Universidad Zaragoza

Trabajo Fin de Grado

Implementación del protocolo DNP3 en una red de sistemas SCADA empotrados para la monitorización de variables y dispositivos.

Implementation of the DNP3 protocol in embedded SCADA systems for the monitorization of variables from sensors and devices.

Autor:
Óscar Clemente Pedrico

Director:
Fernando Tricas Lamana

Ponente:
José Luis Briz Velasco

Escuela de Ingenieros y Arquitectos
2016

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Óscar Clemente Pedrico,

con nº de DNI 72983004W en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado de Ingeniería Informatica, (Título del Trabajo) Implementación del protocolo DNP3 en una red de sistemas SCADA empotrados para la monitorización de variables y dispositivos.,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 21 de Septiembre de 2016

Fdo: Óscar Clemente Pedrico

Implementación del protocolo DNP3 en una red de sistemas SCADA empotrados para la monitorización de variables y dispositivos.

Resumen

Este Trabajo de Fin de Grado se enmarca dentro de mi trabajo en Pariver S.A. y está compuesto por tres partes diferenciadas.

Pariver recibió una oferta para un proyecto en El Salvador en la que se buscaba el uso del protocolo DNP3 en la comunicación entre sistemas SCADA. En este Trabajo de Fin de Grado se ha incluido también la creación del propio sistema SCADA y formar redes de comunicación entre ellos.

La primera tarea consistió en el estudio y análisis del protocolo DNP3 buscando toda la documentación posible del protocolo. Para subsanar las lagunas y dudas surgidas de la documentación he recurrido a simuladores del protocolo DNP3 y a un analizador de paquetes de red para realizar ingeniería inversa.

La segunda tarea trató la implementación del protocolo DNP3 en sendas aplicaciones cliente y servidor que leen los valores de un fichero, los traducen al formato del protocolo y los envían para que sean leídos por el servidor. El servidor analiza los datos y verifica que son aceptables, si no lo son responde al cliente comunicando los cambios necesarios.

La tercera y última tarea abarcó la configuración de las placas ARM GuruPlug sobre las que se desplegaron los clientes y servidores DNP3 y la utilización de sensores que comuniquen valores a los SCADA a través de placas Arduino mediante el protocolo Modbus.

El resultado es un sistema capaz de recolectar datos del entorno, enviárselos al servidor de forma encriptada, analizar los datos y corregir posibles problemas en la instalación en la que se ha implantado de forma autónoma.

Índice general

1. Introducción 7
   1.1. Objetivo y alcance del proyecto 7
   1.2. Contexto de desarrollo 7
   1.3. Métodos y técnicas 7
   1.4. Tecnologías empleadas 8
   1.5. Herramientas empleadas 8
2. Análisis 10
3. Diseño 10
   3.1. Arquitectura del sistema 10
4. Estudio del protocolo DNP3 12
   4.1. Características del protocolo DNP3 12
   4.2. Metodología del análisis 12
   4.3. Estructura del protocolo 12
      4.3.1. Objetos 14
      4.3.2. Capa de enlace 14
      4.3.3. Capa de transporte 15
      4.3.4. Capa de aplicación 16
5. Implementación del protocolo DNP3 18
   5.1. Librería de configuración 19
   5.2. Criptografía TLS 20
   5.3. Ejecutivo cíclico 20
   5.4. Script IFTTT 23
6. Sensores 24
   6.1. Arduino Due 24
   6.2. Caudalímetro 25
   6.3. Anemómetro 26
   6.4. Detector de movimiento 26
   6.5. Sensor de humedad y temperatura 26
   6.6. Sensor de ultrasonidos 27
7. Gestión del proyecto 28
   7.1. Planificación 28
   7.2. Tiempo dedicado 28
   7.3. Herramientas de gestión 28
8. Conclusiones 30
   8.1. Resultados 30

8.2. Lecciones aprendidas 30
8.3. Conclusión personal 30
8.4. Futuro del proyecto 31
Bibliografía 32

Anexos **33**
1. Anexo 1 34

Índice de figuras

1. Diagrama de componentes y conectores. 10
2. Diagrama de despliegue. 11
3. Composición de tramas DNP3. 13
4. Object header. 14
5. Link layer header. 14
6. Link layer control byte. 15
7. Transport layer header. 16
8. Application layer. 16
9. Application header. 16
10. Diagrama de clases de la implementación DNP3. 18
11. Establecimiento de una sesión TLS. 20
12. Ejecutivo cíclico Pago de Aylés. 23
13. Campos del protocolo Modbus. 24
14. Diagrama de Gantt del proyecto. 28
15. Tiempo dedicado por tarea. 29

Índice de cuadros

1. Objetos implementados 15
2. Tareas del ejecutivo cíclico en Pago de Aylés 23
3. Requisitos funcionales 34

1. Introducción

1.1. Objetivo y alcance del proyecto

El objetivo propuesto para este proyecto consiste en crear una red de SCADAs que monitorizan las variables dadas por sensores y dispositivos, en la que toda la información se comunica mediante el uso del protocolo DNP3 y criptografía TLS por parte de los clientes y servidores . Las variables llegan a los clientes DNP3, y éstos se las comunican a los servidores DNP3 que monitorizan cada variable. El servidor se encarga de verificar que las variables permanecen dentro de un rango definido mediante tecnología basada en IFTTT. En caso de que el valor se salga fuera del rango, el servidor actuará y usando un mensaje DNP3 se comunicará con el cliente adecuado y modificará su comportamiento para corregir los valores.

Los clientes y servidores DNP3 se despliegan sobre dispositivos ARM GuruPlug que funcionarán como cliente o servidor dependiendo de la necesidad. Las variables llegan a los clientes a través de unos sensores que captan la información, se comunican con una placa Arduino y ésta se los pasa al ARM GuruPlug.

1.2. Contexto de desarrollo

Pariver es una empresa fundada en 1985 en Zaragoza centrada en el desarrollo de servicios TIC(Tecnologías de la Información y la Comunicación) y consultoría de empresas e instituciones.

Obtuvo el Certificado de Calidad de AENOR y han desarrollado aplicaciones y webs para servicios públicos y para empresas internacionales. Destacan los proyectos desarrollados para el gobierno de Aragón, el ministerio de economía, DKV, BSH, Global Credit Solution y Enerland Group, además, Pariver está implantada de forma internacional en Brasil, México, Marruecos, Portugal y Alemania.

El TFG se enmarca en un proyecto entre Pariver y la empresa Del Sur de El Salvador, en la que se pidió establecer comunicación con una planta industrial usando el protocolo DNP3, el TFG se extendió y se añadió la también la tarea de crear y configurar los sistemas SCADA empotrados sobre los que se usaría el protocolo DNP3.

1.3. Métodos y técnicas

El proyecto se ha organizado mediante el empleo de metodologías ágiles para el desarrollo del software que permiten realizar un desarrollo incremental e iterativo.

Tras la realización de cada tarea se lleva a cabo una reunión entre el autor y el director del proyecto, se discuten problemas y posibles mejoras y se formaliza el trabajo para la siguiente tarea.

Se busca que cada tarea se base en lo realizado en la anterior para formar un desarrollo incremental y también mantener el periodo de realización de cada tarea en torno a dos semanas, tiempo suficiente para que una tarea sea interesante y consistente pero también suficientemente pequeño como para que la tarea no sea difícil de abordar y demasiado compleja.

Cada una de las tareas se encuentra dentro de los siguientes rangos:

- Análisis: Estudio de la tarea y búsqueda de requisitos.
- Diseño: Planificación de la arquitectura y la implementación.
- Implementación: Desarrollo de las funcionalidades y los requisitos planificados previamente.
- Pruebas: Evaluación del correcto funcionamiento del sistema.

1.4. Tecnologías empleadas

Las principales tecnologías de las que se ha hecho uso para la elaboración del proyecto se presentan a continuación.

- C: Concretamente C99. Es un lenguaje de programación de propósito general. Se ha utilizado para la implementación del protocolo DNP3.
- C++: En su versión 11 es usada para programar la lógica de los sensores conectados a la placa Arduino.
- BASH: Es un lenguaje compatible con shell para entornos Unix. Se utiliza para definir una lógica al estilo IFTTT (If This Then That) que permite definir rangos de valores correctos para los sensores y definir la acción que se ha de realizar ante valores impropios, es invocado por el código C y puede ser modificado en caliente.
- libconfig: Es una librería para C y C++ que permite la creación y lectura de ficheros de configuración que permiten parametrizar los valores de un sistema de forma rápida y segura.
- openssl: Es una librería para C orientada a criptografía, se encarga de gestionar la comunicación entre dos sockets de forma fiable y segura usando criptografía SSL/TLS. Se usa el último estándar, TLS 1.2 pero en caso de que uno de los dos dispositivos conectados no pueda utilizarla, se negocia el uso de la tecnología más segura disponible ambos dispositivos. Permite también crear certificados y usarlos para la comunicación

1.5. Herramientas empleadas

Para completar el proyecto se han utilizado las siguientes herramientas de trabajo:

- Buildroot: Es una herramienta que facilita la generación de entornos y compiladores para compilación cruzada [1]. En el caso de este TFG se ha necesitado generar un compilador cruzado para la CPU ARM9E del ARM GuruPlug.
- Wireshark: Es un analizador de paquetes de red que permite comprobar el protocolo de cada paquete, su identidad y verificar la correcta estructura de este.
- OpenDNP3: Simulador del protocolo DNP3 que permite generar tramas del protocolo, para en este caso comprobar que la implementación desarrollada cumple el estándar DNP3.
- VirtualBox: Es una herramienta desarrollada por Oracle que permite crear máquinas virtuales para arquitecturas x86.
- Arduino IDE: Entorno de programación para placas Arduino que permite escribir código en C++, compilar y subir binarios a la placa Arduino.
- Vim: Editor de texto basado en Vi. Se ha usado para el código C.

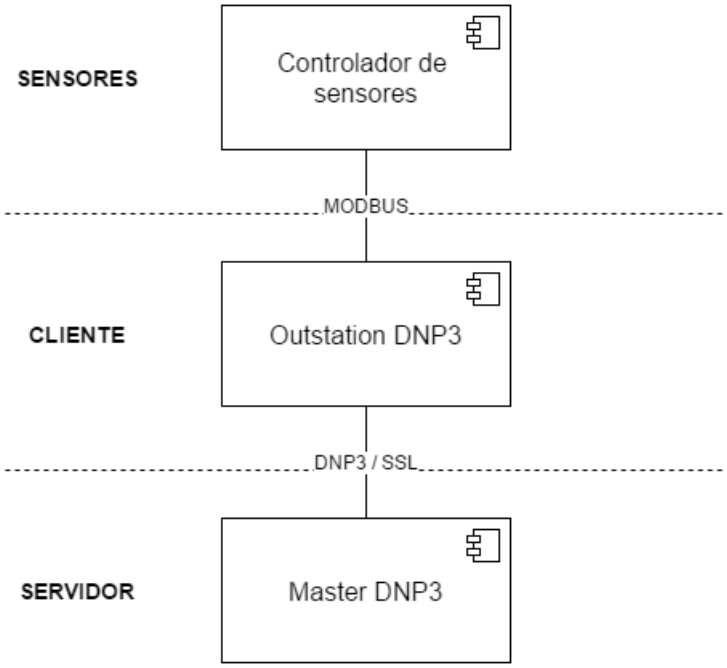


Figura 1: Diagrama de componentes y conectores.

2. Análisis

La fase de análisis determina el alcance del proyecto y las funcionalidades que va a presentar el sistema una vez completado. En el anexo se encuentra una tabla con los requisitos funcionales (Anexo 1).

3. Diseño

Conociendo las funcionalidades que va a cumplir el proyecto se procedió al diseño de alto nivel de la arquitectura y la organización del sistema.

3.1. Arquitectura del sistema

El sistema está compuesto por tres partes diferenciadas: los sensores, el cliente y el servidor (Figura 1). En la sección de los sensores se encuentra el controlador de los sensores que se encarga de recoger de forma periódica la información de cada uno de los sensores para tenerla actualizada y unificada en un solo lugar. El cliente lee los datos dados por el controlador de los sensores y los prepara para ser enviados encapsulándolos en el protocolo DNP3 y finalmente enviándolos al servidor. El servidor recibe mensajes DNP3 que ha de descomponer para poder leer los datos de los sensores, comprueba que los valores son

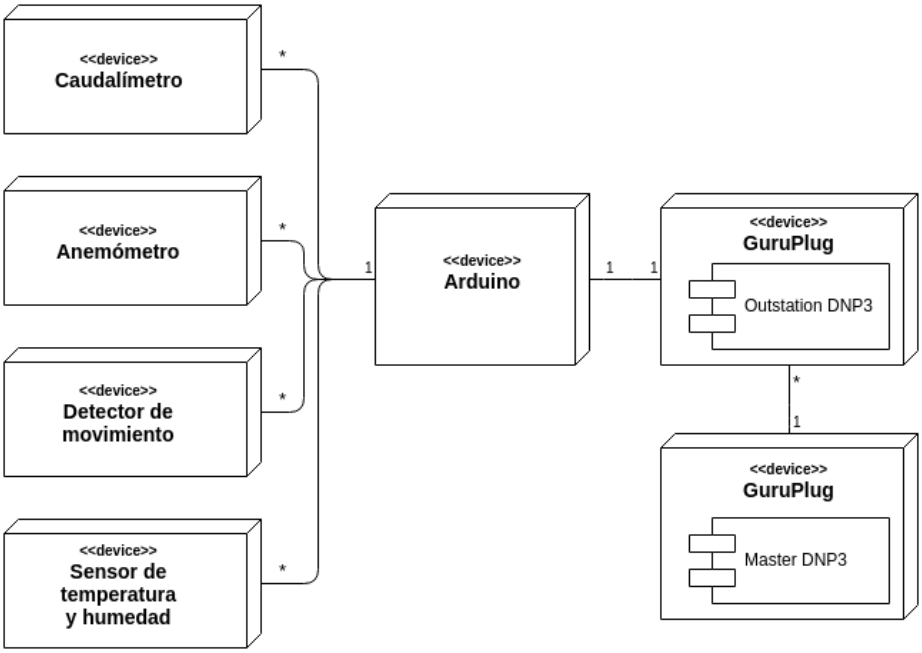


Figura 2: Diagrama de despliegue.

correctos y en caso de que no lo sean responde al cliente con una orden que permita corregir los valores.

El diagrama de despliegue muestra el despliegue físico de cada una de las partes que componen el proyecto incluyendo algunos de los sensores utilizados (Figura 2).

4. Estudio del protocolo DNP3

4.1. Características del protocolo DNP3

DNP3 (Distributed Network Protocol 3) es un protocolo industrial para comunicaciones entre equipos inteligentes, estaciones controladoras y componentes de sistemas SCADA. Es un protocolo ampliamente utilizado en el sector eléctrico y de tratamiento de líquidos, de gran difusión en toda América.

Los sistemas que usan este protocolo se diferencian normalmente en dos: el *máster* y los *outstation*.

- El *máster* es la máquina encargada de recopilar toda la información de las diferentes *outstations* y es usada por los equipos de gestión para visualizar la información, agruparla, analizarla o reenviarla a otro sistema.
- El *outstation* o RTU (Remote Terminal Unit) es la máquina encargada de coger los datos directamente del sistema eléctrico o de agua, transformar estos datos a datos DNP3 correctos, empaquetarlos en el mensaje y enviarlos al *máster*.

El protocolo DNP3 presenta importantes funcionalidades que lo hacen más robusto, eficiente y compatible que otros protocolos más antiguos pero también hacen que sea más complejo [2].

4.2. Metodología del análisis

DNP3 es propiedad de DNP Users Group. Aunque su uso es libre, la documentación oficial es de pago y no ha podido ser accedida para la realización de este TFG. La documentación pública disponible solo ofrece información superficial, sin detalles sobre la estructura del protocolo. Hemos suplido esta falta de información recurriendo a técnicas de ingeniería inversa.

Para ello se han utilizado el analizador de paquetes Wireshark, que es capaz de detectar los paquetes que se envían y descomponerlos para comprender en detalle los valores de cada campo, y el simulador gratuito OpenDNP3. Estableciendo una conexión entre un *máster* y un *outstation* creados con el simulador y usando Wireshark para observar los paquetes se obtiene una idea de cómo se estructura el protocolo y la utilidad de los campos [3].

Sin embargo, las tramas emitidas por el simulador son poco variadas y de tamaño pequeño, y no se utilizan todos los rangos de los campos sino solo unos pocos valores. Por esta razón, para comprender el protocolo completamente hizo falta crear un cliente DNP3 de prueba que se conectase al servidor del simulador y comenzar a iterar entre diferentes valores para cada campo y seguir usando ingeniería inversa con Wireshark para descubrir el significado de cada valor.

4.3. Estructura del protocolo

La especificación DNP3 divide el protocolo en tres capas según el modelo OSI: Nivel de enlace, nivel de aplicación y nivel de transporte. Realmente no

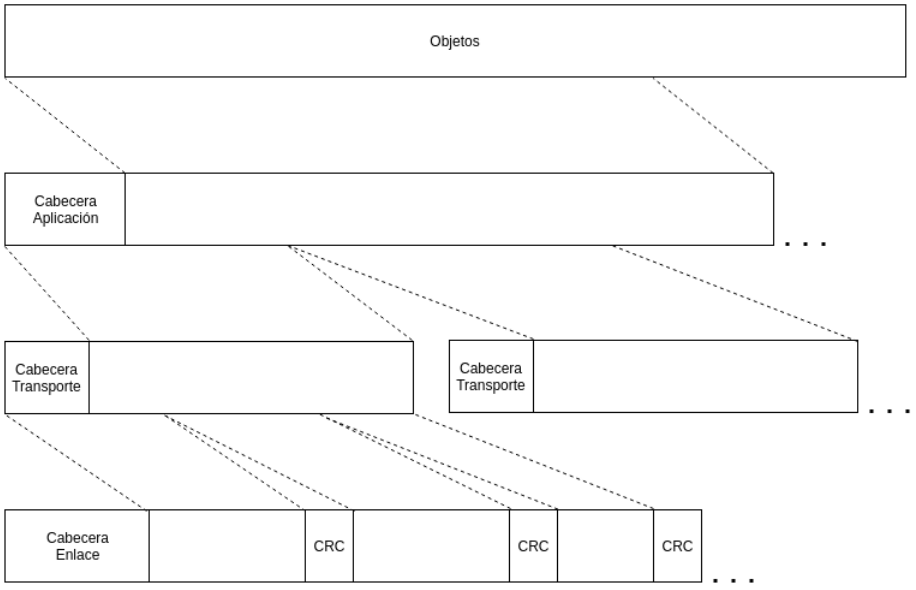


Figura 3: Composición de tramas DNP3.

cumple con todas las especificaciones del modelo OSI y a día de hoy se suele implementar sobre TCP/IP.

La estructura en capas sigue el siguiente esquema (Figura 3):

- Los mensajes a nivel de aplicaciones son denominados fragmentos. El tamaño máximo de un fragmento está establecido en 2014 bytes.
- Los mensajes a nivel de transporte son denominados *segmentos*. El tamaño máximo de un segmento es de 291 bytes.
- Los mensajes a nivel de enlace son denominados *tramas*.

Cuando se transmiten datos desde un *outstation* hacia un *máster*, estos pasan por varias fases antes de ser encapsulados completamente.

- Si el conjunto de datos es mayor que el tamaño máximo del nivel de aplicación. Es necesario dividir los datos.
- Añadir la cabecera del nivel de aplicación a todos los fragmentos.
- Si el fragmento es mayor que el tamaño máximo del nivel de transporte, es necesario dividir los fragmentos.
- Añadir la cabecera del nivel de transporte a todos los segmentos.
- Usando CRC 16 bit DNP [4], cada 16 bytes se insertan 2 bytes de CRC usando los 16 bytes previos para el cálculo.

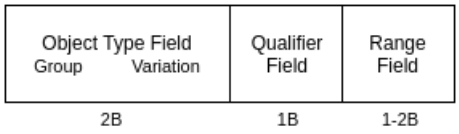


Figura 4: Object header.

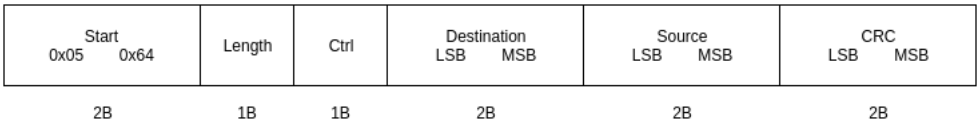


Figura 5: Link layer header.

- Añadir la cabecera del nivel de enlace. Las tramas ya están listas para ser enviadas.

4.3.1. Objetos

Los datos que se envían a través de mensajes DNP3 están encapsulados en el interior de objetos. Un objeto es un conjunto de datos que mantienen características comunes y están identificados por el *Object Type Field* (Figura 4).

- *Object Type Field*: (2 bytes) El primer byte indica de que grupo forma parte el objeto y el segundo byte, la variación de este.
- *Qualifier Field*: (1 byte) Indica la estructura de los datos que llegan. En nuestro caso para simplificar, solo usaremos el valor 0x07 que establece que el Range Field es de un solo byte y que indica el número de objetos de forma numeral (otras formas de hacerlo son con rangos o con índices).
- *Range Field*: (1-2 bytes) Indica el número de objetos en este grupo.

En el caso de esta aplicación se usan los objetos y variaciones visibles en el cuadro 1.

4.3.2. Capa de enlace

Este nivel está formado por una cabecera de 10 bytes en 6 campos distintos (Figura 5).

- Start: (2 bytes) Con valor fijo en hexadecimal, el primer campo es 0x05 y el segundo 0x64, permite a los sistemas detectar el mensaje que llega como DNP3
- Length: (1 byte) Tamaño del mensaje. Este valor no tiene en cuenta los campos Start y Length ni los CRCs.

Cuadro 1: Objetos implementados

Objeto	Variación	Descripción	Tamaño
1	1	Binary Input	1 bit
1	2	Binary Input Status	1 byte
10	1	Binary Output	1 bit
10	2	Binary Output Status	1 byte
30	1	32 bit Analog Input Quality	5 bytes
30	2	16 bit Analog Input Quality	3 bytes
30	3	32 bit Analog Input	4 bytes
30	4	16 bit Analog Input	5 bytes
40	1	32 bit Analog Output Status	5 bytes
40	2	16 bit Analog Output Status	3 bytes
40	3	32 bit Analog Output	4 bytes
40	4	16 bit Analog Output	2 bytes
70	1	File Object Identifier	1 byte
100	1	Short Floating Point	1 byte
100	2	Long Floating Point	2 bytes
100	2	Extended Floating Point	4 bytes
110	1	Octet String	1 byte

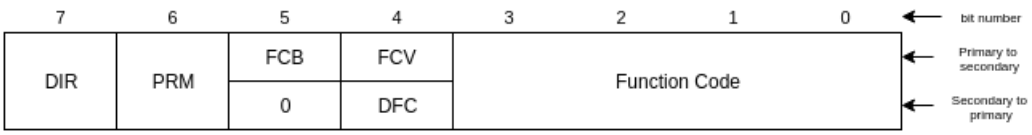


Figura 6: Link layer control byte.

- Control: (1 byte) Código de control. Permite fijar los servicios, el sentido de flujo y el tipo de comunicación (Figura 6).
- Destino: (2 bytes) Contienen un valor que identifica a la maquina con la que se comunica.
- Origen: (2 bytes) Contiene un valor que identifica a la máquina que envía este mensaje.
- CRC: (2 bytes) Código de detección de errores 16 bit CRC DNP. Se calcula con los 8 bytes de los campos anteriores y el polinomio $x^{16} + x^{13} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^2 + 1$.

4.3.3. Capa de transporte

Consiste en un solo byte situado después de la cabecera de enlace y contiene los siguientes campos (Figura 7).

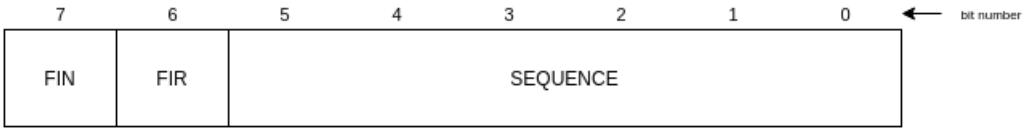


Figura 7: Transport layer header.

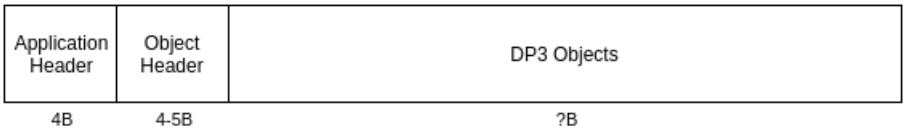


Figura 8: Application layer.

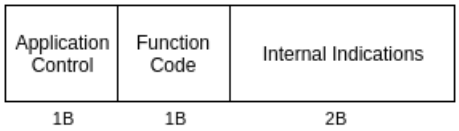


Figura 9: Application header.

- FIN: (1 bit) Bit a 1 indica que el segmento actual es el último de todos los enviados.
- FIR: (1 bit) Bit a 1 indica que el segmento actual es el primero de todos los enviados.
- SEQUENCE: (6 bits) Indica el número de orden del segmento enviado.

4.3.4. Capa de aplicación

La trama de esta capa que aparece a continuación de la capa de transporte está compuesta únicamente por tres campos (Figura 8):

- Application Control: (1 byte) que está compuesto por los siguientes subcampos.
  - FIN: (1 bit) Bit a 1 indica que el segmento actual es el último de todos los enviados.
  - FIR: (1 bit) Bit a 1 indica que el segmento actual es el primero de todos los enviados.
  - Confirmed: (1 bit) Indica si el segmento ha de ser confirmado, es decir si el receptor del mensaje a de mandar un mensaje DNP3 de vuelta mostrando que lo ha recibido
  - Unsolicited: (1 bit) Indica si el segmento que se envía no ha sido pedido por el *máster* si no que se ha enviado de forma esporádica.

    - Sequence: (4 bits) Indica el número de orden del segmento enviado.

- Function Code: (1 byte) Código de la función.
- Internal Indications: (2 bytes) Código usado solo en la respuesta del *outstation* al *máster* con diferentes bits que indican el estado actual del sistema (Problemas, sincronización, overflow, configuración corrupta...).
- Object Type Field: (2 bytes) El primer byte indica de que grupo forma parte el objeto y el segundo byte, la variación de este.
- Qualifier Field: (1 byte) Indica la estructura de los datos que llegan. En nuestro caso para simplificar, solo usaremos el valor 0x07 que indica que el Range Field sea de un solo byte e indique el número de objetos de forma numeral (otras formas de hacerlo son con rangos o con índices).
- Range Field: (1-2 bytes) Indica el número de objetos en este grupo.

A partir de aquí se colocan los datos. El tamaño de los datos enviados depende del Object Type y del Range Field. Una vez acaba el objeto actual, se transmite el siguiente, comenzando con los tres campos anteriores. Conviene notar que cada 16 bytes de la capa de aplicación hay que insertar 2 bytes de CRC, lo que incluye también a las cabeceras de la capa de aplicación. Ha de haber también un CRC final aunque el tamaño de los datos previos sea menor a 16 bytes.

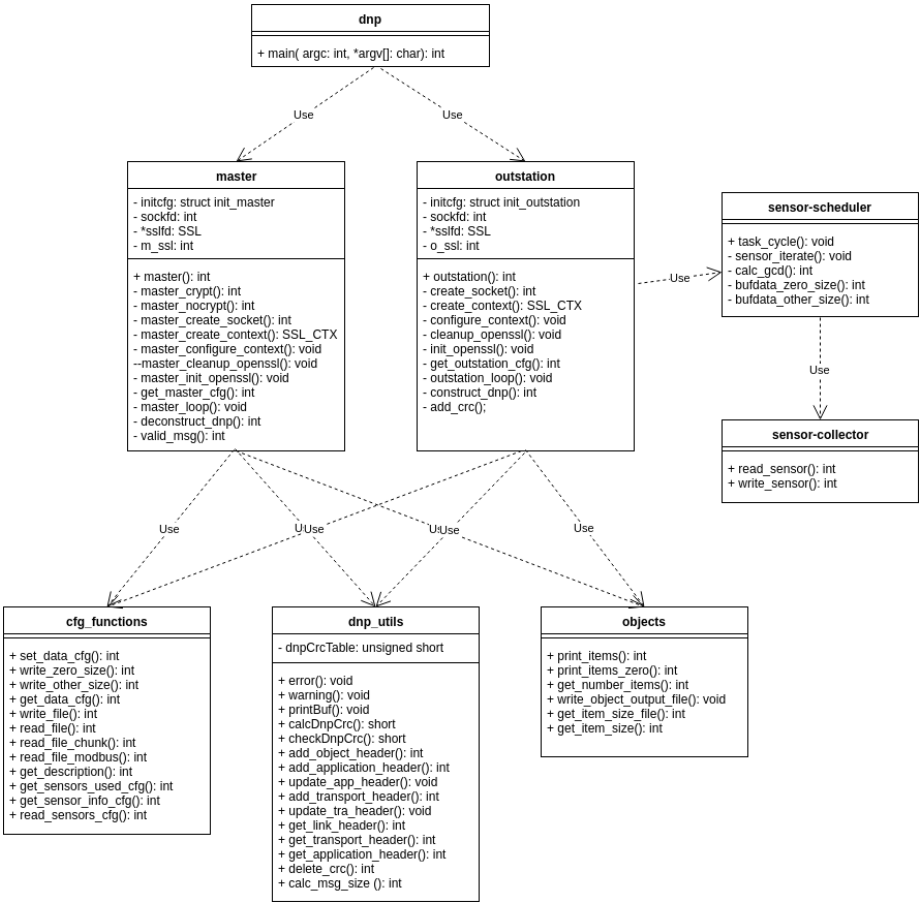


Figura 10: Diagrama de clases de la implementación DNP3.

5. Implementación del protocolo DNP3

La ejecución comienza en `dnp` con la función main, es la encargada de tratar los argumentos y ejecutar el servidor *máster* o el cliente *outstation*.

Tanto *outstation* como *máster* tienen una estructura similar. Ambos comienzan leyendo el fichero de configuración que define valores a nivel de enlace, el puerto de conexión, y valores de la aplicación que establecen la frecuencia de envío. Una vez configurados crean un socket TCP para leer y escribir sobre él, el *máster* llama a la función `loop` y el *outstation* a la función del ejecutivo cíclico.

Estas funciones se encargan de iniciar el servicio DNP. A partir de aquí la estructura del código es similar en ambas máquinas pero la funcionalidad es la opuesta. El *outstation*, mediante el ejecutivo cíclico, se encarga de obtener valores y de componer mensajes, el *máster* por su parte, los descompone y los analiza.

El *outstation* inicia el ejecutivo cíclico y recibe los valores de los sensores ya encapsulados en objetos DNP3. El siguiente paso es dividir los datos de los sensores si superan el tamaño máximo de la capa de aplicación. A continuación, se incluye la cabecera de la capa de aplicación y se comprueba que no sobrepasa el tamaño máximo de la capa de transporte, si lo supera se divide de nuevo. Tras asegurar que los segmentos tienen un tamaño valido, se añade la cabecera de la capa de transporte a todos ellos. Tanto la cabecera de aplicación como la de transporte contienen valores que informan del orden del segmento para la correcta recepción de la trama completa.

En este momento el *outstation* tiene preparada una trama a nivel de transporte que aún requiere cambios. Cada 16 bytes se calcula un CRC DNP de 16 bits y se insertan estos dos bytes después de los 16 bytes. Por último se añade la cabecera de enlace dejando el mensaje listo para ser enviado hacia el *máster*.

El *máster* por su parte hace lo mismo que el *outstation* pero en orden inverso. Elimina la cabecera de enlace y los CRC, comprobando que estos últimos sean correctos. En caso de no serlos se deniega el mensaje completo y se espera a que llegue un nuevo mensaje. Se elimina la cabecera de transporte pero con atención a los valores de secuencia que contienen, que ayudan para reensamblar la trama de objetos en el orden original. Lo mismo ocurre con la cabecera de aplicación. El resultado final es una trama de objetos DNP3. El *máster* llama a la función que descompone los objetos y los envía al script `IFTT` para ser analizados, en caso de que algún valor no entre dentro de los rangos aceptables se enviará al *outstation* una respuesta DNP3 con órdenes para los sensores.

5.1. Librería de configuración

Se usan varios ficheros de configuración en el sistema, accediéndose a ellos mediante la librería `libconfig` [5]. Al iniciar el servicio DNP3 tanto el *máster* como el *outstation* necesitan leer sus ficheros de configuración. Estos ficheros de configuración difieren para el *máster* y el *outstation* y definen ip, puerto, intervalo entre envío de tramas y parámetros que se usaran en los campos de la capa de enlace DNP3.

El conjunto de sensores también tiene sus propios ficheros de configuración. Existen dos ficheros. Uno de ellos contiene todos los sensores que se pueden utilizar, estén instalados o no en el sistema actual, y define la cantidad de variables y el tipo de cada variable que envía cada sensor. El otro fichero de configuración solo contiene el nombre de los sensores que se están usando en ese momento.

El uso de estos ficheros de configuración hace que sea trivial añadir nuevos sensores. Solo es necesario añadir la definición del sensor al primer fichero, definir el tipo de objeto DNP3, variación y el número de datos que se reciben en cada muestreo.

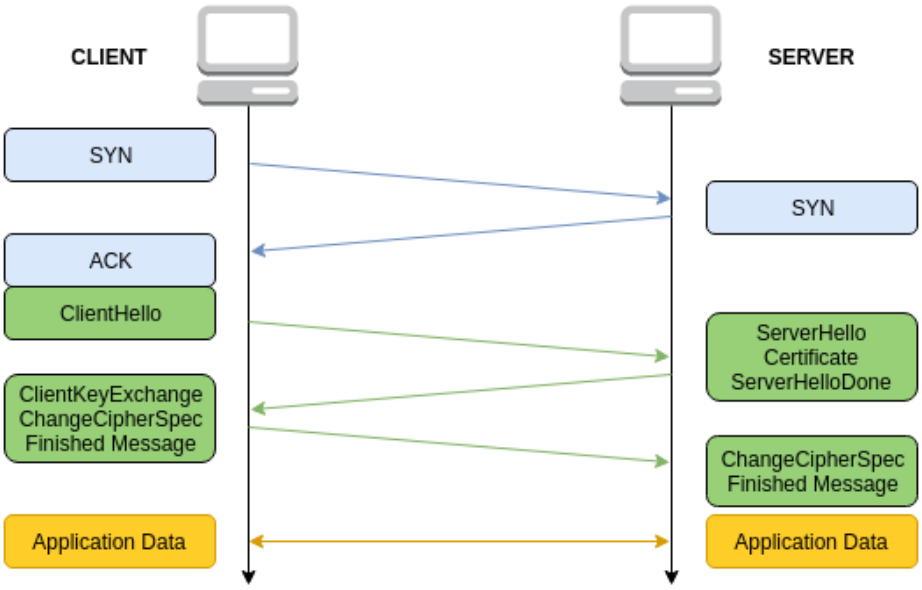


Figura 11: Establecimiento de una sesión TLS.

5.2. Criptografía TLS

La aplicación diseñada hace uso de criptografía TLS para la comunicación entre *outstation* y *máster*. TLS es un protocolo de criptografía basado en SSL que se emplaza sobre el protocolo TCP en la capa OSI de transporte [6].

Se usa `OpenSSL` para facilitar la integración de la capa de criptografía. El uso de la criptografía en la aplicación es opcional, pudiéndose cambiar el fichero de configuración para evitar su utilización, pero el uso de criptografía es recomendado.

Cuando ambos sockets han sido generados usando `OpenSSL` es el cliente *outstation* quien comienza la comunicación. Lo hace especificando una lista de conjuntos de cifrados, métodos de compresión y la versión del protocolo SSL/TLS más alta permitida. El servidor responde eligiendo los parámetros a partir de las opciones del cliente. Tras esto se intercambian los certificados, generalmente X.509, que autentican a ambas partes. Por último, el cliente y el servidor negocian una clave secreta simétrica que se ha originado con el algoritmo Diffie-Hellman. A partir de este momento ambos extremos son capaces de enviar las tramas DNP3 cifradas.

5.3. Ejecutivo cíclico

El proceso *outstation* ha de realizar varias tareas que se ejecutan debiendo cumplir un periodo de tiempo. Existen dos tipos de tareas que conforman el ejecutivo cíclico.

Algoritmo 1: Algoritmo ejecutivo cíclico.

```
1  function task_cycle(mcd, mcm, timecycle, tasklist, dnp3period)
2  step = mcm;
3  while always do
4  |  timeleft = time_compare(timecycle);
5  |  if !is_time_zero(timeleft) then
6  |  |  clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME,
   |  |  timeleft, NULL);
7  |  end
8  |  for i in sensorlist.length do
9  |  |  if step % sensorlist[i].period == 0 then
10 |  |  |  get_sensor_data(sensorlist[i].id, message);
11 |  |  end
12 |  end
13 |  if step % dnp3period == 0 then
14 |  |  send_dnp3(message);
15 |  |  receive_dnp3_response();
16 |  end
17 |  step -= mcd;
18 |  if step ≤ 0 then
19 |  |  step = mcm;
20 |  end
21 |  add_seconds(timecycle, mcd);
22 end
```

Las tareas de tipo *sensor* se encargan de leer los valores de un sensor realizando una comunicación Modbus con la placa Arduino que, una vez ha identificado la petición que ha recibido, selecciona el valor y lo devuelve al *outstation* donde la propia tarea lo encapsula en un objeto DNP3. Todas tienen el mismo nivel de prioridad y se ejecutan en el orden que han sido declaradas en el fichero de configuración. Hay dos ficheros de configuración que son utilizados para poder editar y añadir nuevas tareas de tipo sensor.

El primer fichero de configuración, `sensors.cfg`, contiene todos los sensores que se han implementado y que pueden o no ser usados en el sistema, además se encarga de definir el tipo de sensor del que se trata con una descripción y con el tipo de objeto DNP3 en el que se va a encapsular. El segundo, `sensors_used.cfg`, determina los sensores que se van a usar en el sistema actual y el periodo de la tarea que lee el valor del sensor.

El otro tipo de tareas son las que denominamos *tareas de comunicación*. Constan de dos tareas ya definidas, la escritura de los mensajes DNP3 que van a ser enviados al *máster* y la lectura de las respuestas que llegan del *máster*. Estas dos tareas tienen su periodo definido en el fichero de configuración del proceso *outstation*.

El ejecutivo cíclico encargado de organizar y ejecutar cada una de las tareas cumpliendo sus periodos es un ejecutivo `soft real-time`, por lo que se aligeran las constricciones de tiempo. Se permite que alguna tarea sobrepase su *deadline* ya que no es un sistema crítico. Como reloj se usa `CLOCK_MONOTONIC` para las mediciones de tiempo, es un reloj absoluto que garantiza su valor linealmente incremental durante casi 50 años.

El ejecutivo usa el Algoritmo 1. Los valores *mcd* y *mcm* representan el máximo común divisor y el mínimo común múltiplo de los periodos de las tareas que se han seleccionado en el fichero de configuración `sensors_used.cfg`. Tanto *mcd* como *mcm*, al existir las restricciones establecidas en el fichero de configuración, tendrán siempre un valor de segundos. A continuación se describen funciones relevantes del ejecutivo.

- clock_gettime(clockid_t clk, struct timespec *t): Asigna el valor del reloj al `struct timespec *t`. En este ejecutivo se usa el reloj `CLOCK_MONOTONIC`.
- clock_nanosleep(clockid_t clock_id, int flags, const struct timespec *request, struct timespec *remain): Suspende la ejecución del thread que llama a la función durante el tiempo `req` usando el reloj establecido en `clock_id` que en este caso será el reloj usado en la función clock_gettime(). Se usa el flag `TIMER_ABSTIME` y el parámetro `remain` es `NULL`.
- time_compare(struct timespec *t): Compara el valor del parámetro *t con el valor del reloj en el momento actual obtenido con clock_gettime(), devolviendo un `struct timespec` con la diferencia entre ambos.
- add_seconds(struct timespec *t, int sec): Añade a `*t` el valor de sec.
- is_time_zero(struct timespec *t): Devuelve uno si `*t` tiene un valor igual o menor a cero, devuelvo cero si el tiempo es mayor.

Para que el ejecutivo cíclico cumpla las restricciones tiempo real de las tareas, debe de lanzar una nueva instancia de cada una de acuerdo con su periodo, y de modo que finalice antes del siguiente. Para ello se ha tomado como tiempo de ejecución de cada tarea (C) el WCET (*Worst Case Execution Time*) de la misma, usando una función que devuelve un valor temporal con precisión de nanosegundos antes y después de la ejecución de la tarea (Cuadro 2). El tiempo de ejecución de las tareas de tipo sensor es el mismo ya que solo se encargan de obtener el valor del sensor en la placa Arduino, que al tener los valores de los sensores guardados en memoria, no necesita calcularlos cada vez que se realiza una petición. Se ha utilizado el contexto de la instalación realizada en la planta depuradora de viñedos Pago de Aylés.

Se ha incluido un diagrama que muestra la organización de la ejecución de tareas en el marco temporal (Figura 12).

Cuadro 2: Tareas del ejecutivo cíclico en Pago de Aylés

Tarea	C(ms)	T(ms)	D(ms)
Caudalímetro	12	2000	2000
Detector mov.	12	2000	2000
Nivel líquido	12	2000	2000
Sensor PH	12	4000	4000
Anemómetro	12	8000	8000
Veleta	12	8000	8000
Humedad	12	8000	8000
Temperatura	12	8000	8000
Escribir DNP3	19	2000	2000
Leer DNP3	104	2000	2000

Figura 12: Ejecutivo cíclico Pago de Aylés.

5.4. Script IFTTT

La lógica que permite reaccionar ante ciertos valores de los sensores está basada en la tecnología IFTTT (If This Then That) en la que existe un script o receta que contiene unas instrucciones simples que ejecutan una acción según los valores de entrada. Existe un único script que está implementado en BASH ejecutado a través del código C del máster DNP3. En el caso del script que funciona en este sistema los parámetros de entrada son pares ïd_de_sensor valor_muestreado", tras ser analizado por el script la salida será un par ïd_de_sensor código_de_acción"que será encapsulado en un mensaje DNP3 para ser devuelto al *outstation* como respuesta.

La principal ventaja de usar este método en lugar de incluir la funcionalidad en el programa principal es que no hace falta recompilar el sistema entero cada vez que sea necesario cambiar algún elemento en la lógica, ni detener el sistema para usar nuevos valores, ya que es editable en tiempo de ejecución.

Algoritmo 2: Extracto de script IFTT.

```
1  if [$sensor == "a"] then
2      #sensor de nivel
3      if $value -lt 20 then
4          #nivel alto
5          #desactivar bomba de líquido
6          echo "c 0"
7      end
8      if $value -gt 100 then
9          #nivel bajo
10         #activar bomba de líquido
11         echo "c 1"
12     end
13 end
```

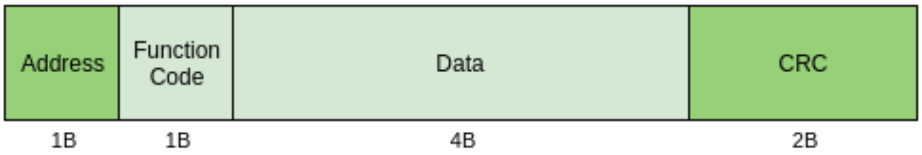


Figura 13: Campos del protocolo Modbus.

6. Sensores

6.1. Arduino Due

La placa Arduino Due contiene una CPU ARM Cortex-M3 de 32 bits con una frecuencia de 84 MHz. Los pines I/O permiten una tensión máxima de 3,3V que es diferente del resto de Arduinos disponibles ya que funcionan a 5V.

El lenguaje de programación usado para la placa Arduino es C++. Se ha utilizado el Arduino IDE para programar sobre la placa haciendo uso de las librerías de terceros TimeLib y TimeAlarms para la implementación de valores históricos de los sensores.

La comunicación entre el GuruPlug *outstation* y la placa Arduino Due se realiza mediante un cable USB usando el protocolo de comunicación Modbus (Figura 13). Modbus es un protocolo simple pero robusto para comunicaciones en puertos serial compuesto por cuatro campos: La dirección del dispositivo al que se dirige la acción, el código de función que especifica el tipo de acción, un conjunto de datos y por ultimo dos bytes de CRC [8].

Existen dos tipos de comunicación.

- Lectura: El *outstation* realiza una petición y el código de función Modbus contiene un código que corresponde con la lectura de un sensor. El Arduino verifica que es un mensaje Modbus correcto y comprueba el CRC, tras esto

comprueba cual es el sensor al que se ha realizado la petición y coge el valor correspondiente de la posición de memoria donde está almacenado el valor. Compone un mensaje Modbus y lo devuelve al *outstation* por el USB.

- Escritura: El *outstation* envía un mensaje por el Modbus con un código de función que corresponde al de la escritura de un sensor. Tras verificarse que el mensaje es correcto, se comprueba cual es la posición de memoria en la que está el registro del sensor sobre el que se va a actuar y se sobrescribe usando el valor del campo Data del mensaje Modbus.

Inicialmente la comunicación serial del Arduino a través del USB es lenta estando siempre por encima del segundo, para mejorar la velocidad de las comunicaciones es necesario cambiar dos valores en la placa Arduino. Las funciones de lectura del serial tienen como estándar un timeout de un segundo que garantiza que el mensaje a leer va a llegar de forma completa y que no se va a detener la lectura cuando aún faltan datos por llegar. Se ha reducido este timeout a un milisegundo, esto es posible cambiando el valor de `Serial.setTimeout` y haciendo que antes de realizar una lectura se compruebe que se han recibido en el serial tantos bytes como tamaño tiene el mensaje de Modbus que se está usando. El otro cambio que mejora la velocidad de la comunicación es aumentar la velocidad de la transmisión de datos al serial, inicialmente el valor son 9600 bps y se ha cambiado a 115200 bps.

El código consiste en una función de setup que activa los pines a los que están conectados los sensores, prepara las interrupciones e inicializa variables necesarias para la ejecución. Una vez acaba la ejecución de la función setup se inicia la función loop que se ejecuta de forma continua y permanente, desde aquí se llama al resto de funciones necesarias para el muestreo de los sensores.

6.2. Caudalímetro

El modelo FS300A G3/4.[es] un caudalímetro simple que funciona por el efecto Hall al producirse una corriente en presencia de un electroimán perpendicular que rota empujado por el flujo del interior del caudalímetro. Tiene tres cables, el rojo contiene la tensión de 5V para alimentar el caudalímetro, el amarillo es el cable de salida que sacará 5V cada vez que se reproduzca el efecto Hall y por último el cable negro que funciona como tierra.

La implementación se ha realizado mediante interrupciones, cada vez que llega un flanco de subida del sensor la interrupción ejecuta una función que aumenta el valor de una variable. El muestreo se realiza cada segundo, existe un timeout preparado para que después de un segundo ejecute una función que calcule el número de litros que han pasado por el caudalímetro según el número de interrupciones usando la siguiente ecuación.

$$Caudal = \frac{Interrupciones * 60}{5{,}5Q}$$

Usando la librería para Arduino TimeAlarm se ha implementado un histórico de los datos del caudalímetro, pudiéndose observar el caudal de las últimas horas.

6.3. Anemómetro

El anemómetro 6410 Davis Vantage Pro 2 tiene dos funcionalidades, medir la velocidad del viento y con la veleta medir la orientación desde la que sopla el viento.

Para la velocidad del viento se usan interrupciones en el flanco de subida que aumentan el valor de una variable. Debido a la variabilidad del viento que cambia continuamente de velocidad, el muestreo se realiza cada 10 segundos para aumentar la fiabilidad de la medición. Para convertir los pulsos de la interrupción en velocidad real del viento existe una ecuación del fabricante que produce millas por hora, para convertirlo a metros por segundo es necesaria añadir otra operación de división a la fórmula del fabricante.

$$Velocidad = \frac{\frac{Pulsos*2{,}25}{Periodo}}{0{,}44704}$$

La veleta devuelve un entero entre 0 y 360 siendo 180 el soporte de la veleta por lo que para obtener un valor correcto es necesario instalar la veleta en una pared que apunte al norte o si no es posible usar un offset para corregir el valor.

6.4. Detector de movimiento

El PIR HC-SR501 es un detector de movimiento preparado para placas Arduino con dos potenciómetros para regular la sensibilidad y el tiempo de activación tras detectar movimiento. Existe un jumper que permite cambiar la modalidad de disparo, disparo repetido en el que la señal está constantemente alternando o disparo único en el que la señal permanece activada. Requiere 5V de alimentación, un cable a tierra y el cable de señal con salida a 5V que deberá ser convertido a 3,3V para el Arduino Due.

Cada vez que el dispositivo detecta movimiento, se dispara una interrupción en el Arduino que aumenta la variable que contiene el número de detecciones, GuruPlug lee los datos cada 10 segundos.

6.5. Sensor de humedad y temperatura

El sensor DHT22 permite medir simultáneamente la humedad y la temperatura debiendo dejar más de dos segundos entre lecturas. Tiene cuatro pines: El pin de alimentación de 5V, pin de salida de datos a 5V, un pin cuyo uso no es necesario para este proyecto y por último el pin final para tierra.

En el código hace falta usar la librería cactus_io_DHT22 que se encarga de facilitar la lectura de los valores del sensor. Es necesario inicializar el sensor con la función dht.begin() y a partir de ahí usar las funciones disponibles para leer los valores del sensor, teniendo en cuenta siempre, que el sensor está preparado

para solo actualizarse cada dos segundos por lo que al realizar lecturas con intervalos menores se recibe el mismo valor.

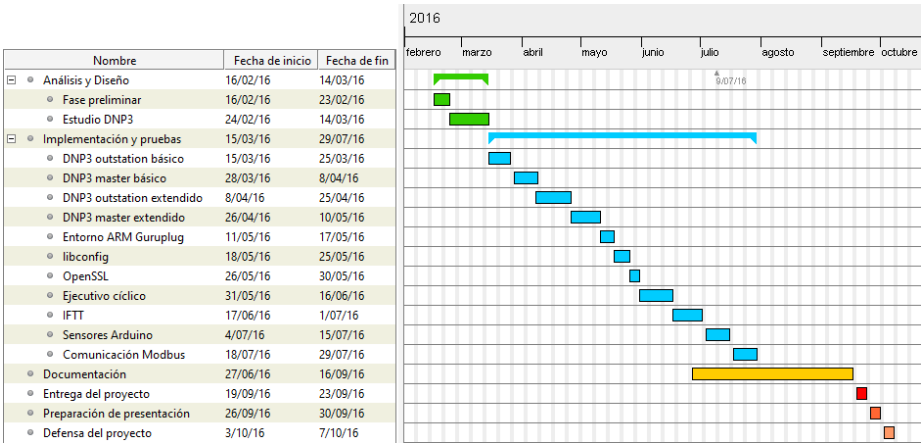
6.6. Sensor de ultrasonidos

El sensor de ultrasonidos HC-SR04 para Arduino mide la distancia a la que se encuentra un objeto o líquido. Tiene un rango mínimo de dos centímetros y un máximo de cuatro metros [9].

Tiene cuatro pines para conexiones, uno de alimentación de 5v, uno para tierra, un pin de *trigger* y otro de *echo*, estos dos últimos pines son el que se usa para realizar la petición de medición de distancia y la señal que avisa de la recepción del ultrasonido respectivamente.

El sensor funciona de la siguiente manera, hay un emisor de ultrasonidos y un receptor de ultrasonidos, cuando el sensor recibe una señal en el pin de *trigger*, este envía el ultrasonido. El ultrasonido rebotará contra alguna superficie y volverá al sensor donde será detectado por el receptor de ultrasonidos que activará el pin de *echo*.

En el código del Arduino es necesario llamar a una función de librería que mide el tiempo que tarda hasta que llega la señal de *echo*. Con el tiempo y conociendo la velocidad del sonido en el aire podemos calcular la distancia del objeto sobre el que ha rebotado el ultrasonido



Nombre	Fecha de inicio	Fecha de fin
Análisis y Diseño	16/02/16	14/03/16
Fase preliminar	16/02/16	23/02/16
Estudio DNP3	24/02/16	14/03/16
Implementación y pruebas	15/03/16	29/07/16
DNP3 outstation básico	15/03/16	25/03/16
DNP3 master básico	28/03/16	8/04/16
DNP3 outstation extendido	8/04/16	25/04/16
DNP3 master extendido	26/04/16	10/05/16
Entorno ARM Guruplug	11/05/16	17/05/16
libconfig	18/05/16	25/05/16
OpenSSL	26/05/16	30/05/16
Ejecutivo cíclico	31/05/16	16/06/16
IFTT	17/06/16	1/07/16
Sensores Arduino	4/07/16	15/07/16
Comunicación Modbus	18/07/16	29/07/16
Documentación	27/06/16	16/09/16
Entrega del proyecto	19/09/16	23/09/16
Preparación de presentación	26/09/16	30/09/16
Defensa del proyecto	3/10/16	7/10/16

Figura 14: Diagrama de Gantt del proyecto.

7. Gestión del proyecto

Como se expuso previamente se ha utilizado una metodología ágil para la realización del proyecto. A continuación se muestra en detalle la planificación del proyecto, las horas de trabajo y las herramientas utilizadas para la administración y gestión del trabajo.

7.1. Planificación

El proyecto comenzó el día 15 de febrero de 2016, principalmente definiendo los objetivos y las funcionalidades a ser desarrolladas. Acabada esta fase preliminar se comenzó con el análisis del proyecto para realizar posteriormente un primer diseño.

Las iteraciones en las que se realizan análisis, diseño, implementación, pruebas y documentación concluyen el día 29 de julio. Tras las iteraciones se dedicó un tiempo a acabar la memoria que ya se había comenzado previamente y dejarla preparada para el depósito (Figura 14).

7.2. Tiempo dedicado

Para la realización del proyecto se han invertido unas 392 horas, estas incluyen las fases de análisis, diseño, implementación y documentación (Figura 15).

7.3. Herramientas de gestión

Se han utilizado las siguientes herramientas para administrar y gestionar la elaboración del proyecto.

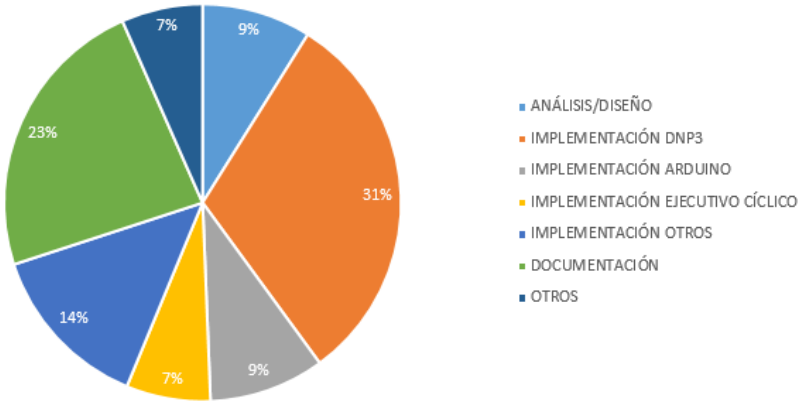


Figura 15: Tiempo dedicado por tarea.

- GitLab. Es una plataforma de desarrollo colaborativo para alojar proyectos en un servidor privado de la empresa usando el sistema de control de versiones Git. Se ha usado para almacenar el código del proyecto.
- Google Drive. Servicio de alojamiento de archivos donde se ha almacenado documentación sobre el proyecto.
- Gantt Project. Herramienta para crear diagramas de Gantt.
- draw.io. Herramienta hospedada en una página web que permite realizar diagramas de todo tipo incluyendo diagramas UML.

8. Conclusiones

Una vez finalizado el proyecto, se realizan las valoraciones sobre el desarrollo y el resultado obtenido del proyecto.

8.1. Resultados

Con el proyecto acabado se ha conseguido implementar el sistema diseñado cumpliendo con todas las especificaciones definidas al comienzo del proyecto. El sistema final permite obtener datos de sensores y dispositivos, comunicarlos con el protocolo DNP3 y TLS a un máster que monitoriza y analiza los datos y que es capaz de responder ante valores impropios de los dispositivos enviando una respuesta que corrige el funcionamiento de los dispositivos y sensores.

Se ha desarrollado un ejecutivo cíclico soft real-time que obtiene las tareas de un fichero de configuración y que permite cambiar las tareas sin necesidad de recompilar el código.

Por último se ha implementado una comunicación Modbus con una placa Arduino para obtener los valores de los sensores que también han sido programados y calibrados.

8.2. Lecciones aprendidas

Durante el desarrollo del proyecto se han aprendido lecciones que serán útiles para futuros proyectos.

- Documentarse minuciosamente antes de implementar: Usando librerías en ocasiones la solución parece simple y obvia, pero en varias ocasiones ha ocurrido que ha sido necesario cambiar la implementación de la librería tras comprobar que la implementación inicial no daba el resultado esperado y la documentación ya explicaba el posible problema.

- Realizar pruebas completas al finalizar cada funcionalidad: En alguna ocasión después de implementar una funcionalidad simple las pruebas realizadas no cubrían todos los casos y poco después habiendo avanzado a otras funcionalidades, la funcionalidad inicial falló en un caso especial que no se había comprobado.

- Metodología de ingeniería inversa: Previamente a este proyecto nunca había intentado usar ingeniería inversa. Al principio la investigación fue lenta, pero con organización y con una metodología establecida se aumentó la productividad.

8.3. Conclusión personal

El proyecto ha requerido bastante organización, especialmente debido a la envergadura que tiene ya que hasta ahora nunca había realizado un proyecto tan grande, no solo ha sido un proyecto de desarrollo de software sino que también

ha requerido implementar sensores físicos. Además el proyecto requería que se ejecutase en los GuruPlug ARM por lo que ha sido necesario crear un entorno de compilación cruzada con BuildRoot para poder ejecutar el código en máquinas ARM.

He aprendido a usar tecnologías y librerías que no conocía pero también he mejorado mi conocimiento de otras que ya conocía como el lenguaje de programación C, que aunque ya lo había usado varias veces, nunca lo había hecho a este nivel de profundidad.

Además me he tenido que desenvolver en un entorno laboral real. Aunque el proyecto ha sido individual, he debido de colaborar con compañeros para ciertas tareas y cada pocas semanas mostrar como avanzaba el proyecto a mi director de TFG.

Por último, la realización del proyecto me ha aportado madurez, capacidad organizativa y la paciencia para enfrentarme a nuevos proyectos de gran envergadura.

8.4. Futuro del proyecto

Este proyecto de SCADA con comunicación DNP3 ha sido completado y ya está preparado para ser incluido en una instalación industrial, de hecho la parte de Arduino y sensores ya forma parte de la instalación de la depuradora de Aylés.

No obstante, es posible ampliar el proyecto añadiendo nuevos objetos DNP3, nuevos sensores u otras funcionalidades que se crean adecuadas. El protocolo DNP3 está preparado para transmitir cualquier tipo de dato que sea necesario.

Bibliografía

[1] J. Thomas, *How to Set Up Buildroot/QEMU/ARM Cross-Development Environment*, 28 Febrero 2010, [En línea], http://processors.wiki.ti.com/images/f/f5/Aaa.pdf

[2] V. Prakash, *Advantages of the DNP3 Communications Protocol in Water and Wastewater Telemetry Systems*, Enero de 2012, [En línea], http://www.automation.com/pdf_articles/1261002_DNP3WaterWastewaterWP.pdf

[3] Citrix, *How to Decrypt SSL and TLS Traffic Using Wireshark*, 25 Noviembre 2015, [En línea], http://support.citrix.com/article/CTX116557

[4] J. McFadyen, *Calculating a CRC in DNP3.0 protocol*, 1 Noviembre 2000, [En línea], https://www.experts-exchange.com/questions/11722859/Calculating-a-CRC-in-DNP3-0-protocol.html

[5] Mark A. Linder, *A Library For Processing Structured Configuration Files*, 16 Mayo 2015, [En línea], http://www.hyperrealm.com/libconfig/libconfig_manual.html

[6] OpenSSL Validation Services, Inc., *OpenSSL FIPS Object Module v2.0*, 10 Mayo 2016, [En línea], https://www.openssl.org/docs/fips/UserGuide-2.0.pdf

[7] E. Oswald, *What is IFTTT and How Does It Work?*, 30 Junio 2016, [En línea], http://www.digitaltrends.com/cool-tech/what-is-ifttt-and-how-does-it-work/

[8] Modicon, *Modicon Modbus Protocol Reference Guide*, Junio 1996, [En línea], http://modbus.org/docs/PI_MBUS_300.pdf

[9] L. Llamas, *Medir distancia con arduino y sensor de ultrasonidos*, 16 Junio 2015, [En línea], http://www.luisllamas.es/2015/06/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/

Anexos

1. Anexo 1

Cuadro 3: Requisitos funcionales

Código	**Requisito funcional**
RF1	El sistema estará compuesto por tres sistemas empotrados distintos: El Arduino, el GuruPlug con funcionalidad *outstation* y el GuruPlug con funcionalidad *máster*. Un arduino está conectado a un sólo GuruPlug con funcionalidad *outstation* y varios Guruplug con funcionalidad *outstation* pueden estar conectados a un Guruplug con funcionalidad *máster*.
RF2	La comunicación entre el *máster* y el *outstation* se realizará mediante el protocolo DNP3 sobre TCP/IP.
RF3	La comunicación entre la Placa Arduino y el GuruPlug *outstation* se realizará usando el protocolo Modbus.
RF4	La comunicación DNP3 podrá opcionalmente realizarse de forma encriptada usando el protocolo de seguridad TLS.
RF5	El arduino se encargará de controlar las comunicaciones con los sensores, muestreando los valores según el tipo de sensor y recolectando todos los valores en la memoria del arduino para después poder enviárselos al GuruPlug *outstation*.
RF6	El GuruPlug *outstation* se encargará de comunicarse con el Arduino mediante el protocolo modbus para coger los datos, empaquetará los datos en un mensaje DNP3 y los enviará al GuruPlug máster, cuando reciba una respuesta del *máster*, el GuruPlug *outstation* se encargará de analizarla y realizar las acciones necesarias en los sensores.
RF7	El GuruPlug *máster* recibirá el mensaje DNP3 del GuruPlug *outstation*, procesará la información recibida y devolverá una respuesta al GuruPlug *outstation*.
RF8	Al recibir un mensaje DNP3, el *máster* ha de ejecutar un script basado en tecnología IFTTT que permita comprobar que el rango de los datos recibidos es válido. Generará una respuesta que contendrá únicamente la palabra response.^{en} caso de ser todos los valores válidos. Si se ha detectado un valor inadecuado se añadirá además una acción que los sensores/actuadores deberán realizar para corregir el valor inadecuado componiéndose de la siguiente manera ïd_del_sensor código_de_acción".
RF9	Tanto el *máster* como el *outstation* tendrán su propio fichero de configuración que permita variar el comportamiento del sistema: Uso o no de criptografía, puertos e ips en los que se realizará la conexión o frecuencia de envió de mensajes DNP3.
RF10	Existirán dos ficheros de configuración para los sensores. El primero define los sensores que han sido preparados para poder ser utilizados, contienen una descripción del sensor y define el objeto y variación que se usará para transmitir el valor en la comunicación DNP3. El segundo fichero de configuración define los sensores que se van a utilizar en el sistema actual y el periodo de muestreo de cada sensor.