VNIVERSIDAD NACIONAL AVTNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

PROGRAMA DE MAESTRIA Y DOCTORADO EN INGENIERIA

FACULTAD DE INGENIERIA

DISEÑO DE UN SERVIDOR WEB EMBEBIDO

T E S I S

QUE PARA OPTAR POR EL GRADO DE

MAESTRO EN INGENIERIA

INGENIERIA ELECTRICA - SISTEMAS ELECTRONICOS

P R E S E N T A :

MIGUEL ANGEL MONTAÑO MEDINA

TUTOR:
M.I. ANTONIO SALVA CALLEJA

2006

JURADO ASIGNADO:

Presidente: Dr. Rangel Licea Víctor

Secretario: M.I. Bañuelos Saucedo Miguel Ángel

Vocal: M.I. Salvá Calleja Antonio

1er. Suplente: Dr. Gómez Castellanos Javier

2do. Suplente: M.I. Valeriano Assem Jorge

Lugar o lugares donde se realizó la tesis:

México, D.F.

TUTOR DE TESIS:

M.I. Antonio Salvá Calleja

_____

FIRMA

En memoria de
Esperanza Montaño Esquivel

Gracias por la educación, el amor y el tiempo que me dedicaste

Agradecimientos

Quiero agradecer

A mis padres por el apoyo que me han proporcionado al realizar estos estudios.

A Keyla por estar siempre a mi lado apoyándome y motivándome a terminar este trabajo.

A todos mis amigos del bloque 6 por su amistad y apoyo.

Al CONACYT por el apoyo económico que me permitió realizar este trabajo.

Al Maestro Antonio Salvá por compartir conmigo sus conocimientos, su experiencia y su tiempo al realizar este trabajo.

A la UNAM por la educación que tan generosamente me brindó.

Miguel Angel Montaño

Índice general

Índice de figuras

Índice de cuadros

Introducción

Internet es una red de redes, a escala mundial, de computadoras interconectadas entre sí, que transmiten datos utilizando el protocolo estándar *internet protocol* (IP), además de otros protocolos. Se compone de miles de pequeñas redes comerciales, domésticas, escolares y gubernamentales. El servicio más conocido que ofrece internet es el de *World Wide Web*.

La web es un servicio de información a gran escala que permite a un usuario buscar información; ofrece un sistema de hipermedios para almacenar recursos como texto, imágenes y cualquier otro archivo de computadora. Para ver la información se utiliza un navegador web para extraer los recursos de los servidores web y mostrarlos en la pantalla del usuario.

Un servidor web es un programa que implementa el protocolo HTTP (*hypertext transfer protocol*). Se encarga de mantenerse a la espera de peticiones HTTP llevada a cabo por un cliente HTTP (navegador web). El navegador realiza una petición al servidor y éste le responde con el contenido solicitado.

Los servidores web pueden generar contenido estático o dinámico. El contenido estático es cualquier archivo que no cambia su contenido entre una petición y otra; por ejemplo, un archivo de texto, una imagen, una animación, etc. Las páginas web de los servicios de noticias; de las respuestas de los buscadores de internet (google, yahoo, etc.); de los servicios financieros (bolsa, cotizaciones, etc.), son ejemplos de contenido dinámico ya que son generados en el momento de responder a la petición del navegador.

Otro ejemplo de contenido dinámico que puede generar un servidor web es el de un mapa de nuestra casa que indique que luces están prendidas en cada habitación, así como el nivel de agua en la cisterna y el tinaco. Para que un servidor web pueda generar este tipo de información debe contar con la ayuda de dispositivos electrónicos externos que le proporcionen los datos necesarios. Una manera de diseñar este sistema sería tener una computadora personal en nuestro hogar conectada a internet enlazada a una serie de sensores por toda la casa. Otra forma sería brindar de mayor inteligencia a los dispositivos del hogar de manera que se pueda acceder desde internet a cada uno de ellos, sin la necesidad de una PC. De esta manera cada dispositivo que se desea controlar estaría conectado a internet, ya sea independientemente (el sistema de iluminación y alarma contra robo) o en un grupo (todos los electrodomésticos de un mismo fabricante).

Algunos dispositivos o sistemas del hogar requerirán de un servidor web para su control y monitoreo, esto plantea la necesidad de que los dispositivos sean *internet-aware*, o sea que tengan la capacidad de acceso desde internet.

Para que un dispositivo pueda conectarse a internet, debe contar con algún medio de acceso a la red: ethernet, wifi, modem (ppp, slip), etc. Además debe soportar el conjunto de protocolos TCP/IP y en el caso de implementar un servidor web debe soportar el protocolo HTTP.

El objetivo de este trabajo fue desarrollar un servidor web embebido en un microcontrolador y crear con ello un componente reusable que permita que nuevos diseños cuenten

con la capacidad de red.

El servidor web en esta tesis comprendió el diseño y la implementación de una pila de protocolos TCP/IP (uinet), el diseño y programación de un servidor HTTP sobre la pila desarrollada (Petit), y el diseño y construcción de un prototipo de hardware basado en un microcontrolador para crear el bloque funcional.

La pila uinet se encarga de manejar la comunicación en internet y el servidor Petit de atender a las peticiones de los usuarios del servidor web. El servidor web se diseñó a partir de la tarjeta de desarrollo Easy Ethernet y se creó un hardware complementario a ésta, para agregar las características necesarias para la ejecución del servidor Petit.

Como ejemplo de aplicación y demostración del servidor web, se diseñó también una red de sensores inalámbricos de temperatura. La red cuenta con 4 nodos y una estación base. Los nodos se sitúan en los lugares donde se desea medir la temperatura y a una distancia no mayor de 10 metros de la estación base, la cuál se conecta a una red ethernet. Cada nodo transmite sus mediciones a la estación base en un tiempo establecido; cuando ésta cuenta con las mediciones de todos los nodos las transmite, a través de la red, a los usuarios que estén monitoreando el sistema. Cualquier computadora conectada a la red puede acceder a la estación base desde un navegador web.

La red de sensores se opera desde una applet[1] que captura y despliega las mediciones de los sensores en pantalla y le permite al usuario guardar los datos para un análisis posterior.

Un número, virtualmente, ilimitado de usuarios pueden al mismo tiempo sin que se vea afectado el desempeño general, ya que el sistema puede enviar las mediciones de los sensores a varios usuarios a la vez.

En esta aplicación el servidor web se encarga de las siguientes tareas:

- Atender las peticiones HTTP de los navegadores. Envía las páginas web solicitadas por cada cliente, así como los recursos enlazados (applets, imagenes, etc.).
- Transmite las mediciones de los sensores a los clientes utilizando el protocolo UDP (*User Datagram Protocol*), y las extensiones multicast del protocolo IP.
- Responde a las solicitudes de los protocolos ICMP (*internet control message protocol*) y ARP (*address resolution protocol*). Responde a las solicitudes del programa *ping*[2] y a la solicitud de la dirección IP por parte de otras computadoras.

Este trabajo está integrado en 6 capítulos que se resumen a continuación:

- Capítulo 1: Es una introducción a los protocolos implementados en la pila TCP/IP de este trabajo en donde se describe la manera en que éstos funcionan y la relación de cada uno con el modelo de internet.
- Capítulo 2: En este capítulo se presenta el hardware que compone al servidor web: la tarjeta de desarrollo Easy Ethernet y el hardware complementario que se desarrolló para la misma.
- Capítulo 3: Aquí se describe la implementación realizada de los protocolos de la pila TCP/IP uinet. Trata sobre como funciona el software de la pila uinet y la manera en que se implementó. Al final del capítulo se presenta una comparación de la pila con otros trabajos relacionados.

[1] Un programa escrito en el lenguaje Java que se ejecuta en el navegador web del usuario.

[2] *Ping* es una utilería que se encuentra presente en la mayoría de los sistemas operativos y sirve para detectar la presencia de un dispositivo en la red.

- Capítulo 4: Este capítulo describe el diseño y la implementación del servidor HTTP Petit. De igual manera que con el capítulo 3, éste trata sobre los detalles del software, su diseño y su implementación. Finaliza también con una comparativa del servidor Petit con otros trabajos relacionados.

- Capítulo 5: En este capítulo se describe la red de sensores inalámbricos desarrollada como ejemplo de aplicación del servidor web. Se describen los componentes de la red, tanto el software como el hardware de los prototipos desarrollados.

- Capítulo 6: Este capítulo es una guía de operación y puesta en marcha de los prototipos de la red de sensores inalámbricos.

- Capítulo 7: Aquí se presentan las conclusiones obtenidas al desarrollar el servidor web.

Esta tesis viene acompañada por un disco compacto en donde se encuentra el código fuente del software desarrollado, así como los diagramas esquemáticos del hardware desarrollado.

Capítulo 1

Introducción a TCP/IP

En este capítulo se hace una descripción general del funcionamiento de la pila de protocolos TCP/IP, con la finalidad de ofrecer una visión general de los protocolos que la conforman e introducir los términos utilizados a lo largo de este trabajo.

Este capítulo comienza describiendo el modelo de internet de cuatro capas, continuando con el concepto de encapsulamiento de la pila de protocolos. Posteriormente se comenta brevemente el esquema de direcciones de internet, para pasar a la descripción general de los protocolos implementados por la pila TCP/IP de éste trabajo. Los detalles del diseño y la implementación de éstos se presentan en el capítulo 3.

La descripción formal de los protocolos está fuera del alcance de este trabajo. En la colección de documentos *Request for Comments* (RFC) se pueden encontrar las especificaciones formales de los protocolos descritos en este capítulo.

1.1. El modelo de internet

El conjunto de protocolos TCP/IP permite que computadoras de todos los tamaños, incluso con sistemas operativos distintos, se puedan comunicar entre sí. Estos protocolos forman la base de lo que es Internet. TCP/IP es una combinación de diferentes protocolos en varios niveles. Generalmente se considera que TCP/IP es un sistema de cuatro niveles como se muestra en la figura 1.1.

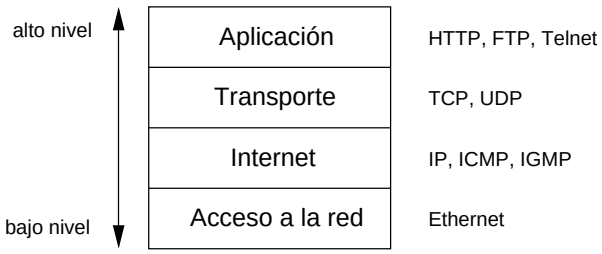


Figura 1.1: Modelo de referencia de internet

Cada capa o nivel soluciona una serie de problemas relacionados con la transmisión de datos, y proporciona un servicio bien definido a los niveles más altos. Los niveles superiores son los más cercanos al usuario y tratan con datos más abstractos, dejando a los niveles más bajos la labor de traducir los datos de forma que se puedan manipular físicamente.

1.1.1. Capa de acceso a la red

Es la capa de más bajo nivel en el modelo de internet. Contiene los protocolos necesarios para que los dispositivos conectados a una red puedan comunicarse entre sí. Los protocolos en esta capa realizan tres tareas principales:

- Definen como se usa la red para transmitir un paquete de datos a través de una conexión física.
- Intercambian datos entre un dispositivo y la red física.
- Entregan datos entre dos dispositivos en la misma red. Para hacer la entrega de datos en la red local, se utilizan las direcciones físicas de los nodos. Éstas direcciones están almacenadas en la interfaz de red del dispositivo y son asignadas por el fabricante.

Por el contrario a las capas de más alto nivel, los protocolos de la capa de red deben comprender los detalles de la red física así como la estructura de los paquetes, el esquema de direcciones utilizado, el tamaño máximo de los datos a transmitir, etc. Esto asegura que los protocolos puedan dar formato a los datos de manera que se puedan transmitir correctamente en la red. En este nivel se encuentra Ethernet.

1.1.2. Capa de internet

Esta capa es responsable de mandar los mensajes a través de las redes internet. En este nivel se encuentra el *internet protocol* (IP), que provee el servicio de red de datagramas. Los datagramas son paquetes de información que contienen una cabecera, datos y una cola (*trailer*). La cabecera contiene información como la dirección del destino, necesaria para encaminar el paquete por la red. La cola por lo general, pero no en el caso de IP, contiene el valor de la suma de verificación que se utiliza para asegurar la integridad de los datos.

El IP empaqueta un mensaje en un datagrama y lo envía por la red. El servicio de datagramas de IP, no soporta el concepto de sesión o conexión. Una vez que el mensaje es enviado o recibido, el servicio no retiene información de la entidad con la que se comunicó. Si ésta información es requerida se puede encontrar en los protocolos de transporte. Las habilidades de retransmitir datos y verificar errores son mínimas o no existentes en los servicios de datagramas. Si se detecta un error en la recepción de un datagrama, éste se descarta sin notificar a la capa de nivel superior.

Los protocolos *Internet Control Message Protocol* (ICMP) e *Internet Group Management Protocol* (IGMP) también se ubican en esta capa. El ICMP se utiliza para detectar y reportar errores de conexión entre los huéspedes, mientras que IGMP es un protocolo utilizado entre los huéspedes y los gateways para administrar los grupos de multidifusión en internet y el tráfico de información entre los miembros de grupo, como se explica en la sección 1.6.

1.1.3. Capa de transporte

En esta capa se encuentra el *transfer control protocol* (TCP), y es el responsable de mantener un flujo de datos seguro y confiable entre dos anfitriones, garantizando la entrega de datos. Para lo anterior, el TCP provee varios mecanismos, como el uso de números de secuencia y retransmisiones de los datos.

El TCP provee funciones para abrir y cerrar una conexión, así como para enviar y recibir datos por la misma. Una conexión o circuito virtual, es el estado del protocolo de transporte,

entre el tiempo en que una entidad receptora acepta el comando abrir, y el tiempo en que se ejecuta el comando cerrar por alguna de las dos partes.

Adicionalmente al TCP, en este nivel se encuentra también el protocolo *user datagram protocol* (UDP). Éste provee un servicio mucho más simple a la capa de aplicación: envía datagramas entre dos anfitriones, sin garantía de que los datagramas lleguen a su destino. Las aplicaciones que emplean este protocolo deben encargase de la seguridad y confiabilidad en la transmisión.

1.1.4. Capa de aplicación

Esta capa provee los servicios que las aplicaciones de usuario utilizan para comunicarse a través de la red. Este nivel abarca a todos los protocolos de aplicación que utilizan a los protocolos de transporte para enviar datos. Los procesos que se realizan sobre los datos del usuario como la encripción y desencripción; la compresión y descompresión también pueden ser parte de esta capa.

En el nivel más alto, los usuarios llaman a una aplicación que accesa los servicios disponibles en internet. Una aplicación interactúa con uno de los protocolos de transporte para enviar o recibir datos. Cada programa de aplicación selecciona el tipo de transporte necesario, ya sea un flujo de datos constante (TCP) o una secuencia de mensajes individuales (UDP). El programa de aplicación pasa los datos, de la forma requerida, al protocolo de transporte para su entrega.

Para que las aplicaciones puedan intercambiar datos deben coincidir en el formato de los mismos. La capa de aplicación es responsable de estandarizar la presentación de los datos. En este nivel se pueden encontrar varios ejemplos de protocolos: HTTP, FTP, Telnet, etc.

1.2. Encapsulación de datos

Cuando una aplicación envía datos utilizando TCP, los datos pasan por cada uno de los niveles de la pila de protocolos TCP/IP, hasta que son enviados como un flujo de bits por la red. En cada nivel se agrega información a los datos preponiendo una cabecera y agregando una cola a los datos que se reciben. En la figura 1.2 se ilustra el proceso.

Cada protocolo agrega en la cabecera que genera un identificador para indicar a que capa pertenecen los datos. En el caso de IP y de ethernet se hace almacenando un valor en el campo de protocolo de la cabecera, de 8 y 16 bits respectivamente.

En la capa de transporte el identificador indica la aplicación a la que pertenecen los datos. Tanto en TCP como en UDP el identificador es un número de puerto de 16 bits. Ambos almacenan el número de puerto de origen y destino en sus respectivas cabeceras.

Existe una serie de números de puertos *bien conocidos*, en donde se asocia una aplicación conocida. Por ejemplo un servidor de archivos FTP tiene asignado el puerto TCP 23, mientras que un servidor web tiene el puerto TCP 80. La mayoría de las implementaciones de TCP/IP alojan puertos transitorios, que se usan para iniciar una conexión, entre el 1024 y el 5000. Los números de puertos mayores al 5000 están pensados para otros servicios que no son tan bien conocidos a través de internet. Los números de puerto bien conocidos son administrados por el *Internet Assigned Numbers Authority* (IANA).

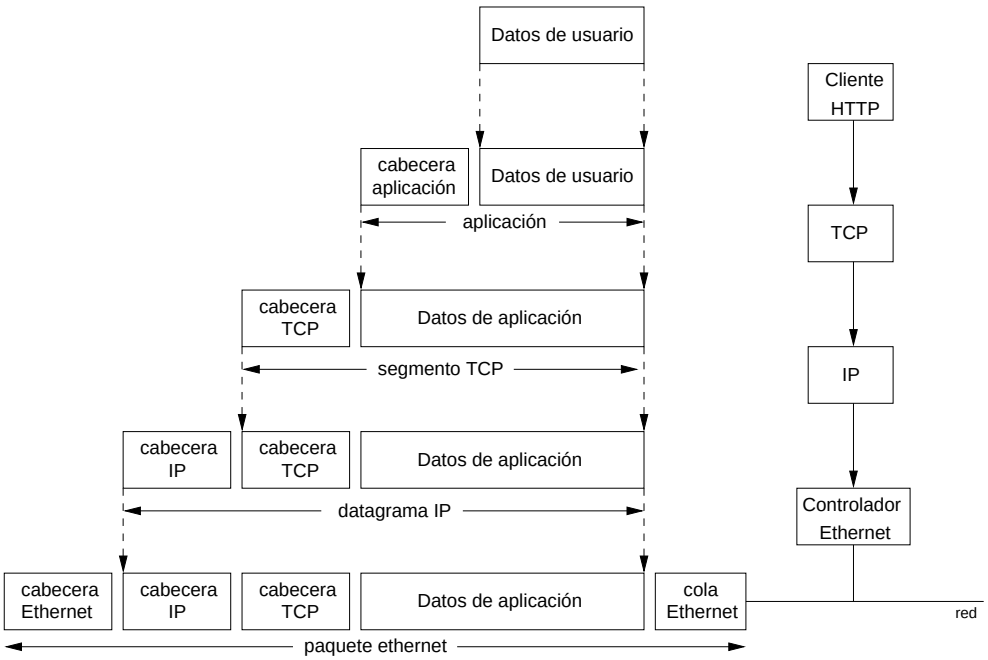


Figura 1.2: Encapsulado de datos al pasar por la pila TCP/IP

1.3. Direccionamiento en internet

Cada interfaz en internet tiene asignada una dirección IP única. La dirección es un número entero de 32 bits. La figura 1.3 muestra la estructura de las direcciones, así como las cinco clases de direcciones en que se clasifican.

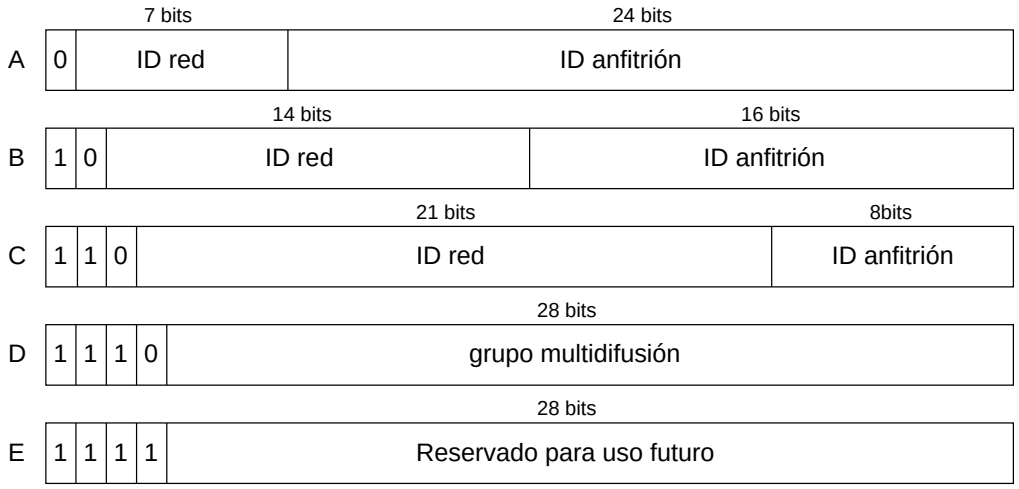


Figura 1.3: Las cinco clases de direcciones IP

Las direcciones IP normalmente se escriben como cuatro números decimales separados por puntos, cada número es un byte de la dirección. Como cada interfaz debe tener una dirección IP única, debe existir una autoridad para designar las direcciones de las redes conectadas a internet, esta es la *Internet Network Information Center* (NIC).

Conceptualmente cada dirección IP es un par (id red, id anfitrión), en donde id red identifica a una red e id anfitrión identifica a un anfitrión dentro de la red. Por ejemplo, la dirección IP 192.168.1.150 identifica al anfitrión 150 de la red 192.168.1.0. Un anfitrión puede ser cualquier dispositivo conectado a internet, como una computadora personal, un servidor dedicado, una impresora de red, etc.

En la práctica una dirección IP puede ser de clase A, B o C.La dirección 192.168.1.150 es una dirección de clase C según la clasificación mostrada en la figura 1.3. Según los tres bits más significativos se puede saber de que clase es una dirección IP. El rango de direcciones que abarca cada clase se muestra en la tabla 1.1.

Clase	Rango
A	0.0.0.0 a 127.255.255.255
B	128.0.0.0 a 191.255.255.255
C	192.0.0.0 a 223.255.255.255
D	224.0.0.0 a 239.255.255.255
E	240.0.0.0 a 255.255.255.255

Cuadro 1.1: Rango de las diferentes clases de direcciones IP

En las siguientes secciones se describe brevemente cada uno de los protocolos que conforman a la pila TCP/IP uinet que se desarrolló en este trabajo.

1.4. Protocolo IP

El protocolo IP provee un servicio de entrega de datagramas no asegurado, es decir no hay garantía de que un datagrama IP llegue siempre y correctamente a su destino. Cuando se detecta algún error en la recepción de un datagrama, IP cuenta con un algoritmo muy sencillo: descartar el datagrama e intentar enviar un mensaje ICMP de regreso al emisor. Las capas superiores se encargan de asegurar la entrega de datagramas. Cada datagrama es manejado de manera distinta, lo que puede provocar que los datagramas sean entregados fuera de orden.

Durante la transmisión de los datagramas IP por internet, éstos pueden transitar por uno o varios gateways. Los gateways o *routers* son dispositivos que conectan una red con otra. Se puede dar el caso de que un datagrama sea muy largo como para pasar por una red en específico. Por ello, los datagramas IP se pueden dividir en fragmentos más pequeños para pasar por una red que así lo requiera, a este proceso se le conoce como fragmentación de datagramas. Los gateways son los encargados de fragmentar un datagrama cuando así se requiera. Cuando un anfitrión recibe un datagrama fragmentado, lo reensambla para formar el datagrama original antes de pasarlo a un protocolo de mayor nivel como TCP o UDP.

Cabe mencionar aquí, que el módulo IP implementado en este trabajo no soporta la fragmentación de datagramas, debido a que esto aumentaría los recursos necesarios de memoria RAM en el microcontrolador utilizado. Aunque esto podría parecer una limitación importante, en la práctica ya casi no se presenta la fragmentación de datagramas, debido a que las redes actuales cuentan con mayores capacidades que las existentes en el tiempo de la especificación del protocolo IP.

Versión (4 bits). La versión del protocolo IP. La versión actual es la 4, por ello IP también se conoce como IPv4.

IHL (4 bits). *Internet header length*. La longitud de la cabecera IP expresado en palabras de 32 bits. El valor mínimo es de 5. Este campo sirve para saber en donde comienzan los datos.

Tipo de servicio (8 bits). Este campo provee una indicación de los parámetros abstractos sobre la calidad del servicio deseado. Algunas redes proveen un servicio de red con precedencia, es decir pueden diferenciar el tráfico más importante y transmitirlo primero. Si el valor de este campo es 0 significa servicio normal. La descripción completa de este campo se encuentra en la especificación de IP [2].

Longitud (16 bits). Este campo indica la longitud total del datagrama IP en bytes. Con este campo y el IHL se puede saber cuantos datos contiene el datagrama. El tamaño máximo de un datagrama es 65536 bytes.

Identificación (16 bits). Este campo contiene un valor asignado por el emisor que identifica al datagrama. Normalmente se incrementa en 1 cada vez que se envía un datagrama.

Flags (3 bits). Banderas de control, indican si un datagrama se puede fragmentar en unidades mas pequeñas y si el datagrama actual está fragmentado o no.

Offset (13 bits). Este campo indica a que parte del datagrama pertenece este fragmento. Se mide en unidades de 64 bits, el primer fragmento tiene el offset 0.

TTL (8 bits). *Time to live*. Este campo sirve para limitar el tiempo de vida de un datagrama. El emisor fija el valor de este campo. Cada vez que el datagrama atraviesa por un gateway este valor se decrementa en 1. Cuando el valor llega a cero, el datagrama se descarta y el emisor es notificado con un mensaje ICMP. Esto permite limitar la propagación de un datagrama y a la vez que el paquete circule indefinidamente por la red.

Protocolo (8 bits). Este campo indica el protocolo, del siguiente nivel, de la porción de datos del datagrama. Para TCP este campo vale 0x06 y para UDP vale 0x11.

Checksum (16 bits). Este campo es la suma de comprobación de la cabecera IP.

Dirección origen (32 bits). La dirección IP del emisor del datagrama.

Dirección destino (32 bits). Ladirección IP del destino del datagrama.

Opciones (longitud variable). Las opciones pueden o no aparecer en los datagramas. Raramente se utilizan. En este trabajo no se usan ni se implementan.

Figura 1.4: Formato de la cabecera IP

En la figura 1.4 se muestra el formato de la cabecera IP. La cabecera mide normalmente 20 bytes, a menos que se encuentren presentes las opciones. Se puede apreciar en la misma figura que la cabecera está ordenada en bloques de 4 bytes, donde el bit menos significativo está numerado con un 0, y el más significativo como 31. Los bytes de la cabecera se transmiten en el siguiente orden: bits del 0 al 7, luego del 8 al 15 y así hasta terminar. Este orden se conoce como *Network byte order* y es equivalente al *big-endian byte order*, y es usado para transferir todos los bytes de las cabeceras TCP/IP por la red.

Cada datagrama IP cuenta con una suma de verificación para la detección de errores, esta suma "protege" solamente a la cabecera IP; los protocolos TCP y UDP proveen su propia suma de verificación.

El algoritmo para calcular la suma de verificación consiste en agrupar los datos del datagrama en palabras de 16 bits. Si el número de bytes en el datagrama es impar, se agrega un byte al final igual a 0x00 y se procede a agrupar las palabras. Se realiza la suma en complemento a uno de todas las palabras, y el complemento a uno de la suma es el valor que se coloca en el campo de checksum. Para realizar el cálculo, el valor del campo de checksum es cero. La ventaja de este algoritmo es su simplicidad al realizar la verificación en el receptor: se comprueba sumando todas las palabras de 16 bits en complemento a uno (incluyendo el campo de checksum) y comprobando que el resultado sea 0xFFFF.

1.5. Protocolo ICMP

Ocacionalmente un gateway o un anfitrión destino se comunica con el anfitrión de origen para reportar un error en el procesamiento de un datagrama, por ejemplo cuando un anfitrión se ha desconectado sorpresivamente de la red, o cuando un enlace entre dos redes se rompe debido a la falla en un gateway. Para estos propósitos se utiliza el protocolo *Internet Control Message Protocol* (ICMP).

ICMP usa el soporte básico de IP como si fuese un protocolo de más alto nivel; sin embargo, ICMP es una parte integral de IP. Los mensajes ICMP son enviados en diferentes situaciones: cuando un datagrama no puede llegar a su destino, cuando un gateway no cuenta con la memoria suficiente para procesar y redireccionar un paquete, entre otras.

El protocolo IP no se diseñó para ser ciento por ciento confiable. El propósito de los mensajes ICMP es informar sobre los problemas en el entorno de comunicación, mas no para hacer a IP confiable. Algunos datagramas pueden perderse sin que se genere un reporte ICMP sobre el hecho. Los protocolos de más alto nivel, como TCP, son los que se encargan de asegurar que se cree un canal de comunicación confiable.

Para prevenir un ciclo infinito de generación de mensajes, no se envían mensajes ICMP acerca de otros mensajes ICMP.

El protocolo ICMP puede enviar los siguientes mensajes:

Echo (petición y respuesta): Este es un mensaje de prueba y se usa para determinar si un anfitrión está activo en la red. Funciona enviando una petición Echo con datos aleatorios al anfitrión, y si éste está activo debe contestar con los mismos datos que se le enviaron en una respuesta Echo. La petición es generada normalmente por la utilería `ping`.

Destination Unreachable: Este es un mensaje que puede ser enviado tanto por los gateways como por los anfitriones destino. Indican que el datagrama no puede ser entregado o procesado por el anfitrión de origen. Las causas por las que un gateway

podría enviar este mensaje son: la red no está disponible, el destino es inalcanzable, no pudo llevarse a cabo la fragmentación o se presentó una falla en el encaminamiento del datagrama. Si el huésped genera este mensaje será porque el puerto solicitado no está disponible o porque el protocolo señalado en la cabecera IP no es soportado.

Source Quench: Si un gateway recibe un datagrama y no tiene espacio suficiente en memoria como para poner el datagrama en la cola de salida para la siguiente red, descarta el datagrama y podría enviar este mensaje al emisor del datagrama. Un anfitrión destino también podría enviar este tipo de mensaje si los datagramas llegan demasiado rapido como para procesarlos. Cuando un anfitrión recibe este mensaje debe limitar la velocidad a la que está enviando los datagramas hasta que deje de recibir este tipo de mensajes.

Redirect: Cuando un gateway (G1) recibe un datagrama, examina las rutas que puede tomar el datagrama en su camino hacia la red X. Como resultado, selecciona la dirección del siguiente gateway (G2) . Si el gateway G2 se encuentra en la misma red que el huésped que envió el datagrama, el gateway G1 envía este mensaje al huésped indicando que envíe el trafico directamente al gateway G2. El gateway G1 reenvía el datagrama original hacia el gateway G2.

Time Exceeded: Cuando el tiempo de vida de un datagrama (TTL) llega a cero al ser procesado por un gateway, este puede notificar al emisor con este tipo de mensaje.

Parameter Problem: Si un gateway o un anfitrión destino encuentra un error al procesar los parámetros de la cabecera IP que impida el procesamiento del datagrama, este se descarta. Este problema se presenta cuando se indican argumentos incorrectos en las opciones de la cabecera. Si el error hace que el datagrama se descarte, se envía este mensaje.

Timestamp (petición y respuesta): Este mensaje es similar al mensaje de echo, salvo que en lugar de datos aleatorios se envía un número de 32 bits con la cuenta del número de milisegundos transcurridos desde la media noche.

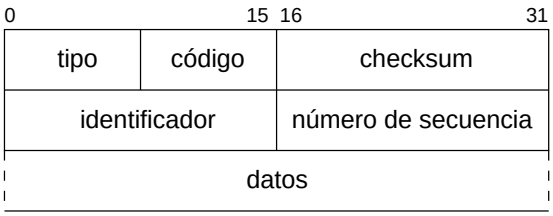
Information (petición y respuesta): Estos mensajes se usan para que un huésped pueda saber los números de la red en la que se encuentra.

Los mensajes ICMP se envían utilizando la cabecera básica IP especificando en el campo de protocolo el valor 0x01.

Para propósitos de este trabajo solo se implementaron los mensajes de *Echo* y *Destination Unreachable*. Por ser los que se presentan con mayor frecuencia en la comunicación real. Todos los demás mensajes ICMP son ignorados por el módulo de IP de la pila uinet.

El formato del mensaje ICMP *Echo* se muestra en la figura 1.5. Este formato se utiliza al enviar mensajes *Echo* y *echo reply*. La pila uinet está diseñada para proveer las funciones necesarias de comunicación en internet a aplicaciones del tipo servidor. Debido a lo anterior el modulo IP de la pila uinet puede responder a las peticiones *echo* con mensajes *echo reply* pero no a la inversa, ya que esto sería parte de la operación de un cliente.

El formato del mensaje *Destination Unreachable* se muestra en la figura 1.6, en la cuál se aprecia que el formato es distinto al del mensaje *Echo*. Este mensaje podría en algún momento ser recibido por la pila uinet si se presenta una falla en la comunicación. Es de utilidad procesarlo ya que informa a los protocolos de más alto nivel de la situación. Si la



Tipo (8 bits). Si el mensaje es una petición el valor de este campo es 0x08, si es respuesta es 0x00.

Código (8 bits). Este campo tiene el valor 0x00.

Checksum (16 bits). La suma de verificación del mensaje. Se calcula desde el campo tipo, hasta el final de los datos. El algoritmo es el mismo que para el checksum de la cabecera IP.

Identificador (16 bits). Si el código es cero, un valor que ayude a identificar las respuestas de las peticiones.

Número de secuencia (16 bits) Si el código es cero, un valor que ayude a identificar las respuestas de las peticiones.

Datos (longitud variable). Estos son datos aleatorios. La cantidad de datos es establecida desde la aplicación que envía este mensaje. Generalmente son 56 bytes.

Figura 1.5: Formato del mensaje ICMP *Echo*

pila recibe este mensaje cierra automáticamente las conexiones con los clientes actuales ya sea en TCP o en UDP.

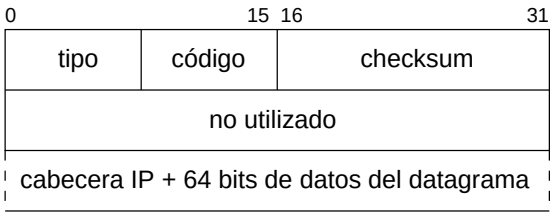
1.6. Extensiones multidifusión

Las extensiones de multidifusión se incluyen en la pila TCP/IP de este trabajo ya que permiten enviar un datagrama IP a múltiples destinos sin la necesidad de mantener un registro con todos los clientes a los que se envía la información. El uso de las extensiones IP disminuye la carga de procesamiento y los requerimientos de memoria en el microcontrolador que se utilizó para implementar la pila, cuando se requiere enviar la misma información a varios destinatarios; ya que en lugar de generar y enviar un paquete por cada destino, solo se genera y se envía un único paquete de multidifusión que llegará a todos los destinos, como se describe en el resto de esta sección.

La multidifusión IP es la transmisión de un datagrama a un grupo de anfitriones identificados por una sola dirección IP. Un datagrama de multidifusión se entrega a todos los miembros del grupo al que está dirigido con los mismos esfuerzos utilizados para entregar un datagrama IP normal; es decir, no se garantiza que el datagrama llegue intacto a todos los miembros.

La membresía de un grupo es dinámica: los anfitriones pueden unirse o dejar un grupo en cualquier momento. No existe restricción en cuanto a la ubicación o al número de miembros en un grupo. Un anfitrión puede ser miembro de uno o más grupos a la vez, y un anfitrión no necesita ser miembro de un grupo para enviar datos a éste.

Los grupos pueden ser permanentes o transitorios. Un grupo permanente tiene asignada una dirección IP bien conocida. La dirección del grupo puede ser permanente, pero no



Tipo (8 bits). Para este mensaje el valor es 0x03.

Código (8 bits). Los valores de este campo indican la razón del mensaje:

- 0x00 = red inalcanzable.
- 0x01 = huésped inalcanzable.
- 0x02 = protocolo inalcanzable.
- 0x03 = puerto inalcanzable
- 0x04 = fragmentación requerida pero datagrama marcado como No fragmentar.
- 0x05 = fallo en la ruta.

Checksum (16 bits). La suma de verificación del mensaje. Se calcula desde el campo tipo, hasta el final de los datos. El algoritmo es el mismo que para el checksum de la cabecera IP.

Cabecera IP + 64 bits de datos del datagrama (longitud variable). Estos datos son usados por el huésped para saber a cual de sus procesos corresponde este mensaje. Si la capa de nivel superior utiliza puertos, el número de puerto estará contenido en los 64 bits de datos del datagrama.

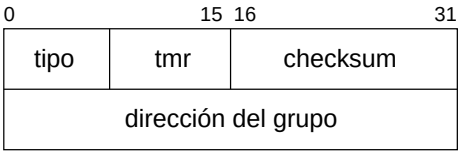
Figura 1.6: Formato del mensaje ICMP *Destination Unreachable*

así la membresía. Puede existir un grupo permanente que no tenga miembros en cierto momento. Las direcciones que no están asignadas permanentemente corresponden a los grupos transitorios. Los grupos transitorios solo existen cuando tienen algún miembro.

Las direcciones de los grupos de multidifusión son de clase D, véase la figura 1.3. Cuando un datagrama de multidifusión se transmite en la red local, se usa la cabecera IP especificando el grupo en el campo de la dirección IP de destino. Además el campo de tiempo de vida de la cabecera se establece en 1 si la multidifusión es solo para la red local, o mayor que 1 si la multidifusión debe propagarse a otras redes.

El campo de tiempo de vida (TTL) en la cabecera IP, juega un papel importante en la multidifusión, ya que permite limitar la multidifusión hasta ciertas redes: un datagrama con TTL=1 solo se distribuirá en la red local, mientras que si el TTL es menor que 1 será distribuido en las demás redes conectadas a la red local. Los gateways son los encargados de difundir el datagrama en las redes que tengan miembros del grupo al que está dirigido. El TTL de un datagrama es decrementado en 1 por cada gateway por donde atraviesa, y cuando llega a cero es descartado y no se propaga mas.

Las redes ethernet soportan directamente el envió de paquetes de multidifusión haciendo uso de las direcciones de multidifusión ethernet. Lo único que se requiere es un mapeo entre las direcciones IP de grupo y las direcciones de multidifusión ethernet. El mapeo se realiza colocando los 23 bits menos significativos de la dirección del grupo en los 23 bits menos significativos de la dirección de multidifusión ethernet: 01-00-5E-00-00-00 (hex). Por



Tipo (8 bits). Los tres tipos de mensajes que existen son:

- Petición de membresía = 0x11
- Reporte de membresía (versión 2) = 0x16
- Abandono de grupo = 0x17

TMR (8 bits). Tiempo máximo de respuesta. Especifica el tiempo máximo, en unidades de 1/10 de segundo, para responder a un mensaje de petición de membresía. En los demás mensajes vale cero y es ignorado por los receptores.

Checksum (16 bits). La suma de verificación del mensaje. Se calcula sobre el mensaje IGMP desde el campo de Tipo. El algoritmo es el mismo que para la cabecera IP.

Dirección del grupo (32 bits). En un mensaje de petición de membresía, este campo contiene la dirección del grupo en cuestión. En un reporte de membresía o abandono de grupo, se especifica el grupo que se reporta o se abandona.

Figura 1.7: Formato de los mensajes IGMP

ejemplo, el grupo de multidifusión 239.192.0.1 se expresa en hexadecimal como EF-C0-00-01, y sus 23 menos significativos son 40-00-01, colocando éstos en la dirección 01-00-5e-00-00-00 resulta la dirección 01-00-5e-40-00-01. Debido a que hay 28 bits significativos en una dirección de grupo IP, más de una dirección de grupo se puede mapear a la misma dirección ethernet.

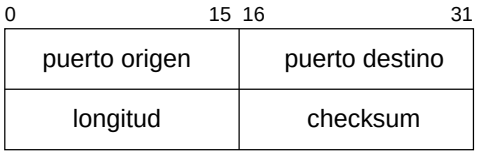
1.7. Protocolo IGMP

El protocolo *Internet Group Management Protocol* (IGMP) se usa para reportar la membresía de un grupo multidifusión a los gateways de multidifusión inmediatos. No todos los gateways en internet soportan la multidifusión, por ello la multidifusión generalmente está limitada a las redes locales. La versión 2 del protocolo IGMP es la que se implementó en este trabajo, y es la que se describe en los siguientes párrafos.

Al igual que ICMP, IGMP es una parte integral del protocolo IP. Los mensajes IGMP son encapsulados en datagramas IP utilizando el número de protocolo 0x02. Los mensajes IGMP se envían con el TTL=1 y la opción IP *router alert* en la cabecera del datagrama IP. Los mensajes IGMP concernientes a los anfitriones tienen el formato mostrado en la figura 1.7. El formato es el mismo para todos los mensajes, y solo cambian los valores del mismo dependiendo del mensaje del que se trate.

Para que un anfitrión se una a un grupo debe enviar un reporte de membresía especificando el grupo al que desea unirse dirigido a la dirección IP 224.0.0.1 Esta es una dirección especial que identifica a los todos los gateways multidifisión en la red. Con esto los gateways sabrán que deben pasar el tráfico del grupo al anfitrión que lo solicitó.

El gateway debe saber si el anfitrión aún está presente en la red recibiendo el tráfico multidifusión y para ello envía periódicamente el mensaje de petición de membresía en



Cabecera UDP

Puerto origen (16 bits). Es un campo opcional. Cuando tiene significado indica el puerto del proceso emisor y se asume que es el puerto al que la contestación por parte del receptor debe dirigirse. Si no se usa, se inserta el valor 0x00.

Puerto destino (16 bits). El puerto de destino tiene significado en el contexto de una dirección de internet particular.

Longitud (16 bits). La longitud en bytes del datagrama de usuario incluyendo la cabecera y los datos.

Checksum (16 bits). Es la suma de verificación de una pseudo cabecera, la cabecera UDP y los datos de la aplicación. El algoritmo de la suma de verificación es el mismo que para la cabecera IP. Se calcula utilizando el valor 0x0000 en el campo de checksum y comenzando en la pseudo cabecera hasta el último byte de datos. Si el cálculo de la suma de verificación resulta en 0x0000, se transmite como 0xFFFF. Si se transmite una suma de verificación igual a 0x0000 se entiende que el emisor no generó o que no usa la suma de verificación.

Figura 1.8: Formato de la cabecera UDP

donde pregunta si en la red hay miembros del grupo multidifusión. Los miembros al recibir este mensaje contestarán con un reporte de membresía.

Finalmente, cuando un anfitrión ya no desea recibir más tráfico del grupo, envía el mensaje de abandono de grupo a la dirección especial IP 224.0.0.4, con lo que si el anfitrión es el último miembro de grupo en una red el gateway detendrá el tráfico de multidifusión a la red. Pero si aún hay mas miembros en la red, el tráfico continuará y el anfitrión que abandonó el grupo seguirá recibiendo el tráfico, aunque ya no lo utilice.

El módulo IP desarrollado en este trabajo puede responder a la petición de membresía de grupo y generara un reporte de membresía al iniciar su funcionamiento.

1.8. Protocolo UDP

El protocolo *User Datagram Protocol* (UDP) proporciona un servicio de entrega de datagramas sin conexión y no confiable, utilizando el protocolo IP como el protocolo subyacente. Proporciona puertos para distinguir entre las aplicaciones, de un mismo anfitrión: cada mensaje contiene el número de puerto destino, así como el de origen. Un programa de aplicación que utiliza UDP debe responsabilizarse por los problemas en la comunicación: pérdida, retraso, duplicación y desorden en la entrega de datagramas.

El formato de la cabecera UDP se muestra en la figura 1.8. La cabecera es muy simple solo consta de la dirección de los puertos UDP de origen y destino, la longitud del mensaje UDP y una suma de verificación que es opcional.

Si la aplicación decide utilizar la suma de verificación se debe utilizar una pseudo cabecera, figura 1.9, para realizar el cálculo de la misma. La pseudo cabecera se usa solo para calcular la suma de verificación, no se incluye al enviar el mensaje. Ésta incluye las

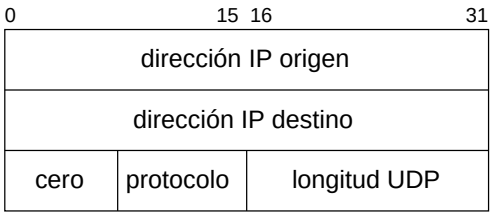


Figura 1.9: Pseudo cabecera UDP

direcciones IP del origen y del destino, el número de protocolo y la longitud del datagrama UDP sin incluir la longitud de la misma pseudo cabecera. El número asignado al protocolo UDP es 0x11, y es usado tanto en la pseudo cabecera UDP como en la cabecera IP.

En este trabajo el protocolo UDP se usa junto con las extensiones multidifusión para transmitir los datos de las lecturas de los sensores inalámbricos a través de internet. Véase el capítulo 5.

1.9. Protocolo TCP

El *Transmission Control Protocol* (TCP), es un protocolo orientado a conexión, confiable de extremo a extremo y diseñado para encajar en una jerarquía de protocolos que soporten múltiples aplicaciones de red. Provee una comunicación confiable entre procesos de anfitriones en redes distintas pero interconectadas entre sí. TCP asume muy poco sobre la confiabilidad de los protocolos de niveles inferiores. De hecho con obtener un datagrama de las capas inferiores es suficiente.

TCP es capaz de transferir un flujo continuo de bytes en cada dirección entre sus usuarios, empaquetando el flujo en segmentos de un tamaño apropiado para su transmisión. El tamaño de los segmentos está limitado por la unidad máxima de transmisión de la capa de acceso a la red, que en el caso de ethernet es de 1500 bytes. Los paquetes resultantes se pasan al protocolo IP para su entrega a través de internet.

Para asegurar que los paquetes no se pierdan en el camino, y evitar que lleguen en desorden, TCP le asigna a cada paquete un número de secuencia. El receptor puede con este número reacomodar los segmentos y reconstruir el flujo.

En un momento dado el receptor tendrá cero o más bytes reconstruidos desde el inicio del flujo. Por los bytes bien recibidos, el receptor envía un acuse de recibo especificando el número de secuencia del siguiente byte que espera recibir. Cada vez que se envía un segmento, el TCP inicia un temporizador y espera un acuse de recibo. Si se termina el tiempo antes de que se acusen de recibidos los datos del segmento, el TCP asume que el segmento se perdió o se corrompió y lo retransmite.

El TCP provee al receptor de una forma de gobernar la cantidad de datos que envía el emisor. Esto lo hace mediante el uso de la ventana deslizable. Supóngase que un anfitrión tiene lista una secuencia de paquetes para transmitir y que el receptor de los mismo le ha anunciado una ventana de 8 paquetes. Esto significa que el anfitrión enviará hasta 8 paquetes y esperara un acuse de recibo por todos. La ventana indica la cantidad aceptable de bytes que el emisor puede transmitir sin esperar un acuse de recibo. Este concepto hace que la transmisión de flujo sea eficiente. Si TCP utilizase un esquema simple de acuse de recibo positivo, (una ventana de 1 paquete) ocuparía un gran ancho de banda de red, debido a que retrasaría el envío de un nuevo paquete hasta que reciba un acuse del paquete anterior.

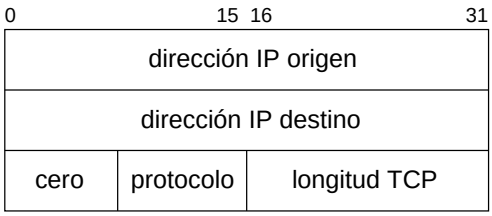


Figura 1.10: Pseudo cabecera TCP

Los acuses de recibo, ACK's, no se generan inmediatamente después de recibir un segmento debido al algoritmo de los ACK's retrasados. Este algoritmo retarda el envío del ACK durante un tiempo, alrededor de 300 ms[1]. Este retraso, aunque a primera vista parezca un tope máximo de la velocidad de transferencia, es en realidad una optimización que permite reducir el tráfico y mejorar el desempeño. Por ejemplo si durante la espera se reciben nuevos datos, con un solo ACK se pueden reconocer todos los datos recibidos. Una desventaja de los ACK's retrasados es que si el receptor tarda mucho en enviar un ACK, el emisor podría retransmitir el segmento y con ello generar más tráfico del necesario.

Los datos recibidos en el receptor se almacenan en un bufer de memoria y son pasados al protocolo de mayor nivel (la aplicación) cuando el emisor emplea la función *push*. Esta función obliga al módulo TCP a entregar los datos inmediatamente a la aplicación. La función garantiza que los datos se transferirán, pero no garantiza una frontera. Por ello las aplicaciones deben estar de acuerdo en un formato antes de iniciar una conexión.

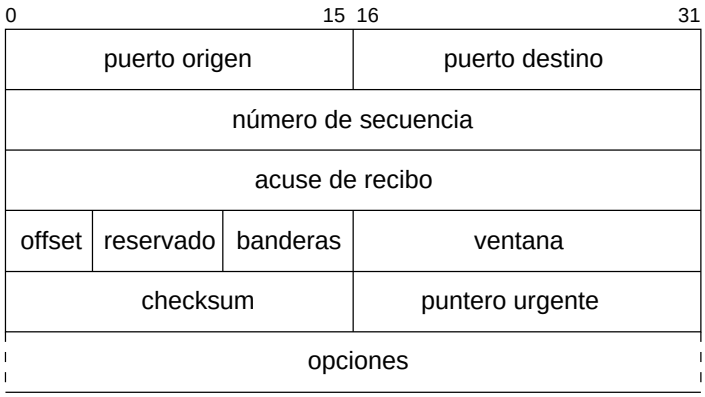
La abstracción fundamental de TCP es la conexión y no el puerto de protocolo. Las conexiones se identifican por medio de un par de puntos extremos. Un punto extremo es un par de números enteros (anfitrión, puerto), en donde anfitrión es la dirección IP de un anfitrión y puerto es el puerto TCP en dicho anfitrión. Por ejemplo el punto extremo (192.168.1.150,80) se refiere al puerto 80 en la máquina con dirección IP 192.168.1.150. Como el TCP identifica una conexión por el par de puntos extremos, varias conexiones en la misma máquina pueden compartir un número de puerto TCP.

Las conexiones proporcionadas por TCP se conocen como *full duplex*. Desde el punto de vista de un proceso de aplicación, una conexión full duplex consiste en dos flujos independientes que se mueven en direcciones opuestas, sin ninguna interacción aparente. La ventaja de una conexión full duplex es que el software subyacente de protocolo puede enviar en los datagramas información de control al origen, llevando datos en la dirección opuesta. Este procedimiento reduce el tráfico en la red.

Otra característica de TCP es que provee los medios para comunicarle al receptor de los datos que en algún punto más lejano en el flujo de datos que el receptor esté leyendo, hay datos urgentes. El TCP no define lo que el usuario debe hacer al ser notificado de que hay datos urgentes pendientes, la noción general es que el receptor debe tomar las acciones necesarias para procesar los datos urgentes lo más rápido posible.

La cabecera de TCP se muestra en la figura 1.11. A diferencia de UDP, el cálculo de la suma de verificación es obligatoria en TCP, y al semejanza de TCP también se utiliza una pseudo cabecera, figura 1.10, para realizar el cálculo del checksum. La pseudo cabecera contiene las direcciones IP del emisor y receptor, el número de protocolo, y la longitud de la cabecera TCP más los datos (sin contar la longitud de la pseudo cabecera).

[1]Este tiempo es solo una aproximación y su valor real se calcula de acuerdo al tráfico que detecta el receptor de los segmentos



Puerto origen (16 bits). El puerto de origen.

Puerto destino (16 bits). El puerto de destino.

Número de secuencia (32 bits). El número de secuencia del primer byte de datos en este segmento (excepto cuando la bandera SYN está presente). Si la bandera SYN está presente el número de secuencia corresponde al número de secuencia inicial (ISN), y el primer byte de datos tiene la secuencia ISN+1.

Acuse de recibo (32 bits). Si la bandera ACK está presente, este campo contiene el siguiente valor de número de secuencia que el emisor del segmento espera recibir. Una vez que la conexión está establecida, siempre se envía.

Offset de datos (4 bits) El número de palabras de 32 bits de la cabecera. Indica en donde comienzan los datos. La longitud de la cabecera TCP, aún con opciones incluidas, siempre es un múltiplo de 32 bits.

Reservado (6 bits). Reservado para uso futuro. Debe valer cero.

Banderas (6 bits). Cuando estas banderas están verificadas, su significado es el siguiente:

- URG: el campo del puntero urgente tiene significado.
- ACK: el campo del acuse de recibo tiene significado.
- PSH: Efectuar la función push.
- RST: Reajustar la conexión.
- SYN: Sincronizar los números de secuencia.
- FIN: El emisor no enviará más datos.

Ventana (16 bits). El número de bytes de datos, comenzado con el segmento indicado en el campo de acuse, que el emisor de este segmento puede aceptar.

Checksum (16 bits). La suma de verificación del segmento TCP.

Puntero urgente (16 bits). Este campo indica el valor actual del puntero urgente como un offset positivo desde el número de secuencia en este segmento. El puntero señala al número de secuencia del siguiente byte de los datos urgentes.

Opciones (longitud variable). Las opciones pueden ocupar espacio al final de la cabecera TCP y su longitud es un múltiplo de 8 bits. Todas las opciones se incluyen al calcular la suma de verificación.

Figura 1.11: Formato de la cabecera TCP

El algoritmo de la suma de verificación es el mismo que para la cabecera IP. Se calcula utilizando el valor 0x0000 en el campo de checksum y comenzando en la pseudo cabecera hasta el último byte de datos.

En la cabecera se aprecia un campo de opciones. El TCP define varios parámetros opcionales que se pueden incluir en la cabecera, si la aplicación requiere hacer uso de ellas. Una de las opciones más importante es el tamaño máximo de segmento MSS, la cuál ocupa 16 bits. Si esta opción está presente, indica el tamaño máximo de segmento que puede recibir el emisor del mismo. Este campo solo se envía al establecer la conexión. Si no está presente, entonces se asume que el MSS es 536 bytes. Finalmente cabe mencionar que el número de protocolo que se usa en la cabecera IP para encapsular al protocolo TCP es 0x06.

1.9.1. Etapas de una conexión TCP

En términos generales una conexión TCP atraviesa por tres etapas: establecimiento de la conexión, transferencia de datos y cierre de la conexión.

Establecimiento de la conexión

Para establecer una conexión, el TCP utiliza un saludo de 3 etapas (*3-way handshake*). El procedimiento es iniciado, normalmente, por un punto extremo y contestado por otro. El caso más sencillo se muestra en la figura 1.12.

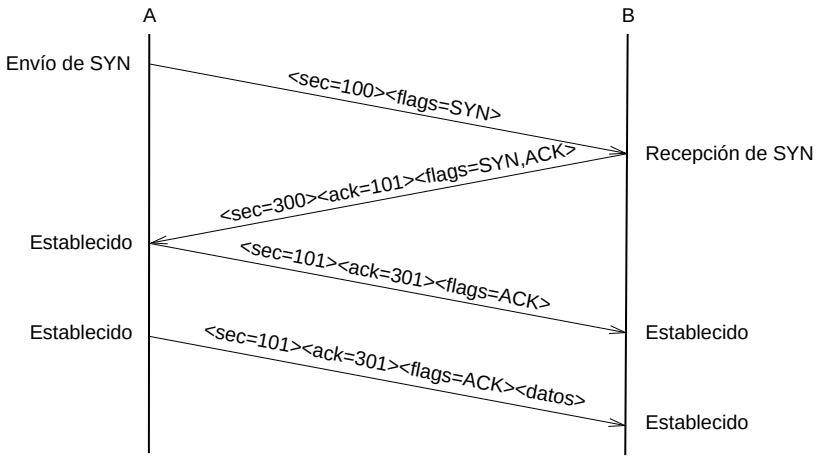


Figura 1.12: Saludo básico de 3 etapas de TCP

El primer segmento del saludo se identifica porque está verificada la bandera SYN. El segundo mensaje tiene verificados las banderas SYN y ACK, indicando el acuse del primer segmento como el hecho de que se continúa con el saludo. El último segmento del saludo es solo un acuse de recibo y se usa para indicar que ambos extremos están de acuerdo en establecer la conexión.

El saludo realiza dos funciones importantes: garantiza que ambos lados estén listos para transferir datos y permite a las partes acordar el ISN. Cada extremo selecciona un ISN aleatorio. En la figura 1.12, el extremo A escogió el ISN=100, mientras que B escogió ISN=300. El ISN no puede comenzar siempre con el mismo valor, en el documento RFC1122 [8] se sugiere escoger el ISN basado en la lectura del reloj de la máquina. Además de ponerse de acuerdo en el ISN, ambos extremos anuncian su MSS.

Transferencia de datos

Una vez que la conexión está establecida, la comunicación se da por el intercambio de segmentos. Dado que los segmentos se pueden perder por errores en la transmisión o congestión de la red, el TCP hace retransmisiones (después de un tiempo fuera) para asegurar la entrega de cada segmento. Cada entidad mantiene un registro con los números de secuencia que debe usar, y otro con los números de secuencia que espera recibir. El cierre de la conexión implica una función push y la llegada de un segmento con la bandera FIN verificada.

Cierre de la conexión

En esta fase se usa un saludo de 4 etapas (*4-way handshake*), con cada lado terminando independientemente. Cuando un extremo desea terminar la conexión, envía un paquete con la bandera FIN verificada que el otro extremo acusa de recibido enviando un segmento ACK.

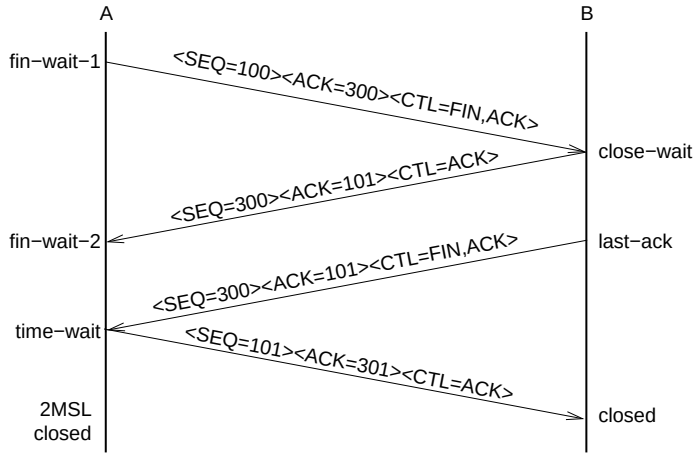


Figura 1.13: Cierre de conexión básico de 4 etapas de TCP

Por lo tanto, un cierre típico requiere un par de segmentos FIN y ACK desde cada extremo, como en la figura 1.13. Una conexión puede estar "medio abierta", este es el caso en que un extremo ha cerrado pero el otro no. El lado que ha terminado no puede enviar más datos por la conexión, mientras que el otro si puede.

Cabe mencionar que las banderas SYN y FIN ocupan un lugar en el espacio de número de secuencia. Cuando estas banderas se transmiten el número de secuencia del emisor se incrementa en 1.

1.9.2. Máquina de estados TCP

La operación de TCP se puede explicar mejor mediante el uso de una máquina de estados. La figura 1.14 muestra la máquina de estados TCP, donde los estados se muestran como un cuadrado redondeado y las transiciones como flechas entre los estados.

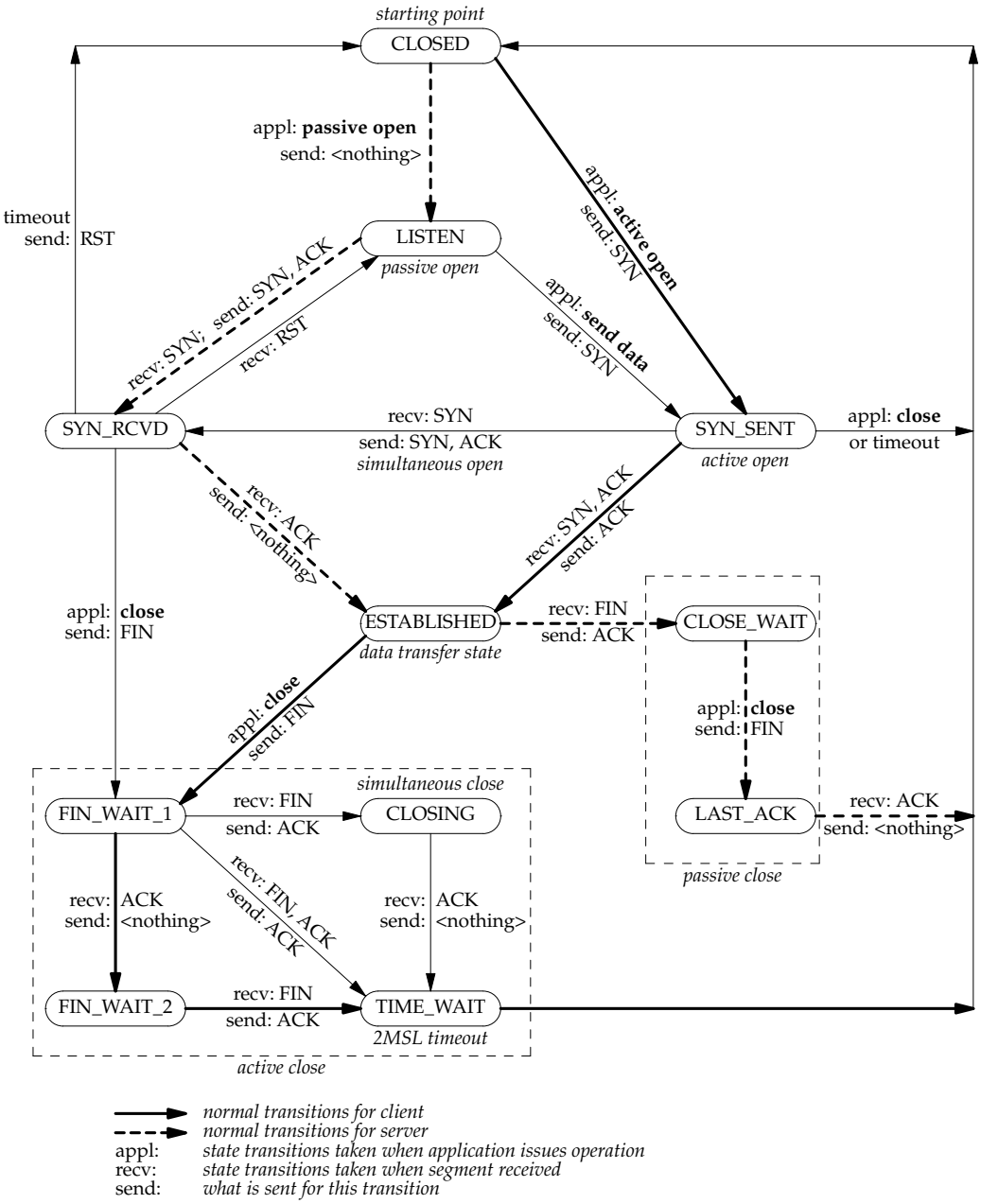


Figura 1.14: Diagrama de estados de TCP

Una conexión atraviesa por varios de los estados mostrados en la figura 1.14, éstos se describen a continuación:

LISTEN Representa la espera de una solicitud conexión desde otro punto extremo.

SYN-SENT Representa esperar por una conexión después de haber enviado la solicitud de conexión.

SYN-RECEIVED Representa esperar el acuse de la confirmación de conexión, después de haber recibido y enviado la petición de conexión.

ESTABLISHED Representa una conexión abierta, los datos se pueden entregar al usuario. El estado normal para realizar conexiones.

FIN-WAIT-1 Representa la espera de la solicitud de cierre de conexión del otro punto extremo, o el acuse de la petición de cierre enviada con anterioridad.

FIN-WAIT-2 Representa la espera por una solicitud de cierre del otro punto extremo.

CLOSE-WAIT Representa la espera por una solicitud de cierre de usuario local.

CLOSING Representa la espera el acuse de la solicitud de cierre del otro punto extremo.

LAST-ACK Representa la espera el acuse de la solicitud de cierre enviada al otro punto extremo.

TIME-WAIT Representa la espera de un tiempo suficiente para asegurar que el otro punto extremo haya recibido el acuse de la solicitud de cierre.

CLOSED Representa un estado en el que no hay conexión.

1.10. Ethernet

En la capa de acceso a la red se encuentra Ethernet. Ethernet (IEEE 802.3) es una especificación de cableado y señalización para redes de área local. Los nodos de la red se conectan entre sí utilizando una topología en estrella o en bus. Las velocidades de transmisión estándares para ethernet son 10 Mbps, 100 Mbps, y 1 Gbps.

En una red ethernet un nodo se comunica con otros enviando paquetes, estos paquetes llegan a todos los nodos de la red. En cada paquete se especifica el nodo destino utilizando una dirección y solo el nodo que tiene la dirección indicada procesará el paquete.

Cada dispositivo conectado a una red ethernet se identifica con una dirección de 6 bytes llamada dirección ethernet. La dirección ethernet también se conoce como dirección MAC (*Media Access Control*). La dirección está asignada por el fabricante del dispositivo y a su vez, el rango de direcciones asignado a los fabricantes es controlado por la IEEE.

Cuando una dirección ethernet contiene todos sus bits en 1 se conoce como dirección *broadcast*. Un paquete que tenga como destino la dirección broadcast será procesado por todos los nodos de la red.

En una red ethernet solo un nodo transmite a la vez. Si dos nodos transmiten al mismo tiempo se provoca una colisión en donde los paquetes se pierden. Ethernet utiliza el mecanismo de control *Carrier Sense Multiple Access with Collision Detection* (CSMA/CD) para detectar colisiones y recuperarse de las mismas.

Cuando se detecta una colisión se llevan a cabo los siguientes pasos:

1. El nodo escucha el canal antes de transmitir para determinar si otro nodo está transmitiendo.

2. Cuando el nodo determina que el canal está libre comienza a transmitir el paquete. Si dos nodos detectan el canal libre al mismo tiempo se provocará una colisión.

3. Mientras el nodo está transmitiendo el paquete también está escuchando si se produce una colisión.

4. Si se deteca una colisión los nodos involucrados dejan de transmitir.

5. Los nodos reintetarán la transmisión después de un tiempo de espera logarítmico. Este proceso se repite hasta que se logra transmitir el paquete, en un máximo de 16 intentos, al diecisieteavo intento se descarta el paquete.

Existen dos tipos de formatos de paquetes ethernet: Ethernet II y IEEE 802.3. Estos se muestran en las figuras 1.15 y 1.16 respectivamente. Para identificar el tipo de paquete utilizado en una red, se lee la palabra formada por los bytes 13 y 14. Si ésta es menor que 1500 el formato es 802.3 y si es mayor es Ethernet II.

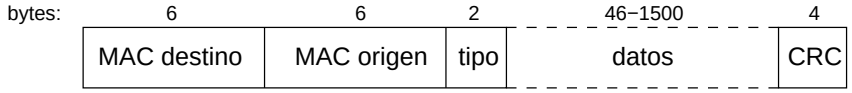


Figura 1.15: Formato de paquete Ethernet II

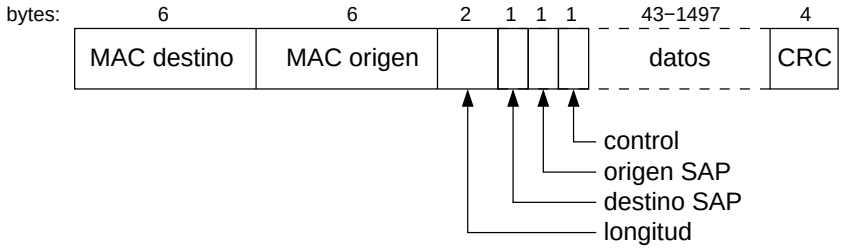


Figura 1.16: Formato de paquete IEEE 802.3

En este trabajo se utilizó el formato Ethernet II, por ser el más utilizado. Partiendo de este formato, el tamaño mínimo de un paquete es 64 bytes, en éste se pueden transferir 48 bytes de datos. Si se requiere transferir menos de 48 bytes, se agregan bytes de relleno hasta lograr 48. Por otra parte, el tamaño máximo de un paquete Ethernet II es de 1518 bytes, lo que permite hasta 1500 bytes de datos. Finalmente, la unidad máxima de transmisión (MTU) es el número máximo de datos que se pueden transmitir en un solo paquete es de 1500 bytes para una red ethernet.

Cabe mencionar que al generar un paquete ethernet, la cola del mismo (el CRC) es generada por el hardware controlador de ethernet y no por el software del sistema de cómputo que genera el paquete.

1.11. Protocolo ARP

El protocolo *Address Resolution Protocol* (ARP), sirve para mapear una dirección de protocolo de la capa de internet, a una dirección de hardware en la capa de acceso a la red. Si no se cuenta con la dirección de hardware de un anfitrión, no se le pueden enviar paquetes.

El protocolo funciona enviando una petición a la red especificando la dirección IP del anfitrión que se busca. Si el huésped está presente en la red, éste enviará una respuesta indicando su dirección ethernet.

El formato del paquete de petición y respuesta ARP es el que se muestra en la figura 1.17

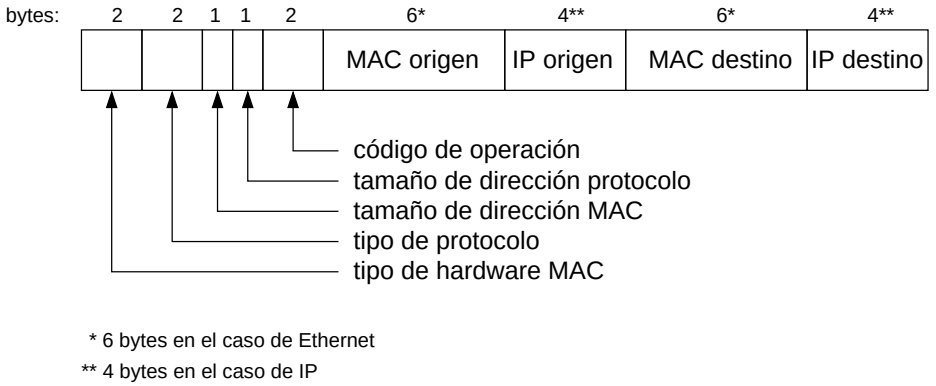


Figura 1.17: Formato del mensaje ARP

En el caso de una red ethernet el tipo de hardware tiene el valor 0x0001 y el número de bytes de la dirección es 0x06. Si el protocolo es IP, el tipo de protocolo es 0x0800 y el número de bytes de la dirección es 0x04. El código de operación es 0x01 para una petición y 0x02 para una respuesta.

Cuando un huésped envía una solicitud ARP para resolver su propia dirección IP se conoce como un *gratuitous ARP*. Se hace colocando la misma dirección IP en los campos de IP emisor y receptor y utilizando la dirección de broadcast de ethernet FF:FF:FF:FF:FF:FF. En una red propiamente configurada no debería de haber una respuesta a este tipo de solicitud, de haberla significa que hay otro huésped con la misma dirección del emisor. Esto es útil cuando el huesped inicia, ya que con ello se detectan las direcciones duplicadas en la red.

Los mensajes ARP se envían a la red tal cual el formato mostrado, es decir no se utiliza en conjunto con la cabecera de algún otro protocolo.

1.12. Protocolo HTTP

El *Hypertext Transfer Protocol* (HTTP) es el protocolo de aplicación que utilizan los navegadores y los servidores web para intercambiar información. En un principio, el propósito del protocolo era proveer una manera de publicar y recibir páginas HTML; sin embargo, las páginas HTML por si solas no eran llamativas ni ofrecían una experiencia muy amigable con el usuario. Por ello, el HTTP se extendió para soportar la transferencia de recursos. Los recursos pueden ser imágenes, sonidos, videos, animaciones, programas, etc., y se identifican mediante un *Uniform Resource Location* (URL) como por ejemplo: http://www.unam.mx/index.html o http://192.168.1.150/miguel/fi.jpg.

HTTP es un protocolo basado en mensajes de petición y respuesta. Típicamente un navegador web inicia una conexión TCP en el puerto 80 de un servidor web y le envía un mensaje de petición solicitando un URL. El servidor recibe la solicitud y contesta con el URL solicitado y cierra la conexión.

Los mensajes de petición y respuesta tienen un formato sencillo: están compuestos por una cabecera y un cuerpo opcional. La cabecera comienza con la línea de petición y una o varias etiquetas adicionales para describir los aspectos de la petición. La cabecera está delimitada del cuerpo del mensaje por los caracteres: `\r\n\r\n`. Por ejemplo, cuando un usuario solicita el URL: http://www.ejemplo.org/index.html, el navegador envía el mensaje de pe-

tición, listado 1.1, al servidor www.ejemplo.org. El servidor procesa la solicitud y envía una respuesta, listado 1.2.

```
1 GET /index.html HTTP/1.1\r\n
2 Host: www.ejemplo.org\r\n
3 \r\n
```

Listado 1.1: Ejemplo de petición HTTP

```
1  HTTP/1.1 200 OK\r\n
2  Date: Mon, 23 May 2005 22:38:34 GMT\r\n
3  Server: Apache/1.3.27 (Unix)  (Red-Hat/Linux)\r\n
4  Content-Length: 438\r\n
5  Connection: close\r\n
6  Content-Type: text/html; charset=UTF-8\r\n
7  \r\n
8  <html>
9  <head>Pagina de ejemplo </head>
10 <body>
11 Esta es una página de ejemplo
12 .
13 .
14 .
15 </body>
16 </html>
```

Listado 1.2: Ejemplo de respuesta HTTP

El mensaje de petición del listado 1.1 no incluye un cuerpo de mensaje, solo la cabecera. La línea 1 es la línea de petición y está formada por tres partes: método, recurso y versión. El método indica la acción que debe realizarse sobre el recurso especificado. El protocolo define los métodos siguientes:

GET Solicita un recurso en específico. GET genera un mensaje indicando en la cabecera las características del recurso solicitado, y en el cuerpo el contenido mismo del recurso.

POST Envía datos del usuario al recurso especificado. Se utiliza por ejemplo: para enviar los datos que el usuario introdujo en el formulario de alguna página, al servidor web.

HEAD Cuando un servidor recibe un mensaje con el método HEAD, produce una respuesta idéntica a la generada por GET, salvo que la respuesta no incluye el cuerpo del mensaje, solo la cabecera.

PUT Es un método que permite transferir un recurso al servidor.

DELETE Elimina un recurso de un servidor. Rara vez se implementa.

TRACE Este método genera un eco de la petición recibida.

OPTIONS Hace que el servidor indique los métodos que soporta.

Por otra parte, en el mensaje de respuesta (listado 1.2), la cabecera abarca desde la línea 1 hasta la línea 6 y el cuerpo del mensaje comienza en la línea 8. La línea 1 es la línea de estado y las líneas 2 a 6 son etiquetas que permiten al servidor agregar información sobre la respuesta que no puede saberse solo con la línea de estado, p. ej. el tamaño del recurso incluido en el mensaje. La línea de estado consiste de la versión del protocolo seguida de un código numérico y una frase aclaratoria. El primer dígito del código identifica el tipo de respuesta de que se trata:

1XX Informativo Este tipo de código indica una respuesta provisional.

2XX Exitoso Indica que la petición del cliente fue recibida, entendida y aceptada exitosamente.

3XX Redireccionamiento Indica que se requieren de más acciones por parte del cliente para satisfacer la petición. Por ejemplo que el recurso ahora se encuentra alojado en otro servidor y el cliente deberá dirigir su petición al nuevo servidor.

4XX Error del cliente Indica que el servidor detectó una petición incompleta o incomprensible.

5XX Error del servidor Indica que el servidor no pudo satisfacer una petición debido a incompatibilidades con el cliente, como versiones distintas de HTTP o métodos no implementados.

Los métodos GET y HEAD se consideran seguros, ya que solo permiten la descarga de información. En cambio, los métodos POST, PUT y DELETE son considerados como inseguros ya que pueden modificar los recursos de un servidor. Los métodos GET, HEAD, PUT y DELETE se definen como idempotentes: múltiples peticiones idénticas sucesivas tienen el mismo efecto en el servidor que una sola petición. Sin embargo, en la práctica el método GET también se usa para pasar los valores de los formularios al servidor, por lo que puede causar cambios en el servidor debido a la ejecución de programas (como los CGI) que pueden terminar en respuestas distintas a peticiones idénticas sucesivas. Los servidores HTTP deben implementar por lo menos los métodos GET y HEAD.

En la versión HTTP/1.1, las conexiones TCP entre el cliente y el servidor son persistentes: a través de la misma conexión el cliente puede realizar varias peticiones. En versiones anteriores, el cliente habría una conexión TCP, enviaba una petición, recibía la respuesta y cerraba la conexión.

El HTTP/1.1 también ofrece un mecanismo que facilita la generación de contenido dinámico: la codificación *chunked-encoding*. Esta codificación trata con el problema de la generación de contenido dinámico cuyo tamaño es desconocido. Al usar la codificación *chunked*, no es necesario conocer el tamaño de un recurso en el momento de generar la respuesta; ya que conforme se genera la respuesta, se transmite en tramos, donde cada tramo lleva un prefijo anunciando el tamaño del mismo.

Otra característica de HTTP/1.1 es la compresión de contenido: el servidor puede comprimir un recurso antes de enviarlo con el fin de que la transferencia sea mas corta. Esto es una optimización que disminuye el trafico en la red y hace un mejor uso del espacio de datos de cada paquete.

En el capítulo 4 se describe con mayor detalle el funcionamiento de las características antes mencionadas.

Capítulo 2

Hardware del servidor web

El servidor web es un sistema embebido y como tal el software y el hardware están hechos a la medida. Antes de discutir los detalles del software de este trabajo: la pila TCP/IP uinet y el programa del servidor HTTP Petit, es necesario conocer los detalles más importantes del hardware donde se ejecutan.

En éste capítulo se describe el diseño de hardware del que constituye al prototipo del servidor web. La tarjeta de desarrollo Easy Ethernet se utilizó como base para crear el prototipo debido a que contó con el hardware necesario para desarrollar la aplicación.

El prototipo final integra al servidor web junto con el hardware de la aplicación de ejemplo: el controlador central de la red de sensores inalámbricos que se describe en el capítulo 5.

El servidor web embebido que se describe en este trabajo, esta pensado para ser un bloque funcional que añada a nuevos diseños la capacidad de operar en internet. Este bloque permite que una aplicación se comunique en una red ethernet utilizando la pila de protocolos TCP/IP sin tener conocimiento alguno sobre los protocolos.

2.1. Requerimientos

El diseño del servidor web se realizó a partir de los requerimientos descritos en esta sección. Estos requerimientos se determinaron a partir del análisis de una PC utilizada como servidor web.

Un servidor web contiene recursos (imágenes, archivos, etc.), que deben ser enviados a los usuarios a petición de los mismos; por lo cual, el servidor debe contar con algún medio de almacenamiento no volátil para mantener esta información.

El servidor web en la PC es una implementación del protocolo HTTP. Este protocolo requiere imprimir en cada uno de sus mensajes la hora y fecha actual del sistema. Por ello es necesario que el servidor lleve un registro preciso de la fecha y hora aún cuando éste no cuente con alimentación.

Para utilizar el servidor web como un componente dentro de una aplicación, éste debe contar con una interfaz de comunicación sencilla y eficiente, como por ejemplo un bus serial. El *Inter-Integrated Circuit*[1] (I2C) o el *serial peripheral interface* (SPI)[2] son buses seriales

[1] El bus I2C utiliza dos líneas de comunicación una de datos (bidireccional) y una de reloj, a estas líneas se pueden conectar hasta 127 dispositivos diferentes, cada uno de ellos accesible con una dirección diferente

[2] El bus SPI provee direccionamiento mediante una línea de selección, es más veloz que el I2C; pero la lógica de selección puede complicarse si se usan varios dispositivos en el bus.

que permiten el direccionamiento de los dispositivos en el bus y proveen una buena velocidad de transferencia de datos sin la necesidad de muchas líneas de comunicación. El servidor utilizará el bus I2C, porque ello facilita el diseño del hardware, y tendrá disponibles las líneas del bus SPI en caso de que la aplicación así lo requiera.

Como el servidor se conectará a una red ethernet es necesario que cuente con un controlador dedicado para manejar la comunicación con la red ethernet, debido a que ésta es una tarea muy demandante. Este controlador actuará como interfaz entre el microcontrolador del servidor y la red ethernet.

El microcontrolador del servidor deberá contar con la memoria RAM suficiente como para ensamblar un paquete ethernet completo. Un paquete ethernet completo ocupa 1514 bytes, un estimado de 2.5Kbytes de RAM serán necesarios en el microcontrolador si se considera la memoria requerida para los procesos de software.

Finalmente, para efectos de la configuración del servidor en la aplicación final, éste deberá contar con un puerto serie RS232, mismo que permitirá la configuración y la carga de contenido desde una PC con programa emulador de terminal.

2.2. Modelo conceptual

Considerando los requerimientos mencionados en la sección anterior se diseñó un modelo conceptual de servidor web, figura 2.1, en donde se aprecian los componentes del mismo.

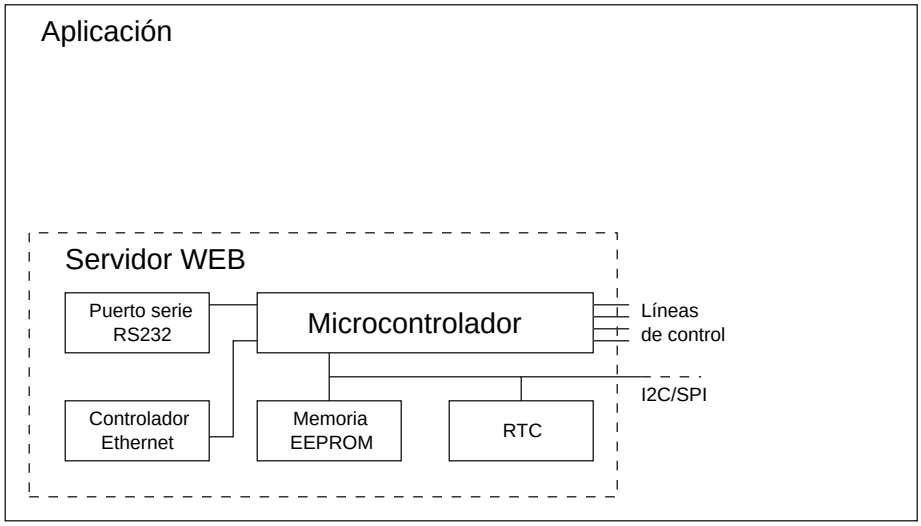


Figura 2.1: Modelo conceptual del servidor web

2.3. Tarjeta de desarrollo Easy Ethernet

Para llevar a cabo este proyecto se escogió a la familia de microcontroladores PIC de 8 bits; principalmente porque se contaba con la experiencia en su uso y con las herramientas de desarrollo necesarias: compilador de lenguaje C, ensamblador y el software y hardware necesario para programar los microcontroladores. La tarjeta de desarrollo Easy Ethernet se escogió para desarrollar el prototipo del servidor web precisamente por estar basada en un microcontrolador PIC.

La tarjeta Easy Ethernet, figura 2.2, está diseñada en torno al microcontrolador PIC16F877 en su versión DIP de 40 pines; cuenta con un puerto ethernet y un puerto serie RS-232.

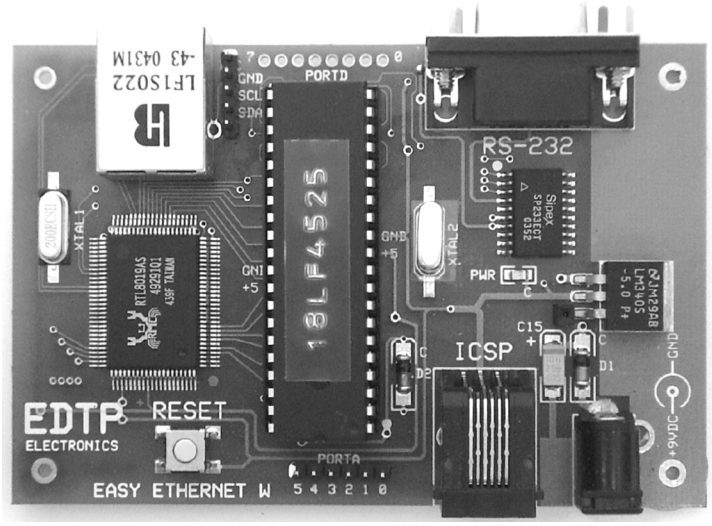


Figura 2.2: Tarjeta de desarrollo Easy Ethernet

El microcontrolador PIC16F877 de la tarjeta no cuenta con las características de memoria que se requieren para desarrollar el servidor, por ello en su lugar[3] se utilizó el PIC18F4525. Este microcontrolador tiene una arquitectura RISC de 8 bits. Cuenta con 48KB de memoria flash de programa, 3986 bytes de memoria RAM y 1024 bytes de memoria EEPROM. Cabe mencionar que cada instrucción tiene una longitud de 16 bits, lo que hace que en los 48KB se puedan almacenar 24576 instrucciones.

El PIC18F4525 tiene integrados los siguientes periféricos:

- Multiplicador 8x8
- 1 temporizador de 8 bits y 3 de 16 bits.
- 2 modulos de captura/comparacion/PWM
- 1 puerto serie *Master Synchronous Serial Port* (MSSP) que puede operar en modo SPI o I2C a 100KHz, 400KHz y 1MHz.
- 1 puerto serie *Enhanced Universal Synchronous Asynchronous Receiver Transmitter* (EUSART).
- 1 convertidor analógico digital de 10 bits.
- 2 comparadores analógicos.
- 1 comparador de voltaje para generar referencias de voltaje al convertidor a/d y a los comparadores.
- 1 detector de voltaje alto/bajo.

[3]El PIC18F4525 es compatible pin con pin con el PIC16F877, por lo que se pueden intercambiar

- 36 líneas de entrada/salida.

A la tarjeta se le cambió el cristal del microcontrolador para soportar al nuevo micro; se reemplazó el original de 20Mhz por uno de 10Mhz. Esto porque el 4525 tiene un generador de reloj interno con *Phase Lock Loop* (PLL) que cuadruplica la velocidad del cristal, logrando operar hasta los 40MHz.

En la tarjeta están disponibles las líneas del bus I2C del microcontrolador mediante un conector, de igual manera se encuentra disponible el puerto A. La tarjeta cuenta con un conector ethernet RJ45, que incorpora componentes magnéticos de aislamiento, y un circuito convertidor de nivel para RS232 con conector DB9 hembra.

En el apéndice D se encuentra el diagrama esquemático completo de la tarjeta.

2.3.1. Controlador de ethernet

La tarjeta Easy Ethernet cuenta con el chip controlador de ethernet RTL8019AS. Las características principales de este chip son:

- Cumple con las normas Ethernet II y IEEE802.3 10Base5, 10Base2, 10BaseT.
- Soporte full-duplex.
- Cuenta con 8 líneas de interrupción.
- Bus de datos configurable de 8 o 16 bits.
- Configuración por jumpers o *Plug and Play*.
- 16KB de memoria SRAM
- Auto-detección de interfaz UTP, AUI & BNC

El aspecto más importante de este controlador es la capacidad de configurar el tamaño del bus de datos. Esto permite su uso en sistemas embebidos en donde los microcontroladores o microprocesadores sean de 8 bits, ya que este controlador se utilizaba principalmente en las tarjetas de red de PC que utilizaban el bus ISA de 16 bits.

El controlador se encarga de transmitir y recibir los paquetes en la red ethernet, del manejo de las retransmisiones y de la discriminación de paquetes. En la hoja de especificación del controlador [23] se detallan las características del mismo, y en la nota de aplicación 475 de National Instruments [22] se presenta una guia para la configuración y el uso del controlador.

2.3.2. Memoria y reloj de tiempo real

El servidor web debe contar con un reloj de tiempo real y una memoria no volátil para almacenar el contenido web. El reloj y la memoria no son parte de la tarjeta Easy Ethernet, pero si son parte del prototipo desarrollado.

Tomada la decisión de utilizar el bus I2C para comunicar el servidor web con la aplicación, se optó por montar la memoria y el reloj también al bus. Con lo cual, surgió el requerimiento de que la memoria y el reloj soportaran la interfaz I2C a la misma velocidad, ya que la velocidad del bus está limitada por el dispositivo más lento. Es deseable que el

bus operara a la máxima velocidad posible, ya que esto reduciría los tiempos de acceso a los datos y mejoría el desempeño general del servidor.

El microcontrolador puede operar el bus I2C hasta 1MHz; sin embargo, no fue posible encontrar un reloj de tiempo real que operara a esa velocidad. El reloj que se utilizó en el servidor opera a 400KHz, y debido a ello es el componente que limita la velocidad del bus. El reloj que se usó fue el DS1338 de Maxim y tiene las siguientes características:

- Interfaz I2C a 400Khz.
- Cuenta horas, minutos, segundos, fecha, mes, año, día de la semana.
- 56 bytes de NVRAM para almacenamiento.
- Salida de señal cuadrada programable.

Dada la limitación del reloj en cuanto a la velocidad del bus I2C se escogió la memoria EEPROM 24LC512 de Microchip que también opera a 400KHz para ser parte del servidor web. Las características más importantes de esta memoria son:

- Interfaz de 2 líneas compatible con I2C.
- Entradas Schmitt Trigger para supresión de ruido.
- 1,000,000 de ciclos de borrado/escritura.
- Empaquetado DIP de 8 pines.

Los demás componentes necesarios para crear el servidor web, siguiendo el modelo conceptual, se encuentran en la tarjeta Easy Ethernet, por lo que no fue necesario diseñar ni agregar más componentes extra.

Se escogió una memoria EEPROM serial como medio de almacenamiento para facilitar el diseño. Se pudo haber escogido una memoria Flash comercial, como una tarjeta SD, XD, MMC, etc., pero esto implicaría más trabajo de software lo que retrasaría el tiempo de desarrollo de esta tesis.

2.4. Interfaz de hardware

Originalmente la tarjeta Easy Ethernet contaba con un conector de 3 pines para el bus I2C, donde se encontraban las lineas SCL, SDA y GND. La tarjeta se modificó para agregar la línea de alimentación +5V y la línea SDO (pin 24 del microcontrolador). Para que de esta manera el servidor web provea una interfaz de hardware a la aplicación que incluya: la alimentación de 5 Volts, un puerto I2C o bien[4] un puerto SPI y 6 líneas de control extra[5].

Cabe destacar que el prototipo del servidor web está formado por la tarjeta Easy Ethernet, el reloj de tiempo real y la memoria EEPROM. La tarjeta Easy Ethernet *por si sola no es el servidor web.*

Así la interfaz de hardware que el servidor ofrece para el diseño de una aplicación queda como en la figura 2.3.

[4]Las líneas del puerto I2C se pueden están multiplexadas internamente en el microcontrolador con las líneas del puerto SPI, mediante un registro de configuración se puede escoger entre uno u otro.

[5]La tarjeta Easy Ethernet cuenta con un conector para utilizar un puerto de 6 bits de entrada/salida digital o analógica, el cual se utilizó para proveer las líneas de control.

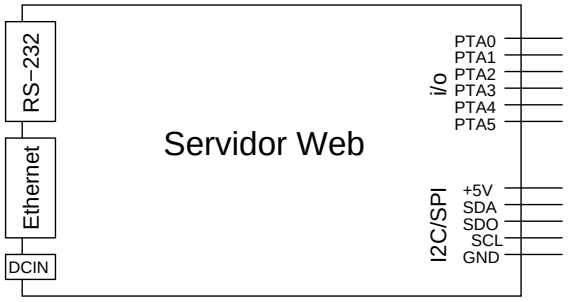


Figura 2.3: Interfaz de hardware del servidor web

En la aplicación de ejemplo del servidor web, descrita en el capítulo 5, se diseñó un circuito impreso que agrupa los componentes del servidor no presentes en la tarjeta Easy Ethernet, así como al hardware de la aplicación. En el apéndice E se encuentra el diagrama esquemático de dicho circuito así como las máscaras de pistas creadas para construir el circuito impreso.

En una siguiente versión del servidor web se puede crear una sola tarjeta que integre a la tarjeta Easy Ethernet junto con el reloj y la memoria EEPROM.

2.5. Diagrama esquemático

En la figura 2.4 se muestra el diagrama esquemático de los componentes del servidor web. Se aprecia en ésta los conectores que vienen de la tarjeta Easy Ethernet así como los conectores de la interfaz de hardware del servidor. Este diagrama esquemático es conceptual, el diagrama esquemático real del prototipo se encuentra en el apéndice E, así como las máscaras PCB y la lista de componentes.

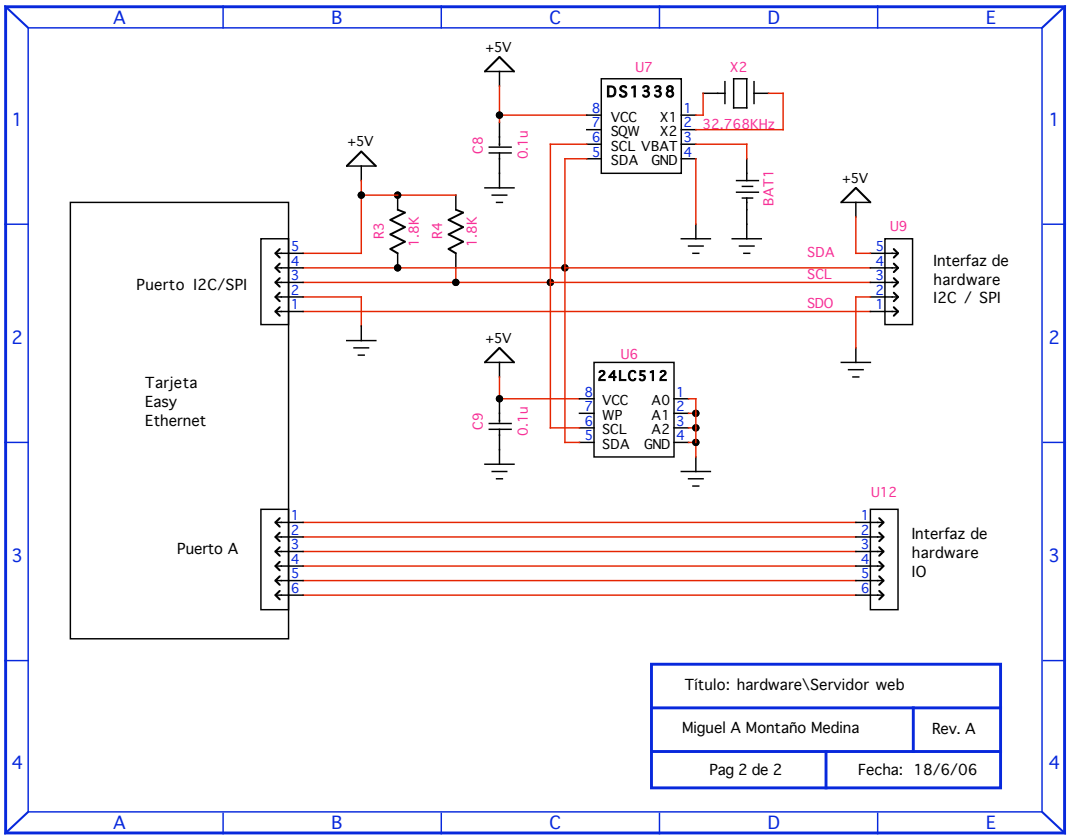


Figura 2.4: Diagrama esquemático conceptual del servidor web

Capítulo 3

Diseño e implementación de la pila TCP/IP uinet

El núcleo de la comunicación en internet reside en la pila de protocolos TCP/IP. La pila es una pieza integral del servidor web ya que provee a éste de las funciones necesarias para comunicarse a través de internet. Este capítulo describe el diseño y los detalles de implementación de la pila de protocolos desarrollada y denominada *uinet*.

La descripción del software de la pila se realiza mediante diagramas de flujo, por su simplicidad y efectividad. En algunas secciones de este capítulo se hace referencia a archivos de código; éstos se encuentran en el disco compacto que acompaña a este trabajo.

La pila uinet consta de los protocolos IP, TCP, UDP, ICMP e IGMP. El protocolo HTTP corresponde al servidor HTTP Petit y se describe en el capítulo 4.

3.1. Descripción general

El diseño de la pila TCP/IP se hizo considerando las restricciones de la plataforma de hardware destino: un microcontrolador. Esta plataforma impone severas restricciones en cuanto a memoria de programa y memoria RAM. Por ello, la implementación de los protocolos debe ser óptima: usando el mínimo de espacio de programa y la menor cantidad de memoria RAM posible. Partiendo de este requerimiento se implementó la pila TCP/IP por módulos, donde cada módulo corresponde a un nivel en el modelo de internet: aplicación, transporte, internet y acceso a la red.

La programación de la pila, así como de todo el software desarrollado para los microcontroladores de este trabajo, se realizó en lenguaje C empleando el compilador CCS PIC C.

La pila uinet la conforman los módulos: ARP, IP, TCP y UDP. Éstos no son completamente independientes uno del otro, ya que con el fin de reducir la copia de datos y las llamadas a funciones, un módulo puede acceder a los datos de los demás, o realizar tareas sencillas que idealmente serían parte de otro módulo.

Considérese una implementación típica de TCP/IP con un servidor web como aplicación. Cuando el servidor recibe una petición de parte de un usuario, genera una respuesta en consecuencia. Los datos que genera el servidor son copiados en un espacio de memoria reservado para el módulo TCP, donde éste divide los datos en segmentos y se encarga de su transmisión. Cuando un segmento TCP está listo para ser enviado, se copia en otra zona de memoria donde el módulo IP forma un datagrama y de allí pasa al módulo de acceso a

```
1  void aplicacion(void){
2    char bufer[1000];
3    int num_car;
4    socket s=crear_socket(5555);   /* crear un socket TCP en el puerto 5555 */
5    read(s,bufer,1000);            /* leer datos del socket y colocarlos en bufer */
6    num_car=strlen(bufer);         /* calcular el numero de caracteres recibidos */
7    write(s,num_car, sizeof(int)); /* escribir el numero de caracteres */
8    close(s);                      /* cerrar el socket */
9  }
10
11 int main(int argc, char **argv){
12   ...
13   aplicacion();
14   ...
15 }
```

Listado 3.1: Programa para calcular el número de caracteres recibidos

la red, para adecuarlo al tipo de red y transmitirlo. TCP no se deshace del segmento que envió a IP, sino que lo conserva en memoria hasta que recibe un acuse de recibo. Como TCP puede enviar varios segmentos y esperar un solo acuse por todos, debe mantenerlos en memoria con el fin de retransmitirlos en caso de perdida o daño de alguno. Este tipo de implementación utiliza un espacio de memoria para cada módulo, lo que ocasiona que se realicen muchas copias de datos, al pasar los datos de un espacio a otro. Por otra parte establece divisiones bien definidas entre los módulos, lo que hace que el mantenimiento del software sea más sencillo.

Los microcontroladores en general no disponen de una memoria RAM extensa, y el utilizado en este trabajo no es la excepción. Debido a ésto, la implementación descrita anteriormente no seria factible. Por ello se optó por un diseño de la pila en el que los módulos compartieran el mismo espacio de memoria minimizando el uso de RAM y evitando la copia de datos. El diseño contempla un bufer de memoria único para la transmisión, recepción y generación de paquetes. El bufer, denominado el bufer principal, tiene un espacio de memoria continuo y estático asignado en tiempo de compilación.

La pila uinet funciona de la siguiente manera: cuando se recibe un paquete de la red se coloca en el bufer principal donde es examinado y procesado por el módulo correspondiente, si éste determina que se debe enviar un paquete de respuesta, éste se genera en el bufer principal y se transmite a la red. De ésta manera se minimiza el uso de memoria RAM, pero se introduce un nuevo problema: no hay espacio para que el módulo TCP retenga los segmentos en memoria mientras espera el acuse de recibo. Para solucionar este problema, la aplicación debe tomar parte activa en la retransmisión de datos: debe poder recrear los últimos datos que envió, a solicitud de TCP. Así, la responsabilidad de la retransmisión es compartida entre TCP y la aplicación. Este esquema obliga a TCP a esperar un acuse por cada segmento que envía, lo que ocasiona que el mecanismo de la ventana deslizante no tenga efecto en la comunicación.

El diseño de la pila no requiere el uso de un sistema operativo en el microcontrolador. Los módulos de la pila se ejecutan secuencialmente y no requieren de procesos paralelos para su ejecución. La repercusión principal es que la API de uinet es diferente a la API de BSD. Esta última provee funciones para abrir y cerrar una conexión y para escribir y leer datos de la misma. La API de uinet no provee las mismas funciones pero pretende brindar la misma funcionalidad. La diferencia fundamental entre éstas es la forma en que se determina el flujo del programa. En BSD el flujo del programa de aplicación está definido

```
1  #define  PTO_TCP_H  0x15 /* puerto TCP , parte alta*/
2  #define  PTO_TCP_L  0xb3 /* parte baja. 0x15B3 = 5555d */
3
4  char packet[1500];               /* bufer principal */
5  int num_car;                     /* int es de 16 bits */
6
7  void ev_tcp_data(void){  /* evento que se dispara al recibir datos TCP */
8    num_car=strlen(&packet[TCP_data]);   /* calcular el numero de caracteres recibidos */
9    packet[TCP_data]=num_car>>8;         /* escribir el resultado de vuelta al bufer, parte alta */
10   packet[TCP_data+1]=num_car;          /* parte baja */
11   tcp_enviar(PSH,2);                   /* enviar el segmento */
12 }
13
14 void ev_tcp_rtx(void){ /* evento que se dispara al necesitarse  una retransmision */
15   packet[TCP_data]=num_car>>8;         /* escribir el ultimo resultado, parte alta */
16   packet[TCP_data+1]=num_car;          /* parte baja */
17   tcp_re_enviar(PSH,2);                /* reenviar el segmento */
18 }
```

Listado 3.2: Programa para calcular el número de caracteres recibidos

por el programador, mientras que en uinet el flujo es por eventos.

Por ejemplo, supongase un programa de aplicación que reciba una cadena de caracteres de una conexión TCP, y envíe, como respuesta, el número de caracteres que contiene la cadena. Esta aplicación se podría escribir en lenguaje C utilizando la API BSD como se muestra en el listado 3.1. En este caso la función `aplicacion` se llama explícitamente desde `main`, crea un nuevo socket, lee y escribe datos con las funciones `read` y `write` y cierra el socket con `close`. Al usar la llamada a `write` se efectúa una copia de datos al espacio reservado para TCP, que está operando en un proceso diferente a `aplicacion`. Como TCP tiene una copia de los datos en su espacio de memoria puede realizar, si es necesaria, una retransmisión sin intervención del usuario.

Por otra parte la misma aplicación se escribe con la API uinet como se muestra en el listado de programa 3.2. En uinet, el socket siempre existe y no hay necesidad de crearlo.

El módulo TCP define varios eventos, entre ellos `ev_tcp_data()` y `ev_tcp_rtx()`. El primero se dispara cuando el módulo TCP ha recibido un paquete exitosamente y está listo para ser procesado en el bufer principal, mientras que `ev_tcp_rtx()` se dispara cuando TCP ha determinado que es necesaria una retransmisión.

No existe en la API uinet una función para leer datos como `read`, ya que los datos están disponibles en el bufer principal al dispararse el evento `ev_tcp_data()`.

Para realizar la escritura de datos, en lugar de proveer una función `write`, en uinet se escriben directamente en el bufer principal. La función `tcp_enviar` crea el segmento TCP, llama al módulo IP y envía el paquete a la interfaz ethernet.

Para el manejo de las retransmisiones, el módulo TCP dispara el evento `ev_tcp_rtx()` cuando es necesario; la función asociada a este evento es `re_aplicacion`. Como `aplicacion` guardó el último dato que escribió en la variable `num_car`, `re_aplicacion` solo vuelve a colocar este valor en el bufer principal e invoca a `tcp_re_enviar`. La pila uinet provee la función `tcp_re_enviar` para usarse solamente en las funciones que hacen retransmisiones.

El diseño de uinet solo permite definir un puerto por donde esperar conexiones TCP, por este puerto escuchan todos los sockets. El número de puerto por defecto se define al asignar un valor hexadecimal a `PTO_TCP`, líneas 1 y 2 del listado de programa 3.2.

3.2. Organización de la pila

La pila está dividida en módulos, donde cada módulo provee soporte para un protocolo en específico. Cada uno de ellos reside en 2 archivos: uno de código y uno de cabecera. Además de éstos, la pila incluye otros archivos que complementan las funciones de los módulos. Los archivos que componen a la pila son los siguientes:

- Módulo TCP: `tcp.h` y `tcp.c`
- Módulo UDP: `udp.h` y `udp.c`
- Módulo IP: `ip.h` e `ip.c`
- Módulo ARP: `arp.h` y `arp.c`
- Interfaz ethernet: `rtl8019.c`
- Temporizadores: `timer.c`
- Configuración global: `globales.h`

Los módulos son dependientes entre sí y las dependencias entre ellos están relacionadas con el modelo de internet, como se aprecia en el diagrama de componentes, figura 3.1. Los archivos de cabecera de los módulos no se muestran en el diagrama para simplificarlo.

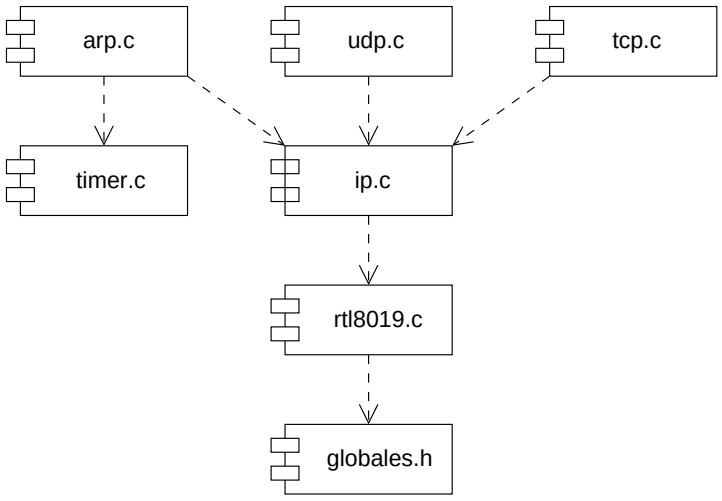


Figura 3.1: Organización de los archivos de la pila uinet

Todos los archivos de la pila pueden incluirse en una aplicación utilizando la cabecera `inet.h`.

3.3. Parámetros de configuración

En la figura 3.1, se aprecia que todos los archivos de la pila dependen al final del archivo `globales.h`. Éste contiene los valores por defecto de los parámetros de configuración de la pila uinet mostrados en el cuadro 3.1. Cuando el sistema se ejecuta por primera vez, utiliza los valores definidos en `globales.h`.

Parámetro	Valor	Descripción
DEFIP	192.168.1.150	Dirección IP
DEFMC	239.192.0.1	Dirección IP grupo multidifusión
TTLMC	1	Tiempo de vida para datagramas de multidifusión
MAC_ADDR	00-0D-93-B7-1E-A6	Dirección MAC ethernet
TCP_PORT	80	Puerto TCP
UDP_PORT	13068	Puerto UDP
MC_PORT	12300	Puerto UDP multidifusión
MAX_MSS	1460	Tamaño máximo de datos TCP en un segmento
MAX_CLIENTES	10	Número de clientes simultáneos

Cuadro 3.1: Parámetros de configuración de uinet

Si mientras el sistema está en ejecución el usuario modifica los parámetros, los nuevos valores son guardados en la memoria EEPROM del microcontrolador y utilizados en las siguientes ejecuciones del sistema. En la figura 3.2 aparece un esquema de la memoria EEPROM del micro que muestra las direcciones en donde se almacenan los valores de los parámetros.

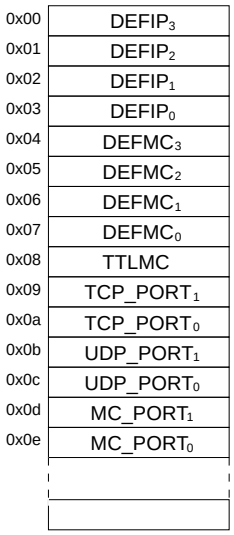


Figura 3.2: Mapa de la memoria EEPROM del microcontrolador

3.4. Bufer principal

El bufer principal es el espacio de memoria en donde se coloca un paquete al recibirlo y donde se genera un paquete para su transmisión. El tamaño del bufer debe ser lo suficientemente grande como para albergar un paquete completo.

El tamaño máximo de un paquete está limitado por el tamaño de la unidad máxima de transmisión (MTU). El MTU para una red ethernet es de 1500 bytes, lo que significa que la red puede transmitir en cada paquete hasta 1500 bytes de datos. Dado que la cabecera ethernet es de 14 bytes de longitud, el tamaño de un paquete completo es de 1514 bytes; éste sería el tamaño ideal del bufer.

En la pila el tamaño del bufer principal se asigna en relación al tamaño máximo de segmento TCP (`MAX_MSS`). En la mayoría de los casos la cabecera IP y la cabecera TCP ocupan en conjunto 40 bytes, lo que hace que el tamaño máximo de segmento TCP sea de 1460 bytes. En consecuencia el tamaño máximo del bufer (`MAX_PACKET`) se asigna a partir del valor de `MAX_MSS`:

$$\begin{aligned} MAX\_MSS &= 1460 \\ MTU &= MAX\_MSS + 40 \\ MAX\_PACKET &= MTU + 14 \end{aligned}$$

Si se utiliza el tamaño máximo del bufer de 1460 bytes, se transmitirán la mayor cantidad de datos en cada paquete. Si la aplicación no requiere enviar tanta información en un paquete, simplemente se disminuye el valor del `MAX_MSS` y el tamaño del bufer principal se ajustará al nuevo valor.

La definición del bufer principal se encuentra en el archivo `globales.h`.

3.5. Secuencia de inicio

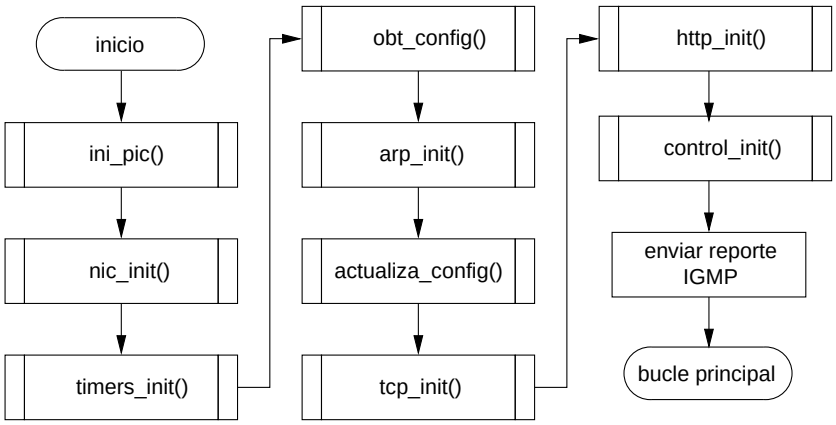


Figura 3.3: Secuencia de inicio del sistema

Para entender el funcionamiento de la pila es necesario conocer la secuencia de inicio del sistema completo. Esta se muestra en el diagrama de flujo de la figura 3.3. Cada una de las funciones que aparecen en la secuencia de inicio realiza lo siguiente:

- `ini_pic()`: Inicializa los periféricos internos del microcontrolador.
- `nic_init()`: Inicializa la interfaz ethernet.
- `timers_init()`: Inicia los contadores de tiempo.
- `obt_config()`: Recupera la configuración inicial del sistema desde la EEPROM del microcontrolador y provee al usuario la posibilidad de cambiarla
- `arp_init()`: Configura la dirección IP del sistema.
- `actualiza_config()`: Si el usuario cambió la configuración, esta función actualiza la configuración en la memoria EEPROM del microcontrolador.

- `tcp_init()`, `http_init()`: Inicializa el módulo TCP y el servidor web.
- `control_init()`: inicializa el controlador de la red inalámbrica.

Posteriormente, el sistema envía un reporte de membresía al grupo de multidifusión configurado en la pila, para finalmente entrar en un bucle infinito denominado el bucle principal.

3.6. Contadores de tiempo

El microcontrolador utilizado en éste trabajo cuenta con temporizadores que pueden generar una interrupción periódica. La pila uinet hace uso de esta característica para llevar un contador de tiempo (`c_tiempo`) que se incrementa libremente cada 100[ms]. Este contador se usa para programar la ejecución de las tareas periódicas de los módulos. Cada módulo mantiene un registro de tiempo (`HTTP_fuera, TCP_fuera, IGMP_fuera`) que se utiliza para programar la ejecución de las tareas periódicas en el bucle principal.

Por ejemplo, si es necesario que IGMP envíe un reporte de membresía en 3 segundos se hace lo siguiente:

$$IGMP\_fuera = c\_tiempo + 30$$

de esta forma, cuando se cumpla:

$$c\_tiempo = IGMP\_fuera$$

abrán transcurrido 3 segundos y se generará el reporte IGMP. Un proceso análogo se lleva a cabo con los demás registros de tiempo, con lo que se puede determinar el momento en que se genera el tiempo fuera para cada tarea.

3.7. Bucle principal

El núcleo del funcionamiento de la pila uinet y las aplicaciones basadas en ella es el bucle principal del sistema, figura 3.4. El bucle polea la interfaz ethernet, para saber si ha recibido un paquete de la red y ejecuta las tareas periódicas de los módulos que se encargan del manejo de las retransmisiones en TCP, el envío de reportes IGMP en IP, el manejo de sesión HTTP y el envío de datagramas multidifusión UDP.

Cuando la interfaz ethernet recibe un paquete, éste se descarga en el bufer principal de uinet y se invoca a la función `selector()` para determinar el tipo de paquete del que se trate. El diagrama de la figura 3.5 muestra como se realiza lo anterior.

La pila acepte paquetes dirigidos a su dirección IP, a la dirección del grupo o a la dirección IGMP *all systems* (224.0.0.1). Un paquete dirigido a cualquier otra dirección será ignorado.

Después de determinar si el paquete recibido será procesado según su dirección IP, `selector()` comprueba la suma de verificación de la cabecera IP y de ser incorrecta descarta el paquete, de lo contrario el paquete es procesado según el campo de protocolo de la cabecera IP.

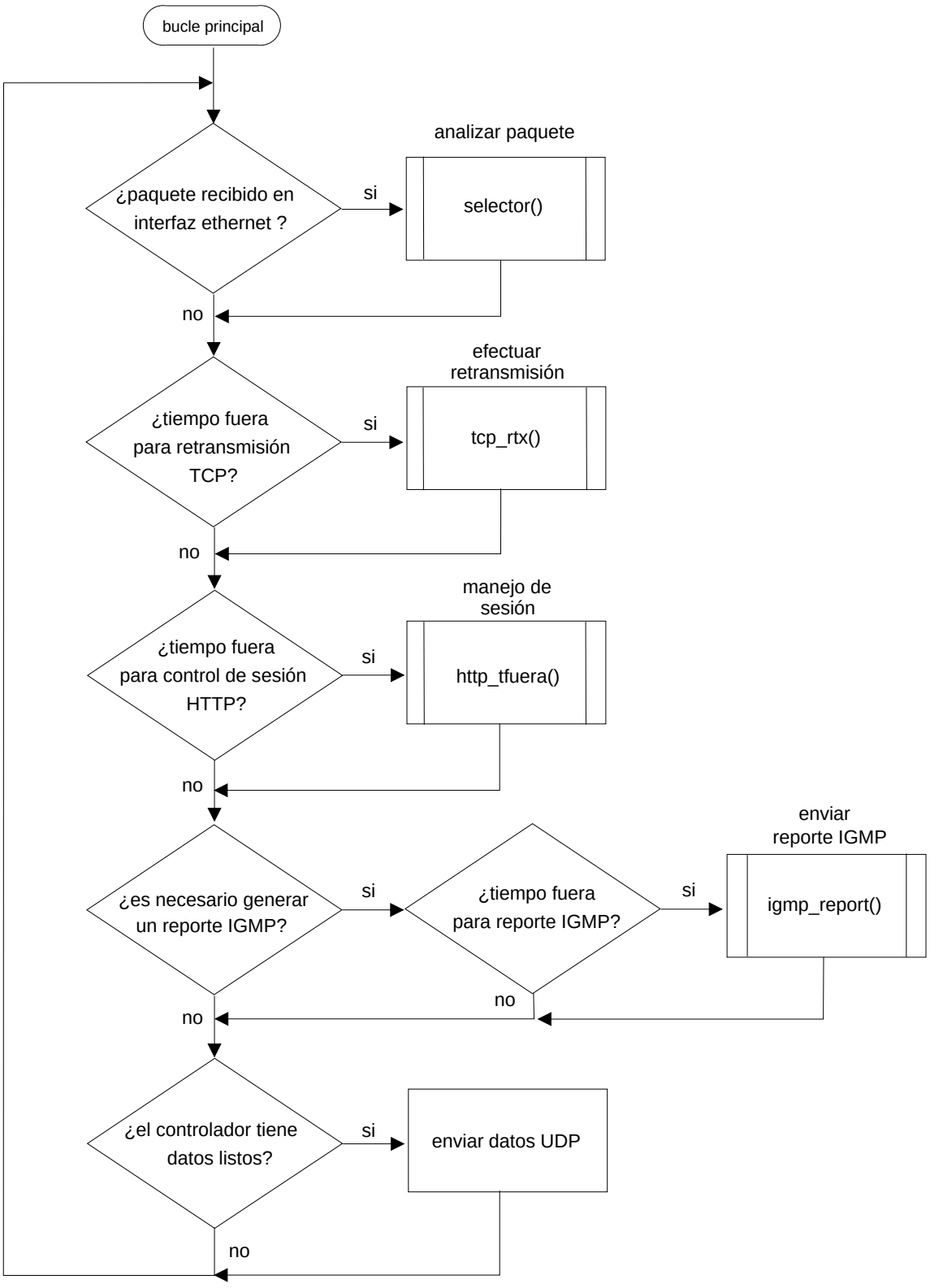


Figura 3.4: Bucle principal

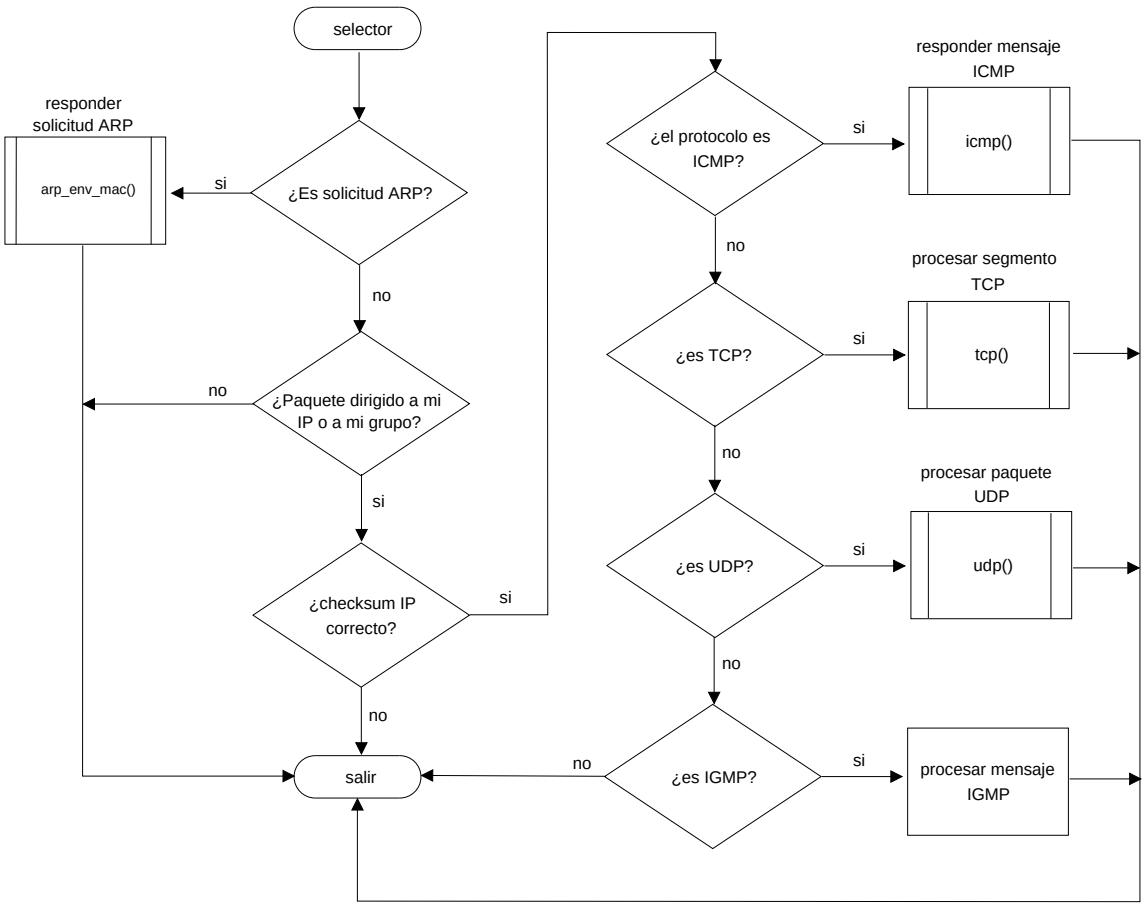


Figura 3.5: Análisis de un paquete

3.8. Modulo IP

El módulo IP provee la funcionalidad básica del protocolo IP y se encarga de 3 tareas principales:

1. Crear la cabecera IP de todos los paquetes de salida.
2. Generar los paquetes de respuesta a peticiones ICMP e IGMP.
3. Pasar los datagramas formados a la interfaz ethernet para su transmisión.

El protocolo IP define un soporte para la fragmentación de datagramas; sin embargo, el módulo IP de uinet no soporta esta característica, debido a que sería necesario un espacio de memoria adicional para colocar los fragmentos mientras se reensambla un datagrama.

Cada datagrama generado lleva en la cabecera IP un número de identificación de 16 bits. Este se obtiene del registro `ip_id_num`, que se incrementa en 1 cada vez que se envía un nuevo datagrama. El valor inicial de éste no está establecido y pretende ser arbitrario.

El módulo IP tiene el registro de la dirección IP propia `MYIP`, del grupo de multidifusión `MCIP` y el tiempo de vida que debe establecerse al enviar datagramas de multidifusión `ttl_mc`. Los valores por defecto de los registros son los establecidos en el archivo de configuración `globales.h`.

El módulo IP provee las siguientes funciones[1]:

- `ip()`: Esta función crea la cabecera IP del datagrama contenido en el bufer principal y lo transmite a la red.
- `icmp()`: Esta función genera el mensaje de respuesta eco a partir de los datos contenidos en el bufer principal y lo transmite por la red.
- `igmp_report()`: Genera un reporte de membresía del grupo multidifusión y lo transmite por la red.
- `igmp_leave()`: Genera un mensaje de abandono del grupo de multidifusión y lo transmite por la red.
- `chksum()`: Calcula la suma de verificación según el algoritmo estándar de IP.

La función `chksum` es utilizada por los demás módulos para calcular la suma de verificación de sus respectivas cabeceras. Dado que esta función es invocada varias veces durante el proceso de recepción y transmisión de un paquete, su implementación debe ser muy eficiente para no afectar el rendimiento de la aplicación. Por ello su programación se efectuó en lenguaje ensamblador.

El protocolo ICMP define varios mensajes de control que pueden generarse al detectarse un error en la transmisión de datagramas. Los mensajes que puede entender el módulo IP son: petición de eco y destino inalcanzable. La recepción del mensaje de petición de eco provoca que se genere una respuesta eco en consecuencia. La petición de eco viene acompañada por datos aleatorios, los cuáles deben repetirse en el mensaje de respuesta eco. Normalmente el número de datos que acompañan a una petición es de 64 bytes, pero puede variar. El módulo IP puede procesar mensajes de eco que contengan hasta `MAX_MSS+12` bytes de datos, si `MAX_MSS=1460`, entonces se pueden procesar mensajes de petición de eco hasta con 1472 bytes de datos.

La recepción de un mensaje de destino inalcanzable indica que la comunicación con el anfitrión especificado en el mensaje no puede continuar. Cuando ésta situación se presenta, los registros de la comunicación con el anfitrión son eliminados: en el TCB del módulo TCP se eliminan los datos del anfitrión, y en el módulo UDP se detiene la transmisión de datos al cliente (en el caso de conexión directa).

Cuando las funciones del módulo IP tienen listo un paquete en el bufer principal para ser enviado a la red, utilizan la API de la interfaz ethernet para enviarlo. La intefaz de software del controlador ethernet es parte del compilador CCS PIC C y se describe en el apéndice C.1.

[1]En el apéndice B.2 se describen todos los registros y las funciones del módulo IP.

3.9. Modulo UDP

El módulo UDP está diseñado para soportar la comunicación con un único cliente remoto. Dado que UDP no define la forma en la que se lleva a cabo una conexión, se diseñó un protocolo para que un cliente establezca una conexión directa con la aplicación UDP.

El protocolo para iniciar, mantener y finalizar una conexión directa está basado en mensajes UDP dirigidos al puerto UDP 13068, éstos se muestran en el cuadro 3.2.

Mensaje	byte (decimal)
solicitud de conexión	55
conexión aceptada	44
conexión rechazada	66
cliente presente	88
fin de conexión	33
datos	0

Cuadro 3.2: Mensajes UDP para conexión directa

Salvo el mensaje de datos, todos los demás son de 1 byte de longitud. El mensaje de datos tiene el formato mostrado en la figura 3.6, contiene un campo de 8 bits para especificar la longitud de datos del mensaje y un campo de 16 bits para indicar el número de secuencia que corresponde al mensaje. El número de secuencia se incrementa en 1 al enviarse un nuevo mensaje de datos.

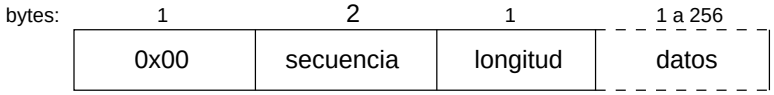


Figura 3.6: Formato de los datos en un paquete udp

Para que un cliente establezca una conexión UDP directa, debe enviar una solicitud de conexión y esperar la respuesta, ya sea de aceptación o rechazo. Cuando la conexión es aceptada, los datos del cliente son guardados en los registros del módulo. Para mantener la sesión, el cliente debe enviar el mensaje de presencia una vez cada 5 segundos. Si 5 segundos después de haber recibido el último paquete de presencia, no se recibe otro, la sesión se dará por terminada. El cliente puede terminar explícitamente la sesión enviando un paquete de fin de sesión. La conexión directa UDP no provee soporte para retransmisión, ni algún otro tipo de control de paquetes perdidos. El diagrama de la figura 3.7 ilustra una sesión de conexión directa UDP.

El puerto UDP por donde el módulo espera las conexiones es `UDP_PORT` y por defecto es el 13068.

Los registros y las funciones más importantes del módulo UDP[2] son los siguientes:

- `ip_udp`: La dirección IP del cliente.
- `pto_udp`: El puerto UDP del cliente.
- `eth_udp`: La dirección MAC del cliente.

[2]En el apéndice B.4 se describen todos los registros y las funciones del módulo UDP

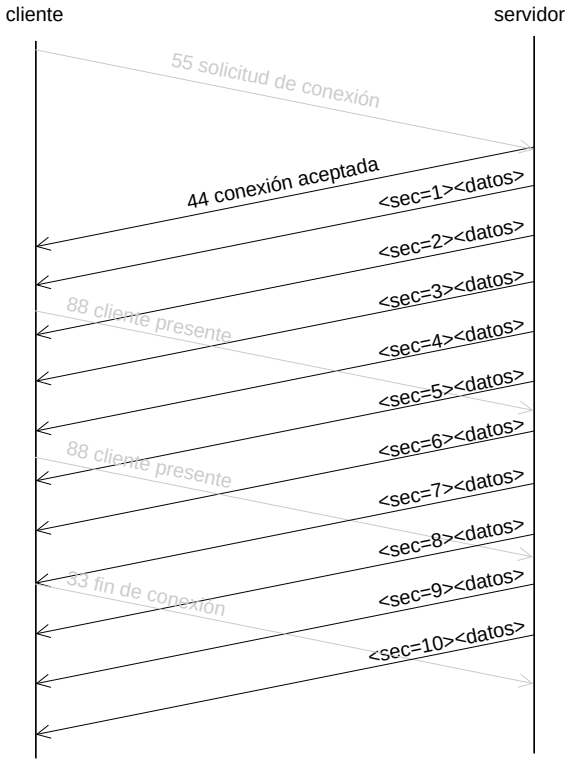


Figura 3.7: Ejemplo de conexión directa UDP

- `*usr_udp`: Es el apuntador de datos para lectura y escritura en el bufer principal.
- `udp()`: Se encarga del manejo de la conexión directa con el cliente.
- `udp_enviar()`: Esta función forma un paquete UDP a partir de los datos del bufer principal y lo transmite por la red.

La aplicación UDP desarrollada para la pila se ejecuta desde el bucle principal cuando se detecta que el controlador de la red de sensores tiene datos disponibles. La aplicación UDP se describe en detalle en el capítulo 5.

3.10. Modulo TCP

El módulo TCP de uinet permite la conexión simultánea de varios clientes a la vez, provee un soporte para retransmisión, define un único puerto TCP para aceptar conexiónes y maneja la comunicación mediante eventos. Está diseñado bajo la restricción de contar con solo un espacio de memoria para la recepción y generación de segmentos, así como para la retransmisión del último segmento enviado en caso de perdida.

Estas restricciones provocan que no se pueda utilizar completamente el mecanismo de la ventana deslizante, ya que no se cuenta con la memoria necesaria para ello. El protocolo TCP establece que en cada segmento transmitido se anuncia el tamaño de la ventana del emisor del segmento. El tamaño de ésta, rige la cantidad de datos que el anunciante puede recibir. Si el tamaño es mayor al de un segmento, el anunciante podría recibir más de un segmento sin generar un acuse de recibo a cambio. El módulo TCP siempre anuncia una ventana del tamaño de un segmento `MAX_MSS`, por lo cuál, los clientes solo enviarán un segmento TCP y esperarán el acuse del mismo antes de enviar uno nuevo.

Cada cliente conectado tiene asignado un registro *transfer control block* (TCB) que se muestra en el listado 3.3. Los registros TCB se agrupan en el arreglo `clientes`, el cúal es del tamaño `MAX_CLIENTES`.

El campo `id` es un identificador propio del TCB que se utiliza en el nivel de aplicación para asociar una estructura TCB con los datos de la aplicación. La dirección IP, el puerto TCP y el MSS del cliente se guardan en los campos `ip`, `pto` y `MSS` respectivamente. El MSS del cliente es el número máximo de datos que el cliente puede recibir en un segmento TCP. La aplicación debe considerar este dato para limitar el tamaño de los segmentos que envía.

El campo `mac` corresponde a la dirección MAC del cliente. Idealmente esta información sería parte del caché ARP; sin embargo, con el fin de simplificar la implementación se optó por incluirla en el TCB.

Los campos `sa`, `sv`, `tempo`, `rto`, y `nrtx` se utilizan para llevar la cuenta del número de retransmisiones del último segmento, calcular el tiempo de viaje *round trip time* (RTT) y el tiempo fuera para una retransmisión. El tiempo de viaje se calcula según el algoritmo de Jacobson [14] y el algoritmo de Karn [13]. El primero es un algoritmo actualizado respecto al utilizado en RFC793 [4] para calcular el RTT, y el segundo trata sobre la forma de efectuar el muestreo para calcular el RTT.

```
1  typedef struct {
2     uint  id;            // mi identificador
3     uint  ip[4];         // ip de mi cliente
4     uint  pto[2];        // puerto de mi cliente
5     uint  mac[6];        // direccion mac de mi cliente
6     uint16 MSS;          // el MSS de mi cliente
7     uint  estado;        // estado actual tcp
8     uint32 sndnxt, rcvnxt;  // numero de secuencia y acknowledge
9     uint16 lude;         // numero de datos que mande en el ultimo paquete
10    uint sa;             // estimadores
11    uint sv;             // estimadores
12    uint tempo;          // rtt
13    uint rto;            // el tiempo fuera para una retransmision
14    uint nrtx;           // el numero de retransmisiones que llevo del paquete
15 } tcb ;
16
17 tcb clientes[MAX_CLIENTES];
```

Listado 3.3: *Transfer control block* del módulo TCP

Además del registro TCB del cliente, el módulo TCP define los siguientes registros:

- `*cnte`: Es un apuntador al registro TCB de `clientes` que representa al cliente con el que se esta comunicando la aplicación.
- `ini_seqnum`: El número de secuencia inicial TCP que se utiliza al iniciar la conexión con algún cliente.
- `*tcp_buf`: Es un apuntador que las aplicaciones pueden utilizar para leer y escribir datos en el bufer principal.
- `TCP_PORT`: El número de puerto TCP por el que están escuchando todos los sockets de la pila.

La característica más importante de la pila uinet recide en el hecho de que la aplicación funciona mediante eventos, que son disparados desde el módulo TCP conforme a los estados de la conexión. Estos son:

- `ev_tcp_ini()`: Ocurre al finalizar el saludo de tres etapas TCP con el cliente. Tiene como finalidad permitir a la aplicación incializar sus variables de entorno.
- `ev_tcp_data()`: Ocurre al recibirse un nuevo segmento del cliente. Le indica a la aplicación que ha llegado una petición por parte del cliente.
- `ev_tcp_ack()`: Ocurre al recibirse el acuse de recibo del último segmento enviado. Permite continuar enviando la respuesta a una petición que ocupa más de 1 segmento.
- `ev_tcp_rtx()`: Ocurre cuando es necesario retransmitir el último segmento enviado. La aplicación debe poder recrear el último segmento que emitió.
- `ev_tcp_fin()`: Ocurre al cierre de la conexión, ya sea por parte del cliente o por algún error en la comunicación. Anuncia un cierre inesperado en la conexión y permite a la aplicación realizar las tareas propias del cierre.

La aplicación debe implementar el cuerpo de los eventos. Estos no llevan ningún argumento de función para pasar datos de la pila a la aplicación, ya que ésta puede acceder a los registros del módulo TCP directamente.

Los segmentos recibidos son examinados para discriminarlos según sus números de secuencia: solo los segmentos con un número de secuencia que pertenezcan a la ventana serán aceptados, según las reglas definidas por el protocolo TCP. Cuando un segmento es aceptado, las banderas del mismo determinarán el siguiente estado de la conexión. Los estados de una conexión, las condiciones de transición y el momento en que se disparan los eventos TCP se aprecian en la máquina de estados del módulo TCP, figura 3.8. El campo `estado` del registro TCB indica el estado de la conexión. El estado inicial de todos los sockets es *listen*.

La máquina de estados se encuentra implementada dentro de la función `tcp()`, la cuál es llamada desde `selector` cuando se determina que un paquete contiene un segmento TCP.

El módulo TCP aporta tres funciones[3] principales que pueden utilizarse en el cuerpo de la definición de los eventos, éstas son:

[3]El módulo TCP cuenta además con otras funciones, en el apéndice B.3 se describen todos los registros y las funciones del módulo TCP

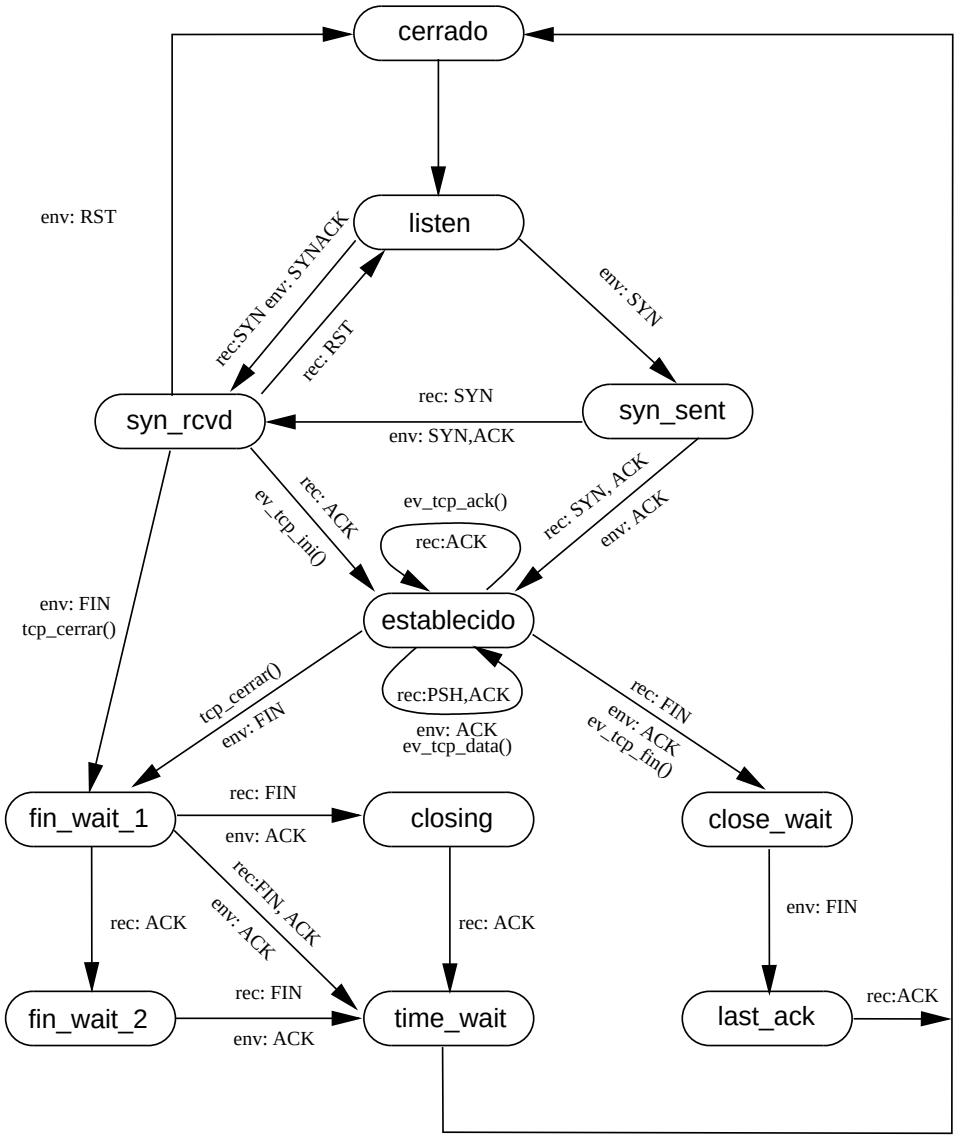


Figura 3.8: Máquina de estados del módulo TCP.

- `tcp_enviar()`: Esta función forma un paquete al crear las cabeceras TCP, IP y ethernet a partir de los datos contenidos en el bufer principal, y envía el paquete a la interfaz ethernet. Esta función altera el registro TCB al actualizar los números de secuencia y de acuse, así como el registro de datos enviados.
- `tcp_re_enviar()`: Esta función es igual a la anterior pero su uso es exclusivo para el cuerpo del evento `ev_tcp_rtx()`, ya que no altera el registro TCB.
- `tcp_cerrar()`: Esta función se utiliza en el cuerpo de los eventos cuando es necesario terminar la conexión con el cliente.

En la figura 3.8, se aprecian las transiciones que disparan los eventos del módulo TCP con excepción de `ev_tcp_rtx()`. Este evento se dispara desde la tarea periódica `tcp_rtx()`. Esta tarea tiene la finalidad de determinar cuándo es necesario efectuar una retransmisión.

Para ello, la tarea se ejecuta cada 100 [ms] y efectúa lo siguiente: busca en todos los elementos del arreglo `clientes` a aquellos cuyo estado de la comunicación sea diferente a cerrado y verifica si en la conexión hay datos sin acuse de recibo. Posteriormente calcula si se ha excedido el tiempo límite para una retransmisión y de ser así se incrementa el contador de retransmisiones del segmento. Si se han realizado 7 retransmisiones[4] del segmento y aún no se recibe ningún acuse, la conexión con el cliente se da por terminada, de lo contrario se efectúa la retransmisión dependiendo del estado de la comunicación. El tiempo límite para la retransmisión se incrementa exponencialmente (*exponential back-off*) y se dispara el evento `ev_tcp_rtx()` solo si el estado de la comunicación es *establecido*. Todo el proceso se resume en el diagrama de flujo de la figura 3.9.

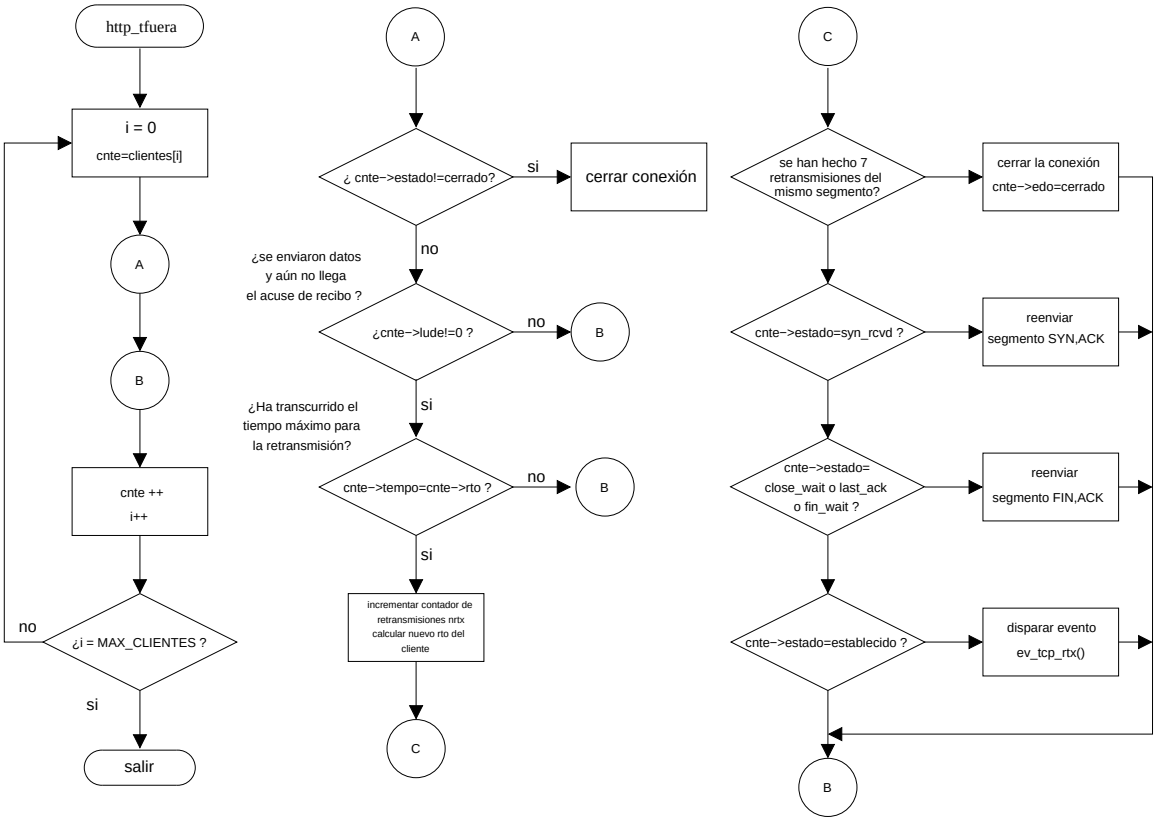


Figura 3.9: Diagrama de flujo de `tcp_rtx()`

El protocolo TCP define un mecanismo para el transporte de datos urgentes, pero dado que la aplicación desarrollada para la pila no requiere de esta característica, ésta no se implementó en el módulo TCP.

3.11. Modulo ARP

En TCP/IP los anfitriones se comunican mediante el uso de las direcciones IP, pero cuando lo hacen en una red ethernet deben dirigirse unos a otros con la dirección MAC. El protocolo ARP se utiliza para mapear las direcciones IP con las correspondientes direcciones MAC.

[4]El valor de 7 retransmisiones implica una espera aproximada de 5 minutos en espera del acuse de recibo.

El módulo ARP provee funciones para responder a las solicitudes ARP, generar peticiones ARP y configurar la dirección IP del sistema. El módulo define los registros `tmac` y `tip` que se utilizan para pasar argumentos a las funciones del módulo. Las funciones[5] son:

- `arp_env_mac()`: genera y envía un mensaje de respuesta ARP anunciando la dirección IP: `MYIP`, y la dirección MAC: `MAC_ADDR`.
- `arp_sol_mac()`: genera y envía un mensaje de solicitud ARP solicitando la dirección MAC del anfitrión cuya IP sea `tip`.
- `arp_buscar_ip()`: verifica que la dirección IP `tip`, no esté ocupada por otro dispositivo en la red. Si está ocupada coloca en `tmac` la dirección MAC del dispositivo que la usa.
- `arp_init()`: verifica que la dirección IP `MYIP`, no esté ocupada por otro dispositivo en la red, de estarlo informa al usuario de la situación.

Cuando el sistema inicia su ejecución emite un paquete destinado a todos los dispositivos de la red anunciando su dirección IP; se utiliza para prevenir que existan direcciones duplicadas en la misma red. El paquete se conoce como *gratuitous* ARP. Al iniciar, el sistema enviará el paquete *gratuitous* hasta 3 veces con una espera de 500 ms entre ellos. Si se recibe una respuesta al paquete entonces habrá una dirección duplicada y el sistema interrumpirá la inicialización. Cuando esto pasa, el usuario es informado de la situación con el parpadeo de los leds de estado y un mensaje en la terminal. Desde ésta, el usuario podrá entonces configurar una nueva dirección IP, como se aprecia en la figura 3.10.

La función responsable de realizar lo anterior es `arp_init()` y utiliza a las demás funciones del módulo para realizar su tarea.

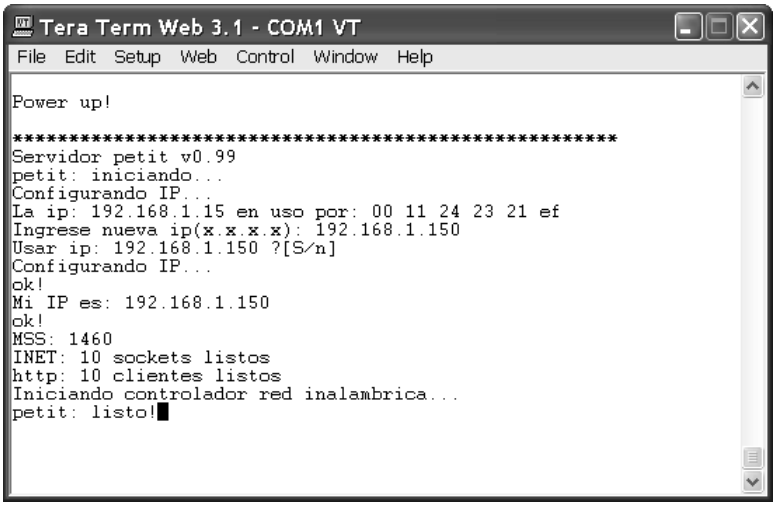


Figura 3.10: Captura de configuración de la dirección IP

[5]En el apéndice B.1 se describen todos los registros y las funciones del módulo ARP

3.12. Trabajos relacionados

Existen algunas implementaciones de la pila TCP/IP diseñadas para usarse en sistemas embebidos. Algunas de ellas elaboradas por empresas de software, otras como proyectos independientes sin fines de lucro y otras mas desarrolladas solo por pasatiempo. La pila uinet se desarrolló tomando como punto referencia tres implementaciones de la pila TCP/IP: μIP, TCP/IP Lean y Open TCP; estas se describen en los párrafos siguientes.

TCP/IP Lean

La implementación TCP/IP Lean es una de las más conocidas y se explica en el libro "TCP/IP Lean" de J. Bentham [20]. Ésta fue diseñada para ejecutarse en el microcontrolador PIC16F877 con la meta de crear una pila que ocupase el mínimo de recursos posibles. El microcontrolador es de 8 bits y cuenta con solo 368 bytes de RAM, debido a ésto, la pila debe valerse de estrategias de software complicadas para poder enviar un paquete ethernet que contenga más de 200 bytes: utiliza paquetes prefabricados almacenados en la memoria de programa a los cuales les cambia las direcciones IP, los números de puerto TCP, los checksums y otros datos antes de enviarlos al cliente. Esta estrategia provoca que la interfaz de software de la pila sea complicada y difícil de utilizar en la creación de nuevas aplicaciones, además de que el uso de las respuestas prefabricadas hace imposible la generación de paquetes con información dinámica.

La pila TCP/IP Lean incluye un servidor web, un cliente de correo electrónico y un servidor telnet que muestra como utilizar la pila para desarrollar nuevas aplicaciones.

OpenTCP

La pila OpenTCP es un proyecto desarrollado inicialmente como un proyecto de software libre, pero ahora es propiedad de la empresa Viola Systems. Esta pila es la más parecida a la implementación del sistema operativo BSD ya que provee una interfaz de programación muy similar a ésta. Este trabajo es el más completo y robusto. La pila Open TCP no está basada en torno a un microcontrolador en especial, sino en el uso del leguaje ANSI C estándar con números enteros de 16 bits. En teoría esta pila podría funcionar en cualquier microcontrolador para el que exista un compilador de C estándar. Sin embargo muy pocos microcontroladores cuentan con un compilador de C que soporte completamente el estandar ANSI C. Por ejemplo, el compilador utilizado en este trabajo PIC CCS C solo soporta un subconjunto del estándar ANSI C y no puede compilar la pila Open TCP dado que no soporta el uso de apuntadores a funciones, además que los tipos de datos enteros que maneja son de 8 bits.

Open TCP requiere aproximadamente 32 Kbytes de RAM para su ejecución, lo que hace que se requiera de un microcontrolador o un microprocesador que pueda direccionar esta cantidad de memoria, además de que dependiendo del tamaño de la instrucción puede requerir de 30 a 35 Kbytes de memoria de programa.

Dado que esta pila ahora es un producto comercial, su código fuente ya no está disponible al publico en general, lo que es una desventaja para utilizarla en nuevos diseños debido al pago de *royalties*.

Características	μIP	TCP/IP Lean	uinet	OpenTCP
Arquitectura de hardware	8 bits	8 bits	8 bits	16 bits
Número de clientes simultáneos	Configurable	1	Configurable	Configurable
Tamaño máximo de paquete ethernet	1514	1514	1514	1514
Configuración IP	Estática	Estática	Configurable	Configurable
Protocolo IP	Si	Si	Si	Si
Extensiones IP Multicast	No	No	Si	No
Protocolo ICMP	Parcial	Parcial	Parcial	Si
Protocolo IGMP	No	No	Si	No
Protocolo TCP	Si	Si	Si	Si
Protocolo UDP	Si	Si	Si	Si
Protocolo ARP	Si	Si	Si	Si

Cuadro 3.3: Comparativa entre las diferentes implementaciones de la pila TCP/IP

μIP

La pila μIP es otro trabajo importante, realizado por Adam Dunkels [21], la cual es una implementación completa de los protocolos IP, TCP y UDP desarrollada inicialmente para la familia de microcontroladores AVR. Este trabajo cuenta con varias aplicaciones desarrolladas con la pila: un servidor web, un cliente de correo electrónico SMTP, y un servidor de archivos FTP entre otras aplicaciones. Ésta pila μIP también está desarrollada con la idea de explotar el uso del lenguaje ANSI C estándar; su diseño no es tan robusto como el de la pila Open TCP, pero es lo suficiente como para establecer una comunicación confiable ya sea mediante TCP o UDP. Está escrita considerando el uso de números enteros de 8 bits. Puede mantener la comunicación con más de un cliente al mismo tiempo.

El código fuente de μIP está optimizado para reducir las llamadas a funciones con lo que se logra reducir el tiempo de respuesta y el tamaño final del código ejecutable. Sin embargo, la interfaz de programación es confusa y difícil de entender, ya que solo hay una función que se ejecuta para manejar la comunicación TCP cada vez que arriba un paquete, cuando se requiere una retransmisión o cuando se dispara un evento de tiempo, sin que exista una manera clara de diferenciar el evento; lo que provoca que el diseño de nuevas aplicaciones y su mantenimiento sea complicado.

Comparativa con las características de uinet

La pila uinet se diseñó y programó desde cero considerando las ideas y limitaciones de los trabajos mencionados anteriormente, se diferencia de los demás trabajos principalmente por la interfaz de programación basada en eventos de TCP, la implementación de las extensiones IP multidifusión y la implementación del protocolo IGMP. Las características más importantes de uinet, así como de los otros trabajos relacionados se resumen en el cuadro 3.3.

En conclusión, la pila Open TCP eleva el costo final de un sistema ya que requiere de un microcontrolador muy potente solo para su ejecución, llegando incluso al punto en el que el hardware para correr la pila es más caro que todo el hardware de la aplicación. La pila μIP dificulta el diseño y mantenimiento de nuevas aplicaciones debido a que la interfaz de programación es muy limitada. La pila TCP/IP Lean solo puede utilizarse en el microcontrolador para el que fue diseñado debido a que está optimizada para ello, cambiar el microcontrolador implica muchos cambios en el código fuente de la misma. La pila uinet es tan robusta como la pila μIP, tiene una interfaz de programación basada en eventos que

es sencilla de utilizar, puede emplearse en microcontroladores de 8 bits y con muy pocos cambios se puede emplear en un microcontrolador distinto al PIC18F4525.

3.13. Requerimientos de huésped de Internet RFC1122 para uinet

El documento RFC1122 [8] define los requerimientos que un huésped debe cumplir antes de ser conectado a internet. Este documento define los requerimientos que a lo largo de la experiencia técnica de varios proveedores de software, universidades e investigadores han determinado en consenso como los mas importantes a cumplir; ésto sin llegar a ser todos los necesarios, debido a que algunas características de los protocolos son más importantes que otras y algunas son incluso opcionales.

Los requerimientos en el RFC1122 están expresados en términos de *debe*, *debería*, *podría*, *no debería* y *no debe*. Por ejemplo: la capa de internet *debe* implementar el protocolo IP y el ICMP. Como la pila uinet está pensada para soportar aplicaciones del tipo servidor, solo un subconjunto de todos los requerimientos especificados en el RFC1122 se aplican a la implementación.

En los cuadros 3.4, 3.5 y 3.6 se muestran los requisitos marcados como *debe* en el RFC1122 que la pila uinet debe cumplir en la capa de internet y en la capa de transporte. Como se puede apreciar en los cuadros, la pila uinet cumple con la mayoría de los requisitos: los necesarios para el funcionamiento de la capa de aplicación, el servidor HTTP. Los requerimientos no cubiertos, no se implementaron debido a que para la aplicación, el servidor Petit, no son estrictamente necesarios para su funcionamiento; mientras que otros, no se implementaron debido a las restricciones de memoria en el microcontrolador.

Como se menciona en el RFC1122, no todos los requerimientos aplican para todas las implementaciones; en el caso de la pila uinet los requerimientos mostrados en los cuadros 3.4, 3.5 y 3.6 son los mínimos para operar el servidor HTTP que se describe en el siguiente capítulo.

Requisitos protocolo IP	Implementado
Descartar paquetes con versión diferente de la 4	si
Verificar IP checksum y descartar datagramas erróneos	si
Dirección de origen es la dirección IP propia	si
Descartar datagramas con dirección de destino errónea	si
Descartar datagramas con dirección de origen errónea	si
Reensamblado de paquetes IP	no
Permitir capa de transporte especificar el valor de TTL	no
TTL configurable	si
Permitir a la capa de transporte enviar opciones IP	si
Pasar todas las opciones IP a la siguiente capa	si
Descartar opciones IP no implementadas	si
Descartar mensajes ICMP desconocidos	si
Regresar mensajes ICMP de error	no
Pasar mensajes de destino inalcanzable a capa superior	si
Servidor eco y cliente eco	no (solo servidor)
Enviar mismos datos en respuesta eco	si
Pasar mensajes de respuesta eco a capa superior	no
Soporte multidifusión	si
Protocolo IGMP	si
Unirse al grupo all-host al inicio	no
Capas superiores pueden utilizar multidifusión	si

Cuadro 3.4: Requisitos de la capa de Internet

Requisitos protocolo UDP	Implementado
Enviar puerto no disponible	no
Generar y verificar checksum	si
Descartar mensajes con checksum erróneo	si
Capa de aplicación puede especificar dirección IP local	si
Enviar únicamente direcciones IP válidas	si
Interfaz IP completa disponible a la aplicación	si

Cuadro 3.5: Requisitos capa de transporte para UDP

Requisitos protocolo TCP	Implementado
Bandera PUSH	si
Llamada a enviar puede especificar el uso de PUSH	si
Último segmento con bandera PUSH	si
Ventana TCP	no
Datos urgentes	no
Opciones TCP	si
Recepción de opciones TCP en cualquier segmento	no
Ignorar opciones no implementadas	si
Envío y recepción de opción MSS	si
Enviar siempre MSS	si
MSS por defecto es 536	no
Calcular checksum al enviar y recibir segmentos	si
ISN generado en base a reloj	si
Abrir conexiones simultáneas	no
Descartar mensajes para direcciones broadcast o multicast	si
Informar a aplicación de conexión abortada	si
Esperar en estado Time-wait 2 x MSL segundos	no
Algoritmo Karn para retransmisiones	si
Algoritmo de Jacobson para calcular RTO	si
Exponential backoff	si
Procesar todos los paquetes en cola antes de ACK	no
Enviar ACK para paquetes en desorden	si
Acks retrasados	no
TTL configurable	si
Envío de paquetes keep-alive	no
Procesar paquetes ICMP de IP	si
Validación de direcciones IP	si

Cuadro 3.6: Requisitos de la capa de transporte para TCP

Capítulo 4

Servidor web Petit

La pieza de software más compleja de este trabajo es el programa servidor web Petit que se describe en este capítulo. El servidor Petit se diseñó conforme a la pila uinet y junto con ésta proveen a la aplicación de ejemplo, la red de sensores, la funcionalidad necesaria para transmitir datos por internet.

El servidor Petit es la implementación de un servidor HTTP/1.1. El servidor Petit, puede servir recursos estáticos y/o dinámicos. Soporta conexiones persistentes, transferencia de archivos de más de 1500 bytes y compresión de contenido entre otras características.

En este capítulo se describe el diseño y la implementación del servidor Petit. Al igual que con la pila uinet la descripción del software se hace mediante diagramas de flujo. El código fuente del servidor, así como los programas listos para cargar en el microcontrolador también se encuentran en el disco compacto anexo.

El servidor Petit está diseñado para integrarse con la pila uinet. Mediante el uso de los eventos de la pila, el servidor toma parte activa en las retransmisiones de segmentos perdidos. También gracias al diseño de la pila, el servidor puede enviar respuestas HTTP que abarcan más de un segmento TCP.

El servidor Petit corresponde al nivel de aplicación del modelo de internet. Se encarga de responder a las peticiones HTTP hechas por un navegador web, enviando una respuesta HTTP acorde a la petición. Los navegadores Internet Explorer, Firefox, Netscape Navigator y Safari se utilizaron en durante el desarrollo y pruebas del servidor Petit.

4.1. Registro de sesión web

Cuando un navegador envía una petición a Petit, éste almacena la información más relevante de la sesión: tiempo de inactividad, datos de la última respuesta y del recurso solicitado por el cliente; en un registro `web`. La estructura de éste se muestra en el listado 4.1. Los registros `web` de todos los clientes se almacenan en el arreglo `usuarios`.

```
typedef struct {
   uint tkeepalive;  // contador keepalive
   uint peticion:4;  // registro del tipo de la ultima peticion
   uint resp:4;      // tipo de la ultima respuesta generada
   uint flags;       // banderas servidor
   uint npaq;        // indicador de inicio de respuesta
   freg archivo;     // el archivo que estoy enviando
   tiempo tstamp;    // el registro de tiempo de la ultima respuesta
} web;

web usuarios[MAX_CLIENTES];
```

Listado 4.1: Estructura del registro `web`

A lo largo de las siguientes secciones se explicará el uso de los campos del registro `web`.

4.2. Sistema de archivos

Las páginas HTML y todo el contenido web del servidor Petit se encuentran almacenados en una memoria no volátil externa al microcontrolador. El sistema de archivos utilizado en ésta, se diseñó de acuerdo a las necesidades del servidor Petit y cuenta con las siguientes características:

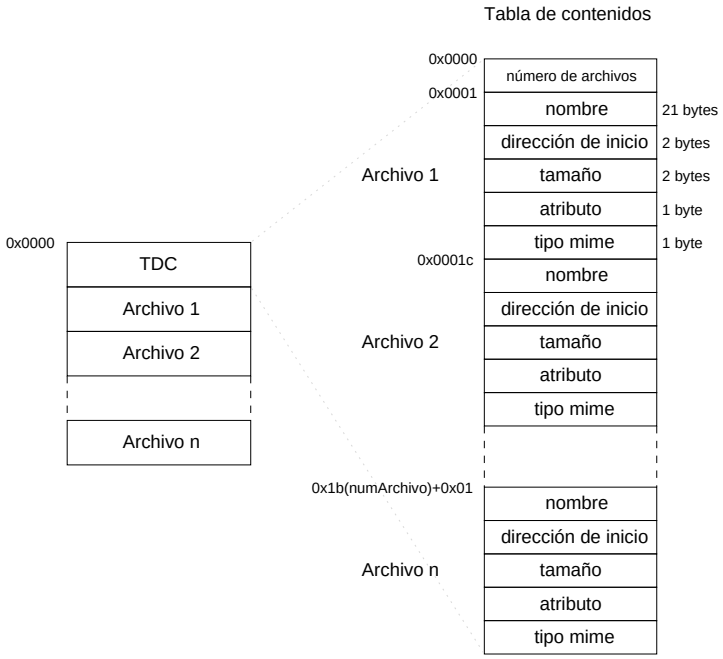


Figura 4.1: Sistema de archivos

Código	Tipo	Descripción
0x00	binario	Archivos con extensión distinta de .html, .htm, .js y .egi.
0x01	dinamico	Archivos cuya extensión sea .egi o .js.
0x02	comprimido	Archivos con extensión .html o .htm

Cuadro 4.1: Tipos de archivos

- Sistema de solo lectura.
- Soporte para nombres de archivos de hasta 21 caracteres (con la extensión incluida).
- Puede almacenar hasta 256 archivos.
- Especificación del tipo de archivo (binario, dinámico y comprimido).
- Especificación del tipo MIME del contenido del archivo.

El sistema de archivos está estructurado como se muestra en la figura 4.1. Básicamente éste se compone de la tabla de contenidos (TDC) seguida del contenido de cada archivo. El primer byte de la TDC indica el número de archivos que contiene la memoria. Por cada archivo hay una entrada en la TDC que contiene: el nombre del archivo, la dirección de la memoria en donde comienzan los datos del archivo, el tamaño, el tipo de archivo y el tipo MIME[1].

El tipo de archivo determina la forma en que el servidor procesará y transmitirá el archivo: permite identificar el contenido estático del dinámico, así como seleccionar la codificación de transferencia y la información que el servidor enviará sobre el archivo. Los archivos `.html` y `.htm` se encuentran comprimidos en el sistema de archivos y por ello marcados como `comprimidos`. El tipo de un archivo puede ser alguno de los mostrados en el cuadro 4.1.

Cuando el servidor envía un archivo a un cliente debe especificar el tipo MIME del contenido del archivo. Los tipos MIME definidos en el servidor[2] son los mostrados en la tabla 4.2. Esta información se incluye en el sistema de archivos como una mera optimización: evita analizar el archivo para determinar el tipo.

Se desarrolló una API para acceder al sistema de archivos desde el software. Esta API provee funciones para abrir, cerrar y leer un archivo, y está diseñada para mantener abierto solo un archivo a la vez. Las funciones de la API del sistema de archivos se detallan en el apéndice C.3.

Un archivo se identifica completamente con la estructura de datos `freg`, mostrada en el listado 4.2. Al abrir un archivo el registro `factual` se inicia con la información del archivo. El campo `ap` es un apuntador al contenido del archivo, que con cada lectura avanza hasta el último byte leído. Si al efectuar una lectura se llega al fin del archivo la lectura se detiene y la bandera `feof` se verifica en 1.

El sistema de archivos se crea en una PC con la herramienta `makerom`, diseñada específicamente para esta tarea. Esta herramienta crea la imagen ROM del sistema de archivos a

[1] *Multipurpose Internet Mail Extensions* es un estándar de internet para describir el tipo de contenido de un mensaje. Los mensajes MIME pueden contener texto, imágenes, audio, etc. En los documentos RFC2045 a 2049 se describen los tipos MIME.

[2] La lista de tipos MIME es muy extensa y solo se definieron los tipos correspondientes al contenido web desarrollado para el servidor.

Código	Tipo
0x00	text/plain
0x01	text/html
0x02	image/gif
0x03	image/png
0x04	image/jpeg
0x05	image/x-icon
0x06	application/x-javascript

Cuadro 4.2: Definición de tipos MIME

```
typedef struct {
   int16 tam;  // tamano en bytes del archivo
   int16 ap;   // offset al contenido del archivo (donde voy)
   int16 dini; // direccion de inicio del archivo en la eeprom
   int8  tipo; // tipo del archivo (binario, dinamico, comprimido)
   int8  mime; //  content-type
} freg;

freg factual;

int1 feof;
```

Listado 4.2: Estructura del registro `freg`

partir del contenido de una carpeta. Al crear la imagen los archivos con extensión `.html` son comprimidos mediante el algoritmo `gzip` (GNU zip). En la sección A.5 se describe la manera de utilizar esta herramienta.

4.3. Manejo de eventos de la pila

Petit está diseñado en torno a la pila uinet y su esquema de eventos. El primer evento que se dispara en el transcurso de una conexión es `ev_tcp_ini()`. En ese momento se sabe que un cliente ha establecido una conexión exitosa con el servidor y que está por mandar una petición. En el evento, el servidor asocia la estructura `TCB` del cliente con un registro `web`; en otras palabras, el cliente se registra con el servidor.

Cuando se recibe la petición del cliente, se dispara el evento `ev_tcp_data()`, indicando que el servidor debe procesar la solicitud y generar una respuesta en consecuencia. En el evento, el servidor verifica la versión HTTP de la petición y procesa la solicitud según el método indicado en la misma como se ilustra en el diagrama de flujo de la figura 4.2. La respuesta enviada en este punto abarca la cabecera HTTP y una parte del recurso solicitado. Si la versión especificada es la 1.0, el servidor cerrará la conexión después de enviar la respuesta debido a que esa versión no soporta conexiones persistentes, véase la sección 4.9.

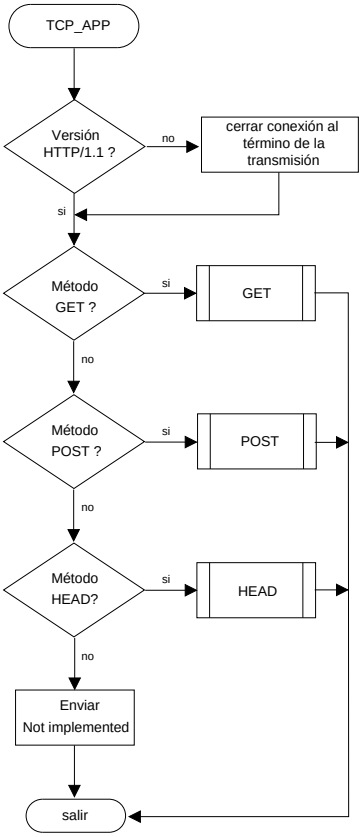


Figura 4.2: Diagrama de flujo del evento `ev_tcp_data()`.

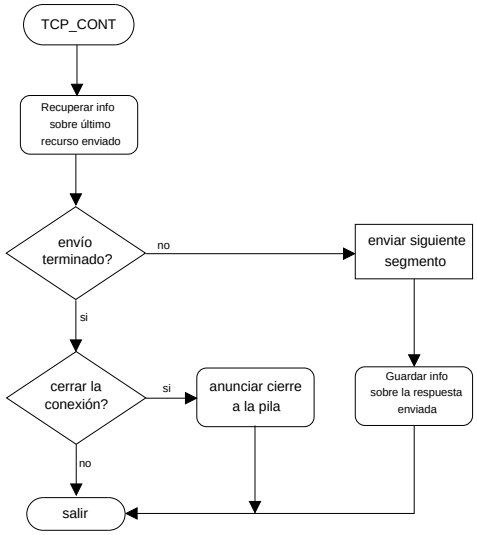


Figura 4.3: Diagrama de flujo del evento `ev_tcp_ack()`.

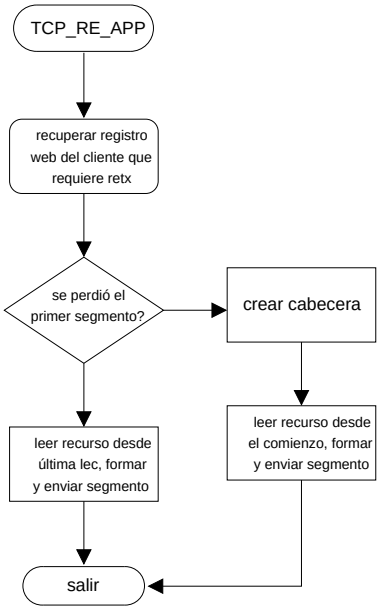


Figura 4.4: Diagrama de flujo del evento `ev_tcp_rtx()`.

Si la respuesta del servidor incluye un recurso, muy probablemente la respuesta abarcará más de un segmento TCP. El primer segmento se envía en el evento `ev_tcp_data()` e incluirá la cabecera y una parte del cuerpo del mensaje. El resto del mensaje se transmitirá en segmentos posteriores que serán enviados desde el evento `ev_tcp_ack()`. El cliente enviará un acuse de recibo por cada segmento que reciba. El acuse ocasionará en el servidor que se disparare el evento `ev_tcp_ack()`, con lo que se podrá enviar el siguiente segmento del mensaje. Este proceso continuará hasta que se haya enviado toda la respuesta.

El registro `web` se utiliza en el evento `ev_tcp_ack()` para determinar desde que punto se debe continuar con el envío del recurso solicitado. Esto se logra gracias al campo `freg` que mantiene la información de la posición en la que se leyó por última vez el recurso en el sistema de archivos. Con esta información el archivo es abierto nuevamente y leido hasta formar un nuevo segmento. La figura 4.3 contiene el diagrama de flujo del proceso.

Si alguno de los segmentos de la respuesta se perdiera durante su transmisión, el evento `ev_tcp_rtx()` se dispararía con el fin de que el servidor vuelva a enviarlo. En este evento, el registro `web` es asociado con el cliente que necesita la retransmisión. El servidor puede entonces recrear a partir del registro `web` el último segmento que envió. Esto se ilustra en la figura 4.4.

Finalmente, el cierre de la conexión por parte del cliente disparará el evento `ev_tcp_fin()`, en el cuál el servidor simplemente eliminará el registro `web` asociado con el cliente.

4.4. Respuestas HTTP

Cuando el servidor Petit recibe la petición de un cliente, contesta con una respuesta formada por una cabecera y un cuerpo de mensaje opcional como lo establece el protocolo HTTP. La primera línea de la cabecera de respuesta es la línea de estado y es la más importante por que indica el resultado de la petición. El formato de la línea de estado comienza con la versión HTTP del servidor, seguida del código de respuesta y una frase explicativa.

El servidor Petit puede responder a las peticiones con alguna de las siguientes líneas de estado:

- `HTTP/1.1 200 OK`: Indica que la petición fue recibida y comprendida; la respuesta a la petición se encuentra en el cuerpo del mensaje.
- `HTTP/1.1 404 Not Found`: La petición fue recibida y comprendida, pero el recurso solicitado no se encuentra en el servidor.
- `HTTP/1.1 501 Not Implemented`: La petición fue recibida, pero no comprendida. Este mensaje puede enviarse cuando el método indicado en la línea de petición no está implementado o cuando la línea de petición no fue comprendida.

Después de la línea de estado la cabecera incluye etiquetas que indican aspectos importantes acerca de la comunicación. Las etiquetas que Petit incluye en la cabecera son distintas según el contenido del mensaje, éstas se describen en las secciones 4.6 y 4.7.

El registro `web` contiene un campo denominado `resp`. Este campo se utiliza para guardar la respuesta que se envió y poder reconstruir la cabecera en caso de ser necesario. El campo `resp` adquiere los valores mostrados en el cuadro 4.3.

`resp`	Respuesta
0x00	404 Not Found
0x01	501 Not Implemeneted
0x02	200 OK

Cuadro 4.3: Valores del campo `resp` del registro `web`

4.5. Métodos implementados

El protocolo HTTP define varios métodos que operan sobre los recursos del servidor, estos se describieron en la sección 1.12. Según la especificación del protocolo, la implementación de los métodos HTTP es opcional excepto por los métodos GET y HEAD. El servidor Petit implementa estos métodos además del método POST.

El procesamiento de una petición con el método GET se ilustra en el diagrama de flujo de la figura 4.5. El servidor busca en el sistema de archivos el recurso especificado en la petición, si lo encuentra lo incluye en una respuesta `200 OK` y se lo envía al cliente. De no encontrarlo, el servidor envía la respuesta `404 Not found`.

El procesamiento del método HEAD, figura 4.6, es idéntico al método GET excepto por que las respuestas enviadas al cliente no incluyen contenido.

El método POST tiene una relación directa con los formularios HTML, ya que este método se utiliza para enviar los datos de un formulario al servidor. Cuando el servidor recibe una petición con el método POST, en lugar de enviar el recurso solicitado al cliente, ejecuta el recurso con los datos enviados en la petición y envía la salida de la ejecución al cliente (genera contenido dinámico).

Los recursos solicitados desde el método POST son recursos ejecutables y se encuentran embebidos en el código del servidor. El servidor incluye dos recursos ejecutables: `tcpip.egi` y `conf.egi`, éstos se describen en detalle en la sección 4.10. El diagrama de flujo de la figura 4.7 muestra como se procesa una solicitud POST.

El protocolo HTTP señala que un servidor debería verificar la línea de petición y buscar una etiqueta `Host` con el nombre del servidor. Por conveniencia y debido a las limitaciones de memoria de programa y memoria RAM, Petit solo procesa la línea de petición y el cuerpo del mensaje (en el caso del método POST) e ignora las etiquetas de la cabecera, incluida la etiqueta `Host`.

El procesamiento de la línea de petición es sensible al uso de letras minúsculas y mayúsculas: los métodos y la versión del protocolo deben estar escritas en letras mayúsculas, de lo contrario Petit enviará la respuesta `501 Not implemented`.

Petit está diseñado específicamente para soportar la versión 1.1 del protocolo HTTP. Si un cliente con una versión diferente hace una petición, la respuestas de Petit pueden no ser las esperadas.

Los métodos implementados pueden generar contenido dinámico, estático o vacío (solo la

Contenido	Método HEAD	Método POST	Método GET
Estático			*
Dinámico		*	*
Vacío	*		

Cuadro 4.4: Tipo de contenido que puede generar cada método en una respuesta exitosa

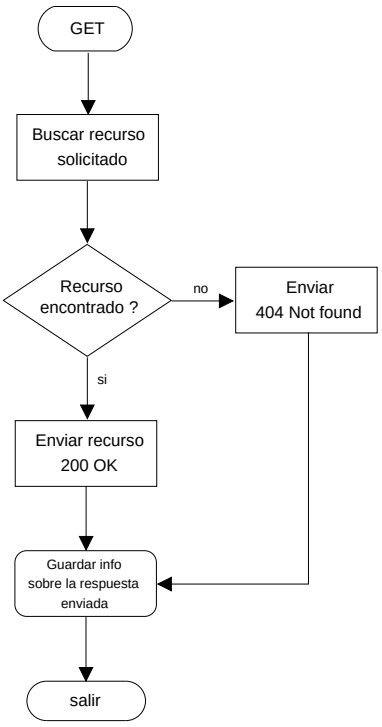


Figura 4.5: Diagrama de flujo del método GET

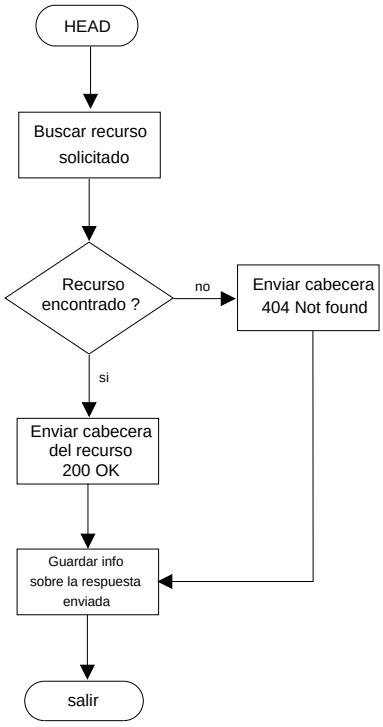


Figura 4.6: Diagrama de flujo del método HEAD

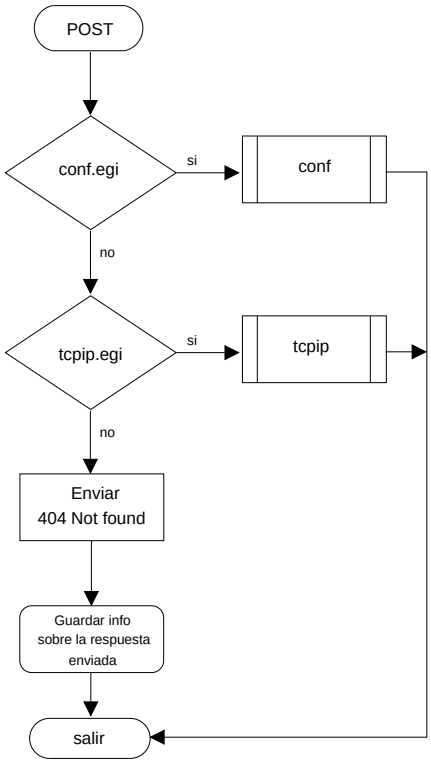


Figura 4.7: Diagrama de flujo del método POST

contenido	Descripción
0x01	Recurso estático
0x02	Recurso dinámico
0x03	Vacío, sólo cabecera

Cuadro 4.5: Valores que puede tomar el campo `contenido` del registro `web`

cabecera). En la tabla 4.4 se muestra el tipo de contenido que puede generar cada método en una respuesta exitosa (`200 OK`). Las demás respuestas incluirán siempre contenido estático, salvo en el caso del método HEAD que siempre enviará respuestas sin contenido, solo con la cabecera. El contenido estático se describe en la sección 4.6.

El registro `web` contiene el campo `contenido` para guardar el tipo de contenido que se envío en la respuesta. Los valores que este campo adquiere se muestran en el cuadro 4.5.

4.6. Contenido estático

Los archivos marcados con el tipo `binario` o `comprimido` en el sistema de archivos, son considerados como contenido estático ya que no cambian entre una petición y otra.

Petit transmite los archivos de tipo binario "tal cual": sin aplicar ninguna modificación al contenido. La cabecera del mensaje en el que se incluirá el archivo `binario` incluirá las siguientes etiquetas:

- `Server`: Indica el nombre del servidor, petit.
- `Date`: Indica la hora y la fecha en la que fue generado el recurso.
- `Content-Type`: Indica el tipo MIME del cuerpo del mensaje.
- `Content-Length`: Indica la longitud del mensaje.

La hora[3] que se incluye en `Date` corresponde a la hora del meridiano de Greenwich. El formato de la misma es el descrito en RFC822 [5]: es una cadena de texto de longitud fija que incluye: día de la semana, día del mes, mes, año, horas, minutos y segundos. P. ej: `Sun, 06 Nov 1994 08:49:37 GMT`.

El `Content-Type` y el `Content-Length` corresponden a los datos del archivo enviado en el mensaje, y son obtenidos del sistema de archivos.

La memoria externa del servidor está limitada en tamaño y no está pensada para albergar páginas web muy grandes que cuenten con un diseño complejo. Los archivos `.html` y `.htm` son propiamente las páginas web del servidor y los archivos de mayor tráfico. Con la finalidad de poder albergar más paginas web en la memoria del servidor y hacer las transferencias de los mismos más eficientes, las páginas web se encuentran comprimidas en el sistema de archivos con el programa de compresión GNU zip como se describe en RFC1952 [10].

Un mensaje del servidor que incluya la etiqueta: `Content-encoding: gzip`, indica que el contenido del mensaje está comprimido y por lo tanto el receptor debe descomprimirlo antes de presentarlo al usuario.

La compresión de las páginas web hace que una página web ocupe menos espacio en el sistema de archivos y que la transferencia de la misma se realice en menos segmentos TCP.

[3]La API para accesar a los datos del reloj del servidor se describe en el apéndice C.2.

Las etiquetas que el servidor incluye al enviar un archivo marcado como `comprimido` (páginas web) son:

- `Server: petit`: Indica el nombre del servidor.
- `Date`: Indica la hora y la fecha en la que fue generado el recurso.
- `Content-Type`: Indica el tipo MIME del cuerpo del mensaje.
- `Content-Encoding: gzip`: Indica que el cuerpo del mensaje está comprimido con gzip.
- `Content-Length`: Indica la longitud del mensaje comprimido.

El uso de la codificación gzip está definida en la versión 1.1 del protocolo HTTP. Si un cliente con un navegador HTTP/1.0 hace una petición al servidor solicitando un recurso que se encuentra `comprimido`, el servidor enviará un mensaje indicando que es necesario actualizar el navegador a la versión[4] HTTP/1.1.

4.7. Contenido dinámico

Parámetro	Significado	Ejemplo de sustitución
`@1`	Dirección IP actual	`192.168.1.150`
`@2`	Dirección IP grupo multicast actual	`239.168.0.1`
`@3`	Número de conexiones TCP activas actuales	`2`
`@4`	Hora actual en formato local	`16/09/2005 14:45:09 GMT -06`
`@5`	Dirección MAC actual	`00-0D-93-B7-1E-A6`
`@6`	Versión de firmware	`0.99`
`@7`	Puerto HTTP actual	`80`
`@8`	Puerto UDP actual	`13068`
`@9`	Dirección IP y puerto HTTP futuros	`192.168.1.152:81`
`@0`	Dirección MAC temporal	`00-0D-93-B7-FF-FF`
`@a`	RTC Hora	`14`
`@b`	RTC Minuto	`45`
`@c`	RTC Segundo	`09`
`@d`	RTC día	`16`
`@e`	RTC mes	`09`
`@f`	RTC año	`2005`
`@g`	RTC GMT	`-6`

Cuadro 4.6: Parámetros permitidos en documentos dinámicos

Los archivos residentes en el sistema de archivos marcados con el tipo `dinamico` son reconocidos por Petit como recursos dinámicos. En la implementación actual solo los archivos con extensión `.egi` y `.js` son marcados de esta forma. Este tipo de archivos pueden contener parámetros que en el momento de la generación de una respuesta son sustituidos por información útil. Los parámetros tienen el formato: @[a-zA-Z0-9], ej.: `@1`, `@4`, `@a`, etc.

Cuando el servidor procesa algún archivo marcado como dinámico sustituye los parámetros del archivo por su significado. Por ejemplo en la figura 4.8a se lista el archivo `ip.egi`. Si el servidor recibiera la petición de este recurso, lo procesaría y generaría el contenido dinámico de la figura 4.8b.

[4]Todos los navegadores comerciales actuales soportan HTTP/1.1.

Como el contenido de la figura 4.8b no ocupa más de 100 bytes: el servidor pudo haberlo mantenido en el bufer principal, calcular su tamaño, incluir la etiqueta `Content-length` y enviarlo. Pero si el contenido hubiera ocupado más de un segmento TCP (más de `MAX_MSS`) no habría sido posible generarlo ni calcular su tamaño final dada la falta de memoria RAM. Para los clientes es necesario conocer el tamaño del cuerpo del mensaje para poder interpretar correctamente el mismo. Cuando no es posible calcular el tamaño de un recurso, Petit utiliza la codificación de transferencia *chunked-encoding* como se describe en la sección 4.8.

```
<html>
<head>
Servidor petit v@6
</head>
<body>
Mi dirección IP es @1
</body>
</html>
```

(a) Archivo `ip.egi`

```
<html>
<head>
Servidor petit v0.99
</head>
<body>
Mi dirección IP es 192.168.1.150
</body>
</html>
```

(b) Contenido dinámico

Figura 4.8: Ejemplo de generación de contenido dinámico

4.8. Codificación chunked-encoding

La codificación *chunked-encoding* permite que conforme el contenido se vaya generando se comience a transmitir, es decir, no se necesita generar todo el contenido antes de enviarlo. Gracias a ésto, Petit puede generar contenido dinámico que abarque más de un segmento.

La codificación chunked funciona enviando tramos de la información. Los cuáles están formados por una cabecera, el tramo de información y una cola. La cabecera contiene el tamaño del tramo (no incluye los bytes de la cabecera ni de la cola), expresado en un número hexadecimal seguido por los caracteres `\r\n`. La cola la forman los caracteres `\r\n`. Cuando se ha transmitido el último tramo de información se transmite un tramo con longitud cero. La figura 4.9 ilustra lo anterior.

El protocolo HTTP no sugiere un tamaño específico para los tramos de información lo que deja libertad para la elección del mismo. Por conveniencia los tramos se hacen para

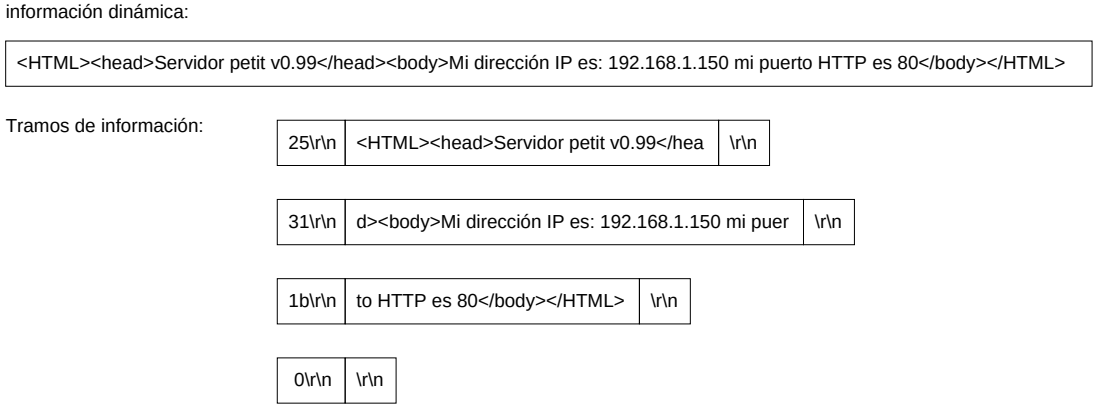


Figura 4.9: Funcionamiento de la codificación *chunked*

ocupar el mayor espacio posible de un segmento TCP. Por lo tanto, Petit envía cada tramo de la información en un segmento TCP, a excepción de los dos últimos tramos (el último tramo de información y el tramo de longitud cero) que podrían enviarse en el mismo segmento.

Cuando un cliente solicita un archivo con contenido dinámico, el servidor utilizará la codificación *chunked* para enviar su respuesta y anunciará el uso de la misma incluyendo la etiqueta `transfer-coding: chunked` en la cabecera de respuesta. Por lo tanto cuando se envía un mensaje con contenido dinámico, Petit incluye en la cabecera las siguientes etiquetas:

- `Server: petit`: Indica el nombre del servidor.
- `Date`: Indica la hora y la fecha en la que fue generado el recurso.
- `Content-Type`: Indica el tipo MIME del cuerpo del mensaje.
- `Transfer-Encoding: chunked`: Indica que el cuerpo del mensaje se transmite por tramos.

En el caso del contenido generado a partir del archivo `ip.egi` listado en la figura 4.8b, el mensaje de respuesta completo sería el mostrado en el listado 4.3.

```
HTTP/1.1 200 OK\r\n
Server: petit\r\n
Date: Wed, 01 Mar 2006 21:10:03 GMT\r\n
Content-Type: text/html\r\n
Transfer-Encoding: chunked\r\n
\r\n
44\r\n
<html>\r\n
<body>\r\n
Mi dirección IP es 192.168.1.150\r\n
</body>\r\n
</html>\r\n
\r\n
0\r\n\r\n
```

Listado 4.3: Respuesta del servidor a la petición del archivo `ip.egi`

4.9. Conexiones persistentes

El servidor Petit soporta el concepto de conexiones persistentes de HTTP/1.1. Esto se refiere al hecho de mantener una conexión abierta y recibir varias peticiones por la misma. En versiones anteriores de HTTP, una conexión solo servía para realizar una petición, lo que ocasionaba una carga extra de trabajo en los servidores, al abrir y cerrar conexiones en cada petición, y hacia imposible el concepto de sesión.

Para el soporte de las conexiones persistentes se utiliza el campo `tkeepalive` del registro `web`. Este campo sirve para limitar el tiempo en el que un cliente puede permanecer inactivo.

Cuando un cliente inicia una conexión al servidor o hace alguna petición, el `tkeepalive` se inicia en 50 segundos, si en ese tiempo el cliente no interactúa nuevamente con el servidor, su conexión será cerrada. Esto permite que cualquier página HTML junto con sus imágenes puedan descargarse utilizando solo una conexión. Sin embargo, en la práctica esto no es en

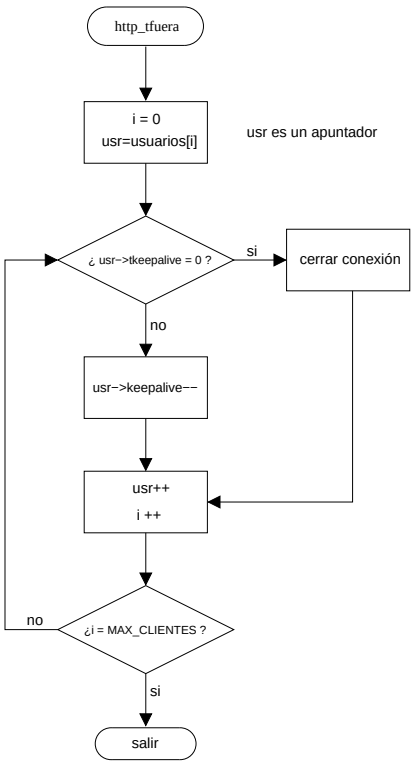


Figura 4.10: Diagrama de flujo de la tarea periódica `http_tfuera`

verdad lo que sucede, ya que cuando los navegadores actuales descargan una página HTML comienzan a descargar en paralelo las imágenes de la misma en conexiones separadas.

Para que el contador `tkeepalive` funcione correctamente se debe decrementar su valor en 1 cada segundo. Para ello, el servidor define la tarea periódica `http_tfuera`. Esta tarea es invocada cada segundo desde el bucle principal y realiza lo siguiente: examina el campo `tkeepalive` de todos los registros `web` del arreglo `usuarios` y decrementa en 1 su valor si éste es distinto de cero. Si el resultado es cero la conexión con el cliente se cerrará y su registro `web` estará libre para albergar a un nuevo cliente. Este proceso se ilustra en el diagrama de flujo de la figura 4.10.

4.10. Recursos ejecutables

Una de las primeras maneras de crear contenido dinámico en un servidor web fue mediante el uso de los CGI (Common Gateway Interface). Esta tecnología permite a un cliente solicitar datos de un programa ejecutado en el servidor web. Los CGI definen un mecanismo de comunicación entre el servidor web y una aplicación externa. En una aplicación CGI, el servidor web pasa las solicitudes del cliente a un programa externo. La salida de dicho programa es enviada al cliente en lugar del archivo estático tradicional.

En el caso del servidor Petit no existe un sistema operativo residente y por lo tanto no es posible ejecutar un programa externo al servidor. Esto hace que el concepto de CGI no se pueda implementar en el servidor. Sin embargo, el servidor provee una interfaz análoga a los CGI: Embedded Gateway Interface (EGI). La interfaz EGI permite ejecutar código en el servidor en respuesta a los formularios enviados por un cliente. Los CGI se ejecutan

como un proceso externo, mientras que los EGI se ejecutan como una llamada de función interna al servidor. El concepto de EGI implementado en este trabajo se basa en el descrito por Bentham [20].

Los EGI son recursos ejecutables accesibles solo desde el método POST. El servidor Petit cuenta con dos de estos recursos: `tcpip.egi` y `rtc.egi`. Para ejecutarlos se diseñaron respectivamente los formularios frmIP y frmReloj mostrados en la figura 4.11. Estos formularios se encuentran en el archivo `config.html`.

Cuando el usuario presiona el botón «Aplicar cambios» de los formularios, el navegador envía los datos del formulario al servidor, éste al recibirlos ejecuta el EGI correspondiente al formulario. Los datos del formulario se convierten ahora en los argumentos de ejecución para el EGI.

Cada argumento tiene un nombre y un valor separados por el caracter '`=`'. El nombre corresponde a un campo del formulario y el valor a la información introducida por el usuario en el campo. El caracter '`&`' se usa para delimitar los argumentos.

Por ejemplo, los datos del formulario `frmIP` se transmiten como los siguientes argumentos para `tcpip.egi`:

```
a=192.168.1.150&b=80&Submit=Aplicar+cambios
```

El argumento `a` indica la dirección IP, `b` el puerto HTTP y `Submit` el botón que presionó el usuario para enviar el formulario.

El `tcpip.egi` sirve para cambiar la dirección IP y el puerto HTTP del servidor desde el navegador. Petit verifica que la dirección IP especificada en el argumento `a` no esté ocupada por otro dispositivo en la red antes de utilizarla. Si la dirección está libre, el servidor enviará al cliente una página con un enlace a la nueva dirección, figura 4.12.

Posteriormente el servidor se reiniciará automáticamente para utilizar la nueva dirección y todos los clientes conectados serán desconectados sin previo aviso. Si la dirección IP especificada se encuentra ocupada, el servidor enviará una página al cliente informando la situación, figura 4.13. Las páginas enviadas por `tcpip.egi` son páginas dinámicas generadas a partir de los archivos `dir.egi` y `ndir.egi`.

El diagrama de flujo de la figura 4.15 resume el funcionamiento de `tcpip.egi`.

El otro recurso ejecutable con que cuenta el servidor es `rtc.egi`. Éste tiene la finalidad de ajustar el reloj de tiempo real del servidor. Los datos necesarios para ajustar el reloj se capturan en el formulario frmReloj. Estos datos se convertirán en argumentos como los siguientes:

```
a=15&b=4&c=2006&h=16&m=56&s=50&u=-6&Submit=Aplicar+cambios
```

Los nombres de los argumentos corresponden a día, mes, año, hora, minuto, segundo y uso horario respectivamente.

El servidor procesa los argumentos y ajusta el reloj a la hora y fecha indicadas. Posteriormente genera una página dinámica, figura 4.14 en la que indica la hora a la que se ajustó el reloj. Esto se ilustra en el diagrama de flujo de la figura 4.16.

Cabe mencionar que el cuando el servidor procesa los argumentos, para cualquiera de los EGI, no verifica que éstos sean válidos. La verificación se realiza en los formularios y no en el servidor. La verificación en el servidor se omitió para simplificar el código del mismo.

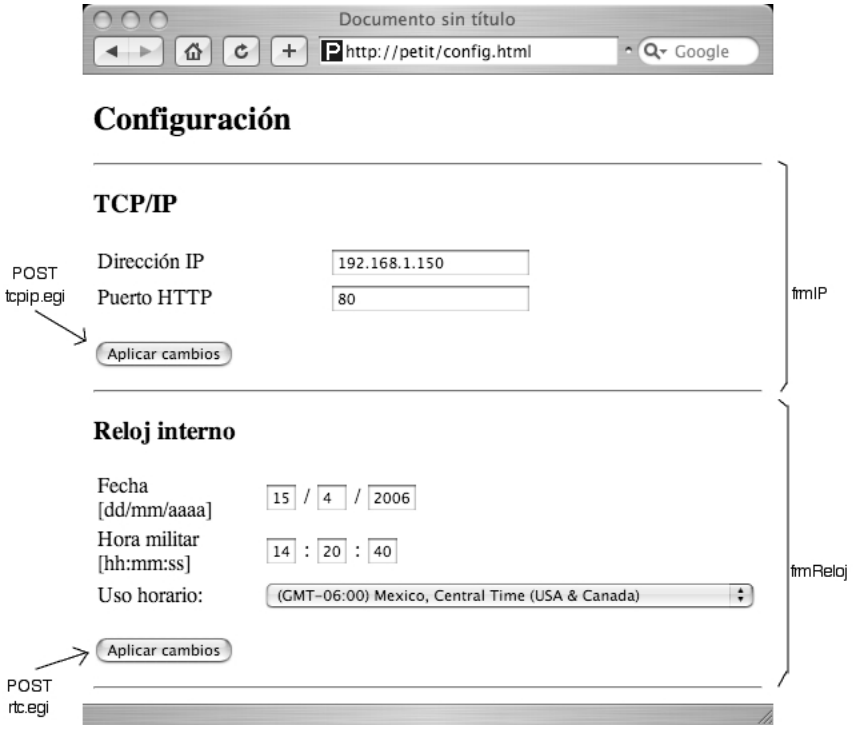


Figura 4.11: Formulario `config.html`

Figura 4.12: Página `dir.egi`

Figura 4.13: Página `ndir.egi`

Figura 4.14: Página `ndir.egi`

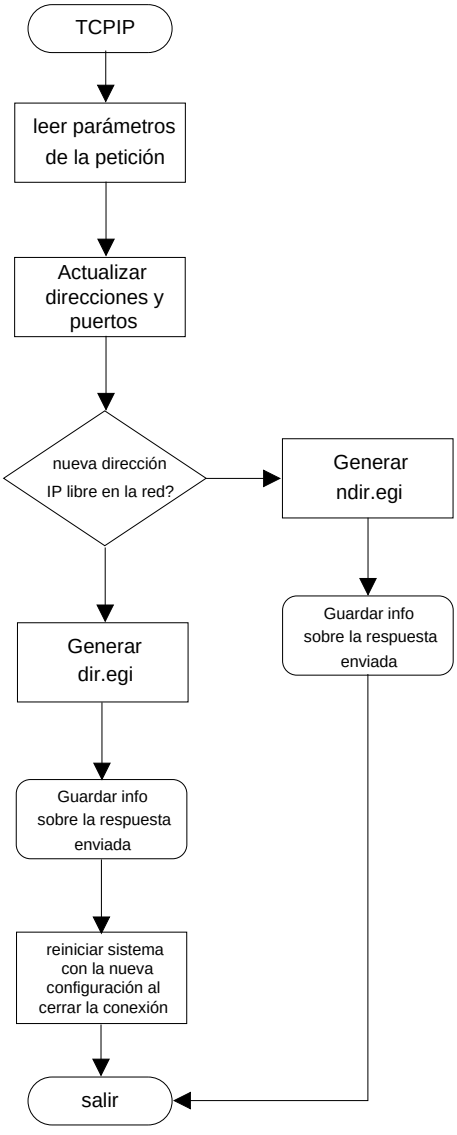


Figura 4.15: Diagrama de flujo del recurso ejecutable `tcpip.egi`

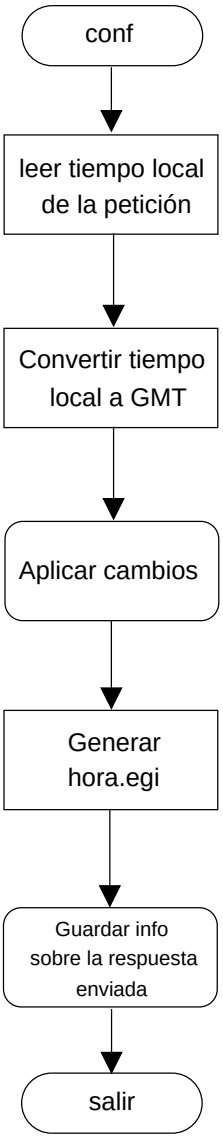


Figura 4.16: Diagrama de flujo del recurso ejecutable `rtc.egi`

4.11. Trabajos relacionados

Al igual que en el caso de la pila uinet, existen algunas implementaciones de servidores HTTP para sistemas embebidos. Algunas de éstas son piezas de software similares al servidor Petit, mientras que otras son componentes comerciales de hardware todo en uno que no permiten la modificación o alteración del software.

En la sección 3.12 se mencionaron algunas implementaciones de la pila TCP/IP similares a uinet, que también incluyen un servidor web. El servidor Petit cuenta con algunas características que no se encuentran en ninguno de esos trabajos. En el cuadro 4.7 se presenta una comparativa de las características de Petit respecto a los servidores HTTP que acompañan a las implementaciones de la pila TCP/IP presentadas en la sección 3.12.

Características	TCP/IP lean	μIP	OpenTCP	uinet
Versión HTTP	1.0	1.0	1.0	1.1
Conexiones persistentes	no	no	no	si
Clientes simultáneos	1	configurable	configurable	configurable
Soporte de codificación *chunked-encodig*	no	no	no	si
Contenido web limitado a 1 segmento TCP	si	no	no	no
Método GET	si	si	si	si
Método POST	no	no	no	si
Método HEAD	no	no	no	si
Interfaz similar a CGI	si	si	no	si
Generación de contenido dinámico	no	si	si	si
Sistema de archivos	no	embebido en código	no	en memoria externa
Interfaz de configuración web	no	no	no	si
Interfaz de configuración serial	no	no	no	si

Cuadro 4.7: Comparativa de las características del servidor Petit con otros trabajos relacionados

4.12. Pruebas de funcionamiento

Para probar todas las características del servidor web se utilizaron tres configuraciones de red en las que participó el servidor.

La primera configuración empleada se muestra en la figura 4.17. Esta configuración se empleó para probar la implementación de los protocolos de la pila uinet, así como la implementación del servidor web. Básicamente es una red ethernet, no conectada a internet, con cuatro nodos: A, B, C y el servidor web.

La segunda configuración se muestra en la figura 4.18. En esta configuración la computadora B está configurada como gateway entre las redes 192.168.1.0 y 192.168.2.0. Con esta configuración se pudo probar la conexión multicast con el servidor web, es decir se utilizó para probar el protocolo IGMP, así como las extensiones multidifusión de la pila uinet. Mediante la configuración apropiada del TTL para los paquetes multidifusión, se logró que la computadora A recibiera el tráfico multicast del servidor web.

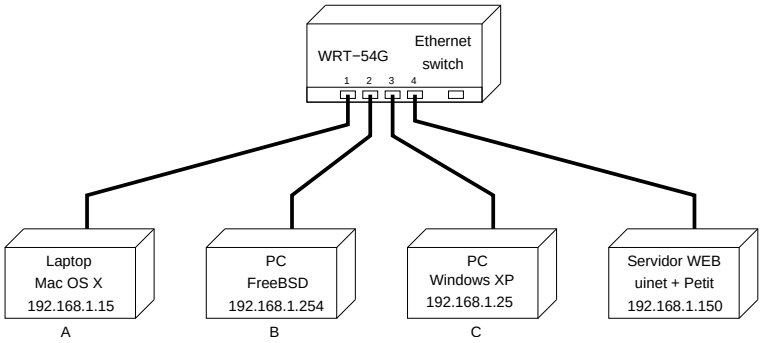


Figura 4.17: Red ethernet utilizada para pruebas del servidor web

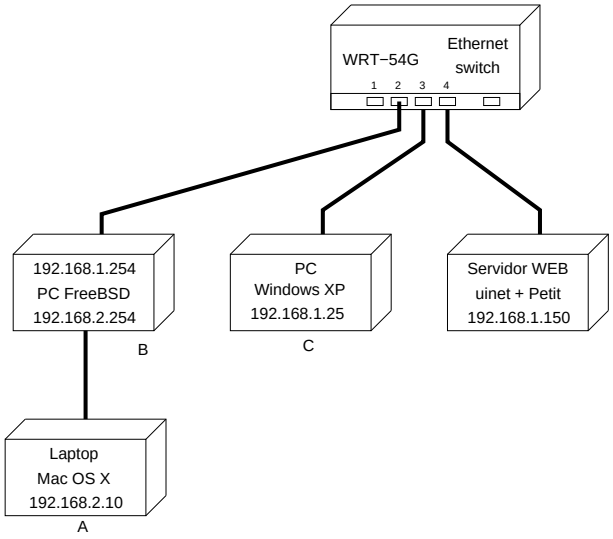


Figura 4.18: Red ethernet utilizada para pruebas de multidifusión

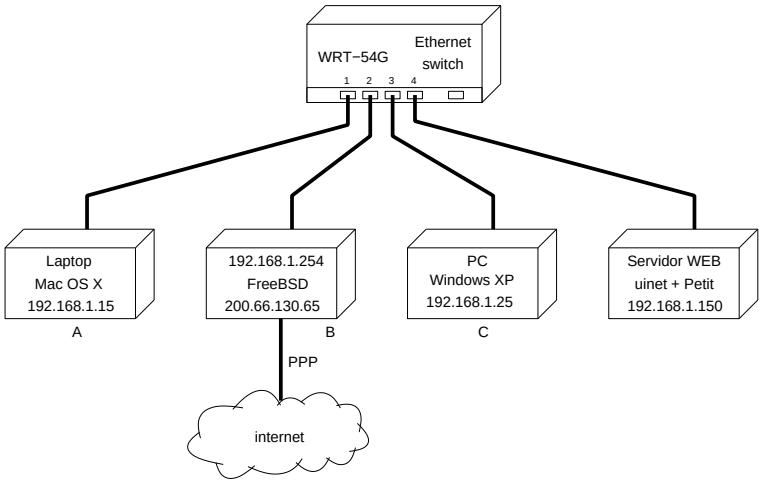


Figura 4.19: Red ethernet utilizada para pruebas desde internet

La última configuración utilizada se muestra en la figura 4.19. En esta configuración la computadora B también está configurada como gateway, salvo que en este caso entre la red local e internet, está conecta a internet mediante un enlace PPP sobre línea telefónica y configurada para redireccionar el trafico TCP en el puerto 80 (puerto HTTP) hacia el servidor web. De esta manera, cualquier computadora conectada a internet puede acceder al servidor web, utilizando la dirección IP de la computadora B. Esta configuración se probó exitosamente desde una computadora ubicada en Ciudad del Carmen, Campeche, otra ubicada en la ciudad de Toluca y una más desde la red UNAM en la ciudad de México, mientras el servidor se encontraba físicamente en el municipio de Xonacatlán, en el estado de México.

Herramientas de software

Para realizar la depuración de la pila uinet y del servidor Petit, se utilizaron las configuraciones de red mencionadas en la sección anterior junto con algunas herramientas de software para analizar el tráfico generado en las redes utilizadas.

Los programas que se usaron para analizar el tráfico en la red fueron `Ethereal` y `tcpdump`. Estos programas capturan el tráfico que llega a la computadora donde se ejecutan y permiten realizar un análisis del mismo. `Ethereal` cuenta con una interfaz gráfica basada en menús y ventanas, figura 4.20; mientras que `tcpdump` se usa desde la línea de comandos. Ambos programas se utilizaron para verificar el funcionamiento de la pila uinet, así como del servidor Petit.

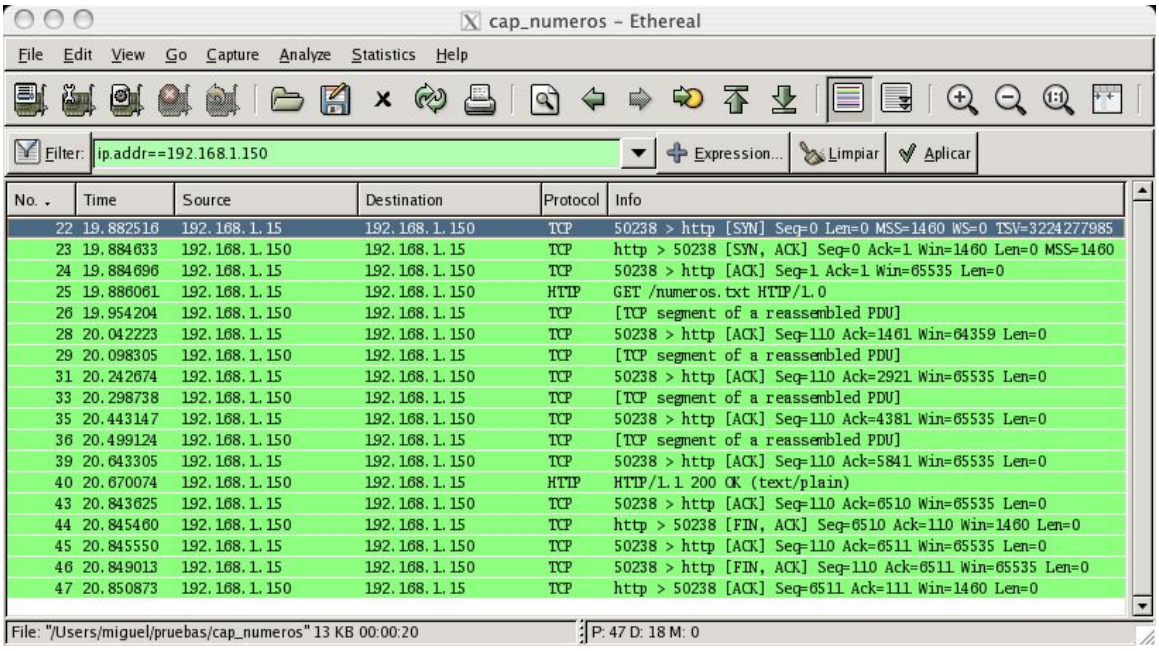


Figura 4.20: Ventana principal de Ethereal

Las pruebas de funcionamiento se realizaron ejecutando el analizador de red en una computadora para capturar el tráfico mientras desde ésta se accedía al servidor web. Los analizadores de red permitieron detectar la mayoría de los errores en la implementación de los protocolos, como por ejemplo:

- Errores en el cálculo de la suma de verificación IP, TCP, UDP, etc.
- Segmentos TCP fuera de secuencia.
- Errores en la máquina de estados TCP.
- Información incompleta en las cabeceras de los protocolos.
- Transferencias de datos incompletas.
- Errores en la secuencia de ejecución.

Específicamente para probar la implementación del protocolo HTTP, servidor Petit, se emplearon los siguientes navegadores web:

- Netscape Navigator 7.2.
- Internet explorer 6.
- Safari 1.3.
- Firefox 1.0.
- Lynx.

Después de varias correcciones al software del servidor, todos los navegadores mencionados pudieron descargar contenido web del servidor Petit sin problemas. Ademas de los navegadores, también se utilizaron los programas `wget` y `curl` para generar peticiones HTTP/1.0 y HTTP/1.1 respectivamente. La ventaja de estos programas respecto a los navegadores web es que después de descargar un recurso del servidor Petit, cierran inmediatamente la conexión TCP al terminar la descarga, y mediante la captura del tráfico generado con estas herramientas se puede determinar el *throughput*[5] efectivo del servidor web.

Finalmente para analizar el desempeño del servidor web, se utilizó el programa `tcptrace`. Este programa analiza las capturas de Ethereal y de tcpdump y genera un reporte con información acerca del desempeño de las conexiones detectadas.

4.13. Análisis de desempeño

El desempeño del servidor web se analizó mediante la herramienta *tcptrace*. Esta herramienta examina los paquetes capturados con ethereal o tcpdump y genera un reporte con la información más relevante sobre las conexiones encontradas.

En el listado 4.4 se muestra el reporte de tcptrace para una conexión realizada entre una computadora y el servidor web, utilizando la configuración de red de la figura 4.17. En este reporte el servidor web se identifica como el host f con la dirección IP 192.168.1.150 y la computadora cliente como el host e (computadora A en la configuración de red) con la IP 192.168.1.15. En la conexión, el cliente solicitó al servidor web el recurso “prueba.jpg”, el cuál es una imagen jpg de 25229 bytes.

[5]La cantidad máxima de datos que puede transmitir el servidor por unidad de tiempo.

```
TCP connection info:
3 TCP connections traced:
================================
TCP connection 3:
        host e:        192.168.1.15:50083
        host f:        192.168.1.150:http
        complete conn: yes
        first packet:  Tue Jun 27 22:09:56.513977 2006
        last packet:   Tue Jun 27 22:10:00.161235 2006
        elapsed time:  0:00:03.647257
        total packets: 44
        filename:      cap_foto
   e->f:                               f->e:
     total packets:            23           total packets:            21
     ack pkts sent:            22           ack pkts sent:            21
     pure acks sent:           20           pure acks sent:            1
     sack pkts sent:            0           sack pkts sent:            0
     dsack pkts sent:           0           dsack pkts sent:           0
     max sack blks/ack:         0           max sack blks/ack:         0
     unique bytes sent:       107           unique bytes sent:     25349
     actual data pkts:          1           actual data pkts:         18
     actual data bytes:       107           actual data bytes:     25349
     rexmt data pkts:           0           rexmt data pkts:           0
     rexmt data bytes:          0           rexmt data bytes:          0
     zwnd probe pkts:           0           zwnd probe pkts:           0
     zwnd probe bytes:          0           zwnd probe bytes:          0
     outoforder pkts:           0           outoforder pkts:           0
     pushed data pkts:          1           pushed data pkts:          1
     SYN/FIN pkts sent:       1/1           SYN/FIN pkts sent:       1/1
     req 1323 ws/ts:          Y/Y           req 1323 ws/ts:          N/N
     adv wind scale:            0           adv wind scale:            0
     urgent data pkts:          0 pkts      urgent data pkts:          0 pkts
     urgent data bytes:         0 bytes     urgent data bytes:         0 bytes
     mss requested:          1460 bytes     mss requested:          1460 bytes
     max segm size:           107 bytes     max segm size:          1460 bytes
     min segm size:           107 bytes     min segm size:           529 bytes
     avg segm size:           106 bytes     avg segm size:          1408 bytes
     max win adv:           65535 bytes     max win adv:            1460 bytes
     min win adv:           65535 bytes     min win adv:            1460 bytes
     zero win adv:              0 times     zero win adv:              0 times
     avg win adv:           65535 bytes     avg win adv:            1460 bytes
     initial window:          107 bytes     initial window:         1460 bytes
     initial window:            1 pkts      initial window:            1 pkts
     ttl stream length:       107 bytes     ttl stream length:     25349 bytes
     missed data:               0 bytes     missed data:               0 bytes
     truncated data:            0 bytes     truncated data:            0 bytes
     truncated packets:         0 pkts      truncated packets:         0 pkts
     data xmit time:        0.000 secs      data xmit time:        3.294 secs
     idletime max:          200.9 ms        idletime max:          208.6 ms
     throughput:               29 Bps       throughput:             6950 Bps

   RTT samples:               3           RTT samples:              20
     RTT min:                 1.9 ms        RTT min:                  0.1 ms
     RTT max:                69.3 ms        RTT max:                175.3 ms
     RTT avg:                24.4 ms        RTT avg:                127.0 ms
     RTT stdev:              38.8 ms        RTT stdev:               47.7 ms

     RTT from 3WHS:           2.2 ms        RTT from 3WHS:           0.1 ms

     RTT full_sz smpls:          1          RTT full_sz smpls:         17
     RTT full_sz min:         69.3 ms        RTT full_sz min:         63.0 ms
     RTT full_sz max:         69.3 ms        RTT full_sz max:        144.9 ms
     RTT full_sz avg:         69.3 ms        RTT full_sz avg:        139.1 ms
     RTT full_sz stdev:        0.0 ms        RTT full_sz stdev:       19.7 ms
```

Listado 4.4: Reporte generado por tcptrace para una conexión al servidor web

En el reporte se aprecia que el tiempo que el servidor tarda en enviar el recurso solicitado es de 3.294 segundos, con un *throughput* efectivo de 6950 Bps (55.6 Kbps). Esta velocidad de transferencia se ve afectada por dos factores principales: la velocidad de procesamiento del microcontrolador del servidor web y la implementación del protocolo TCP.

En este caso la implementación de TCP del cliente es el factor que limita más la velocidad de transferencia del servidor, debido a la forma en la que se implementó la ventana TCP en el servidor, y a que la pila TCP del cliente implementa el algoritmo de ACK's retrasados. Debido a este algoritmo, el cliente podría esperar hasta 500ms antes de enviar un ACK por un segmento recibido. Debido a que la pila uinet no puede enviar más de un segmento TCP sin esperar un ACK a cambio, el cliente con el algoritmo de ACK's retrasados es el que limita la taza de transferencia de datos del servidor web. Esto se aprecia en el reporte al revisar los tiempos de viaje RTT, como el servidor web no implementa el algoritmo de ACK's retrasados, genera un ACK por cada segmento recibido tan rápido como le es posible, por esto es que el RTT de los paquetes que van del cliente al servidor (e->f en el reporte) es menor que el de los que viajan del servidor al cliente (f->e).

En la figura 4.21 se muestra una grafica del comportamiento del *throughput* del servidor web, y se aprecia que este se mantiene constante durante toda la transferencia del archivo.

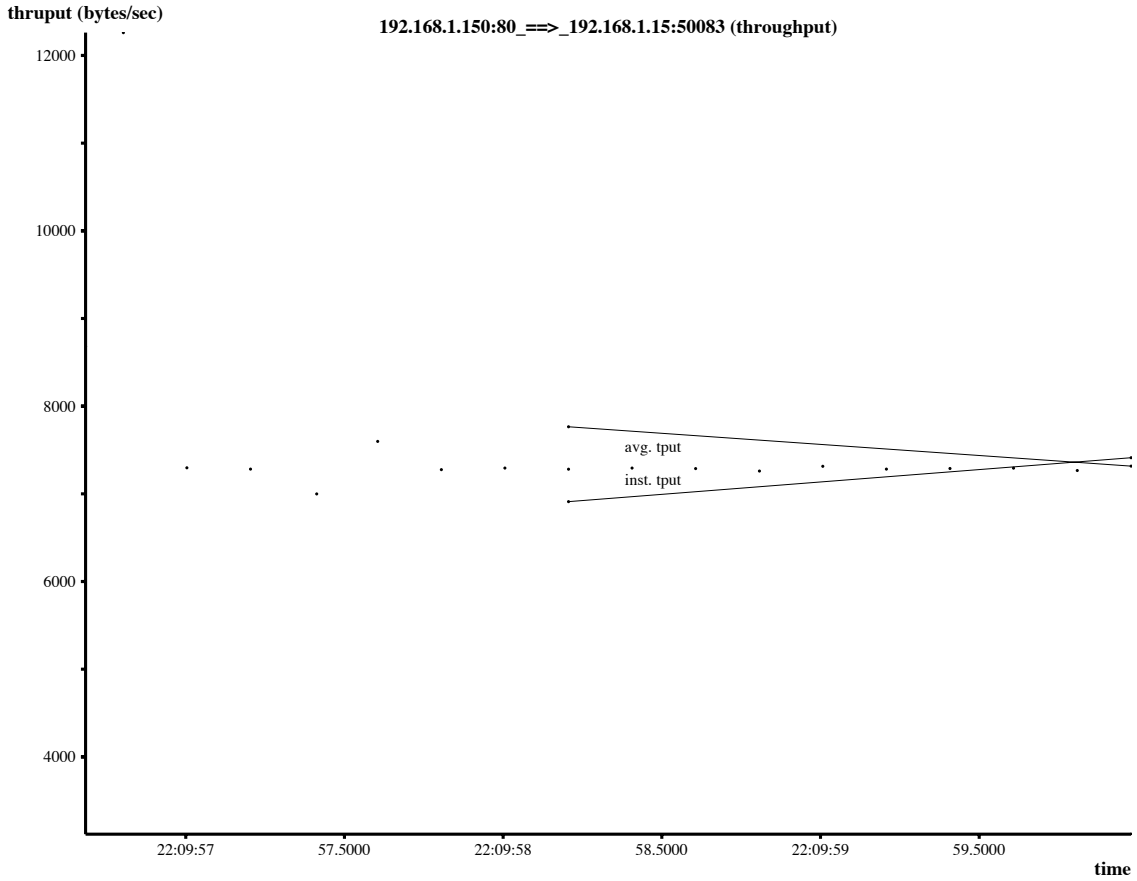


Figura 4.21: Representación del *throughput* del servidor web

En el reporte se aprecia también que la ventana TCP anunciada por el cliente es de 65536 bytes, mientras que la del servidor web es de 1460 bytes. El servidor anuncia esta ventana

para que el cliente envié solo un segmento TCP y espere por un ACK, de lo contrario el cliente podría enviar mas datos pero el servidor web no estaría en condiciones de procesar toda la información debido a las limitaciones en la implementación de TCP en uinet.

Durante la conexión, el servidor generó 18 segmentos TCP para enviar el archivo y 21 ACK's para controlar la comunicación. Dada la configuración de la red utilizada, no se generaron paquetes perdidos o retransmisiones.

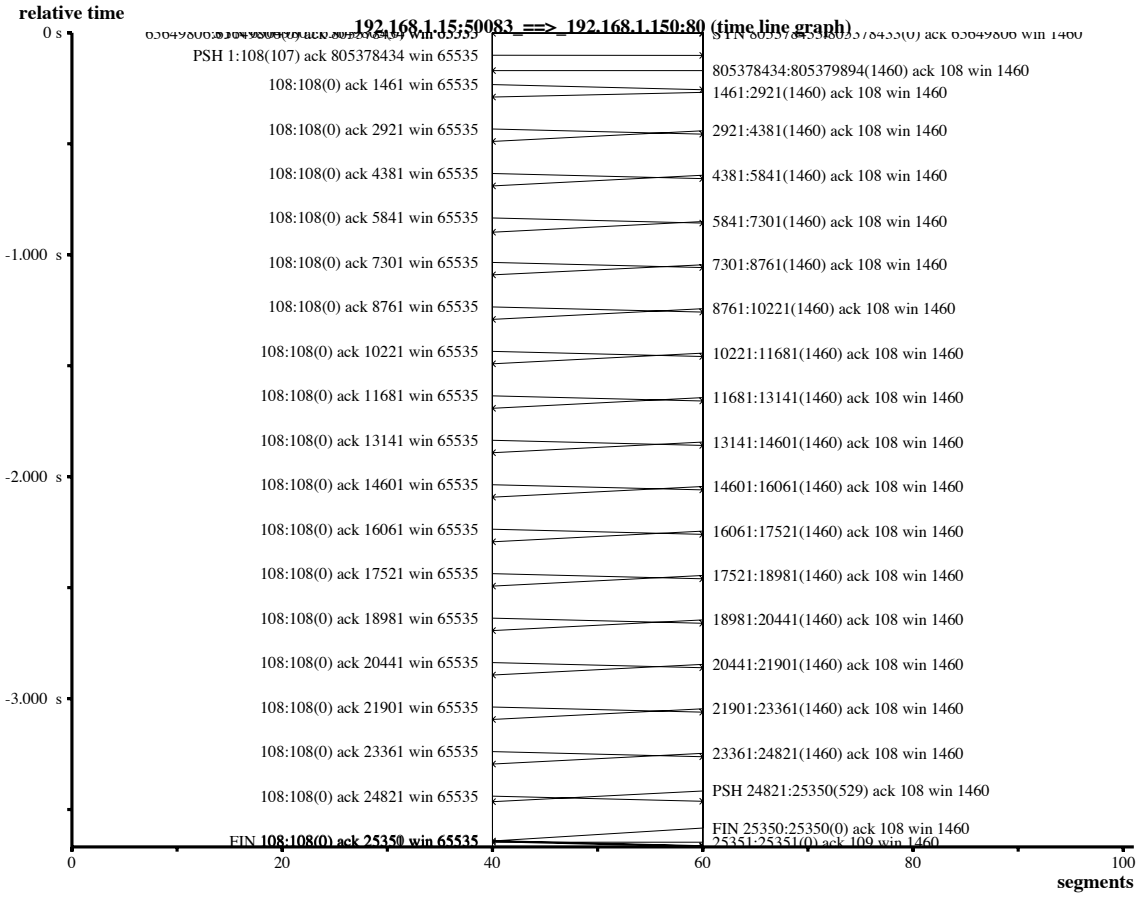


Figura 4.22: Representación del intercambio de paquetes en la comunicación

En la figura 4.22 se muestra una representación del intercambio de paquetes entre el servidor web y el cliente, en ésta se aprecia que el servidor web envía un ACK por cada segmento TCP que recibe. De igual manera, en la figura 4.23 se muestra una representación de los números de secuencia de TCP utilizados por el servidor y por el cliente durante la conexión, esta tiene una forma escalonada bien definida debido a que por cada segmento se genera un ACK. Si la implementación de la ventana TCP en uinet fuese mas robusta, en la figura se mostrarían escalones más altos ya que los números de secuencia aumentarían en magnitud más rápido, debido a que por cada dos o más segmentos se generaría solo un ACK.

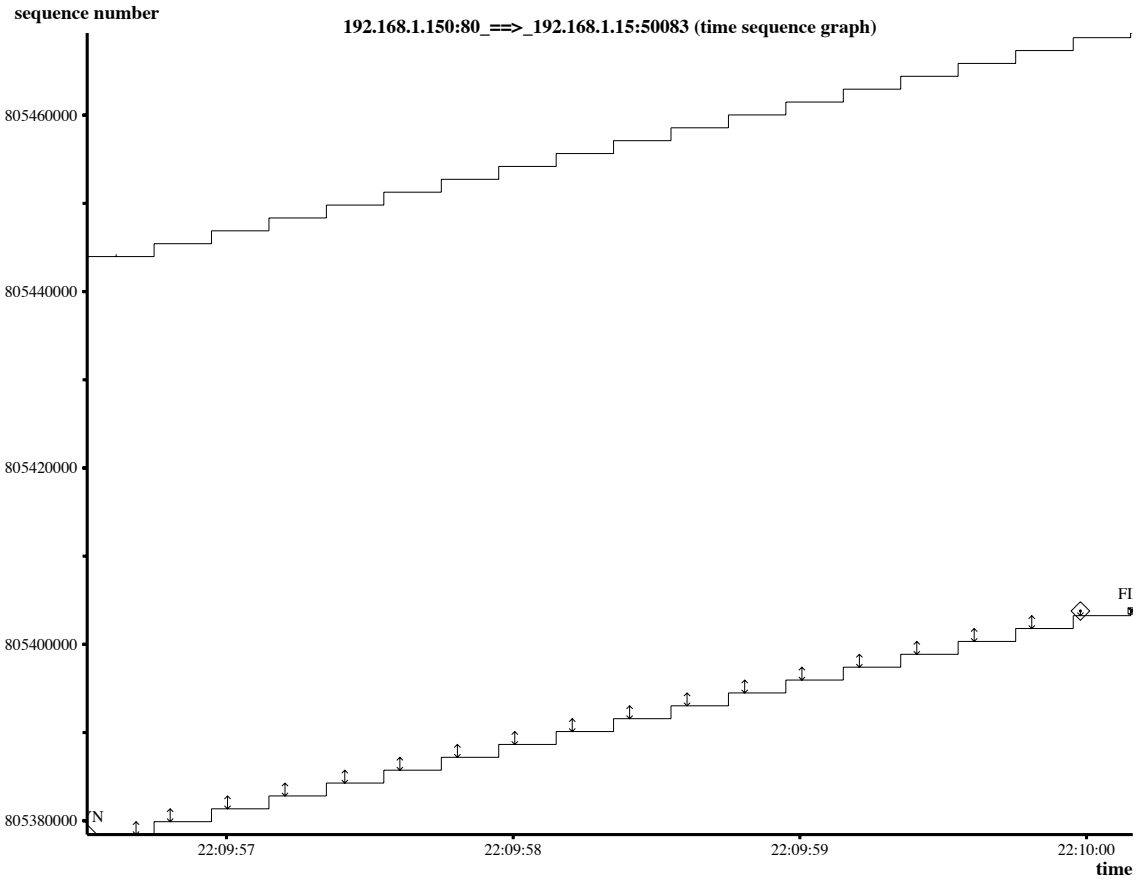


Figura 4.23: Comportamiento de los números de secuencia TCP

Capítulo 5

Ejemplo de aplicación: red de sensores inalámbricos

En este capítulo se describe un ejemplo de aplicación que utiliza el hardware y software del servidor web para comunicarse a través de internet: una red centralizada de sensores de temperatura inalámbricos que transmiten sus mediciones a través de internet.

La red de sensores se podría utilizar para supervisar la temperatura en algún proceso industrial; como un *datalogger* de temperatura en un experimento de laboratorio o como monitores de temperatura en un área climatizada, entre otros usos.

La red de sensores se desarrolló con fines ilustrativos: solamente para mostrar un diseño que emplee al servidor web y haga uso de todas sus capacidades, y no como una aplicación robusta en sí. El capítulo comienza con una descripción general de la red, continúa con la descripción del protocolo que utilizan los sensores para comunicarse con el controlador central. Posteriormente se explica el software y hardware de los sensores y se presenta un análisis de la duración de las baterías en éstos. Finalmente se describe el hardware y software del controlador central, así como la manera en la que se comunica con el servidor web.

5.1. Descripción general

La red de sensores la forman cuatro sensores inalámbricos y un controlador central. El controlador central es el dispositivo que hace uso del servidor web para conectarse a una red ethernet y transmitir los datos de los sensores. La figura 5.1 muestra un esquema general de la red desarrollada.

Los sensores transmiten una muestra por segundo de la temperatura en una superficie al controlador central; el cuál, recopila las muestras y envía la información a los usuarios de la red con la ayuda del servidor web.

Los usuarios se conectan al controlador desde una red ethernet o incluso desde internet y descargan una applet de java que les permite capturar las mediciones de los sensores. El applet es la interfaz de usuario de la red, en el capítulo A se muestra la forma de utilizar el applet, mientras que el diseño de la misma se describe en el apéndice F.

Los sensores utilizan un transductor en un chip para medir la temperatura en el rango de -50 a 150 ^{o}C. Éste se encuentra montado en una tablilla que puede atornillarse a la superficie de interés.

Los sensores se comunican con el controlador utilizando un enlace de RF en la frecuencia de 2.4465 GHz. El alcance del enlace es de 30 metros en exteriores y 10 metros en interiores.

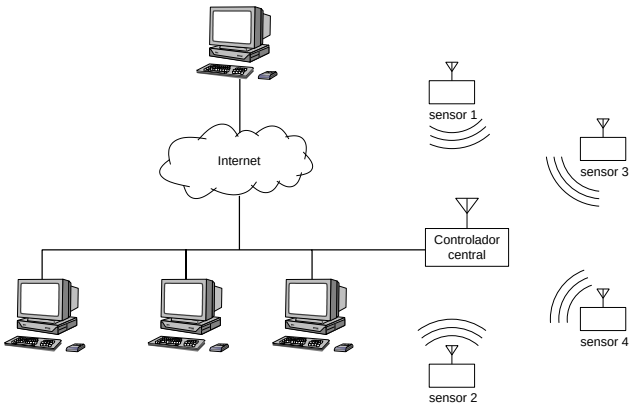


Figura 5.1: Red de sensores inalámbricos conectada a internet

Para evitar que los sensores transmitan al mismo tiempo y se produzca la pérdida de datos por colisiones, el controlador emite cada segundo una señal de sincronización que los sensores utilizan como punto de referencia para iniciar la transmisión.

Cuando los sensores detectan la señal de sincronización comienzan a transmitir en orden: el sensor 1 comienza a transmitir inmediatamente después de detectar la señal de sincronización, el sensor 2 lo hace a los 62.5 ms, el sensor 3 a los 125 ms y así sucesivamente. De esta manera cada sensor tiene asignado un espacio de tiempo de 62.5 ms en el que puede comunicarse con el controlador. Durante este tiempo el sensor transmitirá la muestra y esperará un acuse de recibo de la misma. Si la transmisión falla el sensor tiene tiempo de realizar 3 intentos más antes de descartar la muestra. Lo anterior se ilustra en la figura 5.2. Debido a que cada sensor transmite solo una vez por segundo, la red puede tener hasta 16 sensores activos.

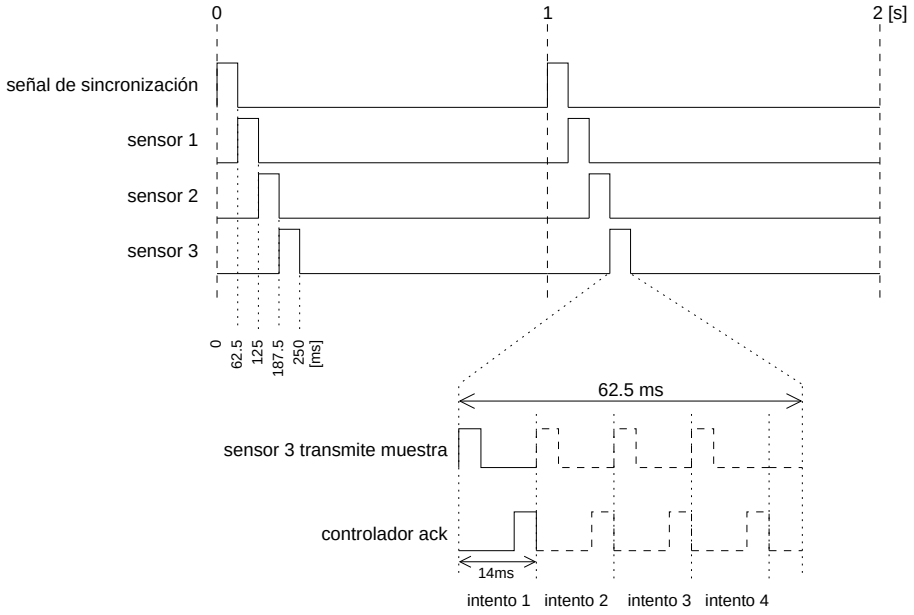
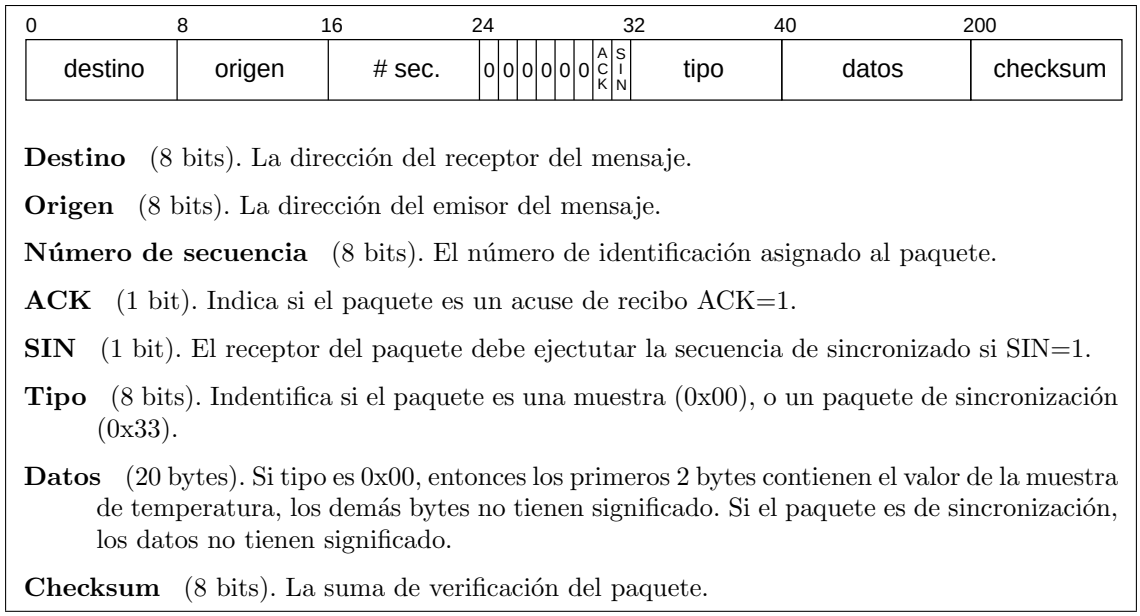


Figura 5.2: Espacios de tiempo asignados para transmisión



Destino (8 bits). La dirección del receptor del mensaje.

Origen (8 bits). La dirección del emisor del mensaje.

Número de secuencia (8 bits). El número de identificación asignado al paquete.

ACK (1 bit). Indica si el paquete es un acuse de recibo ACK=1.

SIN (1 bit). El receptor del paquete debe ejectutar la secuencia de sincronizado si SIN=1.

Tipo (8 bits). Indentifica si el paquete es una muestra (0x00), o un paquete de sincronización (0x33).

Datos (20 bytes). Si tipo es 0x00, entonces los primeros 2 bytes contienen el valor de la muestra de temperatura, los demás bytes no tienen significado. Si el paquete es de sincronización, los datos no tienen significado.

Checksum (8 bits). La suma de verificación del paquete.

Figura 5.3: Formato de los paquetes del protocolo RSI

5.2. Protocolo de comunicación RSI

Para lograr la comunicación de los sensores con el controlador central, se diseñó el protocolo de la red de sensores inalámbricos (RSI), el cuál es un protocolo simple basado en acuses de recibo: por cada paquete de datos recibido se genera un acuse de recibo. Los paquetes del protocolo tienen el formato mostrado en la figura 5.3.

Cada uno de los elementos en la red tiene asignada una dirección de 8 bits. La dirección 0xFF está reservada para *broadcast* y sirve para enviar un paquete a todos los elementos de la red.

Todos los paquetes incluyen un número de secuencia que sirve para asociar un paquete de datos con su acuse de recibo. Cada paquete de datos tiene un número de secuencia distinto: se incrementa en 1 por cada paquete transmitido.

Los paquetes pueden ser de 3 tipos: de sincronización, de datos o de acuse de recibo. Cuando un paquete es de sincronización su tipo es 0xAA y su dirección de destino es la dirección broadcast. Los paquetes de sincronización solo los envía el controlador central y sirven para que los sensores se sincronicen y transmitan en orden. Si el paquete es de datos su tipo es 0x10 y puede estar dirigido a cualquier elemento de la red. El acuse de recibo se forma colocando el bit de ACK en 1 e incluyendo los mismos datos del mensaje original.

Finalmente cada paquete incluye un checksum para verificar la transmisión correcta de los datos. El cheksum es el complemento a uno del byte menos significativo de la suma de todos los bytes del paquete. Para calcularlo, el byte de checksum se toma como 0x00.

El protocolo RSI se diseñó para permitir la operación autónoma de los sensores, éstos transmiten las mediciones constantemente mientras estén prendidos. Dado que los sensores funcionan con baterías, éstos entran en un modo de ahorro de energía después de transmitir su medición para prolongar la vida útil de las mismas.

En el modo de ahorro de energía los sensores tienen apagado el radio transmisor y no es posible comunicarse con ellos. Cuando los sensores están activos la única forma en la que el controlador central se comunica con ellos es mediante los paquetes de acuse de recibido.

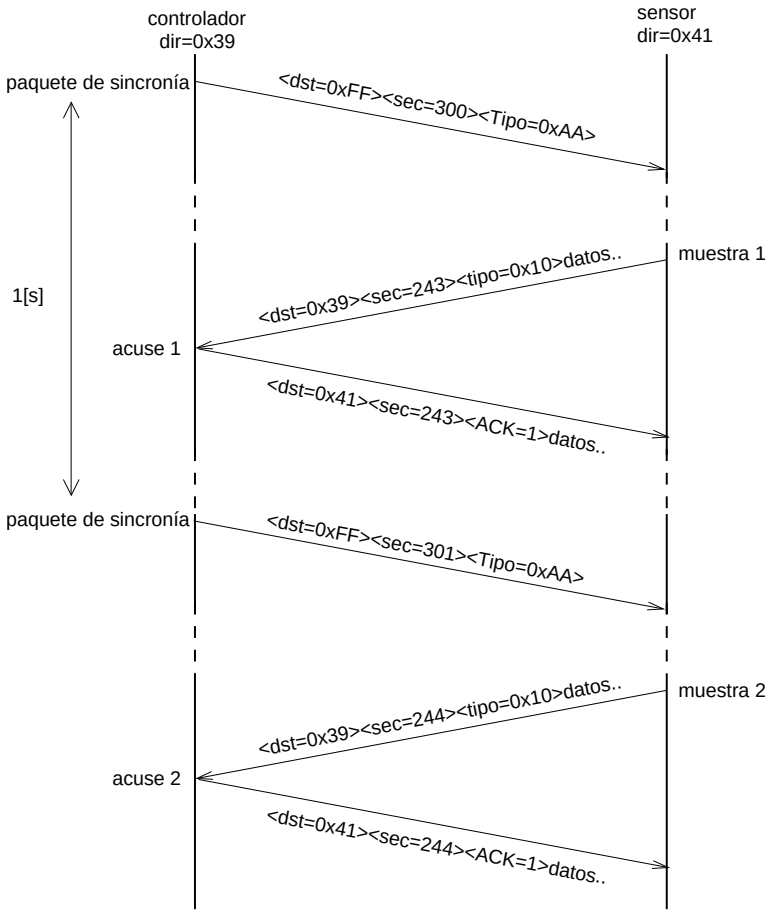


Figura 5.4: Comunicación tipica entre un sensor y el controlador central

Esta comunicación es limitada y solo permite al controlador enviar a los sensores la orden de resincronizarse con él.

La figura 5.4, muestra como se lleva a cabo la comunicación típica entre un sensor y el controlador central. El primer paquete que envía el controlador es un paquete de sincronía. Un tiempo después, el sensor envía la primera muestra de temperatura, y el controlador le responde con un acuse. Un segundo después, la comunicación se repite de la misma manera y continuará así mientras el sensor esté prendido.

5.2.1. Secuencia de sincronización

Un sensor debe sincronizarse con el controlador central antes de comenzar a transmitir las muestras de temperatura; para ello, el sensor espera a recibir el paquete de sincronización. Cada sensor puede hacer uso del canal inalámbrico durante 62.5msy es en ese tiempo en el que puede comunicarse con el controlador. Cuando el sensor detecta el paquete de sincronía determina, según su dirección, el momento en el que puede comenzar a transmitir.

La dirección de cada sensor está asignada de manera que los 4 bits menos significativos de la misma determinan el comienzo de la transmisión. De esta manera, en un segundo 16 sensores se pueden comunicar con el control central. El tiempo exacto en que cada sensor comienza a transmitir se determina con la siguiente ecuación:

$$Tinicio = (0x0F \ \& \ (dirección)) * 62.5[ms] \tag{5.1}$$

P. ej. con la dirección 0x40 un sensor comienza a transmitir justo después de recibir el paquete de sincronización, con 0x41 comienza a los 62.5ms después, con 0x42 a los 125 ms y así sucesivamente hasta 0x4E que transmite a los 937.5 ms.

Un sensor se sincroniza al comienzo de su ejecución; cada 2 minutos después de su última sincronización; cuando no fue posible transmitir una muestra, o a petición del controlador central.

Cuando el controlador central inicia su ejecución, solicita a cada sensor que se resincronize. Para ello, coloca el bit SIN en 1 en el siguiente paquete que envía al sensor (típicamente en un acuse de recibo).

5.3. Sensor de temperatura

Los sensores están diseñados en torno al microcontrolador PIC16F819 en su versión DIP de 18 pines; cuenta con un convertidor analógico-digital de 10 bits, 3584 bytes de memoria flash, 256 bytes de RAM, oscilador interno y opera en el rango de 2 a 5 Volts, entre otras características. El microcontrolador se encarga de realizar la medición de la temperatura y transmitirla al controlador central.

Todos los sensores tienen un hardware y software idéntico. La única diferencia entre ellos es la dirección que tiene cada uno asignado. En la figura 5.5 se muestra el diagrama esquemático del sensor[1] en el que se observan las partes que lo conforman.

El microcontrolador utiliza dos señales de reloj distintas: una generada por el oscilador interno (CLK_{int}), usada para la ejecución de instrucciones, y la otra generada por el cristal X1 (CLK_{x1}), utilizada para salir del modo de ahorro de energía. La señal CLK_{int} es de 4 Mhz, mientras que la señal CLK_{x1} es de 32768 Hz[2].

El microcontrolador entra al modo de ahorro de energía ejecutando la instrucción *sleep*. Durante este modo, el oscilador interno se detiene y el consumo de corriente desciende hasta 0.5μA. La generación de un *reset*, el disparo de una interrupción o el temporizador *watchdog*, ocasionan que el microcontrolador salga del modo de ahorro de energía.

El microcontrolador posee un contador libre de 16 bits (timer1) que se configuró para incrementarse libremente con cada pulso de CLK_{x1}, inclusive durante el modo de ahorro, y para generar una interrupción en cada *overflow* (cuando la cuenta del mismo pasa de 0xFFFF a 0x0000). El disparo de la interrupción ocasiona que el micro salga del modo de ahorro y continúe con la ejecución del programa, la interrupción se programa para dispararse la primera vez al tiempo obtenido con la ecuación 5.1 y posteriormente durante cada segundo.

5.3.1. Transmisor y receptor de RF

Los sensores y el controlador central se comunican mediante un enlace inalámbrico de radio frecuencia. Para ello utilizan el *transceiver* TRF-2.4G de Laipac Technology, el cuál engloba un radio transmisor y receptor programable que opera en la banda ISM de 2.4

[1] En el apéndice E se incluyen los diagramas esquemáticos, las mascaras PCB y la lista de materiales del sensor.

[2] La frecuencia del cristal X1 se seleccionó para generar una espera exacta múltiplo de 62.5 ms.

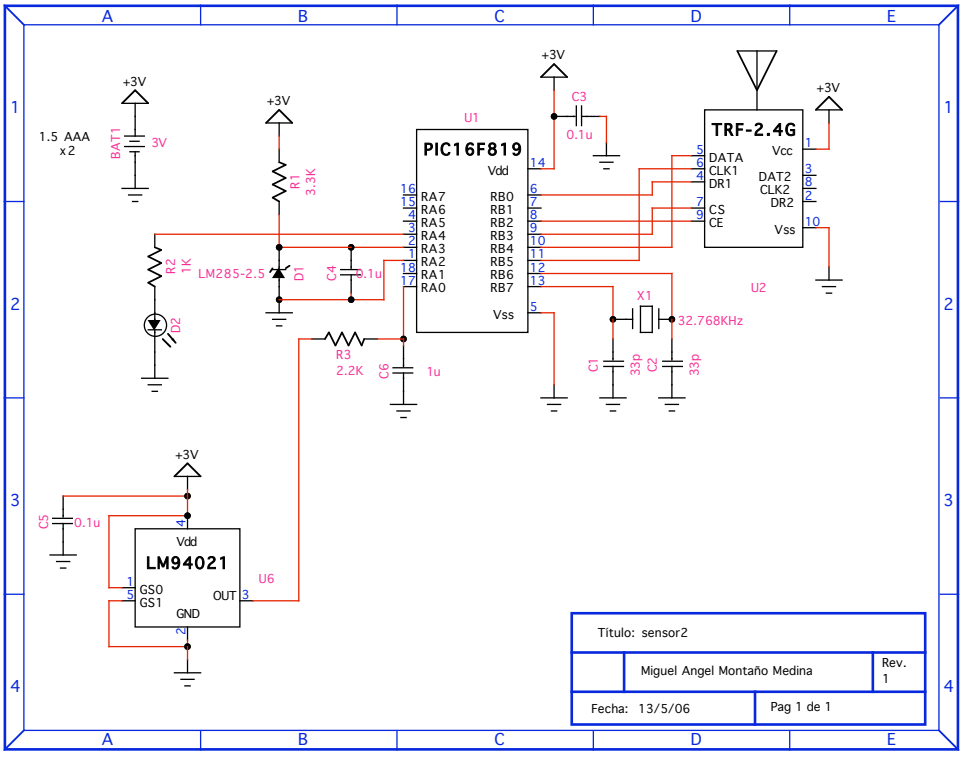


Figura 5.5: Diagrama esquemático de los sensores

GHZ, y permite una comunicación *half-duplex* entre los elementos de la red. El transceiver incorpora una antena en PCB y está diseñado en torno al chip NRF2401 de Nordic Semiconductor.

La frecuencia específica para la transmisión y recepción es programable, así como el modo de operación (ráfaga o directo) y la función deseada: recepción o transmisión.

Modos de operación

En el modo de operación directo, el transceiver opera de manera casi transparente al microcontrolador: en el mejor de los casos es como si hubiese una conexión serie cableada entre dos microcontroladores. Mediante el uso de una interfaz serie similar al SPI, el microcontrolador envía datos al transceiver para que los transmita por el canal inalámbrico. La velocidad de transmisión puede ser 250 Kbps o 1 Mbps, es fijada por el microcontrolador, y corresponde a la velocidad con la que éste carga datos al transceiver: si el micro carga datos en el transceiver a 250 Kpbs, éste los transmitirá a 250Kbps. De igual manera en la recepción de datos el transceiver pasa los datos al microcontrolador a la misma velocidad con la que los recibe.

En el modo directo el microcontrolador debe encargarse además de la verificación de errores en la recepción y el direccionamiento de los datos. Esto provee al microcontrolador con la libertad de emplear técnicas sofisticadas para la transmisión de datos, pero a la vez provoca que el software del micro se vuelva más complejo.

Por otra parte, en el modo de operación por ráfaga el transceiver ofrece al microcontro-

lador una capa de acceso al medio que facilita la transmisión y recepción de datos. La capa incluye el uso de direcciones físicas de longitud programable y la generación y verificación de checksums adicionales a la información transmitida. No incluye el manejo de errores en la transmisión como: colisiones o ruido.

En el modo por ráfaga, el transceiver provee un bufer de memoria en donde el microcontrolador coloca los datos que desea transmitir. La velocidad de transmisión en este modo no depende de la velocidad de la carga de datos, ahora es configurable en el transceiver y puede ser 250Kbps o 1Mbps. Esto permite que el microcontrolador realice la carga de datos, a su ritmo. Cuando el micro termina con la carga de datos, el transceiver se encarga de transmitirlos a la velocidad programada.

Cuando el transceiver recibe un paquete lo mantiene en el bufer de memoria e informa al microcontrolador, mediante un pin de interrupción, que tiene un paquete pendiente por procesar. La descarga del paquete también se realiza a una velocidad conveniente para el micro.

Los sensores y el controlador central operan el transceiver en el modo por ráfaga ya que facilita el diseño del software y hardware de éstos.

Capa de acceso al medio

La capa de acceso al medio en el modo por ráfaga permite el uso de direcciones físicas simples: a cada paquete de datos se le antepone la dirección del destinatario antes de cargarlo en el transceiver. Posteriormente, éste se encarga de colocar un preámbulo al paquete y añadir un checksum de 16 bits al final del mismo. Para ello, el transceiver debe saber el tamaño de los paquetes que ha de transmitir y recibir, por lo cual, el tamaño del paquete debe programarse en el transceiver antes de comenzar la carga o recepción del mismo, para que este pueda calcular y verificar el checksum.

En la figura 5.6 se aprecia como un paquete de datos del protocolo RSI se adecua para la transmisión en el transceiver. Cuando el transceiver recibe un paquete exitosamente, el mismo es entregado al microcontrolador sin el preámbulo, la dirección física y el checksum, es decir solo los datos del paquete.

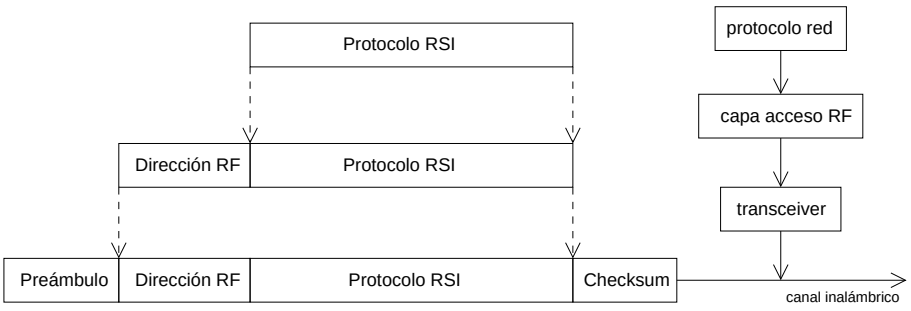


Figura 5.6: Encapsulamiento de un paquete de datos para transmisión con el transceiver

Las direcciones agregadas a los paquetes en la figura 5.6, son direcciones físicas y son distintas a las manejadas por el protocolo de la red inalámbrica. Haciendo una analogía con la pila TCP/IP, estas direcciones corresponderían a la dirección MAC y las direcciones utilizadas en el protocolo de la red de sensores serían como las direcciones IP.

La longitud de las direcciones físicas de los transceivers se puede programar para ser desde 1 a 5 bytes. En la recepción de un paquete en el modo ráfaga, el transceiver compara

la dirección física del paquete recibido con la propia y si corresponde le informa al microcontrolador de la recepción del mismo, de lo contrario el paquete se descarta silenciosamente. De esta manera todos los transceivers reciben el paquete, pero solo el que tiene la dirección correcta lo procesará.

En el protocolo RSI es necesario que un paquete sea recibido y procesado por todos los elementos de la red (el paquete de sincronización); pero con el esquema de direcciones físicas de los transceivers esto no es posible. Debido a esto se optó por asignar a todos los elementos de la red la misma dirección física de 4 bytes: 75-6e-61-6d y pasar al microcontrolador la responsabilidad de seleccionar los paquetes que debe procesar utilizando solo las direcciones del protocolo RSI.

Registro de configuración

El transceiver posee un registro de configuración de integrado por 15 bytes que se debe inicializar antes de la transmisión o recepción. Los parámetros más importantes del registro son:

- Número de bytes del paquete a transmitir.
- La dirección física del transceiver.
- La longitud de la dirección física.
- La generación de checksum de 8 o 16 bits.
- La velocidad de transmisión, 250Kbps o 1Mbps.
- El modo de operación, ráfaga o directo.
- La selección entre transmisión o recepción.
- El canal de frecuencia deseado entre 2400MHz y 2524MHz[3].

El registro de configuración se inicializa enviando serialmente una cadena de 15 bytes con los valores del registro. De igual manera para cambiar entre la recepción y la transmisión se cambia el bit correspondiente en el registro de configuración.

En este trabajo los transceivers transmiten a 250 Kbps en el modo de operación por ráfaga con la dirección física de 4 bytes 75-6e-61-6d, generación de checksums de 16 bits, paquetes de 26 bytes de longitud y el canal de 2492 MHz.

La cadena de configuración completa se encuentra en el archivo `nrf2401.c` y el formato de la misma se detalla en la hoja de especificación del transceiver [26] y del chip de NRF2401 [25].

Interfaz de software

Para utilizar el transceiver se diseñó una interfaz de software residente en el archivo `nrf2401.c` que provee las siguientes funciones:

- `set_tx_mode()`: Prepara el transceiver para una transmisión.

[3]En incrementos de 1MHz

- `set_rx_mode()`: Fija el transceiver en recepción, la recepción comienza al término de la ejecución de esta función
- `init_nrf()`: Envía la cadena de configuración de 15 bytes.
- `write_nrf(char *paq)`: Carga un paquete en el transceiver para transmisión. La transmisión ocurre al término de la ejecución de la función.
- `read_nrf(char *paq)`: Descarga el paquete recibido en el transceiver.

Para efectuar una transmisión se coloca el transceiver en el modo de transmisor empleando la función `set_tx_mode` y luego se envía el paquete con la función `write_nrf`.

El micro pone al transceiver en el modo de receptor con la función `set_rx_mode` para que éste comience a escuchar por una transmisión en el aire. El transceiver informa al microcontrolador que un paquete se recibió y está listo para ser procesado disparando la interrupción externa del micro, con lo cual el micro puede utiliza la función `read_nrf` para leer el paquete. Ésto se explica más a detalle en la sección 5.3.4.

El formato completo de las funciones de la interfaz de software del transceiver se presentan en el apéndice C.4.

5.3.2. Transductor de temperatura

El transductor de temperatura del sensor es el circuito integrado LM94021 de National Instruments, figura 5.7. Este componente tiene un rango de operación de 1.5 a 5.5 Volts y un rango de medición de -50 a 150 °C. Provee un voltaje de salida lineal que es inversamente proporcional a la temperatura medida, cuya ganancia es seleccionable mediante los pines de control: GS0 y GS1.

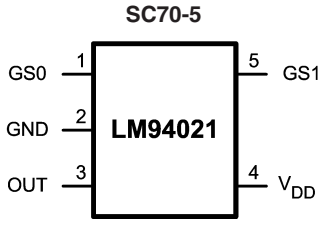


Figura 5.7: Transductor de temperatura LM94021

La salida del transductor es muy lineal y puede modelarse con una recta. La gráfica de la figura 5.8 muestra como se comporta el voltaje de salida respecto a la temperatura para las diferentes configuraciones de ganancia.

El transductor posee un excelente rechazo al ruido, y la especificación señala que en ambientes ruidosos puede ser necesario el uso de un capacitor bypass entre las terminales Vdd y GND lo más próximo al mismo para aumentar el rechazo al ruido. La especificación sugiere un capacitor de 1nF para obtener un rechazo de -51dB en el rango de 200 Hz a 1 MHz. Esto se consideró y se utilizó en el diseño final del sensor como se aprecia en el diagrama esquemático del sensor, figura 5.5.

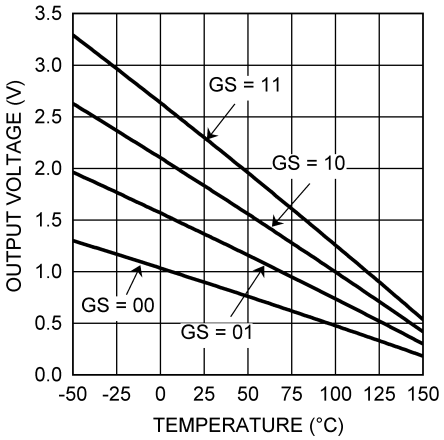


Figura 5.8: Voltaje de salida vs temperatura

5.3.3. Conversión analógica-digital

El convertidor AD del microcontrolador es de 10 bits y 4 canales analógicos, el rango de voltaje que puede leer se establece fijando el valor de las referencias V_{ref-} y V_{ref+}. El nivel de las referencias se fija considerando el rango del voltaje que se quiera leer. Las referencias se pueden fijar en 0 y Vdd o en un voltaje externo aplicado a los pines 2 y 3 del microcontrolador, siempre y cuando $V_{ref+} \leq$Vdd y $V_{ref-} \geq 0$.

Los sensores funcionan con 2 baterías AAA que proveen un voltaje de alimentación de 3 Volts. Con el uso, el voltaje de las mismas disminuye hasta un nivel en el que ya no son útiles. En los sensores este nivel corresponde a V_{ref+}. Mientras el nivel de las baterías se mantenga mayor o igual a V_{ref+}, las mediciones hechas en el convertidor AD serán confiables.

El voltaje que se desea leer es el entregado por el transductor de temperatura. En la figura 5.8 se aprecia que seleccionando la ganancia del transductor GS=01, el voltaje de salida de éste tiene un rango aproximado de 0 a 2 V. Debido a ésto, las referencias de voltaje se establecieron en:

$$V_{ref-} = 0[\text{V}] \tag{5.2}$$

$$V_{ref+} = 2.525[\text{V}] \tag{5.3}$$

En la hoja de especificación del transductor [31] se incluye una tabla con los valores de la función de entrada-salida del mismo. Con esta información se obtiene la gráfica del comportamiento real del transductor para la ganancia GS=01, figura 5.9. En ésta se observa que la salida traza una parábola con cambios pequeños de pendiente en el rango de interés que se abre hacia abajo.

Con la finalidad de que las mediciones del convertidor AD reflejen lo más posible el comportamiento de la figura 5.9, se desarrolló mediante mínimos cuadrados la función $T(v)$ (ecuación 5.4) que relaciona el valor de la temperatura (T) respecto al valor del voltaje leído en el convertidor AD (v).

$$T(v) = \begin{cases} -0.352(v) + 229.277 & \text{para } 764 < v \leq 793 \\ -0.310(v) + 197.107 & \text{para } 635 < v \leq 764 \\ -0.302(v) + 192.034 & \text{para } 470 < v \leq 635 \\ -0.292(v) + 187.322 & \text{para } 299 < v \leq 470 \\ -0.283(v) + 184.518 & \text{para } 122 \leq v \leq 299 \end{cases} \tag{5.4}$$

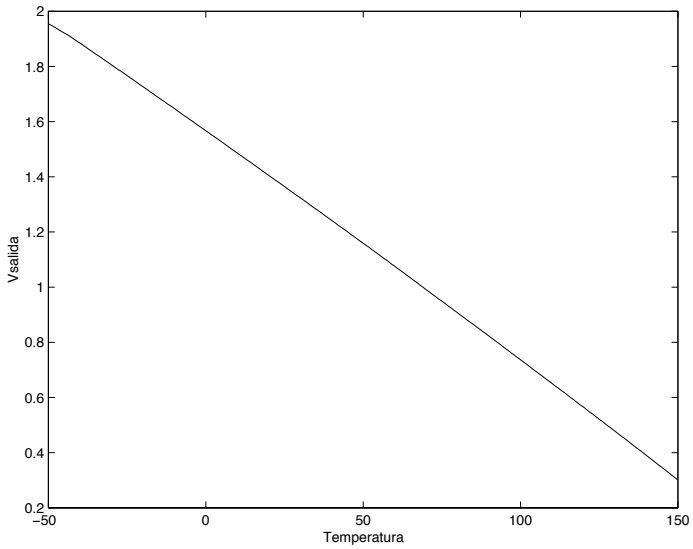


Figura 5.9: Comportamiento real del transductor para GS=01

La función $T(v)$ se compone de 5 modelos de rectas obtenidos a partir de mediciones experimentales y de la tabla de valores de la hoja de datos del transductor [31]. En la figura 5.10 se muestra la representación gráfica de esta función. Inicialmente $T(v)$ se componía de una sola recta, pero mediante pruebas experimentales se determinó que utilizando 5 rectas en lugar de una se obtenía un mejor ajuste, como se puede apreciar en las figuras 5.11 y 5.12.

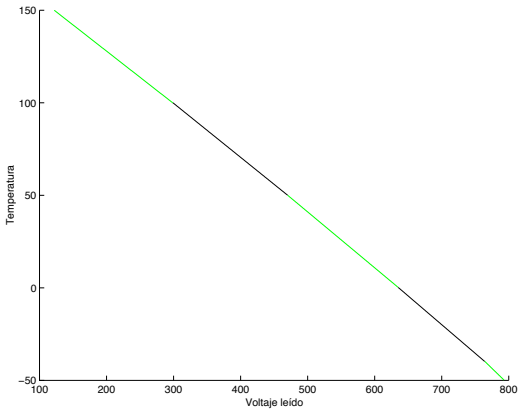


Figura 5.10: $T(v)$. Temperatura respecto al voltaje leído en el convertidor AD.

Dado que el convertidor AD es de 10 bits puede diferenciar entre 1024 niveles de voltaje diferente en el rango $[V_{ref-}, V_{ref+}]$, como este rango corresponde a [0,2.525] la resolución del convertidor es de 2.46 mV. Por otra parte el voltaje de salida del transductor es de 343 a 1955 mV que corresponde al rango de temperaturas de 150 a -50 °C. Cuando un sensor toma una muestra de temperatura, evalúa el voltaje leído en el convertidor AD en la función $T(v)$, y el resultado lo convierte en un número flotante de 16 bits que representa la lectura obtenida. El formato de éste número se muestra en la figura 5.13.

El formato consta de un bit de signo, la parte entera y la parte fraccionaria. La parte

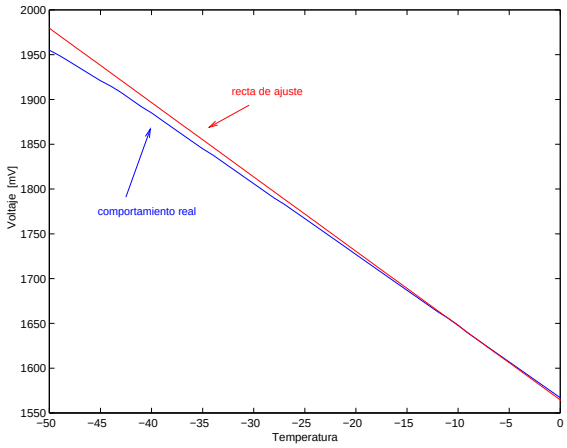


Figura 5.11: Ajuste mediante mínimos cuadrados, una recta.

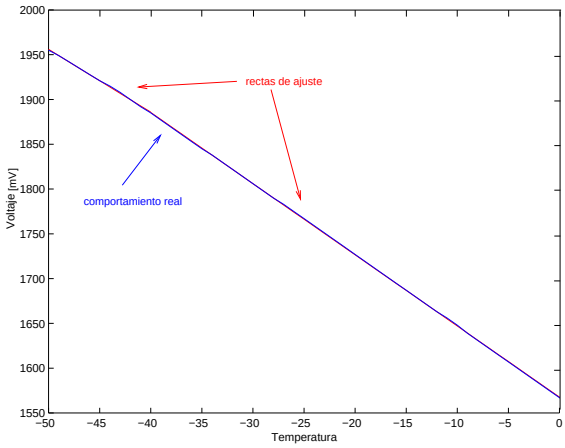


Figura 5.12: Ajuste mediante mínimos cuadrados, 5 rectas.

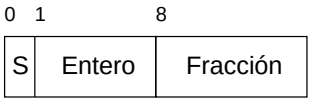


Figura 5.13: Formato de punto flotante de la muestra de temperatura.

entera es de 8 bits signada y puede representar un número entre -127 y 127. La parte fraccionaria representa 2 dígitos de precisión y se transmite como un número entre 0 y 99. Dado que el rango de medición de temperatura es de -50 a 150, la parte entera del valor obtenido al evaluar $T(v)$ se transmite como $T(v) - 50$, y la parte fraccionaria redondeada a dos dígitos de precisión, como se muestra en los ejemplos siguientes:

$$\begin{aligned} 0 &\rightarrow 0xCE00 \rightarrow 11001110\ 00000000 \\ 34.564 &\rightarrow 0xF038 \rightarrow 11110000\ 00100110 \\ -48.339 &\rightarrow 0x9E22 \rightarrow 10011110\ 00100010 \\ 145.960 &\rightarrow 0x5F60 \rightarrow 10011110\ 00100010 \end{aligned}$$

Este formato es de fácil despliegue y reconstrucción por software, y no está pensado para efectuar operaciones.

5.3.4. Software

El software del microcontrolador es sencillo y reside en dos archivos distintos: `sensorx.c` y `nrf2401.c`. El primer archivo contiene el programa principal y el segundo la interfaz de software desarrollada para el transceiver TRF-2.4G, las funciones de ésta se describen en el apéndice C.4.

En la figura 5.14 se muestra el diagrama de flujo del programa principal, en éste se aprecia lo que realiza el sensor antes y después de entrar en el modo de ahorro de energía. Lo anterior se representa mediante el ingreso a *dormir* y *despertar*. El microcontrolador "despierta" cuando se presenta el overflow en el timer 1, lo que ocasiona se ejecute la interrupción asociada `timer_overflow`.

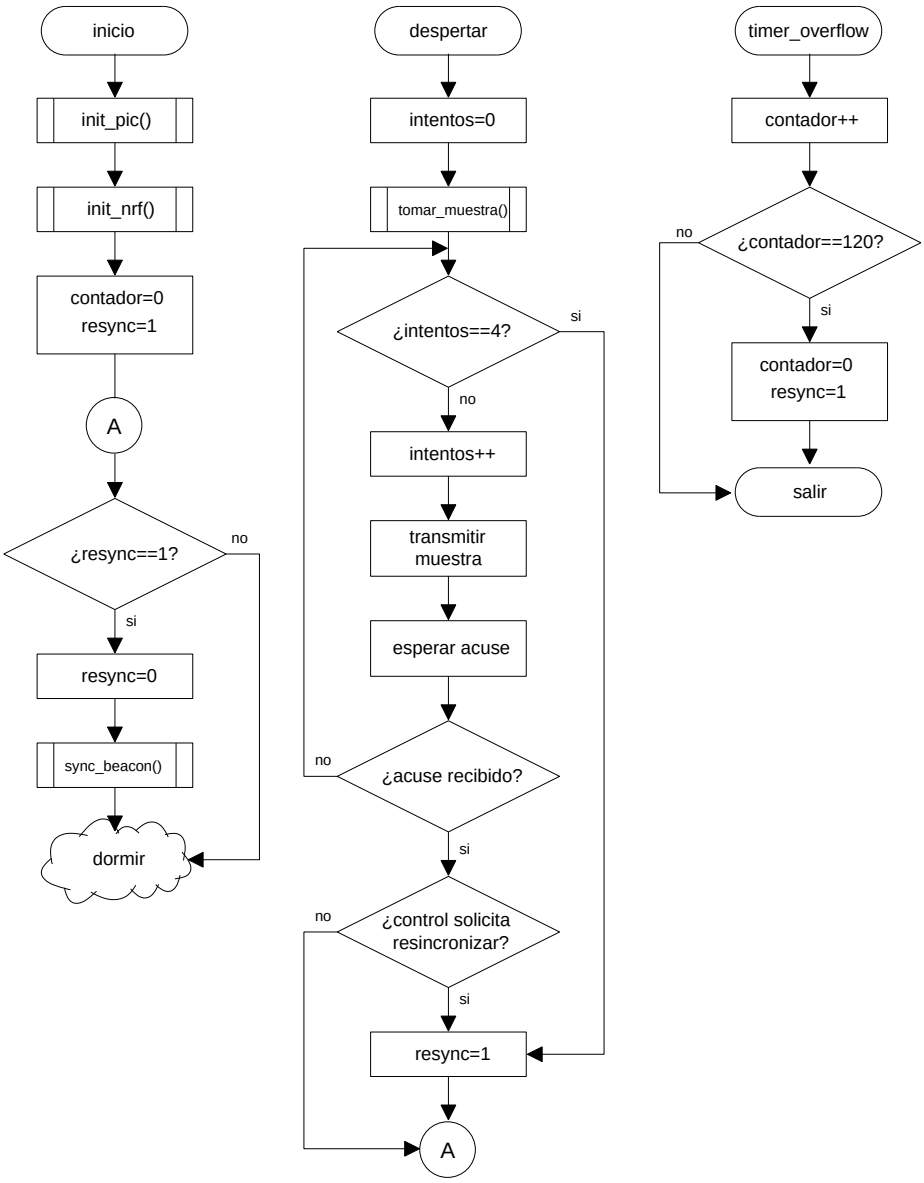


Figura 5.14: Diagrama de flujo del sensor

5.3.5. Consumo de energía

El consumo promedio de corriente del sensor es un parámetro que es importante conocer, ya que con éste se puede hacer el cálculo aproximado de la duración de las baterías.

Los componentes del sensor se seleccionaron tomando en cuenta su consumo de energía. Por ejemplo el microcontrolador PIC16F819 consume 530 μA en el modo activo y apenas 0.2 μA en el modo de ahorro de energía, mientras que el transductor de temperatura requiere apenas 10 μA para operar. En constraste, el transceiver es el componente que más corriente consume.

El consumo del transceiver es variable y depende de la operación que esté llevando a cabo:

- 12 [μA] en espera (stand by).
- 13 [mA] en una transmisión.
- 18[mA] en la recepción.

El uso del transceiver afecta directamente el consumo promedio del sensor más que cualquier otro componente, como se aprecia en la gráfica de consumo de corriente[4] del sensor, figura 5.15. Esta gráfica representa el consumo normal del sensor en el que el microcontrolador se despierta, realiza una medición, la transmite y recibe el acuse de recibo. En la gráfica, 6 volts corresponden a 20 mA de corriente.

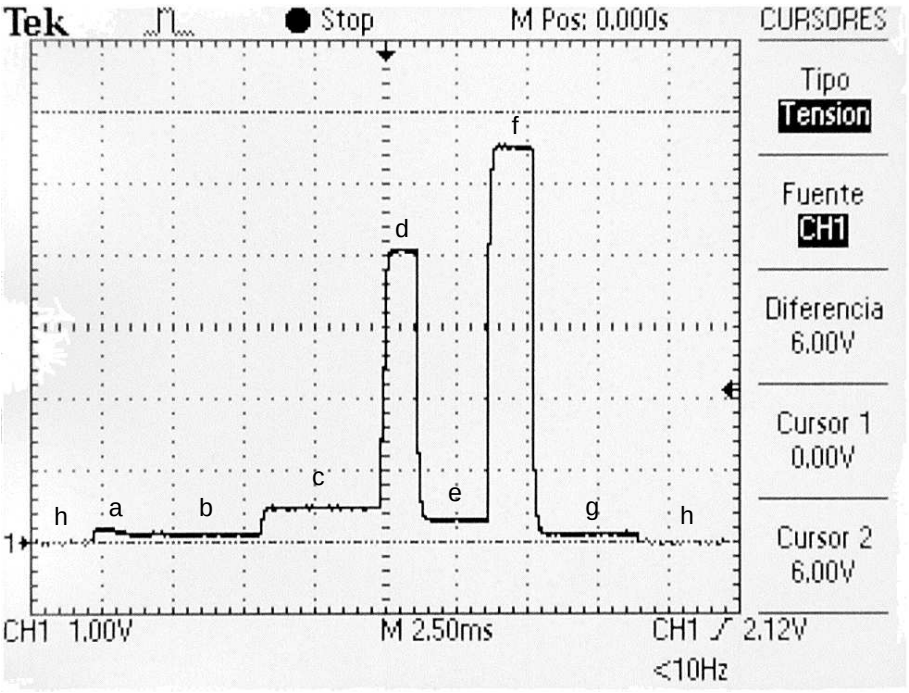


Figura 5.15: Consumo de corriente respecto al tiempo

En la gráfica de consumo se aprecian 8 secciones distintas etiquetadas de la *a* a la *h*. Las secciones *a*-*g* representan el modo activo del controlador, mientras que la sección *h*

[4]La gráfica de consumo se obtuvo con la ayuda de un circuito convertidor de corriente a voltaje

representa el modo de ahorro de energía (la duración completa de esta sección no aparece en la gráfica). Las secciones d y f representan la transmisión de la muestra, y la recepción del acuse de recibo respectivamente. El tamaño de cada una de las secciones de la gráfica se describe en el cuadro 5.1.

Sección	Nivel máximo [mA]	Duración [ms]
a	0.602	1.234
b	0.43	4.675
c	1.591	2.922
d	13.59	1.494
e	1.032	2.305
f	18.32	1.688
g	0.43	3.604
h	0.035	980.844

Cuadro 5.1: Tamaño de las secciones de la gráfica de consumo, figura 5.15.

Utilizando los datos del cuadro 5.1 se puede calcular el consumo promedio de corriente multiplicando la corriente utilizada en cada sección por su duración[5] y sumándolas todas. De esta manera, el consumo promedio es 96.86[μA]. Si se consideran las retransmisiones y las sincronizaciones en un escenario optimista, el consumo promedio se incrementa aproximadamente en un 5 %, con lo que se obtiene un consumo de 101.7 [μA]

Por otra parte cada sensor está alimentado con dos baterías alcalinas AAA de 1.5 Volts. Cada una de ellas tiene una capacidad típica de 1250 mAh, a un voltaje límite de 0.7V. Los sensores requieren un voltaje mínimo de operación de 2.5 Volts para su operación, por ello el voltaje límite al que cada batería puede llegar es 1.25 V. A éste límite las baterías solo permiten el acceso al 59 % de la capacidad de la batería, 735 mAh aproximadamente como se puede apreciar en la gráfica de descarga de una batería Energizer AAA, figura 5.16. Por lo tanto 2 baterías proveen 1470 mAh con lo cual la duración aproximada es de 14454 horas o 602 días.

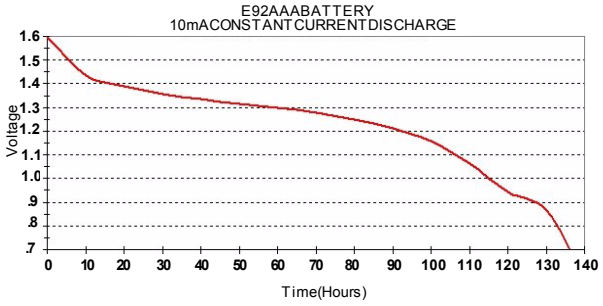


Figura 5.16: Ejemplo de la curva de descarga de una batería AAA

Finalmente, cuando el voltaje de las baterías ya no es suficiente para operar el sensor, éste informa al usuario con un parpadeo constante del led de estado.

[5]La duración de las secciones se calculó en base a un ciclo de trabajo de 1 segundo: el sensor realiza la misma tarea segundo a segundo.

5.4. Controlador central

El controlador central, al igual que los sensores, está diseñado en torno al PIC16F819. El diseño es similar en algunos aspectos al de los sensores, la diferencia principal es que el controlador central se alimenta con 5 Volts[6] y cuenta con líneas de control y comunicación con el servidor web.

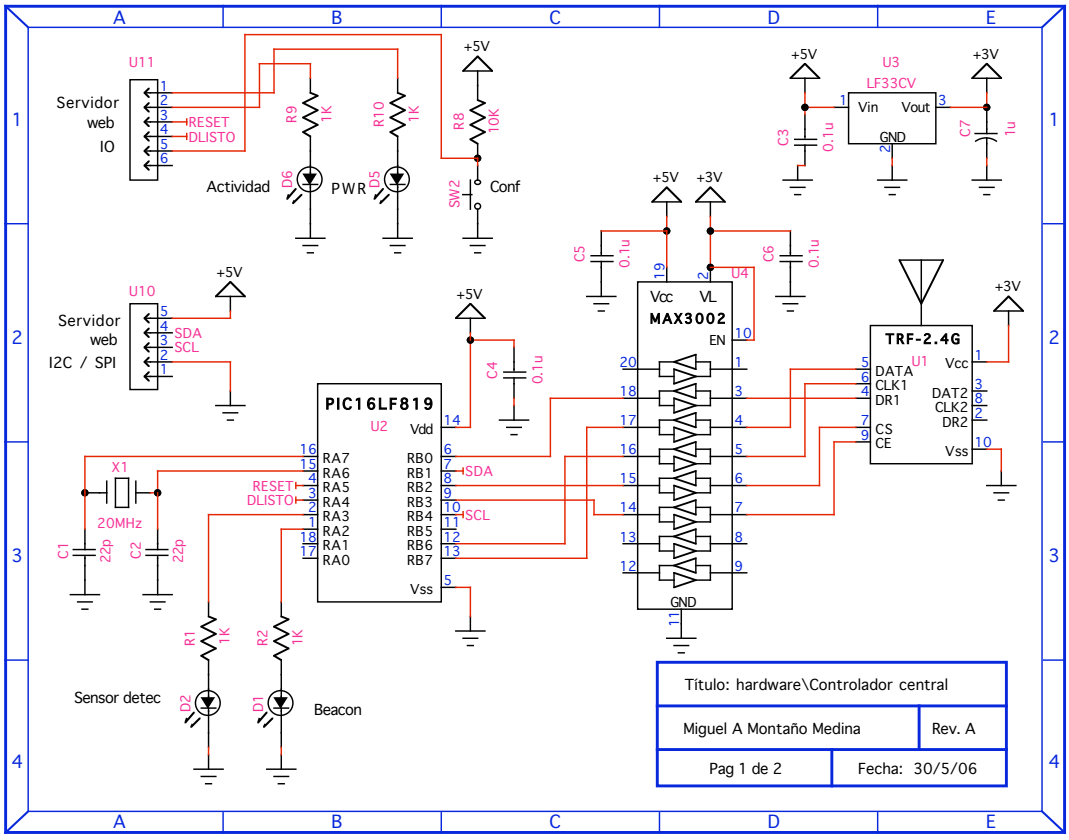


Figura 5.17: Diagrama esquemático del controlador central

En la figura 5.17, se muestra el diagrama esquemático del controlador central[7]. El transceiver de radio frecuencia es el mismo que se utilizó en los sensores. Como éste solo opera en el rango de 2.6 a 3.3 Volts, fue necesaria la inclusión de un regulador de voltaje de 3 Volts para su alimentación: el LF33CV. Dado que el microcontrolador opera con 5 Volts y el transceiver a 3 Volts se utilizó el convertidor de nivel lógico MAX3002 para acoplar los niveles de las señales. Éste es un convertidor de 8 líneas bidireccional que no requiere una lógica de control para establecer la dirección de la conversión.

El microcontrolador opera a 20 MHz, a diferencia de los sensores en los que opera a 4 Mhz. La señal de reloj se genera con la ayuda del cristal X1, es importante que el microcontrolador opere a una velocidad mayor que en los sensores, ya que el controlador

[6]El hardware del servidor web provee al controlador la alimentación de 5 volts.

[7]En el apéndice E se incluyen los diagramas esquemáticos, las mascaras PCB y la lista de materiales del prototipo del controlador central.

además de recopilar datos de los sensores y mantener la red sincronizada, debe también comunicarse con el servidor web.

La comunicación con el servidor web se hace mediante el bus I2C[8]: cuando el controlador ha recopilado las muestras de todos los sensores presentes, le informa al servidor web de la situación colocando la línea DLISTO en 1. El servidor web polea esta línea y cuando detecta que está en uno efectúa la lectura de datos del controlador central para obtener las muestras de los sensores. Las líneas de comunicación que utiliza el controlador central para con el servidor web se listan a continuación:

- DLISTO (salida): Le indica al servidor web que la recopilación de muestras de temperatura está lista.
- SDA (bidireccional): Transmisión y recepción de datos mediante el bus I2C.
- SCL (entrada): Señal de reloj para la transmisión serial en el bus I2C.
- RESET (entrada): Reinicia el controlador central.

Finalmente, el controlador cuenta con dos leds de estado: SINC (D1) y RX (D2), conectados a los pines PTA2 y PTA3 del microcontrolador respectivamente. El primero parpadea cada vez que el controlador emite el paquete de sincronización, y el segundo cada vez que se recibe exitosamente la transmisión de datos de un sensor.

El controlador central almacena las muestras de temperatura en un bufer de memoria RAM e implementa la máquina de estados I2C de manera que cuando éste es direccionado en el bus y comienza una lectura de datos, los datos que se transmiten al bus son los de las muestras en el bufer.

El controlador central no tiene conocimiento alguno sobre los detalles de la conexión a la red, ya que el servidor web es el que se encarga de toda la comunicación. En esta aplicación el controlador central solo tuvo que implementar la comunicación mediante el bus I2C y utilizar dos líneas de control del servidor web para integrarse con éste.

5.4.1. Software

El software del microcontrolador hace uso de tres interrupciones del microcontrolador: la interrupción externa, la del temporizador 1 (timer overflow) y la generada por el puerto MSSP (lectura del bus I2c).

Cuando el transceiver recibe un paquete de datos, establece en 1 la línea DR1. Ésta está conectada al pin de interrupción externa, lo que ocaciona el disparo de la interrupción `int_externa` en cada recepción exitosa. La rutina de interrupción se encarga de validar los paquetes y colocar los datos del mismo en el bufer de memoria `muestras`. Cuando todos los sensores han transmitido sus muestras, la rutina de interrupción establece la variable `nuevo=1`, lo que en el programa principal ocasionará que se informe al servidor web mediante la línea DLISTO (pin PTA4).

La interrupción del MSSP del microcontrolador se genera cuando el servidor web direcciona al controlador en el bus I2C y efectúa la lectura de datos. La interrupción se dispara en repetidas ocasiones, ya que la comunicación mediante I2C atraviesa por varios estados.

[8]El bus I2C funciona siguiendo el modelo maestro-esclavo, y en éste caso el servidor web es el maestro y todos los demás dispositivos conectados al bus son esclavos.

Los estados de ésta se describen en la nota de aplicación AN734 [27]. En cada lectura, la máquina de estados I2C entrega los datos almacenados en el bufer `muestras`.

El controlador emite el paquete de sincronización una vez por segundo y para ello hace uso del contador timer1 del microcontrolador para generar la interrupción `timer_overflow` en el tiempo adecuado y así transmitir el paquete en el tiempo deseado.

El software del controlador central reside en el archivo `controlador.c` y se encuentra en el disco compacto anexo a esta tesis. En la figura 5.18 se presenta el diagrama de flujo del software del controlador.

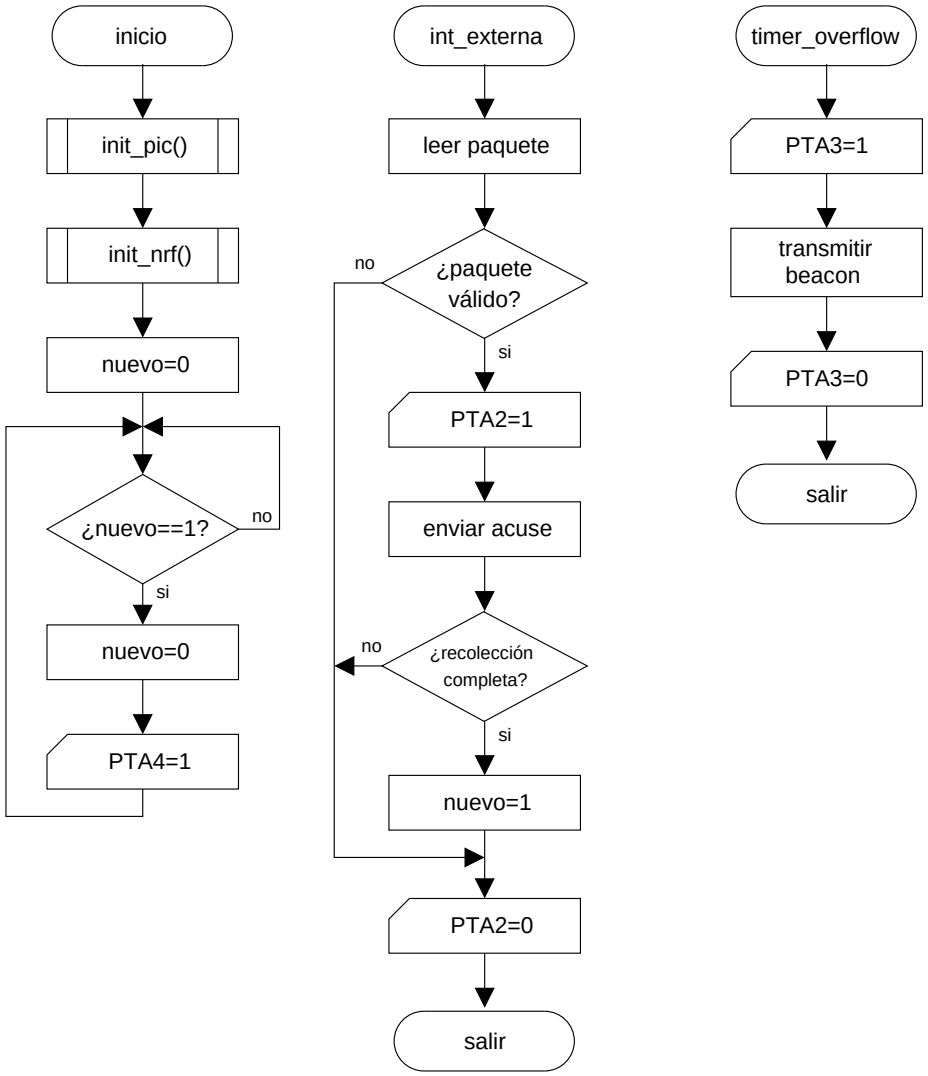


Figura 5.18: Diagrama de flujo del controlador

Capítulo 6

Conclusiones

El objetivo de esta tesis fue crear un servidor web, como un componente de hardware, que pudiera utilizarse en nuevos diseños que requirieran la funcionalidad de red. El desarrollo del servidor web comprendió la realización del siguiente trabajo:

- Se diseñó la pila uinet, siendo una implementación de la pila de protocolos TCP/IP, para usarse en sistemas embebidos basados en microcontrolador. La pila uinet se creó desde cero e incluyó una implementación de los protocolos IP, TCP, UDP, ICMP e IGMP que se llevó a cabo considerando los requerimientos de comunicación descritos en el documento RFC1122.

- Se diseñó e implementó el servidor web Petit, basado en el protocolo HTTP/1.1, para funcionar con la pila uinet, y se creó un contenido web para el mismo que consta de páginas html, imágenes y programas al que se pudo acceder exitosamente desde un navegador web.

- Se diseñó un hardware que complementa a la tarjeta de desarrollo Easy Ethernet, utilizada para diseñar la pila uinet y el servidor Petit, y que en conjunto forman el bloque de hardware que constituye al servidor web.

Para probar el diseño en conjunto del servidor web, se desarrolló una red de sensores en la que el controlador central de la misma utilizó exitosamente al hardware del servidor web para conectarse a una red ethernet, a la pila uinet para enviar datos a través de internet y al servidor Petit como puente para acceder a la interfaz de usuario: un applet de Java.

Los requerimientos especificados en el RFC1122 se tomaron en cuenta al diseñar la pila, pero solo se implementó un subconjunto de los mismos ya que algunos requerimientos, como la implementación del puntero urgente en TCP, no eran necesarios para el funcionamiento del servidor Petit; mientras que otros, como la implementación completa del mecanismo de la ventana de TCP y el algoritmo de ACK's retrasados, que mejorarían en gran medida el desempeño del servidor, no se implementaron debido a que la cantidad de información que el servidor Petit envía a la red es poca y no tendrían un impacto considerable. Las pruebas realizadas en las que el servidor web se conectó exitosamente a internet comprueban que la implementación de requerimientos fue exitosa.

La velocidad máxima de transferencia que se encontró en pruebas experimentales fue de 55.6 Kbps. Esta velocidad está limitada por varios factores: la implementación limitada de la ventana TCP en la pila uinet, la velocidad de procesamiento y el algoritmo de los ACK's retrasados en el receptor. Debido a que la pila uinet solo puede enviar un nuevo

segmento hasta recibir el acuse del segmento anterior, no puede aprovechar las ventajas de la implementación de los acks retrasados en el receptor, lo que ocasiona un efecto adverso ya que el receptor genera un retraso antes de enviar un ACK limitando con ello la velocidad de transferencia del servidor web.

La parte más laboriosa del desarrollo de este trabajo fue la depuración de errores y las pruebas de funcionamiento de la pila uinet y el servidor Petit, ya que cada corrección al código tomaba un tiempo considerable en realizarse, debido a que se debía reprogramar el microcontrolador de la tarjeta de desarrollo con el nuevo software y verificar que éste funcionara de la manera esperada.

6.1. Trabajos futuros

La pila uinet se diseñó para proveer a aplicaciones del tipo servidor las funciones necesarias para comunicarse en la red internet. Un trabajo interesante sería agregar al módulo TCP la funcionalidad necesaria para soportar aplicaciones tipo cliente, como por ejemplo un cliente SMTP que pueda enviar correos electrónicos.

Otro trabajo que se podría realizar sería modificar el código fuente de la pila uinet y recompilarlo para utilizarse en otro microcontrolador distinto al utilizado en esta tesis. Como la programación de la misma se hizo en lenguaje C, con muy pocos ajustes se podría llevar a cabo esta tarea.

En cuanto al servidor web el paso siguiente sería diseñar un nuevo prototipo que incluya el hardware de la tarjeta easy ethernet junto con el hardware complementario realizado en esta tesis, en una tarjeta más pequeña.

También se podría aumentar el espacio de almacenamiento del servidor web, cambiando la memoria EEPROM por una tarjeta de memoria flash comercial (SD, XD, MMC, etc.) e implementar en el servidor el software necesario para acceder al sistema de archivos utilizado en la tarjeta por otros fabricantes, de manera que desde una PC se pueda copiar directamente el contenido web en la tarjeta sin la necesidad de emplear herramientas como el programa `makerom` de esta tesis.

Apéndice A

Guía rápida de operación de la red de sensores

En este capítulo se presenta una guía rápida para la instalación y configuración de la red de sensores desarrollada; así como una guía de configuración y solución de problemas que podrían presentarse en la operación de ésta.

A.1. Descripción de partes

Las partes que conforman a cada uno de los sensores de temperatura se aprecian en la figura A.1. La figura A.2 muestra las partes de la tarjeta Easy Ethernet; la cuál, está conectada al controlador central mediante dos cables planos, uno para el conector de E/S y otro para el conector I2C. El controlador central se muestra en la figura A.3.

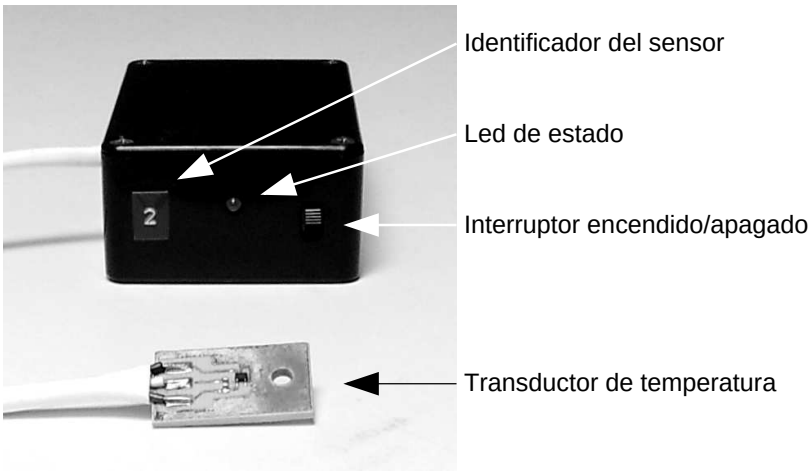


Figura A.1: Sensor de temperatura

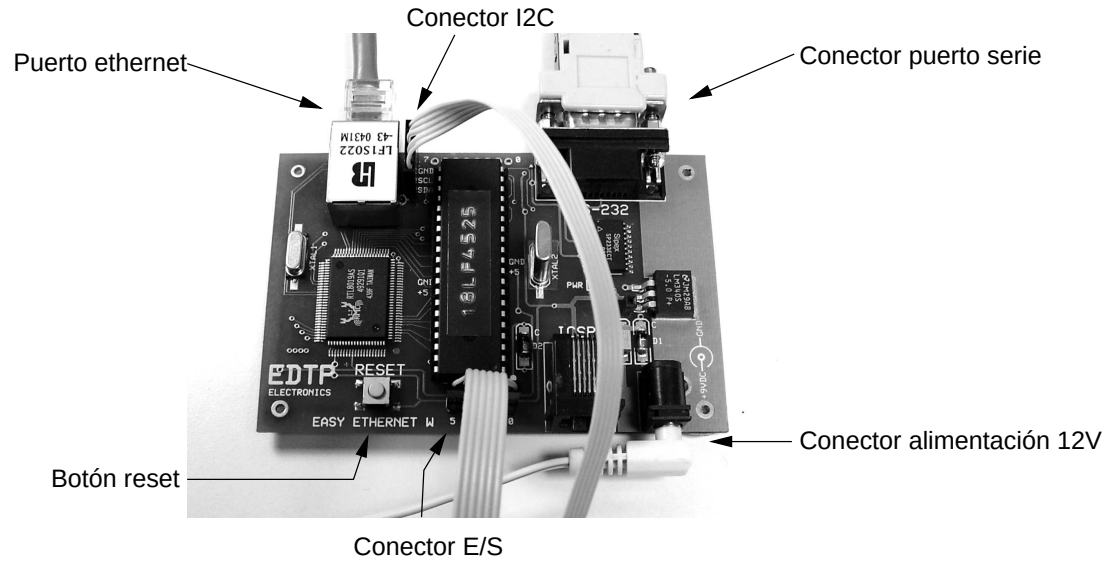


Figura A.2: Tarjeta Easy Ethernet

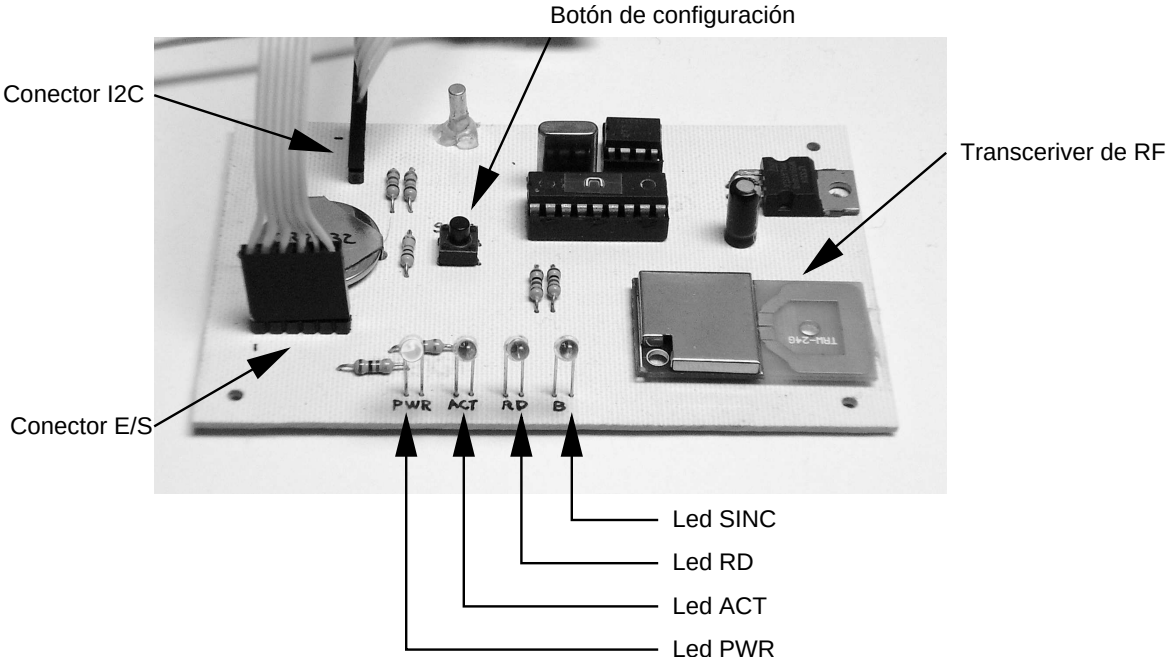


Figura A.3: Controlador central

A.2. Requerimientos mínimos de ejecución

La red de sensores incluye un software para su operación, el cual se encuentra almacenado en la memoria EEPROM del servidor web en el controlador central. El software es un Applet de Java y para poder ejecutarlo se requiere una computadora que cumpla con los siguientes requerimientos:

- Maquina virtual de Java versión 1.4.2 o superior.
- Java Plug-in versión 1.4.2 o superior.
- Navegador web compatible con HTTP/1.1. (ej: Internet explorer 6, Netscape Navigator 7.2, Firefox 1.0, Safari 1.3).

Los sistemas operativos en los que se ha probado exitosamente la applet son: Mac OS X 10.3.9, MS Windows XP SP2 y MS Windows Milenium, en otros sistemas la applet podría no funcionar de la forma esperada.

Adicionalmente para configurar el servidor web del controlador central se requiere:

- Programa emulador de terminal (ej: Hyperteminal, Tera Term, etc.)
- Un puerto serie que soporte 115200 bps.

A.3. Guía rápida de operación

A continuación se presentan los pasos necesarios para operar la red de sensores.

A.3.1. Instalación de los sensores

1. Colocar baterías 2 baterías alcalinas AAA en cada uno de los sensores a utilizar.
2. Instalar los sensores en el lugar de interés, procurando que el transductor haga contacto con la superficie de interés.
3. Ubicar el controlador central a una distancia no mayor a 10 metros de los sensores.
4. Conectar el controlador central a la red ethernet y conectar la alimentación del mismo (12 Vdc).
5. Verificar que el led PWR esté prendido y que el led SINC parpadee.
6. Prender los sensores y verificar que su led de estado parpadee después de unos segundos. El parpadeo del led, indica que el sensor logró la comunicación con el controlador central. Si el led no parpadea y se queda prendido, cambiar la orientación del sensor o colocar el controlador central más cerca de los sensores hasta que el led de estado parpadee.

A.3.2. Captura de datos

1. Desde una computadora[1] conectada a la misma red ethernet que el controlador central, abrir un navegador web e ingresar a la siguiente dirección:

   http://192.168.1.150

   Aparecerá la página mostrada en la figura A.4.

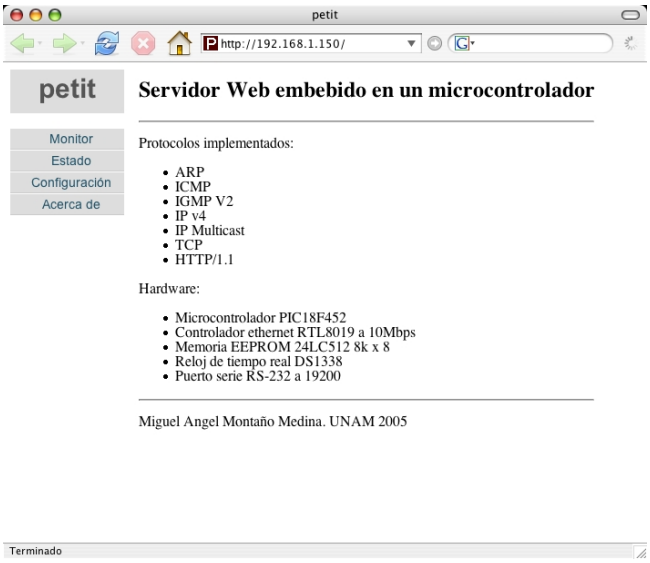


Figura A.4: Página principal

2. Seleccionar en el menú principal, la opción *Monitor*.

Figura A.5: Menú principal

3. A continuación aparecerá una ventana solicitando permiso para la ejecución del programa, figura A.6, en la cual se debe permitir la ejecución de la applet.

[1]Revisar que la computadora cumpla con los requerimientos mínimos, sección A.2

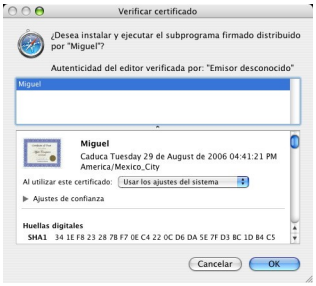


Figura A.6: Verificación del autor y permiso para ejecución del applet

4. Después de conceder el permiso de ejecución, aparecerá la ventana principal del sistema, figura A.7.

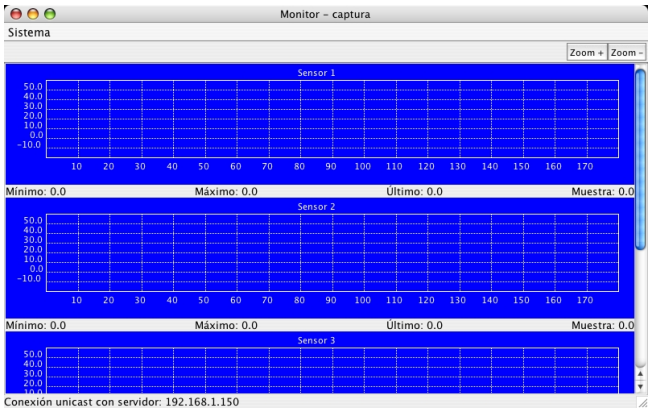


Figura A.7: Ventana principal del sistema de captura

5. Seleccionar en el menú *Sistema* la opción *Configurar conexión.*

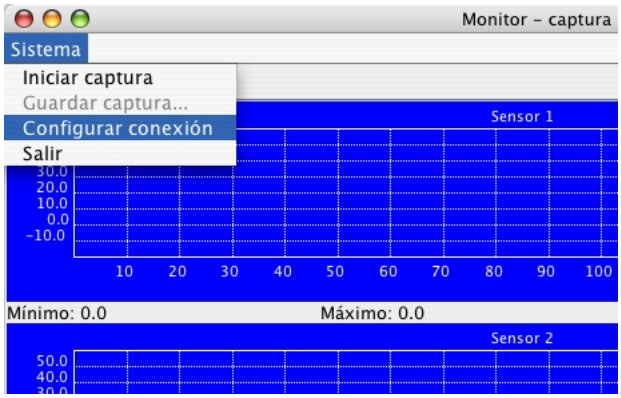


Figura A.8: Menú Sistema

6. Seleccionar de la lista de conexión en la figura A.9, la opción *Unicast* y hacer clic el botón Aceptar.

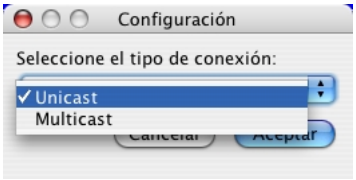


Figura A.9: Selección del tipo de conexión deseada

7. En la ventana principal del sistema seleccionar en el menú *Sistema* la opción *Iniciar captura*, con lo que comenzará la captura y el despliegue de las mediciones de cada sensor en la pantalla.

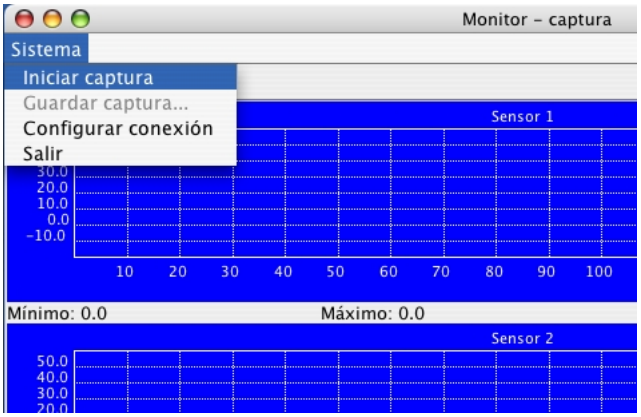


Figura A.10: Menú Sistema

A.3.3. Finalizar la captura de datos

Las mediciones capuradas se guardarán en un archivo al terminar la captura de datos. Para detener la captura efectuar los siguientes pasos:

1. Seleccione la opción *Detener Captura* en el menú *Sistema*.

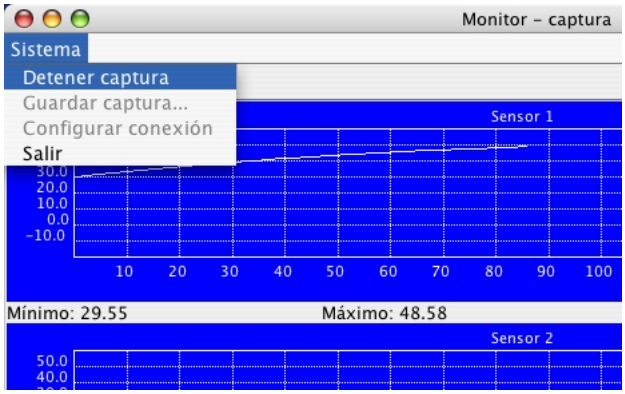


Figura A.11: Menú Sistema

2. Aparecerá un cuadro de dialogo similar a la figura A.12. Seleccione un directorio y un nombre para el archivo de muestras en el campo *Guardar como* y haga clic en el botón *Guardar*. Este archivo contendrá todas las muestras, en formato de texto, capturadas por el sistema durante el tiempo en que operó.

Figura A.12: Seleccionar un archivo para guardar los datos capturados

Nota: Las ventanas mostradas en esta sección corresponden a la ejecución de la applet en el navegador Firefox en el sistema operativo Mac OS X 10.3.9. Con otros navegadores y sistemas operativos, las ventanas pueden tener una apariencia distinta a la aquí presentada.

A.4. Modo de configuración

El controlador central tiene asignada por defecto la dirección IP 192.168.1.150, si esta dirección no es adecuada para la red en donde se comunicará, se puede cambiar ingresando al modo de configuración. Además de la dirección IP también se pueden realizar los siguientes ajustes:

- Cambiar el contenido web del servidor cargando una nueva imagen ROM en la memoria del servidor.
- Restablecer la configuración por defecto.
- Modificar la dirección IP.
- Modificar la dirección IP del grupo de multidifusión.
- Modificar el puerto TCP del servidor web.
- Modificar el puerto UDP para conexiones unicast.
- Modificar el puerto UDP para conexiones multicast.
- Modificar el valor del TTL para los paquetes de multidifusión.

Para ingresar al modo de configuración es necesario llevar a cabo los siguientes pasos:

1. Conectar el controlador central (conector hembra en la tarjeta easy ethernet) al puerto serie de una computadora utilizando un cable RS-232 de 9 pines, figura A.13

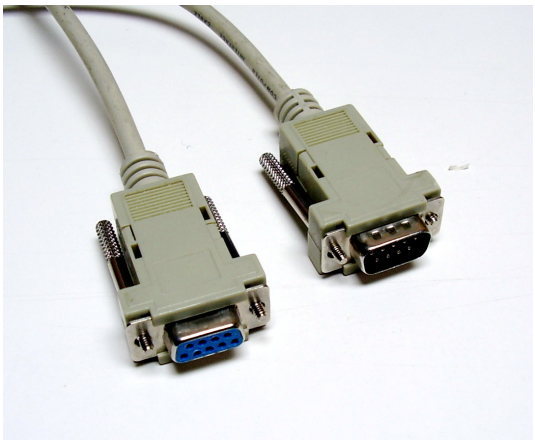


Figura A.13: Clabe serie de 9 pines macho-hembra

2. Configurar en la computadora el programa emulador de terminal Tera Term[2] para utilizar el puerto serie a 115200 bps, 8,N,1, sin control de flujo, figura A.14.

[2]También puede utilizarse el emulador HyperTerminal o cualquier otro emulador de terminal

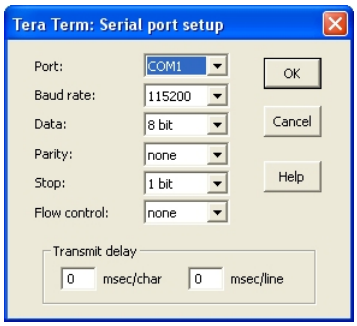


Figura A.14: Configuración del puerto serie

3. Conectar la alimentación de la tarjeta Easy Ethernet.
4. Presionar y mantener oprimido el botón RESET en la tarjeta Easy Ethernet.
5. Presionar y mantener oprimido el botón de configuración en el controlador central y soltar el botón de RESET.
6. Soltar el botón de configuración.
7. En la ventana del emulador de terminal aparecerá un mensaje como el mostrado en la figura A.15.

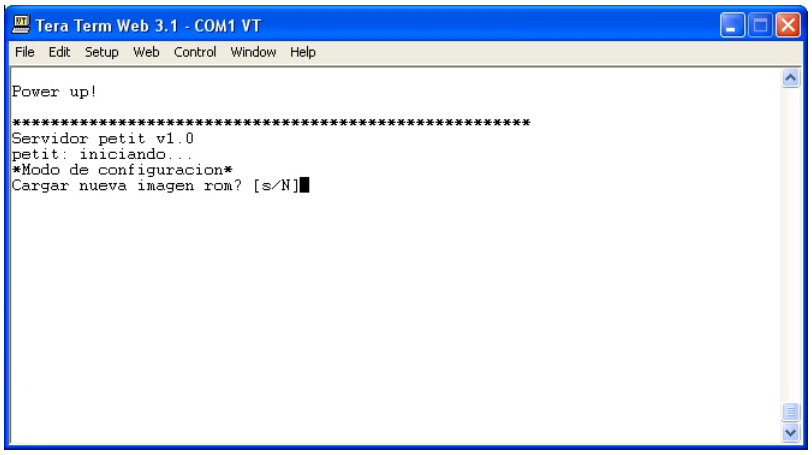


Figura A.15: Modo de configuración

La selección del ajuste deseado se lleva a cabo respondiendo a una serie de preguntas con si ('s') o no ('n'). En cada pregunta se sugiere la respuesta por defecto indicada en letras mayúsculas, por ejemplo en la pregunta: `Utilizar configuración por defecto?(s/N)`, el presionar la tecla enter es equivalente a responder no.

Cuando las preguntas del modo de configuración terminen el servidor iniciará su ejecución utilizando los nuevos ajustes.

A.5. Creación de una imagen ROM

Las páginas html, imágenes y demás archivos del servidor web en el controlador central se almacenan en la memoria (EEPROM) del mismo. La utilería `makerom` es un programa en java que genera el archivo `imagen.rom` con el contenido web que será cargado en el servidor.

La utilería genera la imagen a partir de un directorio llamado `html`. En este directorio deben existir como mínimo los archivos `index.html`, `404.html` y `505.html`. El archivo `index.html` es la página de inicio del servidor, mientras que los archivos `404.html` y `505.html` contienen, respectivamente, las páginas que el servidor mostrará cuando un archivo no se encuentre en el servidor y cuando un navegador invoque un comando HTTP no soportado.

En el directorio `html` no deben existir subdirectorios y se debe tomar en cuenta al crear los archivos del servidor que el tamaño de la memoria de éste es de 65536 bytes.

Los pasos para generar una imagen ROM con la utilería `makerom` son:

1. Dependiendo el sistema operativo del usuario abrir una ventana de comandos (un shell en unix o la ventana de comandos msdos en Windows).

2. Crear un subdirectorio llamado `html`.

3. Copiar en el subdirectorio `html` todos los archivos que se cargarán en el servidor web.

4. Ejecutar el siguiente comando:

```
java -jar makerom
```

El programa generará el archivo `imagen.rom` en el directorio de trabajo actual y desplegará el nombre de los archivos que contiene así como el tamaño y la capacidad utilizada de la memoria del servidor, figura A.16

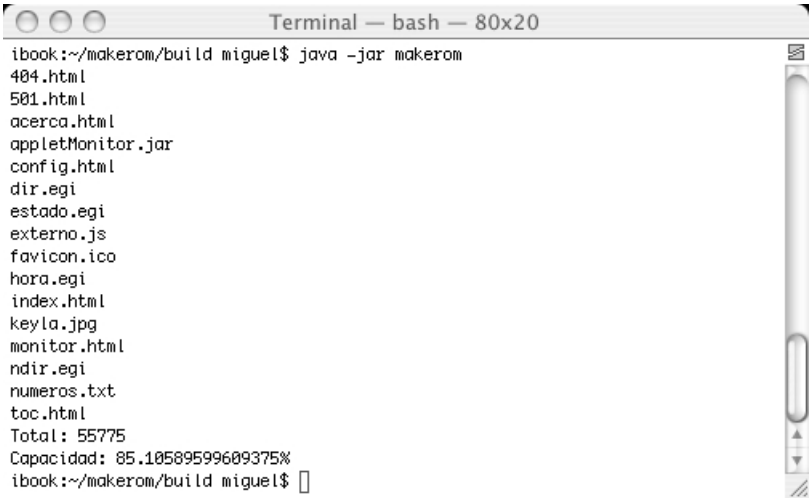


Figura A.16: Ejemplo de ejecución de la utilería `makerom`

El archivo `imagen.rom` se carga en la memoria del servidor como se explica en la sección A.6.

A.6. Carga de una imagen ROM

Para cargar una imagen ROM en la memoria del servidor web en el controlador central, se debe utilizar un programa emulador de terminal que soporte el protocolo de transferencias XMODEM y efectuar los siguientes pasos:

1. Ingresar al modo de configuración.

2. Responder con 's' a la pregunta: `Cargar nueva imagen rom?(s/N)`

3. En el emulador de terminal seleccionar la transferencia mediante XMODEM, figura A.17

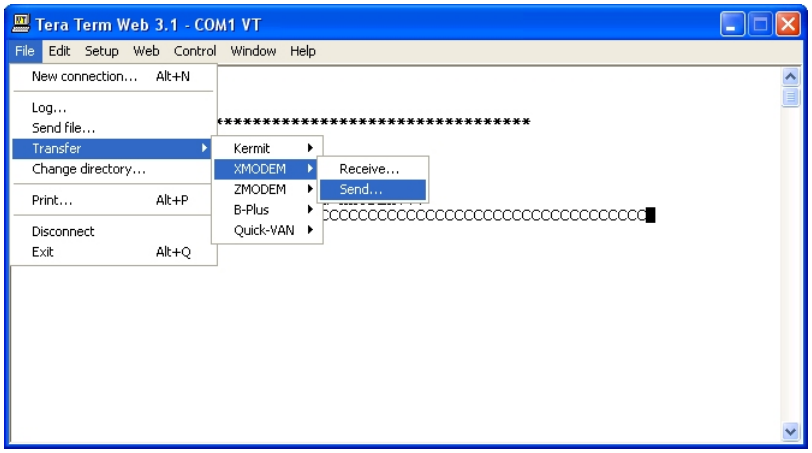


Figura A.17: Selección de transferencia XMODEM

4. Escoger el archivo con la nueva imagen rom, figura A.18.

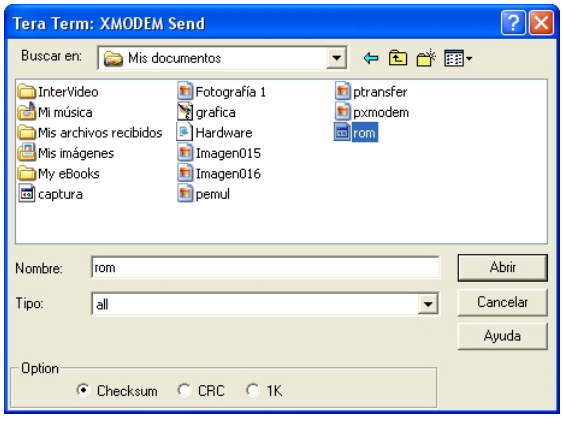


Figura A.18: Escoger imagen ROM

Al escoger el archivo comenzará la transferencia del mismo, figura A.19.

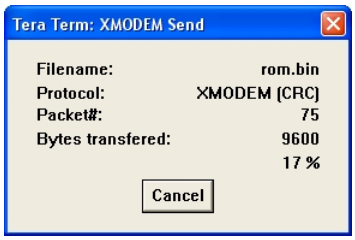


Figura A.19: Transferencia de la imagen

5. Cuando la transferencia esté completa, aparecerá en la ventana del emulador el mensaje: `Nueva imagen rom lista!`. El sistema continuará con las preguntas del modo de configuración.

Adicionalmente al modo de configuración, desde una navegador web también se puede ajustar el valor de la dirección IP, el puerto TCP y fijar la hora y la fecha del servidor, ingresando a la sección Configuración en el menú principal. figura A.20.

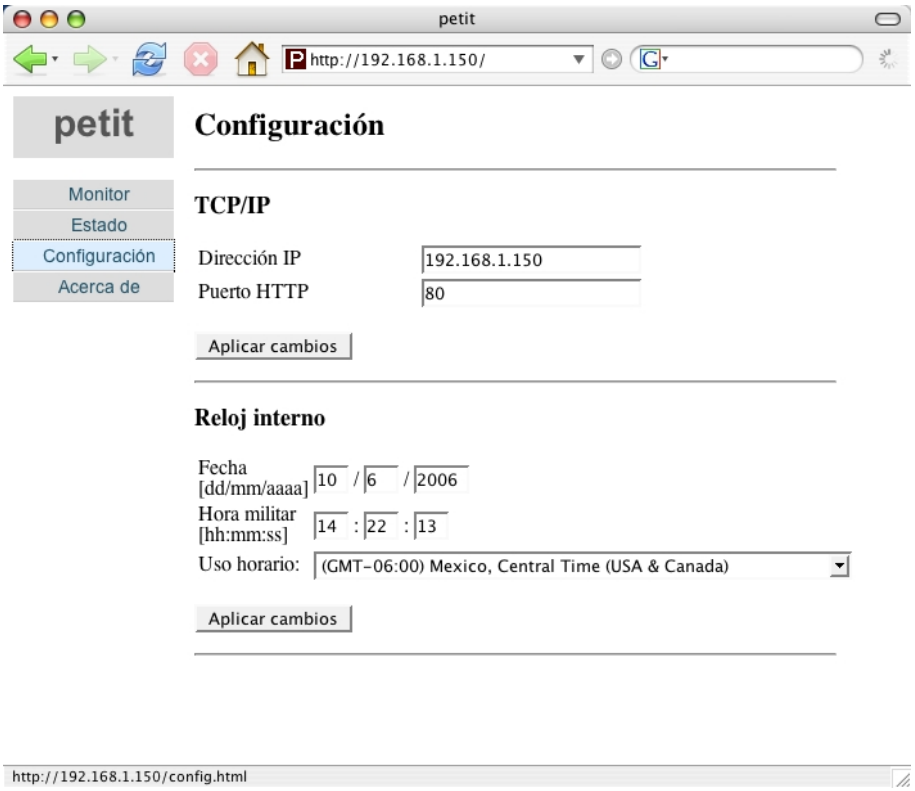


Figura A.20: Configuración desde un navegador web

Apéndice B

Interfaz de programación de la pila uinet

B.1. Módulo ARP

El módulo ARP reside en los archivos `arp.h` y `arp.c`. Los símbolos y las funciones de éste se presenta a continuación:

`uint t_mac[6]`
: Parámetro usado por la función `arp_buscar_ip`

`uint t_ip[4]`
: Parámetro usado por las funciones `arp_buscar_ip` y `arp_sol_mac`.

`uint arp_resp`
: Parámetro usado por la función `arp_sol_mac`.

`uint arp_fuera`
: Parámetro usado por la función `arp_sol_mac`.

`int1 ARP_PET`

`es_arp`
: Macro que se utiliza para determinar si un paquete es un mensaje ARP.

`es_respuesta_arp`
: Macro para determinar si un paquete es un mensaje de respuesta ARP.

`es_peticion_arp`
: Macro para determinar si un paquete es un mensaje de petición ARP.

`void arp_sol_mac()`
: Esta función genera y envía un mensaje de solicitud ARP solicitando la dirección MAC del anfitrión cuya IP sea `t\_ip`. Establece la bandera `arp_resp` en 0 y fija el contador `arp_fuera` en el valor de `c_tiempo+30`. Esto es usado en el bucle principal para determinar si una respuesta ARP llega como consecuencia a la solicitud aquí enviada.

`void arp_env_mac()`
: Esta función genera y envía un mensaje de respuesta ARP anunciando la dirección IP `MYIP`, y la dirección MAC `MAC_ADDR`.

`uint arp_buscar_ip()`
: Verifica que la dirección IP `t_ip`, no esté ocupada por otro dispositivo en la red. Envía tres paquetes *gratuitous ARP* esperando 700 ms entre cada uno, bloquea el control ethernet hasta recibir una respuesta. Si no recibe un mensaje de respuesta regresa un 0, de lo contrario regresa un 1 y coloca en `t_mac` la dirección MAC del dispositivo que tiene cuya IP es `t_ip`.

`void arp_init()`
: Verifica que la dirección IP `MYIP`, no esté ocupada por otro dispositivo en la red utilizando para ello la función `arp_buscar_ip`, de estarlo informa al usuario de la situación a través del puerto serie.

B.2. Módulo IP

El módulo IP reside en los archivos `ip.c` e `ip.h`. Contiene las funciones necesarias para enviar un datagrama IP, así como para mandar mensajes ICMP e IGMP. Las variables y funciones de este módulo son:

`uint16 ip_id_num`
: El contador que lleva la cuenta del último número de secuencia usado en un datagrama IP.

`uint MYIP[4]`
: La dirección actual IP utilizada por el módulo.

`uint MCIP[4]`
: La dirección actual del grupo de multidifusión utilizada por el módulo IP.

`uint ttl_mc`
: El tiempo de vida anunciado en los datagramas de multidifusión.

`int1 IGMP_REPORT`
: Bandera para determinar cuando enviar un mensaje de membresía en el bucle principal.

`uint igmp_fuera`
: Tiempo límite para enviar el mensaje de membresía, utilizado en el bucle principal.

`es_ip`
: Macro para determinar si un paquete es un datagrama IP.

`es_mi_ip`
: Macro para determinar si un datagrama IP está dirigido a la dirección `MYIP`.

`es_all_systems`
: Macro para determinar si un datagrama IP está dirigido a la dirección 224.0.0.1.

`es_mi_grp`
: Macro para determinar si un datagrama IP está dirigido a la dirección del grupo de multidifusión `MCIP`.

`uint16 cksum(int8 *addr,uint16 len)`
: Regresa la suma de verificación empleando el algoritmo estandar de IP de un grupo de `len` datos que comienzan en la dirección `addr`.

`void ip(uint protocolo,uint16 num_datos,uint ttl,uint *ap_IP,uint *ap_MAC)`
: Forma un datagrama IP dirigido a la dirección IP referida por `ap_IP` cuya dirección MAC es `ap_mac` especificando el campo de protocolo el protocolo `proto`, con un tiempo de vida de `ttl` y que contiene el número de datos especificado por `num_datos`.

`void icmp()`
: Envía un mensaje de respuesta echo IGMP a la dirección IP del paquete que se encuentra en el bufer principal.

`void igmp_report()`
: Envía un mensaje de membresía del grupo `MCIP`.

`void igmp_leave()`
: Envía el mensaje de abandono del grupo `MCIP`.

B.3. Módulo TCP

El módulo TCP es el más complejo de la pila uinet, reside en los archivos `tcp.c` y `tcp.h`. Las variables y las funciones de este módulo se describen a continuación:

`struct tcb`
: Esta estructura representa el *transfer control block* de un cliente TCP. Contiene los siguientes campos:

  `uint id`
  : Un número de identificación para el TCB, corresponde al índice de éste elemento en el arreglo `clientes`

  `uint ip[4]`
  : La dirección IP del cliente.

  `uint pto[2]`
  : El puerto IP del cliente.

  `uint mac[6]`
  : La dirección MAC del cliente.

  `uint16 MSS`
  : El tamaño máximo de segmento anunciado por el cliente.

  `uint estado`
  : El estado actual de la máquina de estados TCP para esta conexión.

  `uint32 sndnxt, rcvnxt`
  : El próximo número de secuencia y acuse de recibo que se utilizará y se esperará en la conexión.

`uint16 lude`
: El número de datos que se mandaron en el último paquete, datos que aún no tienen acuse.

`uint sa`
: Estimador de tiempo utilizado para calcular el tiempo de una retransmisión.

`uint sv`
: Estimador de tiempo utilizado para calcular el tiempo de una retransmisión.

`uint tempo`
: El tiempo de viaje del último segmento enviado.

`uint rto`
: El tiempo fuera para una retransmisión.

`uint nrtx`
: El número de retransmisiones que lleva el último paquete enviado.

`tcb clientes[MAX_CLIENTES]`
: El arreglo que contiene los TCB de los clientes.

`tcb *cnte`
: Es un apuntador al TCB del cliente actual en el arreglo `clientes`.

`uint32 ini_seqnum`
: El número de secuencia inicial utilizado cuando un cliente se conecta por primera vez.

`uint* tcp_buf`
: Es un apuntador al contenido del bufer principal en la posición en donde comienza la sección de datos TCP.

`uint tcp_activas`
: El número de clientes activos.

`uint MY_PORT_ADDRESS_H, MY_PORT_ADDRESS_L`
: El puerto TCP en donde escucha el módulo TCP.

`void tcp_env_reset()`
: Genera y envía un segmento TCP de reset acorde al último segmento recibido en el bufer principal.

`void tcp_re_enviar(uint flags, uint16 num_datos)`
: Esta función reenvía el segmento TCP de longitud `num_datos` que está en el bufer principal utilizando las banderas `flags`. Se utiliza exclusivamente en el cuerpo del evento `ev_tcp_rtx`.

`void tcp_enviar(uint flags, uint16 num_datos)`
: Esta función envía el segmento TCP de longitud `num_datos` que está en el bufer principal utilizando las banderas `flags`. No debe utilizarse en el cuerpo del evento `ev_tcp_rtx`.

`void tcp_init()`
: Inicializa las estructuras de datos del módulo TCP, debe invocarse antes de utilizar alguna otra función.

`sint tcp_conectar()`
Esta función busca en el arreglo `clientes` un elemento cuya dirección IP y puerto TCP coincidan con los del paquete recibido en el bufer principal. Si lo encuentra regresa el indice correspondiente al arreglo `clientes`, de lo contrario regresa -1.

`void tcp_cerrar(uint id)`
Cierra la conexión con el cliente cuyo número de indentificación es `id`.

`void tcp()`
Esta función examina el paquete recibido en el bufer principal y dispara el evento correspondiente. Hace uso de las demás funciones del módulo para realizar su tarea. Se encarga de mantener la conexión en el estado correspondiente.

`void tcp_rtx()`
Esta función determina examinado el arreglo clientes si alguno de ellos necesita reenviar un segmento. Si es el caso dispara el evento `ev_tcp_rtx`.

B.3.1. Eventos TCP

La función `tcp` se encarga de disparar alguno de los siguientes eventos según el estado de la máquina de estados TCP y el segmento analizado en el bufer principal.

`void ev_tcp_ini()`
El cliente ha realizado correctamente el saludo de 3 etapas.

`void ev_tcp_fin()`
El cliente ha cerrado la conexión TCP.

`void ev_tcp_rtx()`
Este evento es disparado por la función `tcp_rtx` cuando es necesario que un cliente efectúe una retransmisión.

`sint ev_tcp_psh()`
El cliente ha enviado un segmento con la bandera PSH en 1.

`sint ev_tcp_ack()`
El cliente ha enviado un segmento con la bandera ACK en 1.

B.4. Módulo UDP

El módulo UDP de la pila uinet reside en el archivo `udp.c`. Las funciones y datos de éste son:

`uint ip_udp[4]`
La dirección IP del cliente UDP conectado actualmente.

`uint pto_udp[2]`
La dirección del puerto UDP del cliente.

`uint eth_udp[6]`
La dirección ethernet del cliente.

`int1 usr_udp`
: Esta bandera indica si hay un cliente conectado.

`uint udp_fuera`
: El tiempo fuera para cerrar la conexión con un cliente.

`char *udp_buf`
: El apuntador al principio de los datos UDP del segmento en el bufer principal.

`void udp()`
: Esta función es invocada cada vez que se reciben datos UDP, es la que implementa el protocolo UDP desarrollado en este trabajo.

`void udp_enviar(uint16 num_datos,uint uni_multi)`
: Esta función envía el segmento UDP que se encuentra en el bufer principal que contiene `num_datos`, ya sea en un datagrama unicast `uni_multi=0` o de multidifusión `uni_multi=1`.

Apéndice C

Interfaz de programación del hardware del servidor.

C.1. Interfaz ethernet

La interfaz ethernet utilizada en éste trabajo es parte del compilador CCS PIC C y está desarrollada específicamente para utilizarse con el controlador ethernet RTL8019. Esta interfaz define las siguientes funciones:

- `void nic_init()`: Inicializa los registros internos del controlador.
- `long nic_begin_packet_rx()`: Esta función debe llamarse para comenzar a leer los paquetes recibidos. Regresa el tamaño del paquete recibido, de haber alguno.
- `nic_packet_get_data(char *destino)`: Lee del controlador el paquete recibido en el bufer apuntado por `destino`.
- `void nic_end_packet_rx()`: Termina la lectura de un paquete y regresa el controlador a su operación normal.
- `void nic_putd(char *origen, long tam)`: Se utiliza para cargar un paquete en el controlador y transmitirlo en la red. Lee del bufer `origen` la cantidad de `tam` caracteres y los coloca en el bufer interno del controlador ethernet. Al termino de la lectura el controlador envía el paquete a la red ethernet.

Si el controlador se cambia por uno distinto, solo es necesario implementar la interfaz para el nuevo controlador y la pila uinet podrá ser reutilizada.

C.2. Interfaz del reloj de tiempo real DS1338

Las funciones que utiliza el servidor Petit para accesar a los datos del reloj de tiempo real se encuentran en el archivo `ds1338.c`. El reloj mantiene la información del tiempo representada en formato BCD y es necesario hacer el cambio entre binario y BCD al utilizar el reloj. Las funciones de la interfaz del reloj son las siguientes:

`char DS1338_regs[DS1338_DATE_TIME_BYTE_COUNT]`
: Este arreglo es utilizado por las funciones `set_date` y `read_date` para fijar y leer la hora del reloj. Los datos que representa cada elemento del arreglo son:

- Segundos [0]: 0-59.
- Minutos [1]: 0-59.
- Horas [2]: 0-23.
- Día de la semana [3]: 1-7.
- Día del mes [4]: 1-31.
- Mes [5]: 1-12.
- Año [6]:00-99, basado en el año 2000.
- Registro de control [7].
- GMT [8]: -12 a +12.

`void DS1338_set_date_time(void)`
Esta función fija la hora y fecha del reloj a los valores del arreglo `DS1338_regs`.

`void DS1338_read_date_time(void)`
Lee del reloj la hora y fecha en el arreglo `DS1338_regs`.

`char DS1338_read_byte(char addr)`
Lee el byte en la dirección `addr` del reloj, mediante el bus I2C.

`void DS1338_write_byte(char addr, char value)`
Escribe en la dirección interna `addr` del reloj el valor `value`.

`char bcd2bin(char bcd_value)`
Regresa el valor binario del caracter en BCD `bcd_value`.

`char bin2bcd(char bin_value)`
Regresa el valor BSD del caracter en binario `bin_value`.

C.3. Interfaz del sistema de archivos

Los archivos del servidor Petit residen en una memoria EEPROM. Los tipos de datos y las funciones para accesar al sistema de archivos utilizado en ésta, se encuentran en el archivo `fs.c` y se describen a continuación:

`struct freg`
Esta estructura representa una entrada del sistema de archivos y contiene los siguientes campos:

`int16 tam`
El tamano en bytes del archivo.

`int16 ap`
El offset a la posición actual en el contenido del archivo.

`int16 dini`
La direccion de inicio del archivo en la eeprom.

`int8 tipo`
El tipo del archivo: 0=binario, 1=dinámico y 2=comprimido.

`int8 mime`
El tipo mime del archivo. Los valores se encuentran en el cuadro 4.2

`freg factual`
El archivo abierto actualmente.

`int1 feof`
Este bit se pone en 1 cuando la lectura del archivo actual llega al final.

`int8 fopen(char *archivo)`
Abre el archivo especificado por la cadena `archivo`, e inicializa con los datos de éste el registro `factual`. Regresa 0 si el archivo se encontró en el sistema de archivos y 1 de lo contrario.

`void freopen(void)`
Reabre el archivo especificado en `factual`.

`int16 fread(char *bufer, int16 num_bytes)`
Coloca en el `bufer` el número de caracteres especificado por `num_bytes` del archivo `factual`. Regresa el número de caracteres leídos.

`int16 freadtoken(char *bufer, char token, char *parm,int16 lim)`
Coloca en el `bufer` hasta un máximo de caracteres especificado por `num_bytes` o hasta encontrar el caracter `token` en el archivo `factual`. Regresa el número de caracteres leídos.

`void fclose()`
Cierra el archivo `factual`.

C.4. Interfaz del transceiver TRF-24G

El transceiver de RF TRF-24G de Laipac Technology es un componente basado en el chip nrf2401 de Nordic Semiconductor, este chip es el que realiza todas las tareas del transceiver. La interfaz de software para utilizar este chip se describe en esta sección. Las funciones y los símbolos se encuentran en el archivo `nrf2401.c`.

`BCDIR`
La direccion broadcast de la red de sensores, su valor por defecto es 0xFF.

`MAX_PAYLOAD`
El tamaño en bytes de los paquetes a enviar y recibir, por defecto es 26.

`int const pconfig[15]`
La cadena de configuración del nrf2401. Por defecto esta cadena configura al transceiver en el modo de operación por ráfaga con:

- Direcciones físicas de 4 bytes, valor por defecto: 75-6e-61-6d (código ASCII de 'u','n','a','m')
- Generación de checksum de 16 bits.
- Canal de frecuencia: 2492 MHz.

- Velocidad de transmisión de 250 Kbps.

`void set_tx_mode()`
Configura al transceiver en el modo de transmisor, debe invocarse antes que las funciones `write_nrf` y `write_byte_nrf`. Las escrituras al bufer interno del transceiver comienzan en el principio de éste.

`void set_rx_mode()`
Configura al transceiver en el modo de receptor. Cuando el transceiver recibe un paquete exitosamente fija la línea DR1 del mismo en 1. El microcontrolador puede polear esta línea o utilizarla como fuente de interrupción.

`void write_byte_nrf(int dato)`
Escribe un byte en la posición actual del bufer de transmisión del transceiver y avanza una posición dentro de éste.

`int read_byte_nrf()`
Regresa un byte del bufer de recepción interno del transceiver en la posición actual y avanza una posición en el mismo. Cuando el bufer se ha leído completamente el transceiver fija el pin DR1 en 0. El microcontrolador puede polear esta línea o utilizarla como fuente de interrupción.

`void init_nrf()`
Inicializa el registro de configuración del transceiver enviando la cadena de configuración `pconfig`. Esta función debe invocarse antes que cualquier otra.

`void write_nrf(char *paq)`
Escribe la cadena de caracteres `paq` en el bufer de transmisión del transceiver. Cuando se han escrito `MAX_PAYLOAD` caracteres el transceiver, éste comienza la transmisión de los mismos.

`void read_nrf(char *paq)`
Escribe en `paq` hasta `MAX_PAYLOAD` caracteres leídos del bufer de recepción interno del transceiver.

Apéndice D

Diagramas esquemáticos de la tarjeta Easy Ethernet

En este apéndice se presentan los diagramas originales de la tarjeta Easy Ethernet. En la figura D.2 se aprecia el microcontrolador, el puerto serie y el regulador de voltaje que provee la alimentación de 5 volts de la tarjeta.

El cristal del microcontrolador aparece como de 20 MHz, pero para la realización del proyecto se cambió por uno de 10 MHz para operar con el microcontrolador PIC18F4525. Aunque esto pareciera que reduce la velocidad del nuevo micro, éste cuenta con un PLL que cuadriplica la velocidad del reloj logrando operar hasta los 40 Mhz.

Los pines I/O2 a I/O5 corresponde al conector de entrada/salida al que se hace referencia en la sección 2.4.De igual manera el conector SPI/I2C no aparece en el esquemátic, pero se compone de los pines +5, SDA, SDO, SCL y GND.

La figura D.3 muestra el controlador ethernet junto con el conector del puerto, los leds LED1 y LED2 se encuentran dentro del conector del puerto.

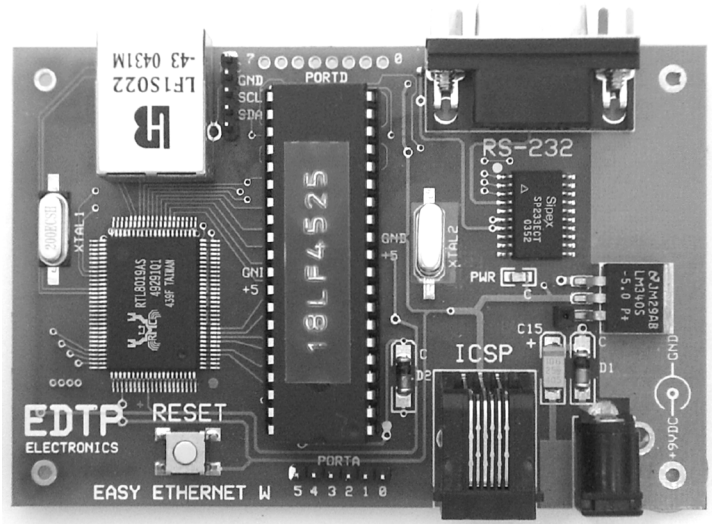


Figura D.1: Tarjeta de desarrollo Easy Ethernet

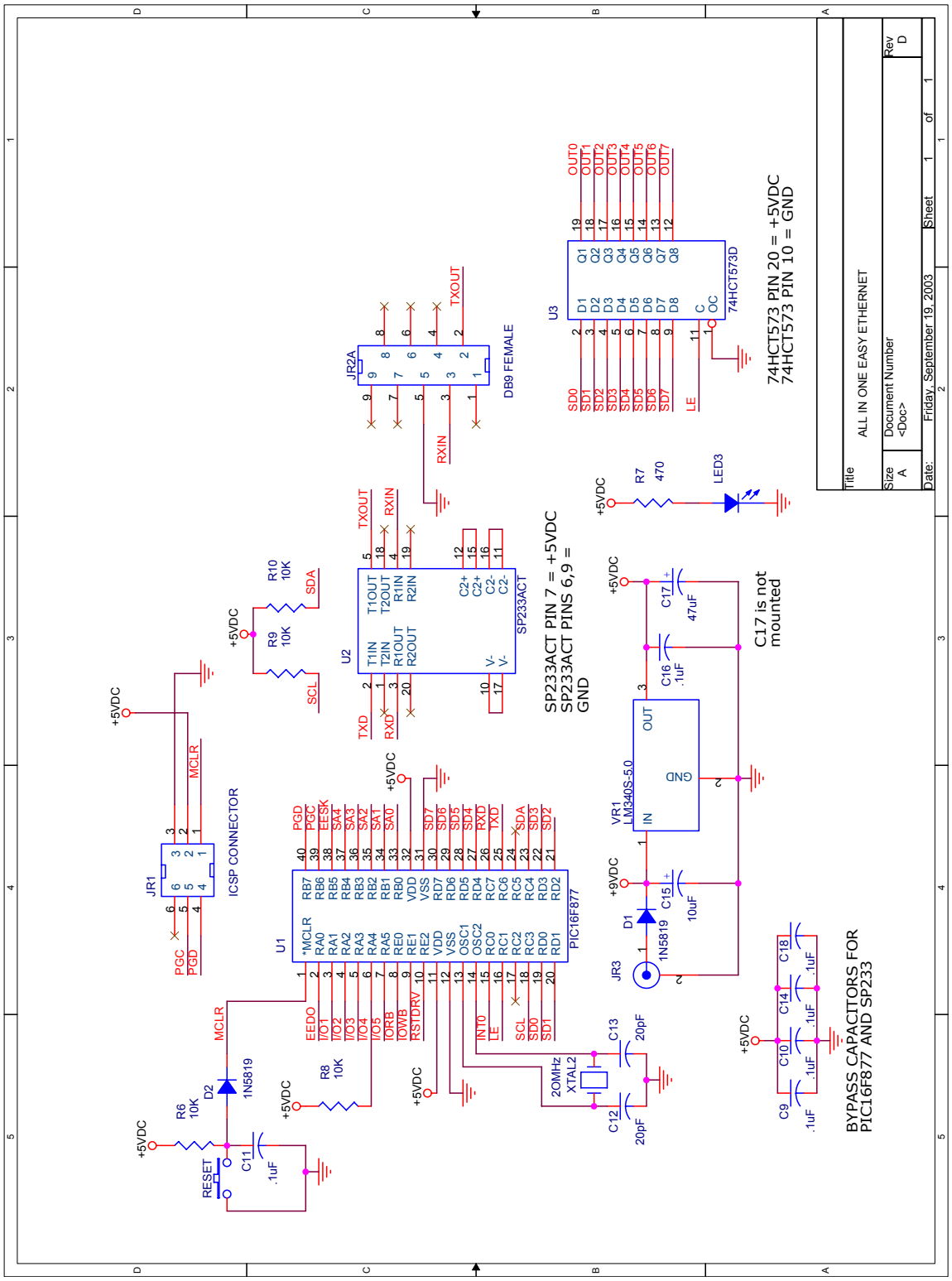


Figura D.2: Diagrama esquemático Easy Ethernet, página 1.

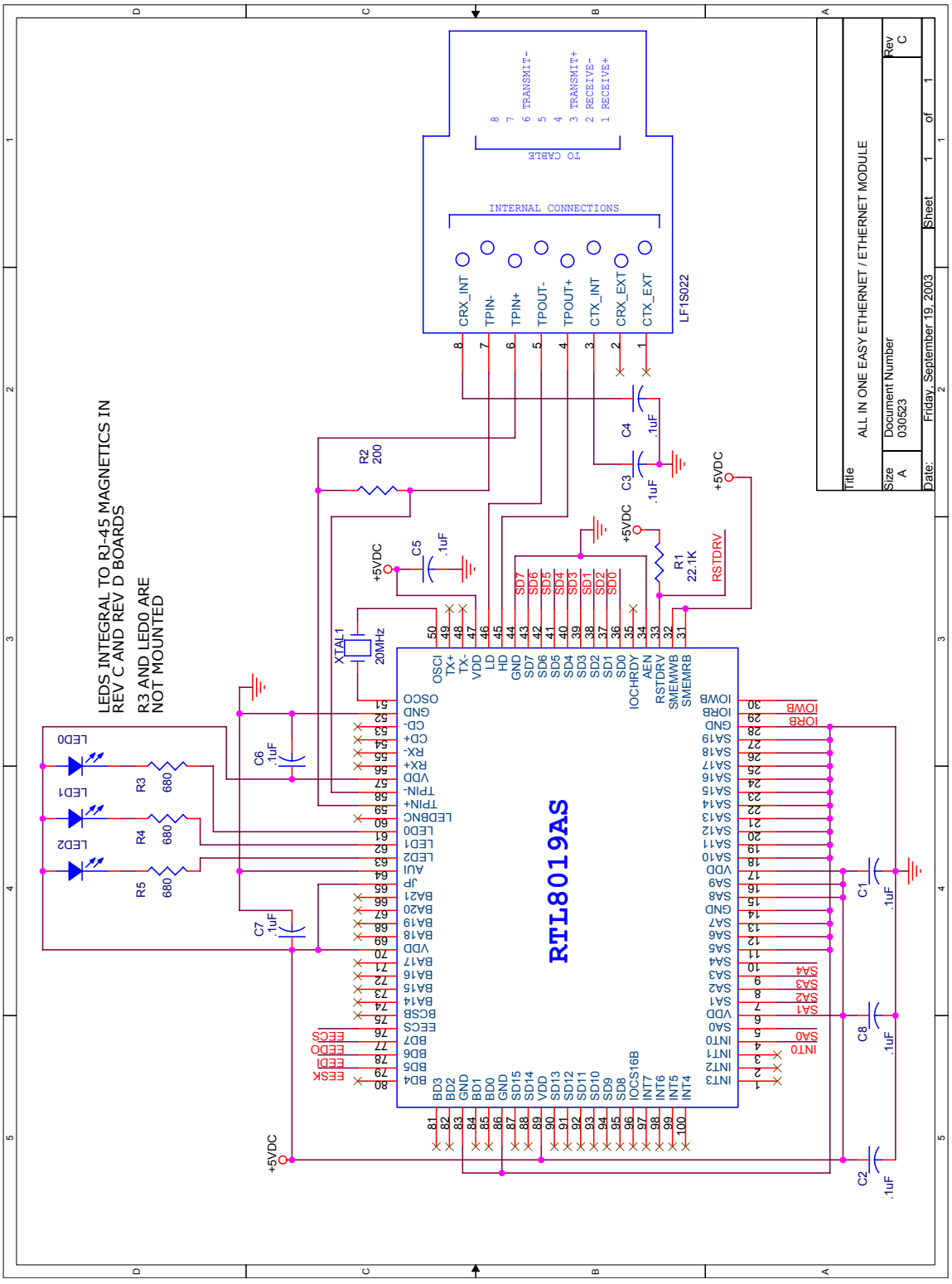


Figura D.3: Diagrama esquemático Easy Ethernet, página 2.

Apéndice E

Diagramas esquemáticos y circuitos impresos de la red de sensores

En este apéndice se incluyen los diagramas esquemáticos del prototipo del controlador central y de los sensores de temperatura. El prototipo del controlador central, figura E.3, incluye los componentes del servidor web que no se encuentran en la tarjeta easy ethernet y los del controlador mismo.

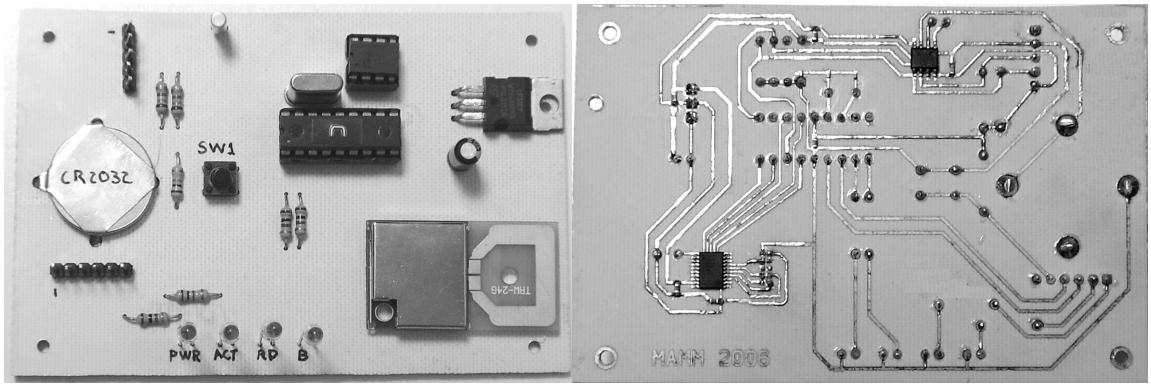


Figura E.1: Prototipo controlador central con componentes del servidor web, cara superior (izquierda), cara inferior(derecha)

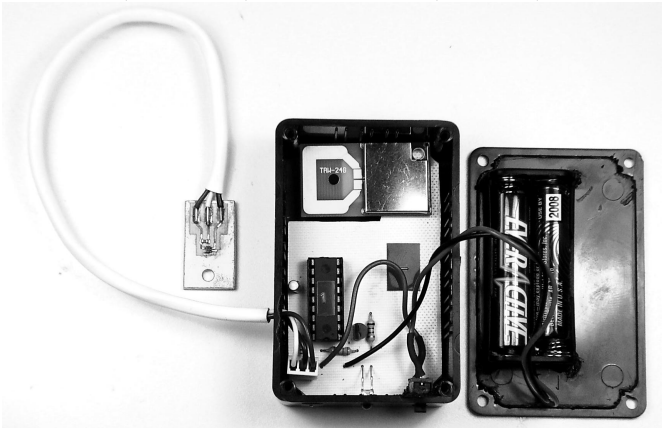


Figura E.2: Sensor de temperatura

E.1. Lista de materiales del controlador central y servidor web

Parte	Utilizados	Valor	Referencias
Capacitor 0403	2	22p	C1 C2
Capacitor 0603	6	0.1u	C3 C4 C5 C6 C7 C8
Capacitor elect. 16V	1	1u	C9
Conector SIP macho	1	5 pines	U4
Conector SIP macho	1	6 pines	U5
Cristal	1	20MHz	X1
Cristal LP	1	32.768KHz	X2
DS1338	1		U7
LF33CV	1		U9
Led 3mm	4	Verde	D1 D2 D3 D4
MAX3002	1		U2
Memoria 24LC512	1		U6
Microswicth	1		SW1
PIC16LF819	1		U1
Porta batería CR2032	1		BAT1
Resistencia 1/4W	2	1.8K	R3 R4
Resistencia 1/4W	1	10K	R5
Resistencia 1/4W	4	1K	R1 R2 R6 R7
Transceiver TRF-2.4G	1		U3

E.2. Lista de materiales del sensor de temperatura

Parte	Utilizados	Valor	Referencias
Cristal LP	1	32.768KHz	X1
Capacitor 0403	2	33p	C1 C2
Capacitor 0603	2	0.1u	C3 C4
Capacitor 0603	1	1u	C6
LM285-2.5	1		D1
Led 3mm	1	Verde	D2
PIC16LF819	1		U1
Portabatería 2 AAA	1	3V	BAT1
Resistencia 1/4W	1	1K	R2
Resistencia 1/4W	1	2.2K	R3
Resistencia 1/4W	1	3.3K	R1
Transceiver TRF-2.4G	1		U2

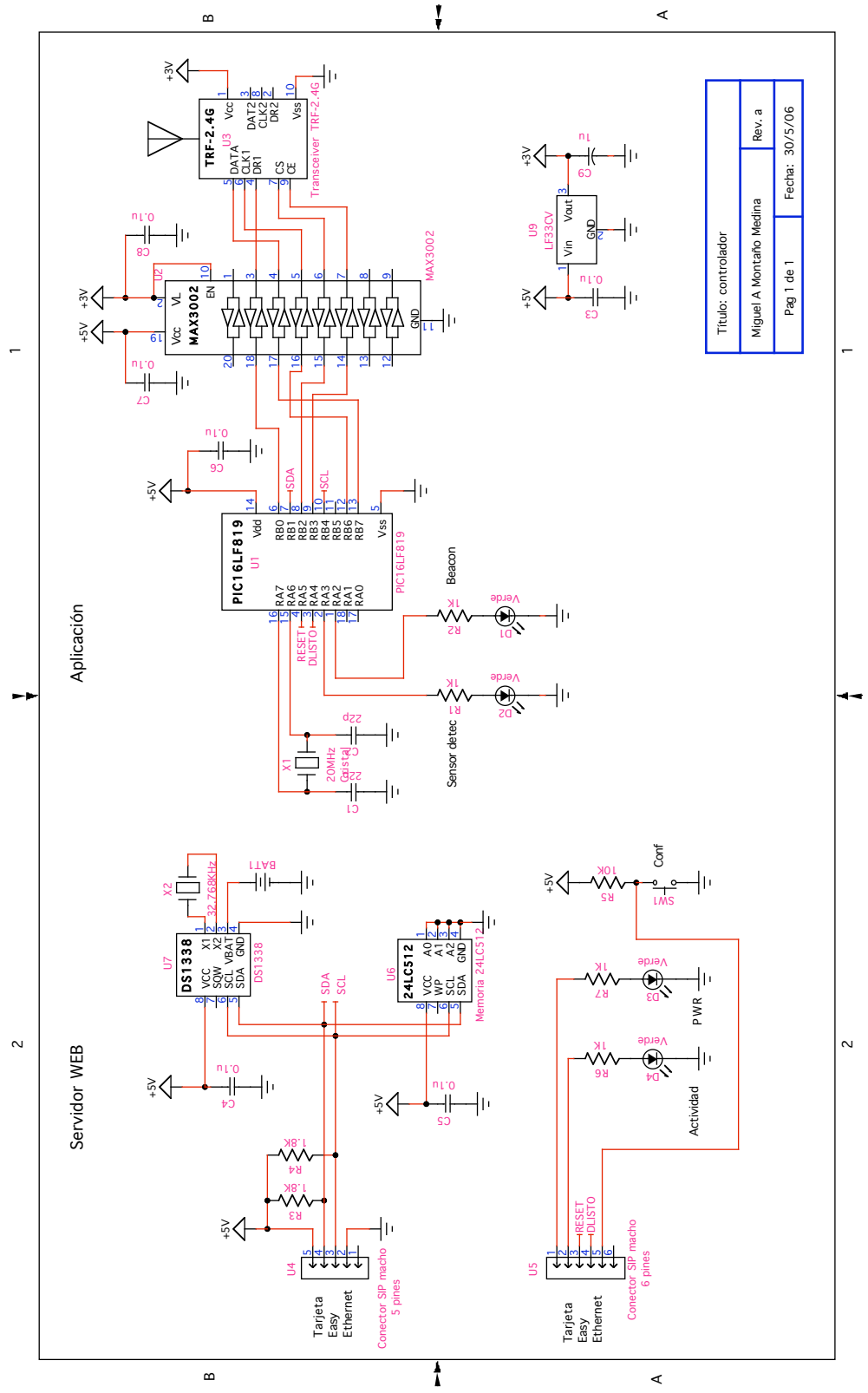


Figura E.3: Diagrama esquemático del controlador central junto con los componentes del servidor web.

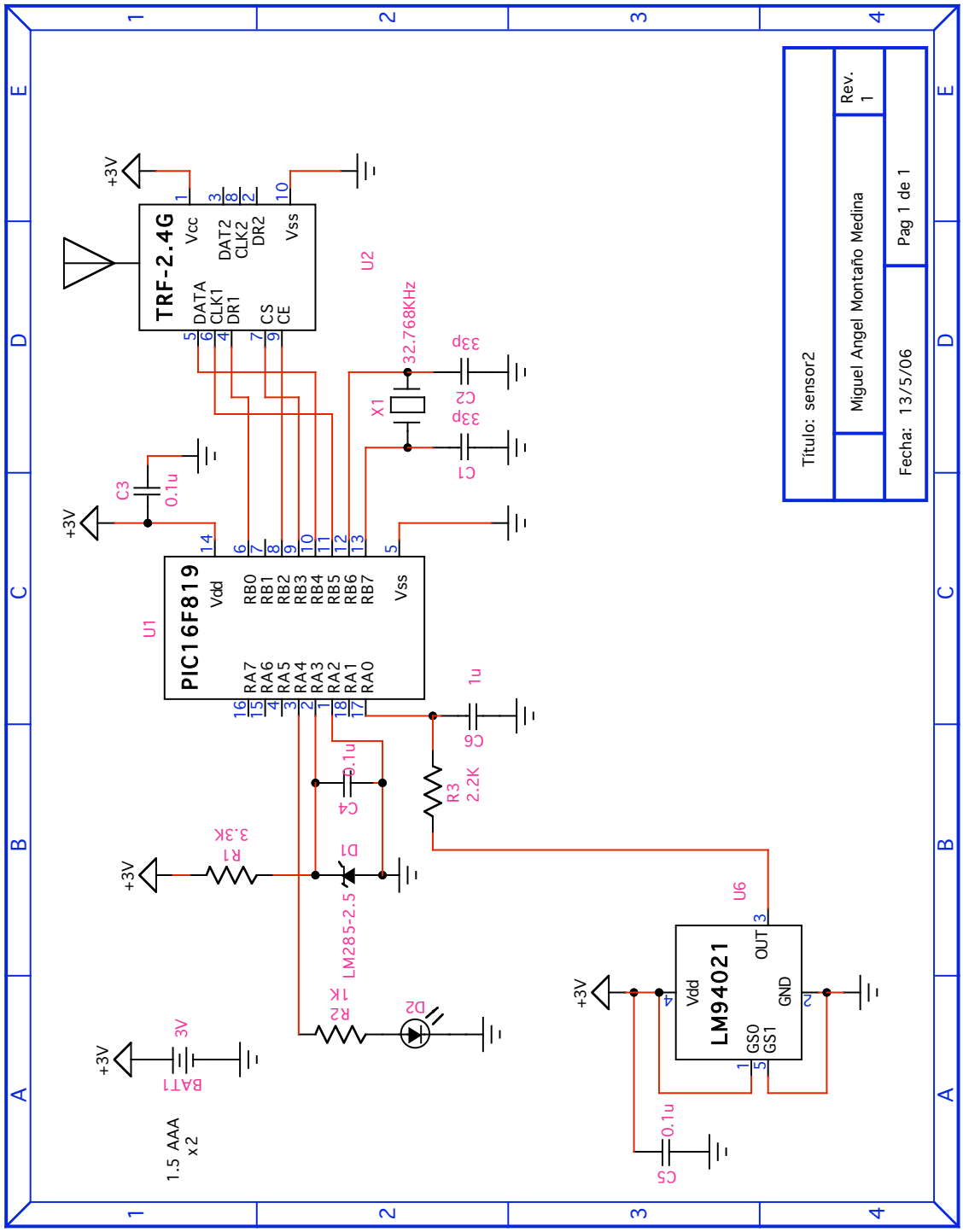


Figura E.4: Diagrama esquemático del sensor de temperatura

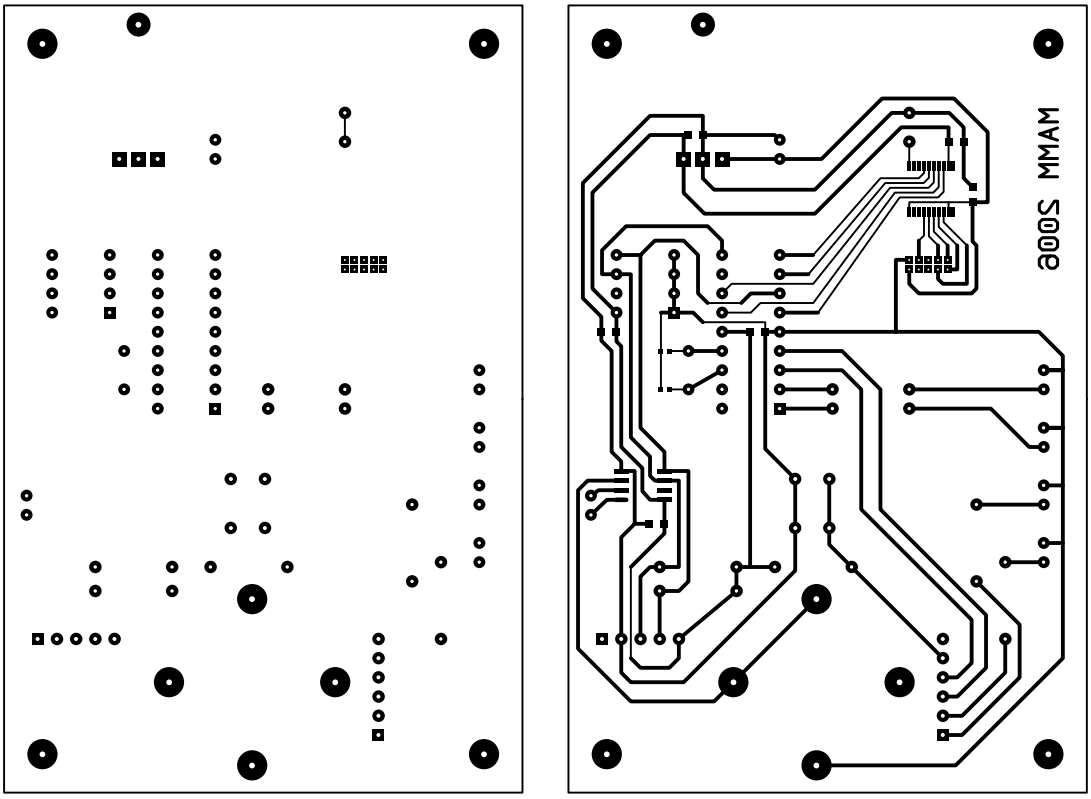


Figura E.5: Circuito impreso del controlador central y servidor web, capa superior (izquierda), capa inferior (derecha).

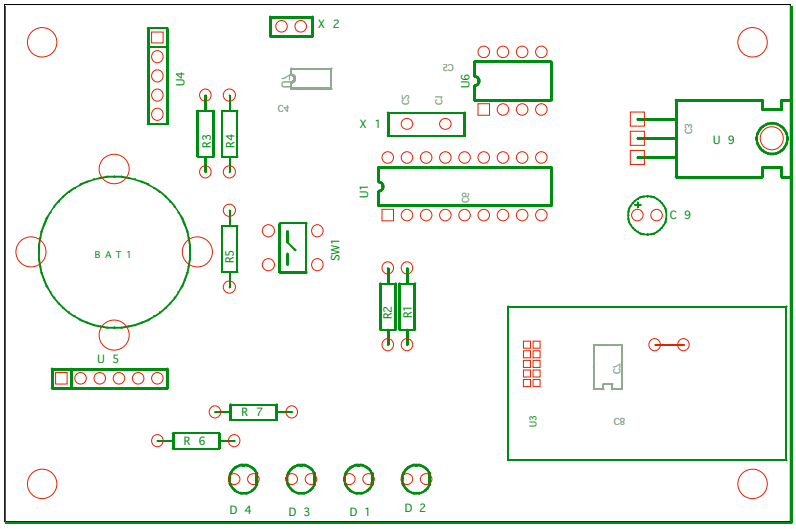


Figura E.6: Circuito impreso del sensor de temperatura, orientación de componentes (vista superior)

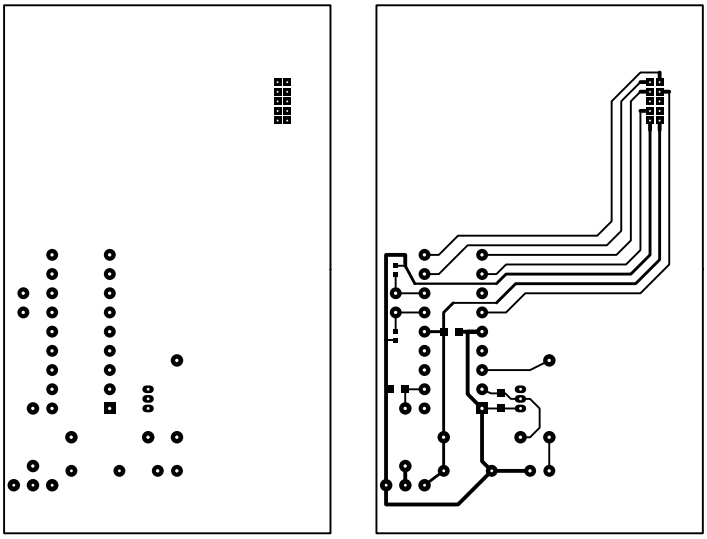


Figura E.7: Circuito impreso del controlador central, cara superior (izquierda), cara inferior (derecha).

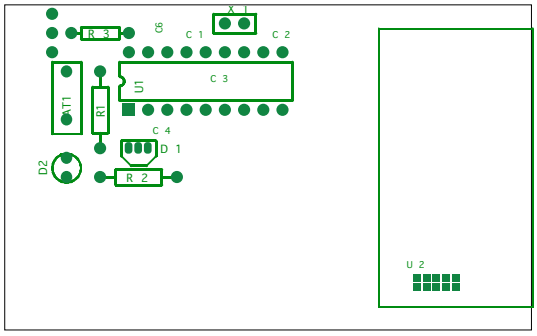


Figura E.8: Circuito impreso del controlador central, orientación de componentes (vista superior)

Apéndice F

Applet Monitor

El programa que se utiliza para capturar las mediciones de los sensores de temperatura es una applet de Java denominada appletMonitor. Esta applet se guarda en la memoria eeprom del servidor web para ser descargada por el usuario cuando éste requiera utilizar la red de sensores.

La applet efectúa conexiones UDP unicast o multicast al controlador central para capturar las mediciones que éste envía, las grafica en pantalla y puede guardar las mediciones en un archivo en la computadora del usuario.

La máquina virtual de Java no permite que un applet lea o escriba archivos en la computadora donde se ejecuta, ni que ésta se comunique con servidores distintos al servidor de donde se descargó, por motivos de seguridad. Pero para aplicaciones en las que es necesario realizar estas tareas, la plataforma java provee el mecanismo de firma electrónica y certificado de autenticidad, con lo cual a una applet se le puede permitir realizar las tareas mencionadas.

Debido a lo anterior es por lo que aparece una ventana solicitando al usuario el permiso de ejecución de la applet cada vez que ésta se inicia, figura F.1. En el disco compacto que acompaña a este trabajo se encuentra el código fuente con las clases y los archivos compilados[1], además del archivo `.jar` que agrupa a las clases de la applet junto con la firma electrónica que permite la ejecución de las mismas.

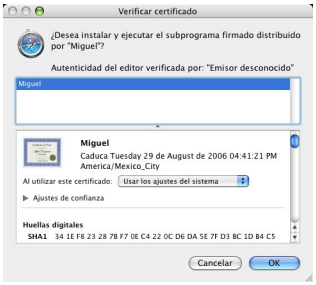


Figura F.1: Verificación del autor y permiso para ejecución del applet

El diseño de la applet se describe en este apéndice mediante los artefactos del lenguaje UML: diagrama de casos de uso, diagrama de clases y diagrama de secuencia.

[1]Si fuese necesario recompilar el applet, ésta se debe firmar nuevamente para permitir su ejecución. El artículo de Pawlan y Dodda [33], describe los pasos necesarios para firmar electronicamente un applet

F.1. Casos de uso

En esta sección se presentan los casos de uso que ayudaron a definir el funcionamiento del appletMonitor. En la figura F.2 se presenta el diagrama de casos de uso de la applet.

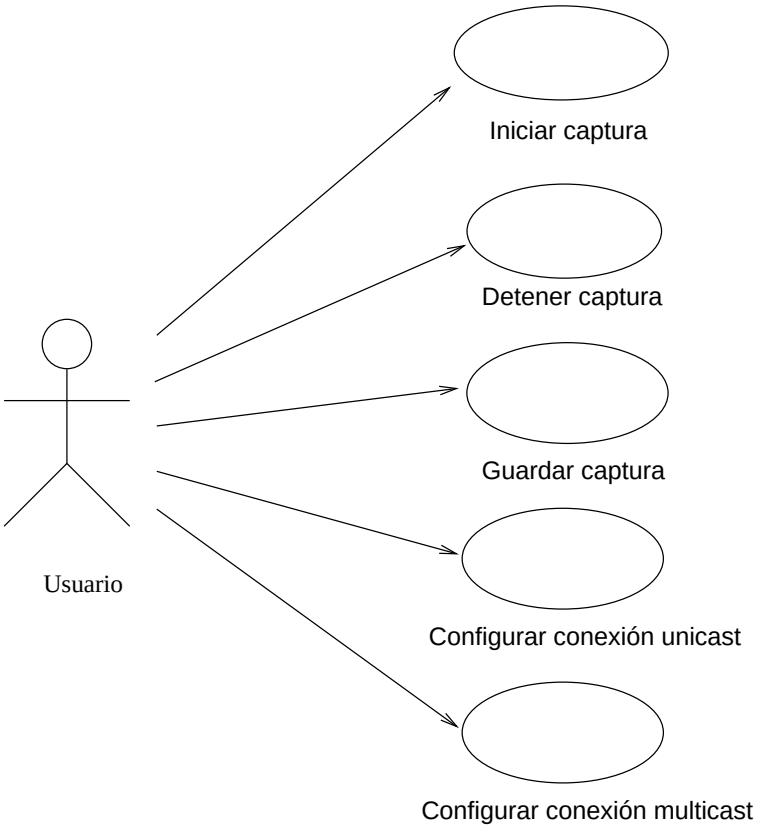


Figura F.2: Diagrama de casos de uso

F.1.1. Descripción de casos de uso

1. Iniciar captura

1.1. Breve descripción.
Es el caso en el que el usuario inicia la captura de datos.

1.2. Flujo de eventos

1.2.1. Comienza cuando el usuario selecciona la opción “Iniciar captura” en el menú “Sistema”.

1.2.2. El sistema abre un socket y espera a que lleguen los datos del controlador central.

1.2.3. El sistema crea un archivo temporal y guarda en él cada una de las muestras recibidas.

1.2.4. El sistema interpreta cada uno de los datos recibidos y los grafica en la pantalla.

1.2.5. Según los datos recibidos se actualizan los valores de las etiquetas “Máximo: ”, “Mínimo” y “Último: ” en cada una de las gráficas en pantalla.

1.2.6. Fin del caso de uso.

1.3. Precondiciones

1.3.1. Se debe escoger el tipo de conexión: unicast o multicast antes de iniciar este caso de uso.

1.4. Poscondiciones

1.4.1. Las opciones “Guardar captura como..” y “Configurar conexión” del menú “Sistema” deben deshabilitarse mientras la captura está en progreso.

1.4.2. Se debe actualizar la gráfica en pantalla para reflejar los datos recibidos.

2. Detener captura

2.1. Breve descripción.

Es el caso en el que el usuario detiene la captura de datos.

2.2. Flujo de eventos

2.2.1. Comienza cuando el usuario selecciona la opción “Detener captura” en el menú “Sistema”.

2.2.2. El sistema cierra el socket donde esperaba los datos del controlador central.

2.2.3. El sistema cierra el archivo temporal.

2.2.4. Fin del caso de uso.

2.3. Precondiciones

2.3.1. Se debe iniciar el caso de uso Iniciar captura antes que este caso de uso.

2.4. Poscondiciones

2.4.1. Las opciones “Guardar captura como..” y “Configurar conexión” del menú “Sistema” deben habilitarse al terminar la captura.

3. Guardar captura

3.1. Breve descripción.

Es el caso en el que el usuario guarda los datos capturados.

3.2. Flujo de eventos

3.2.1. Comienza cuando el usuario selecciona la opción “Guardar captura” en el menú “Sistema”.

3.2.2. El sistema despliega un cuadro de dialogo con el título “Guardar captura como...”.

3.2.3. El usuario selecciona un archivo en el cuadro de diálogo y éste desaparece.

3.2.4. El sistema copia los datos del archivo temporal al archivo seleccionado por el usuario.

3.2.5. Fin del caso de uso.

3.3. Precondiciones

3.3.1. Se debe contar con datos para guardar: haber efectuado el caso de uso 1. y luego el caso 2..

4. Configurar conexión unicast

4.1. Breve descripción.

Es el caso en el que el usuario configura una conexión unicast para capturar las mediciones del controlador central.

4.2. Flujo de eventos

4.2.1. Comienza cuando el usuario selecciona la opción “Configurar conexión” en el menú “Sistema”.

4.2.2. El sistema despliega un cuadro de selección con el título “Seleccione el tipo de conexión” y las opciones “Unicast” y “Multicast”.

4.2.3. El usuario selecciona la opción unicast.

4.2.4. El sistema utilizará un socket UDP para la conexión con el controlador central, y mostrará en la ventana principal un mensaje indicando la interfaz unicast junto con la dirección IP del controlador central.

4.2.5. Fin del caso de uso.

5. Configurar conexión Multicast

5.1. Breve descripción.

Es el caso en el que el usuario configura una conexión multicast para capturar las mediciones del controlador central.

5.2. Flujo de eventos

5.2.1. Comienza cuando el usuario selecciona la opción “Configurar conexión” en el menú “Sistema”.

5.2.2. El sistema despliega un cuadro de selección con el título “Seleccione el tipo de conexión” y las opciones “Unicast” y “Multicast”.

5.2.3. El usuario selecciona la opción “Multicast”.

5.2.4. El sistema desplegará un nuevo cuadro de selección con el título “Seleccione la interfaz de conexión” y en las opciones se desplegarán todas las interfaces de conexión con que dispone la computadora del usuario.

5.2.5. El usuario escogerá una interfaz y el sistema mostrará en la ventana principal un mensaje con la interfaz seleccionada.

5.2.6. El sistema utilizará un socket UDP multicast en la interfaz seleccionada para la conexión con el controlador central.

5.2.7. Fin del caso de uso.

F.2. Diagramas de secuencia

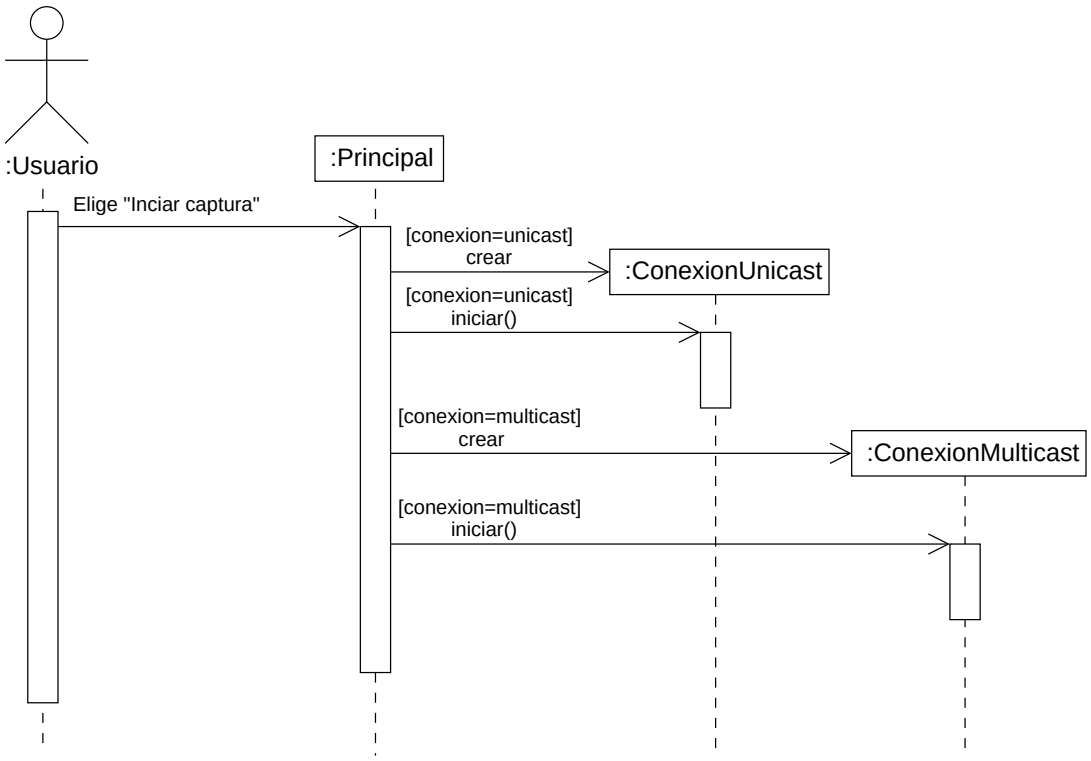


Figura F.3: Diagrama de secuencia del caso de uso 1.

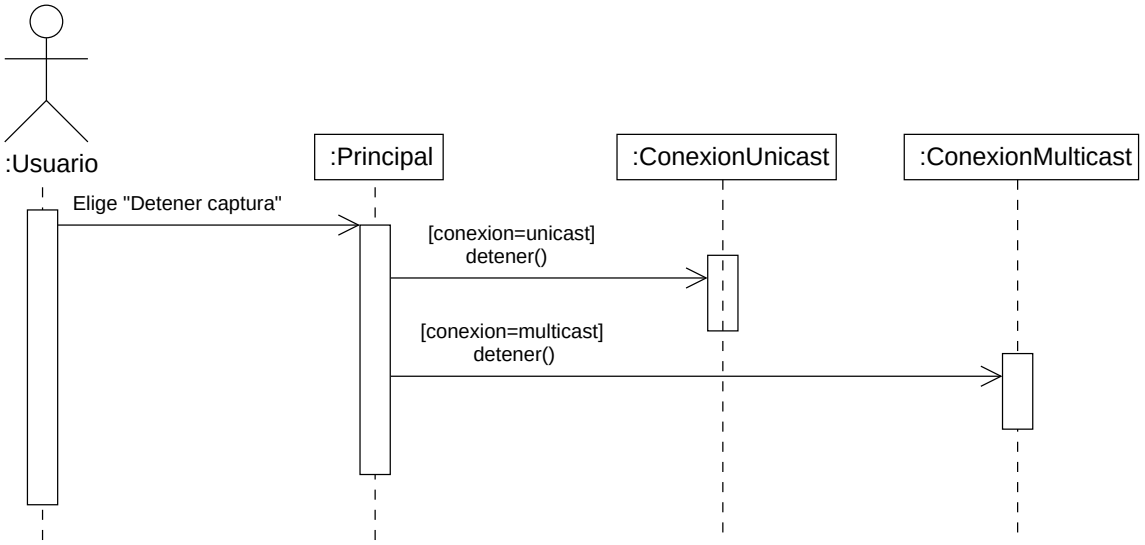


Figura F.4: Diagrama de secuencia del caso de uso 2.

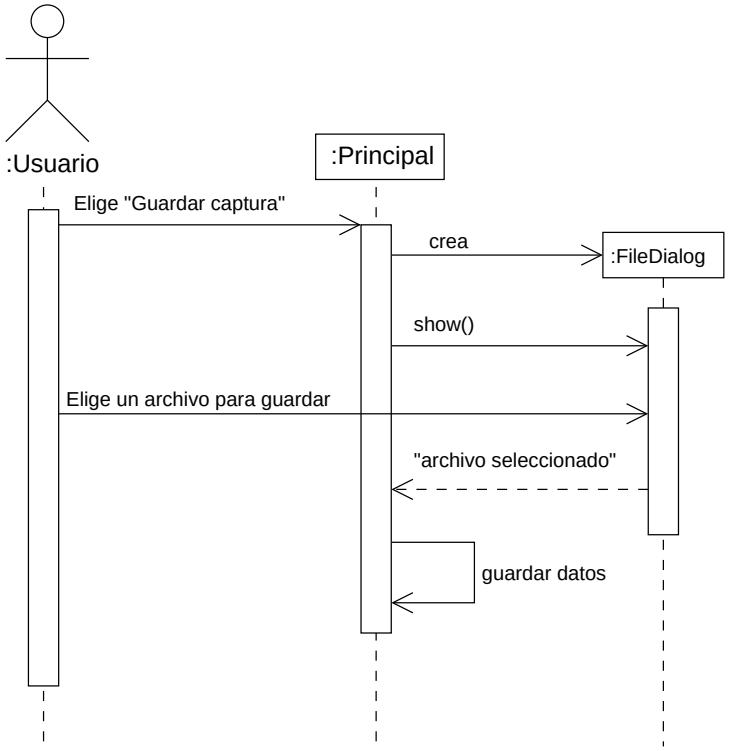


Figura F.5: Diagrama de secuencia del caso de uso 3.

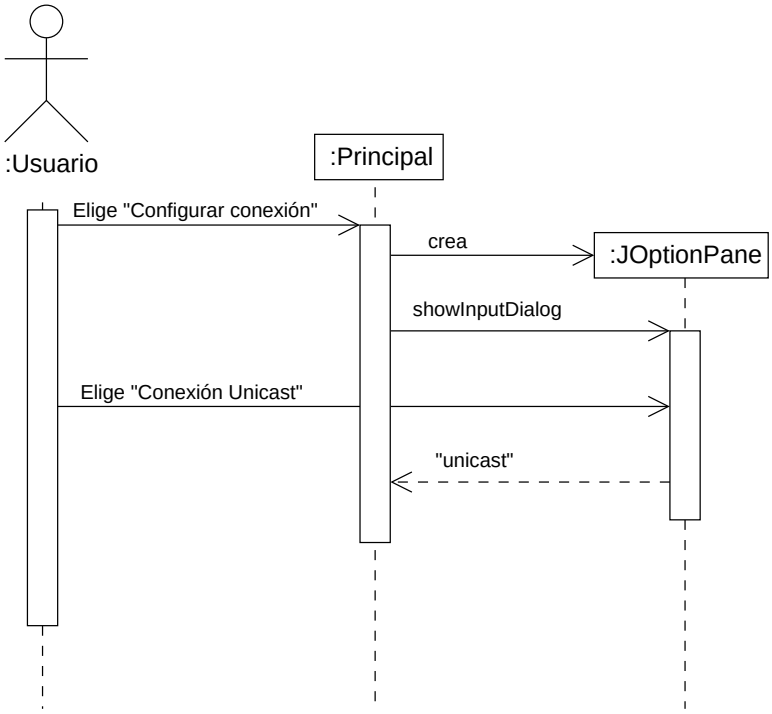


Figura F.6: Diagrama de secuencia del caso de uso 4.

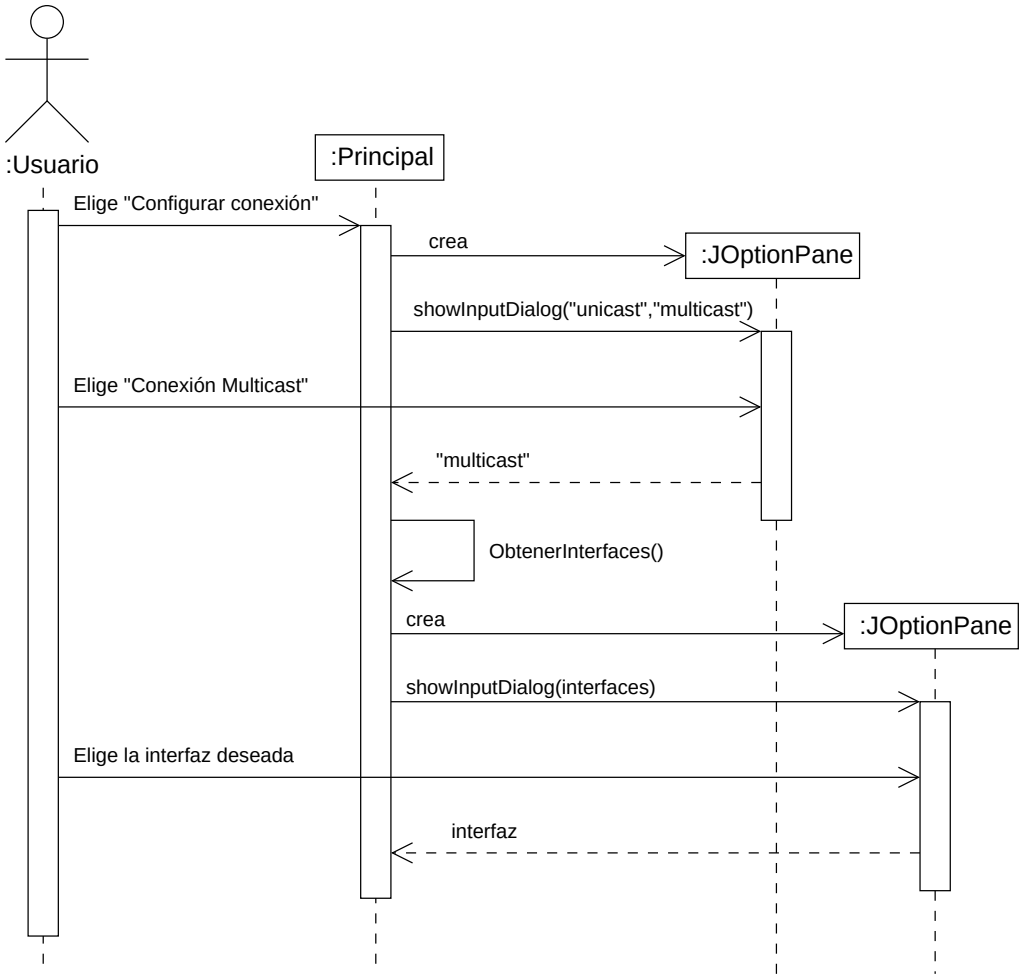


Figura F.7: Diagrama de secuencia del caso de uso 5.

F.3. Diagrama de clases

El diagrama de clases de implementación del appletMonitor se muestra en la figura F.8.

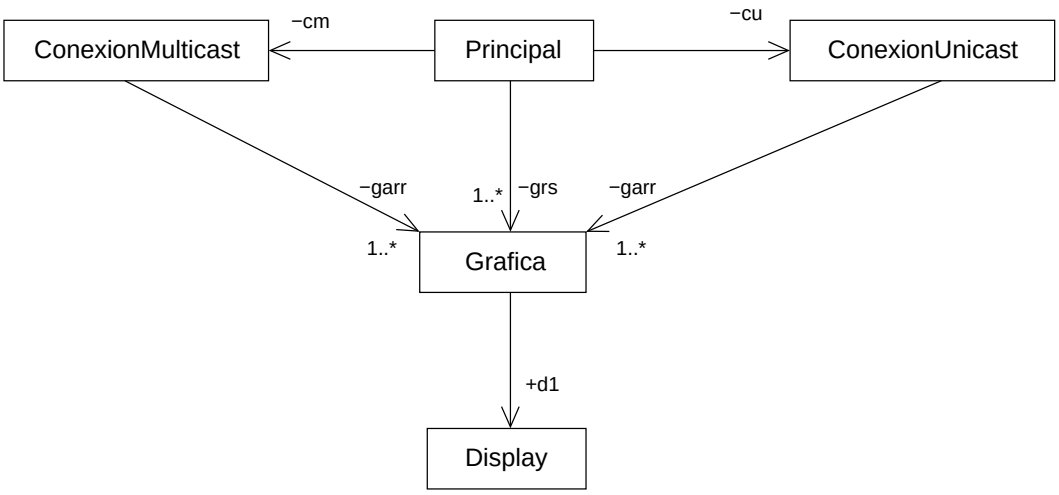


Figura F.8: Diagrama de clases de la applet Monitor

Las clases que componen al appletMonitor son:

- Principal: Es una extensión de la clase JFrame, y representa la ventana principal del sistema.
- ConexionUnicast: Esta clase se encarga de manejar la conexión unicast utilizando el protocolo de comunicación UDP que se presentó en la seccion 3.9.
- ConexionMulticast: Esta clase maneja la conexión multicast.
- Grafica: Es una extensión de la clase JPanel y contiene a los elementos que forman a una gráfica: el display y las etiquetas de “Máximo”, “Mínimo” y “Ultimo”.
- Display: Es una extensión de la clase JPanel y se encarga de mostrar una gráfica con los datos recibidos de las mediciones.

En el disco compacto anexo a esta tesis se encuentra el código fuente de las clases aquí mostradas.

Bibliografía

[1] Postel, J., *User Datagram Protocol*, RFC 768, USC/Information Sciences Institute, Agosto 1980.

[2] Postel, J. (ed.), *Internet Protocol - DARPA Internet Program Protocol Specification*, RFC 791, USC/Information Sciences Institute, Septiembre 1981.

[3] Postel, J., *Internet Control Message Protocol - DARPA Internet Program Protocol Specification*, RFC 792, USC/Information Sciences Institute, Septiembre 1981.

[4] Postel, J. (ed.), *Transmission Control Protocol -DARPA Internet Program Protocol Specification*, RFC 793, USC/Information Sciences Institute, Septiembre 1981.

[5] Crocker, David H. *Standard for the format of ARPA internet text messages*, RFC 822, University of Delaware, agosto 1982.

[6] Braden, R.; Borman, D.; Partridge, C.; *Computing the Internet Checksum*, RFC 1071, Network Working Group, Septiembre 1981.

[7] Deering, S., *Host Extensions for IP Multicasting*, RFC 1112, Stanford University, Computer Science Department, Agosto 1989.

[8] Braden,R. (ed.), *Requirements for Internet Hosts – Communication Layers* , RFC 1122, USC/Information Sciences Institute, Octubre 1989.

[9] Berners-Lee, T., Fielding, R. and H. Frystyk, *Hypertext Transfer Protocol – HTTP/1.0*, RFC 1945, Network Working Group, Mayo 1996.

[10] Deutsch, P. *GZIP file format specification version 4.3*, RFC1952, Aladdin Enterprises, Mayo 1996.

[11] Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H. and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2068, Enero 1997.

[12] Fenner, W. *Internet Group Management Protocol, Version 2*, RFC 2236, Xerox PARC, Noviembre 1997.

[13] Karn, P.; Partridge, C. *Estimating round-trip times in reliable transport protocols.* In Proceedings of SIGCOMM '87, ACM, Agosto 1987.

[14] Jacobson, V.; Karels, Michael J. *Congestion avoidance and control.* In Proceedings of SIGCOMM '88, ACM, Agosto 1988.

[15] Atmel Corp., *AVR460: Embedded Web Server*, Rev. 2396C-AVR-05/02. Junio 2002.

[16] Kupris, G.; Kreidl, H.; Gutknecht, N.; Lill, D.; Braun, N., *Implementation of a UDP/IP (User Datagram Protocol/Internet Protocol) Stack on HCS12 Microcontrollers*, AN2304/D, Motorola. Octubre 2002.

[17] Richey, R.; Humberd S., *Embedding PICmicro Microcontrollers in the Internet*, AN731, Microchip Technology Inc., Enero 2002.

[18] Gibbs, M.,*Internet Group Management Protocol*, Riverstone Advanced Technical Paper Series, Riverstone Networks, Enero 2003.

[19] G. Wright and W. Stevens. *TCP/IP Illustrated Volume 2*. Ed. Addison-Wesley. Reading, MA. 1995.

[20] Bentham, J. *TCP/IP Lean: Web Servers for Embedded Systems*, Segunda Edición, CMP Books, 560 págs., Marzo 2002.

[21] Dunkels, A., *The uIP Embedded TCP/IP Stack*. Disponible en `http://www.sics.se/~adam/uip/`

[22] National Semiconductor. *DP8390 Network Interface Controller: An Introductory Guide*. Application Note 475. 1993.

[23] Realtek Semiconductor. *Realtek Full-Duplex Ethernet Controller with Plug and Play Function*. Documento 8019AS.doc. 2001.

[24] Comer, D., *Redes globales de información con internet y tcp/ip*, 3ra edición, Ed. Pearson Educación, 621 págs., México, 1996.

[25] . Nordic Semiconductor. *Single chip 2.4 GHz Transceiver*. Rev. 1.1. Documento nrf2401.pdf. 2004.

[26] . Laipac Technology. *TRF-2.4G Reference Guide*. Rev. 1.01. Documento RF-24G.pdf. 2004.

[27] Microchip Technology Inc. *Using the PICmicro® SSP for Slave I2C Communication*. AN734. Documento DS00734A. 2000.

[28] Microchip Technology Inc. *PIC18F2525/2620/4525/4620 Data Sheet*. Documento DS39626B. 2004.

[29] Microchip Technology Inc. *24AA512/24LC512/24FC512 Data Sheet*. Documento DS21754F. 2005.

[30] Maxim/Dallas Semiconductor. *DS1338 I 2C RTC with 56-Byte NV RAM*. REV: 091404. Maxim Integrated Products. 2004.

[31] National Semiconductor. *LM94021 Multi-Gain Analog Temperature Sensor*. Documento DS201086. 2005.

[32] Cypress Semiconductor. *Calculating Battery Life in WirelessUSB Systems*. Rev. 0.A. 2004.

[33] Pawlan, M.; Dodda S. *Signed Applets, Browsers, and File Access*, Sun Microsystems. Disponible en: `http://java.sun.com/developer/technicalArticles/Security/Signed/`