Universitat de les Illes Balears

Escola Politècnica Superior

**Memòria del Treball de Fi de Grau**

# Disseny i implementació d'una xarxa domòtica basada en Ethernet Industrial

Diego Navarro Bibiloni

Grau de Enginyería Electrònica Industrial i Automàtica

Any acadèmic 2017-18

DNI de l'alumne: 43218283H

Treball tutelat per Ignasi Furió i Alberto Ballesteros
Departament de Matemátiques i Informàtica

| S'autoritza la Universitat a incloure aquest treball en el Repositori Institucional per a la seva consulta en accés obert i difusió en línia, amb finalitats exclusivament acadèmiques i d'investigació | Autor | | Tutor | |
|---|---|---|---|---|
| | Sí | No | Sí | No |
| | X | | X | |

# Índice

# Resumen

Hoy en día la cantidad de sistemas empotrados que rigen la gran mayoría de nuestras actividades es abrumadora. Ya no sólo se encuentran en el ámbito tecnológico e industrial, si no que cada vez más se integran estos dispositivos a las actividades cotidianas de la gran mayoría de la población. Esta variedad de sistemas empotrados distribuidos conlleva un problema, ya que al ser entre sí diferentes, cada uno tiene unos requisitos y necesidades distintas.

Gracias a un paradigma de la universidad de Aveiro llamado FTT y más específicamente una implementación de éste, HaRTES, se puede dotar a redes ethernet, cada vez más presentes en multitud de ámbitos, con la capacidad de soportar tráfico con requisitos de tiempo real.

En este trabajo se realizará la implementación de una red en la casa *Ca ses Llúcies*, un proyecto de restauración cerca de la universidad que dotará de aplicaciones domóticas, de control y multimedia a la casa. Como cada aplicación necesitará una comunicación distinta, demostraremos que somos capaces de integrar los distintos tráficos en la misma red implementada en la casa.

# 1. Introducción

En este capítulo introductorio, se comenta en primera instancia cuál es el entorno y las motivaciones que nos llevan a realizar el proyecto. Después se enumeran las tareas necesarias para dar por satisfechos dichos objetivos y finalmente, se comenta la estructura de la memoria y una explicación concisa de sus partes.

## 1.1 Entorno y motivaciones del proyecto

Hoy en día, cada vez se utiliza más la palabra *smart*. Empezaron los teléfonos móviles creando una auténtica revolución en la sociedad occidental, y después esta idea *smart* se fue extrapolando a otros objetos cotidianos, ya sean relojes, coches o televisiones.

En la actualidad se utiliza el término *smart-home* para referirnos a una vivienda dotada de automatización. Las aplicaciones son infinitas, ya que van desde levantar las persianas al amanecer para despertarnos hasta realizar tareas de control para mantener una temperatura estable en la vivienda o incluso llevar un inventario en la nevera.

Esta automatización requiere de un tráfico de datos de los diversos aparatos que automatizan las acciones anteriormente realizadas por humanos y sus respectivos sensores. Para ello los datos suelen canalizarse en forma de bus, ya que requiere menos cableado. Además, suelen ser varios sistemas federados cada uno encargado de una función concreta, ya sea climatización, vídeo, luces... El problema reside en canalizar todos los datos por la misma red y que ésta sea capaz de dar soporte a los diferentes tipos de tráfico.

Para solucionar este problema, se puede utilizar un paradigma de comunicación denominado *Flexible-Time Triggered*. FTT es un paradigma que nos permite dotar una red física, por ejemplo, una red ethernet, de mecanismos para llevar a cabo una comunicación con requisitos de tiempo real. Más adelante se entrará en

detalle para explicar el paradigma, pero las principales ventajas que otorga FTT son que nos permite dotar a una red de comunicaciones, únicamente cambiando su método de acceso al medio, con la capacidad de soportar tráfico de tiempo real y una flexibilidad que permite a la red adaptarse a los estímulos exteriores.

Además, FTT tiene la capacidad de integrar diferentes tipos de tráfico por la red en una red de diferentes dispositivos de los cuales cada día se depende más. Unos necesitarán una red con requisitos de tiempo real, otros necesitarán simplemente utilizar datos de internet sin requisitos de tiempo real, y otros tendrán simplemente que enviar un dato en el momento en el que se pulse un botón.

Más específicamente, se utilizará HaRTES, que es una implementación de FTT sobre una red en topología de estrella, que además utiliza el *switch* central para facilitar la integración de tráfico ajeno al paradigma FTT. En la sección 3 se explicará HaRTES más en detalle.

Este tipo de soluciones se irá implementado cada vez más y más, ya que hoy en día el avance de los sistemas empotrados y la interconexión entre ellos es inexorable. La inmensa mayoría de procesadores del mundo se encuentran en sistemas empotrados, todos realizando tareas diversas y con requisitos diferentes, y es cada vez más necesario el intercambio de información entre ellos.

## 1.2 Objetivos del proyecto

El objetivo del proyecto es demostrar que se pueden realizar comunicaciones entre diferentes aplicaciones, pero utilizando únicamente un mismo canal de comunicaciones FTT compartido entre todas las aplicaciones para dotar a *Ca ses Llúcies*, un proyecto de rehabilitación constructiva y energética de un edificio situado en una de las entradas de la UIB, de una red que soporte diferentes tipos de tráfico. De esta manera, se logra canalizar los datos por el mismo canal, aunque sean datos completamente distintos, ya que cada aplicación tendrá una funcionalidad diferente y necesitará unos requisitos de tiempo real u otros dependiendo de las necesidades de comunicación y de la criticalidad de la tarea.

Para simular diferentes tipos de tráfico, se ha decidido plantear tres tipos de aplicación diferentes que utilicen el mismo canal FTT.

La primera aplicación que se implementará es una aplicación domótica sencilla, que se encargará de llevar un estímulo obtenido en un sitio a otro, en el que se realizará una acción en función de este estímulo. El tipo de tráfico de esta aplicación es de tiempo real no estricto, ya que debe llegar al otro nodo, pero si se retrasa seguramente no será percibido. Un ejemplo de este tipo de tráfico podría ser activar el limpiaparabrisas de un coche. El canal FTT en este caso sólo deberá llevar un

dato obtenido en un nodo, por lo que no es necesario un gran ancho de banda ni tampoco que la frecuencia con la que se envían los mensajes sea muy elevada.

La segunda aplicación deberá realizar una tarea de control para mantener una planta estable. Para ello deberá recibir mensajes de la planta, que transportarán los valores de posición y ángulo actuales para compararlos con los valores de consigna y obtener el error. Después deberá aplicar un algoritmo de control para calcular la actuación necesaria para llevar a la planta al estado deseado, y después enviar el valor de actuación de nuevo a la planta para corregir el error. Para realizar esta aplicación se deben utilizar diferentes comunicaciones, que pueden ser clasificados en dos tipos principalmente.

- El primer tipo de comunicaciones que se abordan son las comunicaciones entre la planta y los nodos que serán utilizados.

- El segundo tipo son las comunicaciones FTT internas entre los diferentes nodos. Como cada nodo tendrá funciones diferentes, deberán intercambiar información para que cada uno pueda realizar su función. Por ejemplo, si un nodo obtiene los valores de la planta, luego deberá pasarlos a otro nodo para que realice el control.

La tercera aplicación de la red transmitirá datos multimedia, como podría ser un vídeo en *streaming*. Una aplicación así no suele requerir un ancho de banda muy grande, y gracias a las características del *switch* o conmutador que se mostrará a continuación, es fácil integrar este tipo de tráfico en la red. Así se demostrará que se puede integrar tráfico que no utiliza el paradigma FTT en la red.

Es importante definir lo que es un *switch* antes de proceder. Un *switch* es un dispositivo lógico que permite la interconexión de dos o varios equipos, y es responsable de la transmisión de información entre ellos.

## 1.3 Tareas

A continuación se describen brevemente las tareas que se han realizado para demostrar que pueden ser integrados diferentes tráficos de datos en la red:

- Lectura y documentación sobre FTT y HaRTES: La primera del trabajo parte fue leer la información sobre FTT que fue proporcionada para ir familiarizándose con el paradigma, las normas y conceptos clave necesarios para poder luego realizar las comunicaciones entre nodos.

- Comunicación simple entre dos nodos: Una vez obtenidos los conocimientos básicos, el siguiente paso fue realizar una comunicación sencilla entre dos nodos.

Para ello se tuvieron que tener en cuenta las características del paradigma para transmitir la información, ya que, como se explicará más adelante, se necesita la creación de una conexión virtual para que ambos nodos transfieran información

- Diseño de las tareas: Una vez obtenidas las bases para empezar a programar con FTT, se puede empezar a diseñar el funcionamiento de las aplicaciones de la red en función de las características del paradigma explicadas más adelante y de las necesidades que tendrán las tres aplicaciones.

- Programación de la aplicación domótica: Una vez diseñado el funcionamiento y establecidas las necesidades de la primera tarea, se puede proceder a la programación. Después, se realizan las pruebas pertinentes para ver que realiza bien su trabajo y se procede a la siguiente aplicación.

- Programación de la aplicación de control: La aplicación de control requiere un poco más de trabajo que la anterior, ya que tiene unos requisitos más exigentes. Una vez programada se debe comprobar que funciona correctamente.

- Instalación del tráfico multimedia: Tras haber verificado el correcto funcionamiento de las tareas, se realizan las conexiones necesarias para que pueda fluir por la red tráfico que no utilice FTT y que sea dirigido por el *switch*.

- Unificación de las comunicaciones: Una vez terminadas las tres aplicaciones, se unifican todas en el mismo código y se realiza una prueba conjunta una vez más, para ver que todo funciona correctamente.

## 1.4 Estructura del documento

En el resto del documento se profundiza más en los aspectos comentado hasta ahora, dividiendo el contenido en cinco partes.

La primera es la parte de fundamentos y trabajo previo. En esta se explicarán las partes más importantes de FTT, HaRTES, que es una implementación de FTT sobre un *switch* cuyos detalles se comentarán más adelante, y se mostrará el punto de partida del proyecto en cuanto a código proporcionado por los tutores.

La segunda parte es la etapa de diseño. En esta sección se comentará en detalle el proceso de diseño de las aplicaciones y de sus características para poder utilizarlas en la red FTT.

La tercera sección es la de implementación. Se indicará cuál ha sido el proceso de implementación de las aplicaciones, las decisiones tomadas en función de los problemas que iban surgiendo, y cómo se han acabado solucionando éstos.

La cuarta parte es la de validación de la red. En ésta se observará si se han cumplido los objetivos especificados previamente tras haber realizado la implementación de la red diseñada. Se realizarán diferentes pruebas para verificar que la funciona de manera correcta.

La quinta y última sección es la de conclusiones, constando de un resumen del trabajo realizado y de los objetivos logrados, y una opinión personal.

# 2. Fundamentos de FTT (Flexible-Time triggered)

Como se ha comentado anteriormente, FTT es un paradigma de comunicación propuesto por la Universidad de Aveiro que permite el tráfico de datos en tiempo real para los sistemas empotrados distribuidos (*distributed embedded systems* en inglés o DES) de manera flexible, permitiendo que se realicen cambios en los requisitos de la comunicación si la situación así lo requiere. Necesita un despliegue de red a nivel de capa física del modelo OSI, como mínimo, para sustentar la transmisión de datos, ya que FTT trabaja en las tres capas superiores a dicha capa. En este trabajo se utilizará una red ethernet con topología de estrella y un *switch* que centraliza todas las comunicaciones, facilitando el control de las comunicaciones enormemente.

La arquitectura de FTT está compuesta por un conjunto de nodos denominados esclavos, que son los diferentes nodos encargados de transmitir, procesar o recibir información; y un nodo maestro, que en este caso coincide con el *switch* y, que se encarga de organizar las comunicaciones y de administrar los recursos de la red. Un esquema simple de la arquitectura de una red con topología de bus se muestra en la figura 1.


*Figura 1: Arquitectura FTT*

## 2.1 Elementary Cycle (EC)

El *elementary cycle* es la división temporal que hace el maestro para que los esclavos transmitan y reciban sus mensajes. Cada *elementary cycle* tiene unas ranuras determinadas para cada tipo de tráfico, y es un mecanismo de

sincronización entre las diferentes aplicaciones. Se estructura según la siguiente figura [1]:

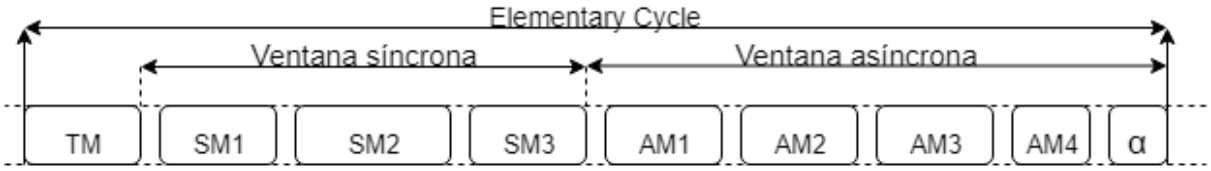


*Figura 2: Elementary Cycle*

La primera parte y la más importante es el *trigger message* (TM). El maestro, al principio de cada EC, envía un TM a todos los esclavos conectados a la red. El *trigger message* contiene datos para los que los esclavos transmitan sus mensajes síncronos en orden. De esta manera todos los nodos están sincronizados al maestro y saben cuándo pueden transmitir sus tramas [1].

La segunda parte es donde se transmiten los datos entre los nodos, que se divide en dos partes. La ventana de transmisión síncrona, que sirve para transmitir mensajes periódicos o síncronos entre los diferentes nodos; y la ventana de transmisión asíncrona, que sirve para enviar tráfico aperiódico.

Por lo tanto, el patrón de comunicaciones definido por FTT sería el mostrado en la siguiente figura. Primero el maestro marca el inicio del *elementary cycle* gracias al TM, que reciben todos los esclavos (Figura 3a). Después empieza la ronda de comunicaciones periódicas donde todos los nodos intercambian datos necesarios para sus aplicaciones de acuerdo con la planificación contenida en el TM (Figura 3b). Después se da paso a la ventana asíncrona, donde los nodos, esclavos y maestro, intercambian datos entre sí sin una organización a priori (Figura 3c). Estos mensajes además de servir para las aplicaciones ejecutándose en los diferentes nodos le sirven al maestro para coordinar la transmisión y satisfacer las necesidades de los nodos. Se añade al final del EC una ventana de guarda para los nodos más lentos, ya que pueden sufrir una pequeña desincronización respecto al maestro (Figura 3d) [1].

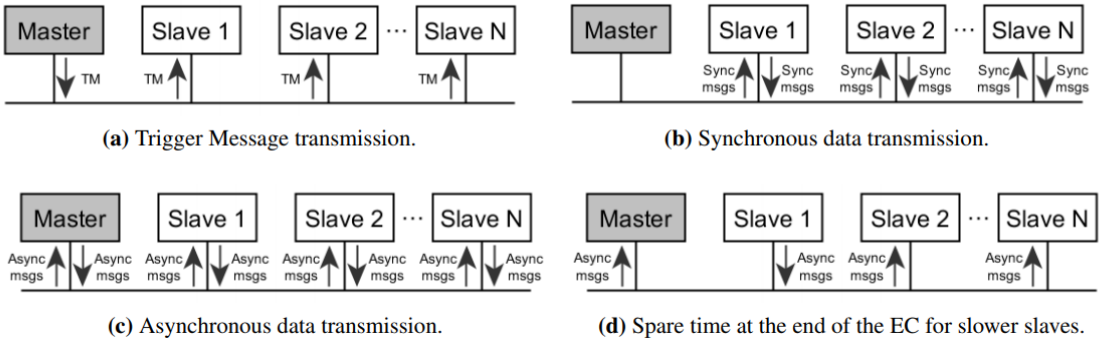


*Figura 3: Esquema de comunicaciones FTT [1]*

## 2.2 Mensajes FTT

Hay dos tipos de mensajes en FTT. Mensajes de datos y mensajes de control. Los primeros se intercambian entre esclavos y simplemente transmiten datos entre nodos esclavos pertenecientes a las aplicaciones que se ejecutan en cada uno de ellos. Los mensajes de control se utilizan para gestionar las comunicaciones. Estos mensajes son, además del TM, los siguientes tres: *Plug-and-Play messages*, *slave request messages*, *Master command messages*. Se muestran todos los tipos de mensajes en la siguiente figura [1]:

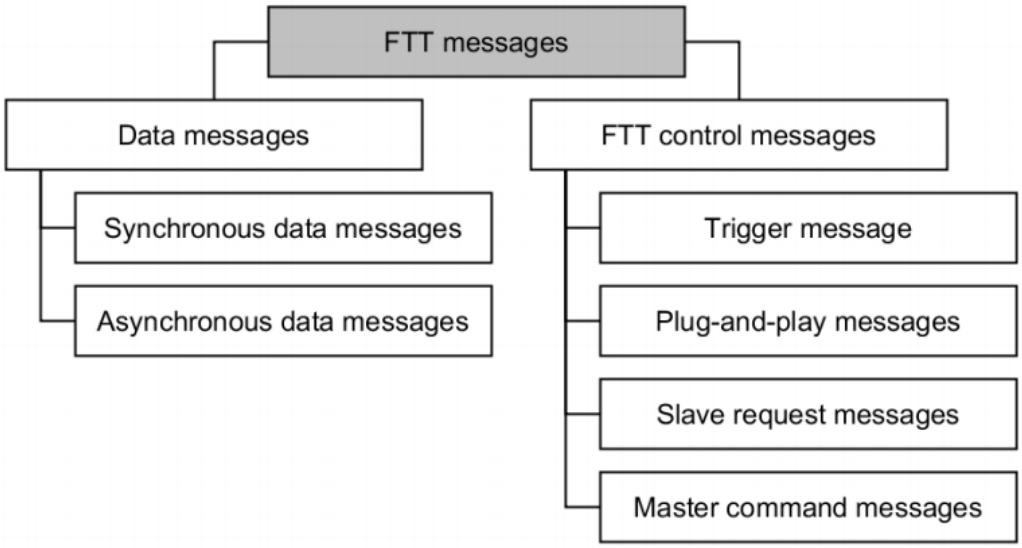


*Figura 4: Tipos de mensaje FTT [1]*

Los mensajes de Plug-and-Play se utilizan para implementar el mecanismo *plug-and-play*. Más específicamente, un esclavo que quiera unirse a la red FTT deberá pedir permiso al maestro mediante este tipo de mensaje, y cuando el maestro acepte la petición, enviará al esclavo el estado actual de la red. Cabe destacar que en la

implementación de FTT llevada a cabo en este trabajo este tipo de mensajes no existe, la conexión de los nodos no se realiza durante la ejecución del programa. Los mensajes de *slave request* sirven para que los esclavos puedan pedir una modificación del estado actual de la red, por ejemplo, un aumento en el ancho de banda debido a un inusual crecimiento en el tráfico de datos. El mensaje de tipo *Master command* permite al maestro responder a las peticiones de los esclavos y transmitir a todos los nodos los cambios realizados en la red.

## 2.3 *Streams* FTT

Los mensajes en FTT se encuentran confinados en canales de comunicaciones virtuales llamados *streams*. Los nodos esclavos que quieran comunicarse deben conectarse a un *stream* como transmisores o receptores. Un *stream* FTT sólo puede tener un nodo emisor conectado a un *stream*, mientras que puede tener varios nodos receptores por *stream*. De esta manera FTT puede transmitir sin tener que preocuparse por la transmisión punto a punto, es decir, el receptor no sabe cuál es el emisor y viceversa.

Los *streams* son gestionados por el nodo maestro, aunque las operaciones son activadas por los nodos esclavos. Más específicamente, la creación, modificación o destrucción de *streams* FTT se pide desde los nodos esclavos, así como el acople o desacople de un nodo a un *stream*. [1]

## 2.4 Flexibilidad en FTT

La flexibilidad en FTT viene dada por las siguientes características [1]:

- El maestro incluye un on-line *scheduler*, que calcula la planificación dinámicamente de la parte síncrona cada EC, dependiendo de las necesidades de los nodos a diferencia de un planificador tradicional, que tiene una planificación fija.

- Los esclavos únicamente deben leer el TM del maestro y limitarse a seguir el orden que se les impone.

- Los *streams* se gestionan desde el maestro, pero por medio de mensajes de petición por parte de los esclavos. De esta manera puede adaptar la red a las necesidades de los esclavos, en función de los recursos disponibles.

- El sistema *PnP* permite a los esclavos adherirse a la red sin necesidad de tener conocimientos previos sobre el estado actual de la red. El maestro se

encarga de transmitir el estado y las modificaciones o actualizaciones en la red.

- El maestro es capaz de gestionar los recursos de la red desde un punto de vista de calidad de servicio (*QoS*). Esto significa que es capaz de modificar los *streams* con información obtenida desde el punto de vista del usuario, los nodos, otorgando los recursos mínimos a cada uno y una máxima eficiencia de la red.

# 3. Hard Real-Time Switching Ethernet (HaRTES)

Hoy en día Ethernet está enormemente extendido alrededor del globo gracias al gran ancho de banda que proporciona. El problema es que no proporciona requisitos de tiempo real y su fiabilidad es limitada, lo que se puede arreglar si se utiliza FTT, más específicamente HaRTES.

HaRTES se basa en el paradigma FTT, con lo que las comunicaciones utilizan todas las normas mentadas anteriormente, pero en vez de utilizar un *switch* ethernet al que se le conectan los diferentes nodos en topología de estrella, se utiliza el mismo *switch* como nodo maestro, al que se le comunican los esclavos para transmitir en forma de estrella. Así, todas las comunicaciones pasan por el *switch* y este puede llevar un control riguroso de éstas [2].

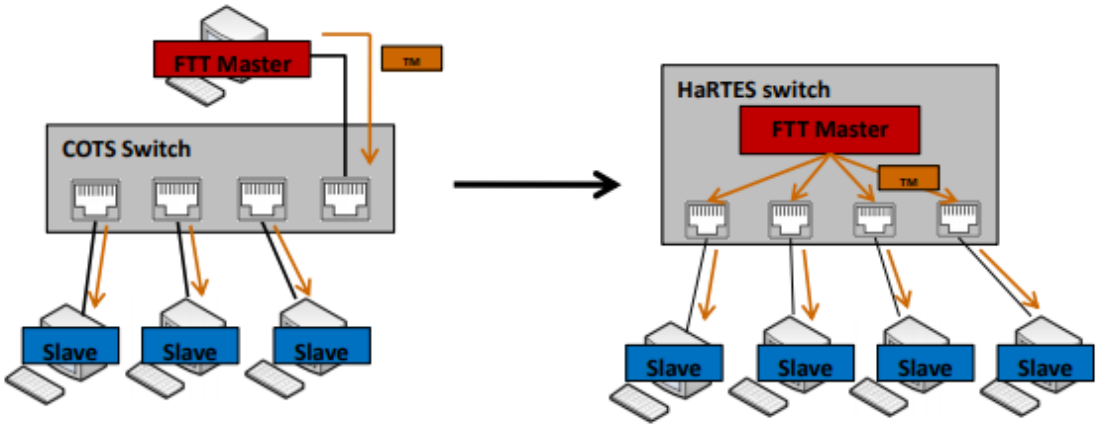


*Figura 5: Comparación switch Etherney y switch HaRTES [2]*

Las características más importantes son las siguientes:

- El tráfico asíncrono se envía de forma automática por los esclavos en vez de ser sondeado por el maestro. El *switch* tiene la capacidad de guardarlo en sus bancos de memoria y enviarlo más adelante al destinatario, incluso si la información no utiliza el paradigma FTT para comunicarse.

- Transmisiones no autorizadas o erróneas pueden ser fácilmente restringidas, ya que el *switch* puede cerrar uno o varios de sus puertos si un nodo se comporta de forma errónea. Por ejemplo, si un nodo fallase, el *switch* podría contener la información transmitida por ese *switch* y así proteger a los demás de crear más errores.

- Ausencia de colisiones, ya que cada nodo tiene una conexión dedicada y no deben compartir el medio de comunicación. El *switch* se encarga de organizar las comunicaciones con cada nodo.

- Integración de tráfico que no utiliza el paradigma FTT, sin poner en riesgo los servicios de tiempo real.

# 4. Trabajo previo

En éste apartado se comentará que componentes, ya sean hardware o software, se han heredado y que por lo tanto no deben considerarse parte de éste trabajo.

Para realizar este trabajo se han utilizado varios programas como punto de partida para realizar la red, ya que el objetivo principal de este sólo es canalizar el tráfico de datos de diferentes aplicaciones a través de la misma red. La parte más importante heredada es el código de HaRTES.

Como punto de partida se ha utilizado una implementación software de un *switch* HaRTES, con todas las primitivas necesarias para realizar las comunicaciones. Gracias a ello se pueden utilizar primitivas necesarias para realizar las transmisiones de los mensajes, crear *streams*, y además es un *switch* que gestiona las comunicaciones entre nodos. Ese código es la base de partida, y el trabajo realizado se implementa en forma de aplicaciones sobre esta implementación software.

Luego se ha utilizado una planta simulada en Simulink para realizar la tarea de control. Se trata de un péndulo invertido, que es básicamente una base móvil que sostiene un elemento alargado que se va inclinando por la acción de la gravedad. esta planta debe ser controlada por un controlador PID, que se encargará de corregir el error para mantener el péndulo derecho.

También se tiene como punto de partida el código de los PID que se encarga de realizar el control, además de sus constantes K para dos controladores diferentes, uno para corregir la posición y otro para corregir el ángulo. Más adelante se explicará el funcionamiento de los PID.

Además se ha obtenido el diseño de la planta de los tutores del trabajo. Éste diseño vino dado en un documento de guía proporcionado al principio del trabajo.

Por último, también se tienen las comunicaciones UDP entre el nodo que contiene el Simulink y los nodos que funcionarán como muestreador y como actuador ya que es necesario transmitir datos entre las aplicaciones y la planta Simulink.

# 5. Diseño de las aplicaciones

Tras haber introducido todos los conceptos necesarios para comprender el contexto en el que se va a trabajar, se puede empezar a hablar del diseño de las aplicaciones que se ejecutarán en la red representando el tráfico de datos de una vivienda inteligente. Para ser más realistas se debería haber creado una gran cantidad de aplicaciones para que el tráfico de datos fuera más denso y las exigencias a la red fueran más elevadas, ya que sería el tráfico real de una casa inteligente. Como no era posible programar tal cantidad de aplicaciones, se optó por realizar tres distintas, cada una con exigencias diferentes, siguiendo el siguiente esquema propuesto por los tutores:

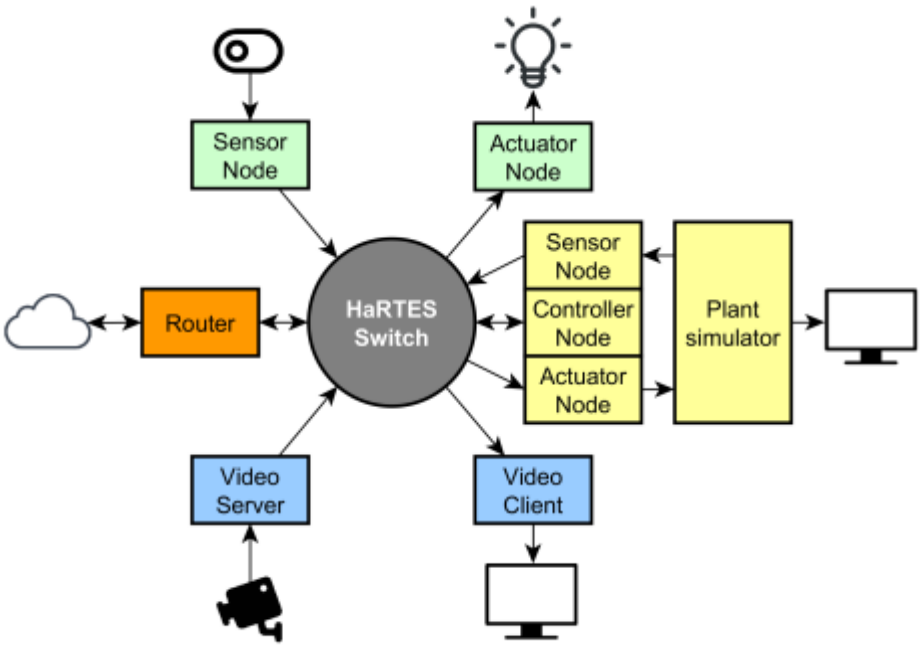


*Figura 6: Esquema de la red*

A continuación, se especificará el diseño de cada una de las tres aplicaciones, cuáles son sus funciones, que requisitos tienen, y con qué podrían ser comparadas en una vivienda real.

## 5.1 Aplicación domótica

La primera parte de la red son las aplicaciones domóticas, que suele ser la más específica de las casas inteligentes ya que funciona para automatizar una acción de la casa cuándo un estímulo es captado por un sensor, un interruptor, una palanca, un botón.... En este caso, se programarán los nodos para que uno reciba una señal analógica que le indique que el estímulo ha sido realizado en la casa y envíe la información al otro nodo, que transformará la información en una salida analógica.

Este tipo de aplicación tiene infinidad de usos, ya que es una manera simple de enviar información de un lado a otro. Se podría utilizar para abrir puertas con un botón, encender la luz con una palmada, o activar algún electrodoméstico con el móvil.

Para simular una aplicación de este estilo, se ha decidido utilizar dos nodos. Uno de ellos se ocupará de recibir la información del exterior. Para recibirla necesita acceso a los pines de entrada-salida del ordenador, en este caso para leer el valor analógico de este. Una vez leído lo transmitirá vía FTT al segundo nodo, encargado de transformar el valor recibido en un valor analógico en una de sus interfaces de entrada-salida, para realizar un estímulo en el exterior.

En un principio la idea era utilizar un sensor de presencia que nos diera un valor de voltaje cuándo notara una persona en las inmediaciones. El nodo leería ese valor, lo transmitiría al segundo nodo, y este se encargaría de poner uno de sus pines a cinco voltios para activar un relé que permitiría el paso de corriente para encender una bombilla.

La aplicación simula un interruptor automático en una casa, activado cuando una persona entra en una habitación. Es una aplicación sencilla de domótica, pero muy útil. También cabe decir que para que la aplicación pudiera implementarse en una vivienda real, tendría que tener en cuenta la luminosidad ambiente, ya que la luz no debe encenderse si no es necesario.

Para simplificar el modelo, al final se decidió realizar una configuración similar pero más sencilla a nivel de montaje, ya que utilizar relés y la red eléctrica es peligroso en cierta medida, y los sensores de presencia no eran baratos.

Finalmente se decidió en hacer un esquema que se componía de un pulsador, que nos daría un valor de entrada a leer en el primer nodo, y un LED que se encendería si el segundo nodo le transmitía un valor de voltaje alto. El diseño es más sencillo, pero el tráfico de datos es idéntico, y la funcionalidad también, por

lo que se considera que es un diseño adecuado para conseguir los objetivos. Para añadir una funcionalidad más real a la luz, se le añade un valor de *time out* o fin de cuenta, que sirve para apagar la luz si pasa un cierto tiempo sin recibir un valor del nodo 1. De esta manera se simula que ya no hay presencia de personas en la habitación, y por razones de ahorro energético se apaga la luz después de un cierto tiempo.

### 5.1.1 Arquitectura de comunicaciones de la aplicación domótica

Esta primera parte tiene unas comunicaciones bastante sencillas. Primero se necesitan dos nodos además del *switch*. Un nodo se encargará, primero, de crear un *stream* entre los dos nodos para poder realizar las comunicaciones, y después de muestrear los valores de entrada proporcionados por las interfaces de entrada. Una vez obtenido el valor, le enviará el valor obtenido al segundo nodo a través del canal creado al principio. El segundo nodo recibirá el valor que le ha enviado el primer nodo, y en función del valor que haya recibido, pondrá una de sus salidas a uno o a cero encendiendo o apagando el LED.

Los datos transmitidos en esta aplicación son simplemente unos o ceros, lo que hace que la comunicación sea bastante sencilla. Además, sólo se utiliza un único canal de comunicaciones ya que sólo hay dos nodos y tan solo es necesaria una comunicación de tipo símplex.

El ancho de banda de esta primera aplicación es bastante pequeño debido a la poca cantidad de datos que se requieren y a la escasa necesidad de una frecuencia de envío alta, ya que a ojos humanos, que el LED tarde en encenderse 6 milisegundos o 150 milisegundos es despreciable.

Cabe decir que en un primer diseño se intentó realizar esta comunicación con mensajes asíncronos, que sería la manera más eficiente de realizar esta tarea en vez de enviar periódicamente el valor leído del GPIO, pero las primitivas necesarias para crear *streams* asíncronos y enviar mensajes de dicho tipo daba errores, y no era posible su implementación.

Un GPIO (*Generic Purpouse Input/Output)* es un pin genérico en un chip cuyo comportamiento puede ser controlado por una persona durante el tiempo de ejecución. En este caso se trata de los GPIO de los ordenadores que funcionan como nodos, que se utilizarán para leer o escribir dependiendo de la función del nodo.

## 5.2 Aplicación de control.

La segunda aplicación que se ha escogido para la red es una aplicación de control. Las aplicaciones de control se caracterizan por tener un flujo de datos constante, ya que deben estar constantemente procesando los datos que le llegan del muestreo además de recibirlas de los sensores y enviar los resultados al actuador. Si se reduce el ancho de banda necesario de este tipo de aplicaciones y los mensajes enviados fueran pesados, normalmente resulta en un control más lento que muchas veces desemboca en un descontrol de la planta e inestabilidad. Cuanta más frecuencia tenga la ejecución del algoritmo de control, el muestreo y la actuación, más preciso es el control.

Se ha utilizado una simulación *hardware in the loop*, que utiliza una planta simulada con modelos matemáticos para comprobar que se realiza el control correctamente y que las comunicaciones son correctas sin la necesidad de tener una instalación real de toda la planta. Como planta simulada se ha utilizado un software programado en Simulink que funciona como un péndulo invertido en un cuarto nodo. Para que se mantenga derecho, necesita un flujo constante de datos con una frecuencia mínima determinada, ya que si los datos de actuación tardan demasiado en llegar, el péndulo podría caer. Para mantenerlo derecho se ha utilizado un par de controladores PID que en función de la posición de la base y el ángulo del péndulo calcularán las actuaciones pertinentes para volver al valor de consigna.

Se debe hacer hincapié en que ésta es una aplicación que pondrá a prueba la los servicios de tiempo real de la red FTT. El péndulo podría seguir funcionando si algún mensaje se perdiera, pero si una gran cantidad de ellos no alcanza su destino el péndulo podría dejar de funcionar. Aunque se utilice una planta simulada y las consecuencias no sean graves, no se puede decir lo mismo de otras aplicaciones de control que nos mantienen a salvo día a día, como pudiera ser el sistema ABS de los vehículos de uso personal. A continuación se muestra el esquema de conexiones físicas de la aplicación

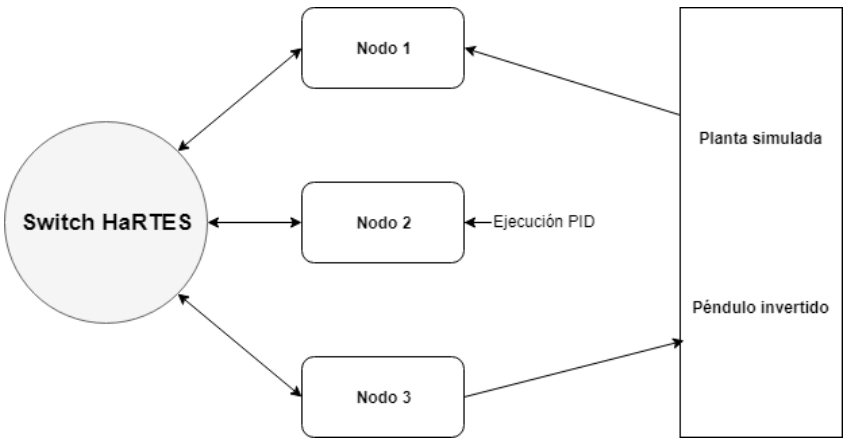


*Figura 7: Esquema de la aplicación de control*

### 5.2.1 Arquitectura de comunicaciones de la aplicación de control

A diferencia de la aplicación anterior, ésta requiere utilizar un nodo más, y en consecuencia, un *stream* más ya que se tienen dos comunicaciones FTT símplex entre los nodos del sistema con diferentes orígenes y destinos, una del primero al segundo, y otra del segundo al tercero. Se podría realizar la misma funcionalidad con un solo nodo, que realizara las tres tareas de la aplicación, pero entonces no se tendría una implementación realista de un sistema distribuido, ya que en muchas plantas reales no se tiene el sensor, la CPU y el actuador localizados en el mismo sitio, si no que suelen estar separados y deben transmitir los datos entre sí.

Para que funcionen de manera adecuada, los nodos necesitarán una sincronización para realizar sus tareas en orden cíclico. Si no se sincronizan los nodos, podría ser que muestreara el nuevo valor de error del péndulo después de enviar un mensaje con un error obsoleto. Para lidiar con este problema, se utiliza como método de sincronización el número de EC. De esta manera se podrá organizar a los tres nodos en el orden adecuado para realizar adecuadamente el control del péndulo.

Un detalle importante es que los nodos de muestreo y de actuación se comunican con la planta simulada mediante otra red, utilizando el protocolo UDP el cual se explicará más adelante en la sección de implementación.

El primer nodo funciona como nodo de muestreo, simulando que recoge datos del entorno para que la aplicación. En este caso recogerá los datos del canal UDP que viene desde la aplicación del péndulo invertido. Como se ha anunciado anteriormente, depende de la importancia de la aplicación de control, el muestreo debe ser más o menos frecuente. A esta aplicación se le da una gran importancia al módulo de control para que el tráfico sea pesado y para controlar el péndulo de la manera más segura posible. Para finalizar, los datos deben ser transmitidos al siguiente nodo, que será el encargado de calcular la actuación necesaria para corregir el error, gracias al PID.

El segundo nodo es el encargado de calcular la actuación acorde a los PID. Para ello se tiene una función denominada pid_update, que pasándole como parámetros los valores de error de ángulo y posición nos devuelve el valor de actuación acorde a los valores K de cada parte del controlador. Los valores de inclinación y posición del péndulo llegan desde el primer nodo, y con ellos debe calcular cómo debe mover la base del péndulo para equilibrarlo con la función que se ha nombrado. Posteriormente debe pasar el valor de actuación al tercer nodo.

El tercer nodo debe comunicarse de vuelta con la planta para que ésta corrija el error. Para ello recibe la actuación del nodo dos y, por otro canal UDP, comunica la decisión de los PID a la planta para que ésta actúe en consecuencia.

Para tener el concepto claro, un controlador PID es un algoritmo de control por realimentación que basa su resultado en sumar tres actuaciones distintas, obtenidas por tres parámetros diferentes y el valor del error que pretende corregir el controlador. El primero es el proporcional, que depende del error actual. El segundo es el integral, que depende de los errores pasados. El tercero es el derivativo, y es una predicción de los errores futuros. Ajustando la importancia que se le da a cada parámetro, se puede calibrar el controlador para obtener una planta estable.

## 5.3 Aplicación multimedia.

Para realizar la implementación del tráfico menos prioritario por la red HaRTES se decidió que la reproducción de un vídeo a través de la red sería más adecuado que realizar una implementación cliente-servidor. Este tipo de tráfico requiere un ancho de banca elevado, pero no tiene una prioridad alta y se puede ejecutar de forma aperiódica cuando el *switch* considere que es correcto transmitir los datos. Si se tuviera que prescindir de una transmisión, sería de ésta ya que no tiene requisitos de tiempo real. Para realizar esta parte, se decidió documentarse primero sobre la transmisión de un vídeo entre un servidor y un cliente.

Como se utilizan nodos estándar que utilizan protocolo de red estándar en Internet, se demuestra claramente un objetivo clave del trabajo, que es la capacidad de HaRTES de trabajar no sólo con tráfico FTT, sinó también con tráficos y nodos estándar.

Para realizar la transmisión de vídeo servidor-cliente se pensó utilizar VLC , un reproductor de vídeo que tiene funcionalidades adicionales que permiten tanto enviar como recibir streams a través de una red. Para ello será necesario encontrar la mejor manera de comprimir el vídeo para enviarlo y que el otro nodo recibiera los datos. De todos los códecs, que son la manera de comprimir el vídeo para luego realizar su transmisión, el mejor para comprimir el archivo de video es H264 que tiene un retardo muy corto y alta calidad de codificación-decodificación, manteniendo la calidad del vídeo al máximo posible.

Al final se decidió realizar una visualización de un vídeo de internet, ya que era más interesante observar cómo la red estaba conectada con el exterior además de realizar transmisiones más prescindibles de manera muy parecida al formato cliente-servidor. Por lo que se conecta el *switch* a internet y después también se conecta un nodo al mismo *switch* para darle la conexión a internet necesaria para reproducir el vídeo.

# 6. Aplicaciones

En este apartado se explicará cómo ha sido el proceso de implementación de las tres aplicaciones en la red. Los nodos que se utilizan como esclavos son todos *Intel Atom* con procesador Intel(R) Atom(TM) CPU D525 @ 1.80GHz, 2 GB de RAM y 4 interfaces ethernet. El nodo que realiza el trabajo de *switch* y de maestro se trata de un ordenador *multicore* estándar con un procesador Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 8 GB de RAM y 10 interfaces de red. El nodo que funciona como simulador es similar al *switch*, pero sin las interfaces de red. Todos los nodos están físicamente conectados al *switch*, de manera que están todos interconectados a nivel de capa física.

El lenguaje de programación utilizado es C para todas las aplicaciones. Como se han desarrollado las aplicaciones en Linux, se ha utilizado el compilador GCC y compilado los archivos con el comando *make* desde la consola para poder ejecutar los programas. También se ha utilizado programación concurrente con hilos para realizar el muestreo de la tarea de control así como lecturas y escrituras en los GPIO de los nodos para comunicarse con el exterior en la aplicación domótica.

Para realizar las comunicaciones con el nodo que contiene la planta simulada, se utilizará un protocolo a nivel de transporte llamado UDP, que se basa en el intercambio de datagramas enteros en vez de bytes sin necesitar una conexión previa, ya que la cabecera especifica el direccionamiento. No tiene confirmación ni control de flujo. UDP utiliza puertos para permitir la comunicación entre aplicaciones, trabaja de manera asíncrona y provoca poco *overhead* debido a su sencillez y sus cabeceras simples.

## 6.1 Aplicación domótica

La idea de la aplicación domótica era la de encender una luz gracias a un sensor de presencia, que generaría un valor que podría ser transmitido a través de un *stream* hasta otro nodo que sería el encargado de activar la bobina de un relé y

cerrar un circuito con una bombilla conectada a la red eléctrica. Un esquema de la idea primera sería el siguiente:

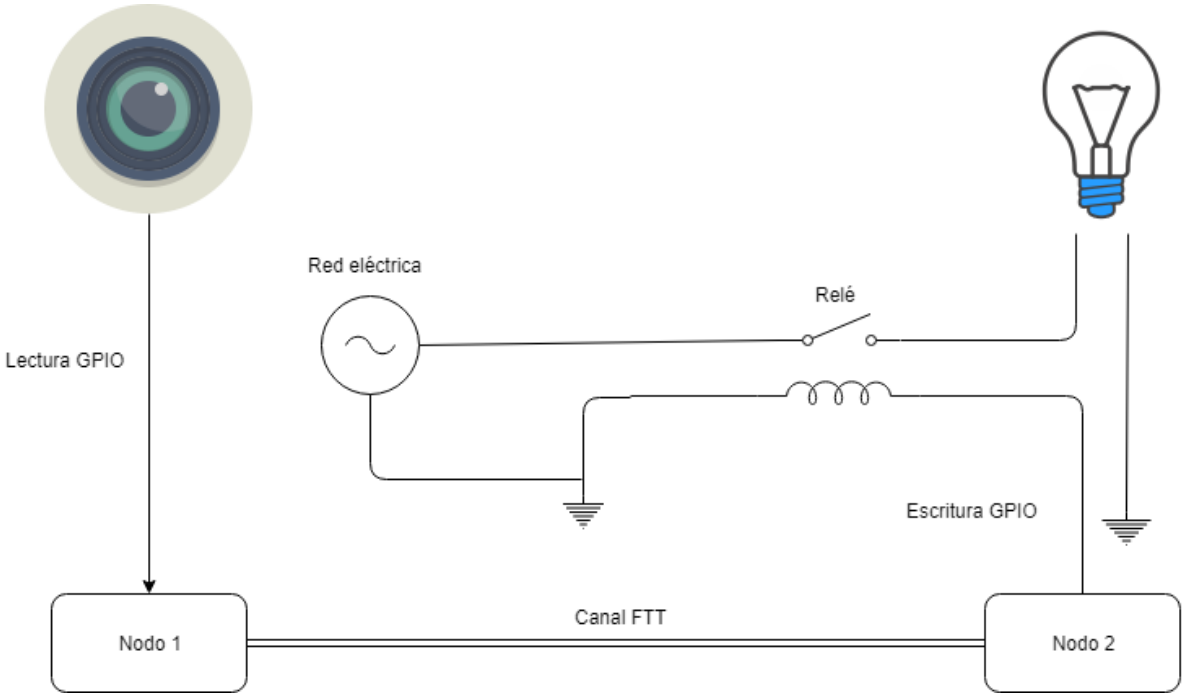


*Figura 8: Esquema del circuito de la aplicación domótica*

Al final, como se ha comentado antes, se ha decidido simplificar el diseño y realizar el montaje a una escala más reducida, con un pulsador como sensor y un LED blanco como sustituto del relé que enciende una bombilla. Esto se ha implementado simplificadamente ya que debido a los retrasos en *Ca ses Llúcies* no se ha tenido acceso, y por lo tanto ésta implementación, aunque simplificada, implementa toda la lógica necesaria para que cuando se despliegue sólo sea necesario cambiar el hardware y un ajuste menor en el código.

La frecuencia con la que se ejecuta el programa puede ser controlada si se obtiene el número de TM que se está enviando, y actuar en función de éste. Por ejemplo, se puede implementar una cláusula condicional que sólo permita al programa enviar los datos en números de TM. De esta manera se obtiene la organización de los tres nodos necesaria para su correcto funcionamiento. Pese a ello, se ha configurado la ejecución para cada *elementary cycle* ya que se tiene un ancho de banda suficiente, pero si fuera más restrictivo podría implementarse de la manera indicada.

Para realizar el montaje se han utilizado dos *protoboards* diferentes, cada una para realizar el circuito que se requiere para que cada componente realice su función.

Los esquemas de los circuitos se muestran a continuación:

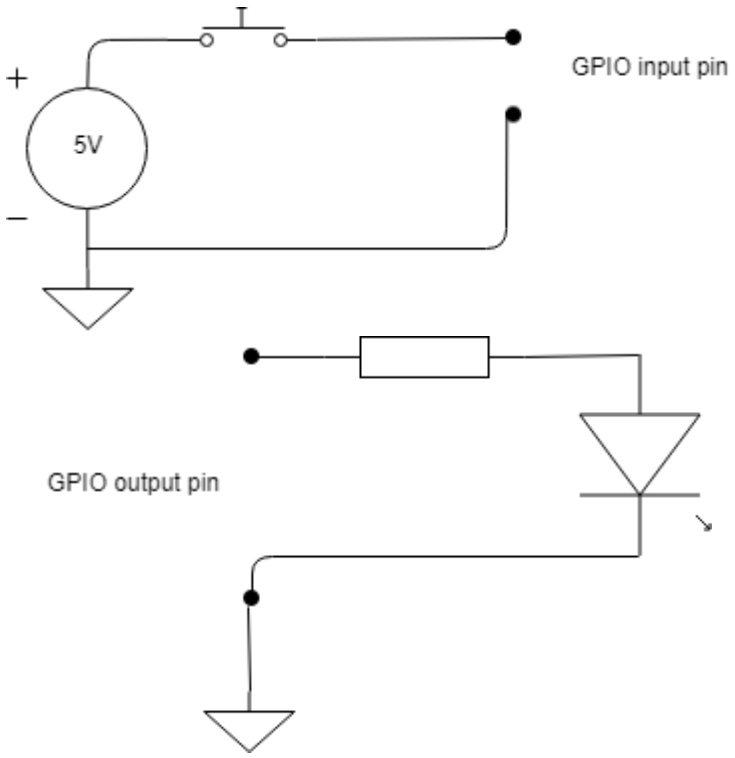


*Figura 9: Circuitos de entrada y salida de los GPIO*

Los pines de entrada y salida de la GPIO de los *Intel Atom* funcionan a 3.3 voltios, lo que nos ayuda ya que no serán necesarios conversores de tensión para cambiar la alimentación ni la lectura del valor otorgado. Además, para alimentar el primer circuito, se puede utilizar uno de los pines del ordenador que otorga constantemente cinco voltios y otro que siempre tiene el valor de tierra. Entonces el primer circuito deja pasar el valor cuando se pulsa y el segundo se enciende cuando se realiza una escritura del GPIO.

Para realizar el transporte se ha utilizado un *stream* FTT que comunica el nodo que lee el valor de la GPIO hasta el segundo nodo, que actúa escribiendo el valor sobre la GPIO para realizar una acción. Cómo era únicamente un valor, no se ha tenido que tratar los datos para el transporte y simplemente se han transmitido a través del canal.

Para finalizar se ha implementado una funcionalidad de *timeout* o cuenta regresiva que apaga la luz cuando un cierto tiempo se ha cumplido sin que haya una lectura positiva de la GPIO de entrada. Esta funcionalidad intenta simular un apagado automático de las luces cuando no detecta una presencia en la sala durante un cierto tiempo. De esta manera, se ahorra energía cuando no se está utilizando la sala.

Un ejemplo en pseudocódigo de las dos aplicaciones se muestra a continuación:

```
app1 (){
     inicialización;
     inicialización GPIO;
     crear stream1;
     attach stream1 como publisher;
     while(true){
          leer_GPIO;
          transmitir;
     }
}

app2 (){
     inicialización;
     inicialización GPIO;
     attach stream1 como subscriber;
     while(true){
          recibir;
          if(valor recibido == 1){
               Escribir_GPIO_1;
               t_out=0;
          }
          else{
               tout++;
               if(t_out == máximo){
                    Escribir_GPIO_0;
                    t_out=0;
               }
          }
     }
}
```

Explicado brevemente, la primera aplicación inicializa las variables para poder leer del GPIO. A continuación, crea el *stream* necesario para la transmisión y se conecta como emisor. Una vez inicializada, la aplicación lee constantemente el valor del GPIO y lo transmite.

La segunda aplicación inicializa su GPIO y se conecta al *stream* como *subscriber*, ya que debe obtener información de él. Después va leyendo constantemente el valor recibido por FTT y si éste es un 1, escribe en la GPIO un uno encendiendo la salida. También se ha implementado un *timeout* que apaga la salida si durante un tiempo no han llegado unos por el canal FTT.

La inicialización de los GPIO consta de escrituras en un fichero para establecer las características de la interacción con los pines. Se escribe en el fichero de que GPIO se trata y si el GPIO funcionará como entrada o salida. Después se puede leer el valor escrito en el fichero si éste fuera de entrada, o escribir uno si éste fuera de salida, afectando así al valor de voltaje que saldrá por dicho pin.[4]

## 6.2 Aplicación de control

Para probar el tráfico de datos de una red de control, se necesita una planta que requiera una regulación o corrección de un cierto error. Para ello se podría haber utilizado una planta real, con sensores y actuadores, pero eso es voluminoso y costoso, así que para probar el funcionamiento de la aplicación se utilizará la técnica del *Hardware in the Loop*, que simula una planta gracias a modelos matemáticos. Esta planta está programada en Simulink y consiste de un péndulo invertido, que tiene un ruido que lo desequilibra automáticamente, y además una entrada de un joystick que permite modificar el ángulo de inclinación de la barra para forzar una actuación más agresiva de los PID, su controlador.

Esta aplicación tiene tres partes fundamentales, ya que sin una de ellas el control sería imposible: La fase de muestreo, dónde obtiene los valores de error de la planta; la fase de control, donde obtiene el valor de actuación en función de los errores gracias al PID; y la fase de actuación, que con la actuación obtenida en la fase anterior se corrige el error de la planta. Cómo se utiliza *Hardware in the Loop* las simulaciones deben suplir las acciones de muestreo y de actuación, ya que la planta no es real.

Serán necesarios tres nodos para simular las tres partes imprescindibles del control del sistema. La sincronización es muy importante en este apartado, ya que se debe saber en cada momento cuándo envía o recibe cada uno de los nodos, además de cuándo ejecuta su parte del trabajo, ya que debe ser siempre lo más reciente posible para evitar datos obsoletos pero nunca ejecutarse después del envío. Para ello, una posible solución es usar el número de *elementary cycle* enviado por el maestro.

Para ello primero se utilizan como offset para ordenar el acoplamiento y así ordenar los nodos para que transmitan cada tres *elementary cycles* gracias al período de la aplicación. Después también debe estar sincronizada la ejecución del código de cada una de las tareas. Para ello simplemente se bloquean realizando una espera activa hasta que llegara al EC necesario para realizar su función. Cabe destacar un problema en la implementación escogida, y es que una espera activa no es el método más idóneo, pero las librerías no contemplaban la posibilidad de utilizar semáforos para realizar las esperas, así que se realizó con esperas activas.

La frecuencia de ejecución en este caso es 3, es decir, cada nodo ejecuta su código cada tres *elementary cycles*, ya que es necesario realizar esta separación para tener los tres nodos en forma de ciclo ordenados según su función. En la siguiente figura se muestra en azul las transmisiones UDP entre la planta y los nodos, y en rojo las transmisiones FTT entre los nodos utilizando el *switch* HaRTES.

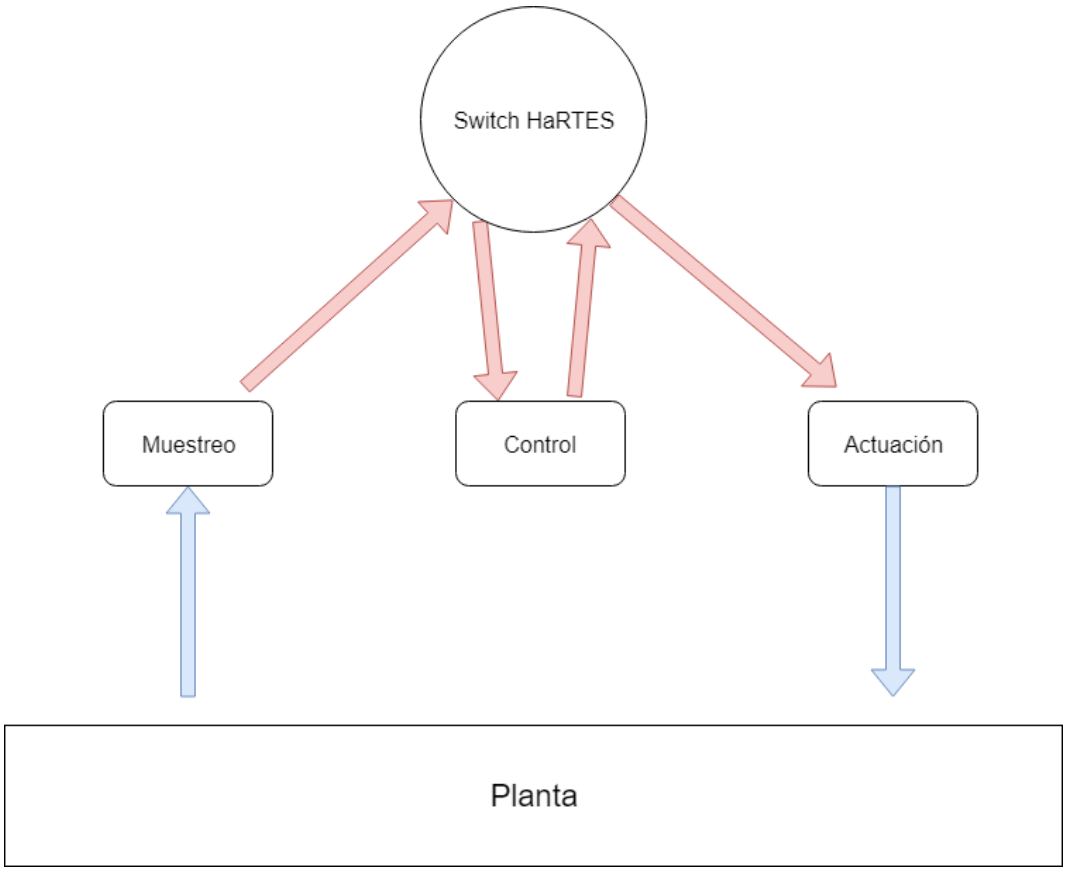


*Figura 10: Esquema de comunicaciones de la aplicación de control*

El primero de los nodos simula el sensor. No se puede realizar ningún control si no se observa la planta y se mide el error, si el muestreo tiene más precisión, los datos serán más reales y se podrá realizar una actuación acorde al error real. Este sensor utiliza un *thread* encargado de realizar el muestreo, obteniendo los valores enviados usando el protocolo UDP desde Simulink. El nodo coge el último valor muestreado por el *thread* y lo envía por el *stream* hacia el nodo de control. Para enviarlos coge ambos valores obtenidos del muestreo, ángulo y posición, los concatena en un mismo buffer de transmisión, y los envía a través del primer canal.

Se utiliza UDP sobre otros protocolos, como podría ser TCP, debido a su bajo *overhead* ya que tiene un preámbulo mucho más pequeño que TCP y además la capacidad de enviar los mensajes en cuestión de microsegundos, aunque éste tiempo sigue dependiendo del hardware y el sistema operativo.[5]

El primer nodo también crea ambos *streams* necesarios para el funcionamiento de la aplicación, aunque él sea simplemente el *publisher* de uno de ellos. Como es la primera tarea que se activa, así es más rápido para las demás acoplarse a los canales virtuales para transmitir

El *thread* es imprescindible para que el muestreo se realice de forma adecuada, ya que para que los nodos se sincronicen, se deben realizar esperas, ya sean activas o en forma de semáforo. Mientras una aplicación realiza una espera, no puede ejecutar parte de su código, por lo que no podría recibir los mensajes de Simulink y no sería capaz de realizar un muestreo correctamente, ya que utilizaría valores obsoletos, previos a la espera.

Así que se utiliza el *thread* o hilo para ir leyendo periódicamente los valores que se envía desde Simulink usando el protocolo UDP y que los vaya guardando en dos variables globales que la aplicación de muestreo utilizará para enviar los datos a la aplicación de control.

Para leer los datos del nodo que ejecuta la planta se utiliza una comunicación UDP. Para inicializar esta comunicación se necesita tener la dirección IP a la que se va a comunicar, que en este caso es la del nodo que ejecuta la planta, y además definir dos puertos que deben coincidir con los puertos de entrada y de salida de la aplicación en Simulink. Entonces se inicializa un socket con el puerto de recepción para recibir, y después obtiene la dirección del socket de envío hacia la planta con la dirección IP y el puerto. Para recibir lee los datos proporcionados del socket de recepción (punto de vista de la aplicación), y para transmitir envía a la dirección de envío obtenida con la dirección IP y el puerto.

A continuación, se muestra un esquema en pseudocódigo del funcionamiento de la tarea de muestreo:

```
double ángulo;
double posición;


app1 (){
     inicialización;
     crear streams;
     thread_create;
     espera hasta EC indicado; //Para la sincronización
                                                entre nodos
     attach stream1 como publisher;
     while(true){
          unificar ángulo y posición;
          transmitir;
```

```
                bloquearse dos EC; // Para seguir el orden
                                                    cíclico
        }
}

thread(){
        while(true){
                obtener valores puerto UDP;
                guardar en ángulo y posición;
        }
}
```

El segundo nodo ejecuta los PID, que realizan el control a partir de los valores muestreados por el primer nodo. Al recibir el mensaje del primer nodo, debe separarlo en dos variables, ya que en el mismo mensaje se enviaban ambos valores de muestreo. Una vez separados, se llama dos veces a la función pid_update con una variable de tipo *struct* que le indica sobre qué error actúa, para obtener el valor de actuación deseado. Se llama una vez dando el valor del ángulo y otra vez el valor de la posición. De esta manera ambos PID intenta corregir tanto el error de ángulo, que es más importante, como el error de posición.

Una vez se obtienen as dos actuaciones obtenidas de cada uno de los errores, se deben sumar sus actuaciones en una sola, ya que la planta sólo es capaz de mover la base del péndulo para corregir el error porque sólo tiene un actuador.

Una vez obtenido el valor de actuación de la función, es necesario transmitir ese valor de nuevo al Simulink. Para ello se utiliza un tercer nodo, conectado por UDP a Simulink, que transmitirá el valor obtenido para que actúe corrigiendo el error. Para que el tercer nodo reciba el valor de actuación se necesita también un segundo *stream*, ya que un *stream* sólo puede tener un *publisher*, o nodo emisor, y los dos anteriores ya sirven para otros propósitos. Por lo que se envía el valor de actuación obtenido de los PID a través del segundo *stream*, desde el nodo uno hasta el nodo dos.

Un esquema en pseudocódigo se muestra a continuación:

```
app2 (){
        inicialización;
        espera hasta EC indicado; //Para la sincronización
                                                entre nodos
        attach stream1 como subscriber;
        attach stream2 como publisher;
```

```
     while(true){
          recibir_stream1;
          separar datos;
          actuación                                        =
pid_update(ángulo)+pid_update(posición);
          transmitir_stream2;
             bloquearse dos EC; // Para seguir el orden
                                                  cíclico
     }
}

double pid_update(tipo, error){
     cálculo error diferencial;
     cálculo error integral;
     cálculo error proporcional;
     guardar error actual como error anterior;
     return e_dif + e_prop + e_int;
}
```

El tercer nodo simula la actuación, aunque él mismo no realice la actuación per se. Para ello, recibe a través del *stream* dos el valor de actuación que ha calculado previamente la función pid_update, y transmite a través de UDP el valor de vuelta a la planta simulada en Simulink. Este nodo no debe separar variables, ya que la actuación es simplemente un valor.

A continuación se muestra su funcionamiento en pseudocódigo:

```
app1 (){
     inicialización;
     espera hasta EC indicado; //Para la sincronización
                                          entre nodos
     attach stream2 como subscriber;
     while(true){
          recibir_stream2
          transmitir_UDP;
          bloquearse dos EC;
     }
}
```

## 6.3 Aplicación multimedia

La tercera aplicación es la aplicación multimedia, y para implementarla se puede utilizar como punto de partida la programación del *switch* HaRTES para conectar la red a internet y de esta manera transmitir datos de un vídeo obtenido de una página de video bajo demanda. Para conectar el *switch* a internet basta con conectar un cable conectado a Internet a uno de los puertos del *switch*, ya que éste es capaz de gestionar el tráfico estándar.

Lo más importante del tráfico multimedia es que aunque sea muy exigente en cuanto a niveles de ancho de banda, no requiere un tráfico estricto ni requisitos de tiempo real por lo que si el ancho de banda se ve reducido, el vídeo será el que pierda cantidad de datos.

Para atenuar éste efecto se utilizan métodos de compresión de video que reducen mediante algoritmos la cantidad de datos enviados, lo que alivia la carga de ancho de banda y permite al vídeo operar normalmente con un mayor margen de posibles sobrecargas.

La implementación de esta aplicación es muy sencilla, ya que únicamente se debe conectar un cable Ethernet del *router* hasta el *switch*, y después conectar el *switch* al nodo que funciona como cliente de vídeo. De esta manera se tiene el nodo conectado a Internet a través del *switch* HaRTES y simplemente se debe mostrar en una pantalla un vídeo.

Una vez conectado el nodo a una pantalla para poder visualizar el vídeo y se reproduce desde el nodo *Intel Atom*. Así se puede observar como se ha integrado tráfico que no es FTT a la red y el *switch* se encarga de hacer el direccionamiento gracias a las conexiones dedicadas entre cada nodo y el maestro.

# 7. Pruebas

En este apartado se explicarán cuáles han sido las pruebas que se han ido realizando en cada uno de los apartados anteriores para verificar que su funcionamiento es correcto y poder dar por completados los objetivos que se han propuesto al principio del documento.

El proceso de validación no ha sido inmediato, aplicando una prueba final para ver que el programa funciona y dar por concluido el apartado. La validación ha sido un proceso por el cual se ha ido comprobando que cada uno de los objetivos más pequeños, como una sincronización o un envío, funcionaban. Por eso en este apartado se comentarán cada una de las pruebas realizadas para verificar los objetivos claves y los resultados de éstas.

## 7.1 Pruebas de la aplicación domótica

La aplicación domótica envía datos en cada *elementary cycle* para ver si el valor de entrada del GPIO ha cambiado y por lo tanto alguien ha pulsado el pulsador. Para ir verificando su funcionamiento, mayoritariamente se hicieron tres pruebas:

- La primera fue enviar un valor de una placa a otra con un mensaje síncrono. Para ello se creó el *stream* pertinente, los nodos se acoplaron y se envió el mensaje. Una vez recibido se imprimió el valor en la otra placa, y se observó que era correcto, por lo que el envío síncrono funcionaba.

- La segunda fue intentar lo mismo, pero esta vez con un mensaje asíncrono, que sería la manera correcta de realizarlo en un sistema domótico ya que es un evento espontáneo y si se hace de manera síncrona requiere un flujo constante de mensajes indicando el valor constantemente. Nos daba errores de creación del *stream* con características asíncronas debido a problemas con las librerías HaRTES, por lo que se decidió no implementar la aplicación con las características asíncronas y se realizó la transmisión de manera síncrona.

- La última prueba se realizó tras haber realizado el montaje de las *protoboards* con los componentes necesarios. Se comprobó el funcionamiento del envío utilizando los valores leídos desde el GPIO y escribiendo el valor en el GPIO de salida. Se intentó hacer funcionar el programa varias veces, realizando varias pulsaciones con un tiempo entre ellas. Como el valor de las variables y el funcionamiento del circuito era correcto, se dio esta parte por concluida.

## 7.1 Pruebas de la aplicación de control

Esta segunda aplicación tiene unas dificultades un poco superiores a la primera. Tiene diferentes *streams*, con diferentes datos y entre diferentes parejas de nodos, además de requerir de una sincronización extra para su correcto funcionamiento. También requiere tratar más las variables a enviar ya que estas son *doubles* y además se deben enviar en el mismo mensaje por lo que hay que concatenarlas y separarlas una vez lleguen al destino.

La pruebas realizadas fueron las siguientes:

- Comprobar que la sincronización de los tres nodos es correcta. Para ello se utiliza, como se ha mencionado anteriormente, el número de *trigger message* actual para que el nodo sepa en qué momento se encuentra, y si debe actuar o no. De esta manera ya se ha conseguido esta planificación cíclica necesaria. Para ver que funcionaban correctamente, se mostraban mensajes por pantalla en cada uno de los nodos, y cómo se enviaban en el EC correcto se pudo confirmar que funcionaba

- La segunda prueba constaba de verificar los envíos entre las diferentes placas. Para ello se probó a crear una variable en coma flotante con un valor en el primer nodo y transmitirla al segundo nodo mediante FTT. Después este segundo nodo obtenía el valor del mensaje y lo enviaba hacia el tercer nodo, mostrando los valores por pantalla para monitorizar que los envíos y las recepciones son correctos.

- La tercera prueba fue obtener los valores y enviar valores de los puertos UDP conectados con Simulink. Para ello se preparó el Simulink habilitando los puertos pertinentes para recibir y enviar, y a continuación se muestran por pantalla mensajes para ver si se realizan correctamente las transmisiones.

- La última prueba fue probar toda la aplicación junta, obteniendo los valores de los puertos UDP y enviando datos por los *streams* FTT. También se añadió el control de la planta en forma de PID y se probó el programa. Cómo se observaba que el péndulo se mantenía equilibrado, incluso con el uso del joystick, se puede confirmar que la segunda aplicación funciona.

## 7.3 Pruebas de la aplicación multimedia

Las pruebas de la aplicación multimedia fueron primero para el modelo cliente-servidor, que al final fue descartado para implementar la reproducción de un vídeo mediante una conexión a Internet.

Las pruebas cliente-servidor fueron pruebas de calidad de vídeo. Al emitir un vídeo a través de VLC, te permite configurar la transmisión y elegir cuál será el algoritmo de compresión utilizado para transmitir. Después de las pruebas se determinó que el mejor *códec* era H264.

Las pruebas que se realizan para comprobar el correcto funcionamiento de la aplicación conectada a internet fueron simplemente monitorizar la salida del vídeo a través de la pantalla y comprobar que la calidad era correcta, además de la fluidez y la velocidad de carga de este. Una prueba que podría haberse hecho sería probar de sobrecargar más la red con tráfico de tiempo real que no debe retrasarse, y ver cómo a medida que el tráfico aumenta la calidad del vídeo se ve comprometida y se va reduciendo hasta que no puede reproducirse. Así se demostrará que el *switch* permite el tráfico de comunicaciones menos prioritarias siempre que no haya comunicaciones más importantes que deban llevarse a cabo.

## 7.4 Pruebas finales.

La última prueba a realizar una vez se han verificado todas las aplicaciones por separado es unificarlas y probarlas todas en el mismo código. Esto no llevó casi problemas ya que únicamente había que cambiar el nombre de los *streams* que se llamasen igual o el nombre de alguna variable cuya denominación fuera la misma. Una vez realizados los cambios, se intentó ejecutar el programa y se vieron las tres aplicaciones funcionando correctamente. El control funcionaba, el interruptor también y el vídeo de internet se visualizaba correctamente. Con esto se pueden dar por concluidas las pruebas y afirmar que se han cumplido los objetivos propuestos.

# 8 Conclusiones

Este documento ha expuesto todos los procesos llevados a cabo para implementar una red FTT que pudiera utilizar diferentes tipos de tráfico de datos con diferentes necesidades.

La parte más interesante de este trabajo es el potencial que tiene para adaptar todos los tipos de datos que fluyen hoy en día en infinidad de objetos cotidianos que hacen la vida humana más sencilla. FTT potencia Ethernet, que tiene ya unas características muy buenas de ancho de banda y simpleza en lo relativo al cableado, otorgándole ventajas de las que antes carecía, como la capacidad de trabajar en tiempo real, gracias al esquema organizativo de los mensajes determinista de FTT. Además gracias a utilizar HaRTES e integrar el nodo maestro en un *switch* conectado en topología de estrella con los esclavos se puede integrar tráfico que no sea FTT en la red ampliando su utilidad y su rango de tráficos. Esto permite implementar este tipo de redes en múltiples sitios, como podría ser una casa, donde se combinaría el tráfico domótico de las diferentes funcionalidades automatizadas con el tráfico multimedia de los habitantes de la casa, realizando transmisiones entre diferentes dispositivos como hacer un display de un vídeo del móvil en la televisión.

Además, podría implementarse en entornos más estrictos donde los requerimientos de eficacia, velocidad y determinismo fueran mayores. Gracias a su gran versatilidad, el rango de aplicaciones de esta red es enorme.

Otra ventaja a tener en cuenta es que, hoy en día, las redes Ethernet están cada vez más extendidas por el mundo realizando conexiones de red local, y cada vez más se implementa en entornos industriales. Adaptar esas conexiones a FTT es una tarea sencilla, ya que simplemente se tendría que retocar el esquema de conexión, en caso de que no fuera en estrella, y después programar el *switch* y los nodos con el paradigma para que utilizaran las especificaciones FTT.

El esquema en particular sería muy fácil de extrapolar a una casa, e incluso de expandir para que tuviera diferentes funcionalidades extra. Por ejemplo, en este trabajo se ha utilizado un péndulo invertido para la aplicación de control, que no es algo que se utilice en una casa normalmente. Pero si en vez de eso se utiliza la aplicación para realizar un

control de la temperatura de la casa con una bomba frío-calor y sensores repartidos por las habitaciones, se tiene una aplicación muy útil para una vivienda. Se podrían automatizar la mayoría de las luces de la casa con sensores de presencia en combinación con sensores lumínicos o relojes, o realizar un control de riego en función de la humedad, el sol o de la lluvia.

Si se quisiera adaptar la red a *Ca ses Llúcies*, simplemente se debería elegir un microcontrolador, ya que los Intel Atom, aunque pequeños, son demasiado grandes; e instalarlos alrededor de la casa donde una funcionalidad fuera requerida. Una vez realizado esto se debería programar el microcontrolador para realizar la tarea, e interconectarlos todos en un mismo *switch* gobernado por FTT para que formaran una red.

Personalmente, realizar este trabajo ha sido muy interesante a la par que satisfactorio. La parte más interesante a mi parecer ha sido poder aprender cómo funcionaba FTT, ya que es una manera de mejorar Ethernet y solucionar sus flaquezas. Otra cosa que también me ha motivado a lo largo del trabajo ha sido mejorar mis habilidades de programación, tratar de utilizar correctamente primitivas de infinidad de librerías y realizar comunicaciones entre aparatos que iban más allá de un microcontrolador. Durante la implementación ha habido momentos de frustración, ya que no siempre encontraba la solución rápidamente, pero los tutores estaban para echarme una mano y podía seguir adelante.

Me ha gustado tener la oportunidad de realizar algo más complicado y realista que fuera capaz de ser implementado, aunque fuera un primer esquema y no la instalación completa de la red en la *casa Ca ses Llúcies.*

# Bibliografia

[1] Julián Proenza Arenas, Alberto Ballesteros, A description of FTT-SE protocol. [Online]. Available: http://srv.uib.es/a-description-of-the-ftt-se-protocol/

[2] Rui Gabriel Viegas dos Santos, Enhanced ethernet switching technology for adaptive hard real-time applications. [Online]. Available: http://ria.ua.pt/handle/10773/7142

[3] Blaise Barney, Lawrence Livermore National Laboratory. POSIX threads programming.

[4] Raspberry Pi And The IoT In C - SYSFS The Linux Way To GPIO. [Online]. Available: https://www.iot-programmer.com/index.php/books/22-raspberry-pi-and-the-iot-in-c/chapters-raspberry-pi-and-the-iot-in-c/57-raspberry-pi-and-the-iot-in-c-sysfs-the-linux-way-to-gpio?showall=&start=1

[5] Michael Pan, Real time communications over UDP protocol [Online]. Available: https://www.codeproject.com/Articles/275715/Real-time-communications-over-UDP-protocol-UDP-RT