SENDERO

UN SISTEMA DE ILUMINACIÓN PARA PRODUCCIÓN ARTÍSTICA

Informe de Proyecto de Grado presentado al Tribunal Evaluador como requisito de graduación de la carrera Ingeniería en Computación.

Autores

Christian Bouvier

Diego Ernst

Alvaro Larrosa

Tutor

MSc. Ing. Christian Clark

Instituto de Computación

Facultad de Ingeniería

Universidad de la República

Montevideo – Uruguay

Julio 2016

Resumen

En los últimos años, se ha demostrado un gran interés por el desarrollo de arte con nuevos medios. Impulsado por la amplia accesibilidad tecnológica y la voluntad por explorar nuevas formas de expresión artística, cientos de nuevas creaciones son presentadas cada año en los diversos festivales, eventos y conferencias que tienen lugar a lo largo de todo el mundo.

Sendero, es un sistema de iluminación LED RGB, *open-source* y *open-source hardware*, que ha servido como herramienta para el desarrollo de este tipo de obras, y gracias a la gran difusión y éxito de sus resultados se identificados una serie de aspectos a mejorar, siendo estos la principal motivación del presente proyecto.

Como parte del mismo, fueron estudiadas y aplicadas técnicas de transferencia de video sobre redes de datos y comunicación *Machine-To-Machine*, se migró el esquema de comunicación de *hardware* hacia uno inalámbrico, se desarrollaron aplicaciones que permiten la interacción con el sistema de manera remota y se llevaron a cabo pruebas que garantizan el funcionamiento del sistema bajo un rango de aplicaciones que extiende ampliamente las posibilidades anteriormente ofrecidas.

Estos desarrollos implicaron la modificación y creación de múltiples componentes de *hardware* y *software*, que son entregados como una herramienta *open-source y open-source hardware* brindada por el Laboratorio de Medios de la Facultad de Ingeniería de la Universidad de la República.

Palabras Clave

Arte con nuevos medios , *streaming* multimedia, comunicación M2M, iluminación LED, WebSockets, arte interactivo, *open-source*.

Índice de Contenidos

Índice de Tablas

Índice de Figuras

Lista de Acrónimos

AP Access Point
API Application Programming Interface
BSS Basic Service Set
DIY Do It Yourself
DMX Digital MultipleX
DTIM Delivery Traffic Indication Message
ESTI European Telecommunications Standards Institute
GPIO General-Purpose Input-Output Port
HLS HTTP Live Streaming
HTTP Hyper Text Transfer Protocol
IEEE Institute of Electrical and Electronics Engineers
IETF Internet Engineering Task Force
IoT Internet of Things
IP Internet Protocol
IPTV Internet Protocol Television
ITU International Telecommunication Union
JPEG Joint Photographic Experts Group
LED Light-Emitting Diode
LR-WPAN Low Rate Wireless Area Network
M2M Machine-To-Machine
MINA Network Management - Artificial Intelligence
MPEG Moving Picture Experts Group
PAN Personal Area Network
PCB Printed Circuit Board
PWM Pulse Width Modulation
QoE Quality of Experience
QoS Quality of Service
RFC Request For Comments
RGB Red-Green-Blue
RTCP Real Time Control Protocol
RTP Real Time Protocol
RTSP Real Time Streaming Protocol
SAP Session Announcement Protocol
SDK Software Development Kit
SDP Session Description Protocol
SIP Session Initiation Protocol
SLP Sendero Lighting Protocol
SPI Serial Peripheral Interface
SWP Sendero Web Protocol
TCP Transport Control Protocol
UDP User Datagram Protocol
URL Unified Resource Locator
USB Universal Serial Bus
W3C World Wide Web Consortium
WHATWG Web Hypertext Application Technology Working Group
Wi-Fi Wireless Fidelity
WiMax Worldwide Interoperability for Microwave Access
WLAN Wireless Local Area Network
WSN Wireless Sensor Network
WWLAN Wide Wireless Local Area Network

Capítulo 1

Introducción

1.1 Descripción del problema

Desde hace algunas décadas, es común oír hablar sobre Arte con Nuevos Medios, Arte Multimedia, Arte Electrónico, o Arte Interactivo [1] entre otro conjunto de términos, cuyo objetivo principal es definir de manera más o menos específica, un mismo concepto: la utilización de la tecnología como medio para la producción artística.

En su libro "*The language of New Media*" [2], Manovich define "Arte con Nuevos Medios" como toda aquella expresión que involucra la utilización de una computadora en alguna etapa del proceso de comunicación, incluyendo adquisición, manipulación, almacenamiento o distribución. Por otra parte, Laurenzo describe en su tesis de doctorado [3] que existen otros conceptos, como la "explicitación" del uso de la tecnología, que deben ser incluidos en la definición de Arte con Nuevos Medios.

Dada la multiplicidad de medios tecnológicos existentes, y naturalmente, de la producción artística, resulta evidente que el Arte con Nuevos Medios (como será referido en este documento) sea un campo diverso y dinámico, que evoluciona constantemente a medida que nuevas formas de expresión son adquiridas y utilizadas, lo que se encuentra directamente vinculado al incesante desarrollo de nuevas tecnologías [4]. Esto da lugar a una gran diversidad de formas de expresión, cuyo denominador común es la apropiación de la tecnología como herramienta para la producción artística, tarea que implica un importante esfuerzo para descontextualizar y reinterpretar el conocimiento tecnológico, transformándolo en un medio para el desarrollo artístico [3].

Es aquí, donde tienen lugar herramientas como Sendero [5]-[6]: un sistema de iluminación LED RGB para la producción artística, cuyo objetivo principal es brindar un *framework* de alto nivel que otorga a los artistas un mayor grado de abstracción tecnológica, permitiéndoles concentrar su esfuerzo en los aspectos más específicos de su obra. Para ello, Sendero resuelve y encapsula aspectos técnicos como la arquitectura general del sistema y la comunicación e interacción entre sus componentes, brindando además mecanismos de interacción extensibles y personalizables.

Bajo estos lineamientos, se identificaron aspectos de Sendero que pueden ser mejorados en pos de transformarlo en una herramienta más versátil, extensible y escalable, y es el objetivo principal de este proyecto atacar algunos de éstos.

1.2 Descripción del proyecto base

Sendero es un sistema de iluminación, *open-source* [7]-[8] y *open hardware* [9], desarrollado en primera instancia por Christian Clark, Tomás Laurenzo, Pablo Gindel y Germán Hoffman, en el marco de la construcción de obras de arte con nuevos medios.

Se trata de un sistema de iluminación integral, abarcando un conjunto de aplicaciones cliente-servidor para la generación de contenidos visuales, el desarrollo de un dispositivo Art-Net DMX [10] basado en la plataforma de microcontroladores mBed [11] y los componentes de *hardware* necesarios para el control de LEDs RGB de alta potencia.

En cuanto al *software*, el servidor se encarga de gestionar las características generales de la obra, recibir información de color desde los clientes, mezclarla, y transmitir el resultado hacia el *hardware* y posibles espectadores remotos. Por su parte, los clientes generan patrones visuales basados en estímulos físicos y/o virtuales que definen el comportamiento de la obra, y los transmiten hacia el servidor.

Tanto el servidor como los clientes se encuentran desarrollados en el lenguaje C++, bajo la plataforma OpenFrameworks [12], y son distribuidos bajo la licencia MIT [13].

Por otra parte, el *hardware* se compone básicamente de dos tipos de dispositivo:

1. Microcontroladores mBed [11] emulando dispositivos DMX [14], y sirviendo de intermediarios entre el servidor y los controladores de LED.
2. Controladores de LED, congregados en componentes denominados Bondibar [9], [15], capaces de controlar hasta ocho tiras de LED RGB de un máximo de cinco metros de largo cada una.

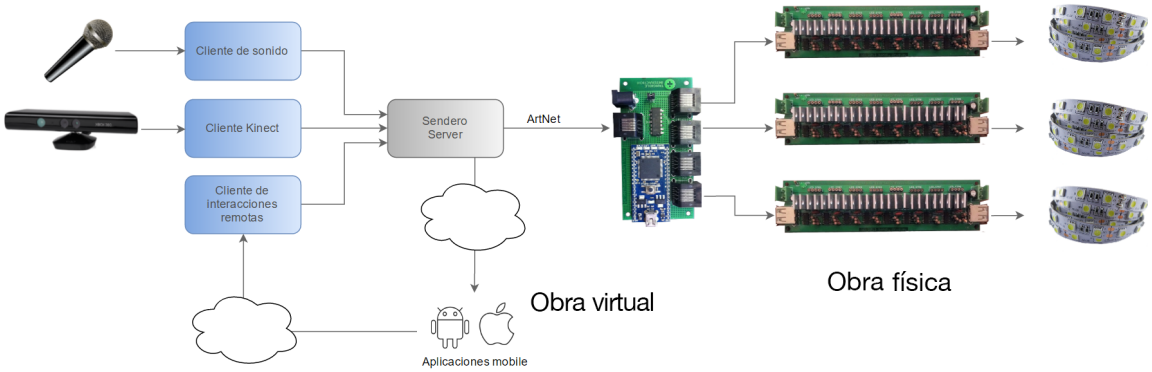


Figura 1-1 - Arquitectura proyecto base

La Figura 1-1 muestra la arquitectura de la solución de hardware/software de Sendero. Una serie de clientes se encargan de obtener estímulos a partir del uso de sensores como ser cámaras, micrófonos o incluso provenientes de usuarios remotos. Estos estímulos son procesados y transmitidos como un patrón de iluminación en forma de un *array* de colores hacia el servidor (Sendero Server). Los mismos son enviados a una tasa de 24 o 30 *fps* para permitir que los patrones visuales generados muestren animaciones y movimientos fluidos sobre los elementos de la obra, denominados píxeles. El servidor mezcla estos patrones y los transmite en simultaneo hacia la **obra física**, compuesta por la solución de hardware y las luces LED, y por otra parte hacia una **obra virtual** que puede ser visualizada como una representación tridimensional de la obra física en la pantalla de un dispositivo móvil (Android y iOS) a través de Internet.

Cada píxel, ya sea físico (luz LED) o virtual (representación en aplicación móvil) necesita conocer en cada momento cual es el color RGB que debe mostrar, de este modo, el flujo de datos de Sendero puede ser interpretado como una imagen descompuesta, donde el color de cada píxel de la imagen se muestra en un píxel de la obra. Dado que esta imagen se refresca a una tasa 24 o 30 *fps* el flujo de datos puede ser visto como un video que se despliega sobre la obra.

En la Sección 3.1, se abordará más en detalle el funcionamiento de esta solución, para luego discutir sobre sus limitaciones y así derivar los requerimientos que se pretenden cubrir con el desarrollo de este proyecto.

1.3 Motivación

La primera versión de Sendero fue desarrollada durante la creación de Celebra [16]–[18], una instalación interactiva comisionada por el Gobierno uruguayo durante los festejos del Bicentenario [19], y desarrollada en conjunto por el Laboratorio de Medios de la Facultad de Ingeniería de la Universidad de la República [20] y el colectivo en diseño de interacción Bondi [21].

Celebra, comprende una red de doscientos globos suspendidos e iluminados desde su interior por LEDs RGB. Esta instalación, ha sido exhibida en repetidas oportunidades desde su creación en 2011, mayormente en Uruguay, aunque también fue presentada en Sídney, Australia, durante el desarrollo del 19° Simposio de Arte Electrónico [22] llevado a cabo en Mayo de 2013. Adicionalmente, Celebra recibió una mención en la conferencia "Laval Virtual 2013" en Francia [23].

Más tarde, Sendero fue mejorado y utilizado para la construcción de una segunda obra, Barcelona [24], [25]. Una escultura interactiva con forma de dodecaedro *pentakis* cuyas aristas se iluminan independientemente en respuesta a estímulos externos. Barcelona fue comisionada por el programa Uruguay Encendido para la inauguración de la conferencia

Singularity University [26], desarrollada en abril de 2013 en el Hotel *Sofitel Montevideo Casino Carrasco*.

Luego del éxito de estos desarrollos, sin duda vinculado a la calidad del trabajo llevado a cabo por sus autores, es indiscutible que Sendero tiene un gran potencial como herramienta para la producción artística, existiendo muchas instancias de aplicación. Esto, sumado a las posibilidades de mejorar y extender el sistema en muchos aspectos, son la motivación principal para el desarrollo de este proyecto de grado.

1.4 Objetivos

Los objetivos específicos del proyecto son:

- **Realizar estudios del estado del arte en transferencia de video sobre redes de datos.** Dada la naturaleza de la información que maneja Sendero, el estudio de técnicas de transferencia de video puede impulsar la reformulación de los mecanismos de transferencia de datos existentes, basado en protocolos y técnicas que han sido objeto de estudio por parte de la comunidad científica.

- **Realizar estudios del estado del arte en transferencia de datos sobre redes Machine-To-Machine (M2M).** Con el creciente interés por la interconexión de objetos a Internet y la reciente explosión de la corriente *DIY* (*Do it Yourself*), son varias las alternativas que se tienen a la hora de comunicar dispositivos entre sí. Se requiere del estudio de las herramientas existentes para evaluar su aplicación en Sendero.

- **Desarrollo de prototipos de los componentes de hardware de Sendero que deban transmitir información de forma inalámbrica.** Con el punto anterior cumplido, se deberían tener las herramientas adecuadas para seleccionar de manera fundamentada los dispositivos y técnicas que permitan implementar una comunicación inalámbrica entre los controladores de LED y el servidor que procesa los patrones de iluminación. Se requieren prototipos de *hardware* que hagan efectiva esta comunicación inalámbrica.

- **Construcción de una interfaz web basada en WebSockets.** Una parte importante de Sendero es la interacción con los espectadores, tanto presenciales como remotos. Para que lo segundo pueda llevarse a cabo de manera efectiva, se plantea la investigación de la tecnología de WebSockets, y la construcción de una interfaz web capaz de *renderizar* en tiempo real una representación gráfica de la obra, al mismo tiempo que permita la interacción con la misma.

- **Desarrollo y construcción de una obra de arte con nuevos medios que utilice el sistema.** Una vez realizados los puntos anteriores se debe construir

una obra de arte con nuevos medios que aproveche las nuevas virtudes del sistema.

- **Posicionamiento de Sendero como una herramienta de producción artística *open-source* y *open-source hardware* provista por el Laboratorio de Medios de la Facultad de Ingeniería de la Universidad de la República.** Los desarrollos llevados a cabo en este proyecto deben ser publicados para su libre utilización.

1.5 Estructura del informe

En el **Capítulo 2**, se exponen los resultados del estudio del estado del arte en técnicas de transferencia de video sobre redes de datos, y comunicación *Machine-To-Machine* (M2M), enfocado en su posible aplicación a Sendero.

El **Capítulo 3**, brinda una descripción de la versión de Sendero tomada como base para el desarrollo de este proyecto, presentando sus limitaciones y los aspectos a mejorar identificados. Luego, se detalla el alcance del proyecto en términos de los productos que deben ser desarrollados.

El **Capítulo 4**, presenta la solución propuesta en términos de diseño e implementación. Se ofrece una contextualización de Sendero en el marco de los estudios realizados en el Capítulo 2, se detallan las decisiones de diseño y los aspectos más importantes de la implementación de cada componente del sistema.

El **Capítulo 5** describe una serie de evaluaciones experimentales y los resultados obtenidos.

Finalmente, en el **Capítulo 6** se discuten las conclusiones del trabajo y se postulan una serie de tareas para trabajo futuro.

Capítulo 2

Estado del Arte

En este capítulo se abordará el estudio del estado del arte en transferencia de video sobre redes de datos y comunicación Machine-to-Machine. Con esto, se pretende encontrar herramientas que permitan desarrollar de manera fundamentada, mecanismos para migrar la comunicación hardware de Sendero a un esquema inalámbrico, a la vez que se utilizan técnicas de streaming de video para asegurar la calidad de la reproducción, tanto en la instalación física como en la interfaz web a desarrollar.

2.1 Introducción

Los dos grandes temas que son presentados en este capítulo, son en sí mismos suficientemente amplios como para escribir libros enteros. Es por esto, que el nivel de detalle con el que serán desarrollados se encuentra enfocado en la consecución de los objetivos planteados en la Sección 1.4.

Por otra parte, a lo largo de este documento será utilizados un conjunto de términos cuyo significado está fuertemente influenciado por el contexto del proyecto. En la Sección 2.2 se introducen y definen estos términos que requieren de explicación contextualizada.

En las secciones 2.3 y 2.4 se presentan los estudios sobre el estado del arte en transferencia de video y comunicación *Machine-To-Machine* respectivamente.

Finalmente, se presenta un breve resumen destacando los elementos relevados con potencial aplicación a Sendero y una visión de los capítulos que continúan.

2.2 Términos y conceptos

En el contexto de Sendero, y en particular de este documento, se hará alusión a los siguientes términos bajo la definición que sigue.

- **Obra o instalación.** Hace referencia a la instalación física y/o virtual que ha sido (es, o será) desarrollada utilizando a Sendero como herramienta.
- **Píxel.** Sendero ha sido diseñado para cierto tipo de instalaciones u obras, donde existe un elemento constitutivo bien definido que se repite [6]. Cada uno de estos elementos será referido como *Píxel*.

- **Sendero Server.** Es el componente de *software* principal de Sendero. Encargado de mezclar y distribuir los patrones de iluminación que alimentan la obra (ver Sección 3.1.2.1).
- **Sendero Client.** Es el componente de *software* de Sendero cuyo rol es aportar los patrones de iluminación a Sendero Server (ver Sección 3.1.2.2).
- **LED, tira LED.** Componente electrónico emisor de luz. Los *píxeles* de Sendero son agrupaciones (tiras) de estos componentes.
- **LED RGB.** Es un LED capaz de emitir cualquier color a partir de su especificación en el formato RGB.
- **LED driver.** Dispositivo electrónico capaz de controlar un LED, haciendo que este cambie su color.
- **PCB.** Del inglés "*Printed Circuit Board",* hace referencia a placas de circuito impreso.
- **Bondibar.** Componente de *hardware* desarrollado por Pablo Gindel en el marco de la construcción de Celebra [16], [17] y Barcelona [15], [24]. Es una parte fundamental de Sendero, encargada de agrupar y conectar *LED Drivers*.
- **DMX o DMX512.** Es un protocolo electrónico utilizado para el control de iluminación. Permite la comunicación entre los equipos de control y los propios componentes emisores de luz [14].
- **Art-Net DMX.** Es un protocolo de capa de aplicación que permite el transporte de datos DMX sobre redes de datos [10].
- ***Blending.*** Refiere al proceso de mezclado que realiza Sendero Server durante la generación del contenido visual de una obra.

2.3 Transferencia de video sobre redes de datos

Internet fue diseñado para soportar comunicaciones de datos tales como e-mails y archivos. A medida que fue creciendo en términos de nodos, aplicaciones y usuarios, emergió como una necesidad la inclusión de mecanismos para el intercambio de datos multimedia: imágenes, audio y video [27]. Actualmente millones de usuarios utilizan servicios de Internet para cubrir sus necesidades de telefonía y videoconferencia, sintonizar canales de televisión o radio, y hasta incluso compartir imágenes y videos en las múltiples redes sociales. Este explosivo crecimiento de las aplicaciones multimedia en Internet, causado por la creciente penetración del acceso residencial a la banda ancha y el acceso inalámbrico de alta velocidad [28], ha impulsado el desarrollo de una serie de protocolos capaces de satisfacer, empleando el servicio de "mejor esfuerzo" de Internet, los muy distintos requerimientos de red que presentan las aplicaciones multimedia. En particular, la **sensibilidad a los retardos terminal a terminal**, su **variación (*jitter*)**, y el hecho de que estas aplicaciones suelen ser tolerantes a **pérdidas** [27]**.**

Sendero no transmite datos multimedia como ser audio o video, sin embargo, sus requerimientos de red son muy similares a los que presentan las aplicaciones de este tipo, par-

ticularmente a las denominadas flujos multimedia en tiempo real. Es por esto que resulta de gran interés conocer el estado del arte en ésta área de estudio.

En el Anexo A se ofrece una categorización de las aplicaciones multimedia, junto con una serie de ejemplos. En lo que sigue, se presentan las limitaciones que el servicio de "mejor esfuerzo" de Internet posee para la transmisión de datos multimedia, un conjunto de técnicas de utilidad para superar estas limitaciones y finalmente una exposición de los protocolos desarrollados para contemplar los distintos escenarios donde las aplicaciones multimedia se presentan.

2.3.1 Limitaciones del servicio de "mejor esfuerzo" de Internet

El servicio de "mejor esfuerzo" que ofrece la capa IP de Internet, como sugiere su nombre, hace todo lo posible por transferir cada datagrama desde el emisor hacia el receptor de la manera más rápida posible. El modelo propuesto mantiene a los nodos intermedios tan simples como se puede, logrando una arquitectura de red muy escalable, lo que ha sido crucial para el gran éxito de Internet [27].

Sin embargo, esta solución no ofrece ninguna garantía respecto a:

- Retardo terminal a terminal. Suma de los retardos de transmisión, procesamiento y puesta en cola en *routers*; propagación en los enlaces, y procesamiento en los sistemas terminales.
- Variabilidad entre retardos (*jitter*). Retardos aleatorios generados por la puesta en cola dentro de los *routers*. Hace que el retardo terminal a terminal sea diferente entre un paquete y otro del mismo flujo de datos.
- Pérdidas de paquetes. Las colas de los *routers* que atraviesa un paquete pueden no tener espacio para almacenarlo hasta que es procesado, esto lleva a que sea descartado, provocando que el paquete no llegue a destino o deba ser retransmitido.

Dado que las aplicaciones multimedia imponen una serie de exigencias respecto a los aspectos mencionados, la utilización del servicio de "mejor esfuerzo" de Internet implica un gran desafío en el diseño de las aplicaciones, que deben tomar partido para superar estas dificultades [28]. Esto ha motivado el surgimiento de múltiples técnicas para la optimización del uso de la red bajo las restricciones mencionadas.

2.3.2 Técnicas de optimización del uso de la red

Si se ignoran las dificultades presentadas en la sección anterior, probablemente se obtenga un resultado poco satisfactorio de cualquier flujo de datos multimedia. Afortunadamente, existen ciertas técnicas que pueden ayudar a resolver estos problemas, como ser: UDP como protocolo de transporte, números de secuencia, marcas de tiempo, retardo de reproducción, corrección de errores y compresión.

2.3.2.1 UDP como protocolo de capa de transporte

El protocolo de capa de transporte más comúnmente utilizado en Internet es TCP (*Transport Control Protocol [29]*). Este añade un gran número de funcionalidades al servicio de entrega no fiable de IP, realizando una abstracción para brindar entrega fiable y secuencial de un flujo de datos entre dos puertos. Para ello, TCP implementa mecanismos de fragmentación, adaptación de la tasa de envío y retransmisiones, que evidentemente contrasta con las características de las aplicaciones multimedia: **sensibles a los retardos** y **tolerantes a las pérdidas**. Por otro lado, se encuentra UDP (*User Datagram Protocol [30]*), este protocolo, apenas extiende las funcionalidades de IP, añadiendo información sobre los puertos y un campo para validar el estado de la información transportada. UDP no realiza ningún control sobre la congestión de la red, ni regula la tasa de transmisión en base esto. Tampoco tiene mecanismos para implementar retransmisiones, haciendo que los paquetes puedan llegar tarde, desordenados o incluso perderse. Sin embargo, el compromiso entre fiabilidad y latencia que se plantea al elegir entre TCP y UDP hace a éste último un protocolo más adecuado para transportar información multimedia en tiempo real [31].

2.3.2.2 Números de secuencia

Un número de secuencia consiste en un valor, típicamente entero, que es incrementado por el emisor en una unidad por cada paquete generado. Su objetivo principal es identificar a un paquete dentro de un flujo, permitiendo determinar si se ha perdido o fue recibido fuera de orden.

2.3.2.3 Marcas de tiempo

Una marca de tiempo consiste en el registro de la hora en la cual el paquete fue generado en el emisor. Su objetivo es ser utilizada para planificar la reproducción en el receptor.

2.3.2.4 Retardo de reproducción

Todo paquete debe haber sido recibido antes de poder reproducirse. Debido a la incertidumbre sobre el retardo terminal a terminal y la variación del mismo, esto no puede garantizarse, dando lugar al **retardo de reproducción**. El cometido de retardar la reproducción, o lo que es lo mismo, mantener un *buffer* de reproducción, es intentar asegurar que la mayor cantidad de paquetes sean recibidos antes de su tiempo de reproducción, añadiendo para ello un retardo que puede ser fijo o adaptativo.

Retardo de reproducción fijo

Esta estrategia propone las postergación de la reproducción de cada paquete recibido en un tiempo fijo dado. En caso de que este paquete no haya sido recibido antes de su reproducción, será descartado. Esto implica un compromiso entre la interactividad y la tasa de pérdida de paquetes. Si el valor de retardo elegido es muy pequeño, los paquetes se intentarán reproducir casi inmediatamente después del momento en que fueron genera-

dos, lo que permite gran interactividad. Sin embargo, si las condiciones de la red generan retardos terminal a terminal mayores a este tiempo, o la variabilidad entre estos retardos es muy grande, entonces muchos paquetes no llegarán a tiempo para ser reproducidos. En el caso que el valor de retardo de reproducción sea grande, es probable que la mayoría de los paquetes lleguen a destino antes de su tiempo de reproducción, pero se pierde en interactividad.

Esta estrategia de planificación de la reproducción tiene como ventaja que es muy simple, siendo adecuada para entornos donde las fluctuaciones de la red son predecibles y estables. En situaciones de gran variabilidad, como es normal en Internet (y en la comunicación con los clientes remotos de Sendero), esta estrategia puede ser mejorada estudiando y estimando el comportamiento de la red para aplicar una política de retardo adaptativo [28].

Retardo de reproducción adaptativo

Como se mencionó en el punto anterior, un retardo muy grande atenta contra la interactividad pero disminuye la tasa de pérdidas de paquetes, mientras que un retardo más pequeño causa el efecto contrario. Una solución a esta problemática es entonces estimar el retardo de la red y su variabilidad para ajustar el retardo de reproducción en base al comportamiento de la red.

En el Capítulo 4 se describe el mecanismo utilizado para resolver la reproducción en los clientes remotos de Sendero, solución fuertemente inspirada en las explicaciones brindadas por Perkins en su libro "*RTP: Audio and Video for the Internet*" [31] sobre los mecanismos utilizados por el protocolo *RTP* y las exposiciones de M. Xing, X. Min, X. Siyuan, y C. Lin [32], A. A. Razzac, S. E. Elayoubi, C. Tijani, y E. Bachar [33] y C. J. Sreenan, C. Jyh-Cheng, P. Agrawal, y B. Narendran [34].

2.3.2.5 Recuperación frente a pérdidas

En el contexto de una red donde existe la posibilidad de que los paquetes se pierdan, ya sea porque fueron descartados en el *buffer* de alguno de los *routers* de su trayectoria, o porque fueron entregados luego del tiempo al que fue planificada su reproducción, la utilización de mecanismos de recuperación frente a pérdidas puede contribuir a mejorar la calidad percibida de la reproducción, disimulando la falta de los paquetes perdidos.
Existen múltiples técnicas para llevar a cabo esta tarea y las mismas pueden clasificarse según donde sean aplicadas, emisor o receptor. Una excelente exposición sobre este tema es *"A Survey of Packet Loss Recovery Techniques for Streaming Audio"* de C. Perkins, P. Colin, H. Orion, y H. Vicky [35].

2.3.2.6 Compresión

El contenido multimedia, ya sea audio o video requiere de importantes cantidades de espacio de almacenamiento. Dado que este factor afecta directamente en el ancho de ban-

da, y éste a su vez en la performance de la red, la utilización de técnicas de compresión de datos es un factor crucial en el uso óptimo de la red.

La compresión de contenido multimedia está basada en distintos algoritmos que pueden ser implementados tanto en *software* como en *hardware*. Existen cientos de estos algoritmos, y una muy extensa bibliografía sobre el tema [36].

Las técnicas de compresión se pueden clasificar en dos categorías: **Sin pérdidas** y **con pérdidas**. Los primeros, permiten la recuperación exacta del contenido comprimido, mientras que los segundos retornan una aproximación. Al mismo tiempo, las técnicas de compresión con pérdidas permiten mayores factores de compresión, siendo ampliamente utilizados con imágenes y video [37], [38]. Estas técnicas, explotan una característica del contenido multimedia que es la redundancia, espacial y temporal. La redundancia espacial refiere a la relación entre los valores espacialmente cercanos dentro de por ejemplo una imagen. La redundancia temporal refleja la repetición parcial de contenido entre un *frame* y el siguiente, por ejemplo en un video. Distintas técnicas utilizan esto a su favor para reducir el tamaño del contenido y así hacer un uso más eficiente de la red. Por ejemplo JPEG [39] para imágenes, MPEG [40] para videos.

Las técnicas introducidas en esta sección, junto a otras, han servido como base para el desarrollo de múltiples protocolos y estándares que son relevados a continuación.

2.3.3 Protocolos de red para aplicaciones multimedia

La capacidad de intercambiar información multimedia de manera exitosa sobre redes de datos, se basa en la utilización de múltiples protocolos y estándares que trabajan de manera conjunta para satisfacer los requerimientos de funcionalidad y performance que los distintos tipos de aplicaciones exigen.

Desde el establecimiento, control y descripción de la sesión, hasta el transporte del contenido multimedia, estos protocolos agregan valor al modelo básico de entrega de paquetes brindado por IP, haciendo posible la utilización de las aplicaciones multimedia sobre el servicio de "mejor esfuerzo" de Internet.

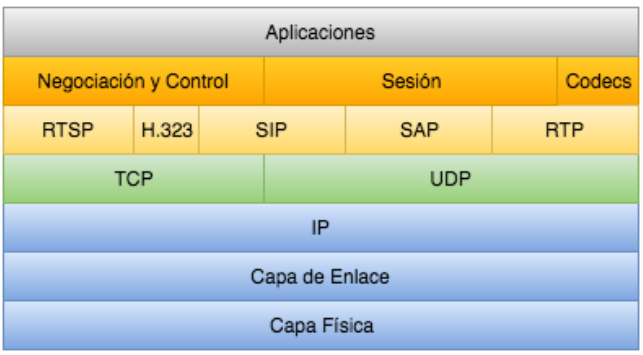


Figura 2-1 – Pila de protocolos multimedia.

La Figura 2-1 es una adaptación de la presentada en "*RTP: Audio and Video for the Internet*" de C. Perkins. La misma muestra una pila de protocolos que trabajan en conjunto con el fin crear, administrar y describir una sesión de datos multimedia. En lo que sigue se presentan los protocolos más destacados de cada categoría: establecimiento y control de la sesión, descripción de la sesión y transporte de contenido.

2.3.3.1 Protocolos para el establecimiento y control de la sesión

Muchas aplicaciones requieren de la creación y manejo de una sesión o de la utilización de mecanismos de anuncio para dar comienzo a una sesión de transmisión. Múltiples protocolos y estándares han sido diseñados para satisfacer este objetivo, y su utilización depende del escenario de la aplicación.

Con el objetivo de inicializar una sesión interactiva, como puede ser una llamada o video conferencia, existen dos estándares de facto [31]. La recomendación UIT H.323 [41], y el protocolo SIP (por sus siglas en inglés de *Session Initiation Protocol*) definido por IETF en el RFC 3261 [42].

Si la finalidad de la aplicación es iniciar una sesión no interactiva, como puede ser un flujo de audio/video almacenado entonces el principal estándar es el *Real-Time Streaming Protocol* (RTSP) definido en el RFC 2326 [43]. Este es un protocolo de control no orientado a conexión cuyo objetivo principal es establecer y manejar flujos multimedia de reproducción sincronizada. Su utilización se encuentra ampliamente extendida y es conocido como "el control remoto" de los servidores multimedia [28], debido a sus funcionalidades *PLAY, STOP, PAUSE, FORWARD* y *BACKWARD.*

Por último, para anunciar a un grupo sobre el comienzo de una sesión de *streaming*, como puede ser un flujo en vivo (IPTV o radio por Internet por ejemplo) el estándar es el protocolo SAP, por sus siglas en inglés de *Session Announcement Protocol*. Definido en el RFC 2974 [44] es un protocolo de aviso usado para asistir a la publicación de anuncios de sesiones *multicast*, y para comunicar información importante de configuración de la sesión a los posibles participantes.

Los requisitos para estos protocolos son bastante distintos, en términos del número de participantes en la sesión y el nivel acoplamiento entre los mismos. Por otro lado, algunos tipos de sesión apenas aplican restricciones sobre sus participantes, mientras que otros implican controlar y administrar explícitamente permisos sobre las acciones que cada uno puede ejecutar. Estos tan distintos requerimientos han desembocado en el desarrollo de muy distintos protocolos, y es una responsabilidad del diseñador de una aplicación multimedia encontrar el protocolo más adecuado para su aplicación.

En el Anexo B se presenta una descripción más detallada de cada uno de los protocolos mencionados.

2.3.3.2 Descripción de sesiones y contenido

Común a todo protocolo de inicialización de sesión es la necesidad de poder describir de alguna manera los detalles a acordar, como medios, direcciones de transporte y otros metadatos de las sesiones y los participantes. A causa de esto, varios protocolos fueron desarrollados para satisfacer esta necesidad, uno de los más destacados es el *Session Description Protocol* (SDP), actualmente redactado en el RFC 4566 [45]. Su objetivo es proporcionar una representación estándar de la información necesaria para permitir a los participantes conocer sobre la existencia de una sesión, y al mismo tiempo brindar la información que estos necesitan para unirse a la misma.

Una descripción de sesión SDP incluye: nombre y propósito de la sesión, tiempo en que la sesión permanecerá activa, información sobre el tipo de contenido involucrado en la sesión e información para obtener el contenido (direcciones de red, puertos, etc.).

2.3.3.3 Protocolos de transporte de contenido multimedia

En el corazón de la pila de protocolos multimedia, se encuentran los protocolos de transporte. Su objetivo es implementar los mecanismos que permiten la utilización del servicio de "mejor esfuerzo" de Internet para transportar datos en tiempo real de manera robusta.

Dentro de estos, se destaca el protocolo RTP [31], [46], por sus siglas en inglés de *Real-Time Transport Protocol*, cuya utilización se encuentra ampliamente extendida. Este protocolo provee funcionalidades de transporte de datos multimedia en tiempo real, tales como audio o video. Dichas funcionalidades incluyen identificación de tipos de contenido, números de secuencia, marcas de tiempo y monitoreo de la entrega. RTP es utilizado típicamente sobre UDP, con el fin de aprovechar sus servicios de multiplexación y validación de correctitud, aunque su implementación no está limitada al uso de este protocolo de capa de transporte. Por otra parte, no provee en sí mismo ningún mecanismo para garantizar la entrega a tiempo de los datos ni ninguna otra garantía de calidad de servicio (QoS, *Quality-of-Service*); ni siquiera garantiza la entrega de paquetes o evita su entrega desordenada. Sin embargo, provee detección de pérdidas y reportes sobre la calidad del flujo, información sobre temporización y sincronización, tipo de carga útil e identificación de la fuente [31].

En el Anexo C se ofrecen más detallas sobre este protocolo y su compañero RTCP.

2.3.3.4 Tecnologías para *streaming* web

Hasta aquí se presentaron una serie de estándares y protocolos cuya utilización conjunta permite establecer, administrar y transmitir flujos de datos sobre la red de Internet. Actualmente la mayor parte de las aplicaciones de Internet ofrecen acceso a través de la web. Esto implica que existe una gran cantidad de contenido multimedia que es consumido a través de navegadores. Ya sea contenido almacenado, en vivo o interactivo en

tiempo real (ver Anexo A), las aplicaciones multimedia en la web conllevan importantes desafíos tecnológicos, para adaptar la arquitectura tradicional de pedido/respuesta a un esquema válido de transmisión de información en tiempo real.

En el esquema tradicional planteado por HTTP [47], un cliente establece conexiones con un servidor con el propósito de solicitar información. Los servidores aceptan conexiones desde los clientes con el fin de recibir y responder a sus pedidos. En este modelo, un servidor no puede iniciar una conexión con un cliente ni enviar información que no haya sido solicitada por este. Por lo tanto, para obtener información en tiempo real un cliente debe consultar al servidor periódicamente (*polling*). Sin embargo, la consulta continua hacia un servidor puede aumentar significativamente el ancho de banda, forzando una secuencia pedido/respuesta cuando no hay nuevos datos disponibles. Además, el servidor no puede notificar al cliente cuando obtiene nuevos datos, disminuyendo la interactividad de la comunicación.

Con el objetivo de resolver estas problemáticas, en los últimos años han sido (y siguen siendo) desarrollados diversos mecanismos, protocolos y técnicas. Como parte de la investigación llevada a cabo se relevaron las tecnologías HTTP *Live Streaming* [48], [49], HTTP Long-polling [50], HTTP *Streaming* [50], WebSockets [51] y WebRTC [52], [53]. Una descripción de cada uno de estas puede ser encontrada en el Anexo D.

2.4 Transferencia de datos en redes *Machine-To-Machine*

Además de compartir muchas similitudes con las aplicaciones multimedia, Sendero, involucra la interacción de múltiples componentes de *hardware* y *software* que se comunican para cumplir con el objetivo de reproducir patrones de iluminación sobre los píxeles de una instalación. De este modo, el estudio de arquitecturas, protocolos y tecnologías en el área de la comunicación *Machine-To-Machine* resulta de especial interés, al fin de poder concretar el objetivo de migrar el esquema de comunicación de *hardware* hacía uno inalámbrico.

2.4.1 Comunicación *Machine-To-Machine* (M2M)

El término *Machine-To-Machine*, identificado por el acrónimo M2M, hace referencia a un conjunto de tecnologías que hacen posible el intercambio bidireccional de información entre máquinas remotas, sin necesidad de que exista intervención humana, y que utilizan para ello redes de comunicación cableadas o inalámbricas.

Dada la amplitud del término y sus múltiples aplicaciones, se conocen muchas acepciones del acrónimo M2M: *Machine-To-Mobile*, *Mobile-To-Machine* y *Machine-To-Man* son algunas de éstas [54]. En lo que respecta a este documento, se utilizará M2M para hacer referencia a *Machine-To-Machine*, como una abreviación de *Machine-To-Machine communications.*

La definición dada puede parecer poco exhaustiva o demasiado amplia, y lo es. En las secciones que continúan se procura brindar un poco de claridad sobre lo que es M2M, ofreciendo una caracterización de las redes y los dispositivos participantes, sus múltiples campos de aplicación, una arquitecturas de red estándar y las tecnologías que actualmente permiten su implementación.

2.4.1.1 Características de M2M

Las comunicaciones *Machine-To-Machine* presentan un conjunto de características distintivas en lo que refiere a los dispositivos involucrados, lo cual a su vez genera un gran impacto en los requerimientos de red y aplicaciones.

- **Multitud.** La característica más importante es el número de dispositivos. Se sostiene que el mismo se incrementará sustancialmente en los próximos años, alcanzando los 50.000.000.000 de dispositivos conectados para el año 2020 [55], lo que supone un problema para las redes de datos actuales en lo que refiere la cantidad de tráfico que podría generarse.
- **Variedad.** Existe una gran cantidad de posibles aplicaciones y por lo tanto, una gran cantidad de dispositivos diferentes que dan soporte a sus necesidades. Esto implica que existirá también una amplia variedad de requerimientos distintos, en términos de tasas de transferencia de información, capacidades de cómputo y alimentación. Lo que deriva uno de los mayores obstáculos de M2M, la heterogeneidad en sus componentes, que requiere de un alto grado de interoperabilidad y estandarización en las redes y sus aplicaciones.
- **Invisibilidad.** Muchas aplicaciones de M2M, tienen como requerimiento que sus dispositivos trabajen de manera invisible, esto es, con muy poca o ninguna intervención humana. Esto pone a la administración de los dispositivos en un rol clave, debiendo ser una parte perfectamente integrada de cualquier solución.
- **Criticidad.** Muchos dispositivos se utilizan para tareas críticas. Su utilización impone estrictos requerimientos de latencia y confiabilidad que pueden desafiar o incluso exceder las capacidades de las redes actuales.
- **Privacidad.** La importancia de la privacidad en las comunicaciones no es un tema nuevo, sin embargo, su implementación puede presentar un obstáculo no menor en diversos campos de aplicación de M2M.

2.4.1.2 Características de los dispositivos M2M

La forma en la que los dispositivos involucrados en las comunicaciones M2M se comunican a través de las redes depende directamente de sus características y restricciones. A continuación se listan las más importantes [54].

- **Funcionalidad limitada.** La mayoría de los dispositivos M2M tienen capacidades de cómputo varios órdenes por debajo que las computadoras de escritorio, portátiles e incluso que los teléfonos inteligentes. Una de las principales razones que impulsa esto es el costo y la posibilidad de competir en el mercado. Por otro

lado, muchos dispositivos están diseñados para realizar tareas simples que no requieren demasiado poder computacional, como por ejemplo sensores cuyo fin es capturar una muestra y transmitir su valor.

- **Bajo consumo.** Si bien muchos dispositivos pueden estar conectados a una red eléctrica, es probable que otros muchos no, dependiendo de la aplicación. En el segundo caso, se requiere de fuentes de alimentación alternativas y mecanismos para minimizar la interacción con las aplicaciones, de modo de hacer un uso óptimo de la energía.
- **Embebidos.** Muchos dispositivos son, y serán, diseñados en sistemas que deben operar en condiciones que hacen difícil la realización de cambios sin afectar al sistema en sí mismo.
- **Vida útil.** Dependiendo del campo de aplicación donde un dispositivo M2M se desenvuelva, la expectativa sobre su vida útil puede variar en el orden de días hasta decenas de años.

2.4.1.3 Conceptos relacionados: IoT y WSN.

El concepto M2M puede ser muy amplio y genérico, por lo que suele ser confundido con otros términos que se encuentran estrechamente relacionados: *Wireless Sensors Networks* (WSN) e *Internet of Things* (IoT).

WSN hace referencia a una colección de sensores distribuidos en el espacio con el fin de monitorear condiciones físicas como pueden ser, temperatura, presión o ruido, y transmitir la información recolectada a través de una red inalámbrica a un servidor, encargado de almacenarla y procesarla. Las conexiones entre estos nodos son implementadas a través de servicios de radio de corta distancia, y los mismos no requieren ser identificados globalmente.

M2M refiere a las tecnologías que permiten la comunicación entre dispositivos similares, de manera cableada o inalámbrica. Estas tecnologías pueden proveer de mecanismos de identificación global a sus dispositivos y pueden ser utilizadas para interconectar redes WSN.

Finalmente, IoT puede ser descrito como la conexión de todos los dispositivos electrónicos conocidos en una estructura similar a Internet. Los dispositivos poseen una identificación única, como la que brinda el protocolo IP, que puede ser utilizada para establecer comunicaciones directas entre los dispositivos.

Por lo tanto, WSN es una forma de M2M y M2M es una forma de IoT [56].

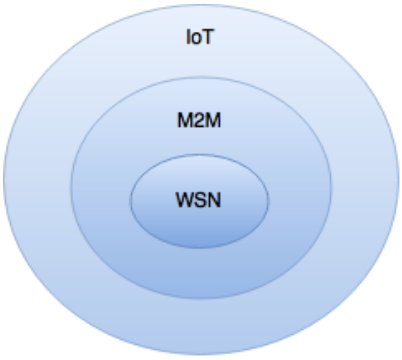


Figura 2-2– Relación entre WSN, M2M e IoT

2.4.2 Aplicaciones de M2M

Las posibles aplicaciones de las comunicaciones M2M son muy amplias, abarcando múltiples sectores como Transporte, Energía, Industria, Entretenimiento, Salud y Seguridad entre otros. *Beechman Research* [57] publicó una investigación en 2009, donde se detalla un conjunto de sectores, los campos de aplicación dentro de estos, y los lugares y dispositivos involucrados. Una versión interactiva de los resultados de esta investigación se encuentra disponible en el sitio web de esta organización [58].

2.4.3 Arquitectura y componentes

En un universo tan amplio de aplicaciones es inevitable que se presenten diversos tipos de requerimientos y dispositivos con distintas restricciones de costos, tamaños y capacidades. Esto impulsa la necesidad de definir protocolos y estándares que permitan el desarrollo de aplicaciones M2M de manera escalable y robusta. Tarea a la cual se han abocado en las últimas décadas múltiples organizaciones y empresas, entre ellas, ETSI, *European Telecommunications Standards Institute [59]* quien ha desarrollado una propuesta para una arquitectura genérica de alto nivel para sistemas M2M [60].

La Figura 2-3 muestra un esquema de esta arquitectura que se separa en dos grandes dominios. Uno para los dispositivos y *gateways,* y otro para la red.

- El **dominio de dispositivos y *gateways*** está compuesto por:
    - **Dispositivos M2M.** Dispositivos que ejecutan aplicaciones M2M. Se pueden distinguir en dos grupos: dispositivos creados específicamente para dar soporte a aplicaciones M2M, habitualmente sensores o actuadores que reportan información; o módulos de comunicaciones embebidos en otros dispositivos (ver Sección 2.4.5). Los dispositivos M2M se conectan con las aplicaciones a través del dominio de red, y a esté de manera directa, o empleando un *gateway* como proxy. A quien se conectan a través de una red de área local.

- **Red de área local *(M2M Area Network)***: provee conectividad entre los dispositivos M2M y *gateways*. Ejemplos: IEEE 802.15.1, ZigBee, Bluetooth, Wi-Fi, PLC, M-BUS. (Ver *Redes de área local (M2M area networks)*)
- ***Gateway* M2M**: Un *hardware/software* que actúa como *proxy* entre los dispositivos M2M y el dominio de red.

- El **dominio de red** se compone de:
  - **Red de Acceso.** Una red que permite a los dispositivos M2M y *Gateways* comunicarse con la Red Central (*Core Network*).
  - **Red Central.** Se encarga de proveer conectividad IP como mínimo, y potencialmente otros tipos, otorgar **f**unciones de control y servicios de red, e interconexión con otras redes.
  - **M2M Service Capabilities.** Responsable de proveer funcionalidades que pueden ser compartidas por diferentes aplicaciones M2M, exponer funciones a través de una serie de interfaces, utilizar las funcionalidades ofrecidas por la Red Central y optimizar el desarrollo de aplicaciones.
  - **Aplicaciones M2M**. Aplicaciones que implementan la lógica de los servicios y utilizan los *M2M Service Capabilities* accesibles a través de una interfaz abierta.

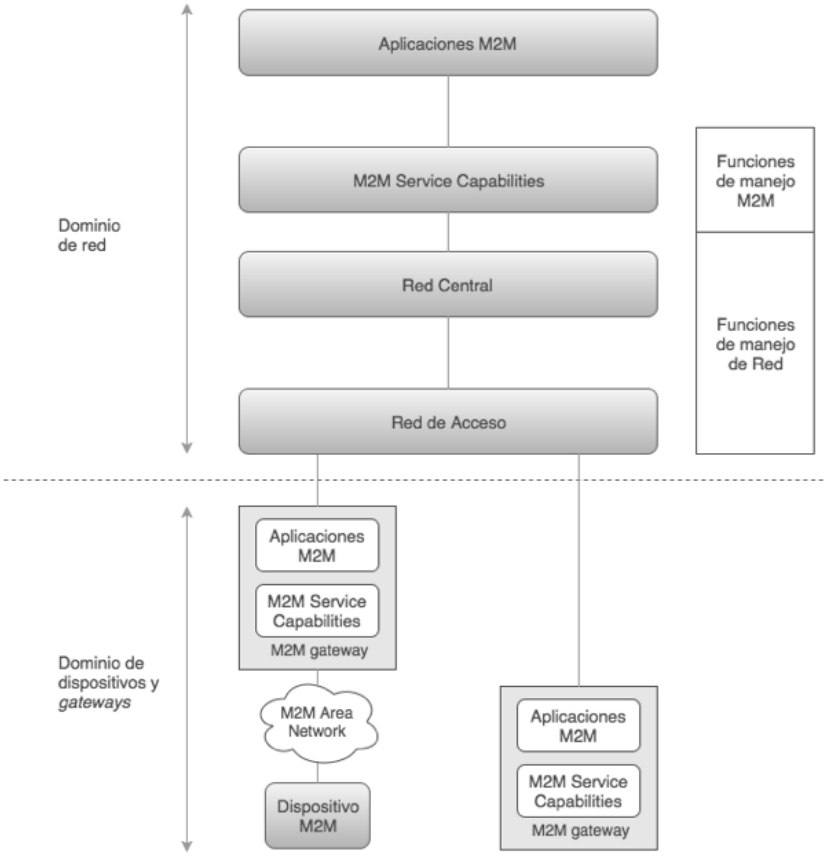


Figura 2-3 - Esquema de arquitectura ETSI TS 102 690.

2.4.4 Tecnologías y Estándares

Como fue mencionado en la sección anterior, las características de las comunicaciones M2M hacen que sea necesario el desarrollo de tecnologías, protocolos y estándares que permitan satisfacer las necesidades existentes y brindar marcos de trabajo para el desarrollo de esta área de manera escalable en el futuro próximo. La estandarización requerida alcanza distintos niveles, como ser modelos de datos, redes de área local, red de acceso, red central y certificación de módulos y terminales.

En particular para este proyecto son de gran interés el estudio de las tecnologías y estándares vinculados a los dispositivos M2M y las redes de área local, en especial aquellas que utilizan tecnologías inalámbricas. En lo que sigue se describen más en profundidad estos aspectos.

2.4.5 Dispositivos M2M: Terminales y módulos

Los dispositivos M2M pueden ser distinguidos principalmente en dos grupos, terminales y módulos [54]. Los terminales son dispositivos creados para dar soporte específico a una determinada aplicación M2M, habitualmente sensores o actuadores que recogen y reportan información. Los mismos poseen dos componentes lógicos bien diferenciados, el primero es el formado por el *software* y *hardware* específico de la aplicación M2M a la que pertenece y el segundo es un módulo M2M. Por su parte, los módulos M2M son componentes cuya principal responsabilidad es proveer de conectividad a otros dispositivos. Por lo que es particularmente importante el conjunto de interfaces de *hardware* que estos módulos ofrecen para interactuar con otros dispositivos.

2.4.5.1 Interfaces de *hardware*

Dado que el objetivo de los módulos M2M es acoplarse a otros dispositivos ya existentes, los proveedores de este tipo de tecnologías ofrecen una diversa gama de interfaces de *hardware* para permitir a sus módulos interactuar con otros dispositivos. En el Anexo E se listan la mayoría de los tipos de interfaces de *hardware* disponibles hasta el momento para módulos M2M [54]. Este aspecto puede cumplir un rol fundamental en la selección de un módulo para Sendero, debido a que es necesario conectarlo con ciertos componentes cuyas interfaces ya se encuentran establecidas.

2.4.5.2 Categorización de módulos

Además de la categorización que se desprende del conjunto de interfaces de *hardware* que cada módulo pueda tener, y dada la gran diversidad de aplicaciones que tienen las comunicaciones M2M, surge la necesidad de poder clasificar estos módulos de algún modo, y así facilitar el proceso de selección para determinado campo de aplicación.

En el libro "*M2M Communications: A systems approach*" de David Boswarthick et al. [54] se plantean métodos para clasificar los distintos tipos de módulos en grupos, no necesariamente excluyentes, según sus propiedades de conectividad y comunicación.

Se presentan distintas categorías, según tecnología de acceso (inalámbrica o cableada), en el caso de los módulos inalámbricos según su alcance y dentro de cada categoría según otros aspectos de las tecnologías.

Es de particular interés conocer la clasificación de módulos inalámbricos según su alcance (y también según su capacidad de procesamiento y ancho de banda). La Tabla 2.1 resume las principales categorías mencionadas en el libro de David Boswarthick et al.

Categoría	**Rango de alcance**	**Tecnologías de red (ejemplos)**
PAN	< 10m	Bluetooth, ZigBee, Infrarrojo.
WLAN	< 100m	Wi-Fi (802.11)
WWLAN	orden de kilómetros	GSM, LTE

Tabla 2.1- Tecnologías de acceso inalámbrico según rango de alcance.

2.4.6 Redes de área local (*M2M area networks*)

El rol de las redes de área local en la arquitectura presentada es el de comunicar los dispositivos M2M entre si y/o con los *gateways*. Estos a su vez comunican a los dispositivos con la red de comunicación, para alcanzar finalmente a la aplicación M2M que transformará los datos recibos en información de utilidad para sus usuarios.

En esta presentación se hará énfasis en las tecnologías de área local inalámbricas, por ser de especial interés para el desarrollo del proyecto. No obstante, no son la única opción disponible.

Los dispositivos M2M suelen ser diseñados con el propósito de soportar un único tipo de tecnología de comunicación de red [61], por lo que la selección de un módulo restringe el tipo de red de área local y viceversa. Existen múltiples investigaciones [62][63][64][65] que comparan y evalúan distintas tecnologías de red de área local. En el contexto de este proyecto, se seleccionaron para su evaluación las siguientes: Wi-Fi (IEEE 802.11) [66], Bluetooth (IEEE 802.15.1) [67], WiMAX (IEEE 802.16) y ZigBee (IEEE 802.15.4) [68]. En el Anexo F se presenta un resumen de las características más destacadas de cada una.

2.5 Resumen

A lo largo de este capítulo fueron relevados distintos protocolos, estándares y técnicas cuya aplicación o estudio pueden ser de gran utilidad a la hora de cumplir los objetivos encomendados por este proyecto.

En particular, en lo que refiere al flujo de datos de Sendero, los protocolos SAP (para el anuncio y configuración de la sesión), SDP (para la descripción de la sesión) y RTP (para el transporte del contenido) conforman una base teórica que de alguna manera se ajusta a las necesidades, siendo objeto de mayor estudio y aplicación. Por otro lado, la arquitectura del sistema y las tecnologías de *hardware* e infraestructura de red pueden articularse dentro de los esquemas presentados en el estudio del estado del arte en comunicaciones M2M presentado.

En el próximo capítulo se brinda una introducción a la versión de Sendero tomada como base, y se describen detalladamente los requerimientos que serán abordados. Luego, en el Capítulo 4, se brinda una contextualización de Sendero en el marco de los estudios realizados, para dar lugar a la solución propuesta, fuertemente inspirada en los resultados obtenidos de estudiar el estado del arte de estas disciplinas.

Capítulo 3

Análisis de Requerimientos

En este capítulo se brinda una descripción de la versión de Sendero tomada como base para el desarrollo de este proyecto, se presentan sus limitaciones y los requerimientos que se deben satisfacer, dando lugar a la descripción del alcance del proyecto en términos de los productos que deben ser entregados.

3.1 Descripción de Sendero

Sendero, es un sistema de iluminación integral, que abarca un conjunto de aplicaciones cliente-servidor para la generación de contenidos visuales, un dispositivo Art-Net DMX [10] basado en la plataforma de microcontroladores mBed [11][69] y los componentes de *hardware* necesarios para el control de LEDs RGB de alta potencia.

En las subsecciones siguientes se brindará un panorama más detallado de cada uno de estos componentes y su interacción.

3.1.1 *Hardware* de Sendero

Varios componentes de *hardware* deben establecer una comunicación para hacer efectivo el funcionamiento de Sendero. En la Figura 3-1 se muestra un esquema que introduce estos componentes y su interacción. Los mismos son descritos individualmente más adelante. Por una explicación detallada se sugiere consultar el blog de Pablo Gindel [70] y el trabajo de maestría de Christian Clark [6].

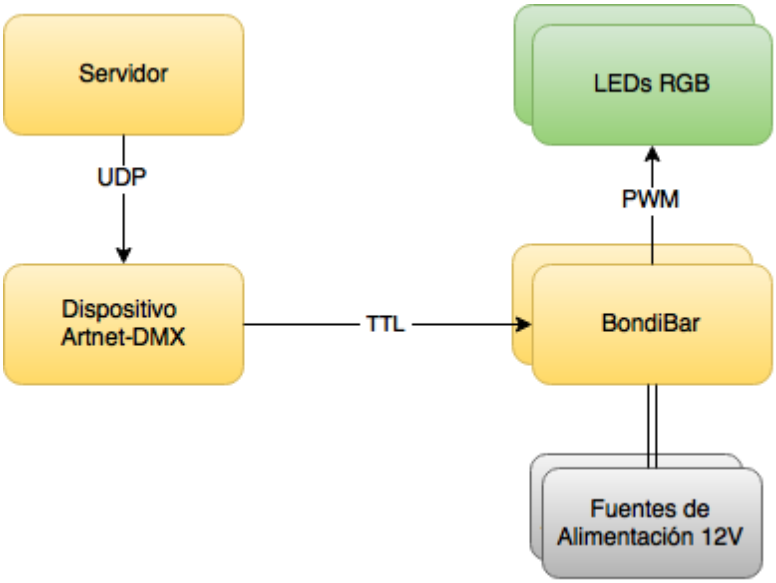


Figura 3-1 - Esquema de componentes de *hardware* de Sendero.

3.1.1.1 Servidor

Típicamente una computadora (o varias) que ejecuta(n) el *software* de Sendero, esto es, Sendero Server y una o varias implementaciones de Sendero Client (ver secciones 3.1.2.1 y 3.1.2.2 respectivamente).

3.1.1.2 Dispositivo Art-Net DMX

Un microcontrolador mBed NXP LPC17680 [69] con un *software* que le permite emular un dispositivo DMX [14] implementando el protocolo Art-Net DMX [10]. Este dispositivo oficia de mediador entre Sendero Server y múltiples controladores Bondibar (ver Sección 3.1.1.3). Para ello, convierte los paquetes Art-Net DMX con el contenido visual a reproducir, en señales comprensibles por los LED *drivers* alojados en los controladores Bondibar.

La conexión entre este microcontrolador y los controladores Bondibar se da a través de un cable de datos especialmente adaptado que se acopla a un conector USB integrado en el Bondibar. En el Anexo G se brindan detalles sobre esta comunicación. Por otro lado, los datos recibidos desde Sendero Server llegan al microcontrolador a través de una interfaz Ethernet [71].

3.1.1.3 Bondibar

Bondibar es el manejador de LEDs de Sendero. Se trata de un circuito impreso que agrupa un conjunto de *LED drivers* WS2801 [72] capaces de convertir señales seriales de datos en pulsos PWM para el control de las tiras LED RGB. Cada Bondibar es capaz de manejar ocho de estos *LED drivers*, y replicar la información que recibe a otros Bondibar

permitiendo un esquema de conexión en *daisy chain* (esquema de cableado que permite una conexión serial de dispositivos que replican una señal).

En el Anexo G se describe en detalle este componente. Adicionalmente, en el blog de Pablo Gindel [15] y la tesis de maestría de Christian Clark [6] se ofrecen explicaciones detalladas de su construcción y uso.

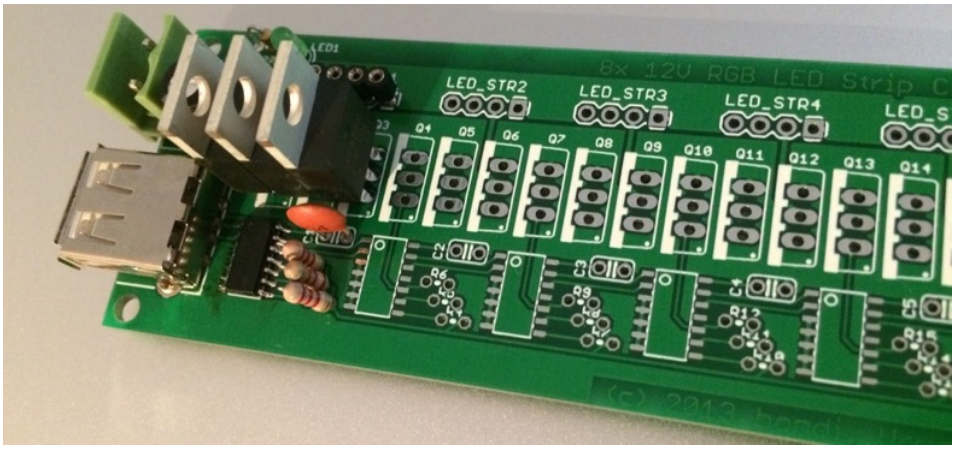


Figura 3-2 – Bondibar en proceso de construcción

3.1.2 *Software*

El *software* de Sendero comprende la interacción entre varias aplicaciones y servicios. La Figura 3-3 ilustra e introduce los componentes principales y cómo se vinculan, dando lugar a su descripción en las subsiguientes secciones. Una explicación detallada puede encontrarse en la tesis de maestría de Christian Clark [6].

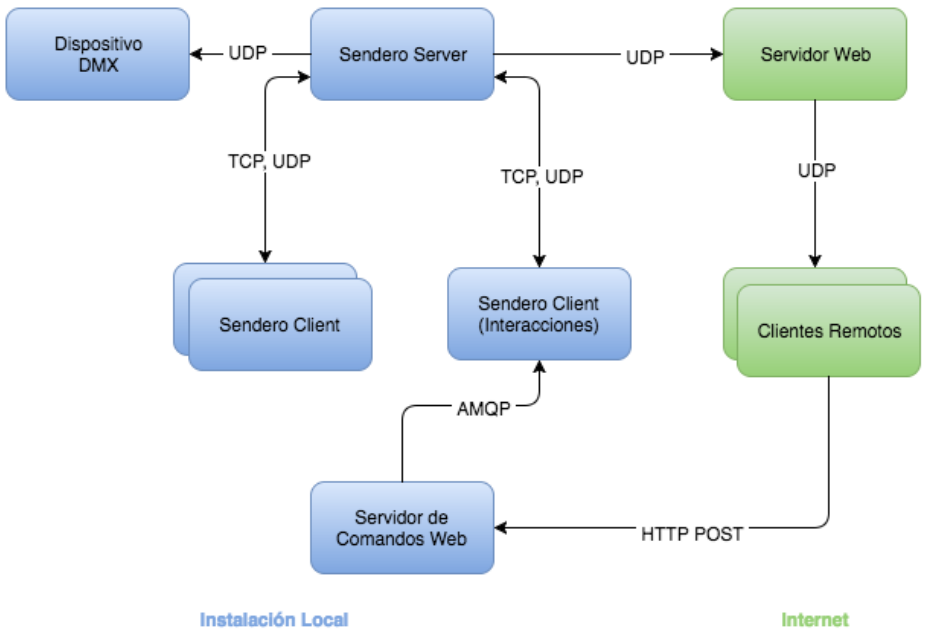


Figura 3-3 - Esquema de arquitectura de *software* de Sendero

3.1.2.1 Sendero Server

Sendero Server, es una aplicación C++ desarrollada bajo OpenFrameworks [12] cuyos objetivos principales son:

1. Proveer información sobre las características físicas de la instalación a las instancias de Sendero Client conectadas.
2. Obtener información de entrada desde las instancias de Sendero Client para la generación de contenidos visuales.
3. Realizar la mezcla (*blending*) de la información recibida desde las instancias de Sendero Client.
4. Comunicar el contenido generado a la obra (controladores Bondibar), a través del dispositivo Art-Net DMX.
5. Transmitir el contenido generado a un Servidor Web, para su distribución a los Clientes Remotos (ver Sección 3.1.2.3).

Adicionalmente, ofrece una interfaz de usuario que permite visualizar en tiempo real el contenido generado y configurar los parámetros del *blending*. La Figura 3-4
muestra una captura de pantalla de esta interfaz gráfica durante un prueba con la estructura de la obra Barcelona [24].

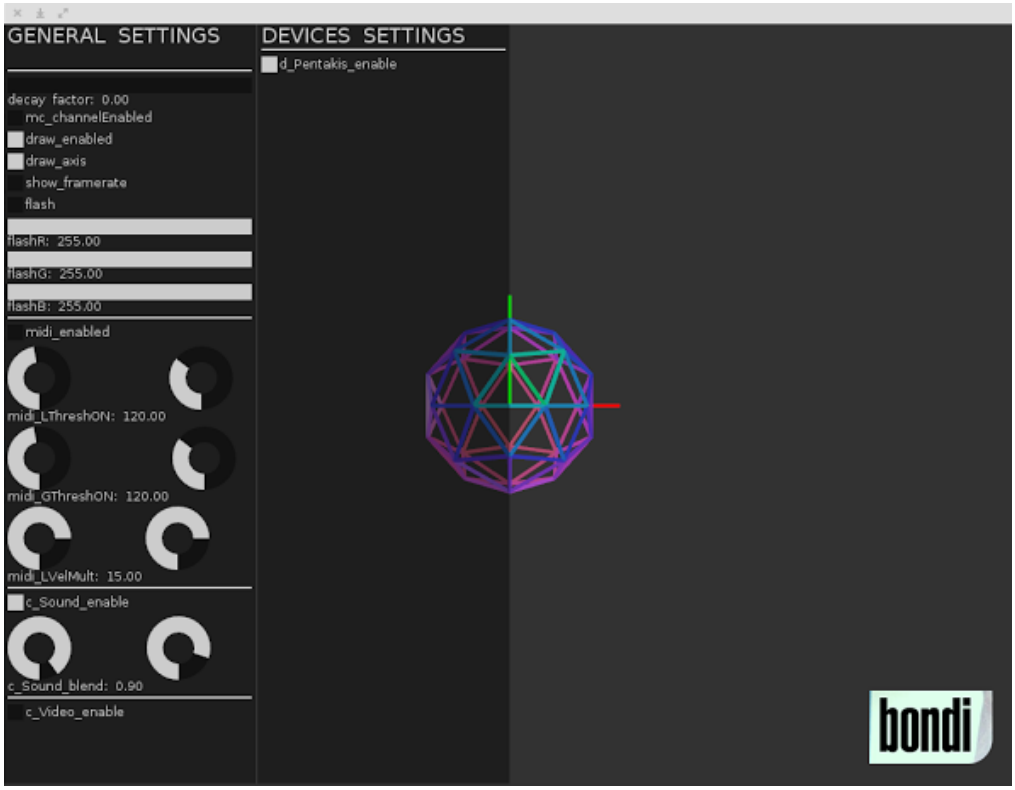


Figura 3-4 - Interfaz de usuario de Sendero Server.

3.1.2.2 Sendero Client

Sendero Client, son aplicaciones cuya finalidad es generar el contenido visual que sirve de entrada a Sendero Server.

Múltiples instancias de Sendero Client pueden conectarse simultáneamente a Sendero Server, y su comportamiento se encuentra fuertemente vinculado a la obra y a la forma en la cual los espectadores pueden interactuar con esta.

Cuando una instancia de Sendero Client se conecta a Sendero Server, lo hace a través de una conexión TCP dedicada, sobre la cual recibe la configuración de la obra y un número de puerto UDP que deberá utilizar para enviar la información de color que genere.

Distintas implementaciones de Sendero Client fueron desarrolladas para Barcelona [24][15] y Celebra [16][17], pero ninguna de estas es parte de la distribución *open-source* de Sendero. No obstante, se ofrece un *template* [8] que cubre todos los aspectos fundamentales de su integración.

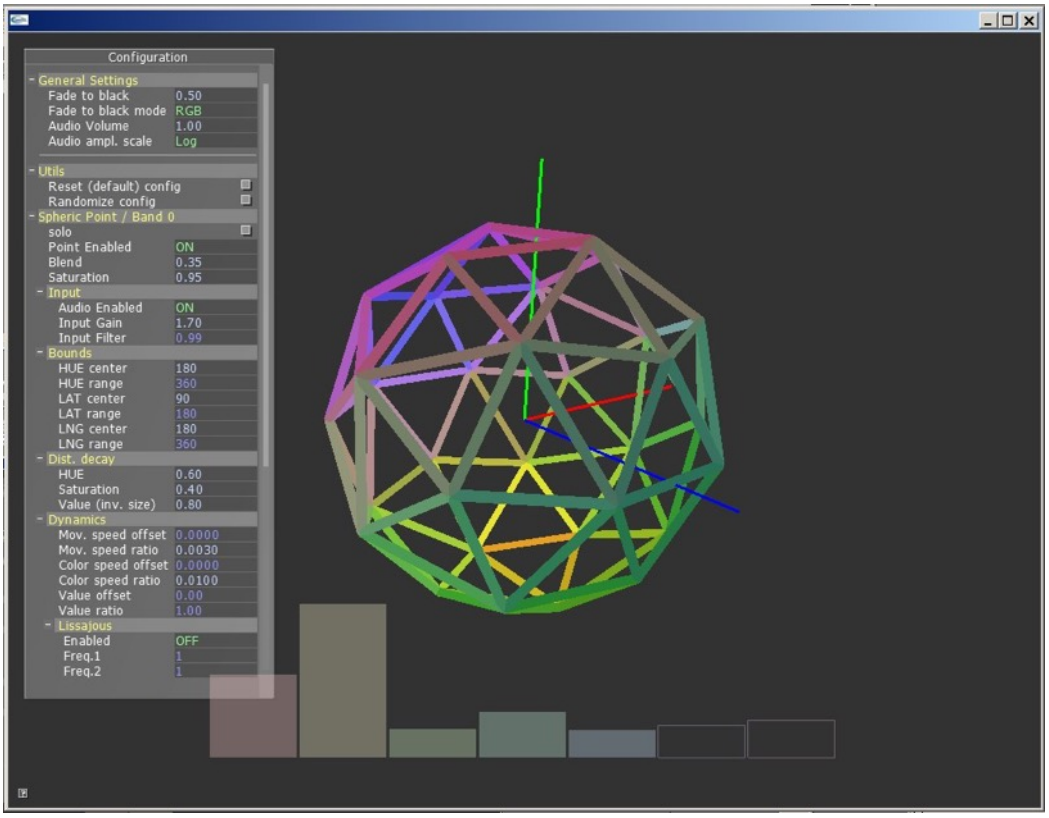


Figura 3-5 - Sendero Client desarrollado por Pablo Gindel para Barcelona.

3.1.2.3 Cliente Remoto, Servidor Web y Servidor de Comandos Web

Además de la posibilidades de interacción que se ofrecen a través de la implementación de instancias de Sendero Client, una serie de aplicaciones para *Smartphone* permiten que espectadores remotos reproduzcan y manipulen en tiempo real el estado de la obra.

Para ello, el contenido visual generado por Sendero Server es distribuido por Internet a través de un Servidor Web, y reproducido sobre una representación virtual en aplicacio-

nes iOS [73] y Android [74]. A esto se suma un Servidor de Comandos Web, que recibe señales desde los clientes remotos y las añade a una cola de mensajes.
Finalmente, una implementación de Sendero Client especializada, consume e interpreta los comandos de la cola de mensajes y los transforma en señales que sirven de entrada a Sendero Server, permitiendo de este modo que los clientes remotos contribuyan en la generación de contenido.

La comunicación entre Sendero Server y el Servidor Web se da utilizando el protocolo de capa de transporte UDP, al igual que la distribución del contenido hacia los Clientes Remotos. Por otro lado, los comandos enviados desde los Clientes Remotos hacia el Servidor de Comandos, son mensajes POST HTTP. Clark brinda detalles sobre la arquitectura y el funcionamiento de este sub-sistema en su trabajo "*Impressionism and New Media*" [6].

Ninguno de estos elementos fueron provistos como parte de la distribución base de Sendero, y tampoco forman parte de la versión *open-source*. Los desarrollos vinculados a este subsistema serán completamente nuevos.

3.2 Limitaciones

Distintas limitaciones fueron identificadas en este sistema y se detalladas a continuación:

- **Comunicación a nivel de *hardware*.** El esquema de comunicación de *hardware* propuesto presenta ciertas limitantes de conectividad. Tanto desde Sendero Server hacia el microcontrolador mBed (conexión punto a punto a través de una cable Ethernet), cómo desde este a los controladores de LED (conexión directa a través de cables de datos). Esto presenta varios problemas que dificultan la **escalabilidad** del sistema.
    - Dependencia del microcontrolador como componente central.
    - Susceptibilidad a fallas de cables y conectores.
    - Costo de utilización de cables.
    - Dificultad y tiempo de instalación.
    - Dificultad para el despliegue de grandes cantidades de píxeles.

- **Reproducción.** Si bien el contenido generado por Sendero no es explícitamente video, existen muchas similitudes en cuanto a la utilización y los requerimientos que este tipo de datos exige a la red. Actualmente se confía en los medios y mecanismos de transmisión, derivando en problemas en la reproducción que afectan su **robustez, performance y usabilidad**, en particular en los clientes remotos.

- **Interacción Remota**. La utilización de mensajes POST HTTP para el envío de comandos implica un importante *overhead* en la red, que disminuye la interactividad y la calidad percibida por el usuario, esto afecta de manera significativa la **usabilidad** de los clientes remotos.

3.3 Requerimientos

En base a estas problemáticas se plantean los siguientes requerimientos:

- **Migración del esquema de conectividad hacia uno inalámbrico**. La tecnología seleccionada debe permitir:
  - Conectar varios cientos (o incluso miles) de píxeles.
  - Operar en un rango de alcance mínimo de 50 m.
  - Transmitir (y recibir) datos a una tasa mínima de 24 *frames* por segundo.

  Al mismo tiempo se requiere la modificación y/o desarrollo de los componentes de *hardware* que deban transmitir datos de manera inalámbrica.

- **Aplicación de técnicas de *streaming* de video.** Estas deben otorgar robustez a la reproducción, mejorando la calidad percibida, tanto para el control de los píxeles de la obra, como para los clientes remotos.
  Su aplicación deberá tener dos grandes objetivos:
  - la red local inalámbrica que controlará la obra.
  - la red (Internet) por la que se conectarán los clientes remotos.

  Esto implica la utilización o desarrollo de protocolos de red que faciliten la distribución de contenido multimedia sobre redes de datos.

- **Desarrollo de una interfaz de usuario que oficie de cliente remoto.** La misma debe cumplir como mínimo con los siguientes puntos:
  - Permitir la visualización de una reconstrucción 3D de la obra.
  - Permitir la reproducción del contenido visual de la obra con la menor latencia posible.
  - Permitir la interacción en tiempo real con la obra utilizando la tecnología de WebSockets.
  - Ofrecer alta compatibilidad con dispositivos móviles.

- **Desarrollo y despliegue de algún mecanismo para la transmisión remota de la obra.** El mismo debe ser capaz de distribuir con la menor latencia posible los *frames* de la obra, a una tasa de transmisión mínima de 24 paquetes por segundo, y a una cantidad potencialmente grande de usuarios remotos simultáneos.

- **Desarrollo y despliegue de algún mecanismo que permita la interacción de los clientes remotos.** Del mismo modo que la versión base de Sendero ofrecía un Servidor Web (basado en *Python/Django* [75]) con el objetivo de recibir comandos en representación de las interacciones de los usuarios remotos, se requiere del desarrollo de algún mecanismo capaz de recibir las interacciones remotas empleando la tecnología de WebSockets.

3.4 Alcance del proyecto

En base a los objetivos generales presentados en la Sección 1.4, las limitaciones identificadas en la Sección 3.2 y los requerimientos detallados anteriormente, el alcance del proyecto queda constituido por los siguientes puntos:

Investigación	Estado del arte en transferencia de video sobre redes de datos.
	Estado del arte en comunicación M2M
Hardware	Modificación y/o construcción de los dispositivos de *hardware* que deban transmitir datos de manera inalámbrica.
Software	Aplicación de técnicas de *streaming* de video para el flujo de control de los píxeles de la obra.
	Aplicación de técnicas de *streaming* de video para el flujo que reciben los clientes remotos.
	Modificación del *software* de Sendero que requiera adaptación.
	Desarrollo y despliegue de un mecanismo para la transmisión de los *frames* de obra a clientes remotos.
	Desarrollo y despliegue de mecanismos para la recepción de las interacciones remotas empleando WebSockets.
	Implementación de interfaz que oficie como cliente remoto y haga uso de la tecnología de WebSockets.
Presentación	Validar la capacidad del sistema para la construcción de la obra Barcelona. (*)
Distribución	Posicionamiento de Sendero como una herramienta *open-source* y *open-hardware* provista por el Laboratorio de Medios de la Facultad de Ingeniería de la Universidad de la República.

Tabla 3.1 - Descripción agrupada del alcance del proyecto.

(*) Antes de que este proyecto comenzara, los desarrolladores iniciales del sistema (Christian Clark, Tomás Laurenzo, Pablo Gindel y Germán Hoffman) presentaron una propuesta para que la escultura interactiva Barcelona, desarrollada con Sendero en 2013 [24] sea exhibida en el Espacio de Arte Contemporáneo [76]. Dicha propuesta salió favorecida recibiendo un presupuesto para su construcción, lo que motivó el planteamiento por parte del tutor de este proyecto y creador de Sendero, de utilizar Barcelona para cumplir el objetivo de desarrollar una obra que utilice al sistema. La presentación será el

Jueves 1 de Diciembre, en el Espacio de Arte Contemporáneo, ubicado en Arenal Grande 1930, Montevideo, Uruguay. Mientras tanto, se determinó quitar del alcance del proyecto el objetivo de construir una obra de arte con nuevos medios, y en su lugar, validar que el sistema desarrollado tienen la capacidad de funcionar apropiadamente bajo las condiciones de Barcelona, lo que será presentado a través de una serie de pruebas experimentales (ver Capítulo 5 y Anexo M).

Capítulo 4

Diseño e Implementación

Este capítulo presenta la solución propuesta en términos de diseño e implementación. En primer lugar se brinda una contextualización de Sendero en el marco de los estudios realizados en el Capítulo 2. Luego se presentan las decisiones de diseño tomadas, y finalmente se detalla la implementación de la solución, junto a una descripción de cada uno de sus componentes.

4.1 Introducción

La solución propuesta se compone de un conjunto de componentes de *hardware* y *software* que operan de manera conjunta para cumplir con los objetivos del sistema. Antes de presentar las decisiones de diseño tomadas, es preciso comprender y contextualizar a Sendero en el marco de los estudios realizados sobre transferencia de video en redes de datos y comunicación M2M.

4.1.1 Sendero como una aplicación multimedia

En el Capítulo 2 se presentaron y clasificaron las aplicaciones multimedia. Las mismas se diferencian de otras aplicaciones por el tipo de datos que intercambian, pero particularmente, por las exigencias que deben cumplir las redes que los transportan.

Si bien Sendero en ningún momento transfiere información multimedia convencional (como ser audio o video), las características del contenido transmitido lo hacen **sensible a los retardos** y su **variación**, y **tolerante a pérdidas**. Esto lo asemeja mucho a una aplicación multimedia, en particular, a aquellas que fueron categorizadas como flujos interactivos en tiempo real.

Sendero Server (emisor) mezcla y genera un cierto contenido visual, que es enviado a una tasa constante (de por ejemplo 24 o 30 *frames* por segundo) en una red de datos, tal y como lo haría cualquier aplicación de video en tiempo real. Al mismo tiempo, los controladores de LED y los clientes remotos (receptores), esperan por estos *frames* para su

reproducción. Ambos escenarios se comportan como *flujos interactivos en tiempo real,* y en este contexto, es viable y pertinente la aplicación de las técnicas estudiadas en el Capítulo 2, sobre transferencia de video en redes de datos.

4.1.2 Sendero como una aplicación M2M

Al mismo tiempo, Sendero puede ser pensado como una aplicación M2M. En el marco de la exposición brindada en la Sección 2.4.3, se pueden asignar los roles de la arquitectura genérica propuesta por ETSI de la siguiente manera:

- **Aplicación M2M.** Sendero Server.
- **Red de área local.** La red en la cual operan los controladores de LED.
- **Dispositivo M2M.** Una serie de dispositivos a ser introducidos, constituidos por controladores de LED y módulos que le añaden conectividad.
- **Gateway M2M.** Dispositivo que interconecta a Sendero Server con la red de área local, y en efecto con los dispositivos.
- **Red de comunicación.** La red empleada para comunicar a Sendero Server con el Gateway M2M, y en definitiva, con los Dispositivos M2M.

Bajo estos términos, se identificaron las siguientes necesidades: seleccionar alguna tecnología para implementar la red de área local, seleccionar un módulo capaz de conectarse a esta red y comunicarse con un controlador de LEDs, seleccionar un *gateway* apropiado a la tecnología elegida y elegir algún mecanismo para implementar la red de comunicación. Decisiones que serán detalladas a continuación.

4.2 Decisiones de Diseño

En esta sección, se describen las decisiones de diseño tomadas para la implementación de los requerimientos del proyecto.

4.2.1 Tecnología de red inalámbrica

Uno de los requerimientos más importantes y desafiantes de este proyecto es la migración del esquema de comunicación de *hardware* hacia uno inalámbrico. La primera decisión que se tomó respecto a este tema fue la eliminación del microcontrolador mBed como componente central de la solución de *hardware*. Luego, se procedió con la selección de una tecnología de red inalámbrica apropiada, para lo cual se consideraron los siguientes aspectos:

- Topologías de red posibles (posibilidad de transmitir de uno a muchos).
- Infraestructura necesaria (evaluación de costos, accesibilidad).
- Ancho de banda (suficiente para transmitir al menos mil píxeles a 24 fps hacia los nodos receptores).
- Cantidad de nodos posibles en la red (varios cientos).
- Rango de alcance (mayor a 50 m).

Se evaluaron distintas alternativas, como ser Bluetooth, ZigBee, WiMax y Wi-Fi. Finalmente, y gracias a la colaboración de los docentes Matías Richart y Javier Baliosian, del grupo MINA del Instituto de Computación [77], se decidió utilizar Wi-Fi (IEEE 802.11).

Esta tecnología cumple con todos los requerimientos planteados: ofrece topologías de red capaces de comunicar un nodo con muchos otros, anchos de banda que van desde los 11 Mbps, capacidad para la conexión de varios cientos de nodos y posibilidad de operar en rangos de alcance que pueden sobrepasar los 100 metros [63][78]. Además, los componentes necesarios para montar una red de este tipo son fácilmente obtenibles, es viable extender su alcance empleando dispositivos repetidores y existen múltiples estudios sobre la utilización de esta tecnología como medio de transmisión de contenido multimedia que dan respaldo a su elección [79][80].

4.2.2 Método de transmisión en red local

En Sendero, cada píxel requiere de 3 bytes de información para ser representado. Cada Bondibar es capaz de controlar 8 píxeles, por lo que si cada una de estas placas fuera considerada un nodo de la red recibiría paquetes cuya carga útil sería de hasta 24 bytes.

La red de Sendero debería proveer la capacidad de soportar varios cientos (o incluso miles) de píxeles, lo que finalmente derivaría en las múltiples problemáticas a las que se enfrentan las redes con estas características, o redes densas [81][82].

Para evadir estos problemas y hacer un uso eficiente de la red, se propone utilizar una política de enrutamiento en multidifusión (*multicast*) con una topología conformada por un BSS (*Basic Service Set* [83]). De modo que, en lugar de enviar paquetes (*unicast*) con 24 bytes a cada nodo, se enviarán paquetes más grandes, que agrupen la información de color de múltiples nodos, siendo responsabilidad de cada uno extraer los datos que deben procesar.

Esto permite gran escalabilidad, introduciendo poco *overhead* en la red [79], aunque aplica ciertas exigencias sobre el poder de cómputo necesario en los nodos.

4.2.3 Nodos de la red inalámbrica

En el contexto de la arquitectura M2M planteada, se determinó designar el rol de Dispositivo M2M a un componente constituido por un controlador de LEDs (Bondibar), y un módulo que provee a este de conectividad inalámbrica y capacidad de procesamiento.

En cuanto al diseño de este dispositivo se plantearon dos opciones:

1. Desarrollar un nuevo circuito para el controlador de LEDs que incluya un módulo de conectividad inalámbrica dentro del mismo PCB.

2. Desarrollar un componente externo que pueda ser conectado a un controlador Bondibar, en particular, utilizando la interfaz USB que expone los pines SPI de los LED *drivers* en su interior.

Asumiendo que se tiene un módulo capaz de participar en cualquiera de estos diseños, se dio prioridad a la minimización de las modificaciones de *hardware* requeridas en el controlador Bondibar, y a la flexibilidad de la solución.

Esto lleva a optar por el segundo diseño planteado. Dicha solución independiza al módulo inalámbrico, pudiendo ser cambiado o actualizado, y solo exige al controlador de LEDs poder exponer a través de su interfaz USB una fuente de alimentación. En la Sección 4.3.3.1 se ofrece una explicación detallada de su implementación.

4.2.4 Módulo inalámbrico

Con las decisiones anteriores tomadas, es preciso obtener un módulo inalámbrico programable, con capacidad para conectarse a redes IEEE 802.11 y llevar a cabo comunicaciones seriales a través del protocolo SPI.

Se relevaron múltiples alternativas disponibles en el mercado: ESP-8266 [84], EMW-3165 [85], CC3200MOD [86], SOM9331 [87], Ralink RT5350 [88], NanoPi [89], Arduino + Wifi Shield [90], Raspberry B+ [91], ARM mBed [69]. De éstas, fueron evaluadas los siguientes aspectos: velocidad del CPU, voltaje de operación, cantidad de puertos GPIO, SPI, otros protocolos seriales, tamaño de comunidad, costo y modo de programación.

Finalmente, se decidió avanzar sobre el módulo ESP8266 [84], en particular con la placa ESP-12E DevKit [92], y construir un circuito adaptador para acoplarlo a un controlador Bondibar. En el Anexo H se brinda una descripción más detallada sobre este dispositivo.

4.2.5 Migración de plataforma

La versión base de Sendero se encuentra desarrollada para la plataforma MacOS. Debido a que no se contaba con suficientes dispositivos compatibles, se decidió migrar el código para que pueda ser compilado en sistemas Linux; específicamente, en Ubuntu 14.04 [93].

4.2.6 *Middleware*

Dado que la comunicación con los controladores de LED será ahora responsabilidad de los módulos inalámbricos, y se desea realizar un tratamiento personalizado de los datos para su agrupación y distribución en grupos *multicast*, sin perder la capacidad de emplear Art-Net DMX como protocolo de salida desde Sendero Server, se decidió implementar un *middleware* en reemplazo del dispositivo mBed NXP LPC17680.

Esta pieza de *software*, será responsable de implementar el protocolo Art-Net DMX y comunicar los datos de manera apropiada a los dispositivos de la red, conformados por

los módulos inalámbricos y los controladores de LED. Adicionalmente, este *middleware* será el administrador de la sesión de *streaming* y las configuraciones de los nodos.

4.2.7 Clientes remotos

Otro de los grandes objetivos de este proyecto implica el desarrollo de los clientes remotos, aplicaciones capaces de reproducir e interactuar con la obra en tiempo real de manera no presencial.

En la versión de Sendero tomada como base, fueron desarrolladas aplicaciones Android y iOS que ofician de cliente remoto, recibiendo un flujo de datos a través de sockets UDP y enviando comandos de interacción como mensajes POST HTTP a un servidor web.

Con el fin de ofrecer mayor accesibilidad, y compatibilidad con múltiples plataformas y dispositivos, se propone la implementación de una aplicación web accesible desde Internet.

En cuanto a la comunicación, esta aplicación necesita recibir el flujo de datos a reproducir, y enviar comandos de interacción, ambos con la menor latencia posible.
Adicionalmente, se debe brindar soporte para la conexión de una cantidad potencialmente grande de usuarios. Se evaluaron las tecnologías estudiadas (ver Sección 2.3.3.4), y en base a estudios sobre su escalabilidad y performance [94][95], se decidió optar por WebSockets, tanto para el flujo de reproducción como para el envío de interacciones.

4.2.8 Servidor de *Streaming*

Con el fin de que los clientes remotos puedan obtener la información de color de la obra desde cualquier parte del mundo, esta debe ser accesible a través de Internet. Para ello, se desplegará un servidor web capaz de recibir dicha información desde Sendero Server y re-transmitirla, con la menor latencia posible, hacia los clientes web conectados.

4.2.9 Servidor de interacciones

Del mismo modo que con el flujo de reproducción, para dar soporte a la participación de los espectadores remotos en la generación del contenido de la obra, será desplegado un servidor web capaz de recibir comandos de interacción, a través de conexiones WebSocket. Este servidor además tendrá la tarea de comunicar los comandos apropiadamente a una instancia de Sendero Client especializada, que se será responsable de integrar estas entradas al ciclo de *blending* de Sendero Server.

4.2.10 Técnicas de transferencia de contenidos multimedia

Con las decisiones planteadas hasta el momento, existen dos flujos de datos que deben ser atendidos. El flujo *multicast* en la red local, y el requerido por los clientes remotos.

Bajo los lineamientos de los estudios realizados en el Capítulo 2 (ver Sección 2.3.2), se resolvieron una serie aspectos listados a continuación.

4.2.10.1 UDP, número de secuencia, marca de tiempo, planificación.

Se utilizará UDP como protocolo de capa de transporte, tanto en la red local como en la conexión entre Sendero Server y el servidor de *streaming*. Desde este último hacía los clientes remotos se utilizará la tecnología WebSockets para minimizar la latencia (con respecto a HTTP) y evadir las dificultades que los navegadores web plantean respecto al uso de sockets UDP (por razones de seguridad).

Para ambos flujos serán desarrollados protocolos que incluyan un cabezal con números de secuencia y marcas de tiempo que serán utilizados para la planificación de la reproducción y la recuperación frente a pérdidas, de forma análoga al protocolo RTP estudiado pero utilizando un conjunto mínimo de sus campos.

En cuanto a la planificación de la reproducción, se propone implementar un esquema de planificación con retardo fijo para la red local, y un esquema con retardo adaptativo para el flujo enviado hacia los clientes remotos (ya que este será transmitido a través de Internet). De este modo, se puede absorber cualquier *jitter*, ya sea ocasionado por la red o por el procesamiento de los datos.

4.2.10.2 Compresión

La compresión de contenido multimedia es en sí mismo es un tema muy amplio. Dado que la red local implementada con la tecnología IEEE 802.11 provee un ancho de banda suficientemente holgado para los requerimientos planteados, y los módulos inalámbricos poseen restricciones de procesamiento, se optó por no utilizar compresión en los datos que circulan por esta red.

Por otra parte, debido a que los clientes remotos poseen una considerable capacidad de procesamiento, resulta viable la utilización de mecanismos de compresión. Sin embargo, se tiene una restricción en cuanto a la minimización de la latencia de reproducción, lo que motiva la búsqueda de algún algoritmo particularmente rápido.

Por su simplicidad, compatibilidad y velocidad, el algoritmo seleccionado fue LZ4 [96].

4.2.10.3 Recuperación frente a pérdidas

Dado que se utiliza UDP como protocolo de capa de transporte, no se tiene ninguna garantía de que los paquetes enviados a través de la red lleguen a destino. De este modo, se implementará un mecanismo para la recuperación frente a pérdidas.

4.2.10.4 Control y estadísticas

La utilización de estas técnicas, y la necesidad de realizar configuraciones y/o ajustes sobre los parámetros de la red y la reproducción, motivan la decisión de implementar un canal de control que permita la obtención de estadísticas.

4.2.11 Sincronización de la reproducción entre módulos inalámbricos

Dado un *frame* de datos, todos los nodos de la red deben reproducirlo en el mismo momento (aproximadamente). La versión base de Sendero comunica el contenido visual directamente a los controladores de LED, empleando un cable de datos. Al recibir esta información, es reproducida de inmediato lo que garantiza el sincronismo entre los nodos.

La migración del esquema de comunicación, y la consecuente distribución del procesamiento de los datos en los nodos de la red, introduce la problemática de que el mismo *frame* puede ser reproducido en diferentes momentos, según como haya sido procesado por cada nodo.

Por esto, se decidió implementar un mecanismo de sincronización de la reproducción, que permita garantizar la reproducción sincronizada de los *frames*. Se relevaron varias alternativas, como la sincronización de los relojes de los nodos, utilizando por ejemplo NTP [97], sin embargo, se optó por no introducir *overhead* en la red, y sacar provecho del mecanismo de multidifusión utilizado, el algoritmo desarrollado se explica más adelante.

4.3 Desarrollo de la solución

En esta sección se brinda una descripción detallada de los desarrollos llevados a cabo para satisfacer los requerimientos del proyecto, bajo los lineamientos de las decisiones anteriormente presentadas.

4.3.1 Descripción general y arquitectura

La solución propuesta comprende la modificación de ciertos componentes de *software* y *hardware* existentes, la creación de nuevos, y el desarrollo de protocolos de comunicación que permitan implementar de manera efectiva las técnicas, mecanismos y tecnologías relevadas en el Capítulo 2.

La Figura 4-1, resume de manera simplificada los componentes involucrados en la solución, y el modo en el cual estos interactúan. Adicionalmente, denota (con diferentes colores) los componentes que fueron creados, modificados o resultaron inalterados.

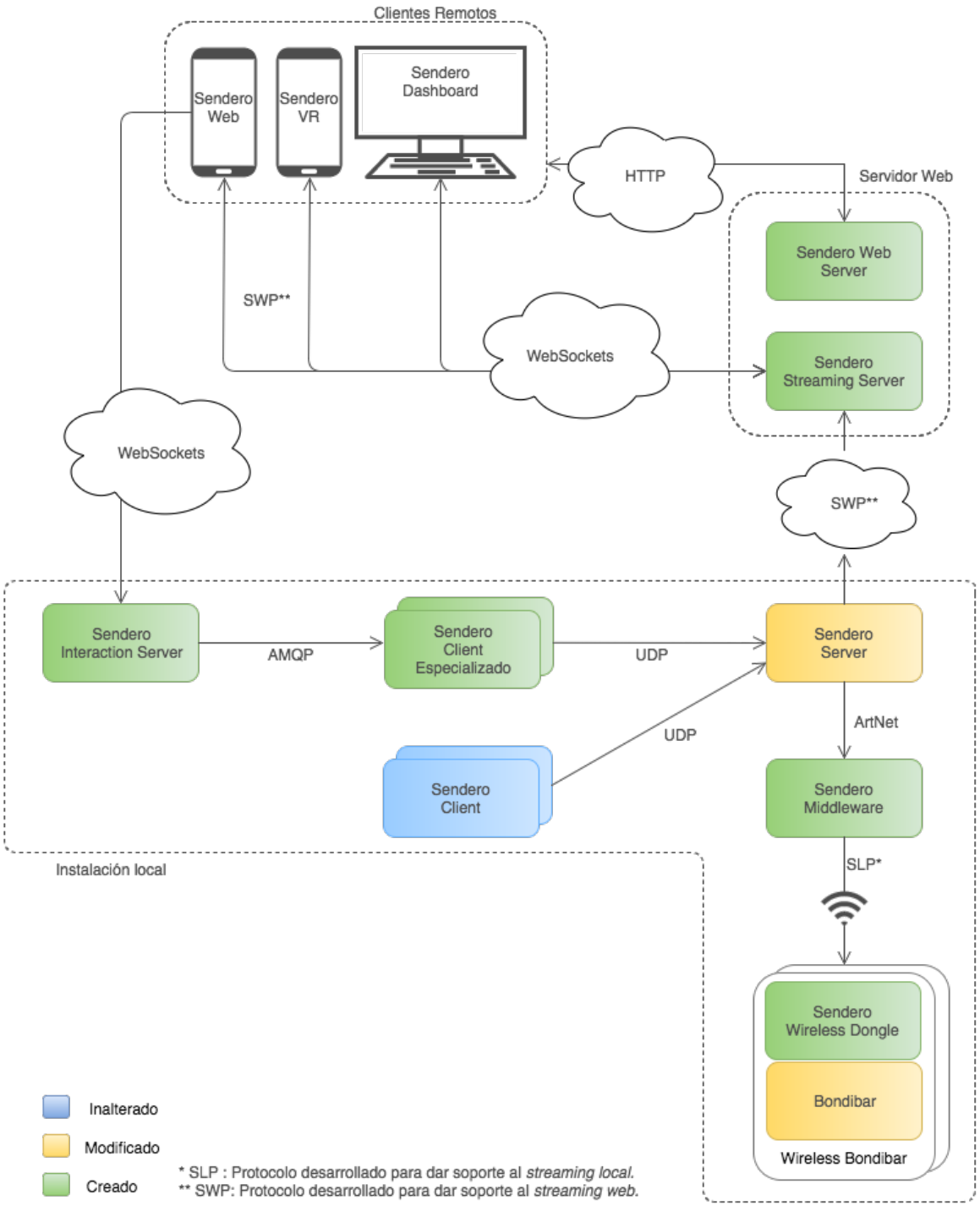


Figura 4-1 - Arquitectura general de la solución

En primer lugar, **Sendero Server,** sirve como componente central de la solución de *software*. Es el responsable de generar el contenido visual a reproducir, a partir de entradas proporcionadas por distintas implementaciones de **Sendero Client** (de manera análoga a como se hace en la versión de Sendero tomada como base). **Sendero Server** se comunica con **Sendero Middleware**, empleando el protocolo Art-Net DMX, con el

fin de transmitir el contenido generado a los controladores de LED, y con **Sendero Streaming Server** para hacer accesible este contenido a los clientes remotos.

Sendero Middleware, es la pieza de *software* construida en reemplazo del microcontrolador mBed NXP LPC17680 eliminado. Sus principales objetivos son recibir, procesar y retransmitir el contenido de los paquetes Art-Net DMX enviados por **Sendero Server** y gestionar los dispositivos de la red inalámbrica.

Sendero Streaming Server es un servidor web cuya finalidad es difundir el contenido visual generado por **Sendero Server** hacia todos los clientes remotos conectados. Adicionalmente, administra una interfaz web (**Sendero Dashboard**) que permite visualizar en tiempo real el estado del flujo de datos.

Sendero Web Server es otro servidor web encargado de servir las aplicaciones que ofician de cliente remoto, **Sendero Web** ***(http://web.sendero.uy)*** y **Sendero Cardboard** ***(http://vr.sendero.uy)***.

Por otra parte, **Sendero Interaction Server** es un servidor web que tiene como objetivo recibir comandos en representación de las interacciones llevadas a cabo por los usuarios remotos en la aplicación **Sendero Web.** Dichas interacciones son comunicadas a una instancia de **Sendero Client** desarrollada específicamente para tal fin, que las procesa e integra al ciclo de *blending* de **Sendero Server**.

Finalmente, los nodos **Wireless Bondibar** están compuestos por *firmware* y una serie de componentes de *hardware* constituida por uno o varios controladores de LED (**Bondibar**), conectados a un **Sendero Wireless Dongle** (el cual posee una placa ESP-12E montada) que les provee conectividad inalámbrica y capacidad de procesamiento. Estos son los nodos de la red inalámbrica de Sendero, y su principal responsabilidad es manejar los píxeles de las obras.

Cada uno de estos componentes, al igual que los protocolos con los que se comunican, serán descritos en detalle en lo que resta de este capítulo.

4.3.2 Protocolos

En esta sección se describirán los protocolos diseñados con el objetivo de cumplir los requerimientos de red del sistema. Básicamente, se desarrollaron dos protocolos de capa de aplicación: Sendero Lighting Protocol (SLP) y Sendero Web Protocol (SWP). La Figura 4-1 muestra un esquema de su función dentro de la arquitectura general de la solución.

4.3.2.1 SLP (Sendero Lighting Protocol)

Con el objetivo de satisfacer los requerimientos de la comunicación inalámbrica entre los componentes Sendero Middleware y los nodos receptores, Wireless Bondibar, se desarrolló un protocolo de comunicación bajo las siguientes premisas:

1. Simple.
2. *Overhead* mínimo.
3. Que permita sincronizar la reproducción entre los receptores.
4. Que permita la reproducción ordenada de los *frames*.
5. Que permita, potencialmente, la corrección de errores.
6. Que permita el envío de comandos de control hacia los receptores.
7. Que permita el descubrimiento de nuevos nodos receptores en la red.

De forma análoga a los RTCP y FTP, se decidió dividir la implementación de estas funcionalidades en distintos canales de comunicación: un **canal de datos** y un **canal de control**. Adicionalmente, el protocolo provee un mecanismo de descubrimiento para contemplar la conexión de nuevos receptores.

Canal de datos

En este canal serán transmitidos los paquetes denominados **SLP Data Packet,** cuyo contenido principal es la información de color de los píxeles de la obra. Debido a que este contenido se comporta como un *streaming* multimedia, se decidió utilizar UDP como capa de transporte para su implementación. Adicionalmente, si bien el protocolo desarrollado no restringe su utilización en modo *multicast*, (como fue mencionado en las decisiones de diseño) la sincronización entre receptores se basa en esta modalidad de emisión.

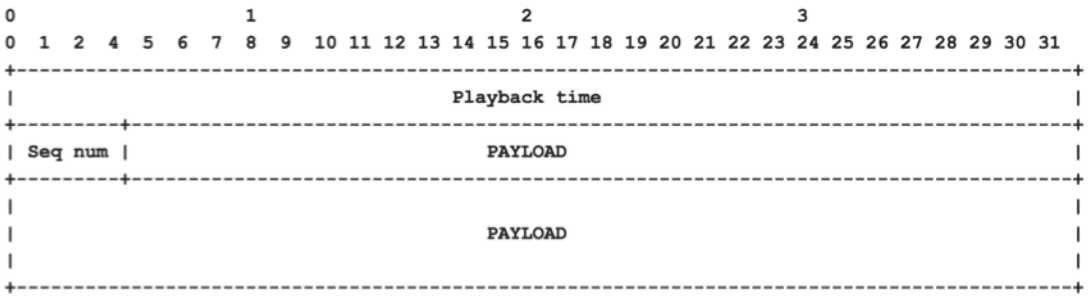


Figura 4-2 - Formato de paquete SLP Data Packet.

En la Figura 4-2 se presenta el formato del paquete SLP Data Packet, donde:

- **Playback time:** Es el tiempo de generación del *frame* expresado en milisegundos en referencia al reloj del emisor (Entero sin signo de 4 *bytes*).
- **Seq num:** Es el número de secuencia del *frame* dentro del flujo (Entero sin signo de 1 *byte*).
- **Flags:** 1 *byte* de flags. En la implementación actual del protocolo no es utilizado.
- **Payload:** *Array* de *bytes* que contiene la información de color en formato RGB donde cada componente de color ocupa 1 *byte*. En total, 3 *bytes* por píxel.

Canal de control

El canal de control es implementado como una conexión TCP paralela al flujo de datos, que se utiliza para diversos fines tales como configuraciones e intercambio de estadísticas. En la implementación de esta versión de Sendero, se utilizará para configurar el comportamiento, proporcionar las configuraciones de la obra y especificar qué píxeles controla cada receptor Wireless Bondibar. Adicionalmente, será utilizado para la recolección de estadísticas calculadas en cada nodo receptor. Cabe aclarar que en esta sección, se referirá como emisor a la entidad responsable de administrar los dispositivos (Sendero Middleware) y como receptor a los componentes inalámbricos receptores (Wireless Bondibar), pero en el contexto del canal de control los roles pueden intercambiarse ya que es considerado un flujo bidireccional de datos. Por otra parte, se introduce aquí el concepto de sesión de un dispositivo como el ciclo de vida del mismo con una conexión de control activa.

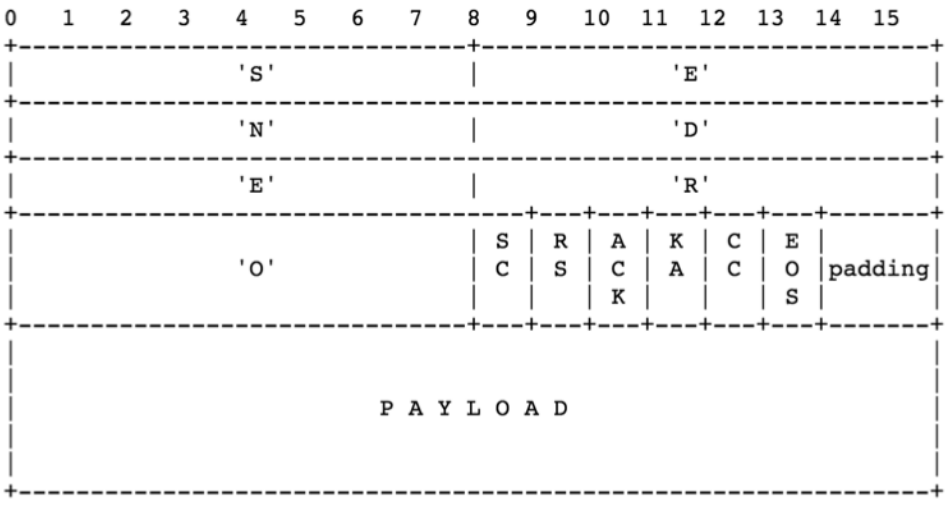


Figura 4-3 - Formato de paquete SPL Control Packet

Con el fin de satisfacer este tipo de requerimientos, se diseñó el paquete **SLP Control Packet** (ver Figura 4-3) donde:

- [**SC**] **Set Configuration Flag:** Bandera utilizada por el emisor para configurar el dispositivo receptor. En este caso, el *payload* contiene información de configuración en el formato descrito por la siguiente expresión regular:

$$((\w+):(\w+)\ )*((\w+):(\w+))\n$$

- [**RS**] **Request Statistics Flag:** Bandera utilizada para solicitar a un receptor inalámbrico estadísticas de su funcionamiento. El conjunto de estadísticas es enviada en el payload del paquete en el mismo formato que las configuraciones.
- [**ACK**] **Ack Flag:** Bandera utilizada por el receptor (en caso de existir un paquete de respuesta) para indicar la correcta recepción y procesamiento del mismo.
- [**KA**] **Keep Alive Flag:** Bandera utilizada para indicar que el paquete es un

Keep Alive. Estos paquetes son utilizados para indicar a los receptores que se encuentran en una sesión activa.

- **[CC] Close Connection Flag:** Bandera utilizada por Sendero Middleware para indicar a un receptor que debe cerrar la conexión TCP de control. La misma podrá ser reiniciada posteriormente y conceptualmente sigue activa.
- **[EOS] End of Session Flag:** Bandera utilizada por el emisor para indicar a un módulo que debe cerrar la conexión y finalizar la sesión, y por lo tanto, la sesión no podrá ser reiniciada. En este estado la conexión no se considera activa, y el dispositivo no recibirá más paquetes por el canal de datos hasta iniciar una nueva sesión.
- **Payload:** *Array* de bytes cuyo contenido depende exclusivamente de las banderas activas en el paquete.

Los paquetes con la *flag* SC o RS activa, exigen al receptor emitir una respuesta con la *flag* ACK activa, de lo contrario el intercambio de información es considerado no exitoso.

El diseño de este canal exige que la responsabilidad de iniciar intercambios de información sea del nodo emisor. Dicho de otra forma, y hablando estrictamente de una conexión TCP, el nodo receptor oficiará el rol de servidor y el rol de cliente será implementado por el emisor.

Mecanismo de descubrimiento

Como será explicado posteriormente, una de las funcionalidades más importantes del componente Sendero Middleware es la administración de los módulos inalámbricos (Wireless Bondibar) existentes en la red. Para evitar la utilización de IPs estáticas en los dispositivos (debido a la poca flexibilidad que esto conlleva), cuando un nodo receptor se conecta a la red, deberá anunciar su existencia a través de la emisión de paquetes **SLP Discovery Packet** en modo *multicast* UDP al puerto 8888. Estos anuncios serán emitidos en intervalos de 2 segundos conteniendo el identificador único de dispositivo como carga útil, hasta recibir una conexión TCP al puerto 8889. Estos paquetes de descubrimiento utilizan el formato estándar del protocolo SDP utilizado para la descripción de sesiones [45].

Este mecanismo de descubrimiento fue inspirado en el protocolo SAP (*Session Announcement Protocol* [44]), pero a diferencia de éste (en donde existe una entidad que anuncia la existencia de sesiones y los interesados en participar realizan peticiones a ésta), cada nodo interesado en participar anuncia su interés activamente. Las razones de esta modificación se basan en brindar soporte al escenario en donde los nodos receptores puedan reiniciarse por alguna razón (condiciones de error) y requieran iniciar la sesión nuevamente sin la necesidad de que Sendero Middleware este continuamente anunciando la sesión (lo cual introduciría tráfico innecesario en la red durante el desarrollo de una obra).

4.3.2.2 SWP (Sendero Web Protocol)

El objetivo de este protocolo es resolver la transferencia de datos (a través de Internet) desde Sendero Server hacia Sendero Streaming Server y desde éste hacia los clientes remotos. El mismo debe proveer metadatos para la correcta reproducción de los *frames* generados por Sendero Server. Para ello, se utiliza un paquete con el mismo formato que SLP Data Packet (ver Figura 4-2). La única diferencia radica en que el campo *flags* es utilizado para indicar si el *payload* se encuentra comprimido o no. Como fue mencionado en las decisiones de diseño, el algoritmo utilizado es LZ4.

La capa de transporte utilizada depende del par de componentes que ésta comunica entre sí. En el tramo desde Sendero Server hacia Sendero Streaming Server se utiliza como protocolo de capa de transporte UDP. Por otro lado, la comunicación desde Sendero Streaming Server hacia los clientes remotos se encuentra limitada al uso de TCP, ya que no es posible utilizar UDP en conexiones con navegadores web (a menos de la utilización de *plugins*). Debido a esta limitante y al relevamiento de tecnologías desarrollado en el Capítulo 2, se utilizó la tecnología de WebSockets (bajo TCP). Como ventajas de su utilización se puede destacar la baja latencia de comunicación, un *overhead* por cabeceras mínimo y la existencia de implementaciones nativas en la mayoría de los navegadores web actuales. En el Anexo D se brinda una exposición detallada sobre esta tecnología.

4.3.3 Implementación

Las secciones a continuación describen los detalles de implementación del *hardware* y *software* desarrollado.

4.3.3.1 Hardware

La solución de *hardware* propuesta comprende:

1. La eliminación del microcontrolador mBed como componente central de la solución.
2. El montaje de una red WLAN 802.11.
3. El desarrollo de un componente de *hardware* (Sendero Wireless Dongle) capaz de acoplarse a una Bondibar para dotarla de conectividad inalámbrica y procesamiento.
4. La realización de las modificaciones pertinentes en el componente Bondibar para dar soporte a la conexión del dispositivo inalámbrico.

Red inalámbrica - WLAN 802.11

La red WLAN desplegada se conformada por un único BSS (*Basic Service Set*), compuesto por un *Access Point* que puede ser cualquier *router* doméstico, y una serie de nodos, Wireless Bondibar, constituidos por uno o varios controladores Bondibar conectados a un Sendero Wireless Dongle (como se ilustra en la Figura 4-4).

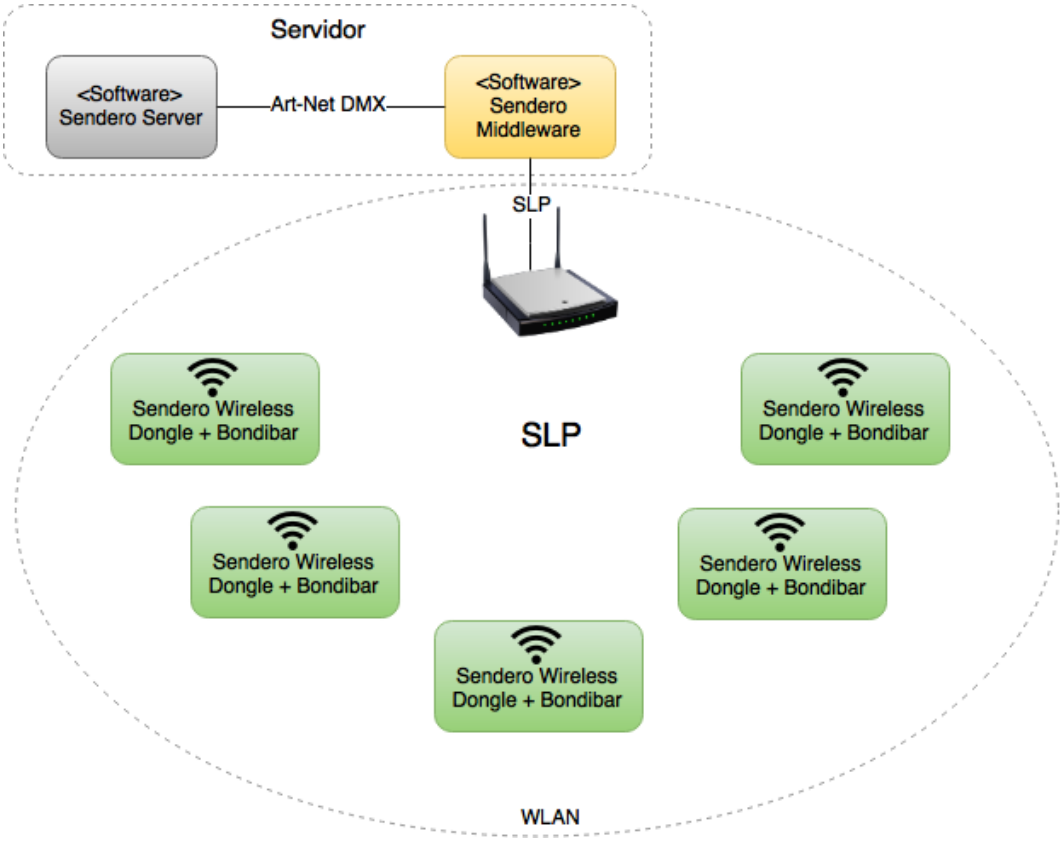


Figura 4-4 - Esquema de componentes de la red WLAN de Sendero.

Sobre esta red se transmite a una frecuencia de 24 o 30 veces por segundo la información de color emitida por Sendero Server. Esta información es recibida y procesada por los nodos que implementan el protocolo SLP y reproducida sobre los LEDs de la obra. La frecuencia de transmisión impone una exigencia sobre la utilización del protocolo IEEE 802.11, en particular respecto al modo en que este realiza la gestión de la energía.

El protocolo IEEE 802.11 fue diseñado para dispositivos móviles, muchos de los cuales son alimentados por baterías. Por ello, este protocolo incluye una serie de parámetros que permiten que sus estaciones ahorren energía. Uno de estos parámetros, llamado DTIM, es un valor entero positivo configurado en el punto de acceso por su administrador y es notificado a las estaciones por medio de las tramas *Beacon*. Solamente si alguna de las estaciones conectadas posee el modo de ahorro de energía habilitado, el punto de acceso envía las tramas *multicast/broadcast* cada DTIM intervalos *Beacon*. De esta forma, las estaciones pueden permanecer en modo *sleep* y "despertar" únicamente para recibir estas tramas, mejorando así el rendimiento de sus baterías a costas del tamaño de *buffering* en el punto de acceso y retrasos en la recepción [98].

Este funcionamiento introduce *jitter* a la entrega de paquetes por parte del *router*, y por lo tanto, se tuvo en cuenta deshabilitar el modo de gestión de energía de Wi-Fi, o bien configurar el envío de tramas DTIM con la mayor frecuencia posible.

Sendero Wireless Dongle

Sendero Wireless Dongle, es un componente de *hardware* que consiste en un módulo ESP-12E DevKit [92] y un circuito adaptador que le permite acoplarse a una placa Bondibar a través de la interfaz USB de entrada que esta última posee.

Sus principales objetivos son:

- Proveer de conectividad inalámbrica y procesamiento a un controlador de LEDs.
- Implementar el protocolo SLP.
- Comunicar de manera apropiada el contenido recibido al controlador conectado.

Respecto al módulo ESP-12E DevKit (Figura 4-5), el mismo encapsula un microcontrolador inalámbrico programable ESP8266 [99], con una CPU RISC de 32 bits que puede ejecutar a 80 o 160 MHz, conectividad inalámbrica 802.11 b/g/n, y soporte para múltiples interfaces de *hardware* como GPIO, I2C, ADC, SPI y PWM. La programación del módulo puede ser llevada a cabo a través de una interfaz micro USB para tal propósito, y desarrollada bajo una adaptación del IDE Arduino [100].

Para la implementación del protocolo SLP y la comunicación SPI con los controladores Bondibar, se desarrolló un *firmware,* del cual se brindan detalles en el apartado Sendero Wireless Bondibar de la Sección 4.3.3.2.

Figura 4-5 - Módulo inalámbrico ESP-12E DevKit

Por otra parte, el circuito adaptador desarrollado tiene como fin facilitar la conexión entre el módulo ESP-12E y el controlador Bondibar, brindando además la alimentación adecuada al módulo. Para ello, conecta los pines D5 (GPIO13, SPI *clock*), D7 (GPIO14, SPI MOSI), VIN (5V) y GND del módulo inalámbrico a los pines DATA+, DATA-, VCC y GND respectivamente de una interfaz USB, que puede ser fácilmente acoplada a un controlador de LEDs de Sendero (Figura 4-6). Esto permite la comunicación con los LED *drivers* WS2801 que controlan los píxeles de la obra, y la obtención de energía para alimentar al módulo.

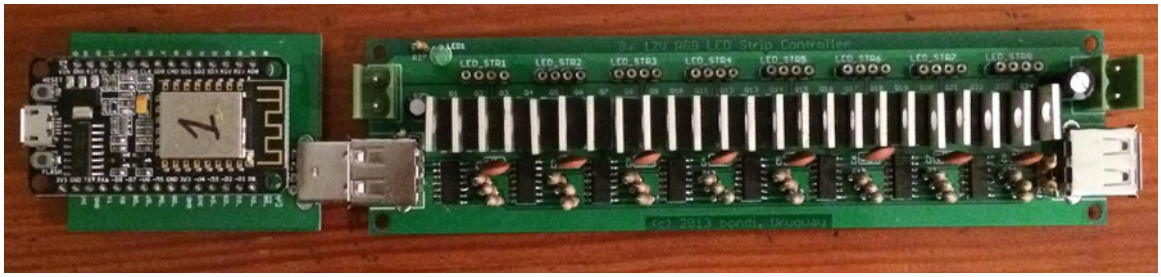


Figura 4-6 - Sendero Wireless Dongle conectado a controlador Bondibar

La Figura 4-7 muestra, a la izquierda el esquemático del circuito adaptador desarrollado con Eagle [101], y a la derecha una fotografía de la placa construida.

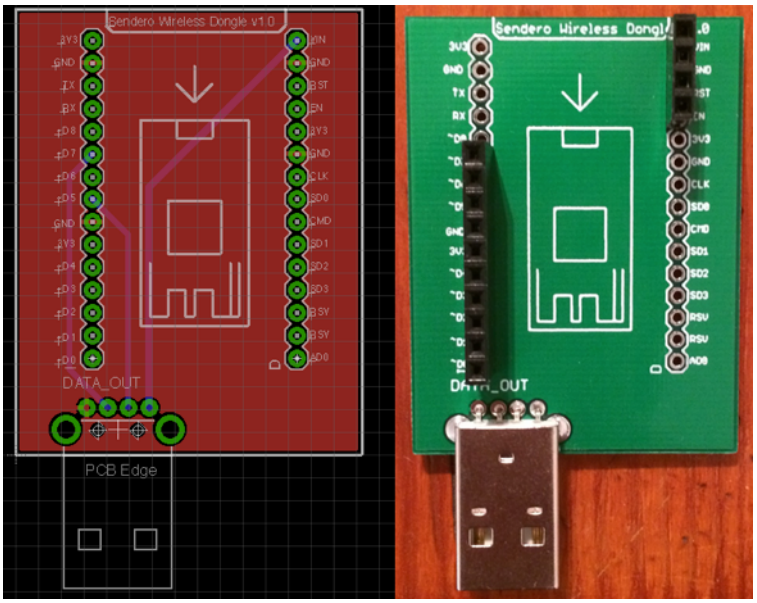


Figura 4-7 - Sendero Wireless Dongle.

Modificaciones en los controladores de LEDs (Bondibar)

En el esquema de conexión de la versión base de Sendero, los controladores de LEDs recibían la información de color a través de un cable de datos proveniente del microcontrolador mBed, conectado a la interfaz USB del componente Bondibar de modo tal, que sólo los pines de datos de esta interfaz (DATA+ y DATA-) eran considerados, conectando a tierra los pines restantes (VCC y GND).

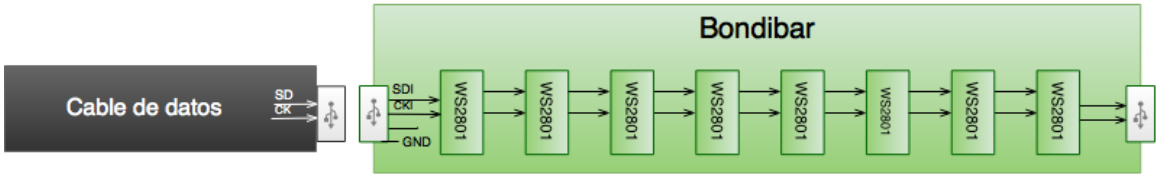


Figura 4-8 - Esquema de conexión de *hardware* legacy.

Con el objetivo de minimizar los cambios en este componente, y poder alimentar los módulos ESP-12E montados en Sendero Wireless Dongle, se sustituyó el regulador de voltaje 78L05 [102] por un estándar 7805 [103] capaz de proveer una corriente continua de 5v y 1A, suficiente para cubrir los requerimientos del módulo inalámbrico (4.5V ~ 9V, ≈70mA y 200mA como máximo). Este regulador conecta su salida al pin VCC de la interfaz USB donde se acopla Sendero Wireless Dongle, y de este modo, el módulo ESP-12E puede recibir la alimentación que necesita (Figura 4-9).

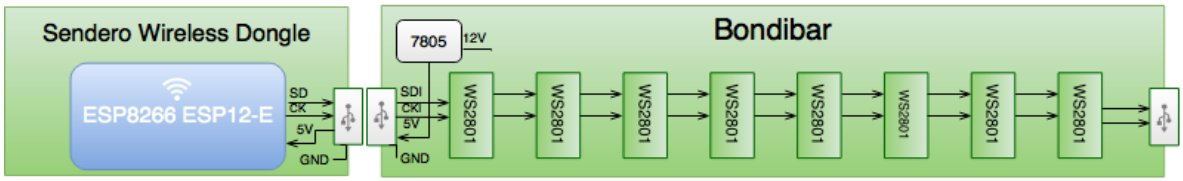


Figura 4-9 - Esquema de comunicación de *hardware* nuevo.

Para la realización de esta modificación se contó con el apoyo del diseñador del componente Bondibar, Pablo Gindel. La gestión de los componentes (Anexo I) y la construcción de las placas fue llevada a cabo por los desarrolladores de este proyecto.

La nueva versión de esta placa y el esquemático del circuito de Sendero Wireless Dongle se entregan como parte de la distribución *open-source* de Sendero.

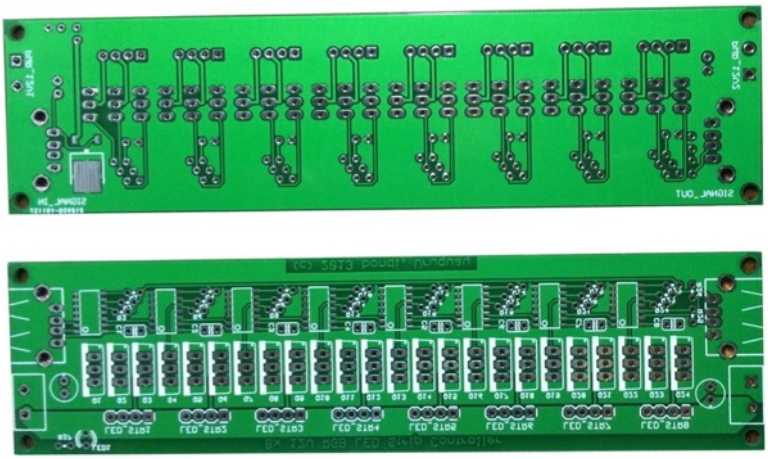


Figura 4-10 - Bondibar modificada para alimentar a Sendero Wireless Dongle.

4.3.3.2 *Software*

Sendero Server

En primer lugar, Sendero Server debió ser adaptado para dar soporte a la comunicación con los clientes remotos. Para ello se implementó la clase ***StreamServerManager***, encargada de recibir, comprimir, empaquetar y enviar los *frames* de la obra a Sendero Streaming Server. Estas tareas se realizan en un *thread* independiente, siguiendo el flujo representado en el diagrama de actividad mostrado en la Figura 4-11.

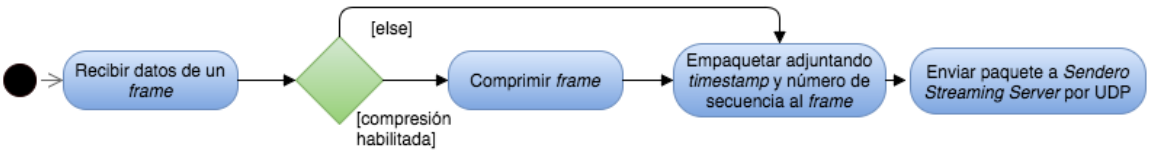


Figura 4-11 - Flujo del *thread* principal de StreamServerManager

La compresión de los *frames* es opcional, y en caso de estar habilitada se realiza utilizando la librería multiplataforma [104]. Por su parte, el empaquetado consiste en la creación de un paquete SWP. Aparte de esto, Sendero Server no sufrió ninguna otra modificación mayor.

Sendero Middleware

Sendero Middleware es el componente creado en reemplazo del microcontrolador mBed (ver Sección 3.1.1.2), y su rol principal es funcionar como intermediario entre Sendero Server y los módulos inalámbricos. Este componente es configurado como un *Device* de Sendero Server, y es capaz de recibir y decodificar paquetes ArtNet-DMX para retransmitirlos en el formato de Sendero Lighting Protocol (SLP) hacia los nodos inalámbricos, Wireless Bondibar. Adicionalmente, gestiona y controla la conexión con estos nodos.

Para la implementación de este componente se escogió el lenguaje de programación Python [105], en su versión 3.4.

Se desarrollaron dos grandes modos de ejecución: **modo *Art-Net*** y **modo desarrollo**. El primero se utiliza para recibir, procesar y transformar datos Art-Net DMX enviados por Sendero Server. Mientras que el segundo es utilizado principalmente para propósitos de *testing*, permitiendo generar y enviar flujos de prueba hacia los nodos inalámbricos.

Arquitectura

Sendero Middleware se divide en 4 sub-componentes: *Devices, Networking, Streaming y Configuration.*

El módulo *Devices* es el gestor de los dispositivos inalámbricos que se conectan a Sendero Middleware. Registra y/o rechaza a aquellos dispositivos que intenten establecer la conexión con él y, a aquellos que establecen la conexión, les envía las configuraciones necesarias para que puedan comenzar a recibir el *streaming* de datos. Además es responsable de enviar paquetes de control y solicitar estadísticas a los dispositivos.

Networking es el módulo encargado de manejar los grupos *multicast* a los que los dispositivos inalámbricos se conectan, asignando direcciones IP y administrando a qué grupo se debe conectar cada dispositivo. A su vez, es responsable de enviar los paquetes SLP.

El módulo *Streaming* recibe el flujo de paquetes Art-Net DMX (enviado por Sendero Server), lo decodifica, encapsula los datos del *frame* en un SLP Data Packet y envía al módulo *Networking* para que sea retransmitido a los dispositivos inalámbricos.

Por último, en el módulo *Configuration* es donde se configuran parámetros como la cantidad de dispositivos a conectar, qué cantidad de LEDs maneja cada uno, y configuraciones acerca de la sincronización de la reproducción entre otras.

Devices

El objetivo de este módulo es gestionar la conexión de los dispositivos inalámbricos.

Antes de iniciar un sesión de reproducción, los dispositivos inalámbricos deben registrarse y obtener una serie de configuraciones por parte de Sendero Middleware. Para ello, estos dispositivos emiten una señal de descubrimiento en modo *multicast*, utilizando paquetes SLP Discovery Packet que contienen su identificador único.

Por su parte, Sendero Middleware espera (en un *thread*) por estos paquetes de descubrimiento, y concede o deniega el establecimiento de una conexión verificando la validez del dispositivo solicitante en una configuración brindada por el módulo *Configuration*. Si la conexión es concedida, entonces se establece una conexión TCP con el dispositivo para comunicarle las configuraciones de la sesión y finalizar el proceso de descubrimiento. El diagrama de actividad de la Figura 4-12 muestra la secuencia descrita.

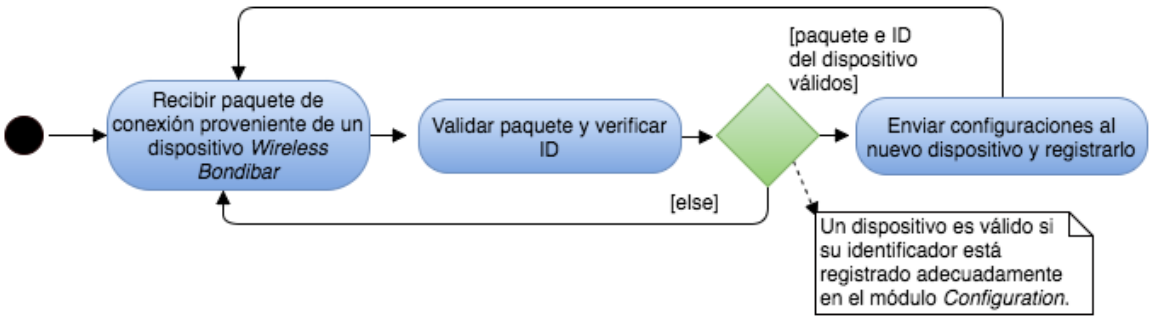


Figura 4-12 - Flujo principal del módulo Devices.

Para el registro de los dispositivos se implementa una clase (denominada *Device*) cuyas instancias se almacenan en una lista de dispositivos conectados. Esta clase encapsula cierta información de los dispositivos como ser su identificador único y dirección IP, información que es necesaria para el manejo del canal de control, a través del cual es posible modificar configuraciones de la sesión y solicitar estadísticas.

La solicitud de estadísticas es opcional y, en caso de estar habilitada, se realiza a intervalos regulares según cómo se defina en el módulo *Configuration*. El formato y contenido de las mismas se detalla más adelante.

Streaming

Como ya se mencionó, la principal tarea de Sendero Middleware es recibir un *streaming* Art-Net DMX proveniente de Sendero Server, procesarlo y re-transmitirlo hacia los dispositivos inalámbricos respetando el protocolo SLP. Dicha tarea es llevada a cabo por el módulo *Steaming*, como se describe en el diagrama de actividad de la Figura 4-13.

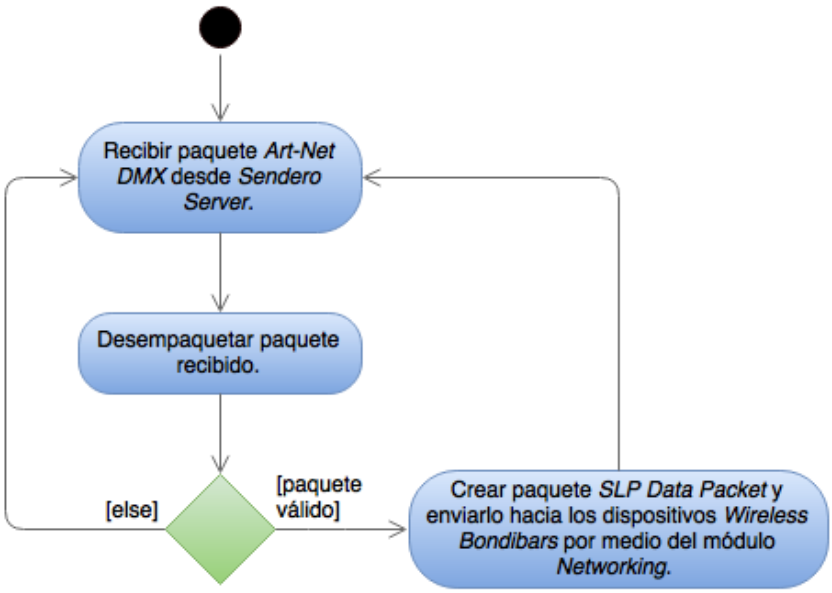


Figura 4-13 - Secuencia de ejecución del modo Art-Net del módulo Streaming

Los paquetes Art-Net DMX enviados por Sendero Server, son recibidos por el módulo *Streaming* de Sendero Middleware a través de un *socket* UDP en un puerto configurable. Cada paquete recibido es procesado y re-empaquetado en forma de un SLP Data Packet que será enviado hacia los dispositivos inalámbricos usando el módulo *Networking*.

Networking

El módulo *Networking* se encarga de transmitir paquetes del tipo SLP Data Packet hacia los dispositivos inalámbricos, respetando ciertas configuraciones referentes a la organización de la red *multicast*.

Con el fin de favorecer la escalabilidad de la solución, Sendero Middleware plantea la organización de la red inalámbrica en grupos *multicast*. Para ello, se debe especificar en el módulo *Configuration* la cantidad de grupos que se desea utilizar, desencadenando una distribución automática cuyo objetivo es obtener una repartición uniforme. Con esto, la información de cada *frame* de la obra se divide en tantos paquetes como grupos *multicast* se hayan configurado.

En el siguiente ejemplo, se propone una obra con 1000 píxeles controlados por 125 Wireless Bondibar, que manejan 8 píxeles cada una y se organizan como indica la Tabla 4.1.

Wireless Bondibar ID	Grupo	Cantidad de píxeles por paquete
0 - 41	224.0.0.116	336
42 - 83	224.0.0.117	336
84 - 124	224.0.0.118	328

Tabla 4.1 - Partición automática de grupos *multicast*

Nota: las direcciones *multicast* utilizadas pertenecen el rango sin asignar 224.0.0.116-224.0.0.250 del espacio de direccionamiento *multicast* para redes locales [106].

En este contexto, Sendero Middleware recibe de Sendero Server la información de 1000 píxeles (3 kilobytes de datos de color) por cada *frame*. Estos datos son repartidos entre los 3 grupos definidos siguiendo la estructura establecida en la Tabla , es decir, los primeros 1008 bytes correspondientes a los primeros 336 píxeles serán enviados con la IP destino 224.0.0.116, los bytes 1009 al 2016 se encapsulan en otro paquete SLP Data Packet con destino a la IP 224.0.0.117, mientras que los restantes bytes se agrupan en un tercer paquete que se envía con IP destino 224.0.0.118. Este comportamiento es representado en la Figura 4-14.

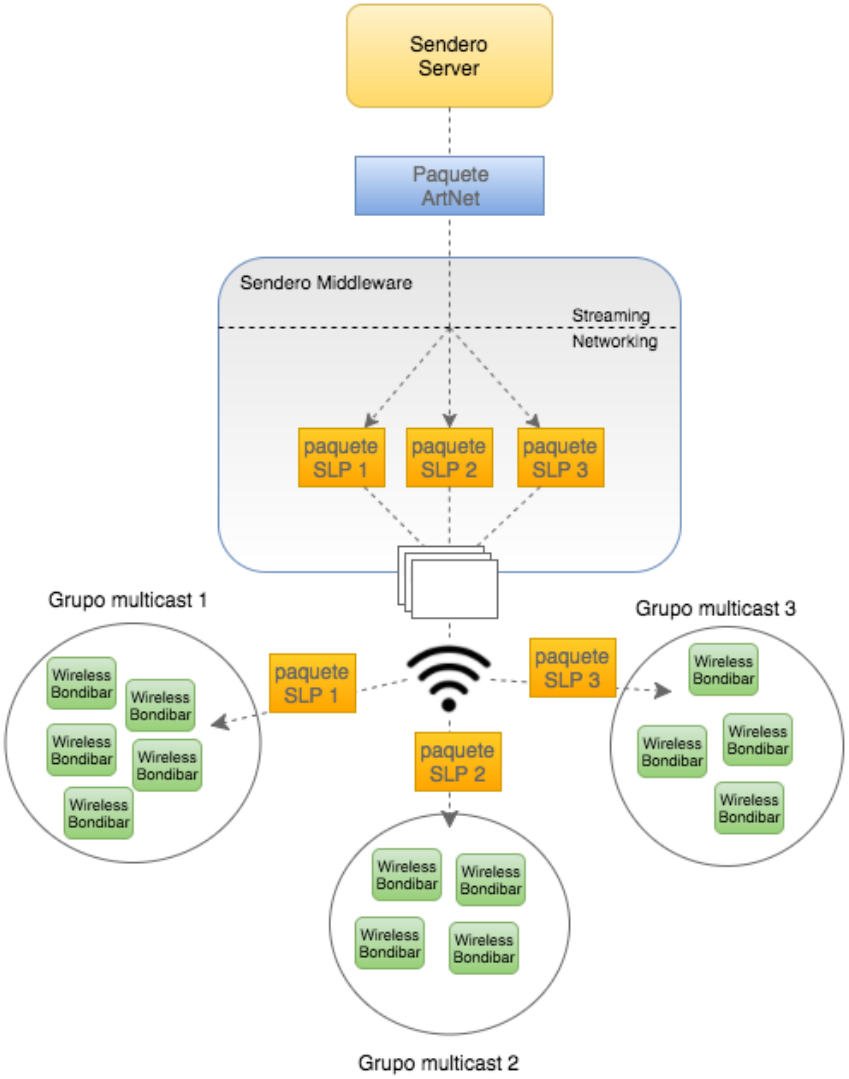


Figura 4-14 - Flujo de datos en red local de Sendero

Configuration

Finalmente, el módulo *Configuration* es el responsable de administrar las variables necesarias para la configuración de los módulos inalámbricos, la estructura de la obra y el control del flujo de datos.

La Tabla 4.2 resume las principales configuraciones, de las cuales pueden encontrarse detalles en la documentación de la distribución *open-source* de Sendero Middleware.

Categoría	Variables
Estructurales de la obra	Cantidad total de píxeles de la obra.
	Identificadores de los dispositivos inalámbricos habilitados.
	Cantidad y posición dentro del paquete de datos de los píxeles que maneja cada Wireless Bondibar.
	Ordenamiento de los bytes de información (RGB, BGR, GRB)
Módulos inalámbricos	Intervalo de solicitud de estadísticas.
	Intervalo de mensajes *Keep Alive*.
Flujo de datos	Puerto en el que se reciben los paquetes Art-Net DMX enviados por Sendero Server.
	Cantidad de grupos *multicast*.
	Dirección IP base de la red *multicast*.
	Retardo de reproducción fijo.

Tabla 4.2 - Principales configuraciones de Sendero Middleware

Sendero Wireless Bondibar

La principal funcionalidad de este módulo es implementar el protocolo Sendero Lighting Protocol (SLP) y a través de éste oficiar de controlador inalámbrico de las luces LED RGB. Para ello se desarrolló un *firmware* capaz de implementar las funcionalidades requeridas por el protocolo y a su vez traducir la información de color de los píxeles recibidos en los paquetes de datos que se envían hacia los controladores de LEDs que éste maneja.

Concretamente, el *firmware* fue implementado utilizando el lenguaje de programación C++ bajo el *framework* de Arduino [100] adaptado para el microcontrolador *ESP8266.*

En la Figura 4-15 se presenta un diagrama arquitectónico de su implementación, en donde se ilustran los principales módulos que lo componen.

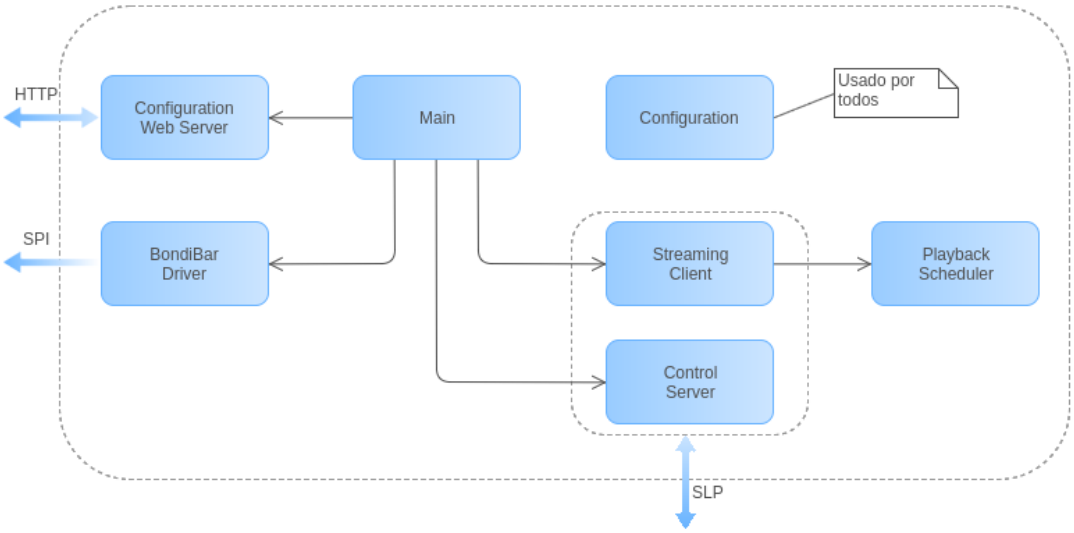


Figura 4-15 - Diagrama arquitectónico del firmware desarrollado.

Módulos

Main

Este módulo representa el componente de más alto nivel dentro del *firmware*. Su principal funcionalidad es la de inicializar y orquestar la utilización de los restantes módulos. Debido a la limitante de contar con un único *thread* de ejecución, las interfaces que ejecuta son implementadas con funciones no bloqueantes.

En la Figura 4-16 se presenta un diagrama de flujo, que describe el comportamiento implementado en este módulo (el cual, en definitiva, determina el comportamiento general del *firmware*).

Este comportamiento puede ser clasificado en dos grandes etapas: **inicialización** y **loop principal**. Por un lado, la etapa de **inicialización** se compone de las siguientes tareas (en el orden dado):

1. Inicializar *Configuration*: se inicializan las configuraciones por defecto del dispositivo, tales como tamaños de *buffers*, identificador de dispositivo, estadísticas, etc.
2. Inicializar *Bondibar Driver*: se configuran los puertos y pines del módulo inalámbrico con el objetivo de poder controlar, a través del protocolo SPI, los controladores Bondibar conectados.
3. Inicializar *Configuration Web Server:* se brinda acceso al módulo *Configuration* a través de una interfaz web HTTP.
4. Inicializar *Wi-Fi Manager*: establece una conexión Wi-Fi con el punto de acceso utilizando las credenciales (*ssid* y contraseña) almacenadas en el módulo *Configuration*.
5. Inicializar *Control Server*: se anuncia la existencia del dispositivo utilizando el protocolo SLP y se establece una conexión de control SLP con el componente Sendero Middleware. Dicha conexión, establece configuraciones adicionales tales

como la cantidad de píxeles total de la obra, cuáles de ellos son controlados por el dispositivo en cuestión, etc.

6. Inicializar *Streaming Client*: el dispositivo se une al grupo *multicast* configurado y se inicializa el puerto UDP a través del cual se recibirá el flujo de datos SLP.
7. Inicializar Playback Scheduler se inicializan variables de estado para la gestión de la reproducción.

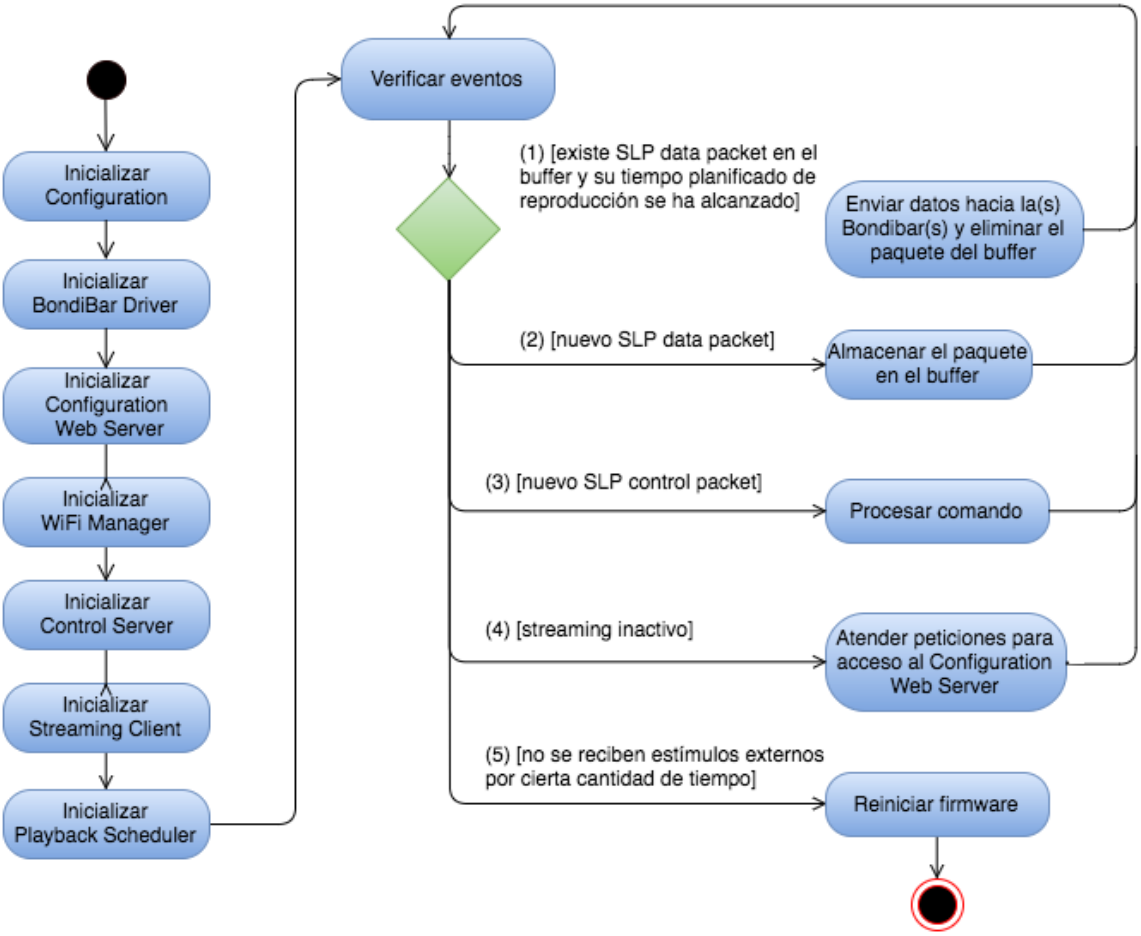


Figura 4-16 - Diagrama de flujo del módulo Main.

Luego de realizadas estas tareas, el dispositivo queda configurado y listo para ser partícipe de una sesión de reproducción. Es aquí donde comienza la segunda etapa denominada **loop principal**. En ella, el dispositivo permanece en un bucle infinito atendiendo los eventos enumerados en la Figura 4-16, respetando la prioridad definida por los números indicados (números más pequeños representan un mayor nivel de prioridad).

Configuration

Este módulo es el encargado de almacenar todas las variables de configuración y estadísticas del dispositivo. El mismo provee los parámetros necesarios para el correcto funcionamiento de cada uno de los restantes módulos, y debido a tal responsabilidad, durante su implementación se hizo hincapié en los siguientes aspectos:

1. **Accesibilidad.** Todos los módulos tienen una dependencia directa hacia el módulo *Configuration* y adicionalmente el mismo es accedido a través de las entidades externas Sendero Middleware (a través del canal de control) y clientes web (a través del módulo *Configuration Web Server)*. Por esta razón la interfaz de acceso al mismo fue diseñada de forma tal que, los componentes de *software* internos al *firmware* puedan acceder a las variables de manera estática (esto es, en tiempo de compilación), mientras que las entidades externas puedan direccionar las mismas a través de un *tag* identificador (como por ejemplo "*Device.number*", "*Streaming.port*", "*Stats.packetLossRate*", etc) y potencialmente proporcionar un valor del tipo adecuado.
2. **Escalabilidad.** Durante el desarrollo del *firmware* se fueron constantemente quitando y agregando variables de configuración al sistema debido al crecimiento de sus funcionalidades y al proceso de desarrollo en sí. Es por ello, que se implementó de forma tal que agregar nuevas variables de configuración sea una tarea sencilla.
3. **Abstracción.** Este módulo provee dos tipos de variables, el tipo *Variable* que representa una variable (y su tipo asociado) y el tipo *Persistent Variable* que extiende el concepto anterior, haciendo que la variable sea a su vez persistente y se mantenga almacenada en la memoria EEPROM disponible en el módulo inalámbrico.

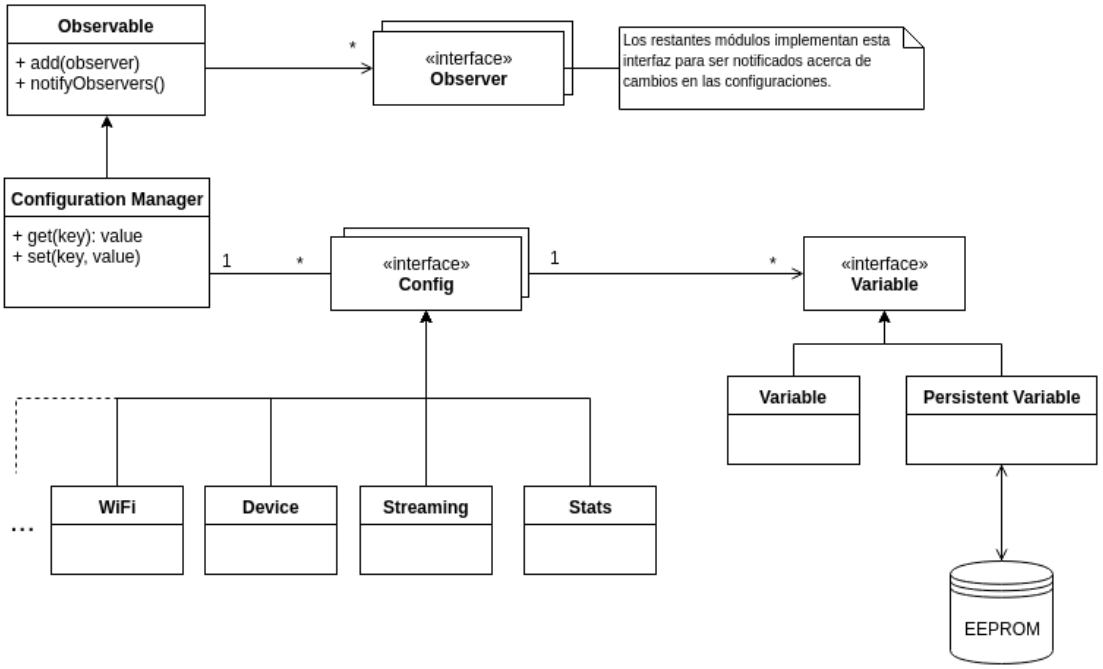


Figura 4-17 - Diagrama de clases del módulo *Configuration*.

La Figura 4-17 presenta un diagrama estructural del módulo implementado. El principal punto de acceso al mismo es la clase *Configuration Manager*, la cual puede ser vista como un gran diccionario de variables, que a su vez permite recibir notificaciones cuando

éstas cambian su estado. Por ejemplo, cuando un cliente web se conecta al *Configuration Web Server* y modifica las configuraciones actuales del dispositivo, todos los módulos del *firmware* (interesados) recibirán la correspondiente notificación y serán reconfigurados en el mismo orden en el que fueron inicializados. Para ello, se utilizó el patrón de diseño *Observer* [107].

Bondibar Driver

Este módulo es el encargado de realizar la comunicación con los controladores de LEDs que el módulo inalámbrico tiene conectados. Como fue mencionado anteriormente, esta comunicación se realiza utilizando el protocolo de *hardware* SPI. Para ello, el *driver* implementado inicializa los puertos y pines SPI correspondientes para oficiar el rol de maestro SPI, es decir, configura el *hardware* SPI contenido en el módulo ESP8266 para utilizar el pin GPIO14 como salida de *clock* SPI y el pin GPIO13 como MOSI (*master output - slave input*) o en otras palabras la salida de datos.

Dado el diseño de la solución, fue posible abstraer el *driver* de la cantidad de controladores de LEDs conectados. Básicamente los elementos determinantes para el funcionamiento del *driver* son los pines por los cuáles se hablará SPI y la cantidad de bytes a ser enviados (determinado por la cantidad de píxeles controlados).

Configuration Web Server

Con el objetivo de poder acceder a las configuraciones presentes en el *firmware* se decidió exponer las mismas mediante una interfaz web. Dicha interfaz permite la configuración de todos los módulos del *firmware* del mismo modo en que lo hace Sendero Middleware, pero de forma directa.

El módulo ESP-12E ofrece dos modos de operación, no excluyentes entre sí: modo **AP** (*access point*) y modo **STATION**. El modo AP hace posible que el dispositivo sea visible como *Access Point* en una WLAN de manera análoga a los *routers* Wi-Fi. Esto resulta especialmente útil para cambiar las credenciales Wi-Fi (*ssid* y contraseña) sin la necesidad de grabar nuevamente el *firmware*. Por otra parte, el módulo ingresa en modo STATION una vez que se establece la conexión Wi-Fi con las credenciales almacenadas en el módulo *Configuration*.

Control Server

Como fue mencionado anteriormente, uno de los principales objetivos del componente Wireless Bondibar es implementar el protocolo SLP y es este módulo el encargado de implementar el canal de control y el proceso de descubrimiento de nuevos emisores.

El proceso de descubrimiento del dispositivo es llevado a cabo durante la inicialización del módulo y consiste en anunciar su existencia a través del envío de paquetes del tipo SLP Discovery Packet, cada cierto intervalo de tiempo, hasta recibir una conexión proveniente del componente Sendero Middleware. Una vez realizada esta conexión, el canal

de control SLP queda establecido, y a partir de allí, Sendero Middleware enviará un paquete de control con la *flag SS* (*Set Configuration*) activa, conteniendo las configuraciones de la obra en cuestión. Luego de la correcta recepción del mismo y la correspondiente actualización de configuraciones, el componente Wireless Bondibar se encuentra en condiciones de continuar la etapa de inicialización de módulos. La Figura 4-18 muestra un esquema de este proceso.

Luego del proceso de inicialización, el canal de comunicación queda establecido durante el restante ciclo de vida del *firmware*. Cabe aclarar que la conexión TCP utilizada para implementar el canal de control, no necesariamente permanece activa durante toda la sesión del dispositivo. Ésta puede ser terminada y reiniciada a demanda, según las *flags* enviadas en los paquetes de control.

Adicionalmente, el canal de control es utilizado para la obtención de estadísticas y métricas recolectadas por los dispositivos durante las sesiones en las que participa. Esto cumplió un rol fundamental para la correcta calibración y estudio del desempeño del prototipo desarrollado (ver Capítulo 5).

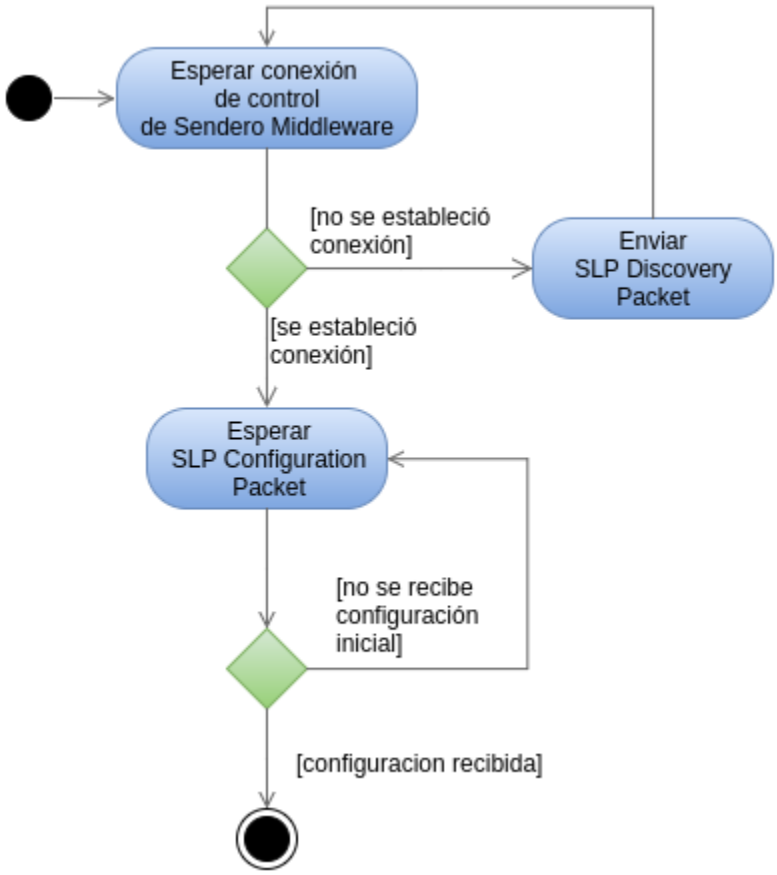


Figura 4-18 - Proceso de inicialización del firmware.

Streaming Client

El módulo *Streaming Client* es el encargado de implementar el rol de receptor en el canal de datos del protocolo SLP y, consecuentemente, es considerado uno de los módulos más importantes. Su principal responsabilidad es aplicar técnicas de *streaming* para garantizar la correcta reproducción de los paquetes SLP Data Packet.

Como fue mencionado anteriormente, el esquema de *streaming* de datos diseñado entre los componentes Sendero Middleware y Wireless Bondibar supone una planificación con retardo de reproducción fijo y determinado por el componente emisor del flujo, en este caso Sendero Middleware. En otras palabras, el emisor adiciona un tiempo de retraso fijo (por ejemplo 200 ms) al *timestamp* en el que cada paquete es generado y obtiene así, el tiempo planificado de reproducción del paquete en cuestión. Dicho paradigma asume que la latencia de la red en la que se lo utiliza se encuentra acotada, y dado que su implantación en este proyecto es sobre una red WLAN de un *hop*, su utilización se encuentra justificada.

La implementación realizada implementa dos técnicas utilizadas comúnmente en transferencia de video sobre redes de datos: *buffering* y recuperación de errores.

Buffering

Como fue relevado en el Capítulo 2, los principales objetivos al aplicar técnicas de *buffering* son: la absorción de los retardos variables introducidos por la red en la cual se transmiten los datos (o dicho de otra forma, la absorción del *jitter)* y los retardos de procesamiento. Para dicha tarea, se cuenta con un *buffer* en el cual permanecen almacenados los paquetes que han sido correctamente recibidos y su tiempo de reproducción aún no ha sido alcanzado.

Debido a que el valor del campo *playbackTime* (contenido en cada SLP Data Packet recibido) es obtenido a partir del reloj de referencia utilizado en el emisor, es necesario trasladar ese sistema de referencia al reloj local en el receptor (ya que no puede asumirse su sincronismo). Esta responsabilidad se delegó al módulo *Playback Scheduler* y será explicado en detalle más adelante. A efectos del funcionamiento de éste módulo, se asumirá que existe un *offset* conocido entre ambos relojes.

Bajo dichos lineamientos, un paquete puede tener asociado uno de los siguientes estados:

1. **Recibido.** El paquete ha sido recibido correctamente y se encuentra listo para ser almacenado en el *buffer* de reproducción. Es aquí cuando el *timestamp* contenido en el paquete de datos es utilizado por el módulo *Playback Scheduler* para actualizar, en caso de ser necesario, el *offset* entre los relojes. Por otro lado, debido a que los paquetes de datos viajan a través de una WLAN de un *hop*, no existe la posibilidad de sean entregados en desorden. Finalmente, el paquete es insertado en el *buffer*.
2. **Almacenado en el *buffer*.** En este estado, el paquete permanecerá almacenado en el *buffer* hasta que su tiempo planificado de reproducción sea alcanzado. Debi-

do a que la precisión utilizada para representar tiempos es de milisegundos, al momento de verificar si un paquete debe ser reproducido, se compara el tiempo actual con el planificado utilizando un cierto margen de tolerancia δ (1 ms en la implementación actual). Es decir, dado un tiempo planificado de reproducción $\boldsymbol{pt}$ y el tiempo actual $\boldsymbol{t}$ (al momento de verificar la condición):

1. $|pt - t| < \delta$ determina que el paquete debe ser reproducido. En este caso el paquete es considerado **Reproducido a tiempo**.
2. $t - pt >= \delta$ determina que el tiempo de reproducción planificado para el paquete ha sido superado, incluso más allá del margen de tolerancia δ. En este caso, el paquete es considerado **Retrasado**.

Dicho margen de tolerancia resulta conveniente y necesario debido a la restricción de contar con un único *thread* de procesamiento. El *firmware* no puede asegurar la reproducción de un *frame* al instante exacto al que fue planificado.

3. **Reproducido a tiempo.** Aquí el paquete es removido del *buffer* y enviado al módulo *Bondibar Driver* para ser transmitido hacia los controladores de LEDs.
4. **Retrasado.** Al igual que el caso anterior, el paquete es removido del buffer pero esta vez no es reproducido ya que se lo considera retrasado.

En la Figura 4-19 se muestra un diagrama de estados de un paquete de datos SLP.

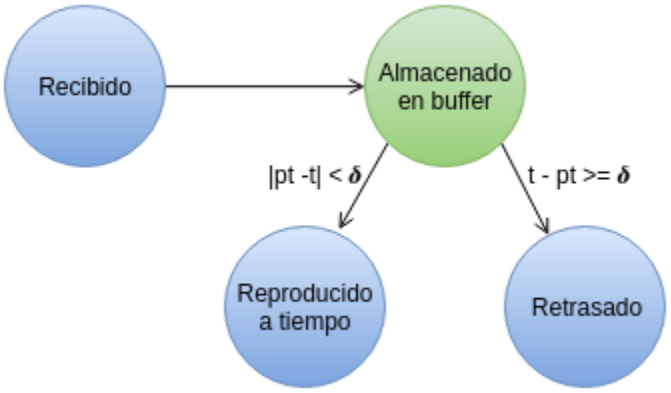


Figura 4-19 - Diagrama de estados de un paquete de datos SLP.

Recuperación de errores

El protocolo de capa de transporte utilizado para la transmisión de los paquetes SLP es, como ya se mencionó, UDP. Este protocolo permite la validación de la integridad de los datos a partir de la utilización de un *checksum*. En este contexto, el único caso de error considerado es la pérdida de paquetes, y para lidiar con este fenómeno se implementó un mecanismo de recuperación frente a pérdidas similar al que será descrito más adelante para la interfaz web Sendero Web.

Playback Scheduler

Como fue mencionado anteriormente, este módulo es el encargado de realizar la planificación de la reproducción de los paquetes recibidos. Para ello, es necesario realizar una traslación del tiempo de reproducción contenido en el paquete SLP Data Packet, desde el

reloj de referencia del emisor hacia el reloj de referencia local en el receptor. Dicho de otra forma, se determina cuánto tiempo permanecerá cada paquete en el *buffer*.

Para ello, dado un paquete recibido n, se define:

- $T(n)$: *timestamp* contenido en el paquete SLP Data Packet expresado en referencia al reloj del emisor.
- $R(n)$: *timestamp* de recepción del paquete expresado en referencia al reloj local.
- $D(n) = T(n) - R(n)$: *offset* (con signo) relativo entre los valores definidos anteriormente.

La diferencia $D(n)$ incluye los siguientes factores:

1. Retardo constante debido a la diferencia de tiempo entre los relojes.
2. Retardo variable debido al procesamiento y preparación de datos en el nodo emisor y receptor.
3. Retardo constante debido al mínimo tiempo de transmisión en la red.
4. Retardo variable debido al *jitter* en la red.
5. *Clock skew* (desviación entre relojes debido a diferencias de *hardware* e imperfecciones o diferencias en su construcción).

Esta diferencia es calculada en cada paquete, y el receptor lleva registro del mínimo (en valor absoluto) observado, denominado *offset*. Luego, el tiempo base de reproducción es calculado como sigue:

$$basePlayoutTime(n) = R(n) + offset$$

Conceptualmente, $basePlayoutTime$ representa el tiempo de reproducción más cercano posible (a partir del tiempo de recepción) para los nodos receptores. Dicho de otra forma, contempla el tiempo planificado de reproducción donde los factores variables en *D(n)* son mínimos. Luego, para absorber estos retardos variables, se adiciona un retardo fijo denominado *playbackDelay*. Por lo tanto, el tiempo planificado de reproducción (pt) para el paquete n se obtiene a partir de la siguiente fórmula:

$$pt(n) = basePlayoutTime(n) + playbackDelay$$

Un detalle omitido hasta aquí es el cálculo del *offset*. Como fue expresado anteriormente, el *offset* es definido como el mínimo *D(n)* observado, pero dicha definición resulta algo idealista principalmente por las posibles desviaciones entre relojes. Para mitigar este problema, *RTP*, por ejemplo, calcula este *offset* como el mínimo de una cierta cantidad de los últimos *D(n)* observados. No obstante, debido a la capacidad de procesamiento limitada disponible en los dispositivos, se optó por implementar ciclos de expiración de este *offset* (por ser computacionalmente menos costoso). Básicamente el cálculo del *offset* se compone de las siguientes tareas:

1. Se mantiene una cota mínima *CM*, de forma tal que si el valor *D(n)* observado es menor, se actualiza *CM* y *offset* con dicho valor.

2. En cada ciclo de expiración se incrementa *CM* utilizando una función exponencial con exponente *q*, siendo *q* la cantidad de períodos de expiración ocurridos desde la última actualización del *offset*. Cabe aclarar que, de no utilizar una función exponencial aquí, podría suceder que la desviación de uno de los relojes sea más rápida que el incremento de *CM* y por lo tanto el *offset* nunca sería actualizado nuevamente.

En la Figura 4-20 se muestra un esquema de este comportamiento. Nota: si *offset* no es dibujado, se asume que es igual a CM.

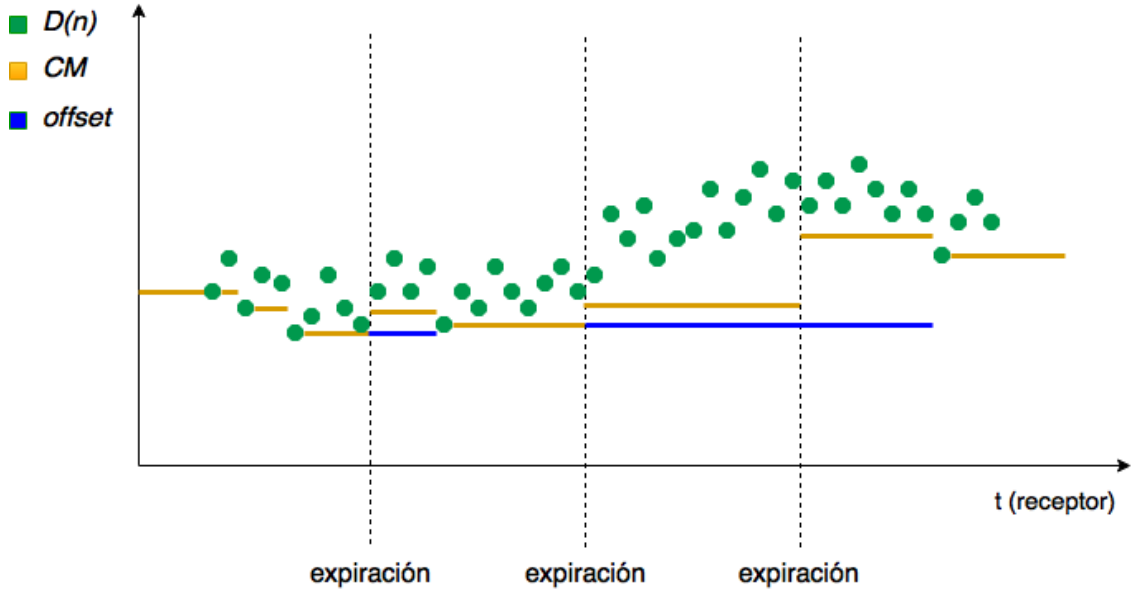


Figura 4-20 - Esquema de Expiración de offset.

Nota: si *offset* no es dibujado, se asume que es igual a CM.

Período de calibración

Un fenómeno detectado durante la implementación de este módulo es que cuando se comienza la recepción de paquetes del tipo SLP Data Packet, los *timestamps* de recepción *R(n)* contienen valores inadecuados o erróneos (no correspondientes con la realidad) durante cierta cantidad de tiempo inicial, provocando cálculos erróneos del tiempo de reproducción de paquetes. Para mitigar este comportamiento, se diseñó un período de calibración en el cual se reciben y descartan paquetes hasta normalizar la situación.

Básicamente, se mantiene un promedio acumulativo (al cual denominaremos *CA*) a partir de los *D(n)* calculados. Sea *CA(n)* el promedio acumulativo calculado hasta el paquete n-ésimo recibido, y calculado mediante la siguiente fórmula:

$$CA(n) = \left(\frac{n-1}{n}\right) \times CA(n-1) + \frac{D(n)}{n}$$

Notar que su definición recursiva hace posible su implementación de manera eficiente, lo cual es un requerimiento presente en toda la implementación del *firmware*. Finalmente, con el objetivo de detectar la normalización de los valores *D(n)* se espera que la derivada del promedio acumulativo *CA(n)* esté por debajo de cierto umbral por determinada cantidad de tiempo.

Sincronismo entre receptores

Como fue mencionado anteriormente, uno de los principales objetivos del protocolo SLP es permitir la sincronización entre los nodos receptores, es decir, que el paquete SLP Data Packet n-ésimo sea reproducido por todos los receptores en aproximadamente el mismo instante. Este objetivo se logra prácticamente como efecto secundario de la utilización del modo *multicast* UDP para su envío, ya que una única emisión por parte de Sendero Middleware basta para que todos los nodos Wireless Bondibar reciban ese mismo paquete en el mismo instante (a menos de diferencias de procesamiento particulares en cada nodo). Respecto al caso en que Sendero Middleware realiza más de una emisión por *frame* de la obra (esto es, cuando se tiene más de un grupo *multicast* definido), la sincronización entre estos grupos se logra mediante la alteración de los retardos fijos de reproducción configurados para cada grupo *multicast.*

Sendero Streaming Server

Sendero Streaming Server tiene como tarea principal recibir *frames* emitidos por Sendero Server y distribuirlos a todos los clientes remotos conectados. Los datos son recibidos mediante UDP, empleando el protocolo SWP. Cada paquete recibido es procesado y retransmitido por medio de WebSockets a cada cliente web conectado.

El componente se desarrolló en Node.js [108], y se utiliza Socket.IO [109] como implementación de la tecnología de WebSockets. Actualmente corre en una máquina virtual de Amazon, específicamente en una instancia t2.micro [110] situada en Oregon, Estados Unidos. La misma cuenta con 1 CPU Intel Xeon de 3.3 GHz y 1 GiB de memoria RAM. Se espera poder migrar hacia algún servidor situado en Uruguay para disminuir la latencia de comunicación (~230ms) con los clientes remotos y mejorar la interactividad.

Sendero Web Server

Sendero Web Server es un servidor web cuya finalidad es servir las aplicaciones web que ofician como clientes remotos, Sendero Web y Sendero Cardboard, y la interfaz administrativa Sendero Dashboard. Para ello se adquirió el dominio **sendero.uy** y se configuraron los sub-dominios **web.sendero.uy**, **vr.sendero.uy.**

Su implementación se reduce a la utilización del servidor *web/proxy* inverso *Nginx* [111], como multiplexor de contenidos estáticos. Para ello, este servidor escucha en el puerto 80

por peticiones HTTP, que sirve de manera diferente según los sub-dominios mencionados.

Luego, cada aplicación establece conexiones con Sendero Streaming Server y/o Sendero Interaction Server según corresponda. La Figura 4-21 muestra un esquema que resume el funcionamiento de este componente, alojado físicamente en el mismo servidor que Sendero Streaming Server.

Las aplicaciones son descritas más adelante en el apartado Sendero Web App.

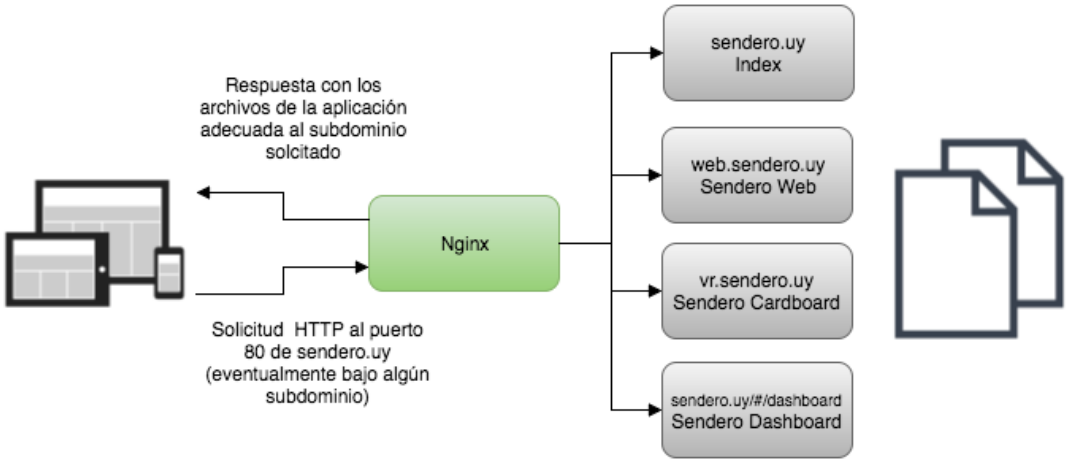


Figura 4-21 - Esquema de funcionamiento de Sendero Web Server.

Sendero Interaction Server

Sendero Interaction Server es un servidor Web que, mediante el uso de WebSockets [112], recibe interacciones remotas realizadas por los clientes web de Sendero mediante el uso de la aplicación Sendero Web (ver Sendero Web más adelante), para luego enviarlas a una instancia de Sendero Client especializada empleando el protocolo de mensajes AMQP [113]. Para su desarrollo se utilizó Node.js y la librería Socket.io para el uso de *WebSockets*. Como implementación del estándar AMQP se utiliza RabbitMQ [114].

Los usuarios remotos pueden interactuar virtualmente con Sendero por medio de la aplicación Sendero Web. Estas interacciones se envían a Sendero Interaction Server utilizando WebSockets. Cuando el servidor recibe esta información, decodifica el tipo de interacción y publica su contenido, junto con un identificador único del cliente web, en la cola RabbitMQ asociada a esa interacción. Este mapeo entre el tipo de la interacción y la cola a la que se debe enviar, se configura en un archivo de configuración del servidor de interacción que sirve para este propósito. Cada cola mantiene una conexión con una única instancia de Sendero Client que recibe la información de las interacciones para crear los patrones de iluminación, generando *frames* que son enviados a Sendero Server. La Figura 4-22 muestra el ciclo descrito.

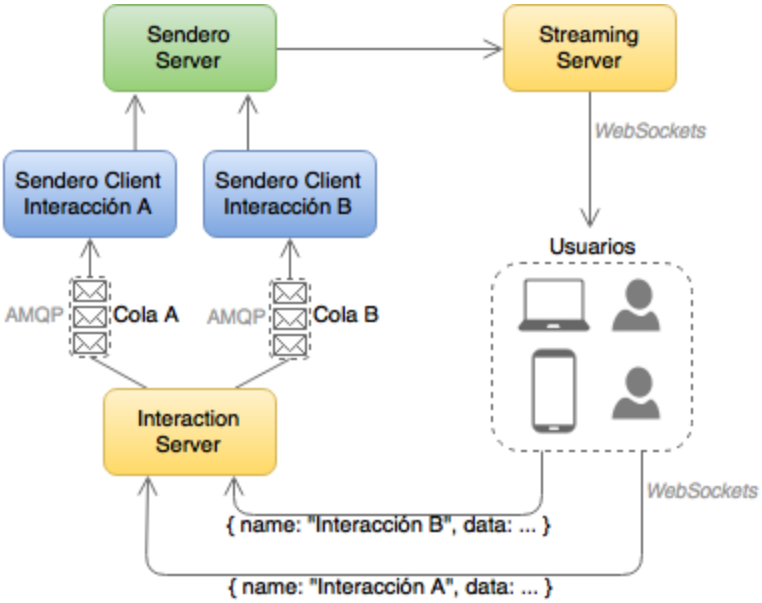


Figura 4-22 - Esquema de flujo de interacciones remotas.

Sendero Web App envía los datos de la interacción hacia Sendero Interaction Server en la forma de un JavaScript *Object* que debe tener dos claves: *name* y *data*. El valor de la primera es un *string* con el nombre identificador de la interacción, que sirve para hacer el mapeo de la cola a la que se deben publicar los datos de la interacción. Asociado a la segunda clave, viaja la carga útil de la interacción y quien debe ser capaz de interpretarlo es la implementación Sendero Client que lo recibe.

Dada esta arquitectura de implementación, para desarrollar una nueva interacción se deben realizar los siguientes pasos:

1. En la aplicación Sendero Web, generar datos y enviarlos en el formato adecuado junto con un nombre identificador de interacción hacia Sendero Interaction Server.
2. Configurar la interacción en el servidor, agregando su nombre y el nombre de la cola RabbitMQ en la que se deben añadir sus datos. Esto se realiza en el archivo de configuración.
3. Desarrollar una implementación de Sendero Client que consuma de la cola RabbitMQ configurada en el paso dos y genere *frames* que luego envía a Sendero Server.

Sendero Web App

Sendero Web App hace referencia a un conjunto de aplicaciones web, desarrolladas con el objetivo de permitir a usuarios remotos visualizar en tiempo real una obra desarrollada con Sendero, al mismo tiempo que es posible interactuar con la misma. Las distintas aplicaciones que constituyen a Sendero Web App se relacionan con Sendero Streaming Server y Sendero Interaction Server del modo explicado en la secciones anteriores.

A continuación se describen las aplicaciones que integran Sendero Web App y sus componentes.

Sendero Web

Sendero Web es una aplicación web compuesta por una serie de módulos que le permiten:

- Recibir la configuración de la obra y generar una representación tridimensional de la misma, accesible a través de un sitio web optimizado para el uso de dispositivos móviles (http://web.sendero.uy).
- Recibir un flujo de datos desde Sendero Streaming Server empleando la tecnología de WebSockets.
- Reproducir esta información en la representación tridimensional de la obra, haciendo uso de las técnicas de *streaming* multimedia presentadas en el Capítulo 2 mediante la utilización del protocolo SWP.
- Interactuar en tiempo real con la obra. Esto es, comunicar de manera apropiada a Sendero Interaction Server sobre los eventos recibidos.
- Generar y enviar estadísticas sobre su funcionamiento.

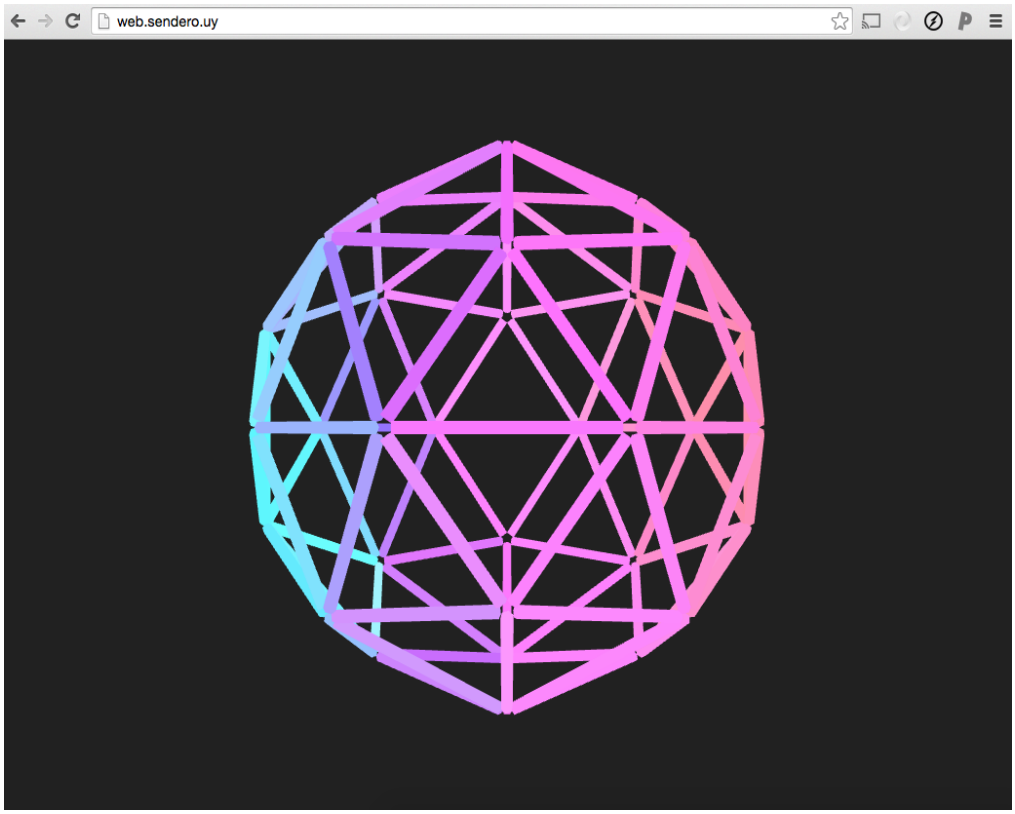


Figura 4-23 - Captura de pantalla de Sendero Web

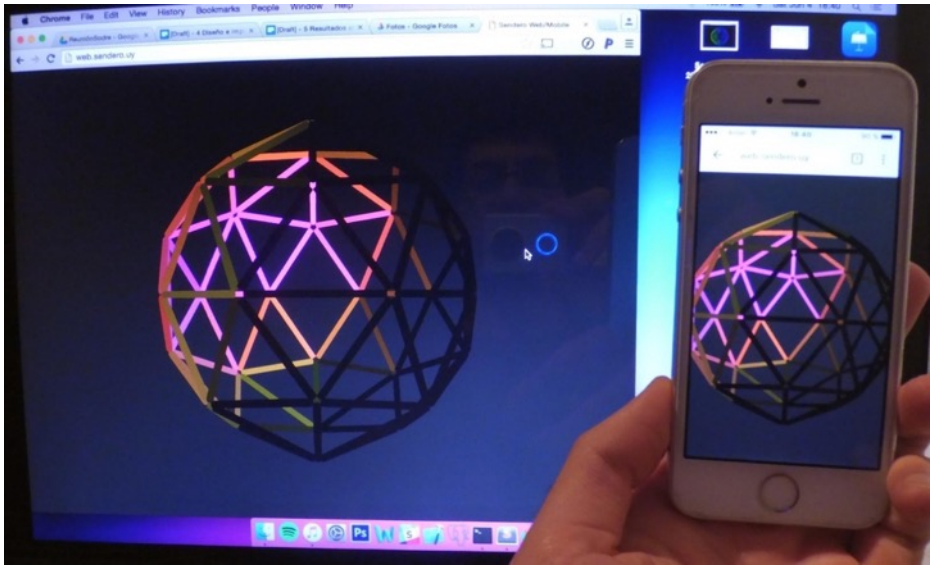


Figura 4-24 - Sendero Web desde dos navegadores

Los principales módulos de Sendero Web se encuentran desarrollados bajo el patrón Module [115] de JavaScript, y encapsulan la lógica referida a alguno de los siguientes aspectos: configuración (*config.js*), reproducción (*streaming.js*), interacción (*interaction.js*), dibujado (*renderer.js*) y estadísticas (*stats.js*).

Sendero Web tiene dos flujos principales. El de inicialización, donde se configura la escena y los parámetros de la sesión, y el de reproducción, encargado de recibir, procesar y reproducir los *frames* de la obra haciendo uso de las técnicas de *streaming* de video estudiadas.

Config

Módulo encargado de obtener los parámetros de configuración de la obra y la sesión, esto es:

- Dirección IP y puerto de Sendero Streaming Server.
- Dirección IP y puerto de Sendero Interaction Server.
- Cantidad de píxeles en la obra.
- Modelo tridimensional de los píxeles de la obra.
- Posición espacial de cada píxel.

Esta información llega a Sendero Web en un archivo con formato XML, siguiendo la estructura del archivo de configuración de Sendero Server (ver Anexo J). Una vez procesados estos parámetros quedan accesibles para que el resto de los módulos hagan uso de los mismos.

Renderer

Módulo construido sobre la librería Three.js [116], cuyo principal objetivo es brindar métodos que faciliten el dibujado y actualización de la obra virtual en el navegador.

Three.js es una librería *open-source* [117], *cross-browser* desarrollada sobre JavaScript, que utiliza WebGL [118] y ciertos elementos de HTML para el despliegue de gráficos 3D en navegadores.

El módulo *Renderer* tiene como objetivos principales:

1. Generar una escena virtual de la obra que permita su visualización desde distintas perspectivas, respetando la configuración de la obra obtenida por el módulo *Config*.
2. Actualizar la iluminación de los píxeles según la información de la obra recibida desde Sendero Streaming Server.

Streaming

Este módulo es el responsable de la reproducción de los *frames* recibidos desde Sendero Streaming Server.

Como fue descrito anteriormente, Sendero Server transmite los *frames* de la obra, utilizando UDP como protocolo de capa de transporte, hacía Sendero Streaming Server, quien al recibirlos los difunde a través de WebSockets hacía todas las instancias de Sendero Web y Sendero Cardboard que existan.

La forma en que las aplicaciones web reciben los datos desde Sendero Streaming Server respeta al protocolo SWP, por lo que cada paquete contiene un número de secuencia, una marca de tiempo y un conjunto de *flags* de configuración. Los campos de número de secuencia y marca de tiempo son utilizados para planificar la reproducción, e implementar un esquema de recuperación frente a pérdidas.

Retardo de reproducción (*buffering*)

Los *frames* recibidos son almacenados en un *buffer* hasta su reproducción. El momento en que serán reproducidos es calculado en base a una estimación del *jitter* de la red, empleando una política (simplificada) de retardo adaptativo basada fuertemente en el protocolo RTP, en particular, en la exposición brindada por C. Perkins en [31].

Dado un paquete n, su tiempo de reproducción planificado se calcula como sigue:

- **Mapeo del tiempo del emisor al tiempo local.** El tiempo del emisor (Sendero Server) es mapeado al tiempo local (Sendero Web/Cardboard), compensando el *offset* relativo entre sus relojes. Para calcular este *offset*, el receptor mantiene registro de las diferencias, $\mathbf{d(n)}$, entre el tiempo en el cual el n-ésimo paquete fue generado, $\mathbf{T_R(n)}$, y el tiempo en el cual fue recibido, $\mathbf{T_L(n)}$.

  La diferencia, $\mathbf{d(n)}$, incluye:
  - un factor constante debido a que los relojes pudieron ser inicializados en distintos momentos,

- un factor variable debido a la desviación entre los relojes ocasionada por imperfecciones de *hardware*,
- un *delay* variable debido al procesamiento en el emisor y receptor,
- un factor constante debido al mínimo tiempo de tránsito en la red,
- y otro *delay* variable debido al *jitter* de la red.

Esta diferencia es calculada en cada paquete, y el receptor lleva registro del mínimo observado en una ventana de tamaño fijo, denominado *offset*. Luego, el tiempo base (*base_playout_time*), es calculado como sigue:

$$base\_playout\_time(n) = T_R(n) + offset$$

Este cálculo, es la primera estimación del tiempo de reproducción planificado para el paquete n-ésimo. Y su objetivo es absorber el factor constante debido a las diferencias de los relojes, y la desviación entre los mismos (por lo que el offset se calcula sobre una ventana y no sobre todos los datos observados).

- **Compensación por *jitter* en la red.** Dado que el tiempo de arribo varía entre un paquete y otro, se necesita realizar alguna compensación que permita amortiguar estas variaciones. La forma de hacerlo es introduciendo un retardo adicional a la reproducción.

  Según lo expuesto por Perkins en [31], en la mayoría de los casos se puede asumir que el *jitter* de la red sigue una distribución Gaussiana, y por esto, si se consigue una estimación de la desviación estándar del *jitter*, se puede seleccionar un valor de retardo que asegure que la mayoría de los paquetes sean recibidos antes de que deban ser reproducidos. Esto se apoya en la teoría estadística de que el 99.7% de un conjunto de muestras normales cae en un rango de 3 desviaciones estándar de la media [119].

  Debido a esto, es posible estimar el *jitter* de la red [31] y compensar el *delay* variable introducido por el mismo.

- **Compensación por tiempos de procesamiento.** Finalmente, se añade un tiempo fijo para compensar el tiempo de procesamiento de los paquetes en el navegador. Este valor fue calculado mediante pruebas, de modo de afectar lo menos posible la interactividad de la aplicación intentando asegurar la correcta reproducción de los *frames*.

  Con el tiempo de reproducción planificado, los paquetes son almacenados en el buffer hasta que este tiempo es alcanzado. En ese momento, el paquete es retirado del buffer y utilizado para actualizar el estado de la obra virtual.

Compresión

El contenido de los *frames* puede llegar a las aplicaciones web comprimido, utilizando el algoritmo LZ4 [120].

Se decidió enviar en cada *frame* la información acerca de si su contenido fue comprimido o no para permitir mayor flexibilidad, puesto que de esta forma, el *streaming* puede variar el formato de codificación de su carga útil, sin alterar la reproducción.

Adicionalmente, la implementación de WebSockets utilizada (Socket.io 1.4.5) integra un mecanismo de compresión selectiva, que luego de una etapa de negociación de los parámetros de compresión, es capaz de determinar individualmente que paquetes serán comprimidos y cuales no en base a su contenido. En caso de aplicarse la compresión, su resultado se logra combinando el algoritmo LZ77 [121] con codificación de Huffman [122].

Recuperación ante pérdidas

Dado que una parte de la trayectoria de los *frames* sucede sobre el protocolo de capa de transporte UDP, no existen garantías de que todos los paquetes lleguen a destino. Por esto, se desarrolló un algoritmo de recuperación basada en el receptor, que realiza una interpolación de los paquetes anteriores y posteriores al que se quiere recuperar, generando una aproximación que disminuye el efecto visual de su pérdida.

Para ello, se utiliza un arreglo circular cuyo tamaño es igual a la cantidad de posibles números de secuencia. Cada vez que llega un nuevo *frame* con número de secuencia s y tiempo base de reproducción t_s, se introduce en el arreglo en la posición s.

Si al momento de recibir el *frame* con número de secuencia s (léase *frame* s) se detecta que existe en el arreglo un espacio vacío entre la posición s y la posición del *frame* recibido previo a s (posición r), se procede a recorrer las posiciones de dicho espacio, iterando desde r + 1 hasta s - 1, con el propósito de llenar estos espacios con *frames* estimados (ver Figura).

Asumiendo que la cantidad de paquetes perdidos es representada por la variable P, durante la iteración, en cada posición intermedia i se introduce un *frame* con número de secuencia i y tiempo base de reproducción t_i:

$$t_i = t_{i-1} + \frac{t_s - t_r}{p + 1}$$

De la misma forma que se interpola el tiempo base, se calcula la información de color RGB para cada uno de los píxeles de los *frames*:

$$R_i = R_{i-1} + \frac{R_s - R_r}{p+1}$$
$$G_i = G_{i-1} + \frac{G_s - G_r}{p+1}$$
$$B_i = B_{i-1} + \frac{B_s - B_r}{p+1}$$

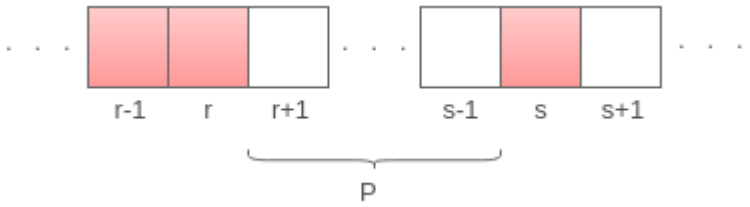


Figura 4-25 - Esquema de mecanismo de recuperación frente a pérdidas

En caso de que posteriormente se reciba alguno de los *frames* que fueron calculados, simplemente se lo reemplaza en la posición correspondiente del arreglo. Esto sucede siempre y cuando el tiempo de reproducción de dicho *frame* aún no haya sido alcanzado, pues de lo contrario se descarta.

Interaction

El módulo *Interaction* se encarga de capturar eventos del navegador y transmitirlos hacia Sendero Interaction Server como un objeto JavaScript cuyo contenido debe ser interpretado por alguna implementación de Sendero Client.

Las interacciones se encuentran fuertemente vinculadas a la obra, y es responsabilidad de quien la desarrolle, implementar interacciones deseadas.

Stats

Finalmente el módulo *Stats* se encarga de generar estadísticas sobre la reproducción del contenido y enviarlas periódicamente hacía Sendero Streaming Server donde son almacenadas en una base de datos no relacional MongoDB [123] y desplegadas por la aplicación Sendero Dashboard (descrita en el apartado Sendero Dashboard).

Las estadísticas que se registran son resumidas en la Tabla 4.3 y fueron basadas en las propuestas de C. Perkins, en *"RTP: Audio and Video for the Internet"* [31].

Grupo	Estadísticas
Paquetes	Número de paquetes recibidos
	Número de paquetes demorados
	Número de paquetes descartados
Jitter	Estimación actual del offset
	Estimación actual del *jitter*
Buffer	Promedio del tamaño del buffer
Reproducción	Promedio y Desviación estándar de los períodos de: reproducción, planificación, generación.

Tabla 4.3 - Estadísticas generadas en Sendero Web y Sendero Cardboard.

Sendero Cardboard

Sendero Cardboard es otra aplicación web, que se diferencia de Sendero Web solo en el módulo *Renderer*, y sus diferencias están enfocadas en permitir que esta aplicación funcione con el dispositivo de realidad virtual Google Cardboard [124], a través de un sitio web (http://vr.sendero.uy).

Para ello, se utiliza una librería de *Three.js* que permite la generación de imágenes estereoscópicas simulando la percepción de profundidad [125].

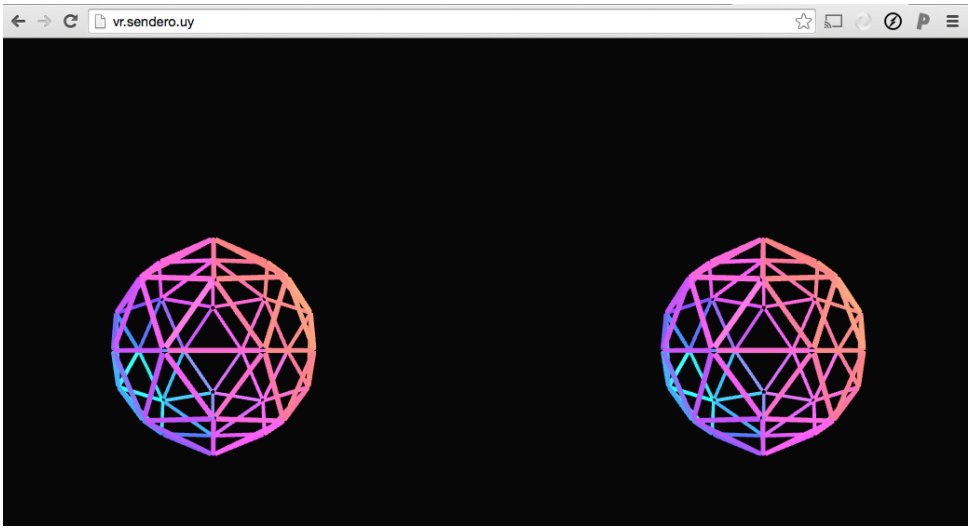


Figura 4-26 - Captura de pantalla de Sendero Cardboard en un navegador.

Esta aplicación fue desarrollada para la presentación de Sendero en el festival de ciencia y tecnología Equinoccio [126], llevado a cabo desde el 22 al 26 de septiembre de 2015 en el Instituto Nacional de Artes Escénicas [127] (ver Anexo K).

Sendero Dashboard

Finalmente, fue desarrollada una aplicación AngularJS [128]+ Socket.IO para visualizar en tiempo real el estado de la reproducción en base a las estadísticas generadas por las instancias de Sendero Web y Sendero Cardboard que se encuentren en ejecución.

Este *dashboard* resume la siguiente información:

- Cantidad de clientes conectados.
- Promedio y desviación estándar del período de muestreo de la generación de *frames* por Sendero Server (*ts*).
- Promedio y desviación estándar del período de muestreo del arribo de paquetes a las aplicaciones web (*arr*).
- Promedio y desviación estándar del período de muestreo de reproducción planificada (*pt*).
- Promedio y desviación estándar del período de muestreo de reproducción efectiva (*rpt*).
- Promedio del tamaño del buffer (*buff*).
- Un listado de las últimas estadísticas recibidas filtrable por ID de cliente.

Cada vez que Sendero Streaming Server recibe un nuevo conjunto de estadísticas, las almacena en una base de datos y emite a través de una conexión de WebSockets un resumen general a todas las instancias de Sendero Dashboard en ejecución.

La muestra una captura de pantalla de la aplicación durante un prueba con 20 clientes remotos.

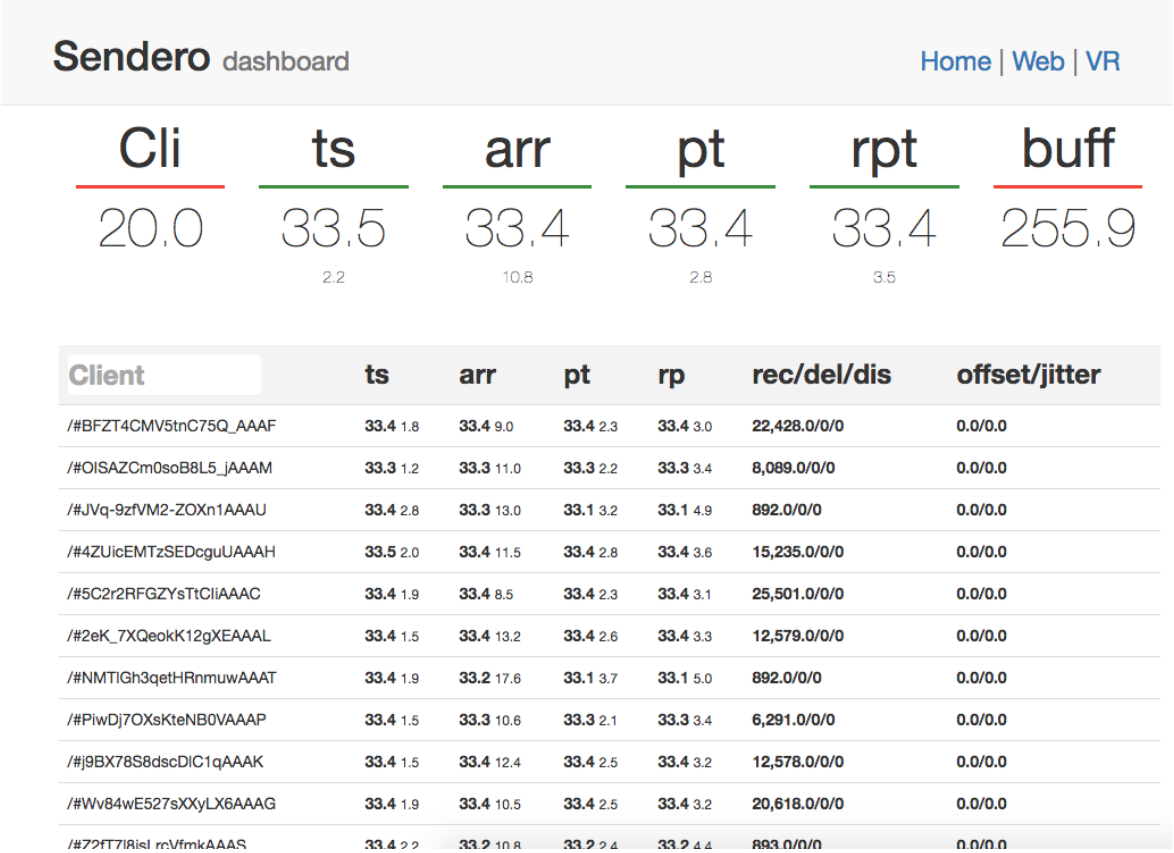


Figura 4-27 - Captura de pantalla de Sendero Dashboard.

4.4 Desarrollos complementarios

Adicionalmente a la implementación de la solución propuesta, se desarrollaron una serie de elementos complementarios con el fin de utilizar y evaluar el sistema, estos son, una instancia de Sendero Client controlado por el sensor de profundidad *Kinect* [129], una interacción remota que permite ver en acción al subsistema web de Sendero, y un dispositivo basado en la plataforma ESP-12E que permite medir objetivamente el nivel de sincronismo entre los receptores inalámbricos, denominado *Sync Meter*.

4.4.1 Sendero Client: Kinect

Se implementó un cliente que permite a un espectador presencial controlar el conjunto de píxeles de la obra Barcelona a través del movimiento de sus manos. La Figura 4-28 muestra la interacción desarrollada a modo de demostración ya que no se contó con la estructura real al momento su implementación.

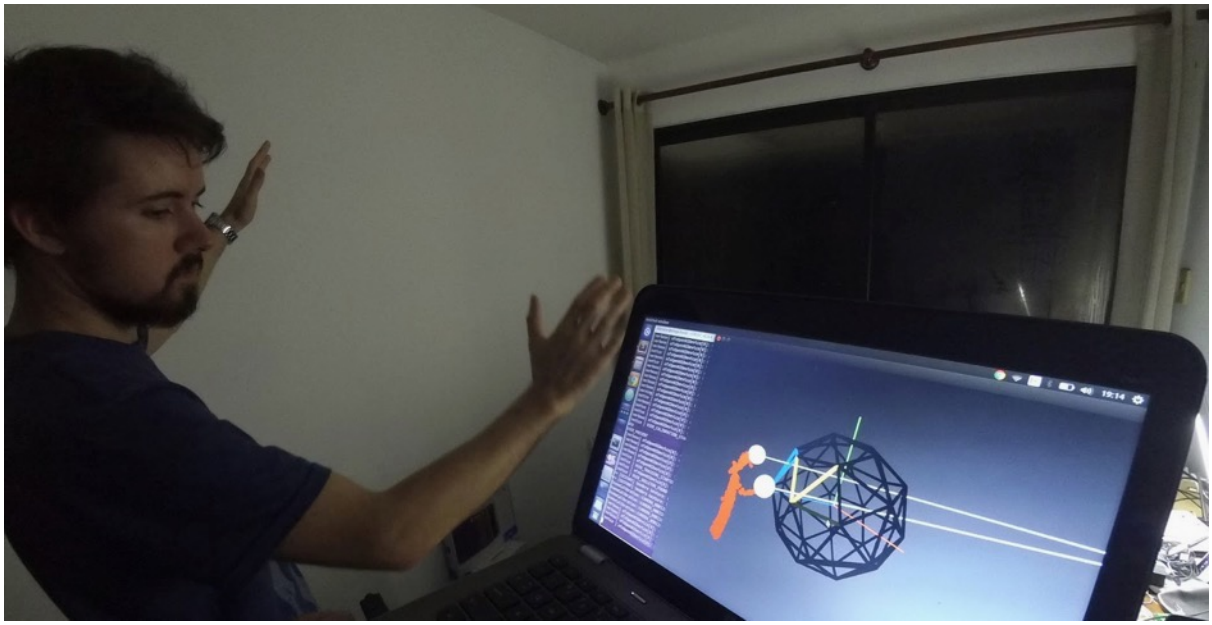


Figura 4-28 - Sendero Client con sensor de profundidad Kinect.

Esta interacción consta de las siguientes etapas:

1. **Detección de usuario.** En esta etapa el cliente se encuentra en un estado de búsqueda de un nuevo usuario. Para la detección del mismo, éste debe acercarse a la estructura de la obra y permanecer parado (a una distancia aproximada de 3 metros). Una vez que el sistema detecta su presencia, informa al usuario a través de un cambio en la intensidad de color en sus píxeles y desde allí comienza la siguiente etapa.
2. **Detección de gesto.** Aquí el sistema se encuentra a la espera de la realización de un gesto (por parte del usuario) para iniciar el seguimiento de manos. Este gesto es reconocido cuando el usuario levanta cualquiera de sus manos.
3. **Seguimiento de manos.** Aquí el usuario es capaz de controlar qué píxeles son iluminados con el movimiento de sus manos. Básicamente, las posiciones espaciales de las manos son proyectadas en la estructura de la obra, proporcionando una experiencia interactiva de fácil asimilación para el usuario.
4. **Finalización de sesión.** Una vez que el sistema pierde el rastro del usuario presente, la etapa 1 vuelve a comenzar.

La implementación fue realizada en base a el *template* de Sendero Client ofrecido en la distribución *open-source* de Sendero y la librería *OpenNI* [130] como SDK para el control del sensor de profundidad *Kinect.*

4.4.2 Interacción Remota

Con el objetivo de poner a prueba la infraestructura para la interacción remota de clientes, se desarrolló una interacción basada en los eventos JavaScript [131] *mousemove* (para navegadores web) y *touchmove* (para dispositivos móviles). La interacción consiste en

reflejar el seguimiento del puntero (cursor o dedo) con una estela de color que permanece asociada a cada usuario interactuando. Las estelas de todos los usuarios participando de la interacción son mezcladas en una instancia de Sendero Client e introducidas como entrada a Sendero Server, haciendo visible el resultado en la obra física y el resto de los clientes conectados de manera instantánea.

La implementación de esta interacción consta de:

- Métodos *JavaScript* en Sendero Web.
- Implementación de una instancia de Sendero Client capaz interpretar estos datos.

Adicionalmente, debe estar en funcionamiento Sendero Interaction Server y la cola de tareas rabbitMQ que oficia de intermediario entre este servidor y la implementación de Sendero Client encargada de interpretar las interacciones.

Este desarrollo fue presentada durante el festival de arte y ciencia Equinoccio, al igual que en el evento Ingeniería de Muestra 2015, presentaciones detalladas en el Anexo K.

La Figura 4-29 muestra una estela de color rojo siguiendo el movimiento trazado por un usuario, mientras que otro usuario hace los mismo en otro dispositivo y es visualizado como una estela de color azul.

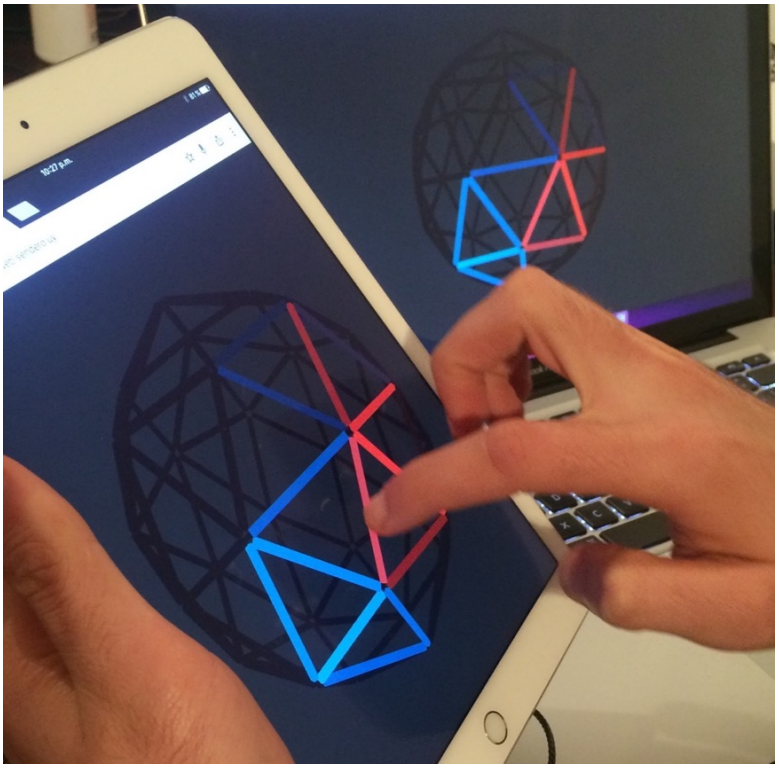


Figura 4-29 - Interacción remota con dos clientes.

4.4.3 Sync Meter

Un aspecto sumamente importante a evaluar en la implementación realizada, es la sincronización en la reproducción entre los nodos de la red. Es crucial que cada *frame* transmitido sea reproducido al mismo tiempo (con un cierto margen) por todos los controladores de LED.

Es aquí donde surge una gran problemática: ¿cómo medir objetivamente el nivel de sincronismo entre receptores? Ante esta pregunta surgieron varias alternativas tales como registrar el comportamiento de las luces a través de una filmación en cámara lenta, para posteriormente corroborar su correctitud. Finalmente, se llegó al diseño e implementación de un instrumento capaz de registrar (desde el punto de vista de un observador externo) el momento en el cual los *frames* son reproducidos por cada nodo receptor, este instrumento es denominado *Sync Meter*.

En en Anexo L se brinda una descripción detallada de su funcionamiento y uso.

Capítulo 5

Evaluación Experimental

En este capítulo se presenta de manera resumida las pruebas experimentales que fueron realizadas al sistema y los principales resultados obtenidos.

5.1 Introducción

Sendero es un sistema complejo que involucra la interacción de múltiples componentes. Con el objetivo de validar las funcionalidades implementadas y probar las capacidades de la nueva versión, se llevaron a cabo una serie de pruebas experimentales que evalúan los aspectos más importante de cada componente y la forma en que estos interactúan entre sí.

Se recomienda fuertemente ver el **Anexo M, donde puede encontrarse el desarrollo de las pruebas con una explicación detallada, datos, gráficos y análisis de las mismas.** Dicho contenido fue omitido en el documento principal debido a su nivel de detalle y extensión.

5.2 Objetivos

Las pruebas desarrolladas persiguen los siguientes objetivos:

- Evaluar el comportamiento del subsistema de control de LEDs, constituido por los componentes Sendero Middleware y Sendero Wireless Bondibar.
- Evaluar el comportamiento del subsistema Web, formado por Sendero Server, Sendero Streaming Server, Sendero Web y Sendero Cardboard.
- Evaluar el comportamiento del subsistema de interacción Web, conformado por Sendero Server, Sendero Interaction Server, Sendero Streaming Server, Sendero Web y una instancia específica de Sendero Client.
- Verificar la capacidad del sistema para desarrollar la obra Barcelona.

5.3 Entorno

Como parte de la infraestructura empleada para la realización de las pruebas se construyeron 12 Sendero Wireless Dongle e igual cantidad de controladores Bondibar.

Para el montaje de la red WLAN se utilizó un *router* inalámbrico doméstico TP-Link WR941ND cargado con el *firmware* DD-WRT v24-sp2.

Por otra parte, Sendero Streaming Server ejecuta en un instancia de Amazon AWS t2.micro que corre Ubuntu 14.04.3 LTS con un CPU Intel Xeon de hasta 3.3 GHz y 1 Gib de memoria RAM. Dicho servidor se encuentra en Oregon, Estados Unidos y tiene una modalidad *Free Tier* por un año.

Sendero Server, Sendero Interaction Server y la instancia especial de Sendero Client ejecutan en una laptop con un CPU Intel Core™ i7-4700MQ de 2.40 GHz quad-core con HyperThreading y 8 GB de RAM, ubicado en Montevideo, Uruguay y conectado a una red DSL de 30 Mbps de bajada y 4 Mbps de subida.

5.4 Plan de pruebas

La Tabla 5.1 lista las pruebas realizadas y su objetivo.

Subsistema	**Prueba**	**Objetivo**
Control de LEDs	Máxima capacidad de píxeles.	Determinar la capacidad máxima de píxeles soportados por el sistema para las distintas configuraciones de *frame rate* y cantidad de grupos *multicast*.
	Sincronismo de la reproducción entre receptores.	Medir y evaluar el nivel de sincronismo en la reproducción de *frames* entre receptores. Concretamente, evaluar las diferencias de tiempo transcurrido entre las reproducciones de un mismo *frame* entre todos los receptores.
	Latencia de reproducción.	Medir y evaluar el nivel de latencia en la reproducción entre los paquetes generados desde el emisor hacia cada nodo receptor. Concretamente, medir el tiempo transcurrido desde que un *frame* es generado hasta que es reproducido.
Web	Estimación del ancho de banda y recursos requeridos en un servidor.	Estimar el ancho de banda y recursos necesarios en un servidor para dar soporte al subsistema web de Sendero.
	Calidad de la reproducción.	Evaluar la calidad general de la reproducción en un cliente remoto en base a las métricas asociadas a las estadísticas que cada cliente genera.

	Recuperación frente a pérdidas.	Evaluar el beneficio del mecanismo de recuperación frente a pérdidas implementado.
	Latencia de reproducción.	Evaluar la latencia de la reproducción.
	Compresión.	Evaluar el beneficio del mecanismo de compresión empleado.
Interacción Web	Prueba de Stress de interacción remota.	Evaluar el comportamiento del sistema ante ráfagas constantes de interacciones concurrentes.

Tabla 5.1 - Plan de pruebas experimentales

5.5 Resumen de resultados

En lo que refiere al subsistema de control de LEDs, se pudo constatar que el mismo puede brindar soporte a un máximo de 976 píxeles a 30 *fps* y 1200 píxeles a 24 *fps*. Esto supera ampliamente las dimensiones de las obras desarrolladas anteriormente con Sendero (Barcelona 90 píxeles y Celebra 200 píxeles) y significa un resultado muy satisfactorio. Luego, se determinó que la diferencia de tiempo entre la reproducción de un mismo *frame* por parte de los nodos de la red (sincronización de la reproducción) puede alcanzar en promedio el valor de 1 ms, con una desviación estándar de 0.8 ms. Si bien no se consiguieron fuentes que lo respalden, se observó que diferencias menores a 15 ms son imperceptibles, por lo que la sincronización se percibe como perfecta. Por último, la latencia de reproducción obtenida se comporta de manera coherente a lo esperado, encontrándose entorno a los 103 ms, lo cual es considerada una latencia imperceptible para un flujo multimedia en tiempo real [132].

Con respecto al subsistema web, se estimó que en el contexto de la obra Barcelona el servidor empleado puede dar soporte a 285 clientes remotos simultáneos. Este valor sirve como referencia pero no tiene un significado en sí mismo, ya que la limitación es la capacidad del servidor utilizado, y la cantidad de clientes esperada dependerá exclusivamente de la obra y el lugar donde la misma sea expuesta. Por otra parte, se evaluó la calidad de la reproducción de un flujo de prueba y se constató que la utilización de las técnicas de *streaming* aplicadas permiten absorber la disparidad de la entrega de los paquetes (ocasionada por su transmisión sobre Internet), obteniéndose una tasa de reproducción muy aproximada a la esperada de 30 *fps*. En cuanto a la recuperación frente a pérdidas, el mecanismo implementado funciona correctamente, ya que la mayoría de los paquetes faltantes a su momento de reproducción son reemplazados por una estimación. Sin embargo, el hecho de no complementar esta medición con métricas de calidad de la experiencia, no permite afirmar que los paquetes reemplazados se adapten al flujo. Con respecto a la latencia de reproducción, se consiguió un valor promedio de 295 ms, que se encuentra por encima del aceptado para flujos de video en tiempo real de 250 ms [132], [133]. No obstante, un análisis de este valor permitió observar que el mismo está afectado principalmente por el tiempo de transmisión en la red, lo cual puede ser fácilmente solucionado configurando a Sendero Streaming Server en un servidor físicamente más cercano a Sendero Server. Por último, el mecanismo de compresión de datos utilizado puede al-

canzar una importante disminución en el tráfico transmitido, aunque naturalmente depende de los datos en cuestión y éstos de la obra que se desarrolle.

Finalmente, se validó la interacción remota con resultados satisfactorios empleando hasta 200 clientes simúlatenos. Nuevamente, las pruebas realizadas para este aspecto del sistema sirven como referencia para futuros desarrollos, ya que se encuentran estrechamente asociadas al *hardware* empleado y a las interacciones programas, la red, servidores, etc.

5.6 Conclusión

Las pruebas experimentales realizadas permitieron cuantificar las capacidades más importantes de Sendero, y garantizar que los requerimientos de la obra Barcelona pueden ser satisfechos sin dificultad, superando ampliamente el número de píxeles que esta obra utiliza, y consiguiendo una calidad de la reproducción muy satisfactoria técnicamente, tanto en lo que refiere al *streaming* en la red inalámbrica local, como al que se da a través de Internet con los clientes remotos. Además se validó la posibilidad de interacción simultánea de un importante número de clientes remotos.

Por otro lado, estas pruebas sirvieron para constatar varios puntos a mejorar del sistema y de las pruebas en sí. En primer lugar, los nodos receptores Wireless Bondibar presentan una gran sensibilidad a los espacios con interferencia, presentando altas tasas de pérdidas de paquetes. Luego, si bien la tecnología Wi-Fi es en teoría capaz de operar en rangos de más de 100 metros, no se realizaron pruebas que lo validen. Por último, varias de las pruebas llevadas a cabo validan aspectos técnicos que requieren del respaldo de métricas de la calidad de la experiencia (QoE) que se ofrece para dar un panorama más completo de los resultados obtenidos. Tareas que pueden formar parte de un trabajo futuro.

Capítulo 6

Conclusiones y Trabajo Futuro

En este capítulo se presentan las conclusiones del trabajo y se describen posibles mejoras que pueden ser fuente de un trabajo futuro.

6.1 Conclusiones

El objetivo central de este proyecto fue desarrollar mejoras y extensiones al sistema de iluminación LED RGB Sendero. Con este cometido, se ejecutaron tareas de investigación, desarrollo y adaptación de *software*, diseño y construcción de *hardware*, configuración de servidores e infraestructura de red, documentación, y finalmente, posicionamiento de la nueva versión del sistema como una herramienta *open-source* y *open-source hardware* para el desarrollo de arte con nuevos medios, provista por el Laboratorio de Medios de Facultad de Ingeniería de la Universidad de la República.

En lo que refiere a investigación, se llevaron a cabo estudios sobre el estado del arte en transferencia de video sobre redes de datos y comunicaciones *Machine-To-Machine* (M2M). Se constató que Sendero puede ser enmarcado en el contexto de estas disciplinas, y en consecuencia, el relevamiento y aplicación de técnicas, protocolos y mecanismos cuya utilización y estudio se encuentran ampliamente extendidos, resultaron de gran aporte a los desarrollos implementados. En particular, los protocolos SAP, RTP y SDP empleados para el *streaming* de contenido multimedia, sirvieron de inspiración para el desarrollo de los protocolos SLP y SWP, que implementan un subconjunto de sus funcionalidades con el fin de posibilitar la distribución del contenido de Sendero. Por otra parte, la arquitectura propuesta por ETSI, el estudio de módulos M2M y tecnologías de red de área local, contribuyeron con la búsqueda de una solución al objetivo de migrar el esquema de comunicación de *hardware* de Sendero hacia uno inalámbrico. Ambos temas comprenden un extenso campo de estudio, que fue imposible abarcar con total profundidad durante este proyecto, sin embargo, la información relevada resultó fundamental a la hora de decidir qué, cómo y por qué implementar cada componente, y en los resultados obtenidos.

En cuanto a la migración del esquema de comunicación de *hardware*, se llevó a cabo un profundo análisis de la implementación tomada como base de Sendero Server y se decidió implementar la solución de modo tal que no se altere ninguna de las funcionalidades existentes. Para ello, se desarrolló un *middleware* (Sendero Middleware) capaz de oficiar de adaptador entre Sendero Server y los componentes de la red inalámbrica de Sendero que esperan contenido SLP. De este modo, Sendero conserva la compatibilidad con dispositivos Art-Net, a la vez que mantiene un bajo acoplamiento con la solución de *hardware*, pudiendo ser reemplazada o actualizada sin necesidad de modificar a Sendero Server.

La solución de *hardware* propuesta implicó la selección de un módulo inalámbrico programable (ESP-12E DevKit), la modificación de los controladores de LED (Bondibar), y el desarrollo de un circuito adaptador que permite conectar estos componentes (Sendero Wireless Dongle). Proveer de conectividad inalámbrica al controlador Bondibar a través de un dispositivo externo, hizo posible mantener al mínimo las modificaciones requeridas en dicho controlador, con un bajo nivel de acoplamiento que permite fácilmente reemplazar el módulo inalámbrico (contemplando la velocidad del avance tecnológico en este tipo de dispositivos). Adicionalmente, se desarrolló un *firmware* capaz de recibir información mediante el protocolo desarrollado SLP, aplicar técnicas de *streaming* para su reproducción, implementar mecanismos que permitan la sincronización de la reproducción con otros nodos y finalmente, comunicar el contenido apropiadamente al controlador de LEDs. Estas tareas involucraron programación de bajo nivel, estudio de protocolos de comunicación de *hardware*, y el diseño y ensamblaje de los circuitos creados/modificados, lo cual significó un gran esfuerzo de investigación en áreas ajenas al foco principal de la carrera, extendiendo la base de conocimiento adquirida y logrando una visión más completa de la interacción *hardware-software*.

Por otro lado, con el objetivo de permitir la interacción de espectadores remotos, se desarrolló un conjunto de aplicaciones que permiten la distribución del contenido de la obra a través de Internet, la reproducción del mismo y la interacción en tiempo real con el sistema. Específicamente, se implementaron aplicaciones web (Sendero Web y Sendero Cardboard) capaces de desplegar una representación virtual de la obra, y brindar mecanismos de interacción que contribuyen en la generación del contenido visual de Sendero. Por otra parte, se implementó y configuró un servidor para la distribución del contenido (Sendero Streaming Server), que además ofrece una interfaz administrativa para evaluar la calidad del *streaming* (Sendero Dashboard). Finalmente se desarrollaron mecanismos para la recepción y manejo de comandos de interacción provenientes de los espectadores (Sendero Interaction Server). El desarrollo de este subsistema basado en la tecnología de WebSockets, brinda una solución más adecuada al problema de la interacción en tiempo real a través de Internet, a la vez que permite obtener buenos resultados en lo que refiere al *streaming* de contenidos. En cuanto a los resultados obtenidos, se puede afirmar que los mecanismos implementados funcionan exitosamente, y se considera que es necesaria

la realización de pruebas que permitan medir la calidad de la experiencia (QoE) con el fin de generar una evaluación más completa.

Al comienzo del proyecto se planteó como objetivo el desarrollo de una obra de arte con nuevos medios que haga uso de las nuevas características del sistema. Como se estableció en el Alcance del Proyecto (ver Sección 3.4), esta tarea fue postergada y reemplazada por la realización de pruebas que garanticen la capacidad de Sendero para montar la obra Barcelona. Las pruebas llevadas a cabo constatan que las capacidades del sistema superan ampliamente los requerimientos de esta obra, por lo que se puede concluir que no existen limitaciones para su implementación. La presentación fue pactada para el día 1 de diciembre de 2016, en el marco de una exposición a desarrollarse en el Espacio de Arte Contemporáneo, ubicado en Arenal Grande 1930, Montevideo, Uruguay.

Aparte de las tareas de desarrollo, investigación y evaluación llevadas a cabo, durante el transcurso del proyecto se realizaron presentaciones públicas de los avances, en el marco del festival de ciencia y tecnología Equinoccio, y en el evento Ingeniería de Muestra 2015. La participación en estas instancias involucró el desarrollo de instalaciones, cuya evaluación por parte de espectadores reales sirvió para realizar una validación temprana de ciertos aspectos y decisiones. Además, permitió exponer las capacidades del sistema y acercarlo a potenciales usuarios.

Con respecto al posicionamiento de la nueva versión de Sendero como una herramienta *open-source* y *open-source hardware*, se publicó el código y los esquemas de diseño de *hardware* en la plataforma *Github,* bajo la organización perteneciente al Laboratorio de Medios [134], empleando la licencia de software MIT. Los repositorios cuentan con la documentación necesaria para comprender la estructura de la solución y realizar la configuración de cada componente.

Finalmente, en base a lo expuesto en los párrafos anteriores, es posible afirmar que el sistema cumple con los objetivos planteados y el alcance estipulado. Se extendieron y mejoraron las capacidades del sistema original, transformando a Sendero en una herramienta más versátil, extensible y escalable. Al mismo tiempo, durante el desarrollo y posterior análisis se descubrieron aspectos que aún pueden ser mejorados, y son postulados como tareas para un trabajo futuro.

6.2 Trabajo futuro

Durante el desarrollo y evaluación del proyecto surgieron propuestas de extensión o mejora de determinados aspectos que por quedar fuera del alcance, o no significar una tarea de prioridad no se pudieron llevar a cabo. Es por esto que son postulados como actividades a desarrollar en el marco de un trabajo futuro.

6.2.1 Wireless Bondibar

1. Evaluar el desempeño utilizando *hardware* más potente. Dado que la solución implementada utiliza C++ como lenguaje y sus librerías estándar, existiría solo una pequeña porción del código a ser adaptada utilizando otro *hardware*. Durante el desarrollo de este proyecto, salió al mercado el módulo Wi-Fi ESP32 [135] desarrollado por el mismo fabricante, el cual provee (entre otras prestaciones) un procesador *dual-core*. Adicionalmente los módulos UART en este nuevo diseño, se encuentran implementados por DMA (*Direct Memory Access*) lo cual liberaría el uso de CPU al momento de transferir información de depuración hacia una computadora. Con el *hardware* utilizado en la actualidad, transferir información de depuración mientras se utiliza el módulo implica un notorio deterioro en su funcionamiento.
2. Evaluar técnicas de compresión basadas en la redundancia espacial y temporal del contenido transmitido.
3. Con el objetivo de superar la máxima cantidad de datos posible por paquete SLP y asumiendo que la misma está restringida por la utilización de la implementación de lwIP [136] (*stack* para TCP/IP), se propone estudiar la realización de una compilación particular para superar este límite.
4. Utilizando mejor *hardware*, estudiar el rendimiento enviando estadísticas de monitoreo periódicamente.
5. Realizar un log interno y persistente en los receptores para determinar las causas de las fallas (por ejemplo conocer las últimas 20 operaciones realizadas).
6. Realizar estudios para determinar la máxima distancia soportada y evaluar a su vez la utilización de repetidores Wi-Fi.

6.2.2 Sendero Middleware

1. Extender la implementación actual del protocolo Art-Net DMX para considerar el escenario de más de un universo por *frame*, esto incrementaría la cantidad posible de píxeles a ser enviada desde Sendero Server.
2. Contando con mejor *hardware* en los módulos receptores, se podría exponer un panel web con el estado de conexión y estadísticas de cada nodo en la sesión. Esto ayudaría a tener una visión global y en tiempo real del funcionamiento del sistema.

6.2.3 Servidores/Clientes Web

1. Evaluar la utilización de servidores más cercanos geográficamente a los utilizados con el objetivo de disminuir la latencia de reproducción e interacción, brindando una mejor experiencia al usuario.
2. Ahondar en técnicas de compresión que saquen provecho de la localidad temporal existente entre *frames*.
3. Realizar un estudio y estimar métricas de QoE (*Quality of Experience*) frente a usuarios reales.

6.2.4 General

1. Debido a la gran cantidad de sub-sistemas distribuidos que operan conjuntamente para brindar la solución global del sistema, resulta de gran importancia contar con un servidor/gestor de la configuración centralizado que unifique y brinde accesibilidad de manera estandarizada a la totalidad de subsistemas (por ejemplo una API REST).
2. Con respecto a la instalación de una obra, estudiar la viabilidad de métodos automáticos para realizar la asociación entre los píxeles físicos en el espacio real de la obra, y los virtuales manejados por los distintos sub-sistemas para dibujar la representación de la obra. Actualmente esta tarea es realizada manualmente.
3. Estudiar la utilización de mecanismos de autenticación en las distintas comunicaciones entre subsistemas.
4. Respecto a la comunicación entre Sendero Middleware y los nodos receptores Wireless Bondibar, evaluar la utilización de canales TCP para implementar la transmisión de los *frames* de la obra con el objetivo de soportar entornos con interferencia donde la comunicación a través de UDP se ve degradada significativamente.

Referencias

[1] R. Jana y M. Tribe, *NEW MEDIA ART*. Taschen, 2006.

[2] L. Manovich, *The Language of New Media*. MIT Press, 2001.

[3] T. Laurenzo, "Decoupling and context in new media art", Ph.D. Thesis, Facultad de Ingeniería, Universidad de la República, 2014.

[4] Andrew Donovan, Sarah Miller, "New Media Arts Scoping Study, Report to the Australia Council for the Arts", Australia Council for Arts, Government Report, 2006.

[5] "Sendero - clark.uy". [En línea]. Disponible en: http://clark.uy/blog/view/sendero.html. [Consultado: 17-may-2016].

[6] C. Clark, "New Media and Impressionism", Masters Thesis, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, 2015.

[7] LaboratorioDeMedios, "LaboratorioDeMedios/Sendero". [En línea]. Disponible en: https://github.com/LaboratorioDeMedios/Sendero. [Consultado: 17-may-2016].

[8] LaboratorioDeMedios, "LaboratorioDeMedios/Sendero_GenericClient". [En línea]. Disponible en: https://github.com/LaboratorioDeMedios/Sendero_GenericClient. [Consultado: 17-may-2016].

[9] P. Gindel, "Bondibar - Github". [En línea]. Disponible en: https://github.com/laboratoriodemedios/bondibar. [Consultado: 17-may-2016].

[10] "Art-Net". [En línea]. Disponible en: http://art-net.org.uk/. [Consultado: 17-may-2016].

[11] "Home | mbed". [En línea]. Disponible en: www.mbed.com/en. [Consultado: 17-may-2016].

[12] "openFrameworks". [En línea]. Disponible en: http://openframeworks.cc/. [Consultado: 17-may-2016].

[13] Massachusetts Institute of Technology, "MIT License". Massachusetts Institute of Technology, 1988.

[14] "DMX512 - USITT". [En línea]. Disponible en: http://old.usitt.org/DMX512.aspx. [Consultado: 17-may-2016].

[15] "Barcelona by Bondi | Palmera blog". [En línea]. Disponible en: http://www.pablogindel.com/2013/06/un-solido-de-catalan/. [Consultado: 17-may-2016].

[16] T. L. C. Clark, "Celebra", presentado en Proceedings of International Symposium on Electronic Art.

[17] "Medialab // Celebra". [En línea]. Disponible en: http://www.fing.edu.uy/grupos/medialab/projects/celebra/project.html. [Consultado: 17-may-2016].

[18] "Celebra - clark.uy". [En línea]. Disponible en: http://clark.uy/blog/view/Celebra.html. [Consultado: 17-may-2016].

[19] "Bicentenario". [En línea]. Disponible en: http://www.bicentenario.gub.uy/bicentenario-uruguay/comision/. [Consultado: 17-may-2016].
[20] "Medialab". [En línea]. Disponible en: http://www.fing.edu.uy/grupos/medialab/. [Consultado: 17-may-2016].
[21] "Bondi - An interaction design collective". [En línea]. Disponible en: http://bondi.cc/. [Consultado: 17-may-2016].
[22] "ISEA2013". [En línea]. Disponible en: http://www.isea2013.org/. [Consultado: 17-may-2016].
[23] "Laval virtual - International conferences and exhibition of Virtual technologies and uses". [En línea]. Disponible en: http://www.laval-virtual.org/en/. [Consultado: 17-may-2016].
[24] "Barcelona - clark.uy". [En línea]. Disponible en: http://clark.uy/blog/view/barcelona.html. [Consultado: 17-may-2016].
[25] "art, design, and research by Tomás Laurenzo". [En línea]. Disponible en: http://laurenzo.net/projects/barcelona. [Consultado: 17-may-2016].
[26] "Singularity University - Solving Humanity's Grand Challenges". [En línea]. Disponible en: http://singularityu.org. [Consultado: 17-may-2016].
[27] A. Durresi, D. Arjan, y J. Raj, "RTP, RTCP, and RTSP — Internet Protocols for Real- Time Multimedia Communication", en *Industrial Electronics*, 2004, pp. 398–408.
[28] J. F. Kurose y K. W. Ross, *Redes de Computadoras: Un enfoque descendente*, 5ta ed. Ribera del Loira, 28. 28042 Madrid (España): PEARSON EDUCACIÓN, S. A., 2010.
[29] University of Southern California, "Transmission Control Protocol". The Internet Engineering Task Force (IETF).
[30] J. Postel, "User Datagram Protocol". The Internet Engineering Task Force (IETF), 28-ago-1980.
[31] C. Perkins, *RTP: Audio and Video for the Internet*. Addison-Wesley Professional, 2003.
[32] M. Xing, X. Min, X. Siyuan, y C. Lin, "A Real-Time Adaptive Algorithm for Video Streaming over Multiple Wireless Access Networks", *IEEE J Sel Areas Commun*, vol. 32, núm. 4, pp. 795–805, 2014.
[33] A. A. Razzac, S. E. Elayoubi, C. Tijani, y E. Bachar, "Impact of Playout Buffering on Mobile TV Performance", *IEEE Trans Mob Comput*, vol. 15, núm. 2, pp. 377–391, 2016.
[34] C. J. Sreenan, C. Jyh-Cheng, P. Agrawal, y B. Narendran, "Delay reduction techniques for playout buffering", *IEEE Trans Multimed.*, vol. 2, núm. 2, pp. 88–100, 2000.
[35] C. Perkins, P. Colin, H. Orion, y H. Vicky, "A Survey of Packet Loss Recovery Techniques for Streaming Audio", en *Readings in Multimedia Computing and Networking*, 2002, pp. 607–615.
[36] J. Crowcroft, *Internetworking Multimedia*. CRC Press, 1999.

[37] B. Furht y F. Borko, “A Survey of Multimedia Compression Techniques and Standards. Part I: JPEG Standard”, *Real-Time Imaging*, vol. 1, núm. 1, pp. 49–67, 1995.
[38] B. Furht, “A Survey of Multimedia Compression Techniques and Standards. Part II: Video Compression”, *Real-Time Imaging*, vol. 1, núm. 5, 1995.
[39] “ISO/IEC 10918-1:1994 - Information technology -- Digital compression and coding of continuous-tone still images: Requirements and guidelines”. [En línea]. Disponible en: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=18902. [Consultado: 22-may-2016].
[40] “ISO/IEC 13818-2:2013 - Information technology -- Generic coding of moving pictures and associated audio information -- Part 2: Video”. [En línea]. Disponible en: http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=61152. [Consultado: 22-may-2016].
[41] UIT-T, “H.323 : Sistemas de comunicación multimedia basados en paquetes”, E 35738, jul. 2010.
[42] R. Zurawski, J. Rosenberg, G. Camarillo, y A. Johnston, “SIP: Session Initiation Protocol”. The Internet Engineering Task Force (IETF), jun-2002.
[43] H. Schulzrinne, “Real Time Streaming Protocol (RTSP)”, abr. 1998.
[44] M. Handley, C. Perkins, y E. Whelan, “SAP: Session Announcement Protocol”. oct-2000.
[45] M. Handley, C. Perkins, y V. Jacobson, “SDP: Session Description Protocol”, jul. 2006.
[46] G. F. H. Schulzrinne S. Casner, R. Frederick, V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, ene. 1996.
[47] P. J. Leach, T. Berners-Lee, J. C. Mogul, L. Masinter, R. T. Fielding, y J. Gettys, “Hypertext Transfer Protocol -- HTTP/1.1”, jun. 1999.
[48] Apple Inc, “HTTP Live Streaming (HLS) - Apple Developer”. [En línea]. Disponible en: https://developer.apple.com/streaming/. [Consultado: 22-may-2016].
[49] R. Pantos y W. May, “HTTP Live Streaming”, abr. 2016.
[50] G. Wilkins, S. Salsano, S. Loreto, y P. Saint-Andre, “Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP”, abr. 2011.
[51] I. Fette y A. Melnikov, “Rfc 6455: The websocket protocol”, *IETF Dec.*, 2011.
[52] “WebRTC Home | WebRTC”. [En línea]. Disponible en: https://webrtc.org/. [Consultado: 22-may-2016].
[53] A. B. Johnston y D. C. Burnett, *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-time Web*. 2013.
[54] D. Boswarthick, O. Elloumi, y O. Hersent, *M2M Communications: A Systems Approach*. John Wiley & Sons, 2012.
[55] B. S. A. K. S. Luis Barriga, “M2M Remote-Subscription Management”, Ericsson Review, Technical Report, 2011.
[56] S. Ajah, A. Sylvester, A. Al Sherbaz, T. Scott, y P. Phil, “Machine-to-machine communications energy efficiencies: the implications of different M2M communications specifications”, *Int J Wirel. Mob. Comput*, vol. 8, núm. 1, p. 15, 2015.

[57] “Beecham Research”. [En línea]. Disponible en: http://www.beechamresearch.com. [Consultado: 23-may-2016].
[58] B. Research, *M2M/IoT Sector Map*. 2011.
[59] “ETSI - European Telecommunications Standards Institute”. [En línea]. Disponible en: http://www.etsi.org/. [Consultado: 23-may-2016].
[60] ETSI, “Machine-to-Machine communications (M2M); Functional architecture”, ETSI, 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE, ETSI TS 102 690 v2.1.1, oct. 2013.
[61] S. S. Prakash y C. M. Rao, “A Comprehensive Study of M2M Area Networks and Challenges”.
[62] R. Challoo, A. Oladeinde, N. Yilmazer, S. Ozcelik, y L. Challoo, “An Overview and Assessment of Wireless Technologies and Co- existence of ZigBee, Bluetooth and Wi-Fi Devices”, *Procedia Comput Sci*, vol. 12, pp. 386–391, 2012.
[63] J.-S. Lee, L. Jin-Shyan, S. Yu-Wei, y S. Chung-Chou, “A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi”, presentado en IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society, 2007.
[64] M. Chen, C. Min, W. Jiafu, G. Sergio, L. Xiaofei, y V. C. M. Leung, “A Survey of Recent Developments in Home M2M Networks”, *IEEE Commun. Surv. Tutor.*, vol. 16, núm. 1, pp. 98–114, 2014.
[65] O. Hersent, D. Boswarthick, y O. Elloumi, *The Internet of Things: Key Applications and Protocols*. John Wiley & Sons, 2011.
[66] Institute of Electrical and Electronics Engineers (IEEE), “Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”, 802.11-2012, 2012.
[67] Institute of Electrical and Electronics Engineers, “Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifi- cations for Wireless Personal Area Networks (WPANs)”, 802.15.1-2005, jun. 2005.
[68] Institute of Electrical and Electronics Engineers, “Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifi- cations for Low-Rate Wireless Personal Area Networks (WPANs)”, 802.15.4, sep. 2006.
[69] “mbed LPC1768 | mbed”. [En línea]. Disponible en: https://developer.mbed.org/platforms/mbed-LPC1768/. [Consultado: 12-may-2016].
[70] “Palmera blog”. [En línea]. Disponible en: http://www.pablogindel.com/. [Consultado: 19-may-2016].
[71] “IEEE-SA -IEEE Get 802 Program - 802.3: Ethernet”. [En línea]. Disponible en: https://standards.ieee.org/about/get/802/802.3.html. [Consultado: 12-may-2016].
[72] World-Semi, “WS2801 Datasheet”. [En línea]. Disponible en: http://www.world-semi.com/en/Driver/Lighting_LED_driver_chip/WS2801/. [Consultado: 17-may-2016].
[73] Apple Inc, “iOS 9”. [En línea]. Disponible en: http://www.apple.com/es/ios/. [Consultado: 12-may-2016].
[74] “Android”. [En línea]. Disponible en: https://www.android.com/. [Consultado: 12-may-2016].

[75] "Django", *Django - The web framework for perfectionists with deadlines.* [En línea]. Disponible en: https://www.djangoproject.com/. [Consultado: 25-may-2016].
[76] "Espacio de Arte Contemporáneo". [En línea]. Disponible en: http://www.eac.gub.uy/. [Consultado: 14-may-2016].
[77] "MINA - Network Management - Artificial Intelligence - Facultad de Ingeniería | Universidad de la República - Uruguay". [En línea]. Disponible en: https://www.fing.edu.uy/inco/investigacion/grupos/MINA. [Consultado: 17-may-2016].
[78] N. Fourty, T. Val, P. Fraisse, y M. J.-J., "Comparative analysis of new high data rate wireless communication technologies 'From Wi-Fi to WiMAX'", presentado en Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services - (icas-isns'05), 2005.
[79] Yeonchul Shin, S. Yeonchul, C. Munhwan, K. Jonghoe, K. Young-Doo, I. Jong-Tae, y C. Sunghyun, "Empirical analysis of video multicast over WiFi", presentado en 2011 Third International Conference on Ubiquitous and Future Networks (ICUFN), 2011.
[80] S. U. Rehman, T. Thierry, y D. Walid, "Multicast video streaming over WiFi networks: Impact of multipath fading and interference", presentado en 2011 IEEE Symposium on Computers and Communications (ISCC), 2011.
[81] P. Gupta y P. R. Kumar, "The capacity of wireless networks", *IEEE Trans Inf Theory*, vol. 46, núm. 2, pp. 388–404, 2000.
[82] S. R. Kulkarni y P. Viswanath, "A deterministic approach to throughput scaling in wireless networks", presentado en Proceedings IEEE International Symposium on Information Theory.
[83] "IE 802.11 Network Topology". [En línea]. Disponible en: http://www.cs.uccs.edu/~gsc/pub/master/pjfong/UCCS%20Project/Articles/IE%20802_11%20Network%20Topology.htm. [Consultado: 05-jun-2016].
[84] ESP8266.com, "Everything ESP8266". [En línea]. Disponible en: http://www.esp8266.com/. [Consultado: 18-may-2016].
[85] "EMW3165 Dev Forum - Index page". [En línea]. Disponible en: http://www.emw3165.com/. [Consultado: 18-may-2016].
[86] "CC3200MOD - SimpleLink™ Wi-Fi and Internet-of-Things Module Solution, a Single-Chip Wireless MCU". [En línea]. Disponible en: http://www.ti.com/product/cc3200MOD. [Consultado: 07-sep-2016].
[87] 10o7, "SOM9331". [En línea]. Disponible en: https://github.com/10to7/SOM9331. [Consultado: 18-may-2016].
[88] "Ralink RT5350 - Wikidev". [En línea]. Disponible en: https://wikidevi.com/wiki/Ralink_RT5350. [Consultado: 18-may-2016].
[89] "CoreWind Embedded system for ARM SBC,Computer-on-Module and Custom Design". [En línea]. Disponible en: http://www.nanopi.org/. [Consultado: 18-may-2016].
[90] "Arduino - Compare board specs". [En línea]. Disponible en: https://www.arduino.cc/en/Products/Compare. [Consultado: 18-may-2016].

[91] "Raspberry Pi Microcontroller Kit". [En línea]. Disponible en: http://www.makershed.com/pages/raspberry-pi-comparison-chart. [Consultado: 18-may-2016].
[92] yichone, "User Manual for ESP-12E DevKit", may 2015.
[93] Ubuntu, "Ubuntu 14.04.4 LTS (Trusty Tahr)", *Ubuntu 14.04.4 LTS (Trusty Tahr)*. [En línea]. Disponible en: http://releases.ubuntu.com/14.04/. [Consultado: 06-may-2016].
[94] Kaazing, "HTML5 WebSocket - A Quantum Leap in Scalability for the Web". [En línea]. Disponible en: http://www.websocket.org/quantum.html. [Consultado: 22-may-2016].
[95] E. Estep, "Mobile HTML5: Efficiency and Performance of WebSockets and Server-Sent Events", Masters Thesis, Aalto University, 2013.
[96] "LZ4 - Extremely fast compression". [En línea]. Disponible en: http://www.lz4.org. [Consultado: 20-may-2016].
[97] N. Webmaster, "ntp.org: Home of the Network Time Protocol". [En línea]. Disponible en: http://www.ntp.org/. [Consultado: 02-jun-2016].
[98] M. Gast, *802.11 Wireless Networks The Definitive Guide*. 1005 Gravenstein Highway North, Sebastopol, CA 95472.: O'Reilly, 2005.
[99] Espressif Systems IOT Team, "ESP8266EX Datasheet", 2015.
[100] "esp8266/Arduino". [En línea]. Disponible en: https://github.com/esp8266/Arduino. [Consultado: 28-may-2016].
[101] CadSoft, "CadSoft EAGLE PCB Design Software". [En línea]. Disponible en: http://www.cadsoftusa.com/. [Consultado: 31-may-2016].
[102] "MC78L00A Series, NCV78L00A", sep. 2015.
[103] "MC7800, MC7800A, MC7800AE, NCV7800", nov. 2014.
[104] Cyan, "Cyan4973/lz4". [En línea]. Disponible en: https://github.com/Cyan4973/lz4. [Consultado: 27-may-2016].
[105] "Welcome to Python.org". [En línea]. Disponible en: https://www.python.org/. [Consultado: 27-may-2016].
[106] S. Venaas, "IPv4 Multicast Address Space Registry", 07-jul-2016. [En línea]. Disponible en: http://www.iana.org/assignments/multicast-addresses/multicast-addresses.xhtml. [Consultado: 11-jul-2016].
[107] C. Larman, *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and the Unified Process*. Prentice Hall Professional, 2002.
[108] N. js Foundation, "Node.js". [En línea]. Disponible en: http://nodejs.org. [Consultado: 15-may-2016].
[109] "Socket.IO". [En línea]. Disponible en: http://socket.io. [Consultado: 15-may-2016].
[110] Amazon, "Tipos de instancias de Amazon EC2". [En línea]. Disponible en: https://aws.amazon.com/es/ec2/instance-types/. [Consultado: 06-may-2016].
[111] "NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy". [En línea]. Disponible en: https://www.nginx.com/. [Consultado: 03-jun-2016].
[112] I. Fette, "The WebSocket protocol", dic. 2011.

[113] C. Trieloff, C. McHale, G. Sim, H. Piskiel, J. O'Hara, J. Brome, K. van der Riet, M. Atwell, M. Lucina, P. Hintjens, y Others, "Advanced Message Queuing Protocol Protocol Specification. amq-spec", *AMQP Org*, 2006.

[114] "RabbitMQ - Messaging that just works". [En línea]. Disponible en: http://rabbitmq.com. [Consultado: 15-may-2016].

[115] S. Stefanov, *JavaScript Patterns*. O'Reilly Media, Inc., 2010.

[116] "ThreeJS". [En línea]. Disponible en: http://threejs.org/. [Consultado: 12-may-2016].

[117] mrdoob, "mrdoob/three.js". [En línea]. Disponible en: https://github.com/mrdoob/three.js. [Consultado: 28-may-2016].

[118] "WebGL Specifications". [En línea]. Disponible en: https://www.khronos.org/registry/webgl/specs/latest/. [Consultado: 28-may-2016].

[119] D. S. Moore, *Estadística aplicada básica*. Antoni Bosch editor, 2005.

[120] Y. Collet, "Lz4: Extremely fast compression algorithm", *Code Google Com*, 2013.

[121] "Loseless Data Compression", *Loseless Data Compression*. [En línea]. Disponible en: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2000-01/data-compression/lossless/lz77/algorithm.htm. [Consultado: 07-mar-2016].

[122] D. Marshall, "Huffman Coding", *Huffman Coding*, 04-oct-2001. [En línea]. Disponible en: https://www.cs.cf.ac.uk/Dave/Multimedia/node210.html. [Consultado: 07-mar-2016].

[123] "MongoDB for GIANT Ideas". [En línea]. Disponible en: https://www.mongodb.com/index. [Consultado: 28-may-2016].

[124] "Google Cardboard – Google VR". [En línea]. Disponible en: https://www.google.com/get/cardboard/. [Consultado: 27-may-2016].

[125] "Calculating Stereo Pairs". [En línea]. Disponible en: http://paulbourke.net/stereographics/stereorender/. [Consultado: 27-may-2016].

[126] Equinoccio, "Inicio - equinoccio". [En línea]. Disponible en: http://equinoccio.uy/. [Consultado: 29-may-2016].

[127] "INAE | Instituto Nacional de Artes Escénicas". [En línea]. Disponible en: http://www.inae.gub.uy/. [Consultado: 03-jun-2016].

[128] "AngularJS — Superheroic JavaScript MVW Framework". [En línea]. Disponible en: https://angularjs.org/. [Consultado: 27-may-2016].

[129] "Kinect para Xbox 360". [En línea]. Disponible en: http://www.xbox.com/en-US/xbox-360/accessories/kinect. [Consultado: 24-may-2016].

[130] "Open-source SDK for 3D sensors - OpenNI". [En línea]. Disponible en: http://openni.ru/. [Consultado: 24-may-2016].

[131] "Javascript - Event reference". [En línea]. Disponible en: https://developer.mozilla.org/en-US/docs/Web/Events. [Consultado: 28-may-2016].

[132] I. K. Ibrahim y D. Taniar, *Mobile Multimedia: Communication Engineering Perspective*. Nova Publishers, 2006.

[133] A. Mellouk, *End-to-End Quality of Service Mechanisms in Next Generation Heterogeneous Networks*. John Wiley & Sons, 2010.

[134] Laboratorio de Medios, Facultad de Ingeniería, UdelaR., "Sendero Project", *Sendero*, 07-nov-2016. [En línea]. Disponible en: https://github.com/LaboratorioDeMedios/SenderoProject. [Consultado: 09-jul-2016].

[135] Espressif Systems, "ESP32", *ESP32 | Overview*. [En línea]. Disponible en: https://espressif.com/en/products/hardware/esp32/overview. [Consultado: 09-jul-2016].

[136] "lwIP - A Lightweight TCP/IP stack - Summary [Savannah]". [En línea]. Disponible en: http://savannah.nongnu.org/projects/lwip/. [Consultado: 02-jul-2016].

Anexos

Anexo A

Categorización de aplicaciones multimedia

Las aplicaciones multimedia pueden ser categorizadas bajo la siguiente taxonomía[1]:

- **Flujos de audio/video almacenado.** En este tipo de aplicaciones, los clientes solicitan bajo demanda archivos con contenido multimedia que se encuentran almacenados en servidores. Dicho contenido se puede pausar y pasar hacia adelante o atrás. El tiempo de respuesta aceptado para cualquiera de estas opciones es del orden de 1 a 10 segundos.
  Ejemplos: Netflix[2], YouTube[3], Spotify[4].

- **Flujos de audio/video en vivo.** Este tipo de aplicaciones son similares a la difusión convencional de televisión y radio, con la excepción de que el medio de transmisión es internet y no ondas de radiofrecuencia. Conocidas como IPTV y Radio por Internet, estas aplicaciones son utilizadas por millones de usuarios a diario. Retrasos de hasta decenas de segundos desde que se inició la solicitud del contenido pueden ser tolerados.
  Ejemplos: Vera+[5].

- **Flujos de audio/video interactivo en tiempo real.** Este tipo de aplicaciones permite a los usuarios intercambiar contenido multimedia en tiempo real. Su principal utilización es en el área de la comunicación, a través de servicios de telefonía o videoconferencia. Los tiempos de respuesta tolerados son mucho más exigentes que para los flujos de audio/video almacenados y en vivo. Por lo general son del orden de algunos cientos de milisegundos.
  Ejemplos: *Skype*[6], *Google Hangouts*[7].

---

[1] J. F. Kurose y K. W. Ross, Redes de Computadoras: Un enfoque descendente, 5ta ed. Ribera del Loira, 28. 28042 Madrid (España): PEARSON EDUCACIÓN, S. A., 2010.

[2] "Netflix". [En línea]. Disponible en: https://www.netflix.com/. [Consultado: 19-may-2016].

[3] "YouTube". [En línea]. Disponible en: https://www.youtube.com/. [Consultado: 19-may-2016].

[4] "Spotify". [En línea]. Disponible en: https://www.spotify.com/uy/. [Consultado: 19-may-2016].

[5] "Veramas". [En línea]. Disponible en: http://veramas.com.uy/. [Consultado: 21-may-2016].

[6] "Skype". [En línea]. Disponible en: https://www.skype.com. [Consultado: 21-may-2016].

[7] "Google Hangouts". [En línea]. Disponible en: https://hangouts.google.com. [Consultado: 21-may-2016].

Anexo B

Protocolos de establecimiento y control de la sesión

SIP

SIP, por sus siglas en Inglés de *Session Initiation Protocol*, es un protocolo de capa de aplicación para establecer, modificar y terminar sesiones multimedia tales como video conferencias, llamadas y distribución de contenidos.

Definido en el RFC 3261[1], este protocolo hace uso de elementos llamados *proxy servers* para facilitar el ruteo de peticiones hacia la posición actual de un usuario, independientemente de que éste se encuentre en movimiento y cambie su dirección de red. Autentica y autoriza usuarios a servicios, e implementa políticas de enrutamiento de llamadas a proveedores de telefonía. Normalmente, este protocolo es utilizado en conjunto con algún protocolo de transporte como RTP[2], y emplea mensajes de descripción de la sesión cuyo formato respeta el protocolo SDP[3].

H.323

H.323 es un estándar propuesto por *UIT-T*[4] para audio y video en tiempo real entre sistemas terminales de Internet. Adicionalmente, permite la conexión a redes telefónicas de conmutación de circuitos. A diferencia de SIP, H.323 es una serie completa de estándares y protocolos integrados para conferencias multimedia e implica mecanismos de señalización, registro, control de admisión, transporte y códecs:

- **Codificación.** Una especificación acerca de cómo los nodos terminales negocian los mecanismos de codificación de audio y/o video.
- **Transporte.** Una especificación acerca de cómo se encapsulan y se envían los fragmentos de audio y video a través de la red. En particular, H.323 obliga el uso de RTP para este propósito.
- **Canalizadores.** Una especificación acerca de cómo los puntos terminales se comunican con sus respectivos canalizadores.

---

[1] R. Zurawski, J. Rosenberg, G. Camarillo, y A. Johnston, "SIP: Session Initiation Protocol". The Internet Engineering Task Force (IETF), jun-2002.

[2] H. Schulzrinne, R. Frederick, y V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications". The Internet Engineering Task Force (IETF), jul-2003.

[3] M. Handley, C. Perkins, y V. Jacobson, "SDP: Session Description Protocol", jul. 2006.

[4] UIT-T, "H.323 : Sistemas de comunicación multimedia basados en paquetes", E 35738, jul. 2010.

H.323 exige que los terminales soporten el estándar de compresión de voz G.711[5]. El soporte para video es opcional, pero en caso de incluirse el sistema terminal debe soportar, como mínimo el estándar de video QCIF H.261[6].

SAP

SAP, por sus siglas en inglés de *Session Announcement Protocol*, es un protocolo definido en el RFC 2974[7], usado para asistir a la publicación de anuncios de sesiones *multicast*, y para comunicar información importante de configuración de sesión a los posibles participantes.

Para llevar a cabo esta tarea, SAP utiliza un directorio distribuido de sesiones. Una instancia de tal directorio periódicamente enviará paquetes *multicast* con la descripción de la sesión, permitiendo que los potenciales participantes configuren las herramientas necesarias para suscribirse a dicha sesión.

RTSP

El *Real-Time Streaming Protocol (RTSP)* definido en el RFC 2326[8] es un protocolo de control no orientado a conexión cuyo objetivo principal es establecer y manejar flujos multimedia de reproducción sincronizada, provenientes desde potencialmente múltiples servidores. Su utilización se encuentra ampliamente extendida y es conocido como "el control remoto" de los servidores multimedia[9].

Este protocolo es intencionalmente muy similar a HTTP en cuanto a su sintaxis y facilidad de extensión. Para iniciar la reproducción de un flujo multimedia, un cliente obtiene desde un servidor (por ejemplo web) un archivo con la descripción de la presentación, donde se define en algún formato los aspectos vinculados a los flujos de audio y/o video: como se encuentran sincronizados, su calidad, codificación etc. Con esto, el cliente se puede contactar a el/los servidores de contenido que proporcionarán los flujos a través de mensajes SETUP, iniciando así una sesión que será administrada por el servidor. Luego, el usuario dispone de una serie de comandos similares a los de un control remoto de televisión, que le permitirán controlar la reproducción del contenido.

---

[5] UIT-T, "G.711 : Modulación por impulsos codificados (MIC) de frecuencias vocales", E 7107, jun. 1990.

[6] UIT-T, "H.261 : Video codec for audiovisual services at p x 64 kbit/s", E 1934, may 2007.

[7] M. Handley, C. Perkins, y E. Whelan, "Session Announcement Protocol", Session Announcement Protocol, oct. 2000.

[8] H. Schulzrinne, "Real Time Streaming Protocol (RTSP)", abr. 1998.

[9] J. F. Kurose y K. W. Ross, Redes de Computadoras: Un enfoque descendente, 5ta ed. Ribera del Loira, 28. 28042 Madrid (España): PEARSON EDUCACIÓN, S. A., 2010.

Anexo C

Real-Time Transport Protocol (RTP)

RTP[1], por sus siglas en inglés de *Real-time Transport Protocol,* es un protocolo que provee funcionalidades de transporte de datos multimedia en tiempo real, tales como audio o video. Estas funcionalidades incluyen identificación de tipos de contenido, números de secuencia, marcas de tiempo y monitoreo de la entrega. La exposición siguiente esta basada en el libro "*RTP: Audio and Video for the Internet*" de C. Perkins[2].

RTP es utilizado típicamente sobre *UDP*, con el fin de aprovechar sus servicios de multiplexación y *checksum*, aunque su implementación no está limitada al uso de este protocolo de transporte. Por otra parte, no provee en sí mismo ningún mecanismo para garantizar la entrega a tiempo de los datos ni ninguna otra garantía de calidad de servicio (QoS, *Quality-of-Service*); ni siquiera garantiza la entrega de paquetes o evita su entrega desordenada. Sin embargo, provee detección de pérdidas y reportes sobre la calidad del flujo, información sobre temporización y sincronización, tipo de carga útil e identificación de la fuente.

Comportamiento de un emisor RTP

En RTP, un emisor es responsable de capturar y transformar información audiovisual para su transmisión. Puede además participar en la corrección de errores y el control de la congestión, adaptando la transmisión del flujo en respuesta a la reacción de los receptores. Un diagrama del proceso de envío de información seguido por un emisor es mostrado en la Figura C.1 (adaptada de [2]).

[1] H. Schulzrinne, R. Frederick, y V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications". The Internet Engineering Task Force (IETF), jul-2003.

[2] C. Perkins, RTP: Audio and Video for the Internet. Addison-Wesley Professional, 2003.

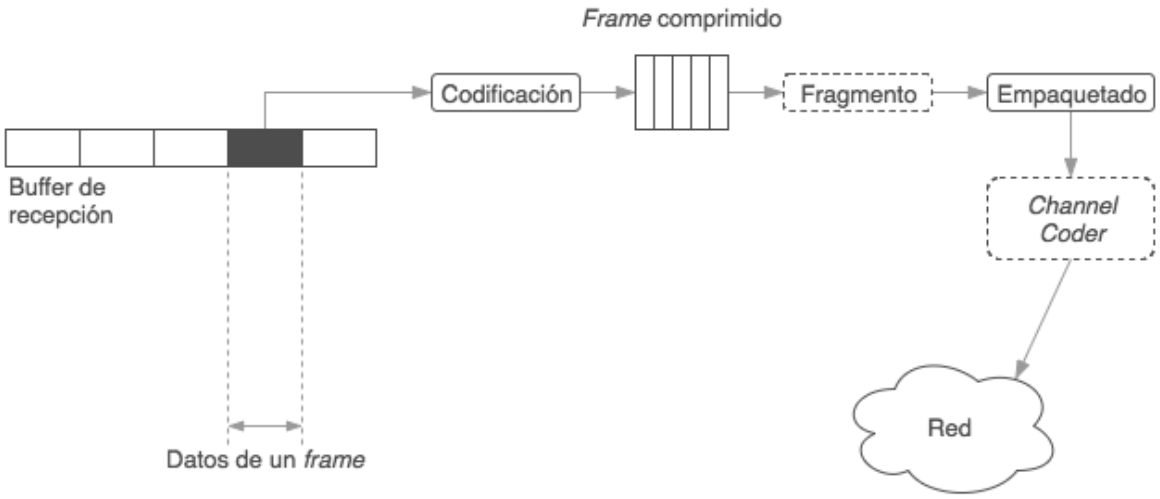


Figura C.1 – Comportamiento de un emisor RTP.

Información multimedia es capturada e insertada en un *buffer,* desde el cual se generan los *frames* comprimidos, que pueden ser codificados de distintas maneras dependiendo del algoritmo de compresión utilizado.

Estos *frames* comprimidos son incluidos como carga útil en paquetes RTP, listos para enviar. Si los mismos son demasiado grandes, entonces se fragmentan en varios paquetes. Si son pequeños, múltiples *frames* pueden ser agrupados en un paquete RTP.

Dependiendo del esquema de corrección de errores empleado, un canal de codificación (*channel coder*) puede ser utilizado para regenerar o reordenar los paquetes antes de su envío.

Por otra parte, el emisor es responsable de la generación periódica de reportes de estado sobre el flujo que está transmitiendo. Además, recibe retroalimentación sobre la calidad del servicio por parte de los receptores. Información que puede ser utilizada para adaptar la transmisión.

Comportamiento de un receptor RTP

Un receptor RTP es responsable de recolectar paquetes provenientes de la red, corregir cualquier pérdida, reconstruir la información de temporización, descomprimir los datos y presentar los resultados al usuario. Además, envía información sobre la calidad del flujo a los emisores, permitiendo que estos adapten la transmisión. Un diagrama de bloques del proceso de recepción y procesamiento de paquetes RTP es presentado en la Figura C.2 (adaptada de [2]).

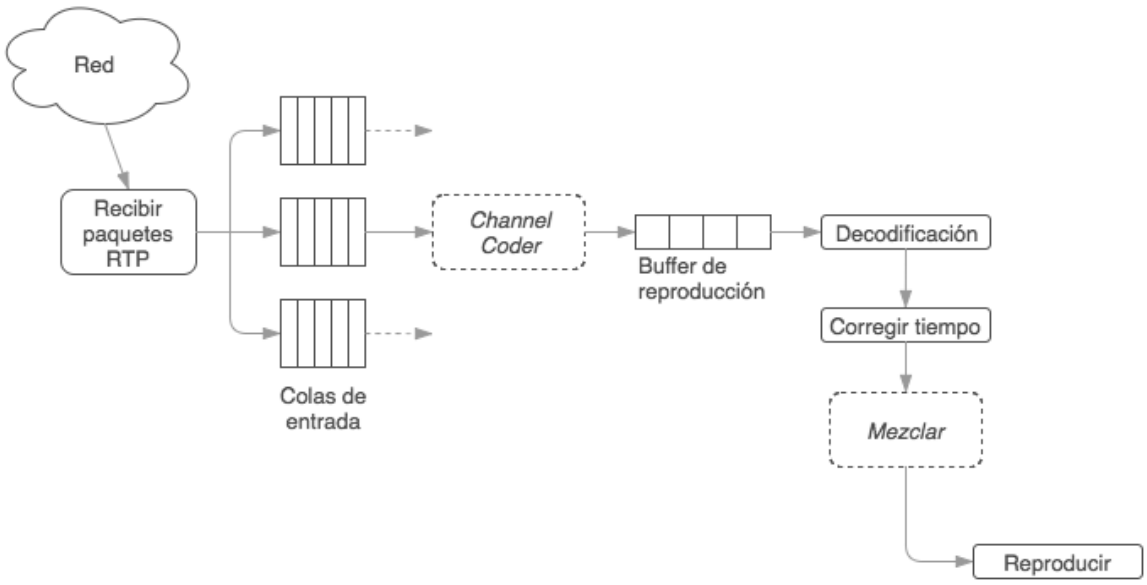


Figura C.2 – Comportamiento de un receptor RTP.

En primer lugar, los paquetes son recuperados desde la red y es validada su correctitud, para ser luego insertados en un *buffer* específico de cada flujo emisor. Los paquetes recogidos son pasados opcionalmente por un canal de codificación para recuperar daños. A continuación, los paquetes son insertados en un *buffer* de reproducción, donde son ordenados por su marca de tiempo para corregir cualquier desorden provocado por la red. Los paquetes permanecen en este *buffer* hasta que todos sus fragmentos son recibidos, y luego un tiempo adicional necesario con el fin de eliminar la variación del retardo entre paquetes. El cálculo de la cantidad de tiempo que un paquete debe permanecer en el *buffer* de reproducción, es una de las tareas más críticas en el diseño de una aplicación multimedia.

Una vez que el tiempo planificado de reproducción para un paquete es alcanzado, sus fragmentos son agrupados para formar un paquete completo, reparando cualquier fragmento dañado y/o generando los que no hayan sido recibidos. Luego el paquete es decodificado y presentado al usuario.

RTCP

Junto con la especificación de *RTP,* en el RFC 3550 se describe RTCP, el protocolo de control de RTP. Su finalidad es el intercambio de información relevante sobre la sesión, entre emisores, y entre emisores y receptores.

Los paquetes *RTCP* son enviados "fuera de banda" por un puerto distinto al utilizado para el flujo RTP (normalmente, el número de puerto se establece para que sea el empleado por RTP más uno). Estos paquetes no encapsulan fragmentos con contenido multimedia. En lugar de ello, anuncian estadísticas que pueden ser útiles para la aplicación que utiliza el protocolo. La especificación de *RTP* no establece qué debe hacerse con estas estadísticas, por lo que depende del desarrollador de la aplicación.

RTCP define varios tipos de paquetes que incluyen:

- **Informes del receptor**. En estos, se envía información sobre la recepción de un flujo RTP. Para ello se indica el identificador del flujo, la fracción de paquetes perdidos, el último número de secuencia recibido y una estimación de la fluctuación entre llegadas de los paquetes RTP recibidos, entre otros.
- **Informes del emisor.** Estos incluyen información sobre el flujo. Como ser su identificador, la cantidad de paquetes enviados, la marca de tiempo y número de secuencia del último paquete RTP enviado, entre otros.
- **Descripción del origen.** Estos paquetes contienen información acerca del origen, como la dirección de correo electrónico del emisor, el nombre del mismo y la aplicación que genera el flujo RTP. También incluye el identificador del flujo RTP asociado. Estos paquetes proporcionan una correspondencia entre el identificador del origen y el nombre del usuario/host.

Anexo D

Tecnologías para streaming Web

HTTP Live Streaming (HLS)

HTTP Live Streaming[1] es un protocolo de comunicación de *streaming* multimedia basado en el protocolo HTTP implementado por Apple[2]. El protocolo se encuentra como propuesta para ser estandarizado por la IETF[3].

HLS sigue una arquitectura cliente/servidor y plantea que cualquier presentación multimedia puede ser especificada por un archivo *Playlist*, compuesto por una lista ordenada de URLs y *tags* de información que representan una serie de segmentos multimedia. Para reproducir el flujo, un cliente deberá obtener el archivo *Playlist* y a continuación descargar y reproducir cada segmento multimedia. Dado que el archivo *Playlist* puede cambiar, el mismo debe ser actualizado periódicamente por el cliente.

Este protocolo está diseñado especialmente para flujos de audio/video almacenado o en vivo.

HTTP Long-polling

Con la técnica tradicional o *short polling*, un cliente envía peticiones al servidor y este responde a cada uno con la información que tiene disponible. Si no hay información, el servidor devuelve una respuesta vacía y el cliente espera por cierto tiempo antes de volver a hacer un pedido. El tiempo que un cliente espera entre un pedido y otro depende exclusivamente del tipo de información en cuestión, siendo la principal desventaja de este enfoque. Potencialmente muchos recursos de red y procesamiento son consumidos sin obtener la información buscada.

En contraste, la técnica de *long polling* intenta minimizar la latencia de los mensajes desde el servidor hacia el cliente y disminuir el uso injustificado de recursos de cómputo y red. Para ello, se plantea que el servidor responda a un pedido solo después de que cierto evento, estado o *timeout* haya ocurrido. Cuando el servidor da respuesta a un pedido de este tipo, el cliente automáticamente envía otro pedido, que quedará abierto en

---

[1] Apple. Inc, "HTTP Live Streaming (HLS) - Apple Developer". [En línea]. Disponible en: https://developer.apple.com/streaming/. [Consultado: 22-may-2016].

[2] Apple. Inc, "Apple". [En línea]. Disponible en: http://www.apple.com/. [Consultado: 22-may-2016].

[3] R. Pantos y W. May, "HTTP Live Streaming", abr. 2016.

el servidor a la espera de algún suceso que desencadene su respuesta.

Si bien este enfoque mejora la situación tradicional de *short polling*, presenta algunos problemas. Existe un importante *overhead* debido a que tanto los pedidos como las respuestas son paquetes HTTP completos. Por otro lado, se tiene una limitación a la latencia de los mensajes provocada por el hecho de que el servidor debe esperar un nuevo pedido luego de la respuesta. En el contexto de una conexión TCP, esto puede implicar la pérdida y retransmisión de paquetes, retrasando el flujo siguiente[4].

HTTP Streaming

HTTP *streaming*[4], es un mecanismo que permite mantener un petición abierta indefinidamente. Una petición nunca es terminada ni la conexión correspondiente cerrada, incluso luego de que el servidor haya respondido al cliente. Este mecanismo reduce significativamente la latencia debido a que clientes y servidores no necesitan abrir y cerrar nuevas conexiones por cada petición.

El ciclo de vida normal de una aplicación que utiliza *HTTP streaming* se puede describir en una serie de pasos:

1. El cliente realiza una petición inicial y espera por la respuesta.
2. El servidor demora la respuesta hasta tener información disponible, o hasta que se cumpla determinado suceso como un *timeout* o cambio de estado.
3. Siempre que exista nueva información, el servidor la enviará a los clientes como parte de la respuesta a la petición inicial.
4. El envío de la respuesta no termina la petición ni cierra la conexión. El servidor vuelve al paso número 3.

Este mecanismo se basa en la capacidad que tiene un servidor de enviar varias partes como respuesta a un mismo pedido. Dicha capacidad se encuentra disponible tanto en HTTP 1.0 como en HTTP 1.1.

Web Sockets

Los WebSockets permiten la creación de una conexión de datos persistente, bidireccional y orientada a mensajes entre cliente y servidor. Son la API más cercana a un *socket* "puro" en los navegadores web, aunque una conexión de WebSockets es mucho más que eso, ya que abstrae la complejidad bajo una API simple y minimal a la vez que provee un conjunto de servicios adicionales como:

- Negociación de los parámetros de conexión.

[4] G. Wilkins, S. Salsano, S. Loreto, y P. Saint-Andre, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", abr. 2011.

- Interoperabilidad con la infraestructura HTTP existente.
- Comunicación basada en mensajes y fragmentación eficiente de mensajes.
- Compresión selectiva.
- Negociación de subprotocolos y extensibilidad.

Esta tecnología provee una enorme reducción de tráfico y latencia en comparación con otras soluciones poco escalables para la comunicación en tiempo real, como ser el *polling* y *long polling*.

La Figura D.1, extraída del sitio oficial[5] de WebSockets, muestra una comparación de la latencia con la técnica de polling. En la misma se refleja la ventaja de no ser necesario realizar una petición por cada mensaje recibido, lo cual impacta en el tiempo de respuesta y los recursos necesarios en el servidor.

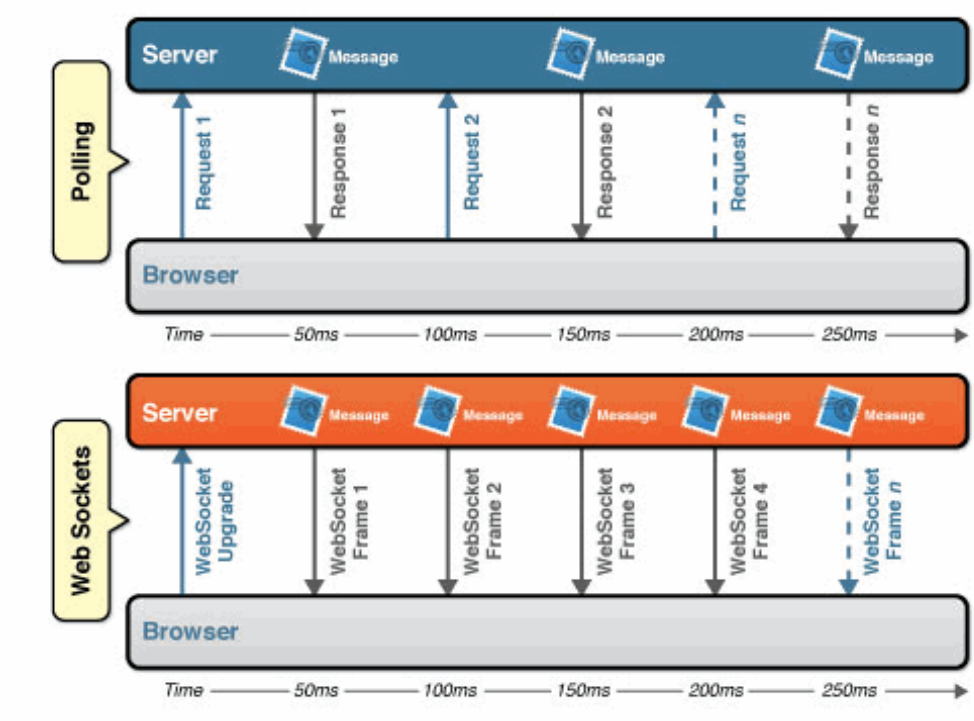


Figura D.1 - Comparación de latencia entre WebSockets y Polling.

En el mismo sitio, es posible constatar a través de una prueba simple una reducción de tráfico por overhead de cabezales de 500:1, que dependiendo del tamaño del cabezal HTTP podría alcanzar una relación de 1000:1. En la Figura D.2 se muestra una gráfica

[5] Kaazing, "HTML5 WebSocket - A Quantum Leap in Scalability for the Web". [En línea]. Disponible en: http://www.websocket.org/quantum.html. [Consultado: 22-may-2016].

extraída del mencionado sitio donde se refleja esta diferencia, comparando tres casos de uso donde 1000 clientes simultáneos hacen *polling* cada 1 segundo, o reciben un mensaje cada 1 segundo según corresponda al protocolo HTTP o WebSocket respectivamente.

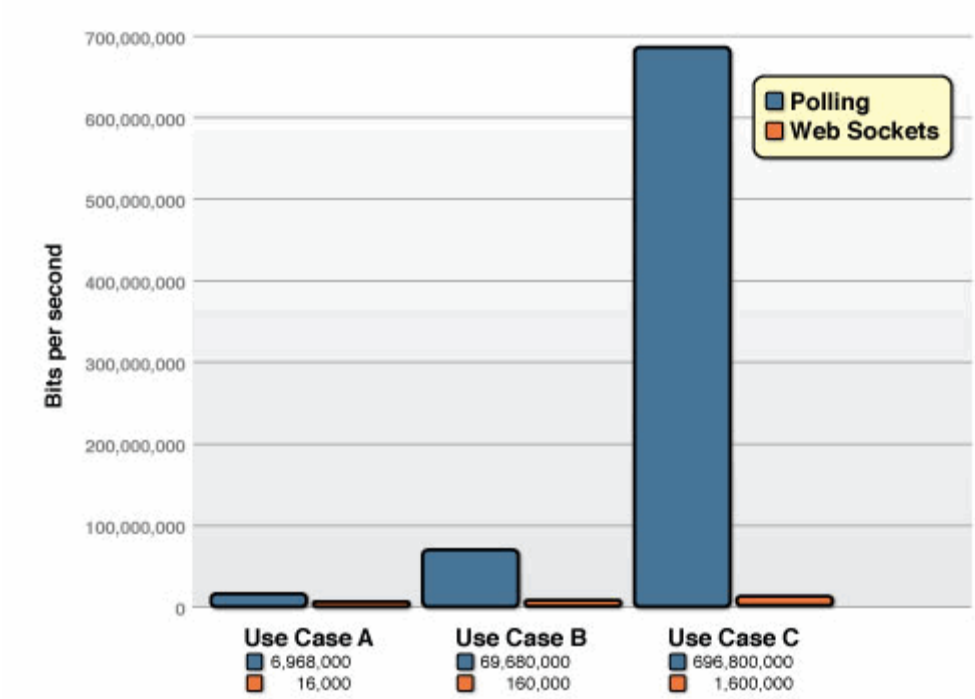


Figura D.2 - Resultados de prueba comparativa de overhead.

Protocolo WebSocket

El protocolo de WebSockets, fue originalmente creado por la *Web Hypertext Application Technology Working Group* (WHATWG[6]) e incluido en la especificación inicial de HTML5[7]. Más tarde, sería estandarizado por la IETF dando lugar al RFC 6455[8].

El mismo fue diseñado para funcionar sobre la infraestructura web existente. Como parte del principio sobre el que fue diseñado, el protocolo especifica que una conexión WebSocket comienza su ciclo de vida como una conexión HTTP, garantizando la compatibilidad con entornos pre-Websockets. Luego de un proceso de negociación (*handshake*) el protocolo puede cambiar (*upgrade*) de HTTP hacia WebSocket.

Para ello, el navegador envía un pedido al servidor indicando su intención de cambiar al protocolo WebSocket. El cliente expresa su deseo a través de un campo de cabecera de-

[6] "Web Hypertext Application Technology Working Group". [En línea]. Disponible en: https://whatwg.org/. [Consultado: 22-may-2016].

[7] "HTML5". [En línea]. Disponible en: https://www.w3.org/TR/html5/. [Consultado: 22-may-2016].

[8] I. Fette y A. Melnikov, "Rfc 6455: The websocket protocol", IETF, December, 2011.

nominado *Upgrade*. Si el servidor soporta el protocolo, entonces acepta la petición cambiando al header Upgrade. La Figura D.3 muestra un flujo capturado en con Wireshark en Sendero Streaming Server al momento de establecerse una conexión WebSocket.

```
GET /socket.io/?EIO=3&transport=websocket&sid=OQL3R5TOKS6fymn7AABR HTTP/1.1
Host: app.sendero.uy:8080
Connection: Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
Origin: http://web.sendero.uy
Sec-WebSocket-Version: 13
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
51.0.2704.103 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,es;q=0.6,gl;q=0.4,pt;q=0.2
Cookie: io=OQL3R5TOKS6fymn7AABR
Sec-WebSocket-Key: /5Tkn/Xmfzwvon5OV2fr0g==
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: d8Yu83TQ04E5eicyUQprt/DklZY=
Sec-WebSocket-Extensions: permessage-deflate
```

Figura D.3 - Captura de paquete que muestra cabecera *Upgrade*.

En este punto, la conexión HTTP es reemplazada por un conexión WebSocket sobre el mismo puerto TCP/IP. Luego, el cabezal WebSockets luce como se ilustra en la Figura D.4.

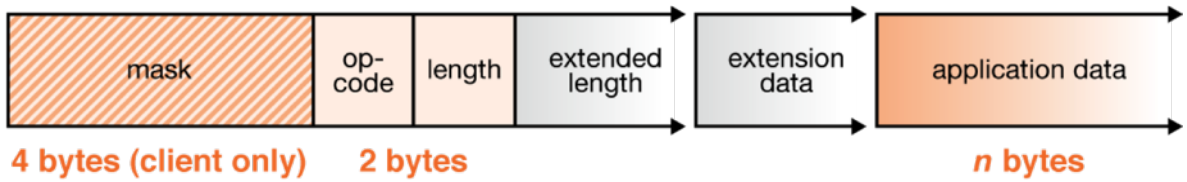


Figura D.4 - Formato de cabecera WebSocket.

Compresión

Luego de varias versiones sin compresión, fue desarrollada una extensión del protocolo capaz de realizar compresión por mensaje de manera selectiva. Este mecanismo incluye un proceso de negociación bi-direccional que determina cómo serán comprimidos los mensajes desde el servidor hacia el cliente y viceversa.

A su vez, permite a clientes y servidores negociar el algoritmo y parámetros de la compresión. Luego de establecidos estos parámetros, cada paquete es individualmente evaluado y en caso de resultar conveniente su carga útil será comprimida. Dicho mecanismo puede ser activado o desactivado, y por defecto utiliza una combinación del algoritmo de deflación (*deflate*) LZ77 y codificación de Huffman.

Compatibilidad

Según el sitio de W3C[9] el protocolo de WebSockets se encuentra soportado por los navegadores y versiones mostrados en la Figura D.5.

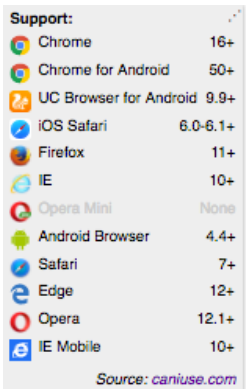


Figura D.5 - Compatibilidad de WebSockets en navegadores.

Implementaciones y librerías

Existen múltiples implementaciones del protocolo, tanto para navegadores Web como para servidores. Entre ellas se destacan las mostradas en la Tabla D.1.

Lenguaje	**Librería**
Node.js	Socket.IO, WebSocket-Node, ws
Java	Jetty
Ruby	EventMachine
Python	pywebsocket
Erlang	Shirasu
C++	libwebsockets
.NET	SuperWebSoket
Swift	Starscream

Tabla D.1 - Implementaciones de WebSockets en múltiples lenguajes.

[9] W3C, "The WebSocket API". [En línea]. Disponible en: https://w3c.github.io/websockets/.

WebRTC

WebRTC[10] consiste en una tecnología, que se encuentra siendo desarrollada por la *World Wide Web Consortium (W3C)* y la IETF, cuyo objetivo es proveer herramientas para la comunicación de audio y video en tiempo real entre navegadores web de forma nativa, sin la necesidad de configurar librerías ni complementos.

El desarrollo de una tecnología como esta se ha visto impulsado por la creciente cantidad de aplicaciones que requieren de comunicación en tiempo real, y satisfacen esta necesidad mediante la instalación de *plugins*. Por ejemplo, *Skype*[11], *Facebook*[12] y *Google Hangouts*[13].

Para cumplir con la consigna de proveer mecanismos nativos de comunicación en tiempo real a los navegadores, WebRTC se apoya en un conjunto de protocolos (muchos ya mencionados en este documento). La Tabla D.2 lista los más importantes, con el objetivo de destacar el desafío que ofrece tal meta[14].

Protocolo	Uso	Especificación
HTTP	Hyper-Text Transfer Protocol	RFC 2616
SRTP	Secure Real-time Transport Protocol	RFC 3711
SDP	Session Description Protocol	RFC 4566
ICE	Interactive Connectivity Establishment	RFC 5245
STUN	Session Traversal Utilities for NAT	RFC 5389
TURN	Traversal Using Relays around NAT	RFC 5766
TLS	Transport Layer Security	RFC 5246
TCP	Transmission Control Protocol	RFC 793
DTLS	Datagram Transport Layer Security	RFC 4347
UDP	User Datagram Protocol	RFC 768
SCTP	Stream Control Transport Protocol	RFC 2960
IP	Internet Protocol	RFC 791, RFC 2460

Tabla D.2 - Protocolos involucrados en WebRTC.

[10] "WebRTC Home". [En línea]. Disponible en: https://webrtc.org/. [Consultado: 22-may-2016].

[11] "Skype". [En línea]. Disponible en: https://www.skype.com. [Consultado: 21-may-2016].

[12] "Facebook". [En línea]. Disponible en: https://www.facebook.com/. [Consultado: 22-may-2016].

[13] "Google Hangouts". [En línea]. Disponible en: https://hangouts.google.com. [Consultado: 21-may-2016].

[14] A. B. Johnston y D. C. Burnett, WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-time Web. 2013.

Anexo E

Interfaces de Hardware

Dado que el objetivo de los módulos M2M es acoplarse a otros dispositivos ya existentes, lo proveedores de este tipo de tecnologías ofrecen una diversa gama de interfaces de *hardware* para permitir a sus módulos interactuar con otros dispositivos. A continuación se listan la mayoría de los tipos de interfaces de *hardware* disponibles hasta el momento para módulos M2M[1].

- **Alimentación.** Evidentemente, todo módulo requiere de una interfaz de alimentación de energía. El rango de alimentación soportada por la mayoría de estos dispositivos varía entre los 3v y 5v.
- **USB (*Universal Serial Bus*).** Las interfaces USB son de las más comunes en módulos M2M. Su utilización está ligada a tecnologías con altas tasas de transmisión de datos y en particular, a técnicas de *debugging*. Existen múltiples versiones de esta interfaz que ofrecen distintas características en cuanto a velocidad de transmisión y forma física de los conectores[2].
- **UART (*Universal Asynchronous Receiver/ Transmitter*).** Las interfaces UART son utilizadas principalmente con fines de comunicación serial y/o *debugging*. Si bien pueden ser encontradas en la mayoría de los módulos M2M, están siendo reemplazadas por las interfaces USB, entre otras razones, por sus tasas de transferencia inferiores.
- **Antena.** Todos los módulos inalámbricos deben proveer al menos una interface para la conexión de la antena. Su posición y diseño es usualmente responsabilidad del desarrollador que utiliza el módulo, teniendo en consideración los beneficios de performance, tamaño e interferencia. Actualmente, múltiples tecnologías de acceso requieren de más de una antena para llevar a cabo técnicas de MIMO (*multiple-input, multiple-output*).
- **GPIO (*General-Purpose Input/Output Port*).** Las interfaces GPIO son utilizadas en los módulos M2M principalmente para el control de periféricos externos y la recepción de información de entrada.
- **SPI (*Serial Peripheral Interface*).** SPI es un tipo de conexión sincrónica serial que opera en modo *full-duplex* donde los dispositivos participantes de la conexión pueden ser configurados como maestros o esclavos. Este tipo de interfaces

[1] D. Boswarthick, O. Elloumi, y O. Hersent, M2M Communications: A Systems Approach. John Wiley & Sons, 2012.

[2] "USB.org - Welcome". [En línea]. Disponible en: http://www.usb.org/. [Consultado: 23-may-2016].

pueden soportar frecuencias de transmisión de hasta 70 MHz y son utilizadas principalmente para el control de dispositivos periféricos y el control de pantallas externas.

- **I2C (*Inter-Integrated Circuit Bus*).** De manera similar a SPI, I2C ofrece un tipo de conexión sincrónica serial cuya tasa de transferencia puede alcanzar los 400 Kbps. Este tipo de interfaz es comúnmente utilizada para control de pantallas externas, leer datos de sensores o periféricos de almacenamiento.
- **PCM (*Pulse Code Modulation*).** Este tipo de interfaces se utiliza especialmente para la transferencia de datos digitales de audio.
- **PWM (*Pulse Width Modulation*).** Este tipo de interfaces proveen a los módulos M2M de una salida pseudo-analógica y son principalmente utilizadas para el control de luces y *buzzers*.

Anexo F

Tecnologías de red de área local

Wi-Fi (IEEE 802.11)

Wi-Fi (*Wireless Fidelity*) incluye los estándares IEEE 802.11a/b/g/n/ac[1] para redes de área local inalámbricas (WLAN). La Tabla F.1 resume las características principales de estos estándares.

Protocolo	Rango de frecuencias	Velocidad de datos
802.11a	5 GHz	54 Mbps
802.11b	2.4 GHz	11 Mbps
802.11g	2.4 GHz	54 Mbps
802.11n	2.4 GHz y 5 GHz	600 Mbps (teórico)
802.11ac	5 GHz	1.3 Gbps

Tabla F.1 - Características de estándares IEEE 802.11

El componente fundamental de la arquitectura 802.11 es el BSS *(Basic Service Set)*. Un BSS contiene una o más estaciones inalámbricas y una estación base central, conocida con el nombre de *Access Point (*AP*)*. Los puntos de acceso se conectan a su vez a un dispositivo de interconexión (como un *router*), que lleva a Internet. Normalmente, los AP y *routers* se integran en un mismo dispositivo físico en las redes domésticas[2].

Bluetooth (IEEE 802.15.1)

Bluetooth, también conocido como IEEE 802.15.1[3] es un estándar de comunicación inalámbrica diseñado para distancias cortas y dispositivos económicos con el fin de reemplazar los cables en los periféricos de las computadoras. Este estándar pertenece a la categoría de redes inalámbricas de área personal (WPAN), alcanzando distancias en el orden de los 10 metros.

Bluetooth opera en la banda de frecuencias de 2.4 GHz y permite dos topologías de red: *Piconet* y *Scatternet*.

---

[1] Intel, "Different WiFi Protocols and Data Rates", 14-mar-2016. [En línea]. Disponible en: http://www.intel.com/content/www/us/en/support/network-and-i-o/wireless-networking/000005725.html. [Consultado: 29-may-2016].

[2] J. F. Kurose y K. W. Ross, Redes de Computadoras: Un enfoque descendente, 5ta ed. Ribera del Loira, 28. 28042 Madrid (España): PEARSON EDUCACIÓN, S. A., 2010.

[3] Institute of Electrical and Electronics Engineers, "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)", 802.15.1-2005, jun. 2005.

Una *piconet* es una WPAN formada por un dispositivo Bluetooth que oficia como maestro y uno o varios que se comportan como esclavos. Todos los dispositivos conectados a una *piconet* se encuentran sincronizados por el reloj del maestro. Los esclavos se comunican con el maestro en modo punto-a-punto. Los maestros pueden transmitir en modo punto-a-punto como también en *broadcast*. Los esclavos en un *piconet* pueden estar activos o en modo *parked (o standby)* con el fin de reducir su consumo energético.

Una *scatternet* es una colección de *piconets* que superponen su operativa en tiempo y espacio. Dos *piconets* pueden conectarse para formar una *scatternet.* Un dispositivo Bluetooth puede participar en múltiples *piconets* al mismo tiempo, permitiendo que la información se mueva más allá del alcance de la *piconet* donde se originó. Un dispositivo en una *scatternet* puede ser esclavo en varias *piconets* pero maestro en solo una de estas.

WiMAX (IEEE 802.16)

WiMAX *(World Interoperability for Microwave Access)*, es una familia de estándares IEEE 802.16 cuyo fin es proveer comunicación de datos inalámbricos a un gran número de usuarios en un área extensa, con velocidades de transmisión similares a las redes DSL. La arquitectura 802.16 está basada en la noción de una estación base que sirve de modo centralizado a un número potencialmente grande de clientes asociados con dicha estación base.

ZigBee (IEEE802.15.4)

ZigBee, IEEE 802.15.4[4] define una especificación para redes de área personal con baja tasa de transmisión (LR-WPAN *Low Rate Wireless Personal Area Network*) cuyo foco se centra en el uso dispositivos con consumo mínimo de energía. El mismo provee una red *mesh* auto-organizada, *multi-hop* y confiable enfocada en el ahorro de batería de sus nodos. Dos tipos de dispositivo pueden participar en una LR-WPAN: *full-function devices* (FFD) y *reduced-function devices* (RDF). Los primeros pueden operar en tres modos, coordinador PAN, coordinador o dispositivo. Un FFD puede comunicarse con RFDs u otros FFDs, mientras que un RFD solo puede comunicarse con un FFD.

En cuanto a las bandas de operación, ZigBee opera en las bandas de 868/915 MHz y 2.4 GHz pudiendo transmitir a 40 Kbps y 250 Kbps respectivamente. Su rango de alcance va de 10 a 100 m[5].

---

[4] Institute of Electrical and Electronics Engineers, "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)", 802.15.4, sep. 2006.

[5] J.-S. Lee, L. Jin-Shyan, S. Yu-Wei, y S. Chung-Chou, "A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi", presentado en IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society, 2007.

Anexo G

Bondibar

En el presente anexo se detallan algunas de las características principales del LED *driver* de Sendero.

Los dispositivos Bondibar (ver Figura G.1) son los encargados de realizar el nexo entre el módulo Sendero Wireless Dongle (que acopla al microcontrolador ESP-12E) y las luces LED RGB. Los mismos, proveen una interfaz USB a través de la cual es posible transferir la información de color como una tira de bytes RGB utilizando el protocolo SPI.

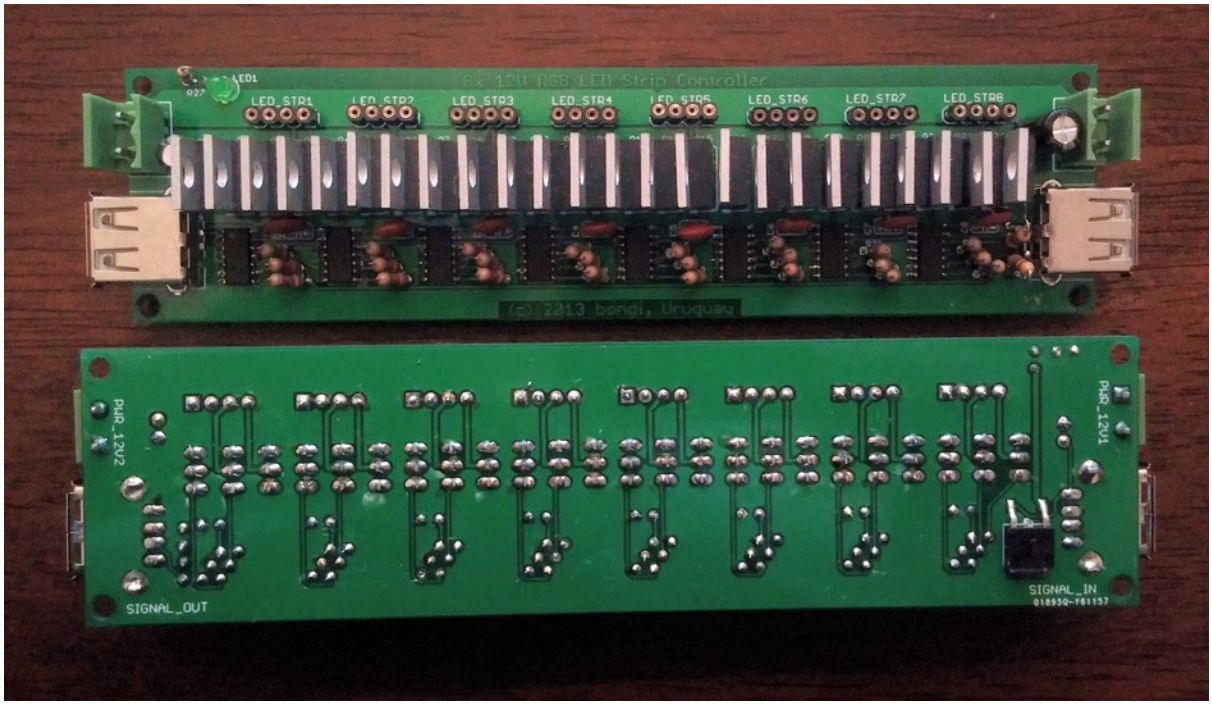


Figura G.1 – Bondibar

Cada Bondibar es capaz de manejar 8 tiras LED, pero adicionalmente, es posible conectar una serie de Bondibars en *daisy-chain*, permitiendo que un solo microcontrolador utilice la cadena de Bondibars a través de la misma interfaz, incrementando así, la cantidad de tiras LED manejadas.

La Figura G.2 muestra el esquema EAGLE[1] del re-diseño del controlador Bondibar.

[1] "CadSoft EAGLE PCB Design Software|Support, Tutorials, Shop". [En línea]. Disponible en: http://www.cadsoftusa.com/. [Consultado: 31-may-2016].

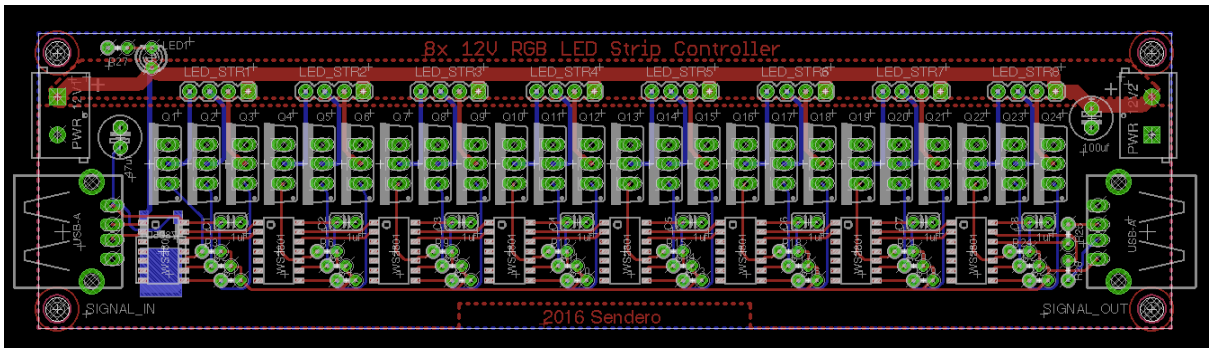


Figura G.2 - Esquema EAGLE de la nueva versión de Bondibar

Alimentación

El controlador Bondibar se alimenta a través de una bornera doble (ver parte superior izquierda de la Figura G.2). Como suministro de energía se utiliza una fuente de 12V.

El LED *driver* permite alimentar hasta 72W por píxel (cantidad necesaria para una tira standard de LEDs SMD 5050 de alta potencia, ver Figura G.3). Con el fin de lograr esta cantidad de potencia, se utilizan tres transistores IRF520 por píxel (uno por cada canal de color).

Figura G.3 - Tira LED RGB SMD 5050.

La circuitería de esta placa también permite encadenar la alimentación fácilmente. Simplemente se utiliza la bornera de salida, como se muestra en la Figura G.4.

Manejo de los LED

Para generar la señal PWM (*pulse-width modulation*, modulación por ancho de pulso), característica clave para el encedido de los LED RGB[2], se utilizan los circuitos integrados WorldSemi WS2801[3]. Estos contienen 3 canales de salida PWM, donde cada uno de ellos es utilizado para controlar una componente de color del píxel (rojo, verde, azul). Estos chips tienen la capacidad de re-transmitir la señal SPI de entrada, permitiendo que los datos ingresen de forma serial en el Bondibar, encargando a los WS2801 la propagación de los mismos. Esta capacidad hace posible la conexión de múltiples Bondibars en cadenas (como fue mencionado anteriormente) a través de sus interfaces USB de entrada y salida. Finalmente, para la conexión de las tiras LED se utiliza un conector estándar de 4 pines.

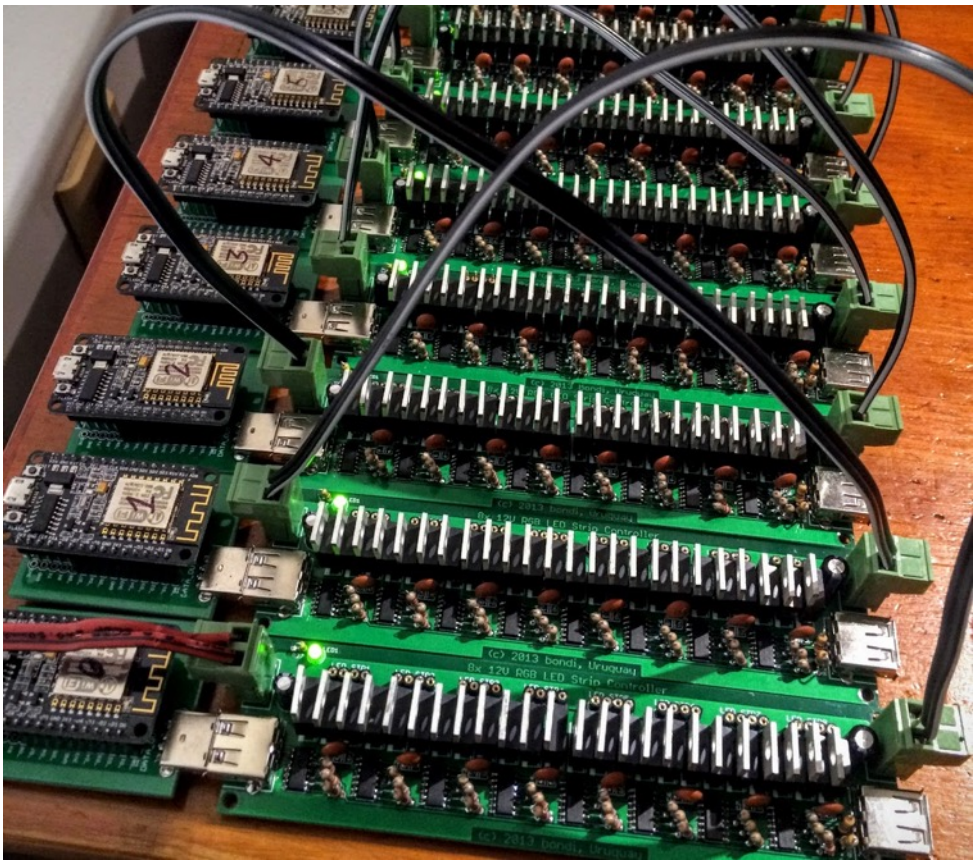


Figura G.4 - Varias Bondibar compartiendo la alimentación.

[2] DigiKey, "How To Dim a LED". [En línea]. Disponible en: http://www.digikey.com/en/articles/techzone/2010/apr/how-to-dim-an-led. [Consultado: 07-jul-2016].

[3] World-Semi, "WS2801 Datasheet". [En línea]. Disponible en: http://www.world-semi.com/en/Driver/Lighting_LED_driver_chip/WS2801/. [Consultado: 17-may-2016].

Comunicación Microcontrolador - Bondibar

La comunicación entre estos componentes se da a través de una interfaz USB A hembra. Esta interfaz alimenta al microcontrolador gracias al regulador de voltaje LM7805[4] que el LED *driver* tiene acoplado (ver parte inferior derecha de la Figura G.1). Éste produce una fuente alimentación de 5V y aproximadamente 1A de intensidad de corriente, lo cual se adapta al estándar de suministro de energía de USB[5], permitiendo alimentar diversos tipos de microcontroladores. Las pistas DATA+ y DATA- de la interfaz están directamente conectadas al primero de los WS2801, específicamente a los pines CKI (*Clock Input*) y SDI (*Serial Data Input*), respectivamente (ver Figura G.5). Por este motivo, la pista DATA+ debe conectarse al SCLK (*SPI clock*) del microcontrolador, mientras que DATA- debe ser acoplado al pin SPI MOSI (*Master Output Slave Input*). Estos últimos ruteos de pistas son llevados a cabo por Sendero Wireless Dongle.

Los pines CKO (*Clock Output*) y SDO (*Serial Data Output*) son conectados a los pines CKI y SDI, respectivamente, del siguiente chip WS2801 en la cadena.

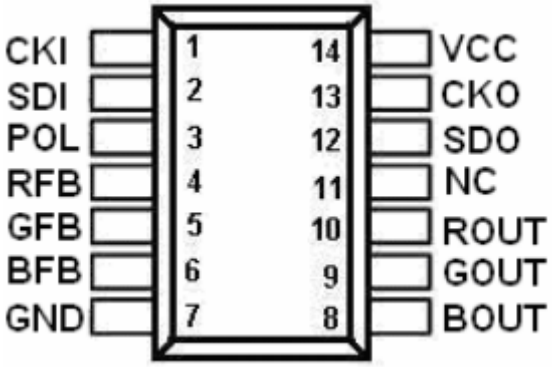


Figura G.5 - Esquema del WS2801 de FairChild Semiconductor Corporation.

Para obtener un mayor detalle de la construcción del Bondibar y su funcionamiento, se recomienda consultar la tesis de Maestría de Christian Clark: “New Media And Impressionism”[6].

[4] Fairchild, “LM78XX/LM78XXA 3-Terminal 1A Positive Voltage Regulator”. Fairchild Semiconductor Corporation, 2006.

[5] Kollman, Robert and Betten, John. “Powering Electronics From the USB Port.” [Internet] Available: http://www.ti.com/lit/an/slyt118/slyt118.pdf

[6] C. Clark, “New Media and Impressionism”, Masters Thesis, Facultad de Ingeniería, Universidad de la República, 2015.

Anexo H

ESP-12E DevKit

NodeMCU ESP-12E DevKit (ver Figura H.1) es un kit de desarrollo diseñado y desarrollado por Shenzhen Doctors of Intelligence & Technology[1]. El mismo se basa en el SoC (*System on Chip*) de bajo costo ESP8266 ESP-12E [2] (ver Figura H.2) y viene precargado con el firmware NodeMCU [3], que permite rápidos desarrollos orientados a Internet of Things (IoT) en el lenguaje interpretado Lua [4]. El SoC embebe un microcontrolador (MCU) Tensilica L106 de 32-bits de direccionamiento, opera a una frecuencia de reloj de 80MHz u 160MHz y adicionalmente posee una antena Wi-Fi integrada.

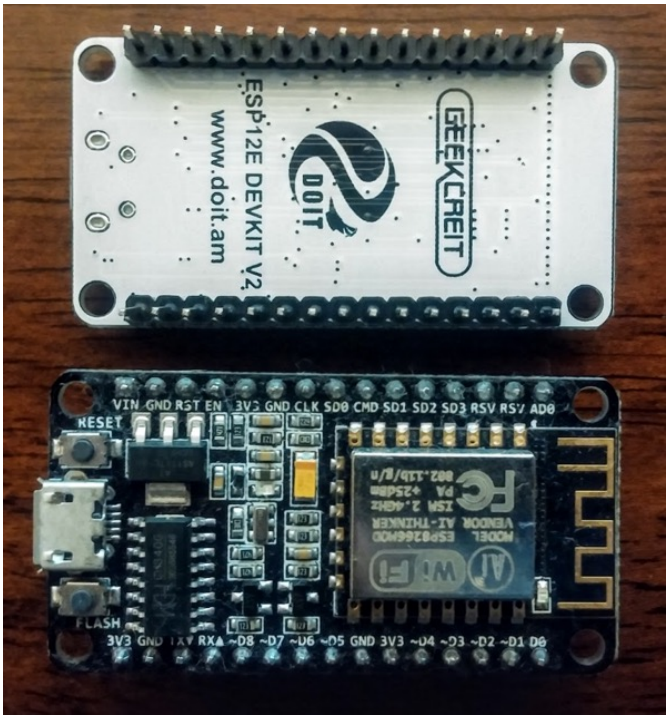


Figura H.1 - NodeMCU ESP-12E *DevKit*

[1] "Doctors of Intelligence & Technology." 2016. [Accessed: 03-Jul-2016]. http://www.doit.am.

[2] "ESP-12E WiFi Module." AI-Thinker, 2015.

[3] "NodeMCU Documentation." [Online]. Available: https://nodemcu.readthedocs.io. [Accessed: 07-May-2016].

[4] R. Ierusalimschy, W. Celes, and de F. Luiz Henrique, "The Programming Language Lua." [Online]. Available: https://www.lua.org/. [Accessed: 07-May-2016].

Figura H.2 - ESP8266 ESP-12E Wi-Fi SoC

El *devkit* facilita la conectividad del microcontrolador añadiendo *pin headers* de 2.54 mm para fácil prototipado, además de una interfaz micro USB que ayuda en la programación y depuración del dispositivo. Permite utilizar herramientas como Arduino IDE [5] o PlatformIO [6] que, en conjunto con las librerías basadas en Arduino, resultan de gran ayuda para el desarrollador [7].

En la figura H.3 se muestra el *pinout* del NodeMCU *devkit*. Las principales características que presenta el módulo ESP8266 ESP-12E se enumeran a continuación:

- Protocolos IEEE 802.11 b/g/n
- Wi-Fi Direct (P2P)
- Pila de protocolos TCP/IP. Permite hasta 5 clientes TCP conectados.
- Señal de salida de +19.5dBm en modo 802.11b
- Modos de ahorro de energía y *wake up* para transmisión de datos.
- SDIO 2.0, (H) SPI, UART, I2C, I2S, IRDA, PWM, GPIO
- STBC, 1x1 MIMO, 2x1 MIMO

[5] "Arduino website." [Online]. Available: https://www.arduino.cc/. [Accessed: 07-May-2016].

[6] "PlatformIO." [Online]. Available: http://platformio.org/. [Accessed: 07-May-2016].

[7] "Arduino core for ESP8266 WiFi chip." [Online]. Available: https://github.com/esp8266/Arduino. [Accessed: 07-May-2016].

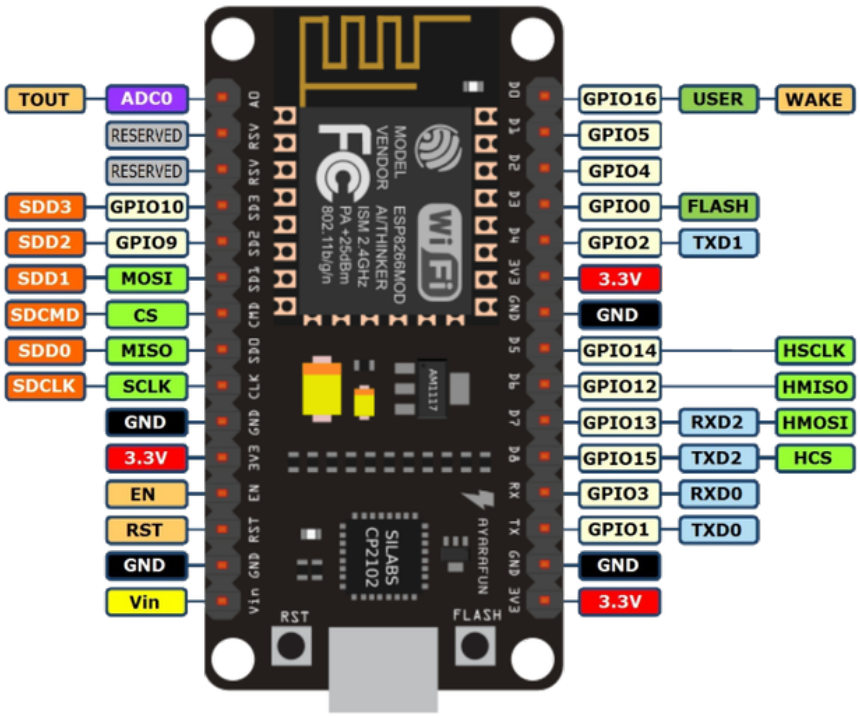


Figura H.3 - *Pinout* de NodeMCU *devkit*

Existe vasta información en la Web acerca de los módulos ESP8266 debido a la popularidad que han ganado gracias a su bajo costo y su buena capacidad de procesamiento. Para más información recomendamos visitar el foro de la comunidad ESP8266[8], y para obtener más detalles técnicos, verificar sus especificaciones[9]. Una descripción más profunda del NodeMCU devkit se encuentra disponible en GitHub[10].

[8] "ESP8266 Community Forum." [Online]. Available: http://www.esp8266.com/. [Accessed: 07-May-2016].

[9] "ESP-12E WiFi Module." AI-Thinker, 2015.

[10] "NodeMCU DEVKIT V1.0." [Online]. Available: https://github.com/nodemcu/nodemcu-devkit-v1.0. [Accessed: 07-May-2016].

Anexo I

BOM Bondibar y Sendero Wireless Dongle

BOM, por sus siglas en inglés de *Bill Of Materials,* es una lista de materiales, subcomponentes o partes y las respectivas cantidades de cada uno que son necesarias para la construcción de un determinado producto final. Como parte del proyecto Sendero, se construyeron nuevos controladores de LEDs Bondibar y por cada uno de estos, un adaptador Sendero Wireless Dongle. Además del diseño e impresión del circuito PCB, se crearon BOMs que sirvieron para organizar y comprar en distintas tiendas los componentes necesarios para la creación de estos productos. La Tabla I.1 muestra el BOM del controlador de LEDs Bondibar junto a el sitio donde se adquirió cada componente. De manera análoga para Sendero Wireless Dongle en la Tabla I.2.

Componente	**nº/placa**	**Tienda**
PCB	1	Seedstudio[1] (Hong Kong)
Condensador 47µf*	1	-
Condensador 100µf*	1	-
LED 3mm verde	1	Eneka[2] (Uruguay)
Regulador 7805DT - TO252	1	Mouser Electronics[3] (EEUU)
Resistencia 1k Ohm 5%	1	Eneka (Uruguay)
Conector USB-A Hembra*	2	-
ScrewTerminal 5mm*	2	-
ScrewTerminal 5mm Macho*	2	-
Resistencia 2.2K Ohm*	24	-
Capacitor de cerámica 0.1µf*	8	-
Conector LED hembra de 4 pines	8	Amazon[4] (EEUU)
Transistor IRF520 - TO220*	24	-
Manejador LED WS2801*	8	-
Resistencia 47 Ohms 5%	2	Eneka (Uruguay)
Conector LED macho de 4 pines	8	Amazon (EEUU)

Tabla I.1 - BOM Bondibar

[1] "Seedstudio". [En línea]. Disponible en: http://www.seeedstudio.com/. [Consultado: 07-abr-2016].

[2] "Eneka". [En línea]. Disponible en: http://www.eneka.com.uy/. [Consultado: 07-abr-2016].

[3] "Mouser Electronics". [En línea]. Disponible en: http://uy.mouser.com/. [Consultado: 07-abr-2016].

[4] "Amazon". [En línea]. Disponible en: https://www.amazon.com/. [Consultado: 07-abr-2016].

* Componentes que no fue necesario comprar gracias a que se contaba con sobrantes de desarrollos previos.

Componente	**nº/placa**	**Tienda**
PCB	1	Seedstudio (Hong Kong)
Conector USB-A PRT-00437	1	HobbyTronics[5] (Reino Unido)
Header hembra 15 pines	2	Amazon (EEUU)

Tabla I.2 - BOM Sendero Wireless Dongle.

[5] "HobbyTronics". [En línea]. Disponible en: http://www.hobbytronics.co.uk/. [Consultado: 07-abr-2016].

Anexo J

Configuración de la Obra

Para configurar una obra en Sendero, es necesario editar un conjunto de atributos en un archivo en formato XML llamado serverConf.xml ubicado en el subdirectorio *bin/data/* del proyecto de Sendero Server. Los atributos configurables son:

- Cantidad total de píxeles
- Nombre de la instalación
- Dispositivos Art-Net (mBed, Sendero Middleware)
- Configuración del modelo 3D de la obra
- Tipos de LED utilizados (para calibrar diferencias de color entre ellos)
- Implementaciones de Sendero Client utilizadas.
- Configuración del Streaming Server.

Las obras de Sendero tienen un modelo 3D asociado que se replica en sus aplicaciones. En la Figura J.1 se observa una captura de Sendero Server corriendo con el set de configuraciones de la obra Barcelona.

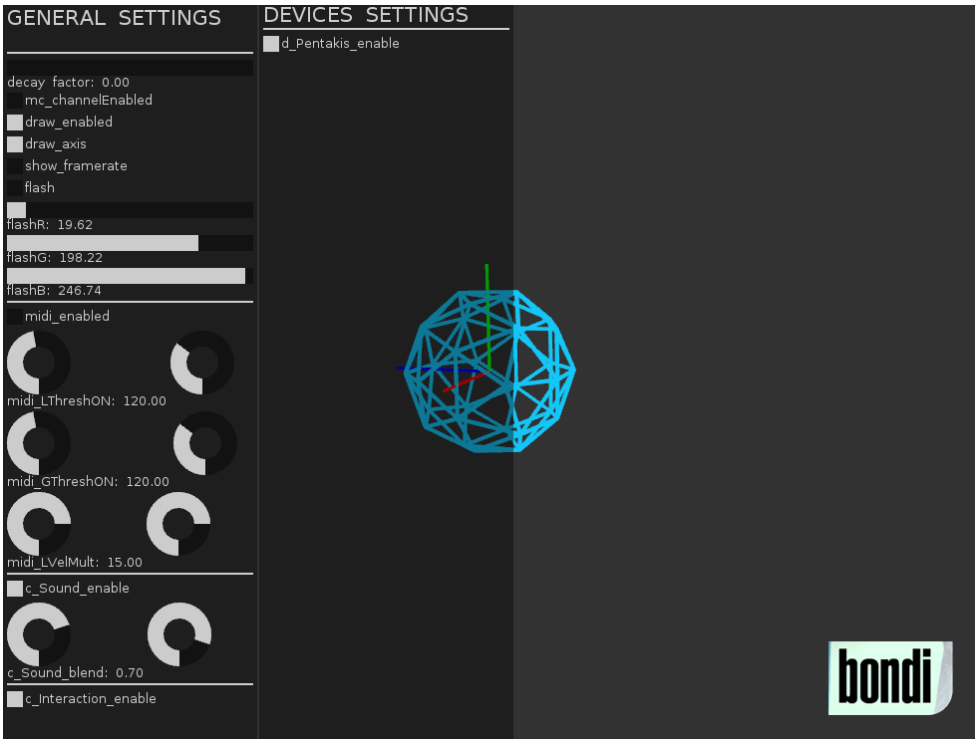


Figura J.1 - Captura de pantalla de Sendero Server con la configuración de Barcelona

A continuación se muestra la configuración de Sendero Server utilizada para el caso mencionado. El mismo tiene configuradas dos instancias de Sendero Client: el cliente de Sonido y el de Interacción. Además se configura el *Streaming Server* con compresión habilitada.

```
<?xml version="1.0" ?>
<!-- Son necesarios 90 píxels  para Barcelona (1 por cada arista del dodecaedro Pentakis) -->
<Configuration pixelQuantity='90' installationName='Pentakis'>
 <Devices>
   <!-- Puede ser ejecutado tanto con un microcontrolador mBed como con Sendero Middleware, puesto
que este emula al primero -->
   <Device id='1' name='Pentakis' type='mBed' ipAddress='localhost'
     UDPPort='7777' enabled='1' />
 </Devices>
 <Meshes>
   <!-- Archivos de mallas poligonales se colocan por fuera -->
   <Mesh name='cylinderShort' path='meshes/cylinderShort100.obj' />
   <Mesh name='cylinderLong' path='meshes/cylinderLong100.obj' />
 </Meshes>
 <LedTypes>
     <LedType id="0">
       <ColorPair r0="255" g0="0" b0="0" r1="255" g1="0" b1="0"/>
       <ColorPair r0="0" g0="255" b0="0" r1="0" g1="255" b1="0"/>
       <ColorPair r0="0" g0="0" b0="255" r1="0" g1="0" b1="255"/>
     </LedType>
     <LedType id="1">
       <ColorPair r0="255" g0="0" b0="0" r1="255" g1="0" b1="0"/>
       <ColorPair r0="0" g0="255" b0="0" r1="0" g1="200" b1="0"/>
       <ColorPair r0="0" g0="0" b0="255" r1="0" g1="0" b1="255"/>
     </LedType>
 </LedTypes>
 <FrameConf>
   <!-- se configura el color inicial de cada píxel, el dispositivo Art-Net que lo conecta y el
tipo de led que tiene conectado -->
   <Pixel id='0' r='255' g='255' b='255' a='255' device='1' ledType="0">
     <!-- mesh: la malla poligonal utilizada para el modelo 3D del píxel (definida en el atributo
'Meshes')  -->
     <Render mesh='cylinderShort'>
       <Front x='-0.0998540000000001' y='0.229397' z='-0.0259085' />
       <Up x='0.546030364592086' y='0.488412566772085' z='2.22000739347465' />
       <Position x='55.8648' y='22.9397' z='-43.28855' />
     </Render>
   </Pixel>
   <Pixel id='1' r='255' g='255' b='255' a='255' device='1' ledType="1">
     <Render mesh='cylinderLong'>
       <Front x='0.141775' y='-0.087622' z='0.229397' />
       <Up x='0.423934187608445' y='0.199571320207026' z='-0.185775887343807' />
       <Position x='60.0569' y='37.1172' z='-22.9397' />
     </Render>
   </Pixel>
   <Pixel id='89' r='255' g='255' b='255' a='255' device='1' ledType="0">
     <Render mesh='cylinderShort'>
       <Front x='0' y='-0.1676835' z='-0.187476' />
       <Up x='0.53360247055402' y='0.708328084653321' z='-0.633547400109695' />
       <Position x='0' y='-57.46605' z='47.1026' />
     </Render>
   </Pixel>
 </FrameConf>
 <!-- Los Sendero Client utilizados y puertos utilizados -->
 <ClientsProxys>
   <Client TCPPort='5002' UDPPort='5003' id='1' name='Sound'
   enabled='1' blendFactor='0.7' protocolType='0' />
   <Client TCPPort='5004' UDPPort='5005' id='2' name='Interaction'
   enabled='1' blendFactor='0.5' protocolType='0' />
 </ClientsProxys>
 <!-- Configuración de direccionamiento del Sendero Server y compresión con LZ4 -->
 <StreamServerConfig enabled='1' ipAddress='app.sendero.uy' port='8080' compression='1'/>
</Configuration
```

Anexo K

Presentaciones

Durante el desarrollo del proyecto, Sendero fue presentado en dos eventos de índole público, el festival de Arte y Ciencia Equinoccio[1], llevado a cabo desde el 22 al 26 de septiembre de 2015 en el Instituto Nacional de Artes Escénicas[2] y en el evento anual Ingeniería de Muestra[3], desarrollado en la Facultad de Ingeniería de la Universidad de la República los días jueves 22, viernes 23 y sábado 24 de octubre de 2015.

La presentación de Sendero en estos eventos aceleró la construcción del subsistema web, del cual se presentaron versiones en desarrollo de los componentes Sendero Web, Sendero Cardboard, Sendero Streaming Server y Sendero Interaction Server. Además para Ingeniería de Muestra se construyó una versión reducida de la instalación interactiva Barcelona, capaz de reflejar la interacción web de los usuarios en los píxeles de una cara de la escultura. Estas instancias fueron de gran utilidad para evaluar el diseño de la implementación propuesta y recibir retroalimentación de espectadores reales.

Las figuras K.1, K.2 y K.3 muestran imágenes de estas instancias.

[1] "Inicio - equinoccio". [En línea]. Disponible en: http://equinoccio.uy/. [Consultado: 29-may-2016].

[2] "INAE | Instituto Nacional de Artes Escénicas". [En línea]. Disponible en: http://www.inae.gub.uy/. [Consultado: 03-jun-2016].

[3] "Ingeniería de Muestra 2015", Ingeniería deMuestra 2015. [En línea]. Disponible en: https://www.fing.edu.uy/ingenieriademuestra. [Consultado: 05-jul-2016].

Figura K.1 - Presentación de Sendero en Equinoccio.

Figura K.2 - Daniel Martínez (Intendente de Montevideo) probando Sendero Cardboard durante Ingeniería de Muestra 2015.

Sendero

Christian Bouvier, Diego Ernst, Alvaro Larrosa
Tutor: MSc. Ing. Christian Clark
Laboratorio de Medios

Características generales

Sendero es un sistema open source de iluminación para la producción artística, desarrollado en primera instancia por el Laboratorio de Medios de la Facultad de Ingeniería de la Universidad de la República, y por el colectivo en diseño de interacción Bondi en el marco de la construcción de obras de arte con nuevos medios.

Se trata de un sistema de iluminación integral, abarcando un sistema de aplicaciones cliente-servidor para la generación de contenidos visuales, el desarrollo de un dispositivo basado en la plataforma mBed que emula un dispositivo DMX, y los componentes de hardware necesarios para el control de LEDs RGB de alta potencia.

Obras realizadas

Celebra es una instalación interactiva que comprende una red de doscientos globos de un metro de diámetro, iluminados independientemente, que reaccionan a la interacción del público y a la música del ambiente.

Barcelona es una escultura de hierro de dos metros de alto con forma de dodecaedro pentakis, cuyas aristas se iluminan independientemente y reaccionan al sonido ambiente y las interacciones del público presencial y remoto.

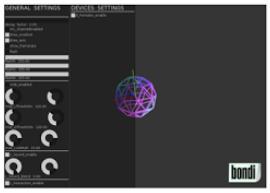
Arquitectura

El software se organiza mediante una arquitectura cliente-servidor. El servidor administra la estructura de la obra (ubicación especial de cada píxel) y orquesta a los clientes, módulos que interpretan señales externas para determinar los parámetros de iluminación de cada píxel.

En cuanto al hardware, un microcontrolador mBed se conecta al servidor y a controladores que manejan las tiras de LED que conforman la obra.

Los controladores LED reciben energía desde varias fuentes de alimentación e información de control desde el microcontrolador mBed.

Nuevos objetivos



- Modificación del esquema de **transmisión de datos** empleando técnicas de compresión y streaming multimedia.
- Refactorización de **interfaz web y mobile** utilizando WebSockets.
- Modificación del esquema de comunicación a nivel de hardware para soportar **transmisión de datos inalámbrica**.
- Realización de una **obra de arte** con nuevos medios que utilice las nuevas funcionalidades de Sendero.

Fundación Julio Ricaldoni INGENIERÍA EN EL URUGUAY
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Figura K.3 - Poster utilizado en Ingeniería de Muestra 2015.

Anexo L

Sync Meter

Sync Meter es un instrumento desarrollado en el marco de este proyecto de grado, capaz de registrar (desde el punto de vista de un observador externo) los *timestamps* en los cuales los paquetes son reproducidos en cada nodo receptor Wireless Bondibar. El mismo significó otro desarrollo de firmware (curiosamente quizás) utilizando el mismo *hardware* que el utilizado para los componentes Wireless Bondibar y su funcionamiento se basa en conectar cada uno de estos a él a través de cables que unen determinados pines GPIO.

Modo A: sincronismo entre receptores.

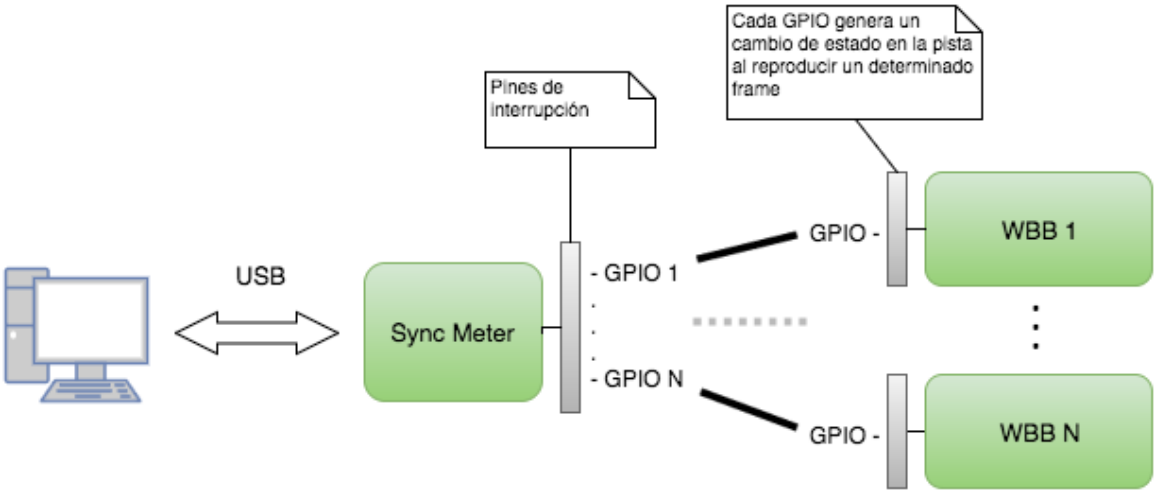


Figura L.1 - funcionamiento Sync Meter. WBB = Wireless Bondibar

En la Figura L.1 se muestra una imagen ilustrativa de sus componentes. Básicamente, cada nodo receptor posee un GPIO conectado a un GPIO del *Sync Meter* a través del cual éste es capaz de recibir interrupciones. Éstas son generadas por los nodos receptores al momento de reproducir paquetes que cumplan cierto criterio global a ellos (por ejemplo, un número de secuencia múltiplo de 30). Por otro lado, el nodo *Sync Meter*, cada vez que recibe una interrupción en determinado GPIO, almacena un *timestamp* asociado a la reproducción en el componente Wireless Bondibar allí conectado. Asumiendo que las interrupciones asociadas a la reproducción de determinado *frame* son generadas de a ráfagas (ya que a priori los receptores se encuentran sincronizados en su reproducción), *Sync Meter* es capaz de resumir el conjunto de *timestamps* de reproducción y enviarlo hacia una computadora a través de una interfaz USB.

En la Figura L.2 se muestra una imagen real análoga al esquema de la Figura L.1.

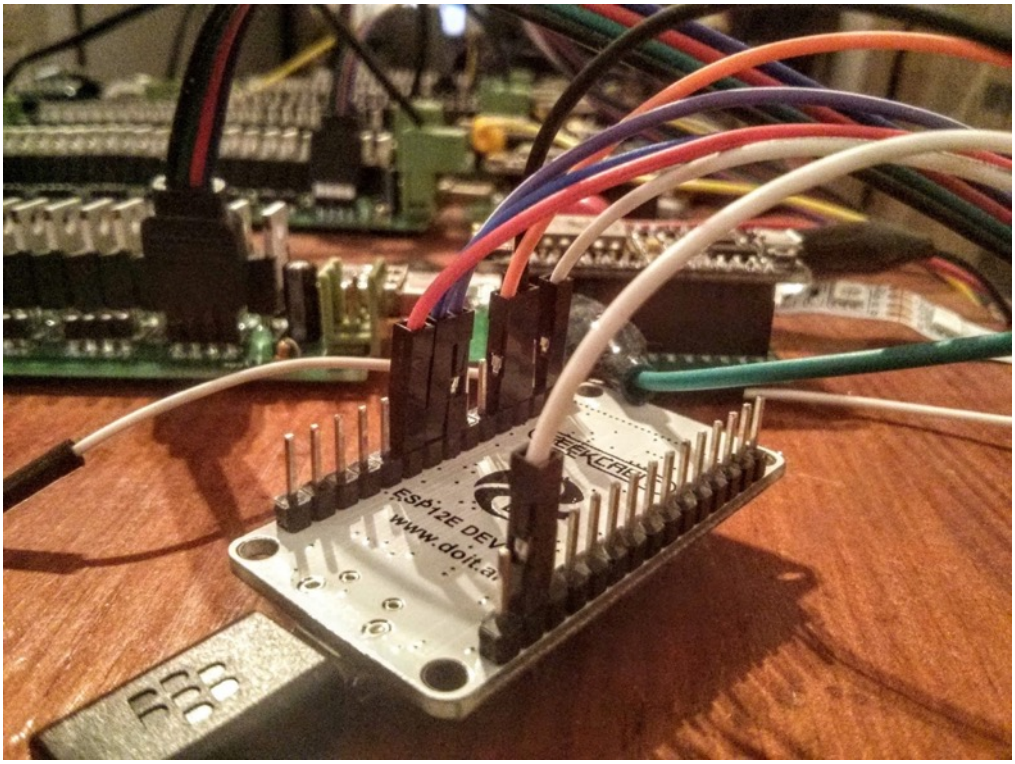


Figura L.2 - Sync Meter con seis receptores conectados a él.

En conclusión, como resultado de su utilización, se obtiene un conjunto de datos con la estructura descrita en la Tabla L.1, donde t_{ij} representa el *timestamp* en microsegundos asociado a la reproducción del *frame i* en el dispositivo *j*.

	WBB 1	...	WBB N
Frame 1	t_{11}	...	t_{1N}
...	...	...	...
Frame J	t_{J1}	...	t_{JN}

Tabla L.1 - estructura de datos resultado de la utilización de *Sync Meter*.

Una de las grandes ventajas de utilizar *Sync Meter* para medir el sincronismo entre receptores es su precisión, ya que ésta es de microsegundos. Dicha precisión es lograda gracias a la utilización de *hardware* para transmitir los eventos de reproducción como fue explicado anteriormente.

Modo B: latencia de reproducción respecto a Sendero Middleware

En este modo se desea utilizar *Sync Meter* para medir el tiempo transcurrido entre que los paquetes son generados en Sendero Middleware hasta que son reproducidos por los nodos receptores.
Si bien, a priori, la utilización del componente *Sync Meter* como instrumento de medición no es viable ya que Sendero Middleware es un componente de *software* ejecutando en una computadora (y no es posible conectarse a un GPIO del *Sync Meter*), se emuló parte de su comportamiento usando el mismo *hardware* que en los receptores. Concretamente, se emuló su rol de emisor en el canal de datos de SLP en un nodo (llamado *Emulador)* separado físicamente de Sendero Middleware que hizo posible su conexión al nodo *Sync Meter*. Las restantes funcionalidades tales como el descubrimiento de nuevos dispositivos y configuración (implementadas en el canal de control de SLP) son provistas por una versión reducida de Sendero Middleware. Dicho esquema de configuración fue posible gracias a que los canales del protocolo SLP no se encuentran restringidos a ser implementados por una única entidad.

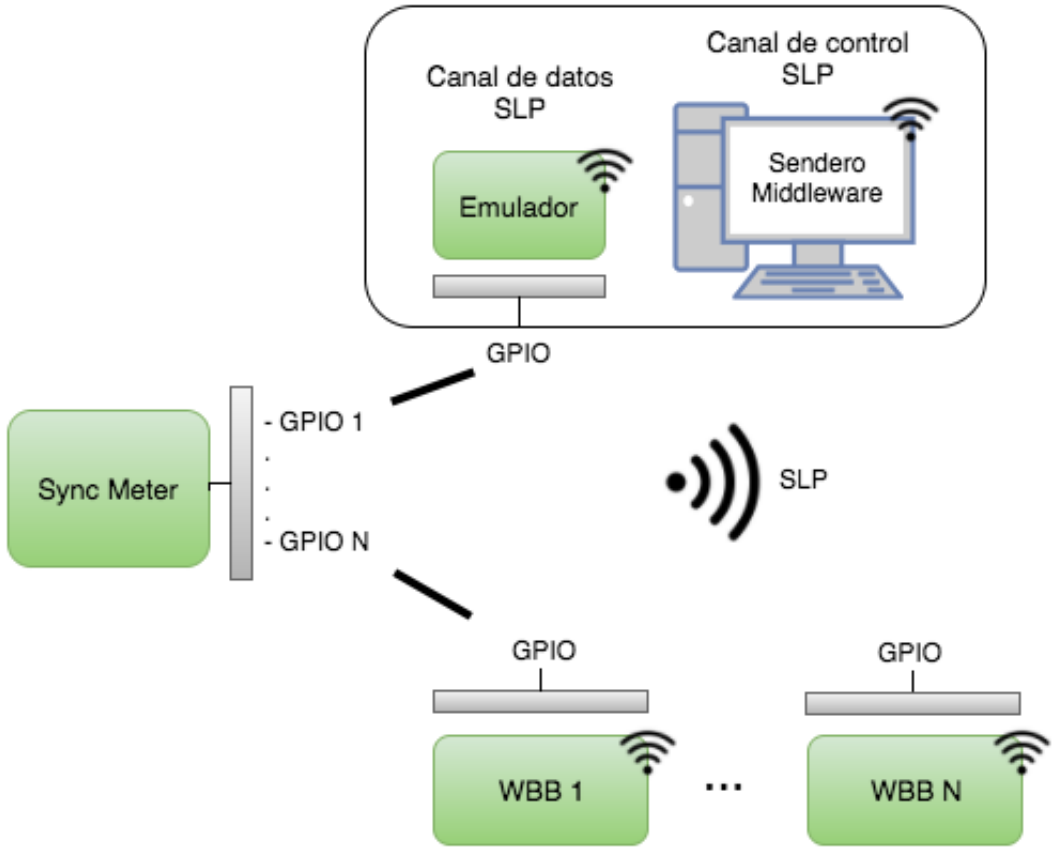


Figura L.3 - Esquema de medición entre Sendero Middleware y receptores Wireless Bondibar.

Los rectángulos en verde representan nodos implementados en el módulo NodeMCU.

En la figura L.3 se presenta el esquema utilizado para hacer posible la utilización de *Sync Meter* como instrumento de medición de la latencia de reproducción entre la entidad fuente de *streaming* y los receptores.

Anexo M

Pruebas Experimentales

En este anexo se describen las principales pruebas realizadas para evaluar el desempeño de los distintos subsistemas que componen la solución desarrollada.

Entorno

Para realizar las pruebas se construyeron 12 Sendero Wireless Dongle, cada una conectada solamente a un controlador Bondibar (también fueron construidos 12). En el contexto de una obra, cada tira LED representa un píxel y por lo tanto, cada nodo receptor Wireless Bondibar en la red controla a lo sumo 8 píxeles.
Se utilizó un *router* inalámbrico doméstico TP-Link WR941ND (con firmware dd-wrt[1]) usado para establecer la red WLAN a través de la cual se realizaron las comunicaciones utilizando el protocolo Sendero Lighting Protocol (SLP). Un aspecto importante respecto a la utilización del *router* es que los dispositivos conectados al mismo, no deben tener activado el modo ahorro de energía de Wi-Fi 802.11.

En cuanto al sub-sistema que involucra a los clientes remotos, Sendero Streaming Server ejecuta en un instancia de Amazon AWS t2.micro con un CPU Intel Xeon de hasta 3.3 GHz y 1 Gib de memoria RAM. Dicho servidor se encuentra en Oregon, Estados Unidos y tiene una modalidad *Free Tier* por un año.

Pruebas

Desempeño de receptores inalámbricos

En primer lugar, se desarrollaron una serie de pruebas para evaluar al subsistema compuesto por Sendero Middleware y los receptores inalámbricos Wireless Bondibar. En particular, se evaluó su capacidad en términos de la cantidad de píxeles y el *frame rate* soportado, el nivel de sincronismo de la reproducción entre los nodos receptores, y la latencia de reproducción.

[1] "DD-WRT | Unleash Your Router". [En línea]. Disponible en: http://www.dd-wrt.com/site/index. [Consultado: 05-jul-2016].

Cantidad máxima de píxeles soportada

Resulta fundamental poder determinar la capacidad máxima de píxeles soportada por la implementación actual del sistema. Si bien existen muchos aspectos como potenciales limitantes de esta capacidad, a efectos de las pruebas realizadas, se asumió que los componentes determinantes son los nodos Wireless Bondibar debido a su capacidad limitada de procesamiento.

Con el objetivo de evaluar el desempeño de los receptores ante las diferentes configuraciones se define el siguiente conjunto de estadísticas:

1. ***Bitrate* teórico por grupo (kbps)**
   Tasa de bits teórica dada la cantidad de píxeles recibidos por cada grupo *multicast* de nodos receptores.
2. ***Bitrate* práctico por grupo (kbps)**
   Tasa de bits efectiva dada la cantidad de píxeles recibidos por cada grupo *multicast* de nodos receptores.
3. **Tasa de paquetes que no fueron reproducidos a tiempo**
   Porcentaje de paquetes que no son capaces de ser reproducidos en una ventana de 1 milisegundo definida en base a su tiempo planificado de reproducción.
4. **Tasa de paquetes tardíos**
   Tasa de paquetes que fueron recibidos luego de su tiempo planificado de reproducción. Los mismos son descartados inmediatamente.
5. **Tasa de paquetes perdidos**
   Porcentaje de paquetes no recibidos por el receptor, ya sea debido a interferencias en la red o a falta de capacidad de procesamiento en el receptor.
6. **Tasa de reproducción**
   El *frame rate* promedio calculado en los receptores.
7. **Desviación estándar de la tasa de reproducción**
   Desviación estándar del *frame rate*.
8. **Tiempo promedio de *buffering***
   Tiempo promedio transcurrido desde que un paquete es recibido hasta que es reproducido.
9. **Desviación estándar del tiempo de *buffering***
   Desviación estándar de los tiempos de *buffering*.

Objetivo: Determinar la capacidad máxima de píxeles soportados para las distintas configuraciones de *frame rate* y cantidad de grupos *multicast*.

Entorno:

- 10 nodos receptores (instancias de Wireless Bondibar).
- Sendero Middleware como emisor ejecutando en PC conectada a través de un cable de red al *router* inalámbrico.
- 100 milisegundos de retraso en la reproducción.

Descripción: Debido a que la adaptación del entorno de Arduino para el chip ESP8266 utiliza lwIP[2] como implementación del *stack* TCP/IP y una de sus principales características es la baja utilización de recursos, la capacidad de recibir paquetes UDP con fragmentación a nivel de capa IP se encuentra deshabilitada. Adicionalmente, la implementación utilizada de lwIP posee un MTU (*Maximum Transmission Unit*) de 1500 bytes y por esta razón, la mayor cantidad máxima de bytes posibles a ser enviados en un paquete SLP evitando fragmentación es de 1500 - 6 (cabezal SLP) - 8 (cabezal UDP) - 20 (cabezal IP) = 1466 bytes. Por lo tanto, la máxima cantidad de pixeles posible en un paquete SLP es de 488.

Se plantea realizar el siguiente procedimiento utilizando 24 y 30 *frames* por segundo:

1. Utilizando la cantidad máxima de píxeles por grupo *multicast*, incrementar la cantidad de grupos y en cada paso analizar las estadísticas obtenidas.
2. Cuando se alcance un nivel no aceptable de funcionamiento en el punto anterior, reducir la cantidad de píxeles teniendo como cota mínima la cantidad de píxeles total utilizada para el nivel anterior de grupos *multicast*.

Resultados y análisis: En caso de no aclarar lo contrario, se asume que la cantidad de píxeles por grupo *multicast* es la máxima. Adicionalmente, el tiempo entre envíos de paquetes hacia los distintos grupos *multicast*, se calculó de tal manera que todos los envíos desde Sendero Middleware resulten equiespaciados en el tiempo. Esta decisión se basa en que en la práctica, cuando un paquete *multicast* es enviado sobre una red Wi-Fi, el mismo es recibido por todos los receptores en la misma y es responsabilidad de la interfaz de red y/o del *stack* IP presente en el receptor, filtrar estos paquetes según la dirección del grupo que se haya configurado[3]. Por esta razón, los nodos receptores requieren mayor tiempo de procesamiento dedicado a descartar paquetes a medida que se incrementan los grupos *multicast*, y se obtuvieron mejores resultados si la distancia entre arribos es la máxima posible.

Pérdidas de paquetes menores a 0,3% suponen una calidad percibida de video de buena a excelente[4], y si bien el flujo de paquetes transmitido no es exactamente video, presenta características similares. En el caso de esta prueba, donde se transmite 1 *frame* por cada paquete transmitido y se utilizan mecanismos de recuperación ante pérdida, este umbral de tolerancia podría extenderse aún más.

---

[2] "lwIP - A Lightweight TCP/IP stack - Summary". [En línea]. Disponible en: http://savannah.nongnu.org/projects/lwip/. [Consultado: 03-jul-2016].

[3] S. E. Deering, "Host extensions for IP multicasting", RFC 1112, ago. 1989.

[4] J. S. Mwela, "Impact of Packet Loss on the Quality of Video Stream Transmission", Ph.D. Thesis, Blekinge Institute of Technology, 2010.

Adicionalmente, la tasa de paquetes recibidos luego de su tiempo de reproducción será considerada como aceptable en valores muy cercanos a 0%, ya que de lo contrario es síntoma de un mal funcionamiento (considerando que se dispone de 100 milisegundo de retraso en la reproducción y los participantes de la comunicación se encuentran ambos en una WLAN de un hop).

Por otro lado, debido a que el retraso en la reproducción es de 100 milisegundos, y dados N grupos *multicast*, el tiempo de *buffering* para determinado grupo *multicast* i, $0 \leq i < N$, debería ser como máximo e idealmente (asumiendo un envio de paquete con tiempos de transferencia y procesamiento mínimo) $100 - \frac{i*1000}{fps*N}$ milisegundos. Esto se debe a que Sendero Middleware configura los retrasos de reproducción para cada grupo *multicast* utilizando esa fórmula para lograr envíos de paquetes equiespaciados en el tiempo como fue mencionado anteriormente. Por lo tanto, tiempos de *buffering* cercanos al máximo son considerados mejores frente a los alejados.

A continuación se presenta la lista de pruebas realizados y finalmente se presenta un estudio resumido y comparativo de la evolución del desempeño en función de las distintas configuraciones. Para cada test realizado, se reportaron los valores promedios de los resultados obtenidos en cada nodo receptor. De todas formas, cabe aclarar que los resultados obtenidos en cada nodo receptor fueron prácticamente iguales; a excepción de los tiempos de *buffering* cuando la configuración establece más de un grupo *multicast*.

Test A: 488 píxeles, un grupo multicast:

	24 fps	**30 fps**
Bitrate teórico por grupo	287,616	359,520
Bitrate práctico por grupo	287,807	355,680
Tasa de paquetes que no fueron reproducidos a tiempo	0,000%	0,000%
Tasa de paquetes tardíos	0,000%	0,000%
Tasa de paquetes perdidos	0,091%	0,039%
Tasa de reproducción	23,811	29,413
Desviación estándar de la tasa de reproducción	0,022	0,022
Tiempo promedio de buffering	98,104	97,872
Desviación estándar del tiempo de buffering	1,651	2,136

Tabla M.1 -Test A

Observaciones:

1. El bitrate obtenido para los distintos fps se encuentra cercano al teórico.
2. Pérdidas de paquetes consideradas imperceptibles.
3. Bajo esta configuración, el tiempo de *buffering* ideal es de 100 milisegundos, y ambas configuraciones de fps reportan valores cercanos al mismo.

Test B: 976 píxeles, dos grupos multicast

	24 fps		**30 fps**	
Bitrate teórico por grupo	287,616		359,520	
Bitrate práctico por grupo	287,974		355,708	
Tasa de paquetes que no fueron reproducidos a tiempo	0,000%		0,000%	
Tasa de paquetes tardíos	0,000%		0,000%	
Tasa de paquetes perdidos	0,031%		0,041%	
Tasa de reproducción	23,811		29,413	
Desviación estándar de la tasa de reproducción	0,027		0,205	
Tiempo promedio de *buffering*	Gr. 0	Gr. 1	Gr. 0	Gr. 1
	97,922	76,817	98,022	80,833
Desviación estándar del tiempo de *buffering*	1,897	1,888	1,867	1,918

Tabla M.2 - Test B

Observaciones:

1. El *bitrate* obtenido para los distintos *fps* se encuentra cercano al teórico.
2. Pérdidas de paquetes consideradas imperceptibles.
3. Tiempos de *buffering*:
    a. 24 fps:
    Bajo esta configuración, los tiempos de *buffering* ideales son de 100 y 79 milisegundos para el grupo 0 y grupo 1 respectivamente.
    b. 30 fps:
    Bajo esta configuración, los tiempos de *buffering* ideales son de 100 y 83 milisegundos para el grupo 0 y grupo 1 respectivamente.

Test C: 1464 píxeles, tres grupos multicast.

Bajo esta configuración, ninguno de los diez nodos receptores presentes en la prueba, fue capaz de iniciar la reproducción. Los mismos permanecen estancados en el período de ca-

libración inicial no pudiendo determinar los tiempos planificados de reproducción para los paquetes recibidos (ver Playback Scheduler). Básicamente, los paquetes recibidos desbordan la capacidad de procesamiento de los nodos receptores provocando grandes diferencias en los *timestamps* de recepción calculados.

Test D: 978 píxeles (326 por grupo), tres grupos multicast.

Debido a que el *Test C* no fue soportado por los receptores, se desea conocer el rendimiento de los nodos utilizando tres grupos *multicast* y la mínima cantidad de píxeles superior a los utilizados en el *Test B.*

	24 fps			**30 fps**		
Bitrate teórico por grupo	194,304			242,88		
Bitrate práctico por grupo	195,354			241,322		
Tasa de paquetes que no fueron reproducidos a tiempo	0,000%			0,000%		
Tasa de paquetes tardíos	0,000%			3,312%		
Tasa de paquetes perdidos	0,049%			0,076%		
Tasa de reproducción	23,811			29,413		
Desviación estándar de la tasa de reproducción	0,144			0,111		
	Gr. 0	**Gr. 1**	**Gr. 2**	**Gr. 0**	**Gr. 1**	**Gr. 2**
Tiempo promedio de buffering	98,023	83,666	69,332	97,096	84,767	74,229
Desviación estándar del tiempo de buffering	1,947	1,966	1,999	4,934	4,364	3,893

Tabla M.3 - Test D

Observaciones:

1. El test realizado para 30 fps, arrojó una tasa de paquetes recibidos luego de su tiempo planificado de reproducción levemente superior al 3% lo cual es **inaceptable**. Por lo tanto, se considera que la máxima cantidad de píxeles soportados en 30 fps fue alcanzada en el *Test B* (976 píxeles).
2. Observaciones para 24 fps:
    a. Mismas observaciones que las realizadas en el *Test B.*
    b. Tiempos de *buffering*:

Bajo esta configuración, los tiempos de *buffering* ideales son de 100, 86 y 72 milisegundos para el grupo 0, 1 y 2 respectivamente.

Test E: 1200 píxeles (400 por grupo), tres grupos multicast. (solo 24 fps)

	24 fps		
Bitrate teórico por grupo	236,928		
Bitrate práctico por grupo	237,657		
Tasa de paquetes que no fueron reproducidos a tiempo	0,000%		
Tasa de paquetes tardíos	0,000%		
Tasa de paquetes perdidos	0,038%		
Tasa de reproducción	23,811		
Desviación estándar de la tasa de reproducción	0,147		
	Gr. 0	**Gr. 1**	**Gr. 2**
Tiempo promedio de buffering	97,722	83,556	69,144
Desviación estándar del tiempo de buffering	2,573	2,491	2,529

Tabla M.4 – Test E.

Observaciones:

1. Mismas observaciones que para el *Test D,* aunque se observan mayores desviaciones estándares en los tiempos de *buffering.*

Test F: 1350 píxeles (450 por grupo), tres grupos multicast. (solo 24 fps)

	24 fps		
Bitrate teórico por grupo	265,728		
Bitrate práctico por grupo	266,108		
Tasa de paquetes que no fueron reproducidos a tiempo	0,000%		
Tasa de paquetes tardíos	0,568%		
Tasa de paquetes perdidos	0,639%		
Tasa de reproducción	23,812		
Desviación estándar de la tasa de reproducción	0,202		
	Gr. 0	**Gr. 1**	**Gr. 2**
Tiempo promedio de buffering	95,809	81,888	67,934
Desviación estándar del tiempo de buffering	7,660	7,071	6,092

Tabla M.5 - Test F

Observaciones:

1. Bajo esta configuración, se comienzan a presenciar paquetes recibidos luego de su tiempo planificado de reproducción. Lo cual es síntoma de mal funcionamiento.
2. Los tiempos de buffering presentan notoriamente más varianza que en el test anterior.
3. La tasa de paquetes perdidos supera el nivel de tolerancia del 0,3%.

Test G: 1410 píxeles (470 por grupo), tres grupos multicast. (solo 24 fps).

Nota: La cantidad de píxeles utilizada en esta configuración se encuentra muy cercana a la cantidad utilizada en el *Test C,* donde los nodos receptores no fueron capaces de iniciar la reproducción.

	24 fps		
Bitrate teórico por grupo	277,248		
Bitrate práctico por grupo	277,561		
Tasa de paquetes que no fueron reproducidos a tiempo	0,000%		
Tasa de paquetes tardíos	21,035%		
Tasa de paquetes perdidos	0,070%		
Tasa de reproducción	23,812		
Desviación estándar de la tasa de reproducción	0,328		
	Gr. 0	**Gr. 1**	**Gr. 2**
Tiempo promedio de buffering	93,330	79,284	65,311
Stdev del tiempo de buffering	10,603	9,912	9,444

Tabla M.6 - Test G

Observaciones:

1. Durante las pruebas se constataron períodos de no reproducción por parte de los nodos receptores.
2. En este nivel de configuración, los paquetes tardíos reportaron una estadística del 20% lo cual es **inaceptable**.

Resumen comparativo

En base a los resultados obtenidos en las distintas configuraciones y tests realizados se puede observar que los principales indicadores de deterioro en el buen funcionamiento de los nodos receptores son la tasa de paquetes tardíos y los tiempos promedio de *buffering* así como también su desviación estándar.

Con el objetivo de comparar los distintos tiempos de *buffering*, se proponen los siguientes indicadores, siendo N la cantidad de grupos *multicast*, y dado un grupo *i,* b_i el tiempo de *buffering*, stdev(b_i) su desviación estándar y d_i el tiempo ideal de *buffering*:

- Indicador de tiempo de *buffering* en función del tiempo ideal (valores cercanos a 1 son favoritos):

$$Tb = \frac{1}{N} * \sum_{i=0}^{N-1} \frac{b_i}{d_i}$$

- Indicador de desviación estándar del tiempo de *buffering*:

$$stdev(Tb) = \frac{1}{N} * \sum_{i=0}^{N-1} stdev(b_i)$$

Las Figuras M.1, M.2, y M.3 presentan las gráficas comparativas entre las distintas configuraciones de cantidad de píxeles y cantidad de grupos *multicast* utilizados, para las métricas *Tb, stdev(Tb)* y tasa de paquetes tardíos respectivamente.

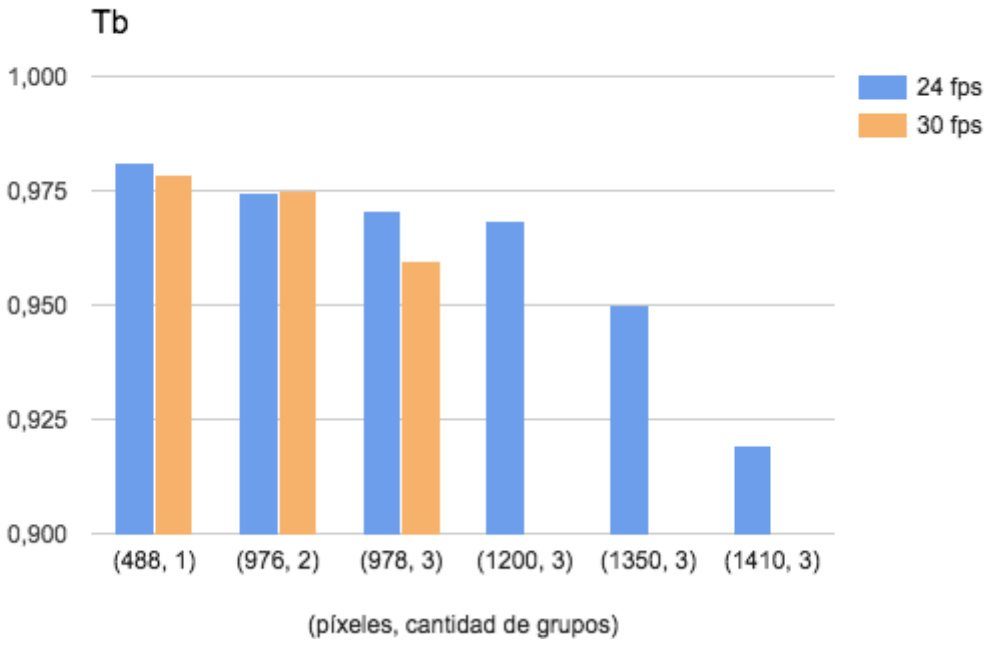


Figura M.1 - Gráfica comparativa para *Tb*.

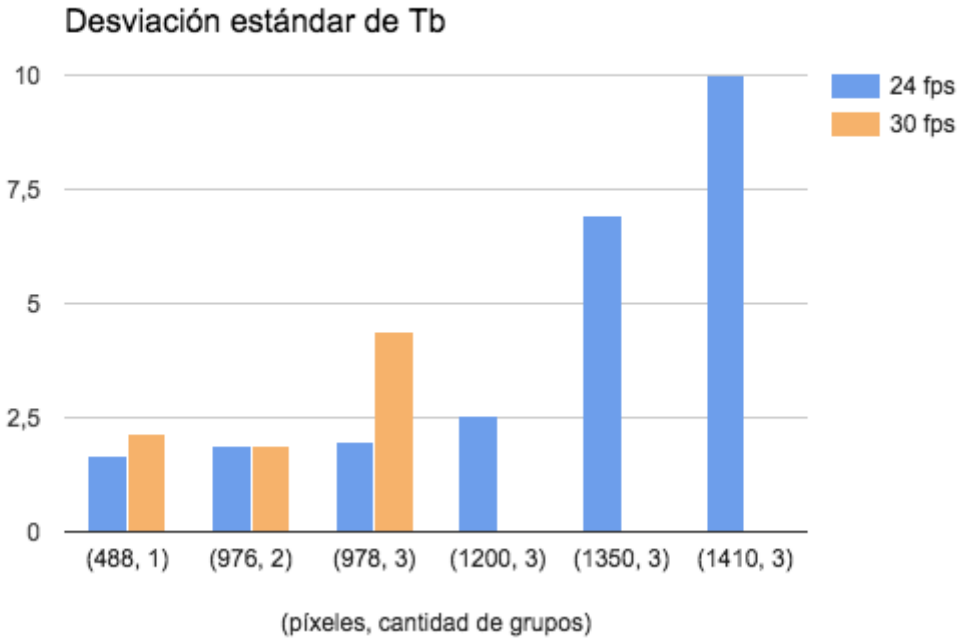


Figura M.2 - Gráfica comparativa para el indicador de desviación estándar del tiempo de *buffering*.

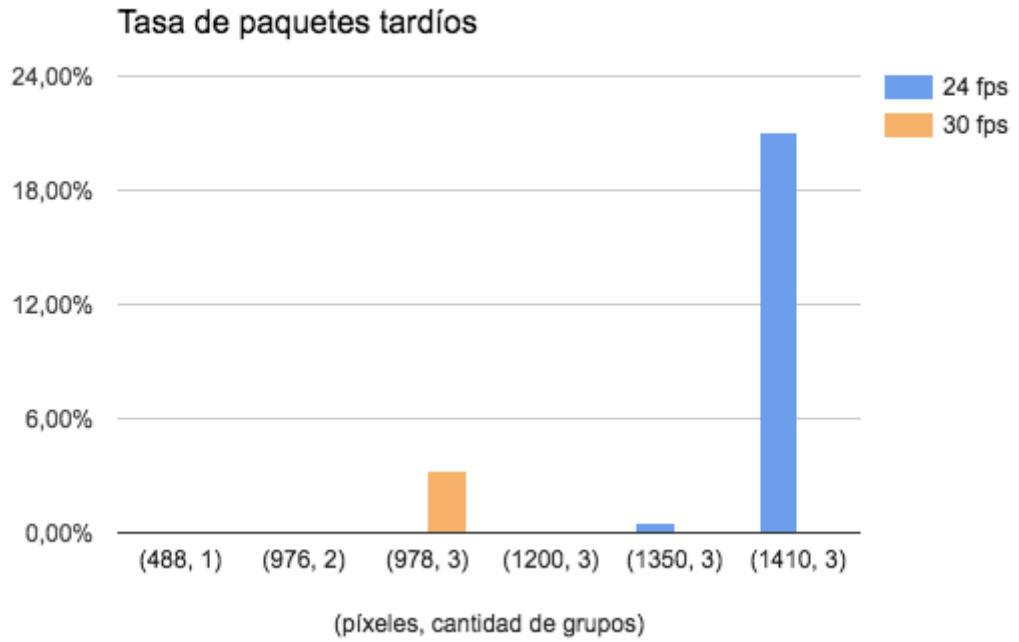


Figura M.3 - Gráfica comparativa de tasas de paquetes tardíos.

Como se puede apreciar en las Figuras M.1 y M.2, a medida que crece la cantidad de grupos y píxeles, el indicador *Tb* disminuye y su desviación aumenta. Esto implica mayores dificultades de procesamiento y recepción de paquetes por parte de los receptores.

Aun así, esta situación no presume un mal comportamiento en la reproducción ya que, justamente, el tiempo de *buffering* de paquetes es dedicado a absorber este tipo de comportamientos.

Por otro lado, en la Figura M.3 se puede apreciar que para la configuración de 978 píxeles y 3 grupos *multicast* a 30 *fps*, se obtuvo un porcentaje no despreciable (alrededor de 3%) de paquetes tardíos lo cual sí se vió reflejado en la calidad percibida de reproducción. Por ello, se considera que la máxima cantidad de píxeles soportada a 30 fps, es de 976 utilizando dos grupos *multicast*. Los resultados obtenidos a 24 fps presentan una pequeña tasa de paquetes tardíos (~ 0,6 %) en 1300 píxeles y 3 grupos *multicast* lo cual puede ser considerado como un primer indicio de mal funcionamiento aunque en la práctica no se visualizaron. Por último, en el siguiente nivel de 1400 píxeles a 24 fps sí se sobrepasó la capacidad de procesamiento y recepción de los nodos receptores, reportando una tasa de paquetes tardíos superior al 20%.

Conclusión: En base a las pruebas realizadas, la implementación actual del protocolo SLP utilizada para la emisión del *streaming* de datos desde Sendero Middleware hacia el conjunto de nodos receptores Wireless Bondibar presentó una máxima capacidad de **976 píxeles a 30 *fps*** y **1200 píxeles a 24 *fps***. Cabe aclarar que estos resultados obtenidos dependen fuertemente del *hardware* utilizado para la implementación de los nodos Wireless Bondibar (ESP-12E).

Sincronismo entre receptores inalámbricos

Objetivo: Medir y evaluar el nivel de sincronismo en la reproducción de *frames* entre receptores. Concretamente, evaluar las diferencias de tiempo transcurrido entre las reproducciones de un **mismo *frame*** entre todos los receptores.

Entorno:

- Configuración A: un único grupo *multicast*.
    - 6 nodos receptores Wireless Bondibar conectados a un controlador Bondibar cada uno.
    - 1 nodo emisor ESP-12E corriendo una instancia parcial emulada de Sendero Middleware (ver Anexo L).
- Configuración B: tres grupos *multicast*.
    - 6 nodos receptores Wireless Bondibar conectados a un controlador Bondibar cada uno.
    - Sendero Middleware como emisor ejecutando en una PC conectada a través de un cable de red al *router* inalámbrico. El mismo emite los *frames* hacia tres grupos *multicast* de igual tamaño con un intervalo de 10 milisegundos entre envíos.

Descripción: Se emite un *streaming* de paquetes SLP Data Packet desde el nodo emisor hacia los 6 nodos receptores con las siguientes características.

- 30 *frames* por segundo.
- 100 milisegundos de *delay* en la reproducción.
- 48 píxeles en total (tiras LED). Cada nodo receptor controla 8 píxeles.
- 5 horas de duración.

Para realizar la medición, se utilizó el componente desarrollado *Sync Meter* (ver Anexo L). En dicha utilización, con el objetivo de no generar una cantidad inmensa de datos y asegurar un correcto funcionamiento, cada nodo receptor comunicó a *Sync Meter* la reproducción de los *frames* con número de secuencia múltiplos de 32. De aquí en más, estos *frames* específicos serán llamados *muestras.*

Resultados: La salida obtenida de la sesión arrojó alrededor de 17500 *timestamps* de reproducción por cada nodo receptor. A partir del conjunto de *timestamps* $\{ ti \}_n$ con $0 \leq i < 6$ asociado a la reproducción del *frame* n en cada nodo receptor i, se define la siguiente métrica:

$$S(n) = máx\{ abs(ti - tj) \}_n \quad \forall i,j \ / 0 \leq i,j < 6$$

donde *S(n)* representa la máxima diferencia entre los tiempos de reproducción para la muestra n-ésima (medidos desde el punto de vista de un observador externo).
Para la configuración A, el total de las muestras obtenidas *S(n)* presenta valor promedio de 1010 microsegundos y una desviación estándar de 581 microsegundos; mientras que para la configuración B el valor promedio y desviación estándar es de 1335 y 777 microsegundos respectivamente.

En las Figuras M.4 y M.5 se presentan los histogramas de los valores obtenidos para las distintas configuraciones de la prueba.

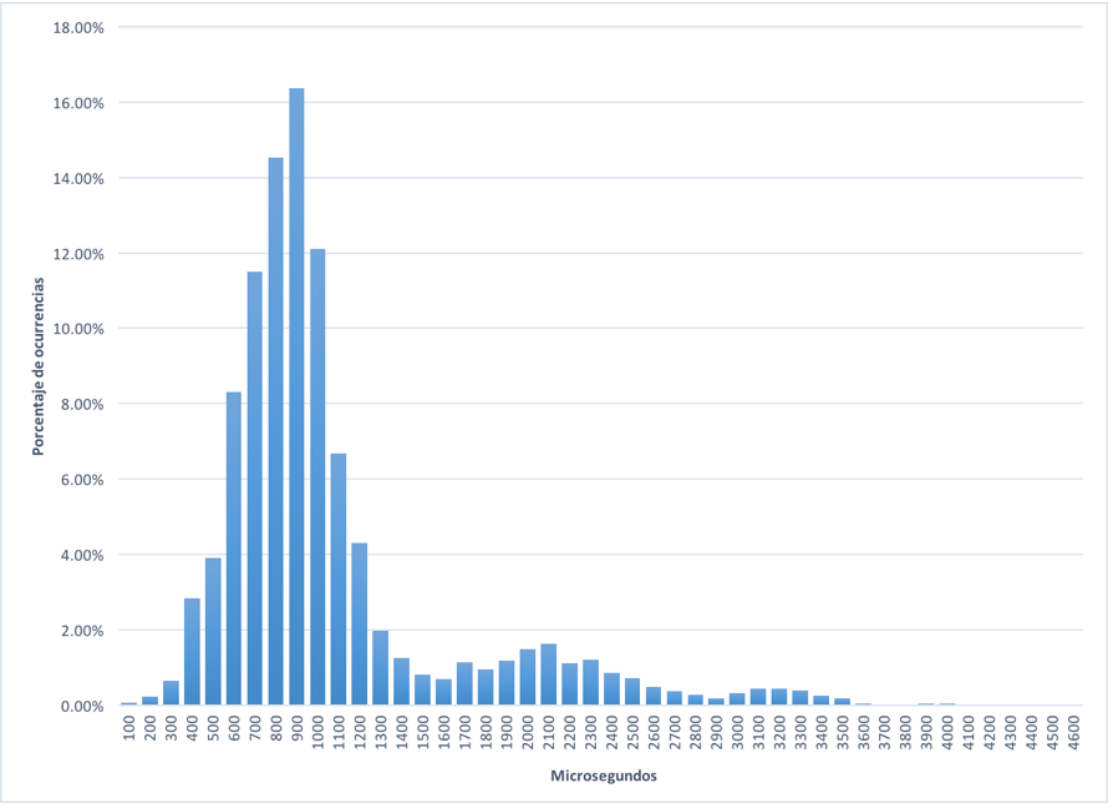


Figura M.4 - Histograma de *S(n)*, configuración A.

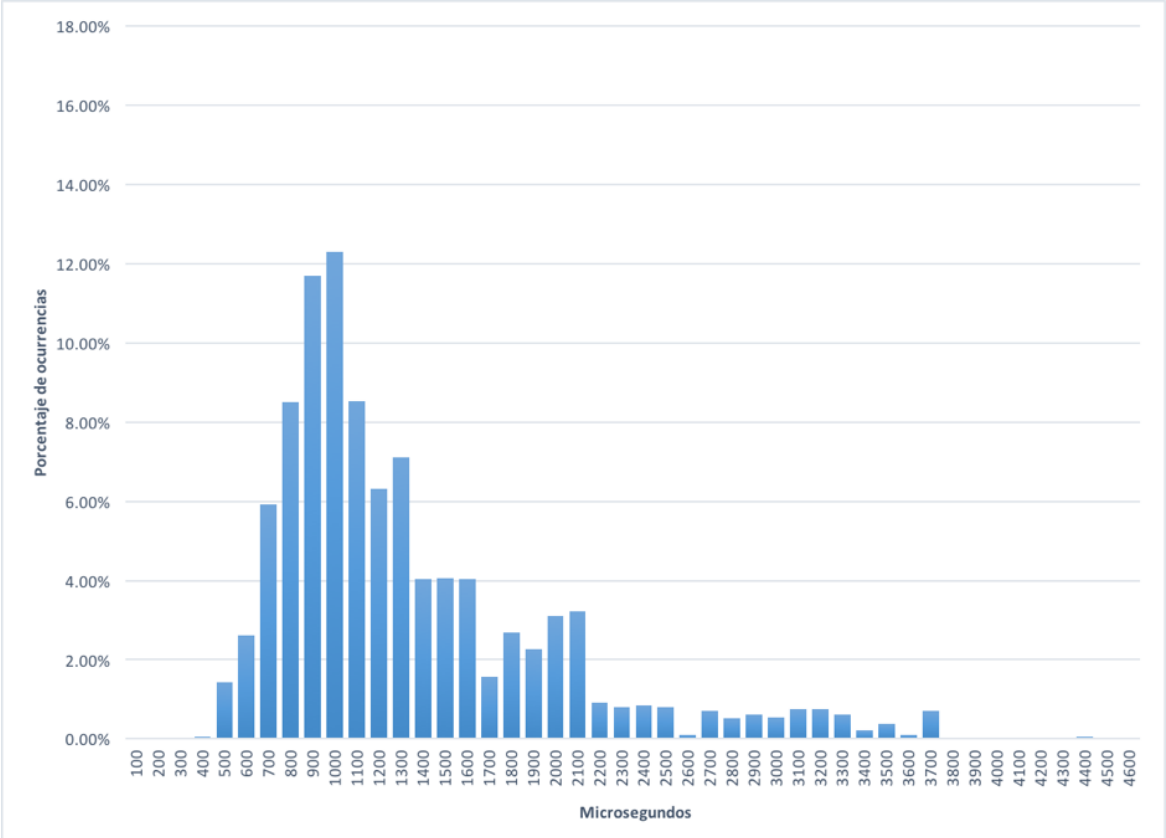


Figura M.5 - Histograma de *S(n)*, configuración B.

Análisis: A partir de los histogramas y datos obtenidos se pueden realizar las siguientes apreciaciones:

1. Las diferencias máximas obtenidas entre receptores es de aproximadamente 3,7 ms.
2. Aproximadamente el 91% de las muestras *S(n)* son menores a 2 ms.
3. Si bien los resultados de las distintas configuraciones no presentan grandes diferencias en cuanto a los valores *S(n)*, es notorio que utilizando un único grupo *multicast* se obtiene una distribución mucho más definida en torno a una media.

Conclusión: Debido a que distribución obtenida de los valores *S(n)* presenta un valor promedio de aproximadamente 1 milisegundo con desviación estándar menor a 0,8 milisegundos podemos concluir que las diferencias de tiempos de reproducción para una misma muestra entre los receptores son prácticamente imperceptibles desde un punto de vista humano y adicionalmente sostenible en el tiempo ya que la duración total de la prueba fue de aproximadamente 5 horas. No se encontraron estudios que clasifiquen este tipo de resultados desde el punto de vista de la percepción humana, sin embargo, se constató durante el proyecto que diferencias entre reproducciones menores a 15 milisegundos son imperceptibles.

Latencia de reproducción

De forma análoga a las tareas realizadas en la sección anterior, se desarrollaron pruebas de utilizando el componente *Sync Meter* pero esta vez, entre el emisor de *streaming* Sendero Middleware y los receptores. Resulta de gran interés estudiar el comportamiento del tiempo transcurrido desde que un *frame* es generado hasta que es reproducido desde un punto de vista **externo**.

Objetivo: Medir y evaluar el nivel de latencia en la reproducción entre los paquetes generados desde el emisor hacia cada nodo receptor. Concretamente, medir el tiempo transcurrido desde que un *frame* es generado hasta que es reproducido.

Entorno:

- 6 nodos receptores Wireless Bondibar conectados a un controlador Bondibar cada uno.
- 1 nodo emisor ESP-12E corriendo una instancia parcial emulada de Sendero Middleware (ver Anexo L).

Descripción: Se emite un *streaming* de paquetes SLP Data Packet desde el nodo emisor hacia los 6 nodos receptores con las siguientes características.

- 30 *frames* por segundo.
- 100 milisegundos de *delay* en la reproducción.
- 48 píxeles en total (tiras LED). Cada nodo receptor controla 8 píxeles.

- 5 horas de duración.

Para realizar la medición, se utilizó el componente desarrollado *Sync Meter* (ver Anexo L). En dicha utilización, cada nodo receptor comunicó a *Sync Meter* la reproducción de los *frames* con número de secuencia múltiplos de 32, pero adicionalmente también lo hizo el nodo emisor al momento de su emisión. Cada uno de ellos será referido como *muestra* de aquí en más.

Resultados: La salida obtenida de la sesión arrojó alrededor de 17500 *timestamps* de emisión y reproducción de muestras por parte del emisor y cada uno de los nodos respectivamente. En base a este conjunto de datos, para cada nodo receptor se definió la siguiente métrica para evaluar la *latencia* de reproducción en cada muestra *n*:

$$L(n) = R(n) - E(n)$$

siendo *R(n)* el tiempo de reproducción de la muestra *n* y *E(n)* el tiempo de emisión de la misma. Cabe aclarar, que todos estos tiempos son medidos utilizando el mismo sistema de referencia temporal ubicado en el nodo *Sync Meter* y representan tiempos de latencia **absolutos**.

Análisis: En la Figura M.6, se presenta un conjunto de medidas *L(n) con* $0 \leq n < 2000$ para cada nodo receptor presente en la prueba. Se decidió restringir visualmente el rango de muestras debido a que la inmensidad de datos obtenida no permitía observar apropiadamente el comportamiento y a que, adicionalmente, el patrón observado presenta características similares a lo largo del tiempo.

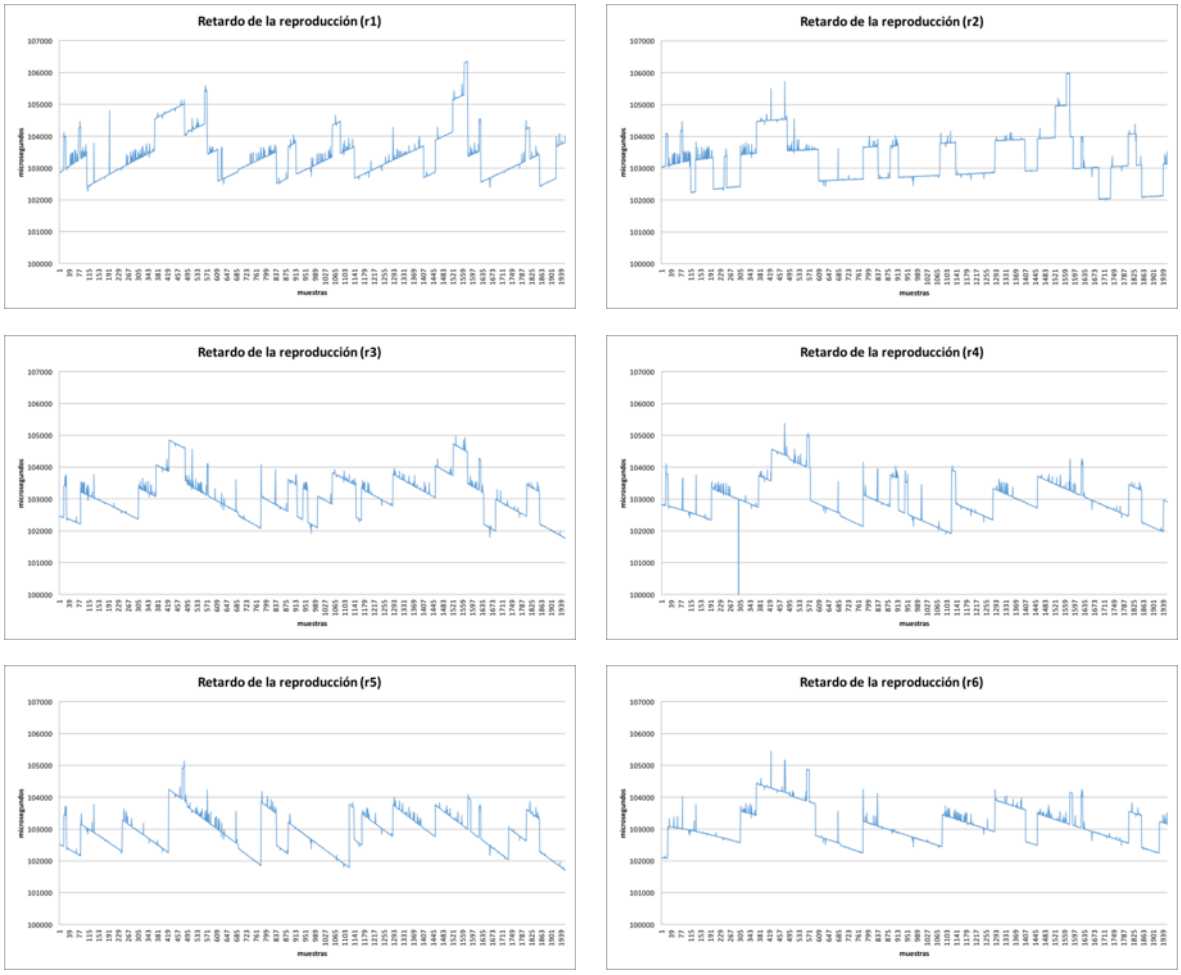


Figura M.6 - Latencia *L(n)* para cada nodo receptor.

En la tabla M.7 se presenta el promedio y desviación estándar de todas las muestras *L(n)* para cada receptor.

Receptor	**Promedio *L(n)* (us)**	**Desviación estándar *L(n)* (us)**
1	103559	974
2	103380	934
3	103167	877
4	103205	872
5	103073	854
6	103262	854

Tabla M.7 - Promedio y desviación estándar de *L(n)* para cada receptor.

Una primer observación resulta ser que *L(n)* presenta una cota inferior de aproximadamente 102 milisegundos. Debido a que el retraso fijo de reproducción añadido en el receptor es de 100 milisegundos, podemos concluir que el tiempo **mínimo** de transferencia y procesamiento entre ambos nodos es de aproximadamente 2 milisegundos.

En cualquier caso, la latencia de reproducción desde el punto de vista de un observador externo será siempre mayor a 100 milisegundos. En la Figura M.7 se presenta una ilustración que explica este fenómeno, donde:

- t_e, t_r y t_{pt} representan los tiempos de emisión, recepción y reproducción, respectivamente, respecto al reloj de referencia en el nodo emisor.
- t'_e, t'_r y t'_{pt} representan los tiempos de emisión, recepción y reproducción, respectivamente, en el nodo receptor.
- escenarios:
    - **a)** El *frame* enviado presenta un tiempo de transmisión y procesamiento mínimo por lo cual, la diferencia de *timestamps* $t'_r - t_e$ es mínima, y como fue explicado en el módulo Playback Scheduler de la Sección 4.3.3.2, esta diferencia representa el offset entre los relojes del emisor y receptor. En este caso, el paquete permanece en el buffer 100 milisegundos hasta su reproducción.
    - **b)** Aquí, el *frame* presenta un tiempo de transmisión y procesamiento mayor al mínimo observado. En este caso el *frame* permanece en el buffer (en espera por su reproducción) $100 - d$ milisegundos.

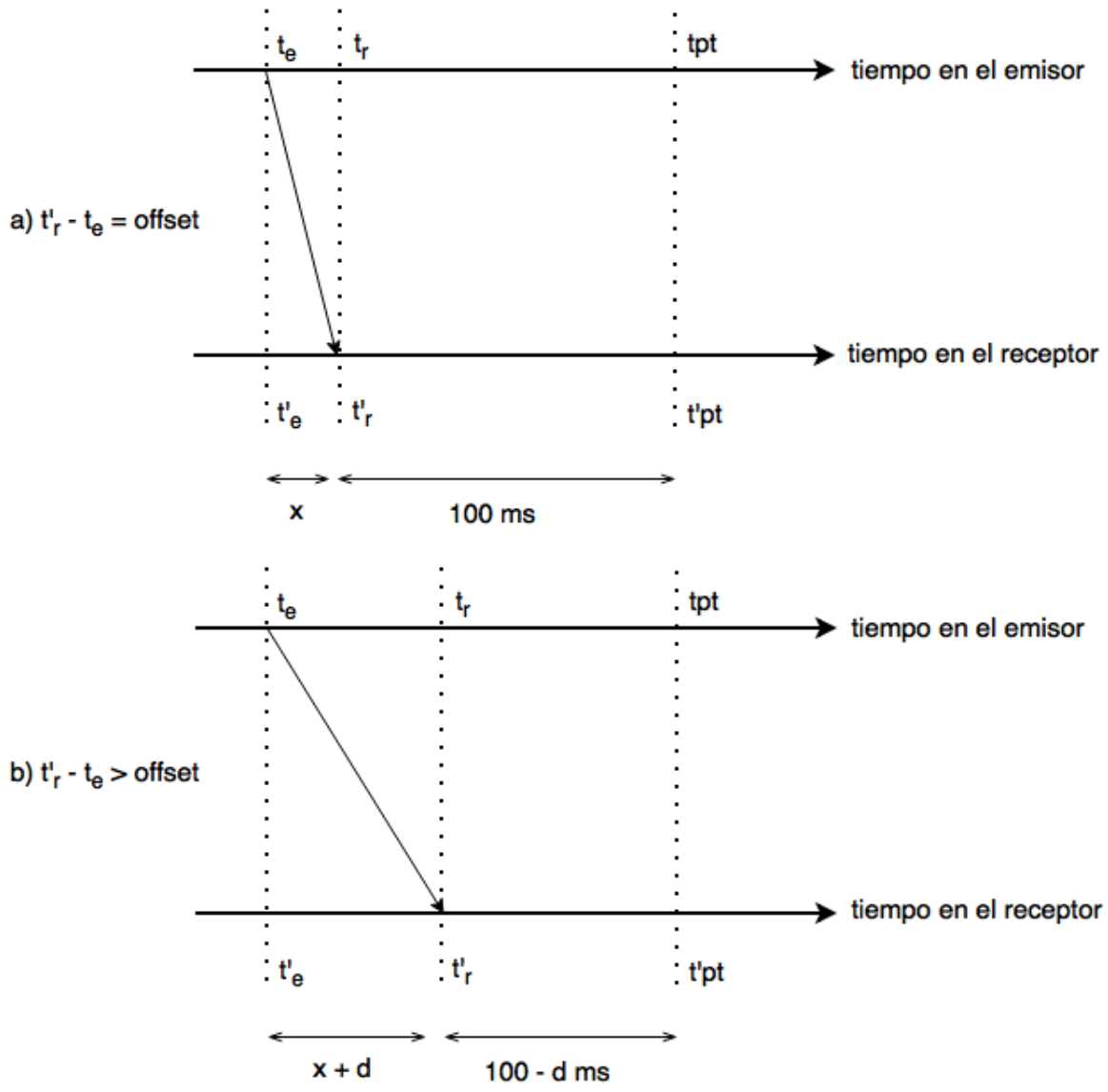


Figura M.7 - Latencia de reproducción. Discusión en casos.

En la Figura M.8, se presenta una imagen ampliada (a modo de ejemplo, para el receptor 6) y a continuación se realiza un análisis de sus características (las cuales pueden ser generalizadas para los restantes receptores).

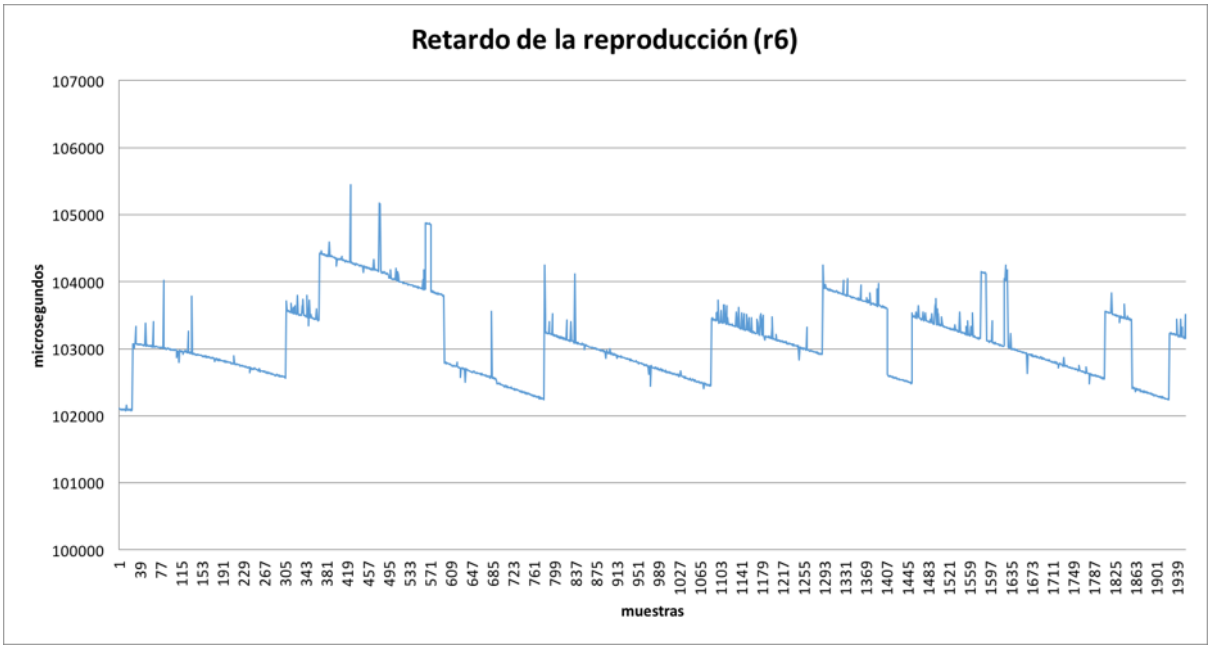


Figura M.8 - Muestras *L(n)* para el receptor 6.

Como principal apreciación se observa el comportamiento escalonado obtenido de *L(n)*. En la Figura M.8, la gráfica presenta escalones verticales en su mayoría de 1 milisegundo de alto, los cuales se deben al método de expiración del *offset* explicado en Playback Scheduler de la Sección 4.3.3.2. Por otro lado, cabe aclarar que debido a que la precisión utilizada en los nodos receptores es de milisegundos, los picos de menor altura probablemente se deban a errores de medición.

Por último, una de las razones por las cuales el nodo *Sync Meter* fue implementado con una precisión de microsegundos, fue para ser capaz de medir el *clock skew* entre relojes, el cual se hace notorio en las pendientes presentes en la gráfica. Desde el punto de vista del receptor, esas pendientes representan períodos en los cuales el *offset* observado se mantiene invariante en el tiempo, y por ello, de no contar con los períodos de expiración implementados, la reproducción y sincronización entre receptores se vería fuertemente afectada conforme avanza el tiempo.

Conclusión: Todos los receptores presentan una latencia de reproducción de aproximadamente 103 +/- 0.9 milisegundos lo cuales son considerados tiempos de latencia imperceptibles para un flujo multimedia en tiempo real[5]. Adicionalmente, estos tiempos de latencia son sostenibles en el tiempo eliminando las diferentes desviaciones de relojes con el emisor (*clock skew*).

[5] I. K. Ibrahim y D. Taniar, Mobile Multimedia: Communication Engineering Perspective. Nova Publishers, 2006.

Desempeño de clientes remotos y streaming web

Otro gran subsistema a ser evaluado es el que involucra a los clientes remotos. El contenido visual mezclado en Sendero Server es transmitido a través de Internet para ser consumido en tiempo real por los clientes remotos, haciendo uso de las aplicaciones web desarrolladas, Sendero Web y Sendero Cardboard.

En este contexto, se pretenden evaluar distintos factores del desempeño de los clientes remotos y del *streaming* de datos. Para ello, serán consideradas múltiples métricas de la calidad de servicio (QoS) brindada y la *performance* de la red.

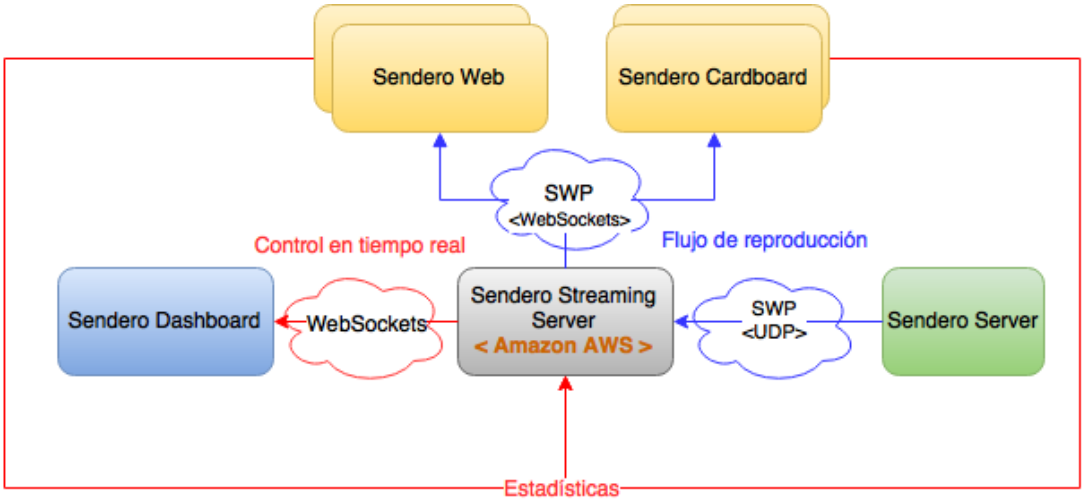


Figura M.9 - Esquema de componentes del subsistema web de Sendero.

La Figura M.9 representa el flujo de datos a evaluar y los componentes involucrados. Sendero Server mezcla y emite a 30 *frames* por segundo el contenido visual en forma de paquetes SWP hacia Sendero Streaming Server a través de un *socket* UDP. Cada paquete recibido es inmediatamente difundido a todos los clientes remotos pero ahora empleando WebSockets. Cada 30 segundos los clientes remotos emiten una serie de estadísticas que son concentradas en una base de datos en Sendero Streaming Server y presentadas en tiempo real por Sendero Dashboard.

Las pruebas presentadas a continuación fueron llevadas a cabo bajo las siguientes condiciones:

- Sendero Server ejecuta en un servidor situado en Montevideo, Uruguay, conectado a una red DSL de 30 Mbps de bajada y 4 Mbps de subida.
- Sendero Streaming Server ejecuta sobre una instancia t2.micro de Amazon AWS con un CPU Intel Xeon de hasta 3.3 GHz y 1 Gib de memoria RAM, localizada en Oregon, Estados Unidos (*rtt* promedio de 230ms).
- Los clientes remotos son evaluados con instancias de Sendero Web (Sendero Cardboard es análogo) y la forma en la que se conectan a la red puede variar en función de la prueba.

Estimación del ancho de banda y recursos del servidor.

Objetivo: Estimar el ancho de banda y recursos necesarios en un servidor para dar soporte al subsistema web de Sendero.

Descripción: Se utilizan las herramientas *tcpdump*[6], *bmon*[7] y *wireshark*[8] para capturar y analizar el tráfico entrante y saliente del servidor donde ejecuta Sendero Streaming Server. De este modo se evalúa el ancho de banda necesario por cliente y se estiman las condiciones para dar soporte a distintas cantidades de usuarios. Al mismo tiempo, se emplea la utilidad *top*[9] para medir el uso de los recursos del sistema, CPU y memoria.

Resultados: La Figura M.10 expone un desglose del tráfico teórico de entrada y salida en Sendero Streaming Server por cliente conectado, sin emplear el mecanismo de compresión implementado (ni la compresión incluida en WebSockets), en el contexto de la obra Barcelona (90 píxeles).

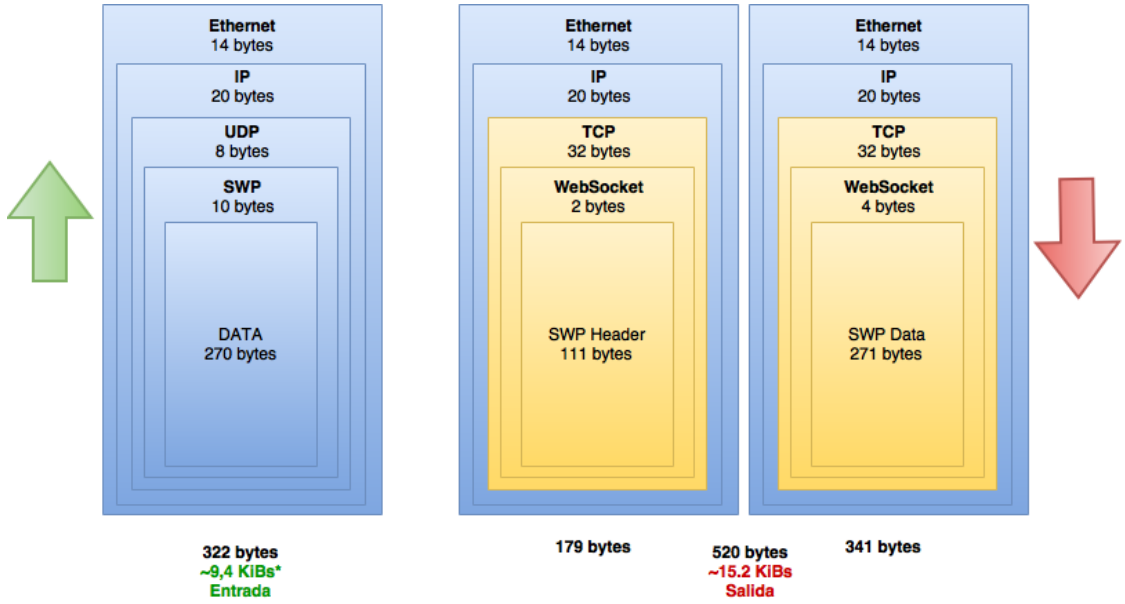


Figura M.10 - Desglose de tráfico Sendero

Nota: La implementación de WebSockets fragmenta la información a enviar por lo que la salida se compone de dos paquetes. El primero envía la cabecera en formato JSON(111 bytes) como parte del procesamiento que realiza Sendero Streaming Server y el segundo envía los datos. Esto provoca la recepción de dos paque-

---

[6] tcpdump, "Tcpdump/Libpcap public repository", 20-sep-2010. [En línea]. Disponible en: http://www.tcpdump.org. [Consultado: 12-jun-2016].

[7] "bmon(1) - Linux man page". [En línea]. Disponible en: http://linux.die.net/man/1/bmon. [Consultado: 12-jun-2016].

[8] "Wireshark · Go Deep". [En línea]. Disponible en: https://www.wireshark.org/. [Consultado: 12-jun-2016].

[9] G. Newell, "top - Linux Command - Unix Command". [En línea]. Disponible en: http://linux.about.com/od/commands/l/blcmdl1_top.htm. [Consultado: 17-jun-2016].

tes de reconocimiento (AKC), lo cual incrementa el tráfico de entrada por cada cliente conectado. Este comportamiento se debe a que la implementación actual envía los datos a través de WebSockets como un objeto JavaScript. Futuras iteraciones en la implementación deberían enviar estos datos como un *array* de bytes para evitar este *overhead* innecesario.

Fuera de este contexto, se mide el tráfico en el servidor (empleando la utilidad *bmon*) y en el cliente para distintas dimensiones de obra en función de la cantidad de píxeles y se. Estos resultados son presentados en la Tabla M.8.

	Servidor		**Cliente**
Cantidad de Píxeles	**Entrada (KiBs)**	**Salida (KiBs)**	**Ancho de banda (KiBs)**
90 (Barcelona)	12.2	15.2	15,6
200	21.8	25.1	25.1
300	30.7	34.7	34.7
500	48.2	52.1	52.1
1000	93.7	99.1	99.1

Tabla M.8 - Ancho de banda con un cliente

Análisis: Con estos resultados se puede estimar el **máximo** ancho de banda requerido por un cliente para la reproducción de Barcelona en ~127 kbps (15.6 KiBs), y para obras con mayor cantidad de píxeles según como se lista en la Tabla M.8. Teniendo en cuenta que los clientes remotos fueron diseñados para funcionar especialmente en dispositivos móviles, y dado que los anchos de banda que manejan las tecnologías de red inalámbrica como WLAN (más de 11 Mbps) o redes celulares 3G (1-3 Mbps) y 4G (3-5 Mbps) superan ampliamente estos valores, la solución propuesta resulta tecnológicamente viable y la reproducción podría ser llevada a cabo exitosamente en múltiples escenarios de red. Por otro lado, es preciso notar que los datos presentados representan una cota superior, que puede ser mejorada utilizando el mecanismo de compresión utilizado.

Por otro lado, la cantidad de clientes que pueden permanecer conectados dependerá de las características del servidor que se utilice. La Tabla M.9 muestra la relación entre el ancho de banda de entrada/salida, el uso de memoria RAM, CPU y la cantidad de clientes remotos soportados para la obra Barcelona en el servidor de prueba.

	Recursos		Ancho de Banda	
Número de clientes	**CPU**(%)	**Memoria**(%)	**Entrada** (Mbps)	**Salida** (Mbps)
1	1.3	4.7	0.0316	0.1278
10	4.7	6.7	0.316	1.278
50	18.6	8.8	1.58	6.39

Tabla M.9 - Análisis de recursos del servidor

En particular, la instancia t2.micro de Amazon tiene un ancho de banda aproximado de 70 Mbps, lo que brinda soporte a un máximo de aproximadamente 550 clientes simultáneos. (Amazon no provee información específica sobre el ancho de banda disponible pero distintos foros dan el estimado mencionado[10]). Por otra parte, la CPU llegaría a su máxima utilización con 285 clientes conectados, mientras que la memoria alcanzaría su límite con aproximadamente 1790 usuarios (cifras estimadas mediante una interpolación lineal). De este modo, la instalación actual tendría la capacidad de soportar hasta unos 285 clientes simultáneos.

Conclusión: Se consiguió una estimación del ancho de banda requerido para el funcionamiento de un cliente remoto ejecutando Sendero Web, y las características que deberá tener un servidor para alojar a Sendero Streaming Server en el marco de la obra Barcelona. En otros contextos, las características del servidor dependerán de la cantidad de público no presencial que se pretenda alcanzar, y en particular de la cantidad de píxeles que contenga la obra.

Calidad de la Reproducción - 1 Cliente

Objetivo: Evaluar la calidad general de la reproducción con un cliente remoto.

Descripción: Se emite contenido visual desde Sendero Server durante un cierto período de tiempo que permita generar más de 50000 *frames*. Se repite el procedimiento 3 veces, en días separados y horarios diferentes para intentar contemplar distintos escenarios de red. Se observa el comportamiento en tiempo real desde Sendero Dashboard y se recuperan los datos de las estadísticas generadas por el cliente para su análisis en el marco de ciertas métricas del rendimiento de la red.

Resultados: La Tabla M.10 muestra el resultado de las 3 ejecuciones de la prueba en términos de las métricas promedio y desviación estándar de: período de generación de paquetes por parte de Sendero Server (ts), período de arribo de paquetes al cliente (arr),

[10] "Amazon EC2". [En línea]. Disponible en: http://serverfault.com/questions/232111/does-anyone-know-the-bandwidth-available-for-different-ec2-instances. [Consultado: 12-jun-2016].

período de planificación de la reproducción en el cliente (pt) y por último el período de reproducción efectivo (rpt). Nota: Estas métricas se encuentran expresadas en función del tiempo transcurrido entre paquetes sucesivos (período de muestreo), en lugar de tasas como es usual encontrar expresadas este tipo de métricas.

Pruebas (~50,000 paquetes)		**ts** (ms)	**arr** (ms)	**pt** (ms)	**rpt** (ms)
Prueba 1	Promedio	33.3	33.5	33.3	33.3
	Des. Estándar	1.9	17.3	3.4	4.0
Prueba 2	Promedio	33.3	33.3	33.3	33.3
	Des. Estándar	0.9	10.2	2.0	2.7
Prueba 3	Promedio	33.4	33.4	33.3	33.3
	Des. Estándar	2.6	11.8	3.5	4.0
Resumen	**Promedio**	**33.3**	**33.4**	**33.3**	**33.3**
	Des. Estándar	**1.8**	**13.1**	**2.9**	**3.6**

Tabla M.10 - Resultados de métricas de reproducción.

Análisis: En primer lugar, el período de generación de paquetes (ts) se comporta como es esperado ya que el *frame rate* configurado en Sendero Server es de 30 *frames* por segundo. El promedio obtenido presenta una desviación estándar de 1.80 ms (5.4%), que puede bien deberse al procesamiento del emisor, o al mecanismo de cálculo empleado para esta estadística que no tiene en consideración la continuidad de los paquetes, haciendo que las pérdidas de los mismos tengan un gran impacto en el valor de esta métrica.

Luego, el período de arribo de paquetes al cliente (arr), mantiene un promedio levemente distinto al de su generación, pero con una desviación estándar ampliamente superior (39.3%), lo cual es justificado por el hecho de que estos paquetes viajan a través de Internet, tanto desde Sendero Server hacia Sendero Streaming Server, como desde este último hacia el cliente remoto (además de lo mencionado antes sobre el cálculo de las estadísticas). En ambos tramos se sufre de problemas de *jitter*, y en particular en el segundo, dado que WebSockets emplea TCP, es posible contar con retransmisiones dando lugar a ráfagas de llegadas de paquetes al cliente. Estos hechos conforman la motivación para la aplicación de las técnicas de *streaming* multimedia estudiadas, y sus resultados se reflejan en el período de planificación de la reproducción (pt).

A pesar de la disparidad de los arribos, la planificación de la reproducción es capaz de absorber estas variaciones planificando el flujo con un período de 33.3 ms y una desviación estándar sustancialmente inferior a 2.97 ms (8.9%). Por otra parte, el período de reproducción efectivo (rpt) posee una dispersión mayor, posiblemente debido a las limitaciones de *JavaScript* para asegurar la ejecución de eventos bajo exigencias de tiempo.

Conclusión: Sendero Server es capaz de generar el *frame rate* apropiado. Las técnicas de *streaming* utilizadas permiten absorber la disparidad de la entrega de los paquetes y de este modo mejorar la calidad de la reproducción, asegurando una tasa de reproducción (*frame rate*) adecuada. Dado que este procesamiento se da en cada cliente, se considera que esta prueba es representativa del comportamiento general de la política de planificación implementada.

Recuperación frente a pérdidas

Objetivo: Evaluar el beneficio del mecanismo de recuperación frente a pérdidas implementado.

Descripción: Dado que el tramo Sendero Server - Sendero Streaming Server se da sobre UDP (vulnerable a pérdidas), y que el tiempo de envío y procesamiento es variable tanto en ese trayecto, como en el que se da desde Sendero Streaming Server hacia los clientes, se implementó un mecanismo de recuperación frente a pérdidas basado en una interpolación para generar los *frames* no recibidos a tiempo. Esta prueba lleva a cabo una medición de la calidad de la reproducción en un escenario donde la tasa de pérdida de paquetes es simulada (25%), y con ello procura evaluar el beneficio de utilizar el mecanismo implementado.

Resultados: Se ejecutan 3 pruebas en las cuales no se tiene en cuenta el mecanismo de recuperación implementado, y luego otras 3 en las que si se considera. Los resultados se muestran en la Tabla M.11, en función del período de reproducción efectivo (rpt).

Pérdida de 25%	**Total de paquetes**	**Paquetes perdidos**	**Paquetes generados**	**rpt(stdev)** ms
Sin recup.	6308	1595	0	45.9(30.4)
	6375	1569	0	45.6(32.7)
	6799	1767	0	46.8(33.2)
Promedio	**6494**	**1643**	**0**	**46.1(32.1)**
Con recup.	6442	1667	1661	33.3(4.7)
	7009	1702	1695	33.3(2.8)
	6541	1632	1630	33.2(2.7)
Promedio	**6664**	**1667**	**1662**	**33.3(3.4)**

Tabla M.11 - Comparación de resultados con/sin recuperación frente a pérdidas.

Análisis y conclusiones: De los resultados conseguidos se desprende que la utilización del mecanismo de recuperación frente a pérdidas implementado ofrece un importante beneficio al momento de asegurar un período de reproducción efectivo dentro de los márgenes deseables (33.3 ms). Por otro lado, la generación de paquetes es levemente inferior a la pérdida, esto se debe a la implementación, que ignora los paquetes cuyo tiempo planificado al momento de llegar está por encima del tiempo actual. Finalmente, esta prueba no tiene en consideración ninguna métrica de la calidad de la experiencia (QoE) que permita determinar si la calidad percibida luego de la interpolación es aceptable para el usuario, no obstante, la prueba si permite evaluar el funcionamiento del mecanismo de recuperación implementado.

Latencia de reproducción

Objetivo: Evaluar la latencia de la reproducción (*end-to-end delay*).

Descripción: Se genera un flujo de datos y se evalúa el tiempo promedio que transcurre desde que cada paquete es emitido hasta su reproducción (*delay*). Se ejecutan 3 pruebas de reproducción y de cada paquete se extrae su tiempo de generación, arribo, planificación y reproducción efectivo. Luego, se calcula la latencia de reproducción como la diferencia de tiempo entre que un paquete es emitido y reproducido, y se presenta un desglose de los elementos constitutivos del *delay* contemplados. Para esta prueba se tuvo en consideración la sincronización de relojes entre Sendero Server y el cliente web.

Resultados: La Tabla M.12 muestra el resultado en términos del promedio y desviación estándar de las tres mediciones realizadas, retardo terminal a terminal (*delay)*, y los componentes considerados incluidos en él, denominados *base* y *jitter.*

Prueba/nº paquetes	**delay** (ms) promedio/stdev	**base** (ms) promedio/stdev	**jitter** (ms) promedio/stdev
1/16914	296.66/10.95	235.88/0.59	8.76/10.83
2/11850	298.42/12.67	236.87/0.53	9.49/12.56
3/14250	291.71/10.94	232.42/1.50	7.28/10.88
Resumen/14338	295.61/11.52	235.06/0.87	8.51/11.42

Tabla M.12 - Latencia y componentes

Análisis y conclusiones: Planificar el tiempo de reproducción para un paquete, o lo que es lo mismo, determinar el tiempo que deberá permanecer en el buffer de reproducción al ser recibido, depende de múltiples factores[11].

El retardo terminal a terminal (*end-to-end delay*) está constituido por un factor constante debido a la diferencia entre los relojes del emisor y receptor, un tiempo variable debido a la preparación del envío en el emisor, un factor constante debido al mínimo tiempo de tránsito en la red, un tiempo variable debido al *jitter* introducido por la red y un factor debido al *clock skew* de los relojes del emisor y receptor. El tiempo denominado *base* en la tabla anterior es el responsable de contemplar la diferencia y el *clock skew* entre los relojes del emisor y receptor y el tiempo mínimo de tránsito en la red. El tiempo denominado *jitter* en la tabla*,* abarca al factor variable introducido por la red. Por otra parte el tiempo variable de procesamiento en el emisor, al igual que en el receptor (procesamiento, descompresión, etc.), son contemplados mediante la adición de un factor extra estimado manualmente (50 ms).

De los datos presentados se desprende que el *delay* promedio alcanza los 295.61 ms, de los cuales un 79.5% lo representa el tiempo *base*, 2.9% el *jitter* y el resto (17.6%) el tiempo adicional para absorber las variaciones en los tiempos de procesamiento en el emisor y receptor.

Por otro lado, los trabajos realizados por Mellouk[12] y Taniar et al.[13] sobre la calidad percibida para aplicaciones de video en tiempo real, exponen que un *delay* que alcanza los

[11] C. Perkins, RTP: Audio and Video for the Internet. Addison-Wesley Professional, 2003.

250 ms es aceptable con un leve impacto visual, mientras que mayor a 250 ms muestra una degradación de la calidad percibible. El valor obtenido durante las pruebas supera este límite, no obstante, el mismo puede ser fácilmente reducido acortando la distancia física entre Sendero Server y Sendero Streaming Server (por ejemplo, alojar a Sendero Streaming Server en un servidor local permitiría disminuir el tiempo base de manera importante, ya que el RTT con un servidor en Uruguay se encuentra en el entorno de los 10 ms, mientras que con el servidor de Amazon es de 230 ms). Otra forma de reducir esta latencia es re-estimando el término adicional añadido, generando un mayor compromiso entre su valor y la tasa de pérdida de paquetes aceptada.

Estos resultados dan la pauta de que se puede mejorar la latencia hasta un nivel aceptable sin mayores dificultades. Por otro lado, los valores tomados como referencia corresponden a video en tiempo real, y deben ser tomados como una aproximación. La realización de pruebas de calidad de la experiencia (QoE) podrían permitir estimar una cota más adecuada para el flujo de Sendero.

Compresión

Objetivo: Evaluar el beneficio en la utilización del mecanismo de compresión empleado.

Descripción: Se genera un flujo de datos (contenido visual) desde Sendero Server utilizando y no utilizando el mecanismo de compresión implementado para varios casos de uso. Se compara el tráfico de datos entrante en Sendero Streaming Server en ambos casos y para cada escenario. Los escenarios que serán considerados son (1) todos los píxeles iguales; (2) uso de la interacción remota implementada, descrita en 4.4.2; (3) una instancia de Sendero Client desarrollada por Pablo Gindel con alta variabilidad de colores generados a partir del sonido[14].

Resultados: La Figura M.11 muestra el resultado de las mediciones realizadas, para los distintos escenarios.

---

[12] A. Mellouk, End-to-End Quality of Service Mechanisms in Next Generation Heterogeneous Networks. John Wiley & Sons, 2010.

[13] I. K. Ibrahim y D. Taniar, Mobile Multimedia: Communication Engineering Perspective. Nova Publishers, 2006.

[14] "Barcelona by Bondi | Palmera blog". [En línea]. Disponible en: http://www.pablogindel.com/2013/06/un-solido-de-catalan/. [Consultado: 15-jun-2016].

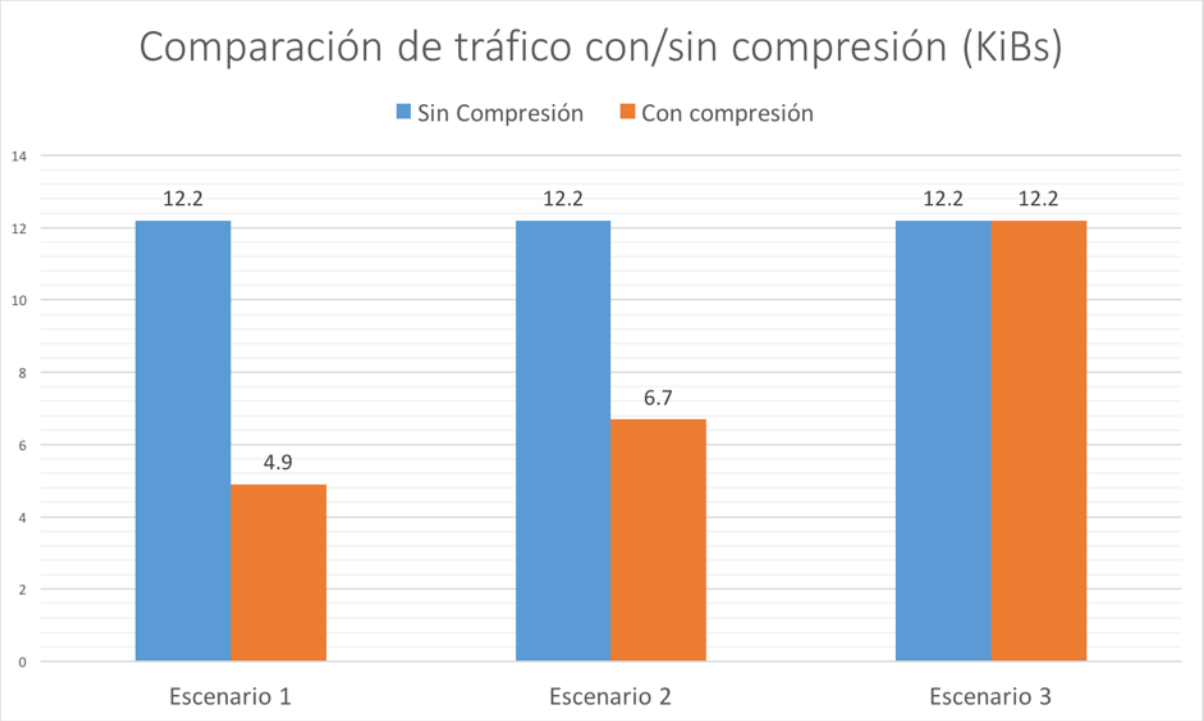


Figura M.11 - Comparación de tráfico con/sin compresión.

Análisis: En primer lugar, cuando no se utiliza compresión, el tráfico entrante en el servidor es el mismo para todos los escenarios, lo cual es coherente con la implementación que envía siempre la información para todos los píxeles de la obra. En cuanto al uso del mecanismo de compresión, se observan mejoras sustanciales en la reducción del tráfico, que corresponden a ~59% y ~45% para los escenarios (1) y (2). Esto puede ser explicado por la naturaleza de estos escenarios, en los cuales existen unos pocos colores para todos los píxeles, y los mismos tienen además una cierta localidad espacial. En lo que refiere al escenario (3), la variabilidad de los colores de los píxeles provoca que la compresión signifique un aumento en la cantidad de información a enviar, por lo cual, la implementación realizada evita la compresión, dando lugar al envío de los datos originales. Se tuvo esta consideración debido a que el contenido comprimido puede ser mayor al no comprimido debido a los metadatos de la compresión.

Por otra parte, la implementación de WebSockets utilizada (socket.io 1.4.5) integra un mecanismo de compresión selectiva, que luego de una etapa de negociación de los parámetros de compresión, es capaz de determinar individualmente que paquetes serán comprimidos y cuales no en base a su contenido. En el contexto de las pruebas presentadas ningún paquete fue comprimido con este mecanismo.

Conclusión: En términos generales, la utilización del mecanismo de compresión empleado resulta beneficioso para reducir el tráfico. Por otra parte, el grado de beneficio dependerá exclusivamente del comportamiento de la obra, en particular de la variabilidad de los colores de los píxeles, y la localidad que estos puedan tener. Dicho beneficio puede ser

variable dentro de un mismo flujo debido a que la compresión se aplica individualmente a cada paquete.

Desempeño de interacción remota

La performance del subsistema de interacción remota depende de múltiples factores tales como:

- Frecuencia de envío y tamaño de los mensajes de interacción que el cliente envía, lo cual impacta en el ancho de banda necesario para poder realizar el procesamiento.
- Del ítem anterior, se desprende la necesidad de una buena conectividad entre los clientes web y el servidor de interacción.
- Cantidad de clientes web emitiendo interacciones de forma concurrente.
- Tiempo de procesamiento de los mensajes de interacción en la instancia de Sendero Client desarrollada. Se debe tener en cuenta que esta instancia encargada de recibir y procesar los datos de interacción, tenga la capacidad de generar el contenido visual a una tasa mayor o igual a la de llegada de los mensajes. De lo contrario, no será capaz de atender largas ráfagas de éstos.

Se realizó una prueba de *stress* para conocer, a través de una interacción de bajo procesamiento, cual es el límite del subsistema en dicho contexto.

Prueba de *stress* de la interacción remota

Objetivo: Evaluar el comportamiento del sistema ante ráfagas constantes de interacciones concurrentes.

Descripción: Se desarrolló un *script* en Node JS capaz de emular la conexión de múltiples clientes web que interactúan con Sendero de forma remota. Cada cliente conectado tiene asignado un único píxel y envía 30 mensajes de interacción por segundo que indican de qué color se desea encender el píxel (blanco o negro). El *script* también recibe (a través del Streaming Server) el flujo generado por estas interacciones y en el arribo de cada *frame*, corrobora si el sistema fue capaz de llevar a cabo la interacción monitoreando el color de cada píxel; cuando estos están coloreados con el color pedido, se considera que la interacción ha funcionado correctamente. Las pruebas fueron realizadas utilizando un conjunto total de 200 píxeles.

Para llevar esto a cabo fue necesario implementar (además del ya mencionado *script*) una instancia de Sendero Client que recibe los mensajes que indican el número de píxel y el color que debe ser asignado a éste. Estos mensajes son leídos de la cola AMQP asignada a esta interacción.

Entorno: El script que oficia de cliente web, el servidor de Interacción, el *broker* AMQP, la instancia de Sendero Client y Sendero Server fueron ejecutados en una misma PC con CPU Intel Core i7-4700MQ de cuatro núcleos de 2.40GHz con HyperThreading

(multithreading simultáneo) y 8 GiB de memoria RAM. Sendero Streaming Server se ejecuta en un instancia de Amazon AWS t2.micro con un CPU Intel Xeon de hasta 3.3 GHz y 1 Gib de RAM.

Resultados: En la Tabla M.13 se presentan los tiempos promedio transcurridos desde que la interacción es emitida hasta que es visualizada en el contenido de la obra recibido (RTT). A su vez, estos valores son reportados para distintas configuraciones de cantidad de clientes, los cuales determinan la cantidad de mensajes por segundo recibidos por el Sendero Client encargado de procesar estas interacciones.

Cantidad de clientes	Mensajes por segundo	RTT (ms)
10	300	286
50	1500	299
100	3000	325
200	6000	332

Tabla M.13 - Pruebas de interacción

Superada la cantidad de 200 usuarios interactuando en simultáneo, el sistema falla. Se constató que los mensajes de interacción emitidos en simultáneo no son recibidos correctamente por el servidor de interacción Sendero Interaction Server.

Análisis y conclusiones: Es destacable en esta prueba que, aún corriendo gran parte del subsistema de interacción en una misma PC, se logra obtener resultados de buena respuesta con 200 clientes interactuando de forma simultánea. Para esta cantidad, se percibe un retardo de procesamiento (se asume el retardo de propagación despreciable debido a que las aplicaciones encargadas del procesamiento de la interacción se ejecutan de forma local) de apenas 46 milisegundos respecto a la prueba con 10 clientes. Teniendo en cuenta que la cantidad de mensajes para satisfacer a los 200 (clientes respecto a 10) es 20 veces mayor (6000 vs. 300), los resultados son considerados satisfactorios.

En la Figura M.12 se observa una captura de pantalla del sitio de gestión de *RabbitMQ* donde se observa el estado de la cola utilizada durante la ejecución con 200 clientes. Notar que la cantidad de mensajes en la cola se mantiene en cero. Esto es debido a la buena tasa de procesamiento de la instancia de Sendero Client, la cual fue capaz de consumir los mensajes de la cola AMQP a una tasa superior a la de llegada.

Figura M.12 - Captura de pantalla del sitio de Gestión de *RabbitMQ* mostrando el estado de la cola durante la ejecución de la prueba con 200 clientes.

Respecto a resultados visuales, en la mayoría de los casos se percibió que los píxeles encendían de forma perfectamente sincrónica entre clientes. En los casos de mayor cantidad de clientes se pudo notar alguna des-sincronización en el cambio de color de los píxeles, pero tan solo en ocasiones aisladas y de forma fugaz.

En la Figura M.13 se observa a Sendero Server durante la ejecución de las pruebas con 100 clientes. Éstos tenían asignado los píxeles numerados del 0 al 99 (de izquierda a derecha en la imagen de la figura).

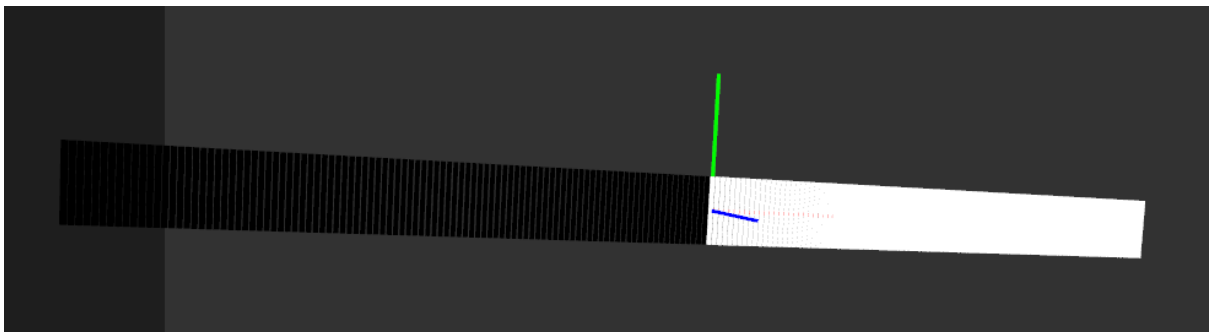


Figura M.13 - Captura de pantalla mostrando la representación 3D de la obra en Sendero Server durante la ejecución de las pruebas (200 píxeles).