicamente diseñada para superar esas limitaciones. Para ello, integra una serie de especificaciones técnicas y funcionales que permiten solventar las barreras de visibilidad, interacción y acceso que suelen presentarse en estos espacios. El sistema utiliza tecnologías de transmisión de video en tiempo real, em este caso WebRTC, que facilitan que todos los estudiantes puedan observar con claridad las demostraciones del profesor, asegurando que cada estudiante tenga la oportunidad de participar y visualizar el contenido desde su propio dispositivo.

Visualizar el contenido desde su propio dispositivo es una ventaja de este sistema, ya que utiliza los recursos tecnológicos de los propios estudiantes para ofrecer una experiencia personalizada y flexible. Esto permite que cada alumno ajuste parámetros de la imagen según sus necesidades, y que pueda seguir la demostración sin obstáculos visuales ni distracciones causadas por la disposición física del aula. Además, al aprovechar los dispositivos personales, se reduce la necesidad de equipamiento adicional por parte de la institución, facilitando la implementación del sistema incluso en laboratorios con recursos limitados.

Otra funcionalidad destacada de la aplicación es la pizarra digital colaborativa creada sobre la transmisión del video, con capacidad para identificar de forma visual a los usuarios que están dibujando sobre esta. Cada estudiante que participa en la pizarra es diferenciado mediante un color específico asignado a su trazo, lo que facilita reconocer quién está interviniendo en tiempo real. Pensando en la accesibilidad, la aplicación incorpora una distinción adicional basada en la forma de los píxeles de cada trazo: así, los usuarios no

solo se diferencian por color, sino también por patrones de pixelado únicos. Esta característica está especialmente pensada para estudiantes con daltonismo o dificultades para distinguir colores, permitiéndoles identificar fácilmente la autoría de cada intervención en la pizarra, independientemente de sus capacidades visuales. De este modo, se garantiza que la experiencia colaborativa sea inclusiva y accesible para todos los participantes.

2. Especificaciones de la aplicación

Debido a sus especificidades y sus diferentes roles, la aplicación puede tener varios casos de uso que a continuación se expone:

- **Caso de uso 1:**

La configuración del sistema es:

- Profesor: es el Administrar y Emisor utilizando un único dispositivo móvil,
- Alumnos: Visualizadores de la emisión y usuarios de la pizarra virtual.

El profesor utiliza su propio dispositivo móvil tanto para emitir la señal de video como para administrar la sesión, mientras que los estudiantes, denominados *viewers*, acceden desde sus propios dispositivos para visualizar la transmisión y participar de forma interactiva. Esta modalidad representa la forma más sencilla de uso de la aplicación, ya que centraliza todas las funciones de emisión y administración en un único dispositivo, facilitando la gestión para el docente.

Sin embargo, esta configuración tiene la limitación de que la interacción entre los estudiantes es más restringida, ya que cada *viewer* solo puede ver su propio dibujo y no los de sus compañeros. Por tanto, aunque es una solución ideal para entornos con pocos recursos o cuando se busca una implementación rápida y sencilla, no favorece tanto la colaboración grupal ni la visualización colectiva de las aportaciones de todos los participantes, quedando la supervisión y la interacción principalmente en manos del profesor.

Figura 12. Caso de uso 1

- **Caso de uso 2:**

La configuración del sistema es:

- Profesor: es el Administrar y Emisor utilizando dos dispositivos,
- Alumnos: Visualizadores de la emisión y usuarios de la pizarra virtual.

En este segundo caso, la aplicación permite separar las funciones de emisión y administración, utilizando dos dispositivos móviles diferentes: uno para emitir la señal de video (por ejemplo, el móvil del profesor) y otro para gestionar la administración de la sesión (como puede ser una tablet o un segundo móvil). Esta separación ofrece al docente mayor flexibilidad y control, ya que puede ajustar la grabación, gestionar la interacción y observar en detalle las anotaciones y dibujos de los estudiantes desde el dispositivo de administración, sin interrumpir la transmisión principal. Los estudiantes, por su parte, continúan accediendo desde sus propios dispositivos para visualizar la transmisión y participar de forma interactiva. Sin embargo, al igual que en el caso anterior, la interacción entre los *viewers* sigue estando limitada, ya que cada estudiante solo puede ver su propio dibujo y no los de sus compañeros. Esta configuración resulta especialmente útil cuando el profesor necesita supervisar de manera más detallada la participación de los estudiantes, o cuando se requiere grabar la sesión para su posterior revisión, permitiendo así una gestión más organizada y eficiente de la clase a pesar de la limitación en la interacción colectiva entre los alumnos.

Figura 13. Caso de uso 2.

- **Caso de uso 3:**

La configuración del sistema es:

- Profesor: Emisor con un dispositivo móvil y Administrador desde un PC,
- Alumnos: Visualizadores de la emisión y usuarios de la pizarra virtual.

El tercer caso de uso es el que ofrece mayor nivel de interacción y visibilidad para todos los participantes. Aquí, el profesor utiliza un dispositivo móvil para emitir la señal de video, mientras que la administración de la sesión se realiza desde un ordenador personal conectado a una pantalla grande, como puede ser un proyector o monitor en el laboratorio. De este modo, el administrador (que puede ser el propio profesor o un asistente) tiene un control más amplio sobre la sesión, pudiendo gestionar usuarios, moderar la interacción y visualizar en tiempo real todos los dibujos y anotaciones que realizan los estudiantes en la pizarra colaborativa. Los *viewers*, además de seguir la transmisión desde sus dispositivos personales, pueden ver reflejadas en la pantalla común tanto sus propias aportaciones como las de sus compañeros, lo que fomenta la participación colectiva y el aprendizaje colaborativo. Esta modalidad es ideal para sesiones prácticas con grupos numerosos, ya que maximiza la visibilidad, la interacción y el aprovechamiento de los recursos tecnológicos disponibles en el aula.

Figura 14. Caso de uso 3

3. Arquitectura del Sistema

Después de hacer varias pruebas con diferentes arquitecturas, P2P, SFU e incluso con Multicast, la opción elegida una arquitectura hibrida con SFU, buscando garantizar una transmisión interactiva y fluida, integrando varios componentes que se comunican de manera eficiente. Node-RED coordina y administra todos los flujos dentro del sistema, mientras que WebRTC se encarga del intercambio de video y audio en tiempo real, y Socket.IO facilita la interacción y sincronización entre dispositivos de usuarios, sean emisores, espectadores o administradores.

3.1. Backend: Node-RED

El servidor backend emplea principalmente Node-RED, Node.js, Socket.IO y la librería wrtc para WebRTC. El motor de flujos de Node-RED gestiona los procesos y comunicaciones, dividiendo las funcionalidades en módulos independientes (como la difusión WebRTC o la gestión de usuarios). Cada usuario conectado, *stream* activo o configuración queda almacenado en el contexto de Node-RED, facilitando el seguimiento y modificación en tiempo real.

Un punto clave es la introducción de la Unidad de Reenvío Selectivo (SFU). Aquí, el punto de emisión (*broadcaster*) envía su señal al servidor que, a través de la SFU, la distribuye eficientemente a todos los consumidores o visualizadores, permitiendo la escala a múltiples usuarios conectados simultáneamente.

3.2. Flujo de Broadcast (Emisor)

El flujo de broadcast es el encargado de recibir y procesar el *stream* enviado por el emisor, permitiendo que la señal de video llegue correctamente al sistema para su posterior distribución a los visualizadores. Este proceso se inicia cuando el emisor realiza una petición HTTP POST al *endpoint* específico /*broadcast*, enviando junto con la solicitud una oferta SDP y la identificación de su sesión. El mensaje o Payload esperado contiene la información necesaria para negociar la conexión WebRTC.

De manera complementaria, existe una ruta de limpieza llamada /*cleanup-broadcast,* también accesible vía POST, cuya función es liberar todos los recursos asociados a la sesión WebRTC cuando el emisor se desconecta. Esto permite asegurar que no queden conexiones abiertas innecesariamente y el sistema mantenga un uso eficiente de los recursos disponibles.

Figura 15. Diagrama de flujo de Broadcast (Emisor)

En el núcleo de este flujo se encuentra la lógica de broadcast, que se encarga de crear una instancia de RTCPeerConnection para el emisor. Tras recibir y procesar la oferta SDP, genera una respuesta SDP que se devuelve al emisor para completar el proceso de negociación y establecer la conexión WebRTC. El *stream* resultante se almacena en el contexto global del sistema, permitiendo que esté disponible para su consumo por parte de otros usuarios. Además, se implementa un temporizador automático de 30 minutos para gestionar la limpieza y se monitorizan los estados de conexión ICE para garantizar la estabilidad de la transmisión. Finalmente, la respuesta al emisor contiene la SDP necesaria para cerrar el ciclo de establecimiento de la sesión WebRTC y poner el stream en línea.

3.3. Flujo de Consumer (Visualizadores)

El flujo de *consumer* está orientado a distribuir el *stream* que ya ha sido recibido y almacenado. Su objetivo principal es gestionar las solicitudes de los visualizadores, permitiendo que cada uno reciba la señal de video de manera gestionada y eficiente.

Cuando un visualizador desea acceder al stream, realiza una petición HTTP POST al *endpoint /consumer*, enviando una oferta SDP generada desde su propio dispositivo. El sistema, al recibir esta solicitud, ejecuta la lógica de *consumer* que primero accede al *stream* almacenado globalmente y luego crea una nueva instancia de RTCPeerConnection para el visualizador. Cada conexión añade las pistas del emisor al *peer* correspondiente mediante pc.addTrack(track, senderStream) y, tras procesar la oferta SDP recibida, genera una respuesta SDP personalizada para ese visualizador.

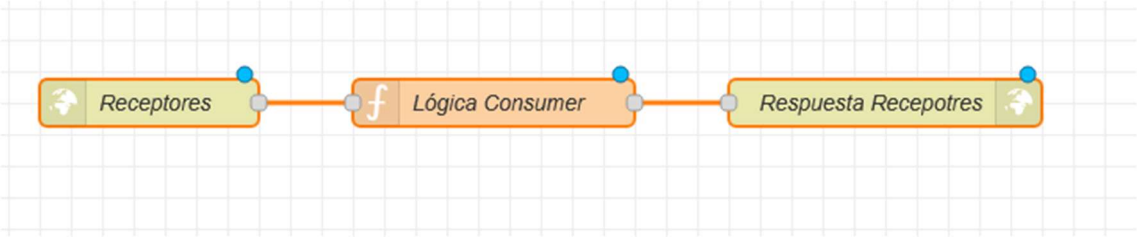


Figura 16. Flujo de Consumer (Visualizadores).

Este proceso implementa el patrón SFU (Selective Forwarding Unit), permitiendo que el sistema distribuya el mismo *stream* a múltiples consumidores de manera eficiente, sin tener que reenviar múltiples veces desde el origen. Por último, la respuesta al receptor se encarga de enviar la SDP resultante para que la conexión WebRTC quede establecida y el usuario pueda visualizar el *stream* en tiempo real.

3.4. Flujo de Gestión de Eventos Socket.IO

El flujo de gestión de eventos mediante Socket.IO es esencial para el funcionamiento en tiempo real del sistema WiWiCast, ya que permite la comunicación instantánea entre todos los participantes y la sincronización de sus acciones. Cuando un usuario se conecta al sistema, ya sea como espectador, administrador o emisor, se genera un evento específico que es capturado por el servidor. Por ejemplo, el evento *viewer-join* indica la llegada de un nuevo espectador, mientras que *admin-join* y *emisor-join* señalan la conexión del administrador y del emisor, respectivamente. Además, eventos como *viewer-started* y *emisor-started* confirman el inicio de la visualización o transmisión, y eventos como draw-data transmiten los trazos realizados en la pizarra colaborativa. También existen comandos como admin-clear-canvas y viewer-clear-canvas para gestionar la limpieza del canvas, ya sea por parte del administrador o a solicitud de un espectador.

El sistema mantiene actualizado en todo momento un registro de los usuarios conectados y de aquellos que están visualizando activamente el contenido. Cada usuario recibe un

identificador único, un color y un patrón de visualización, lo que facilita la distinción visual de las aportaciones individuales en la pizarra colaborativa y garantiza la accesibilidad para usuarios con daltonismo. Además, el sistema procesa y distribuye los eventos de dibujo, asegurando que todos los participantes relevantes reciban las actualizaciones en tiempo real. Cuando un usuario se desconecta, el sistema realiza automáticamente la limpieza de su estado, liberando recursos y manteniendo la coherencia de los listados. Por otro lado, se actualizan estadísticas de uso y conexión en tiempo real, proporcionando información para la administración y el seguimiento del sistema.

El flujo de eventos también se encarga de enviar notificaciones específicas a los diferentes roles. Al administrador se le informa de cambios en la lista de espectadores, de nuevos eventos de dibujo en la pizarra, de estadísticas del sistema y de alteraciones en el estado de las conexiones. Estas notificaciones permiten al administrador tener un control completo y en tiempo real sobre la sesión. Por su parte, los espectadores reciben mensajes personalizados, confirmaciones de que sus acciones han sido procesadas correctamente y comandos para realizar determinadas acciones, como la limpieza de su canvas.

Para garantizar la estabilidad y facilitar el mantenimiento del sistema, se han incorporado nodos de depuración (debug) que monitorean el flujo de eventos, verifican el correcto procesamiento de los datos, detectan errores y analizan el contenido de los mensajes intercambiados. Estos nodos son fundamentales para identificar y resolver incidencias de manera ágil, asegurando que la plataforma funcione de forma óptima en todo momento.

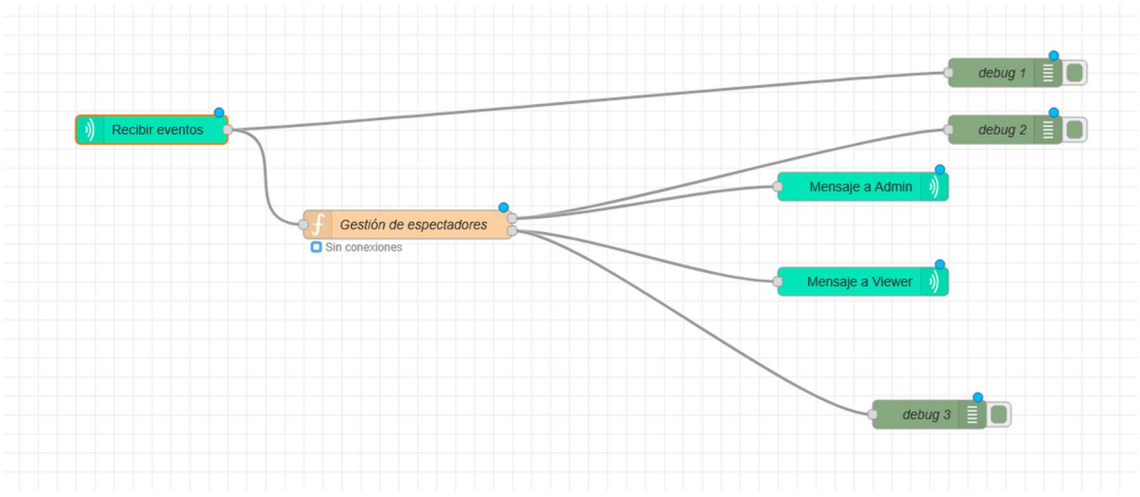


Figura 17. Flujo de Gestión de Eventos Socket.IO

3.5. Flujo de Archivos Estáticos

El flujo de archivos estáticos se encarga de servir las aplicaciones web necesarias para que los usuarios puedan acceder a la plataforma desde sus dispositivos. Este flujo captura todas las solicitudes HTTP GET dirigidas a recursos estáticos, como HTML, CSS y JavaScript, a través de una ruta genérica (por ejemplo, /:path). De este modo, cuando un usuario accede a una URL como /viewer, /admin o simplemente /index.html, la aplicación identifica la ruta solicitada y la mapea al archivo correspondiente en el sistema de archivos del servidor.

Una vez identificada la ruta, el sistema realiza una transformación sencilla para convertir las rutas amigables en nombres de archivo reales. Por ejemplo, la ruta /viewer se traduce a viewer.html, /admin a admin.html, y si no se especifica una ruta concreta, se sirve por defecto el archivo index.html. Esta lógica permite una navegación intuitiva y una gestión flexible de las diferentes interfaces de usuario según el rol del participante (emisor, administrador o espectador).

Tras determinar el archivo a servir, el sistema procede a leer su contenido desde el sistema de archivos, utilizando la información proporcionada en el mensaje de flujo (por ejemplo, msg.filename). Finalmente, el contenido del archivo se envía al cliente que realizó la solicitud, acompañado de los headers HTTP apropiados para garantizar una correcta interpretación por parte del navegador. Este proceso asegura que los usuarios reciban las interfaces web necesarias para interactuar con el sistema, independientemente de su dispositivo o sistema operativo.

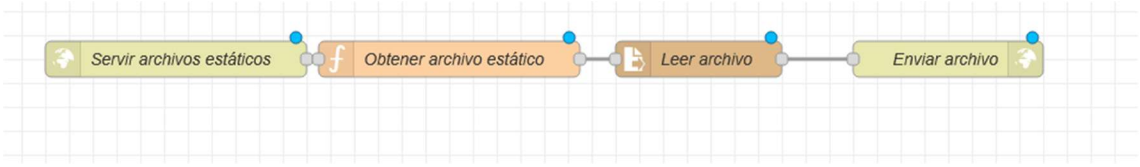


Figura 18. Flujo de Archivos Estáticos

4. Frontend. Navegador web

La integración del *frontend* con el *backend* en WiWiewCast se realiza principalmente a través de dos canales: la comunicación WebRTC para la transmisión de video en tiempo real y la gestión de eventos mediante Socket.IO para la interacción y sincronización entre los participantes. Dentro de esta arquitectura, el sistema de dibujo interactivo constituye una capa adicional fundamental que permite a los visualizadores interactuar directamente

sobre el contenido de video transmitido, creando una experiencia colaborativa e inmersiva que va más allá de la simple visualización pasiva.

En el lado del cliente, el archivo JavaScript principal (index.js) implementa la lógica necesaria para que el usuario pueda seleccionar la cámara deseada, iniciar la transmisión y gestionar la conexión con el servidor. Paralelamente, en la aplicación del visualizador (viewer.html), se implementa un canvas HTML5 superpuesto al elemento de video que captura los eventos táctiles y de mouse del usuario. Cuando el usuario selecciona una cámara en la aplicación emisor, el sistema actualiza el selector correspondiente y, al hacer clic en "Iniciar", se solicita el acceso al dispositivo mediante la API getUserMedia. Del mismo modo, cuando un visualizador toca o hace clic sobre el video, el canvas detecta las coordenadas del evento, las normaliza proporcionalmente al tamaño del canvas y genera un payload que incluye la posición (x, y), el color y patrón asignados únicamente a ese usuario, y otros metadatos como el tamaño del trazo.

Una vez obtenido el flujo de medios (MediaStream), se muestra una vista previa en el elemento de video de la interfaz. A continuación, se crea una instancia de RTCPeerConnection y se inicia la negociación SDP con el *endpoint /broadcast* del servidor. Simultáneamente, el sistema de dibujo envía los datos de interacción a través de Socket.IO usando el evento *draw-data*, que contiene toda la información necesaria para recrear el trazo en otros clientes. El *backend* de Node-RED procesa estos eventos en el nodo "Gestión de espectadores", donde se enriquece la información del dibujo con datos del usuario (nombre, rol, color asignado) almacenados en el contexto de flujo, y posteriormente retransmite estos datos al administrador para su visualización en tiempo real. En el panel de administración (admin.html), el sistema implementa un mecanismo de capas de dibujo donde cada visualizador tiene su propia capa virtual gestionada mediante un canvas HTML5 superpuesto al video. Cuando el administrador recibe un evento *draw-data*, el sistema identifica al usuario emisor por su socket ID y renderiza el trazo en el canvas usando el color y patrón específicos asignados a ese visualizador, manteniendo así una representación visual clara de quién está dibujando qué en cada momento. Cuando la conexión WebRTC se establece correctamente, el indicador de estado cambia a "Online", confirmando que la transmisión está activa y lista para ser recibida por los visualizadores, quienes ahora pueden no solo ver el contenido sino también interactuar dibujando sobre él, mientras que el administrador obtiene una vista completa de todas las interacciones superpuestas.

Además de la transmisión de video, la aplicación utiliza Socket.IO para gestionar eventos en tiempo real que permiten una experiencia interactiva y colaborativa. Por ejemplo, cuando el emisor se conecta, se envía el evento emisor-join con información sobre el usuario y su rol. Al iniciar la transmisión, se emite emisor-started para notificar al servidor y a los demás participantes. En el contexto del sistema de dibujo por capas, cuando un visualizador se conecta, el servidor le asigna automáticamente un color y patrón únicos de una paleta predefinida, enviando esta información a través del evento style-assigned para que el cliente configure su indicador visual y use estos valores en sus trazos. En el panel de administración, esta información se almacena en una estructura de datos que asocia cada socket ID con sus propiedades visuales, permitiendo que el canvas del admin renderice correctamente cada trazo con la identidad visual correspondiente. El frontend también escucha eventos como viewer-count-update, que actualiza en tiempo real el contador de espectadores conectados, así como eventos específicos del canvas como draw-data (para mostrar dibujos de otros usuarios distribuidos por capas virtuales) y admin-clear-canvas (que permite al administrador seleccionar específicamente qué capa de usuario limpiar, enviando un comando dirigido solo a ese visualizador particular.

El flujo de usuario típico en la aplicación de emisión comienza con la carga de la página, durante la cual se detectan las cámaras disponibles y se rellena el selector correspondiente para que el usuario elija el dispositivo deseado. En la aplicación del visualizador, este flujo incluye además la inicialización del canvas interactivo, la configuración de los event listeners para eventos táctiles y de mouse, y la preparación del contexto de dibujo 2D. En el panel de administración, se inicializa un sistema más complejo que prepara el canvas principal y crea una estructura de datos para gestionar las capas virtuales de cada usuario, incluyendo un mapa de identificadores de socket con sus respectivos colores, patrones y estados de actividad. Una vez seleccionada la cámara, al hacer clic en "Iniciar" se solicitan los permisos necesarios, se muestra la vista previa del video y se establece la conexión WebRTC con el servidor, cambiando el estado a "Online" cuando la transmisión está ready. Para los visualizadores, una vez establecida la conexión, pueden comenzar inmediatamente a interactuar tocando o haciendo clic sobre el video, generando trazos que se muestran instantáneamente en su pantalla (actualización optimista) y se envían al servidor para sincronización con otros participantes. El administrador, por su parte, ve estos trazos aparecer en tiempo real en su canvas, cada uno renderizado con el color y patrón distintivo del usuario correspondiente, y puede gestionar individualmente cada capa mediante controles específicos que aparecen en la lista de espectadores activos, incluyendo botones para limpiar la capa de un usuario específico o destacar temporalmente

sus interacciones. Durante la emisión, el usuario puede consultar en tiempo real el número de espectadores conectados a través de un contador actualizado automáticamente. Los visualizadores, por su parte, pueden limpiar sus propios dibujos mediante el botón correspondiente, lo que envía una solicitud viewer-clear-canvas al administrador quien decide si aprobar la limpieza mediante el evento admin-clear-canvas dirigido específicamente a la capa de ese usuario, manteniendo así un control centralizado sobre el contenido visual de cada participante. Finalmente, al hacer clic en "Detener", se cierran todas las conexiones, se liberan los recursos y la aplicación vuelve al estado inicial, lista para una nueva sesión.

Esta integración entre *frontend* y *backend*, combinando WebRTC para la transmisión de medios y Socket.IO para la gestión de eventos, permite ofrecer una experiencia de emisión intuitiva y adaptada a las necesidades de los usuarios, incluso desde dispositivos móviles, garantizando interactividad, bajo retardo y una gestión eficiente de los recursos. El sistema de dibujo por capas aporta una dimensión colaborativa donde cada participante mantiene su identidad visual y el administrador puede supervisar, gestionar y moderar las contribuciones individuales, transformando la experiencia de visualización pasiva en una herramienta de anotación y participación activa controlada y organizada, especialmente valiosa en contextos educativos donde los estudiantes pueden señalar, marcar o comentar visualmente sobre el contenido presentado por el instructor en tiempo real, mientras que el docente mantiene control total sobre la moderación de estas interacciones.

5. Ciclo básico de funcionamiento

El emisor envía su *stream* de video a través de una petición HTTP al *endpoint*/*broadcast*, donde el servidor lo almacena en el contexto global para su posterior distribución. Los visualizadores solicitan el *stream* mediante el endp*oint/consumer*, recibiendo así la transmisión de manera eficiente. Mientras tanto, los eventos gestionados por Socket.IO actualizan el estado del sistema y notifican a administradores y espectadores según corresponda. El servidor también se encarga de servir las aplicaciones web necesarias para que cada cliente pueda cargar la interfaz adecuada a su rol.

La gestión del estado compartido se realiza a través de dos contextos principales: el contexto global, que incluye recursos críticos como el *stream* del emisor y la instancia de la librería WebRTC, y el contexto de flujo, donde se almacena información dinámica como las listas de usuarios conectados y activos, la identificación del administrador y los colores y patrones asignados a cada usuario. La sincronización entre estos contextos se mantiene

gracias a los eventos de Socket.IO, que aseguran la coherencia y consistencia en todo el sistema.

En cuanto a los patrones de comunicación, el sistema utiliza HTTP para la negociación síncrona de las conexiones WebRTC, Socket.IO para la comunicación asíncrona y en tiempo real entre todos los participantes, y WebRTC para la transmisión de medios de baja latencia y alta calidad, apoyándose en una arquitectura P2P centralizada a través de una SFU. Esta combinación de tecnologías y flujos permite una experiencia colaborativa, fluida, adaptada a las necesidades de entornos educativos y de presentación, donde la interacción en tiempo real y la gestión eficiente de recursos son fundamentales.

Capítulo 5. Evaluación y Conclusiones

1. Ejecución y Evaluación del sistema.

La ejecución y evaluación del sistema WiWiewCast se llevó a cabo en un entorno real, concretamente en un aula de laboratorio de Transmisión por Línea del Departamento de Ingeniería Telemática, donde se recrearon las condiciones típicas de un escenario educativo con recursos tecnológicos limitados. Para iniciar el proceso, se conectó la Raspberry Pi al router mediante cable Ethernet, asegurando así una conexión estable y de bajo retardo, y se procedió al encendido de ambos dispositivos.

Caso de uso 1

En el primer caso de uso, se conectaron todos los dispositivos móviles a la red Wi-Fi WiWicast. En esta configuración, un único dispositivo móvil asumió simultáneamente el rol de emisor y administrador. Esto significa que desde este dispositivo se capturó y transmitió el video, al tiempo que se gestionó la sesión, supervisando la lista de espectadores y moderando la interacción. El resto de los dispositivos se conectaron como viewers, permitiendo a los estudiantes visualizar la transmisión y participar en la pizarra colaborativa. Esta modalidad, aunque sencilla de implementar, mostró algunas limitaciones en la interacción entre los viewers, ya que cada uno solo podía ver su propio dibujo y no el de sus compañeros, concentrando la supervisión y el control en el docente. Por otro lado, para el docente, gestionar tanto la emisión de video como el panel de administración desde un único dispositivo móvil puede resultar complicado en la práctica. La pantalla reducida de un teléfono o tablet dificulta la visualización simultánea de la transmisión, la lista de participantes y el contenido de la pizarra colaborativa, lo que puede afectar la eficacia en la moderación y el seguimiento de la sesión.

Caso de uso 2

El segundo caso de uso introdujo una separación de roles entre el emisor y el administrador. En este escenario, un dispositivo móvil se utilizó exclusivamente para emitir el video, mientras que otro dispositivo móvil distinto asumió las funciones de administración. El resto de los dispositivos continuaron conectados como viewers. Esta

configuración permitió al docente tener un mayor control sobre la sesión, pudiendo gestionar la interacción, monitorizar el estado de la transmisión y acceder a estadísticas en tiempo real desde el dispositivo administrador, sin interferir en la emisión principal. A pesar de esta mejora en la gestión, la limitación en la visualización de los dibujos entre viewers persistió, aunque la experiencia para el administrador fue más completa y flexible.

Caso de uso 3

En el tercer caso de uso, se combinó el uso de dispositivos móviles y un ordenador de aula conectado a un proyector. El emisor siguió siendo un dispositivo móvil, pero el rol de administrador lo asumió el ordenador, que proyectaba la interfaz de administración en una pantalla grande visible para todos los asistentes. Los espectadores se conectaron desde sus dispositivos móviles como viewers. Esta configuración maximizó la visibilidad y la interacción grupal, ya que tanto el video como las anotaciones colaborativas podían verse en la pantalla común, fomentando la participación activa y la discusión en el aula. Además, el administrador podía gestionar la sesión de manera centralizada, moderando la participación y resolviendo dudas en tiempo real. Esta modalidad demostró ser la más adecuada para sesiones prácticas con grupos numerosos, ya que aprovecha al máximo los recursos disponibles y facilita la colaboración entre todos los participantes.

En resumen, la ejecución del sistema en estos tres casos de uso permitió evaluar su funcionamiento en diferentes escenarios educativos, identificando tanto las ventajas como las limitaciones de cada configuración. La experiencia práctica confirmó que WiWiewCast es una solución flexible, adaptable y eficiente para la transmisión interactiva de video en entornos con recursos limitados, facilitando la colaboración y mejorando la experiencia educativa tanto para docentes como para estudiantes.

2. Conclusiones

WiWiewCast ha demostrado que es posible crear sistemas de transmisión de video interactivos usando tecnologías web modernas como WebRTC, Node-RED y Socket.IO, y que estos pueden funcionar bien incluso en equipos de bajo coste. Esto es importante porque permite que centros educativos, pequeñas empresas o asociaciones con pocos recursos puedan acceder a herramientas avanzadas de comunicación visual sin tener que invertir mucho dinero en equipos caros. El sistema funciona de manera eficiente en dispositivos sencillos, lo que elimina una barrera económica que antes limitaba el acceso a este tipo de soluciones.

Para que el sistema funcione bien incluso en equipos sencillos, se han tomado decisiones prácticas en el diseño. Por ejemplo, se eligió Node-RED como núcleo porque es ligero y puede manejar varias conexiones a la vez sin que el equipo se ralentice. Además, se usa el patrón SFU (Selective Forwarding Unit) para enviar el video a todos los usuarios, lo que ayuda a ahorrar ancho de banda y evita que el sistema se sature, algo común en otras formas de transmitir video.

El sistema integra varias funciones en una sola plataforma: transmite video en tiempo real, permite que los usuarios dibujen y anoten sobre la imagen que ven, y gestiona quién puede entrar y qué puede hacer cada uno. Todo esto funciona bien incluso en equipos económicos, lo que demuestra que no hace falta gastar mucho en tecnología para tener herramientas modernas de comunicación.

La forma de gestionar quién entra y qué puede hacer cada usuario es sencilla pero efectiva. En vez de usar sistemas de autenticación complejos que requieren servidores potentes, WiWiewCast usa una solución ligera basada en sessionStorage, que funciona bien en equipos con pocos recursos. Esto permite controlar el acceso sin complicar el sistema ni hacerlo más lento.

Una de las características más valoradas es la pizarra colaborativa. Los usuarios pueden dibujar, señalar y escribir sobre lo que ven en la pantalla, y cada uno tiene su propio color y patrón para que se distinga quién hace cada anotación. Esto hace que la experiencia sea más participativa y útil, especialmente en clases donde los estudiantes pueden interactuar directamente con el contenido. Además, todo esto se hace con tecnologías web estándar, sin necesidad de instalar programas extra ni usar equipos especiales.

La estructura del sistema está pensada para que cada parte (transmisión de video, gestión de usuarios, interacción en tiempo real, etc.) funcione de manera independiente pero conectada. Esto facilita el desarrollo, la corrección de errores y la posibilidad de añadir nuevas funciones en el futuro sin tener que cambiar todo el sistema ni comprar equipos nuevos. Además, el sistema incluye mecanismos para limpiar recursos automáticamente y desconectar usuarios inactivos, lo que ayuda a mantener la estabilidad y el buen funcionamiento incluso en equipos con pocos recursos.

Por todo lo anterior queda patente que se han alcanzado, de manera satisfactoria, todos los objetivos planteados al comienzo del desarrollo de este TFG.

3. Trabajo futuro

En un trabajo futuro, sería interesante mejorar el sistema de autenticación para que sea más seguro y flexible, permitiendo, por ejemplo, que un mismo usuario pueda conectarse desde varios dispositivos a la vez o que haya diferentes niveles de permisos (administrador, moderador, profesor, estudiante, etc.). También se podría añadir soporte para que varios dispositivos transmitan video al mismo tiempo, lo que sería útil cuando se necesiten varios ángulos de visión. Para esto, habría que desarrollar formas de mostrar varios videos a la vez (por ejemplo, en pantalla dividida o con una ventana pequeña sobre la principal).

Las herramientas de dibujo y anotación también se podrían ampliar, añadiendo más tipos de pinceles, formas geométricas, texto con formato y la posibilidad de trabajar en capas (por ejemplo, una capa para los profesores y otra para los estudiantes). Incluso se podría reconocer automáticamente lo que se dibuja y convertirlo en formas perfectas o en texto editable.

Otra mejora sería permitir que el administrador decida si los dibujos de los estudiantes son visibles para todos los participantes, no solo para el propio estudiante. Esto podría activarse selectivamente según las necesidades de la sesión y sería especialmente útil en los casos de uso uno y dos, donde actualmente los estudiantes solo ven sus propias anotaciones.

En resumen, WiWiewCast es una plataforma sólida, sencilla y accesible que ya permite hacer cosas avanzadas con pocos recursos. Las mejoras que se plantean para el futuro buscan hacerla aún más potente, flexible y fácil de integrar en distintos entornos, siempre manteniendo la simplicidad y la eficiencia que la hacen especial.

Bibliografía

[1] «WebRTC 1.0: Real-time Communication Between Browsers,» W3C, 2021. [En línea]. Available: https://www.w3.org/TR/webrtc/. [Último acceso: 15 de julio 2025]

[2] «Node-RED Documentation,» Foundation, Node-RED, 2024. [En línea]. Available: https://nodered.org/docs/. [Último acceso: 15 de julio 2025]

[3] «WebRTC Best Practices: Understanding STUN, TURN and ICE Servers,» Technologies, EcosMob, Medium, 2023. [En línea]. Available: https://medium.com/@ecosmobtechnologies/webrtc-best-practices-understanding-stun-turn-and-ice-servers-4836109904ec. [Último acceso: 15 de julio 2025]

[4] A. García Hernández, «Desarrollo de una aplicación web de videoconferencias basada en WebRTC,» Trabajo de Fin de Grado, Dep. Ingeniería Telemática, Universidad Carlos III de Madrid, Madrid, España, 2020. [En línea]. Available: https://e-archivo.uc3m.es/rest/api/core/bitstreams/5fc95880-9f07-41ff-a01c-b0c5783cd336/content. [Último acceso: 15 de julio 2025]

[5] «P2P, SFU and MCU WebRTC Architectures Explained,» DigitalSamba Blog, 2023. [En línea]. Available: https://www.digitalsamba.com/blog/p2p-sfu-and-mcu-webrtc-architectures-explained. [Último acceso: 15 de julio 2025]

[6] «Node.js Documentation,» Foundation, Node.js, 2024. [En línea]. Available: https://nodejs.org/en/docs/. [Último acceso: 15 de julio 2025]

[7] «Socket.IO Documentation,» Socket.IO, [En línea]. Available: https://socket.io/docs/.

[8] «Promise based HTTP client for the browser and node.js,» Axios, 2024. [En línea]. Available: https://axios-http.com/ [Último acceso: 15 de julio 2025].

[9] «Raspberry Pi 5,» Foundation, Raspberry Pi, 2023. [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-5/. [Último acceso: 15 de julio 2025]

[10] M. A. Zabalza, Competencias docentes del profesorado universitario para el siglo XXI, Madrid: Narcea, 2003.

[11] Agencia Nacional de Evaluación de la Calidad y la Acreditación, «ANECA,» mayo 2022. [En línea]. Available: https://www.aneca.es/documents/20123/81865/220106_Informe_RA-V3.pdf/f5988756-632f-db29-c27c-e7b14ad83a8e?t=1656326305105. [Último acceso: 15 Abril 2023]. [Último acceso: 15 de julio 2025]

[12] F. Trujillo Saez, Propuesta para una escuela en el siglo XXI, La Catarata, 2012.

[13] ANECA, 2013. [En línea]. Available: https://www.aneca.es/documents/20123/63546/learningoutcomes_v02.pdf/cc42ff7d-b416-5c32-860e-b3d0a5398f49?t=1654597704825. [Último acceso: 15 de julio 2025].

[14] L. M. S. Srinivas, «Harnessing Media Streams in WebRTC: Capturing and Managing Audio and Video,,» Medium, 2023. [En línea]. Available: https://medium.com/@lmssrinivas/harnessing-media-streams-in-webrtc-capturing-and-managing-audio-and-video-7d69ae43d4f5. [Último acceso: 15 de julio 2025]

[15] J. U. Núñez, «Desarrollo de una aplicación de videoconferencia usando WebRTC,» Proyecto Fin de Carrera, E.T.S. de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid, Madrid, España, , 2015. [En línea]. Available: https://oa.upm.es/37778/1/PFC_JORGE_ULLOA_NU%C3%91EZ_2015.pdf. [Último acceso: 15 de julio 2025]

Pliego de condiciones

El presente pliego de condiciones se estructura en dos apartados claramente diferenciados; las especificaciones técnicas del hardware y los requisitos técnicos del software, ambos necesarios para garantizar el correcto funcionamiento del sistema.

1. Especificaciones de hardware

A continuación, se detalla los componentes físicos necesarios para implementar el sistema WiWiewCast, incluyendo el servidor Raspberry Pi y el equipamiento de red. Las especificaciones garantizan el rendimiento óptimo para transmisiones de video en tiempo real con múltiples usuarios

Raspberry Pi

Componente	Especificación recomendada
Modelo	Raspberry Pi 5
RAM	8GB RAM
Almacenamiento	SSD NVMe 256GB con placa de adaptación (HAT)
Sistema operativo	Raspberry Pi OS 64-bit
Conectividad	Ethernet
Disipación	Ventilador Activo y disipador

Router para red local

Característica	Especificación recomendada
Estándar WiFi	802.11ac
Velocidad	1200 Mbs
Dispositivos	Soporte de al menos de 20 conexiones simultáneas

2. Requisitos de software

Este apartado define el entorno de software base necesario para ejecutar el sistema WiWiewCast. Incluye, por el lado del servidor, el sistema operativo, el runtime de Node.js, librerías específicas y Node-RED, y, por el lado de los clientes el software necesario para los dispositivos móviles.

Sistema operativo base

- Raspberry Pi OS , basado en Debian 11 Bullseye y Arquitectura ARM64
- Kernel Linux 5.10+

Runtime, dependencias y gestores de dependencias

Componente	Versión Recomendada
Node.js	18.x LTS
npm	9.x o superior
Node-RED	3.0.2 o superior
Socket.IO	4.7.0 o superior
Axios	1.4.0 o superior

Nodos Node-RED requeridos

- node-red-contrib-socketio: "^1.1.0",
- node-red-contrib-axios: "^1.3.0",

Dispositivos clientes

Tipo	Especificaciones
Móviles	Android 7+ o iOS 12+, cámara 1080p, RAM 3gb+, navegador actualizado

Tablets	**Android 8+ o iPadOS 13+, cámara 1080p, RAM 3gb+, navegador actualizado**
PC's	**Windows 10/MacOS 10.14/Linux Ubuntu18+, Chrome 90+**

2.1. Configuración de red local

La correcta configuración de los parámetros de la red local es importante y a la vez necesario para garantizar las conexiones estables entre los dispositivos. A continuación, se detalla un ejemplo:

- **Rango IP**: 192.168.1.0/24 (configurable)
- **Gateway**: 192.168.1.1 (router)
- **DHCP**: 192.168.1.100 - 192.168.1.200
- **IP Raspberry Pi**: 192.168.1.88 (estática reservada)

Presupuesto

En este apartado se exponen los costes correspondientes a la elaboración del proyecto. Teniendo en cuenta el contexto académico de este Trabajo Fin de Grado, se ha tomado como guía las directrices del Colegio Oficial de Graduados e Ingenieros Técnicos de Telecomunicación (COITT). Sin embargo, a la hora de la valoración de proyectos en el ejercicio libre de la profesión debe tenerse en cuenta “El Ministerio de Economía y Hacienda remitió a todos los colegios profesionales una nota en la que se nos recordaba que, siguiendo directivas europeas, se debían eliminar los baremos orientativos de honorarios que tradicionalmente veníamos publicando.”

Los costes se han dividido en las siguientes secciones:

- Materiales usados
- Trabajo tarifado por tiempo empleado
- Costes asociados a la redacción del Trabajo Fin de Grado
- Gastos derivados de los impuestos

1. Materiales usados

En este apartado, se examina la utilización de los elementos físicos y lógicos que sustentan el desarrollo del proyecto. Para calcular su coste, se ha empleado un método de depreciación lineal, el cual distribuye equitativamente la pérdida de valor de estos elementos a lo largo de su vida útil estimada. Si bien la vida útil estándar considerada es de 4 años, la duración real de este Trabajo de Fin de Grado ha sido de cuatro meses, lo que ha requerido un ajuste proporcional en los cálculos para reflejar este periodo específico.

Equipos

El Trabajo de Fin de Grado se completó en un periodo de cinco meses, un intervalo temporal significativamente menor en comparación con el periodo de varios años que comúnmente se utiliza como base para calcular la depreciación del equipamiento físico. En consecuencia, los gastos de amortización presentados corresponden exclusivamente al valor de uso durante estos cinco meses específicos.

Debido a la diferencia temporal del periodo de elaboración del TFG y el de vida útil del material usado, se ha tomado la relación entre los dos periodos para el cálculo de la amortización:

$$\text{Vida útil (meses)} = 4 \text{ años} * 12 \text{ meses}/1 \text{ año} = 48 \text{ meses}$$

$$\text{Valor amortizado} = 5 \text{ meses}/48 \text{ meses} * \text{Valor de adquisición}$$

La Tabla 1 detalla los elementos de hardware fundamentales utilizados en el proyecto, indicando tanto su precio de compra original como la porción de su valor que se ha depreciado durante el periodo de desarrollo.

Tabla 1 Tabla de amortización de recursos de hardware

Elemento	Valor de adquisición	Amortización
Portátil	2323,43 €	241,63 €
Raspberry Pi 5	118,99 €	12,37 €
SSD NVMe M2 256GB	39 €	4,056 €
Router Asus RT-AC86U	103,54 €	10,7 €
Raspberry Pi 5 HAT - SSD NVMe 256GB	47,90 €	4,89 €

El coste total de amortización de los materiales físicos es **doscientos setenta y tres con sesenta y cuatro céntimos** (273,64€).

Software

Para el software implementado en este Trabajo de Fin de Grado, la amortización se calcula considerando una utilización de 5 meses dentro de un ciclo de vida útil de 4 años.

Sin embargo, las herramientas de software seleccionadas han sido principalmente de naturaleza gratuita y open source, salvo Claude que tiene un coste mensual:

- IDE VS Code: Al ser usado su versión sin suscripción y no requerir pago de licencia todos los costes de amortización serán nulo.
- Windows 11: Al venir ya instalado en el portátil no requiere el pago de licencias por lo que todos los costes de amortización serán nulos.
- Claude.a (Plan Pro)i: Versión de pago para búsquedas profundas y propuestas de código con un coste (18€/mes * 5 meses) total de **noventa euros** (90€)
- Node-RED: Al ser Software Libre no requiere el pago de licencias por lo que todos los costes de amortización serán nulos.

En consecuencia, el coste total derivado del software utilizado es de **noventa euros** (90€).

2. Trabajo por tiempo empleado

En la ejecución de este proyecto se han dedicado aproximadamente 375 horas distribuidas entre las fases de diseño, desarrollo y creación de la documentación. De acuerdo con las directrices del COITT, la valoración económica del trabajo realizado puede determinarse mediante la siguiente fórmula:

$$H = Ct * 74.88 * Hn + Ct * 96.72 * He$$

- H: Importe total de honorarios correspondientes al proyecto
- Ct: Coeficiente de ajuste en función de las horas empleadas
- Hn: Horas desarrolladas durante jornada laboral ordinaria
- He: Horas desarrolladas en horario extraordinario (para este proyecto su valor es 0al no haberse registrado)

Considerando los baremos establecidos por el COITT, el coeficiente de ajuste correspondiente a las horas empleadas, según se especifica en la, equivale a 0,60.

Tabla 2. Valores del factor de corrección en función a las horas trabajadas

Horas empleadas	Factor de corrección Ct
X < 36	1
36 < X < 72	0,90

72 < X < 108	0,80
108 < X < 144	0,70
144 < X < 180	0,65
180 < X < 360	0,60
360 < X < 540	0,55

Conforme a dicha tabla, al completarse este proyecto en 375 horas, corresponde aplicar el coeficiente de ajuste con valor de 0.60. En base a esto, la expresión matemática anterior se establece de la siguiente forma:

$$H = 0.6 * 74.88 * 375 + 0.6 * 96.72 * 0 = 16.848\ €$$

Los honorarios resultantes del tiempo invertido en el proyecto, sin incluir impuestos, alcanzan un total de **dieciséis mil ochocientos cuarenta y ocho euros (16.848 €)**

Redacción de documentación

Con respecto al coste de la redacción del documento se utiliza la ecuación:

$$H = 0.07 * P * Cn$$

- R: Honorarios correspondientes a la elaboración del documento.
- P: presupuesto total del trabajo.
- Cn: coeficiente de ponderación determinado según el presupuesto.

El importe final resulta de la adición de los costes laborales facturados por tiempo invertido, previamente calculados, y la depreciación de recursos materiales cuyo total se detalla en la Tabla 3.

Tabla 3. Presupuesto del trabajo tarifado y amortización de los recursos materiales

Descripción	Costes
Amortización de recursos materiales	273,64 €
Trabajo tarifado por tiempo empleado	*16.848 €*

Total	17.121,64 €

Dado que el coeficiente de ponderación para presupuestos inferiores a 30.050,00€ se establece por el COITT en 1.00, el coste de elaboración documental del TFG resulta:

H =0,07 17.121,64 * 1= 1.198,51 €*

En definitiva, el coste de redacción del proyecto alcanza un importe de **mil ciento noventa y ocho euros con cincuenta y un céntimos (*1.198,51* €)**

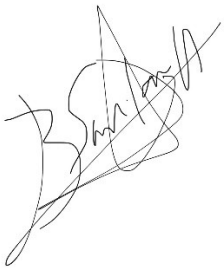
3. Aplicación de impuestos y coste total

A este Trabajo de Fin de Grado se le aplica el Impuesto General Indirecto Canario (IGIC), equivalente al 7% del importe presupuestario. El presupuesto global del proyecto se detalla en la Tabla 4.

Tabla 4.Tabla 24 Aplicación de impuestos a los costes

Concepto	Costes
Amortización de recursos materiales	273,64 €
Trabajo tarifado por tiempo empleado	*16.848 €*
Redacción de documentación	*1.198,51 €*
Subtotal (Sin IGIC)	18.320,15 €
IGIC (7%)	1.282,41 €
Total	19.602,56 €

El Trabajo de Fin de Grado denominado " WiWiewCast. Sistema de streaming interactivo multiusuario con WebRTC utilizando Raspberry Pi ", desarrollado en la Escuela de Ingeniería de Telecomunicaciones y Electrónica de la Universidad de las Palmas de Gran Canaria, presenta un coste total de desarrollo de **diecinueve mil seiscientos dos euros con 56 céntimos (19.602,56 €),** que corresponde a la suma de los importes asignados a los conceptos anteriormente especificados.



Firmado: Benito Santana Díaz Fecha:18/07/2025

Objetivos de Desarrollo Sostenible

Grado de relación del TFG con los objetivos de desarrollo sostenible

Tabla 5. Objetivos de desarrollo sostenible

ODS	Grado de relación con los ODS			
	0 No procede	1 Bajo	2 Medio	3 Alto
ODS 1 Fin de la Pobreza	◉	○	○	○
ODS 2 Hambre cero	◉	○	○	○
ODS 3 Salud y Bienestar	◉	○	○	○
ODS 4 Educación de calidad	○	◉	○	○
ODS 5 Igualdad de género	◉	○	○	○
ODS 6 Agua limpia y saneamiento	◉	○	○	○
ODS 7 Energía Asequible y no contaminante	◉	○	○	○
ODS 8 Trabajo decente y crecimiento económico	◉	○	○	○
ODS 9 Industria, Innovación e Infraestructuras	○	○	◉	○
ODS 10 Reducción de las desigualdades	◉	○	○	○
ODS 11 Ciudades y comunidades sostenibles	◉	○	○	○
ODS 12 Producción y consumo sostenibles	◉	○	○	○
ODS 13 Acción por el clima	◉	○	○	○
ODS 14 Vida submarina	◉	○	○	○
ODS 15 Vida de ecosistemas terrestres	◉	○	○	○
ODS 16 Paz, justicia e instituciones sólidas	◉	○	○	○
ODS 17 Alianzas para lograr objetivos	◉	○	○	○

Justificación del alineamiento del TFT con los ODS con los que se ha indicado que tiene un mayor grado de relación:

ODS04: **Educación de Calidad**

Mejora la calidad de la enseñanza mediante la transmisión de video en tiempo real con anotaciones interactivas, facilitando el acceso a contenidos educativos digitales.

ODS09: Industria, Innovación y Crecimiento Económico

Al emplear tecnologías como WebRTC y Node-RED, el proyecto fomenta el desarrollo de plataformas de comunicación en tiempo real eficientes, escalables y accesibles.

Anexo 1. Manuales

Este anexo tiene como objetivo establecer las condiciones técnicas, funcionales y de calidad para la ejecución del sistema de transmisión interactiva multiusuario denominado WiWiewCast, conforme a las especificaciones definidas en el pliego de condiciones, así como proporcionar un manual de usuario para su correcta operación.

1. Manual de instalación

La siguiente sección proporciona una guía para instalar el sistema desde cero. Incluye la preparación del Raspberry Pi, instalación de Node.js y Node-RED, configuración de certificados de seguridad, y establecimiento del sistema como servicio automatizado. El proceso culmina con la importación de archivos del sistema y las URLs de acceso para los diferentes roles de usuario.

El primer paso es descargar y ejecutar Raspberry Pi Imager: https://rpi.org/imager desde cualquier equipo. Este programa nos ayudará a instalar el sistema operativo de la Raspberry PI dentro de una microSD.

Durante el proceso de ejecución de `Imager`:

- Descargar Raspberry Pi OS (64-bit recomendado).
- Habilitar SSH.
- Habilitar usuario y contraseña (ejemplo “usuario: pi, contraseña: pi”)
- Escribir imagen en microSD

1.1. Primera configuración

A continuación se detallan las acciones a ejecutar para la configuración inicial:

- Insertar la tarjeta microSD y arrancar Raspberry Pi
- Conectar Raspberry Pi a la red local del router WiFi mediante cable Ethernet. Utilizar otro dispositivo conectado a la misma red para saber la IP asignada a la Raspberry a través del comando:

```
$> ping raspberrypi.local
```

- Utilizar la IP para conectarse a través de SSH.

```
$> ssh pi@192.168.1.XX
```

- Actualizar sistema

```
pi@raspberrypi:~$> sudo apt update && sudo apt upgrade -y
```

- Configurar la IP estática

```
$> sudo nmtui
```

1.2. Instalar Node.js y Node-RED

Una vez completadas las acciones anteriores, procedemos a instalar **Node.js** y el entorno gráfico Node-RED.

- Instalar Node.js (versión 18 LTS)

```
$> curl -fsSL https://deb.nodesource.com/setup_18.x
$> sudo apt-get install -y nodejs
```

- Instalar Node-RED globalmente

```
$> sudo npm install -g --unsafe-perm node-red
```

- Instalar dependencias WebRTC

```
$> sudo npm install -g node-red-contrib-socketio sudo npm install
   -g wrtc
```

1.3. Crear certificados autofirmados

Para que los navegadores modernos permitan el uso de dispositivos multimedia como la cámara o el micrófono, es necesario que la conexión esté protegida mediante SSL (HTTPS). A continuación, se detallan los pasos para instalar certificados autofirmados.

- Crear directorio para certificados

```
$> sudo mkdir -p /home/pi/ssl/
$> cd /home/pi/ssl/
```

- Generar clave privada

```
$> sudo openssl genrsa -out server.key 2048
```

- Generar certificado autofirmado (válido por 365 días)

```
$> sudo openssl req -new -x509 -key server.key -out server.crt
-days 365
```

- Configurar Permisos

```
$> sudo chown root:ssl-cert /home/pi/ssl/server.key
$> sudo chown root:ssl-cert /home/pi/ssl/server.crt
$> sudo chmod 640 /home/pi/ssl/server.key
$> sudo chmod 644 /home/pi/ssl/server.crt
```

- Añadir usuario pi al grupo ssl-cert

```
$> sudo usermod -a -G ssl-cert pi
```

1.4. Configurar Node-RED

El paso siguiente será configurar el entorno de trabajo Node-RED para adaptarlo a la conexión protegida (SSL/HTTPS).

- Crear directorio de trabajo

```
$> mkdir ~/.node-red | cd ~/.node-red
```

- Configurar settings.js

```
$> nano settings.js
```

```
javascriptvar fs = require("fs");
module.exports = {
    // Configuración HTTPS
    https: {
        key: fs.readFileSync('/etc/ssl/wiwicast/server.key'),
        cert: fs.readFileSync('/etc/ssl/wiwicast/server.crt')
    },
    // Directorio para archivos estáticos
   httpStatic: '/home/pi/nodered/public',
    // Puerto para HTTPS
    uiPort: process.env.PORT || 1880,
    // Resto de configuración existente...
    functionGlobalContext: {
        wrtc: require('wrtc'),
        os: require('os'),
    },
    // Confiiguración de seguridad
     requireHttps: true
}
```

1.5. Configurar Node-RED como servicio

Una vez modificado el archivo settings.js, procederemos a configurar Node-RED **como un** servicio del sistema, de modo que se inicie automáticamente cada vez que arranque el sistema operativo.

- Crear servicio

```
$> sudo nano /lib/systemd/system/nodered.service;
```

```
[Unit]
     Description=Node-RED
     After=syslog.target network.target

[Service]
     ExecStart=/usr/bin/env node-red-pi --max-old-space-size=256
     Restart=on-failure
     KillSignal=SIGINT
     SyslogIdentifier=node-red
     User=pipi

[Install]
     WantedBy=multi-user.target
```

- Habilitar Node-RED como servicio

```
$> sudo systemctl enable nodered.service
$> sudo systemctl start nodered.service
$> sudo systemctl status nodered.service
```

1.6. Importar Configuración

Una vez instalado todo el ecosistema, solo queda descargar los archivos de la aplicación y alojarlos en el entorno correspondiente.

- Descargar archivos del repositorio GitHub: https://github.com/benitosd/tfg.git

```
$> cd ~/.node-red (Copiar flows.json a la Raspberry Pi)
$> cd ~/nodered/public
```

- Copiar archivos HTML (index.html, viewer.html, admin.html, login.html)
- Copiar Carpetas JS, CSS y todo su contenido

1.7. URLs de Acceso

Una vez el sistema configurado y reiniciado se puede acceder a los diferentes perfiles a partir de las URLS indicadas a continuación:

- **Login:** https://192.168.1.88/login.html
- **Emisor:** https://192.168.1.88/index.html
- **Viewer:** https://192.168.1.88/viewer.html
- **Admin:** https://192.168.1.88/admin.html
- **Node-RED:** https://192.168.1.88:1880

2. Manual del Usuario

El sistema se organiza en tres roles principales que trabajan en conjunto para crear una experiencia de transmisión dinámica, como los siguientes:

- El **Emisor** se encarga de transmitir video en vivo desde su dispositivo, controlando la cámara y la transmisión.

- Los **Viewers** (o espectadores) pueden acceder a la transmisión en tiempo real y participar activamente dibujando sobre el video con colores y patrones únicos que se les asignan automáticamente, creando una capa de interacción colaborativa.
- El **Administrador** supervisa todo el sistema: gestiona a los usuarios conectados, modera el contenido, controla las capas de dibujo y monitorea el rendimiento general. A continuación, se detallan los manuales específicos para cada rol.

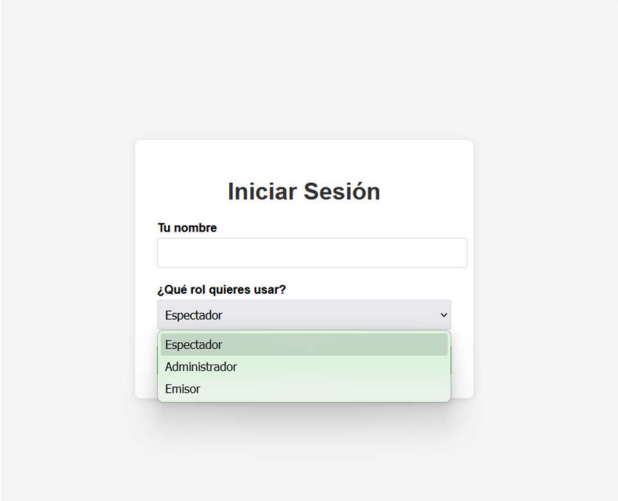


Figura 19. Página de autenticación

La elección de este rol se realiza al autenticarse en la aplicación. El proceso incluye los siguientes pasos:

- Conectar el dispositivo al WiFi de WiWiewCast
- Abrir navegador y escribir la siguiente url: https://192.168.1.88/login.html
- Iniciar sesión eligiendo el rol correspondiente en el desplegable y añadir un nombre para identificación.

2.1. Manual del Emisor

Cuando el interlocutor actúa como emisor transmite video en vivo desde cualquier cámara de su dispositivo. Puede cambiar entre cámaras (frontal o trasera en el caso de un dispositivo móvil) y elegir la orientación horizontal o vertical de la transmisión. Solo se necesita que el dispositivo, con cámara y navegador web, esté conectado al WiFi de WiWiewCast y tener los permisos de cámara habilitados en el navegador.

Las acciones posibles con este perfil son:

- Configuración de la cámara
  - Desplegable "Seleccionar cámara" - lista todas las cámaras disponibles
  - Cambio de cámara: Solo posible cuando NO se está transmitiendo
  - Orientación de Vídeo.
- Para iniciar la transmisión, los pasos a seguir son:
  - Verificar preview - debe verse el video de la cámara
  - Clic en "Iniciar Stream", después de esto el botón cambia a "Detener Stream" y la etiqueta muestra un indicador con el texto "En vivo".

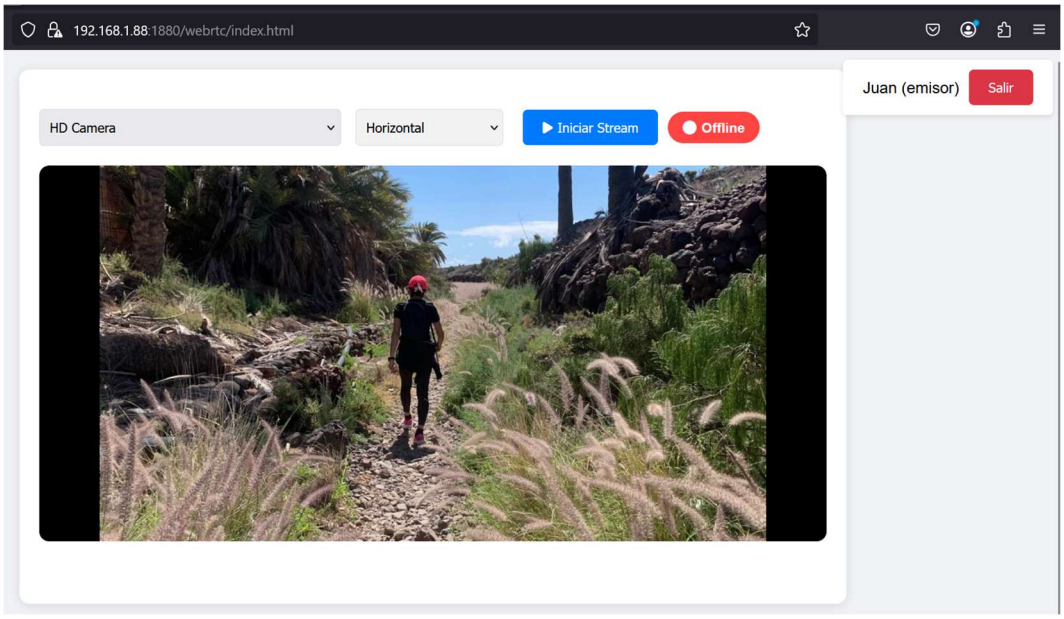


Figura 20. Modo Emisor antes de empezar a transmitir

- Para finalizar la transmisión debemos:
  - Clic en "Detener Stream"
  - **Confirmación automática** - se liberan recursos
  - **Vuelta al estado inicial** botón cambia a "Iniciar Stream"

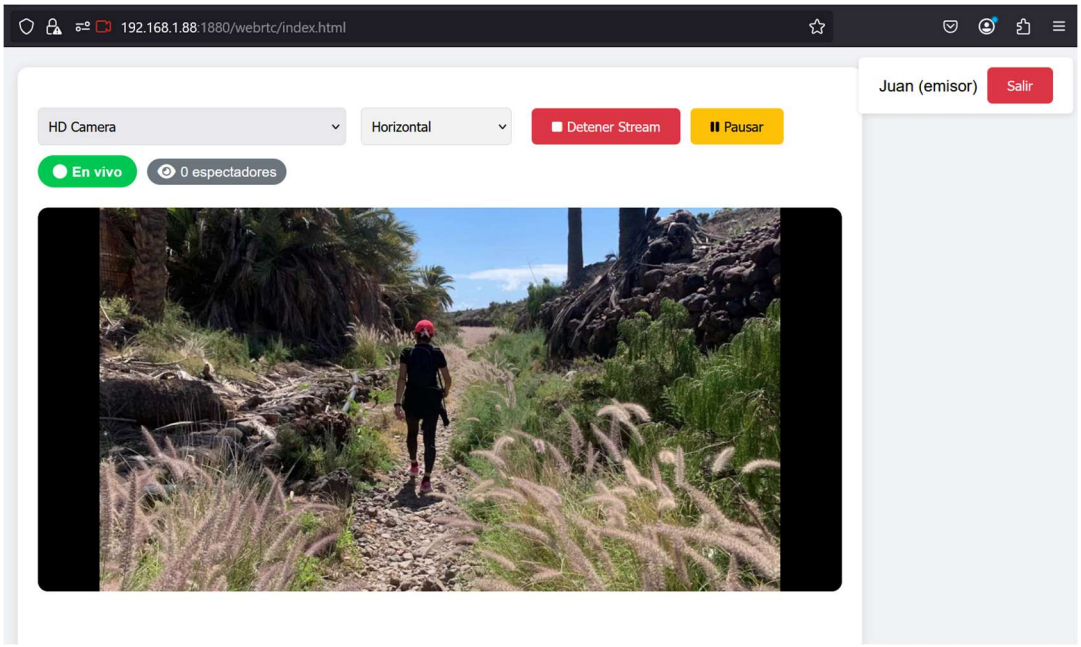


Figura 21. Modo Emisor en estado emitiendo

2.2. Manual del Viewer (Espectador)

Cuando nos logueamos como Viewer o espectador, el usuario puede ver el video en tiempo real y participar activamente en la experiencia colaborativa mediante el sistema de dibujo interactivo. Cada espectador recibe automáticamente un color y patrón único que permite identificar sus contribuciones en la transmisión.

Con este perfil, las acciones posibles incluyen la visualización de video, donde puedes hacer clic en “Ver Stream” y disfrutar de una calidad adaptativa que se ajusta automáticamente según tu conexión.

En cuanto al sistema de dibujo colaborativo, al conectarte recibes automáticamente los elementos que identifican al usuario, incluyendo un color único y un patrón distintivo en la forma del píxel.

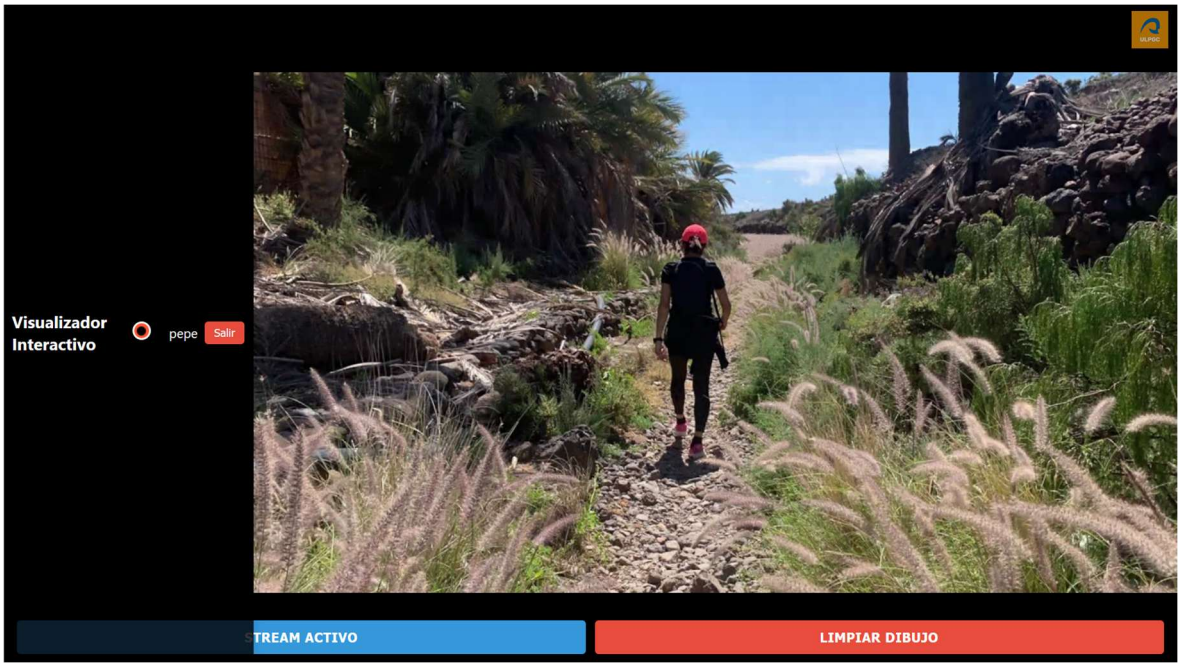


Figura 22. Modo Viewer recibiendo video

Para **dibujar**, simplemente toca o haz clic sobre el video, y arrastra para crear líneas y formas; todos los usuarios verán tus dibujos en tiempo real, incluidos los administradores desde su panel.

En cuanto a la gestión de dibujos, dispones de un botón para “Borrar mi dibujo”, que elimina únicamente tus trazos. Este cambio es inmediato y notifica al administrador quién ha realizado la acción. No puedes borrar los dibujos de otros, pero puedes eliminar y redibujar los tuyos tantas veces se desee.

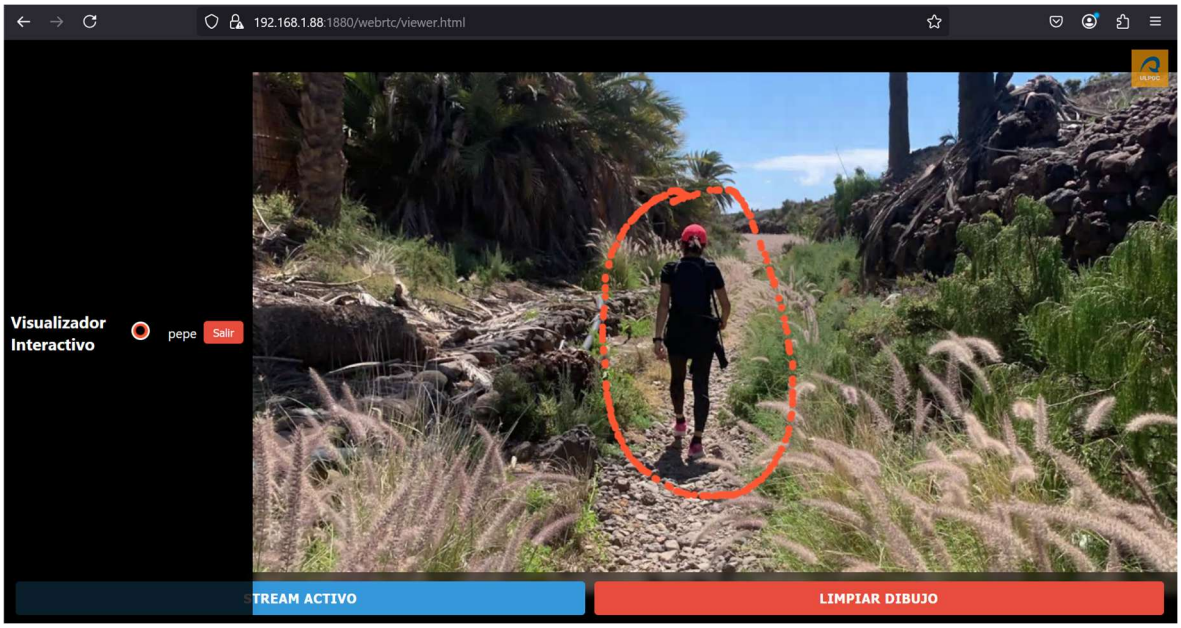


Figura 23. Modo Viewer dibujando

Los controles en los dispositivos táctiles permiten dibujar con un dedo sin interferencia de gestos como el zoom o el pinzado, ya que toda la superficie del video está habilitada como área de dibujo. En ordenadores, los controles con ratón permiten dibujar con el clic izquierdo, mantener presionado para trazos continuos, y mover el puntero sin hacer clic no genera dibujo.

2.3. Manual del Administrador

Como **Administrador** en WiWiCast, tienes control total sobre el sistema de transmisión colaborativa, supervisando todos los usuarios conectados, gestionando las capas de dibujo interactivo, moderando contenido en tiempo real y monitoreando el rendimiento general del sistema. Además, puedes alternar entre diferentes modos de uso para adaptarte a las necesidades de la sesión.

En el panel de control, dispones de acceso a estadísticas generales, como el número total de espectadores conectados (**Viewers Conectados**), los que están visualizando activamente la transmisión (**Viewers Activos**) y las **métricas de red**, que te muestran el uso de ancho de banda en tiempo real.

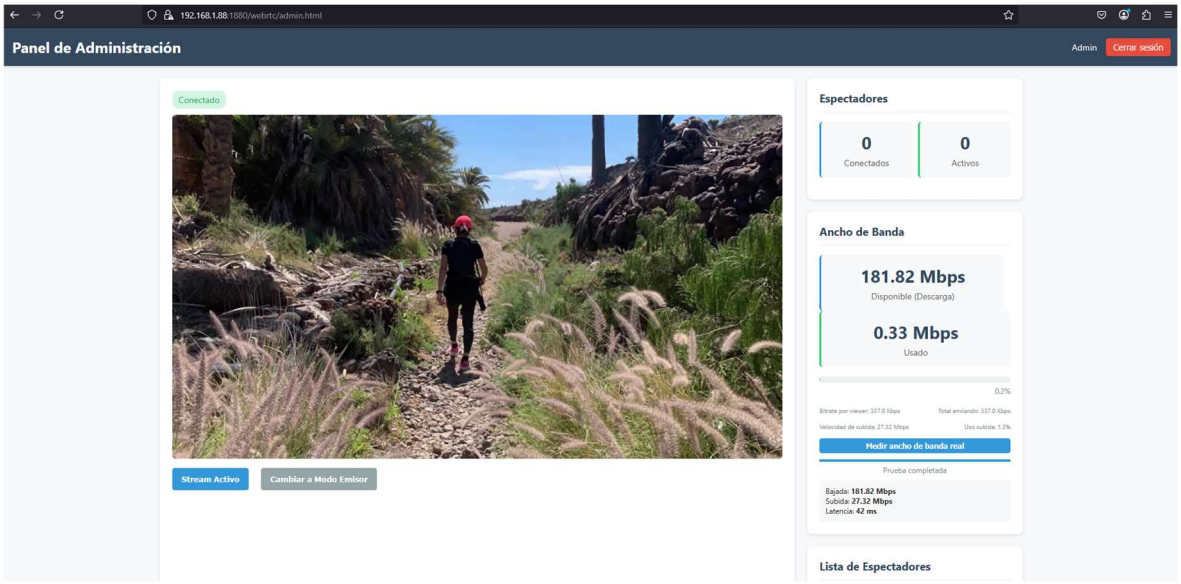


Figura 24. Modo Admin recibiendo emision

La gestión de viewers te permite identificar a cada usuario mediante su nombre o, así como ver el color y patrón asignados automáticamente por el sistema.

En cuanto a la **visualización de capas**, cada usuario tiene su propia capa de dibujo, la cual puedes mostrar u ocultar de forma independiente. Esta visibilidad individual no afecta a la integridad de los dibujos: al ocultar y volver a mostrar una capa, los trazos permanecen intactos. Además, puedes borrar únicamente los dibujos de un usuario concreto con el botón correspondiente, o usar la función **"Resaltar"** para mostrar solo sus trazos durante 2 segundos, facilitando la moderación o el análisis visual.

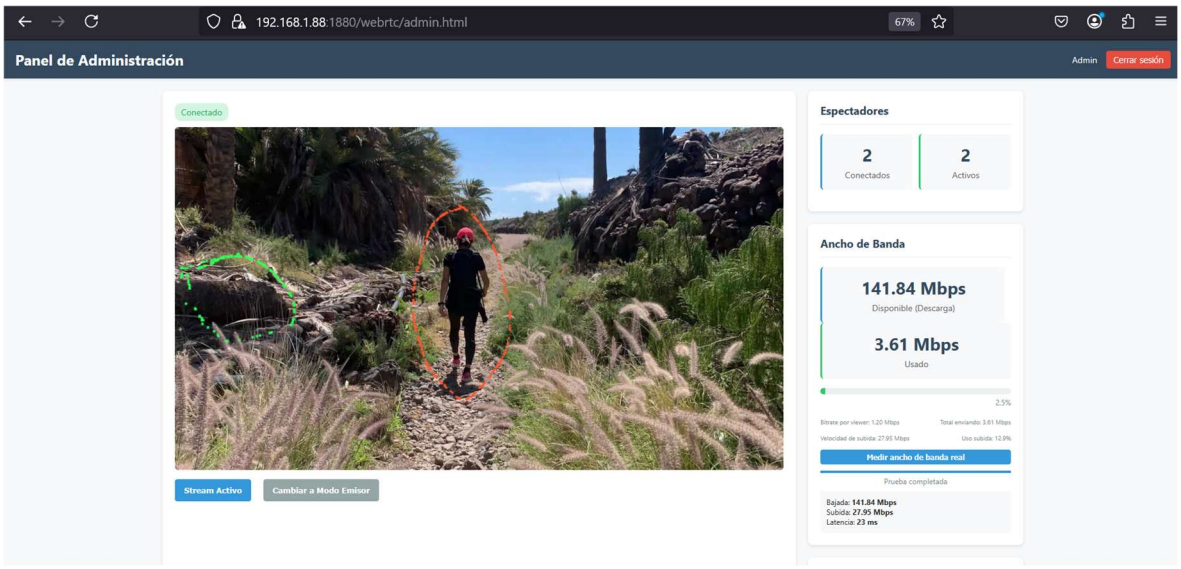


Figura 25. Modo admin recibiendo video y dibujos

Finalmente, el sistema incluye un **Modo Dual (Administrador + Emisor)** que permite al administrador convertirse también en emisor de contenido. Al pulsar el botón "Cambiar a Modo Emisor", se activan los controles de transmisión, como el manejo de la cámara y el inicio del streaming, sin perder las funciones de administración. Puedes volver al modo exclusivamente administrativo, deteniendo la transmisión en cualquier momento y utilizando el botón "Cambiar a Modo Admin".

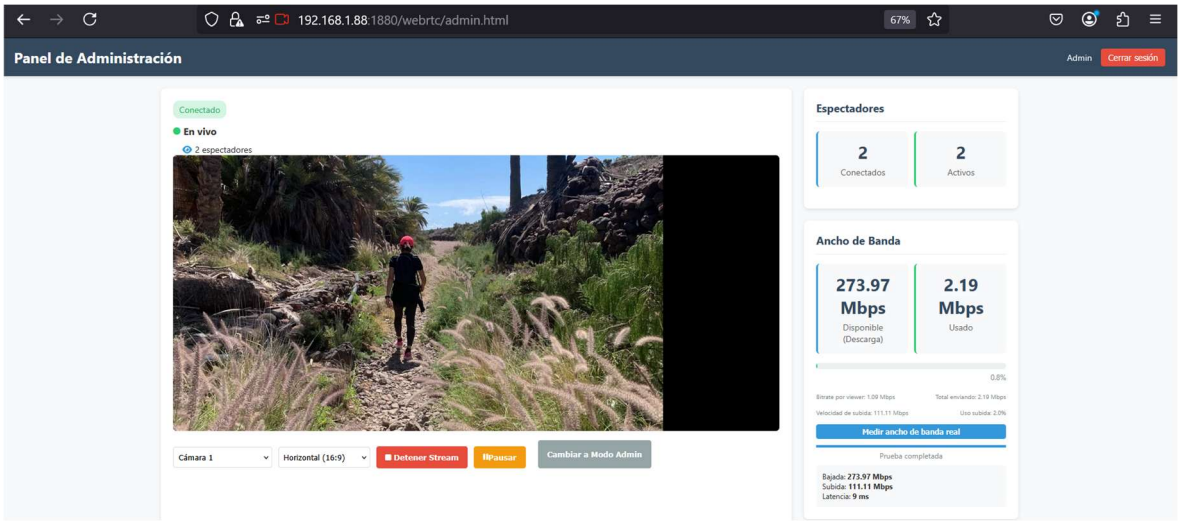


Figura 26. Modo Admin emitiendo video