Tecnura
ISSN: 0123-921X
tecnura@udistrital.edu.co
Universidad Distrital Francisco José de Caldas
Colombia

López Sarmiento, Danilo Alfonso; Garzón Romero, Herney Arturo
Diseño e implementación de IPv6 en un sistema embebido
Tecnura, vol. 17, núm. 37, julio-septiembre, 2013, pp. 167-176
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia

Diseño e implementación de IPv6 en un sistema embebido

IPv6 within an embedded system – design and implementation

Danilo Alfonso López Sarmiento
Ingeniero Electrónico, Magíster en Teleinformática. Docente e investigador de la Universidad Distrital Francisco José de Caldas. Bogotá, Colombia.
Contacto: dalopezs@udistrital.edu.co

Herney Arturo Garzón Romero
Ingeniero Electrónico. Investigador de la Universidad Distrital Francisco José de Caldas. Bogotá, Colombia. Contacto: heagarzonrr@correo.udistrital.edu.co

Fecha de recepción: 28 de septiembre de 2012
Clasificación del artículo: Estudio de Caso
Fecha de aceptación: 21 de mayo de 2013
Financiamiento: Universidad Distrital Francisco José de Caldas

***Palabras clave*:** descubrimiento de vecino, ethernet, IPv6, MAC, pila, sistema embebido.

***Key words*:** neighbor discovery, ethernet, IPv6, MAC, stack, embedded system.

RESUMEN
En este artículo se propone un procedimiento para la implementación de protocolos de nueva generación en un sistema embebido. El objetivo es demostrar que este se puede conectar a Internet mediante IPv6, dando a conocer los aspectos necesarios para su soporte y realizando la implementación de un servidor web como muestra de su funcionamiento. Se eligió un microcontrolador que permite el uso de Ethernet (Coldfire V1 de Freescale), FreeRTOS para facilitar su manejo y el stack de protocolos uIP. Se encontró que es necesaria la implementación de un algoritmo adicional para el manejo de la capa de enlace y que no hay un estándar establecido ni metodología para su desarrollo en los distintos microcontroladores, por lo cual debe caracterizarse el hardware usado, considerar la inclusión de ciertos archivos y garantizar el desempeño de cada una de las capas, teniendo en cuenta el stack de protocolos usado.

ABSTRACT
This paper presents a procedure for the implementation of new-generation protocols in an embedded system. The aim is to show that such an implementation can be connected to the Internet via IPv6. This, for its part, reveals some necessary aspects for supporting the procedure and for actual implementation of a web server (as proof of successful operation). A microcontroller that allows the use of Ethernet (FreescaleColdfire V1) was selected to facilitate handling FreeRTOS and the uIP protocol stack. We found it necessary to write an additional algorithm for the management of the link layer. Since there is no established

standard or methodology for these developments in different microcontrollers, characterization must be achieved depending on the particular hardware being used, including certain files that ensure the performance of each of the layers regarding the protocol stack in question.

1. INTRODUCCIÓN

Con la entrega del último bloque de direcciones IPv4 en febrero de 2011 por parte de la IANA [1], en la actualidad se realiza una transición a la nueva versión del protocolo IP, denominada IPv6, que permite mayor cantidad de estas direcciones disponibles para ser asignadas a usuarios, mejor soporte y prioridad para entrega de datos en tiempo real, soporte para gran variedad de dispositivos además de computadores, buen manejo de tráfico a pesar del crecimiento de la red, entre otras características.

Por otro lado, en la actualidad se está incrementando el uso de los llamados sistemas embebidos en distintas áreas por razones de portabilidad y comodidad, entre otras. Sus aplicaciones van desde el uso doméstico hasta el industrial, caracterizándose por su enfoque en una determinada tarea, menor consumo de energía y fácil fabricación, permitiendo el desarrollo de aplicaciones como celulares, lavadoras, sistema de frenos, etc. En este sentido se puede argumentar que existe la necesidad de generar propuestas de investigación y desarrollo que permitan la integración de IPv6 en sistemas embebidos. Por tal razón, el presente paper pone a disposición de la comunidad académica una propuesta para incluir en los sistemas embebidos el soporte para la navegación en internet con la versión 6 del protocolo IP; de esta manera, se espera contribuir de una manera práctica en el desarrollo de esta importante línea que está en pleno desarrollo. Los elementos fundamentales que cimientan el documento son:

- Herramientas utilizadas.
- Inclusión de archivos necesarios y diseño para el soporte de IPv6 en el microcontrolador.
- Algoritmo de procesamiento de paquetes.
- Implementación en red.
- Análisis y resultados.

2. ESTADO DEL ARTE

De acuerdo con Dunkels y Vasseur [2], IP resulta ser una opción para la intercomunicación de sistemas embebidos y el control de estos debido a varias características, entre las que destacan:

- Interoperabilidad.
- Arquitectura evolutiva y versátil.
- Estabilidad y universalidad de la arquitectura.
- Escalabilidad.
- Configuración y manejo.

Por ello se han desarrollado stacks de protocolos TCP/IP con funcionalidades básicas, considerando el espacio de memoria.

Específicamente en este sentido, Adam Dunkels ha creado dos stacks lwIP [3] y uIP [4], con carácter de distribución libre. El primero está diseñado para microcontroladores de 32 bits, sin embargo no están definidas todas las funciones necesarias para el soporte de IPv6. El segundo se diseñó para microcontroladores de 8 y 16 bits, y permite el soporte para IPv6 e IPv4.

Gran parte de los desarrollos que se pueden encontrar están realizados en IPv4 [5] - [8], algunos autores han realizado modificaciones a lwIP, teniendo en cuenta que la manera en que está escrito su código permite desarrollos con dual stack. Yihua Huang, Zhiping Jia, Xin Li y Hui Xia presentan EIPv6 [9], el cual es un stack desarrollado en base a lwIP con soporte para IPv6, IPv4, ICMP, TCP, ARP y Neighbor Discovery. Debido a su simplicidad, EIPv6 no permite la fragmentación de paquetes y no incluye IPsec, entre otras características. Puede funcionar sobre un sistema operativo o sin él, ocupando un espacio en RAM entre los 24 a 32 KB y 6 a 8 KB en ROM. No se ha publicado su código fuente.

El stack uIP inicialmente se realizó para IPv4, pero dada la necesidad de IPv6 sus desarrolladores incluyeron más módulos, sin embargo esta última versión no se encuentra de manera independiente; está contenida en el sistema operativo contiki [10] [11]. Por otro lado, no permite desarrollos dual stack, ya que el código de IP está escrito bajo compilación condicional, de tal manera que una vez seleccionado IPv6 o IPv4, el sistema embebido no podrá cambiar la versión en tiempo de ejecución.

Gran parte de las aplicaciones que se pueden encontrar con uIP usan IPv4. Wei Li y Ke-Bin Jia [12] realizan la implementación de un servidor web que permite la reproducción de videos en distintos formatos, con lo cual concluyen que este tipo de aplicación permite el manejo de datos en tiempo real, a diferencia de los complejos servidores actuales. Presentan además una metodología para el establecimiento de la comunicación en internet usando las distintas funciones del stack.

Brown, Ferreira y Figueredo [13] sugieren los aspectos para tener en cuenta para la implementación de uIP en un microcontrolador, dentro de los que cabe destacar que es necesario el desarrollo de la aplicación y un controlador para el manejo de la capa de enlace de datos.

Algunos miembros de Cisco y Atmel [14], en colaboración con Adam Dunkels, presentan el stack uIP con soporte para IPv6. Considerando su aplicación en redes de sensores IP, emplean el microcontrolador Atmega1284P y el sistema operativo Contiki. La implementación comprende 32 kB de espacio en memoria incluyendo los controladores para el manejo de capa de enlace de datos.

Otro trabajo con soporte de IPv6 es un proyecto realizado por Guenther Hoelzl con Arduino [15], sin embargo el uso de este software limita su uso en otras arquitecturas de hardware.

3. METODOLOGÍA

Con el fin de dar a entender la propuesta, se presentan a continuación las herramientas de hardware y software utilizadas y posteriormente, las consideraciones realizadas en el diseño, como son las modificaciones al código fuente y el algoritmo principal para la aplicación.

Para el desarrollo de este proyecto se usó el microcontrolador MCF51CN128 [17] de la familia Coldfire V1 marca Freescale, contenido en el kit Tower System [18], que facilitó el desarrollo del prototipo con conexión a internet. Fue utilizado como punto de partida el demo que suministra FreeRTOS [19] para este sistema. La aplicación originalmente funciona como un servidor web, el cual tiene almacenada una página a la que se puede acceder desde el navegador internet de un PC por medio de una conexión Ethernet.

Este demo utiliza el stack uIP para brindar el soporte TCP/IP, pero solo funciona sobre IPv4. En la figura 1 es mostrado un diagrama con los protocolos soportados, junto con aquellos incluidos en el desarrollo de esta investigación (fondo gris, letra blanca). Para poder realizar esto, es necesario adicionar algunos archivos, crear algunas funciones y estructuras.

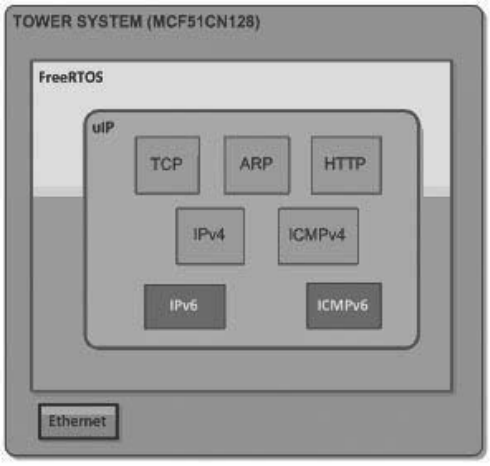

Figura 1. Esquema que representa el software y los protocolos implementados en el sistema propuesto.
Fuente: elaboración propia.

3.1 Archivos de código incluidos

Como ya se mencionó, el demo de FreeRTOS tomado como base para el desarrollo del proyecto no tiene soporte para IPv6, mientras que el stack uIP contenido en el sistema operativo Contiki [10] brinda algunos archivos que contienen las funciones, estructuras y variables necesarias para el procesamiento de paquetes con esta versión de IP. A continuación se dan a conocer los archivos tenidos en cuenta para el soporte de los protocolos ICMPv6 e IPv6:

- uip-ds6.c
- uip-ds6.h
- uip-icmp6.c
- uip-icmp6.h
- uip-nd6.c
- uip-nd6.h
- uip6.c

Dentro del conjunto de archivos uip-ds6 se encuentra la configuración para todas las tablas: caché de vecinos, tabla de enrutamiento, las listas por defecto del router, de prefijos, de direcciones de unidifusión, multidifusión y anycast.

En uip-nd6 se tienen las estructuras correspondientes a los encabezados de los diferentes mensajes "Neighbor Discovery" y las funciones usadas para enviar y procesar estos.

En los archivos uip-icmp6, se encuentran los encabezados de los mensajes ICMPv6, constantes y funciones necesarias para generar y/o responder a estos, los cuales están basados en la RFC 4861 [20] y RFC 4443 [21] .

Se puede decir que uip6.c es el archivo más importante para el soporte de IPv6, porque en él se determina qué tipo de paquete se ha recibido y de acuerdo a ello se envía a las funciones respectivas de procesamiento. En la figura 2 se muestra sobre qué protocolos trabajan algunos de los archivos incluidos, adicionalmente se pueden ver algunas funciones de procesamiento que se aclararán más adelante.

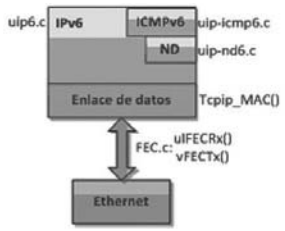

Figura 2. Relación entre archivos, funciones y protocolos
Fuente: elaboración propia.

3.2 Modificaciones de código realizadas

Con el fin de poder establecer la conectividad a nivel de capa de enlace, fue necesario incluir en los archivos de IPv6, los 14 bytes correspondientes a la dirección destino, origen y tipo del encabezado, de acuerdo con la RFC 2464 [22]. Por ello se creó la estructura mostrada en la f dentro del archivo uip_arp.h. La dirección destino y fuente pertenecen a una instancia de la estructura uip_eth_addr, que está definida como un arreglo de 6 bytes (figura 3). El tipo es declarado como una constante simbólica (figura 4) al igual que los 14 bytes del encabezado (figura 5).

```
struct uip_eth_hdr {

struct uip_eth_addr dest; //dirección destino

struct uip_eth_addr src; //dirección fuente

u16_t type; // Tipo

}PACK_STRUCT_END;
```

Figura 3. Estructura encabezado Ethernet
Fuente: elaboración propia.

```
typedef struct uip_eth_addr {

u8_t addr[6]; //variable que almacena la dirección

} uip_eth_addr;
```

Figura 4. Estructura uip_eth_addr
Fuente: elaboración propia.

```
// valor del campo tipo para IPv6

#define UIP_ETHTYPE_IP6 0x86dd


//longitud en bytes del encabezado Ethernet.

#define UIP_LLH_LEN 14
```

Figura 5. Tipo y longitud del encabezado Ethernet
Fuente: elaboración propia.

Para generar el encabezado Ethernet en IPv6 se ha creado la función tcpip_MAC(). Al invocarla, esta busca en la tabla de vecinos (neighbors) la dirección MAC correspondiente a una determinada IP. En la figura 6 se presenta el segmento de código desarrollado para laasignación de la dirección MAC.

```
void tcpip_MAC(void)

{

//verifica si la dirección IP destino está contenida dentro de la tabla de vecinos

if((nbr=uip_ds6_nbr_lookup(&UIP_IP_BUF>destipaddr)) !=NULL)

 {

// asigna a la variable la dirección MAC destino //contenida en el registro de memoria

memcpy(&UIP_IP_LLH->dest.addr, &(nbr->lladdr), sizeof(uip_lladdr));

// asigna a la variable la dirección MAC fuente //contenida en el registro de memoria

memcpy(&UIP_IP_LLH->src.addr,&uip_lladdr, sizeof(uip_lladdr));

//asigna el valor del campo tipo a la variable del //paquete

UIP_IP_LLH->type = HTONS (UIP_ETHTYPE_IP6);

 } else

 {

// ya que no está la IP destino en la tabla se coloca la dicección MAC del gateway

 nbr=uip_ds6_nbr_lookup(&uip_draddr);

memcpy(&UIP_IP_LLH->dest.addr, &(nbr->lladdr), sizeof(uip_lladdr));

memcpy(&UIP_IP_LLH->src.addr, &uip_lladdr, sizeof(uip_lladdr));

UIP_IP_LLH->type = HTONS (UIP_ETHTYPE_IP6);

}

}
```

Figura 6. Función tcpip_MAC()
Fuente: elaboración propia.

Como se puede apreciar, el algoritmo primero verifica si la dirección de destino (IPv6) tiene algún registro en la tabla, si esto sucede, copia la MAC que tiene relacionada con este, después copia la IP propia y finalmente ingresa el tipo. Si por el contrario no la encuentra, entonces coloca la MAC del Gateway.

3.3 Procesamiento de paquetes

El algoritmo diseñado y desarrollado para el procesamiento de paquetes es mostrado en el diagrama de flujo de la figura 7.

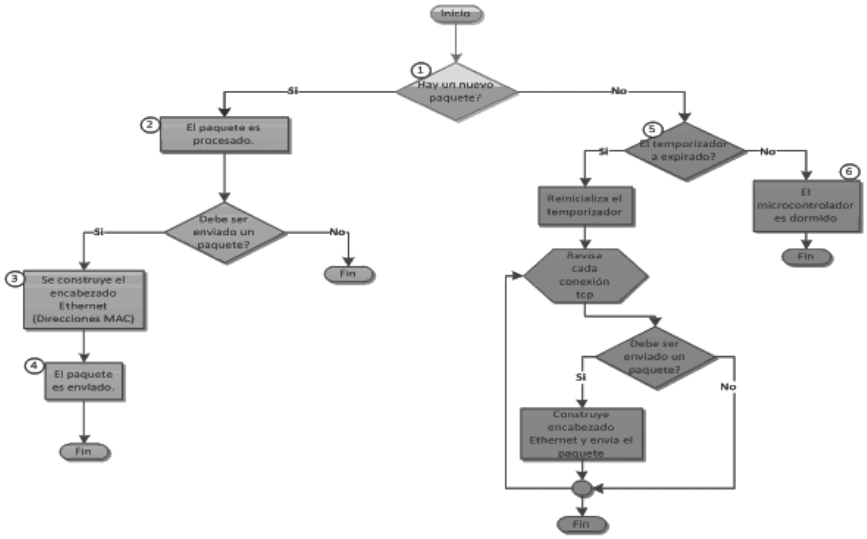

Figura 7. Procesamiento de paquetes
Fuente: elaboración propia.

Para tener una mejor comprensión de la ejecución y las funciones que se encuentran involucradas en dicho algoritmo, se hace una descripción más detallada a continuación, de acuerdo a la numeración que se muestra en el gráfico anterior:

Se debe verificar si ha llegado un nuevo paquete para el microcontrolador, la forma de hacerlo es por medio de las variables uip_len y uip_buf, que deben tener un valor mayor a cero y diferente de cero respectivamente. En la primera se almacena la longitud del paquete que recibe el sistema embebido, en la segunda se encuentra el apuntador de memoria donde es ubicado el paquete. Estas dos variables son un puente fundamental entre los datos recibidos por el microcontrolador y los procesados por el stack, ya que este último procesa aquellos a los cuales apunta uip_buf y cuando requiere enviar algún paquete utiliza estas dos variables, para indicar al Coldfire V1 qué datos debe tener en cuenta y cuál es el tamaño de estos.

En esta etapa del proceso, se realiza el llamado de la función uip_process(), que se encarga de procesar todos los paquetes en el stack. Muchas otras funciones se invocan dentro de la misma con el fin de analizar tipos de paquetes específicos, como es el caso de paquetes ICMP, TCP, UDP, etc. También se verifica la integridad del paquete por medio de la revisión de la suma de confirmación, tamaño máximo del paquete, entre otras características.

Como ya se mencionó en el ítem B, es necesario incluir el encabezado Ethernet mediante la función tcpip_MAC().

En este paso se realiza el envío del paquete, haciendo uso de la función vFECTx (), la cual lo tiene almacenado en la variable uip_buf. Cabe aclarar que para la recepción se usa ulFECRx(), dichas funciones son el medio de comunicación que existe entre los datos que procesa el stack y los que serán enviados por el microcontrolador. Como se puede apreciar en la figura 2, las variables uip_len y uip_buf se escriben al recibir un paquete y son leídas cuando el sistema embebido requiere enviarlo.

Dado el caso de que no se hayan encontrado nuevos paquetes, se realiza una revisión periódica de las conexiones TCP, para verificar que el dispositivo aun está conectado a la red.

Mientras haya conexión pero no se reciban nuevos paquetes, el microcontrolador entra en estado de bajo consumo hasta la llegada de un nuevo mensaje, esto se realiza mediante el uso de un semáforo de FreeRTOS.

3.4 Implementación

Para comprobar el funcionamiento de IPv6 sobre el sistema embebido de acuerdo a lo planteado anteriormente, se ha realizado un montaje que permite hacer un control de temperatura de tipo ON/OFF de manera remota, con el uso de módulos Zigbee como lo muestra el diagrama de la figura 8.

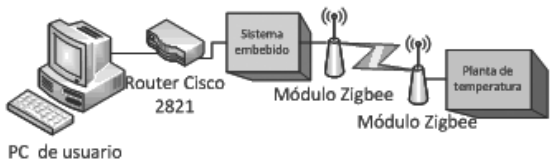

Figura 8. Modelo del sistema de control de temperatura remoto
Fuente: elaboración propia.

Se ha utilizado un router cisco 2821, este dispone de dos interfaces Gigabit Ethernet. En una de ellas es conectado el sistema embebido (Tower System) por medio de un cable Ethernet, en la otra un PC. Este último tiene una tarjeta de red con puerto Ethernet y un navegador de Internet, el montaje real se puede apreciar en la figura 9.

Desde el PC que se encuentra conectado al router se accede a la página web almacenada en el sistema embebido, digitando en la barra de direcciones su IP y adicionándole /index.shtml, para que quede de la siguiente manera [2001:3::2]/index.shtml. Después de que la página ha sido cargada en el navegador del PC, este actualiza el campo de "temperatura actual" cada segundo.

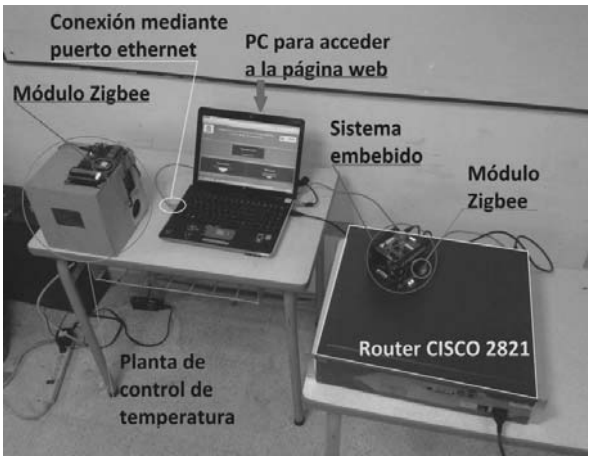

Figura 9. Sistema real
Fuente: elaboración propia.

La topología mostrada en la figura 10 muestra la configuración de direcciones realizada, por lo cual se omite la parte de comunicación inalámbrica con los módulos Zigbee. Como se puede ver, el sistema embebido y el computador del usuario se encuentran conectados en redes distintas.

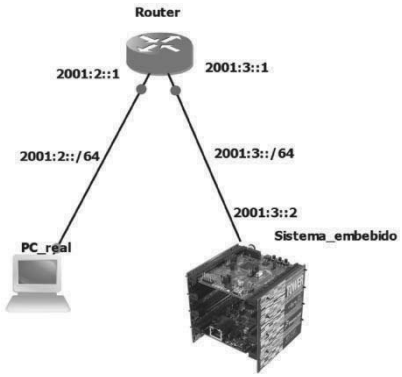

Figura 10. Topología implementada
Fuente: elaboración propia.

Con el objetivo de distinguir los paquetes generados por el sistema embebido de los que puede generar cualquier otra máquina dentro de la red, se ha incluido el valor hexadecimal "CA" en el campo clase de tráfico del encabezado IPv6, como se puede notar en la captura hecha en Wireshark mostrada en la figura 11.

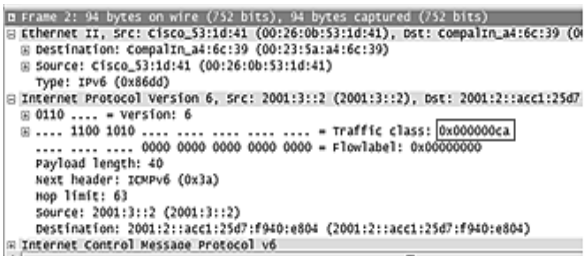

Figura 11. Campo modificado
Fuente: elaboración propia.

4. ANÁLISIS Y RESULTADOS

Para verificar el desempeño del sistema embebido en lo referente al flujo de datos, se ha realizado una gráfica de cantidad de paquetes enviados, teniendo como filtro aquellos que son TCP y que tienen un valor distinto de cero en el campo clase de tráfico (en este caso el valor CA) como se muestra en la figura 12.

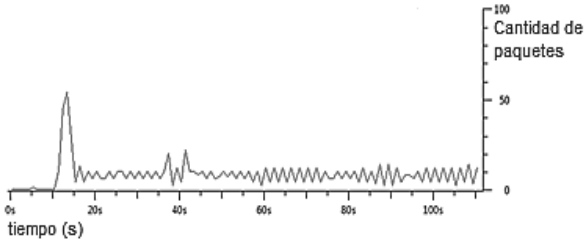

Figura 12. Flujo de datos
Fuente: elaboración propia.

Como puede notarse, hay un continuo flujo de datos, de acuerdo con las peticiones realizadas cada segundo por la página web.

En el intervalo de tiempo considerado (134 segundos) se puede ver que de 1083 paquetes enviados solo dos están malformados, como se muestra en la figura 13.

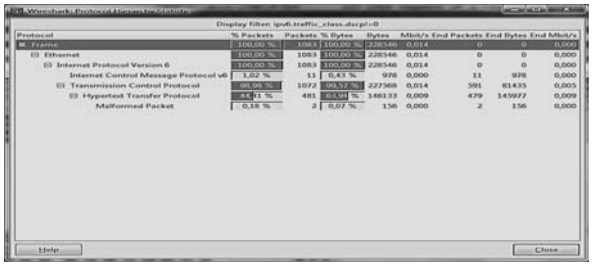

Figura 13. Estadística de paquetes
Fuente: elaboración propia.

En relación con el consumo de memoria en el microcontrolador y de acuerdo con el compilador del software Codewarrior, se obtuvieron los resultados mostrados en la tabla 1. Solo se han tenido en cuenta los archivos del stack necesarios para el soporte de IPv6, de tal manera que se tiene un estimado de 59 KB, superior a la implementación mínima de EIPv6 [9], sin embargo hay que tener en cuenta que en esta aplicación se encuentra una página web con varias características, almacenada en el microcontrolador, lo que hace que HTTP tenga un mayor consumo de memoria (tabla 1).

Tabla 1. Consumo de memoria en bytes.

Módulo	Flash (Bytes)	RAM (Bytes)
HTTP	41 550	74
TCP/IP	6672	3384
ND (Neighbor Discovery)	5968	897
ICMPv6	1044	16
Total	55 234	4371

Fuente: elaboración propia.

5. CONCLUSIONES

A pesar de que existen otros stacks para la implementación de TCP/IP en microcontroladores, la ventaja que tiene uIP es el tipo de licenciamiento, que permite tener un acceso completo a su código, posibilitando mejoras, correcciones y modificaciones, esto también gracias a que su código se encuentra bien organizado y documentado.

Para la realización de cualquier modificación en el código debe tenerse en cuenta la compilación condicional, que permite la ejecución de este de acuerdo a la versión seleccionada.

La implementación del stack con soporte para IPv6 en FreeRTOS, permite que se puedan realizar distintas aplicaciones en base a este, garantizando la conectividad en las redes de nueva generación, mediante el uso de software libre.

Para implementar un stack de comunicación con internet, debe incluirse un archivo de funciones para la configuración del controlador Ethernet del dispositivo con el que se esté trabajando. Esto dependerá de la familia, capacidades y fabricante del dispositivo, razón por la cual el primer paso en cualquier metodología de desarrollo es establecer las características del dispositivo y desarrollar las herramientas de software necesarias para garantizar la conectividad en capa de enlace; en este caso se utilizó el archivo FEC contenido en FreeRTOS, además se tuvo que diseñar un algoritmo para formar el encabezado Ethernet.

El espacio en memoria ocupado por el stack uIP con soporte para IPv6 es de 59 KB, superior en algunos casos a otras implementaciones. No obstante, las diferencias en tamaño, además de las características incluidas en el módulo, dependen del diseño de la página web y el aplicativo y las funcionalidades que se requieran incluir.

REFERENCIAS

[1] R. Sandoval, 1 Abril 2012. [En línea]. Disponible: http://www.mintic.gov.co/index.php/mn-news/197-20110624. [Último acceso: 1 Mayo 2012].

[2] A. Dunkels, "Interconnecting Smart Objects with IP", *Interconnecting Smart Objects with IP*, Oxford: Elsevier, 2010.

[3] A. Dunkels, "Design and Implementation of the lwIP", Swedish Institute of Computer Science, Stockholm, 2001.

[4] A. Dunkels, *The uIP Embedded TCP/IP Stack*, Swedish Institute of Computer Science, Stockholm, 2006, p. 89.

[5] P. Alcantara, "Email Client Using MCF-51CN Family and FreeRTOS", RTAC Americas, México, 2009.

[6] P. Alcantara, "FTP Server Using MCF-51CN Family and FreeRTOS", RTAC Americas, México, 2009.

[7] P. Alcantara, "Serial-to-Ethernet Bridge Using MCF51CN Family and FreeRTOS", RTAC Americas, México, 2009.

[8] P. Alcantara, "Web Server Using the MCF-51CN Family and FreeRTOS", RTAC Americas, México, 2009.

[9] Y. Huang, Z. Jia, X. Li and H. Xia, "EIPv6: A Reduced IPv6 Protocol Stack for Embedded Systems", *3rd International Conference on Advanced Computer Control (ICACC 2011)*, Jinan, 2011.

[10] A. Dunkels, B. Grönvall and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors", *Proceedings of the First IEEE Workshop on Embedded Networked Sensors* (Emnets-I), Tampa, Florida, USA, 2004.

[11] Thingsquare, 4 abril 2011. [En línea]. Disponible: http://www.contiki-os.org/. [Último acceso: 20 marzo 2012].

[12] W. Li, and K. B. Jia, *Implementation of Embedded Web Server*, Beijing: College of Electronic Infonnation & Control Engineering 2007

[13] O. Salcedo, D. López y A. Ríos, "Desempeño de la calidad del servicio (QoS) sobre IPv6", *Tecnura*, vol. 15, no. 28, pp. 32-41, Junio 2011.

[14] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne and A. Dunkels, "Making Sensor Networks IPv6 Ready", *Proceedings of the Sixth ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008)*, Raleigh, North Carolina, USA, 2008.

[15] G. Hoelzl, "ipv6EtherShield - IPv6 for the Arduino platform", 20 agosto 2011. [En línea]. Disponible: http://sites.google.com/site/ghoelzl/ipv6ethershield. [Último acceso: 20 Octubre 2011].

[16] Real Time Engineers, "FreeRTOS", FREERTOS.ORG, 30 abril 2003. [En línea]. Disponible: http://www.freertos.org/a00017.html. [Último acceso: 05 mayo 2012].

[17] Freescale Semiconductor, Inc., "Freescale.com", 13 enero 2012. [En línea]. Disponible: http://cache.freescale.com/files/32bit/doc/ref_manual/MCF51CN128RM.pdf. [Último acceso: 5 mayo 2012].

[18] F. Semiconductor, "Freescale Semiconductor", 13 enero 2012. [En línea]. Disponible: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=TWR-MCF51CN&tid=m32TWR. [Último acceso: 25 abril 2012].

[19] Real Time Engineers Ltd, 15 abril 2012. [En línea]. Disponible: http://www.freertos.org/. [Último acceso: 29 abril 2012].

[20] T. Narten, E. Nordmark, W. Simpson y H. Soliman, "Neighbor Discovery for IP version 6", *The IETF Trust, Research Triangle Park*, 2007.

[21] A. Conta and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6)", Concord, MA, 1998.

[22] M. Crawford, "Transmission of IPv6 Packets over Ethernet Networks", *The Internet Society*, Batavia, IL, 1998.

[23] G. Galeano, *Programación de sistemas embebidos en C*, 1 ed., Bogotá: Alfaomega, 2009, p. 544.