UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DISEÑO Y CONSTRUCCIÓN DE UN SISTEMA DE ALARMAS UTILIZANDO EL PROTOCOLO DNP3

TESIS

PARA OBTENER EL TÍTULO DE
INGENIERO ELÉCTRICO - ELECTRÓNICO

ÁREA
ELECTRÓNICA

PRESENTA
JOSÉ LUIS SILVA PERALES

DIRECTOR DE TESIS
ING. DANIEL MARTÍNEZ GUTIÉRREZ

MÉXICO, D. F. **2012**

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Un sistema SCADA (Supervisory Control And Data Aquisition System), se encarga de recabar información de dispositivos pudiendo provenir de sensores conectados a una estación central para que las señales sean procesadas, mostradas y de acuerdo a esto, tomar ciertas medidas para controlar algún proceso.

Anteriormente los sistemas SCADA consistían de paneles en donde un operador podía observar las variables de un proceso en distintos tipos de indicadores. Estaban conectados directamente a los sensores que proporcionaban la información y a diversos actuadores que podían ser controlados por medio de perillas. Si bien eran sistemas relativamente sencillos y baratos, éstos comenzaron a presentar varias desventajas. Por ejemplo, los datos sólo podían ser recabados en el sitio en donde se encontraba el panel ya que el cableado podía tornarse inmanejable. En algún momento los sistemas SCADA se vieron en la necesidad de migrar a otras tecnologías como la de los PLC (Programmable Logic Controller). Con estos dispositivos la información pasó a ser digital y se eliminaron los problemas anteriores obteniéndose también varias ventajas, como la posibilidad de procesar, almacenar y transmitir los datos a una distancia mayor. Con el tiempo, se integró a los mismos sensores la tecnología de los PLC de tal forma que en algunos casos el PLC, sin embargo, dejó de utilizarse y surgieron así los DEIs (Dispositivo Electrónico Inteligente).

Un sistema SCADA actual consiste entonces de una estación central y múltiples RTU (Unidad Terminal Remota). Estos últimos pueden estar conformados por PLC o por DEI. Para la comunicación entre ellos se emplea un protocolo, que establece un mecanismo de señalización, representación, autenticación y detección de errores, por lo que permite intercambiar datos de manera segura a través de un medio físico imperfecto.

Existe una gran variedad de protocolos, cada uno diseñado para una aplicación en específico. Por ejemplo, están los protocolos de internet, diseñados con la intención de transmitir documentos, bases de datos, audio, video, correo electrónico, etc. Para que el intercambio pueda llevarse a cabo se deben tener equipos que implementen dichos protocolos. En el campo del monitoreo y la adquisición de datos no existía ningún estándar, por lo que cada fabricante de equipo

SCADA dotaba a sus productos de protocolos propietarios generando como consecuencia una gran confusión. En 1994 los protocolos DNP3 junto con el *IEC 60870-5-101*, se presentaron como posibles soluciones a tal problemática, estandarizando la forma de comunicar diversos dispositivos y sensores con estaciones centrales de monitoreo.

Desde su creación, el DNP3 se ha extendido geográficamente y a diferencia del *IEC 60870-5-101* que está destinado únicamente a la industria eléctrica, se ha extendido también a distintas industrias como la del gas y la de tratamiento de aguas residuales.

1.1. Objetivo

Implementar una porción del protocolo DNP3 en un microcontrolador para crear un sistema capaz de comunicar el estado de señales binarias de alarma. Además de esta vía de comunicación, el sistema debe tener salidas a relevador para proporcionar contactos secos, que pueden ser usados para accionar diversos dispositivos.

Capítulo 2

Protocolo DNP3

2.1. Introducción al protocolo

2.1.1. Sistemas SCADA

SCADA son las siglas del inglés *Supervisory Control And Data Acquisition* (control supervisorio y adquisición de datos).

Las funciones de un sistema SCADA pueden resumirse en lo siguiente:

1. Recabar información mediante el uso de un RTU (Unidad terminal remota).
2. Transferirla a una central.
3. Analizarla y de acuerdo a ella realizar procesos de control.
4. Desplegar la información.

En la imagen 2.1 se observa el funcionamiento de un sistema SCADA en donde las unidades terminales remotas están equipadas con una serie de sensores y actuadores. La información es enviada a la central por medio de un protocolo de comunicación para luego ser procesada y desplegada en una pantalla.

Estas funciones permiten llevar a cabo acciones dentro de la central de monitoreo como la supervisión y control remoto de instalaciones y equipos; procesamiento, visualización y almacenamiento de datos y representación de señales de alarma, en donde puede alertarse a un operador alguna situación anormal o de falla.

La comunicación entre los RTU y las centrales es lograda mediante un protocolo de comunicación, como ya se había mencionado, a través de un medio físico. Entonces, los RTU y la central de datos tienen la capacidad de generar e interpretar los mensajes, pero necesitan de algún mecanismo para transmitirlos. El dispositivo que realiza esta acción recibe el nombre de

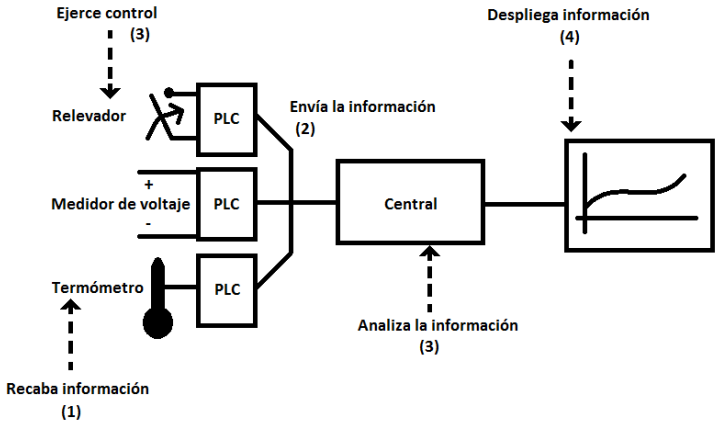


Figura 2.1: Sistema SCADA.

DCE *(Data Communication Equipment)*, el cual sirve de interfaz entre el medio y el RTU o la central de datos, que pueden ser llamados DTE *(Data Terminal Equipment)*. Un DCE puede ser un módem o cualquier dispositivo que permita inyectar la información a un medio de transmisión. Una analogía para entender mejor estos conceptos podría ser en un salón de clases. Tanto el maestro como el alumno los cuales representan dispositivos DTE pueden generar preguntas y respuestas. Éstas son articuladas en un lenguaje, por ejemplo el español, el cual representa el protocolo de comunicaciones. El medio de transmisión utilizado por la voz es el aire, por lo que la boca sería el dispositivo DCE en la analogía.

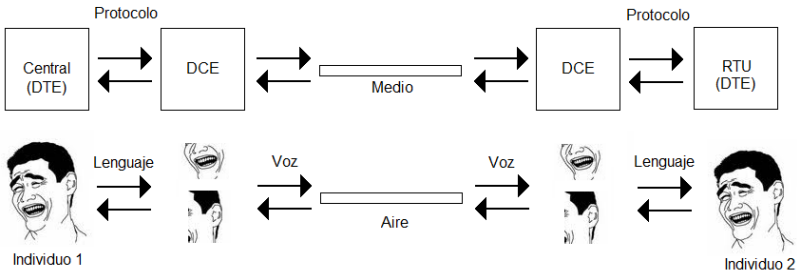


Figura 2.2: Analogía: Comunicación entre dos personas y dos dispositivos *DTE*.

Características de los RTU´s

La figura 2.3, como su nombre lo indica, muestra la estructura del hardware de un RTU y sus aditamentos (PLC´s y radio módems). Algunos de sus elementos son la fuente de poder, el CPU, memoria volátil, puertos de comunicación y módulos de entrada y salida de datos tanto digitales como analógicos. La cantidad de módulos de entrada y salida del RTU dependen del tamaño y éste a su vez depende de la aplicación, por lo que para pequeñas aplicaciones de unas

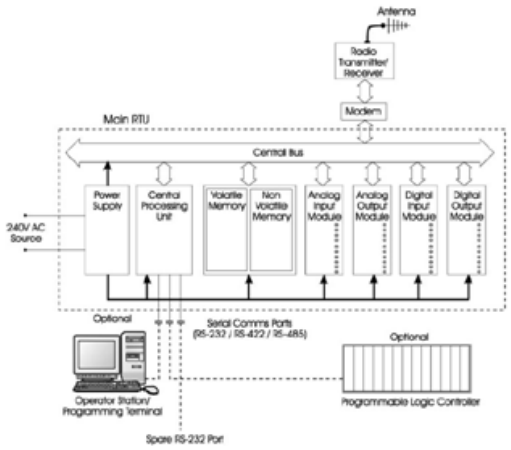


Figura 2.3: Estructura típica del Hardware de un RTU .
[2]

pocas entradas / salidas digitales pueden emplearse microprocesadores pequeños.

Estructuras de comunicación en sistemas SCADA

Estructura de comunicación hace referencia a la manera en que están dispuestos y conectados los RTU´s (al cual me referiré de ahora en adelante como dispositivos esclavo) y las centrales (que ahora serán llamadas dispositivo maestro).

1. Punto a punto: Un dispositivo esclavo intercambia información únicamente con un dispositivo maestro. Ya que el maestro gestiona la comunicación, una ventaja de esta estructura es que no se necesitan protocolos complicados para llevarla a cabo. Una de las desventajas es que si un esclavo necesita información que otro esclavo posee, esta debe pasar a través del maestro.

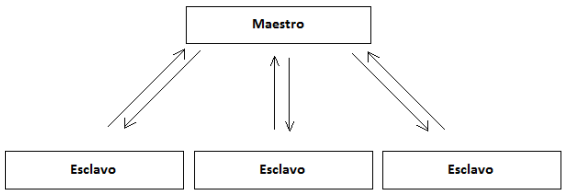


Figura 2.4: Comunicación punto a punto.

2. Comunicación multipunto: Tanto maestros como esclavos intercambian información entre sí. Se necesita un protocolo de comunicaciones sofisticado para prevenir las colisiones de los múltiples dispositivos tratando de comunicarse al mismo tiempo.

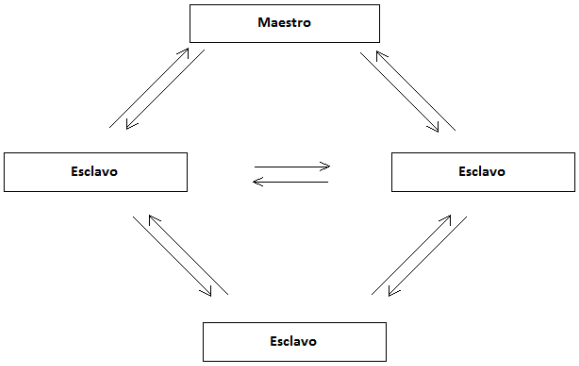


Figura 2.5: Comunicación multipunto.

2.1.2. El modelo de referencia OSI

El protocolo es, para dos dispositivos tratando de lograr una conexión, lo que es el idioma para dos individuos tratando de comunicarse. Al igual que en el idioma, el protocolo consta de ciertas reglas a seguir y se requiere una cierta estructura en las «oraciones». Podría decirse entonces que el modelo OSI es una especie de gramática para protocolos de comunicación.

El modelo OSI (Open Systems Interconnection) fue creado por ISO (International Standard Organization) y es un modelo de referencia utilizado para implementar protocolos de comunicación abiertos, esto quiere decir que las especificaciones pueden ser conocidas por el público. De tal forma que un sistema creado bajo este modelo se puede decir que es un «sistema abierto» y que cualquiera de los dispositivos que lo compongan, pueden ser reemplazados por un dispositivo de cualquier fabricante que cumpla con el modelo.

Funcionamiento

Este modelo consta de 7 capas. Cada capa realiza una función bien definida y puede intercomunicarse únicamente con las capas adyacentes a ésta. Esto le da ciertas ventajas sobre la evolución de las tecnologías en la comunicación ya que debido a la independencia entre capas, es posible sustituir una tecnología por otra sin tener que rediseñar el sistema entero. Por ejemplo, una capa física basada en la comunicación por medio de cables puede ser sustituida por dispositivos inalámbricos sin necesidad de modificar las capas superiores.

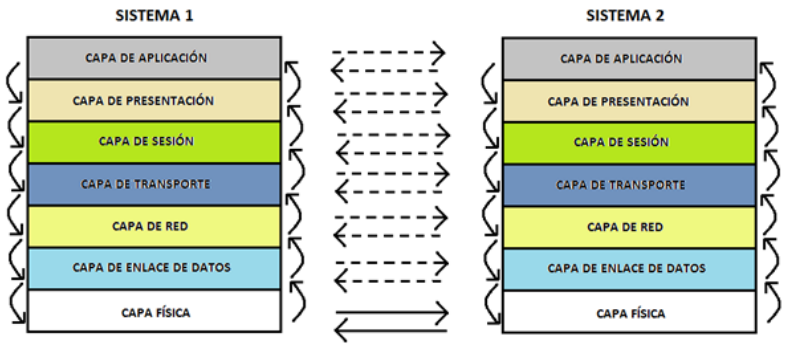


Figura 2.6: Dos sistemas utilizando el modelo de referencia OSI para comunicarse.

La comunicación entre dos sistemas y sus capas se da de la siguiente forma: El usuario 1 comunica algo a la capa de aplicación del sistema 1, y esta se comunica únicamente con la capa adyacente que es la de presentación, esta va pasando la información de manera descendente a través de las capas hasta llegar a la capa física. La capa física se encarga de transmitir la información mediante diversos métodos pudiendo ser eléctricos, ópticos, mecánicos, electromagnéticos, etc hacia la capa física del sistema 2. La información ascenderá a través de las capas hasta llegar a la capa de aplicación para ser presentada al usuario 2.

Un paradigma útil es asumir que las capas del mismo nombre se comunican directamente entre sí (aunque esto no sea verdad). En la imagen, el flujo de información real y la dirección de éste se representan con flechas sólidas mientras que el flujo de información viéndolo con el paradigma se representa con flechas punteadas. De esta forma, se podría ver que el usuario 1 se está comunicando directamente con el usuario 2 aunque en realidad está comunicándose con la capa de aplicación del sistema 1.

Cada capa del sistema 1 agrega una etiqueta a la información proporcionada por la capa superior y esta sólo le concierne y sólo podrá ser leída por la capa del mismo nombre en el sistema 2. Esta etiqueta contiene datos sobre de la capa en cuestión e información sobre la forma en que los datos serán transportados, además de los datos a transportar. Por ejemplo, supongamos que el usuario 1 quiere transmitir un *hola* al usuario 2 (como se muestra en la figura 2.7). La capa de aplicación agregará una etiqueta a *hola*. Esta etiqueta contiene información específica de la capa de aplicación del sistema 1 como por ejemplo el formato del dato *hola* y sólo tendrá significado para la capa de aplicación del sistema 2. La capa de aplicación pasará a la capa de sesión el dato más la etiqueta de la capa de aplicación. La capa de sesión recibirá esta información, sin embargo, no puede distinguir entre el dato original *hola* y la etiqueta de la capa de aplicación. Para ella estas dos representan un mismo dato. Para las siguientes capas se aplica el mismo procedimiento hasta llegar a la capa física. En el sistema 2 se realiza lo contrario ya que la información va ascendiendo. La capa física pasará la información a la capa de enlace de datos, esta leerá la etiqueta de enlace de datos la desprenderá del mensaje y pasara el resto a la capa superior. Lo mismo sucederá con todas las capas y de este modo, si no hubo ningún problema en la comunicación, el usuario 2 recibirá simplemente el dato *hola*.

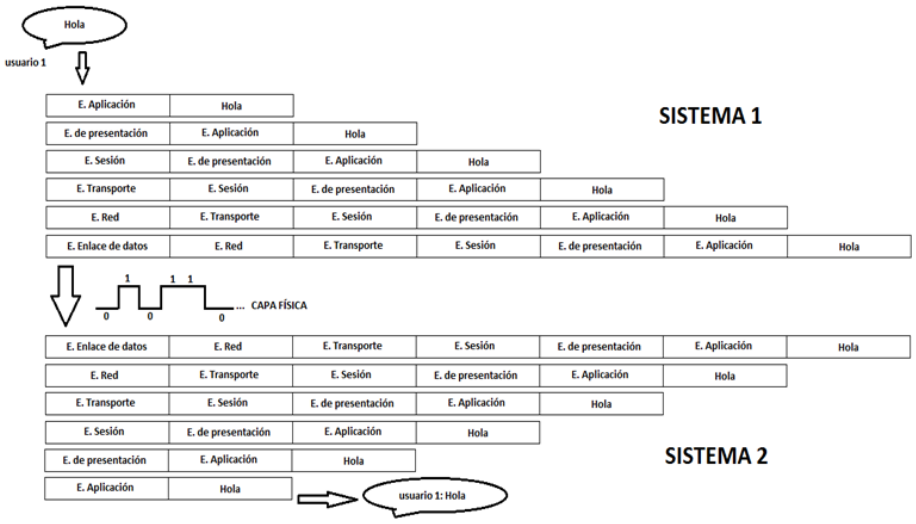


Figura 2.7: Etiquetas agregadas por cada una de las capas del modelo.

2.1.3. El modelo de referencia EPA

A principios de los noventas surge un nuevo modelo de referencia basado en el modelo OSI específicamente diseñado para sistemas SCADA. Fue a partir de aquí que se empezaron a forjar dos de los principales protocolos para la transmisión de datos en este tipo de sistemas: El DNP3 y el IEC 870 -5 - 101.

A diferencia del modelo OSI, EPA (Enhanced Performance Architecture) contiene únicamente tres capas.

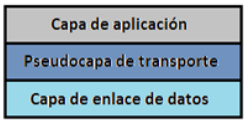


Figura 2.8: Capas del modelo EPA.

Las funciones de cada una de estas capas son:

1. Capa de aplicación: Se comunica con la pseudocapa de transporte y con una capa que no está contemplada dentro del protocolo DNP3 llamada interfaz del usuario. La capa de aplicación puede recibir órdenes de la interfaz de usuario, puede generar sus propias órdenes o recibirlas de la capa de aplicación de otro sistema. Genera conjuntos de datos llamados fragmentos los cuales son transmitidos hacia la pseudocapa de transporte. También puede ser en el sentido inverso: Recibe información de la pseudocapa de transporte, la verifica, la procesa y finalmente la envía a la interfaz de usuario.

2. Pseudocapa de transporte: Divide los fragmentos enviados por la capa de aplicación en

unidades llamadas segmentos y ensambla las unidades de información enviadas por la capa de enlace de datos en fragmentos.

3. Capa de enlace de datos: Recibe los segmentos y los convierte en unidades de datos llamadas tramas, agregándole un mecanismo de detección de errores CRC (Cyclic Redundancy Check), además de otros elementos. Se comunica con una capa física no contemplada en el modelo de referencia. Esta capa se encarga también del direccionamiento guardando la dirección del dispositivo en el que trabaja y la dirección del dispositivo con el que se comunica.

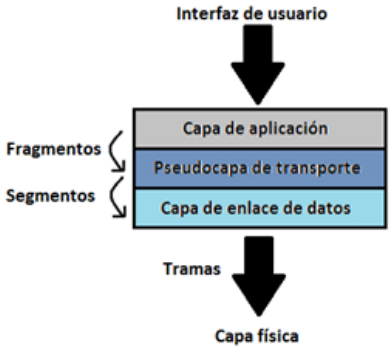


Figura 2.9: Unidades de datos de las capas del modelo EPA.

2.2. El protocolo DNP3

El protocolo de comunicación DNP3 (Distributed Network Protocol version 3) fue creado por Harris Controls Division con la intención de ser utilizado en el sector eléctrico. En 1993 le transmitió los derechos a *DNP3 User Group*, el cual le brinda soporte al protocolo desde entonces.

DNP3 surgió como una posible solución al problema que existía entre la comunicación de IEDs (Intelligent Electronic Device) de diferentes compañías y fue diseñado específicamente para sistemas SCADA. Anteriormente, cada compañía desarrollaba su propio protocolo «cerrado» de comunicaciones y los implantaba en sus dispositivos. A la hora de reemplazarlos, éstos debían ser del mismo fabricante o de lo contrarío un convertidor de protocolos sería necesario. Al ser el DNP3 un protocolo abierto, los fabricantes pueden conocer sus especificaciones y desarrollar equipo de acuerdo a ellas, de esta forma, se incrementa la interoperabilidad y se eliminan los problemas que los protocolos cerrados representan.

2.2.1. Características

El protocolo DNP3 permite la implementación de las siguientes estructuras de comunicación:

1. Maestro-esclavo.
2. Maestro con múltiples esclavos.
3. Múltiples maestros.
4. Maestros intermediarios.

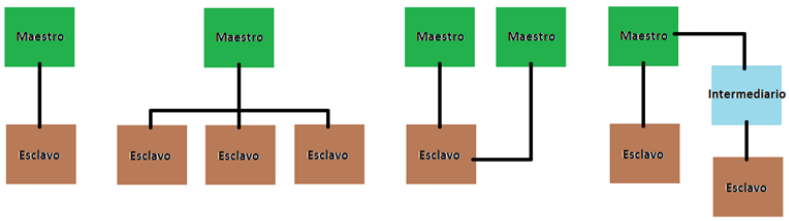


Figura 2.10: Topologías permitidas por el protocolo.

A cada dispositivo DNP3 conectado a una red debe de asignársele una dirección única. Esta puede ir desde el 0 hasta el 65 536. Lo que significa que en una red SCADA con DNP3 pueden conectarse un máximo de 65 537 dispositivos.

La manera en la cual un maestro solicita la información de los esclavos puede de las siguientes formas:

1. Polling: El maestro solicita cierta información a un esclavo determinado. Si el maestro no solicita nada, el esclavo no debe mandar nada. La desventaja de esta técnica es que se hace un mayor uso del ancho de banda de la red, además de que el maestro debe estar programado para estar solicitando la información cada cierto tiempo.
2. Respuestas no solicitadas: El esclavo manda información acerca de un evento importante ocurrido sin que el maestro la haya solicitado. Esto reduce considerablemente el uso del ancho de banda.

Para lograr una comunicación sólida es necesaria una combinación de las dos formas anteriores.

Interoperabilidad

El protocolo es tan extenso que desarrollarlo completamente en todos los dispositivos sería innecesario. Es por esto que el estándar define 4 niveles de implementación, cada uno con un nivel de complejidad mayor al anterior. La regla es que un dispositivo esclavo de nivel x tiene que ser controlado por un dispositivo maestro de nivel $x+1$, por lo que el primer nivel está destinado únicamente para sistemas esclavos sencillos. Esta diferenciación se hace evidente sólo en la capa de aplicación ya que tanto las capas de transporte como la de enlace de datos deben ser prácticamente iguales en todos los dispositivos.

2.2.2. Arquitectura del protocolo

DNP3 está compuesto de tres capas ya que está basado en el modelo EPA. A continuación se presenta una descripción de cada una de las capas incluyendo la capa física, la cual no está especificada por DNP3 por lo que queda a elección del implementador. El propósito de las siguientes descripciones no es el de explicar el protocolo, ya que éste es muy extenso, sino dar las bases que se requieren para el buen entendimiento de los capítulos posteriores.

2.2.3. La capa física

Esta capa especifica el medio y el método de transmisión de la información que se genera en la capa de enlace. De la capa física dependerán las estructuras de comunicación que se podrán implementar. Por ejemplo, una capa física basada en Ethernet dará la posibilidad de crear una estructura de comunicación del tipo maestro con múltiples esclavos y una capa física basada en RS-232 quedaría limitada a estructuras del tipo maestro - esclavo (ver figura 2.10). Una opción interesante es encapsular el protocolo DNP3 dentro de la pila de protocolos TCP/IP ya que de esta forma se podría tener acceso a dispositivos DNP3 desde internet, no obstante, para los fines del proyecto solo se hablará de RS-232.

Existen diversas formas de transmitir información mediante señales eléctricas. Estas consisten en asignar un valor de alguna variable eléctrica a un determinado dato. Por ejemplo, para transmitir ceros o unos se podría asignar un valor de 0 [v] para el cero y 5 [v] para el 1. También podría transmitirse mediante una señal sinusoidal que estuviera variando su frecuencia o su fase y de esta manera representar el 1 ó el 0.

Dentro de la forma de transmisión que utiliza la asignación de tensiones existen dos métodos distintos:

1. Paralelo: Se utilizan varios canales para transmitir valores de tensión al mismo tiempo.
2. Serie: Utiliza un sólo canal para transmitir valores de tensión de manera secuencial. Se subdivide en:
   a) Serial síncrono: Es necesario un reloj para coordinar el envío de datos. Por lo general, este método se utiliza para comunicar dispositivos que se encuentran relativamente cerca.
   b) Serial asíncrono: Los dispositivos se configuran previamente a una determinada velocidad de envío y no es necesario una señal de reloj para coordinar la comunicación. La información se envía generalmente en paquetes de 8 bits (1 byte) pero es necesario un bit de inicio, el cual le indica al dispositivo receptor que hay una transmisión en progreso, y un bit de parada que indica el final de la transmisión, por lo que el tamaño total del paquete de información es de 10 bits. Optativamente se cuentan con mecanismos de seguridad como el bit de paridad y mecanismos de control de flujo, aunque esto significa aumentar la cantidad de bits del paquete de información.

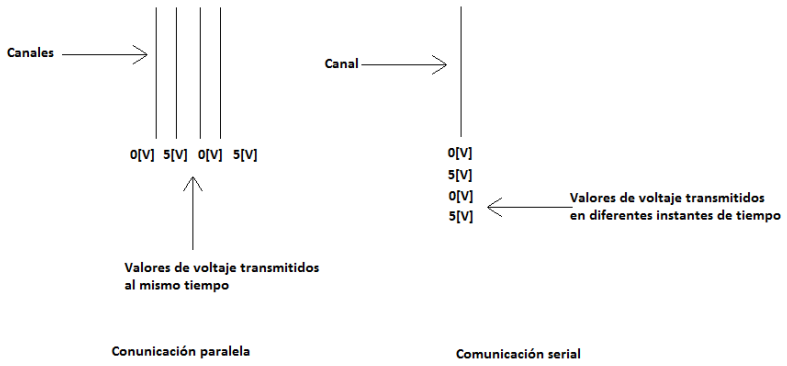


Figura 2.11: Diferencias entre comunicación paralela y serial.

En la imagen inferior se muestra el envío de un paquete de información asumiendo que se ha asignado una tensión de 0 [V] para transmitir el 0, 5 [V] para el 1, que el bit de inicio es siempre un 0 y que el de parada es un 1.

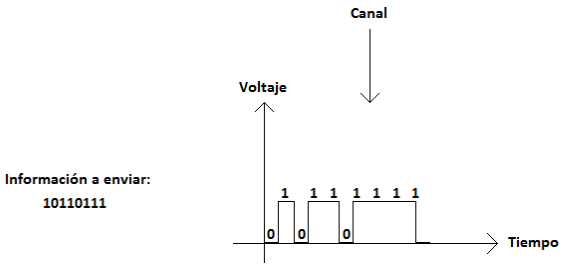


Figura 2.12: Transmisión de un paquete de información.

La comunicación serial también puede subdividirse en los siguientes tipos:

1. Simplex: La comunicación entre dos dispositivos es de un sólo sentido.

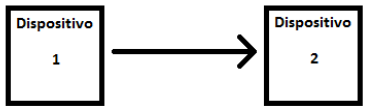


Figura 2.13: Comunicación serial tipo simplex.

2. Semi - dúplex: La comunicación entre dos dispositivos es en ambos sentidos pero sólo se puede dar uno a la vez.

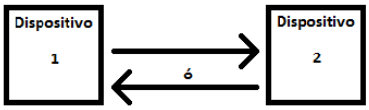


Figura 2.14: Comunicación serial tipo dúplex.

3. Full dúplex: La comunicación entre dos dispositivos es en ambos sentidos a la vez.

El estándar RS-232

RS-232 define la interfaz entre un DTE y un DCE empleando el intercambio serial de datos binarios. No obstante, se puede realizar la conexión entre dos dispositivos DTE sin recurrir a los DCE mediante un artificio llamado NULL-MODEM.

Las especificaciones de este estándar que serán utilizadas en el proyecto son las siguientes:

1. Nivel de tensión de las señales: En la tabla 2.1 se presentan las tensiones utilizadas en el estándar RS-232 para definir el «0» y el «1» lógicos y se comparan con las tensiones utilizadas en circuitos TTL.

Nivel lógico	Tensión RS-232 [V]	Tensión TTL [V]
0	+3 a +15	0 a 0.8
1	-3 a -15	2 a 5

Cuadro 2.1: Niveles de tensión RS-232 y TTL.

2. Las características físicas de la interfaz: El estándar utiliza 11 señales de las cuales únicamente 3 de ellas son necesarias para llevar a cabo una comunicación serial simple.

   a) Transmisión.

   b) Recepción.

   c) Común.

   Aunque el estándar no lo especifica, es común la utilización de los conectores DB-9 en la interfaz. La disposición de los pines del conector DB-9 hembra es la de la figura 2.15.

2.2.4. La capa de enlace de datos

La comunicación entre capas de enlace de datos es una comunicación balanceada porque cualquiera de ellas puede iniciar dicha comunicación, no importando si esta pertenece a una estación esclavo o a una estación maestra. Por esta razón, a este nivel no se hablará de maestros o esclavos sino de primarios o secundarios.

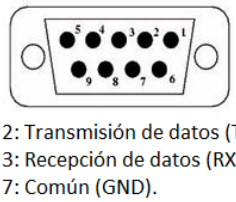


Figura 2.15: Conector DB-9 hembra.

Un primario es aquella capa de enlace que inicia una transmisión y el secundario es el que la recibe y en algunos casos puede generar una respuesta.

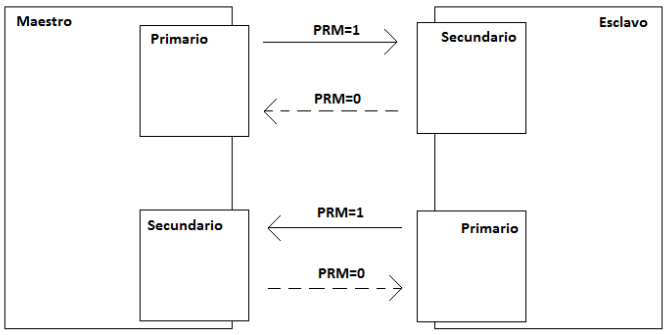


Figura 2.16: Estaciones primarias y secundarias.

La unidad de datos con la que trabaja esta capa es la trama, cuyo tamaño máximo es de 292 bytes:

Figura 2.17: Estructura de una trama.

1. Inicio: Constituido por dos bytes: 0x0564.
2. Largo o *Lenght*: Un sólo byte que representa la longitud de la trama contando a partir del byte de control y sin contar los CRC.
3. Control: El byte de control contiene información sobre la condición de la capa de enlace que emitió el mensaje y sobre ciertas acciones que se deben llevar a cabo para mantener la sincronía y evitar que la comunicación no falle. Este byte será examinado con más detalle.
4. Dirección destino: Dos bytes que contienen la dirección del dispositivo al que va dirigido el mensaje. En caso de que la trama haya arribado, el dispositivo tendrá que verificar que

la dirección de destino sea la suya. El orden de los bytes es invertido, es decir, primero se coloca el byte menos significativo y luego el más significativo (LSB, MSB). Por ejemplo, una dirección destino = 1 se vería en la trama de la siguiente manera: 0x0100.

5. Dirección remitente: Dos bytes que contienen la dirección del remitente. El orden de los bytes es el mismo que el de la dirección destino (LSB, MSB).

6. Datos: El segmento de la pseudocapa de transporte es dividido en grupos de 16 bytes. Sólo el último dato de la trama puede contener un número de bytes distinto. Por ejemplo, si el segmento consistía de 105 bytes, entonces se crearán 6 grupos de 16 bytes y un grupo de 9 bytes.

7. CRC (Cyclic Redundancy Check): Cada CRC consta de dos bytes acomodados de manera similar a los bytes de dirección (LSB, MSB). Estos bytes son el resultado de la aplicación de un algoritmo a cada uno de los bloques de datos y a los bytes correspondientes a inicio, largo, direcciones de remitente y destino juntos. El primer CRC corresponde al los bytes de inicio, largo, control y direcciones, el segundo CRC corresponde al primer bloque de datos, el tercer CRC al segundo bloque de datos y así sucesivamente.

   Cuando una trama es recibida, los valores de CRC para cada bloque de datos y para los bytes de inicio, largo, control y direcciones son re calculados y comparados con los CRC recibidos. Si algún CRC re calculado llega a ser distinto al recibido la trama será descartada ya que contiene errores.

El byte de control de la capa de enlace de datos se divide en:

Figura 2.18: Byte de control.

1. DIR: Consta de un bit. Indica si la trama proviene o será enviada desde una estación maestra (1) o de un esclavo (0).

2. PRM: Consta de un bit. Indica si la trama proviene o será enviada desde una estación primaria (1) o una estación secundaria (0).

3. FCV o DFC: En caso de que la trama sea enviada de una estación primaria este bit toma el nombre de FCV. Si es igual a 1 significa que el valor del bit FCB debe ser verificado pero si es 0 entonces se ignora el valor de FCB.

   Cuando la trama es enviada por un secundario, entonces el bit es llamado DFC y tiene la función de comunicar el estado de los buffers de llegada del secundario (1 para buffers desbordados y 0 para todos los demás casos).

4. FCB: Se utiliza para detectar mensajes duplicados o pérdida de mensaje. El dispositivo que recibe la trama compara el valor del FCB de ésta con el valor del FCB esperado. Si

coinciden, la trama será correcta. En caso contrario se presentará un error. Si el bit FCV no es utilizado entonces el valor de FCB deberá ser siempre 0.

5. FC (Function Code o Código de Función): Consta de 4 bits. Representa la acción que debe realizar un dispositivo secundario cuando recibe una trama.

Código de función en decimal	Nombre de la función
0	RESET_LINK_STATES
2	TEST_LINK_STATES
3	CONFIRMED_USER_DATA
4	UNCONFIRMED_USER_DATA
9	REQUEST_LINK_STATUS

Cuadro 2.2: Códigos de función enviados por un primario.

Los secundarios pueden generar tramas en respuesta a los códigos de función de la tabla 2.2:

Código de función en decimal	Nombre de la función
0	ACK
1	NACK
11	LINK_STATUS
15	NOT_SUPPORTED

Cuadro 2.3: Códigos de función enviados por un secundario.

Direcciones públicas

Las direcciones 0xFFFF, 0xFFFE y 0xFFFD tienen un significado especial. Cuando la capa de enlace detecta que la dirección de destino es alguna de las anteriores debe aceptar la trama como si estuviera dirigida al dispositivo en cuestión. Estas direcciones las utilizan los maestros para enviar el mismo mensaje a todos los dispositivos esclavos que tenga conectados.

De alguna forma y rompiendo un poco con el esquema EPA en el cual está basado el protocolo, la capa de enlace de datos debe ser capaz de comunicar a la de aplicación si el mensaje recibido contenía una dirección pública, ya que ésta última capa procesa de manera distinta este tipo de mensajes.

Algoritmo del CRC

Los dos bytes del CRC se calculan mediante el siguiente procedimiento:

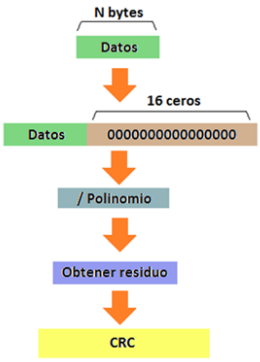


Figura 2.19: Cálculo del CRC.

Dado un dato, se le agregan 16 ceros y se efectúa la división entre el siguiente polinomio generador:

$$P = x^{16} + x^{13} + x^{12} + x^{10} + x^8 + x^5 + x^2 + 1$$

El cociente de la operación es descartado y el residuo es el código CRC.

2.2.5. La pseudocapa de transporte

Se encarga de tomar los fragmentos (unidad de datos de la capa de aplicación cuya extensión puede ser de hasta 2048 bytes), dividirlos en grupos de hasta 249 bytes (el protocolo DNP3 deja a libre elección el tamaño en el que se dividen los fragmentos) y agregarles un byte de control. A esta unidad de datos se le llama segmento y tiene una extensión máxima de 250 bytes.

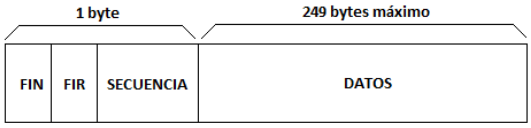


Figura 2.20: Estructura de un segmento.

Para reensamblar los segmentos, la capa hace uso del byte de control que contiene a FIR, FIN y SECUENCIA:

1. FIR y FIN: Identifican a el primer y último segmento que forman parte de un fragmento (ver tabla 1).

FIN	FIR	Descripción
0	0	No es el primer segmento ni el último.
0	1	Primer segmento de la serie.
1	0	Último segmento de la serie.
1	1	La serie consta de un sólo segmento.

Cuadro 2.4: Significado de FIR y FIN.

2. SECUENCIA: Se utiliza para evitar que se repitan o se omitan segmentos al enviar una serie de éstos. El primer segmento de la serie toma cualquier número entre 0 y 63 y se va incrementando en cada segmento enviado. Cuando se llega al 63, el siguiente número de secuencia será 0.

2.2.6. La capa de aplicación

Ésta capa es la que diferencia a todos los dispositivos y evidencia su funcionalidad. Puede estar implementada en diferentes niveles (interoperabilidad) y para diferentes tipos de dispositivo (maestros o esclavos). Su función es generar peticiones, recibirlas, procesarlas y generar respuestas. Esto lo hace mediante unidades de datos llamadas fragmentos, los cuales pueden tener un largo de hasta 2048 bytes, dependiendo de las capacidades de los dispositivos. Dichos fragmentos deben ser procesables por sí mismos, es decir, que en un fragmento debe de estar disponible toda la información para que pueda ser procesado y no debe requerir de otros fragmentos para llevar a cabo tal acción. Puede llegar a darse el caso de que se genere tanta información que no cabe en un fragmento. En este caso dicha información se divide en múltiples fragmentos, sin embargo, cada uno sigue siendo independiente.

Para poder entender la estructura del fragmento es necesario conocer los conceptos que se presentan a continuación.

Tipo de datos

Una de las funcionalidades del protocolo es la capacidad de reportar información por excepción. Esto quiere decir que dicha información contendrá únicamente los cambios ocurridos en el sistema y no el estado del sistema entero. Para esto es necesario dividir los datos en dos tipos:

1. Datos estáticos: Representan el estado actual del sistema.
2. Eventos: Representan los cambios ocurridos en el sistema. Estos cambios pueden ser de importancia crítica o pueden ser completamente irrelevantes, por lo que se subdividen en 4 clases cuyo nivel de importancia es especificado por el usuario. Cuando le es solicitada la información de una clase al sistema, éste debe incluir los puntos asignados a las diferentes clases de acuerdo a la tabla 2.5.

   Nota: Por regla, todos los datos estáticos deben estar asignados a la clase 0. Las clases 1,2 y 3 están reservadas únicamente para datos de tipo evento.

	Clase 0	Clase 1	Clase 2	Clase 3
Solicitud de clase 0	Si	Si	Si	Si
Solicitud de clase 1	No	Si	No	No
Solicitud de clase 2	No	No	Si	No
Solicitud de clase 3	No	No	No	Si

Cuadro 2.5: Respuesta del sistema a la solicitud de las clases existentes.

Existen muchos otros tipos de datos pero se omite su descripción ya que para los alcances del proyecto sólo son necesarios los dos anteriores.

Puntos, tipos de punto, grupos, variaciones, índices y objetos

1. Punto: Es una entidad física o lógica identificable. Por ejemplo, cada una de las entradas del sistema de alarmas representa un punto. Los puntos pueden estar asignados a uno o varios tipos de datos.

2. Tipos de puntos: Sirven para categorizar a los diferentes puntos de un dispositivo. Por ejemplo, las entradas del sistema de alarmas son del tipo entradas binarias ya que únicamente pueden tomar dos valores. Existen muchos otros tipos de puntos como las salidas binarias, entradas analógicas, salidas analógicas, entre otros.

3. Índices: Sirven para identificar a cada uno de los puntos pertenecientes a un grupo de puntos. Por ejemplo, las 5 entradas binarias del sistema pueden identificarse con los índices 1,2,3,4 y 5.

4. Grupos: Sirven para clasificar a los diferentes puntos dentro de un fragmento. Cada grupo contiene un tipo de punto referido a un tipo de dato específico. Por ejemplo, en el sistema de alarmas podría contarse con dos grupos: Las entradas binarias referidas a datos estáticos y las referidas a eventos.

5. Variaciones: Son las diferentes formas de representar a un grupo. Por ejemplo, una entrada analógica que pertenece al grupo x puede ser representada con un número entero de 16 bytes o un flotante de 32 o de 64 bytes.

6. Objeto: Es la representación de los datos de un punto en un fragmento con el formato correspondiente de acuerdo al grupo y variación al que pertenece.

El grupo 0

La función del grupo cero es proporcionar información específica del dispositivo DNP3. Cada una de sus variaciones representa un atributo distinto como el nombre del dispositivo, el software instalado, la versión de software, el nombre del hardware, número de entradas analógicas, número de entradas binarias, fabricante, entre otros. Dicha información es utilizada por el dispositivo maestro para configurarse a si mismo de acuerdo con los atributos del esclavo.

El grupo 1

El grupo 1 con todas sus variaciones agrupa datos estáticos de puntos tipo entrada binaria. Cada una de sus variaciones permite representar la entrada de maneras distintas, sin embargo, en el sistema de alarmas solo se utilizará la variación 1. Ésta a diferencia de otras variaciones, únicamente comunica el valor actual de dicha entrada al momento de recibirse y procesarse la solicitud pudiendo ser 0 ó 1.

El grupo 2

El grupo 2 agrupa eventos de puntos tipo entrada binaria. Mediante la variación 1, permite comunicar el cambio de estado de alguna de las entradas binarias del sistema.

El grupo 52

El grupo 52 permite comunicar intervalos de tiempo.

El grupo 60

EL grupo 60 es un caso especial ya que está diseñado para la solicitud de datos que están asignados a alguna clase. Dicho grupo tiene 4 variaciones: La variación 0 indica objetos de clase 1, la variación 2 objetos de clase 1, la 3 objetos de clase 2 y la 4 objetos de clase 3. Sólo los maestros pueden incluirlo en sus mensajes pues los esclavos deben de utilizar objetos que hagan referencia a los tipos de puntos. Por ejemplo, suponga que un sistema de varias entradas binarias, cuyos eventos están asignados a la clase 2, recibe un mensaje donde se le solicita el valor de los elementos clase 2 (grupo 60, variación 3). Éste deberá responder con objetos del grupo 2 y no del grupo 60.

El grupo 80

Este grupo guarda la información de algunos estados de la capa de aplicación y del sistema. Dicha información corresponde con los valores de las indicaciones internas (IIN).

Estructura del fragmento

La estructura del fragmento generado por un esclavo difiere ligeramente de la del maestro ya que la de estos últimos carece de una sección llamada *Indicaciones internas*. En la figura 2.21 se muestran las partes del fragmento generado por un esclavo:

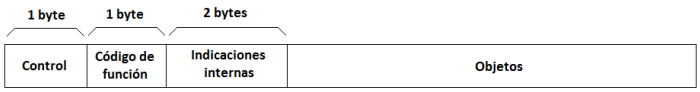


Figura 2.21: Estructura del fragmento generado por un esclavo.

1. Control: En la figura 2.22 se muestran los bits que lo componen.

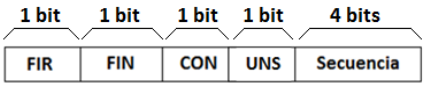


Figura 2.22: Byte de control del fragmento.

   a) FIR y FIN: Su funcionalidad es similar a los de la capa de transporte. Identifican el primer y el último fragmento dentro de una serie de fragmentos (ver tabla 1).

   b) CON: Indica si el fragmento requiere una confirmación de recibido (1 = con confirmación, 0 = sin confirmación).

   c) UNS: Toma el valor de 1 únicamente cuando el fragmento es una respuesta no solicitada.

   d) Secuencia: Se utiliza para evitar que se repitan o se omitan fragmentos al enviar una serie de éstos. El primer fragmento de la serie toma cualquier número entre 0 y 63 y se va incrementando en cada fragmento enviado. Cuando se llega al 15, el siguiente número de secuencia será 0.

2. Código de función: Consta de un byte en el que se indica la operación que se debe realizar con las etiquetas de objetos del fragmento. Existen dos grupos:

   a) Códigos de función de dispositivos maestro: El esclavo debe generar los objetos correspondientes o realizar acciones sobre algún punto del sistema. Algunos ejemplos son: *READ* (leer), *WRITE* (escribir), *COLD_RESTART*(reiniciar).

   b) Códigos de función de dispositivos esclavo: Son enviados por el esclavo en respuesta a una acción que se llevó a cabo. Por ejemplo, si un esclavo recibe un código *READ*, éste generará una respuesta con el código *RESPONSE* (respuesta), *UNSOLICITED RESPONSE* o *CONFIRM*.

En la figura 2.23 se muestra una tabla de los códigos de función utilizados en este proyecto.

Message type	Code	Name	Brief description
Confirmation	0 0x00	CONFIRM	Confirm Function Code: Master sends this to an outstation to confirm the receipt of an Application Layer fragment. Reference: 4.4.1
Request	1 0x01	READ	Read Function Code: Outstation shall return the data specified by the objects in the request. Reference: 4.4.2
Request	2 0x02	WRITE	Write Function Code: Outstation shall store the data specified by the objects in the request. Reference: 4.4.3
Request	13 0x0D	COLD_RESTART	Cold Restart Function Code: Outstation shall perform a complete reset of all hardware and software in the device. Reference: 4.4.9
Request	20 0x14	ENABLE_UNSOLICITED	Enable Unsolicited Responses Function Code: Enables outstation to initiate unsolicited responses from points specified by the objects in the request. Reference: 4.4.13
Request	21 0x15	DISABLE_UNSOLICITED	Disable Unsolicited Responses Function Code: Prevents outstation from initiating unsolicited responses from points specified by the objects in the request. Reference: 4.4.13
Response	129 0x81	RESPONSE	Solicited Response Function Code: Master shall interpret this fragment as an Application Layer response to an Application Layer request sent by the master. Reference: 4.4.20
Response	130 0x82	UNSOLICITED_RESPONSE	Unsolicited Response Function Code: Master shall interpret this fragment as an unsolicited response that was not prompted by an explicit request. Reference: 4.4.24

Figura 2.23: Funciones de la capa de aplicación utilizadas.
[1]

3. Indicaciones internas (IIN): 16 bits correspondientes a 16 indicaciones distintas que contienen el estado de ciertos aspectos de la capa de aplicación y del sistema. A continuación se presentan únicamente las indicaciones que se usan en el sistema de alarmas (se puede prescindir del resto).

   a) *ALL_STATION*: Se activa cuando se recibe un mensaje con una dirección pública.
   b) *CLASS_1_EVENTS*: Eventos clase 1 se encuentran disponibles.
   c) *CLASS_2_EVENTS*: Eventos clase 2 se encuentran disponibles.
   d) *CLASS_3_EVENTS*: Eventos clase 3 se encuentran disponibles.
   e) *DEVICE_RESTART*: El dispositivo acaba de reiniciarse.
   f) *NO_FUNC_CODE_SUPPORT*: El dispositivo no implementa el código de función especificado en el fragmento.
   g) *OBJECT_UNKNOWN*: El dispositivo no implementa uno de los objetos del fragmento.
   h) *PARAMETER_ERROR*: Error de parámetro.
   i) *EVENT_BUFFER_OVERFLOW*: El buffer de eventos está lleno.

4. Objetos: Esta sección, al igual que la de indicaciones internas, difiere entre ambos tipos de dispositivo:

a) Maestro: La sección de objetos contendrá únicamente el encabezado de los objetos. Por ejemplo, si el maestro quiere leer el valor de las entradas binarias coloca únicamente el encabezado de dicho grupo.

b) Esclavo: Puede llegar a contener el encabezado de objetos y su valor. En respuesta al ejemplo anterior, el esclavo genera un fragmento con el encabezado del objeto de las entradas binarias seguido del valor de cada una de ellas.

El encabezado del objeto

La estructura del encabezado de los objetos se muestra en la figura 2.24.

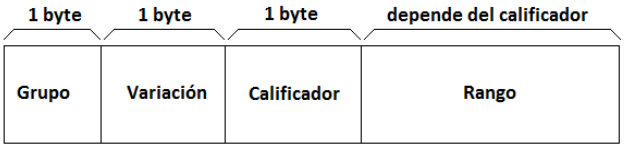


Figura 2.24: Encabezado del objeto.

El calificador y el rango hacen referencia a la forma en que serán acomodados los objetos dentro del fragmento.

1. Calificador: Este byte se subdivide a su vez en dos:

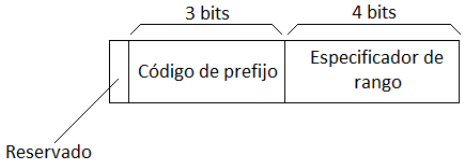


Figura 2.25: Calificador.

a) Código de prefijo: Indica si los objetos serán precedidos por algún prefijo al momento de ser acomodados dentro del fragmento:

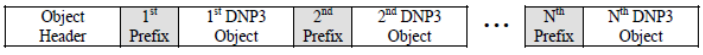


Figura 2.26: Objetos acomodados con prefijo.
[1]

b) Especificador de rango: Indica si se utiliza la sección de *Rango* y la extensión de ésta. Los especificadores utilizados en el proyecto se muestran en la tabla 2.7:

Código de prefijo	Descripción	Tamaño de prefijo (bytes)
0	Sin prefijo	0
1	El prefijo del objeto corresponde a su índice.	1

Cuadro 2.6: Código de prefijos utilizados en el proyecto.

Código del especificador	Descripción	Número de bytes de la sección *Rango* (bytes)
0	La sección de rango contiene un índice de inicio y uno de parada de 1 byte c/u.	2
1	La sección de rango contiene un índice de inicio y uno de parada de 2 bytes c/u.	4
6	Implica todos los valores (sin rango).	0
7	El rango contiene una cuenta de objetos de 1 byte, cada uno precedido por su índice.	1
8	El rango contiene una cuenta de objetos de 2 bytes, cada uno precedido por su índice.	2

Cuadro 2.7: Especificadores de rango utilizados en el proyecto.

La librería de objetos

La libreria de objetos del protocolo contiene información acerca de cómo deben empaquetarse los diferentes objetos que manejan los dispositivos DNP3. En la sección de apéndices se encuentra una descripción detallada del empaquetamiento de los objetos que maneja el sistema de alarmas.

Capítulo 3

Diseño del software

3.1. ¿Programación en un hilo o multi hilo?

La separación del protocolo DNP3 en diferentes capas independientes facilitó enormemente el trabajo de la planeación y estructuración del software. Sin embargo, las especificaciones del protocolo dan a entender que todas sus capas deben estar activas siempre, de tal forma que puedan estar interactuando al mismo tiempo. Esto es difícil de lograr ya que en un principio, los programas en Arduino se ejecutan en un sólo hilo (una acción a la vez), lo que implica que sólo se puede estar ejecutando una capa en un momento dado.

Con las herramientas necesarias es posible crear un programa multi hilo, de tal forma que funcione como un sistema operativo en donde aparentemente se ejecutan diferentes acciones en un mismo tiempo. Librerías como *Plumbing* o *Pyxis OS* nos permiten crear este tipo de aplicaciones pero a un precio elevado ya que además de tener que escribir código en otro lenguaje que no es C (OCCAM - PI para Plumbing), no está bien documentado el uso que hacen de los recursos del microprocesador, además de que son herramientas nuevas en estado experimental y por lo tanto no existe mucha información acerca de ellas.

Después de explorar éstas herramientas se optó por la programación de un sólo hilo, con la desventaja mencionada anteriormente.

3.2. Esquema general del programa

El diagrama de flujo de la figura 3.1 muestra la estructura general del programa. La primera acción que se toma es la de configurar los módulos del microprocesador utilizados. Después puede elegirse pasar a un modo de configuración, en donde se ajustan algunos parámetros del protocolo. Posteriormente se inicializan los parámetros y se entra en un loop infinito en donde se están monitoreando continuamente las entradas del sistema y ejecutándose secuencialmente las diferentes capas.

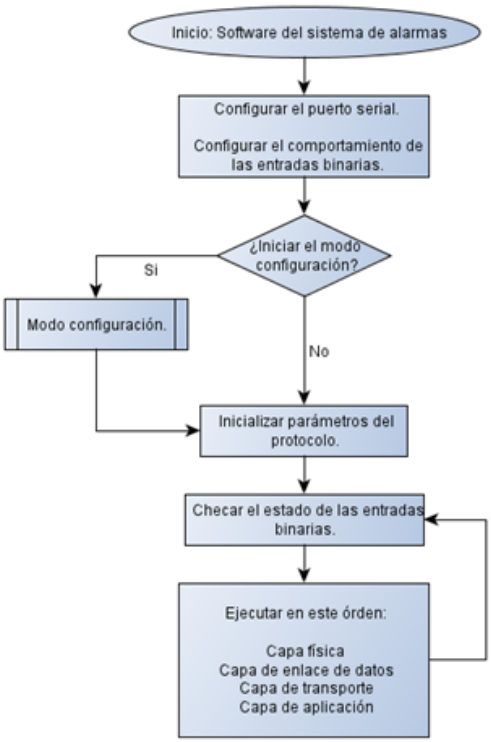


Figura 3.1: Diagrama de flujo del software del protocolo.

3.3. Las capas

Siguiendo la filosofía de los modelos OSI y EPA, en donde una capa interactúa con la capa adyacente se diseñó el siguiente diagrama al que se le llamará *Diagrama de capas* por el resto del documento. En el diagrama se pueden apreciar tres elementos:

1. Las capas, que están representadas por una línea transversal, son en sí un proceso, una serie de instrucciones que necesitan información para operar.
2. Los buffers, que están representados como un recuadro, son esa información que las capas necesitan.
3. El tiempo.

El buffer a la derecha de una capa representa la información proveniente de la capa inferior y el de la izquierda la información proveniente de la capa superior. Cada buffer tiene asignado un nombre que hace referencia a la capa de procedencia y la capa destino. Por ejemplo el buffer CF_CED contiene información proveniente de la capa física y que debe ser procesada por la capa de enlace de datos.

El estado de los buffers a través del tiempo se representa como un punto negro para *lleno* o en blanco para *vacío*. En el diagrama a continuación, se observa el camino que sigue una

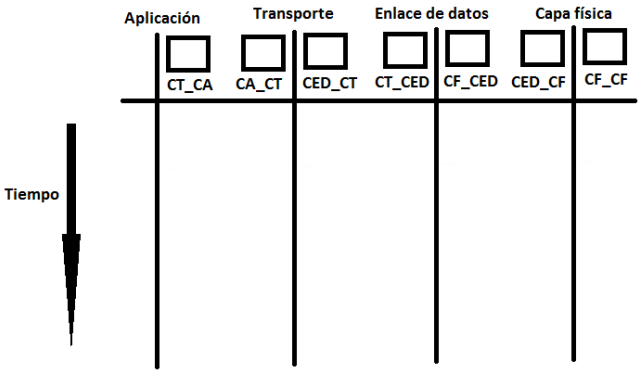


Figura 3.2: Diagrama de capas.

petición recibida por la capa física del sistema proveniente de la capa física de otro sistema. Una vez que llega a la capa de aplicación, se generará una respuesta que viajará en sentido contrario.

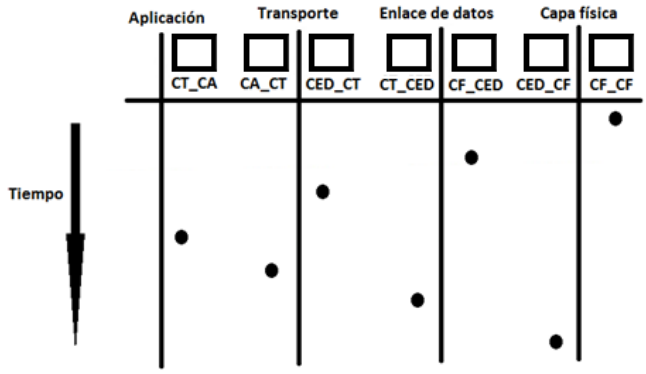


Figura 3.3: Petición y respuesta.

3.4. Los buffers

Debido a las limitaciones en memoria RAM que se tienen en un microprocesador como el ATmega328 (2048 bytes), se pensó utilizar arreglos dinámicos en lugar de estáticos. Esto para no desperdiciar la memoria ya que como se observa en el diagrama, al estarse ejecutando una capa sólo son necesarios 2 buffers. Sin embargo, esta decisión genera otro tipo de problemas que se verán cuando se aborde el tema del manejo de la memoria RAM.

Los buffers son una estructura de 3 elementos que facilitan el manejo de los mismos por parte de las capas.

1. Tamaño.
2. Estado.
3. Apuntador hacia el inicio del buffer.

También cuentan con el método *reset_buffers()*, el cual sirve para vaciar la información de un determinado buffer.

El único buffer que no tiene esta estructura es CF_CF ya que en realidad se trata del buffer de entrada del puerto serie de Arduino.

3.5. Configuración del puerto serial

La capa física hace uso del módulo USART del microprocesador, el cual fue configurado para recibir y transmitir información a 9600 bauds por segundo (utilizando la función *Serial.begin()*), con un bit de stop, sin bit de paridad y 8 bits de información, por lo que cada paquete de información de RS-232 constará de 10 bits.

3.6. Las entradas binarias

Las entradas binarias del protocolo hacen uso de los puertos de entrada / salida del microprocesador. Por medio de las funciones *pinMode()* y *digitalWrite()* se configuran 5 de estos puertos como entradas digitales.

Estas entradas son manejadas por la función *checar_estados()*, la cual actualiza los estados de las 5 entradas y compara los valores anteriores con los actuales para detectar cambios con ayuda de la función *digitalRead()*. En caso de detectarse un cambio, éste es comunicado a la capa de aplicación para la generación de los eventos correspondientes.

3.7. Modo configuración

El protocolo DNP3 deja libre al implementador la elección de ciertos parámetros y éste los determina dependiendo de la aplicación. En el sistema de alarmas, dichos parámetros se hacen libres para el usuario, el cual los seleccionará de acuerdo al tipo de dispositivos con los que se conecte, los canales de comunicación que utilice, el tipo de sistema que se requiera implementar, entre otras variantes.

1. Dirección local: Representa la dirección del sistema de alarmas. Puede variar entre 0 y 65 537.

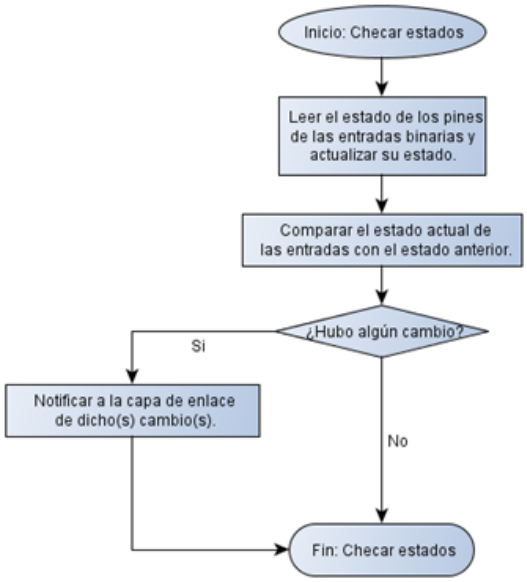


Figura 3.4: Diagrama de flujo de la función *checar_estados()*.

2. Dirección del maestro: Representa la dirección del dispositivo maestro con el cual estará conectado el sistema de alarmas. El rango de direcciones es el mismo que el de la dirección local.

3. Confirmación a nivel de la capa de enlace de datos: Como se verá posteriormente, la capa de enlace de datos puede enviar información sin confirmar o pidiendo una confirmación. Si se utiliza este último método, el intercambio de información se vuelve más seguro pero más lento. En el caso de que se utilice el protocolo TCP/IP como medio de transporte de la información, el uso de confirmaciones a este nivel se vuelve redundante.

4. Número de reenvíos: Cuando se utiliza la confirmación a nivel de enlace de datos, es necesario especificar el número de veces que se reenviará una trama en caso de que ésta no sea confirmada.

5. Tiempo de espera de la capa de enlace: Si se utiliza la confirmación a nivel de enlace de datos, este parámetro especifica el tiempo máximo que el sistema de alarmas esperará una confirmación.

6. Habilitación de respuestas no solicitadas: El usuario puede activar o desactivar el envío de respuestas no solicitadas mediante este parámetro. Se recomienda desactivar esta opción si en la comunicación no se cuenta con algún mecanismo de control de flujo.

7. Tiempo máximo de espera de la capa de aplicación: La capa de aplicación del sistema de alarmas siempre solicitará una confirmación después del envío de información. Este parámetro determina el tiempo máximo de espera de dicha confirmación.

8. Especificación de la clase de cada una de las entradas binarias: Este parámetro permite

asignar una clase de evento (ya sea 1, 2, 3 o no asignado) a cada una de las entradas del sistema.

Debido a que el sistema de alarmas puede reiniciarse varias veces no es conveniente que el usuario deba configurar estos parámetros cada vez que esto ocurra. Es por esto que serán guardados en la memoria EEPROM del microprocesador y podrán ser accedidos y modificados a través del modo configuración.

Para entrar al modo configuración, el sistema de alarmas debe estar conectado a una computadora por el puerto serie, pudiendo ser un puerto USB si se utiliza un cable USB - Serial.

Se debe de contar con un programa capaz de leer y escribir el puerto serial de la PC como el Hyperterminal o el Hercules SETUP utility. El proceso de configuración se describirá detalladamente en la sección de pruebas.

3.8. La capa física

La capa física está contenida en una función cuyo diagrama de flujo es el siguiente: Esta

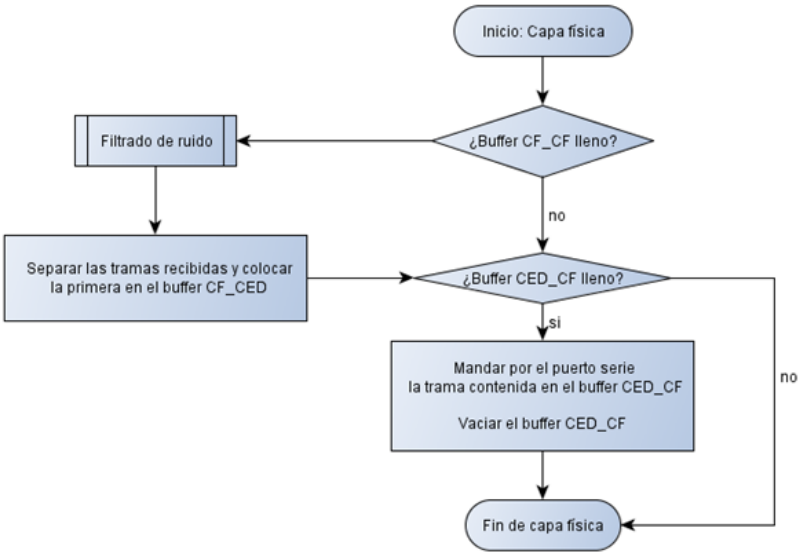


Figura 3.5: Diagrama de flujo de la capa física.

capa se encarga de manejar la recepción y el envío de tramas apoyándose en las funciones para manejar el puerto serial del ATmega328 que el lenguaje Arduino proporciona.

Trabaja en modo full-dúplex ya que tiene la capacidad de recibir una trama mientras envía otra. Por lo general esto es inútil ya que la comunicación entre un esclavo y un maestro en DNP3 se da de la forma: «pregunta - respuesta», sin embargo, cobra importancia cuando las

respuestas no solicitadas están activadas, pudiendo darse el caso de que arribe una petición al esclavo en el momento que se envía la respuesta no solicitada.

3.8.1. Tiempo de recepción

La recepción de información en el buffer CF_CF se realiza en cualquier momento durante la ejecución del programa mediante una interrupción. El microprocesador suspende toda acción que esté llevando a cabo y procede a recibir y a almacenar la trama en CF_CF. Cuando se ejecuta la capa física y el buffer CF_CF se detecta como *lleno*, no se puede saber si la recepción de la trama ya se ha completado o si aún está en proceso. Para evitar que la trama empiece a ser procesada antes de que haya sido recibida por completo, se deja correr un tiempo de espera suficiente como para permitir que llegue una trama del máximo largo posible (292 bytes). Considerando que la velocidad de transmisión es de 9600 bauds (9600 bits por segundo en este caso):

$$T = \frac{1[byte] + 1[bitdeinicio] + 1[bitdeparada] \cdot 292}{9600[bits/s]}$$

$$T = 300[ms]$$

El protocolo especifica que no debe haber espacio de tiempo entre la transmisión de bytes, pero dado el mecanismo de recepción, una trama que no cumpla con esta característica puede llegar a ser admitida si los espacios de tiempo entre bytes no son muy largos.

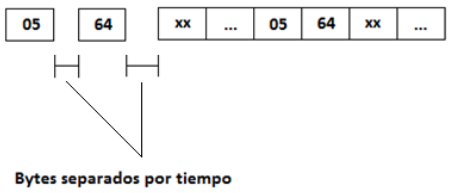


Figura 3.6: Espacio de tiempo entre bytes.

3.8.2. Filtrado de ruido

La información en el buffer CF_CF proviene de un canal físico, por lo que puede contener bytes erróneos debidos al ruido. Si los bytes erróneos se encuentran dentro de una trama no hay nada que se pueda hacer ya que el mismo protocolo la descartará. Sin embargo, si estos se encuentran al principio o al final de la trama, entonces es posible filtrarlos.

El filtrado consiste únicamente en reconocer los bytes de inicio de trama (0x05 y 0x64). Todos los bytes anteriores al inicio de la trama serán descartados. Si se diera el caso en donde se tuviera un inicio de trama erróneo, éste se tomaría como correcto y pasaría una trama basura hacia la capa de enlace de datos, sin embargo, esta la descartaría inmediatamente gracias a los

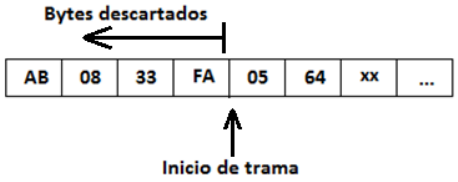


Figura 3.7: Identificación del inicio de la trama.

mecanismos de detección de errores que implementa. El inconveniente aquí sería que es muy probable que la trama verdadera haya sido truncada debido al byte *lenght* falso (utilizado para determinar la longitud de la trama). Esto automáticamente ocasiona que la trama entera se convierta en basura, la cual será descartada en la próxima ejecución de la capa física. Otro caso

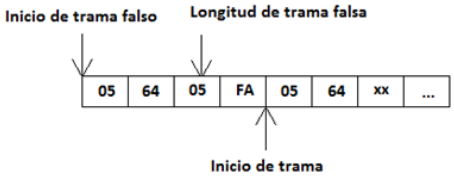


Figura 3.8: Inicio de trama debido a ruido.

especial es cuando se detecta un 0x05 basura. El sistema espera que el siguiente byte sea un 0x64, pero al no ocurrir esto, simplemente se interrumpe el proceso de separación de tramas y se procede a verificar el estado del buffer CED_CF. En la próxima ejecución de la capa física se reanudará la búsqueda de la trama y si no existe otro 0x05 basura esta podrá ser procesada normalmente.

3.8.3. Separación de tramas

Supongamos un escenario como el de la figura 3.9. Puede observarse que el buffer CF_CF

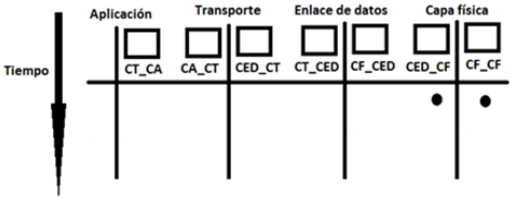


Figura 3.9: Llegada de una trama al sistema.

está lleno, sin embargo, esto no significa que contiene únicamente una trama. El protocolo especifica que pueden enviarse dos tramas continuas sin separación alguna, por lo que para separar estas dos tramas se realiza el siguiente proceso:

1. Se identifica el inicio de la trama (bytes 0x05 y 0x64).

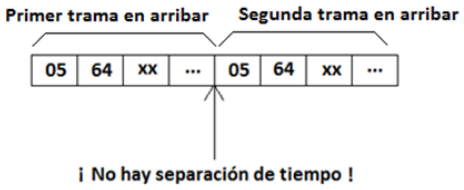


Figura 3.10: Representación de dos tramas arribando al buffer CF_CF.

2. Si el inicio es válido, se procede a leer el byte *lenght*, el cual se representa con un *xx* en la ilustración. Este byte contiene información acerca de la longitud de la trama.

3. Se calcula el número de bytes de la primera trama utilizando la información de *lenght*. Para lograr esto se efectúa el procedimiento de la figura 3.11:

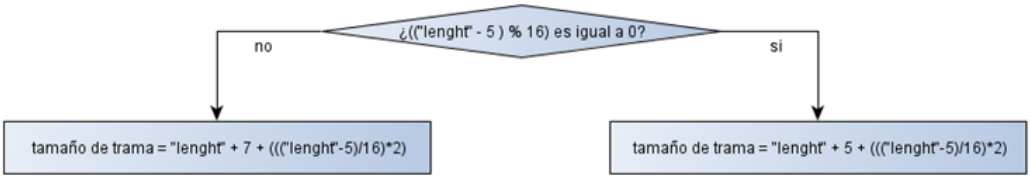


Figura 3.11: Cálculo de la longitud de una trama.

4. La primera trama es copiada al buffer CF_CED para posteriormente ser procesada por la capa de enlace de datos y la trama restante queda almacenada sin alteraciones en CF_CF. Esta será procesada y copiada a CF_CED la próxima vez que la función de la capa física sea invocada.

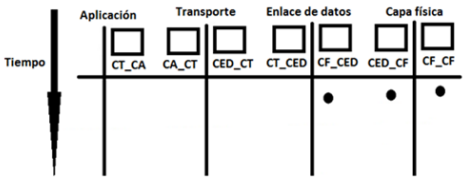


Figura 3.12: Primer trama procesada y enviada a CF_CED.

3.8.4. Envío de tramas

Cuando hay información en el buffer CED_CF, cada uno de los bytes que lo componen es enviado vía serial mediante la instrucción *Serial.write()*. Esta función toma como parámetro un byte (sólo se admiten valores del 0 al 255) y se encarga de enviarlo tal cual, sin ningún tratamiento o conversión (a diferencia de *Serial.print()* o *Serial.println()*). Esto permite que el

proceso de envío sea rápido, de manera que el espaciado de tiempo entre bytes al enviar una trama es nulo (tal como lo especifica el protocolo).

Una vez que la trama ha sido enviada, el buffer CED_CF se destruye para liberar espacio en la memoria RAM.

3.9. La capa de enlace de datos

Nota: Todas las acciones y procedimientos que llevan a cabo la capas de enlace de datos, transporte y aplicación de esta implementación del protocolo van de acuerdo con lo especificado en la norma IEEE 1815 - 2010.

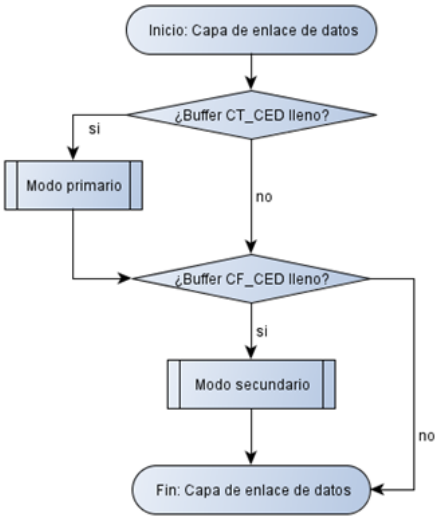


Figura 3.13: Diagrama de flujo de la capa de enlace de datos.

La capa de enlace de datos se encarga de administrar el envío y la llegada de información sirviéndose de la capa física. Esta puede tomar la modalidad primario o la de secundario dependiendo del estado de los buffers CT_CED y CF_CED (ver figura 3.13).

Cuando existe un segmento en el buffer CT_CED se iniciará el modo primario, el cual se encarga de convertir el segmento recibido en una trama (ver figura 3.14) y de gestionar su envío hacia el dispositivo maestro.

Dependiendo del estado del buffer CF_CED puede activarse el modo secundario, responsable de la gestión del arribo de tramas al sistema de alarmas y del procesamiento de éstas (ver figura 3.14).

A continuación se describirá detalladamente el funcionamiento de estos dos modos de operación.

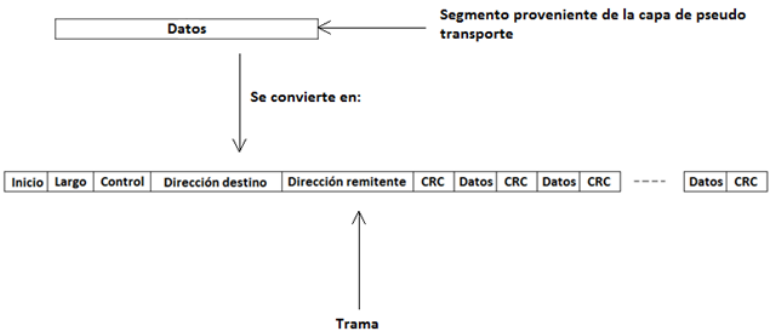


Figura 3.14: Conversión de un segmento a una trama.

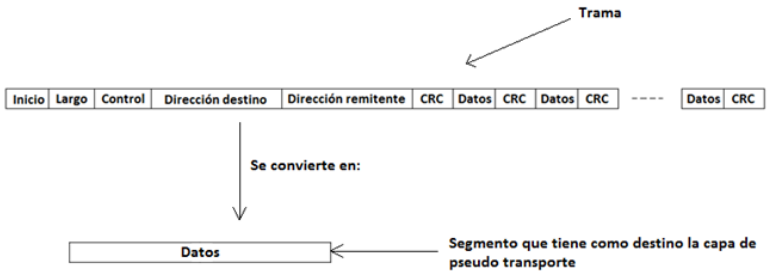


Figura 3.15: Conversión de una trama en un segmento.

3.9.1. Modo primario

El primer paso es verificar el modo de envío de datos, el cuál fue elegido por el usuario al igual que la dirección del destinatario y del emisor dentro del modo de configuración del protocolo. Dependiendo del modo seleccionado, la trama será enviada con el código de función 4 *UNCONFIRMED_USER_DATA* o el 5 *CONFIRMED_USER_DATA*.

Si se eligió el modo sin confirmación, el procedimiento es muy sencillo por que finaliza una vez que la trama es enviada. No obstante, este modo no considera la posibilidad de que el maestro no haya recibido la información debido a un error en el canal de comunicación.

Si el modo elegido fue con confirmación, entonces se verifica que el secundario haya sido inicializado anteriormente. Este proceso consiste en enviar una trama con el código de función 0 *RESET_LINK_STATES* y recibir una con el código 0 *ACK*.

En el diagrama de la figura 3.17 se observa el proceso de inicialización, cuya función principal es sincronizar el valor del bit FCB (ubicado dentro del byte de control). Una vez que llega la respuesta se pasa por un proceso de validación que consiste en la comprobación de los códigos CRC, la de las direcciones y la de las variables del byte de control. Si todas estas son correctas y se recibe un código de función 0 *ACK* entonces se considera que la inicialización tuvo éxito. A partir de este punto, el valor del bit FCB de las tramas enviadas con el código de función *CONFIRMED_USER_DATA* deberá alternarse, empezando con 1 para la primer trama.

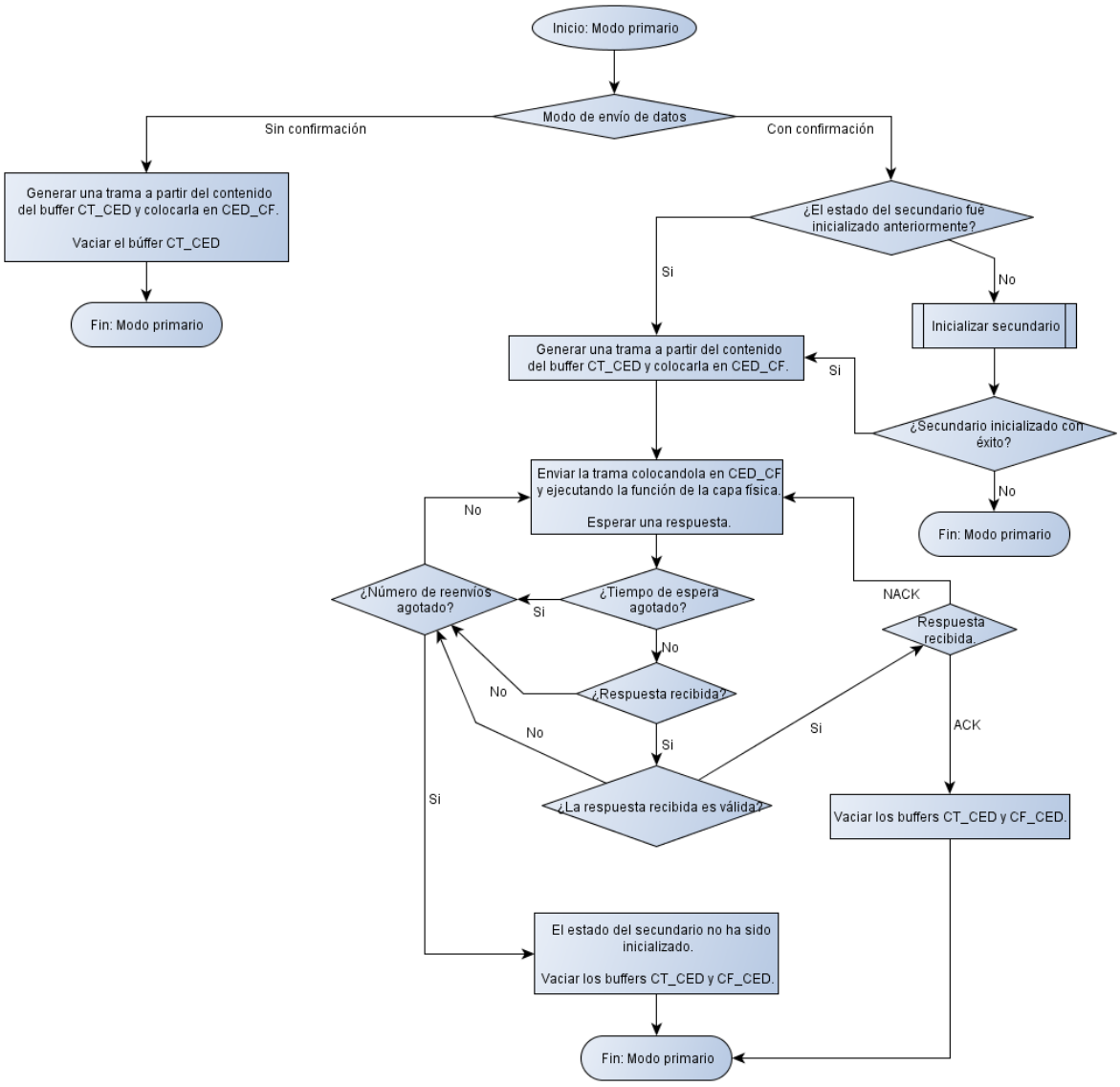


Figura 3.16: Diagrama de flujo del modo primario.

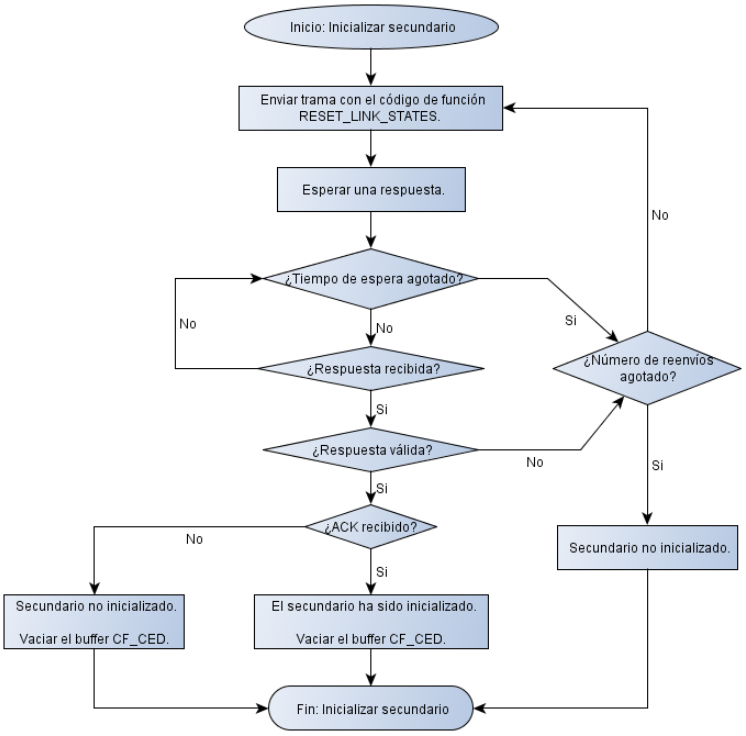


Figura 3.17: Diagrama de flujo del proceso de inicialización.

Regresando al diagrama del modo primario, después de la inicialización, las tramas son generadas y enviadas a través de la capa física. Las figuras 3.19 y 3.18 muestran dos formas de ver el proceso de envío de tramas con las funciones *CONFIRMED_USER_DATA* y *RESET_LINK_STATES*.

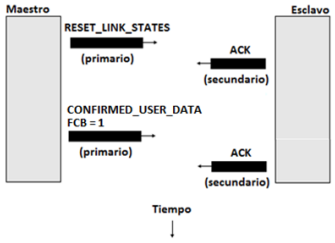


Figura 3.18: Proceso de inicialización del secundario y posterior envío de *UNCONFIRMED_USER_DATA*.

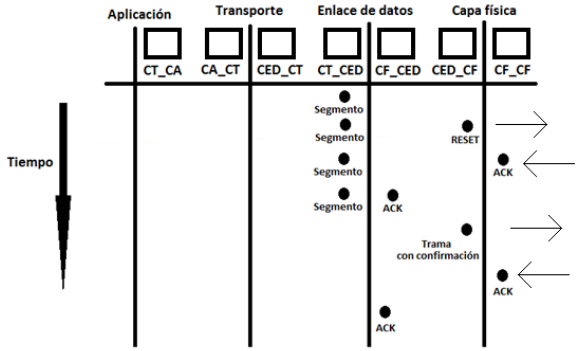


Figura 3.19: Diagrama de capas de la inicialización del secundario y el envío de una trama con confirmación.

3.9.2. Modo secundario

Cuando el buffer CF_CED está lleno significa que ha llegado una trama proveniente del dispositivo maestro que necesita ser procesada. Tal como se indica en el diagrama de flujo del modo secundario (figura 3.20), primeramente se verifica la integridad de la trama mediante la comprobación de los códigos CRC y se comprueba que las direcciones tanto del destinatario como del remitente sean las deseadas (si se recibe una dirección pública, esta se considera como válida y el acontecimiento será documentado en un registro para ser usado posteriormente por la capa de aplicación). En caso de fallar alguna de las dos pruebas anteriores la trama será desechada.

Ya se ha mencionado anteriormente que el secundario puede estar o no inicializado. Si no lo está, las únicas funciones que puede recibir son *UNCONFIRMED_USER_DATA* o *RESET_LINK_STATES*:

1. *UNCONFIRMED_USER_DATA*: Cuando se recibe esta función se comprueba que los bits de DIR, PRM, FCV y FCB sean los correctos. Posteriormente se procesa la trama y se

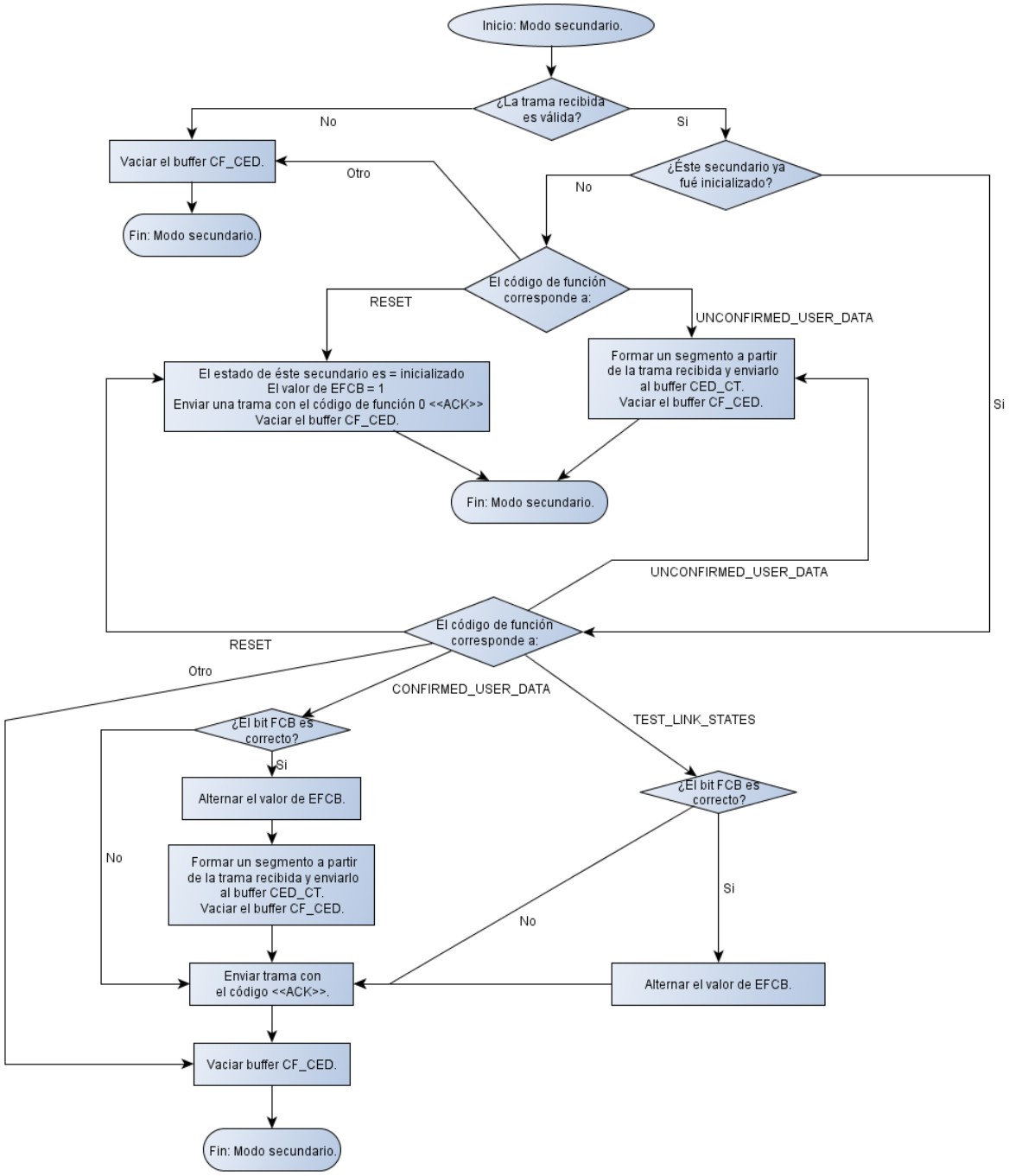


Figura 3.20: Diagrama de flujo del modo secundario.

forma un segmento que será colocado en CED_CT. Los buffers que ya no se ocupan son liberados y el modo secundario llega a su fin.

2. *RESET_LINK_STATES*: A partir de este momento el modo secundario se considera como inicializado y se envía una trama con la función *ACK* para indicar al maestro que se recibió correctamente la instrucción. Un registro de nombre EFCB (FCB esperado) es inicializado con un valor de 1.

Si el secundario fue inicializado, entonces éste podrá recibir además de las funciones descritas anteriormente:

1. *CONFIRMED_USER_DATA*: Se comprueba que el bit FCB sea correcto comparando el valor recibido con el esperado (registro EFCB). Cuando la comprobación es correcta, el valor de EFCB se alterna, la trama es procesada tal como en *UNCONFIRMED_USER_DATA* y se envía un mensaje de confirmación con la función *ACK*. Cuando no es correcta, la trama no es procesada. Sin embargo, el valor del FCB esperado no se alterna y se envía una trama con código *ACK* en un intento por coordinar la comunicación.
2. *TEST_LINK_STATES*: Esta función es parecida a *RESET_LINK_STATES*. Coordina el valor del bit FCB esperado por el secundario teniendo en cuenta el valor recibido.

3.9.3. El cálculo del CRC

Tanto para las tramas (en el buffer CF_CED) como para los segmentos (en el buffer CT_CED), el código CRC se calcula mediante el procedimiento descrito en el capítulo 1. El documento IEEE Standard for Electric Power Systems Communications - Distributed Network Protocol (DNP3) incluye en una de sus secciones la manera de realizar esté cálculo mediante dos métodos distintos: método tabular y por corrimiento de bits. Las desventajas y ventajas de cada uno de los dos son:

1. Método tabular:
   a) Desventajas:
      1) Es necesaria una tabla que ocuparía en memoria 512 bytes.
   b) Ventajas:
      1) Es aparentemente más rápido.
2. Método de corrimiento de bits:
   a) Desventajas:
      1) Realiza un número mayor de iteraciones.
   b) Ventajas:
      1) No necesita de tablas por lo que la memoria que usa es mínima.

Ya que el hardware en el que se está trabajando tiene una memoria RAM muy limitada (2 kilobytes), una tabla de esa extensión no sería aceptable. Tampoco puede usarse la memoria EEPROM ya que debido a las múltiples lecturas podría llegar a desgastarse. El único lugar posible es la memoria Flash del dispositivo, sin embargo, esta usa librerías con funciones especiales para la extracción de información, lo cual podría afectar la rapidez del algoritmo y resultar aún más lento.

Es por estas razones que se elige el método de corrimiento de bits, el cual se implementó dentro de una función que recibe como parámetros un buffer y dos índices: inicio y fin. Dicha función calculará el código CRC de los datos con tenidos entre los índices inicio y fin del buffer en cuestión.

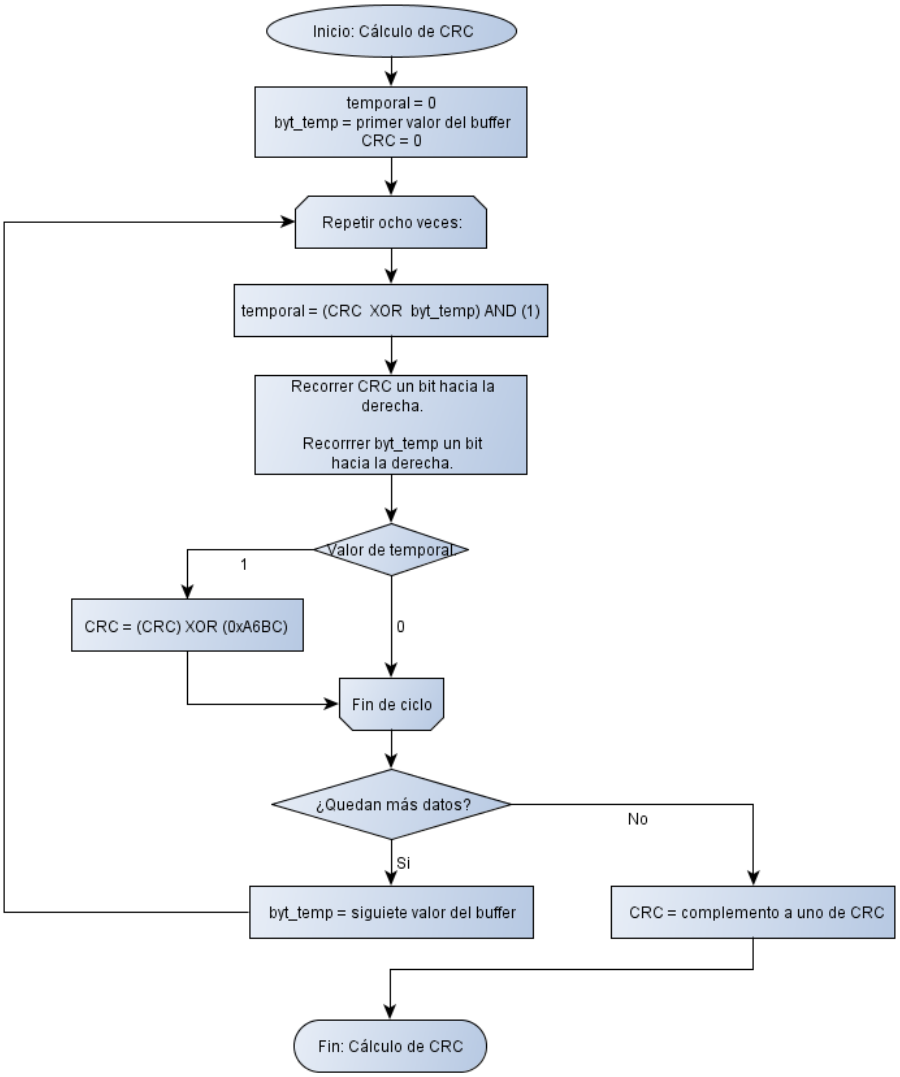


Figura 3.21: Diagrama de flujo del cálculo del CRC por el método de corrimiento de bits.

3.10. La capa de transporte

Las especificaciones del protocolo dictan que un dispositivo esclavo debe de ser capaz de recibir fragmentos de al menos 249 bytes, por lo que se elige este número como la extensión máxima de los fragmentos que puede recibir y enviar el sistema de alarmas. De esta forma se simplifica la implementación de la capa de transporte ya que el máximo fragmento posible tiene la misma extensión de un segmento y no es necesario dividirlos (ver figura 3.22).

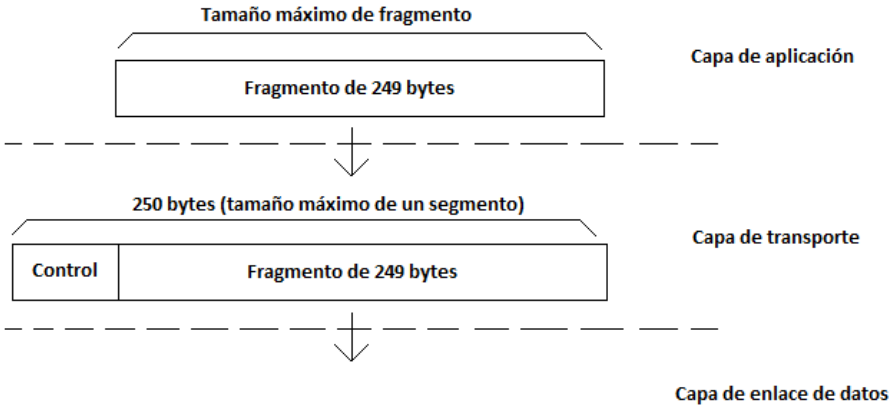


Figura 3.22: Paso de un fragmento a través de la capa de transporte.

De esta forma se reciben y envían series de 1 sólo segmento y la capa únicamente se encargará de verificar el valor de los bits FIR y FIN, el cual deberá ser 1 para ambos. En la figura 3.23 se ilustra el comportamiento de la capa de transporte.

3.11. La capa de aplicación

3.11.1. Implementación nivel 1

La capa de aplicación del sistema de alarmas está diseñada para cumplir con las especificaciones del primer nivel de implementación, por lo que debe de contar con al menos los siguientes aspectos:

1. Debe permitir que un maestro pueda leer mediante la función *READ* objetos del grupo 60 y ciertas variaciones del grupo 0.
2. Modificación del bit *RESTART* de las indicaciones internas.
3. Debe permitir el código de función *COLD_RESTART*.

Además de esto, se implementa la posibilidad de que un maestro lea el estado de las entradas binarias mediante solicitudes del grupo 1. En la tabla 3.1 se presentan todos los objetos y

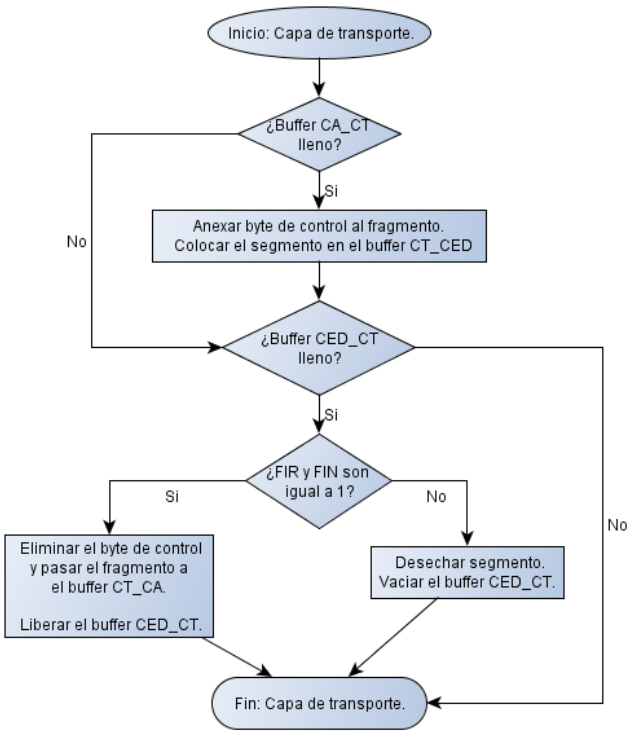


Figura 3.23: Diagrama de flujo de la capa de transporte.

variaciones implementados, además de los calificadores que permiten.

El sistema de alarmas reacciona de manera distinta a cada uno de los códigos de función permitidos y a los objetos a los que se les aplica dicha función:

1. *READ*: El sistema responderá con el valor de cada uno de los objetos requeridos. En caso de detectarse alguna anomalía como por ejemplo un objeto no implementado, el esclavo avisará este acontecimiento al maestro mediante las indicaciones internas (IIN) en una respuesta que puede no contener ningún objeto, es decir, que el fragmento solo estará compuesto por el byte de control, bytes de IIN y código de función. A este tipo de respuestas se les llama «Respuestas nulas».
2. *WRITE*: La reacción del esclavo a este código de función es el envío de una respuesta nula.
3. *COLD_RESTART*: El sistema responderá con un objeto del grupo 52, variación 1 (intervalo de tiempo). Este intervalo de tiempo le indica a la estación maestra el tiempo que el esclavo estará inhabilitado para recibir respuestas ya que se encuentra inicializándose. Se programó un intervalo de 3 segundos, tiempo más que suficiente para tener operativo el microprocesador después de un reinicio.
4. *ENABLE_UNSOLICITED y DISABLE_UNSOLICITED*: Generará una respuesta nula.
5. *CONFIRM*: Cuando es confirmado el envío de una respuesta ya sea solicitada o no solicitada, los eventos que fueron enviados son eliminados del buffer de eventos. Además, después

Grupo	Variación	Descripción	Acepta peticiones con	Genera respuestas con
0	242	Versión del software.	Cod. Func. = 01 Calif. = 0x00	Cod. Func. = 129 Calif. = 0x17
0	243	Versión del Hardware.	Cod. Func. = 01 Calif. = 0x00	Cod. Func. = 129 Calif. = 0x17
0	250	Nombre y modelo del producto.	Cod. Func. = 01 Calif. = 0x00	Cod. Func. = 129 Calif. = 0x17
0	252	Nombre del fabricante.	Cod. Func. = 01 Calif. = 0x00	Cod. Func. = 129 Calif. = 0x17
0	254	Todos los atributos.	Cod. Func. = 01 Calif. = 0x00, 0x06	Cod. Func. = 129 Calif. = 0x17
0	255	Proporciona una lista de atributos. disponnibles.	Cod. Func. = 01 Calif. = 0x00, 0x06	Cod. Func. = 129 Calif. = 0x00
1	0,1	Entradas binarias formato: empaquetado	Cod. Func. = 01 Calif. = 0x00, 0x01, 0x06	Cod. Func. = 129 Calif. = 0x00
2	1	Evento de entrada binaria	Solo se pueden pedir mediante el grupo 60	Cod. Func. = 129, 130 Calif. = 0x17
52	1	Intervalo de tiempo	-	Cod. Func. = 129 Calif. = 0x07
60	1	Datos clase 0	Cod. Func. = 01 Calif. = 0x06	Cod. Func. = 129 Calif. = depende de los datos pedidos
60	2	Datos clase 1	Cod. Func. = 01, 20, 21 Calif. = 0x06	Cod. Func. = 129, 130 Calif. = depende de los datos pedidos
60	3	Datos clase 2	Cod. Func. = 01, 20, 21 Calif. = 0x06	Cod. Func. = 129, 130 Calif. = depende de los datos pedidos
60	4	Datos clase 3	Cod. Func. = 01, 20, 21 Calif. = 0x06	Cod. Func. = 129, 130 Calif. = depende de los datos pedidos
80	1	Indicaciones internas.	Cod. Func. = 02 Calif. 0x00 (índice 7)	Cod. Func. = 129
-	-	Reiniciar sistema.	Cod. Func. = 13	Cod. Func. = 129 Calif. = 07

Cuadro 3.1: Grupos y variaciones aceptados.

de haber sido confirmado el envío de un fragmento, en caso de llegar uno exactamente igual, éste no se tomará como repetido.

La variación cero

Cuando una estación maestra requiere un objeto con variación 0, el esclavo podrá decidir entre las múltiples variaciones que implementa como presentar la información de un determinado grupo. En este caso, cuando se lee el valor de las entradas binarias mediante la variación

0, el sistema responderá con objetos de variación 1, ya que es la única implementada para éste grupo.

3.11.2. Manejo de eventos

Los únicos eventos que se registran en el sistema tienen que ver con el cambio de alguna de las entradas binarias que se detecta durante la fase de chequeo de estados. Dichos eventos son captados en un buffer de 10 localidades como el que se observa en la figura 3.24.

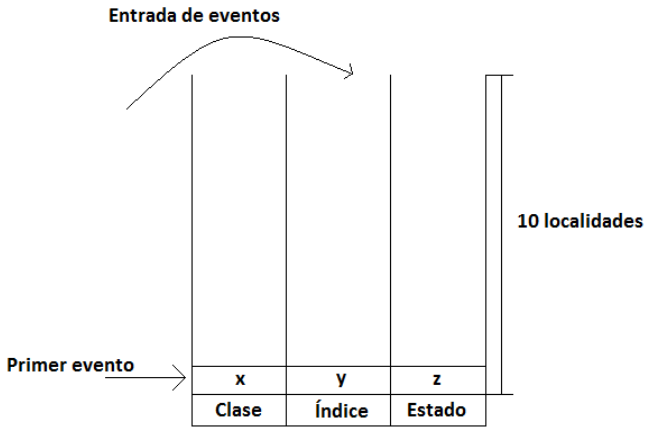


Figura 3.24: Buffer de eventos.

El estado del evento indica el valor al que cambió la entrada en el momento de registrarse dicho evento. El índice representa la entrada binaria en la que se registró, pudiendo ser 1,2,3,4 ó 5. La clase del evento es determinada en el modo configuración del sistema para cada una de las entradas.

El comportamiento del buffer es el siguiente:

1. Los eventos únicamente pueden ser eliminados cuando se recibe una confirmación.
2. Los eventos se almacenan en el buffer por orden de ocurrencia ya que deben de ser reportados en ésta forma.
3. Cuando el buffer se desborda, los eventos guardados permanecen y los nuevos se pierden. Este estado se comunica al maestro mediante el bit EVENT_BUFFER_OVERFLOW de las indicaciones internas.

3.11.3. El programa principal

En la figura 4.2 se presenta el diagrama de flujo del programa principal de la capa de aplicación. Se observa que la primera orden es enviar un mensaje no solicitado sin ningún objeto

(respuesta nula). Su función es la de alertar a la estación maestra que el dispositivo esclavo acaba de reiniciarse. Inmediatamente se comprueba el estado del buffer CT_CA (lugar a donde llegan todas las peticiones) que en caso de estar lleno, se ejecutará una serie de subprocesos que descartarán o validarán el fragmento y que, en este último caso, generarán una respuesta adecuada.

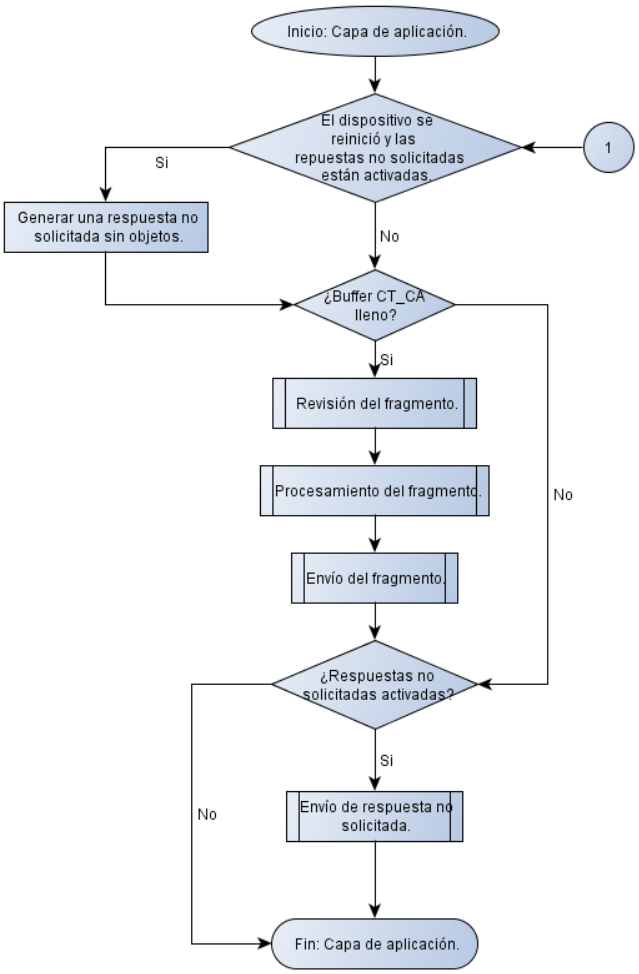


Figura 3.25: Capa de aplicación.

Para facilitar la comprensión de las funciones de esta capa se utilizará como ejemplo el fragmento de la figura 3.26, el cual se localiza en el buffer CT_CA y pretende leer todos los índices del grupo 1, es decir, el valor de las 5 entradas del sistema.

C5	01	3C	01	06

Figura 3.26: Ejemplo.

1. Byte de control: 0xC5 = 0b11000101, por lo que:

   a) FIN = 1
   b) FIR = 1
   c) CON = 0
   d) UNS = 0
   e) Secuencia = 5

2. Código de función (FC): 0x01 = (*READ*)
3. Objeto: 0x3C 0x01 0x06
   a) Grupo: 60
   b) Variación: 1
   c) Calificador: 6

Revisión del fragmento

La sección de revisión del fragmento se encarga de verificar que los datos del byte de control del fragmento recién llegado sean correctos. Si todo está en orden, dicho fragmento es guardado en un buffer auxiliar para que, en caso de llegar otro fragmento con el mismo número de secuencia, pueda ser comparado y así determinar si es repetido o no. Si la dirección que contenía la trama era de una dirección pública, el fragmento no es válido o es repetido, entonces la función llega a su fin.

El fragmento del ejemplo de la figura 3.26 no presenta ningún problema por lo que se toma como válido.

Procesamiento del fragmento

Si no hubo ningún problema con el fragmento durante su revisión, éste pasa a ser procesado. En esta etapa es dónde se generará la respuesta (si es requerida).

Una vez que todos los objetos de la petición fueron procesados, en caso de que la dirección del mensaje no fuera pública, se agrega el byte de control, el código de función y las indicaciones internas. Después de esto el fragmento estará listo para enviarse, pero no sin antes ser almacenado en un buffer auxiliar llamado CA_CT_reserva con el fin de reenviarse en caso de que se reciba la misma petición más de una vez de manera consecutiva.

El proceso por el que pasan cada uno de los objetos se muestra en el diagrama de flujo de la figura 3.29.

El objeto, su variación y el calificador del ejemplo de la figura 3.26 son válidos ya que si están implementados y aceptan la función *READ*, por lo que de acuerdo al diagrama de la figura 3.28, una respuesta podría ser la que se muestra en la figura 3.30.

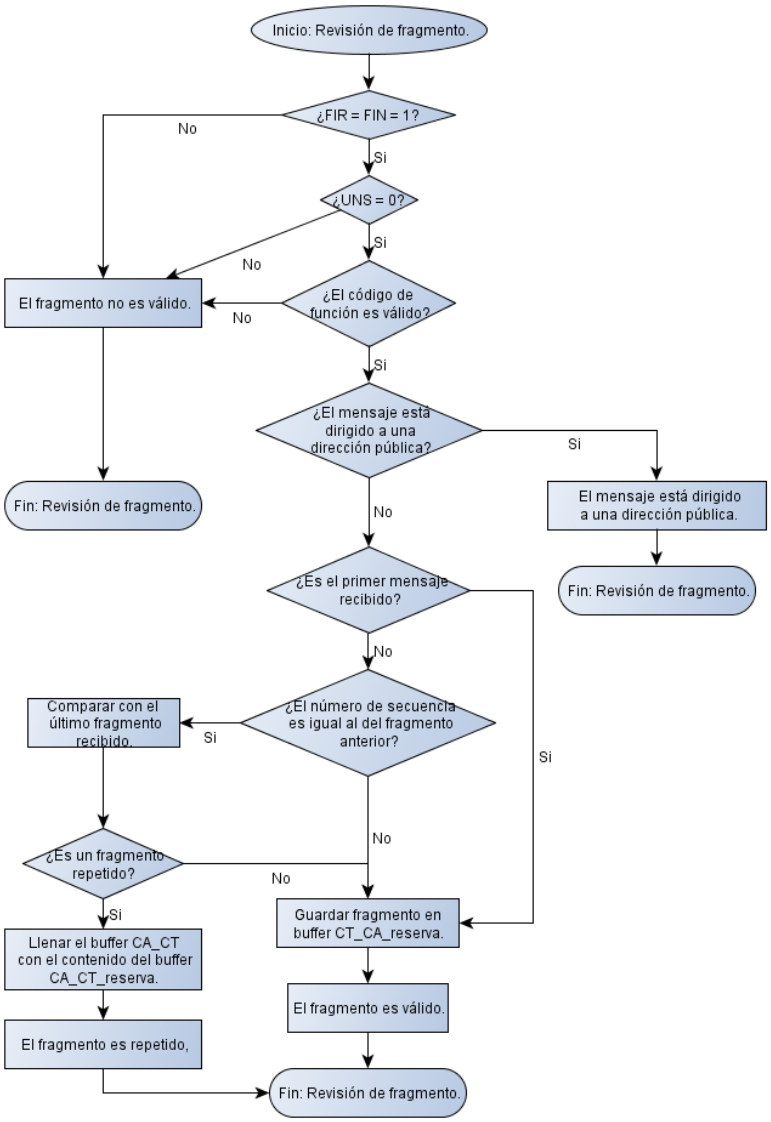


Figura 3.27: Revisión de fragmento.

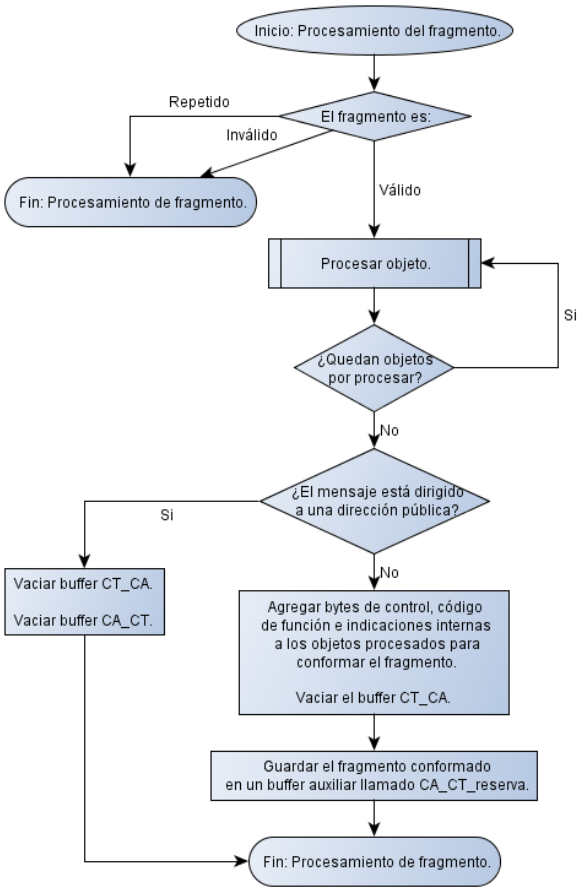


Figura 3.28: Procesamiento de fragmento.

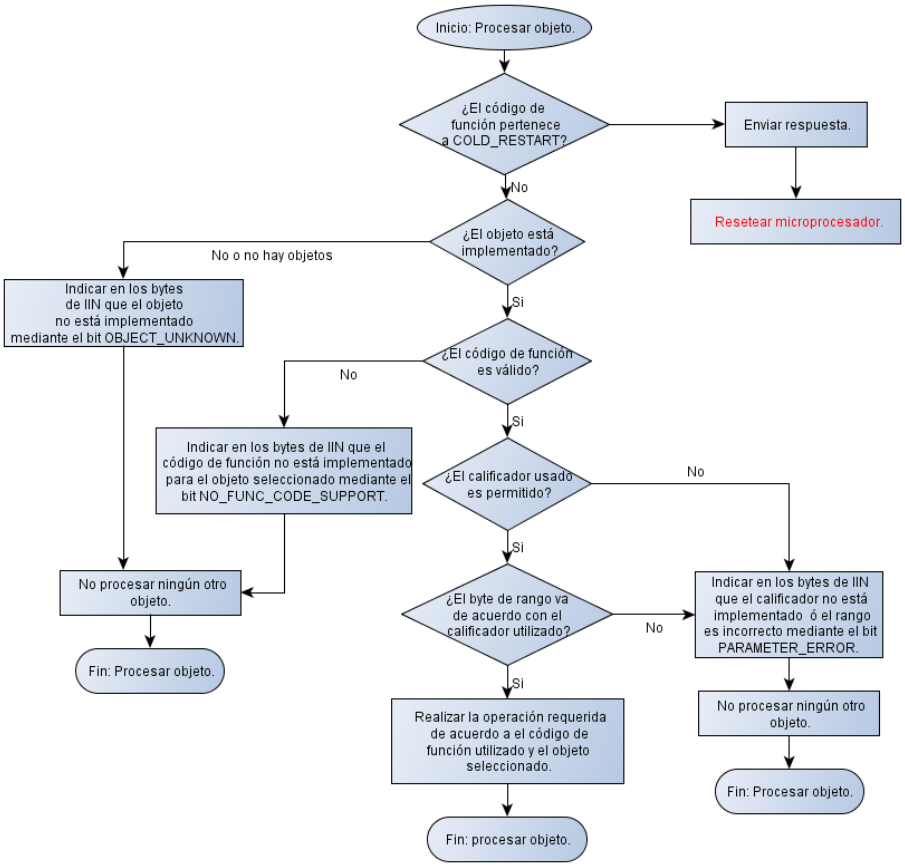


Figura 3.29: Procesamiento de objetos.

E5	81	00	00	01	01	01	01	05	02

Figura 3.30: Fragmento de respuesta.

1. Byte de control = 0xE5 = 0b11100101
   a) FIR = 1
   b) FIN = 1
   c) CON = 1
   d) UNS = 0
   e) SEQ = 5
2. IIN 1 = 0
3. IIN 2 = 0
4. Objeto:
   a) Grupo = 1
   b) Variación = 1
   c) Calificador = 1
   d) Rango = Índices del 1 al 5
   e) Estado = 2 = 00000010 . De acuerdo al empaquetamiento de los datos del grupo 1, variación 1, esto quiere decir que la entrada binaria número 2 tiene un estado igual a 1 mientras que el de las otras es 0 (ver librería de objetos en la sección de apéndices).

Envío del fragmento

La función de este subproceso es la de enviar la respuesta generada en subprocesos anteriores valiéndose de las capas inferiores (transporte, enlace de datos y física), esperar la confirmación a nivel de aplicación (todas las respuestas son enviadas con el bit CON = 1 por lo que todas las respuestas enviadas generan confirmaciones) y validarla. En la validación se toma la importante decisión de conservar o borrar los eventos enviados en la respuesta.

El diagrama de flujo de este subproceso se ilustra en la figura 3.31.

Envío de respuesta no solicitada

Esta es una de las opciones más potentes de la capa de aplicación por que permite enviar los cambios de estado de las entradas del sistema en el momento en que ocurren sin la necesidad de que la estación maestra genere una petición de lectura.

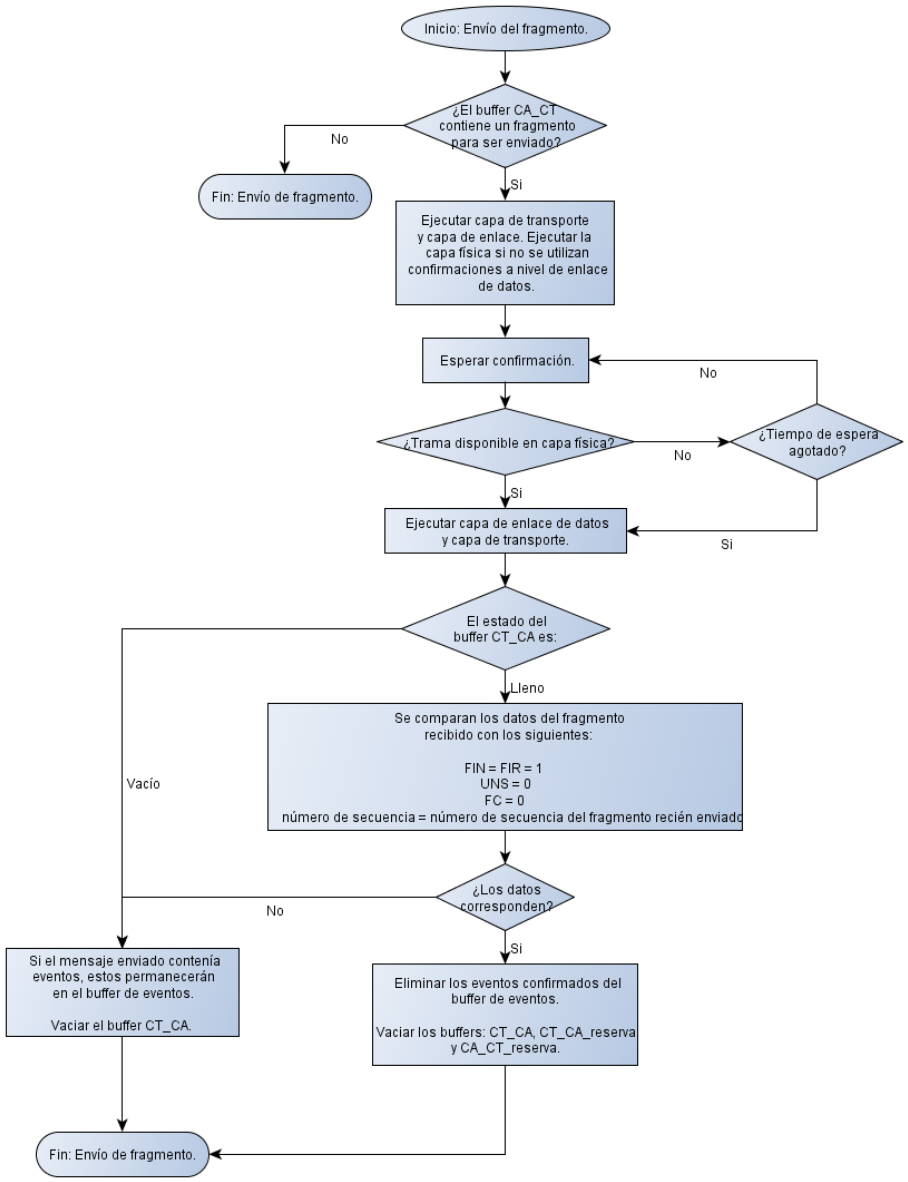


Figura 3.31: Envío de fragmento.

Aunque el módulo DNP3 esté configurado para enviar respuestas no solicitadas, estas se encuentran inhabilitadas en un principio y solamente pueden ser activadas si se recibe la función *ENABLE_UNSOLICITED_RESPONSE*.

Por especificación del protocolo, el bit CON de todas las respuestas es 1, por lo que al enviarse una respuesta de este tipo, siempre se espera una confirmación por parte del maestro. El subproceso determinará si la confirmación es válida o no y de acuerdo a esto se eliminarán los eventos enviados del buffer de eventos.

Las figuras 3.32 y 3.33 contienen el diagrama de flujo del subproceso.

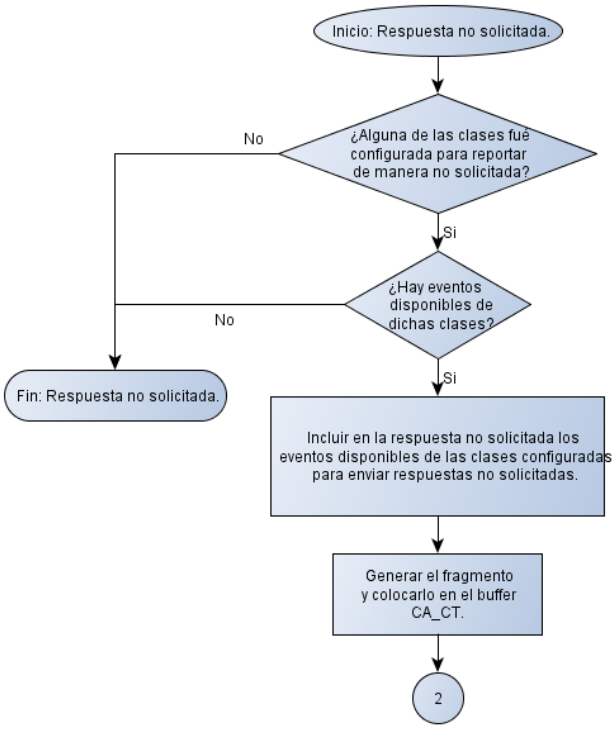


Figura 3.32: Envío de respuesta no solicitada.

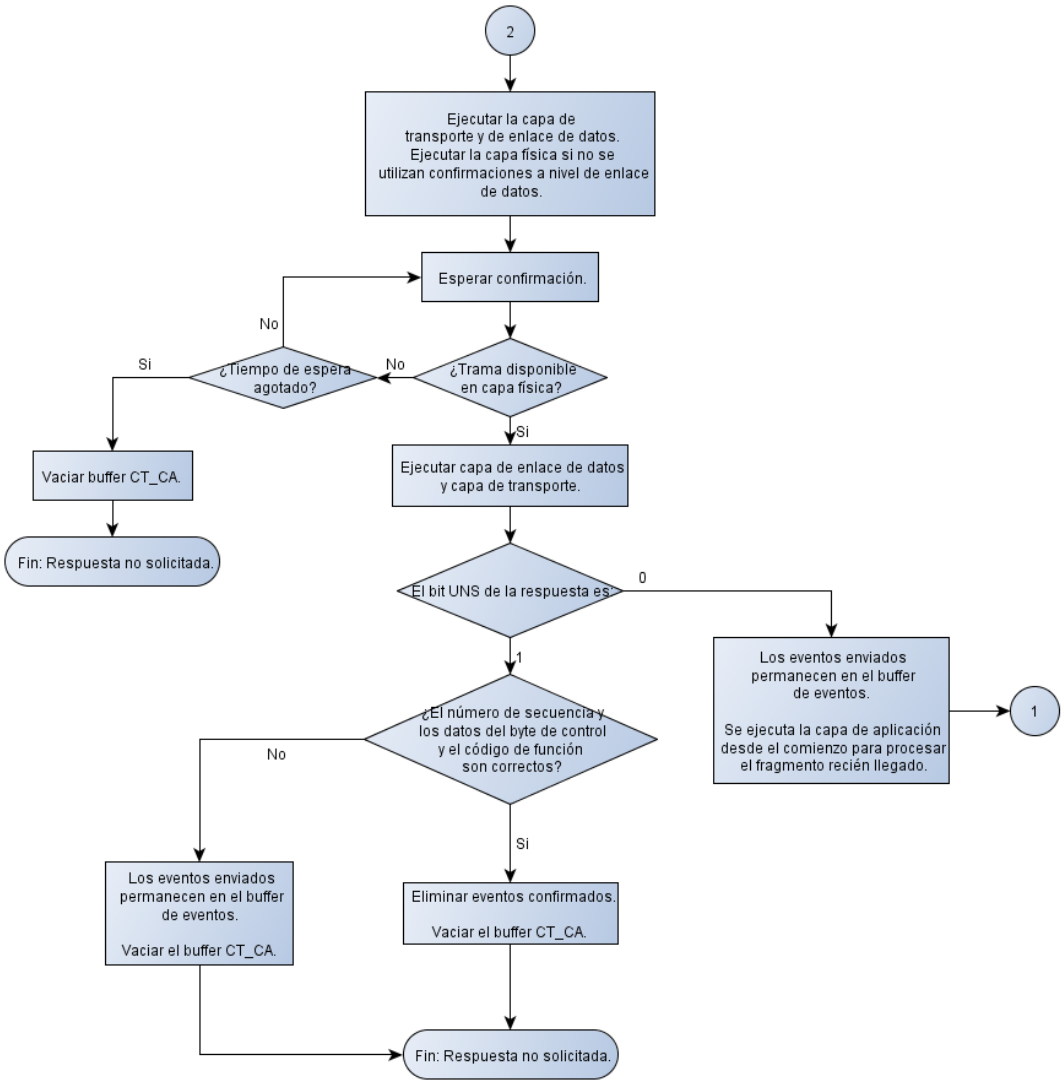


Figura 3.33: Envío de respuesta no solicitada.

Capítulo 4

Diseño del hardware

El diseño consta de 3 partes fundamentales. El microprocesador, los contactos secos y la alimentación. En este capítulo se describe el diseño del hardware de cada uno de esos módulos.

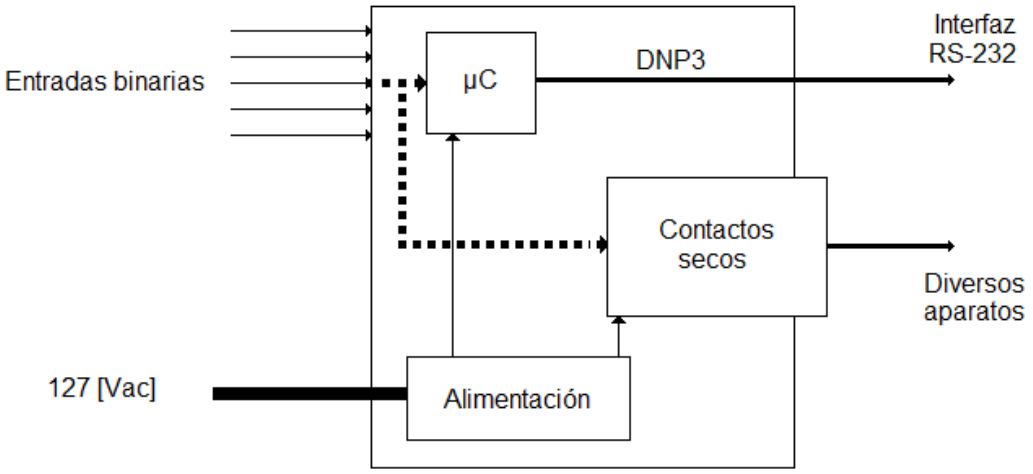


Figura 4.1: Esquema del sistema de alarmas.

4.1. Entorno de desarrollo para el protocolo

Se elige Arduino como la plataforma para la implementación del protocolo DNP3 debido a su bajo costo, versatilidad y facilidad de uso. Dicha plataforma consta de:

1. Software: El entorno de desarrollo permite programar en lenguaje Arduino, el cual está basado en C/C++. Utiliza el compilador avr-gcc de AVR Libc, librería gratuita diseñada para programar microcontroladores ATMEL.
2. Hardware: La tarjeta de desarrollo Arduino UNO incluye un microprocesador ATmega328,

el cual contiene todos los módulos necesarios para el desarrollo del proyecto.

4.1.1. Los módulos del ATmega328

El módulo USART

El módulo USART (*Universal Synchronous and Asynchronous serial Receiver and Transmitter*) se utilizará para implementar la capa física del protocolo. Algunas de sus características son:

1. Puede configurarse para transmitir 5, 6, 7, 8 ó 9 bits. Para éste caso se utilizará la transmisión de 8 bits (1 byte) ya que es la unidad que se maneja en el DNP3.
2. Trabaja con tensiones TTL por lo que para tener una interfaz RS-232 será necesario convertir dichas tensiones.
3. Su velocidad de transmisión es configurable.
4. Permite operación en full-dúplex ya que los registros de transmisión y recepción están separados.

Puertos de entrada/salida digitales de propósito general

Los puertos de propósito general sirven para implementar las entradas del sistema. Cuando son configurados como entradas digitales se comportan como resistencias de un valor muy grande.

El comportamiento que se requiere en las entradas del sistema es el que se muestra en la tabla 4.1, por lo que serán necesarias resistencias *pull-down* para evitar la detección de niveles lógicos aleatorios en caso de presentarse un estado de alta impedancia a la entrada de los puertos (ver figura 4.2).

Tensión aplicada [V]	Nivel lógico detectado [V]
5	1
0	0
alta impedancia	0

Cuadro 4.1: Comportamiento de las entradas del sistema.

Memoria EEPROM y Flash

El microprocesador cuenta con 512 bytes de memoria EEPROM. Ésta es utilizada para guardar variables que deben permanecer aún si el microprocesador es reiniciado pero que también

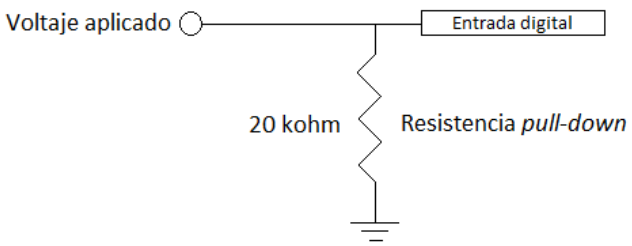


Figura 4.2: Resistencias *pull-down* en las entradas digitales del sistema.

deben ser modificables durante la ejecución del programa como son los parámetros del protocolo. Se debe ser cuidadoso con el uso que se le da ya que únicamente permite aproximadamente cien mil ciclos de lectura/escritura.

También cuenta con 32 kilobytes de memoria Flash, en la cual se encuentra almacenado el programa, las cadenas de caracteres y los atributos del sistema.

Memoria RAM

El microprocesador tiene una memoria RAM de 2048 bytes y es gestionada por la librería AVR Libc de la siguiente forma:

Figura 4.3: Memoria RAM del sistema.

1. En la sección *Variables* se guardan todas las variables globales. El tamaño de esta sección es fijo (no se modifica durante la ejecución del programa) y depende de la cantidad y el tipo de dato de las variables.

   Para determinar el tamaño de ésta sección de memoria haría falta declarar las siguientes variables:

   a) extern unsigned int __data_start

   b) extern unsigned int __bss_end

   Estas dos variables fueron definidas por el compilador y son dos apuntadores que se localizan en el inicio y fin de la sección *Variables* por lo que la resta nos daría la extensión de dicha sección:

```
(int)&__bss_end  - (int)& __data_start
```

Esto arroja un resultado de 432 bytes, dejando así 1616 bytes libres para la sección Heap / Stack.

2. El Stack se encarga de almacenar las variables locales e información sobre las funciones como sus parámetros, valores de retorno y la dirección de la instrucción a ejecutar una vez terminada la subrutina. Comienza al final de la memoria RAM y crece hacia la izquierda, acercándose de manera peligrosa al espacio del Heap. Si el tamaño del Stack y del Heap son grandes pueden llegar a sobreponerse y ocasionar errores.

   En el código del protocolo DNP3 se hace uso de funciones, sin embargo, el tamaño del Stack puede ser despreciado debido a que:

   a) Son pocas las funciones a las que se les pasa un parámetro o que regresan algún valor.

   b) No se tiene un número alto de invocaciones anidadas (llamadas a una función desde alguna otra función). Si el número fuera elevado, el Stack podría llegar a saturarse debido a que tiene que almacenar la información de todas las funciones que han sido invocadas.

   c) El número de variables locales es mínimo y por lo general es de tipo char (con una longitud de 1 byte).

3. En el Heap se guardan las variables asignadas dinámicamente mediante las funciones *malloc*, *calloc*, *realloc* y *free*, además de algunas otras variables del compilador. Comienza justo donde termina la sección de variables globales y su tamaño varia durante la ejecución del programa.

   Es muy difícil predecir su tamaño porque está sujeto a un constante cambio, no obstante, se tiene que asegurar que en ningún momento se va a sobreponer con el Stack. Por esta razón se realizó la siguiente prueba:

   a) Se adjunta la libreria MemoryFree, la cual permite conocer el tamaño de memoria libre RAM en algún momento dado durante la ejecución del programa.

   b) Se eligió un punto del programa en el cual se considera que el tamaño del Heap puede llegar a alcanzar su valor máximo, como por ejemplo en la capa de aplicación.

   c) Se agrega la siguiente línea en el punto anterior:

   ```
   Serial.print(freeMemory());
   ```

   Al ejecutarse el programa y después de haber recibido/enviado varias tramas, el valor de memoria libre mínimo es de aproximadamente 1200 bytes, por lo que en ningún momento se corre el riesgo de que la memoria llegue a agotarse.

Las librerías de Arduino

Las librerías de arduino permiten controlar los diferentes módulos del microprocesador. En el proyecto se utilizan las siguientes:

1. Serial: Permite configurar y controlar el módulo USART del ATmega328.
2. Metro: Hace uso de uno de los temporizadores del microprocesador para generar tiempos de espera.
3. EEPROM: Permite la lectura/escritura de la memoria EEPROM.
4. avr/pgmspace: Permite la lectura de la memoria Flash.

4.1.2. Interfaz RS-232

Dado que el módulo USART del ATmega328 trabaja con niveles de tensión TTL y que una gran cantidad de dispositivos como computadoras o módems hacen uso de ciertas características del estándar RS-232 (como los niveles de tensión y la interfaz física), es necesario convertir dichos niveles de TTL a RS-232 y proporcionar conectores tipo DB-9 para que el sistema sea lo más compatible posible.

La conversión de tensión se realiza con ayuda del CI MAX232 (figura 4.4). Éste circuito es capaz de recibir señales con tensiones TTL para convertirlas en tensiones RS-232 y viceversa a partir de una fuente de 5 [V]. Para aumentar la compatibilidad, los pines de transmisión y de recepción del conector DB-9 (figura 4.5) pueden ser configurados para actuar como un DTE (*Data Terminal Equipment*) ó un DCE (*Data Communication Equipment*).

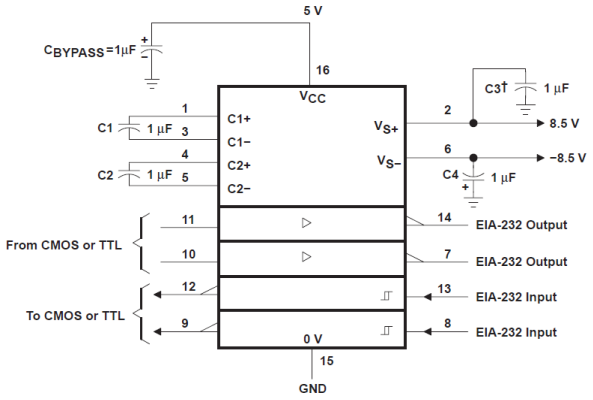


Figura 4.4: MAX232.

Función de los pines para DTE:

1. Transmisión: Conectado al pin Tx del microprocesador mediante el CI MAX232. Se encarga de transmitir la información generada por el protocolo DNP3.
2. Recepción: Conectado al pin Rx del microprocesador mediante el CIMAX232. Se encarga de recibir las peticiones provenientes de otros equipos.

Este modo es utilizado cuando se conectan dispositivos de tipo DCE (p.e. módems) al sistema de alarmas.

Función de los pines para DCE:

1. Transmisión: Conectado al pin Rx del microprocesador mediante el CI MAX232. Se encarga de recibir las peticiones provenientes de otros equipos.
2. Recepción: Conectado al pin Tx del microprocesador mediante el CIMAX232. Se encarga de transmitir la información generada por el protocolo DNP3.

Se utiliza este modo cuando el sistema de alarmas se conecta con dispositivos DTE (p.e. computadoras) para evitar el uso de componentes extra como adaptadores de tipo *NULL-MODEM*.

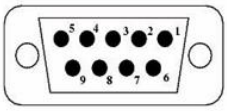


Figura 4.5: Conector DB-9 hembra.

4.1.3. Función de RESET por hardware

Para cumplir con el nivel 1 de implementación del protocolo, es necesario incluir en la capa de aplicación la función *COLD_RESTART*. Ésta le da la capacidad a un maestro de reiniciar los dispositivos esclavo que tenga conectados a él mediante el envío de un simple comando que incluya dicha función.

Existen dos formas de reiniciar el ATmega328:

1. Por software: Es posible reiniciar el microprocesador mediante una serie de instrucciones pero no se utilizará ya que la memoria RAM y algunos registros no son modificados en el proceso por lo que se podrían generar errores.
2. Por hardware: El pin *RESET* permite reiniciar el microprocesador por completo al aplicarsele una tensión de 0 [V] durante 2.5 [µs].

Se descartó la reinicialización por software debido a los inconvenientes que presenta y se implementa una solución por hardware:

Uno de los puertos digitales de propósito general es conectado al pin de *RESET* a través del circuito de la figura 4.6.

Al llegar el comando de reinicio del dispositivo maestro, el puerto digital pasa de un nivel lógico bajo a un nivel alto. Esta acción activa mediante un MOSFET (Q1) el temporizador

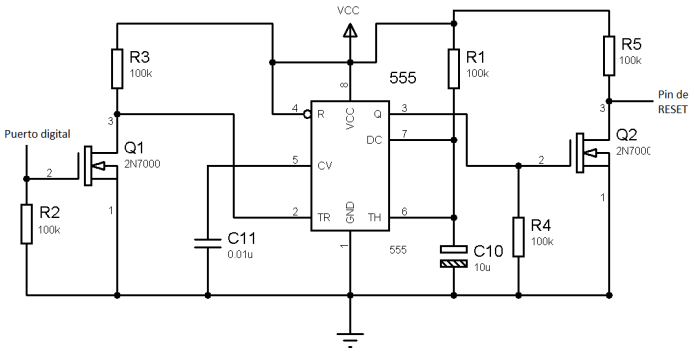


Figura 4.6: Circuito para reiniciar el microcontrolador.

555, cuya salida pasa igualmente de un nivel lógico bajo a uno alto durante un tiempo de 1 segundo. Al mismo tiempo, el MOSFET Q2 comienza a conducir haciendo que el pin de *RESET* quede aproximadamente a 0[V], reiniciando así el sistema. Podría pensarse que conectando directamente el puerto digital con el pin de *RESET* se lograría el mismo cometido, no obstante, esto no es posible ya que durante el tiempo en que el pin *RESET* se encuentra a 0[V], todos los puertos adquieren un estado de alta impedancia.

El valor de R_1 y C_{10} se calculan mediante la ecuación 4.1 dado un tiempo de $t = 3[s]$ (tiempo más que suficiente para llevar a cabo exitosamente el reinicio del microprocesador):

$$t = 1.1R_{10}C_{10} \tag{4.1}$$

El circuito del MAX232, el LM555 y el microprocesador conforman entonces el módulo DNP3, cuyo esquemático se muestra en la sección de apéndices.

4.2. Contactos secos

Un contacto seco es aquel en el cual no existe originalmente una tensión (de ahí el «seco») y es en esencia un interruptor. Éstos contactos respresentan una forma de comunicar una alarma sin la necesidad de transmitir tramas de información ya que el mensaje es enviado por medio del estado del interruptor.

Los relevadores seleccionados para la implementación de los contactos secos son del tipo LY1-DC12. Sus terminales permiten conectarse a bases que pueden ir montadas sobre un carril DIN. La tensión de la bobina para activar estos relevadores es de 12 [Vdc] y consumen una potencia de 0.9 [W], lo que implica una de corriente de 75 [mA].

La figura 4.7 muestra el circuito que permite manejar estos relevadores:

1. R_1: Esta resistencia tiene 2 funciones: Descargar el capacitor parásito de la unión *Com-*

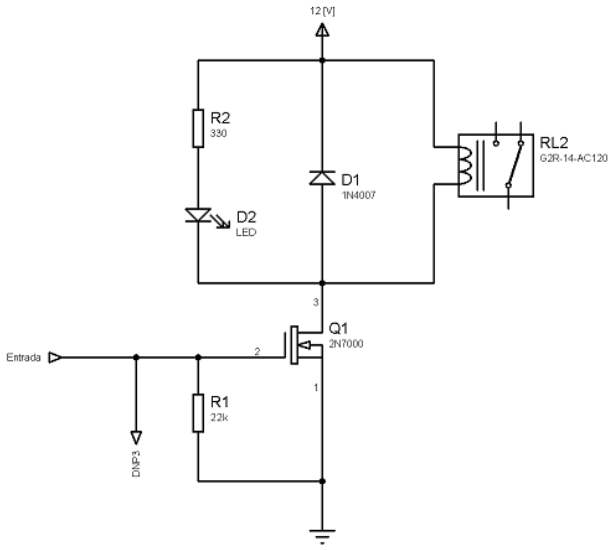


Figura 4.7: Circuito de control del relevador.

puerta - Surtidor y servir como resistencia de *pull - down* para los puertos digitales del microcontrolador.

2. R_2 y D_2: Este LED permite conocer el estado del relevador (activado/desactivado). Si la tensión del LED es de $V_{LED} = 2[V]$, entonces la corriente a través de estos dos elementos será de 30 [mA].

3. Q_1: El MOSFET 2N7000 es capaz de conducir una corriente I_D de hasta 200 [mA] y de ser manejado por tensiones V_{GS} relativamente bajas. Proporciona aislamiento a las entradas del sistema debido a que la compuerta prácticamente no consume corriente. Este dispositivo puede activarse con una tensión mínima de $V_{GS} = 3[V]$, sin embargo, se deben utilizar las tensiones TTL (ver tabla 2.1 en pag. 25) ya que las entradas digitales del ATmega328 se encuentran conectadas a la compuerta.

   El dispositivo funciona como interruptor, por lo que se utiliza en las regiones de corte y lineal. Para colocarlo en la región de corte basta con que V_{GS} sea igual a 0 o esté ausente. Para la región lineal se tiene que $V_{DS} \leq V_{GS} - V_p$.

   Si la corriente $I_D = 105[mA]$ para $V_{GS} = 5[V]$, se tendrá una tensión V_{DS} tan pequeña que puede ser despreciada, por lo que la posibilidad de que la bobina del relevador no alcance la tensión necesaria para activarlo es descartada.

4. D_1: Este diodo sirve para descargar la energía almacenada en la bobina del relevador cuando el transistor Q_1 pasa a corte.

Se requieren 5 circuitos de este tipo para manejar las 5 señales de las entradas del sistema. El diagrama esquemático se muestra en la sección de apéndices e incluye también la fuente de alimentación.

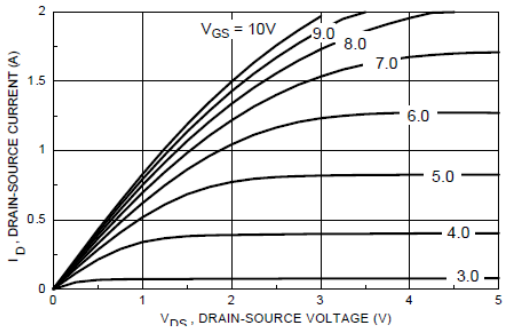


Figura 4.8: 2N7000: I_D vs V_{DS}.
[21]

4.3. La fuente de alimentación

La fuente de alimentación deberá proporcionar la corriente necesaria para alimentar el módulo DNP3 y cada uno de los relevadores. Si cada relevador con su LED indicador consume 105 [mA] al estar encendidos, el consumo de los 5 juntos será de 525 [mA]. El consumo de corriente del módulo DNP3 se tomará como 200 [mA] ya que éste es el máximo consumo del ATmega328. La fuente de alimentación deberá entonces de ser capaz de proporcionar al menos 725 [mA].

Se elige un diseño de fuente de tipo lineal debido a su simplicidad:

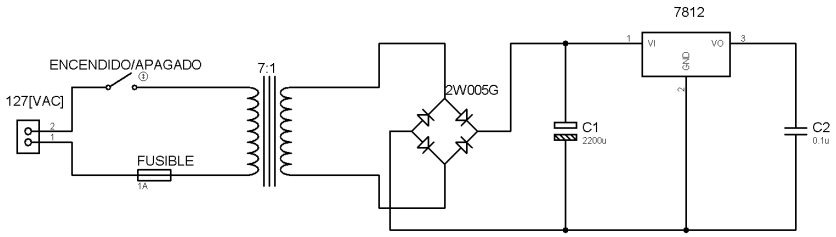


Figura 4.9: Esquema del circuito de la fuente de tensión.

1. Transformador: Se utilizaró un transformador de 127 [VRMS] a 18 [VRMS] @ 1 [A]. La tensión pico de salida será de:

$$Vp = 18[V] \cdot \sqrt{2} = 25.45[V]$$

2. Puente de diodos: El puente de diodos 2W005G conduce un máximo de 2 [A] y tiene una caída de 1.1 [V] por diodo. La tensión de salida tendrá un máximo de 23.25 [Vp] debido a dicha caída.

3. Regulador de tensión: Se elige el LM7812 que proporcionará un máximo de 1 [A] @ 12 [V] en su salida para alimentar los relevadores. La tensión de encendido del integrado es de 14 [V]:

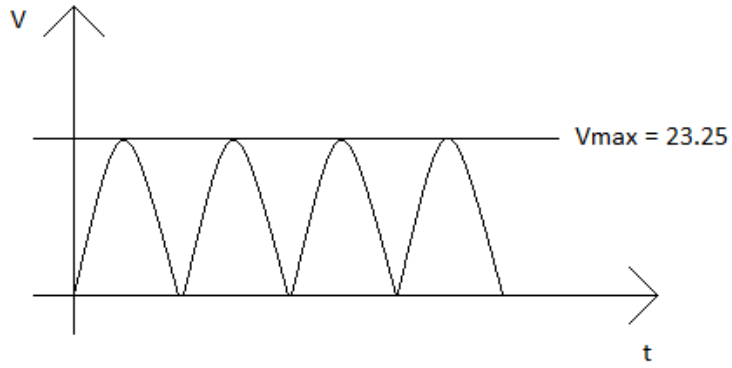


Figura 4.10: Tensión a la salida del puente de diodos.

$$V_{DROPOUT} = 2[V]$$

$$V_{encendido} = V_o + V_{DROPOUT} = 14[V]$$

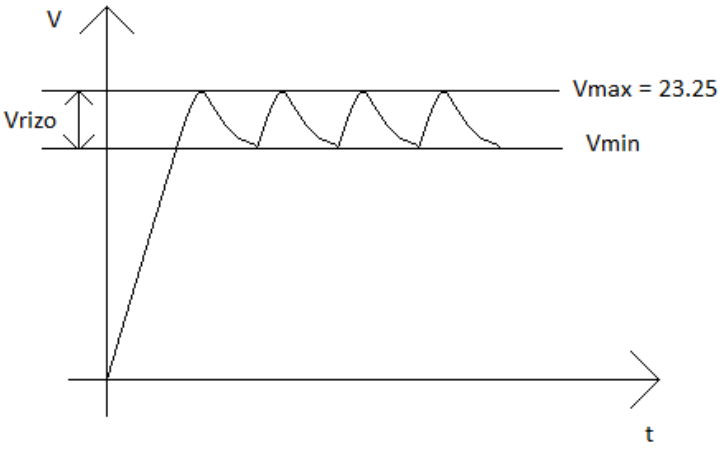


Figura 4.11: Tensión de rizo.

Por lo que la tensión de rizo debe ser menor a 9.25 [V].

$$V_{rizo\_max} = V_{max} - V_{encendido} = 23.25[V] - 14[V] = 9.25[V]$$

4. Capacitor: La ecuación para seleccionar el capacitor de la fuente se obtiene mediante el análisis de un circuito RC:

$$v(t) = k \cdot e^{\frac{-t}{RC}} \tag{4.2}$$

Para un tiempo $t = 0$ se fija:

$$k = V_{max}$$

Despejando la ecuación 4.2 y sustituyendo $v(t)$ por V_{min} se obtiene:

$$C = \frac{-t}{ln\left(\frac{V_{min}}{V_{max}}\right) \cdot R} \tag{4.3}$$

Con la ecuación 4.3 se puede calcular el valor del capacitor C seleccionando la tensión mínima de rizo. El tiempo estará dado por:

$$t = t_a + t_b$$

$$t_a = 4.16[ms]$$

$$t_b = \frac{sin^{-1}\left(\frac{V_{min}}{V_{max}}\right)}{2\pi \cdot 60}$$

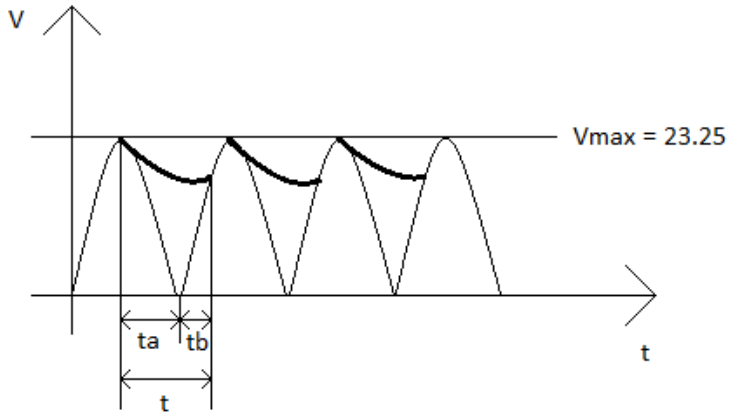


Figura 4.12: Tiempos t, ta y tb.

R es la resistencia equivalente de los componentes que se va a alimentar. Esta puede calcularse con la corriente que debe suministrar la fuente y la tensión a la que lo hace:

$$R = \frac{V_{max}}{I}$$

$$I = 725[mA] \approx 1[A]$$

$$R \approx 23.25[\Omega]$$

Si se elige una tensión de rizado del 15 % del valor máximo, se tendrá un valor de tensión mínimo de:

$$V_{min} = 19.76[V]$$

por lo que de acuerdo a la ecuación 4.3, el valor del capacitor será de:

$$C = 1806[\mu F] \approx 2200[\mu F]$$

5. Disipador: El uso de un disipador será necesario debido a la potencia desperdiciada en forma de calor en el regulador de tensión.

   Para el cálculo de su valor máximo se utiliza el circuito térmico de la figura 4.13:

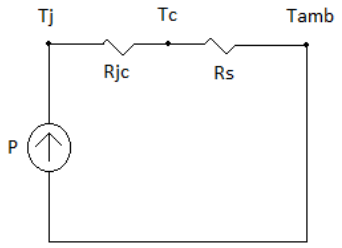


Figura 4.13: Circuito térmico utilizado en el cálculo del disipador.

R_{jc} representa la resistencia térmica que existe entre la juntura del transistor del regulador y la carcasa de éste, mientras que R_s la resistencia térmica entre el disipador y el aire. Se desprecia la resistencia térmica entre la carcasa del regulador y el disipador ya que con el uso de grasa siliconada es posible reducirla prácticamente a 0.

La potencia disipada por el regulador está dada por la siguiente ecuación (la tensión de entrada será la tensión máxima):

$$P = I \cdot (V_{entrada} - V_{salida}) = 725[mA] \cdot (23.25[V] - 12[V]) = 8.1[W]$$

Del circuito térmico se tiene que:

$$R_s = \frac{T_{j\_max} - T_{ambiente}}{P} - R_{jc} \tag{4.4}$$

Además:

$$R_{jc} = 5 \left[\frac{^oC}{W}\right]$$

$$T_{ambiente\_max} = 35[^oC]$$

$$T_{j\_max} = 125[^oC]$$

Por lo que la resistencia máxima del disipador de acuerdo a la ecuación 4.4 deberá ser:

$$R_{s\_max} = 6.1 \left[\frac{^oC}{W}\right]$$

4.4. El chasís

El sistema se colocará en un gabinete para rack de 19 pulgadas cuyas medidas y especificaciones se encuentran en el estándar EIA-310. En la figura 4.14 se muestran las medidas de dicho rack.

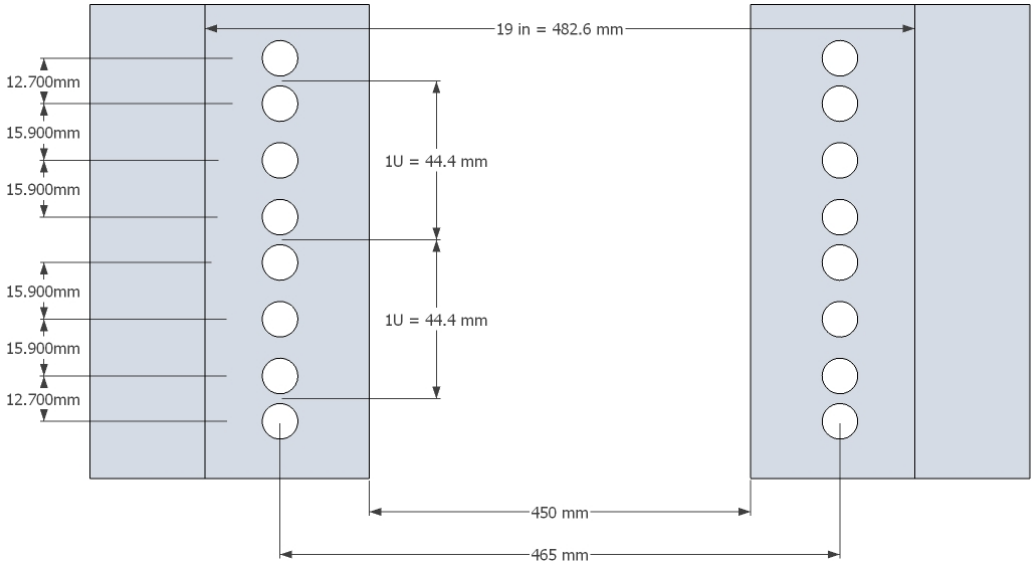


Figura 4.14: Especificaciones del rack de 19 pulgadas.

El rack tiene un ancho de 19 pulgadas y una altura de 42 unidades (1 unidad = 44.4 mm). Cada equipo montado en el deberá cumplir con las medidas del ancho y deberá tener una altura que corresponda a un número entero de unidades. De esta forma, se tiene que el gabinete más pequeño debe tener altura de una unidad.

El gabinete deberá albergar todo el sistema, del cual, los elementos más voluminosos son los relevadores con sus bases. Debido a esto, tuvo que elegirse una altura de dos unidades. En la figura 4.15 se muestra el gabinete armado. Éste consta de 4 partes distintas: frontal, trasera, laterales y superior e inferior. Las medidas de cada uno pueden encontrarse en la sección de apéndices del documento.

Las piezas serán sujetadas entre sí mediante tornillos DIN 7985 M3 de 8 milímetros de largo y tuercas M3 con excepción de la parte superior, la cual deberá ir roscada para poder ser

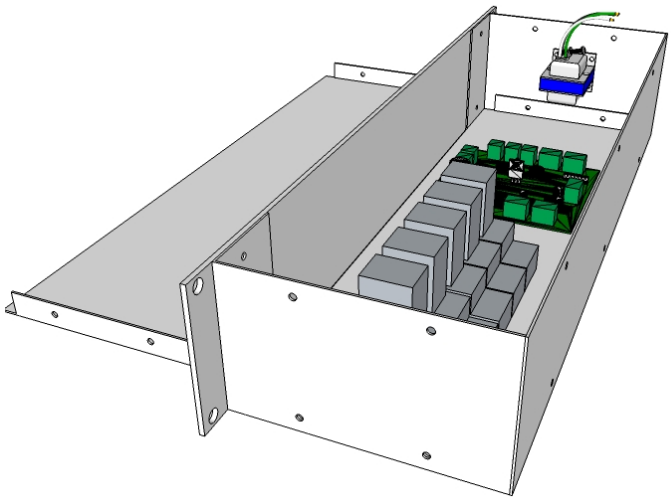


Figura 4.15: Gabinete armado.

atornillada.

La ventilación puede agregarse horadando alguna de las partes del gabinete. Preferentemente la parte inferior o las laterales ya que aperturas en la parte superior facilitarían la entrada de polvo. Opcionalmente se podría instalar un respiradero.

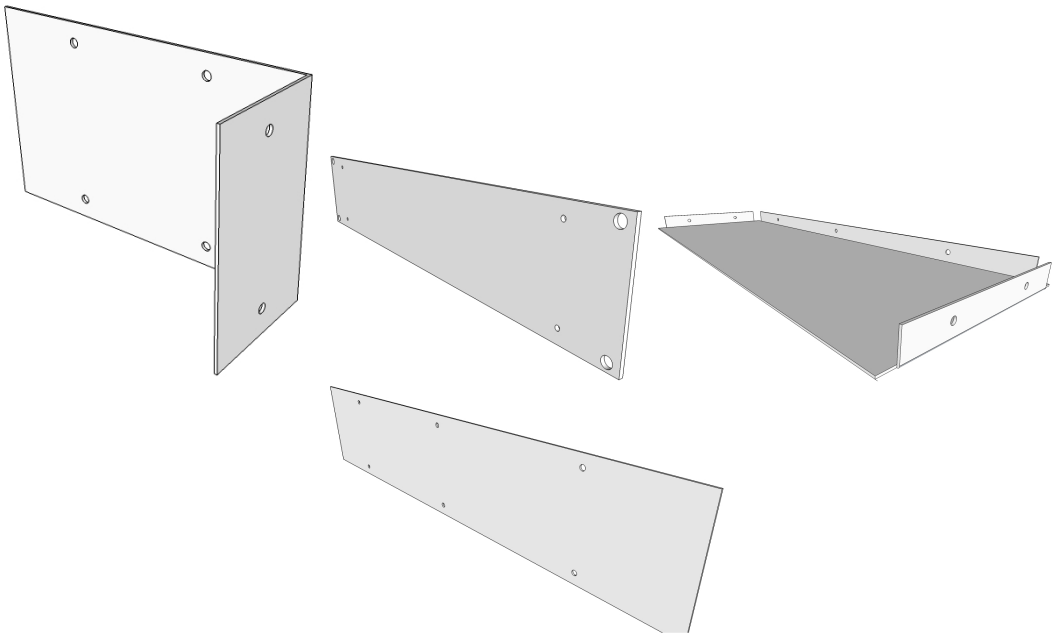


Figura 4.16: Componentes del chasís.

Capítulo 5

Pruebas y resultados

En este capítulo se probará el funcionamiento y el desempeño del sistema de alarmas. Para llevar esto a cabo, las pruebas se dividirán en tres secciones:

1. Configuración del protocolo: Se describirá con detalle el proceso requerido para configurar algunos aspectos del protocolo DNP3 que el usuario debe determinar.
2. Pruebas del módulo DNP3: Se simulará una estación DNP3 maestra mediante el uso de software y se probarán las funciones del módulo DNP3 del sistema de alarmas.
3. Pruebas de la función *COLD_RESTART*: Se verificará que el envío de dicha función reinicie el sistema de la manera esperada.

5.1. Configuración del protocolo

La configuración del módulo DNP3 se realiza mediante un terminal serial que permita el envío de datos en formato hexadecimal. Existe una gran cantidad de programas para llevar a cabo tal acción, sin embargo, la prueba se realizará únicamente utilizando el programa *Hercules Setup Utility*.

A continuación se presentan los pasos para llevar a cabo la configuración:

1. Conectar el módulo DNP3 a la computadora mediante un cable serial, pudiendo ser este un cable USB - serial.
2. Ejecutar el programa *Hercules Setup Utility*.
3. Elegir la pestaña «Serial» y configurar la comunicación de la siguiente forma: nombre del puerto serial: el que haya registrado la computadora; bauds = 9600, tamaño de datos = 8 bits, sin paridad; sin *Handshake* y en modo libre.

4. Se hace clic derecho sobre el espacio en blanco y se selecciona la opción *Special chars* → *Text mode* y se activa la casilla *CR/LF Enable*.
5. Se energiza el módulo DNP3.
6. Abrir el puerto con el botón *Open*.
7. Seleccionar el modo configuración del módulo y reiniciarlo.
8. En la sección en blanco aparecerá algo como lo que se muestra en la figura 5.1.

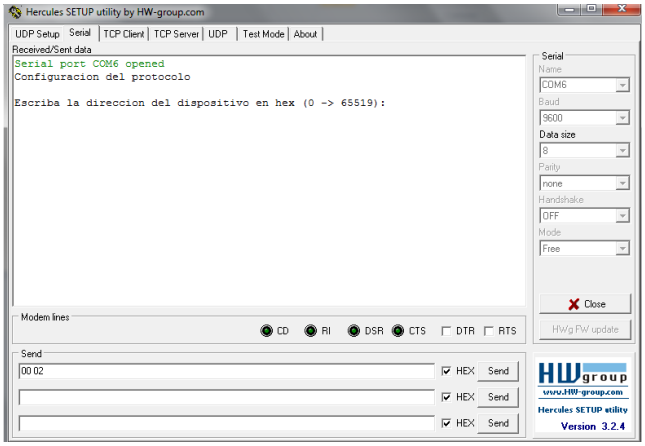


Figura 5.1: Configuración del módulo DNP3 en *Hercules Setup Utility*.

Todos los valores deben ser enviados en hexadecimal mediante la habilitación de las casillas «HEX». El número a enviar depende de la cantidad de bytes de cada parámetro. Por ejemplo, la dirección del dispositivo puede ir desde 0 hasta 65519, por lo que se necesitan dos bytes para especificarla (en la figura 5.1 se muestra el envío del 2, este será la nueva dirección del módulo).

Cuando se detecta un valor válido aparecerá la opción de configurar el próximo parámetro. Si el valor que se introduce es inválido no se podrá progresar en la configuración (ver figura 5.2).

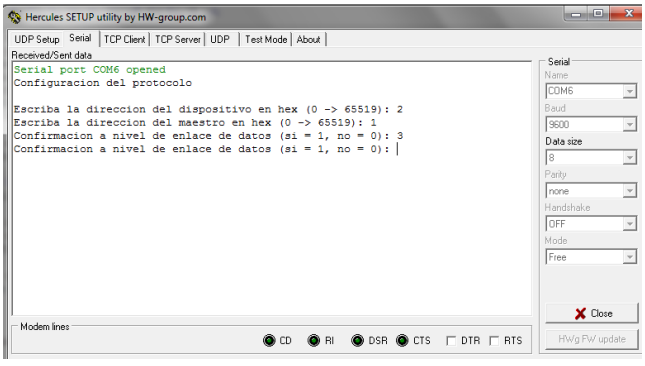


Figura 5.2: Valor de parámetro no válido.

El valor de las clases de cada una de las entradas se introduce como si fuera un solo número hexadecimal (únicamente separados por un espacio).

9. Finalmente se quita la selección del modo configuración del módulo y se reinicia.

5.2. Pruebas del módulo DNP3

Para simular estaciones maestro de DNP3 se utilizarán tres programas distintos:

1. Communication Protocol Test Harness: Diseñado por la compañía Triangle MicroWorks Incorporated, contiene una versión de prueba que se utilizará para probar los diferentes comandos que el módulo DNP3 del sistema de alarmas mediante la simulación de una estación maestra nivel 4.
2. DNP3 TestSet: Es un programa de código abierto que simula dispositivos DNP3 tanto maestros como esclavos. Está muy limitado en cuanto a las funciones que implementa aunque será suficiente para verificar el funcionamiento del sistema de alarmas.
3. Hercules Setup Utility: Se utilizará para probar la función de *COLD_RESTART*, la cual no está implementada en ninguno de los dos programas anteriores.

5.2.1. Communication Protocol Test Harness

El objetivo de esta prueba es el de verificar el buen funcionamiento de algunos de los comandos que una implementación DNP3 nivel 1 debería aceptar (ver tabla 3.1 en la pag. 56).

Los pasos que se siguieron para la realización de la prueba son:

1. Instalar el *Communication Protocol Test Harness*. En este caso se instaló en una computadora con Windows XP.
2. Instalar los drivers del cable USB - Serial. El cable utilizado fue el adaptador de USB a serial.
3. Se configuró el módulo DNP3 del sistema de alarmas con los siguientes parámetros:
   a) Dirección del dispositivo: 2.
   b) Dirección del maestro: 1.
   c) Confirmación a nivel de enlace de datos: si.
   d) Número de reenvíos: 3.
   e) Tiempo máximo de espera en la capa de enlace: 768 [ms] (0x0300).
   f) Habilitar respuestas no solicitadas: si.
   g) Tiempo máximo de espera en la capa de aplicación: 3328 [ms] (0x0D00).
   h) Clase de cada una de las entradas binarias: Todas clase 1.

Esta configuración permitirá probar el módulo en su forma más compleja que es habilitando la confirmación a nivel de enlace de datos y las respuestas no solicitadas.

4. Se configura el *Communication Protocol Test Harness* de la siguiente forma:

   a) Dentro de la pestaña Channel - Advanced Settings se configura el puerto serial de igual forma en el que trabaja el sistema de alarmas (8 bits de datos, 1 bit de parada, sin paridad, sin *handshaking*). El nombre del puerto se elegirá de acuerdo al que le haya asignado Windows al cable USB - Serial (ver figura 5.3). Dentro del mismo menú se configura a la capa de enlace del dispositivo maestro para pedir confirmaciones a nivel de enlace de datos.

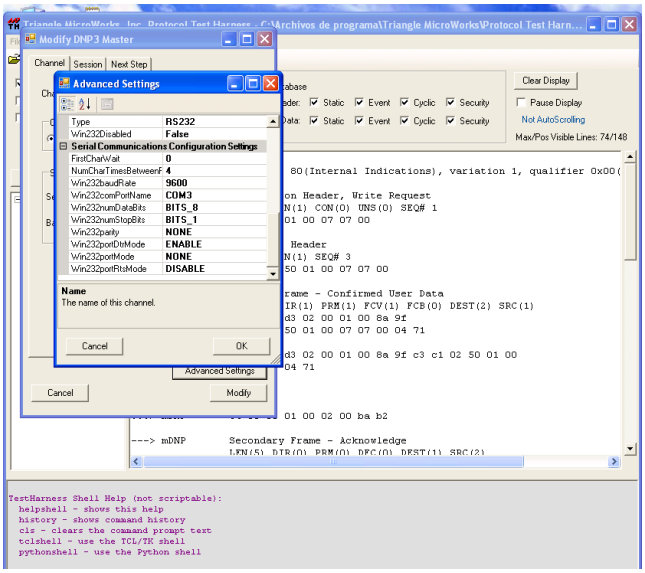


Figura 5.3: Communication Protocol Test Harness - Configuración avanzada de la pestaña Channel.

   b) Dentro de la pestaña Session se eligen las direcciones de la capa de enlace las cuales serán 2 y 1 para el esclavo y el maestro respectivamente (ver figura 5.4).

   c) En el menú de Session - Advanced Settings se configura el programa para enviar confirmaciones automáticas a nivel de aplicación.

De esta forma se tendrá configurado un dispositivo maestro nivel 4 que enviará comandos únicamente cuando se le ordene, que utiliza confirmaciones a nivel de enlace de datos y que envía confirmaciones a nivel de aplicación automáticamente cuando éstas son requeridas. Al finalizar la configuración se tendrá en el escritorio de Windows lo que se muestra en la figura 5.5.

   a) En la ventana principal se muestra la información de cada petición o respuesta en cada una de las diferentes capas del protocolo así como una rudimentaria interfaz humano - máquina.

   b) Estadísticas: Muestra información como por ejemplo el número de mensajes enviados, mensajes recibidos, mensajes fallidos, entre otros.

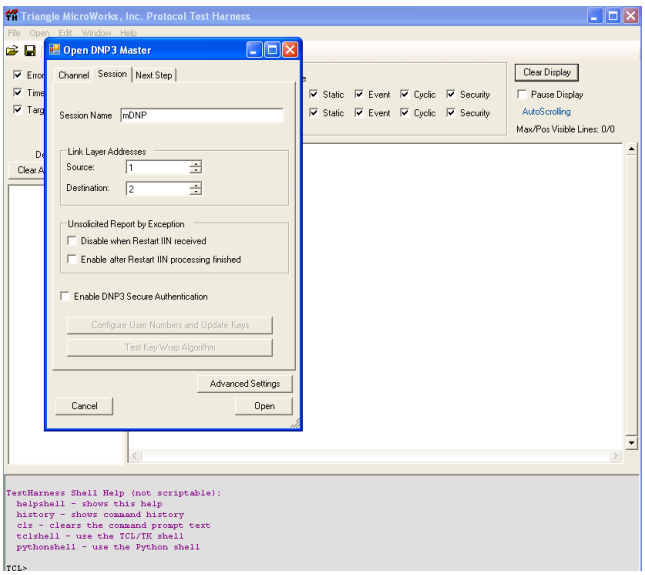


Figura 5.4: Communication Protocol Test Harness - Pestaña Session.

c) Ventana de comandos: Se muestran todos los comandos que pueden enviarse desde el dispositivo maestro. Los comandos que se utilizarán para probar el sistema de alarmas serán únicamente:

1) Solicitud de integridad: Efectúa una lectura de la variación 1 del grupo 60 (Clase 0).
2) Borrado del bit de *RESTART*: Efectúa una escritura del objeto 80 variación 1 (Escribe un 0 en el bit de *RESTART* de las indicaciones internas).
3) Habilitar/Deshabilitar respuestas no solicitadas: Habilita o deshabilita el envío de eventos de determinada clase por medio de respuestas no solicitadas.
4) Leer tipos de datos DNP3 específicos: Permite leer diferentes objetos y seleccionar el calificador para tal acción.

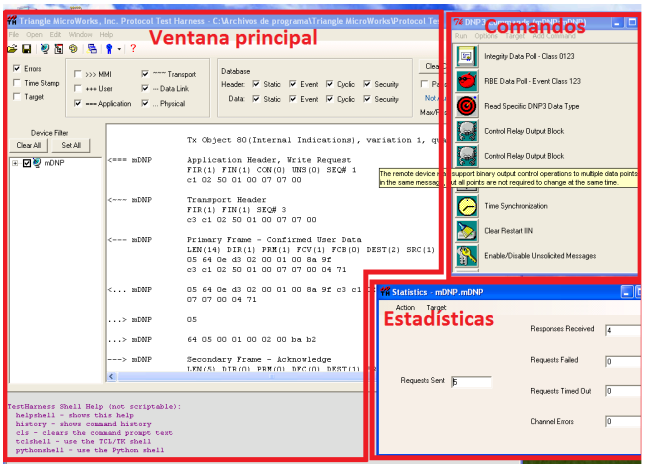


Figura 5.5: Communication Protocol Test Harness (Entorno de trabajo).

5. Finalmente se colocan en un estado lógico alto (5 [V]) todas las entradas binarias del sistema.

Una vez que se tiene todo listo se procede a energizar el módulo DNP3. En la sección de apéndices pueden observarse a detalle las transacciones de información entre el *Communication Protocol Test Harness* y el módulo DNP3 del sistema de alarmas. A continuación se presentan dichas transacciones mostrando únicamente la interfaz humano - máquina del programa:

1. Inicio: Primeramente, ya que las respuestas no solicitadas están activadas, de acuerdo al diagrama de flujo principal de la capa de aplicación (figura 3.25 en la página 58), debe enviarse un mensaje no solicitado vacío cuando el sistema se inicia.

2. Atributos disponibles: Posteriormente se realiza la lectura del grupo 0, variación 255, la cual regresa los diferentes atributos disponibles del módulo DNP3:

```
Rx Object 0(Device Attribute), variation 255, qualifier 0x17(8 Bit Index)

Device Attribute Property 0 Point 0 Variation 242=Device manufacturers
software version string

Device Attribute Property 0 Point 0 Variation 243=Device manufacturers
hardware version string

Device Attribute Property 0 Point 0 Variation 250=Device manufacturers
product name and model

Device Attribute Property 0 Point 0 Variation 252=Device manufacturers name string
```

   Estos atributos son los mínimos necesarios para cumplir con el nivel de implementación 1 y son: Versión del software del sistema; versión del hardware; nombre del producto y modelo; y nombre del fabricante.

3. Valor de los atributos disponibles: Con la lectura del grupo 0, variación 254, debería obtenerse el valor de cada uno de los atributos anteriores:

```
Rx Object 0(Device Attribute), variation 242, qualifier 0x17(8 Bit Index)
Device Attribute Point 0, Variation 242=Device manufacturers software version string
code 1=Visible Characters value 1.0

Rx Object 0(Device Attribute), variation 243, qualifier 0x17(8 Bit Index)
Device Attribute Point 0, Variation 243=Device manufacturers hardware version string
code 1=Visible Characters value Arduino UNO

Rx Object 0(Device Attribute), variation 250, qualifier 0x17(8 Bit Index)
Device Attribute Point 0, Variation 250=Device manufacturers product name and model
code 1=Visible Characters value Sistema de alarmas DNP3

Rx Object 0(Device Attribute), variation 252, qualifier 0x17(8 Bit Index)
Device Attribute Point 0, Variation 252=Device manufacturers name string
code 1=Visible Characters value UNAM FI
```

4. Escritura del bit de *RESTART*: En la sección de apéndices puede observarse que en las transacciones anteriores a esta aparece lo siguiente:

```
IIN Bits:
IIN1.7 Device Restart
```

La lectura del bit de *RESTART* es un mecanismo del cual se valen las estaciones maestras para saber si un dispositivo se reinició. El valor de ese bit en todos los esclavos deberá ser 1, a menos que el maestro lo haya modificado a 0 mediante la escritura del objeto 80, variación 1, índice 7. Hecho esto, la indicación mostrada anteriormente dejará de aparecer.

5. Solicitud de integridad: Esta solicitud deberá regresar el valor de todos los datos ya sea de tipo estático o evento. En este caso, ya que no se ha modificado el valor de ninguna de las entradas, no debería haber eventos y debería de leerse únicamente el estado de las entradas. El resultado de dicha solicitud fue el siguiente:

```
Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
Binary Input 000001 = 0x81
Binary Input 000002 = 0x81
Binary Input 000003 = 0x81
Binary Input 000004 = 0x81
Binary Input 000005 = 0x81
```

El valor de las entradas muestra 0x81 debido al empaquetamiento de la información (es mostrado como si fuera del grupo 1, variación 2), sin embargo el valor actual de la entrada es de 1. De aquí en adelante, el valor de 0x81 deberá interpretarse como un nivel lógico alto (1) y el valor de 0x01 como un valor lógico bajo (0).

6. Primer lectura de entradas: Se efectúa la lectura del grupo 1 (entradas binarias), variación 1, índices 1 y 2 (correspondientes a las entradas 1 y 2 respectivamente) mediante el calificador 0. El resultado fue el siguiente:

```
Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
Binary Input 000001 = 0x81

Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
Binary Input 000002 = 0x81
```

7. Segunda lectura de entradas: Esta lectura se realizó para los índices 3,4 y 5 (correspondientes a las entradas 3,4 y 5 respectivamente) utilizando el calificador 1. El resultado fue el siguiente:

```
Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
Binary Input 000003 = 0x81

Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
Binary Input 000004 = 0x81

Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
Binary Input 000005 = 0x81
```

8. Tercera lectura de entradas: En este caso se realiza la lectura de las entradas binarias mediante el calificador 6, el cual las implica a todas.

```
Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
Binary Input 000001 = 0x81
Binary Input 000002 = 0x81
Binary Input 000003 = 0x81
Binary Input 000004 = 0x81
Binary Input 000005 = 0x81
```

9. Segunda solicitud de integridad: Antes de realizar esta solicitud, se cambio múltiples veces el estado de la entrada 4 de tal forma que se registraran algunos eventos. El resultado fue el siguiente:

```
Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
Binary Input 000004 = 0x01

Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
Binary Input 000004 = 0x81

Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
Binary Input 000004 = 0x01

Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
Binary Input 000004 = 0x81

Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
Binary Input 000001 = 0x81
Binary Input 000002 = 0x81
Binary Input 000003 = 0x81
Binary Input 000004 = 0x81
Binary Input 000005 = 0x81
```

En esta respuesta se aprecian dos objetos. El primero (grupo 2, variación 1) corresponden a los eventos registrados en la entrada 4, en donde se observan los nuevos estados de la entrada después de varios cambios. El segundo objeto (grupo 1, variación 1) corresponde a los datos estáticos del sistema.

10. Solicitud de eventos: Se realiza la lectura del grupo 60, variaciones 2,3 y 4 (clases 1,2 y 3) pero no sin antes haber variado dos veces el estado de la entrada 3 del sistema. EL resultado de la solicitud fue:

```
Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
Binary Input 000003 = 0x01

Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
Binary Input 000003 = 0x81
```

11. Habilitación del envío de eventos en respuestas no solicitadas: Aunque el módulo DNP3 fue configurado para permitir el envío de este tipo de respuestas, la estación maestra debe de «estar de acuerdo» con dicha función. Esto lo hace habilitando el envío de eventos de las diferentes clases en este tipo de respuestas. El comando que se utilizó en este caso permitió la habilitación del envío de los tres tipos de clases, aunque debido a la configuración del sistema, únicamente se tendrán eventos de clase 1.

12. Arribo de respuesta no solicitada: El valor de la entrada 4 del sistema fue modificado y como consecuencia se recibe una respuesta no solicitada indicando dicho cambio:

```
Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
Binary Input 000004 = 0x01
```

13. Verificación de algunas de las indicaciones internas del sistema: Se envía una solicitud para desactivar el envío de eventos de todas las clases en respuestas no solicitadas. Se cambia el estado de alguna de las entradas de tal forma que se generen eventos. Finalmente, se envía una solicitud de lectura de un objeto no implementado, por ejemplo el grupo 1 variación 2. El resultado de estas acciones fue el siguiente:

```
IIN Bits:
IIN1.1 Class 1 Event Data Is Available
IIN2.1 Object Unknown
```

a) IIN1.1 nos indica que hay eventos de clase 1 contenidos en el buffer de eventos y que corresponden al cambio de estado que se hizo de una de las entradas. Éstos no pueden ser enviados vía respuesta no solicitada ya que esta función fue desactivada previamente.

b) IIN2.1 indica que en la solicitud que se le envió al esclavo se contenía un objeto que éste no pudo reconocer (grupo 1, variación 2).

5.2.2. DNP3 TestSet

El objetivo de esta prueba es el de verificar que el comportamiento del módulo DNP3 con una estación maestra a lo largo de un cierto tiempo sea el esperado.

DNP3 TestSet permite configurar una estación maestra automatizada capaz de enviar solicitudes de integridad, solicitudes de lectura de eventos y de recibir respuestas no solicitadas. En esta prueba se utiliza la misma configuración del módulo DNP3 que se utilizó en la prueba con el programa *Communication Protocol Test Harness* (ver pag. 85), además de estar conectado a la computadora de la misma forma (mediante cable USB - Serial).

La prueba consistió en dejar funcionando el sistema durante 4 horas, en las cuales el valor de las entradas binarias fue modificado únicamente momentos antes de finalizar la prueba. La estación maestra está programada para poder recibir respuestas no solicitadas y para realizar una solicitud de integridad cada 30 minutos.

La configuración de la estación maestra se realiza mediante un archivo XML como el que se muestra en la figura 5.6.

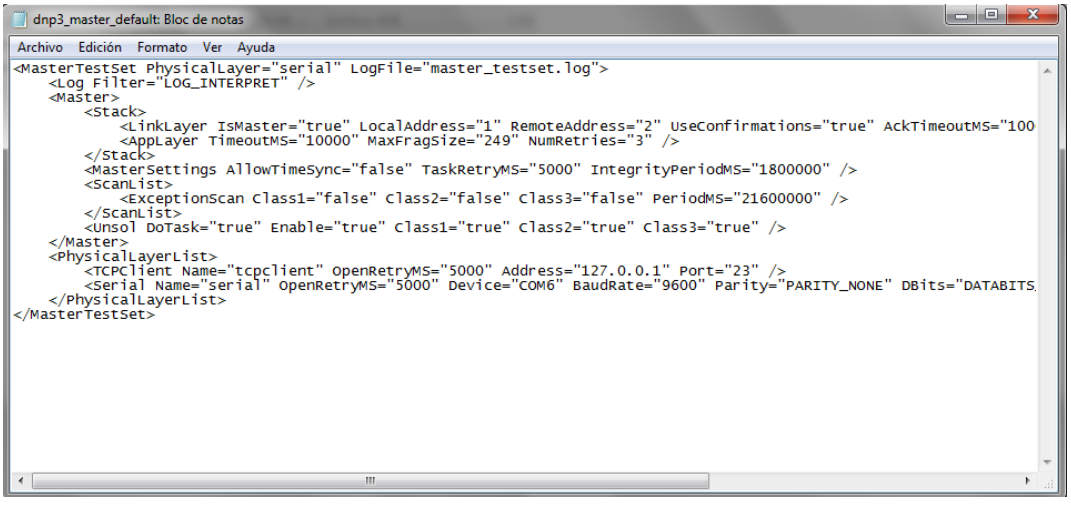


Figura 5.6: Configuración de DNP3 TestSet.

El resultado de la prueba se expresa en el registro de todos los acontecimientos ocurridos en el dispositivo maestro. Para facilitar la explicación, dividí dicho registro en 6 secciones que a continuación explicaré (en la sección de apéndices se incluye el registro completo de la prueba).

1. Sección 0: La comunicación en esta parte del registro es un poco caótica ya que ambos dispositivos se encuentran enviando información al mismo tiempo, por lo que todos los mensajes a nivel de capa de aplicación enviados por el maestro fueron fallidos. Lo único que se logro establecer exitosamente fue la comunicación a nivel de enlace de datos y el reconocimiento por parte del maestro de que el sistema de alarmas acababa de reiniciarse.

2. Sección 1: A partir de esta sección se logra normalizar la comunicación teniendo así la primer petición completada de manera exitosa. Esa petición corresponde a la inhabilitación de las respuestas no solicitadas. En el siguiente cuadro puede observarse dicha transacción a nivel de capa de aplicación:

```
17:31:13.145 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 0, Func: Disable Unsol HdrCount: 3,
    Header: (Grp: 60, Var: 2, Qual: 6, Count: 0)
    Header: (Grp: 60, Var: 3, Qual: 6, Count: 0) Header: (Grp: 60, Var: 4, Qual: 6, Count: 0), Size: 11

17:31:13.793 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 0, Func: Rsp IIN: (LSB: 80 DeviceRestart)
   (MSB: 00) HdrCount: 0, Size: 4

17:31:13.793 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 0, Func: Confirm HdrCount: 0, Size: 2
```

   El primer párrafo constituye la petición del maestro (*Disable Unsol*). *HdrCount* indica el número de objetos en el fragmento, que en este caso son 3: objeto 60, variaciones 2,3 y 4 correspondientes a las clases 1,2 y 3. El segundo párrafo corresponde a la respuesta nula del sistema de alarmas, puede apreciarse que el bit de *RESTART* aún no ha sido modificado. El tercer párrafo corresponde a la confirmación de llegada de la respuesta nula.

3. Sección 2: Se completa con éxito la petición de escritura del bit de *RESTART*:

```
17:31:14.408 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 1, Func: Write HdrCount: 1,
    Header: (Grp: 80, Var: 1, Qual: 0, Count: 1), Size: 8

17:31:14.743 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 1, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 0, Size: 4

17:31:14.743 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 1, Func: Confirm HdrCount: 0, Size: 2
```

4. Sección 3: Se completa con éxito la primera solicitud de integridad.

```
17:31:15.358 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 2, Func: Read HdrCount: 1,
    Header: (Grp: 60, Var: 1, Qual: 6, Count: 0), Size: 5

17:31:15.698 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 2, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 1,
    Header: (Grp: 1, Var: 1, Qual: 0, Count: 5), Size: 10

17:31:15.698 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 2, Func: Confirm HdrCount: 0, Size: 2
```

   El segundo párrafo es la respuesta del sistema a la solicitud. Se observa que únicamente incluye 5 objetos (Count: 5) del grupo 1, variación 1 ya que no se ha registrado ningún evento.

5. Sección 4: Habilitación de envío de respuestas no solicitadas de eventos clase 1,2 y 3.

```
17:31:16.312 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 3, Func: Enable Unsol HdrCount: 3,
    Header: (Grp: 60, Var: 2, Qual: 6, Count: 0)
    Header: (Grp: 60, Var: 3, Qual: 6, Count: 0) Header: (Grp: 60, Var: 4, Qual: 6, Count: 0), Size: 11

17:31:16.644 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 3, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 0, Size: 4

17:31:16.644 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 3, Func: Confirm HdrCount: 0, Size: 2
```

6. Sección 5: En esta sección se incluyen las 7 solicitudes de integridad que realizó el maestro cada 30 minutos. Ya que el estado de las entradas no cambió durante dichas 4 horas, dichas transacciones son prácticamente iguales a la de la sección 3 (primera solicitud de integridad).

7. Sección 6: Esta sección registra 2 respuestas no solicitadas debido a que intencionalmente se cambió el estado de la entrada binaria número 2.

```
21:22:42.721 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 1, SEQ: 0, Func: Unsol Rsp IIN: (LSB: 00) (MSB: 00)
     HdrCount: 1, Header: (Grp: 2, Var: 1, Qual: 23, Count: 1), Size: 10

21:22:42.721 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 1, SEQ: 0, Func: Confirm HdrCount: 0, Size: 2
```

El primer párrafo corresponde a la respuesta no solicitada. Puede observarse que se envía el valor de un objeto de grupo 2, variación 1 (evento de entrada binaria). Si bien en el registro no se aprecia el cambio de dicho valor, este puede constatarse en la ventana principal del programa mediante el comando *show*, el cual muestra el estado de todos los puntos del dispositivo esclavo.

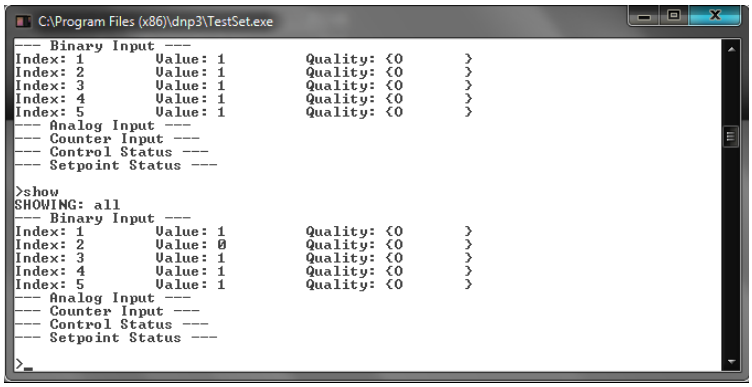


Figura 5.7: Cambio de estado de la entrada binaria número 2.

En la figura 5.7, se ejecuta el comando *show* antes y después de el cambio de valor de la entrada. Posteriormente el valor es regresado a su estado anterior, lo que provoca el envío de una segunda respuesta no solicitada:

```
21:24:11.712 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 1, SEQ: 1, Func: Unsol Rsp IIN: (LSB: 00) (MSB: 00)
     HdrCount: 1, Header: (Grp: 2, Var: 1, Qual: 23, Count: 1), Size: 10

21:24:11.712 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 1, SEQ: 1, Func: Confirm HdrCount: 0, Size: 2
```

Una tercera ejecución del comando *show* muestra dicho cambio (figura 5.8).

5.2.3. Prueba de la función *COLD_RESTART*

Esta prueba verificará el buen funcionamiento del mecanismo de reinicio por hardware. Ya que ninguno de los programas anteriores implementa la función, es necesario utilizar el programa

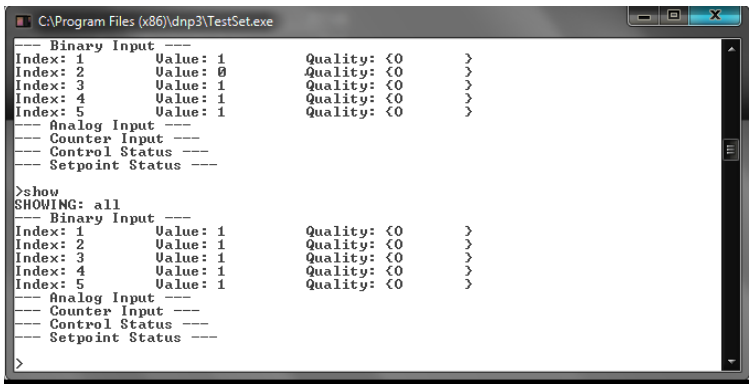


Figura 5.8: Segundo cambio.

Hercules Setup Utility para comunicarse con el módulo. La prueba se realizará mediante el envío de una trama que incluya la función *COLD_RESTART*, para la cual, el sistema debe reiniciarse y estar operativo en un plazo de 3 segundos. Los pasos que se siguieron para su realización son:

1. Configurar el *Hercules Setup Utility* para recibir caracteres hexadecimales. Esto se hace mediante un click derecho en la sección *Received/Sent data* y seleccionando *Special Chars* → *Hexadecimal* y *Hex enable*.
2. Configurar el módulo DNP3 para mandar respuestas no solicitadas e inhabilitar la confirmación a nivel de capa de aplicación.
3. Se envía la siguiente trama:

   `05 64 08 E4 02 00 01 00 8E AF C0 C3 0D 37 36`

   En donde la antepenúltima cifra (0D) es el código de función de capa de aplicación correspondiente a *COLD_RESTART*.
4. Se espera que el sistema responda con un objeto del grupo 52 en el que se indica que el sistema no estará disponible durante tres segundos.
5. Tres segundos después, se recibirá una respuesta no solicitada indicando que el sistema se ha reiniciado.

Resultados

En la imagen 5.9 pueden observarse los resultados exitosos de la prueba. La primera trama corresponde a la respuesta no solicitada del sistema indicando que acaba de reiniciarse. La segunda es la que se envía al sistema con el código de función 0D. La tercera contiene el tiempo en que no estará disponible el sistema y tres segundos después llega la última indicando nuevamente que el sistema se reinició.

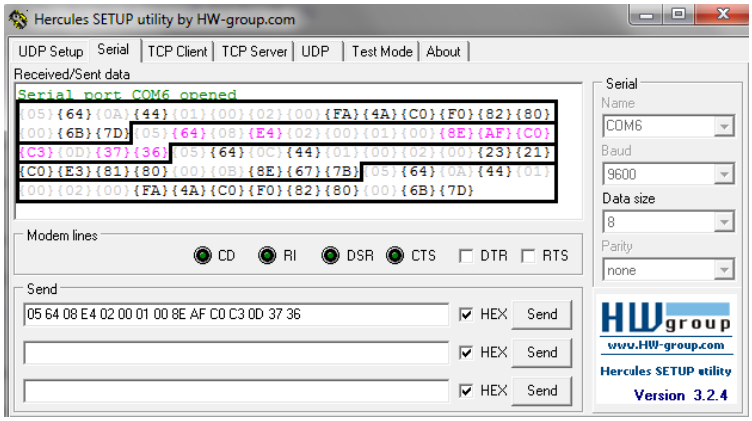


Figura 5.9: Resultados de la prueba.

Capítulo 6

Librería DNP3

El contenido de este capítulo se deriva del estudio del protocolo DNP3 y de la experiencia adquirida en su implementación para el sistema de alarmas. Aunque dicho contenido se aleja un poco de lo que es el objetivo del trabajo considero que su inclusión es importante ya que representa un alternativa a la implementación original del software, además de que aporta nuevas funcionalidades y características que podrían parecerle interesantes al lector.

6.1. Introducción

El objetivo de este trabajo fue el de obtener una librería DNP3 de tal forma que se pudieran configurar puntos de distintos tipos (entradas/salidas binarias y entradas/salidas analógicas) de una manera sencilla. De esta forma se facilitaría la creación de estaciones esclavo configuradas por el usuario y personalizadas para las necesidades de éste.

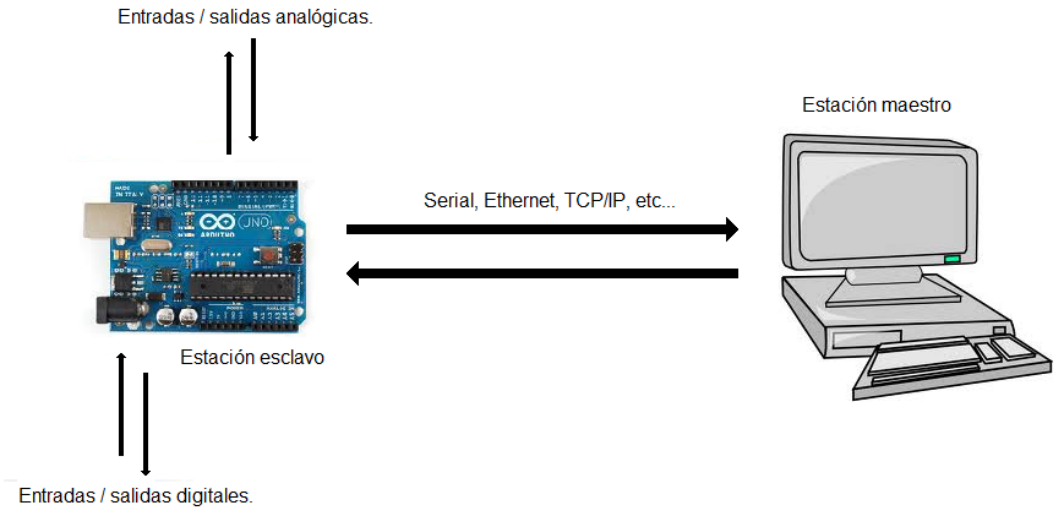


Figura 6.1: Estación esclavo.

6.2. Desarrollo

Teniendo ya la experiencia de la implementación del protocolo de manera secuencial, decidí seguir investigando acerca de la ejecución de múltiples procesos al mismo tiempo sobre microcontroladores, ya que considero que el protocolo puede implementarse de una manera más natural de esta forma. Después de varias opciones que descarté por utilizar una gran cantidad de recursos del sistema, encontré la librería Protothreads. Si bien no permite la ejecución multihilo, permite crear hilos o procesos que pueden ser manejados por eventos. La librería fue diseñada y escrita por Adam Dunkels especificamente para sistemas embebidos o sistemas de recursos limitados, lo cual la hace una herramienta perfecta para esta aplicación. Además de la creación de hilos, permite el uso de semáforos para el manejo de recursos compartidos entre los diferentes procesos.

En cuanto al hardware utilizado se siguió ocupando el microprocesador ATmega328 montado sobre una placa arduino UNO, por lo que el lenguaje Arduino (el cual es C/C++ con algunas funciones para controlar los módulos del ATmega328) fue utilizado.

El esquema propuesto consta de 7 procesos:

1. **Capa física de entrada**: Se encarga de manejar las tramas que arriban vía puerto serial.
2. **Capa física de salida**: Se encarga de enviar las tramas que requieren ser enviadas por la capa de enlace de datos ya sea en modo primario o secundario vía puerto serial.
3. **Capa de enlace de datos modo secundario**: Se encarga de recibir y procesar las tramas recién capturadas por la capa física de entrada para ser convertidas en segmentos de la capa de transporte.
4. **Capa de enlace de datos modo primario**: Se encarga de convertir los segmentos de la capa de transporte en tramas y de enviarlos a la capa física de salida para su transmisión.
5. **Capa de transporte**: Convierte a los fragmentos de la capa de aplicación en segmentos que recibirá la capa de enlace en su modo primario. También recibe los segmentos de la capa de enlace en modo secundario y los convierte en fragmentos.
6. **Capa de aplicación**: Se encarga de procesar las peticiones que llegan en forma de fragmentos y de enviar las respuestas adecuadas. Además monitorea constantemente el estado de los puntos del sistema y de esta forma genera eventos y envía respuestas no solicitadas. Está ejecutándose continuamente.
7. **Asignación de puntos**: Este proceso es ajeno al protocolo al igual que las capas físicas de entrada y salida. Se encarga de asignar y de actualizar el valor de los diferentes puntos del sistema. Se ejecuta continuamente.

De esta forma las capas del protocolo pueden ser implementadas como lo sugiere la norma IEEE 1815 - 2010, en forma de máquinas de estado.

Otra ventaja que proporciona la librería son los semáforos. Al haberse mantenido la estructura original de los buffers (ver página 38), cabe la posibilidad de que un proceso quiera acceder y modificar uno de ellos mientras otro está haciendo uso de él. Con la ayuda de los semáforos, el acceso a los buffers se restringe de tal forma que solo un proceso puede acceder a la vez. En caso de que algún otro proceso intente acceder, tendrá que esperar a que el buffer se desocupe.

6.3. Limitaciones

Durante el desarrollo de este proyecto surgieron principalmente dos limitantes:

1. **Falta de memoria FLASH en el microcontrolador para albergar más código**. Este problema se traduce en la limitación de los objetos que pueden ser utilizados a la vez. Por esta razón, se decidió implementar únicamente las entradas analógicas y digitales y excluir las salidas analógicas y digitales. Por lo tanto, el programa no podrá controlar dispositivos y servirá únicamente para la adquisición de datos. A continuación se presenta la tabla de objetos y calificadores aceptados.

Grupo	Variación	Descripción	Acepta peticiones con	Genera respuestas con
1	1	Entradas binarias formato: empaquetado	Cod. Func. = 01 Calif. = 0x00, 0x06	Cod. Func. = 129 Calif. = 0x00
2	1	Evento de entrada binaria	Cod. Func. = 01 Calif. = 0x06	Cod. Func. = 129, 130 Calif. = 0x17
60	1	Datos clase 0	Cod. Func. = 01 Calif. = 0x06	Cod. Func. = 129 Calif. = depende de los datos pedidos
60	2	Datos clase 1	Cod. Func. = 01, 20, 21 Calif. = 0x06	Cod. Func. = 129, 130 Calif. = depende de los datos pedidos
60	3	Datos clase 2	Cod. Func. = 01, 20, 21 Calif. = 0x06	Cod. Func. = 129, 130 Calif. = depende de los datos pedidos
60	4	Datos clase 3	Cod. Func. = 01, 20, 21 Calif. = 0x06	Cod. Func. = 129, 130 Calif. = depende de los datos pedidos
80	1	Indicaciones internas	Cod. Func. = 02 Calif. 0x00 (índice 7)	Cod. Func. = 129

Cuadro 6.1: Grupos y variaciones aceptados.

2. **Memoria RAM restringida**. Según la norma IEEE 1815 - 2010, una estación esclavo debe de valerse de mensajes multifragmento en caso de que la información requerida por el maestro supere el tamaño de éste. Sin embargo debido a la poca memoria RAM con la que se cuenta, el tamaño de un fragmento se fija en 249 bytes y no se permiten los mensajes multifragmento. De tal forma que cualquier fragmento que requiera de una mayor extensión

será truncado en 249 bytes, perdiéndose así información que pudiera ser importante. Es por esta razón que se debe tener cuidado con el número de entradas que se declaran.

6.4. Instrucciones

Para configurar la librería se tiene que abrir el archivo «planilla», el cuál contiene lo siguiente:

```
#include "dnp3.h"

//---------------------------------------------Aquí se declaran los puntos de la estación esclavo.


void setup()//-----------------------------------Función setup de arduino
{

//---------------------------------------------Configuración del microcontrolador. Aquí se configuran los diferentes módulos del
                                               //microcontrolador como la velocidad del puerto serial y los pines que serán utilizados
                                               //como entradas.

Serial.begin(9600);

//---------------------------------------------Configuración del protocolo. Se configuran diversos parámetros del protocolo así
                                              // como algunas características de los puntos que fueron creados anteriormente.

direccion_local = 2;                          // <----- Dirección de éste dispositivo
direccion_maestro = 1;                        // <----- Dirección del dispositivo maestro
modo_confirmacion_DL = sin_confirmacion;      // <----- Confirmación a nivel de enlace de datos (con_confirmacion / sin_confirmacion)
retry = 3;                                    // <----- Número de reenvíos de la capa de enlace de datos
tiempo_espera_DLL = 800;                      // <----- Tiempo de espera de la capa de enlace de datos (en milisegundos)
respuestas_no_solicitadas = inhabilitadas;    // <----- Envío de respuestas no solicitadas (inhabilitadas / habilitadas)
tiempo_espera_AL = 4000;                      // <----- Tiempo de espera de la capa de aplicación

}

void asignaciones()//--------------------------En esta función se realizan las asignaciones de valores a los puntos de la estación.
{

}

void fallo_aplicacion()//-----------------------Se ejecuta un proceso en caso de que no se reciba una
                                               //confirmación a nivel aplicación en el tiempo especificado.
{

}

void fallo_de_com()//---------------------------Se ejecuta un proceso en caso de que surja una falla en la comunicación entre capas
                                              //de enlace de datos cuando modo_confirmacion_DL = con_confirmacion.
{

}

void loop()//------------------------------------Función loop de arduino
{
iniciar_com();//-------------------------------Esta función inicia la comunicación DNP3
}
```

En la primera parte se declaran los puntos de la estación de la siguiente forma:

```
objeto *"nombre del punto" = new_"tipo de entrada"_var"número de variacion"();
```

En el siguiente ejemplo se declararán 5 variables. Dos entradas binarias variación 1, dos entradas analógicas variación 4 y una entrada analógica variación 5.

```
objeto *entrada_binaria_1 = new entrada_binaria_var1();
```

```
objeto *entrada_binaria_2 = new entrada_binaria_var1();
objeto *entrada_analogica_1 = new entrada_analogica_var4();
objeto *entrada_analogica_2 = new entrada_analogica_var4();
objeto *entrada_analogica_3 = new entrada_analogica_var5();

short contador = 0;// Esta variable se explicará posteriormente.
```

Dentro de la función setup de arduino se realiza la configuración tanto del microcontrolador como la del protocolo. Se comienza configurando la comunicación serial y habilitando todos los pines y módulos que se vayan a utilizar para adquirir los datos. Es importante mencionar que las entradas analógicas y binarias no necesariamente deben de provenir de una medición. Éstas también pueden ser el resultado de un proceso o de una operación. Retomando el ejemplo anterior, el microcontrolador se configurará de la siguiente forma, la cual será explicada posteriormente:

```
pinMode(8,INPUT); //el pin 8 del microcontrolador fue configurado como una entrada digital
pinMode(9,INPUT); //al igual que el pin 9.
```

En la configuración del protocolo se ajustan todos los parámetros de acuerdo a las necesidades del usuario y a las características del hardware. Para el ejemplo se utilizará la configuración que viene por defecto. Dentro de esta misma sección se configuran los parámetros de los puntos que fueron creados anteriormente. Para las entradas binarias se debe especificar un índice y opcionalmente la clase de los eventos que generan (por defecto, las entradas binarias no generarán ningún evento). Para las entradas analógicas únicamente se asignará un índice. Algunas recomendaciones para la asignación de índices son:

1. El primer índice será el cero y se irá incrementando en 1.
2. Cada grupo de puntos deberá tener una serie de índices propia. De esta forma, las entradas binarias estarán numeradas de 0 a N y las entradas analógicas de 0 a M.

Tomando en cuenta estas consideraciones se realiza lo siguiente:

```
entrada_binaria_1->set_indice(0); //El índice de entrada_binaria_1 será 0.
entrada_binaria_2->set_indice(1); //El índice de entrada_binaria_2 será 1.
entrada_analogica_1->set_indice(0); //El índice de entrada_analogica_1 será 0.
entrada_analogica_2->set_indice(1); //El índice de entrada_analogica_2 será 1.
entrada_analogica_3->set_indice(2); //El índice de entrada_analogica_3 será 2.
```

Al no haber seleccionado una clase para las entradas binarias, estas no generarán ningún evento.

En la función *asignaciones()* es donde se le asigna a cada punto un determinado valor. Esta función está incluida dentro del proceso *Asignación de puntos*, por lo que está ejecutándose continuamente actualizando así el valor de éstos. Para el caso del ejemplo, se asignara a entrada_binaria_1 el valor de el pin 8 del microcontrolador (el cual fue configurado como entrada digital) y de manera similar se hará lo mismo con entrada_binaria_2 y el pin 9.

```
entrada_binaria_1->set_valor((unsigned char)digitalRead(8));  //El valor del punto entrada_binaria_1 está ahora ligado al del pin 8.
entrada_binaria_2->set_valor((unsigned char)digitalRead(9));  //El valor del punto entrada_binaria_2 está ahora ligado al del pin 9.
```

En cuanto a las 3 entradas analógicas, sus valores se asignarán de la siguiente forma:

1. A el punto entrada_analogica_1 se le asignará el valor del pin 0, el cuál corresponde a una entrada del convertidor analógico digital (pin A0 del arduino).
2. A el punto entrada_analogica_2 se le asignará el valor de la variable *contador*, la cual fue definida anteriormente.
3. A entrada_analogica_3 se le asignará un valor constante arbitrario (-52.37).

```
entrada_analogica_1->set_valor((short)(analogRead(0)));
entrada_analogica_2->set_valor((short)contador);
entrada_analogica_3->set_valor((float)(-52.37));
```

Es importante verificar el tipo de dato que se le es asignado a cada uno de los puntos. En el ejemplo esto no representa ningún problema con las entradas binarias ya que solo pueden tomar dos valores, sin embargo, un dato con un formato erróneo asignado a una entrada analógica puede representar el envío de datos falsos. Un ejemplo de un posible error sería el de asignar la constante de tipo flotante -52.37 a entrada_analogica_2, ya que al ser ésta de variación 4, solo acepta números enteros de 16 bits.

La función *fallo_aplicacion()* se ejecuta cuando la capa de aplicación no recibe una confirmación en el tiempo especificado anteriormente después de haber enviado una respuesta. De manera similar, *fallo_de_com()* se ejecuta cuando existe una falla en la comunicación a nivel de enlace de datos. Esta opción solo puede darse cuando las confirmaciones en la capa de enlace están activadas (modo_confirmacion_DL = con_confirmacion). Dentro de estas funciones puede o no escribirse las acciones a tomar en caso de un fallo. En el ejemplo, la variable contador aumentara su valor en 1 para cada fallo en la comunicación:

```
void fallo_aplicacion()
{
  contador++;
}

void fallo_de_com()
{
  contador++;
}
```

Por último, dentro de la función *loop()* de arduino se ejecuta la función *iniciar_com()*. Ésta es la encargada de inicializar todos los procesos y de mantenerlos ejecutándose.

La plantilla quedaría entonces de la siguiente forma:

```
#include "dnp3.h"

//-----------------------------------------------Aquí se declaran los puntos de la estación esclavo.

objeto *entrada_binaria_1 = new entrada_binaria_var1();
objeto *entrada_binaria_2 = new entrada_binaria_var1();
objeto *entrada_analogica_1 = new entrada_analogica_var4();
objeto *entrada_analogica_2 = new entrada_analogica_var4();
objeto *entrada_analogica_3 = new entrada_analogica_var5();

short contador = 0;// Esta variable se explicará posteriormente.
```

```
void setup()//----------------------------------Función setup de arduino
{

//-------------------------------------------Configuración del microcontrolador. Aquí se configuran los diferentes módulos del
                                             //microcontrolador como la velocidad del puerto serial y los pines que serán utilizados
                                             //como entradas.

Serial.begin(9600);
pinMode(8,INPUT); //el pin 8 del microcontrolador fue configurado como una entrada digital
pinMode(9,INPUT); //al igual que el pin 9.

//-------------------------------------------Configuración del protocolo. Se configuran diversos parámetros del protocolo así
                                            // como algunas características de los puntos que fueron creados anteriormente.

direccion_local = 2;                        // <----- Dirección de éste dispositivo
direccion_maestro = 1;                      // <----- Dirección del dispositivo maestro
modo_confirmacion_DL = sin_confirmacion;    // <----- Confirmación a nivel de enlace de datos (con_confirmacion / sin_confirmacion)
retry = 3;                                  // <----- Número de reenvíos de la capa de enlace de datos
tiempo_espera_DLL = 800;                    // <----- Tiempo de espera de la capa de enlace de datos (en milisegundos)
respuestas_no_solicitadas = inhabilitadas;  // <----- Envío de respuestas no solicitadas (inhabilitadas / habilitadas)
tiempo_espera_AL = 4000;                    // <----- Tiempo de espera de la capa de aplicación

entrada_binaria_1->set_indice(0); //El índice de entrada_binaria_1 será 0.
entrada_binaria_2->set_indice(1); //El índice de entrada_binaria_2 será 1.
entrada_analogica_1->set_indice(0); //El índice de entrada_analogica_1 será 0.
entrada_analogica_2->set_indice(1); //El índice de entrada_analogica_2 será 1.
entrada_analogica_3->set_indice(2); //El índice de entrada_analogica_3 será 2.

}

void asignaciones()//--------------------------En esta función se realizan las asignaciones de valores a los puntos de la estación.
{
entrada_binaria_1->set_valor((unsigned char)digitalRead(8));  //El valor del punto entrada_binaria_1 está ahora ligado al del pin 8.
entrada_binaria_2->set_valor((unsigned char)digitalRead(9));  //El valor del punto entrada_binaria_2 está ahora ligado al del pin 9.

entrada_analogica_1->set_valor((short)(analogRead(0)));
entrada_analogica_2->set_valor((short)contador);
entrada_analogica_3->set_valor((float)(-52.37));
}

void fallo_aplicacion()//-----------------------Se ejecuta un proceso en caso de que no se reciba una
                                             //confirmación a nivel aplicación en el tiempo especificado.
{
  contador++;
}

void fallo_de_com()//--------------------------Se ejecuta un proceso en caso de que surja una falla en la comunicación entre capas
                                            //de enlace de datos cuando modo_confirmacion_DL = con_confirmacion.
{
  contador++;
}

void loop()//-----------------------------------Función loop de arduino
{
iniciar_com();//-------------------------------Esta función inicia la comunicación DNP3
}
```

6.5. Resultados

La librería fue probada utilizando el mismo software y mediante un procedimiento similar al de la tesis. Se probaron cada uno de los posibles estados de todas las capas que componen al protocolo y se verificó que su comportamiento fuera el adecuado. No obstante, en esta sección se omitirá el análisis de las tramas por ser muy parecido al que se incluye en el capítulo «Pruebas y resultados» y en el apéndice «Datos de la sección de pruebas». Únicamente se describirá el comportamiento de la interfaz HMI (humano - máquina) de dos programas diferentes que implementan el protocolo DNP3 y fungen como dispositivos maestro. Dichos programas se ejecutan en una PC a la cuál se conecto mediante un cable serial a la estación esclavo configurada

anteriormente.

DNP3 TestSet

Este programa, además de que permite analizar el intercambio de tramas entre dos dispositivos DNP3, también permite ver el estado de los puntos de las estaciones conectadas a él. Para el caso del ejemplo, se obtuvo la siguiente información:

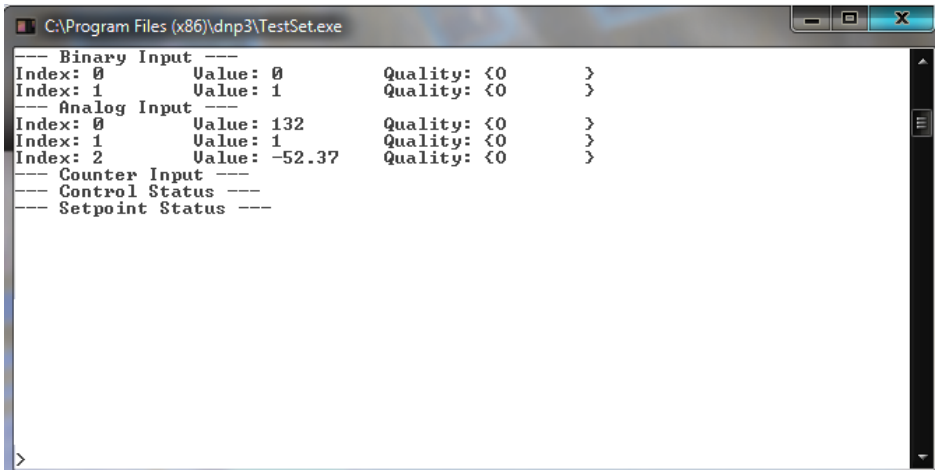


Figura 6.2: Puntos de la estación esclavo en DNP3 TestSet.

Primeramente se tienen las dos entradas binarias. Sus valores corresponden al estado de los pines del arduino 8 y 9. La entrada analógica con el índice 0 puede tomar valores de 0 hasta 1023 ya que está ligada a una de las entradas del convertidor analógico - digital. En este caso, su valor (132) es aleatorio ya que no se encuentra nada conectado al pin A0 del arduino. La entrada analógica con el índice 1 tiene un valor de 1, ya que al parecer ocurrió una falla en la comunicación. Por último, en la entrada analógica con índice 2, se tiene el valor constante que se había definido previamente (-52.37).

SCADA BR

Este software funciona como una estación maestra SCADA que admite distintos protocolos, entre ellos el DNP3. Cuenta con muchas más funciones que el DNP3 TestSet como la de graficar los puntos, ver el valor de los puntos en diferentes estados de tiempo, entre otras, sin embargo, el número de objetos que maneja es mucho más reducido.

El voltaje en los pines 8 y 9 no cambió, por lo que se puede apreciar el mismo valor que en el programa anterior. La entrada analógica 0 tiene un valor aleatorio ya que aún no se ha conectado nada a el pin A0 del arduino. La entrada analógica 1 representa el número de veces que ha fallado la comunicación, que en este caso han sido 7. Por último se tiene que el valor de la entrada analógica 2 no ha podido ser adquirido, esto se debe a que el grupo 30 variación 5

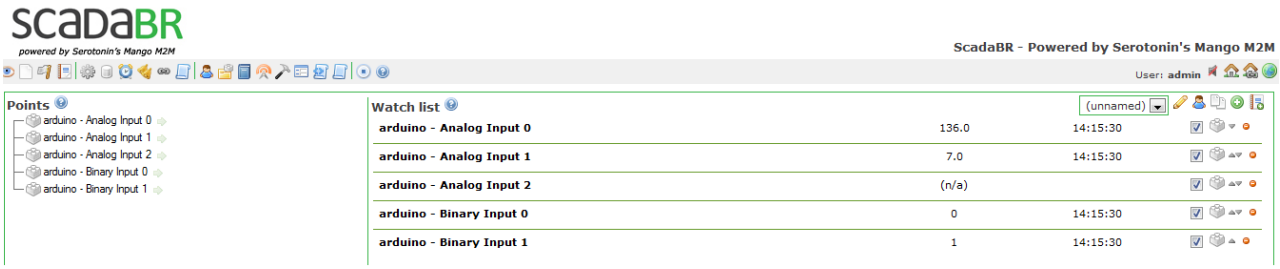


Figura 6.3: Puntos de la estación esclavo en SCADA BR.

(al cual pertenece dicha entrada) no está implementado en SCADA BR.

Capítulo 7

Conclusiones y trabajos futuros

- La interfaz RS-232 permite implementar una conexión de punto a punto que puede extenderse si se añaden dispositivos como adaptadores ethernet, transmisores, receptores de radio, etc.
- La velocidad de una transacción entre un maestro y el módulo DNP3 está determinada por la velocidad de éstos y la del canal de comunicación. Con base en los datos de las pruebas realizadas se determinó que la velocidad de una transacción (serie de mensajes) es de aproximadamente 900 [ms] para una conexión serial con una PC. Este tiempo será entonces el mínimo del que puede obtenerse información del módulo DNP3, por lo que no es apto para aplicaciones que necesiten recabar información en intervalos de tiempo inferiores a éste.
- La implementación del protocolo no incluye la autenticación segura (mecanismo ocpional que provee protección contra ataques). Esto quiere decir que el sistema es susceptible a diversas formas de ataques e intrusiones. Un ejemplo podría ser un ataque de denegación de servicio el cual podría consistir en enviar continuamente mensajes DNP3 basura que impedirían al módulo recibir los verdaderos mensajes (recordemos que solo pueden recibirse 2 mensajes consecutivos). También es relativamente fácil suplantar la identidad del maestro, por lo que mediante ésta técnica podrían conocerse los estados de las entradas del sistema.
- Para implementaciones DNP3 sencillas como la de este proyecto, microprocesadores como el ATmega328 son apenas suficientes. Si se quisiera extender la funcionalidad del protocolo se requerirían componentes adicionales como memorias, relojes, o bien algún otro dispositivo con mayores capacidades y funcionalidades.
- El programa del capítulo «Librería DNP3» permite crear aplicaciones DNP3 sencillas y configurables, sin embargo, se ven limitadas por el hecho de que no cumplen con ninguno de los niveles de implementación especificados en el estándar.
- Un posible trabajo sería la creación de una librería en la que se implemente el protocolo de una manera más modular. Si bien, en la implementación del proyecto las capas están

debidamente limitadas, sería difícil expandirlo. Las capas de enlace de datos y de transporte no representan un gran problema ya que prácticamente deben de ser iguales en todos los dispositivos. Éstas podrían configurarse mediante una serie de comandos para adaptarse y limitarse al tipo de hardware que se decida utilizar. Para la capa de aplicación se podría proporcionar una manera sencilla de programar su comportamiento. Por ejemplo, el usuario de la librería podría programar el proceso a seguir en caso de la llegada de un fragmento con determinado código de función, determinado objeto y determinada variación.

Referencias y obras consultadas

[1] IEEE Power & Energy Society. *IEEE Standard for Electric Power Systems Communications–Distributed Network Protocol (DNP3), IEEE STD 1815-2010.*

[2] Gordon Clarke y Deon Reynders (2004) *Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems.* Gran Bretaña: Editorial Elsevier.

[3] Andrew S. Tanenbaum (2003). *Redes de computadoras.* México: Editorial PEARSON EDUCACIÓN.

[4] Julián Alejandro Villalba Márquez (2010). *ESTUDIO Y PRUEBAS DEL PROTOCOLO DE COMUNICACIÓN DNP3.0 SOBRE TCP/IP PARA LA COMUNICACIÓN ENTRE LA CENTRAL DE GENERACIÓN CUMBAYÁ DE LA EMPRESA ELÉCTRICA QUITO S.A. Y EL CENACE.* Tesis de licenciatura, Escuela Politécnica Nacional, Ecuador, Quito.

[5] Dong-joo y Rosslin John Roble (2009). *Compartmentalization of Protocols in SCADA Communication.* International Journal of Advanced Science and Technology, Volumen 8.

[6] Polo Francisco Padilla Monroy (2006). *DESARROLLO DE UN SISTEMA DE MEDICIÓN DE VARIABLES ELÉCTRICAS PARA UN SISTEMA DE BAJA TENSIÓN TIPO INDUSTRIAL.* Tesis de maestría, Instituto Politécnico Nacional, México.

[7] Ken Curtis (20, marzo, 2005). *A DNP3 protocol primer.* DNP3 Users Group, 2000, 2005.

[8] DPS Telecom (8, agosto, 2011). *SCADA Tutorial: A Quick, Easy, Comprehensive Guide.*

[9] Ing. Henry Mendiburu Díaz. *Sistemas SCADA, Fundamento teórico.* [En línea]. Consultado: [16, julio, 2012] Disponible en: http://hamd.galeon.com/

[10] Thomas M. Lebakken y Dominic R. Orlando, *Substation Automation and Communication Standards: IEC 61850 and DNP3.* [En línea]. Consultado: [16, julio, 2012] Disponible en: http://www.elp.com/index/display/article-display/302693/articles/utility-automation-engineering-td/volume-12/issue-8/features/substation-automation-and-communication-standards-iec-61850-and-dnp3.html

[11] CONTROL MICROSYSTEMS (18, junio, 2007). *DNP3: User and Reference Manual.*

[12] Triangle MicroWorks (22, febrero, 2002). *DNP3 Overview.*

[13] Mr. Jim. Coats (Presidente de Triangle MicroWorks) (19, agosto, 2002). *DNP3 Protocol: AGA/GTI SCADA Security Meeting*. Conferencia presentada en Washington, DC.

[14] Electus Distribution. *Electus Distribution Reference Data Sheet: RELAYDRV.PDF (I), RELAY DRIVING BASICS*.

[15] Héctor Tejeda V. *Manual de C*. [En línea]. Consultado: [16, julio, 2012] Disponible en: http://www.fismat.umich.mx/mn1/manual/node1.html

[16] *Referencia del lenguaje Arduino.* [En línea]. Consultado: [16, julio, 2012] Disponible en: http://arduino.cc/es/Reference/HomePage

[17] *The Server Rack FAQ, information about rack issues and racking servers.* [En línea]. Consultado: [16, julio, 2012] Disponible en: http://www.server-racks.com/eia-310.html

[18] *ARDUINO PLAYGROUND.* [En línea]. Consultado: [16, julio, 2012]. Disponible en: http://arduino.cc/playground/Main/HomePage

[19] Página de la libreria *AVR Libc*. http://www.nongnu.org/avr-libc/

[20] FAIRCHILD SEMICONDUCTOR. *2N7000 datasheet*.

[21] FAIRCHILD SEMICONDUCTOR. *LM78xx datasheet*.

[22] ATMEL. *8 bit AVR microcontroller with 4/8/16/32K bytes In-System Programmable Flash*.

[23] Texas Instruments. *MAX232 datasheet*.

[24] National Semiconductor. *LM555 datasheet*.

[25] Atmel Corporation. *Atmel AVR042: AVR Hardware Design Considerations*.

[26] SEMTECH ELECTRONICS LTD. *2W005G THRU 2W10G datasheet*.

[27] Sitio de descarga del programa *DNP3 Test Set*. http://code.google.com/p/dnp3/

[28] *Online CRC Calculation*: http://www.lammertbies.nl/comm/info/crc-calculation.html#intr

[29] Sitio de descarga del programa *Communication Protocol Test Harness*: http://www.trianglemicroworks.com/downloads.htm

[30] Sitio de descarga del programa *Hercules SETUP Utility*: http://www.hw-group.com/products/hercules/index_en.html

Apéndice A

Librería de objetos utilizados

Se muestra la forma de empaquetar la información de cada uno de los objetos utilizados en este proyecto.

DNP3 Object Library

		Group:	0
		Variation:	242
Group Name:	Device Attributes	Type:	Attrib
Variation Name:	Device manufacturer's software version		

Description

This attribute is the version code of the manufacturer's device software.

The contents of this attribute is a free form string that is formatted according to the manufacturer's normal practice.

Two examples are

"5.3.006" and "1.12:2003-11-25-Standard."

Coding

Pictorial

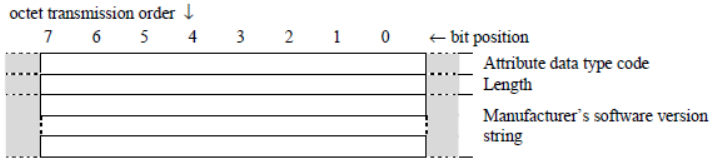


Figura A.1: Grupo 0, variación 242.

DNP3 Object Library

		Group:	0
		Variation:	243
Group Name:	Device Attributes	Type:	Attrib
Variation Name:	Device manufacturer's hardware version		

Description

This attribute is the version code of the manufacturer's device hardware.

The contents of this attribute is a free form string that is formatted according to the manufacturer's normal practice.

Two examples are

"Rev D" and "2004-122."

Coding

Pictorial

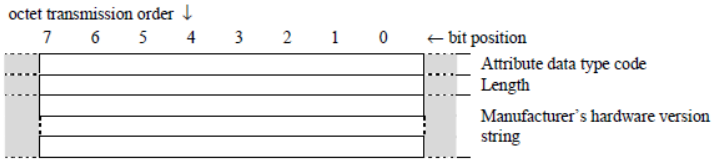


Figura A.2: Grupo 0, variación 253.

DNP3 Object Library

		Group:	0
		Variation:	250
Group Name:	Device Attributes	Type:	Attrib
Variation Name:	Device manufacturer's product name and model		

Description

This attribute is the device manufacturer's product name and model.

The content of this attribute is a free form string that identifies an industry recognizable trade name and model. The string should not include the manufacturer's name as that appears in attribute variation 252.

Several examples are

"Callisto," "D25 IED," "Form 5 Recloser," "SEL-351 Relay," "NTU-7500," "PDS Magna," "IntelliCap," and "Multicomm."

Coding

Pictorial

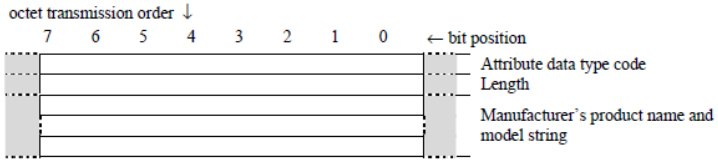


Figura A.3: Grupo 0, variación 250.

DNP3 Object Library		Group:	0
		Variation:	252
Group Name:	Device Attributes	Type:	Attrib
Variation Name:	Device manufacturer's name		

Description

This attribute is the name of the device manufacturer. An example is "123 SCADA, Ltd."

Coding

Pictorial

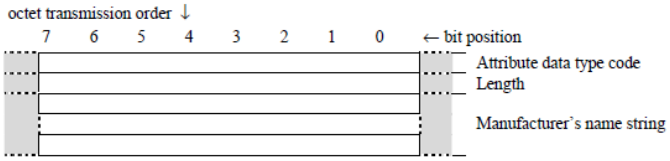


Figura A.4: Grupo 0, variación 252.

DNP3 Object Library		Group:	0
		Variation:	254
Group Name:	Device Attributes	Type:	Attrib
Variation Name:	Non-specific all attributes request		

Description

This attribute is used as a shorthand to request an outstation to return all of its attributes in a single response. The master sends a single object header with this variation in lieu of including a possibly lengthy list of object headers in the request.

Coding

This variation does not have objects.

NOTE 1—This variation may only appear in a master request. It shall not be used in responses from outstations.

NOTE 2—Requests from a master use group 0, variation 254 qualifier code

— 0x00 where the start-stop range field indexes indicate from which set or sets of outstation attributes. Index 0 is the set of attributes defined by the DNP User's Group. Other indexes specify privately defined or vendor-specific attribute sets.

— 0x06.

NOTE 3—Devices that implement attributes are required to support this variation.

Figura A.5: Grupo 0, variación 254.

DNP3 Object Library		Group:	0
		Variation:	255
Group Name:	Device Attributes	Type:	Attrib
Variation Name:	List of attribute variations		

Description

This is a special attribute number that is used to retrieve a list of all of the device attribute variation numbers supported by the outstation at a specified index,[41] and the properties of those attributes. This object has a variable length that depends on the count of attribute variations supported by the outstation.

Coding

Pictorial

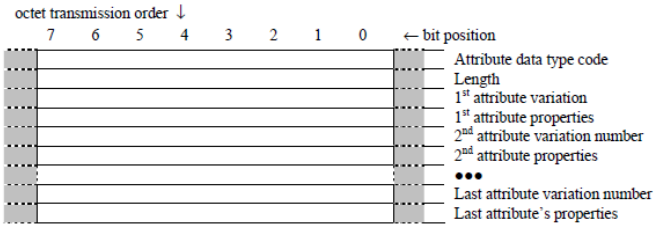


Figura A.6: Grupo 0, variación 255.

DNP3 Object Library		Group:	1
		Variation:	1
Group Name:	Binary Input	Type:	Static
Variation Name:	Packed format		

Description

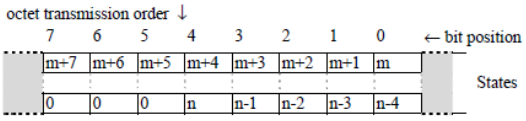
Object group 1, variation 1 is used to report the current value of a binary input point. See 11.1 for a description of a Binary Input Point Type.

Variation 1 objects contain a single-bit binary input state without status flags.

Coding

This object is coded as a single bit, with a possible value of 0 or 1, for each point index. When multiple points are contained in a single response message, they are packed, as illustrated in the following pictorial.

Pictorial



This figure depicts the bit-packing sequence for a response containing bit indexes m through n. As can be seen, the first bit is in the bit 0 position of the first octet, the second bit is in the bit 1 position of the first octet, etc. Any unused bits in the final octet are returned as 0.

Figura A.7: Grupo 1, variación 1.

DNP3 Object Library

		Group:	2
		Variation:	1
Group Name:	Binary Input Event	Type:	Event
Variation Name:	Without time		

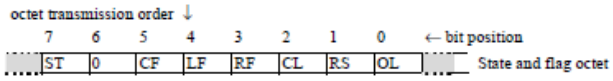
Description

Object group 2, variation 1 is used to report events related to a binary input point. See 11.1 for a description of a Binary Input Point type.

Variation 1 objects contain an octet for reporting the state of the input and status flags.

Coding

Pictorial



Formal structure

BSTR8: Flag Octet

Bit 0:	ONLINE
Bit 1:	RESTART
Bit 2:	COMM_LOST
Bit 3:	REMOTE_FORCED
Bit 4:	LOCAL_FORCED
Bit 5:	CHATTER_FILTER
Bit 6:	Reserved, always 0
Bit 7:	STATE—Has a value of 0 or 1, representing the state of the physical or logical input.

Figura A.8: Grupo 2, variación 1.

DNP3 Object Library

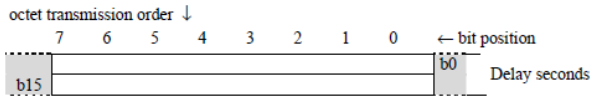
		Group:	52
		Variation:	1
Group Name:	Time Delay	Type:	Info
Variation Name:	Coarse		

Description

This object is used to convey the elapsed time between two absolute times with a resolution of 1 second.

Coding

Pictorial



Formal structure

UINT16: Delay seconds

This is the elapsed time with a 1-second resolution.

Range is 0 to +65 535 seconds.

NOTE—Use of the Fine Time Delay object, group 52, variation 2 is preferred for delay times of less than or equal to 65 seconds.

Figura A.9: Grupo 52, variación 1.

DNP3 Object Library

		Group:	80
		Variation:	1
Group Name:	Internal Indications	Type:	Static
Variation Name:	Packed format		

Description

Object group 80, variation 1 is used to report and in some cases set a device's DNP3 Internal Indication states. Internal indications are described in detail in Clause 4 through Clause 6 of this standard. internal indication states are reported in every response from an outstation. The internal indications reported or set by this object group and variation are the same.

Point indexes are used to specify the specific internal indication bit(s) according to the following table.

Index	Bit	Name
0	IIN1.0	ALL_STATIONS
1	IIN1.1	CLASS_1_EVENTS
2	IIN1.2	CLASS_2_EVENTS
3	IIN1.3	CLASS_3_EVENTS
4	IIN1.4	NEED_TIME
5	IIN1.5	LOCAL_CONTROL
6	IIN1.6	DEVICE_TROUBLE
7	IIN1.7	DEVICE_RESTART

8	IIN2.0	NO_FUNC_CODE_SUPPORT
9	IIN2.1	OBJECT_UNKNOWN
10	IIN2.2	PARAMETER_ERROR
11	IIN2.3	EVENT_BUFFER_OVERFLOW
12	IIN2.4	ALREADY_EXECUTING
13	IIN2.5	CONFIG_CORRUPT
14	IIN2.6	RESERVED_1
15	IIN2.7	RESERVED_2

A device may include a private set of internal indication states beyond index 15.

Coding

This object is coded as a bit string. When multiple indexes are contained in a single message, they are packed, as illustrated in the following pictorial.

Pictorial

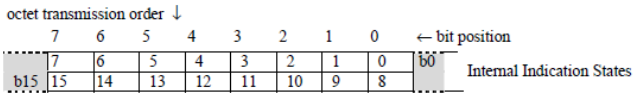


Figura A.10: Grupo 80, variación 1.

Apéndice B

Diagramas de los circuitos

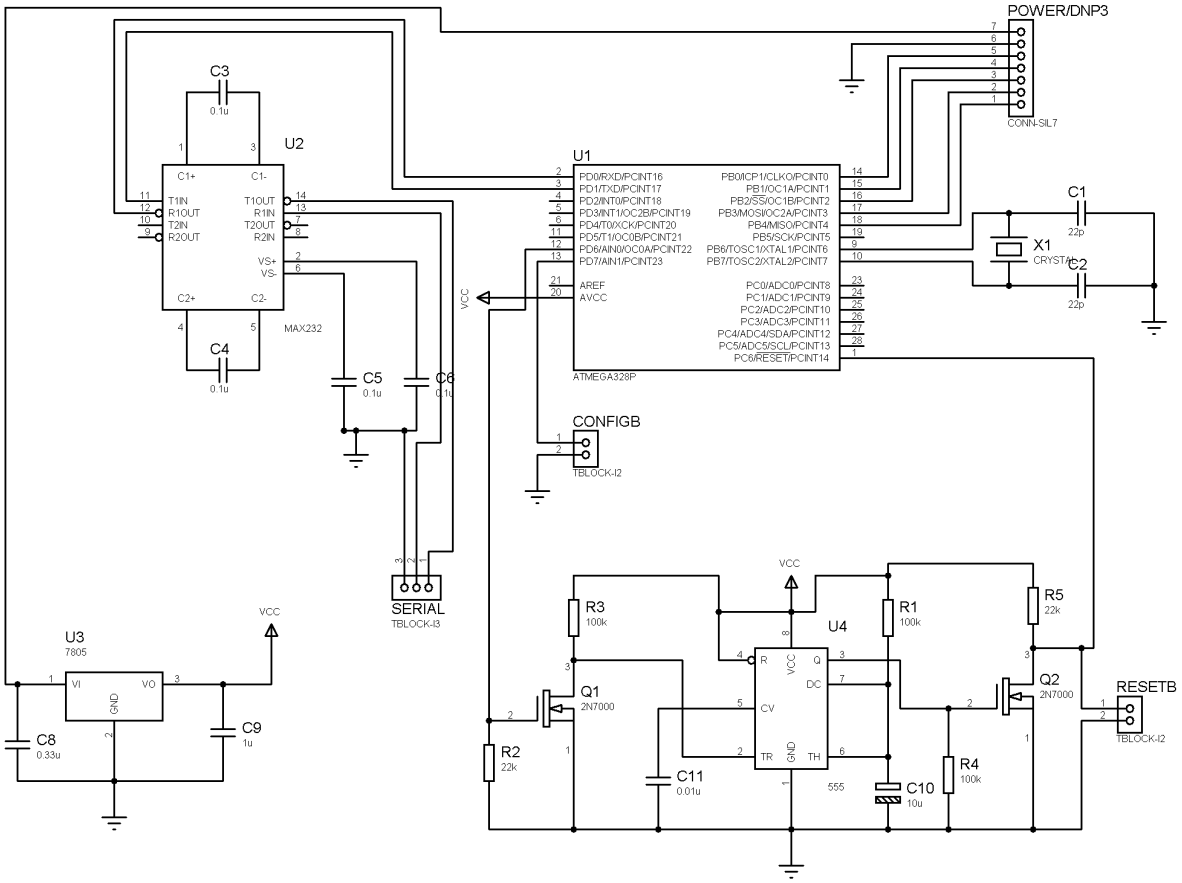


Figura B.1: Módulo DNP3.

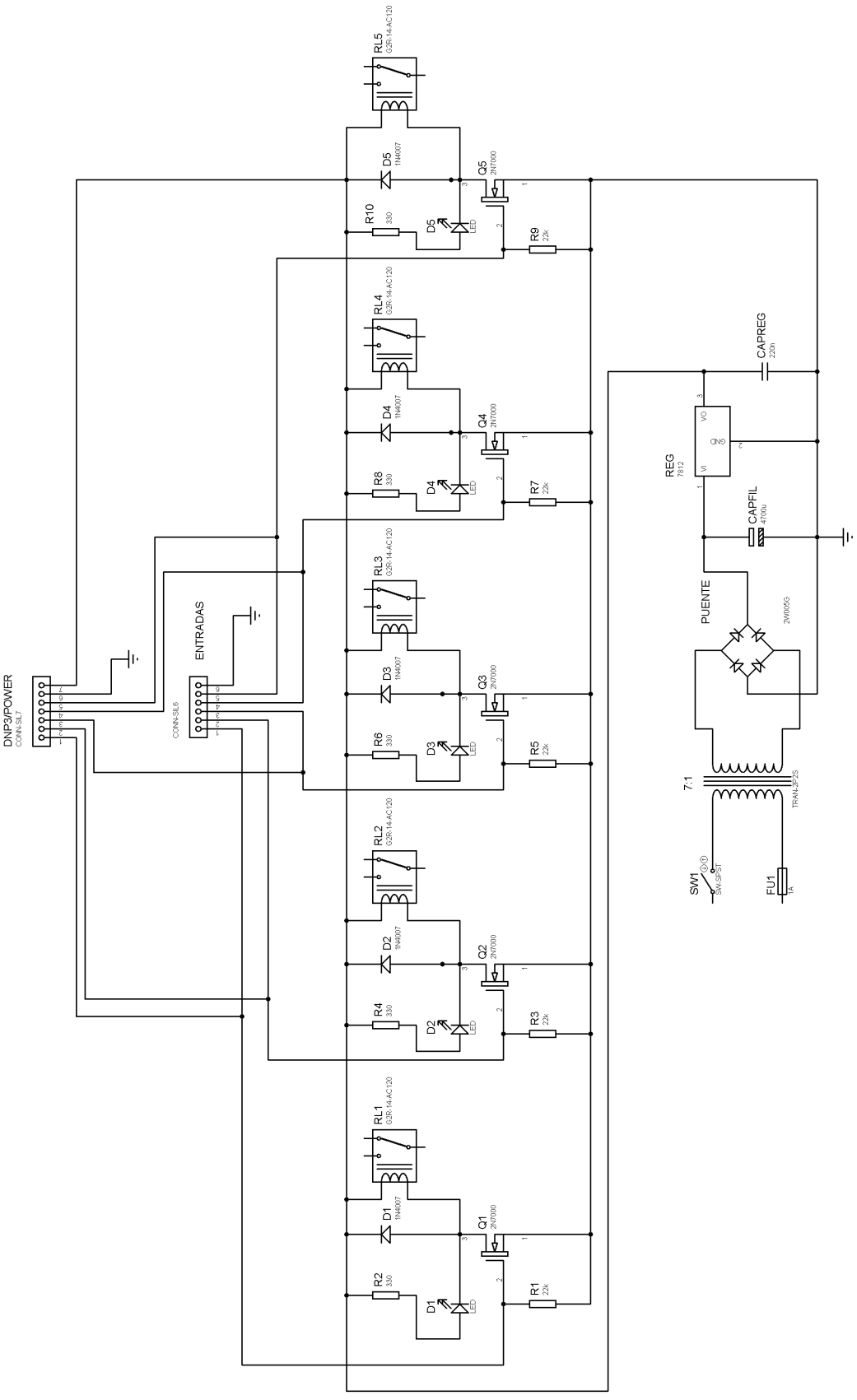


Figura B.2: Fuente y contactos secos.

Apéndice C

Dimensiones de las partes del chasís

Placa frontal

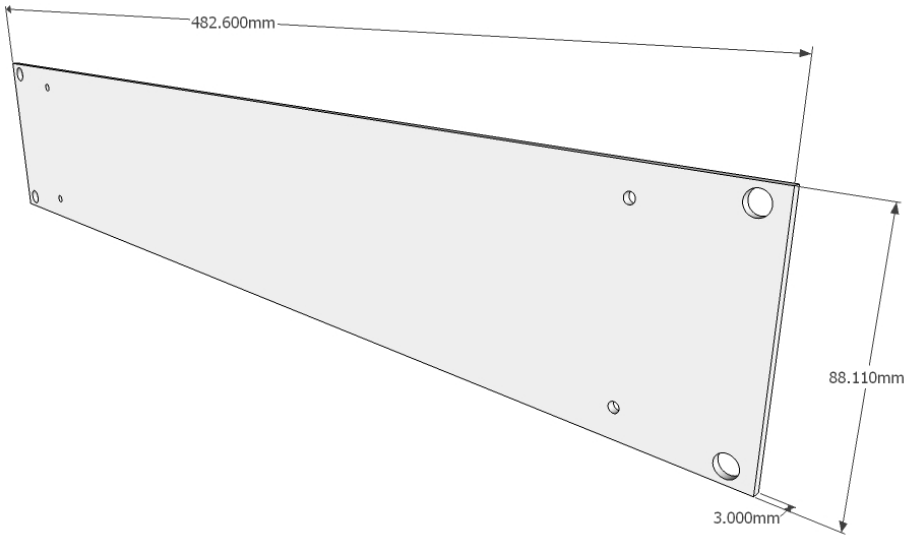


Figura C.1: Dimensiones de la placa frontal 1.

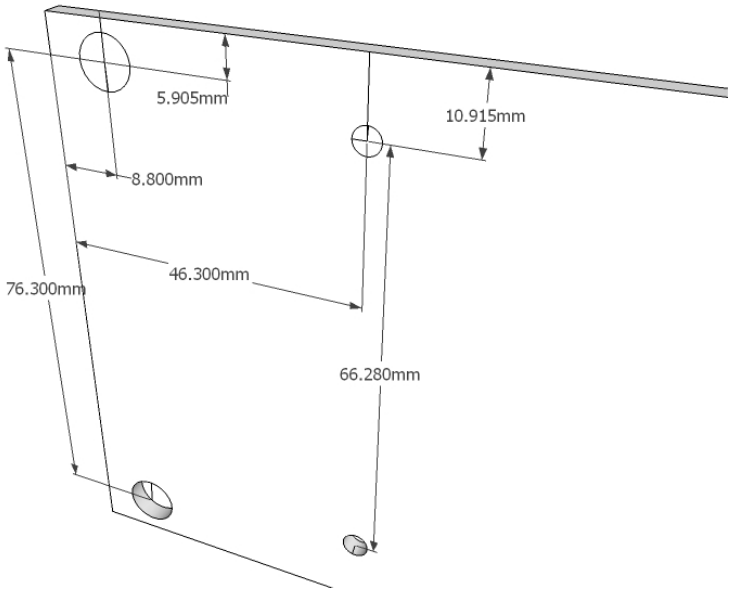


Figura C.2: Dimensiones de la placa frontal 2.

Placas laterales

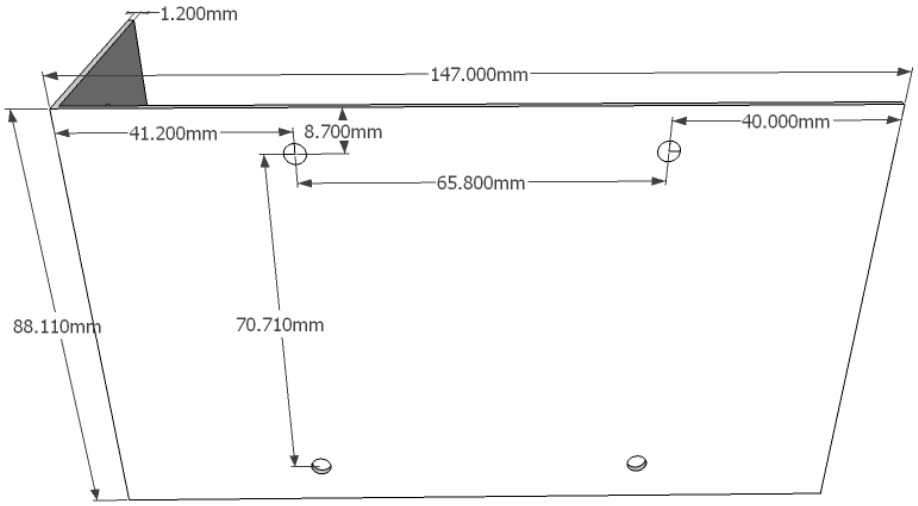


Figura C.3: Dimensiones de la placa lateral 1.

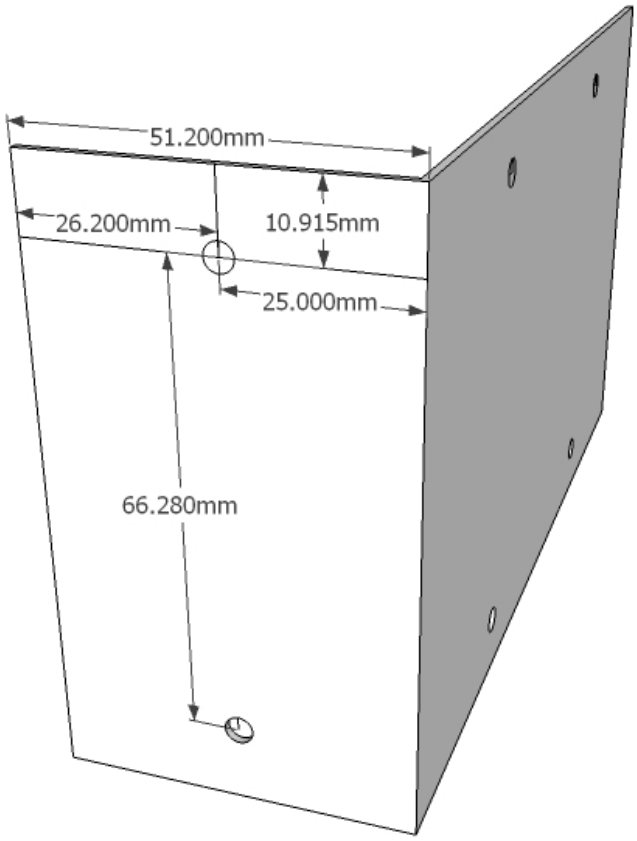


Figura C.4: Dimensiones de la placa lateral 2.

Superior e inferior

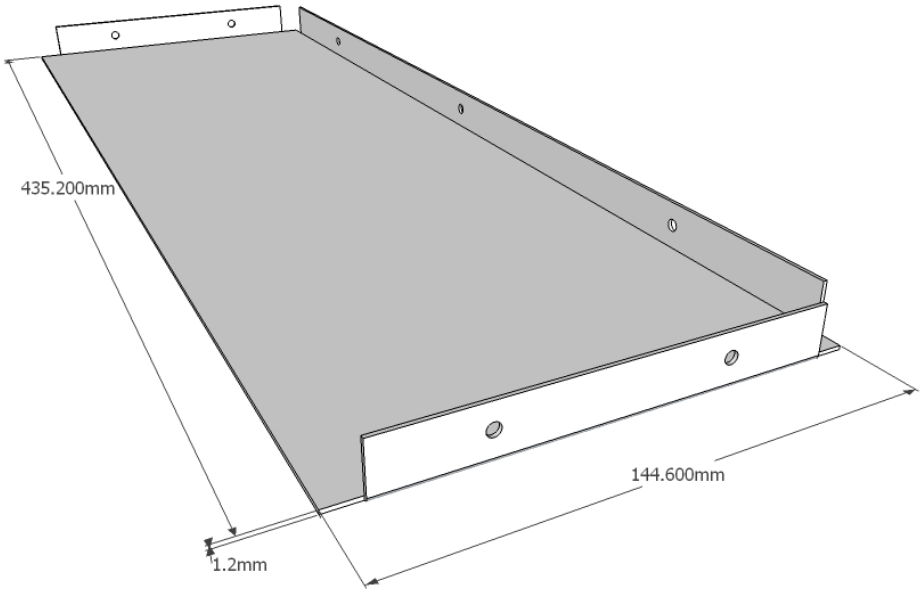


Figura C.5: Dimensiones de las partes superior e inferior 1.

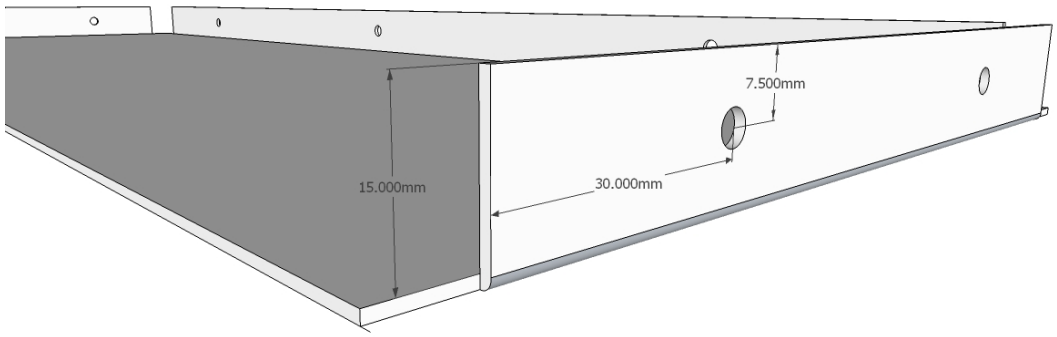


Figura C.6: Dimensiones de las partes superior e inferior 2.

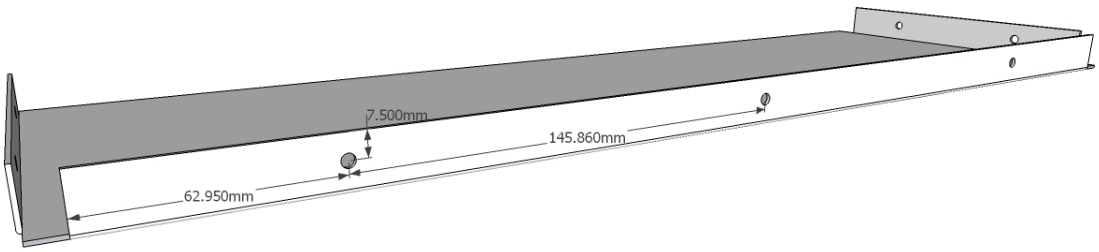


Figura C.7: Dimensiones de las partes superior e inferior 3.

Parte trasera

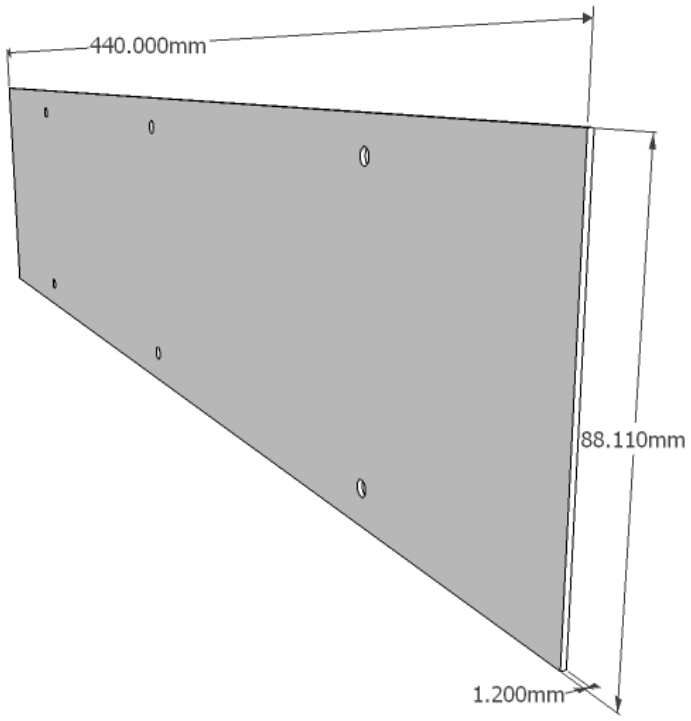


Figura C.8: Dimensiones de la parte trasera 1.

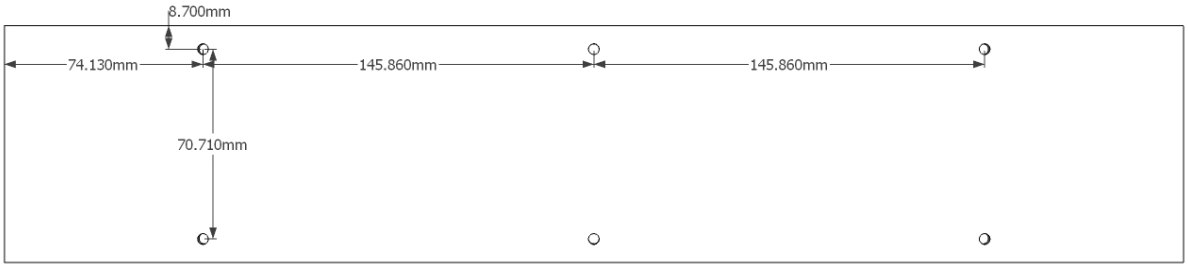


Figura C.9: Dimensiones de la parte trasera 2.

Apéndice D

Datos de la sección de pruebas

En este apéndice se muestran las transacciones de información entre los programas *Communication Protocol Test Harness* y *DNP3 Test Set* con el módulo DNP3 del sistema de alarmas. La configuración tanto del maestro como del esclavo se describe en el capítulo de *Pruebas y resultados*, sección *Pruebas del módulo DNP3*, subsección *Communication Protocol Test Harness*.

Communication Protocol Test Harness

Cada transacción comienza con una flecha que codifica el sentido de la información y la capa a la que va dirigida. Una flecha con sentido a la derecha indica la llegada de información a la estación maestra. Las flechas con sentido a la izquierda indican que el maestro envía una petición hacia el esclavo.

1. Transacciones a nivel de capa física.

   ```
   ...>     <...
   ```

2. Transacciones a nivel de la capa de enlace de datos.

   ```
   --->     <---
   ```

3. Transacciones a nivel de la pseudocapa de transporte.

   ```
   ~~~>     <~~~
   ```

4. Transacciones a nivel de la capa de aplicación.

   ```
   ===>     <===
   ```

Inicio de la comunicación.

```
...> mDNP       05

...> mDNP       64 05 40 01 00 02 00 00 82

---> mDNP       Primary Frame - Reset Link States
                LEN(5) DIR(0) PRM(1) FCV(0) FCB(0) DEST(1) SRC(2)
                05 64 05 40 01 00 02 00 00 82

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

...> mDNP       05

...> mDNP       64 0a 73 01 00 02 00 27 11

...> mDNP       c0 f0 82 80 00 6b 7d

---> mDNP       Primary Frame - Confirmed User Data
                LEN(10) DIR(0) PRM(1) FCV(1) FCB(1) DEST(1) SRC(2)
                05 64 0a 73 01 00 02 00 27 11
                c0 f0 82 80 00 6b 7d

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

^--> mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 0
                c0 f0 82 80 00

===> mDNP       Application Header, Unsolicited
                FIR(1) FIN(1) CON(1) UNS(1) SEQ# 0
                f0 82 80 00

<=== mDNP       Application Header, Application Confirmation
                FIR(1) FIN(1) CON(0) UNS(1) SEQ# 0
                d0 00

<^-- mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 8
                c8 d0 00

<--- mDNP       Primary Frame - Reset Link States
                LEN(5) DIR(1) PRM(1) FCV(0) FCB(0) DEST(2) SRC(1)
                05 64 05 c0 02 00 01 00 9e 59

<... mDNP       05 64 05 c0 02 00 01 00 9e 59

===> mDNP       IIN Bits:
                IIN1.7 Device Restart

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
                05 64 08 f3 02 00 01 00 0e ec
                c8 d0 00 db 84

<... mDNP       05 64 08 f3 02 00 01 00 0e ec c8 d0 00 db 84

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

Petición de lectura del grupo 0 variación 255 mediante el calificador 6.

```
Tx Object 0(Device Attribute), variation 255, qualifier 0x06(All Points)

<=== mDNP       Application Header, Read Request
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 3
                c3 01 00 ff 06

<^-- mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 9
                c9 c3 01 00 ff 06

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(11) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
                05 64 0b d3 02 00 01 00 03 67
                c9 c3 01 00 ff 06 22 58

<... mDNP       05 64 0b d3 02 00 01 00 03 67 c9 c3 01 00 ff 06
                22 58

...> mDNP       05
```

```
...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2

...> mDNP       05

...> mDNP       64 19 53 01 00 02 00 eb e2

...> mDNP       c0 e3 81 80 00 00 ff 17 01 00 fe 08 f2 00 f3 00
                05 7e

...> mDNP       fa 00 fc 00 80 a8

---> mDNP       Primary Frame - Confirmed User Data
                LEN(25) DIR(0) PRM(1) FCV(1) FCB(0) DEST(1) SRC(2)
                05 64 19 53 01 00 02 00 eb e2
                c0 e3 81 80 00 00 ff 17 01 00 fe 08 f2 00 f3 00 05 7e
                fa 00 fc 00 80 a8

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

~~~> mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 0
                c0 e3 81 80 00 00 ff 17 01 00 fe 08 f2 00 f3 00
                fa 00 fc 00

===> mDNP       Application Header, Response
                FIR(1) FIN(1) CON(1) UNS(0) SEQ# 3
                e3 81 80 00 00 ff 17 01 00 fe 08 f2 00 f3 00 fa
                00 fc 00

Rx Object 0(Device Attribute), variation 255, qualifier 0x17(8 Bit Index)
Device Attribute Property 0 Point 0 Variation 242=Device manufacturers software version string
Device Attribute Property 0 Point 0 Variation 243=Device manufacturers hardware version string
Device Attribute Property 0 Point 0 Variation 250=Device manufacturers product name and model
Device Attribute Property 0 Point 0 Variation 252=Device manufacturers name string

<=== mDNP       Application Header, Application Confirmation
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 3
                c3 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 10
                ca c3 00

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
                05 64 08 f3 02 00 01 00 0e ec
                ca c3 00 28 d8

<... mDNP       05 64 08 f3 02 00 01 00 0e ec ca c3 00 28 d8

===> mDNP       IIN Bits:
                IIN1.7 Device Restart

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

Petición de lectura del grupo 0, variación 254 mediante el calificador 6.

```
Tx Object 0(Device Attribute), variation 254, qualifier 0x06(All Points)

<=== mDNP       Application Header, Read Request
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 4
                c4 01 00 fe 06

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 11
                cb c4 01 00 fe 06

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(11) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
                05 64 0b d3 02 00 01 00 03 67
                cb c4 01 00 fe 06 c9 3c

<... mDNP       05 64 0b d3 02 00 01 00 03 67 cb c4 01 00 fe 06
                c9 3c

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2

...> mDNP       05

...> mDNP       64 52 73 01 00 02 00 47 5d
```

```
...> mDNP       c0 e4 81 80 00 00 f2 17 01 00 01 03 31 2e 30 00
                b1 e3

...> mDNP       f3 17 01 00 01 0b 41 72 64 75 69 6e 6f 20 55 4e
                97 c3

...> mDNP       4f 00 fa 17 01 00 01 17 53 69 73 74 65 6d 61 20
                2a e7

...> mDNP       64 65 20 61 6c 61 72 6d 61 73 20 44 4e 50 33 00
                47 06

...> mDNP       fc 17 01 00 01 07 55 4e 41 4d 20 46 49 0e f7

---> mDNP       Primary Frame - Confirmed User Data
                LEN(82) DIR(0) PRM(1) FCV(1) FCB(1) DEST(1) SRC(2)
                05 64 52 73 01 00 02 00 47 5d
                c0 e4 81 80 00 00 f2 17 01 00 01 03 31 2e 30 00 b1 e3
                f3 17 01 00 01 0b 41 72 64 75 69 6e 6f 20 55 4e 97 c3
                4f 00 fa 17 01 00 01 17 53 69 73 74 65 6d 61 20 2a e7
                64 65 20 61 6c 61 72 6d 61 73 20 44 4e 50 33 00 47 06
                fc 17 01 00 01 07 55 4e 41 4d 20 46 49 0e f7

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

~~~> mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 0
                c0 e4 81 80 00 00 f2 17 01 00 01 03 31 2e 30 00
                f3 17 01 00 01 0b 41 72 64 75 69 6e 6f 20 55 4e
                4f 00 fa 17 01 00 01 17 53 69 73 74 65 6d 61 20
                64 65 20 61 6c 61 72 6d 61 73 20 44 4e 50 33 00
                fc 17 01 00 01 07 55 4e 41 4d 20 46 49

===> mDNP       Application Header, Response
                FIR(1) FIN(1) CON(1) UNS(0) SEQ# 4
                e4 81 80 00 00 f2 17 01 00 01 03 31 2e 30 00 f3
                17 01 00 01 0b 41 72 64 75 69 6e 6f 20 55 4e 4f
                00 fa 17 01 00 01 17 53 69 73 74 65 6d 61 20 64
                65 20 61 6c 61 72 6d 61 73 20 44 4e 50 33 00 fc
                17 01 00 01 07 55 4e 41 4d 20 46 49

Rx Object 0(Device Attribute), variation 242, qualifier 0x17(8 Bit Index)
Device Attribute  Point 0, Variation 242=Device manufacturers software version string
code 1=Visible Characters  value 1.0

Rx Object 0(Device Attribute), variation 243, qualifier 0x17(8 Bit Index)
Device Attribute  Point 0, Variation 243=Device manufacturers hardware version string
code 1=Visible Characters  value Arduino UNO

Rx Object 0(Device Attribute), variation 250, qualifier 0x17(8 Bit Index)
Device Attribute  Point 0, Variation 250=Device manufacturers product name and model
code 1=Visible Characters  value Sistema de alarmas DNP3

Rx Object 0(Device Attribute), variation 252, qualifier 0x17(8 Bit Index)
Device Attribute  Point 0, Variation 252=Device manufacturers name string
code 1=Visible Characters  value UNAM FI

<=== mDNP       Application Header, Application Confirmation
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 4
                c4 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 12
                cc c4 00

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
                05 64 08 f3 02 00 01 00 0e ec
                cc c4 00 d9 0a

<... mDNP       05 64 08 f3 02 00 01 00 0e ec cc c4 00 d9 0a

===> mDNP       IIN Bits:
                IIN1.7 Device Restart

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

Petición de escritura del grupo 80, variación 1, índice 7 mediante el calificador 0 (Borrado del bit de ***RESTART***).

```
Tx Object 80(Internal Indications), variation 1, qualifier 0x00(8 Bit Start Stop)

<=== mDNP       Application Header, Write Request
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 5
                c5 02 50 01 00 07 07 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 13
                cd c5 02 50 01 00 07 07 00
```

```
<--- mDNP       Primary Frame - Confirmed User Data
                LEN(14) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
                05 64 0e d3 02 00 01 00 8a 9f
                cd c5 02 50 01 00 07 07 00 5f 61

<... mDNP       05 64 0e d3 02 00 01 00 8a 9f cd c5 02 50 01 00
                07 07 00 5f 61

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2

...> mDNP       05

...> mDNP       64 0a 53 01 00 02 00 7a 09

...> mDNP       c0 e5 81 00 00 68 f5

---> mDNP       Primary Frame - Confirmed User Data
                LEN(10) DIR(0) PRM(1) FCV(1) FCB(0) DEST(1) SRC(2)
                05 64 0a 53 01 00 02 00 7a 09
                c0 e5 81 00 00 68 f5

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

~~~> mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 0
                c0 e5 81 00 00

===> mDNP       Application Header, Response
                FIR(1) FIN(1) CON(1) UNS(0) SEQ# 5
                e5 81 00 00

<=== mDNP       Application Header, Application Confirmation
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 5
                c5 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 14
                ce c5 00

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
                05 64 08 f3 02 00 01 00 0e ec
                ce c5 00 e7 92

<... mDNP       05 64 08 f3 02 00 01 00 0e ec ce c5 00 e7 92

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

Solicitud de integridad (ningún evento ocurrido).

```
Tx Object 60(Class Data), variation 2, qualifier 0x06(All Points)

Tx Object 60(Class Data), variation 3, qualifier 0x06(All Points)

Tx Object 60(Class Data), variation 4, qualifier 0x06(All Points)

Tx Object 60(Class Data), variation 1, qualifier 0x06(All Points)

<=== mDNP       Application Header, Read Request
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 6
                c6 01 3c 02 06 3c 03 06 3c 04 06 3c 01 06

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 15
                cf c6 01 3c 02 06 3c 03 06 3c 04 06 3c 01 06

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(20) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
                05 64 14 d3 02 00 01 00 20 5b
                cf c6 01 3c 02 06 3c 03 06 3c 04 06 3c 01 06 d0 37

<... mDNP       05 64 14 d3 02 00 01 00 20 5b cf c6 01 3c 02 06
                3c 03 06 3c 04 06 3c 01 06 d0 37

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2

...> mDNP       05
```

```
...> mDNP       64 10 73 01 00 02 00 8d d5

...> mDNP       c0 e6 81 00 00 01 01 00 01 05 1f bc 8e

---> mDNP       Primary Frame - Confirmed User Data
                LEN(16) DIR(0) PRM(1) FCV(1) FCB(1) DEST(1) SRC(2)
                05 64 10 73 01 00 02 00 8d d5
                c0 e6 81 00 00 01 01 00 01 05 1f bc 8e

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

~~~> mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 0
                c0 e6 81 00 00 01 01 00 01 05 1f

===> mDNP       Application Header, Response
                FIR(1) FIN(1) CON(1) UNS(0) SEQ# 6
                e6 81 00 00 01 01 00 01 05 1f

                Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
                      Binary Input 000001 = 0x81
                      Binary Input 000002 = 0x81
                      Binary Input 000003 = 0x81
                      Binary Input 000004 = 0x81
                      Binary Input 000005 = 0x81

<=== mDNP       Application Header, Application Confirmation
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 6
                c6 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 16
                d0 c6 00

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
                05 64 08 f3 02 00 01 00 0e ec
                d0 c6 00 e5 6c

<... mDNP       05 64 08 f3 02 00 01 00 0e ec d0 c6 00 e5 6c

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

Petición de lectura del grupo 1, variación 1, índices del 1 al 2 usando el calificador 0.

```
Tx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)

<=== mDNP       Application Header, Read Request
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 7
                c7 01 01 01 00 01 02

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 17
                d1 c7 01 01 01 00 01 02

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(13) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
                05 64 0d d3 02 00 01 00 da 0c
                d1 c7 01 01 01 00 01 02 19 3d

<... mDNP       05 64 0d d3 02 00 01 00 da 0c d1 c7 01 01 01 00
                01 02 19 3d

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2

...> mDNP       05

...> mDNP       64 16 53 01 00 02 00 09 a6

...> mDNP       c0 e7 81 00 00 01 01 00 01 01 01 01 01 00 02 02
                43 8b

...> mDNP       01 a1 c9

---> mDNP       Primary Frame - Confirmed User Data
                LEN(22) DIR(0) PRM(1) FCV(1) FCB(0) DEST(1) SRC(2)
                05 64 16 53 01 00 02 00 09 a6
                c0 e7 81 00 00 01 01 00 01 01 01 01 01 00 02 02 43 8b
                01 a1 c9

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
```

```
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

~~~> mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 0
                c0 e7 81 00 00 01 01 00 01 01 01 01 01 00 02 02
                01

===> mDNP       Application Header, Response
                FIR(1) FIN(1) CON(1) UNS(0) SEQ# 7
                e7 81 00 00 01 01 00 01 01 01 01 01 00 02 02 01

                Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
                      Binary Input 000001 = 0x81

                Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
                      Binary Input 000002 = 0x81

<=== mDNP       Application Header, Application Confirmation
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 7
                c7 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 18
                d2 c7 00

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
                05 64 08 f3 02 00 01 00 0e ec
                d2 c7 00 db f4

<... mDNP       05 64 08 f3 02 00 01 00 0e ec d2 c7 00 db f4

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

Petición de lectura del grupo 1, variación 1, índices de 3 a 5 utilizando el calificador 1.

```
Tx Object 1(Binary Input), variation 1, qualifier 0x01(16 Bit Start Stop)

<=== mDNP       Application Header, Read Request
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 8
                c8 01 01 01 01 03 00 05 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 19
                d3 c8 01 01 01 01 03 00 05 00

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(15) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
                05 64 0f d3 02 00 01 00 6d 2a
                d3 c8 01 01 01 01 03 00 05 00 97 10

<... mDNP       05 64 0f d3 02 00 01 00 6d 2a d3 c8 01 01 01 01
                03 00 05 00 97 10

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2

...> mDNP       05

...> mDNP       64 1c 73 01 00 02 00 3f 02

...> mDNP       c0 e8 81 00 00 01 01 00 03 03 01 01 01 00 04 04
                32 60

...> mDNP       01 01 01 00 05 05 01 fe e3

---> mDNP       Primary Frame - Confirmed User Data
                LEN(28) DIR(0) PRM(1) FCV(1) FCB(1) DEST(1) SRC(2)
                05 64 1c 73 01 00 02 00 3f 02
                c0 e8 81 00 00 01 01 00 03 03 01 01 01 00 04 04 32 60
                01 01 01 00 05 05 01 fe e3

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

~~~> mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 0
                c0 e8 81 00 00 01 01 00 03 03 01 01 01 00 04 04
                01 01 01 00 05 05 01

===> mDNP       Application Header, Response
                FIR(1) FIN(1) CON(1) UNS(0) SEQ# 8
                e8 81 00 00 01 01 00 03 03 01 01 01 00 04 04 01
```

```
              01 01 00 05 05 01

              Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
                    Binary Input 000003 = 0x81

              Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
                    Binary Input 000004 = 0x81

              Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
                    Binary Input 000005 = 0x81

<=== mDNP     Application Header, Application Confirmation
              FIR(1) FIN(1) CON(0) UNS(0) SEQ# 8
              c8 00

<~~~ mDNP     Transport Header
              FIR(1) FIN(1) SEQ# 20
              d4 c8 00

<--- mDNP     Primary Frame - Confirmed User Data
              LEN(8) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
              05 64 08 f3 02 00 01 00 0e ec
              d4 c8 00 be 49

<... mDNP     05 64 08 f3 02 00 01 00 0e ec d4 c8 00 be 49

...> mDNP     05

...> mDNP     64 05 00 01 00 02 00 ba b2

---> mDNP     Secondary Frame - Acknowledge
              LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
              05 64 05 00 01 00 02 00 ba b2
```

Petición de lectura del grupo 1, variación 1 mediante el calificador 6.

```
Tx Object 1(Binary Input), variation 1, qualifier 0x06(All Points)

<=== mDNP     Application Header, Read Request
              FIR(1) FIN(1) CON(0) UNS(0) SEQ# 9
              c9 01 01 01 06

<~~~ mDNP     Transport Header
              FIR(1) FIN(1) SEQ# 21
              d5 c9 01 01 01 06

<--- mDNP     Primary Frame - Confirmed User Data
              LEN(11) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
              05 64 0b d3 02 00 01 00 03 67
              d5 c9 01 01 01 06 12 22

<... mDNP     05 64 0b d3 02 00 01 00 03 67 d5 c9 01 01 01 06
              12 22

...> mDNP     05

...> mDNP     64 05 00 01 00 02 00 ba b2

---> mDNP     Secondary Frame - Acknowledge
              LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
              05 64 05 00 01 00 02 00 ba b2

...> mDNP     05

...> mDNP     64 10 53 01 00 02 00 d0 cd

...> mDNP     c0 e9 81 00 00 01 01 00 01 05 1f 1f a5

---> mDNP     Primary Frame - Confirmed User Data
              LEN(16) DIR(0) PRM(1) FCV(1) FCB(0) DEST(1) SRC(2)
              05 64 10 53 01 00 02 00 d0 cd
              c0 e9 81 00 00 01 01 00 01 05 1f 1f a5

<--- mDNP     Secondary Frame - Acknowledge
              LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
              05 64 05 80 02 00 01 00 24 69

<... mDNP     05 64 05 80 02 00 01 00 24 69

~~~> mDNP     Transport Header
              FIR(1) FIN(1) SEQ# 0
              c0 e9 81 00 00 01 01 00 01 05 1f

===> mDNP     Application Header, Response
              FIR(1) FIN(1) CON(1) UNS(0) SEQ# 9
              e9 81 00 00 01 01 00 01 05 1f

  Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
                    Binary Input 000001 = 0x81
                    Binary Input 000002 = 0x81
                    Binary Input 000003 = 0x81
                    Binary Input 000004 = 0x81
                    Binary Input 000005 = 0x81

<=== mDNP     Application Header, Application Confirmation
              FIR(1) FIN(1) CON(0) UNS(0) SEQ# 9
              c9 00

<~~~ mDNP     Transport Header
              FIR(1) FIN(1) SEQ# 22
              d6 c9 00
```

```
<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
                05 64 08 f3 02 00 01 00 0e ec
                d6 c9 00 80 d1

<... mDNP       05 64 08 f3 02 00 01 00 0e ec d6 c9 00 80 d1

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

Solicitud de integridad. (Se cambió deliberadamente y múltiples veces el estado de la entrada número 4).

```
Tx Object 60(Class Data), variation 2, qualifier 0x06(All Points)

Tx Object 60(Class Data), variation 3, qualifier 0x06(All Points)

Tx Object 60(Class Data), variation 4, qualifier 0x06(All Points)

Tx Object 60(Class Data), variation 1, qualifier 0x06(All Points)

<=== mDNP       Application Header, Read Request
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 10
                ca 01 3c 02 06 3c 03 06 3c 04 06 3c 01 06

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 23
                d7 ca 01 3c 02 06 3c 03 06 3c 04 06 3c 01 06

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(20) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
                05 64 14 d3 02 00 01 00 20 5b
                d7 ca 01 3c 02 06 3c 03 06 3c 04 06 3c 01 06 4d d2

<... mDNP       05 64 14 d3 02 00 01 00 20 5b d7 ca 01 3c 02 06
                3c 03 06 3c 04 06 3c 01 06 4d d2

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2

...> mDNP       05

...> mDNP       64 28 73 01 00 02 00 12 c6

...> mDNP       c0 ea 81 00 00 02 01 17 01 04 01 02 01 17 01 04
                c7 b3

...> mDNP       81 02 01 17 01 04 01 02 01 17 01 04 81 01 01 00
                c9 b5

...> mDNP       01 05 1f bb b4

---> mDNP       Primary Frame - Confirmed User Data
                LEN(40) DIR(0) PRM(1) FCV(1) FCB(1) DEST(1) SRC(2)
                05 64 28 73 01 00 02 00 12 c6
                c0 ea 81 00 00 02 01 17 01 04 01 02 01 17 01 04 c7 b3
                81 02 01 17 01 04 01 02 01 17 01 04 81 01 01 00 c9 b5
                01 05 1f bb b4

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

~~~> mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 0
                c0 ea 81 00 00 02 01 17 01 04 01 02 01 17 01 04
                81 02 01 17 01 04 01 02 01 17 01 04 81 01 01 00
                01 05 1f

===> mDNP       Application Header, Response
                FIR(1) FIN(1) CON(1) UNS(0) SEQ# 10
                ea 81 00 00 02 01 17 01 04 01 02 01 17 01 04 81
                02 01 17 01 04 01 02 01 17 01 04 81 01 01 00 01
                05 1f

                Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
                      Binary Input 000004 = 0x01

                Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
                      Binary Input 000004 = 0x81

                Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
                      Binary Input 000004 = 0x01

                Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
                      Binary Input 000004 = 0x81
```

```
                Rx Object 1(Binary Input), variation 1, qualifier 0x00(8 Bit Start Stop)
                      Binary Input 000001 = 0x81
                      Binary Input 000002 = 0x81
                      Binary Input 000003 = 0x81
                      Binary Input 000004 = 0x81
                      Binary Input 000005 = 0x81

<=== mDNP       Application Header, Application Confirmation
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 10
                ca 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 24
                d8 ca 00

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
                05 64 08 f3 02 00 01 00 0e ec
                d8 ca 00 7b f9

<... mDNP       05 64 08 f3 02 00 01 00 0e ec d8 ca 00 7b f9

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

Petición de lectura del grupo 60, variaciones 2,3 y 4 mediante el calificador 6. (Se cambió deliberadamente un par de veces el estado de la entrada 3).

```
Tx Object 60(Class Data), variation 2, qualifier 0x06(All Points)

Tx Object 60(Class Data), variation 3, qualifier 0x06(All Points)

Tx Object 60(Class Data), variation 4, qualifier 0x06(All Points)

<=== mDNP       Application Header, Read Request
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 11
                cb 01 3c 02 06 3c 03 06 3c 04 06

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 25
                d9 cb 01 3c 02 06 3c 03 06 3c 04 06

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(17) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
                05 64 11 d3 02 00 01 00 a9 a3
                d9 cb 01 3c 02 06 3c 03 06 3c 04 06 90 46

<... mDNP       05 64 11 d3 02 00 01 00 a9 a3 d9 cb 01 3c 02 06
                3c 03 06 3c 04 06 90 46

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2

...> mDNP       05

...> mDNP       64 16 53 01 00 02 00 09 a6

...> mDNP       c0 eb 81 00 00 02 01 17 01 03 01 02 01 17 01 03
                12 8f

...> mDNP       81 1d 6f

---> mDNP       Primary Frame - Confirmed User Data
                LEN(22) DIR(0) PRM(1) FCV(1) FCB(0) DEST(1) SRC(2)
                05 64 16 53 01 00 02 00 09 a6
                c0 eb 81 00 00 02 01 17 01 03 01 02 01 17 01 03 12 8f
                81 1d 6f

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

~~~> mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 0
                c0 eb 81 00 00 02 01 17 01 03 01 02 01 17 01 03
                81

===> mDNP       Application Header, Response
                FIR(1) FIN(1) CON(1) UNS(0) SEQ# 11
                eb 81 00 00 02 01 17 01 03 01 02 01 17 01 03 81

     Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
                      Binary Input 000003 = 0x01

     Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
                      Binary Input 000003 = 0x81
```

```
<=== mDNP       Application Header, Application Confirmation
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 11
                cb 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 26
                da cb 00

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
                05 64 08 f3 02 00 01 00 0e ec
                da cb 00 45 61

<... mDNP       05 64 08 f3 02 00 01 00 0e ec da cb 00 45 61

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

Petición de habilitación del envío de eventos correspondientes a las clases 1,2 y 3 mediante respuestas no solicitadas.

```
Tx Object 60(Class Data), variation 2, qualifier 0x06(All Points)

Tx Object 60(Class Data), variation 3, qualifier 0x06(All Points)

Tx Object 60(Class Data), variation 4, qualifier 0x06(All Points)

<=== mDNP       Application Header, Enable Unsolicited Messages
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 12
                cc 14 3c 02 06 3c 03 06 3c 04 06

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 27
                db cc 14 3c 02 06 3c 03 06 3c 04 06

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(17) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
                05 64 11 d3 02 00 01 00 a9 a3
                db cc 14 3c 02 06 3c 03 06 3c 04 06 6b 15

<... mDNP       05 64 11 d3 02 00 01 00 a9 a3 db cc 14 3c 02 06
                3c 03 06 3c 04 06 6b 15

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2

...> mDNP       05

...> mDNP       64 0a 73 01 00 02 00 27 11

...> mDNP       c0 ec 81 00 00 af c2

---> mDNP       Primary Frame - Confirmed User Data
                LEN(10) DIR(0) PRM(1) FCV(1) FCB(1) DEST(1) SRC(2)
                05 64 0a 73 01 00 02 00 27 11
                c0 ec 81 00 00 af c2

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

~~~> mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 0
                c0 ec 81 00 00

===> mDNP       Application Header, Response
                FIR(1) FIN(1) CON(1) UNS(0) SEQ# 12
                ec 81 00 00

<=== mDNP       Application Header, Application Confirmation
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 12
                cc 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 28
                dc cc 00

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
                05 64 08 f3 02 00 01 00 0e ec
                dc cc 00 b4 b3

<... mDNP       05 64 08 f3 02 00 01 00 0e ec dc cc 00 b4 b3

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2
```

```
---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

Envío de respuesta no solicitada debido al cambio de estado de la entrada 4.

```
...> mDNP       05

...> mDNP       64 10 53 01 00 02 00 d0 cd

...> mDNP       c0 f0 82 00 00 02 01 17 01 04 01 ca bf

---> mDNP       Primary Frame - Confirmed User Data
                LEN(16) DIR(0) PRM(1) FCV(1) FCB(0) DEST(1) SRC(2)
                05 64 10 53 01 00 02 00 d0 cd
                c0 f0 82 00 00 02 01 17 01 04 01 ca bf

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

~~~> mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 0
                c0 f0 82 00 00 02 01 17 01 04 01

===> mDNP       Application Header, Unsolicited
                FIR(1) FIN(1) CON(1) UNS(1) SEQ# 0
                f0 82 00 00 02 01 17 01 04 01

                Rx Object 2(Binary Input Change), variation 1, qualifier 0x17(8 Bit Index)
                      Binary Input 000004 = 0x01

<=== mDNP       Application Header, Application Confirmation
                FIR(1) FIN(1) CON(0) UNS(1) SEQ# 0
                d0 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 29
                dd d0 00

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
                05 64 08 d3 02 00 01 00 53 f4
                dd d0 00 7a 2d

<... mDNP       05 64 08 d3 02 00 01 00 53 f4 dd d0 00 7a 2d

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

Petición de deshabilitar el envío de eventos correspondientes a las clases 1,2 y 3 en respuestas no solicitadas.

```
Tx Object 60(Class Data), variation 2, qualifier 0x06(All Points)

Tx Object 60(Class Data), variation 3, qualifier 0x06(All Points)

Tx Object 60(Class Data), variation 4, qualifier 0x06(All Points)

<=== mDNP       Application Header, Disable Unsolicited Messages
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 14
                ce 15 3c 02 06 3c 03 06 3c 04 06

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 33
                e1 ce 15 3c 02 06 3c 03 06 3c 04 06

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(17) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
                05 64 11 d3 02 00 01 00 a9 a3
                e1 ce 15 3c 02 06 3c 03 06 3c 04 06 9a 8d

<... mDNP       05 64 11 d3 02 00 01 00 a9 a3 e1 ce 15 3c 02 06
                3c 03 06 3c 04 06 9a 8d

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

```
...> mDNP       05

...> mDNP       64 0a 73 01 00 02 00 27 11

...> mDNP       c0 ee 81 00 00 06 0a

---> mDNP       Primary Frame - Confirmed User Data
                LEN(10) DIR(0) PRM(1) FCV(1) FCB(1) DEST(1) SRC(2)
                05 64 0a 73 01 00 02 00 27 11
                c0 ee 81 00 00 06 0a

<--- mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                05 64 05 80 02 00 01 00 24 69

<... mDNP       05 64 05 80 02 00 01 00 24 69

~~~> mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 0
                c0 ee 81 00 00

===> mDNP       Application Header, Response
                FIR(1) FIN(1) CON(1) UNS(0) SEQ# 14
                ee 81 00 00

<=== mDNP       Application Header, Application Confirmation
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 14
                ce 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 34
                e2 ce 00

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
                05 64 08 f3 02 00 01 00 0e ec
                e2 ce 00 73 06

<... mDNP       05 64 08 f3 02 00 01 00 0e ec e2 ce 00 73 06

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

Petición de lectura del grupo 1 variación 2 mediante el calificador 6 (el estado de varias entradas fue cambiado de tal forma que se generaran varios eventos).

```
Tx Object 1(Binary Input), variation 2, qualifier 0x06(All Points)

 <=== mDNP       Application Header, Read Request
                 FIR(1) FIN(1) CON(0) UNS(0) SEQ# 5
                 c5 01 01 02 06

 <~~~ mDNP       Transport Header
                 FIR(1) FIN(1) SEQ# 48
                 f0 c5 01 01 02 06

 <--- mDNP       Primary Frame - Confirmed User Data
                 LEN(11) DIR(1) PRM(1) FCV(1) FCB(1) DEST(2) SRC(1)
                 05 64 0b f3 02 00 01 00 5e 7f
                 f0 c5 01 01 02 06 e3 42

 <... mDNP       05 64 0b f3 02 00 01 00 5e 7f f0 c5 01 01 02 06
                 e3 42

 ...> mDNP       05

 ...> mDNP       64 05 00 01 00 02 00 ba b2

 ---> mDNP       Secondary Frame - Acknowledge
                 LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                 05 64 05 00 01 00 02 00 ba b2

 ...> mDNP       05

 ...> mDNP       64 0a 73 01 00 02 00 27 11

 ...> mDNP       c0 e5 81 02 02 31 82

 ---> mDNP       Primary Frame - Confirmed User Data
                 LEN(10) DIR(0) PRM(1) FCV(1) FCB(1) DEST(1) SRC(2)
                 05 64 0a 73 01 00 02 00 27 11
                 c0 e5 81 02 02 31 82

 <--- mDNP       Secondary Frame - Acknowledge
                 LEN(5) DIR(1) PRM(0) DFC(0) DEST(2) SRC(1)
                 05 64 05 80 02 00 01 00 24 69

 <... mDNP       05 64 05 80 02 00 01 00 24 69

 ~~~> mDNP       Transport Header
                 FIR(1) FIN(1) SEQ# 0
                 c0 e5 81 02 02
```

```
===> mDNP       Application Header, Response
                FIR(1) FIN(1) CON(1) UNS(0) SEQ# 5
                e5 81 02 02

<=== mDNP       Application Header, Application Confirmation
                FIR(1) FIN(1) CON(0) UNS(0) SEQ# 5
                c5 00

<~~~ mDNP       Transport Header
                FIR(1) FIN(1) SEQ# 49
                f1 c5 00

<--- mDNP       Primary Frame - Confirmed User Data
                LEN(8) DIR(1) PRM(1) FCV(1) FCB(0) DEST(2) SRC(1)
                05 64 08 d3 02 00 01 00 53 f4
                f1 c5 00 7d 25

<... mDNP       05 64 08 d3 02 00 01 00 53 f4 f1 c5 00 7d 25

===> mDNP       IIN Bits:
                IIN1.1 Class 1 Event Data Is Available
                IIN2.1 Object Unknown

...> mDNP       05

...> mDNP       64 05 00 01 00 02 00 ba b2

---> mDNP       Secondary Frame - Acknowledge
                LEN(5) DIR(0) PRM(0) DFC(0) DEST(1) SRC(2)
                05 64 05 00 01 00 02 00 ba b2
```

DNP3 Test Set

En este registro no se muestra la información a nivel de bytes pero si algunos aspectos de cada una de las capas. La simbología es similar a la de los resultados del *Communication Protocol Test Harness* con la diferencia del sentido de las flechas. Una dirección hacia la derecha indica información originada en la estación maestra cuyo destino es el esclavo y una flecha con dirección hacia la izquierda indica el arribo de información al maestro proveniente del dispositivo esclavo.

```
17:31:00.497 - EVENT - FileLogger - New Log Started
 17:31:00.511 - INFO - dnp.ports.serial - Port successfully opened

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------ (0)

 17:31:00.511 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 0, Func: Disable Unsol HdrCount: 3,
     Header: (Grp: 60, Var: 2, Qual: 6, Count: 0)
     Header: (Grp: 60, Var: 3, Qual: 6, Count: 0) Header: (Grp: 60, Var: 4, Qual: 6, Count: 0), Size: 11
 17:31:00.511 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #0
 17:31:00.511 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_RESET_LINK_STATES PayloadSize: 0 From Master Pri->Sec FCB=0 FCV=0
 17:31:01.511 - WARNING - dnp.mts.link - Confirmed data timeout, retrying, 2 remaining - 58
 17:31:01.511 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_RESET_LINK_STATES PayloadSize: 0 From Master Pri->Sec FCB=0 FCV=0
 17:31:01.596 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_RESET_LINK_STATES PayloadSize: 0 From Outstation Pri->Sec FCB=0 FCV=0
 17:31:01.596 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
 17:31:01.911 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 5 From Outstation Pri->Sec FCB=1 FCV=1
 17:31:01.919 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
 17:31:01.919 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
 17:31:01.919 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 1, SEQ: 0, Func: Unsol Rsp IIN: (LSB: 80 DeviceRestart)
    (MSB: 00) HdrCount: 0, Size: 4
 17:31:01.919 - WARNING - dnp.mts.master - Device restart detected
 17:31:02.511 - WARNING - dnp.mts.link - Confirmed data timeout, retrying, 1 remaining - 58
 17:31:02.511 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_RESET_LINK_STATES PayloadSize: 0 From Master Pri->Sec FCB=0 FCV=0
 17:31:02.829 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
 17:31:02.829 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 12 From Master Pri->Sec FCB=1 FCV=1
 17:31:03.144 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
 17:31:03.144 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 1, SEQ: 0, Func: Confirm HdrCount: 0, Size: 2
 17:31:03.144 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #1
 17:31:03.144 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
 17:31:03.157 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 5 From Outstation Pri->Sec FCB=0 FCV=1
 17:31:03.164 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
 17:31:03.164 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
 17:31:03.164 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 0, Func: Rsp IIN: (LSB: 80 DeviceRestart)
   (MSB: 00) HdrCount: 0, Size: 4
 17:31:03.164 - WARNING - dnp.mts.app - .\AsyncAppLayer.cpp(274): Unsol flood, 19 - 19
 17:31:04.145 - WARNING - dnp.mts.link - Retry confirmed data - 58
 17:31:04.145 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
 17:31:04.461 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
 17:31:13.145 - WARNING - dnp.mts.app.sol - Timeout while waiting for response
 17:31:13.145 - INFO - dnp.mts.app.sol - App layer retry, 2 remaining

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------ (1)

 17:31:13.145 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 0, Func: Disable Unsol HdrCount: 3,
     Header: (Grp: 60, Var: 2, Qual: 6, Count: 0)
     Header: (Grp: 60, Var: 3, Qual: 6, Count: 0) Header: (Grp: 60, Var: 4, Qual: 6, Count: 0), Size: 11
 17:31:13.145 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #2
 17:31:13.145 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 12 From Master Pri->Sec FCB=1 FCV=1
 17:31:13.461 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
 17:31:13.470 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_RESET_LINK_STATES PayloadSize: 0 From Outstation Pri->Sec FCB=0 FCV=0
```

```
17:31:13.470 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
17:31:13.785 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 5 From Outstation Pri->Sec FCB=1 FCV=1
17:31:13.793 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
17:31:13.793 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
17:31:13.793 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 0, Func: Rsp IIN: (LSB: 80 DeviceRestart)
 (MSB: 00) HdrCount: 0, Size: 4
17:31:13.793 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 0, Func: Confirm HdrCount: 0, Size: 2
17:31:13.793 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #3
17:31:13.793 - WARNING - dnp.mts.master - Device restart detected
17:31:13.793 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
17:31:14.408 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------- (3)

17:31:14.408 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 1, Func: Write HdrCount: 1,
    Header: (Grp: 80, Var: 1, Qual: 0, Count: 1), Size: 8
17:31:14.408 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #4
17:31:14.408 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 9 From Master Pri->Sec FCB=1 FCV=1
17:31:14.723 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
17:31:14.735 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 5 From Outstation Pri->Sec FCB=0 FCV=1
17:31:14.743 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
17:31:14.743 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
17:31:14.743 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 1, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 0, Size: 4
17:31:14.743 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 1, Func: Confirm HdrCount: 0, Size: 2
17:31:14.743 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #5
17:31:14.743 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
17:31:15.358 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------- (4)

17:31:15.358 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 2, Func: Read HdrCount: 1,
    Header: (Grp: 60, Var: 1, Qual: 6, Count: 0), Size: 5
17:31:15.358 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #6
17:31:15.358 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 6 From Master Pri->Sec FCB=1 FCV=1
17:31:15.673 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
17:31:15.685 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 11 From Outstation Pri->Sec FCB=1 FCV=1
17:31:15.698 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
17:31:15.698 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
17:31:15.698 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 2, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 1,
    Header: (Grp: 1, Var: 1, Qual: 0, Count: 5), Size: 10
17:31:15.698 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 2, Func: Confirm HdrCount: 0, Size: 2
17:31:15.698 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #7
17:31:15.698 - INTERPRET - dnp.mts.master - Converting 5 Group1Var1 To class apl::Binary
17:31:15.698 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
17:31:16.312 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------- (5)

17:31:16.312 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 3, Func: Enable Unsol HdrCount: 3,
    Header: (Grp: 60, Var: 2, Qual: 6, Count: 0)
    Header: (Grp: 60, Var: 3, Qual: 6, Count: 0) Header: (Grp: 60, Var: 4, Qual: 6, Count: 0), Size: 11
17:31:16.312 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #8
17:31:16.312 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 12 From Master Pri->Sec FCB=1 FCV=1
17:31:16.627 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
17:31:16.640 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 5 From Outstation Pri->Sec FCB=0 FCV=1
17:31:16.644 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
17:31:16.644 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
17:31:16.644 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 3, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 0, Size: 4
17:31:16.644 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 3, Func: Confirm HdrCount: 0, Size: 2
17:31:16.644 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #9
17:31:16.644 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
17:31:17.258 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------- (6)

18:01:17.918 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 11, Func: Read HdrCount: 1,
    Header: (Grp: 60, Var: 1, Qual: 6, Count: 0), Size: 5
18:01:17.918 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #38
18:01:17.918 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 6 From Master Pri->Sec FCB=1 FCV=1
18:01:18.234 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
18:01:18.243 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 11 From Outstation Pri->Sec FCB=1 FCV=1
18:01:18.259 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
18:01:18.259 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
18:01:18.259 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 11, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 1,
 Header: (Grp: 1, Var: 1, Qual: 0, Count: 5), Size: 10
18:01:18.259 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 11, Func: Confirm HdrCount: 0, Size: 2
18:01:18.259 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #39
18:01:18.259 - INTERPRET - dnp.mts.master - Converting 5 Group1Var1 To class apl::Binary
18:01:18.259 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
18:01:18.873 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0


18:31:22.452 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 3, Func: Read HdrCount: 1,
    Header: (Grp: 60, Var: 1, Qual: 6, Count: 0), Size: 5
18:31:22.452 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #4
18:31:22.452 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 6 From Master Pri->Sec FCB=1 FCV=1
18:31:22.769 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
18:31:22.777 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 11 From Outstation Pri->Sec FCB=0 FCV=1
18:31:22.793 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
18:31:22.793 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
18:31:22.794 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 3, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 1,
    Header: (Grp: 1, Var: 1, Qual: 0, Count: 5), Size: 10
18:31:22.794 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 3, Func: Confirm HdrCount: 0, Size: 2
18:31:22.794 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #5
18:31:22.794 - INTERPRET - dnp.mts.master - Converting 5 Group1Var1 To class apl::Binary
18:31:22.794 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
18:31:23.408 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0


19:01:27.003 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 11, Func: Read HdrCount: 1,
    Header: (Grp: 60, Var: 1, Qual: 6, Count: 0), Size: 5
19:01:27.003 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #34
19:01:27.003 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 6 From Master Pri->Sec FCB=1 FCV=1
19:01:27.320 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
19:01:27.328 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 11 From Outstation Pri->Sec FCB=1 FCV=1
19:01:27.344 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
19:01:27.344 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
```

```
19:01:27.344 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 11, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 1,
 Header: (Grp: 1, Var: 1, Qual: 0, Count: 5), Size: 10
19:01:27.344 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 11, Func: Confirm HdrCount: 0, Size: 2
19:01:27.344 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #35
19:01:27.344 - INTERPRET - dnp.mts.master - Converting 5 Group1Var1 To class apl::Binary
19:01:27.344 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
19:01:27.958 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0


19:31:32.055 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 3, Func: Read HdrCount: 1,
    Header: (Grp: 60, Var: 1, Qual: 6, Count: 0), Size: 5
19:31:32.055 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #0
19:31:32.055 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 6 From Master Pri->Sec FCB=1 FCV=1
19:31:32.405 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
19:31:32.405 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 11 From Outstation Pri->Sec FCB=0 FCV=1
19:31:32.405 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
19:31:32.405 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
19:31:32.405 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 3, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 1,
    Header: (Grp: 1, Var: 1, Qual: 0, Count: 5), Size: 10
19:31:32.405 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 3, Func: Confirm HdrCount: 0, Size: 2
19:31:32.405 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #1
19:31:32.405 - INTERPRET - dnp.mts.master - Converting 5 Group1Var1 To class apl::Binary
19:31:32.405 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
19:31:33.035 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0


20:01:37.617 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 11, Func: Read HdrCount: 1,
    Header: (Grp: 60, Var: 1, Qual: 6, Count: 0), Size: 5
20:01:37.617 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #30
20:01:37.617 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 6 From Master Pri->Sec FCB=1 FCV=1
20:01:37.933 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
20:01:37.942 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 11 From Outstation Pri->Sec FCB=1 FCV=1
20:01:37.958 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
20:01:37.958 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
20:01:37.958 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 11, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 1,
    Header: (Grp: 1, Var: 1, Qual: 0, Count: 5), Size: 10
20:01:37.958 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 11, Func: Confirm HdrCount: 0, Size: 2
20:01:37.958 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #31
20:01:37.958 - INTERPRET - dnp.mts.master - Converting 5 Group1Var1 To class apl::Binary
20:01:37.958 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
20:01:38.572 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0


20:31:43.445 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 3, Func: Read HdrCount: 1,
    Header: (Grp: 60, Var: 1, Qual: 6, Count: 0), Size: 5
20:31:43.445 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #60
20:31:43.445 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 6 From Master Pri->Sec FCB=1 FCV=1
20:31:43.795 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
20:31:43.795 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 11 From Outstation Pri->Sec FCB=0 FCV=1
20:31:43.795 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
20:31:43.795 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
20:31:43.795 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 3, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 1,
    Header: (Grp: 1, Var: 1, Qual: 0, Count: 5), Size: 10
20:31:43.795 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 3, Func: Confirm HdrCount: 0, Size: 2
20:31:43.795 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #61
20:31:43.795 - INTERPRET - dnp.mts.master - Converting 5 Group1Var1 To class apl::Binary
20:31:43.795 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
20:31:44.425 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0


21:01:50.192 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 11, Func: Read
    HdrCount: 1, Header: (Grp: 60, Var: 1, Qual: 6, Count: 0), Size: 5
21:01:50.192 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #26
21:01:50.192 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 6 From Master Pri->Sec FCB=1 FCV=1
21:01:50.509 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
21:01:50.517 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 11 From Outstation Pri->Sec FCB=1 FCV=1
21:01:50.533 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
21:01:50.533 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
21:01:50.533 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 0, SEQ: 11, Func: Rsp IIN: (LSB: 00) (MSB: 00) HdrCount: 1,
    Header: (Grp: 1, Var: 1, Qual: 0, Count: 5), Size: 10
21:01:50.533 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 0, SEQ: 11, Func: Confirm HdrCount: 0, Size: 2
21:01:50.533 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #27
21:01:50.533 - INTERPRET - dnp.mts.master - Converting 5 Group1Var1 To class apl::Binary
21:01:50.533 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
21:01:51.148 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------ (7)

21:22:42.721 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 11 From Outstation Pri->Sec FCB=0 FCV=1
21:22:42.721 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
21:22:42.721 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
21:22:42.721 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 1, SEQ: 0, Func: Unsol Rsp IIN: (LSB: 00) (MSB: 00)
    HdrCount: 1, Header: (Grp: 2, Var: 1, Qual: 23, Count: 1), Size: 10
21:22:42.721 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 1, SEQ: 0, Func: Confirm HdrCount: 0, Size: 2
21:22:42.721 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #8
21:22:42.721 - INFO - dnp.mts.app.unsol - Ignoring repeat unsol seq: 0
21:22:42.721 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=1 FCV=1
21:22:43.351 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0

21:22:47.021 - INFO - terminal - Executing: show - 8388608


21:24:11.712 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 11 From Outstation Pri->Sec FCB=1 FCV=1
21:24:11.712 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_SEC_ACK PayloadSize: 0 From Master Sec->Pri DFC=0
21:24:11.712 - INTERPRET - dnp.mts.transport - <- TL: FIR FIN #0
21:24:11.712 - INTERPRET - dnp.mts.app - <= AL FIR: 1, FIN: 1, CON: 1, UNS: 1, SEQ: 1, Func: Unsol Rsp IIN: (LSB: 00) (MSB: 00)
    HdrCount: 1, Header: (Grp: 2, Var: 1, Qual: 23, Count: 1), Size: 10
21:24:11.712 - INTERPRET - dnp.mts.app - => AL FIR: 1, FIN: 1, CON: 0, UNS: 1, SEQ: 1, Func: Confirm HdrCount: 0, Size: 2
21:24:11.712 - INTERPRET - dnp.mts.transport - -> TL: FIR FIN #17
21:24:11.712 - INTERPRET - dnp.mts.master - Converting 1 Group2Var1 To class apl::Binary
21:24:11.712 - INTERPRET - dnp.serial - ~> DL 1 to 2 : FC_PRI_CONFIRMED_USER_DATA PayloadSize: 3 From Master Pri->Sec FCB=0 FCV=1
21:24:12.342 - INTERPRET - dnp.serial - <~ DL 2 to 1 : FC_SEC_ACK PayloadSize: 0 From Outstation Sec->Pri DFC=0

21:24:14.842 - INFO - terminal - Executing: show - 8388608
```

```
21:27:42.070 - INFO - terminal - Executing: quit - 8388608
21:27:42.070 - WARNING - dnp.ports.serial - La operación de E/S se anuló por una salida de subproceso o por una solicitud de aplicación
```

Glosario, siglas y acrónimos

CON	Bit que indica si un fragmento requiere de confirmación.
CRC	Cyclic Redundancy Check (Código de redundancia cíclica).
DCE	Data Communication Equipment (Equipo de comunicación de datos).
DEI	Dispositivo electrónico inteligente.
DIR	Bit que indica si la trama proviene de una estación maestra o de un esclavo.
DNP3	Distributed Network Protocol version 3 (Protocolo de red distribuida versión 3).
DTE	Data Terminal Equipment (Equipo de terminal de datos).
EEPROM	Electrically Erasable Programmable Read-Only Memory (Memoria de solo lectura eléctricamente programable - borrable).
EFCB	Expected FCB (FCB esperado).
EPA	Enhanced Performance Architecture (Arquitectura de rendimiento mejorado).
FC	Function Code (Código de función).
FCB	Bit que ayuda en la detección de tramas repetidas.
FCV	Bit que ayuda en la detección de tramas repetidas.
FIN	Ayuda a procesar mensajes multisegmento y multifragmento.
FIR	Ayuda a procesar mensajes multisegmento y multifragmento.

Flash Un tipo de memoria del microprocesador.

Handshake Método para realizar el control de flujo de datos.

IIN Internal Indications (Indicaciones internas).

Lenght Byte que contiene el largo de una trama.

LSB Least Significant Bit (El bit menos significativo).

MSB Most Significant Bit (El bit más significativo).

OSI Open System Interconnection (Interconexión de sistemas abiertos).

PLC Programmable Logic Controller (Controlador lógico programable).

PRM Bit que indica si la trama proviene de una estación primaria o secundaria.

RAM Random Access Memory (Memoria de acceso aleatorio).

RTU Remote Terminal Unit (Unidad terminal remota).

SCADA Supervisory Control And Data Acquisition (control supervisorio y adquisición de datos).

TTL Transistor - Transistor Logic (Lógica de transistor - transistor).

UNS Bit que indica si un fragmento es una respuesta no solicitada.

USART Universal Serial Asynchronous Reciever Transmitter (Transmisor - Receptor serial asíncrono universal).