xx(11 6559 1)

CINVESTAV

Centro de Investigación y de Estudios Avanzados del I.P.N.
Unidad Guadalajara

Transmisión Inalámbrica de Datos

TESIS QUE PRESENTA:
Miguel Angel Amador Alvarez

PARA OBTENER EL GRADO DE:
Maestro en Ciencias

EN LA ESPECIALIDAD DE:
Ingeniería Eléctrica

DIRECTOR DE TESIS
Dr. José Luis Leyva Montiel

Guadalajara, Jalisco. Enero de 2004

Transmisión Inalámbrica de Datos

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:
Miguel Angel Amador Alvarez
Ingeniero en Comunicaciones y Electrónica
Universidad de Guadalajara. 1993-1997

Becario del CONACYT, expediente No. **143828**

Director de Tesis:

Dr. José Luis Leyva Montiel

CINVESTAV del IPN Unidad Guadalajara. Enero de 2004

AGRADECIMIENTOS

Agradezco a Dios por haberme permitido terminar una etapa de mi vida.

Agradezco a mi familia por todo su apoyo y comprensión durante este tiempo de esfuerzo y trabajo.

Agradezco a mis amigos, compañeros y conocidos por su ayuda durante el trayecto de esta etapa.

Agradezco al **CONACYT** por el apoyo para la elaboración de esta tesis.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La comunicación inalámbrica permite intercambiar información entre personas o sistemas sin importar el lugar donde se encuentren.

En los orígenes, la transmisión de la información en forma inalámbrica se inicio con la radio y la televisión. Las características de estos son alta potencia y bajas frecuencias, equipos de transmisión de un volumen considerable, antenas fijas de un gran tamaño pero con un rango de alcance del orden de km.

Debido a la necesidad de la comunicación entre sistemas portátiles o transportables se redujeron las antenas y la potencia, provocando que la frecuencia de transmisión se incremente y el área de enlace disminuya. Esto dio origen a los sistemas de enlace de corto alcance para la transferencia de voz y datos. Dichos sistemas están basados en un protocolo de comunicación, que es un procedimiento que define una sintaxis y una secuencia permitiendo a los sistemas comunicarse entre ellos.

El tema de este trabajo se orienta a la trasferencia de datos en forma inalámbrica aplicada al servicio del transporte urbano de la ciudad de Guadalajara. Donde se plantea un sistema de pre-pago por el servicio y se enfoca en resolver la problemática de la comunicación inalámbrica que se presenta entre los autobuses y una estación base. Para determinar la eficiencia de la comunicación se agrega una etapa de pruebas y resultados.

OBJETIVOS

La presente tesis, llamada *"Transmisión inalámbrica de datos"* ha sido desarrollada en CINVESTAV Unidad GDL.

Los objetivos del trabajo son:

- Implementar una red inalámbrica basada en los sistemas MEDV-900-HP.
- Diseñar e implementar un protocolo que permita establecer una comunicación inalámbrica segura y libre de errores, usando los dispositivos.

En el presente trabajo se considera que las estaciones secundarias están instaladas en los autobuses urbanos con ruta determinada y que las estaciones primarias están colocadas en los puntos terminales de cada ruta. Estas últimas están conectadas en red con los bancos y puntos de venta. La comunicación entre autobuses y los puntos terminales de la ruta se establece a través del protocolo PCOMTU-GDL.

El documento proporciona información técnica general (definiciones, nomenclaturas, estándares, arquitecturas) relacionada con las redes inalámbricas de corto alcance.

ESTRUCTURA DE LA TESIS

El presente trabajo de tesis se encuentra organizado en 6 capítulos y 7 apéndices. El capítulo 1 expone la justificación de la presente tesis y se hace el dimensionamiento del sistema desde el punto de vista: velocidad de transmisión y capacidad de almacenamiento.

El capítulo 2 contiene los requerimientos de hardware y de software que debe cumplir el presente trabajo para garantizar una buena funcionalidad.

El capítulo 3 describe el estándar del protocolo PCOMTU-GDL implementado a nivel capa de enlace.

El capítulo 4 explica los algoritmos usados para establecer una comunicación inalámbrica entre una estación primaria y varias estaciones secundarias, utilizando el protocolo PCOMTU-GDL expuesto en el capítulo 3. También, introduce al usuario en el conocimiento y uso de los programas que permiten dicha comunicación.

El capítulo 5 presenta los resultados sobre el desempeño obtenido de las pruebas de transmisión realizadas en campo bajo diferentes condiciones: a diferentes distancias y en diferentes lugares. Además, se muestran las mediciones de potencia obtenidas de los *transceiver* MEDV-900-HP con la finalidad de poder establece una distancia máxima de transmisión.

El capítulo 6 presenta las conclusiones y observaciones sobre el desarrollo del presente trabajo. También, se describe el posible trabajo futuro.

El apéndice A incluye los acrónimos. El apéndice B contiene una introducción a los protocolos de comunicación inalámbrica Bluetooth y IEEE 802.11.

El apéndice C describe, en forma breve, los sistemas disponibles para establecer una red inalámbrica bajo los protocolos Bluetooth e IEEE 802.11. Además, se presenta el sistema MEDV-900-HP que permite el enlace físico de una comunicación inalámbrica.

El apéndice D contiene el código del programa usado por la estación primaria. Y el apéndice E el código del programa utilizado por las estaciones secundarias.

El apéndice F describe la configuración utilizada en la unidad de cobro para evitar que el transmisor cause interferencia con las demás unidades.

El apéndice G muestra el esquemático del receptor RXM-900-HP-II y del transmisor TXM-900-HP-II utilizados en el presente trabajo de tesis.

CAPITULO I

1. DESCRIPCIÓN DEL PROBLEMA

1.1 PLANTEAMIENTO

En el presente trabajo de tesis se plantea una arquitectura que a través de la introducción de un sistema de comunicación inalámbrico en el transporte urbano, permite las siguientes mejoras en el servicio:

- Hacer más práctico el acceso de los usuarios al servicio público a través de una tarjeta de pre-pago.
- Realizar los pagos de forma rápida y eficiente a las cuentas bancarias del conductor, concesionario (transportista) y empresa que proporciona los puntos de venta.
- Reenfocar la atención del conductor en la seguridad de los pasajeros y de la unidad.

El diagrama a bloques del sistema de pago electrónico se muestra en la Figura 1.1.

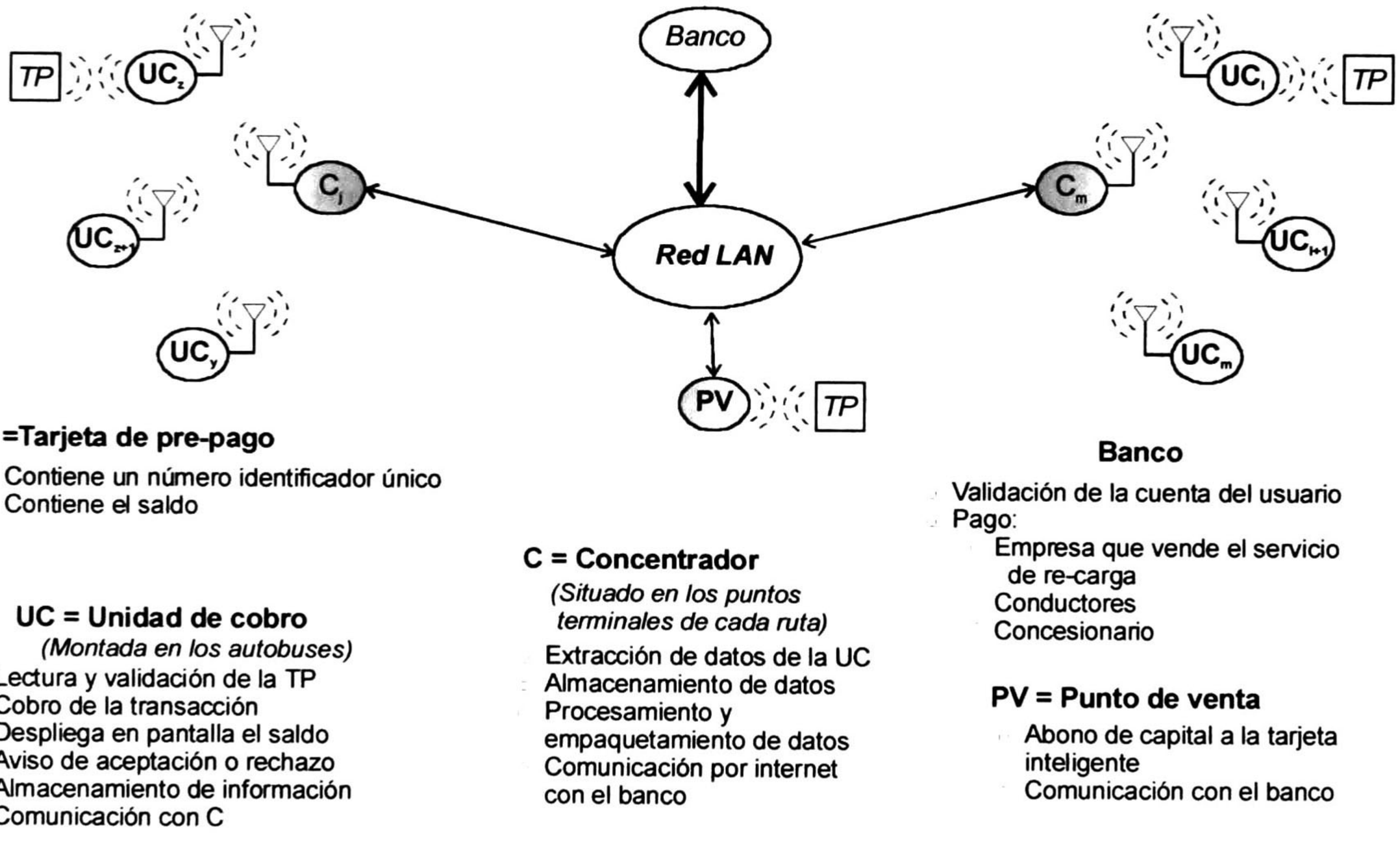


Figura 1.1 Diagrama a bloques del sistema de pago por el servicio de transporte urbano.

El sistema consta de 5 elementos, descritos a continuación:

Tarjeta de pre-pago (TP)

Es una tarjeta de proximidad que contiene un número de identificación, una memoria flash (384 bits) para almacenar el crédito adquirido y un protocolo para establecer una comunicación inalámbrica con la unidad de cobro. El rango de alcance de la tarjeta de pre-pago es de 10 cm. En la Figura 1.2 se observa un diagrama a bloques de la tarjeta de pre-pago.

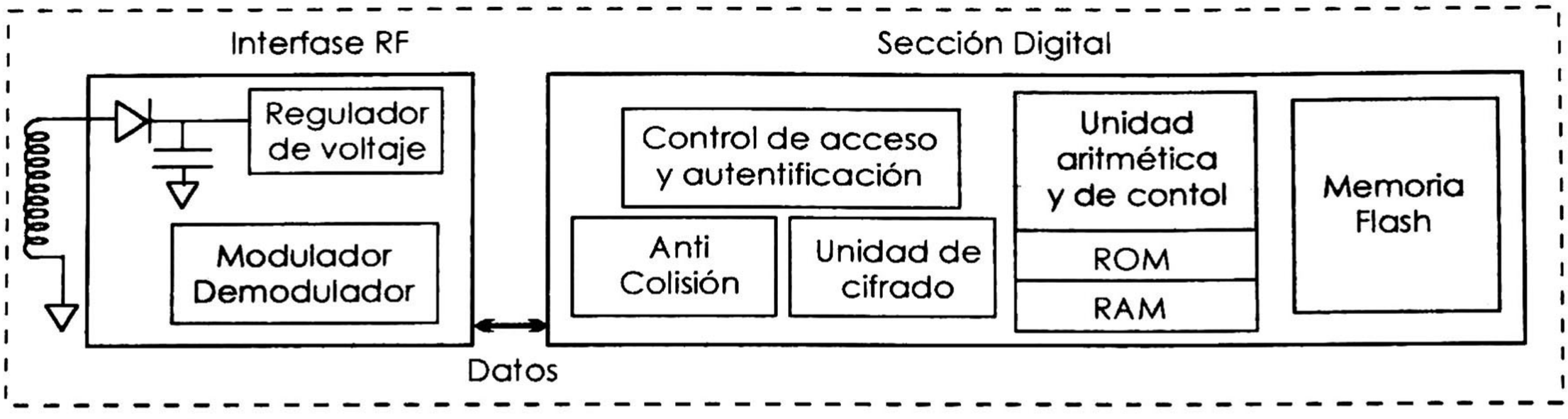


Figura 1.2 Diagrama a bloques de la tarjeta inteligente de pre-pago.

Punto de venta (PV)

Es el sistema propuesto para que el cliente pueda adquirir y abonar a su tarjeta de pre-pago el crédito que requiera. El punto de venta puede estar situado en las terminales de los autobuses, en centros comerciales, en cajeros automáticos, etc., y estará conectado con el banco a través de una red de datos.

Unidad de cobro (UC)

La unidad de cobro está instalada en todos los autobuses. En la Figura 1.3 se muestra el diagrama a bloques de la unidad de cobro. Esta tiene las siguientes características:

- Valida la tarjeta para brindar el servicio.
- Realiza el descuento del servicio y actualiza el saldo en la tarjeta de pre-pago.
- Despliega el costo de la transacción realizada y saldo en la tarjeta del usuario.
- Comunica por RF con el concentrador.
- Almacena la información correspondiente a treinta días para presentar reportes al elemento concentrador de recorridos anteriores.
- Cuenta con un puerto serial tipo RS-232 para conectar una computadora y tener acceso a la información almacenada. También, a través de este puerto se puede hacer el diagnóstico de la unidad de cobro y bajar actualizaciones del sistema operativo.

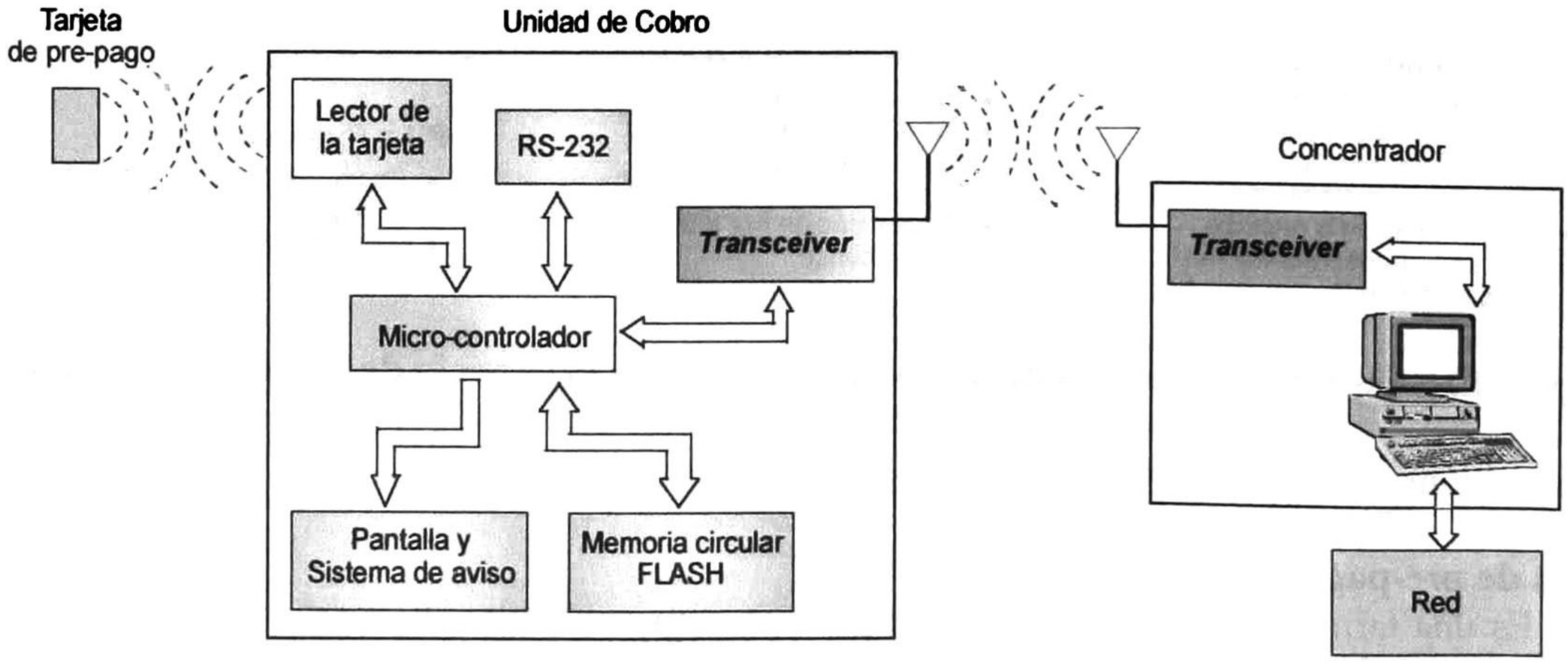


Figura 1.3 Diagrama a bloques de la unidad de cobro y del concentrador.

Concentrador (C)

El concentrador es el sistema basado en una PC que se comunica con las unidades de cobro y se encuentra instalado en las terminales de cada ruta. En la Figura 1.3 se observa el diagrama a bloques del concentrador. Las funciones que realiza son las siguientes:

- Comunica por RF con las unidades de cobro.
- Extrae la información almacenada en las unidades de cobro.
- Almacena la información transmitida por un periodo de hasta 1 año.
- Actualiza en las unidades de cobro la lista de tarjetas no válidas.
- Empaqueta la información extraída a cada unidad y la transmite vía internet al banco.
- Autorización de unidades en una ruta determinada.

Banco

Cuenta con las siguientes funciones:

- Recibe la información del sistema punto de venta de todas las tarjetas de pre-pago abonadas a través de internet.
- Recibe del concentrador las transacciones realizadas durante el recorrido del autobús.
- Abona a las cuentas de cada cliente el pago correspondiente al servicio prestado. Los clientes son: el chofer, el concesionario y los encargados de los puntos de venta.
- Genera una lista de tarjetas no válidas y la envía a los concentradores.

1.2 DIMENSIONAMIENTO DEL SISTEMA

Para determinar algunos parámetros de diseño de las unidades de cobro, por ejemplo: velocidad de procesamiento y cantidad de memoria requerida, se hicieron algunos estudios en sito (autobuses) en algunas de las rutas representativas del transporte urbano.

1.2.1 Información Requerida en el Concentrador

La información que se requiere almacenar a cada transacción es la siguiente:

Celda	Configuración	Bytes
Número asignado al chofer	65536 (valor máx.)	2
Número de ruta	256 (valor máx.)	1
Número del vehículo	65536 (valor máx.)	2
Número de la tarjeta inteligente	16777216 (valor máx.)	3
Fecha	mmddaaaa	3
Hora de la transacción	hhmm	2

Tabla 1.1 Formato de la información.

1.2.2 Caso de Estudio

En Guadalajara las empresas encargadas del transporte urbano son Sistecozome, Alianza de Camioneros, Servicios y Transporte, TUTSA y Características Especiales. En la zona metropolitana existen 183 rutas tomando en cuenta todas las empresas de servicios y transportes [1]. Cada ruta tiene un número variado de unidades, en la Tabla 1.2 se pueden observar algunos casos.

Ruta	Núm. De Unidades
371	60
158-D	28
24	35
632	27

Tabla 1.2 Rutas y número de unidades.

El horario del servicio depende de la ruta, por ejemplo: la ruta 610 ofrece el servicio a partir de las 4:30 AM hasta las 11:00 PM. Otras rutas trabajan desde las 6:00 AM hasta las 10:00 PM de lunes a domingo [2]. Las jornadas de trabajo de los autobuses son de 16 horas en promedio. Existen dos tipos de unidades:

- Minibuses.- Tienen una capacidad máxima de 60 personas donde sólo 35 pueden ir sentadas.
- Autobuses.- Cuentan con una capacidad de 80 personas con sólo 50 asientos.

El número de personas que usan el servicio durante el día depende del horario, y la cantidad de personas aumenta durante las horas pico. Se considera hora pico la hora de la comida, de 1:30 PM a 3:00 M, la hora de entrada al trabajo y escuela, de 7:30 AM a 8:30 AM, y de salida, de 7:00 PM a 8:30 PM. Por lo tanto las horas pico se presentan tres veces durante el día.
A manera de ejemplo se realizó un muestreo en la ruta 632 de la empresa Sistecozome. Esta ruta utiliza minibuses. El horario de servicio de la ruta es de 5:40 AM a 9:30 PM de lunes a domingo. La unidad tarda aproximadamente 120 minutos en hacer su recorrido completo [2]. Tomando en cuenta que de lunes a viernes la afluencia al transporte público es mayor. En el mes de junio del 2001 en el horario comprendido entre las 12:15 PM y las 2:10 PM se realizó el conteo de las personas que utilizaron la unidad de dicha ruta, obteniendo los siguientes datos: el número de usuarios que abordaron la unidad fueron 156, observándose que cada 5 minutos subían un promedio de 7 personas (Figura 1.4).

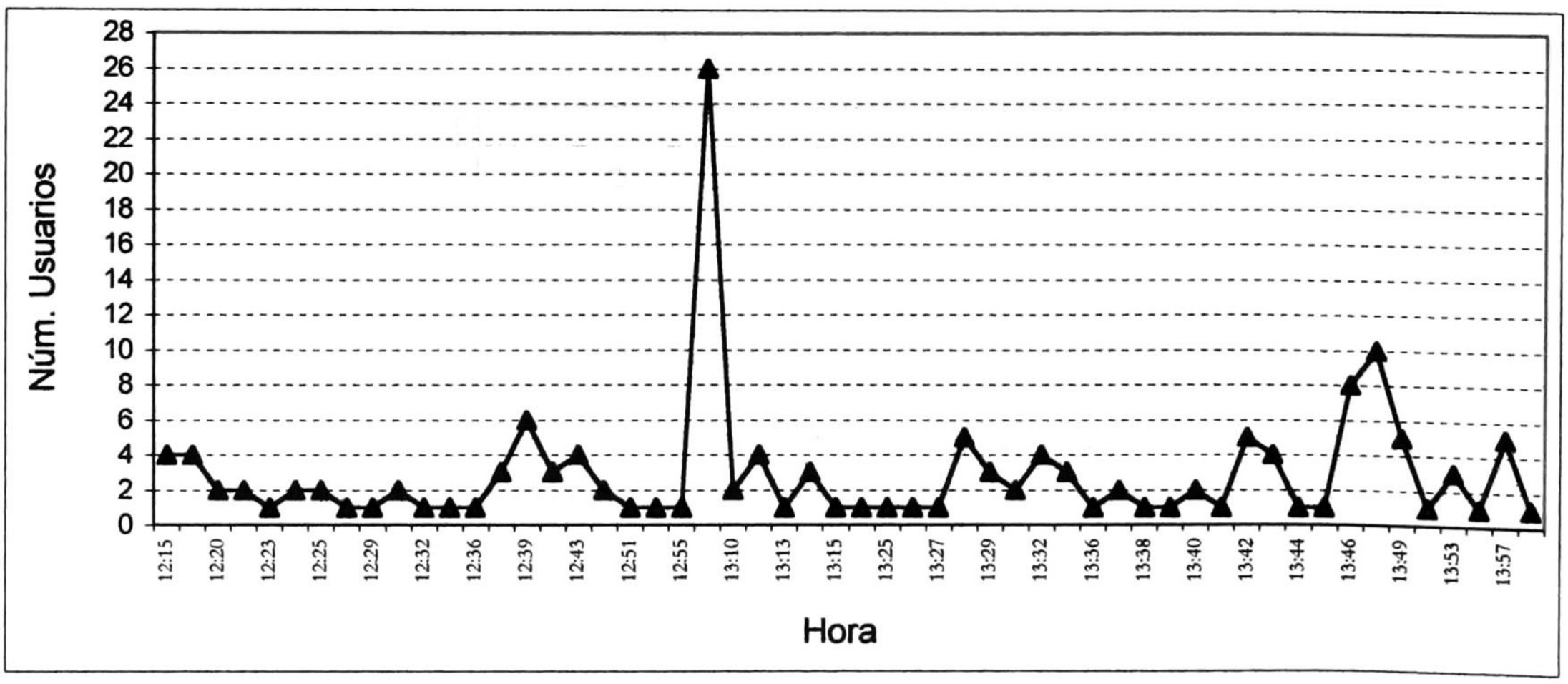


Figura 1.4 Gráfica del muestreo realizado en la ruta 632 entre las 12:15 y 14:10 horas.

La Tabla 1.3 muestra el número de personas contabilizadas en la ruta 632 en un periodo de 1 hora. A partir de este registro se calculó el número de personas que usan el servicio en un día, en una semana y un mes. También se muestra un aproximado del número de usuarios que se tiene con rutas de mayor tiempo de recorrido y unidades más grandes. Como el caso de la ruta 371, que tiene un tiempo de recorrido de 270 minutos en 86 kilómetros de distancia y utiliza unidades que tienen capacidad para 80 personas [3]. En la Tabla 1.3 se muestra los datos recopilados. La columna *parámetro de diseño* corresponde al doble del tamaño de la información recopilada.

Tiempo transcurrido	Número de usuarios atendidos	Tamaño de memoria requerida	Parámetro de diseño
1 hr	78	245 Bytes	489 Bytes
1 día	1,248	5,824 Bytes	11,648 Bytes
1 semana	8,736	40,744 Bytes	81,488 Bytes
1 mes	34,944	162,976 Bytes	325,952 Bytes

Tabla 1.3 Estadística del número de personas que usan el servicio de la ruta 632.

Como resultado de este estudio y de acuerdo a los dispositivos disponibles en el mercado, la memoria adecuada es la de 512 kB con un costo de $3.28 dólares en cantidades de cien. Para nuestros propósitos y por el bajo costo ($5.83 dólares) se seleccionaron memorias de 4 MB que tienen capacidad para almacenar la información de un año.

1.3 BIBLIOGRAFÍA

[1] http://www.ceit.org.mx
[2] http://sistecozome.jalisco.gob.mx
[3] http://www.alianza-camioneros.com.mx

CAPITULO II

2 REQUERIMIENTOS Y OBJETIVOS

2.1 INTRODUCCION

En esta sección se establecen los requerimientos que debe cumplir tanto el concentrador como la unidad de cobro para garantizar una buena funcionalidad. Para la realización del presente trabajo se plantearon los siguientes requerimientos: los requerimientos del hardware (**RH**) y los requerimientos del software (**RS**).

2.2 REQUERIMIENTOS DEL HARDWARE

El hardware proporciona un medio de comunicación entre el concentrador y las unidades de cobro. En el apéndice C se define el equipo a utilizar de acuerdo a las siguientes funciones y características requeridas:

RH1.- Distancia mínima de alcance.
Se requiere que el maestro se comunique con las tarjetas esclavos a una distancia mínima de 100 m en línea directa.

OH1.- Tipo de Modulación.
Equipo basado en una modulación FSK o GFSK dentro de la banda ISM, para lograr cierta inmunidad ante la presencia de ruido.

RH2.- Comunicación full-duplex.
Se requiere un medio de comunicación full-duplex al establecer un enlace entre el maestro y un esclavo.

RH3.- Compatibilidad con computadoras personales como anfitrión.
Se requiere un equipo maestro compatible con PC bajo ambiente Windows 9x.

RH4.- Equipo esclavo tipo stand-alone basado en el microcontrolador 8051.

OH2.- Manejo del Protocolo SyT (Sistemas y Transportes)
Equipo capaz de manejar el protocolo SyT para establecer un enlace en tiempo real.

OH3.- Sistema de seguridad
Se requiere que el equipo cuente con los dispositivos CRC, HEC, cifrado y autentificación.

RH5.- Transferencia de datos digitales.
Equipo capaz de transferir texto, imágenes y voz.

RH6.- Flexibilidad de los protocolos.
Se requiere un equipo capaz de adoptar diferentes protocolos de comunicación.

RH7.- Equipo transportable.
Equipo integrado con dispositivos VLSI para reducir su tamaño, logrando que sea transportable.

RH8.- Equipo Modular.
El equipo debe ser modular para lograr un mejor acoplamiento con otros bloques o sistemas.

RH9.- Voltaje de alimentación.
El voltaje de alimentación en el equipo es de 12 voltios.

RH10.- Potencia máxima de transmisión.
Se requiere un equipo con un nivel de potencia máxima de transmisión a 4 dBm.

RH11.- Velocidad mínima de transmisión.
Equipo con velocidad mínima de transmisión de 9,600 bits por segundo.

RH12.- Desviación de frecuencia.
Equipo con una desviación de frecuencia mínima de 50 a 95 Khz.

RH13.- Sensibilidad de modulación DC.
El nivel de sensibilidad de modulación DC mínima del equipo es de –88 dBm.

RH14.- Alto rechazo a la interferencia.
Se requiere un equipo con un rechazo de interferencia de emisiones espurias máxima de –4.8 dBm.

RH15.- Ganancia RSSI
Equipo con ganancia de 24 mV/dB.

RH16.- Precisión en la frecuencia requerida.
El equipo debe contar con una alta precisión de la frecuencia requerida para transmitir y recibir de ±50 Khz.

RH17.- Impedancia RF de entrada.
Se requiere un equipo que tenga una impedancia RF de entrada de 50Ω.

RH18.- Antena eficiente.
La antena deberá de tener un excelente diseño para atraer lo menos posible las señales no deseadas y radiar la información manera eficiente.

RH19.- Costo del equipo en cantidades de 10,000 unidades.
Costo de la tarjeta esclavo menor a $100 dólares.
Costo de la tarjeta maestro menor a $100 dólares.

2.3 REQUERIMIENTOS DEL SOFTWARE

El software se encarga de establecer una comunicación entre el concentrador y las unidades de cobro. Por lo que las características deberán de ser las siguientes:

RS1.- Interfaz usuario-maestro (IUM)
Generar una pantalla en Windows para que el usuario interactúe con el sistema operativo del maestro.

RS2.- Ventana Estado de la transmisión
Proporcionar una ventana en la IUM para monitorear el estado de la transmisión de la información. La ventana indica el número del mensaje respuesta a petición recibido o retransmitido.

RS3.- Ventana de unidades detectadas
Proporcionar una ventana en la IUM que muestre el número del esclavo que se encuentra dentro del rango de enlace del maestro.

RS4.- Ventana TX/RX
Proporcionar una ventana en la IUM para monitorear el estado actual del maestro.

RS5.- Menú
La IUM deberá proporcionar un menú para configurar la transmisión (Puerto, Conectar, Desconectar y Salir).

RS6.- Ventana # de búsqueda
Proporcionar una ventana en la IUM que indique el número de veces que el maestro ha buscado a todos los esclavos sin recibir respuesta.

RS7.- Ventana de imágenes
Proporcionar una ventana en la IUM que muestre archivos del tipo JPG recibidos.

RS8.- Interfaz usuario-esclavo (IUE)
Generar una pantalla en Windows para que el usuario interactúe con el sistema operativo del esclavo.

RS9.- Ventana Estado de la transmisión
Proporcionar una ventana en la IUE que muestre el número del mensaje respuesta a petición y búsqueda de esclavos disponibles.

RS10.- Ventana de información
Mostrar el nombre del archivo a transmitir y una barra progresiva que indica el porcentaje de la información enviada. También debe mostrar el número identificador de la unidad que le pertenece.

RS11.- Menú
La IUE deberá proporcionar un menú para configurar la transmisión (Puerto, Conectar, Desconectar, Archivo y Salir).

RS12.- Generación del mensaje.
Tipos de mensajes:

- Búsqueda de esclavos disponibles.
- Respuesta de disponibilidad.
- Petición de información.
- Respuesta a petición.
- Acuse de recibo.
- Lista de usuarios no autorizados.

RS13.- Extracción de la información
Proporcionar una secuencia para extraer los bytes del mensaje que llegan byte a byte al buffer serial de entrada.

RS14.- Detección del inicio y final del mensaje.
Determinar el inicio o final de un mensaje mediante la identificación de la bandera de un byte (7Eh).

RS15.- Comprobar el CRC del mensaje.
Calcular e indicar si el mensaje contiene algún error, a través de la prueba de redundancia cíclica.

RS16.- Validación del mensaje.
Indicar el tipo de mensaje obtenido del buffer de entrada serial.

RS17.- Generación de la bandera.
Proporcionar una secuencia para agregar las banderas de inicio y final al mensaje.

RS18.- Envío de bytes.
Generar una función para enviar los bytes del mensaje al buffer de salida serial.

RS19.- Unidades dentro de rango.
Detectar a las unidades que están dentro del rango de alcance del maestro mediante la transferencia de mensajes.

RS20.- Lista de unidades.
Generar una lista de todas las unidades localizadas dentro del rango de alcance del maestro.

RS21- Evitar duplicidad.
Proporcionar un proceso que evite la duplicidad de las unidades en lista de las unidades.

RS22.- Cantidad de información.
El maestro solicita la cantidad de información a recibir por parte del esclavo.

RS23.- Acuse de recibo.
Proporcionar una secuencia para que el maestro informe al esclavo si la transferencia de información fue exitosa.

RS24.- Comprobación del CRC en la carga útil.
Determinar e indicar si la carga útil contiene algún error a través de la prueba de redundancia cíclica de 16 bits.

RS25.- Retransmisiones.
Generar una retransmisión del mensaje cuando finalice el tiempo de espera.

RS26.- Almacenamiento de la información.
Proporcionar una función para almacenar en la memoria del maestro la información recibida.

RS27.- Transmisiones inconclusas.
Determinar e indicar cuando se interrumpa una transmisión maestro-esclavo.

RS28.- Monitoreo de esclavos.
Indicar el número de esclavos detectados por el maestro durante la primer búsqueda.

RS29.- Monitoreo de las retransmisiones.
Determinar e indicar el número de retransmisiones realizadas durante la transferencia de información.

RS30.- Exactitud de la información.
Determinar si la información existente en la memoria del maestro es igual a la información que el esclavo transmitió.

Los requerimientos de hardware y software representan la funcionalidad del tema de tesis que se exhibe. A media que se describa el presente trabajo, se irán cubriendo uno a uno los anteriores requerimientos.

CAPITULO III

3. PROTOCOLO DE COMUNICACIÓN AD-HOC AL PROBLEMA ENTRE UN UC Y UN C

3.1 DESCRIPCIÓN

El esquema propuesto en este trabajo corresponde al de una piconet (apéndice B), donde el maestro (C) puede encender y apagar un número máximo de 8191 esclavos (UC) y atender uno solo a la vez.

El proceso es simple, de acuerdo a una lista autorizada, el maestro llama a cada uno de los esclavos disponibles en el área de enlace. Con aquellos que responden, el maestro hace una lista prioritizada de los esclavos que le interesa atender y comienza en forma secuencial la extracción de la información de las unidades de cobro.

A una velocidad de 19,200 bps, el sistema es capaz de bajar en un segundo la información recolectada por la unidad de cobro (esclavo) en un recorrido de dos horas y tarda aproximadamente tres minutos en bajar la información correspondiente a un mes.

A continuación describimos la arquitectura y el protocolo utilizados para establecer una enlace de comunicación inalámbrica entre el concentrador y la unidad de cobro.

3.2 ARQUITECTURA

3.2.1 Introducción

La arquitectura propuesta sigue el modelo maestro-esclavo donde los esclavos son las unidades de cobro instaladas en los autobuses y el maestro es el sistema concentrador instalado en la terminal de autobuses, ver Figura 3.1.

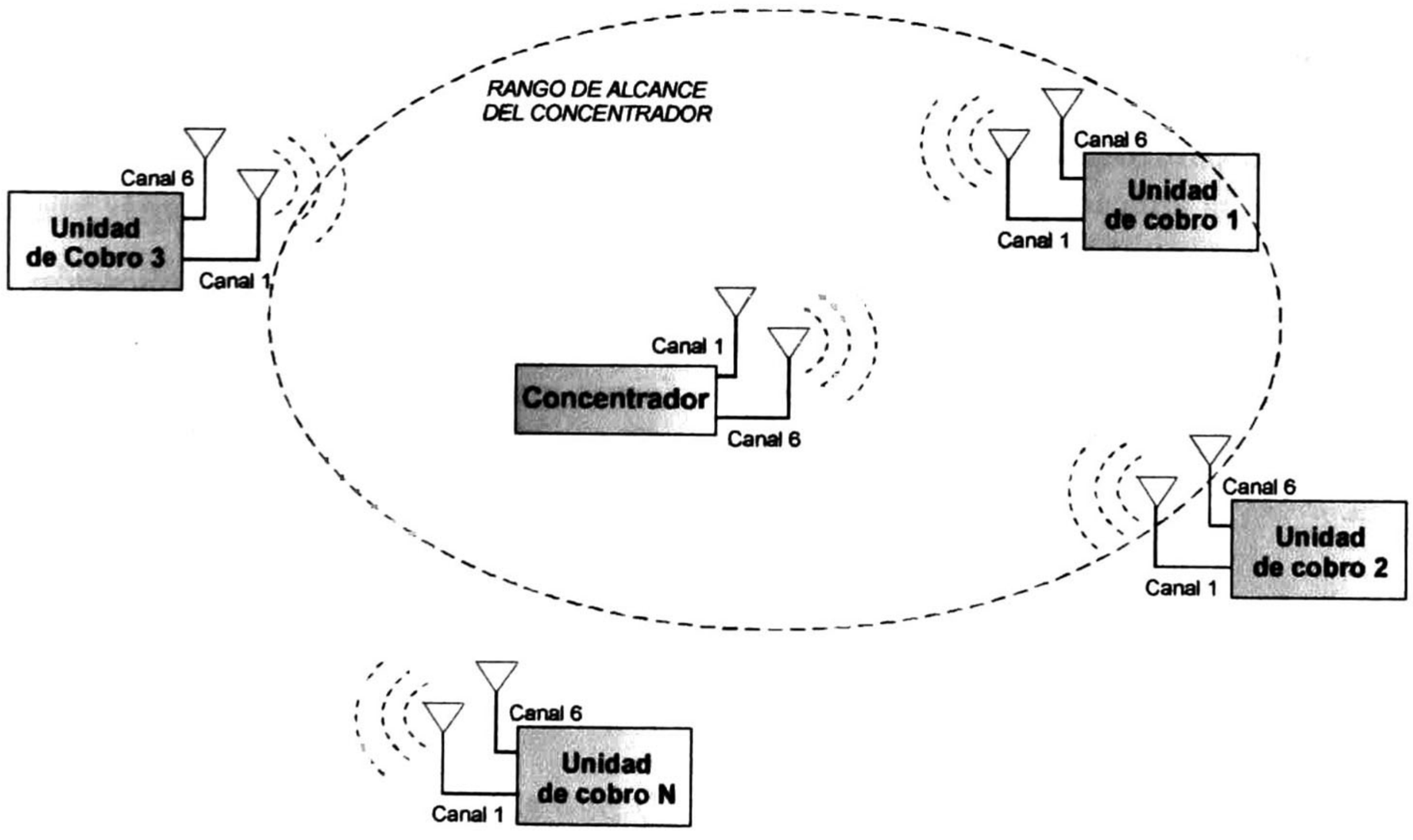


Figura 3.1 Arquitectura de la red con sistemas MEDV-900-HP de LINX.

3.2.2 Selección de Frecuencias

Dados los requerimientos de hardware y de software presentados en el siguiente capítulo, se tomó la decisión de usar dos canales distintos para la transmisión y la recepción.

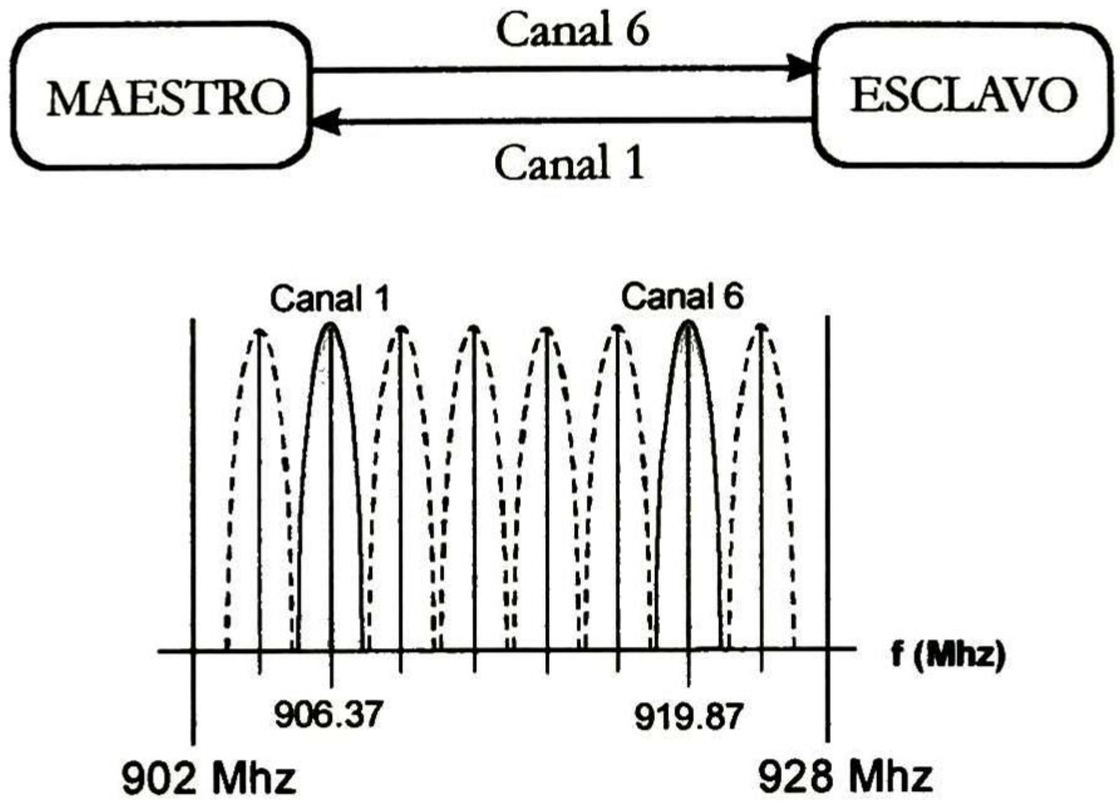


Figura 3.2 Canales de comunicación y espectro del MEVD-900-HP.

De esta manera, se crea una comunicación full-duplex que le permite al maestro controlar de manera eficiente la comunicación, por ejemplo, en caso de detectar un error, el maestro puede parar la transmisión del esclavo y pedir la retransmisión del ultimo bloque.

Los canales uno y seis (ver Figura 3.2) fueron seleccionados para este sistema después de medir el desempeño (interferencia entre canales) en pruebas de campo con los ocho canales disponibles y en diferentes lugares físicos.

3.3 PROTOCOLO

3.3.1 Introducción

La comunicación esta controlada por el maestro el cual tiene en su memoria la identidad de todos los esclavos existentes en la aplicación.

El siguiente proceso se repite continuamente.

i. El maestro pasa lista de todos los esclavos disponibles (máximo 8191) en la aplicación.
ii. El esclavo al escuchar su identificación responde con un mensaje de disponibilidad.
iii. Con cada mensaje de disponibilidad, el maestro forma una lista de los esclavos disponibles.
iv. De acuerdo a la lista, el maestro comienza a pedir a cada uno de los esclavos la información requerida.

3.3.2 Formato de Mensaje

El formato consta de un encabezado, una carga útil y un CRC . Los mensajes están separados por una o más banderas, ver Figura 3.3.

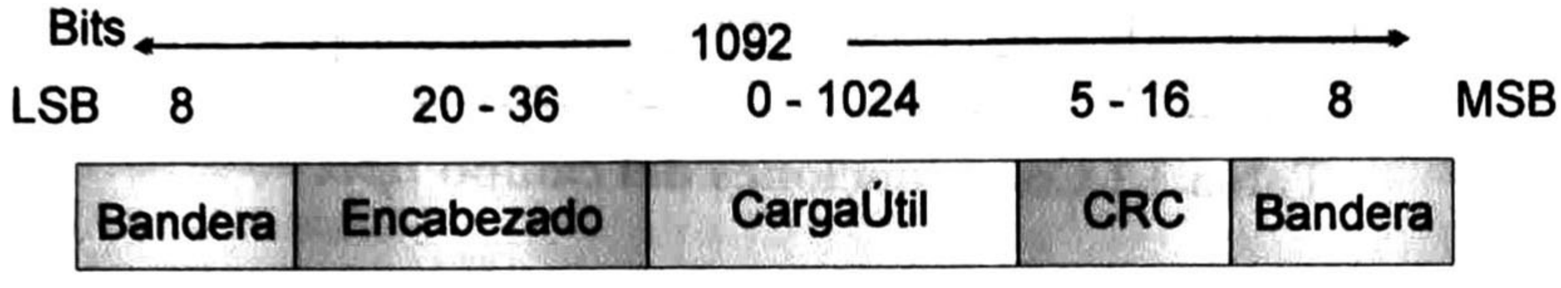


Figura 3.3 Formato general del protocolo para la comunicación inalámbrica.

3.3.2.1 Banderas

Las banderas tienen una longitud de 8 bits y se definen con el patrón 0111 1110 (7E en hexadecimal). La presencia de este patrón esta prohibida dentro del *encabezado*, *carga útil* o *CRC*, ya que la presencia de este patrón indica el fin de mensaje. Para prevenir esto, en la transmisión se aplica un algoritmo de inserción de ceros que inserta un "0" cada vez que encuentra cinco "1's" consecutivos. En la recepción cada vez que se encuentran cinco "1's" consecutivos el sexto bit es removido automáticamente, ver Figura 3.4.

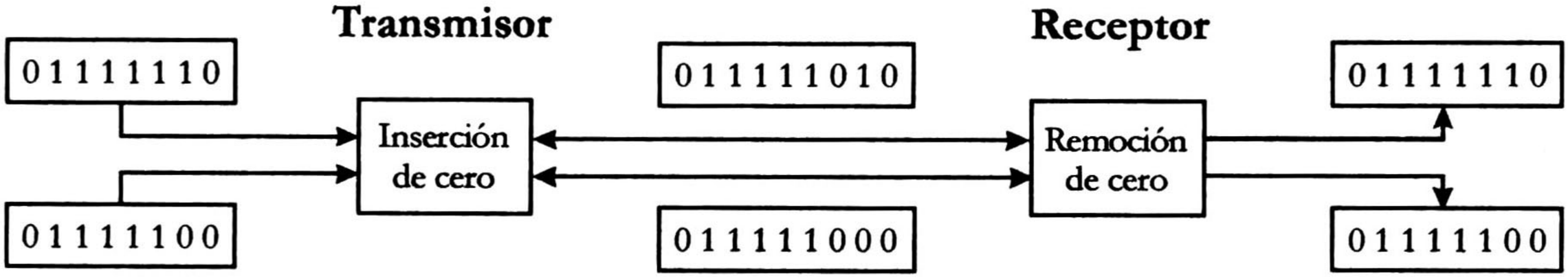


Figura 3.4 Algoritmo de inserción y remoción de cero.

3.3.2.2 Encabezado

En la Figura 3.5 se muestra el formato general del *encabezado*. El formato tiene una longitud variable de 20 o 36 bits, dependiendo del Tipo de mensaje.

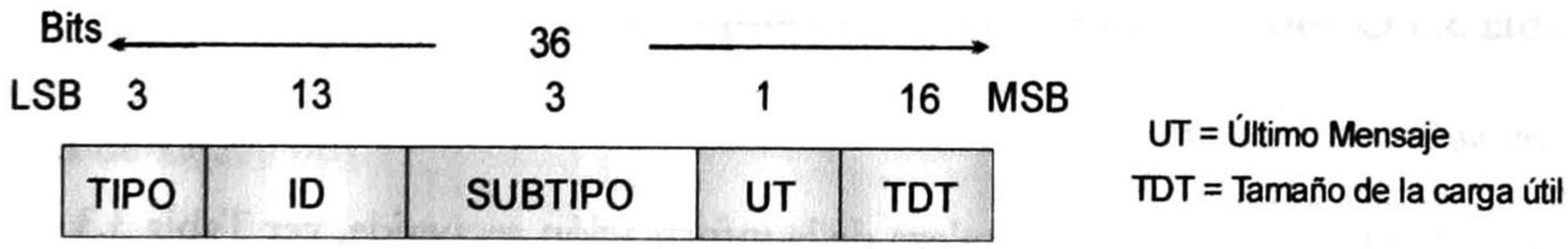


Figura 3.5 Formato general del encabezado.

A continuación se describe cada campo del *encabezado*.

TIPO

Este campo define el tipo de mensaje a usar durante la comunicación y tiene una longitud de 3 bits.

Mensajes iniciados por el maestro.

TIPO $b_0b_1b_2$	Descripción del campo *tipo*
0 0 0	*Uso futuro*
0 0 1	*Búsqueda de esclavos disponibles*
0 1 1	*Petición de la información (ver SUBTIPO)*
1 0 0	*Acuse de recibo por bloque*
1 1 0	*Lista de usuarios no autorizados*
1 1 1	*Uso futuro*

Tabla 3.1 Combinaciones del campo *tipo*.

- *Mensajes iniciados por el esclavo.*

TIPO $b_0b_1b_2$	Descripción del campo *tipo*
0 1 0	*Respuesta de disponibilidad*
1 0 1	*Respuesta a petición (ver SUBTIPO)*

Tabla 3.2 Combinaciones válidas del campo *tipo*.

ID – Identificador

- *Mensajes iniciados por el maestro.*
  El ID carga el identificador del esclavo al cual va dirigido el mensaje.

- *Mensajes iniciados por el esclavo.*
  El ID carga la identificación del esclavo.

SUBTIPO

Este campo está compuesto de 3 bits y tiene 2 acepciones que dependen del campo TIPO.

SUBTIPO $b_{16}b_{17}b_{18}$	Descripción del campo *SUBTIPO*
0 0 0	Uso futuro
0 0 1	Información del último recorrido
0 1 0	Información del último día
0 1 1	Información de la última semana
1 0 0	Información del último mes
1 0 1	Información de los últimos seis meses
1 1 0	Información del último año
1 1 1	Uso futuro

Tabla 3.3 Combinaciones válidas del campo *SUBTIPO* cuando *TIPO* es 011.

Mensajes iniciados por el maestro.
Si el mensaje es del TIPO 011.
El campo SUBTIPO define la naturaleza de la información requerida, ver Tabla 3.3.
Si el mensaje es del TIPO 100.
El campo SUBTIPO define una numeración secuencial (000 a 111) para reconocer las retransmisiones.

Mensajes iniciados por el esclavo.
Solo se utiliza con mensajes TIPO 101.
El esclavo ordena los mensajes con una numeración secuencial (000 a 111) para identificar las retransmisiones.

UT – Último Mensaje

- *Mensajes iniciados por el maestro.*
  Este campo sólo se activa (bit = 1) cuando se transmite el último mensaje *acuse de recibo.*

- *Mensajes iniciados por el esclavo.*
  Este campo sólo se activa (bit = 1) cuando se transmite el último mensaje *respuesta a petición.*

TDT – Tamaño del Mensaje

- *Mensajes iniciados por el maestro.*
  Este campo indica el tamaño en bytes de la carga útil sólo en la transmisión de la *lista de usuarios no autorizados.* TDT tiene una longitud de 15 bits.

- *Mensajes iniciados por el esclavo.*
  El campo indica el tamaño en bytes de la carga útil en el mensaje *respuesta a petición.*

3.3.2.3 Carga Útil

La carga útil transporta bloques de información entre los esclavos y el maestro. Su longitud máxima es de 1024 bits (128 bytes). El campo se utiliza con los mensajes de *respuesta a petición* y *lista de usuarios no autorizados.* La carga útil se divide en dos campos: *cuerpo de carga útil* y *CRC-C,* como se muestra en la Figura 3.6.

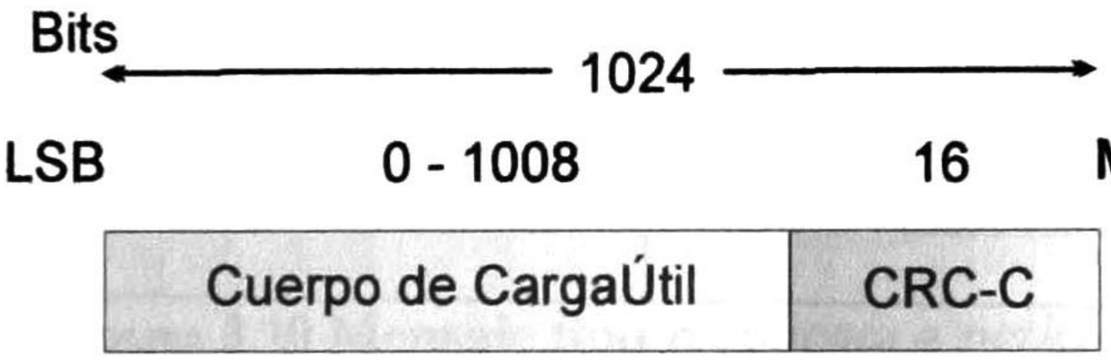


Figura 3.6 Formato de la carga útil.

Cuerpo de carga útil

El campo contiene una determinada cantidad de información a transmitir. La longitud del campo es de 0 a 1008 bits. Cuando la información a transmitir tiene una longitud mayor a la de la carga útil, el mensaje se divide en varios bloques conteniendo el tamaño máximo del campo. La Tabla 3.4 muestra diferentes bloques que están en función al tamaño de la información a transmitir (capacidad); también se indica la duración de la transmisión, en segundos, a una velocidad de transmisión de 19,200 bps.

Tiempo de almacenamiento	Capacidad	Bloques	Duración de la transmisión
2 hrs	978 B	8	0.736 segundos
1 día	11,648 B	93	7.72 segundos
1 semana	81,488 B	637	52.43 segundos
3 semanas	244,464 B	1941	159.6 segundos
1 mes	325,952 B	2587	212.7 segundos

Tabla 3.4 Número de mensajes *respuesta a petición* necesarios para transmitir.

CRC del Cuerpo de Carga Útil (CRC-C)

Este campo contiene el valor obtenido en la prueba de redundancia cíclica aplicada al cuerpo de la carga útil. El polinomio generador utilizado es $1 + D^2 + D^5 + D^{12} + D^{16}$ [6]. La longitud del campo es de 16 bits.

3.3.2.4 Prueba de Redundancia Cíclica (CRC)

El campo se presenta en el formato del mensaje de la Figura 3.3 y contiene el resultado de la prueba de redundancia cíclica aplicada al mensaje sin incluir las banderas. Para los mensajes de *respuesta de petición* y *lista de usuarios no autorizados* se tiene un CRC de 16 bits y el polinomio generador utilizado es $1 + D^2 + D^5 + D^{12} + D^{16}$ [6]. Para el resto de los mensajes se utiliza un CRC con longitud de 5 bits y el polinomio generador es $1 + D^2 + D^5$ [3].

3.3.3 Tipos de Mensajes

Los mensajes que se intercambian en este tipo de comunicación son los siguientes:

Maestro	Esclavo
Búsqueda de esclavos disponibles	Respuesta de disponibilidad
Petición de información Información del último recorrido Información del último día Información del último mes Información del último seis meses Información del último año	Respuesta a petición Información del último recorrido Información del último día Información del último mes Información del último seis meses Información del último año
Acuse de recibo por bloque	
Actualizar lista de usuarios no autorizados en la memoria del esclavo	

A continuación se muestra el formato de cada mensaje.

3.3.3.1 Mensaje tipo búsqueda de esclavos disponibles y respuesta de disponibilidad

El maestro utiliza el mensaje de *búsqueda de esclavos disponibles* para identificar a los esclavos que se encuentran dentro del área de enlace. En la Figura 3.7 se muestra el formato del mensaje.

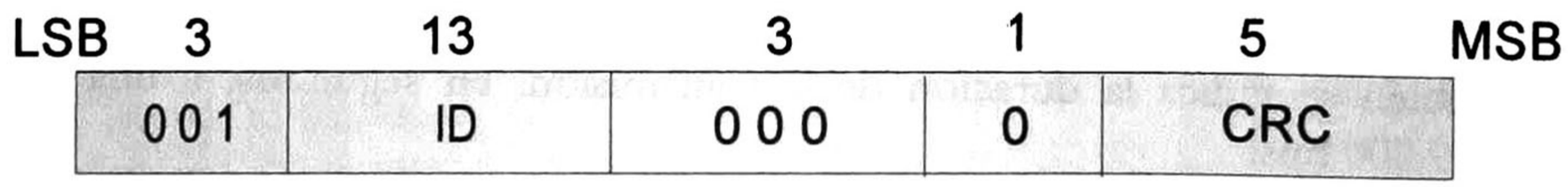


Figura 3.7 Mensaje tipo *búsqueda de esclavos disponibles.*

ID varia de acuerdo a la lista de vehículos autorizados en la ruta que se está vigilando. Cuando un esclavo identifica su ID, responde al maestro con un mensaje de *respuesta de disponibilidad.* El formato se muestra en la Figura 3.8.

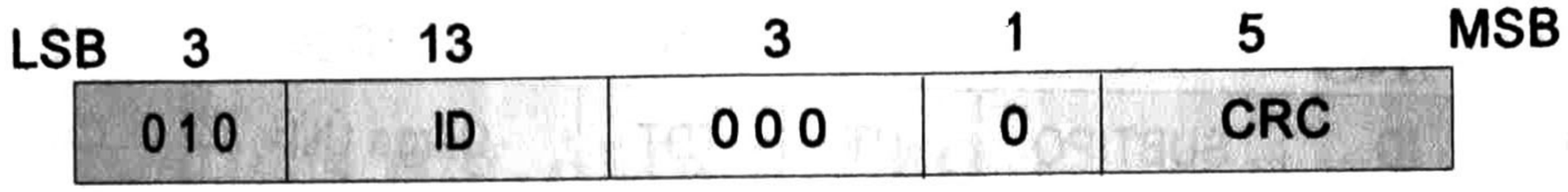


Figura 3.8 Mensaje tipo *respuesta de disponibilidad.*

3.3.3.2 Mensaje tipo petición de información

El maestro utiliza el mensaje para indicarle a un esclavo específico la cantidad de información requerida. La Figura 3.9 muestra el formato del mensaje.

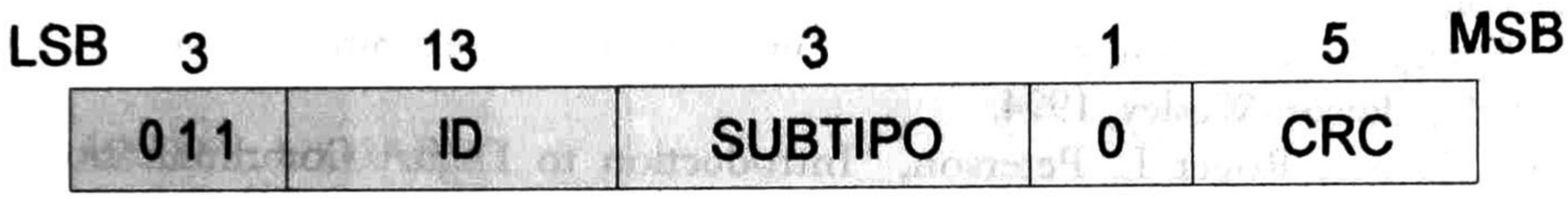


Figura 3.9 Mensaje tipo *petición de información.*

3.3.3.3 Mensaje tipo respuesta a petición y acuse de recibo

El mensaje se utiliza para transmitir la información contenida en los esclavos. En la Figura 3.10 se muestra el formato.

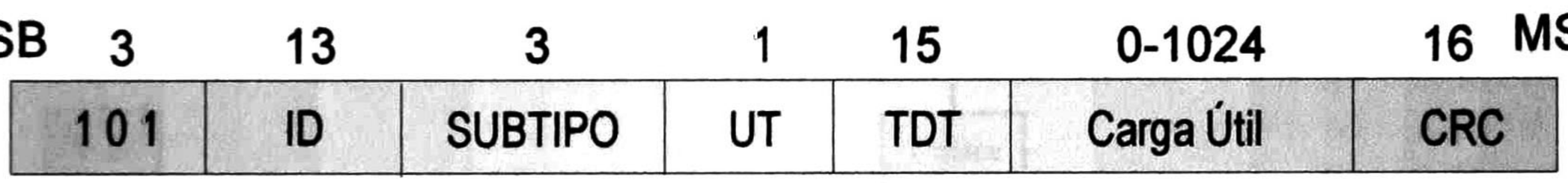


Figura 3.10 Mensaje tipo *respuesta a petición.*

Por cada mensaje tipo *respuesta a petición* que recibe el maestro sin errores, se envía una mensaje tipo *acuse de recibo* al esclavo correspondiente para avisar que puede enviar el siguiente mensaje. El formato del mensaje tipo *acuse de recibo* se muestra en Figura 3.11.

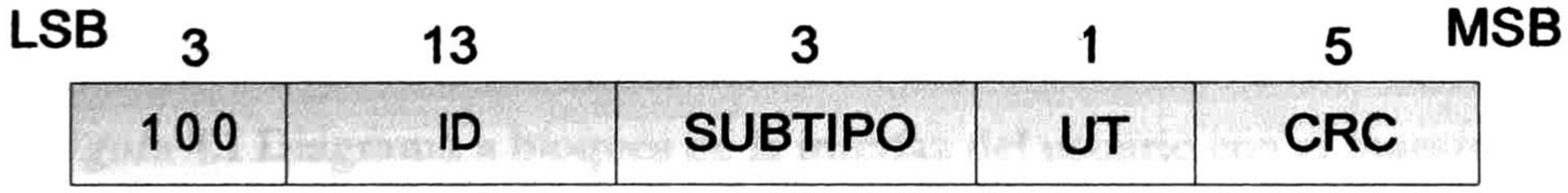


Figura 3.11 Mensaje tipo *acuse de recibo.*

3.3.3.4 Mensaje tipo lista de usuario no autorizado

El mensaje es usado por el maestro para enviar la lista de usuarios no autorizados al esclavo. La Figura 3.12 se muestra el formato.

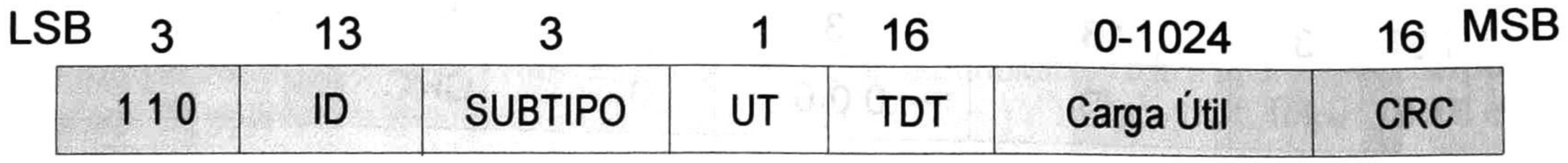


Figura 3.12 Mensaje tipo *lista de usuario no autorizado.*

3.4 BIBLIOGRAFÍA

[1] Uyless D. Black, "Computer Networks: Protocols, Standards and Interfaces", Second Edition, Prentice Hall, 1993.

[2] Jhon D. Spragins, J. Hammond and K. Pawlikowski, "Telecommunications Protocol and Design", Ed. Addison-Wesley, 1994.

[3] Rodger E. Ziemer, Roger L. Peterson, "Introduction to Digital Communication", Second Edition, Prentice Hall, 1992.

[4] Andrew S. Tanenbaum, "Redes de Computadoras", Tercera Edición, Pearson, 1997.

[5] William Stallings, "ISDN and Broadband ISDN", Second Edition, Macmillan Publishing Company, 1992.

[6] http://www.dtr.isy.liu.se/dtr/staff/mikael/irrpoly

CAPÍTULO IV

4. DESCRIPCIÓN DEL ALGORITMO

4.1 DESCRIPCIÓN

En este capítulo se describe el algoritmo usado en el protocolo SyT. El programa se divide en dos bloques:

1. La interfaz con el usuario, pantalla gráfica que interactúa con el usuario.
2. El programa operativo maestro-esclavos.

4.2 INTERFAZ DEL USUARIO

4.2.1 Interfaz del Usuario con el Maestro

La Figura 4.1 muestra el diagrama a bloques del programa que conforma la interfaz gráfica del usuario con el maestro.

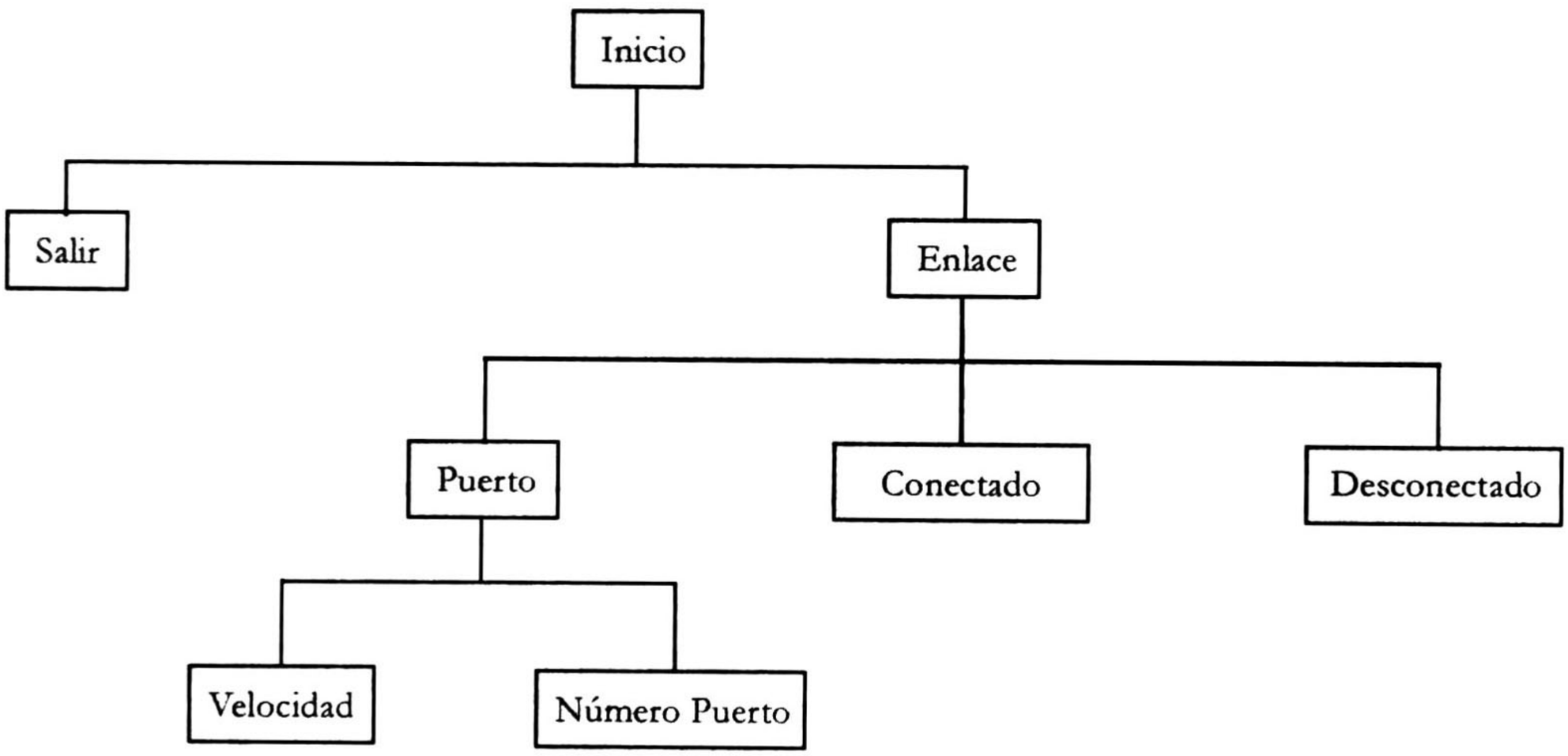


Figura 4.1 Diagrama a bloques de la interfaz del usuario con el maestro.

4.2.1.1 Menú de funciones

En la parte superior de la pantalla del usuario se observa el menú de funciones con dos elementos: *Salir* y *Enlace*, ver Figura 4.2.

- **Salir**
  Cierra la aplicación, libera el puerto serial y la memoria reservada por el programa.
- **Enlace**
  Puerto
  Establece la velocidad de transmisión [por default a 19.2Kbps] y determina el puerto serial a usar [por default el puerto serial 1].

- **Conectado**
  Activa el programa en el maestro para establecer el enlace de comunicación con los esclavos.
- **Desconectado**
  Indica al usuario que el maestro está desconectado.

Figura 4.2 Interfaz gráfica del usuario.

4.2.1.2 Ventanas de diálogo

La pantalla del usuario tiene cinco ventanas que se utilizan para indicar los siguiente eventos:

Ventana "# Búsqueda"
Esta ventana es útil en las etapas de puesta en marcha, verificación de la correcta instalación y pruebas de campo del sistema. Nos indica el número de veces que el maestro ha buscado en vano a los esclavos presentes en el rango de alcance.

Ventana "Estado TX/RX"
Indica el estado actual del maestro:

- **Búsqueda**.- Búsqueda de esclavos que están dentro del área de enlace, ver "Unidad Detectada"
- **Recibiendo**.- Se encuentra recibiendo la información de los esclavos localizados.
- **Desconectado**.- Indica al usuario que el maestro está desconectado.

Ventana de imágenes
Esta ventana fue introducida para llevar a cabo las pruebas de campo y nos permite verificar en forma visual la correcta transmisión de datos entre esclavos y maestro. La ventana ubicada a la derecha de la Figura 4.2 maneja archivos de tipo JPG.

Ventana "Estado de la Transmisión"
Esta ventana central muestra dos tipos de mensajes:

- ***Mensaje recibido***.- indica el número de bloque (1024b) que se está recibiendo de un esclavo en particular. Esta información sólo se despliega cuando se reciben mensajes tipo *respuesta a petición*.
- ***Mensaje retransmitido***.- indica el número de bloque (1024b) que se está retransmitiendo. Esto debido a haber encontrado un error de CRC en el bloque previo.

Ventana "Unidad Detectada"

Muestra la identificación de los esclavos encontrados por el maestro. Los identificadores se despliegan conforme son detectados por el maestro. Posteriormente esta lista y su orden son utilizados para extraer en forma secuencial la información de cada esclavo. Una vez que el esclavo ha entregado la información, el identificador es borrado de dicha ventana.

Ventana Inferior

En la parte inferior de la Figura 4.2 se muestran las siguientes etiquetas:

- **Estado** – Indica si el maestro esta conectado o desconectado.
- **Velocidad** – Muestra la velocidad de transmisión.
- **Puerto** – Presenta el puerto serial que se utiliza.
- **Datos recibidos** – Indica el número de bytes extraídos al esclavo durante la comunicación.

4.2.1.3 Botón de Conexión / Desconexión

El botón con la etiqueta "Desconectar" es un acceso directo a la acción Enlace → Desc*onectar*. Cuando el botón es presionado la etiqueta cambia a "Conectar" representando así la acción futura Enlace→C*onectar*.

4.2.2 Interfaz del Usuario con el Esclavo

Figura 4.3 muestra el diagrama a bloques del programa que conforma la interfaz gráfica del usuario con el esclavo.

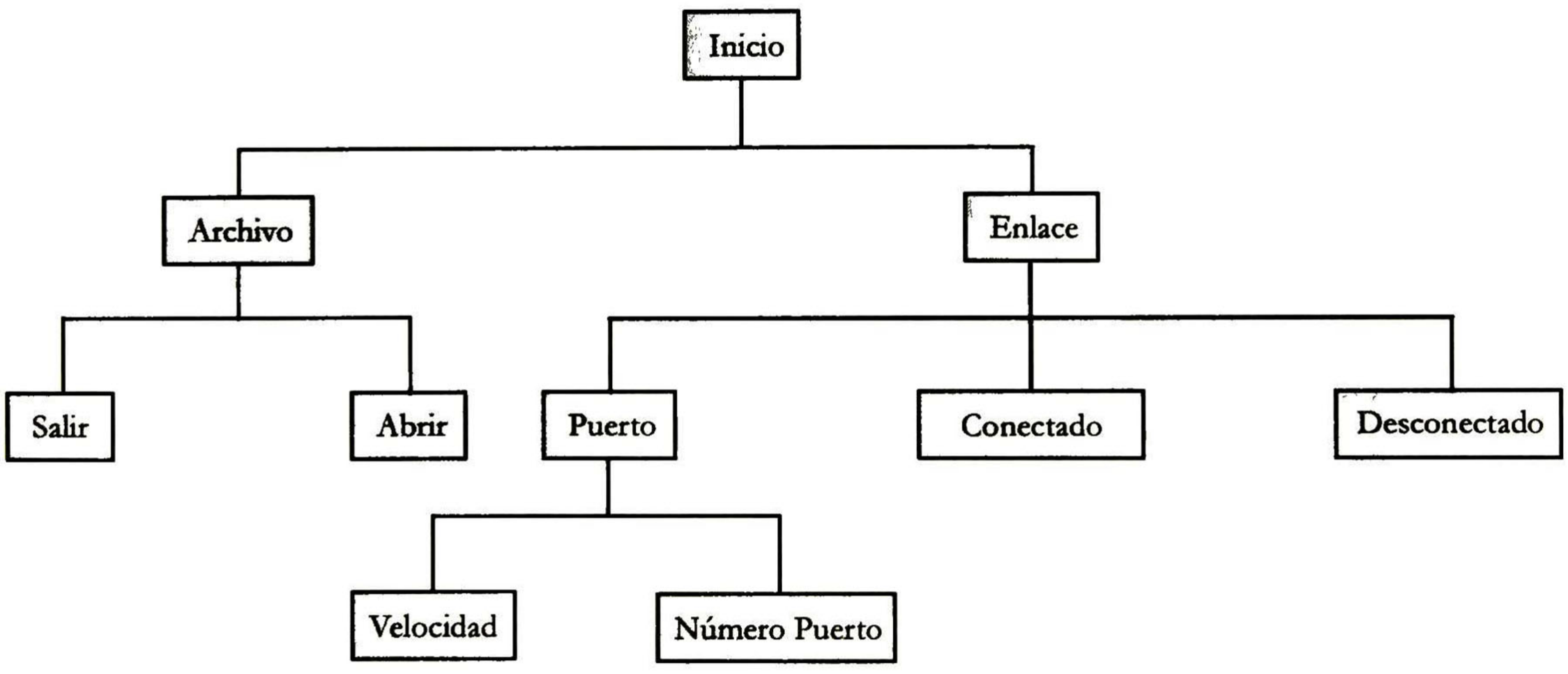


Figura 4.3 Diagrama a bloques de la interfaz del usuario con el esclavo.

4.2.2.1 Menú de funciones

En la parte superior de la pantalla del usuario se observa el menú de funciones con dos elementos: *Archivo* y *Enlace*, ver Figura 4.4.

- **Archivo**
  - **Abrir**
    Selecciona la información que se desea transmitir y le asigna un espacio en la memoria para su posterior procesamiento. Esta opción fue introducida para llevar acabo las pruebas de campo del sistema.
  - **Salir**
    Cierra la aplicación, libera el puerto serial y la memoria reservada por el programa.
- **Enlace**
  - **Puerto**
    Establece la velocidad de transmisión [por default a 19.2Kbps] y determina el puerto serial a usar [por default el puerto serial 1].
  - **Conectado**
    Activa el programa en el esclavo para establecer el enlace de comunicación con el maestro.
  - **Desconectado**
    Indica al usuario que el esclavo está desconectado.

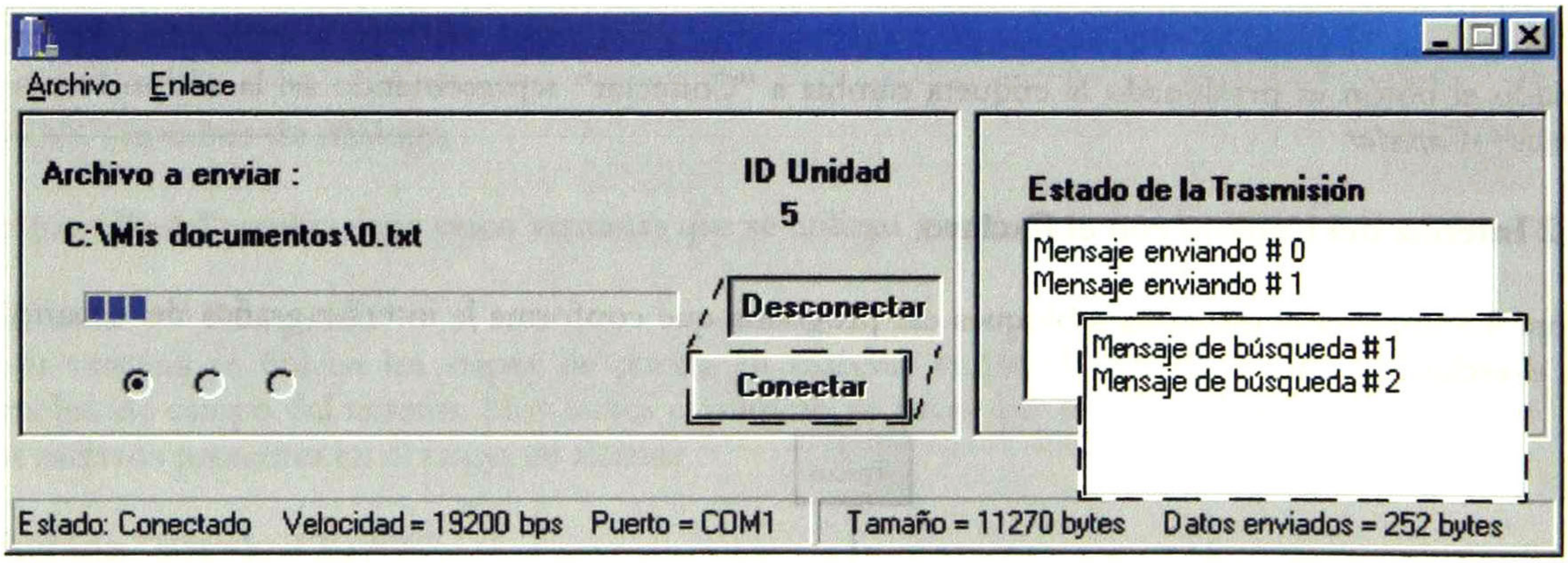


Figura 4.4 Interfaz gráfica del usuario con el esclavo.

4.2.2.2 Ventanas de dialogo

En el marco derecho de la Figura 4.4 se observa:

Ventana "Estado de la Transmisión"
Esta ventana presenta dos tipos de mensajes;

- ***Mensaje enviado***.- Número de bloque (1024b) transmitido al maestro. Durante el proceso de extracción de la información.
- ***Mensaje de búsqueda***.- identificador de los esclavos buscados por el maestro. Durante el proceso de búsqueda de esclavos disponibles.

Ventana Izquierda

En la parte izquierda de la Figura 4.4 se presentan las siguientes etiquetas:

- **ID Unidad 5 –** Representa el número de identificación del esclavo.
- **Archivo a enviar** – Muestra la dirección de acceso y el nombre del archivo que se desea transmitir. Existe también bajo la etiqueta una barra progresiva que muestra el porcentaje de información que se transmite al maestro.

Ventana Inferior

En la parte inferior de la Figura 4.4 se muestran las siguientes etiquetas:

- **Estado** – Indica si el maestro esta conectado o desconectado.
- **Velocidad** – Muestra la velocidad de transmisión.
- **Puerto** – Presenta el puerto serial que se utiliza.
- **Tamaño** – Representa la cantidad de información total, en bytes, a transmitir.
- **Datos enviados** – Indica el número de bytes transmitidos al maestro durante la comunicación.

4.2.2.3 Botones

Por último encontramos una serie de botones como:

Botón de Conexión / Desconexión

El botón con la etiqueta "Desconectar" es un acceso directo a la acción Enlace → Desconectar. Cuando el botón es presionado la etiqueta cambia a "Conectar" representando así la acción futura Enlace → Conectar.

Botones de selección

Se utilizan para habilitar el archivo que se ha de transmitir y dependiendo del botón que se seleccione es el nombre del archivo que se mostrará en la etiqueta "Archivo a enviar" Estos botones se introdujeron para llevar a cabo las pruebas de campo en el sistema.

4.3 PROGRAMA OPERATIVO MAESTRO-ESCLAVO

El programa que define la operación del maestro-esclavo se presenta en el diagrama a bloques de la Figura 4.5. Esta rutina se repite indefinidamente:

1.- "Búsqueda de esclavos" (BE) – el maestro pregunta por cada uno de los esclavos enviando mensajes del tipo *búsqueda de esclavos disponibles*, como se muestra en la Figura 4.5.
2.- "Respuesta del esclavo" (RE) – el esclavo que se encuentra dentro del rango de enlace responde cuando son llamados por el maestro enviando el mensaje *respuesta de disponibilidad.*
3.- "Generación de la lista de esclavos" (GLE) – el maestro recibe el mensaje de disponibilidad y almacena su número identificador para después generar la lista de esclavos encontrados.
4.- "Petición de la información" (PI) – el maestro toma la lista generada y secuencialmente llama a cada uno de los esclavos y les pide la información requerida por el usuario, enviando un mensaje tipo *petición de información* (por default la información del último recorrido).
5.- "Recepción y almacenamiento de la información" (RAI) – el esclavo envía los mensajes *respuesta a petición* para transmitir la información requerida por el maestro (ver Figura 4.5). El maestro transmite un mensaje *acuse de recibo* por cada paquete recibido correctamente.

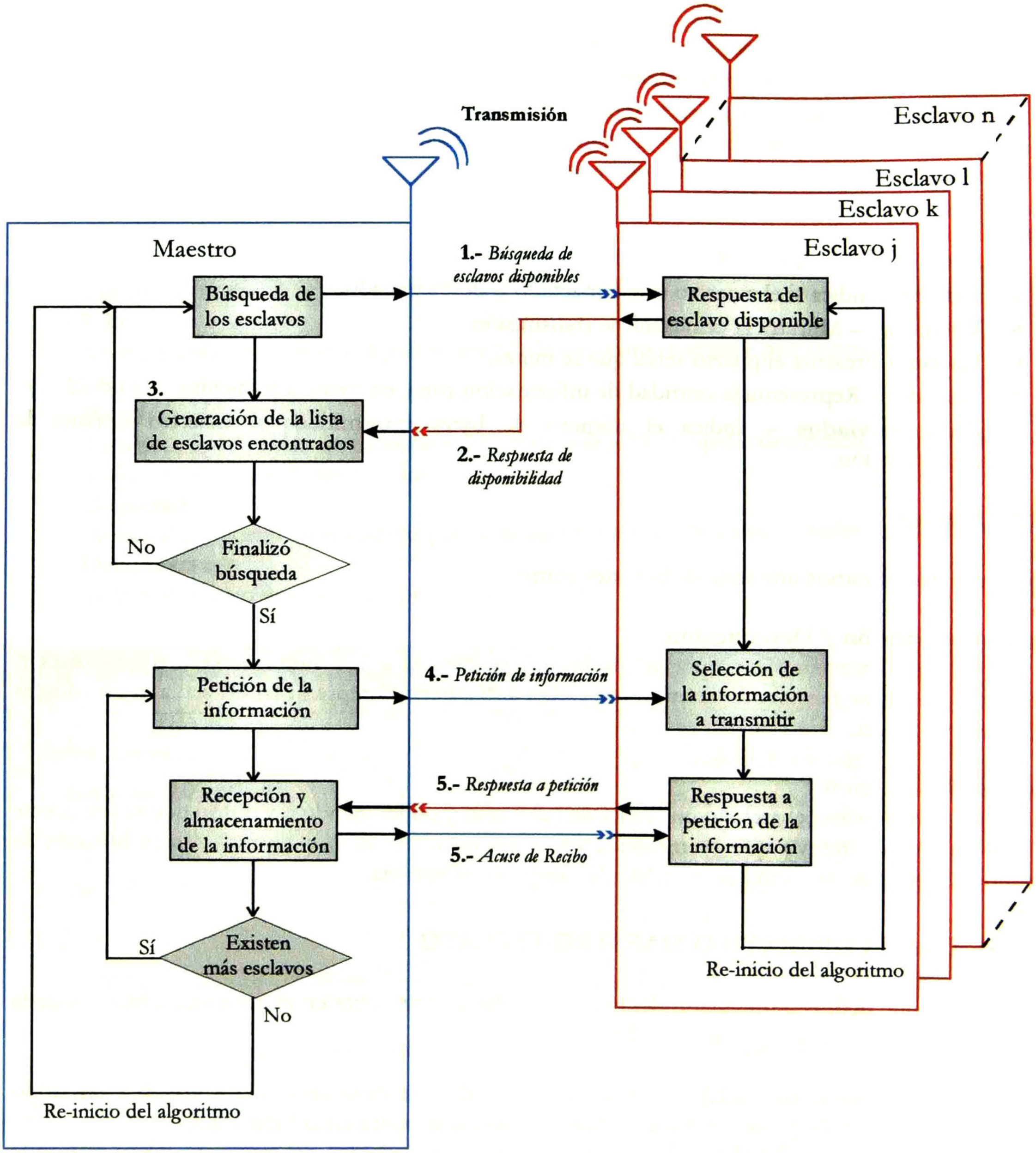


Figura 4.5 Diagrama de intercambio de mensajes del protocolo.

A continuación se describe el programa operativo del maestro y del esclavo.

4.3.1 PROGRAMA OPERATIVO DEL MAESTRO

El programa que define la operación del maestro se divide principalmente en dos bloques: Búsqueda de Esclavos y Recepción-Almacenamiento de la Información.

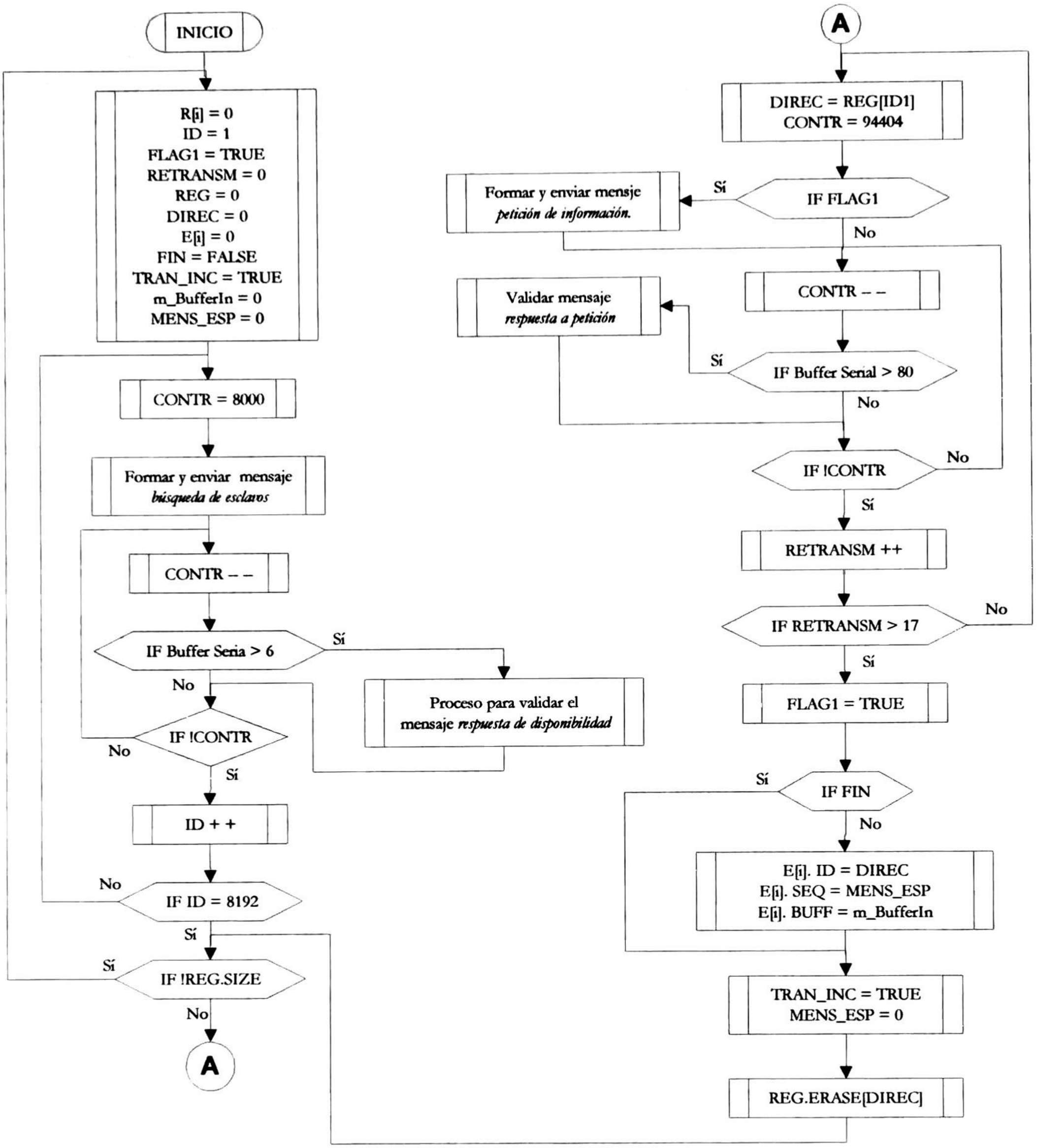


Figura 4.6 Diagrama de flujo del programa existente en el maestro.

En el bloque BE se crea el mensaje *búsqueda de esclavos disponibles*, después se envía al buffer de transmisión serial y se entra a un ciclo de espera (pasos 2a, 2b y 2c). Dentro del ciclo se revisa el buffer de recepción serial, cuando detecta un posible mensaje se valida y se generar la lista de esclavos encontrado (paso 2d). Cuando finaliza dicho ciclo se prepara otro mensaje *búsqueda de esclavos disponibles* para buscar a otro esclavo (pasos 2e y 2f). El bloque se repite hasta que se busquen 8192 esclavos(paso 2g).

Para ejecutar el bloque RAI primero se revisa la lista de esclavos encontrados (paso 3). Una vez dentro de este bloque se extrae de la lista el número de un esclavo para formar y enviar el mensaje *petición de la información* (paso 4b). Se entra a un ciclo de espera para detectar el mensaje

respuesta de la información (paso 4d). Cuando finaliza el ciclo se revisa que la recepción haya concluido, en el caso de que no se reserva un espacio de memoria y se almacena la información necesaria para continuar con la recepción durante otro enlace (paso 4e, 4f y 4g). Se borra de la lista el número del esclavo contactado (paso 4h). El proceso de este bloque se repite hasta que la lista de esclavos encontrados no contenga información.

El diagrama de flujo del programa se muestra en la Figura 4.6. A continuación se mencionan los pasos del procesamiento.

1) Inicialización de las variables

- ID = 1, variable tipo *unsigned long*, identificador del esclavo a buscar (por default =1).
- FLAG1 = TRUE, el esclavo ID no ha respondido, (FALSE el esclavo ID responde a la petición de información). Variable tipo *bool*.
- RETRANSM = 0, var *unsigned int*, registra el número de retransmisiones.
- REG[i] = 0, variable definida como *vector<unsigned long>*, registra la ID de los esclavos encontrados.
- DIREC = 0, var *int*, contiene el identificador del esclavo seleccionado para la petición de la información.
- E[i] = 0 variable definida como *struct* (*unsigned long, unsigned int, unsigned short, bool, std::vector* <*BYTE*>), almacena la información en una transmisión inconclusa.
- FIN = FALSE e l esclavo transmitió toda su información. Var *bool*.
- R[i] = 0 variable tipo *struct* (*unsigned long, unsigned short, unsigned short, bool, unsigned char, unsigned long*), contiene el mensaje extraído del buffer.
- TRAN_INC=TRUE var *bool*, revisa si existen transmisiones inconclusas.
- m_BufferIn = 0, var *std::vector<BYTE>* que almacena la información enviada por el esclavo.
- MENS_ESP = 0, var *unsigned short*, representa el número de la secuencia esperada.

2) Inicia proceso de búsqueda de esclavos

a) CONTR = 8000, establece el tiempo de espera para que un esclavo responda.
b) Se forma el mensaje *búsqueda de esclavos disponibles* y se envía al buffer del puerto serial, ver sección 4.3.1.5.
c) CONTR – –, disminuye el tiempo de espera.
d) Se revisa el buffer de entrada serial para verificar si el esclavo buscado respondió:
   i) Si el buffer contiene más de 6 bytes, se ejecuta una función para validar el mensaje *respuesta de disponibilidad* (sección 4.3.1.1).
   ii) De lo contrario se va al paso siguiente.
e) CONTR igual a 0 (IF !CONTR):
   i) Si es cero, se va al paso (2.f).
   ii) Si es diferente de cero se regresa al paso (2.c).
f) ID ++, para buscar al siguiente esclavo.
g) ID es igual a 8192 (IF ID = 8192):
   i) Si es igual, se continua con el paso 3.
   ii) De lo contrario regresa a paso 2a.

3) Revisión de lista de esclavos encontrados:

a) Si REG.SIZE = 0 se regresa al paso (1).
b) En caso contrario se prosigue con el paso (4).

REG=[ID1, ID2, ID3, ... IDn]

4) Proceso de extracción de la información:

a) DIREC = REG[ID1], se selecciona de la lista al esclavo para la petición de información. CONTR = 94404 (establece el tiempo de espera para que el esclavo transmita).
b) FLAG1 es igual a TRUE (IF FLAG1):
   i) Si es igual, se forma el mensaje *petición de información* y envía al buffer serial, ver sección 4.3.1.5.
   ii) En caso contrario proseguimos con el paso (4.c).
c) CONTR – –, disminuye el tiempo de espera.
d) Se revisa el buffer de entrada serial para verificar si el esclavo seleccionado respondió:
   i) Si el buffer contiene más de 80 bytes, se ejecuta una función para validar el mensaje *respuesta a petición*, ver sección 4.3.1.3.
   ii) De lo contrario se va al paso siguiente.
e) CONTR igual a 0 (IF !CONTR):
   i) Si es cero, RETRANSM ++, incrementa el número de retransmisiones y se prosigue con el paso (4.f).
   ii) En el caso contrario se regresa al paso (4.c).
f) RETRANSM es mayor a 17 (IF RETRANSM > 17):
   i) Si es mayor, FLAG1 = TRUE y se continua el proceso en (4.g).
   ii) Si es menor se regresa al paso (4.a).
g) FIN es igual a TRUE (IF FIN):
   i) Si es igual, se continua con (4.h).
   ii) Si no finalizó, se reserva un espacio de memoria para almacenar la información recibida. E[i].ID = ID del esclavo(DIREC).
   E[i].SEQ = número de la secuencia esperada del mensaje (MENS_ESP).
   E[i].BUFF = información recibida(m_BufferIn).
   Esta opción fue introducida para llevar acabo las pruebas de campo de los sistemas.
h) TRAN_INC = TRUE y MENS_ESP = 0.
i) REG.ERASE[DIREC], se borra de la lista el esclavo que estuvo transmitiendo. Se regresa al paso (3).

Con esto se concluye la secuencia de pasos que realiza el programa operativo del maestro para establecer una comunicación inalámbrica. El apéndice D contiene el código del programa operativo del maestro descrito en lenguaje de alto nivel Builder C++.

4.3.1.1 Validación del Mensaje *Respuesta de Disponibilidad*

La función obtiene el mensaje existente en el buffer de recepción serial (paso 2). Cuando se tiene el mensaje completo se verifica que no contenga errores y se identifica el tipo (paso 3 y 4). Si es del tipo respuesta de disponibilidad el número del esclavo se almacena en la lista de esclavos encontrados (paso 5). En la Figura 4.7 se observa el diagrama de flujo de la función.

Los pasos que se siguen para realizar el procesamiento de la función son:

1) Generación de las variables:

- CRC = 0, valor de la prueba de redundancia cíclica aplicada al mensaje. Variable tipo *unsigned long.*
- COMPT = FALSE, el mensaje no está completo (TRUE, mensaje completo). Var *bool.*
- DIRECC1 = 0, var *unsigned long,* almacena el identificador del esclavo.

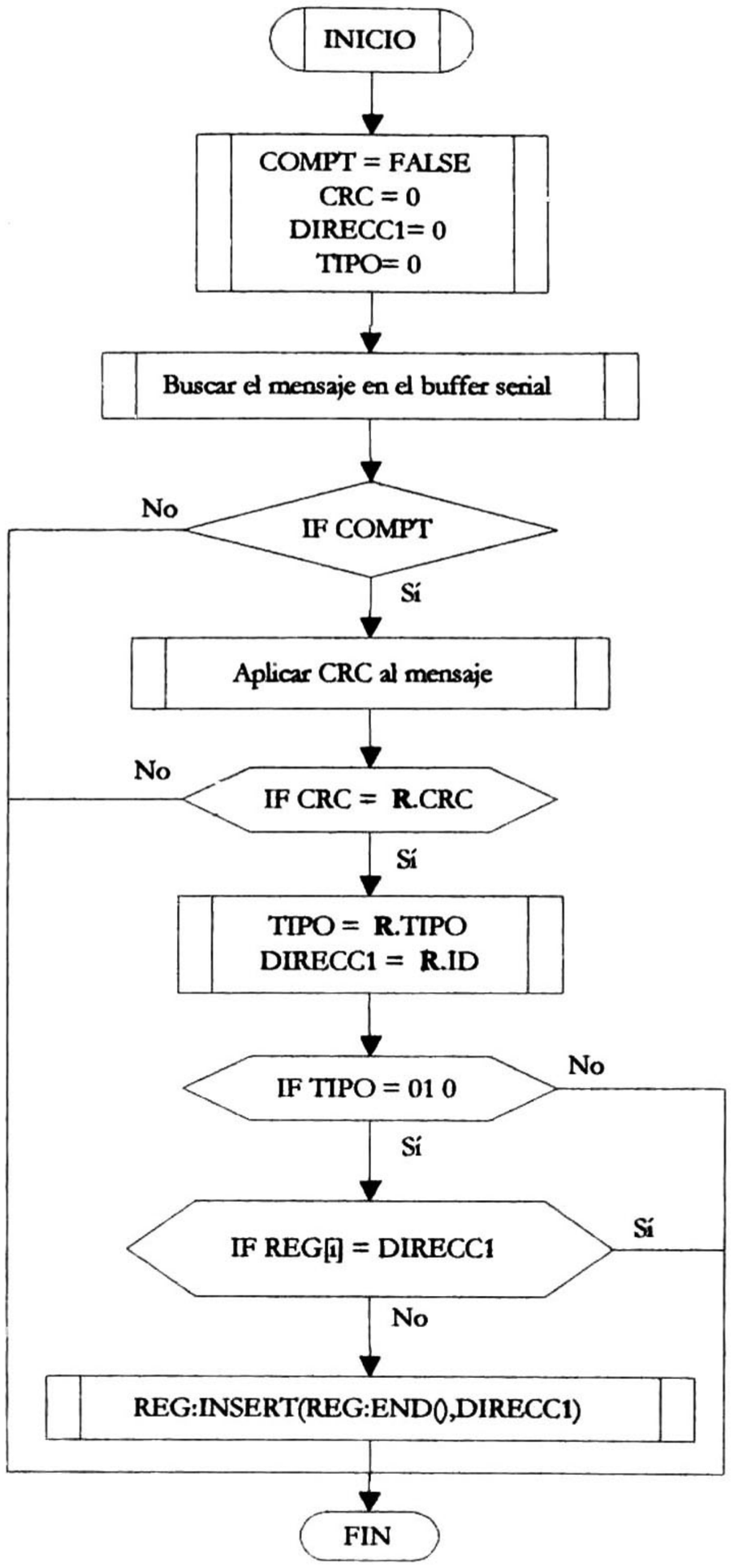


Figura 4.7 Diagrama de flujo para validar mensajes *respuesta de disponibilidad.*

- TIPO = 0, variable definida como *enum*, registra el campo "tipo" del mensaje recibido.

2) **Se busca el mensaje en el buffer de entrada serial,** ver sección 4.3.1.2. Retorna la variable R y COMPT.
3) **COMPT es igual a TRUE (IF COMPT):**
   a) Si es igual, se aplica la prueba de redundancia cíclica al mensaje y el resultado se almacena en CRC. Se va al paso (4).
   b) De lo contrario, se termina el proceso y se retorna al punto donde la función se ejecutó.
4) **CRC es igual a R.*CRC* (IF CRC = R.CRC):**
   a) Si son iguales, TIPO = R. TIPO y DIRECC1 = R. ID, continuando con el paso (5).
   b) En caso contrario, se finaliza la función.
5) **TIPO es igual a 0 1 0 (IF TIPO = 0 1 0);**
   a) Si es igual, se busca el ID del esclavo en la lista del maestro por medio de un ciclo FOR, para evitar duplicidad. Por lo que si REG[i] es igual a DIRECC1 (IF REG[i] = DIRECC1):
      i) Si es igual, se concluye la función.

ii) Si es diferente, se agrega el ID del esclavo a la lista, REG. INSERT(REG.END( ),DIRECC1).

b) En caso contrario, se termina el proceso retornando al punto donde la función fue ejecutada.

4.3.1.2 Búsqueda del Mensaje en el Buffer de Entrada Serial

La función extrae un byte del buffer de recepción serial, lo convierte a bits para identificar la bandera de inicio en el mensaje (pasos 2, 3, 4 y 5). Cuando la encuentra, se extrae el mensaje del buffer y a su vez se aplica el proceso de remoción de cero a cada byte (pasos 6, 8 y 9). El proceso finaliza cuando se encuentra la bandera de fin en el mensaje o el número de bytes extraídos es mayor al número de bytes esperados (pasos 7 al 12). En la Figura 4.8 se muestra el diagrama de flujo de la función. Los pasos que se siguen son:

1) **Generación de las variables:**
   - BYTE_BUFFER=0, variable tipo *BYTE*, almacena un byte extraído del buffer serial.
   - BS[8] = 0, vector tipo *int*, que contiene 8 bits para la remoción del cero.
   - C = 0, var *BYTE*, almacena el byte obtenido del proceso remoción del cero.
   - TAM_MENS = 0, var *int*, representa el tamaño máximo en bytes del mensaje.
   - TAMAÑO = 0, var *int*, contiene el número de bytes extraídos del buffer serial.
2) **TAM_MENS = SIZEOF(mensaje),** define el tamaño del mensaje esperado.
3) **Se revisa si existen bytes en el buffer de entrada serial:**
   a) Si existen, se extrae el byte del buffer (BYTE_BUFFER = Buffer serial). Prosiguiendo al paso (4).
   b) De lo contrario, se finaliza el proceso para retornar al punto donde la función fue ejecutada.
4) **BS[8] = BYTE_BUFFER**, se procede a convertir un byte en 8 bits con el algoritmo de conversión decimal a binario.
5) **BS[8] es igual a 0 1 1 1 1 1 1 0 (IF BS[8] = 0 1 1 1 1 1 1 0):**
   a) Si es igual, continuamos con el paso (6).
   b) En caso contrario se regresa al paso (3).
6) **Se requiere otro byte del buffer:**
   a) Sí se requiere, se revisa si existen bytes en el buffer de entrada serial:
      i) Si existen, se extrae el byte del buffer (BYTE_BUFFER = Buffer serial). Además, BS[8] = BYTE_BUFFER, se procede a convertir un byte en 8 bits.
      ii) En caso contrario, se finaliza función.
   b) De lo contrario se prosigue con el paso (7).
7) **BS[8] es igual a 0 1 1 1 1 1 1 0 (IF BS[8] = 0 1 1 1 1 1 1 0):**
   a) Sí se encontró la bandera, se finaliza el proceso.
   b) De lo contrario se va al paso (8).
8) **BS[8] contiene 1 1 1 1 1 0 (IF BS[8] = ... 1 1 1 1 1 0...):**
   a) Sí la respuesta es si, se elimina el bit en cero.
   b) En caso contrario se prosigue con el paso (9).
9) **C = BS[8]**, se procede a convertir los 8 bits en un byte con el algoritmo de conversión binario a decimal.
10) **R[i] = C,** almacena un byte revisado.
11) **TAMAÑO + +,** incrementa el número de bytes almacenados.
12) **TAMAÑO es mayor que TAM_MENS (IF TAMAÑO > TAM_MENS):**
   a) Si es mayor, COMPT = TURE y se concluye el proceso búsqueda del mensaje.
   b) En caso contrario, se regresa al paso (6).

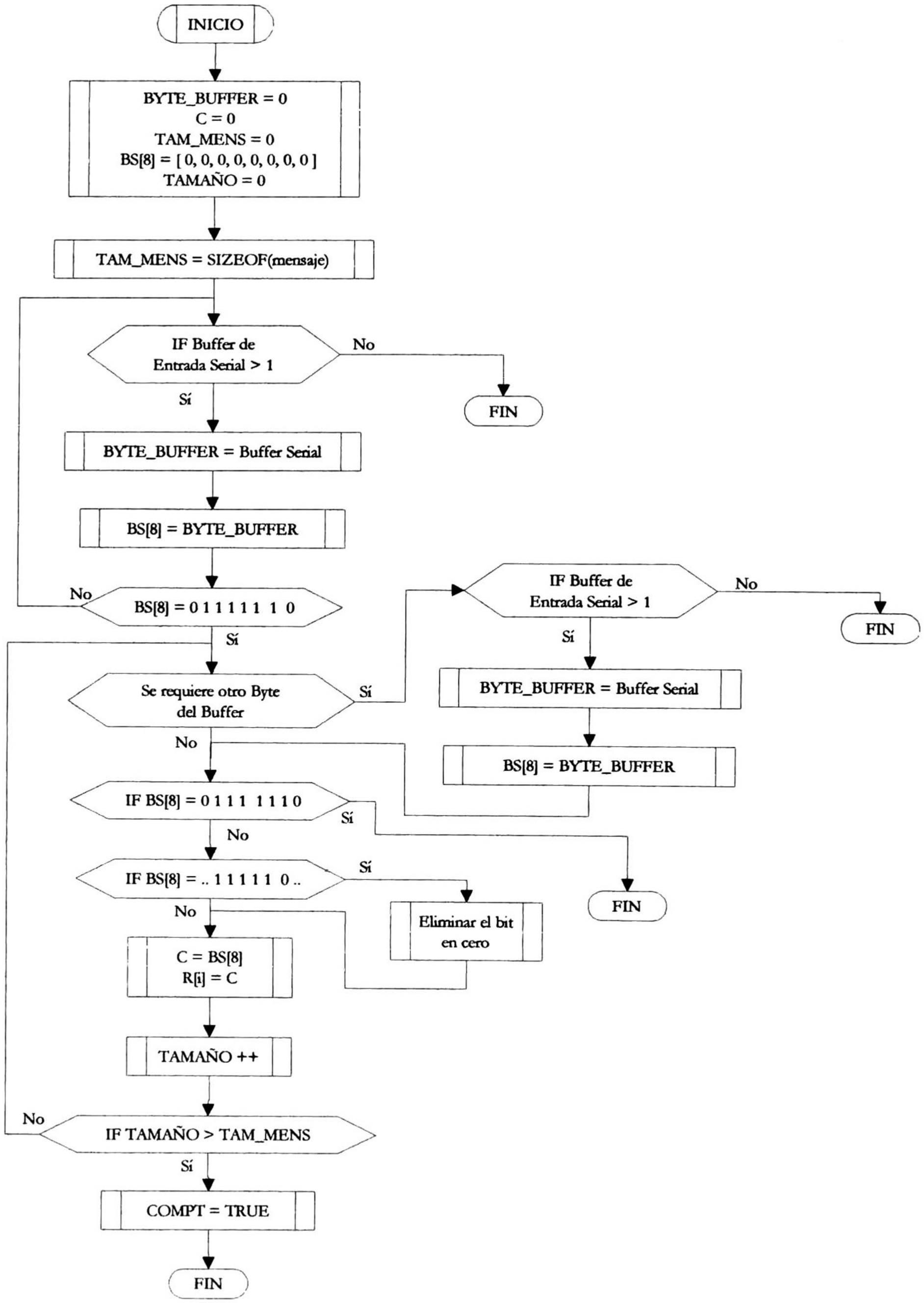


Figura 4.8 Diagrama de flujo para buscar mensajes en el buffer de entrada serial.

4.3.1.3 Validación del Mensaje *Respuesta a Petición*

La función obtiene el mensaje del buffer de recepción serial (paso 2). Cuando se tiene el mensaje completo se verifica que no contenga errores y se identifica el tipo (paso 3 y 4). Si es del tipo *respuesta de petición*, sólo el primer mensaje se revisa para saber si existe una recepción inconclusa con el

esclavo (paso 5). Después se revisa la secuencia y se almacena la carga útil en la memoria (paso 6). Si el mensaje ya había sido procesado se transmite uno del tipo acuse de recibo (paso 7). En la Figura 4.9 se observa el diagrama de flujo de la función.

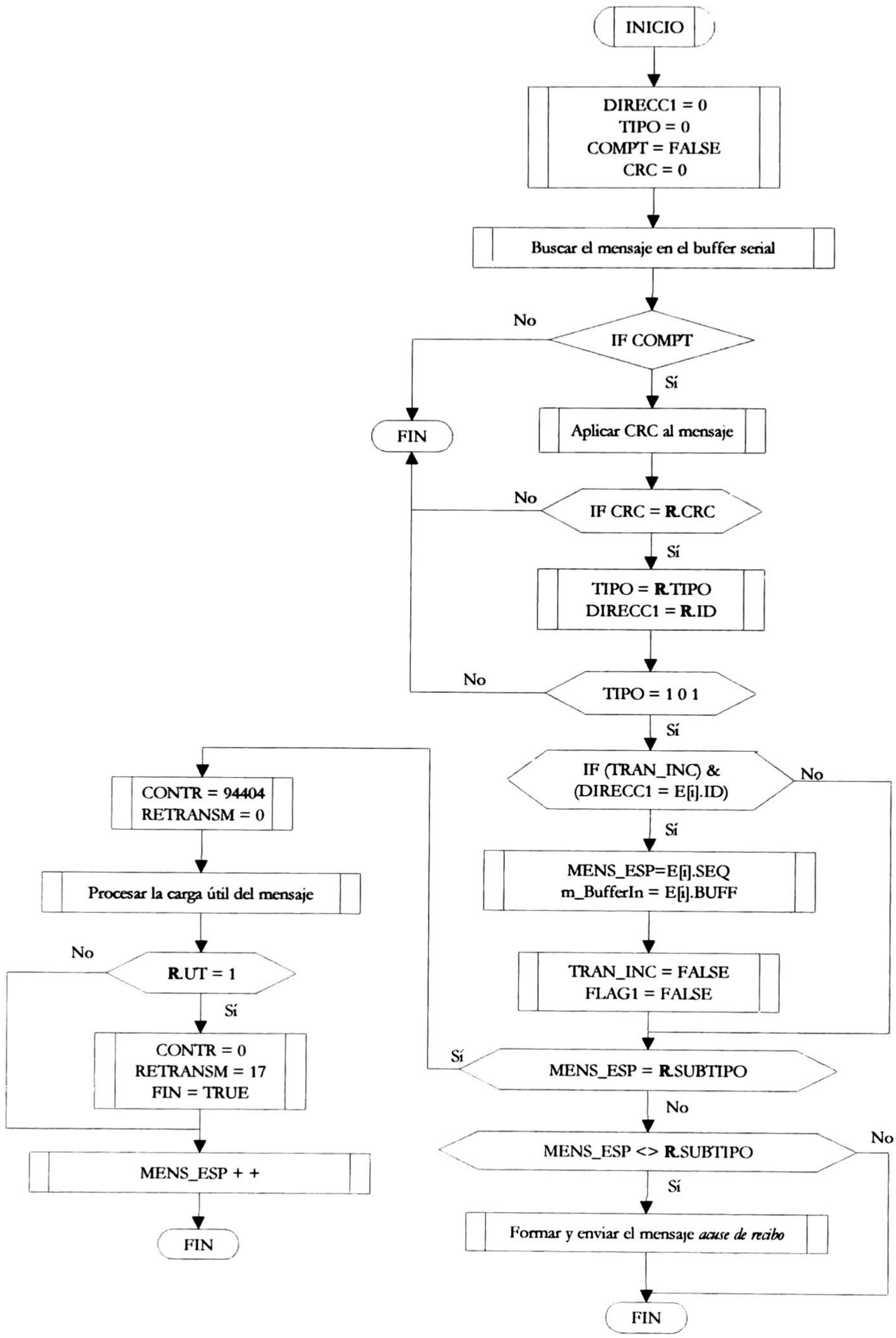


Figura 4.9 Diagrama de flujo para validar el mensaje *respuesta a petición*.

La secuencia que se realiza es la siguiente:

1) **Inicialización de las variables:**
   - COMPT = FALSE, el mensaje no está completo (TRUE, mensaje completo). Variable *bool.*
   - CRC = 0, valor de la prueba de redundancia cíclica aplicada al mensaje. Var *unsigned long.*
   - DIRECC1 = 0, var *unsigned long*, almacena el identificador del esclavo.
   - TIPO = 0, variable definida como *enum*, contiene el campo "tipo" de los mensaje recibido.
2) **Se busca el mensaje en el buffer de entrada serial,** ver sección 4.3.1.2. Retorna la variable R y COMPT.
3) **COMPT es igual a TRUE (IF COMPT):**
   a) Si es igual, se aplica la prueba de redundancia cíclica al mensaje y el resultado se almacena en CRC. Se procede al paso (4).
   b) De lo contrario, se termina el proceso y se retorna al punto donde dicha función se ejecutó.
4) **CRC es igual a R.CRC (IF CRC = R.CRC):**
   a) Si son iguales, DIRECC1 = R. ID y TIPO = R.TIPO, continuando con el paso (5).
   b) En caso contrario, se finaliza la función.
5) **TIPO es igual a 1 0 1 (IF TIPO = 0 1 0):**
   a) Si es igual, se revisa E[i].ID con un ciclo "for" para verificar si existe una transmisión inconclusa con el esclavo. TRAN_INC es igual a TRUE y DIRECC1 es igual a E[i]. ID(IF TRAN_INC & DIRECC1 = E[i].ID):
      i) Si se cumple la condición, MENS_ESP = E[i].SEQ y m_BufferIn = E[i].BUFF. Además, TRAN_INCO = FALSE y FLAG1 = FALSE.
      i) De lo contrario se continua con el paso (6).
   b) En caso contrario se finaliza la función.
6) **MENS_ESP es igual a R.SUBTIPO (IF MENS_ESP = R.SUBTIPO):**
   a) Si es igual, entonces:
      i) CONTR = 94404 y RESTRANSM = 0.
      ii) Se procesa la *carga útil* del mensaje, ver sección 4.3.1.4.
      iii) R.UT es igual a 1(representa el campo *último mensaje* de la trama recibida):
         (1) Si es igual, FIN = TURE, CONTR = 0 y RETRANSM = 17.
         (2) En caso contrario, se va al paso (6.b.iv).
      iv) MENS_ESP + +.
      v) Se finaliza la función.
   b) En el caso contrario, se va al paso (7).
7) **MENS_ESP es diferente de R.SUBTIPO (IF FRAME_EXPECTED <> R.SUBTIPO):**
   a) Si es diferente, se forma el mensaje *acuse de recibo* y se envía al buffer de salida serial, ver sección 4.3.1.5.
   b) En caso contrario, se finaliza la función.

4.3.1.4 Separación de la Carga Útil del Mensaje *Respuesta a Petición*

La función verifica que no existan errores en la carga útil, después transmite un mensaje acuse de recibo (paso 2 y 3). La información de la carga útil se almacena en la memoria del maestro y se revisa si el ultimo bloque de información para generar el archivo (paso 4 y 5). En la Figura 4.10 se muestra el diagrama de flujo de la función.

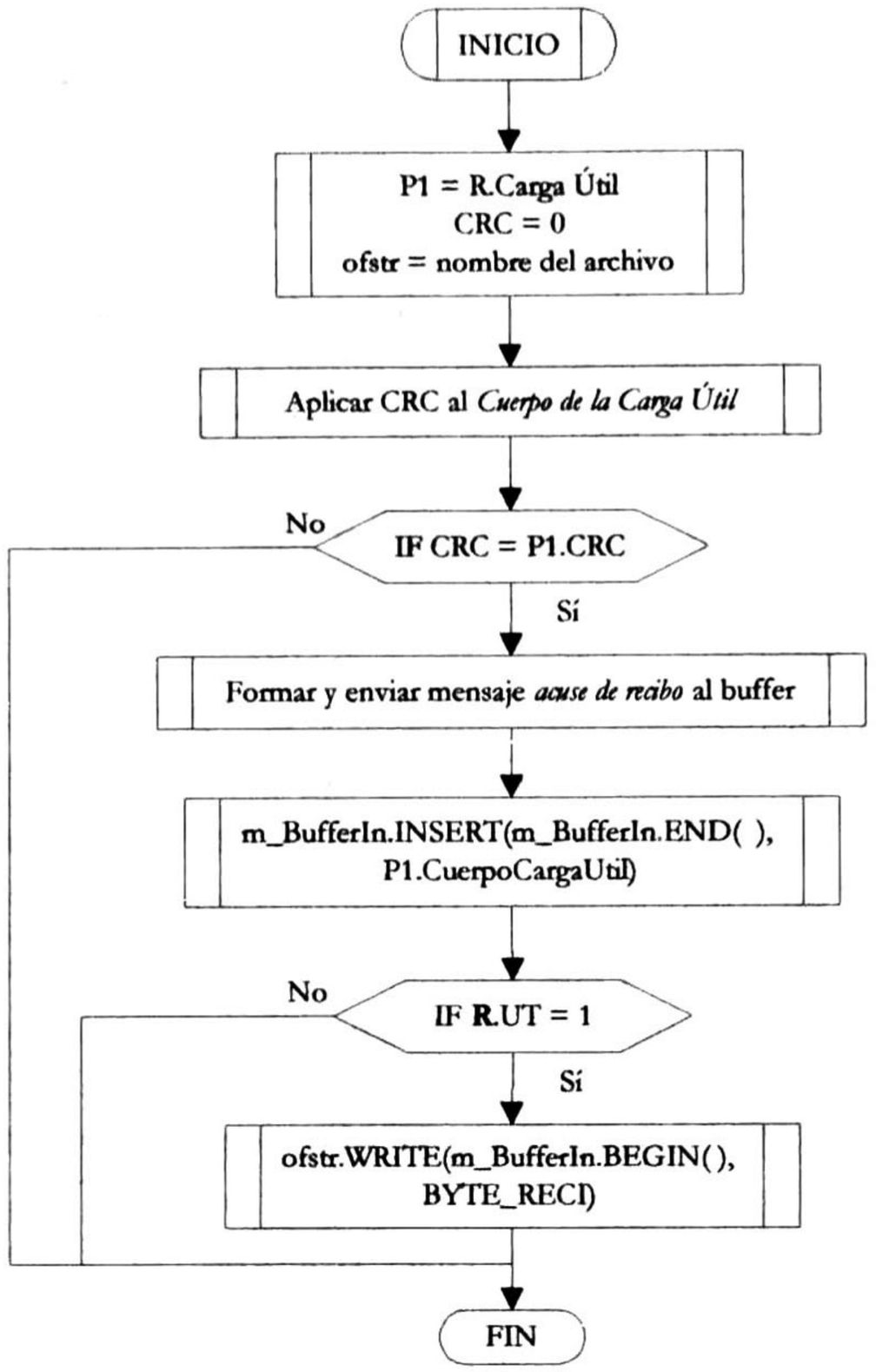


Figura 4.10 Diagrama de flujo para separar la *carga útil* del mensaje.

Los pasos que se siguen para realizar su operación son:

1) **Generación de la variable:**
   - P1=R.CARGA_ÚTIL, variable tipo *struct* (*unsigned char[128]*), almacena la carga útil del mensaje *respuesta a petición.*
   - CRC = 0, valor de la prueba de redundancia cíclica aplicada al cuerpo de la carga útil. Variable tipo *unsigned long.*
   - ofstr = nombre del archivo. var *AnsiString.*
2) **Se aplica la prueba de redundancia cíclica al cuerpo de la carga útil y el resultado se almacena en CRC.**
3) **CRC es igual a P1.CRC (IF CRC = P1.CRC):**
   c) Si es igual, se forma el mensaje *acuse de recibo* y se envía al buffer de salida serial, ver sección 4.3.1.5.
   d) En caso contrario, se finaliza el proceso y se retorna al punto donde la función se ejecutó.
4) **m_BufferIn.INSERT(m_BufferIn.END(),P1.Cuerpo_carga_útil),** se almacena el cuerpo de la *carga útil* en la memoria del maestro en forma secuencial.
5) **R.UT es igual a 1 (IF R.UT):**
   a) Si es igual indica que el esclavo envió el último mensaje. Por lo que toda la información almacenada en memoria se convierte en un archivo con la instrucción: ofstr.WRITE (m_BufferIn.BEGIN(),BYTE_RECI). Este bloque fue introducido para llevar acabo las pruebas de campo del sistema

b) De lo contrario continua con el proceso.

6) Se finaliza el proceso y se retorna al punto donde la función se ejecutó.

4.3.1.5 Enviar los Mensajes al Buffer Serial

La función agrega la bandera de inicio al mensaje (paso 3). Obtiene el siguiente byte del mensaje y realiza el proceso de inserción de cero (pasos del 4 al 7). El bloque se repite hasta que se revise el mensaje completo (pasos 8 y 9). Al finalizar, se agrega la bandera de fin de mensaje y se envía la información al buffer de transmisión serial (paso del 10 al 13). En la Figura 4.11 se muestra el diagrama de flujo de la función. Los pasos que se siguen son:

1) Inicialización de las variables:
- TAM_MENS = 0, variable *int*, representa el tamaño en bytes del mensaje.
- AUXILIAR = 0, var *std::vector <BYTE>*, almacena el mensaje a enviar.
- B = 0, var *BYTE*, almacena un byte extraído del mensaje.
- BS[8] = 0, vector tipo *int* que contiene 8 bits para la inserción del cero.
- C = 0, var *BYTE*, almacena el byte obtenido de la inserción del cero.
- TAMAÑO = 0, var *int*, contiene el número de bytes extraídos del mensaje.

2) TAM_MENS = SIZEOF(mensaje), define el tamaño del mensaje esperado.
3) AUXILIAR.INSERT(AUXILIAR.END(), 7EH), se agrega la bandera de inicio.
4) B = R[i], se extrae un byte del mensaje.
5) TAMAÑO ++, incrementa el número de bytes extraídos.
6) BS[8] = B, se procede a convertir un byte en 8 bits con el algoritmo de conversión decimal a binario.
7) BS[8] contiene 11111 (IF BS[8] = ... 11111 ...):
a) Si la respuesta es si, se inserta un cero después del quinto bit.
b) De lo contrario proseguimos con el siguiente paso.
8) Se analizó por completo BS[8].
a) Si la respuesta es si, se realiza lo siguiente:
i) C = BS[8], se procede a convertir los 8 bits a un byte con el algoritmo de conversión decimal a binario.
ii) AUXILIAR.INSERT(AUXILIAR.END(), C).
b) En caso contrario se avanza al paso (7).
9) TAMAÑO es mayor que TAM_MENS (IF TAMAÑO > TAM_MENS):
a) Si es mayor, se agrega la bandera de final a BS[8].
b) En caso contrario, se regresa al paso (3).
10) Se completa un byte en BS[8].
a) Si se completa se continua con el proceso.
b) De lo contrario, se insertan los ceros necesarios en BS[8].
11) C = BS[8], se procede a convertir los 8 bits en un bytes con el algoritmo de conversión binario a decimal.
12) AUXILIAR.INSERT(AUXILIAR.END(), C), se inserta el byte.
13) WRITEBUFFER(AUXILIAR.BEGIN(), AUXILIAR.SIZE()), se envía el mensaje al serial.
14) Se finaliza el proceso regresando al punto donde la función se ejecutó.

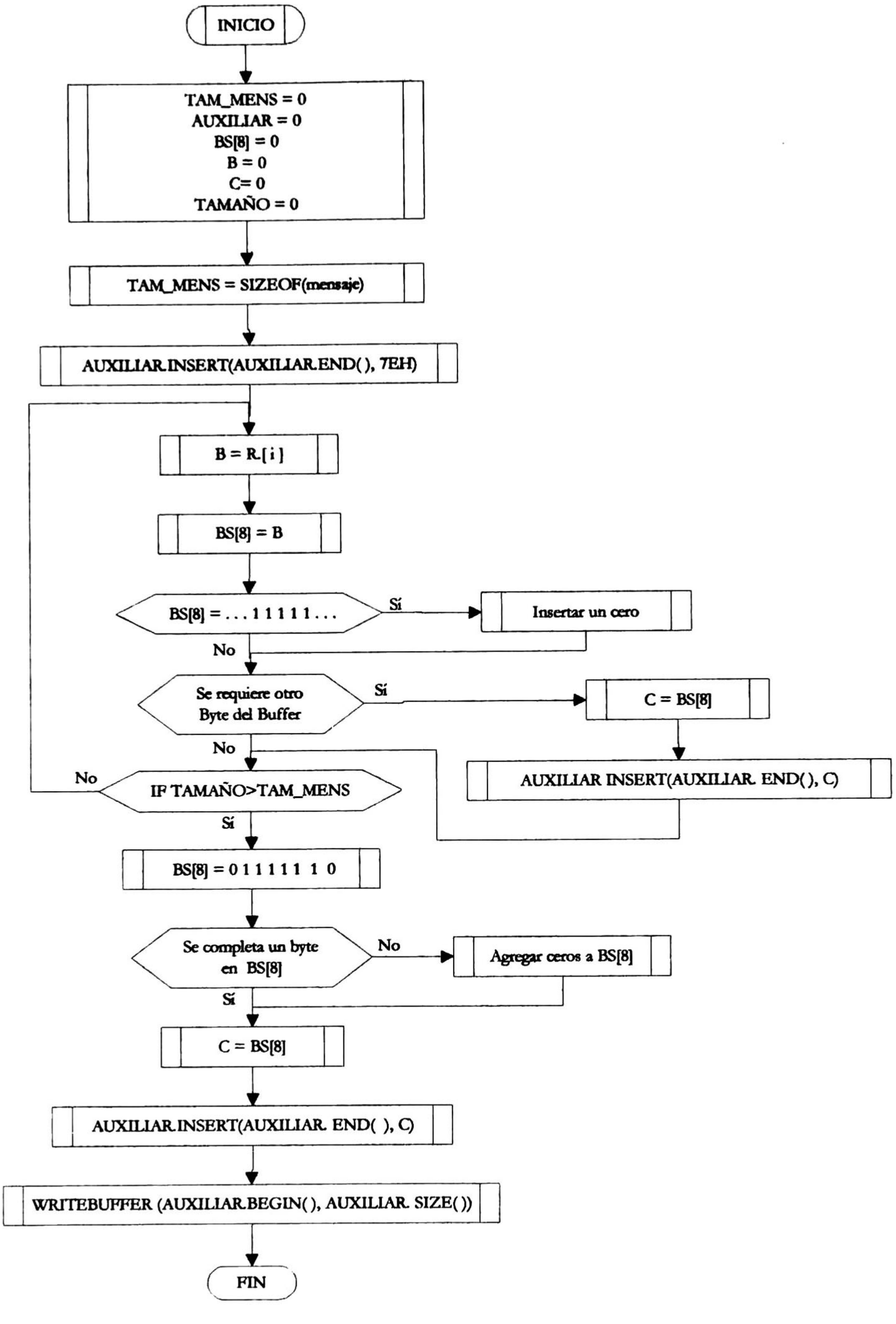


Figura 4.11 Diagrama de flujo para enviar los mensajes al buffer de salida serial.

4.3.2 PROGRAMA OPERATIVO DEL ESCLAVO

El programa que define la operación del esclavo se divide principalmente en dos bloques: RE y RPI.

En el bloque RE, se revisa el buffer de recepción serial para identificar el mensaje correspondiente *búsqueda de esclavos disponibles* o *petición de la información* (paso 2). El esclavo permanece en este bloque hasta que reciba el mensaje *petición de la información* (paso 3).

El bloque RPI extrae un segmento de la información existente en memoria, forma el mensaje respuesta a la petición y entra a un ciclo de espera (pasos 4, 5 y 6). Dentro del ciclo se revisa el buffer de recepción serial y cuando se detecta un posible mensaje se identifica (paso 7). Si finaliza el ciclo de espera sin obtener respuesta del maestro se realiza una retransmisión del mensaje (pasos 8 y 4). Cuando se excede el número de retransmisiones el proceso concluye (paso 9). En la Figura 4.12 se muestra el diagrama de flujo del programa. Los pasos necesarios para realizar su procesamiento son:

1) Inicialización de las variables:

- IDENT = 5, número identificador del esclavo. Variable tipo *int.*
- R[i] = 0, variable definida como *struct* (*unsigned long, unsigned int, unsigned short, bool, std::vector <BYTE>*), contiene el mensaje.
- TRANSM = FALSE, var *bool*, permite al esclavo acceder al bloque de transmisión de la información.
- FINAL = FALSE, variable tipo *bool*, indica que se realizó una transmisión completa.
- CONT1 = 0, var *unsigned int*, registra el número de retransmisiones.
- CLOCK = 0, var *unsigned int*, establece el tiempo de espera para que el maestro responda.
- NBUFFERED = 0, variable tipo *unsigned short*, indica cuando se puede transmitir un mensaje *respuesta a petición.*
- CANT_INF = 0, var *unsigned short*, almacena la naturaleza de la información requerida por el maestro.
- SEQ_ESP = 0, var *unsigned short*, registra la numeración secuencial (000 a 111) esperada en el mensaje *acuse de recibo.*
- CAN_ENV = 0, var *unsigned int*, cantidad en bytes de información transmitida.
- m_BufferOut[1], var *std::vector<BYTE>*, archivo con la cantidad de información del recorrido. Esta variable y las siguientes son introducidas para llevar acabo las pruebas de campo de los sistemas.
- m_BufferOutLength[1], var *unsigned int*, define el tamaño en bytes del archivo.
- m_BufferOut[2], var *std::vector<BYTE>*, archivo con la cantidad de información del día.
- m_BufferOutLength[2], var *unsigned int*, define el tamaño en bytes del archivo.
- m_BufferOut[3], var *std::vector<BYTE>*, archivo con la cantidad de información de dos semanas.
- m_BufferOutLength[3], var *unsigned int*, define el tamaño en bytes del archivo.

2) Se revisa el buffer de entrada serial para verificar si el maestro busca al esclavo:

a) Si el buffer contiene más de 6 bytes, se ejecuta una función para validar el mensaje *búsqueda de esclavos disponibles*, ver sección 4.3.2.1.
b) En caso contrario se prosigue al paso (3).

3) TRANSM es igual a TRUE (IF TRANSM):

a) Si es igual indica que el esclavo fue detectado por el maestro, se continua con el paso (4).
b) De lo contrario se regresa al paso (2).

4) NBUFFERED es igual a cero (IF !NBUFFERED):

a) Si es igual, entonces CONT1 = 17, CLOCK = 15000 y NBUFFERED =1.
b) En caso contrario, se ejecuta la función encargada de las retransmisiones, ver sección 4.3.2.4. Después se prosigue con el paso (7).

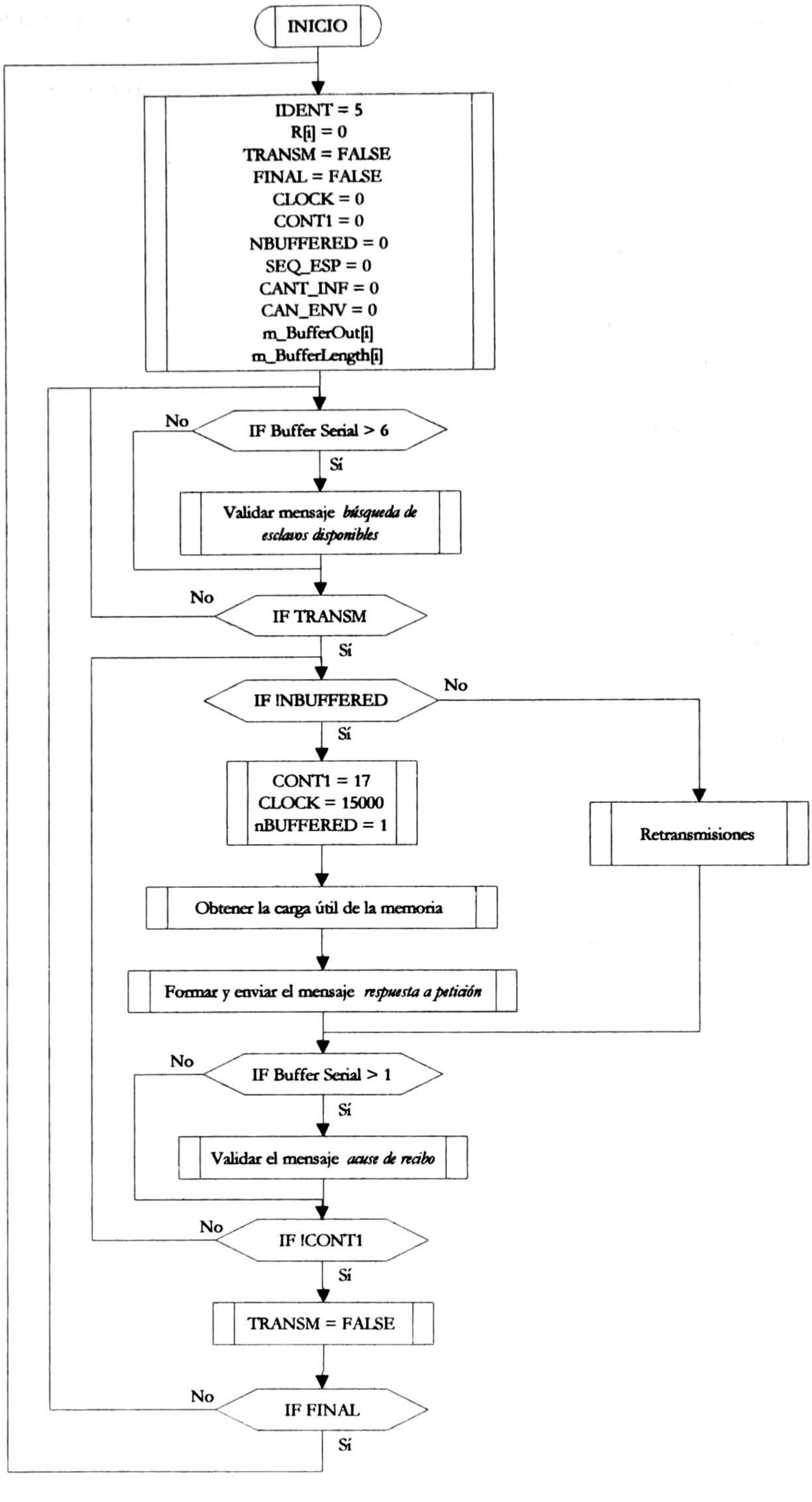


Figura 4.12 Diagrama de flujo del programa existente en el esclavo.

5) **Se extrae de la memoria del esclavo un bloque de información y se procesa en la *carga útil* del mensaje,** ver sección 4.3.2.3.

6) **Se forma el mensaje *respuesta a petición* y se envía al buffer del puerto serial,** ver sección 4.3.1.5.
7) **Se revisa el buffer de la entrada serial para verificar si el maestro recibió el mensaje enviado:**
   a) Si el buffer contiene más de un byte, se ejecuta una función para validar el mensaje *acuse de recibo* (sección 4.3.2.2).
   b) De lo contrario se pasa al siguiente paso.
8) **CONT1 es igual a 0 (IF !CONT1):**
   a) Si es igual detiene transmisión por exceso de retransmisiones, TRANSM = FALSE y continua con el paso (9).
   b) En caso contrario, se posiciona en el paso (4).
9) **FINAL es igual a TRUE (IF FINAL):**
   a) Si es igual la transmisión finaliza y va al paso (1).
   b) De lo contrario regresa al paso (2)

Con esto se concluye la secuencia de pasos que realiza el programa del esclavo para establecer una comunicación inalámbrica. El apéndice E contiene el código del programa operativo del maestro descrito en lenguaje de alto nivel Builder C++.

4.3.2.1 Validar Mensajes: *Búsqueda de Esclavos Disponibles* y *Petición de la Información*

La función obtiene los mensajes existentes en el buffer de recepción serial (paso 2). Cuando se tiene el mensaje completo se verifica que no contenga errores y se identifica el tipo (pasos 3 y 4). Si es del tipo *búsqueda de esclavos disponibles* y le corresponde al esclavo, se forma el mensaje *respuesta de disponibilidad* y se envía al buffer. Pero si es del tipo petición de la información se selecciona la cantidad de información a transmitir (pasos 5, 6 y 7). En la Figura 4.13 se observa el diagrama de flujo de la función. Los pasos que se siguen para su desarrollo son:

1) **Generación de las variables:**
   - CRC = 0, valor de la prueba de redundancia cíclica aplicada al mensaje. Variable tipo *unsigned long.*
   - COMPT = FALSE, el mensaje no está completo (TRUE, mensaje completo). Var *bool.*
   - DIRECC1 = 0, var *unsigned long,* almacena el identificador del esclavo.
   - TIPO = 0, variable definida como *enum*, contiene el campo "tipo" del mensaje recibido.
2) **Se busca el mensaje en el buffer de entrada serial,** ver sección 4.3.1.2. Retorna la variable R y COMPT.
3) **COMPT es igual a TRUE (IF COMPT):**
   a) Si es igual, se aplica la prueba de redundancia cíclica al mensaje y el resultado se almacena en CRC. Se procede al paso (4).
   b) De lo contrario, se termina el proceso y se retorna al punto donde dicha función se ejecutó.
4) **CRC es igual a R.CRC (IF CRC = R.CRC):**
   a) Si son iguales, DIRECC1 = R. ID y TIPO = R.TIPO continuando con el paso (5).
   b) En caso contrario, se finaliza la función.
5) **TIPO es igual a 0 0 1 y DIRECC1 es igual a IDENT (IF TIPO = 001 & DIRECC1 = IDENT):**
   a) Si son iguales, se forma el mensaje *respuesta de disponibilidad* y se envía al buffer del puerto serial (sección 4.3.1.5).
   b) De lo contrario, se proseguirá con el paso (6).

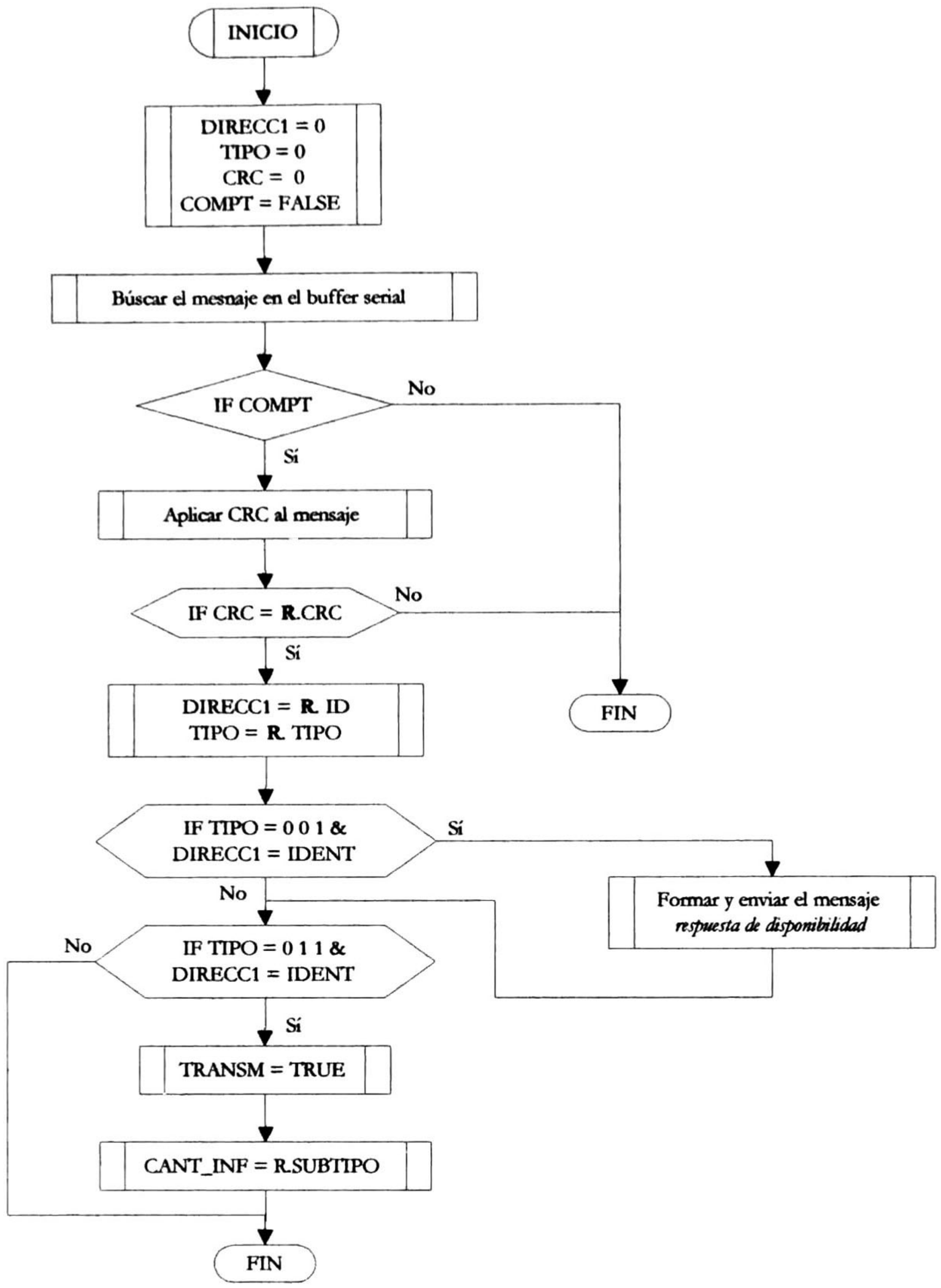


Figura 4.13 Diagrama de flujo para validar mensajes: *búsqueda de esclavos disponibles* y *petición de la información.*

6) **TIPO es igual a 0 1 1 y DIRECC1 es igual a IDENT (IF TIPO = 0 1 1 & DIRECC1 = IDENT):**
   a) Si son iguales, TRANSM = TRUE.
   b) En caso contrario se prosigue al paso (8).
7) **CANT_INF = R. SUBTIPO,** se define la cantidad de información a transmitir.
8) **Se finaliza el proceso.**

4.3.2.2 Validar Mensaje *Acuse de Recibo*

La función extrae los mensajes existentes en el buffer de recepción serial (paso 2). Cuando tiene el mensaje completo verifica que no contenga errores e identifica el tipo (paso 3 y 4). Si es del tipo acuse de recibo y corresponde a la secuencia esperada, permite al esclavo enviar el siguiente

segmento de información (paso 5 y 6). También se revisa si el mensaje corresponde al último para finalizar la transmisión (paso 7). En la Figura 4.14 se observa el diagrama de flujo de la función. Los pasos que se siguen son:

1) Generación de las variables:

- CRC = 0, valor de la prueba de redundancia cíclica aplicada al mensaje. Variable tipo *unsigned long.*
- COMPT = FALSE, el mensaje no está completo (TRUE, mensaje completo). Var *bool.*
- DIRECC1 = 0, var *unsigned long,* almacena el identificador del esclavo.
- TIPO = 0, variable definida como *enum*, contiene el campo "tipo" del mensaje.

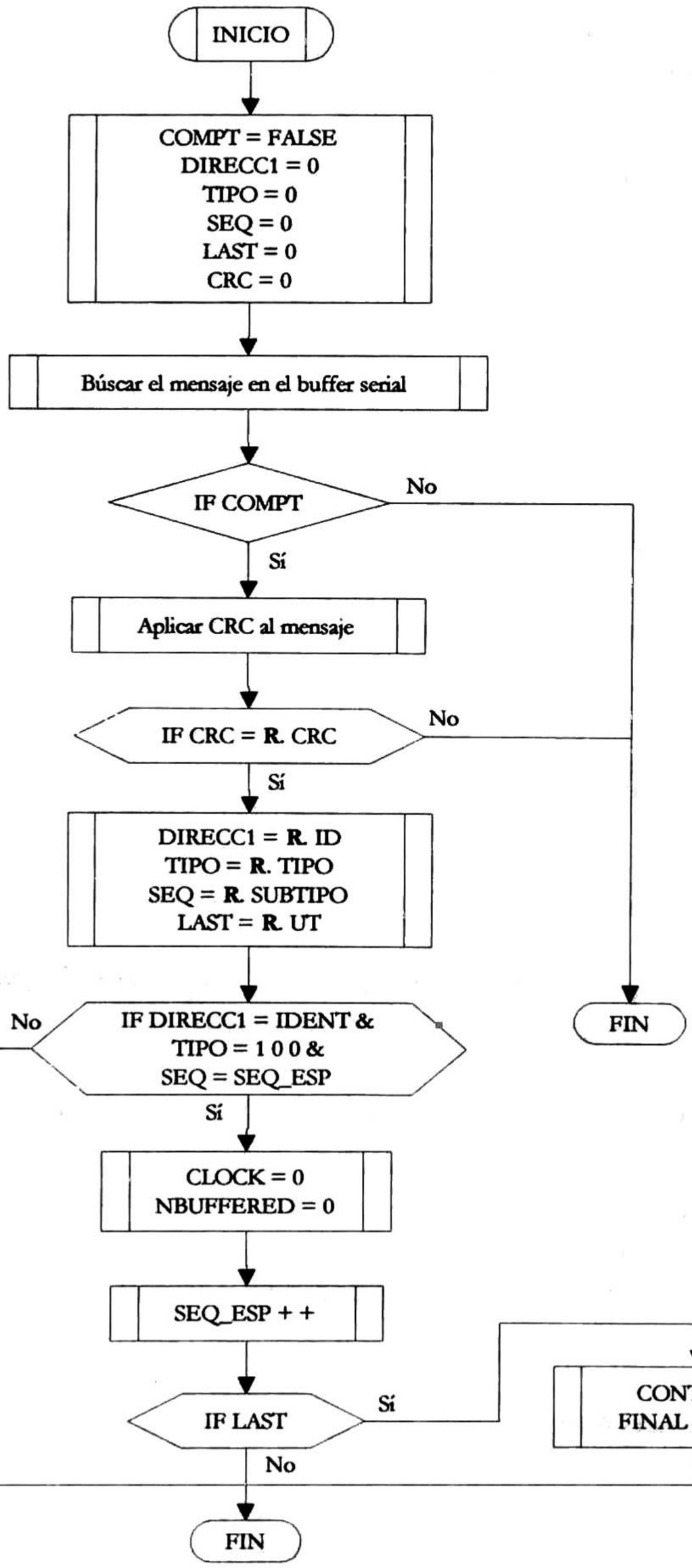


Figura 4.14 Diagrama de flujo para validar el mensaje *acuse de recibo.*

- SEQ = 0, var *unsigned short*, almacena una numeración secuencial del mensaje recibido.
- LAST = 0, var *BYTE*, avisa cuando se recibe el ultimo mensaje de la transmisión.

2) **Se busca el mensaje en el buffer de entrada serial** (sección 4.3.1.2). Retorna la variable R y COMPT.
3) **COMPT es igual a TRUE (IF COMPT):**
   a) Si es igual, se aplica la prueba de redundancia cíclica al mensaje y el resultado se almacena en CRC. Se procede al paso (4).
   b) De lo contrario, se termina el proceso y se retorna al punto donde dicha función se ejecutó.
4) **CRC es igual a R.CRC (IF CRC = R.CRC):**
   a) Si son iguales, DIRECC1 = R. ID, TIPO = R.TIPO, SEQ = R.SUBTIPO y LAST = R.UT continuando con el paso (5).
   b) En caso contrario, se finaliza la función.
5) **DIRECC1 es igual a IDENT & TIPO es igual a 1 0 0 & SEQ es igual a SEQ_ESP (IF DIRECC1 = IDENT & TIPO = 1 0 0 & SEQ = SEQ_ESP):**
   a) Si son iguales, CLOCK = 0 y NBUFFERED = 0.
   b) En caso contrario, se finaliza la función
6) **SEQ_ESP + +,** se incrementa el número de la secuencia esperada.
7) **LAST es igual a 1 (IF LAST):**
   a) Si es igual indica que el maestro recibió el último mensaje, CONT1 = 0 y FINAL = TRUE.
   b) En caso contrario se finaliza la función

4.3.2.3 Extracción de la Carga Útil en la Memoria

La función extrae un segmento (126B) de la información contenida en la memoria (paso 2). Se aplica el CRC al segmento y el resultado se almacena en la carga útil junto con el segmento (paso 3, 4 y 5). Se verifica si es el último segmento de información (paso 6). En la Figura 4.15 se muestra el diagrama de flujo de la función.

Los pasos que se siguen para realizar su procesamiento son:

1) **Inicialización las variables:**
   - CRC = 0, valor de la prueba de redundancia cíclica aplicada al cuerpo de la carga útil. Variable tipo *unsigned long.*
   - P1[i] = 0, variable tipo *struct* (*unsigned char[128]*), almacena un bloque de información existente en memoria.
2) **P1.Cuerpo_Carga_Utili = m_BufferOut[CANT_INF].AT(CAN_ENV),** se extrae un bloque de 126 B de la memoria del esclavo.
3) **Se aplica la prueba de redundancia cíclica al cuerpo de la carga útil y el resultado se almacena en CRC.**
4) **P1.CRC = CRC.**
5) **R.Carga_Util = P1.**
6) **CAN_ENV es igual a m_BufferOutLength[CANT_INF] (IF CAN_ENV = m_Buffer OutLength[CANT_INF]):**
   a) Si el resultado es si, R.UT = 1.
   b) De lo contrario se va al siguiente paso.
7) **Se finaliza el proceso y se retorna al punto donde la función se ejecutó.**

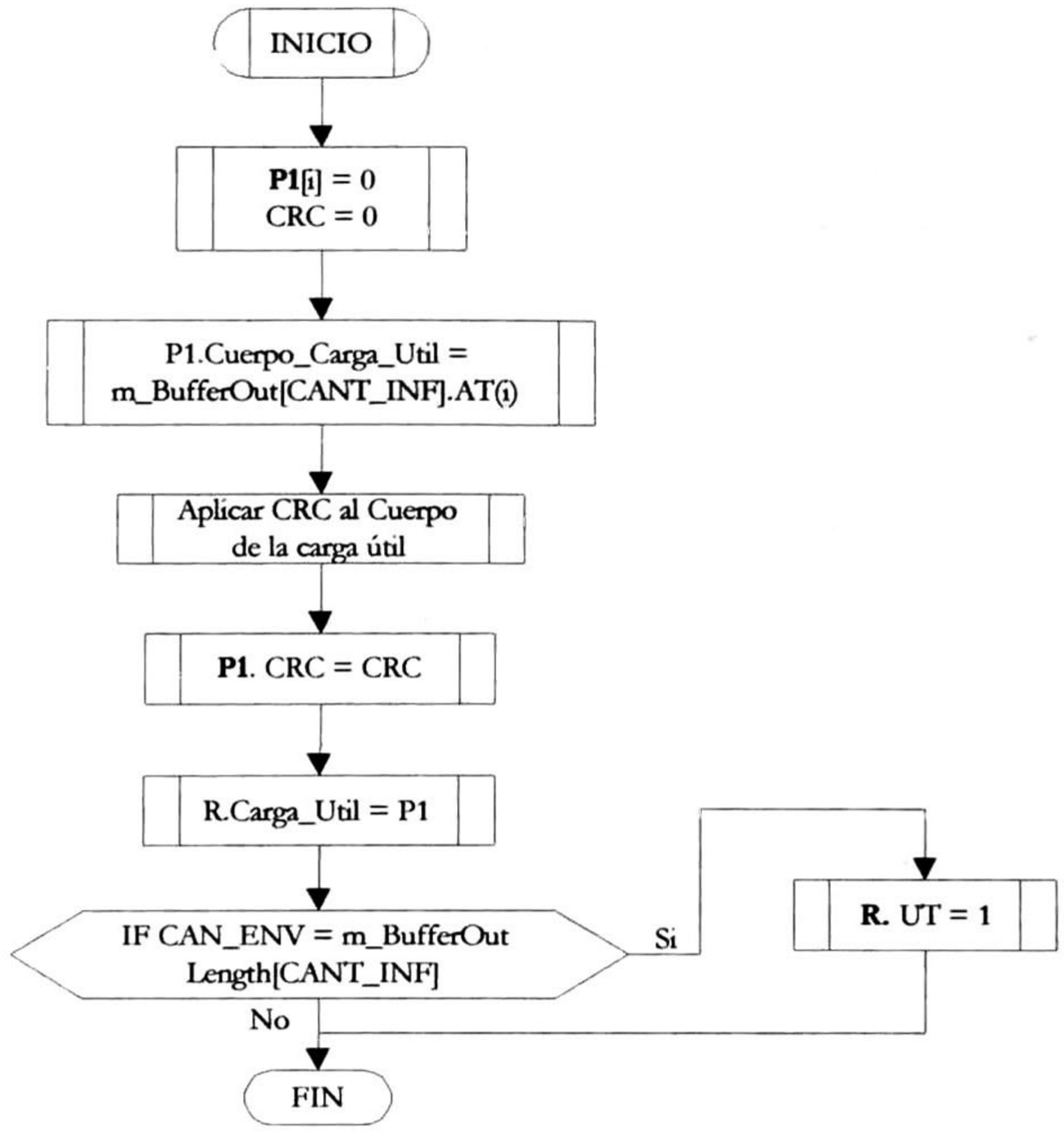


Figura 4.15 Diagrama de flujo de la función encargada de obtener la *carga útil.*

4.3.2.4 Retransmisiones

La función disminuye el tiempo de espera para que el maestro envíe el mensaje *acuse de recibo*. Cuando el tiempo finaliza, el esclavo retransmite el mensaje *respuesta a petición* (paso 1). Los paso que se realizan para su operación son:

1) **CLOCK es mayor que 0 (IF CLOCK > 0):**
   a) Si es mayor, CLOCK – –
   b) Si es cero, se retransmite el mensaje *respuesta a petición* y CONT1 – –.
2) **Se finaliza el proceso y se retorna al punto donde la función se ejecutó.**

4.4 CARACTERÍSTICAS DEL PROGRAMA

Las características de los archivos se muestran en la Tabla 4.1:

Grupo	Archivo	MAESTRO			ESCLAVO		
		Capacidad en Bytes	Núm. de líneas	Núm. de funciones	Capacidad en Bytes	Núm. de líneas	Núm. de funciones
Interfaz Usuario	Main.cpp	5,943	131	11	9,394	263	17
	Main.h	2,579	74	-	3,041	88	-
Programa Operativo	Protocolo.cpp	39,229	1,106	21	37,893	1,120	30
	Protocolo.h	5,721	145	-	5,393	123	-
Archivos para el puerto serial	SerialPort.cpp	3,1654	54	7	3,358	54	7
	SerialProt.h	963	25	-	991	26	-
	Commsetting.cpp	1,728	37	5	1,497	36	5
	Commsetting.h	1,340	36	-	1,340	31	-
	Comm.cpp	21,059	311	29	22,209	337	30
	Comm.h	4,436	92	-	4,517	97	-
Ejecutable	Term.exe	899,584	-	-	694,272	-	-

Tabla 4.1 Características del programa para el maestro y el esclavo.

CAPÍTULO V

5. RESULTADOS

5.1 INTRODUCCIÓN

Este capítulo presenta los resultados obtenidos de la comunicación inalámbrica bajo el protocolo SyT. Para comprobar su funcionalidad se realizaron varias transmisiones con diferentes variantes, donde se monitorea:

- Las distancias a las cuales las estaciones pueden realizar sus transmisiones, sin que la señal disminuya su potencia por debajo de la potencia mínima de recepción (RH1).
- La detección de los esclavos durante la búsqueda por el maestro (RS28).
- Las retransmisiones realizadas por las estaciones durante los enlaces, al momento de enviar los mensajes *respuesta a petición* y *acuse de recibo* (RS29).
- La exactitud con que los datos son recibidos por el maestro, es decir, que la información que transmiten los esclavos no contenga errores al momento de ser almacenados en el maestro (RS30).

5.2 CARACTERIZACIÓN DEL *TRANSCEIVER*

Para determinar el área de alcance real del *transceiver* MEDV-900HP se realizaron varias mediciones de la potencia del transmisor TXM-900-HP-II a diferentes distancias y sin obstáculos. Y de acuerdo con la hoja de especificaciones del receptor RXM-HP-900-II se puede definir la distancia mínima de enlace o alcance.

El equipo de medición utilizado para medir la potencia es el analizador de espectro HP900. En la siguiente tabla se muestran los datos obtenidos del *transceiver* que se encuentra en el concentrador.

Distancia	Potencia	Distancia	Potencia
4.5 mts	-50.68 dBm	43 mts	-63.03 dBm
7.5 mts	-52.39 dBm	50 mts	-66.70 dBm
15 mts	-56.11 dBm	58 mts	-72.59 dBm
21.5 mts	-57.06 dBm	64 mts	-74.62 dBm
29 mts	-60.19 dBm	78.5 mts	-76.66 dBm
36 mts	-61.52 dBm	86 mts	-79.25 dBm

Tabla 5.1 Medición de la potencia del transmisor TXM-900-HP-II ubicado en el concentrador.

La Tabla 5.2 muestra las mediciones de la potencia del transmisor TXM-900-HP-II que se encuentra en las unidades de cobro, a diferentes distancias y sin obstáculos.

Observando las mediciones de las Tablas 5.1 y 5.2, se puede concluir que:

- El transmisor existente en las unidades de cobro tiene una pérdida de la señal equivalente a 2.5 dBm aproximadamente con respeto al transmisor del concentrador, esto se debe a la existencia del interruptor de AsGa en el transmisor de la unidad de cobro utilizado para conectar y desconectar la antena del sistema (ver apéndice F).
- La distancia máxima de transmisión según las hojas de especificaciones de sensibilidad de la modulación DC del receptor es de 70 metros, con una ruta directa de la señal.

Distancia	Potencia	Distancia	Potencia
4.5 mts	-56.88 dBm	43 mts	-66.09 dBm
7.5 mts	-58.11 dBm	50 mts	-67.37 dBm
15 mts	-59.80 dBm	58 mts	-68.52 dBm
21.5 mts	-60.01 dBm	64 mts	-72.39 dBm
29 mts	-64.33 dBm	78.5 mts	-73.49 dBm
36 mts	-65.81 dBm	86 mts	-82.81 dBm

Tabla 5.2 Medición de la potencia del transmisor TXM-900-HP-II existente en las unidades de cobro.

5.3 COMUNICACIÓN CON BAJOS NIVELES DE INTERFERENCIA

En esta sección se muestran los resultados obtenidos durante la comunicación de un maestro con un esclavo. El termino "bajos niveles de interferencia", se refiere a que la comunicación se realiza a través de cables por el puerto serial para evitar interferencias provocadas por el medio. Todas las pruebas se realiza con el *transceiver* MEDV-900-HP a una velocidad de transmisión de 19,200 bps.

Los parámetros de la transmisión son:

- Duración de la prueba, 1 hr.
- Distancia entre el maestro y el esclavo, 1.50 mts.
- El archivo transmitido contiene la secuencia 01010101 (55 en hexadecimal).
- Duración de la transmisión del archivo, 7.397 segundos.
- El número de muestras[1] realizadas es de 230.

El análisis de los resultados se dividen en dos bloques:

1.- Detección de esclavos: Los datos son obtenidos de las secciones búsqueda de los esclavos y respuesta del esclavo disponible, en el programa del maestro y del esclavo, respectivamente.
2.- Retransmisiones realizadas: Los resultados se obtienen de las secciones respuesta a la petición y recepción y almacenamiento de la información, en el programa del esclavo y del maestro, respectivamente.

Los resultados se muestran en una tabla, donde la primer columna (de izquierda a derecha) indica el nombre del mensaje a analizar, la segunda columna representa el número de veces que se presenta un mensaje del mismo tipo durante la muestra; y la siguiente expresa el porcentaje de la segunda columna en función al número de muestras.

En la Tabla 5.3 se muestran los resultados del 1er bloque. El número máximo de mensajes *búsqueda de esclavos disponibles* enviados por el maestro son de 99 por muestra. En el 100% de las 230 muestras, la esclavo U5 recibió 98 mensajes en cada muestra con el número identificador (#ID) diferente al número del esclavo. Asimismo en cada muestra se recibió un mensaje con el campo #ID igual al número del esclavo. Esto significa que el esclavo recibió sin error todas los mensajes enviados por el maestro durante una hora. Por otra parte, el esclavo envió el 100% de las veces el mensaje *respuesta de disponibilidad* al maestro (ver Tabla 5.3).

El número de veces que el maestro repite la sección búsqueda de esclavos sin ejecutar la sección petición de la información depende si detectó o no los mensajes *respuesta de disponibilidad* de los esclavos. En este muestreo siempre se logro pasar a la sección de petición y recepción de la información después de la primer búsqueda.

[1] Se considera una muestra a la ejecución del programa completo, es decir, que el programa realice la primera y segunda etapa.

Tipo de mensaje	# mensajes por muestra	% del mensaje durante el muestreo
Mensajes recibidos por el esclavo 5		
Búsqueda de esclavos #ID != U5	98	100%
Búsqueda de esclavos #ID = U5	1	100%
Mensajes recibidos por el maestro		
Respuesta de disponibilidad	1	100%

Tabla 5.3 Resultados del primer bloque con bajos niveles de interferencia.

Antes de mostrar los resultados del segundo bloque, primero debemos mencionar algunos puntos. Los mensajes que se muestran en la Tabla 5.4 son del tipo *respuesta a petición* y *acuse de recibo* que han sido almacenados por parte del maestro y del esclavo respectivamente. La diferencia radica en las partes del programa donde son obtenidos o detectados, así tenemos que:

- Mensajes enviados por el maestro (o el esclavo).- El valor se obtiene del maestro (esclavo) y representa el número exacto de transmisiones que se han generado.
- Mensajes retransmitidos por el maestro (esclavo).- Se encuentra en la misma situación que el anterior sólo que especifica el número de mensajes re-enviados por el maestro (esclavo).
- Mensajes (*respuesta a petición / acuse de recibo*) recibidos.- Se refiere a los mensajes que llegan al maestro o al esclavo y permanecen en el buffer de entrada serial.
- Mensajes con CRC válido.- En este punto los mensajes tienen un CRC correcto, es decir, pasaron la prueba de redundancia cíclica exitosamente.
- Retransmisiones con CRC válido.- Representa a los mensajes con CRC válido pero que tienen un número de secuencia repetido, es decir, mensajes que ya fueron recibidos y procesados.
- Mensajes esperados.- Se refiere a los mensajes *respuesta a petición* que se les extrajo la *carga útil* y se agregaron al archivo. En el caso de los mensajes *acuse de recibo* se indican que los mensajes *respuesta a petición* llegaron sin errores a su destino. A cada mensaje enviado le corresponde un mensaje *acuse de recibo*.

En la Tabla 5.4a se muestran los resultados del segundo bloque, donde el maestro recibió 90 mensajes *respuesta a petición* (transmitidos por el esclavo U5) en cada muestra de las 320 sin existir el envío de retransmisiones. Los mensajes que tuvieron un CRC válido fueron 90 en cada muestra. Por lo que no existen retransmisiones en la parte de mensajes retransmitidos con CRC válido. Los mensajes esperados fueron 90 en cada muestra, ya que el esclavo fragmentó el archivo en 90 mensajes *respuesta a petición* para su transmisión.

Mensajes recibidos por el maestro		
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	90	100%
Retransmitidos por el esclavo	0	100%
Mensajes recibidos	90	100%
Con CRC válido	90	100%
Retransmisiones con CRC válido	0	100%
Mensajes esperados	90	100%

a)

Mensajes recibidos por el esclavo		
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	90	100%
Retransmitidos por el maestro	0	100%
Mensajes recibidos	90	100%
Con CRC válido	90	100%
Retransmisiones con CRC válido	0	100%
Mensajes esperados	90	100%

b)

Tabla 5.4 Resultados del segundo bloque con bajos niveles de interferencia.

En la Tabla 5.4b, se muestran los resultados del esclavo. Donde existen 90 mensajes *acuse de recibo* en cada muestra enviados por el maestro y cero retransmisiones. Así que se recibieron en el buffer serial del esclavo 90 mensajes de este tipo por cada una de las 320 muestras. Los mensajes con un CRC válido son 90 por muestra y no existieron retransmisiones con CRC valido. Por último, el número de mensajes esperados es de 90, lo cual es correcto ya que el archivo se dividió en 90 fragmentos.

Se puede considerar que la tabla anterior muestra los resultados de una comunicación ideal, donde el ruido del medio ambiente no está presente, por lo tanto todos los resultados mencionados en esta sección son los idealmente esperados.

5.4 COMUNICACIÓN EN ESPACIOS CERRADOS

Se realizaron dos pruebas en espacios cerrados (el termino "espacios cerrados", se define como el lugar que ofrece cierto control de la interferencia) cada uno de ellos con distintas características.

El primero se realizó en el aula F-142 audio visual del CINVESTAV-GDL, donde la probabilidad de interferencia es mínima. El segundo espacio fue una sala de cómputo donde las interferencias, dadas las características del espacio no era de fácil control.

A continuación se muestran los resultados obtenidos durante la comunicación en los espacios cerrados, donde intervienen un maestro y dos esclavos.

5.4.1 Comunicación Inalámbrica Realizada en el Aula F-142.

Los parámetros de la transmisión son:

- Duración de la prueba, 2:53 hr.
- Distancia entre el maestro y los esclavos, 3 mts.
- Distancia entre los esclavos, 5 mts.
- Sin obstáculos.
- Lugar de la prueba, Aula F142 (audio-visual del CINVESTAV-GDL).
- El archivo transmitido contiene la secuencia 11111111 (FF en hexadecimal).
- Duración de la transmisión del archivo, 6.407 segundos.
- El número de muestras realizadas es de 322.

En la Tabla 5.5 se muestran los resultados del primer bloque. El maestro ejecutó la sección búsqueda de esclavos una vez en cada muestra, es decir, una vez el 100% de las muestras.

Tipo de mensaje	# mensajes por muestra	% del mensaje durante el muestreo
Mensajes recibidos por el esclavo 5		
Búsqueda de esclavos #ID != U5	98	100%
Búsqueda de esclavos #ID = U5	1	100%
Mensajes recibidos por el esclavo 15		
Búsqueda de esclavos #ID != U15	98	100%
Búsqueda de esclavos #ID = U15	1	100%
Mensajes recibidos por el maestro		
Respuesta de disponibilidad	2	100%

Tabla 5.5 Resultados del primer bloque (Aula F-142).

La Tabla 5.6 muestra el número de mensajes recibidos en U5 y el maestro en el segundo bloque. Se observa que entre los mensajes *respuesta a petición* enviados por U5 y los mensajes recibidos en el maestro no existen pérdidas, ya que el maestro detecta todas los mensajes enviados. Pero existe un 0.31% de retrasmisiones de los mensajes *respuesta a petición* debido a que llegaron con errores durante

el muestreo. Cabe mencionar que en esta prueba el archivo transmitido se divide en 77 mensajes. En el caso del esclavo U5, se reciben sin errores los mensajes *acuse de recibo* y sin la necesidad de retransmitir durante el tiempo que se realizó la prueba.

Mensajes recibidos por el maestro				
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	77	99.69%	78	0.31%
Retransmitidos por el esclavo	1	0.31%	-	-
Recibidos	77	99.69%	78	0.31%
Con CRC válido	77	100%	-	-
Retransmisiones con CRC válido	0	100%	-	
Esperados	77	100%	-	-

Mensajes recibidos por el esclavo # 5				
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	77	100%	-	-
Retransmitidos por el maestro	0	100%		-
Recibidos	90	11.18%	87	11.18%
Con CRC válido	77	85.61%	78	14.29%
Retransmisiones con CRC válido	0	100%	-	
Esperados	77	100%	-	-

Tabla 5.6 Resultados del maestro-U5 en el segundo bloque (Aula F-142).

En la Tabla 5.7 se muestra el número de mensajes recibidos en el maestro y U15 en el segundo bloque. El maestro recibió en el buffer serial todas los mensajes *respuesta a petición* enviados por el esclavo # 15, pero existe un 8.07% de retransmisiones ya que llegaron con errores durante el muestreo. El esclavo recibió un 7.45% de retransmisiones de los mensajes *acuse de recibo* durante todo el muestreo.

Mensajes recibidos por el maestro				
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	77	91.93%	78	4.66%
Retransmitidos por el esclavo	1	4.66%	2	1.24%
Recibidos	77	91.93%	78	4.66%
Con CRC válido	77	92.55%	78	4.35%
Retransmisiones con CRC válido	1	4.35%	2	1.24%
Esperados	77	100%	-	-

Mensajes recibidos por el esclavo # 15				
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	77	92.55%	78	4.35%
Retransmitidos por el maestro	1	4.35%	2	1.24%
Recibidos	77	39.75%	78	31.99%
Con CRC válido	77	97.52%	78	2.48%
Retransmisiones con CRC válido	0	100%		
Esperados	77	100%	-	-

Tabla 5.7 Resultados del maestro-U15 en el segundo bloque (Aula F-142).

5.4.2 Comunicación Inalámbrica Realizada en el Edificio "A".

Los parámetros de la transmisión son:

- Duración de la prueba, 7 hrs.
- Distancia entre el maestro y el esclavo, 3.5 mts.
- Distancia entre los esclavos, 4.7 mts.
- Sin obstáculos.
- Lugar de la prueba, sala del edificio "A" en la planta baja(CINVESTAV-GDL).
- Se transmitieron 3 archivos, el primero contiene la secuencia 00H, el segundo contiene la secuencia 55H y el tercero tiene AAH.
- Duración de la transmisión por archivo, 7.397 seg.
- El número de muestras realizadas es de 1050.

En la Tabla 5.8 se muestran los resultados del primer bloque. El maestro ejecutó la sección de búsqueda de esclavos una vez por muestra durante el 100% de las muestras.

Tipo de mensaje	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Mensajes recibidos por el esclavo 5				
Búsqueda de esclavos #ID != U5	98	100%	-	-
Búsqueda de esclavos #ID = U5	1	100%	-	-
Mensajes recibidos por el esclavo 15				
Búsqueda de esclavos #ID != U15	98	99.90%	196	0.10%
Búsqueda de esclavos #ID = U15	1	99.90%	2	0.10%
Mensajes recibidos por el maestro				
Respuesta de disponibilidad	2	99.90%	1	0.10%

Tabla 5.8 Resultados del primer bloque (Edificio A).

En la Tabla 5.9 se muestran los resultados obtenidos de los mensajes recibidos en el esclavo U5 y el maestro, durante el segundo bloque. Se observa que entre los mensajes *respuesta a petición* enviados por el esclavo # 5 y los recibidos en el maestro no existen pérdidas, ya que el maestro detectó todos los mensajes recibidos. Pero existe un 12.38% de mensajes *respuesta a petición* retransmitidos ya que llegaron con error. (la cantidad mencionada, se obtiene restando al 100% el valor porcentual del reglón "Recibidos"). En el caso del esclavo # 5, se recibió un 8.19% retransmisiones del mensaje *acuse de recibo* durante el tiempo que se realizó la prueba. El número de segmentos en que se divide el archivo transmitido para esta prueba es de 90 mensajes.

Mensajes recibidos por el maestro				
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	90	87.62%	91	10.29%
Retransmitidos por el esclavo	1	10.29%	2	1.33%
Recibidos	90	87.62%	91	10.29%
Con CRC válido	90	91.81%	91	6.19%
Retransmisiones con CRC válido	1	6.19%	2	1.24%
Esperados	90	100%	-	-

Mensajes recibidos por el esclavo # 5				
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	90	91.81%	91	6.19%
Retransmitidos por el maestro	1	6.19%	2	1.24%
Recibidos	106	7.05%	108	6.57%
Con CRC válido	90	49.90%	91	29.62%
Retransmisiones con CRC válido	0	99.90%	1	0.10%
Esperados	90	100%	-	-

Tabla 5.9 Resultados del maestro-U5 en el segundo bloque (Edificio A).

En la Tabla 5.10 se muestran los resultados de los mensajes recibidos en el maestro y el esclavo U15, durante el segundo bloque.

Mensajes recibidos por el maestro				
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	90	74.38%	91	13.33%
Retransmitidos por el esclavo	1	13.33%	2	7.62%
Recibidos	90	74.67%	91	13.24%
Con CRC válido	90	99.81%	91	0.10%
Retransmisiones con CRC válido	1	0.10%	2	0.09%
Esperados	90	100%	-	-

Mensajes recibidos por el esclavo # 15				
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	90	99.81%	91	0.10%
Retransmitidos por el maestro	1	0.10%		
Recibidos	90	42.38%	91	27.43%
Con CRC válido	90	98.00%	91	1.81%
Retransmisiones con CRC válido	0	99.81%	1	0.10%
Esperados	90	100%	-	-

Tabla 5.10 Resultados del maestro-U15 en el segundo bloque (Edificio A).

En la Tabla 5.10 se observa que el maestro recibió un 25.33% de mensajes *respuesta a petición* retransmitidos durante todo el muestreo debido a que llegaron con error. Mientras que el esclavo U15 recibió un 0.19% de retransmisiones de los mensajes *acuse de recibo* durante el muestreo, estas retrasmisiones se produjeron porque el maestro recibió con errores los mensajes *respuesta a petición* que envió el esclavo.

5.5 COMUNICACIÓN EN ESPACIOS ABIERTOS

Esta sección muestra los resultados obtenidos de una comunicación inalámbrica realizada con un maestro y dos esclavos en espacios abiertos (al aire libre) donde el canal, debido al ambiente, tiene una interferencia mayor.

Se realizan cuatro pruebas variando la distancia en cada una de ellas. La distancia corresponde a 8, 14, 27 y 40 metros del maestro hacia los esclavos, como se muestra en la Figura 5.1. Las transmisiones presentan los siguientes parámetros generales: 1.- las antenas de los esclavos se colocan en ruta directa con el maestro. 2.- todas las pruebas se realiza con el *transceiver* MEDV-900-HP a una velocidad de transmisión de 19,200 bps. 3.- el lugar de prueba es a los alrededores del edificio "A" en el CINVESTAV-GDL.

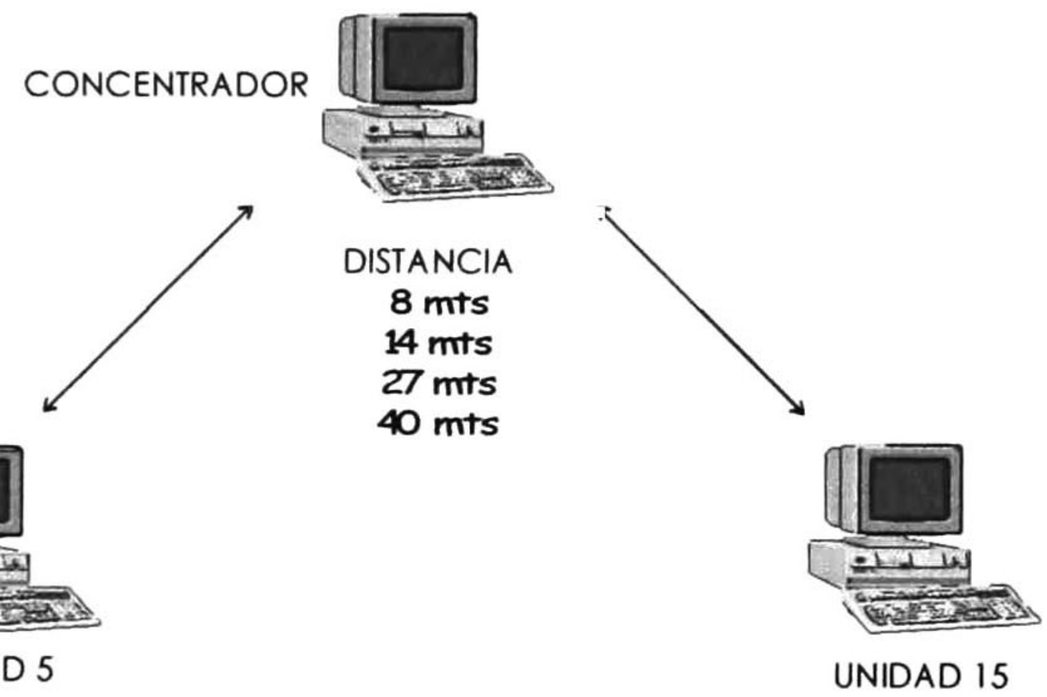


Figura 5.1 Ubicación de las estaciones a diferentes distancias.

5.5.1 Comunicación Inalámbrica a 8 metros

Los parámetros de la transmisión son:

- Duración de la prueba, 3 hrs.
- Distancia entre el maestro y los esclavos, 8 mts.
- Distancia entre los esclavos, 10.5 mts.
- Obstáculos entre los esclavos, piedras y plantas.
- El archivo transmitido contiene la secuencia 11111111 (FF en hexadecimal).
- Duración de la transmisión por archivo, 6.407 seg.
- El número de muestras realizadas es de 349.

En la Tabla 5.11 se muestran los resultados del primer bloque. El maestro ejecuta sólo una vez la sección búsqueda de esclavos en cada muestra, en el 100% de las muestras.

Tipo de mensaje	# mensajes por muestra	% del mensaje durante el muestreo
Mensajes recibidos por el esclavo 5		
Búsqueda de esclavos #ID != U5	48	92%
Búsqueda de esclavos #ID = U5	1	100%
Mensajes recibidos por el esclavo 15		
Búsqueda de esclavos #ID != U15	48	83.38%
Búsqueda de esclavos #ID = U15	1	100%
Mensajes recibidos por el maestro		
Respuesta de disponibilidad	2	100%

Tabla 5.11 Resultados del primer bloque (distancia 8 mts).

La Tabla 5.12 muestra el número de mensajes recibidos en el maestro y el esclavo # 5, durante el segundo bloque. En la tabla mencionada se observa que el maestro recibió todos los mensajes *respuesta a petición* enviados por el esclavo #5 y sólo existe un 0.57% de mensajes retransmitidos debido a que se detectaron errores. En el caso del esclavo #5, se recibió un 0.57% de mensajes *acuse de recibo* retransmitidos durante el tiempo que se realizó la prueba. Cabe mencionar que en esta prueba el archivo transmitido se divide en 77 mensajes *respuesta a petición.*

Mensajes recibidos por el maestro				
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	77	99.43%	78	0.57%
Retransmitidos por el esclavo	1	0.57%		
Recibidos	77	99.43%	78	0.57%
Con CRC válido	77	99.43%	78	0.57%
Retransmisiones con CRC válido	1	0.57%		
Esperados	77	100%	-	-

Mensajes recibidos por el esclavo # 5				
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	77	99.43%	78	0.57%
Retransmitidos por el maestro	1	0.57%		-
Recibidos	93	10%	90	10%
Con CRC válido	72	72%	78	21.14%
Retransmisiones con CRC válido	0	100%		-
Esperados	90	100%	-	-

Tabla 5.12 Resultados del maestro-U5 en el segundo bloque (distancia 8 mts).

La Tabla 5.13 muestra los mensajes del maestro y del esclavo # 15 obtenidos durante el segundo bloque. El maestro recibió en el buffer todos los mensajes *respuesta a petición* enviados por el esclavo #15, pero existe un 51.42% de retransmisiones que durante el muestreo llegaron con errores. El esclavo #15 recibió un 46.42% de retransmisiones en los mensajes *acuse de recibo* durante todo el muestreo.

Mensajes recibidos por el maestro				
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	77	48.58%	78	28.08%
Retransmitidos por el esclavo	1	28.08%	2	11.76%
Recibidos	77	48.58%	78	28.08%
Con CRC válido	77	53.58%	78	26.07%
Retransmisiones con CRC válido	1	26.07%	2	9.46%
Esperados	77	100%	-	-

Mensajes recibidos por el esclavo # 15				
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	77	53.58%	78	26.07%
Retransmitidos por el maestro	1	26.07%	2	9.46%
Recibidos	92	9.17%	91	6.88%
Con CRC válido	77	100%	-	-
Retransmisiones con CRC válido	0	100%	-	-
Esperados	77	100%	-	-

Tabla 5.13 Resultados del maestro-U15 en el segundo bloque (distancia 8 mts).

5.5.2 Comunicación Inalámbrica a 14 metros

Los parámetros de la transmisión son:

- Duración de la prueba, 2:02 hrs.
- Distancia entre el maestro y los esclavos, 14 mts.
- Distancia entre los esclavos, 13.8 mts.
- Obstáculos entre las unidades, ventiladores.
- El archivo transmitido contiene la secuencia 11111111 (FF en hexadecimal).
- Duración de la transmisión por archivo, 6.407 seg.
- El número de muestras realizadas es de 194.

En la Tabla 5.14 se muestran los resultados del primer boque. El maestro ejecutó la sección de búsqueda de esclavos una vez en cada muestra durante el 100% de las muestras.

Tipo de mensaje	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Mensajes recibidos por el esclavo 5				
Búsqueda de esclavos #ID != U5	98	100%	-	-
Búsqueda de esclavos #ID = U5	1	100%	-	-
Mensajes recibidos por el esclavo 15				
Búsqueda de esclavos #ID != U15	98	81.16%	196	13.77%
Búsqueda de esclavos #ID = U15	1	81.16%	2	13.77%
Mensajes recibidos por el maestro				
Respuesta de disponibilidad	2	74.74%	1	25.26%

Tabla 5.14 Resultados del primer bloque (distancia 14 mts).

En la Tabla 5.15 se muestran los resultados obtenidos de los mensajes recibidos en el maestro y el esclavo #5 durante el segundo bloque. En dicha tabla se observa que existe un 3.09% de mensajes *respuesta a petición* retransmitidos durante el muestreo, debido a que un 2.58% llegó con errores y un 0.51% no pudo ser validado. El esclavo #5 presenta un 3.10% de retransmisiones en los mensajes *acuse de recibo* durante todo el muestreo. El número de mensajes *respuesta a petición* en que se divide el archivo transmitido para esta prueba es de 77.

Mensajes recibidos por el maestro				
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	77	97.42%	78	2.06%
Retransmitidos por el esclavo	1	2.06%	3	0.52%
Recibidos	77	96.39%	78	2.58%
Con CRC válido	77	96.91%	78	2.06%
Retransmisiones con CRC válido	1	2.06%	3	1.03%
Esperados	77	100%	-	-

Mensajes recibidos por el esclavo # 5				
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	77	96.90%	78	2.06%
Retransmitidos por el maestro	1	2.06%	3	1.03%
Recibidos	84	9.28%	81	8.25%
Con CRC válido	77	97.42%	78	2.06%
Retransmisiones con CRC válido	0	99.48%	2	0.52%
Esperados	77	100%	-	-

Tabla 5.15 Resultados del maestro-U5 en el segundo bloque (distancia 14 mts).

En la Tabla 5.16 se muestran los resultados obtenidos de los mensajes recibidos en el maestro y el esclavo #5, durante el segundo bloque. El maestro recibió un 74.62% de mensajes *respuesta a petición* retransmitidos durante el muestreo, debido que llegaron con error. Mientras que el esclavo #15 recibió un 56.46% de retransmisiones de los mensajes *acuse de recibo.*

Mensajes recibidos por el maestro				
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	77	25.38%	78	6.52%
Retransmitidos por el esclavo	1	6.52%	9	3.62%
Recibidos	77	31.16%	78	10.14%
Con CRC válido	77	43.48%	78	7.25%
Retransmisiones con CRC válido	1	7.25%	2	3.62%
Esperados	77	77.54%	-	-

Mensajes recibidos por el esclavo # 15				
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	77	43.54%	78	7.25%
Retransmitidos por el maestro	1	7.25%	2	3.62%
Recibidos	80	5.07%	87	4.38%
Con CRC válido	77	52.17%	78	7.97%
Retransmisiones con CRC válido	0	100%	-	
Esperados	77	77.54%	3	2.17

Tabla 5.16 Resultados del maestro-U15 en el segundo bloque (distancia 14 mts).

5.5.3 Comunicación Inalámbrica a 27 metros

Los parámetros de la transmisión son:

- Duración de la prueba, 1:47 hr.
- Distancia entre el maestro y el esclavo, 27 mts.
- Distancia entre los esclavos, 25 mts.
- Obstáculos entre los esclavos, edificio A.
- El archivo transmitido contiene la secuencia 11111111 (FF en hexadecimal).
- Duración de la transmisión por archivo, 6.407 seg.
- El número de muestras realizadas es de 348.

En la Tabla 5.17 se muestran los resultados del primer bloque. El maestro realizó sólo una vez la sección de búsqueda de esclavos en cada muestra, el 100% de las muestras.

Tipo de mensaje	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Mensajes recibidos por el esclavo 5				
Búsqueda de esclavos #ID != U5	48	99.71%	46	0.29%
Búsqueda de esclavos #ID = U5	1	99.71%	2	0.29%
Mensajes recibidos por el esclavo 15				
Búsqueda de esclavos #ID != U15	48	99.71%	46	0.29%
Búsqueda de esclavos #ID = U15	1	99.71%	2	0.29%
Mensajes recibidos por el maestro				
Respuesta de disponibilidad	2	99.43%	1	0.57%

Tabla 5.17 Resultados del primer bloque (distancia 27 mts).

La Tabla 5.18 muestra el número de mensajes recibidos en el maestro y U5, durante el segundo bloque. En la tabla se observa que el maestro recibió 6.88% de retransmisiones. Cabe mencionar que en esta prueba el archivo transmitido se divide en 77 mensajes *respuesta a petición*. En el caso del esclavo #5, se recibió un 0.29% de mensajes *acuse de recibo* retransmitidos durante el tiempo que se realizó la prueba.

En la Tabla 5.19 se muestran los resultados obtenidos de los mensajes recibidos en el maestro y el esclavo #15, durante el segundo bloque. El maestro recibió un 56.03% de mensajes *respuesta a petición* con retransmisión durante el muestreo. En el esclavo #15 se recibió un 19.83% de mensajes retransmitidos del tipo *acuse de recibo*.

Mensajes recibidos por el maestro				
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	77	93.12%	78	2.29%
Retransmitidos por el esclavo	1	2.29%	2	0.57%
Recibidos	77	93.12%	78	2.29%
Con CRC válido	77	99.71%	78	0.29%
Retransmisiones con CRC válido	1	0.29%		
Esperados	77	100%	-	-

Mensajes recibidos por el esclavo # 5				
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	77	99.71%	78	0.29%
Retransmitidos por el maestro	1	0.29%		
Recibidos	89	9.74%	92	9.46%
Con CRC válido	77	59.31%	78	31.23%
Retransmisiones con CRC válido	0	100%		
Esperados	77	100%	-	-

Tabla 5.18 Resultados del maestro-U5 en el segundo bloque (distancia 27 mts).

Mensajes recibidos por el maestro				
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	77	43.97%	78	22.99%
Retransmitidos por el esclavo	1	22.99%	2	11.78%
Recibidos	77	44.83%	78	22.70%
Con CRC válido	77	80.17%	78	10.34%
Retransmisiones con CRC válido	1	10.34%	2	3.74%
Esperados	77	100%	-	-

Mensajes recibidos por el esclavo # 15				
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	77	80.17%	78	10.34%
Retransmitidos por el maestro	1	10.34%	2	3.74%
Recibidos	78	25.00%	79	20.11%
Con CRC válido	77	99.14%	78	0.86%
Retransmisiones con CRC válido	0	100%		
Esperados	77	100%	-	-

Tabla 5.19 Resultados del maestro-U15 en el segundo bloque (distancia 27 mts).

5.5.4 Comunicación Inalámbrica a 40 metros

Los parámetros de la transmisión son:

- Duración de la prueba, 1:52 hrs.
- Distancia entre el maestro y los esclavos, 40 mts.
- Distancia entre los esclavos, 35 mts.
- Obstáculos entre los esclavos, edificio A.
- El archivo transmitido contiene la secuencia 00000000 (00 en hexadecimal).
- Duración de la transmisión por archivo, 7.397 seg.
- El número de muestras realizadas es de 218.

La Tabla 5.20 muestra los resultados del primer bloque. El maestro ejecutó la sección de búsqueda de esclavos una vez en cada muestra durante el 100% de las muestras.

Tipo de mensaje	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Mensajes recibidos por el esclavo 5				
Búsqueda de esclavos #ID != U5	48	98.62%	46	0.46%
Búsqueda de esclavos #ID = U5	1	100%	-	-
Mensajes recibidos por el esclavo 15				
Búsqueda de esclavos #ID != U15	48	100%	-	-
Búsqueda de esclavos #ID = U15	1	100%	-	-
Mensajes recibidos por el maestro				
Respuesta de disponibilidad	2	100%	-	-

Tabla 5.20 Resultados del primer bloque (distancia 40 mts).

En la Tabla 5.21 se muestran los resultados obtenidos de las mensajes recibidos en el maestro y el esclavo #15, durante el segundo bloque. En la tabla se observa que existe un 19.27% de mensajes *respuesta a petición* retransmitidos durante el muestreo. Y en el esclavo #5 existe un 1.83% de mensajes retransmitidos del tipo *acuse de recibo*. El número de mensajes *respuesta a petición* en que se divide el archivo transmitido para esta prueba es de 90.

Mensajes recibidos por el maestro				
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	90	80.73%	91	12.84%
Retransmitidos por el esclavo	1	12.84%	3	2.29%
Recibidos	90	80.73%	91	12.84%
Con CRC válido	90	98.17%	91	0.46%
Retransmisiones con CRC válido	1	0.46%	2	0.46%
Esperados	90	100%	-	-

Mensajes recibidos por el esclavo # 5				
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	90	98.17%	91	0.46%
Retransmitidos por el maestro	1	0.46%	2	0.46%
Recibidos	90	24.77%	91	15.14%
Con CRC válido	90	92.66%	91	4.59%
Retransmisiones con CRC válido	0	98.62%	1	0.46%
Esperados	90	100%	-	-

Tabla 5.21 Resultados del maestro-U5 en el segundo bloque (distancia 40 mts).

En la Tabla 5.22 se observa que el maestro rechazó un 25.23% de los mensajes *respuesta a petición* por lo que provocó retransmisiones de los mensajes en el esclavo. Mientras que el esclavo # 15 recibió un 1.83% de retransmisiones del mensaje *acuse de recibo* durante el muestro.

Mensajes recibidos por el maestro				
Mensaje *respuesta a petición*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el esclavo	90	74.77%	91	9.63%
Retransmitidos por el esclavo	1	9.63%	2	0.46%
Recibidos	90	74.77%	91	10.09%
Con CRC válido	90	98.17%	91	1.38%
Retransmisiones con CRC válido	1	1.38%	2	0.46%
Esperados	90	100%	-	-

Mensajes recibidos por el esclavo # 15				
Mensaje *acuse de recibo*	# mensajes por muestra	% del mensaje durante el muestreo	# mensajes por muestra	% del mensaje durante el muestreo
Enviados por el maestro	90	98.17%	91	1.38%
Retransmitidos por el maestro	1	1.38%	2	0.46%
Recibidos	92	20.64%	91	10.10%
Con CRC válido	90	94.95%	91	5.05%
Retransmisiones con CRC válido	0	99.08%	1	0.92
Esperados	90	100%	-	-

Tabla 5.22 Resultados del maestro-U15 en el segundo bloque (distancia 40 mts).

Para concluir el análisis anterior, a continuación se grafican los resultados de las retransmisiones obtenidos en las secciones 5.3, 5.4 y 5.5 para observar el comportamiento conforme cambia la distancia. En la Figura 5.2 se muestran los resultados del enlace maestro-esclavo 5.

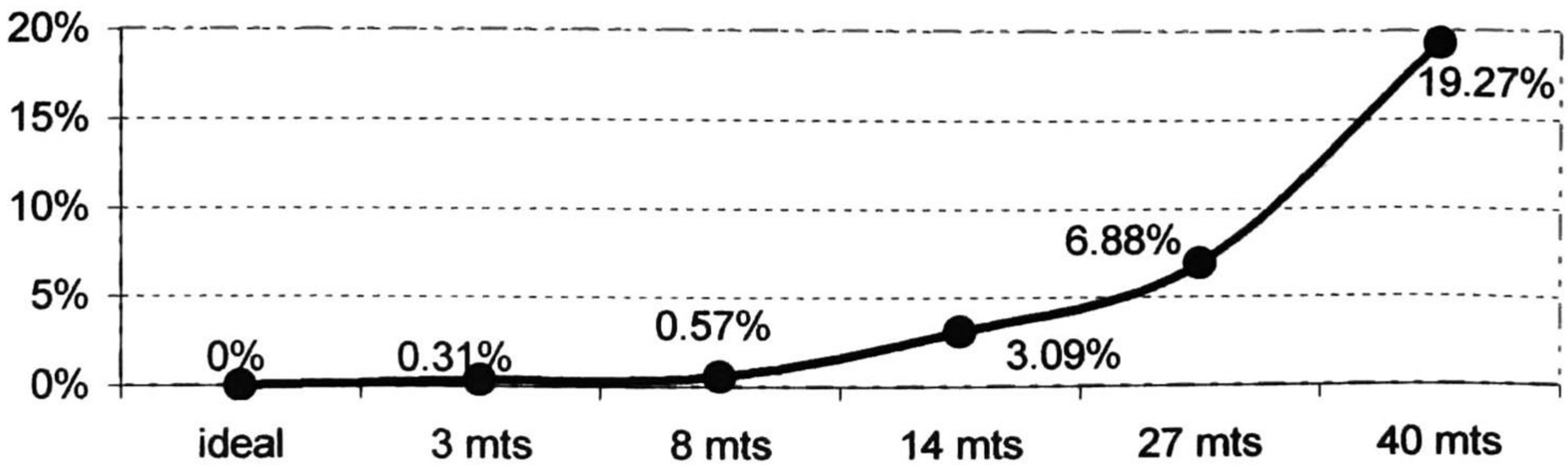


Figura 5.2 Resultados de las retransmisiones realizadas por el esclavo 5.

En la Figura 5.3 se muestran los resultados de las retransmisiones en el enlace maestro-esclavo 15. En el caso de la comunicación en espacios abiertos, se puede observar que entre la distancia de 8 a 27 mts existe una incremento de retransmisiones debido a la existencia de unos motores para la extracción del aire y para el sistema del aire acondicionado del edificio "A" dentro del área de enlace. Pero conforme se aleja el esclavo (40 mts) de la fuente de interferencia las retransmisiones bajan considerablemente.

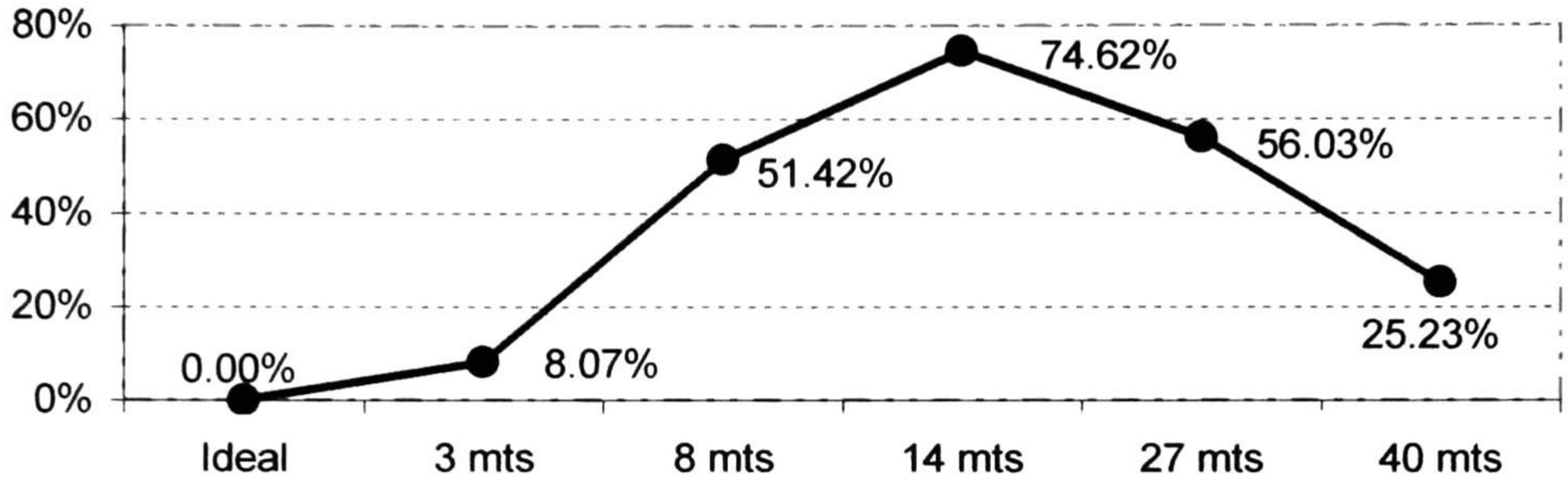


Figura 5.3 Resultados de las retransmisiones realizadas por el esclavo 15.

5.6 VERIFICACIÓN DE LOS ARCHIVOS TRANSMITIDOS

En esta sección se compara la información almacenada en el maestro con los archivos transmitidos de los esclavos, con la finalidad de determinar el porcentaje de similitud entre estos. Se realizó un programa en Builder C++ para revisar byte a byte los archivos existentes en el maestro con los originales, en la siguiente tabla se muestra el porcentaje de igualdad.

	Archivos enviado por:	
Distancia	**Esclavo 5**	**Esclavo 15**
3 mts	100%	100%
8 mts	100%	100%
14 mts	100%	100%
27 mts	100%	100%
40 mts	100%	100%

Tabla 5. 23 Porcentaje de similitud entre los archivos originales y los transmitidos.

Los archivos que se transmitieron se indican en la tabla siguiente.

Nombre del Archivo	Formato	Capacidad	Transmisor (U5 o U15)
00.txt	Código ASCII 0x00	11,270 Bytes	Ambos
55.txt	Código ASCII 0x55	11,266 Bytes	Ambos
AA.txt	Código ASCII 0xAA	11,253 Bytes	Ambos
FF.txt	Código ASCII 0xFF	9,660 Bytes	Ambos
Sol.exe	Ejecutable	171,616 Bytes	Unidad 5
Mshearts.exe	Ejecutable	123,024 Bytes	Unidad 15

Tabla 5.24 Información de los archivos transmitidos.

CAPÍTULO VI

6. CONCLUSIONES Y TRABAJO FUTURO

6.1 CONCLUSIONES

Las conclusiones del presente trabajo de tesis son:

- Se propuso un sistema de cobro electrónico por el servicio del transporte público.
- Se propuso el protocolo SyT para mejorar el sistema de cobro por medio de una comunicación inalámbrica entre concentradores y los autobuses.
- Se implementó un programa en leguaje de alto nivel (builder C++) que representa el protocolo SyT, denominado programa operativo maestro-esclavo.

Las pruebas del programa operativo se realizaron con el *transceiver* MEDV-900-HP por cumplir con los requerimientos de hardware, tales como costo del equipo y por tener una velocidad de transmisión arriba de 9,600 bps.

En las pruebas se tiene una exactitud de recepción de la información del 100% y una retransmisión de 0.00%, ambas realizadas con bajos niveles de interferencia. El termino *bajos niveles de interferencia* se refiere a una comunicación a través de cables por el puerto serial.

También se establece que el dispositivo MEDV-900-HP es susceptible al ruido o interferencias generadas por sistemas externos. Como se muestra en las pruebas realizadas en espacios cerrados (lugar que cuenta con un control de la interferencia) donde se tienen retransmisiones de 0.31% para el caso del aula F-142, pero en la planta baja del edificio A obtenemos 12.38% de retransmisiones. La diferencia entre el aula F-142 y la planta baja del edificio A, es que en este último se cuenta con varias PC's y un sistema de aire acondicionado.

En las pruebas realizadas en espacios abiertos (al aire libre) el *transceiver* presenta una limitante en distancias arriba de los 70 m debido a que la señal baja a –73 dBm, por lo que le resulta imposible demodular la señal recibida.

Una parte importante del programa operativo es al momento de validar los mensajes recibidos ya que en el buffer serial de la PC los mensajes sufren ciertos retrasos y sin una buena técnica de detección tales mensajes pueden ser mal interpretados o perdidos. Otro elemento rescatable es el tiempo de espera de los mensajes en el maestro y los esclavos, dado que este intervalo de tiempo se ve afectado por la velocidad de procesamiento en las computadoras. Se debe obtener un intervalo promedio para lograr la transmisión o recepción del mensaje antes de que el intervalo finalice.

6.2 TRABAJO FUTURO

Como trabajo futuro se recomiendan los siguiente puntos:

1. Agregar al programa operativo maestro-esclavo una etapa para validar a las unidades de cobro y las tarjetas de pre-pago. También se requiere de una etapa para el cifrado de la información a transmitir.

2. Implementar el hardware de la unidad de cobro descrito en el capítulo 1, con el firmware necesario para que el microcontrolador coordine los bloques existentes en la unidad.

3. Convertir el programa operativo del esclavo descrito en leguaje de alto nivel del capítulo 4 a lenguaje máquina para agregarlo al microcontrolador.

4. Realizar pruebas de la comunicación inalámbrica entre el concentrador y la unidad de cobro implementada en un autobús para retroalimentar los resultados finales.

APÉNDICE A

A. ACRÓNIMOS

ABM	*Asynchronous Balanced Mode*
ACL	*Asynchronous Connection-Less*
ACK	*Positive Acknowledge*
ARM	*Asynchronous Response Mode*
ASIC	*Application Specific Integrated Circuit*
AUX	*Auxiliary*
BE	*Búsqueda de Esclavos*
BiCMOS	*Bipolar Complementary Metal Oxide Semiconductor*
BSA	*Basic Service Area*
BSS	*Basic Service Set*
BTDK2	*Bluetooth Development Kit 2.0*
C	*Concentrador*
CAC	*Channel Access Code*
CFP	*Contention-free Period*
CP	*Contention Period*
CRC	*Cyclic Redundancy Check*
CRC-C	*Cyclic Redundancy Check Load*
CSMA/CA	*Carrier Sense Multiple Access with Collision Avoidance*
CTS	*Clear-To-Send*
DA	*Destination Address*
DAC	*Device Access Code*
DBPSK	*Differential Binary Phase Shift Keying*
DC	*Direct Current*
DCF	*Distributed Coordination Function*
DIAC	*General Inquiry Access Code*
DH	*Data-High*
DM	*Data-Medium*
DQPSK	*Differential Quadrature Phase Shift Keying*
DS	*Distribution System*
DSSS	*Direct Sequence Spread Spectrum*
DTBS	*Distributed Time-Bounded Services*
DTC	*Data Transfer Controller*
DV	*Data-Voice*
EEPROM	*Erase Electrical Programming Read Only Memory*
ESS	*Extended Service Set*
FEC	*Forward Error Correction*
FHS	*Frequency Hop Synchronization*
FHSS	*Frequency Hopping Spread Spectrum*
FSK	*Frequency Shift Keying*
GFSK	*Gaussian Frequency Shift Keying*
GIAC	*General Inquiry Access Code*
GLE	*Generación de la Lista de Esclavos*
HCI	*Host Controller Interface*

HDLC	*High-level Data Link Control*
HIPERLAN	*High-Performance European Radio LAN*
HV	*High-quality Voice*
IAC	*Inquiry Access Code*
IBSS	*Independent Basic Service Set*
ID	*Identify*
IFS	*Interframe space*
ISM	*Industrial-Scientific-Medical*
IUE	*Interfaz Usuario-Esclavo*
IUM	*Interfaz Usuario-Maestro*
L2CAP	*Logical Link Control And Adaptation Protocol*
LAPs	*Lower Address Parts*
LLC	*Logical Link Control*
LMP	*Link Manager Protocol*
LNA	*Low Noise Amplifier*
LSB	*Less Significant Bit*
LTCC	*Low Temperature Cofired Ceramic*
MAC	*Medium Access Control*
MODEM	*Modulador-Demodulador*
MPDU	*MAC Protocol Data Unit*
MSB	*More Significant Bits*
MSDU	*MAC Service Data Unit*
NAK	*Negative Acknowledge*
NRM	*Normal Response Mode*
ORX	*Objetivos del Hardware*
OSI	*Open Systems Interconnection*
OSX	*Objetivos del Software*
PCF	*Point Coordination Function*
PCOMTU-GDL	*Protocolo para la transmisión de datos del transporte urbano de Guadalajara.*
PDU	*Protocol Data Unit*
PHY	*Physical*
PI	*Petición de la Información*
PLL	*Phase Locked Loop*
PPM	*Pulse Position Modulation*
PV	*Punto de Venta*
QoS	*Quality of Service*
RA	*Receiver Address*
RAI	*Recepción y Almacenamiento de la Información*
RE	*Respuesta del Esclavo*
REJ	*Reject*
RHX	*Requerimiento de Hardware*
RNR	*Receive Not Ready*
RR	*Receive Ready*
RSSI	*Received Signal Strength Indicator*
RSX	*Requerimiento de Software*
RTS	*Request-To-Send*
SA	*Source Address*
SAW	*Surface Acoustic Wave*
SCO	*Synchronous Connection-Oriented*

SDLC	*Synchronous Data Link Control*
SNR	Signal-Noise Rate
SREJ	*Selective Reject*
STA	*Station*
SyT	*PCOMTU-GDL*
TA	*Transmitter Address*
TDD	*Time Division Duplex*
TDT	*Tamaño de la Carga Útil*
TP	*Tarjeta de Pre-pago*
TPU	*Timer Pulse Unit*
UA/I	*User Asynchronous/Isochronous*
UAP	*Upper Address Part*
UC	*Unidad de Cobro*
UT	*Último Mensaje*
VCO	*Voltage Controlled Oscillator*
VCXO	*Voltage Controlled Crystal Oscillator*
WDS	*Wireless Development System*
WDT	*Watchdog Timer*
WLAN	*Wireless Local Area Network*

APÉNDICE B

B. ESTÁNDAR DE PROTOCOLO INALÁMBRICO

B.1 INTRODUCCIÓN

La comunicación inalámbrica es una tecnología de rápido crecimiento que permite a múltiples tipos de usuarios establecer un enlace a las redes de comunicación. Las redes inalámbricas se consideran una alternativa para la comunicación en lugares de difícil acceso. Este tipo de redes busca la estandarización de los protocolos de comunicación entre computadoras móviles, teléfonos celulares y manejadores de dispositivos que permitan un adecuado intercambio de datos.

En la práctica es importante conocer los estándares porque desarrollan especificaciones en la subcapa de control de acceso al medio (MAC) y en capa física (PHY) para la conectividad inalámbrica en estaciones fijas, portátiles y en movimiento dentro de un área local[1]. El propósito de los estándares son los siguientes:

- Definir los mecanismos y algoritmos que permiten llevar a cabo los enlaces.
- Definir un marco de referencia que regule la interconexión y operatividad entre equipos de diferentes fabricantes.

En este apéndice se estudian dos protocolos con la finalidad de entender sus estándares y especificaciones. Uno de ellos es el Bluetooth, el cual se utiliza para comunicaciones inalámbricas en dispositivos móviles de bajo costo como agendas electrónicas, computadoras portátiles entre otros, mientras que el estándar IEEE 802.11 es utilizado en la mayoría de las redes inalámbricas para la comunicación entre computadoras a alta velocidad, hasta 11 Mbps.

B.2 PROTOCOLO BLUETOOTH

B.2.1 Descripción

Un enlace Bluetooth es un enlace de corto alcance basado en el protocolo Bluetooth, el cual tiene la finalidad de establecer enlaces en sistemas portátiles y dispositivos electrónicos. Las características importantes son: la robustez, poca complejidad, bajo consumo de energía y costo.

El protocolo Bluetooth establece una comunicación entre dispositivos mediante la técnica de conmutación de circuitos y conmutación de paquetes. Bluetooth permite la comunicación de datos con tiempo crítico, como el requerido en voz y audio, el cual es referido como enlace síncrono. También realiza la comunicación de datos con paquetes insensibles al tiempo, comúnmente conocido como enlace asíncrono. Bluetooth puede soportar un enlace asíncrono de datos o hasta tres enlaces síncronos simultáneos de voz. También puede permitir enlaces con datos asíncronos y síncronos. Cada enlace síncrono tiene una velocidad de transmisión de 64 kbps en cada dirección, mientras que un enlace asíncrono asimétrico tiene una velocidad máxima de 723.2 kbps y hasta 57.6 kbps en dirección opuesta o 433.9 kbps si es simétrico[2].

Bluetooth proporciona conexiones punto a punto y punto a multipunto. En la primera conexión sólo dos unidades Bluetooth están involucradas: un maestro y un esclavo, como se muestra en la Figura B.1(a). Por otro lado en una conexión punto a multipunto el canal se comparte con varias unidades Bluetooth, donde una unidad actúa como maestro y las otras unidades se comportan como esclavos, Figura B.1(b). El maestro tiene la tarea de enviar información del reloj para

sincronizarse y la dirección del dispositivo para que los esclavos puedan calcular los saltos de frecuencia[2]. A la agrupación de un maestro con varios esclavos se le conoce como piconet. El número máximo de esclavos que actúan dentro de una piconet es siete. Por otra parte, cuando la piconet tiene más de siete esclavos, los esclavos restantes son bloqueados por el maestro enviándolos al estado de espera. Los esclavos en estado de espera no pueden actuar en el canal de la piconet, pero permanecen sincronizados con el maestro (por medio del reloj y la dirección del dispositivo) esperando su activación. El acceso de los esclavos activos al canal es controlado por el maestro. Los esclavos dentro de la piconet no pueden establecer un enlace directo entre ellos, sólo pueden establecer el enlace con el maestro.

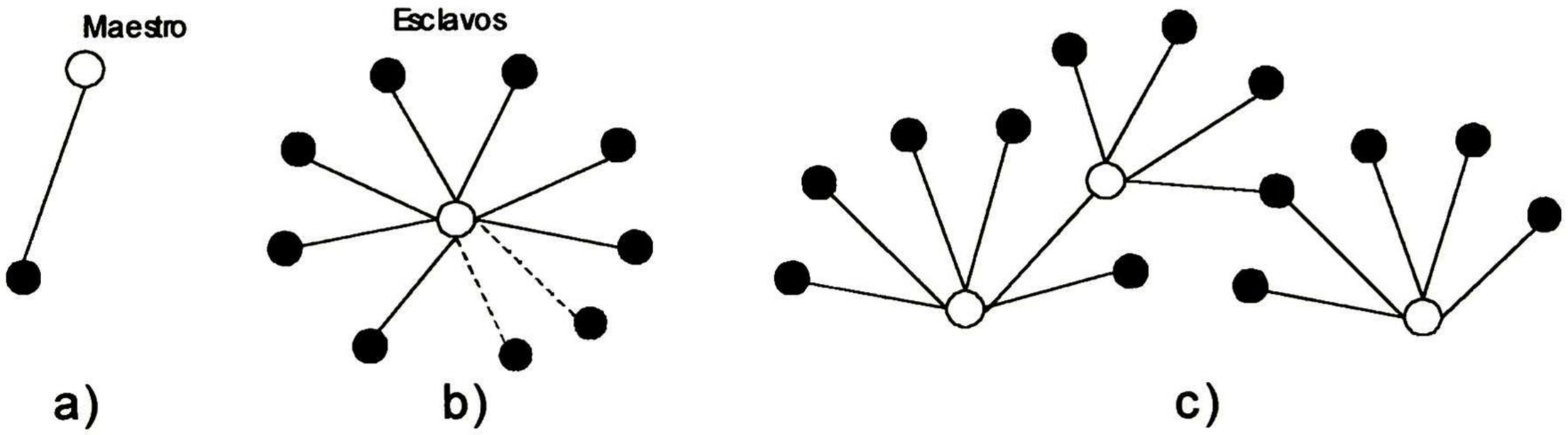


Figura B.1 (a) Piconet punto-punto. (b) Piconet punto-multipunto. (c) Scatternet.

Múltiples piconets con un área de cobertura definida forman un scatternet. Cada piconet tiene sólo un maestro, pero éste puede actuar como esclavo en otra piconet como se ilustra en la Figura B.1(c). Los esclavos pueden participar en diferentes piconets basándose en el multiplexaje de división de tiempo (TDM), es decir, el esclavo está unos periodos en una piconet y otros en la otra piconet.

B.2.2 Canal de Comunicación

El dispositivo bluetooth opera en la banda Medica-Científica-Industrial (ISM) a 2.4 GHz. El ancho de banda es de 83.5 MHz, por lo que el rango es de 2.400-2.4835 GHz. En esta banda existen 79 canales de RF con espacio entre canales de 1 MHz. El esquema de modulación es conmutación de frecuencia Guassian (GFSK), la velocidad de transmisión máxima es de 1Mb/s. Esta característica brinda mayor seguridad y robustez.

El canal es representado mediante una secuencia de salto pseudo-aleatoria a través de 79 canales. La secuencia de salto es única para cada piconet y está determinada mediante el direccionamiento del dispositivo maestro; la fase en la secuencia de salto es determinada mediante el reloj del maestro. El canal es dividido en varios periodos de tiempo donde cada periodo corresponde a un salto de frecuencia RF. Los saltos consecutivos corresponden a diferentes frecuencias RF. Cabe señalar que la relación nominal de saltos es de 1600 saltos por segundo. Además, todas las unidades que participan en una piconet están sincronizadas en tiempo y saltos hacia con el canal[3].

Otra característica importante es el periodo de operación. Cada periodo tiene una duración de 625 microsegundos, y están numerados en función al reloj del piconet maestro. La numeración del periodo tiene un rango de 0 a $(2^{27}-1)$. Dentro de estos periodos de tiempo, el maestro y el esclavo pueden transmitir sus paquetes. Para ésto se utiliza un esquema de división de tiempo (TDD), donde el maestro y el esclavo transmiten alternativamente, como se muestra en la Figura B.2. El maestro iniciará su transmisión sólo en los periodos de tiempo denominados como impares y el esclavo en los pares. El inicio del paquete es alineado con el inicio correspondiente del periodo.

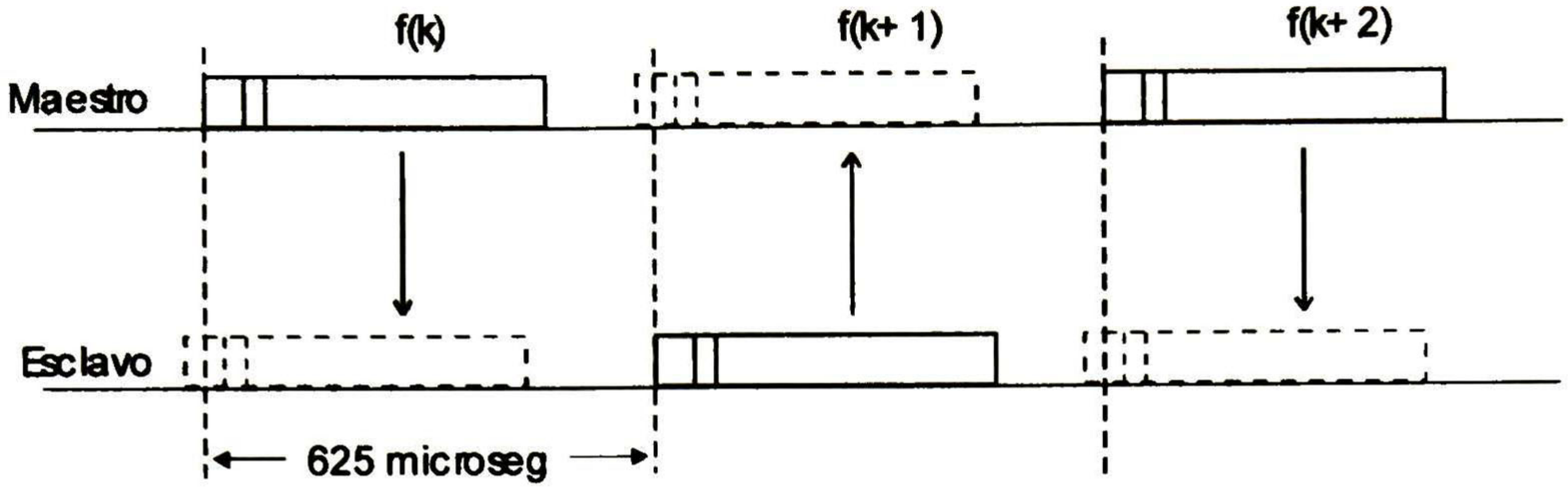


Figura B.2 Transmisión con esquema TDD (síncrono).

Los paquetes transmitidos por el maestro o el esclavo pueden extenderse hasta cinco periodos de tiempo. Por lo que el valor de la frecuencia se mantendrá estable durante la duración del paquete[3]. Por ejemplo, para un paquete de un periodo, el valor de la frecuencia de salto se obtendrá del número actual del reloj maestro Bluetooth; para paquetes multi-periodos, el valor de la frecuencia se obtiene de igual forma, pero esta frecuencia permanecerá durante la transmisión del paquete entero (ver Figura B.3). Una vez que el paquete sea transmitido, la frecuencia del siguiente salto se obtiene del valor del reloj maestro Bluetooth.

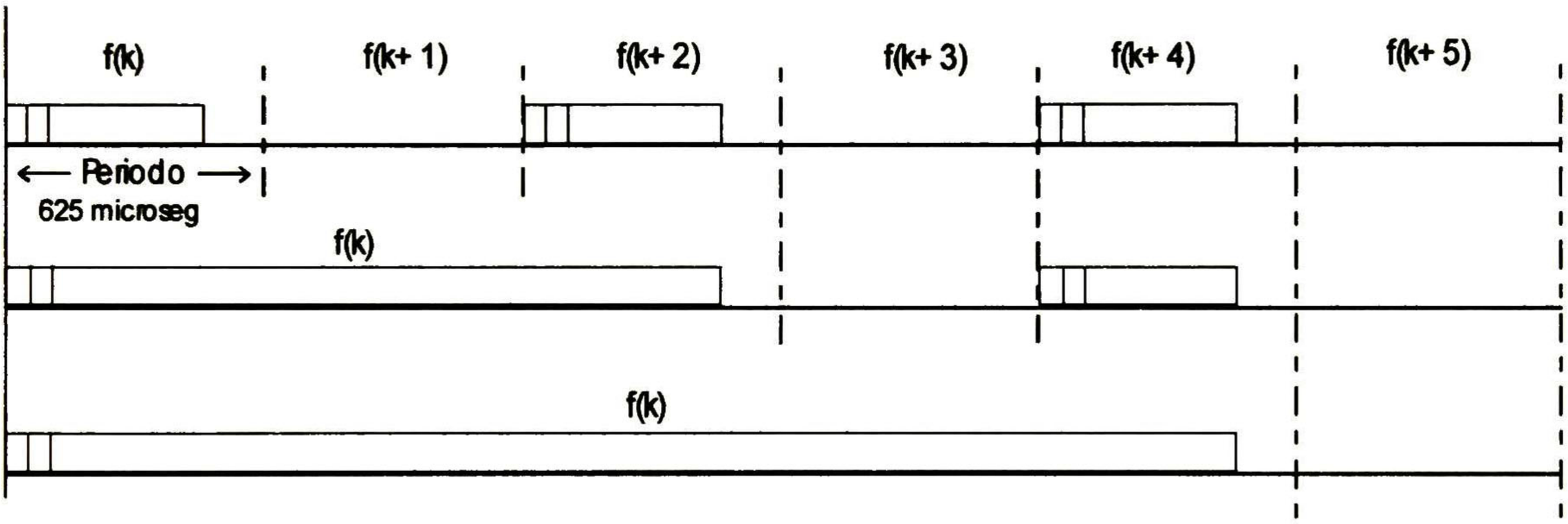


Figura B.3 Transmisión con esquema TDD para paquetes de multi-periodos (asíncrono).

B.2.3 Enlace Físico

Entre un maestro y un esclavo existen diferentes tipos de enlaces, aunque sólo se han definido dos tipos. El tipo de enlace depende del paquete de datos que se quiere transmitir.

B.2.3.1 *Enlace Síncrono Orientado a Conexión (SCO)*

El enlace SCO es simétrico y se utiliza en enlaces punto a punto entre un maestro y un esclavo específico. El enlace síncrono orientado a conexión reserva periodos de tiempo, por lo que se considera una conexión de conmutación de circuitos entre el maestro y el esclavo. Este enlace es usado típicamente para transportar voz. Un maestro soporta hasta 3 enlaces síncronos con el mismo esclavo o con diferentes esclavos. Y un esclavo acepta hasta 3 enlaces con el mismo maestro o dos enlaces con maestros diferentes. Los paquetes SCO nunca son retransmitidos debido a que los paquetes comúnmente pertenecen a procesos que requieren de cierta coordinación con el tiempo.

El maestro manda los paquetes SCO en intervalos regulares, identificado como T_{SCO}, al esclavo en los periodos reservados maestro-esclavo, a lo que el esclavo responde en el siguiente periodo esclavo-maestro a menos que un esclavo diferente haya sido direccionado en un periodo previo maestro-esclavo. En una comunicación ya establecida maestro-esclavo, si el esclavo falla al decodificar su dirección debido a alguna interferencia, el paquete recibido se perderá. Sin embargo, el esclavo podrá continuar enviando su información durante el periodo reservado SCO[4].

Un enlace síncrono orientado a conexión lo establece el maestro, al enviar un mensaje de inicialización SCO mediante el protocolo de manejo de enlace (LMP). Este mensaje contiene parámetros de temporización, tipo de paquete y el formato de codificación a usar durante el enlace. Los parámetros de temporización son el intervalo T_{SCO} y el intervalo del siguiente periodo, definido como D_{SCO}.

B.2.3.2 *Enlace Asíncrono no Orientado a Conexión (ACL)*

En los periodos no reservados a un enlace SCO, el maestro intercambia paquetes con cualquier esclavo que esté participando en la piconet; este enlace se realiza mediante la técnica de conmutación de paquetes entre el maestro y todos los esclavos activos que participan en la piconet. El enlace asíncrono no orientado a conexión soporta servicio tanto asíncrono como isócrono, y comúnmente es usado para transmitir datos. Entre un maestro y el esclavo únicamente puede existir un enlace asíncrono no orientado a conexión. Los paquetes ACL pueden ser retransmitidos para asegurar la integridad del dato.

Un esclavo puede regresar un paquete ACL en el periodo esclavo-maestro únicamente si ha sido direccionado en el periodo anterior maestro-esclavo. Si el esclavo falla al decodificar el direccionamiento del esclavo en el encabezado del paquete, éste no podrá transmitir.

Todos los paquetes no direccionados a un esclavo específico son considerados como paquetes de difusión y son leídos por todos los esclavos. Si no hay dato para mandar en el enlace ACL y no es requerido el protocolo de llamada selectiva (*polling*), entonces no se realiza la transmisión[3].

B.2.4 Paquetes

B.2.4.1 *Formato General.*

El dato en el canal de la piconet es transportado en paquetes. El formato del paquete general se muestra en la Figura B.4, donde el bit menos significativo (LSB) es el primer bit que se transmite. Cada paquete está constituido de 3 entidades: código de acceso, encabezado y carga útil. El agrupamiento del código de acceso y el encabezado tiene una longitud fija de 72 y 54 bits, respectivamente. La carga útil puede variar de 0 a 2745 bits y los paquetes pueden estar constituidos solo del campo denominado código de acceso, o del código de acceso más el encabezado, o también código de acceso más el encabezado más la carga útil.

B.2.4.2 *Formato del Código de Acceso*

Todo paquete inicia con un código de acceso. Si un paquete contiene la entidad de encabezado, entonces el código de acceso es de 72 bits de longitud, de lo contrario el código de acceso es de 68 bits. El código de acceso se utiliza para detectar la presencia de paquetes mediante los flancos del dato recibido y para direccionar el paquete a un dispositivo específico. Con el código de acceso se identifican todos los paquetes intercambiados en el canal de la piconet, ya que todos los paquetes enviados en la misma piconet son precedidos por el código de acceso del mismo canal. El código de

acceso está constituido de un preámbulo (*preamble*), una palabra síncrona y ocasionalmente un elemento de cola (*trailer*), como se muestra en la Figura B.4.

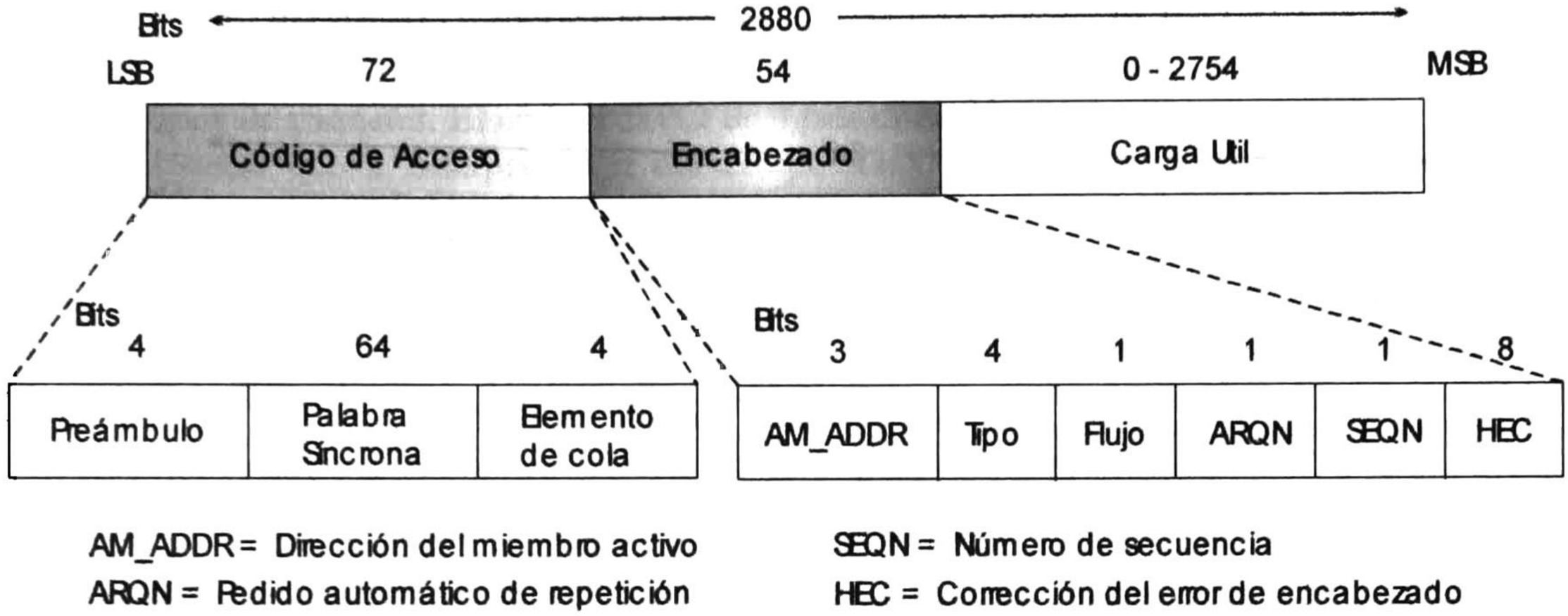


Figura B.4 Formato del paquete para el protocolo Bluetooth.

B.2.4.2.1 Tipos de Código de Acceso.

Los códigos de acceso están definidos en 4 tipos, los cuales son usados en los dispositivos para diferentes modos de operación:

1. Código de Acceso de Canal (CAC) – El CAC se deriva de la dirección de la parte baja (LAP) del dispositivo maestro y lo utilizan todos los dispositivos en la piconet durante el intercambio de los datos en una conexión.
2. Código de Acceso de Dispositivo (DAC) – El DAC se obtiene de la dirección LAP de un dispositivo específico. Este se utiliza en el proceso de paginación a un dispositivo específico y así conocer sus características con más detalle.
3. Código de Acceso de Búsqueda General (GIAC) – El GIAC se usa por todos los dispositivos durante el proceso de indagación o búsqueda, mientras no exista el conocimiento previo algún dispositivo que esté dentro del rango.
4. Código de Acceso de Búsqueda Dedicada (DIAC) – El DIAC es común en procesos de indagación o búsqueda de un grupo específico de dispositivos que comparten una característica en común, como impresoras o agendas electrónicas.

B.2.4.2.2 Preámbulo

El preámbulo es un patrón fijo de cero-uno de 4 bits, se utiliza para detectar los flacos del dato recibido y con ésto recuperar la señal del reloj. La secuencia 1010 es usada cuando el LSB de la palabra síncrona es 1, pero si este bit es cero entonces se usa la secuencia 0101.

B.2.4.2.3 Palabra Síncrona

La palabra síncrona es una palabra código de 64 bits formada con una dirección LAP del dispositivo Bluetooth de 24 bits más 6 bits predefinidos con la secuencia de Barker, con la finalidad de mejorar las propiedades de auto-correlación. Estos 30 bits son codificados en un código de bloques Bose-

Chaudhuri-Hocquenghem (64,30), garantizando una distancia Hamming $d_{min} = 14$ entre las palabras síncronas basadas en las diferentes direcciones LAPs, donde se obtienen 34 bits de paridad que son incluidos en el campo de la palabra síncrona (ver Figura B.5). La buena propiedad de autocorrelación de la palabra síncrona mejora el tiempo del proceso de sincronización. La derivación de la palabra síncrona está descrita en la sección 13.2[3].

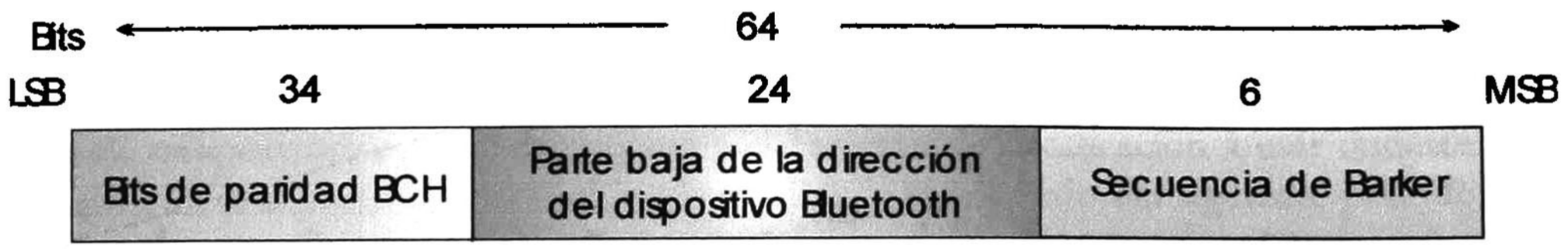


Figura B.5 Estructura de la palabra síncrona.

B.2.4.2.4 Elemento de Cola o Final

El elemento de cola es agregado a la palabra síncrona cuando el paquete contiene el código de acceso más el encabezado para el caso típico CAC. También se utiliza para el DAC, el DIAC y el GIAC cuando estos códigos utilizan el paquete de intercambio de sincronización de salto en frecuencia (FHS), durante el proceso de respuesta de página y de respuesta de indagación.

El elemento de cola es un patrón fijo de uno-cero de 4 bits. El trailer junto con los 3 bits más significativos (MSB) de la palabra síncrona forma un patrón de 7 bits el cual puede ser usado para extender la detección del dato recibido. La secuencia 1010 es usada cuando el MSB de la palabra síncrona es cero, pero si este bit es uno entonces se usa la secuencia 0101.

B.2.4.3 *Paquetes de Encabezado*

El encabezado contiene toda la información de control asociada con el paquete y el enlace, en la capa de control del enlace (LC). El encabezado está constituido de 6 campos:

- AM_ADDR Miembro activo de direccionamiento (3 bits).
- TIPO Código Tipo (4 bits).
- FLUJO Control de flujo (1 bit).
- ARQN Indicación de petición automática de repetición (1 bit).
- SEQN Numeración de secuencia (1 bit).
- HEC Verificación de error del encabezado (8 bits).

El encabezado total, incluyendo el HEC, tiene una longitud de 18 bits, Figura B.4, pero como es protegido con el esquema de corrección de error adelantado (FEC) con una relación de un 1/3 resulta un encabezado de 54 bits, quiere decir, que cada bit del encabezado se repite 3 veces.

B.2.4.3.1 Miembro Activo de Direccionamiento (AM_ADDR)

El AM_ADDR representa una dirección de miembro y es usado para identificar a cada miembro activo en la piconet. La forma de distinguir cada esclavo separadamente, es asignándole una dirección provisional de 3 bits durante el proceso de paginación. La técnica de conmutación de paquetes entre maestro y esclavo se realiza llevando el campo AM_ADDR del esclavo, es decir, el AM_ADDR del esclavo es usado tanto en la transmisión de paquetes maestro-esclavo como en la de esclavo-maestro.

El direccionamiento de ceros es reservado para el paquete de difusión del maestro al esclavo. A cada esclavo se le asigna un nuevo AM_ADDR cuando éstos se reconecten a la piconet.

B.2.4.3.2 TIPO

Existen 6 tipos de paquetes. El código TIPO de 4 bits distingue el tipo de paquete que se utiliza. Es importante notar que la interpretación del campo TIPO depende del enlace físico asociado al paquete. Primero, se debe determinar si el paquete es enviado en un enlace SCO o ACL, entonces, ésto puede establecer cual tipo de paquete ha sido recibido de los diferentes tipos de paquetes que se muestran en la Tabla B.1, Tabla B.2 y Tabla B.3. El código TIPO también revela cuántos periodos de tiempo necesitará el paquete que se recibe. Esto permite al receptor no direccionado abstenerse de usar el canal durante el tiempo restante.

Código TIPO $b_3b_2b_1b_0$	Tipo de Paquete	Encabezado de la Carga Útil (Bytes)	Carga Útil (Bytes)	Periodo
0011	DM1	1	0 – 17	1
0100	DH1	1	0 – 27	1
1001	AUX1	1	0 – 29	1
1010	DM3	2	0 – 121	3
1011	DH3	2	0 – 183	3
1110	DM5	2	0 – 224	5
1111	DH5	2	0 – 339	5

Tabla B.1 Paquetes para un enlace ACL.

Código TIPO $b_3b_2b_1b_0$	Tipo de Paquete	Encabezado de la Carga Útil (Bytes)	Carga Útil (Bytes)	Periodo
0101	HV1	na	10	1
0110	HV2	na	20	1
0111	HV3	na	30	1
1000	DV	1	10+(0-9)	1

Tabla B.2 Paquetes para un enlace SCO.

Código TIPO $b_3b_2b_1b_0$	Tipo de Paquete	Carga Útil (Bytes)	Periodo
0000	NULL	na	1
0001	POLL	na	1
0010	FHS	18	1
0011	ID	na	1

Tabla B.3 Paquetes utilizados en ambos enlaces.

B.2.4.3.3 FLUJO

Este bit controla el flujo de paquetes en un enlace ACL. Cuando el recipiente del buffer de recepción en un enlace ACL está lleno, envía una indicación de alto o paro (FLOW = 0) para detener la transmisión de datos temporalmente. Es de notar, que la señal de alto sólo se presenta en paquetes ACL. Así que los paquetes SCO aún pueden ser recibidos y también los paquetes que incluyen sólo información de control de enlace, como los paquetes de identificación (ID), los de llamada selectiva (*POLL*) y los denominados nulos (*NULL*). Cuando el recipiente del buffer de recepción está vacío, se envía una indicación para continuar (FLOW = 1). Cuando un paquete no es recibido, o cuando el encabezado de éste es erróneo, la indicación de continuar se asume implícitamente.

B.2.4.3.4 Indicación de Petición Automática de Repetición (ARQN)

Un bit indica el acuse de recibo (ARQN) y es utilizado para informar a la fuente una transferencia exitosa de datos de la carga útil con el código redundancia cíclica (CRC). Aquí puede presentarse un acuse de recibo positivo (ACK) o negativo (NAK). Si la recepción fue exitosa el acuse es enviado (ARQN = 1), pero si no lo es se envía un acuse negativo (ARQN = 0). Cuando no se recibe un mensaje respecto al acuse ARQN, se asume que ocurrió un NAK. El acuse negativo también retorna información de incumplimiento. El acuse es incluido en el encabezado del paquete retornado. El éxito de la recepción se verifica mediante el CRC.

B.2.4.3.5 Numeración de Secuencia (SEQN)

El bit SEQN provee un esquema de numeración secuencial para ordenar los paquete de datos recibidos. Para cada paquete nuevo transmitido que contenga datos con CRC, el bit SEQN es invertido. Esto se requiere para filtrar las retransmisiones al destinatario; si una retransmisión ocurre debido a un fallo en el acuse de recibo, el destinatario recibe el mismo paquete dos veces y mediante la comparación de SEQN las retransmisiones recibidas correctamente pueden ser eliminadas. El SEQN es agregado debido a la falta de numeración de paquetes en el esquema no numerado en ARQ.

B.2.4.3.6 Verificación de Error del Encabezado (HEC)

El campo HEC almacena un código de suma de comprobación para verificar la integridad del encabezado. Este está constituido de una palabra de 8 bits generada mediante el código de bloques CRC, donde el polinomio generador es $g(D) = (D + 1)(D^7 + D^4 + D^3 + D^2 + 1) = D^8 + D^7 + D^5 + D + 1$, representación octal 647. Antes de generar el HEC en el transmisor, el generador debe de ser inicializado con un valor de 8 bits obtenido de la dirección de la parte superior (UAP) del dispositivo Bluetooth. En el receptor se realiza el mismo proceso antes de verificar el HEC.

B.2.4.4 *Formato de la Carga Útil*

En la carga útil existen dos campos. El campo de voz (síncrono) y el de datos (asíncrono). En un enlace asíncrono los paquetes sólo tienen el campo de datos y en un enlace síncrono los paquetes sólo tienen el campo de voz, con la excepción del paquete de datos-voz (DV) el cual tiene ambos campos.

B.2.4.4.1 Campo de Voz

El campo de voz tiene una longitud fija. Para paquetes de voz con alta calidad (HV) la longitud del campo es de 240 bits; para los paquetes DV la longitud es de 80 bits. La estructura de la carga útil no contiene encabezado de carga útil y CRC, como en el campo de datos, ver Tabla B.2.

B.2.4.4.2 Campo de Datos

El campo de datos consta de 3 segmentos: el encabezado, el cuerpo de carga útil y el código CRC (sólo el paquete auxiliar1 (AUX1) no contiene el código CRC). En la Figura B.6 se muestra el formato de carga útil para los paquetes de datos.

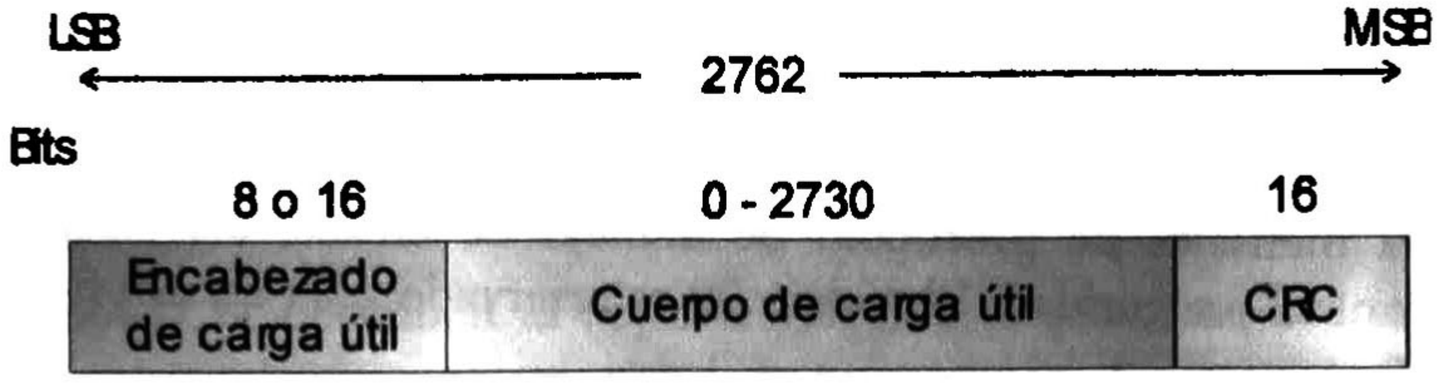


Figura B.6 Formato de la carga útil para paquetes ACL.

B.2.4.4.2.1 Encabezado de Carga Útil

Sólo el campo de datos tiene un encabezado de carga útil. El encabezado puede tener una longitud de uno o dos bytes, que depende de los periodos que comprenda el paquete. Los paquetes de un periodo tienen un encabezado de carga útil de 1 byte, los cuales se observan en la Tabla B.1, Tabla B.2 y Tabla B.3; los paquetes con duración de tres y cinco periodos tienen un encabezado de carga útil de 2 bytes, ver Tabla B.1.

El encabezado de carga útil se forma del canal lógico (*L_CH*), del control del flujo en el canal lógico (*FLOW*), y de la longitud de carga útil (*LENGTH*), como se observa en la Figura B.7. En el caso de un encabezado de 2 bytes, la longitud del indicador es extendida mediante 4 bits dentro del siguiente byte. Los 4 bits restantes de este segundo byte, corresponden al campo *UNDEFINED*, los cuales son reservados para un futuro uso y deberán estar en cero.

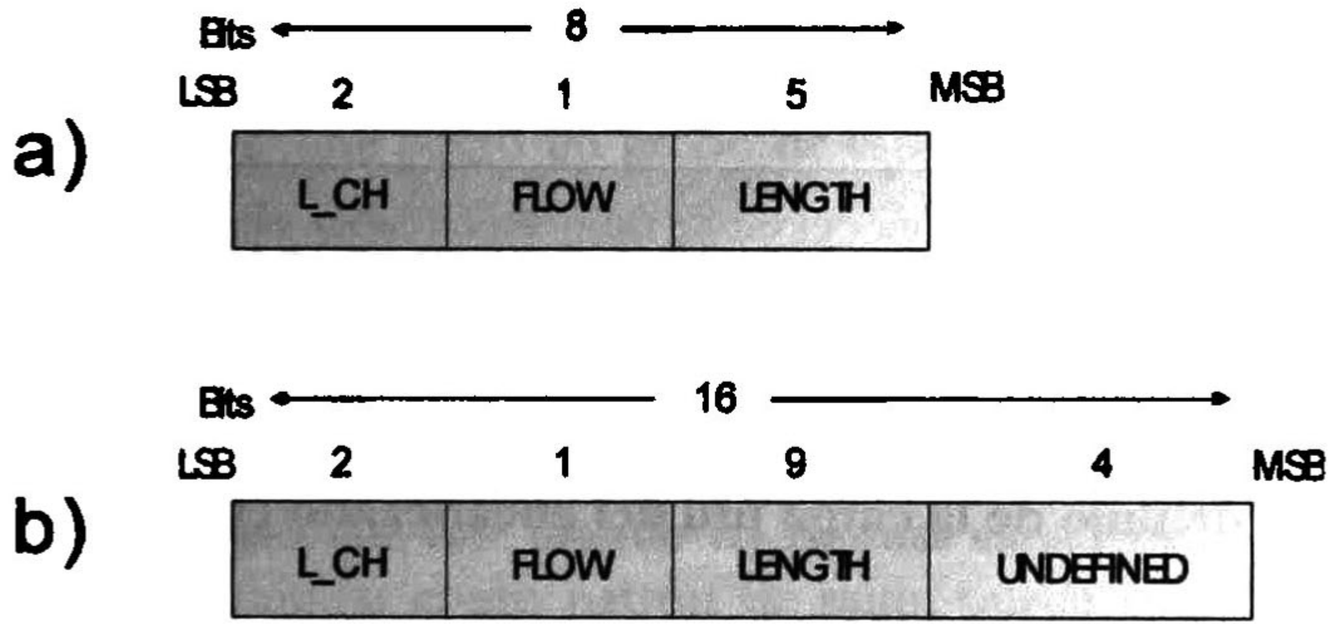


Figura B.7 Formatos del encabezado de carga útil para paquetes: (a) de un periodo y (b) multi-periodos.

- El campo ***L_CH*** indica cuando la carga útil a transmitir es la primera o la continuación de un mensaje del protocolo de adaptación y control de enlace lógico (L2CAP) o si es un mensaje del protocolo de manejo de enlace. En la Tabla B. 4 se muestra con más detalle el contenido del campo L_CH.

Código L_CH b_1b_0	Canal Lógico	Información
00	NA	No definido.
01	UA/UI	Continuación de fragmento de un mensaje L2CAP.
10	UA/UI	Inicio de un mensaje L2CAP o no existe fragmentación.
11	LA	Mensaje de la capa manejo de enlace.

Tabla B. 4 Campo del canal lógico (L_CH).

Un mensaje L2CAP puede ser fragmentado en varios paquetes. El código 10 es usado para transmitir el primer fragmento del mensaje o paquete L2CAP; el código 01 se utiliza para dar continuidad a los fragmentos siguientes. Si no hay fragmentación, el código 10 se utiliza para todo el paquete. El código 11 se usa para mensajes de protocolo de manejo de enlace, los cuales se utilizan para establecer un enlace, control y seguridad. El código 00 es reservado para un uso futuro.

- El campo ***indicador de flujo*** en la carga útil se utiliza para controlar el flujo por el canal lógico a nivel L2CAP. Cuando FLOW = 1 significa que está listo para mandar fragmentos y con FLOW = 0 significa que no puede mandar fragmento alguno.

No hay un requerimiento estricto de tiempo-real en el bit de flujo del encabezado de carga útil, es decir, en cualquier instante durante la transmisión puede ser recibido el bit para detener el flujo. La capa manejo de enlace es responsable de la configuración y procesamiento del bit de flujo en el encabezado de carga útil. El control de flujo en tiempo real es llevado a cabo a nivel de paquete mediante la capa controlador de enlace a través del bit de flujo en el encabezado. Con el bit de flujo de carga útil, el tráfico de una unidad remota puede ser controlada. Esto permite generar y enviar un paquete ACL con una longitud de carga útil de valor cero.

Las indicaciones de inicio y continuo de fragmentación L2CAP (L_CH = 10 y L_CH = 01) también retienen su contenido cuando la longitud de carga útil es igual a cero; por ejemplo, un inicio de fragmento no deberá de enviarse en medio de una transmisión de paquetes L2CAP. Siempre es seguro enviar un paquete ACL con una longitud de carga útil = 0 y L_CH = 10. El bit de flujo de carga útil tiene su propio significado para cada canal lógico (UA/I o LM), ver Tabla B. 5. En un canal LM, el control de flujo no es aplicable y el bit de flujo de carga útil es siempre llevado a uno.

Código L_CH b_1b_0	Uso y semántica del bit de flujo del encabezado de carga útil ACL
00	No definido, reservada para uso futuro.
01 o 10	Control de flujo del canal UA/I (el cual es usado para enviar mensajes L2CAP).
11	Siempre fijo a FLOW = 1 en una transmisión e ignora el bit en la recepción.

Tabla B. 5 Bit de flujo de la carga útil del encabezado en canales lógicos.

- El campo ***longitud*** determina el número de bytes (ej: palabra de 8 bits) en la carga útil, es decir, sólo el cuerpo de la carga útil.

B.2.4.4.2.2 Cuerpo de Carga Útil.

El cuerpo de carga útil incluye la información del usuario y determina la cantidad de información efectiva que llega. La longitud del cuerpo está indicada por el campo de longitud del encabezado de carga útil.

B.2.4.4.2.3 Generación del código CRC.

El código CRC de 16 bits en la carga útil es generado mediante el polinomio generador $g(D) = D^{16} + D^{12} + D^5 + 1$ (representación octal 210041) del esquema CRC-CCITT[3]. Antes de calcular el código CRC de la carga en transmisor y también en el receptor, el dispositivo generador CRC debe de ser inicializado con el valor de ocho bits obtenidos del direccionamiento de la parte superior del dispositivo Bluetooth y asignarle el valor de cero a los bits restantes del dispositivo generador.

B.3 ESTANDAR IEEE 802.11

B.3.1 Descripción

Los estándares IEEE 802.11 describen el soporte obligatorio para una red inalámbrica local a 1 Mbps con capacidad opcional para una velocidad de transmisión de datos de 2 Mbps. También se especifica el soporte obligatorio para una transferencia de datos asíncronos así como para el servicio distribuido de tiempo límite (DTBS). La transferencia de datos asíncrona se refiere al tráfico que es relativamente insensible a tiempos de retraso. Un ejemplo de datos asíncronos está disponible en la relación trafico-bits como es el correo electrónico y la transferencia de archivos. El tráfico de tiempo límite (*time-bounded*), por otro lado, es el tráfico que es limitado por un tiempo de retraso especificado para alcanzar una calidad de servicio aceptable (QoS) como el empaquetamiento de video y voz.

El particular interés en la especificación es el soporte a dos esquemas MAC fundamentales, el transporte asíncrono y servicio de tiempo límite. El primer esquema, función de coordinación distribuida (DCF), está diseñado para el transporte de datos asíncronos, donde todos los usuarios con datos a transmitir tienen la misma oportunidad para acceder a la red. La función de coordinación de punto (PCF) es el segundo esquema MAC. La función PCF está basada en la llamada selectiva, donde sólo se activa un usuario a la vez para transferencia de información mediante una búsqueda especifica. La función es principalmente diseñada para la transmisión de tráfico sensible a retraso[5].

B.3.2 Arquitectura

El servicio básico fijo (BSS) representa un bloque fundamental en la construcción de la arquitectura IEEE 802.11. Este servicio se define como un grupo de estaciones que están bajo el control de un único coordinador como la función de coordinación distribuida o la función de coordinación de punto. El área geográfica cubierta por el servicio básico fijo se conoce como Área de Servicio Básico (BSA). Conceptualmente, todas las estaciones que estén dentro del área de cobertura del servicio básico pueden comunicarse directamente con las estaciones restantes del BSS. Sin embargo, la degradación en el medio de transmisión debido al desvanecimiento del multipath, o a la interferencia de un servicio básico fijo cercano que esté usando las mismas características de la capa física como código de frecuencia y dispersión, o como patrón de salto, puede causar que algunas estaciones aparezcan como ocultas para otras.

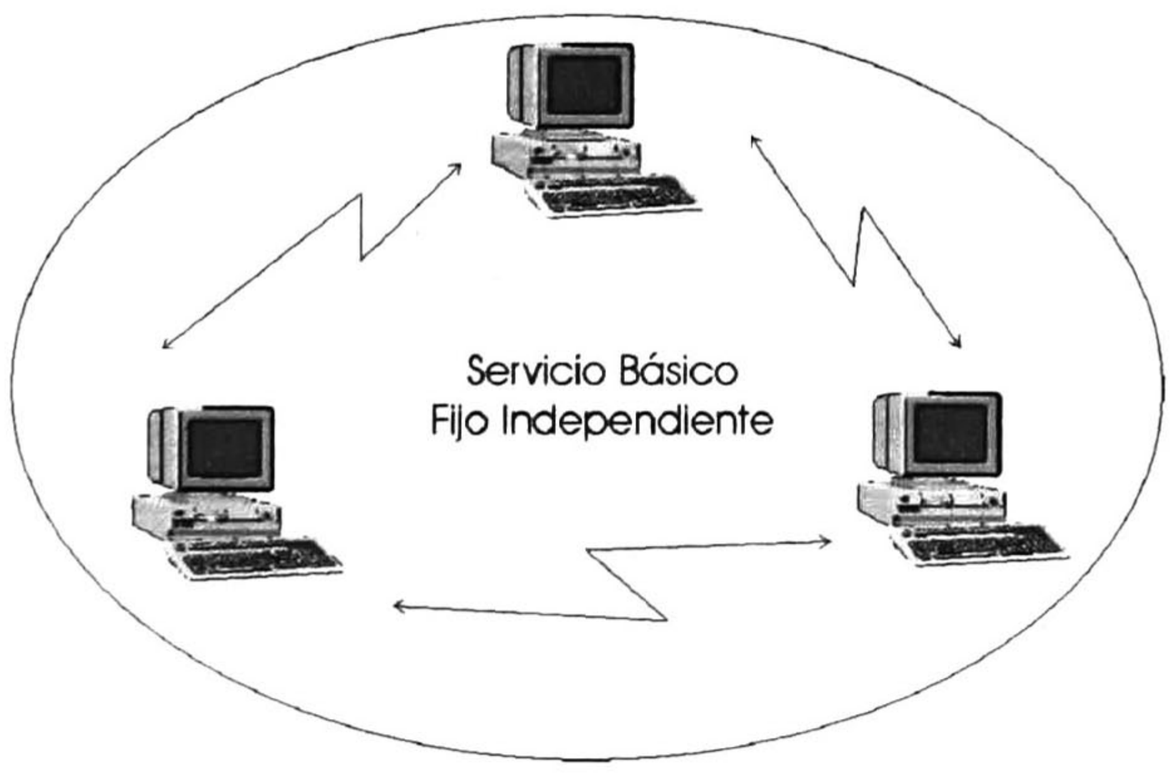


Figura B.8 Esquema de una red ad hoc.

La existencia de un agrupamiento deliberado de estaciones dentro del servicio básico fijo con la finalidad de establecer comunicación interna sin la ayuda de una red de infraestructura (*infrastructure network*) es conocida como una red *ad hoc*. En la Figura B.8 se muestra una red *ad hoc*, que también es conocida como servicio básico fijo independiente (IBSS).

Cada una de las estaciones puede establecer una comunicación directa con cualquier otra estación en el servicio básico fijo, sin el requerimiento de canalización de todo el tráfico a través de un punto de acceso (AP) centralizado. En contraste a la red *ad hoc*, las redes de infraestructura se establecen para proporcionar a los usuarios, de forma inalámbrica, servicios específicos y extensión de rango. La red de infraestructura en el contexto de la IEEE 802.11 está establecida a partir de un punto de acceso. Este último soporta extensiones de rango amplio mediante la disposición de puntos de integración necesarios para la conectividad de redes entre múltiples BSSs, formando con esto un servicio extendido fijo (ESS). El servicio extendido fijo tiene la apariencia de un servicio básico fijo a gran escala por la subcapa de control de enlace lógico (LLC) de cada estación (STA). En otras palabras, un servicio extendido fijo está constituido de múltiples servicios básicos fijos, los cuales están comunicados entre ellos debido a que el punto de acceso de cada servicio básico fijo se enlaza a un mismo sistema de distribución (DS), como se muestra en la Figura B.9. Un sistema de distribución puede ser considerado como una red de estructura principal que es responsable del transporte a nivel MAC de unidades de datos MAC (MSDU), es decir, es un puente que transporta paquetes MSDU entre los servicios básicos fijos y las redes de área local.

Un servicio extendido fijo también puede proporcionar un acceso directo a usuarios inalámbricos dentro de una red como la internet, esto es conocido como un portal. El portal es una entidad lógica que especifica los puntos de integración en el sistema de distribución donde la red IEEE 802.11 se integra con una red no IEEE 802.11. Si la red es una IEEE 802.X, el portal incorpora funciones análogas al puente; es decir, proporciona la extensión de rango y la translación entre diferentes formatos de marcos. La Figura B.9 ilustra un servicio extendido fijo desarrollado con dos servicios básicos fijos, un sistema de distribución y un portal de acceso a una red local cableada.

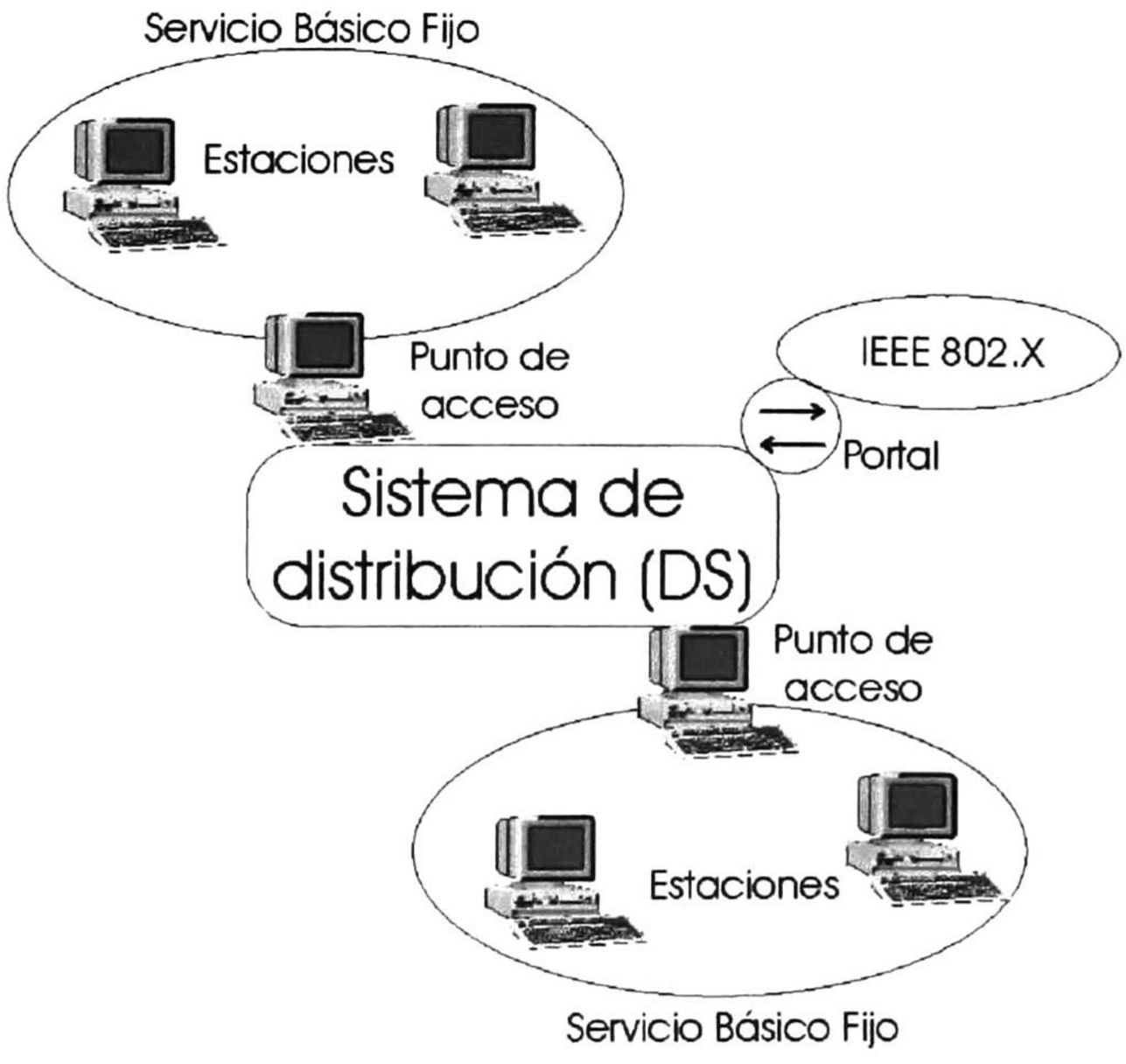


Figura B.9 Esquema de una red de infraestructura.

B.3.3 Capa Física

La especificación de la IEEE 802.11 define tres diferentes tipos de implementaciones para la capa física:

1. Espectro disperso por salto de frecuencia (FHSS)
2. Espectro disperso de secuencia directa (DSSS)
3. Infrarrojo (IR)

El método del espectro disperso por salto de frecuencia utiliza la banda ISM a 2.4000-2.4835 GHz. En los Estados Unidos se especifican 79 canales, las cuales están divididas en tres agrupaciones con 26 secuencias de salto fijo. El primer canal tiene una frecuencia central de 2.402 GHz, y todos los canales subsecuentes están separados a 1 MHz. La separación del canal corresponde 1 Mbps del ancho de banda instantáneo. Se comisionan diferentes secuencias de salto en múltiples servicios básicos fijos para coexistir en la misma área geográfica, lo cual puede llegar a ser importante para evitar la congestión y maximizar el rendimiento total en un único servicio básico fijo. La razón para tener tres diferentes secuencias de salto fijo es evitar prolongados periodos de colisión entre diferentes secuencias de salto en un agrupamiento. La mínima relación de saltos permitidos es 2.5 saltos/s. La relación de acceso básico es de 1 Mbps usando dos niveles de conmutación de frecuencia Gaussian, donde un 1 lógico es codificado usando la frecuencia (Fc+f) y para un cero lógico se utiliza la frecuencia (Fc-f), donde Fc es la frecuencia central y f es una frecuencia de menor valor. Para aumentar la relación de acceso a 2 Mbps se utilizan cuatro niveles GFSK, donde 2 bits están codificados a un tiempo usando cuatro frecuencias.

El sistema del espectro disperso de secuencia directa también utiliza la banda ISM, donde el rango básico de 1 Mbps es codificado con la conmutación de fase binaria diferencial (DBPSK), y para aumentar el rango a 2 Mbps se utiliza la conmutación de fase en cuadratura diferencial (DQPSK). La dispersión se hace mediante la división del ancho de banda disponible en 11 subcanales, separados 11 MHZ, y utilizando la secuencia Barker. La máxima capacidad del canal es por lo tanto (11 chip/símbolo)/(11 MHz) = 1 Mb/s cuando el DBPSK es usado. El código de Barker consiste en una secuencia de 11 bits, también conocidos como chips (10110111000) y representan un bit del dato a modular. Esta secuencia cuenta con propiedades matemáticas ideales para la modulación de ondas de radio. Una superposición y un servicio básico fijo cercanos pueden ser aceptados gracias a que se asegura una separación mínima de 30 MHz entre las frecuencias centrales de cada BSS. Este rígido requerimiento hará posible que sólo existan dos servicios básicos fijos cercanos y superpuestos para operar sin interferencia.

Las especificaciones del infrarrojo identifican un rango de longitud de onda de 850 a 950 nm. La banda del infrarrojo está diseñada para uso interior y opera con transmisiones no directas. Las especificaciones IR fueron diseñadas para hacer posible que las estaciones reciban en onda directa y las transmisiones reflejadas. La codificación de la relación de acceso básico es 1 Mbps si es desempeñada con 16 pulsos para la modulación de posición del pulso (PPM), donde 4 bits de datos son mapeados a 16 bits codificados para transmisión. El aumento de la relación de acceso a 2 Mbps se obtiene usando modulaciones de 4-PPM, donde 2 bits de datos son mapeados a 4 códigos de bits para transmisión.

B.3.4 Subcapa de Control de Acceso al Medio (MAC)

La subcapa de control de acceso al medio es responsable del proceso para la asignación del canal, del direccionamiento de la unidad de datos del protocolo (PDU), de dar formato al marco, de realizar la suma de comprobación, fragmentación y reagrupamiento de paquetes.

El medio de transmisión puede alternar entre el modo de contención, también conocido como periodo de contención (CP), y un periodo sin contención (CFP). En el modo de contención todas las estaciones compiten por el acceso al canal cada vez que necesiten transmitir paquetes. En el modo sin contención el medio es controlado por un punto de acceso, de esa forma se elimina la necesidad de competir por el acceso al canal.

B.3.4.1 *Formato General*

La IEEE 802.11 soporta 3 diferentes tipos de marcos: direccionamiento, control y datos. El marco de direccionamiento es usado para la asociación, desasociación, sincronización, temporización y autentificación de estaciones con el punto de acceso. El marco de control es utilizado para el esquema *handshakring* durante el periodo de contención para avisos de acuse de recibo positivos y al finalizar el periodo sin contención. El marco de datos es utilizado para la transmisión de datos durante el periodo de contención y sin contención. También puede ser combinado con el protocolo de llamada selectiva y el acuse positivo durante el CFP.

El formato del marco definido por el estándar IEEE 802.11 consiste de tres campos: encabezado, cuerpo del marco y CRC (ver Figura B.10).

B.3.4.1.1 *Encabezado*

El encabezado se utiliza para definir el tipo de marco, define el tiempo del uso del canal y establece el origen y el destino del marco dentro de una transmisión. El encabezado tiene una longitud de 30 octetos, es decir, 240 bits. Este se constituye de siete campos: control del marco, duración, dirección receptora, dirección transmisora, dirección destino, secuencia de control y dirección fuente.

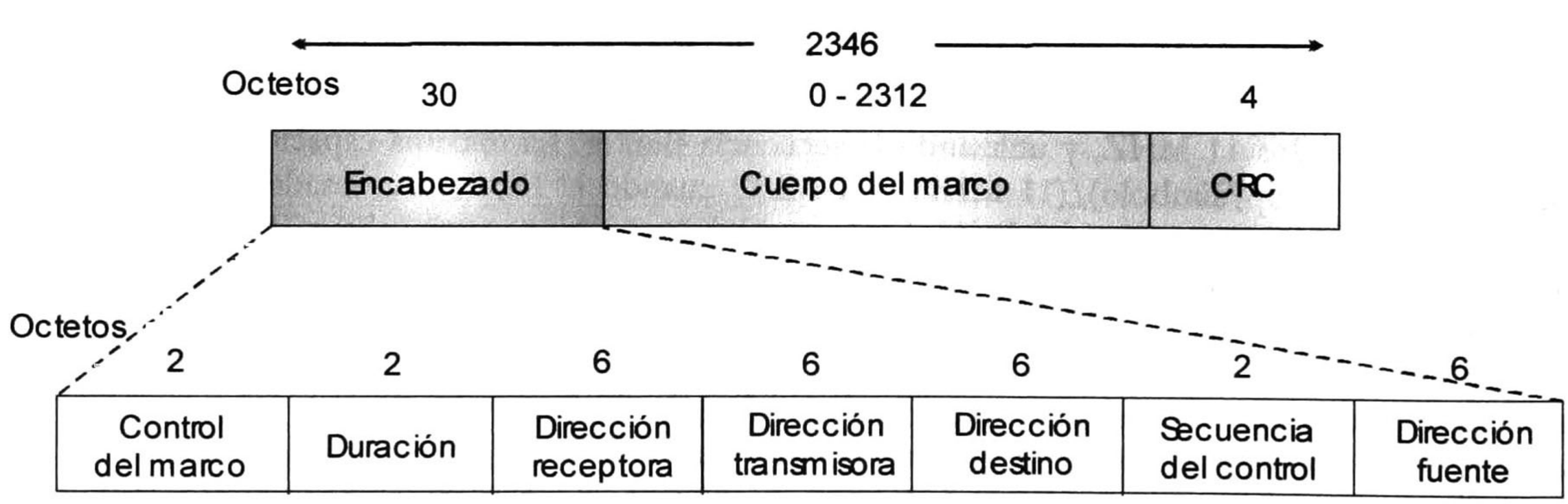


Figura B.10 Formato del marco para el estándar IEEE 802.11.

B.3.4.1.1.1 *Control del Marco*

El control del marco se divide en once campos como se muestra en la Figura B.11, los cuales con:

El campo *tipo* identifican el marco, el cual puede ser de direccionamiento, control o datos.

El campo *subtipo* identifican la modalidad del marco, como en el caso del tipo control que tiene las modalidades: listo para enviar(CTS), solicitud de envío(RTS), acuso de recibo, ahorro de potencia y llamada selectiva.

El campo *de DS* se activa con los marcos del tipo datos, que son destinados a un sistema de distribución.

El campo *para DS* se define para los marcos del tipo datos existentes en el DS.

El campo *más fragmentos* se activa cada vez que existen marcos del tipo datos y direccionamiento para transmitir, es decir, todavía no es último marco a enviar.

El campo *retransmisión* se utiliza cuando un marco tipo datos o direccionamiento se transmite por más de una vez.

El campo *direccionamiento de potencia* cuando tiene el valor de uno lógico indica que la estación está en el modo de ahorro de energía, y si se halla en cero indica que el punto de acceso (AP) se encuentra activo.

El campo *más datos* indica que la estación está en modo de ahorro de energía y que todavía tiene marcos en el buffer para enviar a un punto de acceso.

El campo *WEP* se activa sólo si el cuerpo del marco se ha procesado con el algoritmo de privacidad inalámbrica (WEP).

El campo *orden* se usa en los marcos del tipo datos que contengan un MSDU o un fragmento.

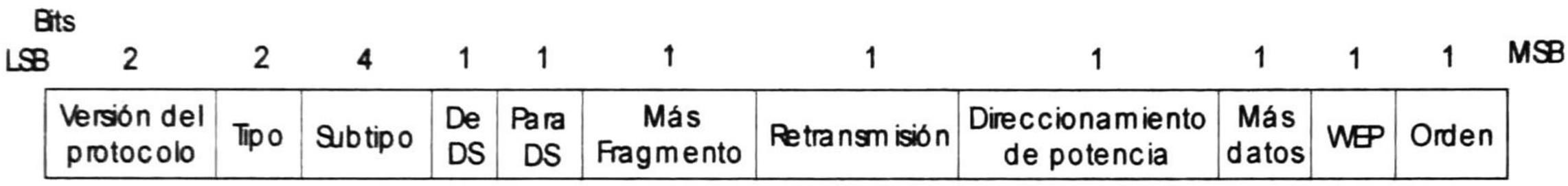


Figura B.11 Formato del campo de control.

B.3.4.1.1.2 *Duración*

El campo duración indica el tiempo, en microsegundos, que el canal estará ocupado a partir de la finalización del marco actual en el canal. Por lo que el canal será asignado durante este tiempo a una estación para la transmisión exitosa de una unidad de datos del protocolo MAC (MPDU). El MPDU es conjunto de paquetes manejados en la capa de control de acceso al medio.

B.3.4.1.1.3 *Dirección*

Los campos de dirección son utilizados para indicar la dirección destino (DA), la dirección fuente (SA), la dirección receptora (RA) y la dirección transmisora (TA). Aquí cada campo tiene una longitud de 48 bits.

B.3.4.1.1.4 *Secuencia de Control*

El campo secuencia de control se divide en dos subcampos, el número de secuencia y el número de fragmento.

El *número de secuencia* indica el número que le corresponde a cada unidad de datos MAC transmitido por una estación. Este campo tiene una longitud de 12 bits y se incrementa en uno para cada unidad de datos MAC.

El *número de fragmento* se utiliza para contabilizar cada fragmento transmitido exitosamente en unidades de datos MAC, el cual se mantiene constante en las retransmisiones. Este campo tiene el valor de cero cuando sólo se tiene un fragmento a transmitir o cuando se transmite el primer fragmento de la unidad de datos MAC. El campo número de fragmento tiene una longitud de 4 bits.

B.3.4.1.2 *Cuerpo del Marco*

El cuerpo del marco (MSDU) es una celda de longitud variable que consta de la carga útil y 56 bits para el cifrado y descifrado, sólo si se implementa el protocolo de privacidad equivalente al cableado (WEP).

B.3.4.1.3 Código de Redundancia Cíclica

El CRC de la Figura B.10 se utiliza para la detección de errores en el encabezado y en el cuerpo del marco; tiene una longitud de 32 bits. Este campo se obtiene con el polinomio generador $D(g) = D^{32} + D^{26} + D^{23} + D^{22} + D^{16} + D^{12} + D^{11} + D^{10} + D^8 + D^7 + D^5 + D^4 + D^2 + D + 1$.

B.3.4.2 Función de Coordinación Distribuida

Esta función es el método de acceso fundamental usado en la transferencia de datos asíncronos. Esta función opera sólo en redes *ad hoc*, aunque también puede operar o coexistir con la función de coordinación a punto en una red de infraestructura. La arquitectura MAC es representada en la Figura B.12, donde se muestra que la función de coordinación distribuida que está ubicada en la parte superior de la capa física y soporta el servicio de contención.

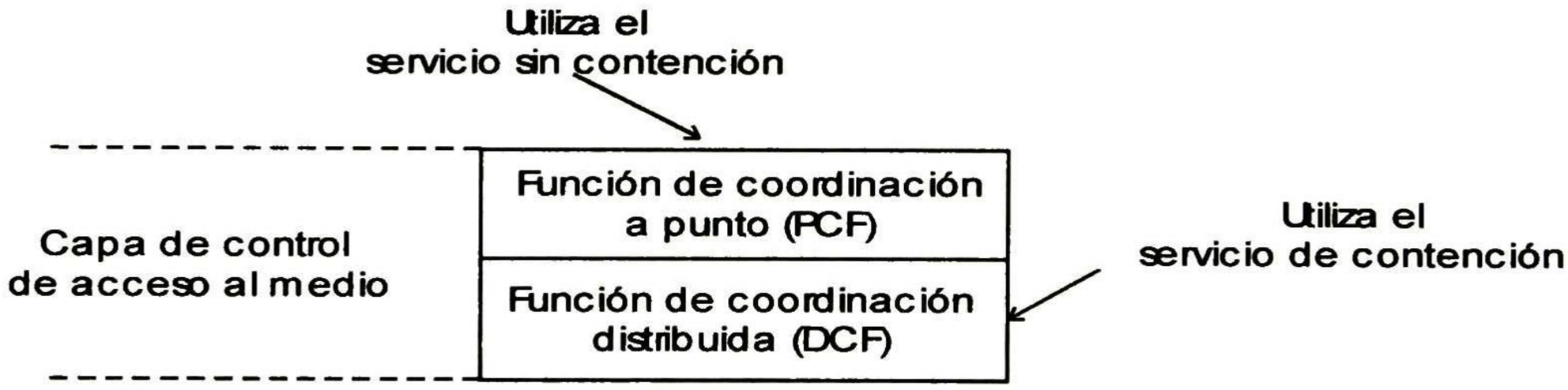


Figura B.12 Arquitectura MAC.

El servicio de contención implica que cada estación con una unidad de datos MAC en cola para transmitir deberá de competir para el acceso al canal y, una vez que la unidad de datos MAC es transmitida, la estación deberá de volver a competir por el acceso del canal para marcos posteriores. El servicio de contención asegura el acceso al canal para todas las estaciones.

La función de coordinación distribuida está basada en el acceso múltiple por detección de portadora evitando colisiones (CSMA/CA). El esquema CSMA/CD (detección de colisión) no se usa debido a que la estación no es capaz de escuchar el canal de colisión mientras transmite. En el estándar IEEE 802.11, la detección de portadora es realizada tanto en la interfase aérea, referida como detección de portadora física, como en la subcapa MAC, referida como detección de portadora virtual. En la detección de portadora física se observa la presencia de otros usuarios de WLAN IEEE 802.11 mediante el análisis de todos los paquetes detectados, y también detectando la actividad en el canal mediante la relación de la fuerza de la señal (*relative signal strength*)de otras fuentes.

Una estación fuente efectúa la detección de portadora virtual enviando información de la duración de una unidad de datos del protocolo MAC, en el encabezado del marco del tipo solicitud de envió, listo para enviar y los marcos de datos. La estación en un servicio básico fijo utiliza la información del campo *duración* para ajustar el vector de distribución en la red (NAV). Este vector indica la cantidad de tiempo que debería transcurrir hasta que la sección de transmisión actual esté completa, y el canal pueda ser muestreado otra vez para un estado desocupado. El canal es marcado

como ocupado si alguno de los mecanismos de detección de portadora, física o virtual, indica que el canal está ocupado.

La prioridad de acceso al medio inalámbrico se controla a través del uso de los intervalos de tiempo de espacio inter-marco (IFS) que existe entre la transmisión del marco. El intervalo inter-marco es un periodo obligatorio de tiempo desocupado en el medio de transmisión. Existen 3 intervalos inter-marco que están especificados en los estándares, el intervalo más pequeño es el SIFS (*Small IFS*), seguido por el PIFS (*PCF-IFS*) y el DIFS (*DCF-IFS*), respectivamente. Las estaciones que sólo requieren de esperar un inter-marco pequeño tienen prioridad de acceso sobre aquellas estaciones que requieren de esperar un inter-marco de punto o distribuido antes de transmitir. En el método de acceso básico, cuando una estación detecta que el canal está desocupado, la estación espera durante un periodo DIFS y checa el canal de nuevo, si el canal continua desocupado, la estación transmite una unidad de datos del protocolo MAC (paquete). La estación receptora calcula la suma de comprobación y determina si el paquete ha sido recibido correctamente. Después de determinar que el paquete está correcto, la estación receptora espera un intervalo inte-marco pequeño y transmite un marco de acuse de recibo positivo de regreso a la estación fuente, indicando que la transmisión fue exitosa. En la Figura B.13 se muestra un diagrama de tiempo. Cuando se transmite el marco de datos (paquete), el campo de *duración* se utiliza para que todas las estaciones del servicio básico fijo conozcan por cuanto tiempo permanecerá ocupado el canal. Todas las estaciones recibirán el marco de datos y ajustarán su vector de distribución al valor del campo *duración*, el cual incluye el intervalo inter-marco pequeño y el acuse siguiente del marco de datos.

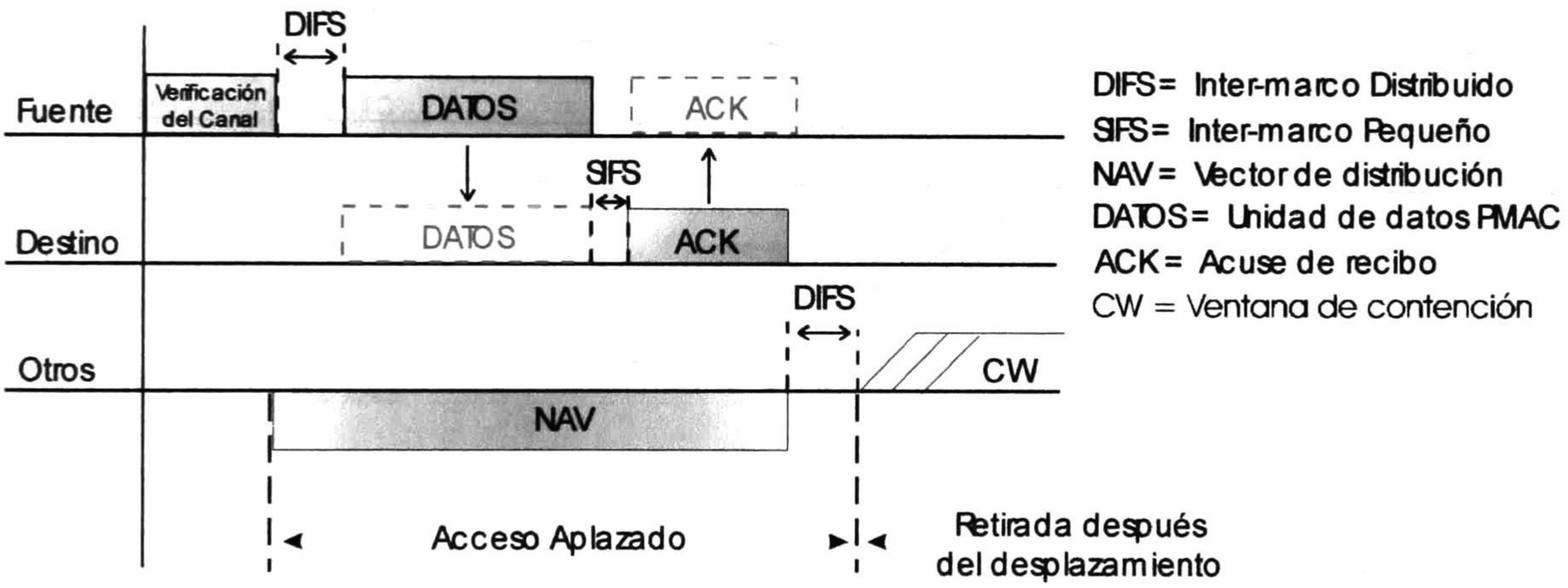


Figura B.13 Transmisión de datos con el protocolo MAC sin el mecanismo RTS/CTS

Dado que una estación fuente del servicio básico fijo no puede escuchar su propia transmisión, cuando una colisión ocurre, la fuente continua transmitiendo la unidad de datos del protocolo MAC completo. Si la unidad de datos es grande (ej., 2300 octetos), gran parte del ancho de banda del canal es desperdiciado debido al daño de dicha unidad de datos. El marco de control para la "solicitud de envío" y "listo para enviar" pueden ser usados para reservar el canal y tener prioridad para la transmisión de un unidad de datos del protocolo MAC. De esta forma se minimiza la cantidad de desperdicio del ancho de banda cuando ocurra una colisión. El marco de control RTS y CTS es relativamente pequeño (el marco de solicitud de envío es de 20 octetos y el marco listo para enviar es de 14 octetos) comparado con el máximo marco de datos (2346 octetos). El marco de control "solicitud de envío" es transmitido primero por la estación fuente, después de una exitosa competición por el canal, con un dato o marco de manejo de cola para transmisiones a una específica

estación destino. Todas las estaciones del servicio básico fijo, escuchan el paquete RTS, interpretan el campo *duración* y ajustan su vector de distribución. La estación destino responde al paquete solicitud de envío con un paquete listo para enviar después de transcurrir un periodo inter-marco pequeño. Las estaciones escuchan el paquete CTS, observan el campo de duración y ajustan de nuevo el vector de distribución. Una vez que la recepción del paquete CTS fue exitoso, la estación fuente está virtualmente segura de que el medio es estable y está reservado para una transmisión exitosa la unidad de datos del protocolo MAC. Las estaciones son capaces de actualizar su vector de distribución con el paquete RTS de la estación fuente y el paquete CTS de la estación destino, lo cual ayuda a combatir el problema de la "terminal oculta". En la Figura B.14 se muestra la transmisión de una unidad de datos del protocolo MAC utilizando el mecanismo RTS/CTS. Las estaciones pueden elegir nunca usar el mecanismo RTS/CTS, o usar el RTS/CTS cuando la unidades de datos MAC exceda el valor umbral del paquete solicitud de envío (parámetro controlable), o siempre utilizar el RTS/CTS. Si ocurre una colisión con un paquete solicitud de envío o listo para enviar, el ancho de banda desperdiciado es mucho menor comparado con un dato grande como la unidad de datos del protocolo MAC.

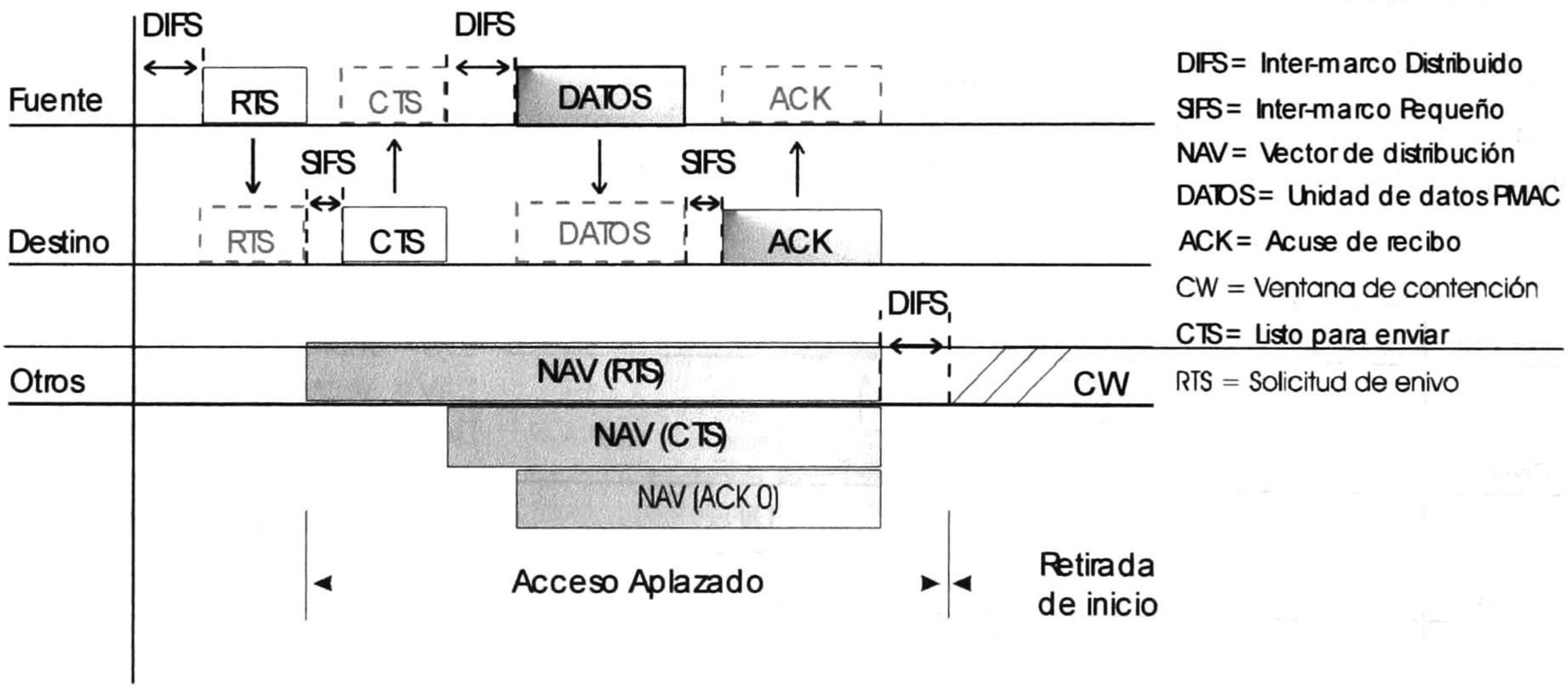


Figura B.14 Transmisión de datos con el protocolo MAC usando el modo RTS/CTS.

Las unidades de datos MAC, de gran tamaño, transmitidas de la capa de control de enlace lógico a la subcapa de control de acceso al medio, son fragmentados para incrementar la confiabilidad de la transmisión. Para determinar si se realiza la fragmentación, la unidad de datos del protocolo MAC es comparado con el parámetro controlable del umbral de fragmentación. Si el tamaño de la unidad de datos MAC excede el valor de umbral de fragmentación, la unidades de datos MAC es separada en múltiples fragmentos. El resultado son varias unidades de datos del protocolo MAC con el tamaño del umbral de fragmentación, con la excepción de la última unidad de datos, el cual tiene un tamaño variable que no excede el umbral de fragmentación. Cuando la unidad de datos es fragmentada, todo los fragmentos son transmitidos secuencialmente como se observa en la Figura B.15.

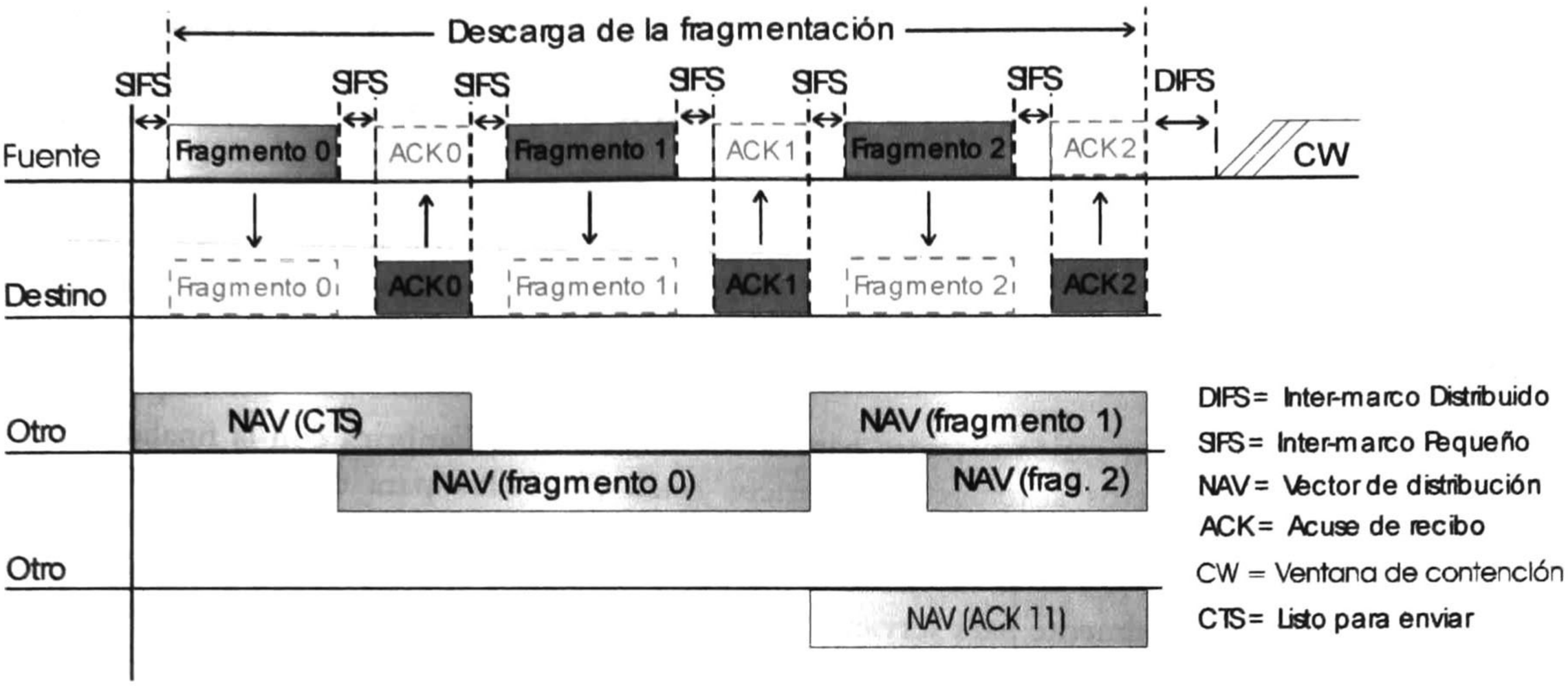


Figura B.15 Transmisión de datos con el protocolo MAC fragmentado.

El canal no queda libre hasta que la unidades de datos MAC ha sido transmitida exitosamente, o en el caso que finalice su tiempo de espera y suspenda la recepción del acuse. La estación destino manda un paquete de acuse de recibo positivo a la estación fuente, sólo cuando el fragmento fue recibido exitosamente. La estación fuente mantiene el control del canal durante toda la transmisión de la unidades de datos MAC por medio de la espera de un sólo periodo inter-marco pequeño después de recibir un acuse positivo y transmitir el siguiente fragmento. Cuando un ACK no es recibido después de haber transmitido un marco, la estación fuente interrumpe la transmisión y vuelve a competir por el canal. Al obtener el acceso al canal, la fuente inicia la transmisión con el último fragmento no recibido por la estación destino.

B.4 BIBLIOGRAFÍAS

[1] Brain P. Crow, Indra Widjaja, "IEEE 802.11 WIRELESS LOCAL AREA NETWORKS"

[2] http://www.ai.ldc.usb.ve/~figueira/Cursos/redes2/EXPO-m01/Bluetooth/bluetooth.htm

[3] Bluetooth, "Specification of the Bluetooth System", Specification vol 1.0 B, December 1st 1999.

[4] Jennifer B., Charles F. S., "Bluetooth Connect Without Cables", Prentice Hall, 2001.

[5] LAN MAN Standards Committee of the Computer Society, "INTERNATIONAL STANDARD ISO/IEC 8802-11, ANSI/IEEE Std 802.11", Part 11: Wireless LAN Medium Access Control and Physical Layer specifications, Frist Edition 1999-08-20.

APÉNDICE C

C. DESCRIPCIÓN DE SISTEMAS PARA LA COMUNICACIÓN INALÁMBRICA

C.1 INTRODUCCIÓN

En este apéndice se presentan varios sistemas para la comunicación inalámbrica con la finalidad de evaluarlos y compararlos. Estos sistemas inalámbricos están diseñados para operar en la banda Industrial-Científica-Médica de libre acceso. Las tres bandas ISM son:

Banda 1. 902 – 928 MHz (26 MHz de ancho de banda)
Usado principalmente para servicios de paginación.

Banda 2. 2.4 – 2.4835 GHz (ancho de banda de 83.5 MHz)
Usado para la transmisión de voz, datos y video;

Banda 3. 5.725 – 5.85 GHz (con un ancho de banda de 125 MHz)
Utilizado para la transmisión (microondas) de video y voz a altas velocidades en LAN.

Los sistemas descritos en este tema son:

- Sistema de desarrollo inalámbrico 2.0 (opera en la banda 2).
- Kit de desarrollo Bluetooth 2.0 (opera en la banda 2).
- Sistema 3COM para redes inalámbricas (opera en las bandas 2 y 3).
- Sistema de evaluación y desarrollo LINK (opera en la banda 1).

Dichos sistemas de comunicación inalámbrica presentan las siguientes características:

- Libertad de movimiento para el usuario.
- Enlace inalámbrico con otros dispositivos.
- Servicios equivalentes a los ofrecidos por las redes fijas.
- Integración entre las redes fijas y las móviles.
- Bajo costo, volumen y consumo de potencia.

C.2 SISTEMA DE DESARROLLO INALAMBRICO 2.0

C.2.1 Introducción

El sistema de desarrollo inalámbrico 2.0, de la empresa Silicon Wave, se utiliza para transmisiones digitales bajo el estándar Bluetooth, versión 1.0B. El sistema contiene: un radio modulador-demodulador, un controlador de enlace, un microcontrolador, y el *software* del protocolo Bluetooth (ver Figura C.1).

Dicho sistema puede establecer una comunicación con una computadora anfitrión a través de un conector universal asíncrono transmisor/receptor o bus serial universal, y con la finalidad de enviar los comandos al controlador de enlace y al radio MODEM.

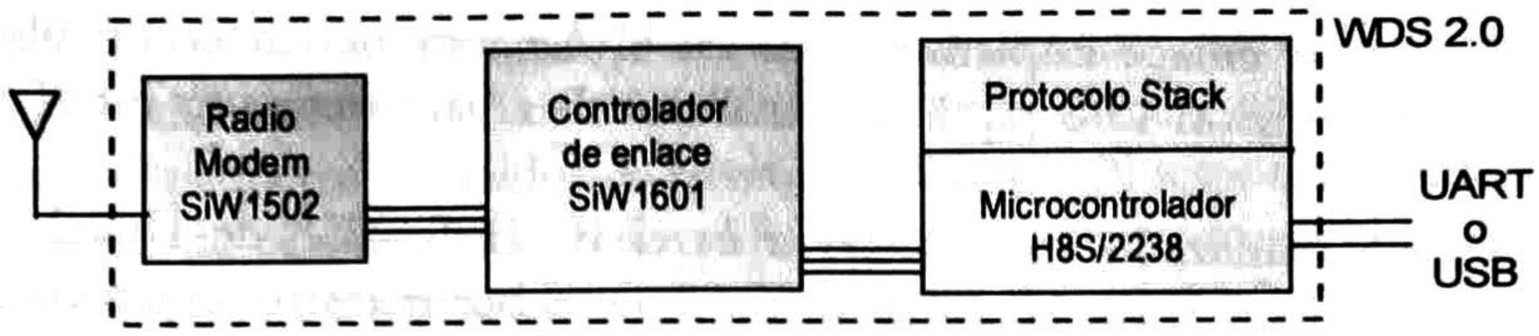


Figura C.1 Diagrama a bloques del Sistema de Desarrollo Inalámbrico WDS 2.0.

C.2.2 Dispositivos del WDS 2.0

El Radio-MODEM SiW1502 combina un *transceiver* a 2.4 GHz y un MODEM GFSK con funciones de control digital bajo las especificaciones Bluetooth. El *transceiver* se encarga de filtrar la señal y de trasladar la frecuencia según sea el caso mediante una estructura heterodina, además transmite con una potencia de 0 dBm y tiene un rango de alcance de 10 metros. El MODEM GFSK realiza la modulación-demodulación digital, filtra el canal y recupera la sincronía del símbolo [1]. El radio y el MODEM están desarrollados en un sólo circuito mixto (analógicos para RF y digitales para las funciones del MODEM). En la Figura C.2 se muestra un diagrama a bloques del CI SiW1502.

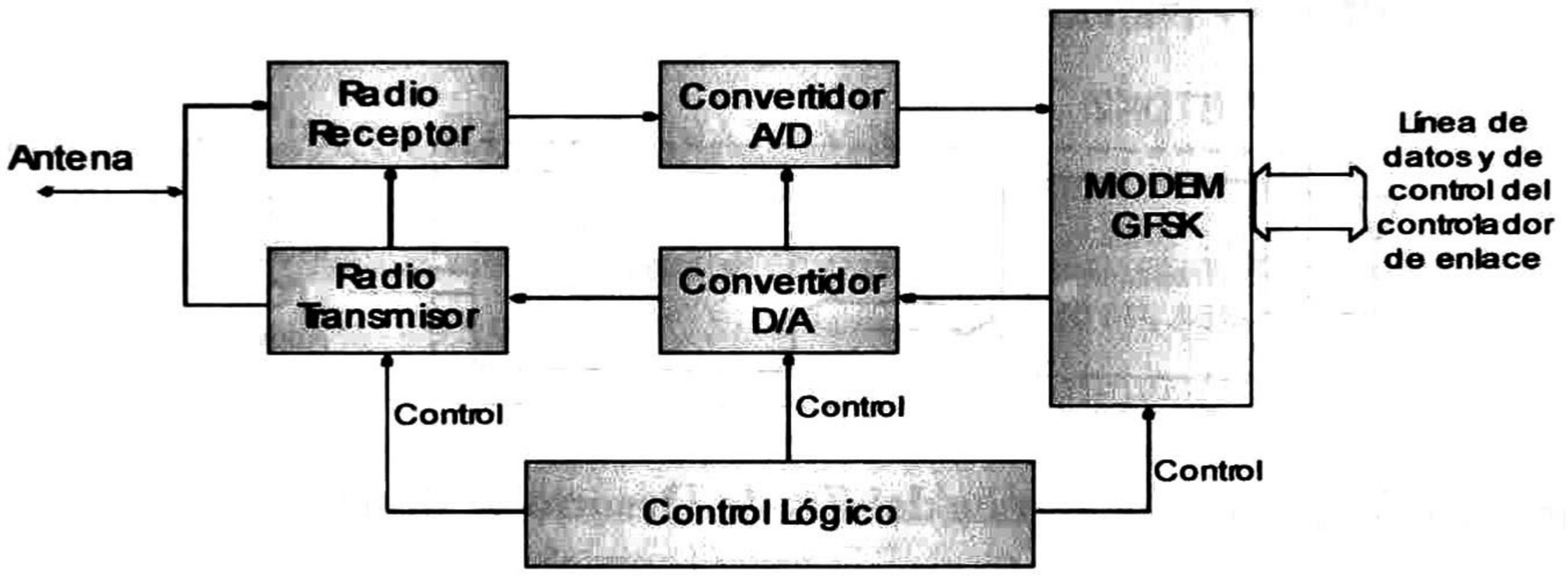


Figura C.2 Diagrama funcional a bloques del Radio MODEM SiW1502.

El controlador de enlace SiW1601 se encarga del manejo de la capa banda base Bluetooth. Por lo que, el dispositivo procesa las transmisiones y recepciones de datos a nivel *hardware*, como se muestra en el diagrama a bloques de la Figura C.3. También cuenta con funciones de control realizadas a nivel *hardware* mediante un circuito integrado de aplicación específica. Este *hardware* realiza el procesamiento del protocolo lógico dentro de la unidad y permite que el anfitrión se comunique sobre un enlace Bluetooth.

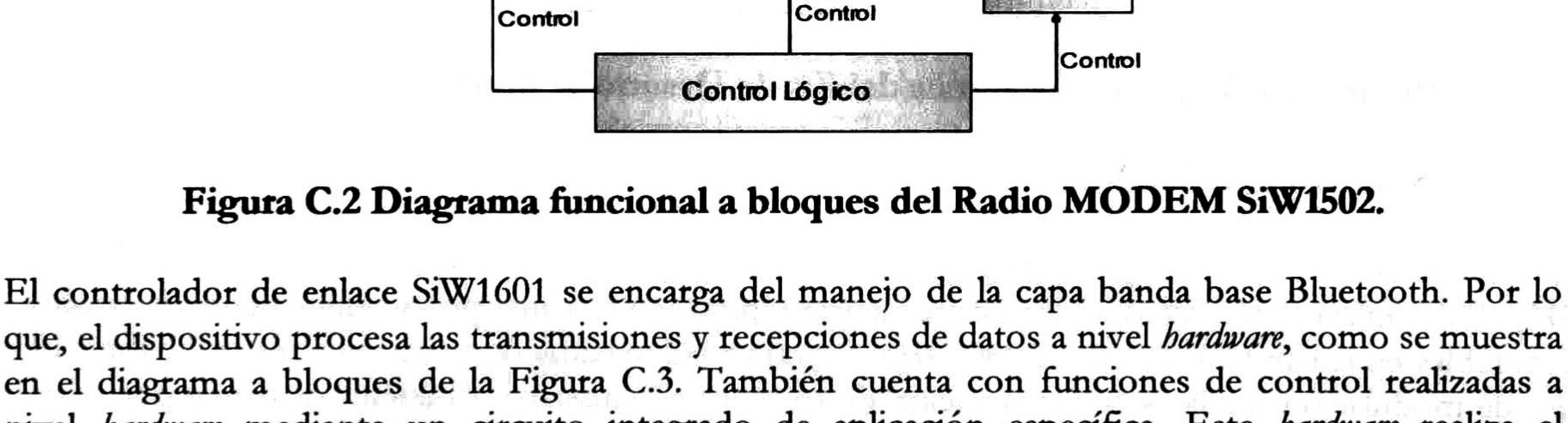


Figura C.3 Diagrama a bloques del Controlador de Enlace SiW1601.

Las funciones del control de enlace implementadas en el *hardware* tienen compatibilidad con las especificaciones Bluetooth, versión 1.0B [2]. El controlador de enlace tiene una interfase directa con el radio MODEM.

El microcontrolador utilizado en el sistema es el IC H8S/2238 de Hitachi que contiene básicamente: un controlador para la transferencia de datos en el bus maestro, una memoria flash (con capacidad de 256 Kbytes), una memoria RAM (capacidad 16 Kbytes), también cuenta con una interfase de comunicación serial y además, éste contiene el *software* del protocolo a nivel HCI preinstalado en la memoria flash [3].

C.3 KIT DE DESARROLLO BLUETOOTH 2.0

C.3.1 Introducción

El *kit* BTDK2 es un sistema diseñado por Ericsson para la transmisión de voz, datos y video. Cada tarjeta incluye: un radio *transceiver*, un dispositivo controlador banda base, componentes discretos y el *software* del protocolo *stack* Bluetooth grabado en el controlador (ver Figura C.4). La tarjeta contiene los siguientes puertos para la comunicación con el anfitrión: 3 puertos RS-232, un puerto USB, un puerto I^2C y una salida de audio [4].

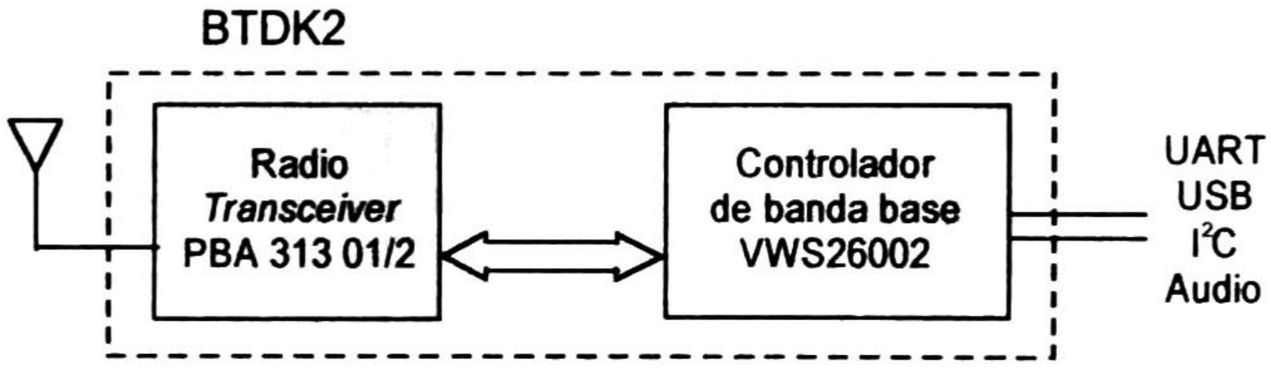


Figura C.4 Diagrama a bloques del Kit de Desarrollo Bluetooth BTDK 2.0.

C.3.2 Dispositivos del BTDK2

El radio *transceiver* PBA 313 01/2 está diseñado para operar en la banda de frecuencias ISM, 2.4 – 2.5 GHz. Este utiliza la técnica de salto de frecuencia rápido (1600 saltos por segundo) con 79 canales disponibles en la banda de 2.402 – 2.480 GHz. La velocidad máxima de transmisión es de 1 Mbps. El tipo de modulación usada es la conmutación de frecuencia Gaussiana. El ancho de banda por canal es de 1 MHz y la desviación de la frecuencia portadora está entre los 140 y 175 KHz. La potencia de salida TX típica es de 1.5 dBm y máxima de 4 dBm. El radio Bluetooth está constituido básicamente de un ASIC (contiene un transmisor y un receptor de estructura heterodina con una frecuencia intermedia baja de 3 MHz [5]), de un filtro para la antena y de una circuitería para realizar la conexión de las líneas de transmisión RX y TX con la antena; ver Figura C.5.

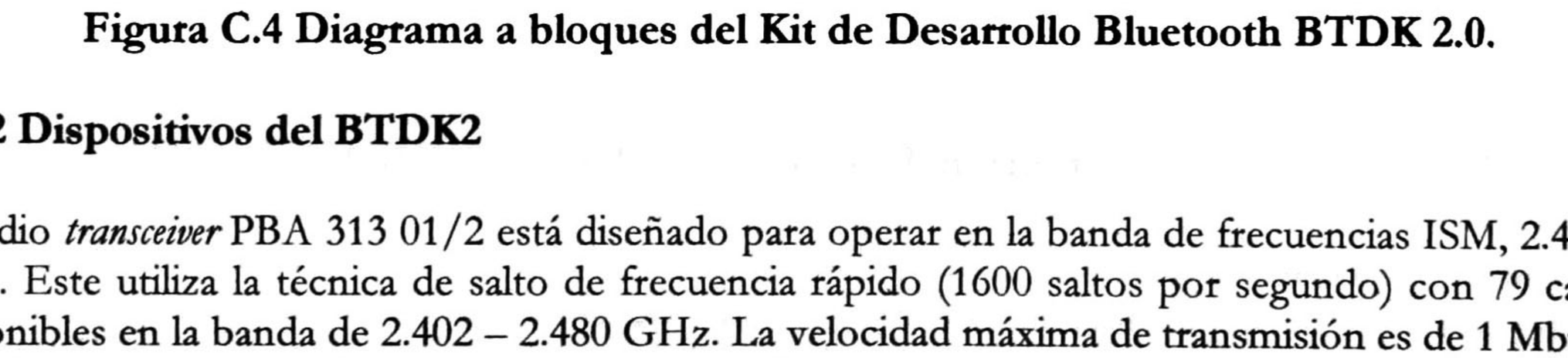


Figura C.5 Diagrama a bloques de la arquitectura del radio Bluetooth PBA 313 01/2.

El controlador VWS26002 es el encargado de controlar la capa del protocolo Bluetooth [6]. Este se encarga de la corrección de error, cifrado, aleatorización, prueba de redundancia cíclica y la autentificación de los paquetes recibidos y enviados al radio Bluetooth (Figura C.6). Además, el dispositivo tiene integrado el microprocesador ARM7TDMI para procesar el *software* del protocolo a nivel HCI. El VWS26002 es perfectamente compatible con el módulo de radio Bluetooth por lo que se reducen considerablemente los riesgos y tiempos de acceso.

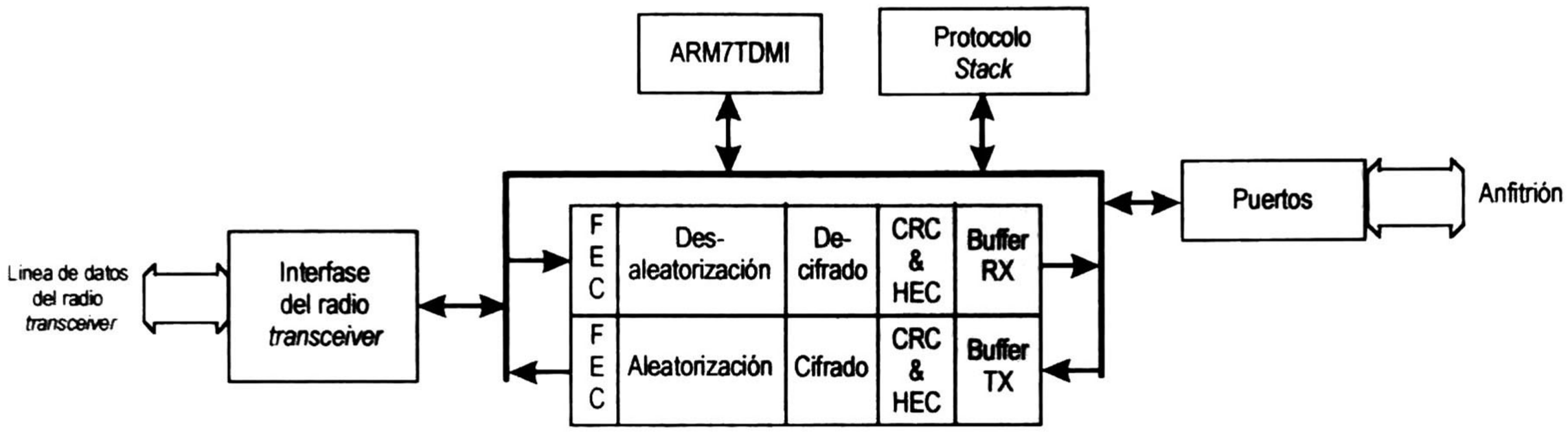


Figura C.6 Diagrama a bloques del Controlador VWS26002.

C.4 SISTEMA 3COM PARA REDES INALAMBRICAS

La empresa 3COM desarrolló un sistema completo para la comunicación de redes inalámbricas bajo el estándar IEEE 802.11b descrito en el capítulo anterior. El sistema está constituido básicamente de un punto de acceso y tarjetas de conexión inalámbrica para computadoras portátiles y de escritorio. Esto es el equivalente a un servicio básico fijo, siempre y cuando las estaciones permanezcan dentro del área del servicio básico, también conocido como red *ad hoc*. Los puntos de acceso *Wireless LAN* se conectan directamente la red cableada, ver Figura C.7.

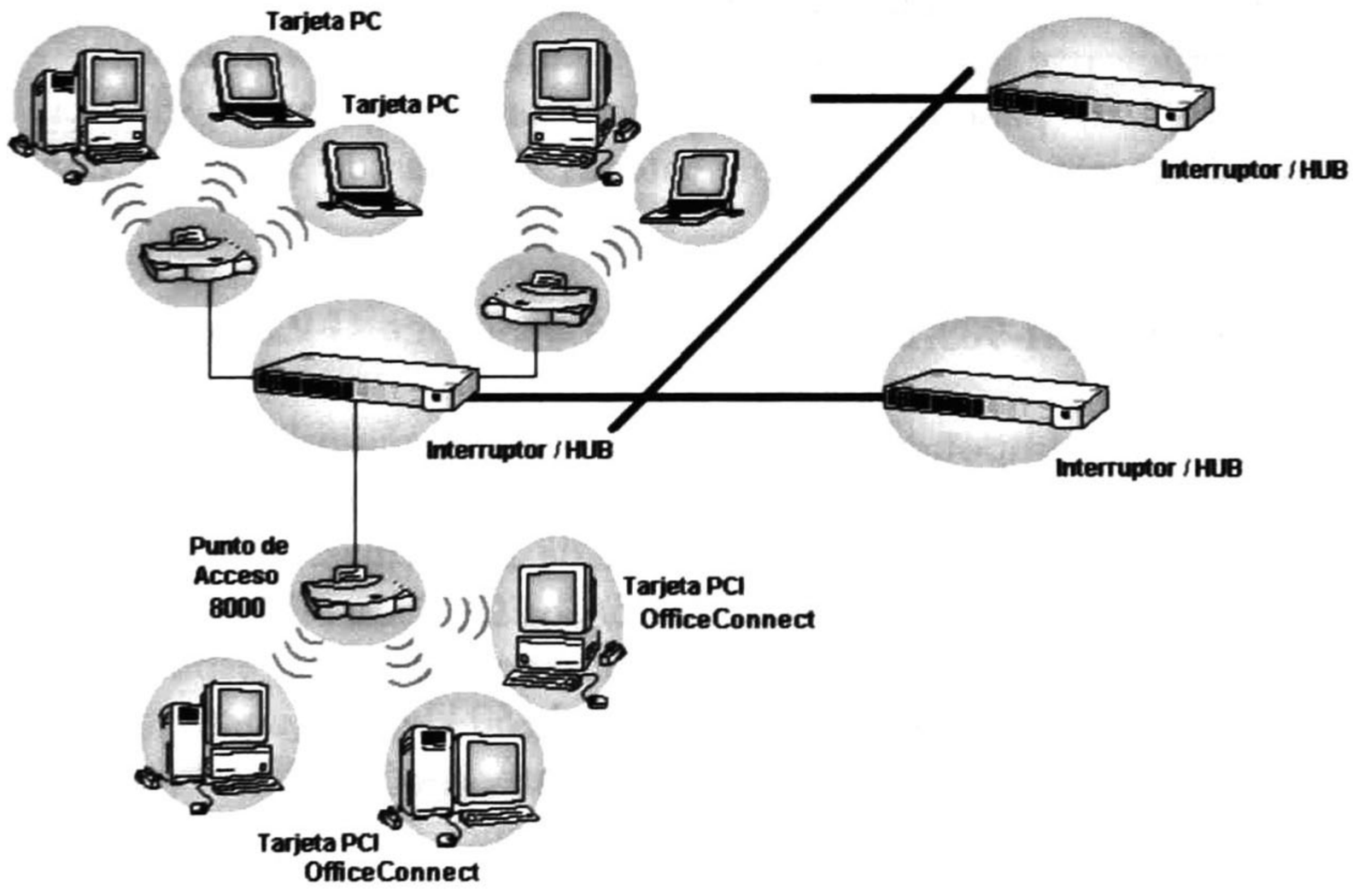


Figura C.7 Red inalámbrica con el estándar IEEE 802.11b.

C.4.1 Punto de Acceso 8000 *Wireless LAN*

El punto de acceso 8000 3COM a 11Mbps para redes inalámbricas, ofrece: seguridad, conectividad flexible y la posibilidad de expandirse, ya que maneja múltiples conexiones simultáneas. Se integra fácilmente con la infraestructura tradicional porque soporta múltiples normas y aplicaciones de gestión de red. El punto de acceso tiene una cobertura inalámbrica hasta 100 metros.

El dispositivo contiene las características de seguridad más avanzadas del mercado, éste soporta la autentificación IEEE 802.1x; el cifrado compartido de 40 bits y 128 bits bajo la norma de privacidad equivalente a la alambrada. También contiene un sistema que le permite cambiar su velocidad de transmisión en función a las condiciones del entorno para garantizar las más rápidas conexiones posibles, ésto es conocido como cambio de velocidad dinámica. Además, el dispositivo soporta hasta 256 usuarios simultáneos, su velocidad de transmisión varía de 11, 5.5, 2 y 1 Mbps dependiendo de la distancia y las condiciones de la red de cables [7].

C.4.2 OfficeConnect 11 Mbps Wireless LAN PC Cards

Esta tarjeta que va conectada al equipo del usuario ofrece las mismas características que el punto de acceso en cuanto a codificación, conectividad y posibilidad de expandirse. Es fácil de instalar y su cobertura es de 100 metros. La velocidad de transmisión varía de 11, 5.5, 2 y 1 Mbps. La tarjeta soporta las mismas normas y estándares que el dispositivo punto de acceso [8]. Así que ésta se conecta con el punto de acceso para proporcionar el servicio de la red. Las tarjetas se instalan en computadoras portátiles y de escritorio.

C.5 SISTEMA DE EVALUACIÓN Y DESARROLLO LINX

C.5.1 Introducción

El sistema MEDV-900-HP está conformado de una tarjeta que contiene un módulo de transmisión RF, un módulo de recepción RF y una interfase de comunicación serial bajo el estándar RS-232. Este ultimo se utiliza para comunicarse con una computadora anfitrión a través del UART. El sistema se puede utilizar junto con otro para establecer una comunicación inalámbrica enviando señales analógicas o digitales. El sistema se alimenta con baterías alcalinas de 9 voltios, con la finalidad de que sea transportable.

C.5.2 Módulo Transmisor TXM-900-HP-II

El transmisor es diseñado con componentes de bajo costo y de alto rendimiento para la transferencia inalámbrica de señales analógicas o digitales, en la banda ISM (902 – 928 MHz). El transmisor emplea una modulación de conmutación o desviación de frecuencia. La modulación FSK ofrece ventajas significativas sobre los métodos de modulación banda base AM, como el incremento a la inmunidad al ruido y la capacidad del receptor para obtener una señal especifica en presencia de múltiples señales. La máxima velocidad de transmisión es de 50 kbps. El ancho de banda por canal es de 2 MHz y la desviación de la frecuencia portadora está entre los 60 y 95 kHz. La potencia de salida mínima es de –3 dBm y máxima de 4 dBm. Bajo condiciones óptimas se puede transmitir a una distancia máxima de 300 metros.

El transmisor cuenta con 8 canales que son seleccionados mediante interruptores por el usuario. La frecuencia central del primer canal es de 903.37 MHz, y los canales subsecuentes están separados aproximadamente entre 2 y 3 MHz (el ancho de banda ocupado es de 32 MHz).

En la Figura C.8 se muestra el diagrama a bloques del transmisor. La modulación se realiza con un oscilador de cristal controlado por voltaje a una frecuencia baja. En el siguiente bloque la señal modulada incrementa su frecuencia al orden de los 900 MHz, por medio de un sintetizador de frecuencia [9].

En el caso de que existan varios transmisores dentro del área de enlace deberán de ser activados en canales separados, lo suficientemente retirados como para que una señal del canal inmediato no entre al receptor a un nivel que sobrepase la capacidad de rechazo del receptor.

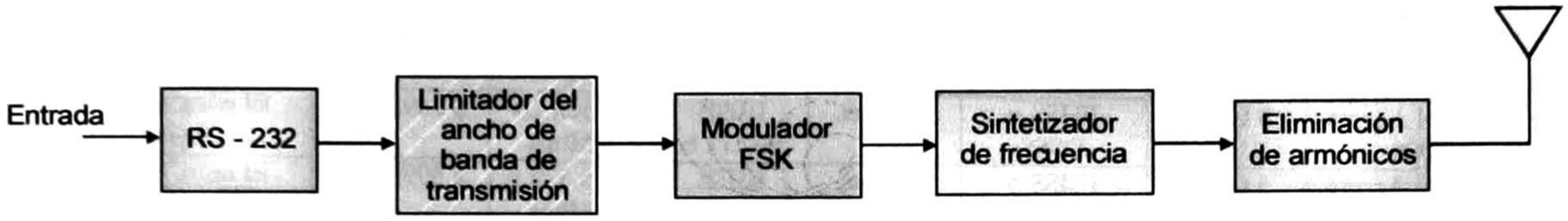


Figura C.8 Diagrama a bloques del transmisor HP serie II.

C.5.3 Módulo Receptor RXM-900-HP-II

El receptor básicamente es una estructura superheterodina de doble conversión, diseñada con componentes de bajo costo y de alto rendimiento para la recepción inalámbrica de señales analógicas o digitales, en la banda popular ISM (902 – 928 MHz). La demodulación se realiza con el método de conmutación o desviación de frecuencia.

En la Figura C.9 se muestra el diagrama a bloques del receptor. La señal RF que llega del exterior a través de la antena, se procesa en el bloque de la estructura superheterodina de doble conversión trasladando la frecuencia original a una frecuencia intermedia. En el bloque demodulador entrega una la señal analógica con un rango de 0 – 25 kHz [10].

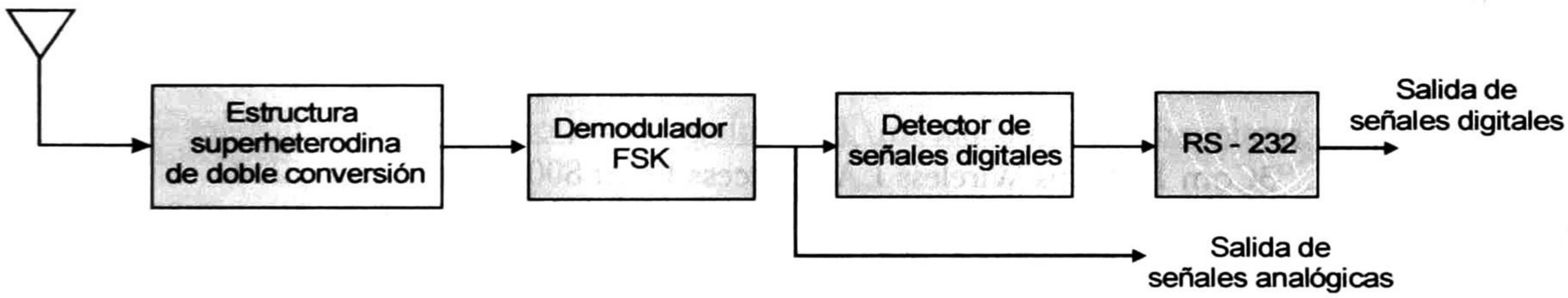


Figura C.9 Diagrama a bloques del receptor HP serie II.

El sistema está originalmente diseñado para manejar una sola vía de comunicación (simplex). Para poder utilizar la forma full-duplex se requieren dos pares de módulos trabajando en canales distintos.

C.6 SELECCIÓN DEL SISTEMA DE COMUNICACIÓN INALÁMBRICA

La Tabla C.1 nos permite comparar los sistemas antes expuestos de acuerdo con las características básicas requeridas en el sistema de comunicación, decidiendo que:

El sistema Bluetooth queda descartado por su alcance limitado de 10 mts.

Desde el punto de vista del ancho de banda, los sistemas basados en el protocolo IEEE 802.11 son ideales para nuestra aplicación, sin embargo, el costo por unidad es elevado.

Por último el sistema LINX 900-HP tiene la ventaja de que los módulos de transmisión y recepción son económicos. Su ancho de banda de 50 kbps es suficiente para manejar la cantidad de información que se puede transmitir entre un autobús y una base durante el tiempo de espera en la terminal.

Sistema	Fabricante	Modulación	Alcance	Protocolo	Vel. Trans.	Precio	Nota
Wireless Development System 2.0	Silicon Wave	GFSK	10 mts.	Bluetooth 1.0	1 Mbps		
Kit Development Bluetooth 2.0	Ericsson	GFKS	10 mts.	Bluetooth 1.0	1 Mbps	$ 8,500 USD	En la compra de otro $ 7,000 USD
Wireless LAN Access Point 8000	3 COM	FHSS o DSSS	100 mts.	IEEE 802.11 b	11, 5.5, 2 y 1 Mbps	$ 749 USD	El núm. máx. de usuarios es 256.
Wireless LAN Access Point 6000	3 COM	FHSS o DSSS	100 mts.	IEEE 802.11 b	11, 5.5, 2 y 1 Mbps	$ 599 USD	El núm. máx. de usuarios es 65.
Office Connect WLAN PC Card	3 COM	FHSS o DSSS	100 mts.	IEEE 802.11 b	11, 5.5, 2 y 1 Mbps	$ 139 USD	
Development Kit MED-900-HP	LINX	FSK	300 mts.		50 kbps	$ 248 USD	Incluye: dos módulos TXM y dos RXM.
TXM-900-HP-II	LINX	FSK	300 mts.		50 kbps	$ 21 USD	
RXM-900-HP-II	LINX	FSK	300 mts.		50 kbps	$ 31 USD	

Tabla C.1 Tabla de comparación con los diferentes sistemas de comunicación inalámbrica.

C.7 BIBLIOGRAFÍA

[1] Silicon Wave, Inc., "SiW 1502 Data Sheet", Preliminary Information, October 10, 2000.
[2] Silicon Wave, Inc., "SiW1601 Data Sheet", Preliminary Information, October 11, 2000.
[3] Silicon Wave, Inc., "Wireless Development System Product Summary", Preliminary Information, June 8, 2000.
[4] http://bluetooth.ericsson.com
[5] Ericson Microelectronics AB, Inc., "PBA 313 01/2 Bluetooth Radio Data Sheet", Preliminary, April 2000.
[6] http://www.semiconductors.philipscom/Technology/bluetooth/systems/controller
[7] 3COM, Inc., "3Com 11 Mbps Wireless LAN Access Point 8000 Data Sheet", 2002.
[8] 3COM, Inc., "3Com OfficeConnect 11 Mbps Wireless LAN PC Cards Data Sheet", 2002.
[9] Linx Technologies, Inc., "HP Series-II Transmitter Module Design Guide Description", 1999.
[10] Linx Technologies, Inc., "HP Series-II Receiver Module Design Guide Description", 1999.

APÉNDICE D

D. PROGRAMA OPERATIVO DEL MAESTRO

D.1 MAIN.CPP

```
//---------------------------------------------------------------------------
//Programa Principal (main.cpp)
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "main.h"
#include "commsettings.h"
#include <memory>
#include <fstream.h>
#include "SerialPort.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TMainForm *MainForm;
//---------------------------------------------------------------------------
//Capacidad del puerto.
const int TextStreamCapacity   = 64*1024;
// Creacion del buffer para enviar y recibir datos
__fastcall TMainForm::TMainForm(TComponent* Owner)
   : TForm(Owner)
{
   m_BufferIn.reserve(TextStreamCapacity);
   // Se trabaja con un hilo llamado SerialPort
   SerialPort = new TSerialPort();
   // Se trabaja con un hilo llamado Protocolo
   Protocolo = new TProtocolo();

   StatusBar1->DoubleBuffered = true;
   UpdateStatusBar();
   m_BufferInLength = 0;
   BytesReceived = 0;
   Finished = false;
}
//---------------------------------------------------------------------------
__fastcall TMainForm::~TMainForm()
{
   // Si Protocolo está ejecutándose lo detiene y espera a que
termine.
   if(!Protocolo->Suspended)
   {
      Protocolo->Terminate();
      Protocolo->WaitFor();
   }
  delete Protocolo;
}
//---------------------------------------------------------------------------
// Codigo para elegir el puerto de transmision y para elegir la tasa de baudios
void __fastcall TMainForm::actCommSettingsExecute(TObject
*Sender)
{
   std::auto_ptr<TCommSettingsForm> form(new
TCommSettingsForm(NULL));
   form->CommPort = SerialPort->GetCommPort().GetCommPort();
   form->BaudRate = SerialPort->GetCommPort().GetBaudRate();
   if(form->ShowModal())
   {
      bool bConnected = SerialPort->GetConnected();
      SerialPort->SetParameters(form->BaudRate, form->CommPort);
      if(bConnected)
         SerialPort->Connect();
   }
   UpdateStatusBar();
}
//---------------------------------------------------------------------------
//Conexion
void __fastcall TMainForm::actConnectExecute(TObject *Sender)
{
   SerialPort->Connect();
   UpdateStatusBar();
   if (SerialPort->GetConnected())
      Protocolo->Resume();
   Button2->Enabled = false;
   Connect1->Enabled = false;
   Settings1->Enabled = false;
}
//---------------------------------------------------------------------------
//Desconexion
void __fastcall TMainForm::actDisconnectExecute(TObject *Sender)
{
   if (!Protocolo->Suspended)
      Protocolo->Suspended = true;
   SerialPort->Disconnect();
   UpdateStatusBar();
   Button2->Enabled = true;
   Connect1->Enabled = true;
   Settings1->Enabled = true;
   Edit1->Text="Inactivo";
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::actConnectUpdate(TObject *Sender)
{
   actConnect->Enabled = !SerialPort->GetConnected();
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::actDisconnectUpdate(TObject *Sender)
{
   actDisconnect->Enabled = SerialPort->GetConnected();
}
//---------------------------------------------------------------------------
//Exit
void __fastcall TMainForm::Exit1Click(TObject *Sender)
{
   Close();
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::StatusBar1Resize(TObject *Sender)
{
   StatusBar1->Panels->Items[0]->Width = StatusBar1->Width - 225;
}
//---------------------------------------------------------------------------
//Actualiza la barra de estado
void TMainForm::UpdateStatusBar()
{
   AnsiString str, str1;
   str = "Estado: ";
   if(SerialPort->GetConnected())
      str += "Conectado       ";
   else
      str += "Desconectado    ";
   str += "Baudios = " + IntToStr(SerialPort-
>GetCommPort().GetBaudRate());
   str += "    Puerto = ";
   str += SerialPort->GetCommPort().GetCommPort().c_str();
   str1 +="    Datos recibidos = " + AnsiString(BytesReceived) +"
bytes";
```

```
   StatusBar1->Panels->Items[0]->Text = str;
   StatusBar1->Panels->Items[1]->Text = str1;
}
//---------------------------------------------------------------------
//Recibo de una trama de la capa de red
void TMainForm::FromDataLinkLayer(packet *p, unsigned int count, bool last)
{
 unsigned int contador = 0;
 while (contador < MAX_PKT-2 &&  contador < count)
 {
  m_BufferIn.insert(m_BufferIn.end(), p->data[contador]);
  contador++;
  BytesReceived++;
 }
 Protocolo->Fin = false;
 if(last){
     AnsiString cad;
     AnsiString Str("\n Transmisión finalizada \n Escribiendo archivo JPG
\n");
     cad = "D:\\Pruebas\\Ar" + IntToStr(Protocolo->direc)+
IntToStr(Protocolo->cont1) + ".txt";//".jpg";
     strcpy(Name,cad.c_str());
     ofstream ofstr(Name, ios::binary);
     if (ofstr)
           {
            ofstr.write(m_BufferIn.begin(), BytesReceived);
            ofstr.close();
           }
     MainForm->RichEdit3->Lines->Append(Str);
     Finished = true;
     Protocolo->retransm = 13;
 }
 if (!Protocolo->Barb)
     Protocolo->Barb = true;
 UpdateStatusBar();
}
//---------------------------------------------------------------------
```

D.2 PROTOCOLO.CPP

```
#include <vcl.h>
#pragma hdrstop

#include "Protocolo.h"
#include "main.h"
#include <exptintf.hpp>
#define lista 50 //100
#pragma package(smart_init)
#define timeout 94404//555320
#define inc2(k) if (k < lista) k = k + 1; else k = 0;
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0;
#define inc3(k) if (k < 7) k = k + 1; else k = 0;
#define inc1(k) if (k < 3) k = k + 1; else k = 1;
////////////////////////////////////////////////
#include <fstream.h>
const int TextStreamCapacity   = 32*2048;
const int TextStreamCapacity2   = 32*2048;
////////////////////////////////////////////////
//---------------------------------------------------------------------
__fastcall TProtocolo::TProtocolo()
   : TThread(true)
{
   retransm = 0;
   frame_expected = 0;  /* numero de marco de entrada esperado */
   ID = 1;   //////8191
   flag1 = true;
   compt = false;
   cont1 = 0;
   Fin = false;
   Barb = false;
   tran_inc = false;
   direc = 0;
   cont2 = "s";   ///Faltan mas datos por recibir y por lo tanto el
archivo no esta completo
   e[0].bandera == false
   e[1].bandera == false;
   e[1].Buff.reserve(TextStreamCapacity2);
   e[0].Buff.reserve(TextStreamCapacity2);
   barro = false;
   MainForm->Edit2->Text = "0";
   grafarhc = false;
   //crcpck = false;
   num_arch = 1;
   algo = 0;
   Garbage();
}
//---------------------------------------------------------------------
//Destructor
__fastcall TProtocolo::~TProtocolo()
{
}
//---------------------------------------------------------------------
void __fastcall TProtocolo::Execute()
{
   while (!Terminated)
   {
      ID = 1;
      DWORD BytesAvailable;
      Sleep (800);
      MainForm->Edit1->Text = "Busqueda";
      MainForm->Edit2->Text = chale6;
      while (ID< lista)
      {
        contr = 8000;
        queondon = false;
        mandar_trama(ID,listado,0,0);
        while (contr && !queondon)
        {
          contr--;
          BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
          if(BytesAvailable > 6)  ////6
          {
            Synchronize(SynchronizeNewBytes2);
          }
        }
        inc2(ID);
        MainForm->Finished = false;
      }
      if (ID == lista)
      {
         MainForm->Edit2->Text = chale6;
      }
      if (ID == lista && reg.size() > 0)
      {
        direc = reg[0];
        MainForm->Edit1->Text = "Recibiendo";
        barro = true;
        if (algo == 350)
        {
          inc1(num_arch);
          algo = 0;
        }
        algo ++;
   }
      while (ID == lista && reg.size() > 0)
```

```
    {
      while (retransm < 17)
      {
        contr = timeout;
        if (flag1)
        {
              mandar_trama (reg[0], final,0,num_arch);
        }
        while (contr)
         {
              BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
              contr--;
              if (BytesAvailable > 80)
              {
                   Synchronize(SynchronizeNewBytes);
              }
      }
      retransm++;
    }
    if (retransm == 17)
    {
      if (reg.size() > 0)
      {
        for ( unsigned int i = 0; i < reg.size(); i++)
        {
          if (reg[i] == direc)
          {
            reg.erase(reg.begin() +i);
            break;
          }
        }
      }
      MainForm->RichEdit->Clear();
      for (unsigned int i = 0; i < reg.size(); i++)
      {
        MainForm->RichEdit->Lines->Add(IntToStr(reg[i]));
      }
      if (!Fin && MainForm->BytesReceived)
      {
            int var;
            for(int i =0; i<2;i++)
              {
                if (e[i].bandera == false)
                {
                      var = i;
                      break;
                }
            }
            e[var].ID = direc;  /// OJO AQUI VA DIREC
            e[var].tamano = MainForm->BytesReceived;
            e[var].seq = frame_expected;
            e[var].bandera = true;
            e[var].Buff.erase(e[var].Buff.begin(),e[var].Buff.end());
e[var].Buff.insert(e[var].Buff.end(), MainForm-
>m_BufferIn.begin(),MainForm->m_BufferIn.end());
                  cont2 = "s";
            }
            retransm = 0;
            flag1 = true;
            frame_expected = 0;
            MainForm->BytesReceived = 0;
            MainForm->m_BufferIn.erase(MainForm-
>m_BufferIn.begin(),MainForm->m_BufferIn.end());
            MainForm->Finished = false;
            MainForm->RichEdit3->Lines->Clear();
            Barb = false;
            tran_inc = false;
            direc = reg[0];//0;   ///Hacer archivo Buffer.clear()
        }
      }
    }
}
//---------------------------------------------------------------------------
```

```
///Validación de presentes
void __fastcall TProtocolo::SynchronizeNewBytes2(void)
{
 tipo_marco tipo;
 WORD direcc1;
 bool flag = false;
//Seguridad para cuando el programa se cierre sin haber terminado
 if(Application->Terminated /*|| MainForm->Finished*/)
     return;
  // llego mensaje del buffer
 from_physical_layer1(&R);  /////definir como hacerle cuando se
reciba en el buffer.....
  // CRC
 if (compt && Cal_CRC_162(&R) == 0)     //definir los marcos
para que sea general.....
 {
   tipo = R.direcc & 0xE000;
   direcc1 = R.direcc & 0x1FFF;
   if (tipo == presente && direcc1 < lista)
   {
     if (reg.size() > 0)
     {
       for (unsigned int i = 0; i < reg.size(); i++)
       {
         if (reg[i] == direcc1)
         {
           flag = true;
           break;
         }
       }
       if (!flag)
       {
         reg.insert(reg.end(), direcc1);
         MainForm->RichEdit->Lines->Add(IntToStr(reg[reg.size()-1]));
       }
       if (direcc1 == ID)
       {
         queondon = true;
       }
     }
     compt = false;
   }
   compt = false;
 }
 //---------------------------------------------------------------------------
 //Sincornizacion de datos
 void __fastcall TProtocolo::SynchronizeNewBytes(void)
 {
  AnsiString Str("Mensaje ");
  tipo_marco tipo;
  WORD direcc1;
 //Seguridad para cuando el programa se cierre sin haber terminado
  if(Application->Terminated /*|| MainForm->Finished*/)
     return;
 from_physical_layer(&r);
 if (compt && Cal_CRC_16(&r) == r.CRC)
 {
    tipo = r.direcc & 0xE000;
    direcc1 = r.direcc & 0x1FFF;
   if (flag1 && direc == direcc1)
   {
    if (!tran_inc)
    {
      for (int i=0;i<2;i++)
      {
          /////Tal ves sea bueno poner un contador aqui
          if (direc == e[i].ID && e[i].bandera)
          {
            MainForm->BytesReceived = e[i].tamano;
            frame_expected = e[i].seq;
            e[i].bandera = false;
```

```
                MainForm->m_BufferIn.erase(MainForm-
>m_BufferIn.begin(),MainForm->m_BufferIn.end());
 MainForm->m_BufferIn.insert(MainForm->m_BufferIn.end(),
e[i].Buff.begin(),e[i].Buff.end());
                e[i].Buff.erase(e[i].Buff.begin(),e[i].Buff.end());
                tran_inc = true;
                flag1 = false;
                break;
             }
          }
       }
    }
    if(r.seq == mens_esp && tipo == datos && direcc1 == direc)
     {
        if(chale2 == 255)
           chale3++;
        chale2++;
        Str = Str + "recibido # "+ r.seq /*+"\n"*/;
        // solo se aceptan marcos en orden
        if (!Barb)   ///////mover esto a despues de la linea siguiente
        {
           flag1 = false;
           tran_inc = true;
        }
        contr = timeout;
        retransm = 0;
        if (!MainForm->Finished)
        {
           to_network_layer(&r.info, r.cont, r.last, mens_esp, direcc1);
           if(r.last && MainForm->Finished)
           {
              Fin = true;
              cont2 = "n";
              if (grafarhc)
                 MainForm->Imagen->Picture->LoadFromFile(MainForm-
>Name);
              grafarhc = false;
             }
           }
           inc(mens_esp);
           MensajeHola(Str);
        }
        else if (tipo == datos && direcc1 == direc)
           {
             send_ack(r.seq, r.last,direcc1,ack1);
             Str = Str + "retransmitido # "+ r.seq /*+"\n"*/;
             MensajeHola(Str);
           }
        if(tipo == final && direc == direcc1)
        {
           contr=0;
           retransm = 16;
         }
         compt = false;
     }
     compt = false;
}
//---------------------------------------------------------------------
// Construccion del marco tipo listado
void TProtocolo::mandar_trama(WORD ID ,tipo_marco tipo, int b, seq_nr
num_arch)
{
 BYTE CRCS;
 marco2 s;
 s.direcc = 0x0000;
 s.seq = 0x00;
 s.CRC = 0x00;
 s.seq = num_arch*0x20;
 s.direcc = ID | tipo;          /*direccion de la unidad a localizar*/
 CRCS = Cal_CRC_162(&s);
 s.seq = s.seq ^ (CRCS / 16);
 s.CRC = CRCS * 0x10;
 to_physical_layer2(&s,b);
}
//---------------------------------------------------------------------
// Acuse
void TProtocolo::send_ack(seq_nr mens_esp, bool last,WORD ID ,tipo_marco
tipo)
{
 BYTE CRCS;
 marco2 s;
 s.direcc = 0x0000;
 s.seq = 0x00;
 s.CRC = 0x00;
 s.direcc = ID | tipo;          /*direccion de la unidad a localizar*/
 s.seq = mens_esp*0x20;
 if (last)
     s.seq = s.seq ^ 0x10;
 CRCS = Cal_CRC_162(&s);
 s.seq = s.seq ^ (CRCS / 16);
 s.CRC = CRCS * 0x10;
 to_physical_layer2(&s,1);
}
//---------------------------------------------------------------------
// Envía un marco a la capa de red
void TProtocolo::to_network_layer(packet *p, unsigned int count, bool last,
seq_nr mens_esp, WORD ID)
{
 packet p1;
 BYTE MSB,LSB;
 WORD CRCCE;
 p1 = *p;
 CRCCE = p->data[MAX_PKT-1];
 MSB = p->data[MAX_PKT-2];
 CRCCE = CRCCE + MSB * 0x100;
 if (Cal_CRC_CE(&p1) == CRCCE)
 {
    send_ack(mens_esp,last,ID,ack1);
    MainForm->FromDataLinkLayer(p, count, last);
    //crcpck = true;
  }
}
//---------------------------------------------------------------------
void TProtocolo::MensajeHola(AnsiString Str)
{
     MainForm->RichEdit3->Lines->BeginUpdate();
     MainForm->RichEdit3->Lines->Append(Str);
     SendMessage(MainForm->RichEdit3->Handle,
EM_SCROLLCARET,0,0);
     MainForm->RichEdit3->Lines->EndUpdate();
}
//---------------------------------------------------------------------
// Suma de comprobación
WORD TProtocolo::Cal_CRC_16(frame* s)
{
 CRC_16 CRC;
 int Message_size;
 BYTE* m = (BYTE*) s;
 Message_size = sizeof(frame);
 if (compt)
 {
    if(chale12 == 255)
      chale13++;
    chale12++;
  }
  CRC.crc = Check(Message_size, m);
  return CRC.crc;
}
//---------------------------------------------------------------------
WORD TProtocolo::Cal_CRC_CE(packet* s)
{
 CRC_16 CRC;
 int Message_size;
 BYTE* m = (BYTE*) s;
 Message_size = sizeof(packet);
 CRC.crc = Check(Message_size, m);
 return CRC.crc;
}
```

```
//---------------------------------------------------------------------
WORD TProtocolo::Cal_CRC_162(marco2* s)
{
 CRC_16B CRC;
 int Message_size;
 Message_size = sizeof(marco2);
 CRC.crc = CheckB(Message_size, (BYTE*) s, false);
 return CRC.crc;
}
//---------------------------------------------------------------------
////realiza la comprobación de CRC
WORD TProtocolo::Check(int Message_size, BYTE* m)
{
 CRC_16 CRC;
 BYTE Aux;
 BYTE Message[sizeof(frame)];
 for(int i = 0; i < Message_size; i++)
    Message[i] = m[i];
 CRC.byte_crc[0] = Message[1];
 CRC.byte_crc[1] = Message[0];
 for(int i = 2; i < Message_size - 2; i++)
 {
   Aux = Message[i];
   for(int j = 0; j < 8; j++)
   {
    if((0x8000 & CRC.crc))
    {
      CRC.crc = CRC.crc * 2;
      if(Aux & 0x80)
          CRC.crc = CRC.crc | 0x0001;
      CRC.crc = CRC.crc ^ Generator;
    }
    else
    {
      CRC.crc = CRC.crc * 2;
      if(Aux & 0x80)
          CRC.crc = CRC.crc | 0x0001;
      CRC.crc = CRC.crc ^ 0x0000;
    }
    Aux = Aux * 2;
   }
 }
 return CRC.crc;
}
//---------------------------------------------------------------------
/////Funcion para realizar la prueba CRC
WORD TProtocolo::CheckB(int Message_size, BYTE* m, bool a)
{
 CRC_16B CRC;
 BYTE Aux;
 BYTE Message[sizeof(marco2)];
 bool band = false, band1 = true;
 int cont = 0;
 Message[1] = m[0];
 Message[0] = m[1];
 for(int i = 2; i < Message_size; i++)
    Message[i] = m[i];
 CRC.crc = Message[0];
 while (cont < Message_size)
 {
   for(int j = 0; j < 4; j++)
   {
    if((0x80 & CRC.crc))
       band = true;
    CRC.crc = CRC.crc * 2;
    if (j == 3)
    {
       if (band1)
       {
          cont++;
          Aux = Message[cont] & 0xF0;
          Aux = Aux / 16;
          CRC.crc = CRC.crc ^ Aux;
          band1 = false;
       }
       else
       {
          Aux = Message[cont] & 0x0F;
          CRC.crc = CRC.crc ^ Aux;
          band1 = true;
       }
    }
    if (band)
    {
       CRC.crc = CRC.crc ^ 0x28;
       band = false;
    }
    if ((cont == Message_size-2 && j == 3 && band1 && a)||(cont ==
Message_size-1 && j == 3))
    {
       cont = Message_size;
       j = 4;
    }
   }
 }
 return CRC.crc;
}
//---------------------------------------------------------------------
// Envía un marco a la capa fisica
void TProtocolo::to_physical_layer2(marco2 *s, int b)
{
 int tam_mens;
 BYTE* fr = (BYTE*)s;
 tam_mens = sizeof(marco2);
 ACapaFisica2(tam_mens, b, fr);
}
//---------------------------------------------------------------------
//---------------------------------------------------------------------
void TProtocolo::Garbage(void)
{
 auxiliar2.insert(auxiliar2.end(), FF);
 auxiliar2.insert(auxiliar2.end(), ETX);
 auxiliar2.insert(auxiliar2.end(), DIEZ);
 auxiliar2.insert(auxiliar2.end(), CIN);
 auxiliar2.insert(auxiliar2.end(), DIEZ);
 auxiliar2.insert(auxiliar2.end(), CIN);
 auxiliar2.insert(auxiliar2.end(), FF);
 auxiliar2.insert(auxiliar2.end(), ETX);
}
//---------------------------------------------------------------------
////función para mandar los datos al buffer del serial
void TProtocolo::ACapaFisica2(int tam_mens, int z, BYTE* fr)
{
 std::vector <BYTE> auxiliar;
 int n, i = 0, a, j, t = 0, k = 0, l = 0;
 BYTE c,b;
 int bs[8];
 auxiliar.insert(auxiliar.end(), FF);
 auxiliar.insert(auxiliar.end(), ETX);
 auxiliar.insert(auxiliar.end(), CERO);
 auxiliar.insert(auxiliar.end(), CERO);
  while (i<tam_mens)
  {
    b = fr[i];
    j = 8;
    while (j)
    {
      a = pow(2,j-1);
      if (b > a-1)
      {
        b-=a;
        *(bs+t) = 1;
        k++;
      }
      else
      {
         *(bs+t) = 0;
```

```
    k = 0;
}
if (k == 5 && t < 7)
{
        t++;
        *(bs+t) = 0;
        k = 0;
}
if (t == 7)
{
    c = 0;
    n = 8;
    t = 0;
    while (n)
    {
        a = pow(2,n-1);
        if (*(bs+t) == 1)
            c += a;
        t++;
        n--;
    }
    auxiliar.insert(auxiliar.end(), c);
  }
  if (k == 5 && t == 8)
  {
        inc3(t);
        *(bs+t) = 0;
        k = 0;
  }
  inc3(t);
  j--;
 }
 i++;
}
*(bs+t) = 0;
while (l<8)
{
 while (tl=7 && l < 6)
 {
    inc3(t);
    *(bs+t) = 1;
    l ++;
 }
 if (t == 7)
 {
    c = 0;
    n = 8;
    t = 0;
    while (n)
    {
        a = pow(2,n-1);
        if (*(bs+t) == 1)
            c += a;
        t++;
        n--;
    }
    auxiliar.insert(auxiliar.end(), c);
    if (l == 7)
        l = 8;
 }
 while (l == 6 && t != 7)
 {
  inc3(t);
  *(bs+t) = 0;
  if (t == 7)
    l = 7;
 }
 }
 if (z == 1)
 {
    MainForm->SerialPort->GetCommPort().WriteBuffer(auxiliar2.begin(),
auxiliar2.size());
    Sleep(5);
    MainForm->SerialPort->GetCommPort().WriteBuffer(auxiliar.begin(),
auxiliar.size());
  }
  else
  {
    MainForm->SerialPort->GetCommPort().WriteBuffer(auxiliar.begin(),
auxiliar.size());
  }
}
//---------------------------------------------------------------------
// Obtiene un marco de la capa fisica
void TProtocolo::from_physical_layer1(marco2 *s)
{
  int tam_mens;
  BYTE* fr1 = (BYTE*)s;
  tam_mens = sizeof(marco2);
  DeCapaFisica(tam_mens, fr1);
}
//---------------------------------------------------------------------
// Obtiene un marco de la capa fisica
void TProtocolo::from_physical_layer(frame *s)
{
  int tam_mens;
  BYTE* fr1 = (BYTE*)s;
  tam_mens = sizeof(frame);
  DeCapaFisica1(tam_mens, fr1);
}
//---------------------------------------------------------------------
///Función para leer el buffer del seria  copia de la unidadd
void TProtocolo::DeCapaFisica(int tam_mens, BYTE *fr)
{
  BYTE Bytebuffer, a;
  BYTE NextByte, c;
  int i=0, j, m = 2, k = 0,t = 0, l = 0;//k = 1,m = 0
  DWORD BytesAvailable;
  bool chido = false;
  int bs[8];
  int chinga = 0;
  BytesAvailable = MainForm->SerialPort->GetCommPort().BytesAvailable();
  while (!chido && BytesAvailable)
  {
     Bytebuffer = MainForm->SerialPort->GetCommPort().GetByte();
     j = 8;
     while (j && !chido /*m < 2*/)/////!chido x m < 2
     {
       a = pow(2,j-1);
       if (Bytebuffer > a-1)
       {
          Bytebuffer -= a;
          *(bs+t) = 1;
          k++;
       }
       else
       {
          *(bs+t) = 0;
          if (k == 6 && m == 1)
          {
             m = 2;///////m = sin valor y k = 0
             chido = true;
          }
          else
          {
             k = 0;
             m = 1;
             }
       }
       inc3(t);
       j--;
     }
       BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
       while (!BytesAvailable && chido && chinga < 4)
       {
          Sleep (1);
```

```
        chinga ++;
        BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
      }
      chinga = 0;
    }
    if (chido && BytesAvailable)
    {
      if (j == 0 && t == 0)
      {
        j = 8;
        Bytebuffer = MainForm->SerialPort->GetCommPort().GetByte();
      }
      m = 1;
      k = 0;
      int hi = 0;
      while (i < (tam_mens+1) && BytesAvailable && chido)
      {
        while (j && i < (tam_mens+1) && chido)
        {
          a = pow(2,j-1);
          if (Bytebuffer > a-1)
          {
            Bytebuffer -= a;
            *(bs+t) = 1;
            k++;
            hi = 0;
          }
          else
          {
            *(bs+t) = 0;
            if (k == 6)
            {
              if (i)
               chido = false;
              else
               k = 0;
               hi = 1;
            }
          }
          else if (k == 5)
          {
            k = 0;
            m = 0;
            hi = 0;
          }
          else
          {
            k = 0;
            hi = 0;
          }
        }
        if (m)
        {
          inc3(t);
          l++;
        }
        else
          m = 1;
        j--;
        if (l == 8)
        {
          if (k>7)
            chido = false;
          c = 0;
          while (l && !hi)
          {
            a = pow(2,l-1);
            if (*(bs+t) == 1)
              c += a;
            inc3(t);
            l--;
          }
          if (i < tam_mens && !hi)
          {
            fr[i] = c;
            m2_BufferIn.insert(m2_BufferIn.end(), c);
            BytesReceived1++;
            if (i == tam_mens-1)
            compt = true;
          }
          l = 0;
          if (!hi)
          i++;
        }
      }
      j = 8;
      BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
      while (!BytesAvailable && chido && i < tam_mens+1 && chinga < 5)
      {
        Sleep (1.4);
        BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
        chinga++;
      }
      chinga = 0;
      if (BytesAvailable && chido && i < tam_mens+1)
      {
        Bytebuffer = MainForm->SerialPort->GetCommPort().GetByte();
      }
    }
  }
}
//---------------------------------------------------------------------
///Función para leer el buffer del seria  copia de la unidadd
void TProtocolo::DeCapaFisica1(int tam_mens, BYTE *fr)
{
  BYTE Bytebuffer, a;
  BYTE NextByte, c;
  int i=0, j, m = 2, k = 0,t = 0, l = 0;//k = 1,m = 0
  DWORD BytesAvailable;
  bool chido = false;
  int bs[8];
  int chinga = 0;
  BytesAvailable = MainForm->SerialPort->GetCommPort().BytesAvailable();
  while (!chido && BytesAvailable)
  {
    Bytebuffer = MainForm->SerialPort->GetCommPort().GetByte();
    j = 8;
    while (j && !chido)
    {
      a = pow(2,j-1);
      if (Bytebuffer > a-1)
      {
        Bytebuffer -= a;
        *(bs+t) = 1;
        k++;
      }
      else
      {
        *(bs+t) = 0;
        if (k == 6 && m == 1)
        {
          m = 2;///////m = sin valor y k = 0
          chido = true;
        }
        else
        {
        k = 0;
        m = 1;
        }
      }
      inc3(t);
      j--;
    }
    BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
    while (!BytesAvailable && chido && chinga < 4)
```

```
    {
      Sleep (1);
      chinga ++;
      BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
    }
    chinga = 0;
  }
  if (chido && BytesAvailable)
  {
    if (j == 0 && t == 0)
    {
      j = 8;
      Bytebuffer = MainForm->SerialPort->GetCommPort().GetByte();
    }
    m = 1;
    k = 0;
    int hi = 0;
    while (i < (tam_mens+1) && BytesAvailable && chido)
    {
      while (j && i < (tam_mens+1) && chido)
      {
        a = pow(2,j-1);
        if (Bytebuffer > a-1)
        {
          Bytebuffer -= a;
          *(bs+t) = 1;
          k++;
          hi = 0;
        }
        else
        {
          *(bs+t) = 0;
          if (k == 6)
          {
            if (i)
              chido = false;
            else
              k = 0;
              hi = 1;
          }
          else if (k == 5)
          {
            k = 0;
            m = 0;
            hi = 0;
          }
          else
          {
            k = 0;
            hi = 0;
          }
        }
        if (m)
        {
          inc3(t);
          l++;
        }
        else
          m = 1;
        j--;
        if (l == 8)
        {
          if (k>7)
            chido = false;
          c = 0;
          while (l && !hi)
          {
            a = pow(2,l-1);
            if (*(bs+t) == 1)
              c += a;
            inc3(t);
            l--;
          }
          if (i < tam_mens && !hi)
          {
            fr[i] = c;
            if (i == tam_mens-1)
            compt = true;
          }
          l = 0;
          if (!hi)
          i++;
        }
      }
      j = 8;
      BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
      while (!BytesAvailable && chido && i < tam_mens+1 && chinga < 2)
      {
        Sleep (12);
        BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
        chinga++;
      }
      chinga = 0;
      if (BytesAvailable && chido && i < tam_mens+1)
      {
        Bytebuffer = MainForm->SerialPort->GetCommPort().GetByte();
      }
    }
  }
}
```

D.3 PROTOCOLO.H

```
//---------------------------------------------------------------------
#ifndef ProtocoloH
#define ProtocoloH
#include <Classes.hpp>
#include "comm.h"
#include <vector>
#include <string>
#include <sys\timeb.h>
#define MAX_PKT 128
#define MAX_SEQ 7
using namespace std;
typedef vector<WORD> vreg;
typedef struct { unsigned char data[MAX_PKT];} packet;
typedef struct
{
    WORD ID;
    unsigned int tamano;
    seq_nr seq;
    bool bandera;
    std::vector <BYTE> Buff;

}memorix;
//---------------------------------------------------------------------
///////Estructura del marco
typedef struct
{
  WORD direcc;   /*localidad de la unidad*/
  seq_nr seq;  /* numero de secuencia*/
  unsigned short cont; /*contador de bytes de la trama*/
  bool last;
```

```
  packet info; /* paquete de la capa de red*/
  WORD CRC;
} frame;

typedef enum
  {
  listado = 8192,
  presente = 16384,
  final = 24576,
  ack1 = 32768,
  datos = 40960
  } tipo_marco; /* definicion de tipo de trama*/
//---------------------------------------------------------------
////// Estructura del marco Listado, Presente y Final
typedef struct
{
  WORD direcc;
  BYTE CRC;
} marco;

///////Estructura del marco ACK
typedef struct
{
  WORD direcc;
  BYTE seq;
  BYTE CRC;
} marco2;

//////////Estructura Para realizar CRC
typedef union{
  BYTE crc;
  BYTE byte_crc;
}CRC_16B;

typedef union{
  WORD crc;
  BYTE byte_crc[2];
}CRC_16;

const BYTE CIN = 85;
const BYTE DIEZ = 170;
const BYTE ETX = 254;
const BYTE FF = 255;
const BYTE CERO = 126;
const WORD Generator = 0x1025;
//---------------------------------------------------------------
class TProtocolo : public TThread
{
protected:
   void __fastcall Execute();
   void __fastcall SynchronizeNewBytes(void);
   void __fastcall SynchronizeNewBytes2(void);
   void TProtocolo::to_network_layer(packet *p, unsigned int count,
bool last, seq_nr mens_esp, WORD ID);
   void MensajeHola(AnsiString Str);
   void mandar_trama(WORD ID,tipo_marco tipo,int b,seq_nr
num_arch);
   void send_ack(seq_nr mens_esp, bool last,WORD ID ,tipo_marco
tipo);
   void to_physical_layer2(marco2 *s, int b);
   void Garbage(void);
   void from_physical_layer(frame *s);
   void from_physical_layer1(marco2 *s);
   void DeCapaFisica(int tam_mens, BYTE *fr);
   void DeCapaFisica1(int tam_mens, BYTE *fr);
   WORD Cal_CRC_16(frame * s);
   WORD Cal_CRC_161(marco* s);
   WORD Cal_CRC_162(marco2* s);
   WORD Cal_CRC_CE(packet* s);
   WORD Check(int Message_size, BYTE* m);
   WORD CheckB(int Message_size, BYTE* m, bool a);
   void ConversionBybi(int* bs, int tam_mens, BYTE* fr);
   void Agregarcero(int* bs1,int* bs2, int size_trama);
   void ACapaFisica2(int tam_mens, int b,BYTE* fr);

//---------------------------------------------------------------
   typedef enum
   {frame_arrival, cksum_err, t1imeout,
network_layer_ready}event_type;
   seq_nr next_frame_to_send; /* MAX_SEQ > 1; se usa para
corriente de salida*/
   seq_nr ack_expected;     /* Marco mas viejo aun no reconocido */
   packet buffer[MAX_SEQ + 1];   /* buffers para corriente de
salida */
   int clock[MAX_SEQ + 1];
   int count[MAX_SEQ + 1];
   seq_nr nbuffered;         /* numero de buffers usados
actualmente*/
   event_type event;
   bool retransmit;

//---------------------------------------------------------------
WORD ID;
  vreg reg;
  bool queondon;
  bool flag1;
  bool compt;
  bool borrar;
  unsigned int contr;
//---------------------------------------------------------------
public:
   __fastcall TProtocolo();
   virtual __fastcall ~TProtocolo();
    marco2 R;   /* siguiente marco esperado en la entrada*/
   frame r;              /* marco de trabajo*/
   memorix e[2];
   seq_nr mens_esp;     /* siguiente marco esperado en la entrada*/
   unsigned int retransm;
   bool Fin;
   unsigned int cont1;
   bool Barb;
   AnsiString cont2;
   bool tran_inc;
   int direc;
   int tam_bs2;
   std::vector <BYTE> m2_BufferIn;
   unsigned int BytesReceived21;
   std::vector <BYTE> m_BufferCRC;
   unsigned int BytesReceived1;
   std::vector <BYTE> m2eta_BufferIn;
   unsigned int BytesReceived1eta;
   unsigned int BytesReceived21eta;
   std::vector <BYTE> meta_BufferCRC;
   char Aname[30];
   AnsiString cad;
   char Aname1[30];
   AnsiString cad1;
   char Anameeta[30];
   AnsiString cadeta;
   char Aname1eta[30];
   AnsiString cad1eta;
   BYTE chale, chale1, chale2, chale3, chale4, chale5, chale6, chale7,
chale8;
   BYTE chale9, chale10, chale11, chale12, chale13, chale14, chale15,
chale16, chale17;
   bool barro,crcpck;
   int algo;
   seq_nr num_arch;
   std::vector <BYTE> auxiliar2;

};
//---------------------------------------------------------------
#endif
//---------------------------------------------------------------
```

APÉNDICE E

E. PROGRAMA OPERATIVO DEL ESCLAVO

E.1 MAIN.CPP

```
//---------------------------------------------------------------------
//Programa Principal
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "main.h"
#include "commsettings.h"
#include <memory>
#include <fstream.h>
#include "SerialPort.h"
#include "mensaje.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TMainForm *MainForm;
//---------------------------------------------------------------------
// Creacion del buffer para enviar y recibir datos
__fastcall TMainForm::TMainForm(TComponent* Owner)
  : TForm(Owner)
{
  // Se trabaja con un hilo llamado SerialPort
  SerialPort = new TSerialPort();

  // Se trabaja con un hilo llamado Protocolo
  Protocolo = new TProtocolo();

  r = 0;
  StatusBar1->DoubleBuffered = true;
  m_BufferOutLength[0] = 0;
  m_BufferOutLength[1] = 0;
  m_BufferOutLength[2] = 0;
  UpdateStatusBar();
  Can_env = 0;
  Final = false;
  Transmiting = false;
  Select = false;
  Select2 = false;
  Select3 = false;
  Select1 = false;
  ultimo = 1;
  RadioButton1->Checked = true;
}
//---------------------------------------------------------------------
__fastcall TMainForm::~TMainForm()
{
  // Si Protocolo está ejecutándose lo detiene y espera a que
termine.
  if(!Protocolo->Suspended)
  {
    Protocolo->Terminate();
    Protocolo->WaitFor();
  }
  delete Protocolo;
}
//---------------------------------------------------------------------
// Codigo para elegir el puerto de transmision y para elegir la tasa de
baudios
void __fastcall TMainForm::actCommSettingsExecute(TObject
*Sender)
{
  std::auto_ptr<TCommSettingsForm> form(new
TCommSettingsForm(NULL));
  form->CommPort = SerialPort->GetCommPort().GetCommPort();
  form->BaudRate = SerialPort->GetCommPort().GetBaudRate();
  if(form->ShowModal())
  {
    bool bConnected = SerialPort->GetConnected();
    SerialPort->SetParameters(form->BaudRate, form->CommPort);
    if (bConnected)
      SerialPort->Connect();
  }

  UpdateStatusBar();
}
//---------------------------------------------------------------------
//Conexion
void __fastcall TMainForm::actConnectExecute(TObject *Sender)
{
  if (Select && Select2 && Select3)
  {
    SerialPort->Connect();
    UpdateStatusBar();
    if (SerialPort->GetConnected())
      Protocolo->Resume();
    Button2->Enabled = false;
    Connect1->Enabled = false;
    Settings1->Enabled = false;
    Select1 = true;
    RadioButton1->Enabled = false;
    RadioButton2->Enabled = false;
    RadioButton3->Enabled = false;
    RadioButton1->Checked = true;
  }
  else
  {
    Error->Label->Caption = "Primero seleccionar los archivos a
enviar.";
    Error->ShowModal();
  }
}
//---------------------------------------------------------------------
//Desconexion
void __fastcall TMainForm::actDisconnectExecute(TObject *Sender)
{
  if (!Protocolo->Suspended)
  {
    Protocolo->Suspended = true;
    SerialPort->Disconnect();
    UpdateStatusBar();
    Button2->Enabled = true;
    Connect1->Enabled = true;
    Settings1->Enabled = true;
    Select1 = false;
    RadioButton1->Enabled = true;
    RadioButton2->Enabled = true;
    RadioButton3->Enabled = true;
  }
  else
  {
    Error->Label->Caption = "El sistema esta actualmente
desconectado.";
    Error->ShowModal();
  }
}
//---------------------------------------------------------------------
```

```
void __fastcall TMainForm::actConnectUpdate(TObject *Sender)
{
  actConnect->Enabled = !SerialPort->GetConnected();
}
//---------------------------------------------------------------
void __fastcall TMainForm::actDisconnectUpdate(TObject *Sender)
{
  actDisconnect->Enabled = SerialPort->GetConnected();
}
//---------------------------------------------------------------
//Exit
void __fastcall TMainForm::Exit1Click(TObject *Sender)
{
  Close();
}
//---------------------------------------------------------------
void __fastcall TMainForm::StatusBar1Resize(TObject *Sender)
{
  StatusBar1->Panels->Items[0]->Width = StatusBar1->Width - 290;
}
//---------------------------------------------------------------
//Actualiza la barra de estado
void TMainForm::UpdateStatusBar()
{
  AnsiString str,str1;
  str = "Estado: ";
  if(SerialPort->GetConnected())
    str += "Conectado      ";
  else
    str += "Desconectado   ";
  str += "Baudios = " + IntToStr(SerialPort->GetCommPort().
GetBaudRate());
  str += "   Puerto = ";
  str += SerialPort->GetCommPort().GetCommPort().c_str();
  str1 ="Tamaño = " + AnsiString(m_BufferOutLength[r])+" bytes";
  str1 +="   Datos enviados = " + AnsiString(Can_env)+" bytes";
  StatusBar1->Panels->Items[0]->Text = str;
  StatusBar1->Panels->Items[1]->Text = str1;
}
//---------------------------------------------------------------
//Seleccion del archivo a enviar
void __fastcall TMainForm::SendFile1Click(TObject *Sender)
{
  if (!Select1)
  {
    if (OpenDialog1->Execute())
    {
      MainForm->RichEdit3->Clear();
      MainForm->ProgressBar1->Max=0;
      MainForm->ProgressBar1->Max=100;
      Protocolo->Inicializacion();
      FileName = OpenDialog1->FileName;
      if (r == 0)
      {
     //Image1->Picture->LoadFromFile(OpenDialog1->FileName);
        Label2->Caption="";
        Label2->Caption = OpenDialog1->FileName;
        Select=true;
 }
  else if (r == 1)
{
   Label6->Caption="";
 //Image2->Picture->LoadFromFile(OpenDialog1->FileName);
   Label6->Caption = OpenDialog1->FileName;
   Select2=true;
}
else if (r == 2)
{
   Label7->Caption="";
 //Image3->Picture->LoadFromFile(OpenDialog1->FileName);
   Label7->Caption = OpenDialog1->FileName;
   Select3=true;
}
      m_BufferOutLength[r] = 0;
      Can_env = 0;
      Final = false;
      CargarArchivo();
      }
    }
    else
    {
      Error->Label->Caption = "Primero debes desconcetar el enlace.";
      Error->ShowModal();
    }
}
//---------------------------------------------------------------
//envio de trama a la capa de red
unsigned int TMainForm::ToDataLinkLayer(packet *p)
{
 unsigned int contador = 0;
 while (contador < MAX_PKT-2 && Can_env < m_BufferOutLength[r])
 {
  p->data[contador] = m_BufferOut[r].at(Can_env);
  contador++;
  Can_env++;
 }
 p->data[contador] = 0x00;
 p->data[contador+1] = 0x00;
 if (Can_env == m_BufferOutLength[r] && Can_env > 0)
 {
   Transmiting = false;
   ultimo = Protocolo->next_frame_to_send;
 }
 ProgressBar1->StepBy(contador);
 UpdateStatusBar();
 return contador;
}
//---------------------------------------------------------------
void __fastcall TMainForm::CargarArchivo(void)
{
  unsigned int contador = 0;
  ifstream Archivo;
  BYTE byte;
  m_BufferOut[r].erase(m_BufferOut[r].begin(),m_BufferOut[r].end());
  Archivo.open(FileName.c_str(), ios::binary);
  while (Archivo.read(&byte, 1))
  {
    m_BufferOut[r].insert(m_BufferOut[r].end(), byte);
    contador++;
  }
  m_BufferOutLength[r] = contador;
  UpdateStatusBar();
  Archivo.close();
  Transmiting = true;
}
//---------------------------------------------------------------
void TMainForm::Desconectar(void)
{
  SerialPort->Disconnect();
  UpdateStatusBar();
  Button2->Enabled = true;
  Connect1->Enabled = true;
  Settings1->Enabled = true;
  Select1 = false;
  if (!Protocolo->Suspended)
     Protocolo->Suspended = true;
}
//---------------------------------------------------------------
void __fastcall TMainForm::RadioButton1Click(TObject *Sender)
{
  r = 0;
  Image1->Visible = true;
  Image2->Visible = false;
  Image3->Visible = false;
  Label2->Visible = true;
  Label6->Visible = false;
```

```
   Label7->Visible = false;
   UpdateStatusBar();
   ProgressBar1->Max = m_BufferOutLength[r];
}
//---------------------------------------------------------
void __fastcall TMainForm::RadioButton2Click(TObject *Sender)
{
   r = 1;
   Image2->Visible = true;
   Image1->Visible = false;
   Image3->Visible = false;
   Label6->Visible = true;
   Label2->Visible = false;
   Label7->Visible = false;
   UpdateStatusBar();
   ProgressBar1->Max = m_BufferOutLength[r];
}
//---------------------------------------------------------
void __fastcall TMainForm::RadioButton3Click(TObject *Sender)
{
   r = 2;
   Image3->Visible = true;
   Image2->Visible = false;
   Image1->Visible = false;
   Label7->Visible = true;
   Label2->Visible = false;
   Label6->Visible = false;
   UpdateStatusBar();
   ProgressBar1->Max = m_BufferOutLength[r];
}
//---------------------------------------------------------
```

E.2 PROTOCOLO.CPP

```
#include <vcl.h>
#pragma hdrstop
#include "comm.h"
#include "Protocolo.h"
#include "main.h"
#include <exptintf.hpp>
#pragma package(smart_init)
#include <fstream.h>
#define espera 17
#define ident 5
#define inc3(k) if (k < 7) k = k + 1; else k = 0;
#define inc(k) if (k < 7/*MAX_SEQ*/) k = k + 1; else k = 0;
const int TextStreamCapacity3   = 16;
//////////////////////////////////////
//---------------------------------------------------------
__fastcall TProtocolo::TProtocolo()
   : TThread(true)
{
   Inicializacion();
   compt = false;
   cont2 = "si";
   MainForm->Label5->Caption = ident;
   auxiliar2.reserve(TextStreamCapacity3);
   mandar_trama (ident, presente);
   camb_arch = true;
   turno = false;
}
//---------------------------------------------------------
//Destructor
__fastcall TProtocolo::~TProtocolo()
{
}
//---------------------------------------------------------
void __fastcall TProtocolo::Execute()
{
   while (!Terminated)
   {
      if (MainForm->Final)
      {
         Inicializacion();
         IniciaBucle();
         camb_arch = true;
      }
      DWORD BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
      if (BytesAvailable > 7)
      {
         Synchronize(SynchronizeNewBytes2);
      }

      while (transms == true)
      {
         DWORD BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
         if (nbuffered == 1 && retransmit)
            dec_timer();
         if(nbuffered == 0 && MainForm->Transmiting)
         {
            cont1 = espera;
            unsigned int contador;
            contador = from_network_layer(&buffer[next_frame_to_send]);
            count[next_frame_to_send] = contador;
            nbuffered = 1;
            send_data(next_frame_to_send, buffer, contador);
            inc(next_frame_to_send);
         }
         if(BytesAvailable > 1)
         {
            Synchronize(SynchronizeNewBytes);
         }
         if (!cont1)
         {
            if (transms)
            MainForm->RichEdit3->Clear();
            transms = false;
            cont1 = espera;
            cont2 = "si";
         }
      }
   }
}

//---------------------------------------------------------
///sincronización de presentes
void __fastcall TProtocolo::SynchronizeNewBytes2(void)
{
 AnsiString Str("Mensaje de ");
 tipo_marco tipo;
 WORD direcc1;
 seq_nr can_env;
//Seguridad para cuando el programa se cierre sin haber terminado
 if(Application->Terminated)
      return;
  // llego un marco desde la capa fisica
 from_physical_layer1(&R);
  // Chek_Sum
 if (compt && Cal_CRC_162(&R) == 0)
 {
   tipo = R.direcc & 0xE000;
   direcc1 = R.direcc & 0x1FFF;
   if (tipo == listado && direcc1 == ident)
   {
```

```
  ACapaFisica3();
  Str = Str + "búsqueda #" + direcc1;
  MainForm->RichEdit3->Lines->Clear();
  MensajeHola(Str);
  turno = true;
}
if (tipo == final && direcc1 == ident && turno)
{
  can_env = (R.seq & 0xE0)/0x20;
  turno = false;
  if (camb_arch)
  {
    if (can_env == 1)
        MainForm->RadioButton1->Checked = true;
    else if (can_env == 2)
        MainForm->RadioButton2->Checked = true;
    else if (can_env == 3)
        MainForm->RadioButton3->Checked = true;
    }
    camb_arch = false;
    transms = true;
    cont1 = espera;
    MainForm->RichEdit3->Lines->Clear();
  }
  if (tipo == listado && direcc1 != ident)
  {
    Str = Str + "búsqueda #" + direcc1;
    MensajeHola(Str);
  }
  compt = false;
 }
 compt = false;
}
//---------------------------------------------------------------------
///sincronización de ACK
void __fastcall TProtocolo::SynchronizeNewBytes(void)
{
 bool cuenta = false;
 //Seguridad para cuando el programa se cierre sin haber terminado
 if(Application->Terminated)
     return;
 // llego un marco desde la capa fisica
 from_physical_layer2(&P);
  // Chek_Sum
 if (compt && Cal_CRC_162(&P) == 0)
 {
   seq_nr seq;
   tipo_marco tipo;
   WORD direcc1;
   seq = (P.seq & 0xE0)/0x20;
   tipo = P.direcc & 0xE000;
   direcc1 = P.direcc & 0x1FFF;
  if (tipo == ack1 && direcc1 == ident)
  {
    while(between(seq_esp, seq, next_frame_to_send))
    {
     cuenta = true;
     BYTE last;
     nbuffered = 0;
     stop_timer();
     inc(seq_esp);
     last = (P.seq & 0x10)/0x10;
     if (!nbuffered && !MainForm->Transmiting && last==1)
     {
       transms = false;
      MainForm->Final = true;
      send_fin(ident);
      MainForm->RichEdit3->Lines->Clear();
      cont1 = 0;
      cont2 = "no";
    }
   }
 }
}
  compt = false;
  }
 compt = false;
}
//---------------------------------------------------------------------
// Construccion del marco tipo listado
void TProtocolo::mandar_trama(WORD profex,tipo_marco tipo)
{
 BYTE CRCS;
 marco2 s;
 s.direcc = 0x0000;
 s.seq = 0x00;
 s.CRC = 0x00;
 s.direcc = profex | tipo;        /*direccion de la unidad a localizar*/
 CRCS = Cal_CRC_162(&s);
 s.seq = s.seq ^ (CRCS / 16);
 s.CRC = CRCS * 0x10;
 to_physical_layer2(&s);     /* transmite marco */
}
//---------------------------------------------------------------------
// Contruccion del marco tipo final del tamaño de 138 B
void TProtocolo::send_fin(seq_nr frame_expected)
{
 frame s;
 s.cont = 0;
 s.direcc = ident | final;
 s.seq = frame_expected;
 s.CRC = 0x0000;
 s.CRC = Cal_CRC_16(&s);
 to_physical_layer(&s);
}
//---------------------------------------------------------------------
// Construccion del marco de daros
void TProtocolo::send_data(seq_nr frame_nr, packet buffer[],
unsigned short contador)
{
 AnsiString Str("Mensaje enviado # ");
 frame s;      /* marco de trabajo*/
 CRC_16 CRR;
 s.direcc = ident | datos;
 s.info = buffer[frame_nr];  /* Inserta un paquete en el marco */
 s.cont = contador;         /* Inserta el numero de bytes del paquete*/
 s.seq = frame_nr;          /* inserta el numero de secuencia en el marco */
 s.CRC = 0x0000;            /* Inicializa el campo de CRC*/
 Str = Str + s.seq;
 if (!MainForm->Transmiting && frame_nr == MainForm->ultimo)
 {
   s.last = true;
   Str = Str + "\n"+ "Ultimo mensaje.";
   MensajeHola(Str);
 }
 else
   s.last = false;
 s.CRC=Cal_CRC_16(&s);  //Calcula el campo de suma de comprobación
 to_physical_layer(&s);     /* transmite marco */
 start_timer(/*0/frame_nr*/);     /* comienza a temporizar*/
 MensajeHola(Str);
}
//---------------------------------------------------------------------
// Obtiene un marco de la capa de red
unsigned int TProtocolo::from_network_layer(packet *p)
{
  int contador;
  packet p1;
  WORD CRCCE;
  BYTE MSB,LSB;
  contador = MainForm->ToDataLinkLayer(p);
  p1 = *p;
  CRCCE = Cal_CRC_CE(&p1);
  LSB = CRCCE;
  MSB = CRCCE/0x100;
  p->data[MAX_PKT-1] = LSB;
  p->data[MAX_PKT-2] = MSB;
```

```
  return (contador);
}
//---------------------------------------------------------------------
//Verifica que el número de trama recibia se encuentre entre las esperadas
bool TProtocolo::between(seq_nr a, seq_nr b, seq_nr c)
{
//  Funcion between
 if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) ||
    ((b < c) && (c < a)))
     return(true);
 else
     return(false);
}
//---------------------------------------------------------------------
// Inicializa el temporizador
void TProtocolo::start_timer()
{
  clock = 15000
}
//---------------------------------------------------------------------
// Detiene el temporizador
void TProtocolo::stop_timer()
{
  clock = 999999;
}
//---------------------------------------------------------------------
// Decrementa el temporizador
void __fastcall TProtocolo::dec_timer(void)
{
    if (clock != 999999)
         clock = clock - 1;
    if (clock == 0)
    {
      if (cont1)
      cont1--;
      send_data(seq_esp, buffer, count[seq_esp]);
    }
    if(MainForm->Final)
    retransmit = false;
}
//---------------------------------------------------------------------
//Pantalla
void TProtocolo::MensajeHola( AnsiString Str)
{
  MainForm->RichEdit3->Lines->BeginUpdate();
  MainForm->RichEdit3->Lines->Append(Str);
  SendMessage(MainForm->RichEdit3->Handle, EM_SCROLLCARET,0,0);
  MainForm->RichEdit3->Lines->EndUpdate();
}
//---------------------------------------------------------------------
//Inicializa variables
void TProtocolo::Inicializacion(void)
{
  seq_esp = 0;        /* Siguiente acuse de entrada esperado */
  next_frame_to_send = 0;   /* siguiente marco de salida */
  clock = 999999;
  retransmit = true;
  nbuffered = 0;
  cont1 = espera;
  transms = false;
}
//---------------------------------------------------------------------
//Bucle
void TProtocolo::IniciaBucle(void)
{
   MainForm->Final = false;
   MainForm->Transmiting = true;
   MainForm->Can_env = 0;
   MainForm->ProgressBar1->Max=0;
   MainForm->ProgressBar1->Max=100;
   MainForm->ProgressBar1->Max = MainForm-
>m_BufferOutLength[MainForm->r];
}
```

```
//---------------------------------------------------------------------
//Suma de comprobacion
WORD TProtocolo::Cal_CRC_162(marco2* s)
{
 CRC_16 CRC;
 int Message_size;
 Message_size = sizeof(marco2);
 CRC.crc = CheckB(Message_size, (BYTE*) s, false);
 return CRC.crc;
}
//---------------------------------------------------------------------
// Suma de comprobación
WORD TProtocolo::Cal_CRC_16(frame* s)
{
 CRC_16 CRC;
 int Message_size;
 BYTE* m = (BYTE*) s;
 Message_size = sizeof(frame);
 CRC.crc = Check(Message_size, m);
 return CRC.crc;
}
//---------------------------------------------------------------------
// Suma de comprobación
WORD TProtocolo::Cal_CRC_CE(packet* s)
{
 CRC_16 CRC;
 int Message_size;
 BYTE* m = (BYTE*) s;
 Message_size = sizeof(packet);
 CRC.crc = Check(Message_size, m);
 return CRC.crc;
}
//---------------------------------------------------------------------
/////Funcion para realizar la prueba CRC
WORD TProtocolo::Check(int Message_size, BYTE* m)
{
 CRC_16 CRC;
 BYTE Aux;
BYTE Message[sizeof(frame)];
for(int i = 0; i < Message_size; i++)
  Message[i] = m[i];
CRC.byte_crc[0] = Message[1];
CRC.byte_crc[1] = Message[0];
for(int i = 2; i < Message_size - 2; i++)
{
  Aux = Message[i];
  for(int j = 0; j < 8; j++)
  {
    if((0x8000 & CRC.crc))
    {
      CRC.crc = CRC.crc * 2;
      if(Aux & 0x80)
         CRC.crc = CRC.crc | 0x0001;
      CRC.crc = CRC.crc ^ Generator;
    }
    else
    {
      CRC.crc = CRC.crc * 2;
      if(Aux & 0x80)
         CRC.crc = CRC.crc | 0x0001;
      CRC.crc = CRC.crc ^ 0x0000;
    }
    Aux = Aux * 2;
  }
 }
 return CRC.crc;
}
//---------------------------------------------------------------------
/////Funcion para realizar la prueba CRC
WORD TProtocolo::CheckB(int Message_size, BYTE* m, bool a)
{
 CRC_16B CRC;
 BYTE Aux;
```

```
BYTE Message[sizeof(marco2)];
bool band = false, band1 = true;
int cont = 0;
Message[1] = m[0];
Message[0] = m[1];
for(int i = 2; i < Message_size; i++)
  Message[i] = m[i];
CRC.crc = Message[0];
while (cont < Message_size)
{
  for(int j = 0; j < 4; j++)
  {
   if((0x80 & CRC.crc))
       band = true;
   CRC.crc = CRC.crc * 2;
if (j == 3)
{
   if (band1)
   {
      cont++;
      Aux = Message[cont] & 0xF0;
      Aux = Aux / 16;
      CRC.crc = CRC.crc ^ Aux;
      band1 = false;
   }
   else
   {
      Aux = Message[cont] & 0x0F;
      CRC.crc = CRC.crc ^ Aux;
      band1 = true;
   }
}
if (band)
{
  CRC.crc = CRC.crc ^ 0x28;
  band = false;
}
    if ((cont == Message_size-2 && j == 3 && band1 && a) | |
    (cont == Message_size-1 && j == 3))
    {
       cont = Message_size;
       j = 4;
    }
   }
  }
 return CRC.crc;
}
//---------------------------------------------------------
// Envía un marco a la capa fisica-----------------------
void TProtocolo::to_physical_layer2(marco2 *s)
{
 int size_frame;
 BYTE* fr = (BYTE*)s;
 size_frame = sizeof(marco2);
 ACapaFisica(size_frame, fr);
}
//---------------------------------------------------------
////función para mandar los datos al buffer del serial
void TProtocolo::ACapaFisica(int size_frame, BYTE* fr)
{
 int n, i = 0, a, j, t = 0, k = 0, l = 0;
 BYTE c,b;
 int bs[8];
 auxiliar2.insert(auxiliar2.end(), FF);
 auxiliar2.insert(auxiliar2.end(), ETX);
 auxiliar2.insert(auxiliar2.end(), CERO);
 auxiliar2.insert(auxiliar2.end(), CERO);
  while (i<size_frame)
  {
     b = fr[i];
     j = 8;
     while (j)
     {
a = pow(2,j-1);
if (b > a-1)
{
   b-=a;
   *(bs+t) = 1;
   k++;
}
else
{
    *(bs+t) = 0;
    k = 0;
}
if (k == 5 && t < 7)
{
     t++;
     *(bs+t) = 0;
     k = 0;
}
if (t == 7)
{
    c = 0;
    n = 8;
    t = 0;
    while (n)
    {
       a = pow(2,n-1);
       if (*(bs+t) == 1)
          c += a;
       t++;
       n--;
    }
    auxiliar2.insert(auxiliar2.end(), c);
}
if (k == 5 && t == 8)
{
        inc3(t);
        *(bs+t) = 0;
        k = 0;
     }
     inc3(t);
     j--;
   }
   i++;
}
*(bs+t) = 0;
while (l<8)
{
   while (tl=7 && l < 6)
   {
     inc3(t);
     *(bs+t) = 1;
     l ++;
   }
   if (t == 7)
   {
     c = 0;
     n = 8;
     t = 0;
     while (n)
     {
       a = pow(2,n-1);
       if (*(bs+t) == 1)
          c += a;
       t++;
       n--;
     }
     auxiliar2.insert(auxiliar2.end(), c);
     if (l == 7)
       l = 8;
   }
  while (l == 6 && t != 7)
  {
   inc3(t);
```

```
      *(bs+t) = 0;
      if (t == 7)
        l = 7;
      }
   }
}
//---------------------------------------------------------------------
void TProtocolo::ACapaFisica3()
{
   MainForm->SerialPort->GetCommPort().Pon0RTS();
   Sleep(2); ///4
   MainForm->SerialPort->GetCommPort().WriteBuffer(auxiliar2.begin(),
auxiliar2.size());
   Sleep(4); //2/5
   MainForm->SerialPort->GetCommPort().Pon1RTS();
}
//---------------------------------------------------------------------
// Envía un marco a la capa fisica
void TProtocolo::to_physical_layer(frame *s)
{
  int size_frame;
  BYTE* fr = (BYTE*)s;
  size_frame = sizeof(frame);
  ACapaFisica2(size_frame, fr);
}
//---------------------------------------------------------------------
////función para mandar los datos al buffer del serial
void TProtocolo::ACapaFisica2(int size_frame, BYTE* fr)
{
  std::vector <BYTE> auxiliar;
  int n, i = 0, a, j, t = 0, k = 0, l = 0;
  BYTE c,b;
  int bs[8];
  MainForm->SerialPort->GetCommPort().Pon0RTS();
  Sleep(2);
  auxiliar.insert(auxiliar.end(), FF);
  auxiliar.insert(auxiliar.end(), ETX);
  auxiliar.insert(auxiliar.end(), CERO);
  auxiliar.insert(auxiliar.end(), CERO);
   while (i<size_frame)
   {
{
   b = fr[i];
   j = 8;
   while (j)
   {
     a = pow(2,j-1);
     if (b > a-1)
     {
        b-=a;
        *(bs+t) = 1;
        k++;
     }
     else
     {
         *(bs+t) = 0;
         k = 0;
     }
     if (k == 5 && t < 7)
     {
          t++;
          *(bs+t) = 0;
          k = 0;
}
if (t == 7)
{
    c = 0;
    n = 8;
    t = 0;
    while (n)
    {
       a = pow(2,n-1);
       if (*(bs+t) == 1)
          c += a;
          t++;
          n--;
       }
       auxiliar.insert(auxiliar.end(), c);
    }
    if (k == 5 && t == 8)
    {
          inc3(t);
          *(bs+t) = 0;
          k = 0;
    }
    inc3(t);
    j--;
  }
  i++;
}
*(bs+t) = 0;
while (l<8)
{
  while (t!=7 && l < 6)
  {
    inc3(t);
    *(bs+t) = 1;
    l ++;
  }
  if (t == 7)
  {
    c = 0;
    n = 8;
    t = 0;
    while (n)
    {
       a = pow(2,n-1);
       if (*(bs+t) == 1)
          c += a;
       t++;
       n--;
    }
    auxiliar.insert(auxiliar.end(), c);
    if (l == 7)
       l = 8;
  }
  while (l == 6 && t != 7)
  {
   inc3(t);
   *(bs+t) = 0;
   if (t == 7)
     l = 7;
  }
 }
  MainForm->SerialPort->GetCommPort().WriteBuffer(auxiliar.begin(),
auxiliar.size());
  Sleep(8);
  MainForm->SerialPort->GetCommPort().Pon1RTS();
}
//---------------------------------------------------------------------
// Obtiene un marco de la capa fisica (del serial)
void TProtocolo::from_physical_layer2(marco2 *s)
{
  int size_frame;
  BYTE* fr1 = (BYTE*)s;
  size_frame = sizeof(marco2);
  DeCapaFisica1(size_frame, fr1);
}
//---------------------------------------------------------------------
// Obtiene un marco de la capa fisica(del serial)
void TProtocolo::from_physical_layer1(marco2 *s)
{
  int size_frame;
  BYTE* fr1 = (BYTE*)s;
  size_frame = sizeof(marco2);
  DeCapaFisica(size_frame, fr1);
}
```

```
//---------------------------------------------------------------
///Función para leer el buffer del seria  copia de la unidadd
void TProtocolo::DeCapaFisica(int size_frame, BYTE *fr)
{
  BYTE CurrentByte, a;
  BYTE NextByte, c;
  int i=0, j, m = 2, k = 0,t = 0, l = 0;
  DWORD BytesAvailable;
  bool chido = false;
  int bs[8];
  int chinga = 0;
  BytesAvailable = MainForm->SerialPort->GetCommPort().BytesAvailable();
  while (!chido && BytesAvailable)
  {
     CurrentByte = MainForm->SerialPort->GetCommPort().GetByte();
     j = 8;
     while (j && !chido )
     {
       a = pow(2,j-1);
       if (CurrentByte > a-1)
       {
          CurrentByte -= a;
          *(bs+t) = 1;
          k++;
       }
       else
       {
          *(bs+t) = 0;
          if (k == 6 && m == 1)
          {
             m = 2;
             chido = true;
          }
          else
          {
          k = 0;
          m = 1;
          }
        }
        inc3(t);
        j--;
     }
     BytesAvailable = MainForm->SerialPort->GetCommPort().BytesAvailable();
     while (!BytesAvailable && chido && chinga < 4)
     {
        Sleep (1);
        chinga++;
        BytesAvailable = MainForm->SerialPort->GetCommPort().BytesAvailable();
     }
     chinga = 0;
  }
  if (chido && BytesAvailable)
  {
     if (j == 0 && t == 0)
     {
        j = 8;
       CurrentByte = MainForm->SerialPort->GetCommPort().GetByte();
     }
  }
  m = 1;
  k = 0;
  int hi = 0;
  while (i < (size_frame+1) && BytesAvailable && chido)
  {
    while (j && i < (size_frame+1) && chido)
    {
      a = pow(2,j-1);
      if (CurrentByte > a-1)
      {
         CurrentByte -= a;
         *(bs+t) = 1;
         k++;
         hi = 0;
      }
      else
      {
        *(bs+t) = 0;
        if (k == 6)
        {
          if (i)
           chido = false;
          else
           k = 0;
           hi = 1;
        }
        else if (k == 5)
        {
           k = 0;
           m = 0;
           hi = 0;
        }
        else
        {
           k = 0;
           hi = 0;
        }
      }
      if (m)
      {
        inc3(t);
        l++;
      }
      else
        m = 1;
      j--;
      if (l == 8)
      {
        if (k > 7)
        chido == false;
        c = 0;
        while (l && !hi)
        {
          a = pow(2,l-1);
          if (*(bs+t) == 1)
            c += a;
          inc3(t);
          l--;
        }
        if (i < size_frame && !hi)
        {
          fr[i] = c;
          if (i == size_frame-1)
          compt = true;
        }
        l = 0;
        if (!hi)
        i++;
      }
    }
    j = 8;
    BytesAvailable = MainForm->SerialPort->GetCommPort().BytesAvailable();
    while (!BytesAvailable && chido && i < size_frame+1 && chinga < 4)
    {
      Sleep (1);
      chinga++;
      BytesAvailable = MainForm->SerialPort->GetCommPort().BytesAvailable();
    }
    chinga = 0;
    if (BytesAvailable && chido && i < size_frame+1)
    {
      CurrentByte = MainForm->SerialPort->GetCommPort().GetByte();
    }
  }
 }
```

```
}
//---------------------------------------------------------------------
///Función para leer el buffer del seria  copia de la unidadd
void TProtocolo::DeCapaFisica1(int size_frame, BYTE *fr)
{
  BYTE CurrentByte, a;
  BYTE NextByte, c;
  int i=0, j, m = 2, k = 0,t = 0, l = 0;
  DWORD BytesAvailable;
  bool chido = false;
  int bs[8];
  int chinga = 0;
  BytesAvailable = MainForm->SerialPort->GetCommPort().BytesAvailable();
  while (!chido && BytesAvailable)
  {
     j = 8;
     CurrentByte = MainForm->SerialPort->GetCommPort().GetByte();
     while (j && !chido )
     {
       a = pow(2,j-1);
       if (CurrentByte > a-1)
       {
          CurrentByte -= a;
          *(bs+t) = 1;
          k++;
       }
       else
       {
          *(bs+t) = 0;
          if (k == 6 && m == 1)
          {
             m = 2;
             chido = true;
          }
          else
          {
          k = 0;
          m = 1;
          }
       }
       inc3(t);
       j--;
     }
     BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
     while (!BytesAvailable && chido && chinga < 4)
     {
        Sleep (1);
        chinga++;
        BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
     }
     chinga = 0;
  }
  if (chido && BytesAvailable)
  {
     if (j == 0 && t == 0)
     {
       j = 8;
       CurrentByte = MainForm->SerialPort->GetCommPort().GetByte();
     }
     m = 1;
     k = 0;
     int hi = 0;
     while (i < (size_frame+1) && BytesAvailable && chido)
     {
        while (j && i < (size_frame+1) && chido)
        {
          a = pow(2,j-1);
          if (CurrentByte > a-1)
          {
             CurrentByte -= a;
             *(bs+t) = 1;
             k++;
             hi = 0;
          }
          else
          {
             *(bs+t) = 0;
             if (k == 6)
             {
               if (i)
                chido = false;
               else
                k = 0;
                hi = 1;
             }
             else if (k == 5)
             {
                k = 0;
                m = 0;
                hi = 0;
             }
             else
             {
                k = 0;
                hi = 0;
             }
          }
          if (m)
          {
             inc3(t);
             l++;
          }
          else
             m = 1;
          j--;
          if (l == 8)
          {
             if (k > 7)
              chido == false;
             c = 0;
             while (l && !hi)
             {
                a = pow(2,l-1);
                if (*(bs+t) == 1)
                   c += a;
                inc3(t);
                l--;
             }
             if (i < size_frame && !hi)
             {
                fr[i] = c;
                if (i == size_frame-1)
                compt = true;
             }
             l = 0;
             if (!hi)
             i++;
          }
        }
        j = 8;
        BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
        while (!BytesAvailable && chido && i < size_frame+1 && chinga <
4)
        {
          Sleep (1);
          chinga++;
          BytesAvailable = MainForm->SerialPort-
>GetCommPort().BytesAvailable();
        }
        chinga = 0;
        if (BytesAvailable && chido && i < size_frame+1)
        {
          CurrentByte = MainForm->SerialPort-
>GetCommPort().GetByte();
        }
```

```
    }
  }
```

E.3 PROTOCOLO.H

```
//---------------------------------------------------------------------------
#ifndef ProtocoloH
#define ProtocoloH
#include <Classes.hpp>
#include "comm.h"
#include <vector>
#include <string>
#include <memory>
#define MAX_PKT 128
#define MAX_SEQ 1
typedef unsigned short seq_nr; /*numero de secuencia o
reconocimiento */
typedef struct { unsigned char data[MAX_PKT];} packet; /*
definicion de paquete*/

//---------------------------------------------------------------------------
///////Estructura del marco tipo DATOS
typedef struct
{
  WORD direcc;   /*localidad de la unidad*/
  seq_nr seq;  /* numero de secuencia*/
  unsigned short cont; /*contador de bytes de la trama*/
  bool last;
  packet info; /* paquete de la capa de red*/
  WORD CRC;
} frame;

typedef enum
  {
  listado = 8192,
  presente = 16384,
  final = 24576,
  ack1 = 32768,
  datos = 40960
  } tipo_marco; /* definicion de tipo de trama*/
//---------------------------------------------------------------------------
///////Estructura del marco ACK
typedef struct
{
  WORD direcc;
  BYTE seq;
  BYTE CRC;
} marco2;

//////////Estructura Para realizar CRC
typedef union{
  BYTE crc;
  BYTE byte_crc;
}CRC_16B;

typedef union{
  WORD crc;
  BYTE byte_crc[2];
}CRC_16;

const BYTE ETX = 254;
const BYTE CERO = 126;
const BYTE FF = 255;
const WORD Generator = 0x1025;
//---------------------------------------------------------------------------
class TProtocolo : public TThread
{
protected:
  void __fastcall Execute();
  void __fastcall SynchronizeNewBytes(void);
  void __fastcall SynchronizeNewBytes2(void);
  bool between(seq_nr a, seq_nr b, seq_nr c);
  unsigned int from_network_layer(packet *p);
  void MensajeHola( AnsiString Str);
  void mandar_trama(WORD profex,tipo_marco tipo);
  void send_fin(seq_nr frame_expected);
  void send_data(seq_nr frame_nr, packet buffer[], unsigned short
contador);
  WORD Cal_CRC_16(frame * s);
  WORD Cal_CRC_162(marco2* s);
  WORD TProtocolo::Cal_CRC_CE(packet* s);
  WORD Check(int Message_size, BYTE* m);
  WORD CheckB(int Message_size, BYTE* m, bool a);
  void to_physical_layer(frame *s);
  void to_physical_layer2(marco2 *s);
  void ACapaFisica2(int size_frame, BYTE* fr);
  void ACapaFisica(int size_frame, BYTE* fr);
  void ACapaFisica3();
  void from_physical_layer(frame *s);
  void from_physical_layer1(marco2 *s);
  void from_physical_layer2(marco2 *s);
  void DeCapaFisica(int size_frame, BYTE *fr);
  void DeCapaFisica1(int size_frame, BYTE *fr);
  void __fastcall dec_timer(void);
  void start_timer(/*seq_nr seq*/);
  void stop_timer(/*seq_nr seq*/);
  typedef enum
  {frame_arrival, cksum_err, t1imeout,
network_layer_ready}event_type;
  seq_nr seq_esp;
  packet buffer[7 + 1];   /* buffers para corriente de salida */
  unsigned int clock;
  int count[7 + 1];
  seq_nr nbuffered;          event_type event;
  bool retransmit;
//---------------------------------------------------------------------------
  unsigned int cont1;
  bool transms;
  bool compt;
  unsigned int cont;
//---------------------------------------------------------------------------
public:
  __fastcall TProtocolo();
  virtual __fastcall ~TProtocolo();
  marco2 R;
  marco2 P;
  seq_nr next_frame_to_send;
  AnsiString cont2;
  bool chole;
  WORD Prueba(int Message_size, BYTE* m, Byte Message[]);
  void Inicializacion(void);
  void IniciaBucle(void);
  std::vector <BYTE> m2_BufferIn;
  std::vector <BYTE> auxiliar2;
  bool camb_arch;
  bool turno;
};
#end
```

APÉNDICE F

F. INTERRUPTOR MESFET GaAs

F.1 IMPLEMENTACIÓN DEL INTERRUPTOR

Los transmisores TXM-900-HP-II de LINX están diseñados para que transmitan continuamente la portadora, aún cuando no tenga la necesidad de transmitir datos. Por lo tanto, cuando existe más de una estación dentro del área de recepción, se tienen problemas de interferencia entre ellos ya que emplean el mismo canal.

Para solucionar este problema es necesario agregar un interruptor MESFET de GaAs a cada sistema para desconectar la antena de los transmisores esclavos, cuando no estén autorizados por el maestro para transmitir datos.

El interruptor MESFET de GaAs se agrega al sistema MDEV-900-HP del esclavo, donde la finalidad es desconectar la antena de la salida del módulo transmisor (ver Figura F.1). El interruptor es controlado mediante el anfitrión. Este activa por *software* la bandera CTS a través del UART. Una vez que el sistema MDEV-900-HP recibe la señal, activa la compuerta AND que habilita el interruptor para la conmutación de un estado a otro. Cuando el sistema está como receptor, el transmisor tiene la salida aislada de la antena por lo que no existe la potencia suficiente para enviar la portadora. La configuración inicial del interruptor es con la antena desconectada, este cambia de posición sólo cuando el maestro permite al esclavo enviar su información.

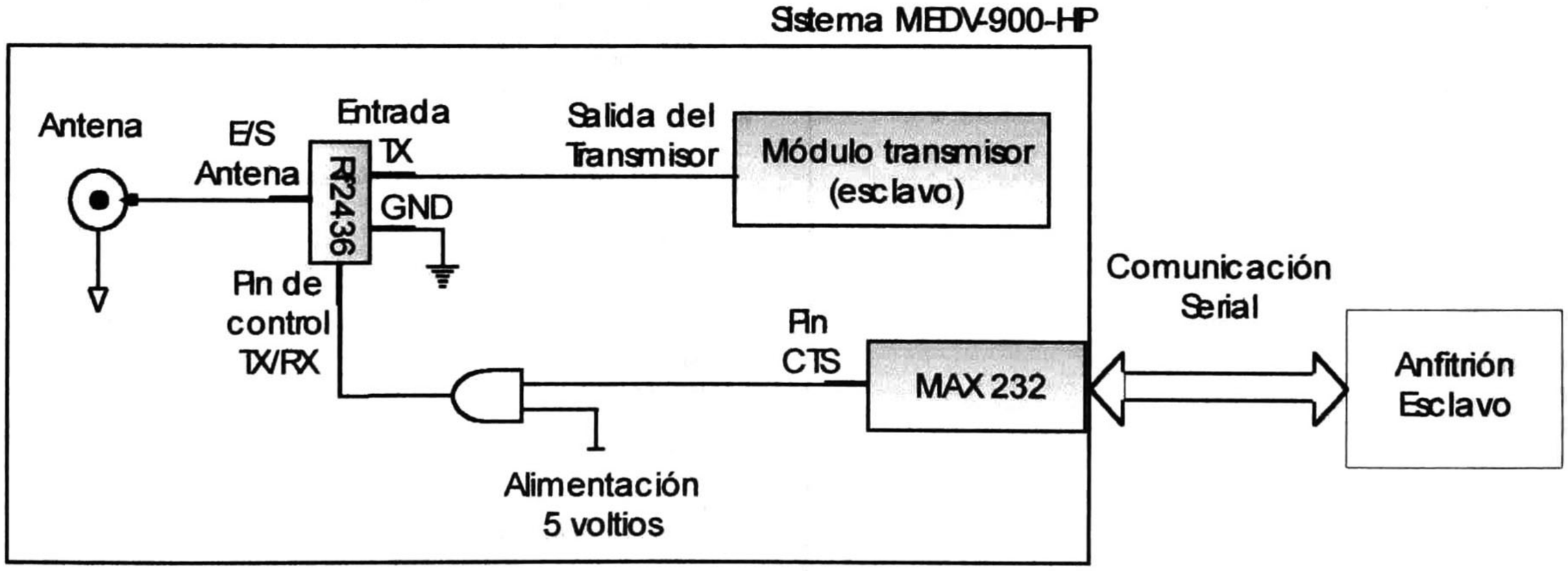


Figura F.1 Diagrama del circuito para controlar el interruptor RF2436.

C.2 CARACTERÍSTICAS DEL INTERRUPTOR MESFET DE GaAS

El RF2436 es un interruptor que conmuta entre el modo de transmisión y de recepción, utilizado en teléfonos y sistemas inalámbricos [1]. El dispositivo cuenta con las siguientes características:

- Soporta niveles de potencia hasta +28 dBm.
- El rango de frecuencia es de 0 hasta 2 GHz.
- Opera con niveles de alimentación mínimos de 1.5 V y máximo de 6 V para niveles lógicos CMOS en el control de la entrada (el nivel estándar es 3 V).
- No requiere de voltajes negativos.

- Consumo de corriente es muy bajo (5 μA en el modo de recepción y 0.5 mA en el modo transmisor).
- El índice de voltaje en una onda estacionaria para un canal activo (transmisor o receptor) es 1:1.
- La pérdida por inserción es de 0.5 dB a 900 MHz.
- El aislamiento entre canal es de 24 dB a 900 MHz.

En la Figura F. 2 se muestra el diagrama funcional del integrado RF2436.

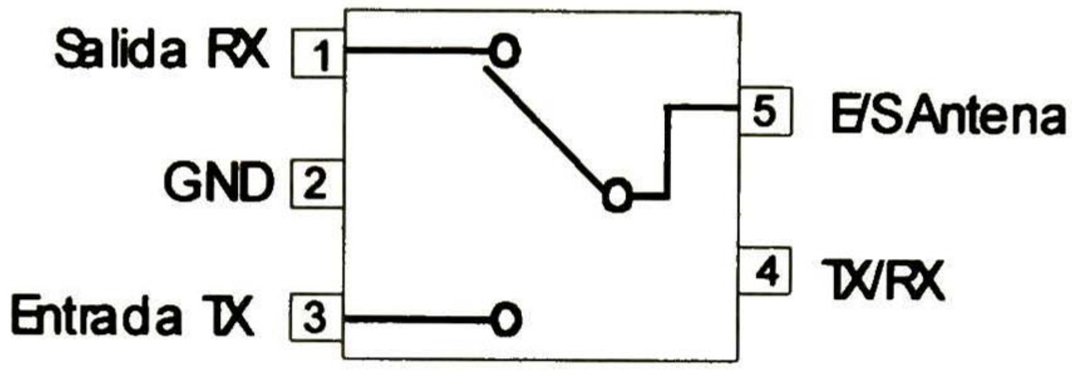


Figura F. 2 Diagrama funcional del IC RF2436.

F.3. BIBLIOGRAFÍA

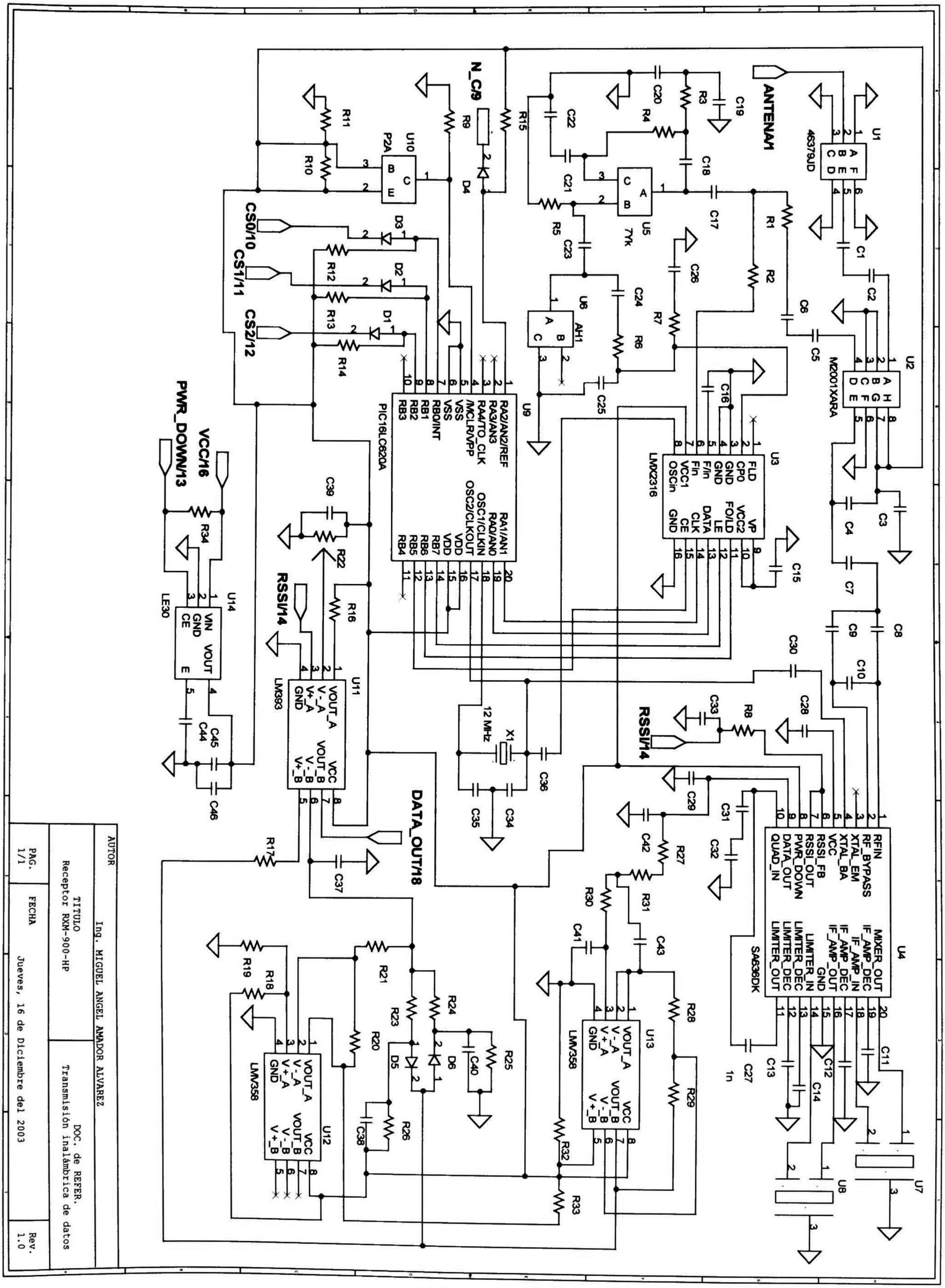
[1] Micro Devices RF, Inc., "RF2436 Transmit/Receive Switch", Rev A0 990118, January 06, 2000.

APÉNDICE G

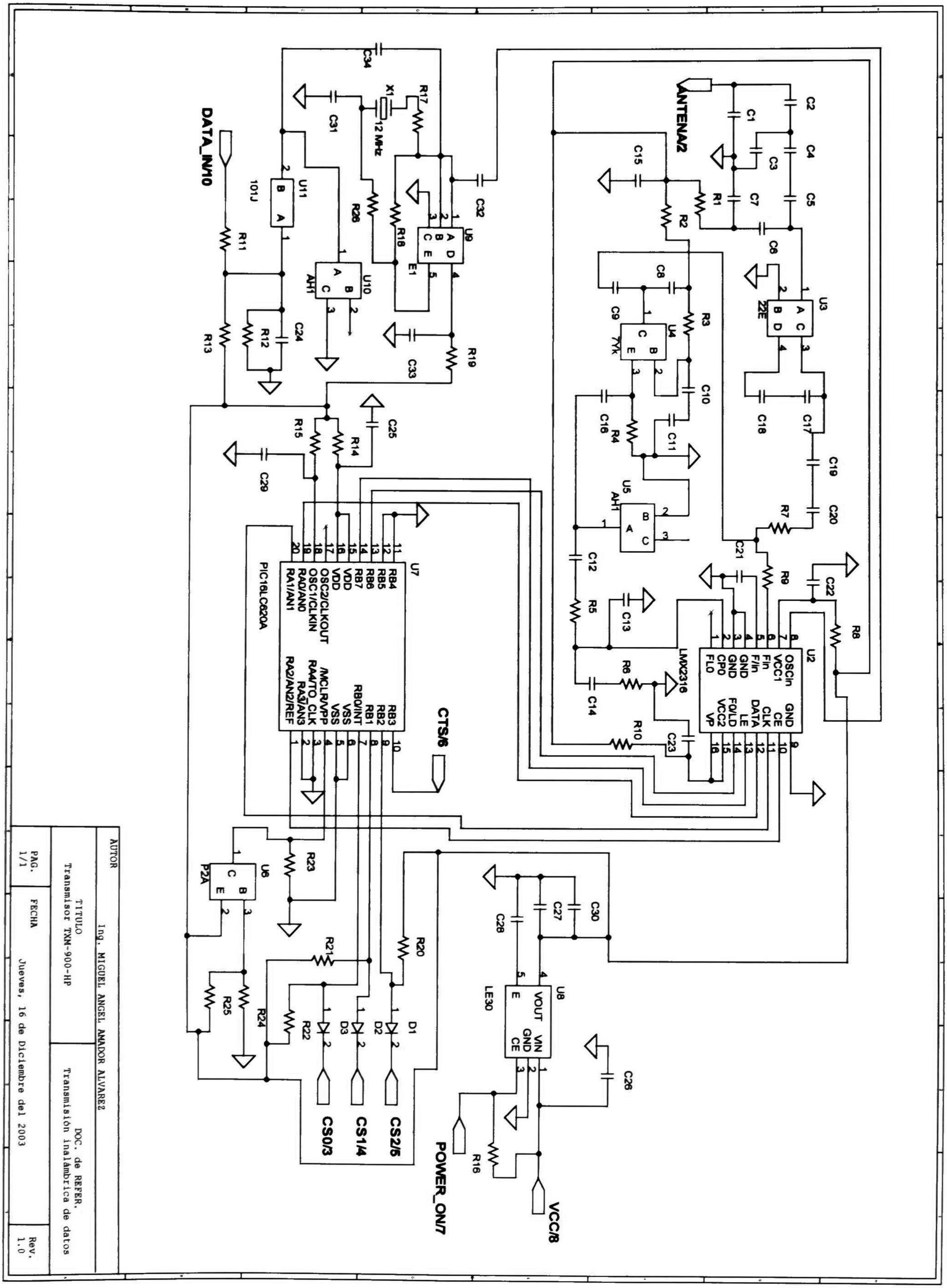
G. ESQUEMÁTICO DEL TRANSMISOR Y RECEPTOR

G.1 ESQUEMÁTICOS

En esta sección se muestra el esquemático del receptor RXM-900-HP-II y del transmisor TXM-900-HP-II utilizados en el presente trabajo de tesis.



AUTOR	Ing. MIGUEL ANGEL AMADOR ALVAREZ	
TITULO Receptor RXM-900-HP		DOC. de REFER. Transmisión inalámbrica de datos
PAG. 1/1	FECHA Jueves, 16 de Diciembre del 2003	Rev. 1.0


ANTENA2
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
C13
C14
C15
C16
C17
C18
C19
C20
C21
C22
C23
C24
C25
C26
C27
C28
C29
C30
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R17
R18
R19
R20
R21
R22
R23
R24
R25
R26
X1
12 MHz
DATA_IN/10
CTS/6
POWER_ON/7
VCC/8
CS2/5
CS1/4
CS0/3
LE30
LMX2316
PIC16LC620A
U2
U3
U4
U5
U6
U7
U8
U9
U10
U11
D1
D2
D3
AUTOR
Ing. MIGUEL ANGEL AMADOR ALVAREZ
TITULO
Transmisor TXM-900-HP
DOC. de REFER.
Transmisión inalámbrica de datos
PAG. 1/1
FECHA
Jueves, 16 de Diciembre del 2003
Rev. 1.0

Centro de Investigación y de Estudios Avanzados del IPN
Unidad Guadalajara

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó la tesis:

Transmisión Inalámbrica de Datos

del (la) C.

Miguel Angel AMADOR ALVAREZ

el día 27 de Enero de 2004.

Dr. Jose Luis LEYVA MONTIEL
Investigador Cinvestav 3B
CINVESTAV GDL
Jalisco

Dr. Deni Librado TORRES ROMÁN
Profesor Investigador 3A
CINVESTAV GDL
Jalisco

Dr. Federico SANDOVAL IBARRA
Investigador Cinvestav 3A
CINVESTAV GDL
Jalisco