UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Trabajo Final de Grado en Ingeniería Informática

TouCAN: Mejoras y Ampliación de Funcionalidades

Aridany Santana Sánchez

Tutores
Antonio Carlos Domínguez Brito
Jorge Cabrera Gámez

30 de noviembre de 2014

Agradecimientos

Me gustaría comenzar este documento mostrando mi agradecimiento a todas aquellas personas que de forma directa o indirecta han colaborado en la realización este proyecto.

En primer lugar agradecer a mis tutores Jorge Cabrera Gámez y Antonio Carlos Domínguez Brito el apoyo y asesoramiento recibido durante el desarrollo de este Trabajo Final de Grado (TFG).

Mis más sincero agradecimiento a mi familia por su ánimo, paciencia y motivación durante la realización de este trabajo.

Por último, me gustaría felicitar a John Wu Wu[24] y Jose Antonio Lardo Domíguez [22] por el trabajo realizado en sus respectivos TFG que dieron origen a la primera versión de la librería TouCan v1.0 y que ha sido la base de este TFG.

Resumen

TouCAN es una librería creada en su primera versión (v1) como Trabajos de Fin de Grado en Ingeniería Informática por John Wu Wu[24] y Jose Lareo Domínguez[22] bajo la tutorización de los profesores Antonio C. Domínguez Brito y Jorge Cabrera Gámez. Define un protocolo de comunicación para la interconexión de una red de microcontroladores basados en la plataforma de prototipado electrónico Arduino y trabaja sobre el protocolo de comunicación CAN Bus (Controller Area Network)[4], ampliamente utilizado por la industria desde la década de los 80.

El objetivo principal de este Trabajo Final de Grado en Ingeniería Informática consiste en proporcionar robustez a la librería incorporando mejoras y nuevas funcionalidades. Entre las principales mejoras destacan el control frente a fallos de comunicación, reinicio o reset de los microcontroladores, así como la caída de los mismos. Otra característica incluida en esta revisión consiste en la asignación dinámica de identificadores de dispositivos que conforman un sistema empotrado distribuido, permitiendo la posibilidad de "conexión en caliente" de nuevos nodos microcontroladores a la red de forma dinámica. A estos cambios, también se han añadido mejoras en la interfaz de la API que simplifica el uso y aprendizaje de la misma. Así como una nueva herramienta denominada TouCANSniffer que permite capturar y analizar todo el tráfico generado en la red.

En nuestra opinión, las nuevas características y funcionalidades añadidas en TouCAN v2.0 proporcionan el potencial necesario para ser considerada seriamente como base de cualquier nuevo proyecto que integre una red distribuida de microcontroladores.

Abstract

TouCAN is library created in its first version as Final Degree Projects in Computer Science by John Wu Wu[24] and Jose Lareo Domínguez[22] under the supervision of professors Antonio C. Domínguez Brito and Jorge Cabrera Gámez. It defines a communication protocol for interconnecting a network of microcontrollers based on the electronic prototyping platform Arduino. It operates using the CAN (Controller Area Network)[4] bus communication protocol that has been widely used by the industry since the 80s.

TouCAN stands out as a light, friendly and robust library. The main goal of this Final Degree Project in Computer Science is to provide robustness to the library including new functionality and improvements. Some of the main included new features are: control against communication failures, restarting and reset of microcontrollers, and detection devices which have stopped operation. Also another included feature is the dynamic assignment of identifiers for devices that allow us to hot plug new nodes dinamically. Moreover, we have also included further improvements which simplify the use and learning. In addition there is also new tool called TouCANSniffer that allow us capture and analyze all packets generated in the network.

All these new improvements included in TouCAN give the library the robustness necessary to be used in any project which integrates a microcontroller network.

Índice General

Capítulo 1
Introducción

1.1. Estado actual

TouCAN es una librería Open Source[17] que define un protocolo de comunicación para la interconexión de una red de microcontroladores basados en la plataforma Arduino[18] y en la tecnología CAN[4] (Controller Area Network). Nació en 2012 motivado por la inexistencia de una librería de código abierto 100 % funcional y sencilla de utilizar. TouCAN está dividida en dos partes bien diferenciadas, pero estrechamente relacionadas denominadas Microcontrolador y Supervisor. El primero proporciona un conjunto de clases que facilitan la comunicación entre los distintos dispositivos de la red. Mientras, el supervisor, permite establecer una comunicación entre un microcontrolador y un PC basado en un sistema Unix como Mac OSX o Linux.

1.2. Objetivos

El trabajo que se recoge en este documento está basado en la librería TouCAN que facilita la integración de redes de microcontroladores Arduino mediante el protocolo de conexión CAN Bus[4]. Tiene como objetivo incorporar nuevas funcionalidades que proporcionen la robustez necesaria para ser integrada en cualquier proyecto real de microcontroladores.

A continuación se detallan más específicamente los objetivos de este Trabajo de Fin de Grado en Ingeniería Informática:

- **Asignacion dinámica ID:**Mejora del protocolo de comunicaciones TouCAN con objeto de permitir la asignación y re-asignación dinámica de identificadores a los nodos microcontroladores que conformen un sistema dado.
- **Control frente a fallos de comunicación, reinicio o reset:** Mejora del protocolo de comunicaciones TouCAN de manera que se detecten situaciones de reinicio de nodos microcontroladores en el sistema, así como fallos de comunicación, añadiendo mecanismos de detección y recuperación.
- **TouCANSniffer:** Diseño y desarrollo de una herramienta que permita capturar los paquetes generados en la red. Con la finalidad de monitorizar el tráfico y facilitar la detección de errores o situaciones irregulares.

Desde este punto de vista académico, este TFG se ha orientado al aprovechamiento de los conocimientos adquiridos durante la titulación. De entre ellos se destacan las siguientes materias

- Algoritmos, programación y estructuras de datos.
- Diseño de Sistemas Basados en Microcontroladores.
- Sistemas Empotrados y de Tiempo Real.
- Algoritmos y Programación Paralela.
- Sistemas Operativos.
- Base de Datos.

Capítulo 2

Análisis del Problema

El principal objetivo de este TFG ha consistido en incorporar mejoras y nuevas funcionalidades al framework TouCAN manteniendo intacto el rendimiento y las características obtenidas en la primera versión del protocolo. Teniendo presente las dos partes bien diferenciadas del framework, microcontrolador y supervisor. Con esta premisa, se realizó un estudio previo de las estructuras de datos y la interfaz de programación, API. En una segunda fase, se implementaron diferentes escenarios donde se verificó el funcionacimiento, rendimiento y características de la librería en un entorno real.

El framework TouCAN está compuesto por dos librerías denominadas: TouCAN Microcontrolador y TouCAN Supervisor. La primera ofrece una API que facilita la interconexión entre dispositivos Arduino en una red bus CAN. Por otro lado, TouCAN Supervisor, proporciona un conjunto de librerías que permite la comunicación entre un PC y nodo Maestro. Esta conexión habilita a un PC, basado en sistema Unix, enviar paquetes a cualquier nodo en la red así como la monitorización del sistema.

2.1. TouCAN Microcontrolador

TouCAN Microcontrolador es una API que facilita la interconexión de la plataforma Arduino en una red con topología CAN Bus. Está basada en la librería ARCAN[23] dotándola de un sencillo protocolo de comunicación. Entre sus principales caracteristicas destacan: la facilidad de uso y la optimización de los recursos del dispositivo.

2.1.1. Topología

En la primera versión de TouCAN v1.0 se definen tres tipos de nodos o dispositivos que pueden coexisitir en una red de microcontroladores:

- **Esclavo o nodo común:** Son los nodos pasivos en la red. No inician peticiones por iniciativa propia, sólo atienden las peticiones de los Masters. Actuan como nodo sensores y/o actuadores en una red de microcontroladores.
- **Maestro:** A diferencia de los nodos Esclavos, puede iniciar una comunicación con otros nodos en la red o conectarse a un Supervisor.
- **Supervisor:** A diferencia de los nodos anteriores, no se conecta directamente a la red bus CAN[4] sino a través de los nodos Maestros. Entre sus tareas cabe destacar la monitorización del bus, iniciar peticiones a través de algunos de los nodos maestros, establecer velocidades de transmisión, etc.

La siguiente imagen 2.1 representa un esquema básico de una red TouCAN compuesta por tres nodos Esclavos, dos Maestros y un Supervisor.

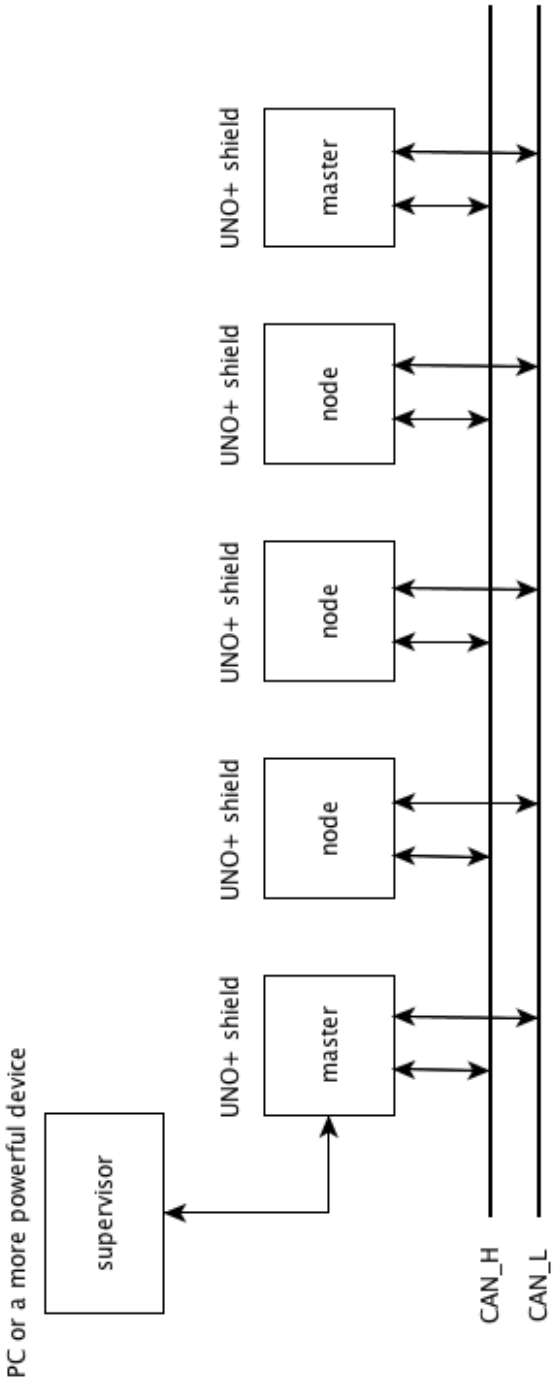


Figura 2.1: Esquema red TouCAN v1.0

2.1.2. TouCAN Trama

El protocolo bus CAN[4] define dos tipos de tramas que difieren en la longitud del campo identificador. La trama estándar definida en la especificación 2.0A tiene un ID de 11 bits mientras el identificador de la trama extendida asciende a 29 bits y está definida en la especificación 2.0B.

TouCAN hace uso de la trama extendida para definir su propio protocolo. Utiliza los 29 bits del campo identificador para establecer el origen, destino y el tipo de trama. La imagen 2.2 muestra las diferencias entre una trama CAN 2.0B[4] y una trama TouCAN. Por otro lado, la figura 2.3 contiene un ejemplo de estructura TouCAN.

- **ID0..ID10:** 11 bits reservados para establecer el origen del mensaje.
- **EID0..EID10:** 11 bits reservados para identificar el destinatario del mensaje.
- **EID11..EID15:** 5 bits utilizados para identificar el tipo de trama. La versión TouCAN v1.0 implementa tan solo 6 tipos de tramas de las 32 posibles.
- **EID16..EID17:** Bits inutilizados en la versión TouCAN v1.0.
- **DLC0..DLC03:** 4 Bits destinados a especificar la longitud del campo de datos.
- **Data field:** El protocolo establece un máximo de 8 bytes para el campo de datos.

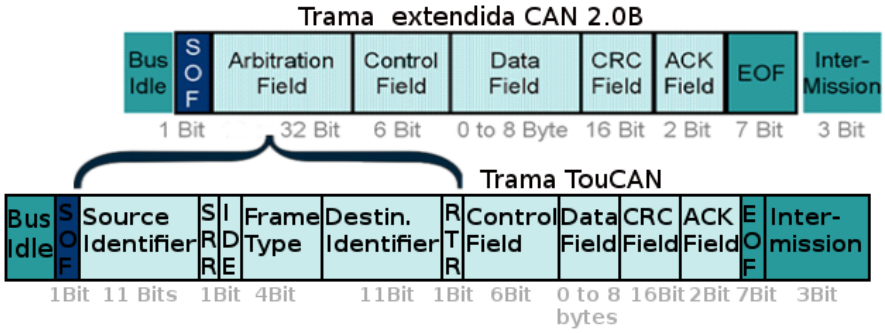


Figura 2.2: Comparativa entre una trama CAN 2.0B y TouCAN Trama

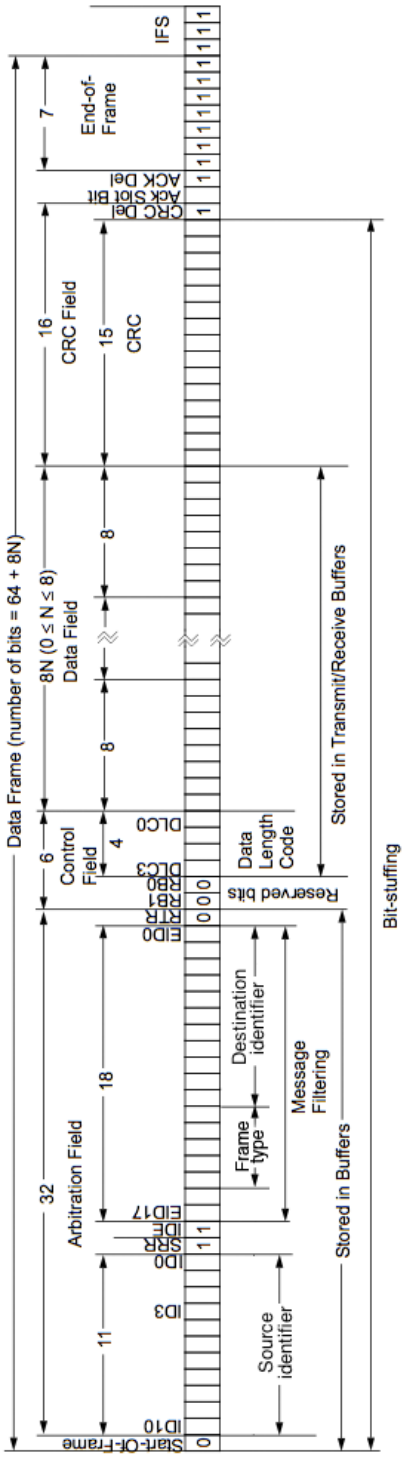


Figura 2.3: Trama TouCAN

2.1.3. TouCAN Tipos de Comunicaciones

El protocolo TouCAN define dos tipos de comunicaciones:

- **síncrona:** Este tipo de petición garantiza la recepción de la respuesta a una determinada solicitud. Útil para solicitar un dato puntual a un dispositivos de la red. La siguiente imagen 2.4 representa un ejemplo de comunicación síncrona.

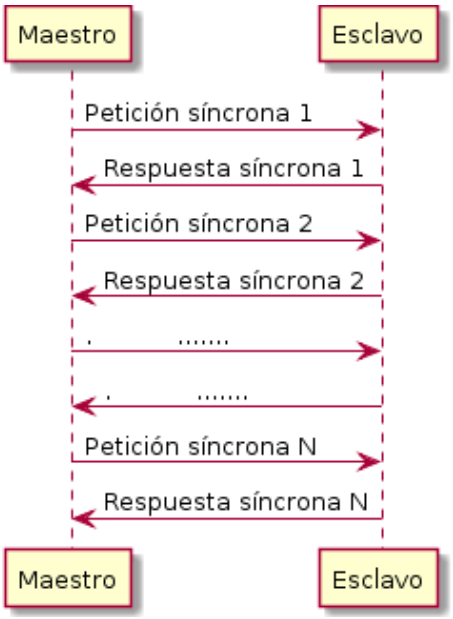


Figura 2.4: Ejemplo petición síncrona

- **asíncrona:** A diferencia de la petición síncrona, no se garantiza la recepción de todos los mensajes. La comunicación es establecida por un nodo master que solicita a otro nodo el envío periódico de un dato hasta que la petición sea cancelada. La siguiente imagen 2.5 muestra un ejemplo de petición asíncrona entre un nodo Maestro y otro Esclavo. El Maestro establece una comunicación asíncrona y espera la recepción de N paquetes antes de finalizar la comunicación con el Esclavo.

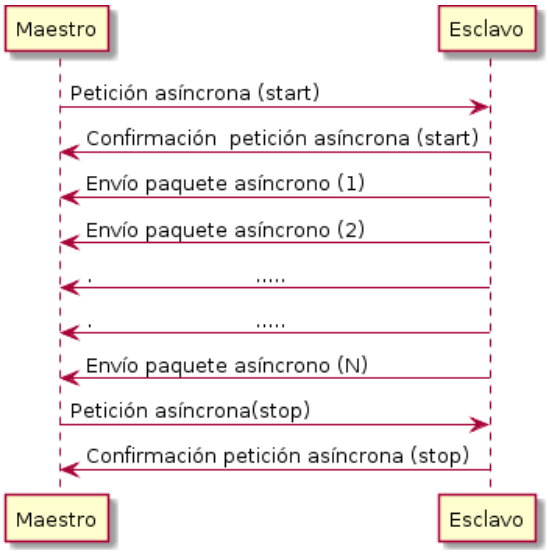


Figura 2.5: Ejemplo petición Asíncrona. Nodo Maestro solicita N paquetes a nodo Esclavo

2.1.4. Tramas síncronas

- **Request (petición):** Codificación en el protocolo,00000 (macro *REQU*). Esta trama es la responsable de realizar una petición síncrona. Únicamente puede ser emitida por un nodo Master o Supervisor. Garantiza la respuesta o en su defecto, informan de la pérdida del paquete para realizar un reintento. El usuario tiene total libertad para utilizar el campo de datos e implementar un protocolo sobre TouCAN.
- **Answer (respuesta):** Esta trama es utilizada por cualquier nodo de la red para responser a una petición *request*. Está codificada con el valor 00001 (macro *ANSW*).

La figura 2.6 muestra un ejemplo de comunicación síncrona entre un nodo Maestro y Esclavo donde se especifica el nombre y el código de cada una de las tramas que intervienen.

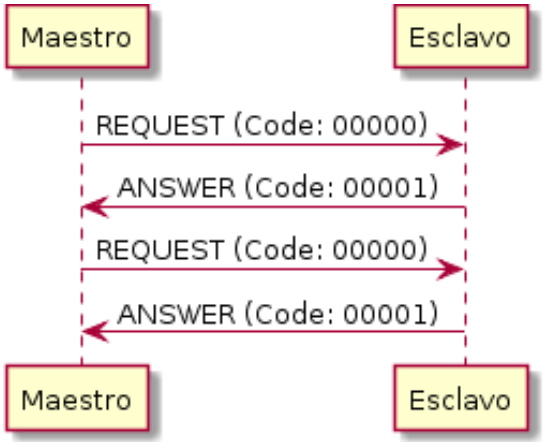


Figura 2.6: Ejemplo petición Síncrona

2.1.5. Tramas asíncronas

- **Start (comienzo):** Codificación en el protocolo: 00011 (macro *STRT*). Es enviada por un nodo Master o Supervisor para solicitar al destinatario que le envie un conjunto de datos de forma periódica. La frecuencia de envío se indica en milisegundos en el primer byte de datos del mensaje. El resto de datos son libres para que el usuario los utilice para implementar otro protocolo a un nivel superior.
- **Acknowledge (reconocimiento):** Codificación en el protocolo 01000 (macro *ACKN*). Esta trama es esperada por un maestro inmediatamente después de enviar un "start" o "stop".
- **Bulk (envío masivo):** Codificación en el protocolo 00101 (macro *BFRM*). Esta trama no garantiza la recepción del paquete y es utilizada como respuesta a una petición de tipo "start". (Respuesta asíncrona)
- **Stop (finalización):** Codificación en el protocolo: 00100 (macro *STOP*). Informa al destinatario el cese de envío de paquetes asíncronos, "bulk".
- **Reject:** Codificación en el protocolo, 00010 (macro *REJE*). Informa que no ha sido posible atender la petición, indicando el motivo por el cual ha sido rechazada. Un Maestro o Supervisor puede mantener comunicaciones síncronas y asíncronas con un mismo nodo de forma simultánea. Sin embargo, un nodo común o Esclavo no puede mantener comunicaciones simultáneas con má de un maestro o supervisor.

La siguiente imagen 2.7 muestra un ejemplo de comunicación asíncrona entre un nodo Maestro y Esclavo donde se especifica el nombre y el código de cada una de las tramas que intervienen.

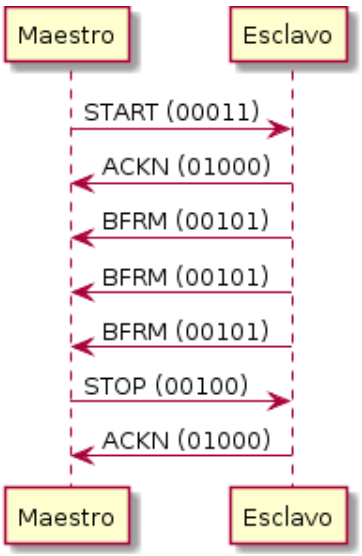


Figura 2.7: Ejemplo petición Asíncrona entre un nodo Maestro y otro Esclavo

2.1.6. TouCAN Estructuras de Datos

TouCAN v1.0 define dos tipos de estructuras de datos denominadas: **node** y **tCAN**. La primera representa un nodo en la red y es utilizada por los dispositivos Maestros. La segunda define la estructura de un mensaje TouCAN utilizada en el envio y recepción de mensajes por los distintos nodos que conforman un determinado sistema.

2.1.6.1. TouCAN - Estructura tCAN

tCAN es la estructura que representa un mensaje en la librería TouCAN. Actúa como buffer de recepción y transmisión del MCP2515[8] al enviar o recibir mensajes a través del bus CAN[4]. El siguiente fragmento de código representa una estructura tCAN.

```
typedef struct{
  unsigned int id_s;
        unsigned int id_d;
        int frametype;
        struct {
          byte rtr : 1;
          byte length : 4;
        }header;

        byte data[8];
}tCAN;
```

- **id_s:** Almacena el ID de origen del mensaje.
- **id_d:** Almacena el ID destinatario del mensaje.
- **frametype:** Identifica el tipo de trama a enviar.
- **struct header:** Almacena el tipo de trama a nivel de CAN y la longitud del campo de datos.
- **data[8]:** Contiene los datos del mensaje.

2.1.6.2. TouCAN - Estructura node

Esta estructura es utilizada exclusivamente por los dispositivos Maestros y representa de forma lógica un nodo en la red. Se define estáticamente en un vector de nodos en cada Maestro.

```
typedef struct{
  unsigned int id;

  //synchronous part
  boolean S_supervisor_op;
  float S_timestamp;
  byte S_answer;
  byte S_length;
  byte S_data[8];

  //Asynchronous part
  boolean A_supervisor_op;
  float A_timestamp;
  byte A_ack;
  byte written;
  byte A_length;
  byte A_data[8];
}tCAN;

//vector de node
node nodes[NNODES + 1];
```

- **id:** ID del dispositivo con el que se ha iniciado una comunicación.
- **S_supervisor_op:** Se trata de un campo que indica si la comunicación síncrona con el nodo ha sido iniciada por el maestro o por el supervisor (recordemos que si bien un supervisor no está conectado directamente a la red, puede realizar peticiones a otros nodos a través del maestro al que se encuentra conectado). Sus valores posibles son true si es una comunicación del supervisor y false si es del maestro.
- **S_timestamp:** Cuando el valor es mayor que 0 indica que existe una comunicación síncrona con otro nodo.
- **S_answer:** Indica que se ha recibido una respuesta síncrona del nodo destino. Los posibles valores son:

  0 No se ha recibido respuesta.

  1 Se ha recibido una respuesta.

  2 Se ha recibido una respuesta, pero esta ha sido rechazada.

  Nota: Si se trata de una petición de otro maestro, esta propiedad adquiere otro significado, siendo 0 que el maestro no ha recibido peticiones de otros maestros ó 1 en caso contrario.
- **S_length:** Almacena la longitud del campo de datos recibidos, corespondiente a una respuesta síncrona.

- **S_data[8]:** Vector de 8 bytes utilizado para almacenar los datos de una respuesta síncrona. Funciona como buffer para almacenar la respuesta y ser procesada con posterioridad.

- **A_supervisor_op:** Se trata de un campo que indica si la comunicación asíncrona con el nodo ha sido iniciada por el maestro o por el supervisor (recordemos que si bien un supervisor no está conectado directamente a la red, puede realizar peticiones a otros nodos a través del maestro al que se encuentra conectado). Sus valores posibles son true si es una comunicación del supervisor y false si es del maestro.

- **A_timestamp:** Cuando el valor es mayor que 0 indica que existe una comunicación asíncrona con otro nodo.

- **A_ack:**Indica si ha habido una trama de reconocimiento emitida desde el nodo destino. Los posibles valores son:

  0 No se ha recibido una respuesta.

  1 Se ha producido una respuesta.

  2 La trama, *start* o *stop* previa ha sido rechazada.

  Nota: Si se trata de la entrada del maestro en la tabla, adquiere otro significado, siendo 0 que el maestro no ha recibido ningún "start" de otro maestro, 1 en caso contrario, 2 que se ha recibido un"stop" válido de otro maestro y 3 que el maestro se encuentra en un estado de envío activo.

- **written:** Es bastante probable que los envíos de tramas asíncronas se realicen a intervalos de tiempo menores a los que el maestro comprueba si ha llegado algún mensaje. Como este tipo de transmisión no garantiza la pérdida de un paquete se considera necesario informar al usuario de una sobreescritura en el buffer.

- **A_length:** Almacena la longitud del campo de datos recibidos, correspondiente a una respuesta asíncrona.

- **A_data[8]:** Vector de 8 bytes utilizado para almacenar los datos de una respuesta asíncrona. Funciona como buffer para almacenar la respuesta y ser procesada con posterioridad.

2.1.6.2.1. Identificadores Para optimizar la búsqueda de nodos se establecieron dos tipos de identificadores: reales y virtuales.

- **Identificador real:** Corresponde al ID real de cada nodo y son conocidos de antemano por el usuario administrador de la red.
- **identificador virtual:** El identificador virtual corresponde a la posición del nodo en el vector de nodos estáticos. El identificador virtual de un nodo puede difererir entre distintos maestros.

  Son asignados en funcion de la posición del nodo en el vector y son utilizados para acceder de forma directa a los nodos del vector.

Las dos imágenes a continuación, muestran un ejemplo del funcionamiento de los identificadores virtuales y reales. Los ID virtuales son utilizados por el programa principal para especificar el destinatario del mensaje. La librería obtiene el ID real del dispositivo accediendo a la tabla de nodes. Esta estrategia requiere añadir por parámetro el vector de nodos en todos los métodos de envío/recepción de mensajes.

```
#define NNODES 3 //Numero maximo de nodos
#define ID 0x06 //Identificador del nodo
tTouCAN::tCAN mensaje;  //Variable global
tTouCAN toucan;
//Vector de estructuras node
node nodes[NNODES+1];
.....

void setup() {
  ....

  //Inicializar nodos
  toucan.init_nodes(nodes, NNODES);

  //Configura los ID de los nodos con lo que
  //deseamos  establecer una comunicacion
  nodes[0].id=2;
  nodes[1].id=1;
  nodes[2].id=5;
  ....
}

void loop () {
 .....
 //Envio de un mensaje sincrono al nodo Esclavo con ID 5
 //El tercer parametro corresponde al ID virtual del nodo
 //con el que queremos comunicarnos.
 toucan.request(&mensaje, ID, 2, 3, nodes, 100,0, 1);
 ....
}
```

Figura 2.8: Programa de usuario - Ejemplo de acceso a través de identificadores virtuales.

```
/**
* Ask for a synchronous data
*
* @parameter tCAN msge
* @parameter int id_s Identificador de origen
* @parameter int id_d Identificador del nodo destino
* @parameter int length número de nodos
* @parameter node vector de nodos
* @parameter float timeout
* @parameter boolean is_supervisor_op
* @parameter boolean is_blockin
*/
int tTouCAN::request(tCAN *msge, int id_s, int id_d, int length,
                     node *nodes, float timeout,
                     boolean is_supervisor_op,
                     boolean is_blocking){

  //accede al nodo a través del identificador virtual
  if((millis()-nodes[id_d].S_timestamp<timeout)
        &&(nodes[id_d].S_timestamp!=0)){
        return PREVSTILLACTIVE;
  }
  //el tercer parámetro corresponde al id real del nodo destinatario.
  if (do_request(msge, id_s, nodes[id_d].id, length, is_blocking)) {
    nodes[id_d].S_timestamp=millis();
    if(is_supervisor_op){
        nodes[id_d].S_supervisor_op=true;
    }
  }
        ....
}
```

Figura 2.9: Fragemto librería TouCAN - Ejemplo de acceso a través de identificadores reales.

2.1.7. TouCAN Plantillas

Con la finalidad de reducir la curva de aprendizaje, TouCAN v1.0 ofrece un conjunto de plantillas que facilitan la implementación de distintos escenarios. Una plantilla consiste en una estructura de código predefinida donde el usario solo debe implementar un conjunto de callbacks. Las plantillas cubren distintos escenarios como la comunicación síncrona/asíncrona entre dos o tres nodos de distinta tipología incluyendo la comunicación con un Supervisor. La siguiente figura muestra una plantilla de un nodo Maestro que solicita peticiones síncronas / asincronas a un nodo Esclavo.

2.2. TouCAN Supervisor

Facilita la conexión de un PC a un nodo Master, vía interfaz serie USB, para realizar operaciones de monitorización del sistema o envío/recepción de paqutes a cualquier nodo de la red a través de un nodo Maestro.

2.2.1. Trama

La comunicación entre el Supervisor y el nodo Master se realiza a través de la interfaz serie mediante el envío secuencial de una serie de bytes que deben ser interpretados por ambas partes. La siguiente imagen representa la estructura de datos utilizada en la comunicación.

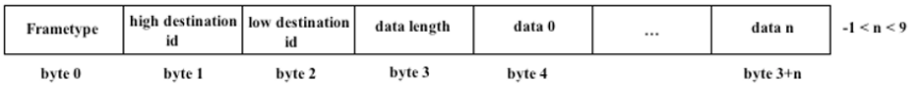


Figura 2.10: Comando utilizado por el nodo supervisor en el envío/recepción de mensajes.

Capítulo 3

Competencias

En este capítulo se enumeran las competencias cubiertas por el presente trabajo de fin de grado.

3.1. CII01

Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

Esta competencia ha sido desarrollada a lo largo del documento en los capítulos de "Introdución" 1,"Análisis del problema" 2 y "Desarrollo" 9. En dichos capítulos se detallan las causas y los hechos derivados de las decisiones de diseño tomadas.

3.2. CII02

Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.

En el capítulo "Metodología y planificación" 7, se indica el sistema de organización utilizado para alcanzar los objetivos propuestos. La valoración del impacto social y económico queda reflejada por su parte en el capítulo "Aportaciones" 4.

3.3. CII04

Capacidad para elaborar el pliego de condiciones técnicas de una instalación informática que cumpla los estándares y normativas vigentes.

Esta competencia queda cubierta en el capťulo "Pliego de Condiciones" 8 donde se especifican cada una de las condiciones asociadas al proyecto.

3.4. CII18

Conocimiento de la normativa y la regulación de la informática en los ámbitos nacional, europeo e internacional.

El cumplimiento de esta competencia queda descrita en el capítulo "Normativa y Legislación" 5.

Capítulo 4

Aportaciones

Actualmente la comunidad Arduino está en auge. Multitud de empresas y usuarios apuestan por esta tecnología para llevar a cabo proyectos de gran envergadura en distintas áreas como la robóticia, domótica, videojuegos, electrodomésticos entre otros. Por otro lado CAN[4] es un protocolo asentado utilizado por la industria desde hace varias décadas.

TouCAN se creó con el objetivo inicial de proporcinar una librería software libre fácil de utilizar y optimizada para microcontroladores con pocos recursos, que tomara ventaja de las capacidades de una red de plataformas Arduinos conectados a través de una red CAN Bus.

Este proyecto ha aportado un nivel más de robutez a la librería TouCAN aportando mejoras relacionadas con la detección y recuperación de fallos de comunicaciones y reset de los microcontroladores. La asignación dinámica simplifica comunicación entre dispositivos y permite la conexión en "Caliente". TouCANSniffer facilita la tarea de debug analizando el tráfico de mensajes en la red. Todas estas aportaciones convierten a TouCAN en una herramienta a ser considerada seriamente en proyectos donde se desee integrar redes de microcontroladres de forma económica, sencilla, eficiente y libre.

Capítulo 5

Normativa y Legislación

A continuación se incluye la legislación vigente que afecta al presente Trabajo Fin de Grado.

5.1. Normativa

5.1.1. Ley Orgánica de Protección de Datos.

La presente Ley Orgánica de Protección de Datos [12] tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar. Es de aplicación a los datos de carácter personal registrados en soporte físico, que los haga susceptibles de tratamiento, y a toda modalidad de uso posterior de estos datos por los sectores público y privado.

5.1.2. Código Tipo

Se definen como códigos deontológico o de buena conducta o práctica profesionales. Están regulados en la Ley Orgánica de Protección de Datos de Carácter Personal[12], que es la que establece en nuestro país los cimientos básicos de los aspectos legales en cuanto a protección de datos de carácter personal se refiere. Tienen por objeto adecuar lo establecido en la LOPD y en el RLOPD a las peculiaridades de los tratamientos efectuados por quienes se adhieren a ellos, conteniendo reglas o estándares específicos con el objetivo de armonizar los tratamientos de datos efectuados por los adheridos; facilitar el ejercicio de los derechos de los afectados y, en definitiva, favorecer el cumplimiento de lo dispuesto en la normativa de protección de datos.

5.1.3. Inscripción de un programa informático en LOPD

A continuación se detalla los requisitos necesarios para la inscripción de un programa informático en la LOPD.

- La totalidad del código fuente, en CD-ROM o en soporte papel, debidamente encuadernada y paginada.
- En los programas de ordenador editados ha de presentarse un resumen por escrito de al menos 20 folios del código fuente, siempre y cuando reproduzcan elementos esenciales del mismo. Deberá ir encuadernado y con portada donde figure el título y autor de la obra.
- Una memoria en soporte papel, debidamente encuadernada y paginada con el resumen de la aplicación, el lenguaje de programación, el entorno operativo, el listado de nombres de los ficheros que contiene y el diagrama de flujo.
- Puede aportarse el ejecutable del programa, de forma opcional.
- Los escritos y solicitudes que se dirijan a cualqueira de las oficinas del Registro podrán presentarse en las formas y ante los órganos que prevé la ley 30/1992, de Régimen Jurídico de las Administraciones Públicas y del Procedimiento Administrativo Común. También podrán presentarse en cualquiera de los registros a que se refiere el presente reglamente, tanto central como territorial, lo cuales los remitirán, el dia siguiente al de su presentación, al registro indicado en la solicitud.

5.2. Licencias

A continuación se detallan algunas de las licencias que afectan al software utilizado durante el desarrollo del presente proyecto.

5.2.1. GNU GPL

La Licencia Pública General de GNU[9] es la licencia más ampliamente usada en el mundo del software y garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios. Esta licencia fue creada originalmente por Richard Stallman fundador de la Free Software Foundation (FSF) para el proyecto GNU.

5.2.2. LGPL

La Licencia Pública General Menor (del inglés: Lesser General Public License o LGPL) [10] es una modificación de la licencia GPL descrita anteriormente. La LGPL permite que los desarrolladores utilicen programas bajo la GPL o LGPL sin estar obligados a someter el programa final bajo dichas licencias.

5.2.3. Eclipse Public License

La Licencia Pública Eclipse (EPL) [7] es una licencia de software de código abierto utilizada por la Funciación Eclipse para su software. Está diseñada para ser una licencia favorable a los negocios. No requiere ningún seguimiento en los cambios y sólo exige la publicación del código fuente cuando las modificaciones

se consideran un trabajo derivado y no una extensión o un módulo separado. Los trabajos derivados deben ser publicados siempre bajo la licencia EPL.

5.2.4. Creative Commons

Las licencias Creative Commons[5] ó CC tienen su origen en la licencia GNU GPL, cuyo objetivo es el de ofrecer al autor de una obra un mecanismo sencillo y eficáz a la hora de establecer una serie de condiciones asociadas a la obra. Las condiciones que se pueden imponer son las siguientes:

- **Reconocimiento (Attribution):** *En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.*

Figura 5.1: Reconocimiento.

- **No Comercial (Non commercial):** *La explotación de la obra queda limitada a usos no comerciales.* ó

Figura 5.2: No Comercial.

- **Sin obras derivadas (No Derivate Works):** *La autorización para explotar la obra no incluye la transformación para crear una obra derivada.*

Figura 5.3: Sin obras derivadas.

- **Compartir Igual (Share alike):** *La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.*

Figura 5.4: Compartir Igual.

Capítulo 6

Requisitos

Para poder conseguir los objetivos marcados inicialmente del proyecto se ha hecho uso, durante el desarrollo de éste, de una serie de recursos. Se han desglosado en dos grupos: hardware y software.

6.1. Hardware

A nivel de hardware las necesidades fueron las siguientes:

1. **Arduino Uno rev3**. 4 unidades.
2. **CAN-BUS shield**. 4 unidades.
3. **Protoboard**.
4. **Resistencias**. 2 unidades de 120 ohm.
5. **Cables**. Par de cobre trenzado.
6. **Cables USB de A a B**. 4 Unidades
7. **PC**. En concreto se ha utilizado un PC con CPU Intel Core QuadCore a 2Ghz, Memoria ram de 8GB y una gráfica GeForce 9400M.

6.2. Sofware

A nivel de software fue necesario:

1. **Distribución GNU/Linux** En concreto se utilizó Debian 6.0 de 64 bits.
2. **Sofware de desarrollo** Eclipse 3.8, GNU g++ y Arduino IDE(processing).
3. **Sofware documental** LaTexT, ImageMagick 6.7.7-10.

Capítulo 7

Metodología y Plan de Trabajo

Como metodología de desarrollo se utilizaron las técnicas de desarrollo software propias del Proceso Unificado (Unified Process)[20]. El Proceso Unificado proporciona un marco de desarrollo genérico adaptable a proyectos específicos cuyas características son las siguientes:

1. **Iterativo e Incremental:** El Proceso Unificado está compuesto por cuatro fases distintas: Inicio, Elaboración, Construcción y Transición. En cada una de las fases se hará un determinado número de iteraciones con el objetivo de mejorar e incluir mejoras de funcionalidades dando lugar a un incremento del proyecto en desarrollo.

2. **Dirigido por los casos de uso:** Por medio de los casos de uso se especificará en las iteraciones los requisitos funcionales y los contenidos.

3. **Centrado en la arquitectura:** En el desarrollo del software no se usará un único modelo que describa todo el sistema. Se optará por un enfoque con múltiples modelos y vistas que describan la arquitectura de software del sistema.

4. **Enfocado en los riesgos:** Se requiere identificar los riesgos críticos en una etapa temprana. De este modo los resultados de cada iteración deben seleccionarse en un orden que asegure que los riesgos principales son considerados primero.

A continuación, en la figura 7.1 se puede observar de forma gráfica la evolución típica de un proyecto que sigue el Proceso Unificado.

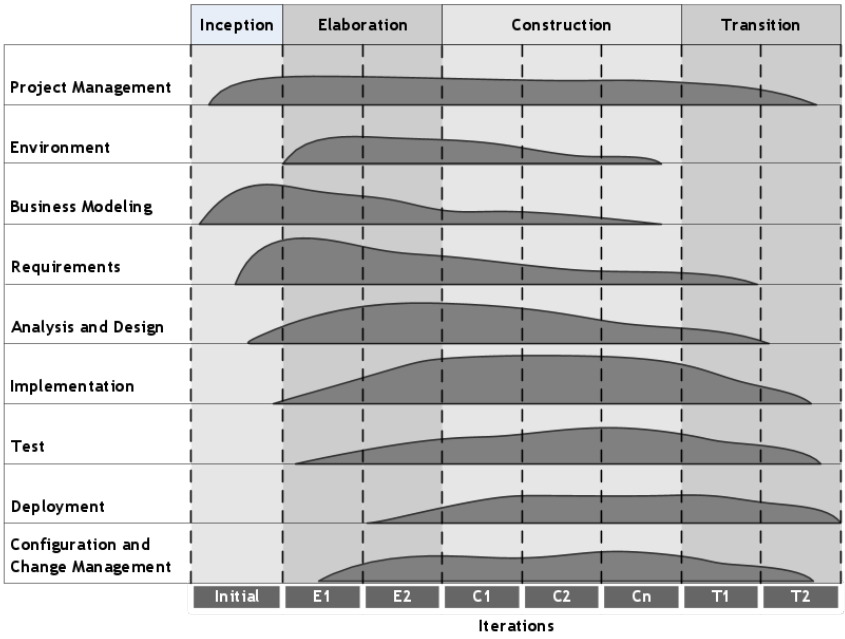


Figura 7.1: Evolución de las diferentes etapas de un proyecto desarrollado mediante el Proceso Unificado. *Fuente: http://es.wikipedia.org/*

En relación al Plan de Trabajo a llevar a cabo, las etapas a cubrir durante el desarrollo del TFG propuesto fueron las siguientes:

1. **Familiarización con la librería TouCAN**. Estudio, familiarización y pruebas del entorno Arduino, CAN Bus y la librería TouCAN.

2. **Diseño y desarrollo**. Análisis, diseño, desarrollo e implementación de cada una de las nuevas características y funcionalidades incorporadas en la librería y descritas en el capítulo 9 "Desarrollo"

3. **Fase de test.** Se ha realizado una amplia batería de test donde se verifica el funcionamiento de las nuevas funcionalizades y su compatibilidad con la versión anterior.

4. **Documentación y Defensa**. Elaboración del Documento de Trabajo de Fin de Grado a partir de toda la documentación y datos obtenidos durante la realización de las diferentes etapas en las que se ha organizado.

La planificación temporal del desarrollo del TFG en base a las tareas previstas en cada etapa que se llevó a cabo fue la siguiente:

Etapas	Dedicación Estimada
Etapa 1: Familiarización Arduino y Bus CAN y Librería TouCAN v1	30 horas
Etapa 2: Refactorización de la librería TouCAN	45 horas
Etapa 3: Mejora protocolo TouCAN (fallos, reinicios y caídas)	75 horas
Etapa 4: Mejora protocolo toucan (Asignación Dinámica de identificadores)	50 horas
Etapa 5: Integración prototipo real de prueba	50 horas
Etapa 6: Documentación y Defensa	50 horas
Dedicación Estimada Total	300 horas

Tabla 7.1: Tabla correspondiente a la dedicación estimada en cada etapa del Plan de Trabajo.

Capítulo 8

Pliego y Condiciones

8.1. Objeto de este pliego

El pliego de condiciones técnicas tiene por objeto la definición de condiciones generales, técnicas y económicas asociadas a la ejecución y utilización de la librería TouCAN.

8.2. Pliego de Condiciones Generales

El presente trabajo está destinado a la incorporación de nuevas funcionalidades y mejoras en la librería TouCAN. Las características principales del presente proyecto son:

- Capacidad de asignar identificadores a los dispositivos de la red de manera dinámica.
- Recuperar el estado de un nodo Server/Master después de un reset o de caida.
- Monitorizar el estado de los dispositivos en la red.
- Optimización de la librería en la utilización de recursos y facilidad de uso.
- Crear una herramienta que permita recopilar, monitorizar y analizar los paquetes generados en una red.

8.3. Pliegos de especificaciones técnicas

8.3.1. Especificaciones de materiales, equipos y sofware

Las especificaciones del hardware y software de los cuales se componen el proyecto se describe ampliamente en el apartado de requisitos.6

8.3.2. Especificaciones de ejecución

El proceso llevado a cabo en la elaboración del proyecto se puede encontrar especificado dentro del capítulo 9, Desarrollo.

8.4. Pliego de Clausulas Administrativas Particulares

El presupuesto asociado al proyecto asciende a un total de 16.980 euros.

Recurso	Coste
Analista/Desarrolador (250 horas)	60 euros/h.
Hardware	180 euros.
Total	15180 euros (IGIG incl.).

Tabla 8.1: Tabla correspondiente al presupuesto asociado al proyecto.

8.5. Licencia de Uso

Al adquirir el presente proyecto, el comprador no adquiere ningún poder ni titularidad sobre el software que contiene. Sin embargo podrá modificarlo y distribuirlo si así lo desea, tal y como establece la licencia pública general GNU versión 3.[9]

Capítulo 9

Desarrollo

En este capítulo explicaremos las nuevas funciones y mejoras incorporadas a la libreríra TouCAN explicando la naturaleza de su diseño.

9.1. TouCAN Topología

Originalmente, TouCAN v1.0 estableció tres tipos de elementos que conforman un sistema: Supervisor, Maestro y Esclavo. En esta revisión se ha incluido tres nuevos elementos a la topología de dispositivos denominados: Servidor, SupervisorSniffer y NodoSniffer. A continuación se describen las principales características de cada uno.

- **Nodo Servidor:**Además de las características de un Master incluye la capacidad de asignar identificadores de forma dinámica y monitorización del estado de los dispositivos en una red. Así, en una red podrán coexistir distintos nodos de tipo Maestro o Esclavo. Sin embargo, solo puede haber un nodo Server y este siempre tiene el identificador 0. Recordar, un nodo Supervisor puede conectarse a un nodo Maestro o Servidor.
- **NodoSniffer:** Este nodo es parte de la herramienta TouCANSniffer y realiza únicamente dos funciones. Recopilar todo el tráfico generado en la red y reenviar los paquetes al SupervisorSniffer, a través del puerto serie, para ser procesados.
- **SupervisorSniffer:** Al igual que el Supervisor, no corresponde a un nodo de la red propiamente dicho. Es una applicación que establece una comunicación serie con el nodo Sniffer para recopilar y monitorizar todo el tráfico generado en la red.

La imagen a continuación representa el esquema de una red TouCAN con los nuevos elementos incluidos.

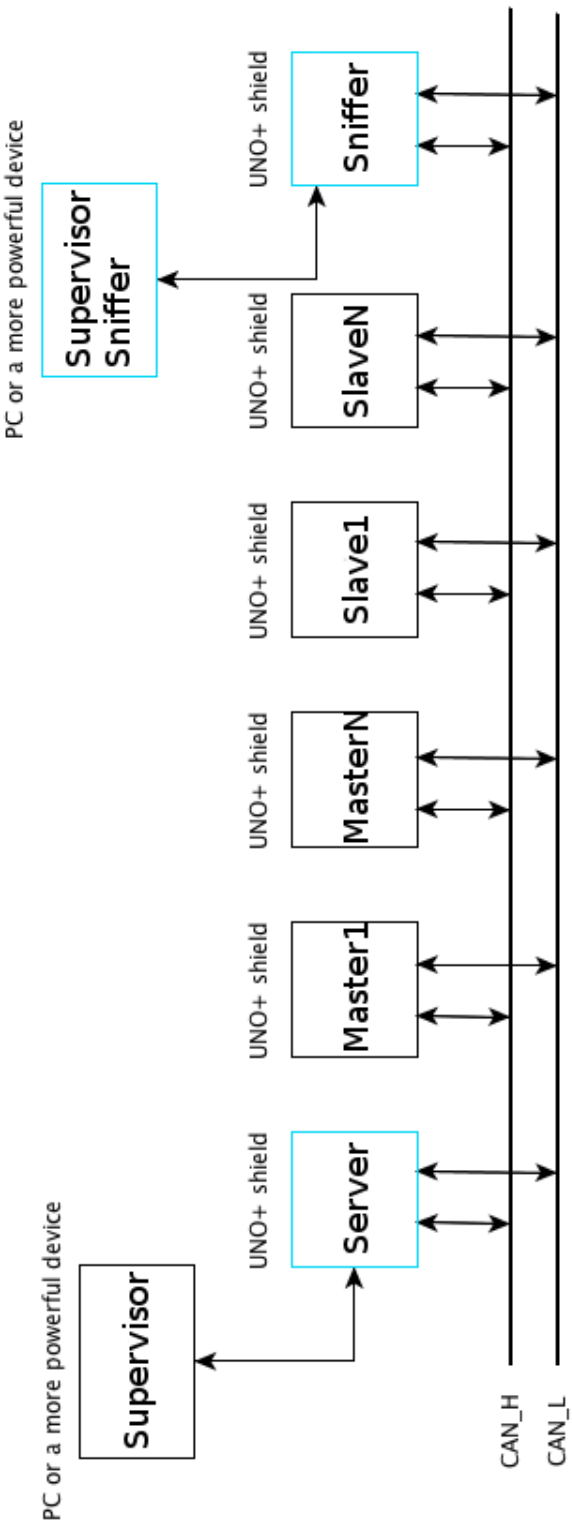


Figura 9.1: Esquema de la red TouCAN v2.0

9.2. Tramas TouCAN

A las tramas ya existentes en la versión v1.0 de TouCAN se han incorporado nuevos comandos responsables de la asignación dinámica de ID y de la monitorización de dispositivos en la red. En esta revisión los nodos son reconocidos por un nombre de cinco caracteres. A continuación se detallan las tramas incorporadas en esta revisión.

9.2.1. Proceso de asignación dinámica de ID

- **Request ID:** Codificación en el protocolo, 11100 (macro REID). Función utilizada por los nodos Master y Slave para solicitar un ID al nodo Server. En los cinco primeros bytes del campo de datos contienen el nombre del dispositivo que realiza la solicitud. La imagen 9.2 corresponde a una trama REID.

  Data0..Data4 : Los primeros cinco bytes contienen el nombre del dispositivo.

  Data5..Data7 : Sin utilizar.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
BROADCAST (11111111)	REID (11100)	0	5	L	U	Z	0	1			

Figura 9.2: TouCAN v2.0 - Ejemplo trama REID

- **Response ID:** Codificado con el valor, 11101 (macro BRID). Trama tipo broadcast, es enviada siempre por el nodo Server a todos los dispositivos de la red, aunque solo es procesada por el nodo que realizó la petición "Request ID". La figura 9.3 es un ejemplo de trama BRID.

  Data0..Data4 : Los primeros cinco bytes contienen el nombre del dispositivo que realizó la petición.

  Data5 : Puede contener dos valores, 0 operación realizada con éxito ó 255 si se ha producido un error.

  Data6..Data7 Los dos últimos bytes del campo de datos almacenan el ID asignado al dispositivo o un código de error según el valor del byte 5.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	BRID (11101)	BROADCAST (11111111)	5	L	U	Z	0	1	0	17	

Figura 9.3: TouCAN v2.0 - Ejemplo trama BRID

- **Confirm ID:** Codificada en el protocolo con el valor, 11110 (macro BCID). Trama tipo broadcast, enviada por el nodo que realizó la petición "Request ID" para confirmar el ID asignado al nodo Server e informar al resto de dispositivos Master de la red. Ejemplo de trama BCID en la imagen: 9.4

  Data0..Data4 : Los primeros cinco bytes contienen el nombre del dispositivo.

  Data5..Data7 : Sin utilizar.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
17	BCID (11110)	BROADCAST (11111111)	5	L	U	Z	0	1			

Figura 9.4: TouCAN v2.0 - Ejemplo trama BCID

9.2.2. Proceso de reconocimiento de dispositivos en la red

- **Request check devices network:** Trama codificada con el código, 11010 (macro BCDN). Es un paquete broadcast, utilizado únicamente por el nodo Server para recuperar la lista de dispositivos en la red después de un reinicio. Ejemplo de trama BCDN en la imagen: 9.5

  Data0..Data7 : Esta trama no hace uso del campo data.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	BCDN (11010)	BROADCAST (11111111)	0								

Figura 9.5: TouCAN v2.0 - Ejemplo trama BCDN

- **Response check devices network:** Codificado en el protocolo con el código, 11011 (macro RCDN). Respuesta de los nodos Master y Slave a una petición "Request check devices network". El destinatario del paquete el el nodo Server. Ejemplo de trama RCDN en la figura 9.6

  Data0..Data4 : Los primeros cinco bytes contienen el nombre del dispositivo.

  Data5..Data7 : Sin utilizar.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
17	RCDN (11011)	0	5	L	U	Z	0	1			

Figura 9.6: TouCAN v2.0 - Ejemplo trama RCDN

9.2.3. Proceso verificación de estados de los dispositivos.

- **Request echo:** Codificada en el protocolo con el valor, 11000 (macro ECSE). Paquete echo enviado por el nodo Server a cada uno de los dispositivos de la red para verificar cuales están activos y cuales no. La imagen 9.7 corresponde a un ejemplo de trama ECSE.

  Data0..Data7 : Esta trama no hace uso del campo data.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	ECSE (11000)	BROADCAST (11111111)	0								

Figura 9.7: TouCAN v2.0 - Ejemplo trama ECSE

- **Response echo:** Codificación en el protocolo con el valor, 11001 macro (REEC). Emitido por los nodos Master y Slave como respuesta a una petición "Request echo". Ejemplo de trama REEC en la figura: 9.8

  Data0..Data7 : Esta trama no utiliza el campo data.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
17	REEC (11000)	0	0								

Figura 9.8: TouCAN v2.0 - Ejemplo trama REEC

- **Remove device:** Codificado en el protocolo con el valor, 10100 (macro ECRD). Paquete broadcast utilizado para informar que un dispositivo ha causado baja en la red. Enviado por el nodo Server y procesado únicamente por los nodos Master. Los dos primeros bytes del campo de datos almacenan el ID del nodo a eliminar. Este paquete permite mantener actualizada la lista de dispositivos de todos los nodos Master de la red.

  Data0..Data1 : Identificador del dispositivo que ha causado baja en la red.

  Data5..Data7 : Sin utilizar.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	ECRD (10100)	BROADCAST (11111111)	2	17							

Figura 9.9: TouCAN v2.0 - Ejemplo trama ECRD

9.2.4. Proceso actualización lista dispositivos - Nodo Maestro

- **Request devices list:** Codificado con el código, 10101 (macro: DLGT). Trama utilizada por un nodo Master para solicitar la lista completa de dispositivos al nodo Server. La imagen 9.10 corresponde a un ejemplo de la trama DLGT

  Data0..Data7 : Esta trama no hace uso del campo data.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
17	DLGT (10101)	0	0								

Figura 9.10: TouCAN v2.0 - Ejemplo trama DLGT

- **Response devices number:** Codificación en el protocolo con el valor, 10110 (macro DLNU). Devuelve el número de dispositivo en en los dos primeros bytes del campo de datos. Enviado por el nodo Server al Master que realizó la petición: "Request devices list". Ejemplo de trama en la figura: 9.11

  Data0..Data1 : Número de registros en la tabla de dispositivos.

  Data2..Data7 : Sin utilizar.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	DLNU (10110)	17	2	13							

Figura 9.11: TouCAN v2.0 - Ejemplo trama DLNU

- **Response device name:** Codificaco en el protocolo con el valor, 10111 (macro DLNA). Este paquete es utilizado por el nodo Server para enviar al Master la lista de dispositivos. Envía tantos paquetes como dispositivos contenga la tabla. El campo de datos contiene el ID y el nombre del nodo.

  Data0..Data4 : Los primeros cinco bytes contienen el nombre del dispositivo.

  Data5..Data6 : Identificador del dispositivo

  Data7..Data7 : Sin utilizar

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	DLNA (10111)	17	7	L	U	Z	0	2	37		

Figura 9.12: TouCAN v2.0 - Ejemplo trama DLNA

9.3. Refactorización y Optimización

Antes de comenzar con la desarrollo de las nuevas funcionalidades se consideraron necesarias la optimización de algunos aspectos de la librería con la única finalidad de facilitar la implementación de las nuevas mejoras. Los siguientes apartados describen los cambios realizados.

9.3.1. Refactorización

La libería TouCAN está compuesta por una única clase con las propiedades y métodos de los distintos nodos en un único archivo. Esto no impide a un nodo Esclavo iniciar una comuniciación con otros dispositivos, cuando por definición los nodos Esclavos son dispositivos pasivos que no pueden comenzar una comunicación. Por otro lado, a medida que aumenta la complejidad de la librería, con mejoras y nuevas funcionalidades, esta estrategia puede resultar difícil de mantener. Con la finalidad de mantener la premisia inicial de una librería amigable y fácil de ampliar se decidió refactorizar la librería aplicando el principio de responsabilidad única SRP [19]. Donde, cada objeto debe tener una simple responsabilidad, y que debe estar contenida únicamente en la clase.

Las dos imágenes a continuación muestran una comparativa entre los diagramas de clases de TouCAN revisión v1.0 y v2.0

Figura 9.13: Diagrama Clases, TouCAN v1.0

9.3.1.1. Descripción Clases

A continuación se describen las características principales de cada una de las clases de la librería TouCAN revisión v2.0.

tCAN: Representa un mensaje en la librería TouCAN. Actúa como buffer de recepción / transmisión del MCP2515[8] al enviar o recibir mensajes a través del bus CAN[4]

TouCANSPIManager: Encapsula todas las operaciones específicas sobre el controlador MCP2515[8]. Es el responsable, en última instancia, en transmitir/recibir los mesajes a través del bus CAN[4].

TouCANNodeBase: Clase abstracta. Encapsula los métodos y propiedades comunes a TouCANNode y TouCANSniffer.

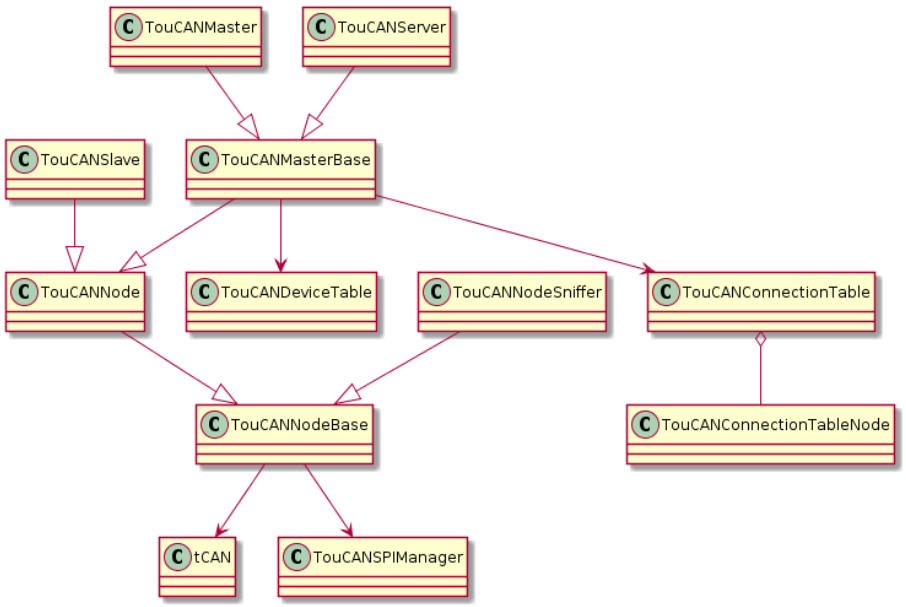


Figura 9.14: Diagrama de clases, TouCAN v2.0

TouCANNode: Clase abstracta. Encapsula todas las operaciones comunes a los distintos nodos. Entre sus responsabilidades destaca la inicialización del dispositivo, la solicitud de un identificador y la respuesta a una petición síncrona/asincrona.

TouCANSniffer: Esta clase deriva de TouCANNode y añade los métodos necesarios para recopilar todos los paquetes de la red y reenviarlos a un nodo SupervisorSniffer a través del puerto serie.

TouCANSlave: Esta clase hereda de TouCANNode y representa un dispositivo Esclavo. Contiene todas las operaciones específicas del nodo.

TouCANMasterBase: Clase abastracta que deriva de TouCANNode y contiene las funciones comunes a los nodos: Maestro y Servidor.

TouCANMaster: Esta clase hereda de TouCANMasterBase y representa un dispositivo tipo Maestro. Contiene todas las operaciones esecíficas del nodo.

TouCANServer: Esta clase hereda de TouCANMasterBase y representa un dispositivo TipoServer. Contiene todas las operaciones específicas del nodo.

TouCANConnectionTable: Esta clase es la responsable de almacenar la información sobre las comunicaciones activas de un nodo tipo Maestro o Servidor con el resto de dispositivos en la red. El tamaño máximo es establecido por la constante: *TOU_CAN_CONNECTION_TABLE_SIZE*. La primera posición del vector de comunicaciones está reservado para atender a las peticiones de otros nodos Maestros o del Servidor.

9.3.2. Tabla de conexiones

La tabla de conexión surge con la finalidad de reducir el uso de recursos necesarios para mantener la lista de dispositivos y limitar el número de conexiones simultáneas. Hay que recordar, que en la primera versión de TouCAN, el programa de usuario es el responsable de declarar y configurar el vector de nodos estáticamente. En esta revisión la operación recae sobre el propio framework. Este último punto está directamente relacionado con la optimización de los métodos de la API responsables de enviar/recibir mensajes. La imagen 9.18 reflejan los cambios que esta mejora implica en un programa de usuario.

Antes de enviar una trama síncrona o asíncrona es necesario llamar al método, getDeviceIDByName(...), para obtener el ID del dispositivo con el que se desea establecer una comunicación. Por otro lado, antes de enviar un paquete, el framework verifica la existencia de una entrada libre en la tabla de comunicación para realizar el envío. Si no hay una disponible el mensaje no es enviado y se notifica al usuario con un mensaje de error. El siguiente diagrama de flujo, imagen:9.15, describe como un nodo Maestro obtiene un nodo de conexión antes de enviar un mensaje.

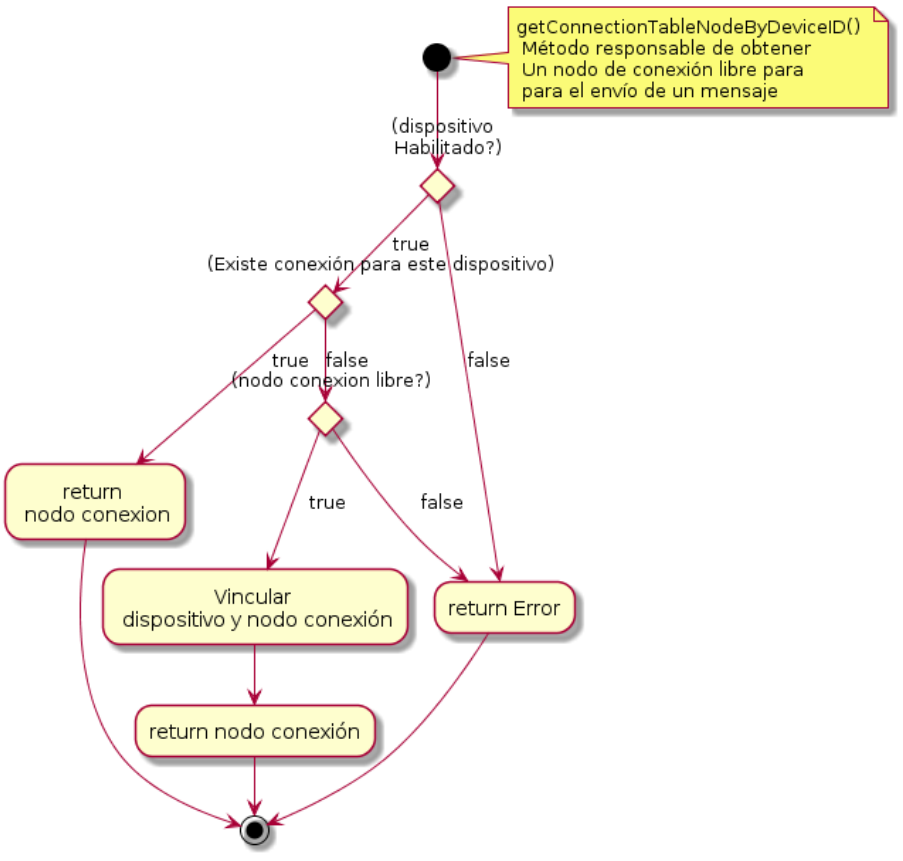


Figura 9.15: Diagrama de flujo - Obtener nodo de conexión

Las entradas en la tabla de conexiones son liberadas cuando un dispositivo causa baja en el sistema o al finalizar una comunicación síncrona o asícrona.

9.3.2.1. Estructura de datos

TouCANDevice es una estructura de datos que representa un nodo en la red. La clase TouCANDeviceTable define un vector de tamaño, *TOU_CAN_DEVICE_TABLE_SIZE*, donde almacena la información correspondiente a cada uno de los dispositivos de la red.

```
typedef struct {
  unsigned int id;
  char name[5];
  byte state;
  byte connection_id;
  byte echo;
  float time_activity;
}TouCANDevice;
```

- **unsigned int id:** Contiene el ID del dispositivo.
- **char name[5]:** Almacena el nombre del dispositivo.
- **byte state:** Almacena uno de los siguientes valores:

  DEVICE_STATUS_UNINITIALIZED Valor por defecto. Dispositivo no inicializado.

  DEVICE_STATUS_PENDING_CONFIRMATION Asignación temporal de ID. Pendiente de confirmación por parte del nodo.

  DEVICE_STATUS_CONFIRMED Dispositivo inicializado correctamente.

- **byte connection_id:**Identificador de la tabla de conexión asociado al dispositivo. Esta propiedad es configurada al iniciar una comunicación.
- **byte echo:** Propiedad utilizada al realizar la tarea de monitorización de dispositivos en la red.

  0 Valor por defecto.

  1 Paqute echo enviado.

  2 Recibida respuesta a paquete echo.

- **float time_activity:** Sello temporal del último paquete recibido.

El número de entrada de las tablas TouCANDevice y TouCANConnectionTable están definidas en las macros TOU_CAN_DEVICE_TABLE_SIZE y TOU_CAN_CONNECTION_TABLE_SIZE respectivamente. En TouCAN v2.0 el **Struct node** pasa a llamarse TouCANTableNode. Las dos imágenes a continuación, figura 9.16 y 9.17, describen las diferencias entre la tabla de conexiones de la versión v1.0 y v2.0

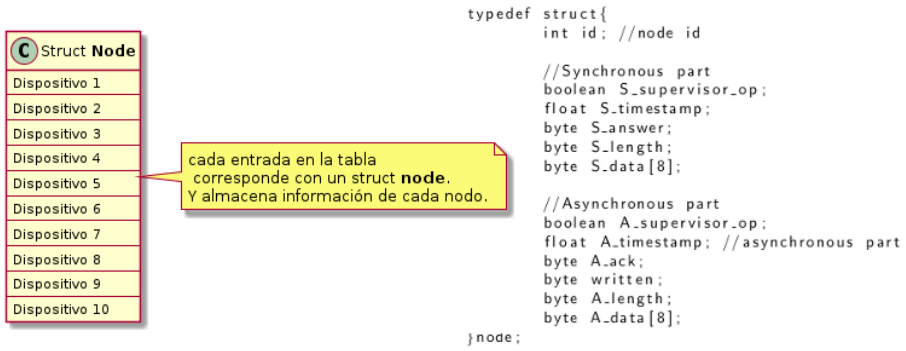


Figura 9.16: TouCAN v1.0 - Tabla Conexion

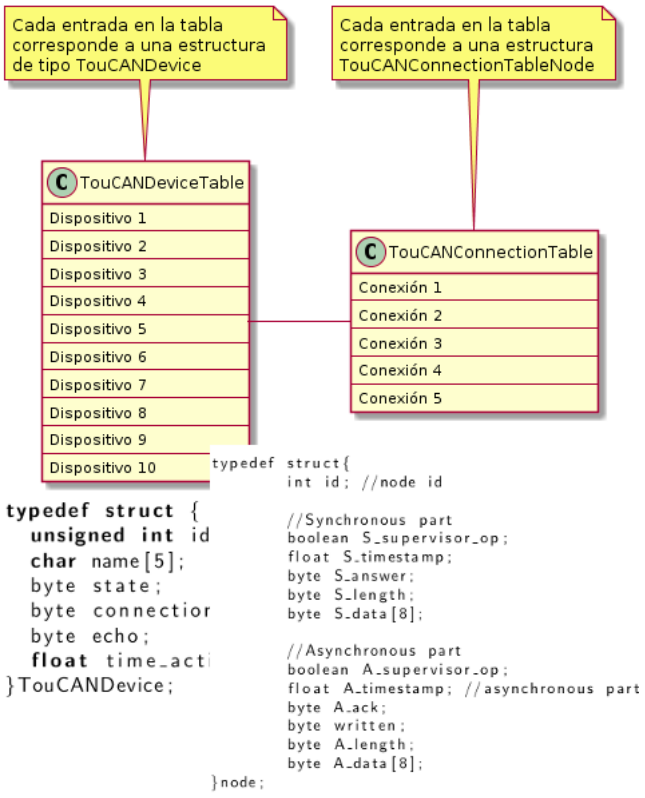


Figura 9.17: TouCAN v2.0 - Tabla Conexion

Figura 9.18: Tabla Conexion - Optimización interfaz

9.4. Asignación dinámica de identificadores

9.4.1. Descripción

Actualmente, en la versión v1.0 de TouCAN, la asignación de identificadores se realiza de forma estática y es responsabilidad del programador asignar un ID válido para evitar duplicados. Esto implica conocer con anterioridad los ID de los dispositivos con los que se desea establecer una comunicación. Por otro lado, es imposible realizar una conexión en caliente debido a la necesidad de parar el sistema por completo y configurar el identificador del nuevo nodo en todos los dispositivos de la red.

La asignación dinámica de ID incorporada en esta revisión delega la responsabilidad de asignar identificadores sobre el nodo Servidor y evita conocer con antelación los ID de los dispositivos con los que establecer una conexión. Los nodos Maestro y Serividor tienen capacidad, en tiempo de ejecución, de obtener el ID de un dispositivo a través de su nombre y de esta forma establecer una comunicación. Además, la asignación dinámica permite la posibilidad de "conexión en caliente" (del inglés "hot plugging") sin que estos queden aislados del sistema o del supervisor.

El diagrama de secuencia de la figura 9.19 corresponde a un sistema compuesto por cuatro dispositivos: un Servidor, dos nodos maestros, MAS01 y LUZ01 y un nodo esclavo, SLAVE. Donde LUZ01 realiza una petición de solicitud de identificador al nodo Servidor.

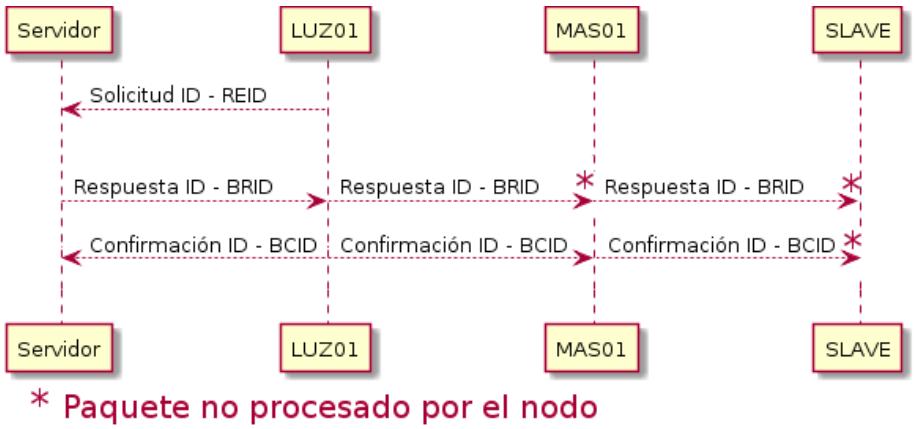


Figura 9.19: Asignación Dinámica ID - Respuesta petición RequestID - ID = 17

9.4.2. Tramas

A continuación se describen las tramas que intervienen en el proceso de asignación dinámica de ID.

- **REID:** Función utilizada por los nodos Maestros y Esclavos para solicitar un ID al nodo Servidor. En los cinco primeros bytes del campo de datos contienen el nombre del dispositivo que realiza la solicitud.
- **BRID:**Trama broadcast, enviada por el nodo Servidor a todos los dispositivos de la red, aunque solo es procesada por el nodo que realizó la petición "Request ID".
- **BCID:** Trama broadcast, enviada por el nodo que realizó la petición "Request ID" para confirmar el ID asignado al nodo Servidor e informar al resto de dispositivos Maestro de la red su ID.

9.4.3. Implementación

El proceso de asignación dinámica de ID comienza con la llamada al método, *requestID()* por parte de un nodo *Maestro* o *Esclavo*. El paquete lleva como destinatario el nodo Servidor (ID = 0), el campo origen tiene el valor 255 (macro *BROADCAST_IDENTIFIER_ID*) y los cinco primeros bytes del campo de datos contienen el nombre del dispositivo. La siguiente imagen, 9.20 corresponde a una trama de solicitud de ID.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
BROADCAST (11111111)	REID (11100)	0	5	L	U	Z	0	1			

Figura 9.20: Asignación Dinámica ID - Petición RequestID

Cuando el nodo Servidor recibe la solicitud, verifica el número de entradas en la tabla de dispositivo y que no exista una con el mismo nombre insertado con anterioridad. Tanto si la verificación es correcta como no, envía un paquete broadcast especificando en el campo de datos la siguiente información:

- **data0..data4:** nombre del dispositivo que realizó la solicitud.
- **data5:** Resultado de la operación. Puede contener dos valores, 0 Operación realizada con éxito ó 255 error al procesar la solicitud.
- **data6..data7:** Contiene el ID del dispositivo si la operación finaliza correctamente o un código de error en caso contrario.

  TOU_CAN_DEVICE_TABLE_DEVICE_SAME_NAME (65282).
  Existe un dispositivo con el mismo nombre insertado con anterioridad.

  TOU_CAN_DEVICE_TABLE_ERROR_TABLE_FULL (65281).
  Tabla de dispositivo completa.

Las dos imágenes 9.21 y 9.22 corresponden a una respuesta de solicitud de ID. La primera asigna correctamente el identificador 17 al nodo con el nombre LUZ01. La segunda muestra un mensaje de error con el código: 65282

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	BCID (11101)	BROADCAST (11111111)	8	L	U	Z	0	1	0	1	7

Figura 9.21: Asignación Dinámica ID - Respuesta petición RequestID - ID = 17

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	BCID (11101)	BROADCAST (11111111)	8	L	U	Z	0	1	255	65282	

Figura 9.22: Asignación Dinámica ID - Respuesta petición RequestID - Error

Todos los nodos de la red recibirán el paquete broadcast anterior, pero sólo será procesado por el dispositivo que realizó la petición. Si el paquete contiene un ID válido, el nodo actualiza el ID, configura los filtros extendidos y envía un paquete broadcast de confirmación, figura 9.23. Este comando es utilizado por el Servidor a modo de confirmación y por los nodos Maestro para actualizar la lista de dispositivos.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
17	BCID (11110)	BROADCAST (11111111)	5	L	U	Z	0	1			

Figura 9.23: Asignación Dinámica ID - Mensaje BROADCAST de confirmación.

9.5. Control frente a fallos de comunicación, reinicio y reset

A continuación se describen las mejoras introducidas en la librería TouCAN revisión v2.0 en la detección de fallos de comunicación y reinico de microcontroladores.

9.5.1. Nodo Server - Reconocimiento dispositivo en la red (Reset)

9.5.1.1. Descripción

Cuando un nodo Servidor sufre un reinicio, las entradas almacenadas en la tabla de dispositivo se pierden al igual que las conexiones activas. Esta situación genera inestabilidad en el sistema al quedar el Servidor aislado del resto de los componentes de la red. Por otro lado, la incorporación de un dispositivo después del reset generaría conflicto al poderse asignar un ID ya existente en la red. La imagen, 9.24 representa este escenario.

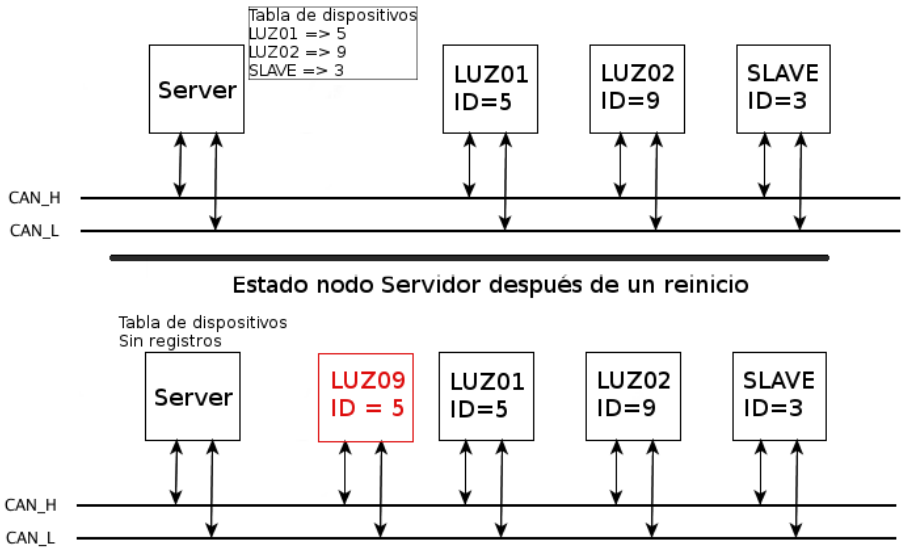


Figura 9.24: Estado nodo Servidor después de un reinicio.

El método *TouCANServer::checkNodesInNetwork()*, disponible únicamente en el nodo Servidor, garantiza la recuperación de la tabla de dispositivos después de un reset. El proceso consiste en solicitar el ID y nombre de los dispositivos activos en la red y actualizar la tabla de dispositivo con la información de cada uno. Es recomendable invocar el método desde la fución setup del programa de usuario para disponer de los datos actualizados al inicio del programa principal en el servidor.

El diagrama de secuencia de la figura 9.25 corresponde a una petición de reconocimiento de dispositivos realizada por el nodo Servidor después de un reinicio.

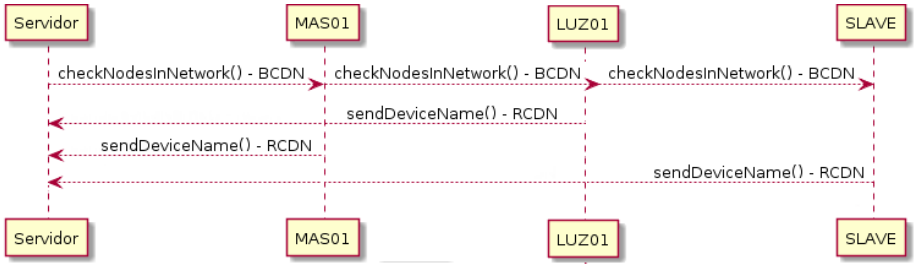


Figura 9.25: TouCAN v2.0 - Diagrama de secuencia - checkNodesInNetwork

9.5.1.2. Tramas

A continuación se describen las tramas que intervienen en el proceso de recuperación de la tabla de dispositivos de un nodo Servidor depués de un reset o reinicio.

- **BCDN:** Paquete broadcast, utilizado únicamente por el nodo Server para solicitar el ID y nombre de cada uno de los dispositivos de la red.
- **RCDN:** Respuesta de los nodos Master y Slave a una petición "Request check devices network".

9.5.1.3. Implementación

Como se comentó en el párrafo anterior, el método responsable de actualizar la lista de dispositivos es *checkNodesInNetwork*. Este método envía un paquete broadcast, identificado por la macro BCDN, a todos los dispositivos de la red solicitando el identificador y el nombre de cada uno. La figura 9.26 corresponde a una trama BCDN.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	BCDN (11010)	BROADCAST (11111111)	0								

Figura 9.26: TouCAN v2.0 - Trama BCDN.

Todos los nodos en la red deben responder a un mensaje *BCDN* con el comando *RCDN* configurando el campo data con el nombre del dispositivo. La imagen 9.27

corresponde a una trama RCDN enviada por un nodo con identificador 17 y nombre LUZ01.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
17	RCDN (11011)	0	5	L	U	Z	0	1			

Figura 9.27: TouCAN v2.0 - Trama RCDN.

El nodo Servidor actualiza la lista de dispositivos con la información obtenida en los paquetes *RCDN*. Durante el proceso de reconocimiento todos los paquetes distintos de *RCDN* son descartados. El diagrama 9.28 describe el proceso completo de detección de equipos en la red.

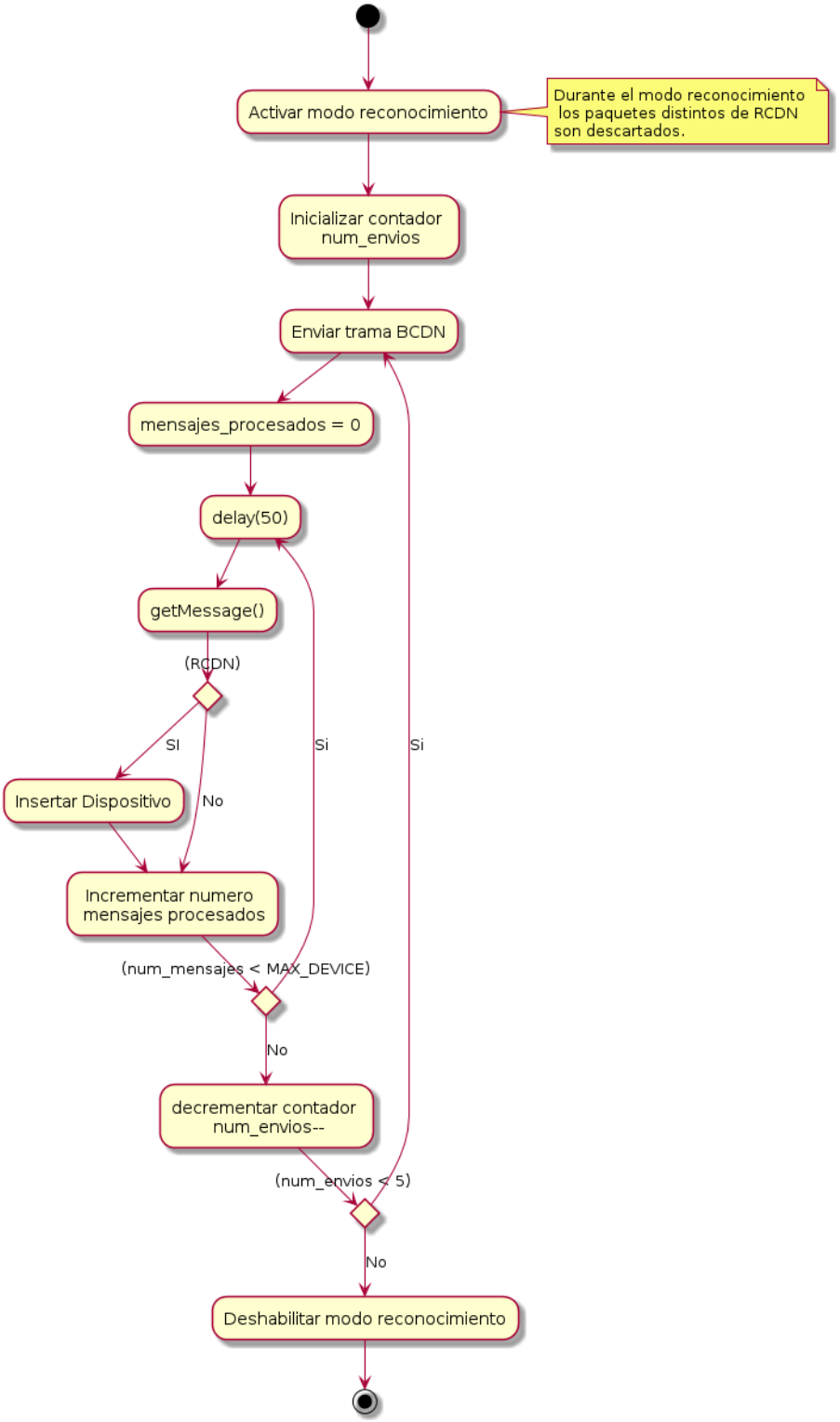


Figura 9.28: Diagrama de actividad. Actualización de la tabla de dispositivo de un nodo Servidor.

9.5.2. Nodo Master - Actualizar lista dispositivo

9.5.2.1. Descripción

Un nodo Maestro debe tener la tabla de dispositivos sincronizada con el nodo Servidor en todo momento. Sin embargo, existen diversos escenarios que pueden ocasionar diferencias entre ambas tablas.

- **Reset:** El reinicio de un nodo Maestro libera las entradas en la tabla de dispositivo.
- **Conexión Supervisor:** La conexión de un Supervisor a un nodo Maestro genera el reinicio de este último.
- **Pérdida de paquetes:** La pérdida de un paquete de confirmación de asignación de ID, BCID.
- **Inicio:** Cuando un nodo Maestro inicia con posterioridad a otros dispositivos.

La revisión v2.0 de TouCAN incorpora un mecanismo que permite a los nodos Maestros solicitar una copia de la tabla de dispositivos a un Supervisor. Esta operación se realiza a través del método *TouCANMaster::getDeviceList* y garantiza la sincronización de la tabla de dispositivos.

En la siguiente figura 9.29 el nodo Maestro, MAS01, solicita la lista de dispositivos al nodo Servidor, después de un reset en una red compuesta por cuatro dispositivos.

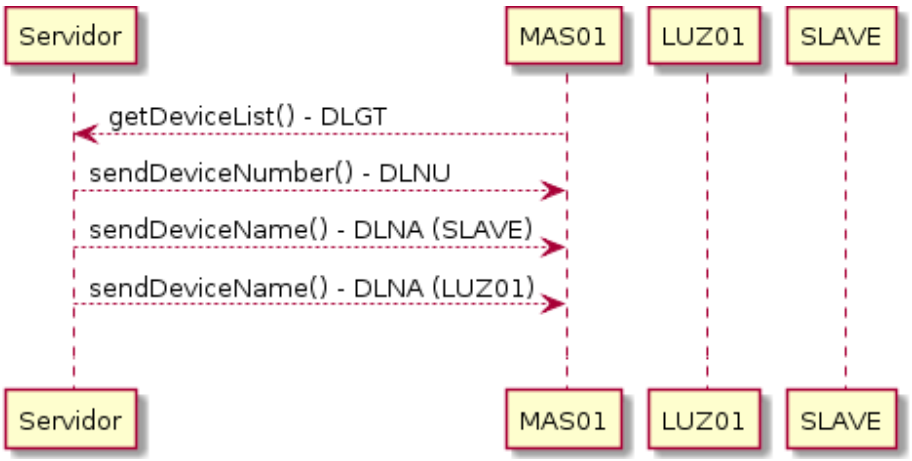


Figura 9.29: TouCAN v2.0 - Diagrama de secuencia - Actualización lista de dispositivos

9.5.2.2. Tramas

A continuación se detallan las tramas que intervienen en el proceso de actualización de la tabla de dispositivo de un nodo Master.

- **DLGT** Trama utilizada por un nodo Master para solicitar la lista completa de dispositivos al nodo Server.
- **DLNU:** Devuelve el número de dispositivo en los dos primeros bytes del campo de datos. Enviado por el nodo Server al Master que realizó la petición: "Request devices list".
- **DLNA:** Este paquete es utilizado por el nodo Server para enviar al Master la lista de dispositivos. Envía tantos paquetes como dispositivos contenga la tabla. El campo de datos contiene el ID y el nombre del nodo.

9.5.2.3. Implementación

El proceso de actualización o sincronización consiste en solicitar la lista de dispositivos al nodo Servidor a través del comando DLGT. Este envıa en primer lugar una trama con el número de dispositivos almacenados y a continuacion envia tantos paquetes DLNA como entradas existan en la tabla de dispositivos. El diagrama 9.34 describe el proceso de actualización al completo.

A continuación se describe el proceso de sincronización de la tabla de dispositivos en una red compuesta por cuatro dispositivos:

1. **SERVER:** Nodo Servidor con identificador 0.
2. **MAS01:** Nodo Maestro con identificador 1.
3. **LUZ01:** Nodo común con identificador 2.
4. **LUZ02:** Nodo Esclavo con identificador 3.

La imagen 9.30 corresponde a una petición *DLNA* del nodo **MAS01**. Después de solicitar la lista de dispositivo el nodo queda a la espera de una respuesta por parte del servidor. Durante este proceso, cualquier paquete distinto a *DLNU* o *DLNA* son descartados.

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
1	DLGT (10101)	0	0								

Figura 9.30: TouCAN v2.0 - Envio trama DLGT

El Server recibe la petición y envía un primer paquete con el número de dispositivos en la red, comando *DLNU*. El número de dispositivos no contempla al servidor y al maestro que realizó la petición. Este escenario está compuesto por 4 dispositivos. Sin embargo, el número de nodos enviados corresponde a 2.(**LUZ01** y **LUZ02**) Imagen 9.31

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	DLNU (10110)	1	2	0	2						

Figura 9.31: TouCAN v2.0 - Envio trama DLNU

A continuación el Servidor envia dos paquetes *DLNA* correspondiente a los nodos: **LUZ01** y **LUZ02** que serán procesados por **MAS01** para actualizar la lista de dispositivos. Imágenes 9.32 y 9.33

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	DLNA (10111)	1	7	L	U	Z	0	1	2		

Figura 9.32: TouCAN v2.0 - Envio trama DLNA

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	DLNA (10111)	1	7	L	U	Z	0	2	3		

Figura 9.33: TouCAN v2.0 - Envio trama DLNA

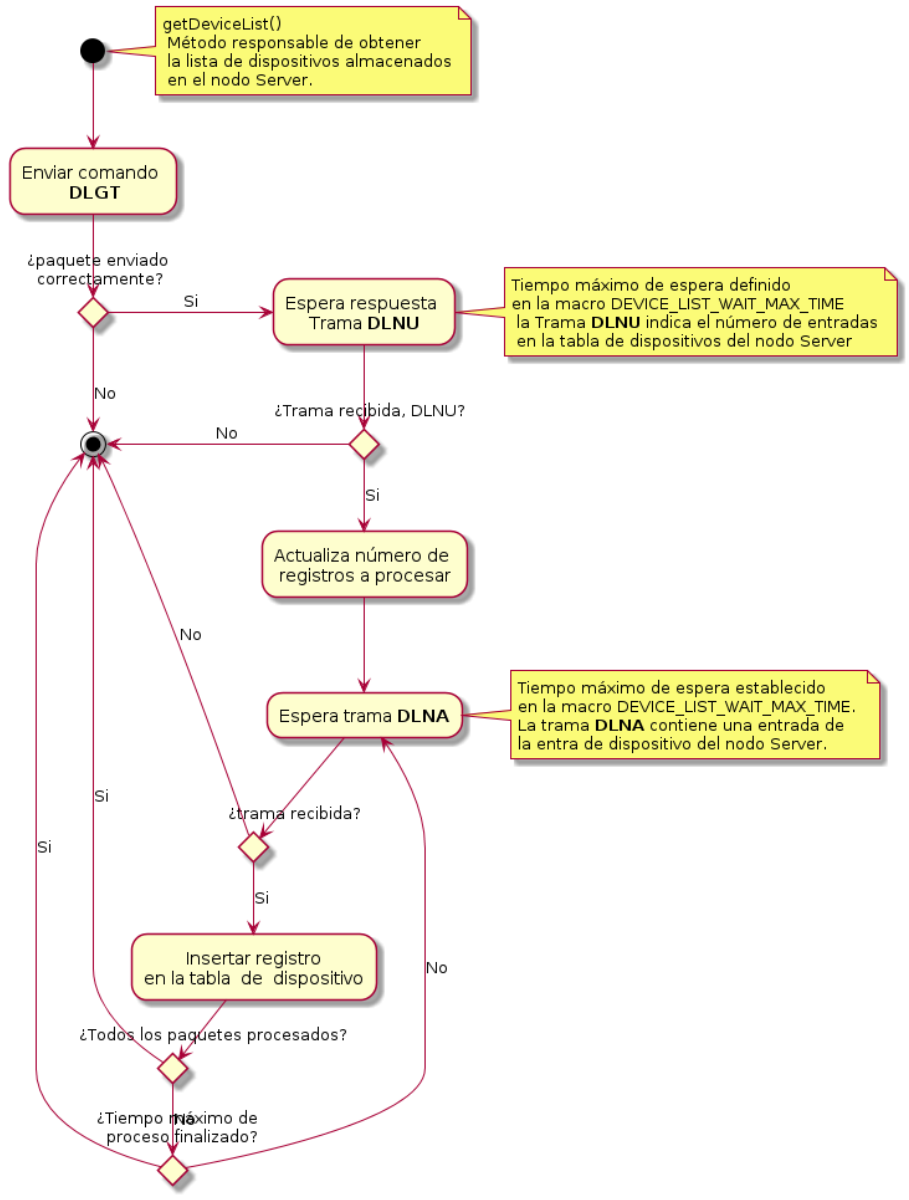


Figura 9.34: Diagrama de actividad. Actualización de la lista de dispositivos de un nodo Master.

9.5.3. Nodo Server - Verificar estado dispositivos (ECHO)

9.5.3.1. Descripción

La versión inicial de la librería carece de un mecanismo que permita verificar el estado de los nodos de la red con cierta periodicidad. Esta situación puede afectar de forma negativa al rendimiento del sistema cuando un nodo causa baja y otros continuan enviandole paquetes. La revisión TouCAN v2.0 incorpora un mecanismo, denominado *ECHO*, que verifica el estado de los nodos con cierta periodicidad para determinar cuales continuan activos y cuales no. Los nodos que han causado baja son notificados a los Maestros para que actualicen sus tabla de dispositivos.

La siguiente imagen 9.35 describe la secuencia de tramas durante el proceso de vierificación de estado de dispositivos, en una red conformada por cuatro dispositivos: un nodo servidor, dos maestros MAS01 y LUZ01 y un nodo esclavo, ESLAVE. Donde el nodo maestro, LUZ01, ha dejado de estar operativo.

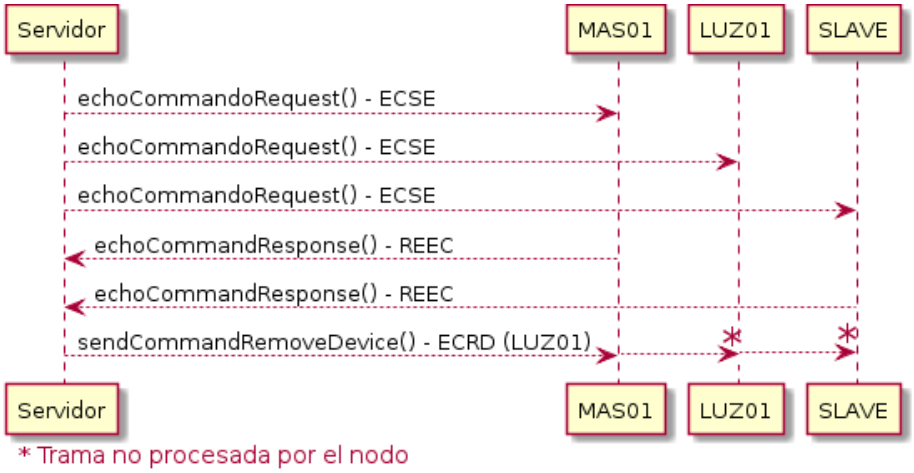


Figura 9.35: TouCAN v2.0 - Diagrama de secuencia comando ECHO

9.5.3.2. Tramas

En esta sección se detallan las tramas que intervienen en el proceso de verificación de dispositivos en la red, *ECHO*.

- **ECSE:** Trama enviada por el nodo Servidor para verificar si algún dispositivo a causado baja en el sistema. El comando es enviado a todos los nodos con los cuales no ha mantenido una comunicación en cierto intervalo de tiempo.
- **REEC:** Respuesta emitida por los nodos Maestros y Esclavos a una petición *ECSE*.
- **ECRD:** Paquete broadcast utilizado para informar que un dispositivo ha causado baja en la red. Enviado por el nodo Servidor y procesado únicamente por los nodos Maestros. Los dos primeros bytes del campo de datos almacenan el ID del nodo a eliminar. Este paquete permite mantener actualizada la lista de dispositivos de todos los nodos Master de la red.

9.5.3.3. Implementación

La característica de verificación de dispositivo es exclusiva de los nodos Servidores y consiste en enviar un paquete *ECSE* a cada uno de los nodos del sistema con una periodicidad establecida por la macro ECHO_TIME_INTERVAL_TO_SEND (5minutos). Después de un tiempo de espera establecido por la macro ECHO_TIME_INTERVAL_TO_UPDATE (2minutos) el Server comprueba que dispositivos han contestado a la solicitud. Los dispositivos que no han respondido se consideran que han causado baja en el sistema y se notifica a todos los nodos Master de la red a través de la trama broadcast *ECRD*. Este comando almacena en el campo de datos el ID del dispositivo que ha causado baja. El diagrama 9.40 describe el proceso completo.

A continuación se describe el funcionamiento del sistema de verificación de estados de dispositivos en un sistema compuesto por un Servidor, dos Maestros y un Esclavo. Donde en el instante de tiempo que el nodo Servidor inicia el comando *ECSE* el dispositivo *SLAVE* no está operativo.

- **SERVER**: Nodo Servidor con identificador 0.
- **MAS01**: Nodo Maestro con identificador 1.
- **SLAVE**: Nodo Esclavo con identificador 2.
- **MAS02**: Nodo Maestro con identificador 3.

El Servidor inicia el proceso de reconocimiento enviando un paquete *ECSE* al nodo **SLAVE** y **MAC02**. Dispositivos con los que no ha mantenido una comunicación en los últimos 5 minutos. Figura 9.36 y 9.37

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	ECSE (11000)	2	0								

Figura 9.36: TouCAN v2.0 - Trama ECSE

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	ECSE (11000)	3	0								

Figura 9.37: TouCAN v2.0 - Trama ECSE

MAS02 responde al nodo Servidor enviando el paquete *REEC*. Imagen: 9.38

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
3	REEC (11001)	0	0								

Figura 9.38: TouCAN v2.0 - Trama REEC

Después de un intervalo de tiempo establecido por la macro: ECHO_TIME_INTERVAL_TO_UPDATE, el Servidor verifica que dispositivos han respondido al comando *ECSE*. En este escenario únicamente recibe respuesta del nodo **MAS02** considerando que el nodo **SLAVE** a cusado baja en el sistema. En ese mismo instante envía una trama *ECRD* que será procesada por los nodos **MAS01** y **MAS02**. Imagen 9.39

				Data							
Source ID	Frame Type	Destin. ID	Length	0	1	2	3	4	5	6	7
0	ECRD (10100)	BROADCAST (11111111)	0	2							

Figura 9.39: TouCAN v2.0 - Trama ECRD

Los nodos **MAS01** y **MAS02** procesan el comando *ECRD* eliminando el nodo **SLAVE** de la tabla de dispositivos.

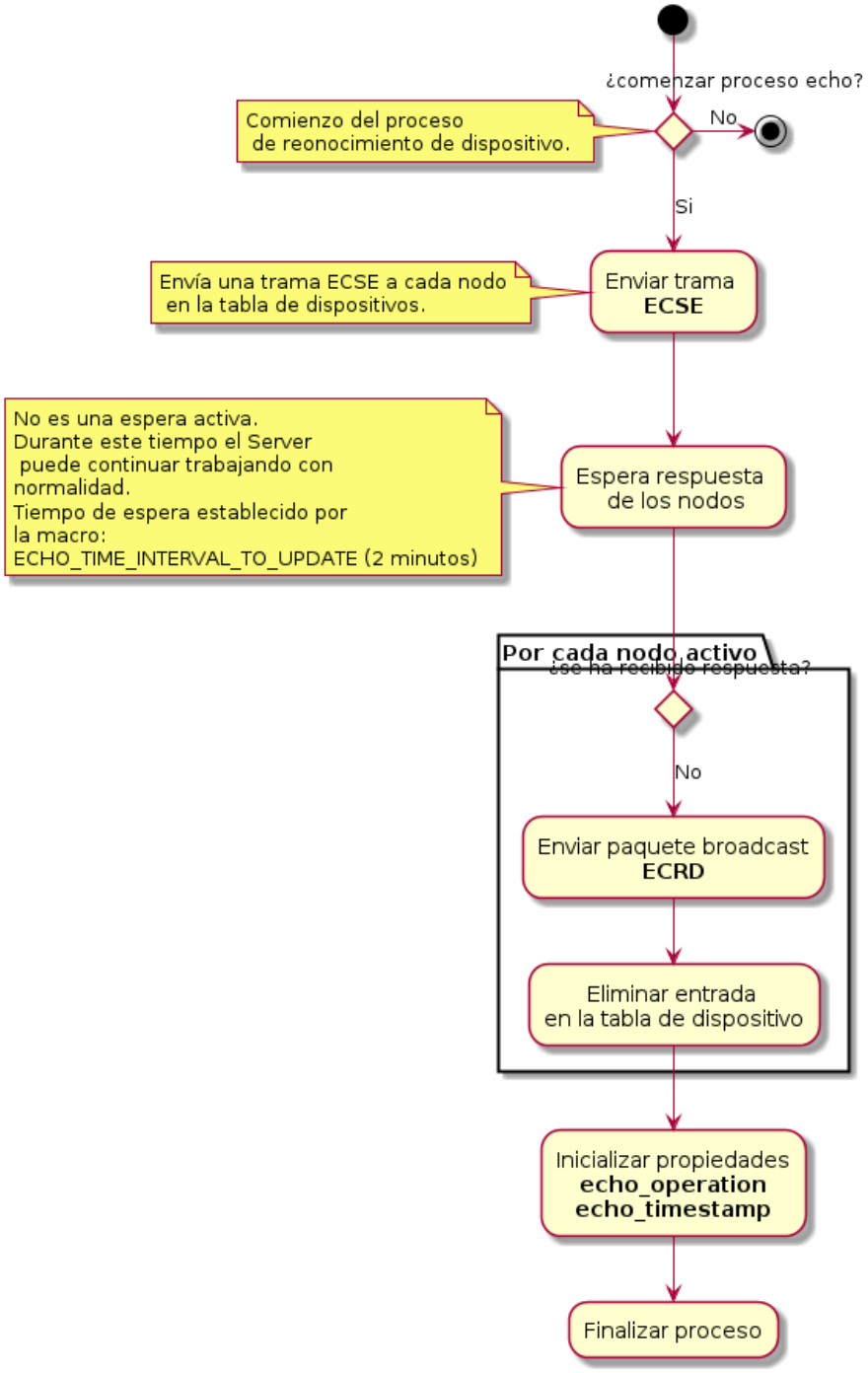


Figura 9.40: Diagrama de actividad. Proceso de reconocimiento de dispositivos en red (echo).

9.6. TouCANSniffer

TouCANSniffer es una herramienta integrada en el framework TouCAN v2.0 que permite capturar, monitorizar y visualizar todo el tráfico generado en un sistema. Está compuesta por tres módulos totalmente independientes, pero en su conjunto proporcionan una herramienta que facilita el análisis del funcionamiento de la librería y la integración de nuevas características.

9.6.1. TouCAN Node Sniffer

NodeSniffer es un nuevo elemento incorporado en la revisión v2.0 de la librería que realiza únicamente dos funciones bien definidas.

- **Capturar tramas:** Capturar todos los paquetes generados en la red.
- **Conexión SupervisorSniffer:** Enviar, al SupervisorSniffer, todos los paquetes capturados en la red a través del puerto serie.

A diferencia de los nodos Servidor, Maestro y Esclavo, NodeSniffer no requiere de un identificador para su correcto funcionamiento y solo puede capturar mensajes teniendo deshabilitado el envío de paquetes.

9.6.1.1. Diagrama de Clases

La figura 9.41 contiene el diagrama de clase correspondiente al nodo Sniffer.

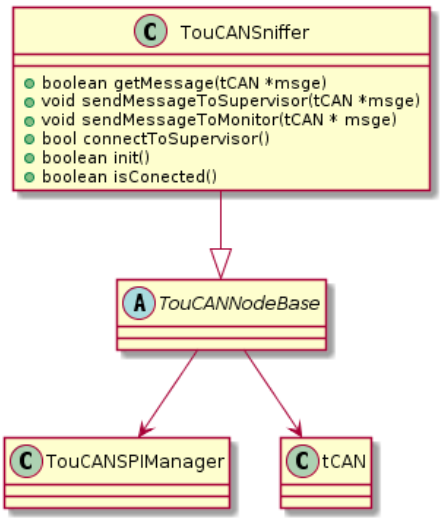


Figura 9.41: TouCAN v2.0 - Diagrama de clases NodeSniffer

A continuación se describen los principales métodos de la clase:

- **getMessage:** Método responsable de capturar un mensaje de la red.
- **sendMessageToSupervisor:** Envía un mensaje al SupervisorSniffer a través del puerto serie.
- **sendMessageToMonitor:** Envía un mensaje a la ventana monitor serial.
- **connectToSupervisor:** Establece una comunicación con el SupervisorSniffer.
- **init:** Inicializa el dispositivo. Este método es el responsable de establecer el filtro y la máscara correcta para la captura de todos los paquetes.
- **isConected:** Verifica la conexión con el SupervisorSniffer.

9.6.1.2. Trama

La comunicación entre el SupervisorSniffer y NodeSniffer se realiza a través de la interfaz serie mediante el envío secuencial de una serie de bytes que deben ser interpretados por ambas partes. La siguiente iamgen 9.42 representa la estructura de datos utilizada en la comunicación.

Figura 9.42: TouCAN v2.0 - Trama utilizada en la comunicación entre SupervisorSniffer y NodeSniffer

- **Time:** Marca de tiempo en milisegundos desde el inicio del nodo Sniffer.
- **Origin:** Identificador del nodo origen:
- **Destination:** Identificador del nodo destino.
- **Frame Type:** Tipo de Frame enviado
- **Length:** Longitud del campo Data.
- **Data:** 8 Bytes reservados al campo data.

9.6.2. SupervisorSniffer

SupervisorSniffer es una aplicación de escritorio implementada en lenguaje C++[11] que recopila todos los paquetes enviados por el nodo Sniffer a través del puerto serie y lo almacena en un archivo en disco, con el formato correspondiente a cada trama. Este archivo es importado por la aplicación, TouCANSnifferView, responsable de visualizar todo el tráfico a través de una interfaz gráfica. La imagen 9.43 contiene el diagrama de secuencia correspondiente a SupervisorSniffer.

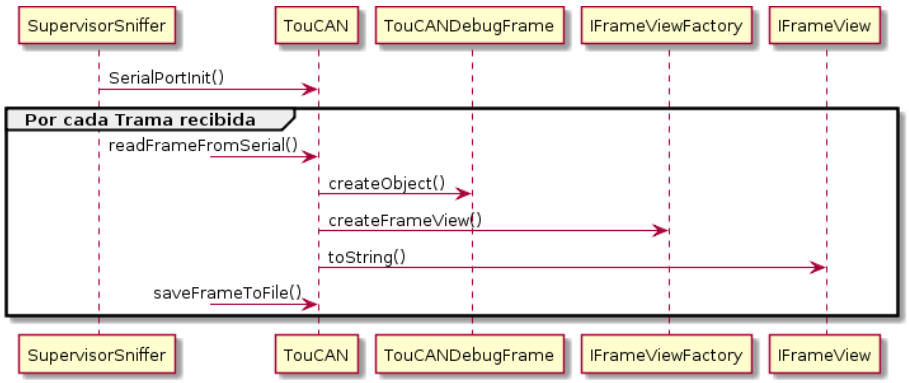


Figura 9.43: TouCAN v2.0 - Diagrama de secuencia SupervisorSniffer

9.6.2.1. Diagrama de Clases

SupervisorSniffer implementa el patrón Abstract Factory [1] en el proceso de generación de las tramas enviadas por el nodo NodeSniffer, facilitando el formato de visualización del campo *DATA* de cada una de las tramas. La figura 9.44 corresponde ak diagrama de clases del SupervisorSniffer.

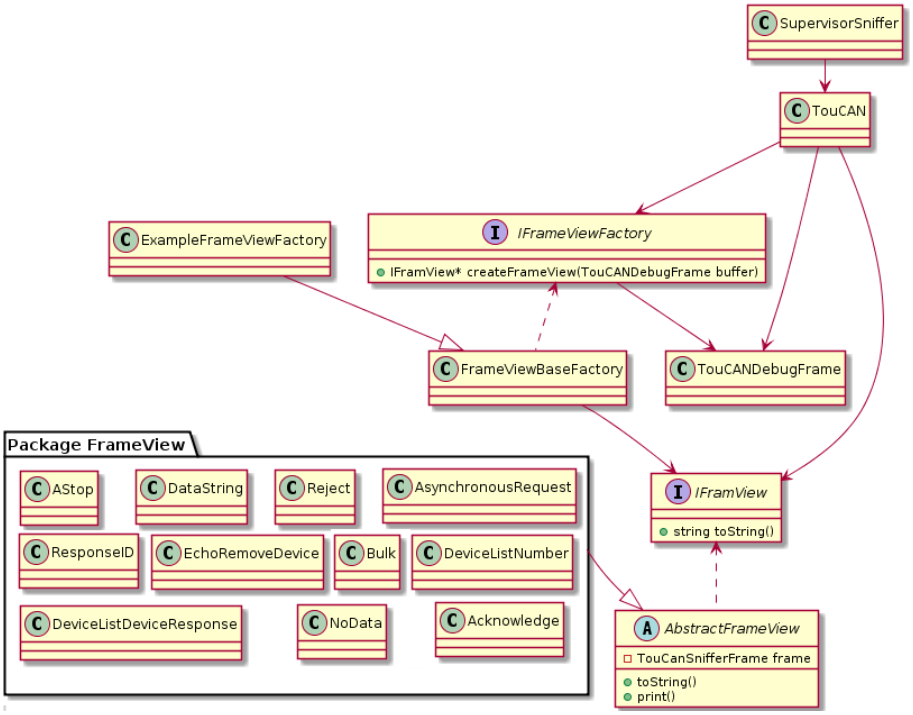


Figura 9.44: TouCAN v2.0 - Diagrama de clases SupervisorSniffer

- **SupervisorSniffer**: Aplicación de usuario responsable de establecer una conexión con el node NodeSniffer y almacenar en disco los paquetes recibidos.
- **TouCAN**: Esta clase proporciona una interfaz de comunicación con el puerto serie y procesa las tramas recibidas dando formato al campo *DATA*.
- **IFrameViewFactory**: Define la interfaz que debe implementar las clases Factory responsables de crear los objetos IFrameView.
- **TouCANDebugFrame**: Clase utilizada como buffer para almacenar la trama enviada por el nodo NodeSniffer.
- **FrameViewBaseFactory**: Clase abstracta que implementa las operaciones comunes de las distintas clases Factory.
- **IFrameView**: Define la interfaz que debe implementar una clase hija de AbstractFrameView para personalizar formato del campo DATA.
- **AbstractFrameView**: Clase abastracta, contiene las operaciones comunes de cada una de las distintas tramas.

- **Package FrameView:** Conjunto de clases que representan cada una de las diferentes tramas del protocolo. La principal responsabilidad de estas clases consiste en dar el formato adecuado al campo DATA antes de almacenar la información en disco.

9.6.3. TouCANSnifferView

TouCANSnifferView es una aplicación web, compatible con la mayoría de los navegadores HTML5, que permite visualizar las tramas generadas por el SupervisorSniffer en una interfaz gráfica con una experiencia de usuario similar a una aplicación de escritorio. A continuación se describen las tecnologias utilizadas en el desarrollo de la aplicación:

- **Symfony PHP[21]:** Es un framework diseñado para el desarrollo de aplicaciones web basado en el patrón Modelo Vista Controlador MVC que separa la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos.
- **Doctrine[6]**: Es un mapeador de objetos-relacional (ORM)[16] escrito en PHP que proporciona una capa de persistencia para objetos PHP. Permite convertir datos entre el sistema de tipos utilizado en un lenguage de programación orientado a objetos y el utilizado en una base de datos relacional. Las tablas de la base de datos pasan a ser clases y los registros objetos.
- **AngularJS[2]**: Es un framework MVC[14] de JavaScript opensource para el desarrollo web en el lado cliente creado por Google. Permite crar aplicaciones SPA, Single Page Applications. Aplicación web que se ejecuta en una única página, logrando una experiencia de usuario más cercana a una aplicación de escritorio.
- **BootStrap[3]**: Es un framework de software libre, desarrollado por Twitter, para el diseño y desarrollo de aplicaciones web. Proporciona un conjunto de plantillas y elementos de diseño basado en HTML.
- **MariaDB[13]**: Sistema de gestión de base de datos derivado de MysSQL[15] con licencia GPL.

La figura 9.45 corresponde a una captura de pantalla de la aplicación TouCANSnifferView. En ella podemos encontrar dos zonas bien diferenciadas. En el lateral izquierdo se encuntra el menú de filtros que permite mostrar/ocultar los registros según el tipo de trama o dispositivo y en el área central se encuentran las tramas generadas por el nodo SupervisorSniffer. En este ejemplo, los registros en las posiciones tres, cuatro y cinco corresponden al tráfico entre los nodos *SERVER* y *SLAVE* durante el proceso de solicitud de ID de este último.

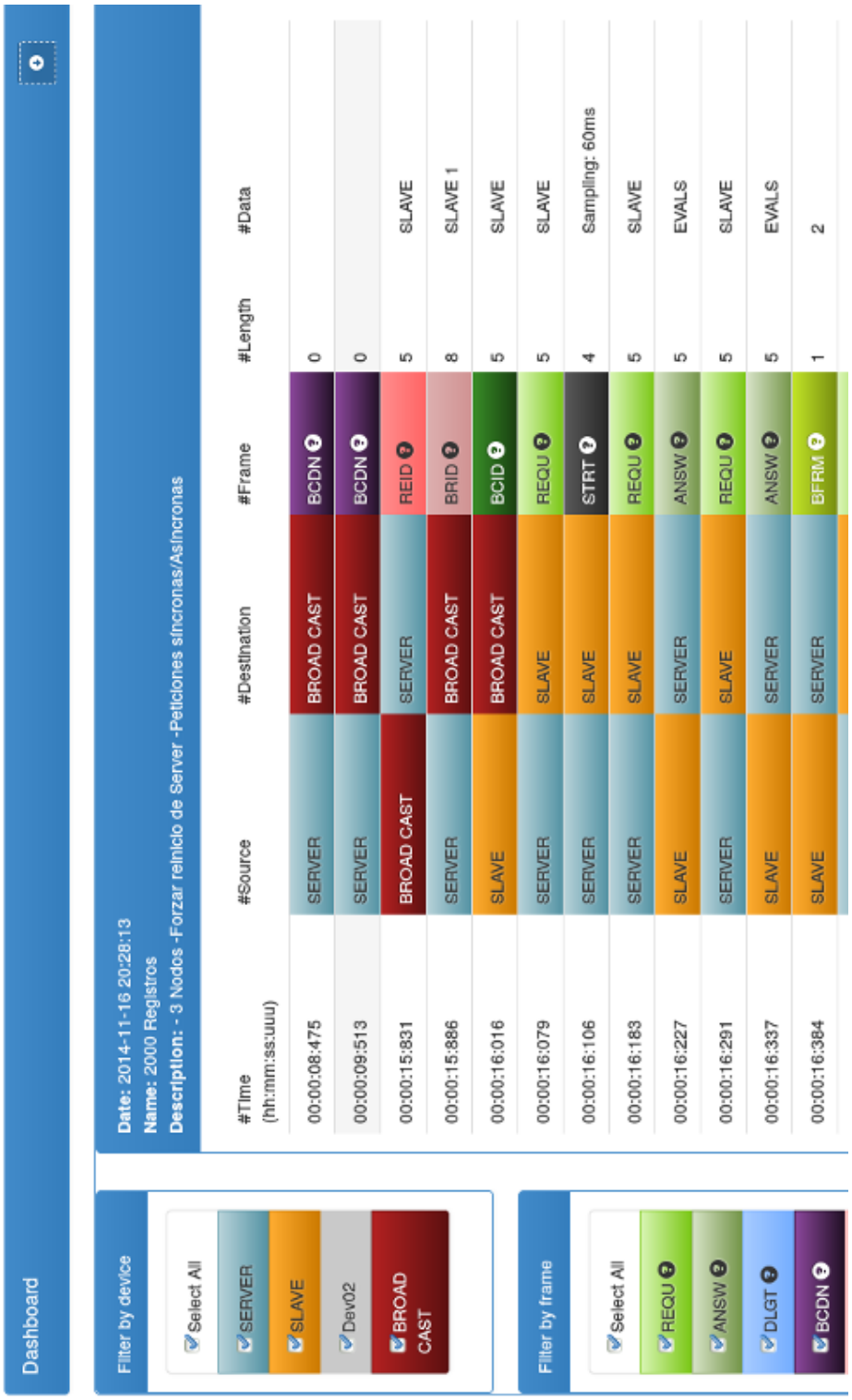


Figura 9.45: TouCAN v2.0 - Captura de pantalla TouCANSnifferView

9.6.3.1. Estructura base de datos

La base de datos está compuesta por cuatro tablas: importfilelog, register, device y frame. Almacenan los registros generados por SupervisorSniffer e información sobre las tramas del protocolo TouCAN v2.0 y los distintos dispositivos que conforman la red. La figura 9.46 muestra al diagrama Entidad Relación de la base de datos. A continuación se describe cada una de las tablas

- **importfilelog**: Esta tabla almacena información sobre cada una de las importaciones.
- **register**: Almacena las tramas generadas por SupervisorSniffer.
- **device**: Almacena información sobre los dispositivos que conforman la red. Los datos se obtienen del campo DATA de la trama BCID.
- **frame**: Contiene información sobre cada una de las tramas TouCAN v2.0.

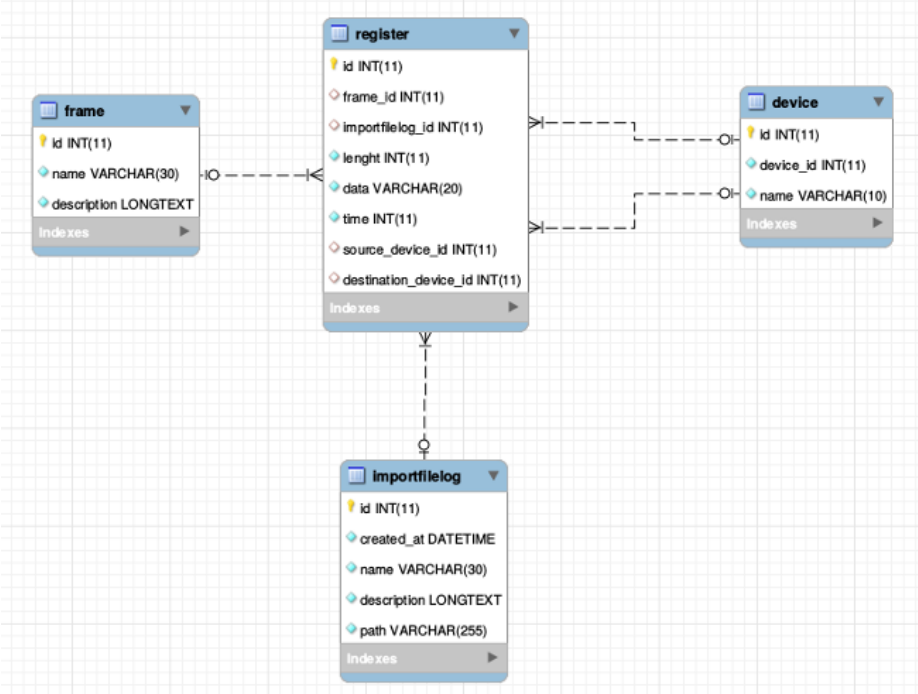


Figura 9.46: TouCAN v2.0 - Diagrama Entidad Relación TouCANSnifferView

9.6.3.2. Diagrama de clases

La figura 9.47 describe las clases principales de la aplicación TouCANSnifferView.

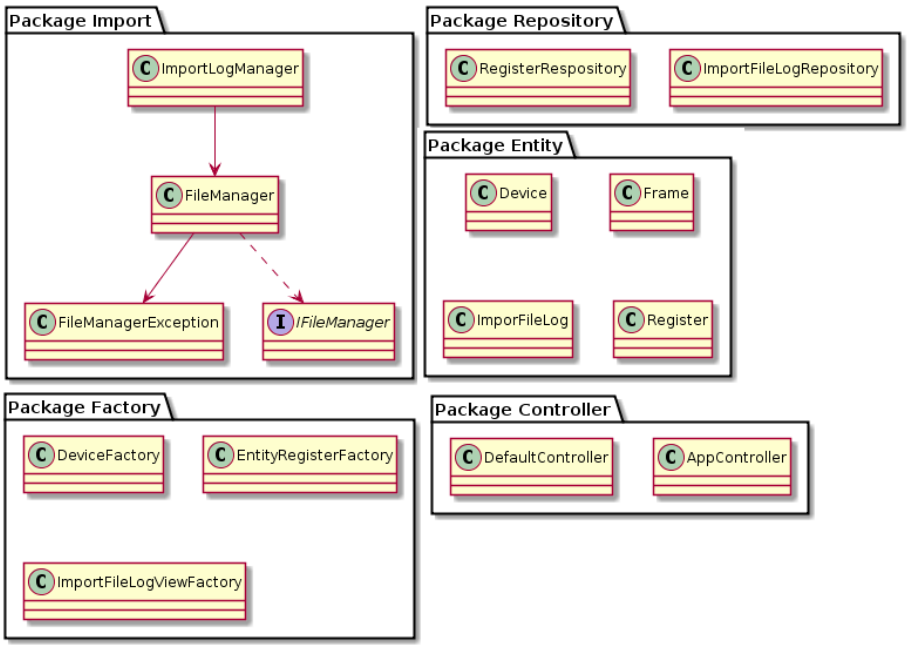


Figura 9.47: TouCAN v2.0 - Diagrama de clases - TouCANSnifferView

- **Package Entity:** Cada una de estas clases corresponde con una tabla en la base de datos y es utilizada por la librería Doctrine[6] para insertar, actualizar, eliminar o consultar un registro.
- **Package Repository:** Conjunto de clases que facilitan las consultas de registros en la base de datos.
- **Package Controller:** Los puntos de entrada o URL de acceso a la aplicación son definidos en estas clases.
- **Package Import:** Conjunto de clases responsable de importar el fichero de tramas generado por el SupervisorSniffer.
- **Package Factory:** Conjunto de clases que facilita la creación de objetos.

Capítulo 10

Pruebas

En este capítulo se describen diferentes escenarios donde se ponen a prueba las mejoras introducidas en este TFG y se verifican que los cambios realizados no afectan a las características ya existentes. Todos los escenarios están compuestos por una red de cuatro dispositivos: Servidor, Maestro, Esclavo y NodeSniffer. (Ver imagen: 10.1).

En la verificación de cada escenario se utilizará la herramienta TouCANSniffer-View descrita en el capítulo 9 Desarrollo. Cada captura de la aplicación TouCANS-nifferView mostrará los registros filtrados por dispositivos y/o tramas para verificar cada una de las distintas funcionalidades de forma independiente.

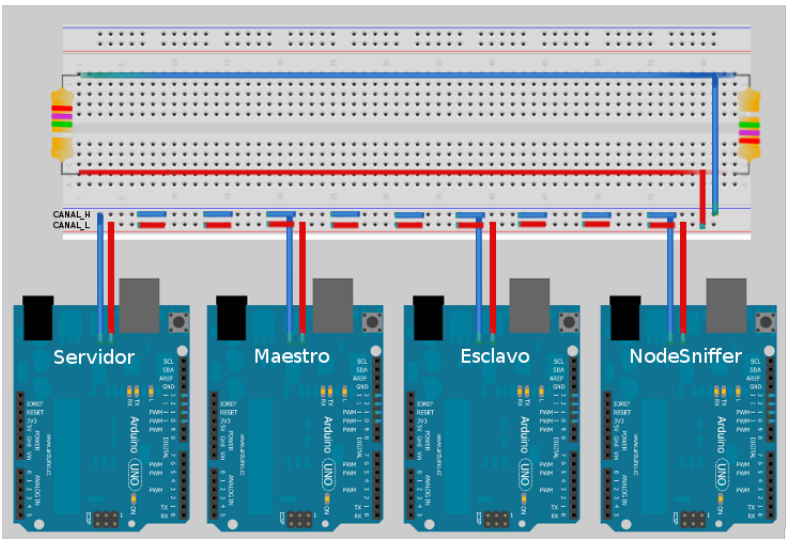


Figura 10.1: TouCAN v2.0 - Escenario Test - Sistema cuatro dispositivos

10.1. Escenario 1 - Asignación Dinámica ID

10.1.1. Descripción

Este escenario tiene por finalidad verificar la asignación dinámica de ID en una red compuesta por tres nodos de tipo: Server, Master y Slave. Es importante destacar el orden en el que debe iniciar cada dispositivo para verificar las distintas situaciones que se producen durante el proceso. A continuación se describe el orden en el que se iniciará los nodos y la función del programa principal.

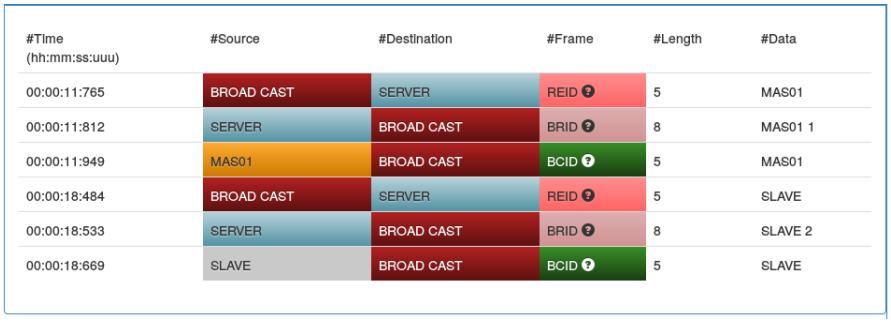
- **Server**: Atiende las peticiones de solicitud de ID. Y establece dos comunicaciones con cada uno de los nodos, una síncrona y otra asíncrona.
- **Master**: Realiza una solicitud de ID y atiende a peticiones síncroncas y asíncronas del nodo Server.
- **Slave**: Realiza una solicitud de ID y atiende a peticiones síncroncas y asíncronas del nodo Server.

10.1.2. Resultados

10.1.2.1. Asignación Dinámica ID

La siguiente imagen 10.2 muestra el tráfico correspondiente a la de solicitud de ID de los nodos MAS01 y SLAVE. En primer lugar, el Servidor atiende la petición del nodo maestro, asignándole el identificador 1. Instantes más tarde, asigna el ID 2 al nodo SLAVE. La columna, DATA, contiene la información enviada por cada nodo en el campo DATA en las distintas tramas. Por ejemplo, los tres últimos registros corresponden a la solicitud del nodo SLAVE.

1. En la solicitud de ID, el nodo almacena el nombre del dispositivo en el campo DATA. [SLAVE]
2. En la respuesta, el Servidor almacena el nombre del dispositivo y el identificador asignado. [SLAVE 2]
3. El tercer paquete corresponde a un paquete broadcast de confirmación de ID. El campo data contiene el nombre del dispositivo [SLAVE]

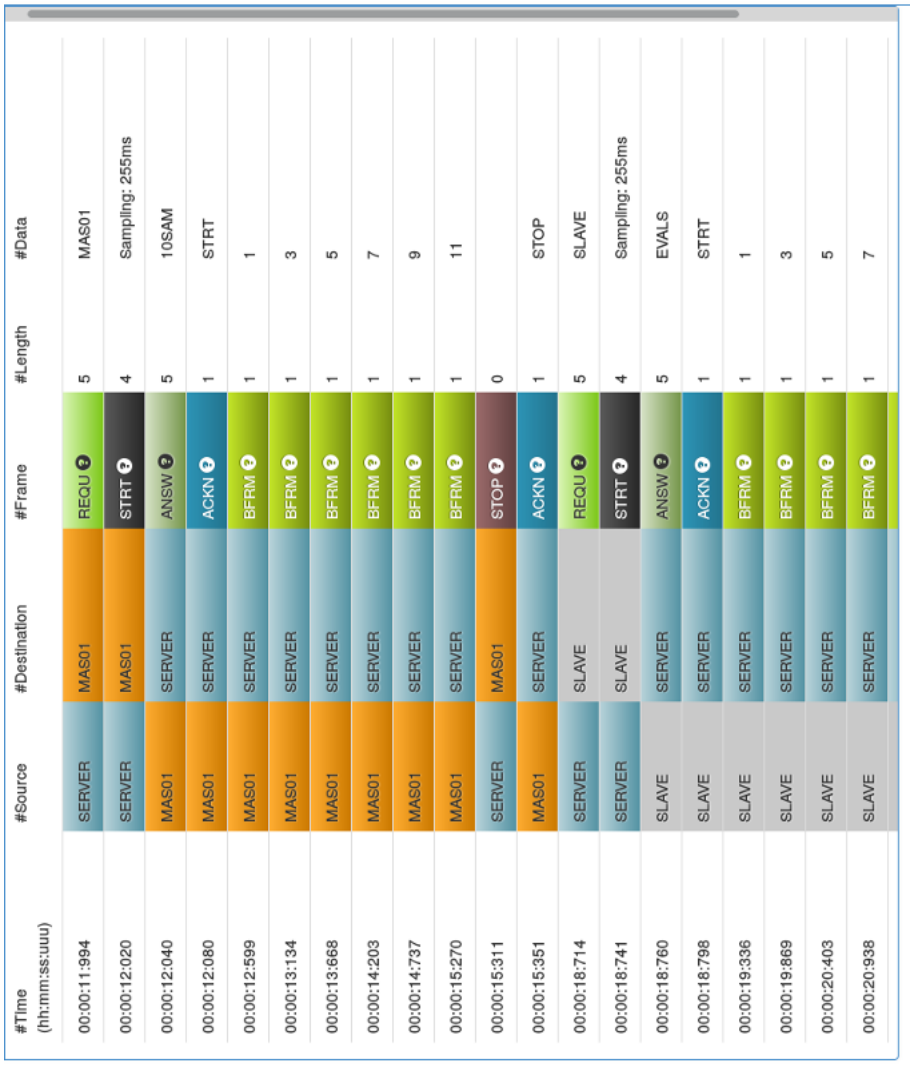


#Time (hh:mm:ss:uuu)	#Source	#Destination	#Frame	#Length	#Data
00:00:11:765	BROAD CAST	SERVER	REID	5	MAS01
00:00:11:812	SERVER	BROAD CAST	BRID	8	MAS01 1
00:00:11:949	MAS01	BROAD CAST	BCID	5	MAS01
00:00:18:484	BROAD CAST	SERVER	REID	5	SLAVE
00:00:18:533	SERVER	BROAD CAST	BRID	8	SLAVE 2
00:00:18:669	SLAVE	BROAD CAST	BCID	5	SLAVE

Figura 10.2: TouCAN v2.0 - TouCANSnifferView - Asignación dinámica ID

10.1.2.2. Petición Síncrona y Asíncrona

La figura 10.3 describe el tráfico generado en las peticiones síncronas y asíncronas entre el nodo Servidor y el resto de dispositivos. Los dos primeros registros corresponden al incio de una petición síncrona y asíncrona respectivamente entre los nodos Server y MAS01. En la imagen se aprecia las distintas tramas de una petición asíncrona y el orden de cada una de ellas. Por ejemplo, después del inico o finalización de una petición asíncrona siempre le sigue una trama de reconocimiento ACKN. (líneas 4 y 12)



#Time (hh:mm:ss:uuu)	#Source	#Destination	#Frame	#Length	#Data
00:00:11:994	SERVER	MAS01	REQU	5	MAS01
00:00:12:020	SERVER	MAS01	STRT	4	Sampling: 255ms
00:00:12:040	MAS01	SERVER	ANSW	5	10SAM
00:00:12:080	MAS01	SERVER	ACKN	1	STRT
00:00:12:599	MAS01	SERVER	BFRM	1	1
00:00:13:134	MAS01	SERVER	BFRM	1	3
00:00:13:668	MAS01	SERVER	BFRM	1	5
00:00:14:203	MAS01	SERVER	BFRM	1	7
00:00:14:737	MAS01	SERVER	BFRM	1	9
00:00:15:270	MAS01	SERVER	BFRM	1	11
00:00:15:311	SERVER	MAS01	STOP	0	
00:00:15:351	MAS01	SERVER	ACKN	1	STOP
00:00:18:714	SERVER	SLAVE	REQU	5	SLAVE
00:00:18:741	SERVER	SLAVE	STRT	4	Sampling: 255ms
00:00:18:760	SLAVE	SERVER	ANSW	5	EVALS
00:00:18:798	SLAVE	SERVER	ACKN	1	STRT
00:00:19:336	SLAVE	SERVER	BFRM	1	1
00:00:19:869	SLAVE	SERVER	BFRM	1	3
00:00:20:403	SLAVE	SERVER	BFRM	1	5
00:00:20:938	SLAVE	SERVER	BFRM	1	7

Figura 10.3: TouCAN v2.0 - TouCANSnifferView - Petición Síncrona y Asíncrona

10.2. Escenario 2 - Reinicio nodo Servidor

10.2.1. Descripción

Una de las nuevas características incorporadas a la librería consiste en recuperar la tabla de dispositivo después de un reset. Este escenario fuerza el reinicio de un Server y la verificación de la tabla de dispositivo después de reiniciar el dispositivo. A continuación se describe el programa principal de cada uno de los nodos.

- **Servidor**: Atiende las peticiones de solicitud de ID. No realiza peticiones síncronas o asíncronas.
- **Maestro**: Realiza una solicitud de ID.
- **Esclavo**: Realiza una solicitud de ID.

10.2.2. Resultados

La siguiente figura 10.4 muestra el intercambio de mensajes entre el Servidor y el resto de nodos en la red con el objetivo de recuperar las entradas de la tabla de dispositivos antes del reinicio. El servidor envía dos paquetes de reconocimiento de dispositivos, BCDN, con un intervalo de tiempo entre cada uno. En la captura se aprecia como los nodos SLAVE y MAS01 responden enviando la trama RCDN.

#Time (hh:mm:ss:uuu)	#Source	#Destination	#Frame	#Length	#Data
00:00:24:238	SERVER	BROAD CAST	BCDN	0	
00:00:24:302	SLAVE	SERVER	RCDN	5	SLAVE
00:00:25:283	SERVER	BROAD CAST	BCDN	0	
00:00:25:348	SLAVE	SERVER	RCDN	5	SLAVE
00:00:25:478	MAS01	SERVER	RCDN	5	MAS01

Figura 10.4: TouCAN v2.0 - TouCANSnifferView - Reinicio nodo Servidor

10.3. Escenario 3 - Master actualizar Tabla Dispositivos

10.3.1. Descripción

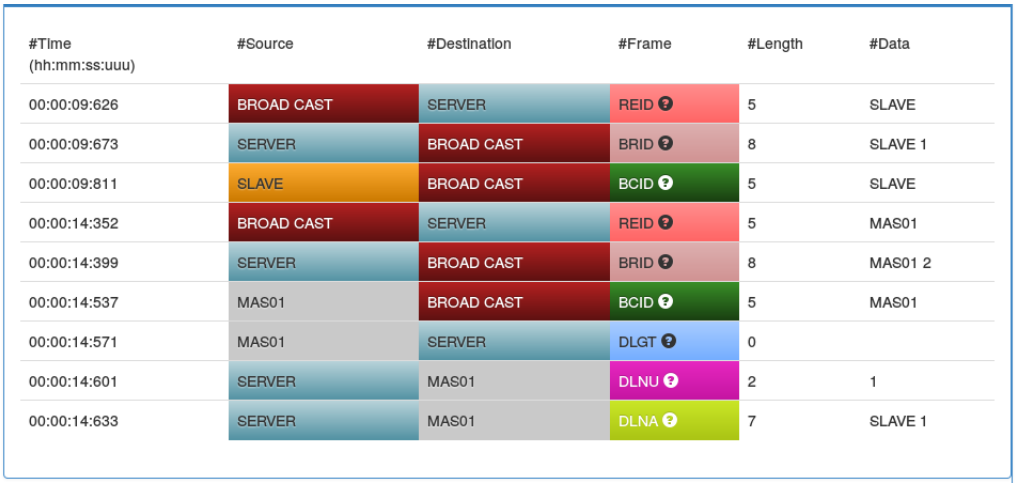
Los nodos Master actualizan la tabla de dispositivo a través de la trama *BCID* que reciben de los procesos de confirmación de ID que se generan en la red. Este mecanismo no asegura la actualización de la tabla en todo momento. La pérdida de un paquete, la sobrecarca del microcontrolador o simplemente iniciar con posterioridad a otros nodos puede causar que la tabla de dispositivo quede obsoleta. Esta revisión proporciona al nodo Master un mecanismo que le permite actualizar la lista de nodos con la del Server y mantenerse actualizado en todo momento.

En este escenario vamos a forzar que la tabla de dispositivo del nodo Master esté obsoleta simplemente iniciando el dispositivo con posterioridad al Slave. A continuación se describe las funcines del programa principal y el orden de inicio de cada uno de los nodos.

- **Server**: Atiende las peticiones de solicitud de ID. No realiza peticiones síncronas o asíncronas.
- **Slave**: Realiza una solicitud de ID.
- **Master**: Realiza una solicitud de ID.

10.3.2. Resultados

En la siguiente imagen 10.5 nos encontramos con los mensajes capturados en este escenario. A través de la marca de tiempo podemos verificar que el nodo MAS01 inició con posterioridad al nodo SLAVE. La primera operación que realiza el maestro después de iniciar es solicitar la lista de dispositivos al Servidor utilizando para ello la trama DLGT. Instantes después recibe dos paquetes del Server. El primero contiene el número de dispositivos en la red, DLNU, y el segundo contiene la información del dispositivo SLAVE.



#Time (hh:mm:ss:uuu)	#Source	#Destination	#Frame	#Length	#Data
00:00:09:626	BROAD CAST	SERVER	REID	5	SLAVE
00:00:09:673	SERVER	BROAD CAST	BRID	8	SLAVE 1
00:00:09:811	SLAVE	BROAD CAST	BCID	5	SLAVE
00:00:14:352	BROAD CAST	SERVER	REID	5	MAS01
00:00:14:399	SERVER	BROAD CAST	BRID	8	MAS01 2
00:00:14:537	MAS01	BROAD CAST	BCID	5	MAS01
00:00:14:571	MAS01	SERVER	DLGT	0	
00:00:14:601	SERVER	MAS01	DLNU	2	1
00:00:14:633	SERVER	MAS01	DLNA	7	SLAVE 1

Figura 10.5: TouCAN v2.0 - TouCANSnifferView - Maestro - Actualizar Lista de Dispositivos

10.4. Escenario 4 - Server - Monitorización

10.4.1. Descripción

Por distintos motivos, en un momento dado, un nodo puede causar baja en la red. Esta situación puede producir el envío masivo de mensajes a un nodo que no está operativo. TouCAN v2.0 incorpora un mecanismo, que verifica el estado de los nodos de la red con cierta periodicidad notificando a los nodos maestros la baja del dispositivo.

Para verificar el correcto funcionamiento del sistema de monitorización es necesario crear un escenario compuesto por tres nodos: Servidor, Maestro y Esclavo. Una vez estén configurados se desconectará el nodo SLAVE de la red iniciando el sistema de verifiación. A continuación se detallan las funciones del programa de usuario.

- **Server**: Atiende las peticiones de solicitud de ID. No realiza peticiones síncronas o asíncronas.
- **Slave**: Realiza una solicitud de ID.
- **Master**: Realiza una solicitud de ID.

10.4.2. Resultado

En la captura 10.6 el nodo Servidor inicia el proceso de verificación de estados a los 25 segundos de iniciar el sistema. La primera operación que realiza consiste en enviar la trama ECSE a los nodos MAS01 y SLAVE. Después de un tiempo de espera, solo recibe confirmación del nodo maestro, dando de baja de forma inmediata al dispositivo SLAVE de la tabla de dispositivo. Seguidamente envía un paquete ECRD para notificar a los nodos maestros la baja del dispositvo.

#Time (hh:mm:ss:uuu)	#Source	#Destination	#Frame	#Length	#Data
00:00:09:706	BROAD CAST	SERVER	REID	5	SLAVE
00:00:09:753	SERVER	BROAD CAST	BRID	8	SLAVE 1
00:00:09:891	SLAVE	BROAD CAST	BCID	5	SLAVE
00:00:15:167	BROAD CAST	SERVER	REID	5	MAS01
00:00:15:216	SERVER	BROAD CAST	BRID	8	MAS01 2
00:00:15:352	MAS01	BROAD CAST	BCID	5	MAS01
00:00:25:577	SERVER	MAS01	ECSE	0	
00:00:25:592	SERVER	SLAVE	ECSE	0	
00:00:25:618	MAS01	SERVER	REEC	0	
00:00:45:381	SERVER	BROAD CAST	ECRD	5	SLAVE
00:01:07:651	SERVER	MAS01	ECSE	0	
00:01:07:690	MAS01	SERVER	REEC	0	

Figura 10.6: TouCAN v2.0 - TouCANSnifferView - Servidor - Monitorización comando ECHO

10.5. Escenario 5 - Supervisor

10.5.1. Descripción

Este escenario está compuesto por un Supervisor y los nodos: Servidor y Slave. El Supervisor establece una comunicación asíncrona con el nodo Slave a través del Server. A continuación se detalla las funciones del programa de usuario.

- **Master**: Establece una comunicación asíncrona con el nodo Slave a través del Server.
- **Server**: Atiende las peticiones de solicitud de ID. No realiza peticiones síncronas o asíncronas.
- **Slave**: Realiza una solicitud de ID. Atiende peticiones Asíncronas

10.5.2. Resultado

Como se aprecia en la siguiente imagen 10.7. El nodo supervisor obtiene la lista de dispositivos a través del Server y establece una comunicación asíncrona con el dispositivo Slave.

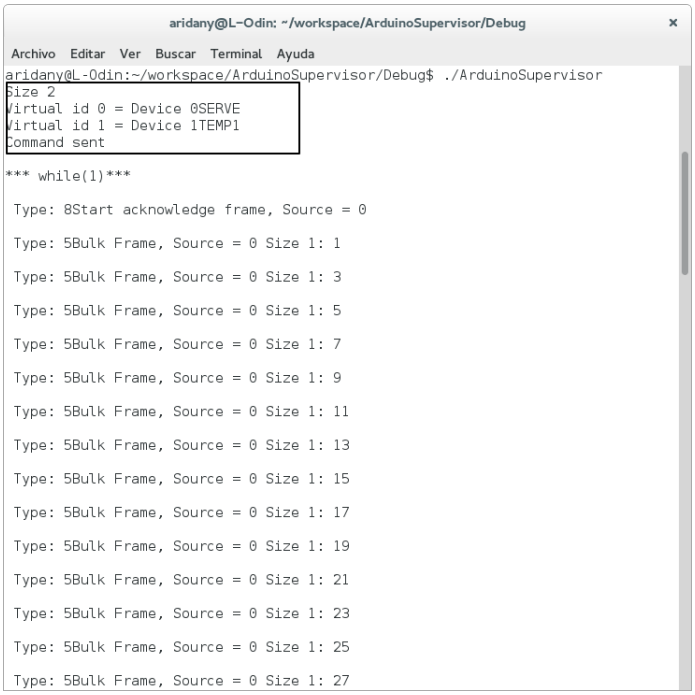


Figura 10.7: Monitor Serial - Supervisor - Petición asíncrona a un nodo de la red.

Capítulo 11

Conclusiones

11.1. Conclusiones

El objetivo de este Trabajo de Fin de Grado en Ingeniería Informática consistió en la implementación de mejoras y nuevas funcionalidades en la librería de comunicación de microcontroladores Arudiono, TouCAN.

Desde el punto de vista del objetivo inicial los resultados obtenidos son satisfactorios. Se ha logrado implementar todas las mejoras incrementando el potencial de la librería.

Durante el desarrollo del trabajo se han extraido las siguientes conclusiones:

- El éxito de un proyecto depende en gran medida del tiempo invertido en las fases previas de investigación, planificación y diseño. De estas fases depende la calidad del producto final y su continuidad.
- Es esencial la selección, estudio y familiarización de herramientas de software que facilite el trabajo durante el desarrollo.
- Dado los escasos recursos de los Arduinos UNO es conveniente tener presente en todo momento la optimización del código.
- Licenciar el proyecto con una licencia libre permite la continuidad del mismo e incluso la creación de una comunidad alrededor de él.

Asi mismo, se consideran alcanzados de forma exitosa los objetivos académicos planteados:

- Algoritmos, programación y estructuras de datos.
- Diseño de Sisteams Basados en Microcontroladores.
- Sistemas Empotrados y de Timpo Real.
- Algoritmos y programación Paralela.
- Sistemas Operativos.

11.2. Trabajo Futuro

A pesar de considerar a TouCAN una herramienta con el suficiente potencial para ser utilizada en cualquier proyecto que integre una red de microcontroladores. Me gustaría proponer una serie de mejoras que proporcionaría a la librería mayor robustez.

- **Interrupciones MCP2515**: Incorporar el uso de las interrupciones en la recepción de mensajes, mejoraría el rendimiento de la librería y simplificaría el proceso de obtención y procesamiento de mensajes.
- **Escalabilidad**: TouCAN actualmente solo funciona sobre placas Arduino UNO rev3. Esta restricción puede impedir el uso de la librería en proyectos que requieran dispositivos con otras características. La adaptación de la librería a otros modelos de microcontrolador podría ser una característica interesante.
- **Comunicación Supervisor:** Actulamente, el nodo supervisor se conecta a un nodo Master o Server a través de un puerto USB. Con el uso cada vez más extendido de dispositivos móviles y las comunicaciones inalámbricas. Se podría implementar la comunicación entre ambos nodos a través de una conexión Wifi, Bluetooth o radio frecuencia.
- **Consola Supervisor**:La incorporación de herramientas podría facilitar su integración en proyectos y simplificar las tareas de monitorización o test. Una de estas herramientas podria consistir en una consola que, a través del supervisor, permitiera monitorizar el estado de los nodos de la red o enviar / recibir paquetes a un determinado dispositivo de forma sencilla.

Bibliografía

[1] Abstract Factory Pattern, artículo en Wikipedia. [http://wikipedia.com].

[2] AnjularJS. [https://angularjs.org/].

[3] Bootstrap Framework. [http://getbootstrap.com/].

[4] CAN BUS (Controller Area Network). [Online; accedida 05-Diciembre-2012][http://en.wikipedia.org/wiki/CAN_bus].

[5] Creative Commons, artículo en Wikipedia. [Online; accedida 25-Julio-2014]http://es.wikipedia.org/wiki/Creative_Commons.

[6] Doctrine Project. [http://www.doctrine-project.org/].

[7] Eclipse Public License, artículo en Wikipedia. [Online; accedida 25-Julio-2014][http://es.wikipedia.org/wiki/Eclipse_Public_License].

[8] Especificaciones del microchip MCP2515. [Online; accedida 20-Diciembre-2012][http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010406].

[9] GNU General Public License, artículo en Wikipedia. [Online; accedida 20-Julio-2014][http://es.wikipedia.org/wiki/GNU_General_Public_License].

[10] GNU Lesser General Public License, artículo en Wikipedia. [Online; accedida 20-Julio-2014][http://es.wikipedia.org/wiki/GNU_Lesser_General_Public_License.

[11] Lenguaje C++. [http://es.wikipedia.org/wiki/C%2B%2B].

[12] Ley Ogánica de Protección de Datos. [Online; accedida 01-Julio-2014]https://www.boe.es/buscar/act.php?id=BOE-A-1999-23750&tn=0&p=20110305&vd=#ti].

[13] MariaDB. [https://mariadb.org/].

[14] Model-View-Controller Pattern, artículo en Wikipedia. [http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller].

[15] MySQL. [http://www.mysql.com/].

[16] Object Relational Mapping, ORM, artículo en Wikipedia. [http://en.wikipedia.org/wiki/Object-relational_mapping].

[17] Open Source, artículo en Wikipedia. [Online; accedida 02-Septiembre-2012][http://en.wikipedia.org/wiki/Open_source].

[18] Plataforma Arduino. [http://arduino.cc].

[19] Principio de Responsabilidad Única. SRP. Artículo en Wikipedia. [Online; accedida 01-Marzo-2013][http://en.wikipedia.org/wiki/Single_responsibility_principle].

[20] Proceso Unificado, artículo en Wikipedia. [Online; accedida 20-Enero-2013][http://es.wikipedia.org/wiki/Proceso_Unificado].

[21] Symfony Framework. [http://symfony.com/].

[22] José Antonio Lareo Domíngez. Integración de redes de microcontroladores distribuidos basados en bus CAN. Lado supervisor. 2012. [http://hdl.handle.net/10553/10146].

[23] Raúl Milla Pérez. Página oficial del proyecto ArCAN. [http://www.arcan.es/].

[24] John Wu Wu. Integración de redes de microcontroladores distribuidos basados en bus CAN. Lado microcontrolador. 2012. [http://hdl.handle.net/10553/9860].