ANEXO 1.
CÓDIGO FUENTE DE LOS WASPMOTES.

En este están los archivos de código fuente de los Waspmotes utilizados en el proyecto para compilarlos con Waspmote IDE.

A1.1. NODO 0041: Ctra. Utrera.

```
/*------------------------------------------------------------------------
 * - Nombre: NODO 0041 - CTRA. UTRERA
 * - Autor: Manuel Jesús Cano Moreno
 *------------------------------------------------------------------------
*/

#define TAM_TABLA 3

// Tipos de MENSAJE
#define MSG_KEEPALIVE  "KEEPALIVE"
#define MSG_ALARMA     "ALARMA"
#define MSG_EST_ENL    "ESTADO_ENLACE"

// Tipos de ALARMA
#define ALM_CAIDA "CAIDA_ENLACE"
#define ALM_TEMP  "TEMP_ALTA"
#define ALM_BATT  "BATERIA_BAJA"
#define ALM_DC    "DUTY_CYCLE"

// Tipos de KEEPALIVE
#define KEEPALIVE_NODE 0
#define KEEPALIVE_LINK 1

#define NUM_SALTOS 0

packetXBee* paq_sent;

// Datos de tiempo
unsigned long t; // Instante actual (valor absoluto)
unsigned long T; // Instante de inicio de periodo (valor absoluto)

// Mensaje a enviar
char data[200];

// -- DIRECCIONAMIENTO Y ENCAMINAMIENTO ----------------------------------------

char NAOrigen[6] = "0041";
char NADestino[6] = "0000";
char NABroadcast[6] = "FFFF";

// Tabla de direcciones MAC
char tablaMAC[TAM_TABLA][18] = {"0013A20040581DBC",
                                "0013A20040581DC1",
                                "0013A200403AB314"};

// Tabla de direcciones NA
char tablaNA[TAM_TABLA][6]  = {"0042",
                               "0043",
                               "0000"};

// Indice que indica el salto siguiente (Next Hop) elegido
uint8_t ind = 0;

// Indice que se usa para los reintentos de envio
uint8_t i;
// Indice para los reenvios de alarma
uint8_t j;
// Indice para el buffer (FIFO) en el que se guardan

// ------------------------------------------------------------------------

// Campos del mensaje de KEEPALIVE
char *fecha_hora;
```

```
uint8_t temp;
char aux_temp[6];
uint8_t batt_level;
uint8_t dc_usado;

// -- CADENAS AUXILIARES ------------------------------------------------------
// Cadenas auxiliares para guardar los campos del mensaje
char dia_aux[15] = {0};
char fecha_hora_aux[40] = {0};
char NAOrigen_aux[6] = {0};
char NADestino_aux[6] = {0};
char tipoMens_aux[15] = {0};
char temp_aux[6] = {0};
char batt_aux[6] = {0};
char dc_aux[6] = {0};
char num_saltos_aux[6] = {0};
long int num_salt_longint;

// ------------------------------------------------------------------------------

// Puesto que los mensajes de alarma por CAIDA_ENLACE son muy importantes,
// se crea un buffer para almacenarlos y reenviarlos cuando se actualice la tabla
// de encaminamiento
char buffer_msg_alarma[TAM_TABLA][200];
uint8_t pos = 0;

/* ---------------------------------------------------------------------------------------
 *  Extrae los campos del mensaje. (NAOrigen, NADestino y tipoMensaje (KEEPALIVE, ...)
 * ---------------------------------------------------------------------------------------
 */

void extraerCampos(char *dia, char *fechaHora, char *tipoMens, char *NAOrig, char *NADest,
                   char *temper, char *bater, char *duty, char *numSaltos)
{
  uint8_t i;
  char *p; // Puntero que recorre el mensaje para extraer los datos
  p = data;

  for (i=0; i<9; i++)
  {
    // Este bucle extrae los campos sin ponerle el '\0' al final de cada uno
    while(*p != ',')
    {
      if(i==0)
      {
         if (*p != '#')
         {
           *dia = *p;
           dia++;
         }
      }
      if (i == 1)
      {  *fechaHora = *p;
          fechaHora++;
      }
      if (i == 2)
      {   *tipoMens = *p;
          tipoMens++;
      }
      if (i == 3)
      {   *NAOrig = *p;
          NAOrig++;
      }
      if (i == 4)
      {   *NADest = *p;
          NADest++;
      }

      // En el caso de que sea un mensaje de KEEPALIVE, extraemos los campos que correspondan.
      if ( (i > 4) && (Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) )
      {
          if (i == 5)
          {   *temper = *p;
              temper++;
          }
          if (i == 6)
          {   *bater = *p;
              bater++;
          }
          if (i == 7)
          {   *duty = *p;
              duty++;
          }
          if (i == 8)
```

```
          {
              if (*p != '#')
              {
                *numSaltos = *p;
                numSaltos++;
              }
              else break; // Si el caracter es '#', quiere decir, que hemos llegado al final del mensaje
                          // salimos del bucle while
          }
      }
      p++; // Se mueve un caracter el puntero que recorre el mensaje
    }
    p++; // Esto sirve para que "salte" la coma y pase al siguiente campo

    // Añadimos los '\0' de finalizacion del campo
    if(i==0)
    {  *dia = '\0';
    }
    if (i == 1)
    {  *fechaHora = '\0';
    }
    if (i == 2)
    {  *tipoMens = '\0';
    }
    if (i == 3)
    {  *NAOrig = '\0';
    }
    if (i == 4)
    {  *NADest = '\0';
    }
    if ( (i > 4) && (Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) )
    {
        if (i == 5)
        {   *temper = '\0';
        }
        if (i == 6)
        {   *bater = '\0';
        }
        if (i == 7)
        {   *duty = '\0';
        }
        if (i == 8)
        {   *numSaltos = '\0';
        }
    }
  }
}

/* ------------------------------------------------------------------------
 *   Funcion que genera un mensaje de KEEPALIVE en el nodo
 * ------------------------------------------------------------------------
 */
void crearMensajeKeepAlive(uint8_t tipo_MSG_KA)
{
    // MENSAJE KEEPALIVE. Tipo NODE
    // + -- + ----------- + --------- + -------- + --------- + ------ + ------ + ------ + ---------- + ---- +
    // | ## | Fecha/Hora  | KEEPALIVE | NAOrigen | NaDestino |  Temp  |  Batt  |   DC   | Num Saltos | #### |
    // + -- + ----------- + --------- + -------- + --------- + ------ + ------ + ------ + ---------- + ---- +

    // MENSAJE KEEPALIVE. Tipo LINK  (Se envía por Broadcast a los nodos adyacentes)
    // + -- + ---- + --------- + -------- + ------------ + --- + --- + --- + --- + --- + ---- +
    // | ## |      | KEEPALIVE | NAOrigen | NA_Broadcast |     |     |     |     |     | #### |
    // + -- + ---- + --------- + -------- + ------------ + --- + --- + --- + --- + --- + ---- +

    // Si el mensaje de KEEPALIVE es de tipo NODO, se rellenan los campos.
    if (tipo_MSG_KA == KEEPALIVE_NODE)
    {
      // Obtenemos el "timestamp": Fecha/Hora
      fecha_hora = RTC.getTime();

      // Obtenemos la temperatura del xBee
      temp = RTC.getTemperature();

      // Obtenemos el nivel de bateria
      batt_level = PWR.getBatteryLevel();

      // Obtenemos el porcentaje de duty cycle disponible que ha sido usado
      xbee868.getDutyCicle();
      dc_usado = xbee868.dutyCicle;

      sprintf(data,"##%s,%s,%s,%s,%d,%d,%d,%d####", fecha_hora, MSG_KEEPALIVE, NAOrigen, NADestino, temp, batt_level, dc_usado,
NUM_SALTOS);
    }
    // Si el mensaje de KEEPALIVE es de tipo LINK, los campos se dejan vacíos y la dirección de destino se pone a FFFF.
```

```
    else if (tipo_MSG_KA == KEEPALIVE_LINK)
    {
      sprintf(data,"##,,%s,%s,%s,,,,####", MSG_KEEPALIVE, NAOrigen, NABroadcast);
    }
}



/* ---------------------------------------------------------------------
 *   Funcion que genera un mensaje de ALARMA en el nodo
 * ---------------------------------------------------------------------
 */
void crearMensajeAlarma(uint8_t tipoAlarma)
{
    // MENSAJE ALARMA.                                                    { Campo opcional }
    // + -- + ----------- + -------- + -------- + --------- + ------------- + ----------- + ---- +
    // | ## | Fecha/Hora  |  ALARMA  | NAOrigen | NaDestino |  TIPO_ALARMA  | Radioenlace | #### |
    // + -- + ----------- + -------- + -------- + --------- + ------------- + ----------- + ---- +

    /* tipoAlarma:
       0 = CAIDA_ENLACE
       1 = TEMP_ALTA
       2 = BATERIA_BAJA
       3 = DUTY_CYCLE
    */

    // Obtenemos el "timestamp": Fecha/Hora
    fecha_hora = RTC.getTime();

    switch(tipoAlarma)
    {
      case 0:
             // ALARMA por CAIDA_ENLACE
             sprintf(data,"##%s,%s,%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_CAIDA, NAOrigen,
tablaNA[ind]);
             break;
      case 1:
             // ALARMA por TEMP_ALTA
             sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_TEMP);
             break;
      case 2:
             // ALARMA por BATERIA_BAJA
             sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_BATT);
             break;
      case 3:
             // ALARMA por DUTY_CYCLE
             sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_DC);
             break;
      default:
             // Opcion no valida...
             break;
    }
}

/* ---------------------------------------------------------------------------------------
 *  Limpiar el mensaje de datos
 * ---------------------------------------------------------------------------------------
 */

void clearData()
{
  uint8_t k;

  for(k=0;k<200;k++)
    data[k] = '\0';
}

/* ---------------------------------------------------------------------------------------
 *  Configuración inicial del Waspmote
 * ---------------------------------------------------------------------------------------
 */

void setup()
{
  // Inicialización de la librería de xBee
  xbee868.init(XBEE_868,FREQ868M,NORMAL);

  // Encender xBee
  xbee868.ON();

  // Poner potencia al máximo
  xbee868.setPowerLevel(4);

  // Inicializamos el RTC
```

```
  RTC.ON();

  // Meter todo el código de setup y dejar para el final el
  // cálculo de los instantes de tiempo para evitar que la llamada
  // a la funcion loop meta mucho retardo
  T = millis();
}

void loop()
{
  // Obtenemos el instante actual.
  t = millis();

  // Lo primero que se hace es enviar el mensaje de KEEPALIVE
  // (Si han pasado 30 segundos desde el ultimo envio de KEEPALIVE)
  if ( t >= T+30000)
  {
    // Actualizamos el instante de inicio de periodo de 30 seg.
    T = t;

    // ------------------------------------------------------------------
    // PASO 1. Siempre se envia el paquete de KEEPALIVE de tipo Node, haya alarma o no.
    // ------------------------------------------------------------------

    // Set params to send en paquete API (se reserva memoria para el paquete primero)
    paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
    paq_sent->mode=UNICAST;
    paq_sent->MY_known=0;
    paq_sent->packetID=0x52;
    paq_sent->opt=0;
    xbee868.hops=0;
    xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

    // Creamos un mensaje de KeepAlive
    crearMensajeKeepAlive(KEEPALIVE_NODE);

    // Retardo antes de enviar de 1 s
    delay(1000);

    xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
    xbee868.sendXBee(paq_sent);

    // Si hay error de transmisión, se realizan 4 reintentos de envio, con retardos de
    // 250ms, 500ms, 750ms y 1000ms (se va aumentando el retardo linealmente)

    if( xbee868.error_TX )
    {
       i = 0;
       while( (i<4) && (xbee868.error_TX) )
       {
         delay(250*(i+1));
         xbee868.sendXBee(paq_sent);
         i++;
       }
       if (i == 4)
       {
         // Si se llega al máximo de reintentos...
         if(ind < TAM_TABLA-1)
         {
           // Liberamos primero la memoria del paquete anterior de KEEPALIVE
           free(paq_sent);
           paq_sent=NULL;
           // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
           clearData();

           // 1) Se crea el mensaje de ALARMA por CAIDA_ENLACE
           crearMensajeAlarma(0);
           // 2) Se actualiza la tabla de encaminamiento si no se ha llegado a la ultima posicion
           ind++;
           // 3) Se envia el mensaje de alarma
           paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
           paq_sent->mode=UNICAST;
           paq_sent->MY_known=0;
           paq_sent->packetID=0x52;
           paq_sent->opt=0;
           xbee868.hops=0;
           xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

          // Retardo antes de enviar de 1 s
          delay(1000);

          xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
          xbee868.sendXBee(paq_sent);
```

```
      // Puesto que el mensaje de CAIDA_ENLACE es importante, si da error al enviar, se reintenta 4 veces.
      if( xbee868.error_TX )
      {
         j = 0;
         while( (j<4) && (xbee868.error_TX) )
         {
           delay(250*(j+1));
           xbee868.sendXBee(paq_sent);
           j++;
         }
         // Si se supera el limite de reintentos de envio del mensaje de alarma por CAIDA_ENLACE...
         if (j == 4)
         {
           // Si el buffer no está lleno...
           if (pos < TAM_TABLA-1)
           {
             // Se copia el mensaje de alarma en el buffer
             strcpy(buffer_msg_alarma[pos], data);
             // Se incrementa el indice que apunta a la siguiente posicion libre en el buffer (LIFO)
             pos++;
           }
         }
      }

      // Liberar memoria de paquete API de alarma
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();

     } // Si se ha llegado a la última posicion de la tabla de encaminamiento, se vuelve a la primera a ver si se puede...
     else
     {
       ind = 0;
     }

   }
}
// En caso de que el KEEPALIVE se haya mandado bien... se comprueba si hay mensajes de ALARMA por caida de enlace
// pendientes en el buffer y se intentan enviar...
else
{
   // Si hay mensajes en el buffer
   if(pos > 0)
   {
      paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
      paq_sent->mode=UNICAST;
      paq_sent->MY_known=0;
      paq_sent->packetID=0x52;
      paq_sent->opt=0;
      xbee868.hops=0;
      xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

      // Retardo antes de enviar de 1 s
      delay(1000);

      // Ojo! El mensaje a enviar estará en "pos-1", ya que "pos" indica la primera posicion libre del buffer.
      xbee868.setDestinationParams(paq_sent, tablaMAC[ind], buffer_msg_alarma[pos-1], MAC_TYPE, DATA_ABSOLUTE);
      xbee868.sendXBee(paq_sent);

      // Liberar memoria de paquete API de alarma
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();

      // Si no ha habido error, se vacia una posicion del buffer (LIFO)
      if( !xbee868.error_TX )
      {
        pos--;
      }
   }

}

// Si no se ha liberado la memoria del paquete de KEEPALIVE, se libera ahora.
if (paq_sent != NULL)
{
  // Liberar memoria de paquete API
  free(paq_sent);
  paq_sent=NULL;
  // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
  clearData();
}
```

```
// ---------------------------------------------------------------------
// PASO 2. En caso de que alguno de los valores susceptibles de generar
// alarmas supere un determinado umbral, mandar alarma.
// Luego se considerara el caso de los mensajes de ESTADO_ENLACE
// ---------------------------------------------------------------------

// ALARMA POR TEMP_ALTA
if(temp >= 55)
{
  // Primero esperamos 1 s antes de enviar la alarma para que
  // todo se procese bien.
  delay(1000);

  // Set params to send en paquete API (se reserva memoria para el paquete primero)
  paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
  paq_sent->mode=UNICAST;
  paq_sent->MY_known=0;
  paq_sent->packetID=0x52;
  paq_sent->opt=0;
  xbee868.hops=0;
  xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

  // Creamos un mensaje de ALARMA por TEMP_ALTA
  crearMensajeAlarma(1);

  xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
  xbee868.sendXBee(paq_sent);
  // Liberar memoria de paquete API
  free(paq_sent);
  paq_sent=NULL;
  // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
  clearData();

}
// ALARMA POR BATERIA_BAJA
if(batt_level <= 20)
{
  // Primero esperamos 1 s antes de enviar la alarma para que
  // todo se procese bien.
  delay(1000);

  // Set params to send en paquete API (se reserva memoria para el paquete primero)
  paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
  paq_sent->mode=UNICAST;
  paq_sent->MY_known=0;
  paq_sent->packetID=0x52;
  paq_sent->opt=0;
  xbee868.hops=0;
  xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

  // Creamos un mensaje de ALARMA por BATERIA_BAJA
  crearMensajeAlarma(2);

  xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
  xbee868.sendXBee(paq_sent);
  // Liberar memoria de paquete API
  free(paq_sent);
  paq_sent=NULL;
  // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
  clearData();

}
// ALARMA POR DUTY CYCLE
if(dc_usado >= 90)
{
  // Primero esperamos 1 s antes de enviar la alarma para que
  // todo se procese bien.
  delay(1000);

  // Set params to send en paquete API (se reserva memoria para el paquete primero)
  paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
  paq_sent->mode=UNICAST;
  paq_sent->MY_known=0;
  paq_sent->packetID=0x52;
  paq_sent->opt=0;
  xbee868.hops=0;
  xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

  // Creamos un mensaje de ALARMA por DUTY_CYCLE
  crearMensajeAlarma(3);

  xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
  xbee868.sendXBee(paq_sent);
```

```
      // Liberar memoria de paquete API
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();
    }
  }

  // Si se reciben mensajes de KEEPALIVE de tipo LINK...
  if( XBee.available() )
  {
    // Se extraen los datos desde la UART (del xBee al ATMega)
    xbee868.treatData();
    if( !xbee868.error_RX )
    {
      // Guardamos los parametros del paquete recibido
      while(xbee868.pos>0)
      {
        // Copiamos los datos recibidos en la cadena "data"
        Utils.strCp( xbee868.packet_finished[xbee868.pos-1]->data , data);

        // Liberamos memoria del paquete
        free(xbee868.packet_finished[xbee868.pos-1]);
        xbee868.packet_finished[xbee868.pos-1]=NULL;
        xbee868.pos--;

        // Extraemos los campos del mensaje
        extraerCampos(dia_aux, fecha_hora_aux, tipoMens_aux, NAOrigen_aux, NADestino_aux, temp_aux, batt_aux, dc_aux,
num_saltos_aux);

        // 1) Se comprueba el origen del KEEPALIVE y si proviene de un nodo cuyo radioenlace asociado
        //    tiene mejor RSSI que nodo seleccionado como Next Hop en la tabla de encaminamiento,
        //    se elige este nodo como Next Hop en la tabla.
        if ((Utils.strCmp(NAOrigen_aux, "0042", 4)==0 ))
        {
            if (ind > 0)
               ind = 0;
        }
        else if ((Utils.strCmp(NAOrigen_aux, "0043", 4)==0 ))
        {
            if (ind > 1)
               ind = 1;
        }
        // En ultimo caso, si se recibe un KEEPALIVE de tipo link del nodo 0000,
        // se ignora y se deja como esta, ya que no hay un caso peor.

      }
    }
  }
}
```

A1.2. NODO 0042: C. Maribáñez.

```
/*-------------------------------------------------------------------------------
 * - Nombre: NODO 0042 – C. MARIBAÑEZ
 * - Autor: Manuel Jesús Cano Moreno
 *-------------------------------------------------------------------------------
*/

#define TAM_TABLA 2

// Tipos de MENSAJE
#define MSG_KEEPALIVE  "KEEPALIVE"
#define MSG_ALARMA     "ALARMA"
#define MSG_EST_ENL    "ESTADO_ENLACE"

// Tipos de ALARMA
#define ALM_CAIDA "CAIDA_ENLACE"
#define ALM_TEMP  "TEMP_ALTA"
#define ALM_BATT  "BATERIA_BAJA"
#define ALM_DC    "DUTY_CYCLE"

// Tipos de KEEPALIVE
#define KEEPALIVE_NODE 0
#define KEEPALIVE_LINK 1

#define NUM_SALTOS 0

packetXBee* paq_sent;
```

```
// Datos de tiempo
unsigned long t; // Instante actual (valor absoluto)
unsigned long T; // Instante de inicio de periodo (valor absoluto)

// Datos del tiempo para KEEPALIVES de tipo 2 (informan de que un nodo se ha vuelto a activar)
unsigned long t2; // Instante actual (valor absoluto)
unsigned long T2; // Instante de inicio de periodo (valor absoluto)

// Mensaje a enviar
char data[200];

// -- DIRECCIONAMIENTO Y ENCAMINAMIENTO -----------------------------------------

char NAOrigen[6] = "0042";
char NADestino[6] = "0000";
char NABroadcast[6] = "FFFF";

// Tabla de direcciones MAC
char tablaMAC[TAM_TABLA][18] = {"0013A20040581DC1",
                                "0013A200403AB314"};

// Tabla de direcciones NA
char tablaNA[TAM_TABLA][6]  = {"0043",
                               "0000"};

// Indice que indica el salto siguiente (Next Hop) elegido
uint8_t ind = 0;

// Indice que se usa para los reintentos de envio
uint8_t i;
// Indice para los reenvios de alarma
uint8_t j;

// ---------------------------------------------------------------------------

// Campo comun a todos (KEEPALIVE, ESTADO_ENLACE, ALARMA)
char *fecha_hora;

// Campos del mensaje de KEEPALIVE
uint8_t temp;
char aux_temp[6];
uint8_t batt_level;
uint8_t dc_usado;

// Campos del mensaje de ESTADO_ENLACE
uint8_t valorRSSI;
char valorRSSI_hex[6];
uint8_t reintentos = 0;
char paqBuenos_hex[6];

// -- CADENAS AUXILIARES -------------------------------------------------------
// Cadenas auxiliares para guardar los campos del mensaje
char dia_aux[15] = {0};
char fecha_hora_aux[40] = {0};
char NAOrigen_aux[6] = {0};
char NADestino_aux[6] = {0};
char tipoMens_aux[15] = {0};
char temp_aux[6] = {0};
char batt_aux[6] = {0};
char dc_aux[6] = {0};
char num_saltos_aux[6] = {0};
long int num_salt_longint;

// ---------------------------------------------------------------------------

// Puesto que los mensajes de alarma por CAIDA_ENLACE son muy importantes,
// se crea un buffer para almacenarlos y reenviarlos cuando se actualice la tabla
// de encaminamiento
char buffer_msg_alarma[TAM_TABLA][200];
uint8_t pos = 0;

/* ---------------------------------------------------------------------------
 *  Extrae los campos del mensaje. (NAOrigen, NADestino y tipoMensaje (KEEPALIVE, ...)
 * ---------------------------------------------------------------------------
 */

void extraerCampos(char *dia, char *fechaHora, char *tipoMens, char *NAOrig, char *NADest,
                   char *temper, char *bater, char *duty, char *numSaltos)
{
  uint8_t i;
  char *p; // Puntero que recorre el mensaje para extraer los datos
  p = data;

  for (i=0; i<9; i++)
```

```
{
  // Este bucle extrae los campos sin ponerle el '\0' al final de cada uno
  while(*p != ',')
  {
    if(i==0)
    {
       if (*p != '#')
       {
         *dia = *p;
         dia++;
       }
    }
    if (i == 1)
    {  *fechaHora = *p;
        fechaHora++;
    }
    if (i == 2)
    {   *tipoMens = *p;
        tipoMens++;
    }
    if (i == 3)
    {   *NAOrig = *p;
        NAOrig++;
    }
    if (i == 4)
    {   *NADest = *p;
        NADest++;
    }

    // En el caso de que sea un mensaje de KEEPALIVE, extraemos los campos que correspondan.
    if ( (i > 4) && (Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) )
    {
        if (i == 5)
        {   *temper = *p;
            temper++;
        }
        if (i == 6)
        {   *bater = *p;
            bater++;
        }
        if (i == 7)
        {   *duty = *p;
            duty++;
        }
        if (i == 8)
        {
            if (*p != '#')
            {
              *numSaltos = *p;
              numSaltos++;
            }
            else break; // Si el caracter es '#', quiere decir, que hemos llegado al final del mensaje
                        // salimos del bucle while
        }
    }
    p++; // Se mueve un caracter el puntero que recorre el mensaje
  }
  p++; // Esto sirve para que "salte" la coma y pase al siguiente campo

  // Añadimos los '\0' de finalizacion del campo
  if(i==0)
  {  *dia = '\0';
  }
  if (i == 1)
  {  *fechaHora = '\0';
  }
  if (i == 2)
  {  *tipoMens = '\0';
  }
  if (i == 3)
  {  *NAOrig = '\0';
  }
  if (i == 4)
  {  *NADest = '\0';
  }
  if ( (i > 4) && (Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) )
  {
      if (i == 5)
      {   *temper = '\0';
      }
      if (i == 6)
      {   *bater = '\0';
      }
      if (i == 7)
```

```
        {   *duty = '\0';
        }
        if (i == 8)
        {   *numSaltos = '\0';
        }
    }
  }
}

/* ----------------------------------------------------------------------
 *   Funcion que actualiza el numero de saltos del campo correspondiente
 *   en los mensajes de KEEPALIVE NODE.
 * ----------------------------------------------------------------------
 */

void actualizarNumSaltos()
{
    // Convertimos a entero la cadena num saltos
    num_salt_longint = strtol(num_saltos_aux, NULL, 10);
    // Incrementamos en 1
    num_salt_longint++;
    // Limpiamos la cadena data por si acaso
    clearData();
    // Actualizamos el mensaje de KEEPALIVE NODE con el nuevo valor de num de Saltos
    sprintf(data,"##%s, %s,%s,%s,%s,%s,%s,%s,%ld####", dia_aux, fecha_hora_aux, tipoMens_aux, NAOrigen_aux, NADestino_aux,
temp_aux, batt_aux, dc_aux, num_salt_longint);
}

/* ----------------------------------------------------------------------
 *   Funcion que genera un mensaje de KEEPALIVE en el nodo
 * ----------------------------------------------------------------------
 */
void crearMensajeKeepAlive(uint8_t tipo_MSG_KA)
{
    // MENSAJE KEEPALIVE. Tipo NODE
    // + -- + ----------- + --------- + -------- + --------- + ------ + ------ + ------ + ---------- + ---- +
    // | ## | Fecha/Hora  | KEEPALIVE | NAOrigen | NaDestino |  Temp  |  Batt  |   DC   | Num Saltos | #### |
    // + -- + ----------- + --------- + -------- + --------- + ------ + ------ + ------ + ---------- + ---- +

    // MENSAJE KEEPALIVE. Tipo LINK  (Se envía por Broadcast a los nodos adyacentes)
    // + -- + ---- + --------- + -------- + ------------ + --- + --- + --- + --- + ---- +
    // | ## |      | KEEPALIVE | NAOrigen | NA_Broadcast |     |     |     |     | #### |
    // + -- + ---- + --------- + -------- + ------------ + --- + --- + --- + --- + ---- +

    // Si el mensaje de KEEPALIVE es de tipo NODO, se rellenan los campos.
    if (tipo_MSG_KA == KEEPALIVE_NODE)
    {
      // Obtenemos el "timestamp": Fecha/Hora
      fecha_hora = RTC.getTime();

      // Obtenemos la temperatura del xBee
      temp = RTC.getTemperature();

      // Obtenemos el nivel de bateria
      batt_level = PWR.getBatteryLevel();

      // Obtenemos el porcentaje de duty cycle disponible que ha sido usado
      xbee868.getDutyCicle();
      dc_usado = xbee868.dutyCicle;

      sprintf(data,"##%s,%s,%s,%s,%d,%d,%d,%d####", fecha_hora, MSG_KEEPALIVE, NAOrigen, NADestino, temp, batt_level, dc_usado,
NUM_SALTOS);
    }
    // Si el mensaje de KEEPALIVE es de tipo LINK, los campos se dejan vacíos y la dirección de destino se pone a FFFF.
    else if (tipo_MSG_KA == KEEPALIVE_LINK)
    {
      sprintf(data,"##,,%s,%s,%s,,,,####", MSG_KEEPALIVE, NAOrigen, NABroadcast);
    }
}

/* ----------------------------------------------------------------------------
 *   Funcion que genera un mensaje de ESTADO DE ENLACE en el nodo de recepcion
 * ----------------------------------------------------------------------------
 */
void crearMensajeEstEnlace()
{
    // MENSAJE ESTADO_ENLACE
    // + -- + ----------- + --------------- + -------- + --------- + ------ + ---------- + ----------- + ---- +
    // | ## | Fecha/Hora  |  ESTADO_ENLACE  | NAOrigen | NaDestino |  RSSI  | MacRetries | Radioenlace | #### |
    // + -- + ----------- + --------------- + -------- + --------- + ------ + ---------- + ----------- + ---- +

    // Obtenemos el "timestamp": Fecha/Hora
    fecha_hora = RTC.getTime();
```

```
    // Guardamos el valor de RSSI
    xbee868.getRSSI();
    valorRSSI = xbee868.valueRSSI[0];

    // Guardamos el numero de reintentos MAC
    reintentos = xbee868.packet_finished[xbee868.pos-1]->retries;

    sprintf(data,"##%s,%s,%s,%s,%d,%d,%s,%s####", fecha_hora, MSG_EST_ENL, NAOrigen, NADestino, valorRSSI, reintentos,
NAOrigen_aux, NAOrigen);
}

/* ------------------------------------------------------------------------
 *   Funcion que genera un mensaje de ALARMA en el nodo
 * ------------------------------------------------------------------------
 */
void crearMensajeAlarma(uint8_t tipoAlarma)
{
    // MENSAJE ALARMA.                                                  { Campo opcional }
    // + -- + ----------- + -------- + -------- + --------- + ------------- + ----------- + ---- +
    // | ## | Fecha/Hora  |  ALARMA  | NAOrigen | NaDestino |  TIPO_ALARMA  | Radioenlace | #### |
    // + -- + ----------- + -------- + -------- + --------- + ------------- + ----------- + ---- +

    /* tipoAlarma:
       0 = CAIDA_ENLACE
       1 = TEMP_ALTA
       2 = BATERIA_BAJA
       3 = DUTY_CYCLE
    */

    // Obtenemos el "timestamp": Fecha/Hora
    fecha_hora = RTC.getTime();

    switch(tipoAlarma)
    {
      case 0:
             // ALARMA por CAIDA_ENLACE
             sprintf(data,"##%s,%s,%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_CAIDA, NAOrigen,
tablaNA[ind]);
             break;
      case 1:
             // ALARMA por TEMP_ALTA
             sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_TEMP);
             break;
      case 2:
             // ALARMA por BATERIA_BAJA
             sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_BATT);
             break;
      case 3:
             // ALARMA por DUTY_CYCLE
             sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_DC);
             break;
      default:
             // Opcion no valida...
             break;
    }
}

/* ------------------------------------------------------------------------------------------
 *  Limpiar el mensaje de datos
 * ------------------------------------------------------------------------------------------
 */

void clearData()
{
  uint8_t k;

  for(k=0;k<200;k++)
    data[k] = '\0';
}

/* ------------------------------------------------------------------------------------------
 *  Configuración inicial del Waspmote
 * ------------------------------------------------------------------------------------------
 */

void setup()
{
  // Se inicializa el xBee
  xbee868.init(XBEE_868,FREQ868M,NORMAL);

  // Powers XBee
  xbee868.ON();

  // Se pone la potencia al maximo (25 dBm)
```

```
  xbee868.setPowerLevel(4);

  // Inicializamos el RTC
  RTC.ON();

  // Meter todo el código de setup y dejar para el final el
  // cálculo de los instantes de tiempo para evitar que la llamada
  // a la funcion loop meta mucho retardo
  T = millis();
  T2 = millis();
}

void loop()
{

  // Obtenemos el instante actual
  t = millis();
  t2 = millis();

  if ( t2 >= T2+120000 )
  {
     // Actualizamos el instante de inicio de periodo de 2 min.
     T2 = t2;

     // Enviamos un KEEPALIVE de broadcast a todos los nodos adyacentes informando de que el nodo esta vivo
     paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
     paq_sent->mode=BROADCAST;
     paq_sent->MY_known=0;
     paq_sent->packetID=0x52;
     paq_sent->opt=0;
     xbee868.hops=0;
     xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

     // Creamos un mensaje de KeepAlive de tipo LINK
     crearMensajeKeepAlive(KEEPALIVE_LINK);

     // Retardo antes de enviar de 1 s
     delay(1000);

     xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
     xbee868.sendXBee(paq_sent);

     // Liberar memoria de paquete API
     free(paq_sent);
     paq_sent=NULL;
     // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
     clearData();
  }

  // Lo primero que se hace es enviar el mensaje de KEEPALIVE
  // (Si han pasado 30 segundos desde el ultimo envio de KEEPALIVE)
  if ( t >= T+30000)
  {
    // Actualizamos el instante de inicio de periodo de 30 seg.
    T = t;

    // -------------------------------------------------------------------
    // PASO 1. Siempre se envia el paquete de KEEPALIVE NODE, haya alarma o no.
    // -------------------------------------------------------------------

    // Set params to send en paquete API (se reserva memoria para el paquete primero)
    paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
    paq_sent->mode=UNICAST;
    paq_sent->MY_known=0;
    paq_sent->packetID=0x52;
    paq_sent->opt=0;
    xbee868.hops=0;
    xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

    // Creamos un mensaje de KeepAlive
    crearMensajeKeepAlive(KEEPALIVE_NODE);

    // Retardo antes de enviar de 1 s
    delay(1000);

    xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
    xbee868.sendXBee(paq_sent);

    // Si hay error de transmisión, se realizan 4 reintentos de envio, con retardos de
    // 50ms, 100ms, 150ms y 200ms (se va aumentando el retardo... algo parecido al
    // exponential backoff pero en lineal)

    if( xbee868.error_TX )
    {
```

```
   i = 0;
   while( (i<4) && (xbee868.error_TX) )
   {
     delay(1000*(i+1));
     xbee868.sendXBee(paq_sent);
     i++;
   }
   if (i == 4)
   {
     // Si se llega al máximo de reintentos...
     if(ind < TAM_TABLA-1)
     {
       // Liberamos primero la memoria del paquete anterior de KEEPALIVE
       free(paq_sent);
       paq_sent=NULL;
       // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
       clearData();

       // 1) Se crea el mensaje de ALARMA por CAIDA_ENLACE
       crearMensajeAlarma(0);
       // 2) Se actualiza la tabla de encaminamiento si no se ha llegado a la ultima posicion
       ind++;
       // 3) Se envia el mensaje de alarma
       paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
       paq_sent->mode=UNICAST;
       paq_sent->MY_known=0;
       paq_sent->packetID=0x52;
       paq_sent->opt=0;
       xbee868.hops=0;
       xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

      // Retardo antes de enviar de 1 s
      delay(1000);

      xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
      xbee868.sendXBee(paq_sent);

      // Puesto que el mensaje de CAIDA_ENLACE es importante, si da error al enviar, se reintenta 4 veces.
      if( xbee868.error_TX )
      {
         j = 0;
         while( (j<4) && (xbee868.error_TX) )
         {
           delay(250*(j+1));
           xbee868.sendXBee(paq_sent);
           j++;
         }
         // Si se supera el limite de reintentos de envio del mensaje de alarma por CAIDA_ENLACE...
         if (j == 4)
         {
           // Si el buffer no está lleno...
           if (pos < TAM_TABLA-1)
           {
             // Se copia el mensaje de alarma en el buffer
             strcpy(buffer_msg_alarma[pos], data);
             // Se incrementa el indice que apunta a la siguiente posicion libre en el buffer (LIFO)
             pos++;
           }
         }
      }

      // Liberar memoria de paquete API
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();

     } // Si se ha llegado a la última posicion, se vuelve a la primera a ver si se puede...
     else
     {
       ind = 0;
     }

   }
}
// En caso de que el KEEPALIVE NODE se haya mandado bien... se comprueba si hay mensajes de ALARMA por caida de enlace
// pendientes en el buffer y se intentan enviar...
else
{
   // Si hay mensajes en el buffer
   if(pos > 0)
   {
      paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
      paq_sent->mode=UNICAST;
```

```
      paq_sent->MY_known=0;
      paq_sent->packetID=0x52;
      paq_sent->opt=0;
      xbee868.hops=0;
      xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);


      // Retardo antes de enviar de 1 s
      delay(1000);

      // Ojo! El mensaje a enviar estará en "pos-1", ya que "pos" indica la primera posicion libre del buffer.
      xbee868.setDestinationParams(paq_sent, tablaMAC[ind], buffer_msg_alarma[pos-1], MAC_TYPE, DATA_ABSOLUTE);
      xbee868.sendXBee(paq_sent);

      // Liberar memoria de paquete API de alarma
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();

      // Si no ha habido error, se vacia una posicion del buffer (LIFO)
      if( !xbee868.error_TX )
      {
        pos--;
      }
   }

}

// Si no se ha liberado la memoria del paquete de KEEPALIVE, se libera ahora.
if (paq_sent != NULL)
{
  // Liberar memoria de paquete API
  free(paq_sent);
  paq_sent=NULL;
  // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
  clearData();
}

// ---------------------------------------------------------------------
// PASO 2. En caso de que alguno de los valores susceptibles de generar
// alarmas supere un determinado umbral, mandar alarma.
// Luego se considerara el caso de los mensajes de ESTADO_ENLACE
// ---------------------------------------------------------------------

// ALARMA POR TEMP_ALTA
if(temp >= 55)
{
  // Primero esperamos 1 s antes de enviar la alarma para que
  // todo se procese bien.
  delay(1000);

  // Set params to send en paquete API (se reserva memoria para el paquete primero)
  paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
  paq_sent->mode=UNICAST;
  paq_sent->MY_known=0;
  paq_sent->packetID=0x52;
  paq_sent->opt=0;
  xbee868.hops=0;
  xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

  // Creamos un mensaje de ALARMA por TEMP_ALTA
  crearMensajeAlarma(1);

  xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
  xbee868.sendXBee(paq_sent);
  // Liberar memoria de paquete API
  free(paq_sent);
  paq_sent=NULL;
  // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
  clearData();

}
// ALARMA POR BATERIA_BAJA
if(batt_level <= 20)
{
  // Primero esperamos 1 ms antes de enviar la alarma para que
  // todo se procese bien.
  delay(1000);

  // Set params to send en paquete API (se reserva memoria para el paquete primero)
  paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
  paq_sent->mode=UNICAST;
  paq_sent->MY_known=0;
```

```
      paq_sent->packetID=0x52;
      paq_sent->opt=0;
      xbee868.hops=0;
      xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

      // Creamos un mensaje de ALARMA por BATERIA_BAJA
      crearMensajeAlarma(2);

      xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
      xbee868.sendXBee(paq_sent);
      // Liberar memoria de paquete API
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();

    }
    // ALARMA POR DUTY CYCLE
    if(dc_usado >= 90)
    {
      // Primero esperamos 1 s antes de enviar la alarma para que
      // todo se procese bien.
      delay(1000);

      // Set params to send en paquete API (se reserva memoria para el paquete primero)
      paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
      paq_sent->mode=UNICAST;
      paq_sent->MY_known=0;
      paq_sent->packetID=0x52;
      paq_sent->opt=0;
      xbee868.hops=0;
      xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

      // Creamos un mensaje de ALARMA por DUTY_CYCLE
      crearMensajeAlarma(3);

      xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
      xbee868.sendXBee(paq_sent);
      // Liberar memoria de paquete API
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();
    }
  }

  // ----------------------------------------------------------------------------
  // PASO 3. Se comprueba si hay paquetes recibidos.
  // En caso afirmativo, se van guardando los paquetes, se crean mensajes
  // de ESTADO_ENLACE a partir de ellos y por último se guardan estos mensajes
  // en la tarjeta SD del nodo.
  // ----------------------------------------------------------------------------
  if( XBee.available() )
  {
    // Se extraen los datos desde la UART (del xBee al ATMega)
    xbee868.treatData();
    if( !xbee868.error_RX )
    {
      // Guardamos los parametros del paquete recibido
      while(xbee868.pos>0)
      {
        // Copiamos los datos recibidos en la cadena "data"
        Utils.strCp( xbee868.packet_finished[xbee868.pos-1]->data , data);

        // Liberamos memoria del paquete
        free(xbee868.packet_finished[xbee868.pos-1]);
        xbee868.packet_finished[xbee868.pos-1]=NULL;
        xbee868.pos--;

        // Extraemos los campos del mensaje
        extraerCampos(dia_aux, fecha_hora_aux, tipoMens_aux, NAOrigen_aux, NADestino_aux, temp_aux, batt_aux, dc_aux,
num_saltos_aux);

        // Si el mensaje es de KEEPALIVE de tipo NODE, se actualiza el numero de saltos y se sobreescribe el mensaje "data"
        if( (Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) && (Utils.strCmp(NADestino_aux, "FFFF", 4)!=0) )
        {
          actualizarNumSaltos();
        }

        // Si el mensaje NO es un mensaje de KEEPALIVE de tipo LINK, se reenvía
        if( (Utils.strCmp(NADestino_aux, "FFFF", 4)!=0) )
        {
          // Reenviamos el paquete
          // Set params to send en paquete API (se reserva memoria para el paquete primero)
```

```
          paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
          paq_sent->mode=UNICAST;
          paq_sent->MY_known=0;
          paq_sent->packetID=0x52;
          paq_sent->opt=0;
          xbee868.hops=0;
          xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

          // Retardo antes de enviar de 1 s
          delay(1000);

          // Se reenvia
          xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
          xbee868.sendXBee(paq_sent);
          // Liberar memoria de paquete API
          free(paq_sent);
          paq_sent=NULL;
          // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
          clearData();
        }
        // Si es un mensaje de KEEPALIVE de tipo LINK...
        else
        {
          // 1) Se comprueba el origen del KEEPALIVE y si proviene de un nodo cuyo radioenlace asociado
          //    tiene mejor RSSI que nodo seleccionado como Next Hop en la tabla de encaminamiento,
          //    se elige este nodo como Next Hop en la tabla.
          if ((Utils.strCmp(NAOrigen_aux, "0043", 4)==0 ))
          {
              // Si como Next Hop se tenía un nodo con un link peor... (en este caso el 0000), se
              // actualiza la tabla de encaminamiento a la mejor ruta
              if (ind > 0)
                 ind = 0;
          }
          // En otro caso... (si el paquete llega desde el 0000), no se hace nada... ya que este
          // broadcast no nos aporta nada. (es la peor ruta siempre la que llega desde el nodo 0000)
        }

        // Si el mensaje recibido es un KEEPALIVE de tipo NODE, proviene de uno de los nodos adyacentes (en este caso el 0041)
        // y el numero de saltos es 1 solo (enrutamiento directo) -> GENERAR MENSAJE DE ESTADO_ENLACE
        if( ( (Utils.strCmp(NAOrigen_aux, "0041", 4)==0) && (Utils.strCmp(NADestino_aux, "FFFF", 4)!=0)
           && (Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) )
           && (num_salt_longint == 1) )
        {
            // Creamos un mensaje de Estado de Enlace
            crearMensajeEstEnlace();

            // Enviamos el paquete de Estado de Enlace
            // Set params to send en paquete API (se reserva memoria para el paquete primero)
            paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
            paq_sent->mode=UNICAST;
            paq_sent->MY_known=0;
            paq_sent->packetID=0x52;
            paq_sent->opt=0;
            xbee868.hops=0;
            xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

            // Retardo antes de enviar de 1 s
            delay(1000);

            xbee868.setDestinationParams(paq_sent, tablaMAC[ind], data, MAC_TYPE, DATA_ABSOLUTE);
            xbee868.sendXBee(paq_sent);
            // Liberar memoria de paquete API
            free(paq_sent);
            paq_sent=NULL;
            // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy)
            clearData();
        }
      }
    }
  }
}
```

A1.3. NODO 0043: San Fernando.

```
/*-------------------------------------------------------------------------------
 * - Nombre: NODO 0043 – SAN FERNANDO
 * - Autor: Manuel Jesús Cano Moreno
 *-------------------------------------------------------------------------------
*/
```

```
// ------------------------------
// MAC del nodo: 0013A20040581DC1
// ------------------------------

// MAC del nodo siguiente. (Esto en realidad seria la tabla de encaminamiento..."
#define MAC_DESTINO "0013A200403AB314"

// Tipos de MENSAJE
#define MSG_KEEPALIVE  "KEEPALIVE"
#define MSG_ALARMA     "ALARMA"
#define MSG_EST_ENL    "ESTADO_ENLACE"

// Tipos de KEEPALIVE
#define KEEPALIVE_NODE 0
#define KEEPALIVE_LINK 1

// Tipos de ALARMA
#define ALM_TEMP  "TEMP_ALTA"
#define ALM_BATT  "BATERIA_BAJA"
#define ALM_DC    "DUTY_CYCLE"

#define NUM_SALTOS 0

packetXBee* paq_sent;

// Datos de tiempo
unsigned long t; // Instante actual (valor absoluto)
unsigned long T; // Instante de inicio de periodo (valor absoluto)

// Datos del tiempo para KEEPALIVES de tipo 2 (informan de que un nodo se ha vuelto a activar)
unsigned long t2; // Instante actual (valor absoluto)
unsigned long T2; // Instante de inicio de periodo (valor absoluto)

// Mensaje a enviar
char data[200];

// -- DIRECCIONAMIENTO Y ENCAMINAMIENTO ---------------------------------------

// NOTA: En este caso no hay abla de encaminamiento, puesto que sólo hay una opción
//       como siguiente salto, el nodo 0000 (Oficina).

char NAOrigen[6] = "0043";
char NADestino[6] = "0000";
char NABroadcast[6] = "FFFF";

// ------------------------------------------------------------------------------

// Campo comun a todos (KEEPALIVE, ESTADO_ENLACE, ALARMA)
char *fecha_hora;

// Campos del mensaje de KEEPALIVE
uint8_t temp;
char aux_temp[6];
uint8_t batt_level;
uint8_t dc_usado;

// Campos del mensaje de ESTADO_ENLACE
uint8_t valorRSSI;
char valorRSSI_hex[6];
uint8_t reintentos = 0;
char paqBuenos_hex[6];

// -- CADENAS AUXILIARES ------------------------------------------------------
// Cadenas auxiliares para guardar los campos del mensaje
char dia_aux[15] = {0};
char fecha_hora_aux[40] = {0};
char NAOrigen_aux[6] = {0};
char NADestino_aux[6] = {0};
char tipoMens_aux[15] = {0};
char temp_aux[6] = {0};
char batt_aux[6] = {0};
char dc_aux[6] = {0};
char num_saltos_aux[6] = {0};
long int num_salt_longint;


/* ------------------------------------------------------------------------------
 *  Extrae los campos del mensaje. (NAOrigen, NADestino y tipoMensaje (KEEPALIVE, ...)
 * ------------------------------------------------------------------------------
 */

void extraerCampos(char *dia, char *fechaHora, char *tipoMens, char *NAOrig, char *NADest,
                   char *temper, char *bater, char *duty, char *numSaltos)
{
```

```
uint8_t i;
char *p; // Puntero que recorre el mensaje para extraer los datos
p = data;

for (i=0; i<9; i++)
{
  // Este bucle extrae los campos sin ponerle el '\0' al final de cada uno
  while(*p != ',')
  {
    if(i==0)
    {
       if (*p != '#')
       {
         *dia = *p;
         dia++;
       }
    }
    if (i == 1)
    {  *fechaHora = *p;
        fechaHora++;
    }
    if (i == 2)
    {   *tipoMens = *p;
        tipoMens++;
    }
    if (i == 3)
    {   *NAOrig = *p;
        NAOrig++;
    }
    if (i == 4)
    {   *NADest = *p;
        NADest++;
    }

    // En el caso de que sea un mensaje de KEEPALIVE, extraemos los campos que correspondan.
    if ( (i > 4) && (Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) )
    {
        if (i == 5)
        {   *temper = *p;
            temper++;
        }
        if (i == 6)
        {   *bater = *p;
            bater++;
        }
        if (i == 7)
        {   *duty = *p;
            duty++;
        }
        if (i == 8)
        {
            if (*p != '#')
            {
              *numSaltos = *p;
              numSaltos++;
            }
            else break; // Si el caracter es '#', quiere decir, que hemos llegado al final del mensaje
                        // salimos del bucle while
        }
    }
    p++; // Se mueve un caracter el puntero que recorre el mensaje
  }
  p++; // Esto sirve para que "salte" la coma y pase al siguiente campo

  // Añadimos los '\0' de finalizacion del campo
  if(i==0)
  {  *dia = '\0';
  }
  if (i == 1)
  {  *fechaHora = '\0';
  }
  if (i == 2)
  {  *tipoMens = '\0';
  }
  if (i == 3)
  {  *NAOrig = '\0';
  }
  if (i == 4)
  {  *NADest = '\0';
  }
  if ( (i > 4) && (Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) )
  {
      if (i == 5)
      {   *temper = '\0';
```

```
        }
        if (i == 6)
        {   *bater = '\0';
        }
        if (i == 7)
        {   *duty = '\0';
        }
        if (i == 8)
        {   *numSaltos = '\0';
        }
    }
  }
}

/* ----------------------------------------------------------------------
 *   Funcion que actualiza el numero de saltos del campo correspondiente
 *   en los mensajes de KEEPALIVE.
 * ----------------------------------------------------------------------
 */

void actualizarNumSaltos()
{
    // Convertimos a entero la cadena num saltos
    num_salt_longint = strtol(num_saltos_aux, NULL, 10);
    // Incrementamos en 1
    num_salt_longint++;
    // Limpiamos la cadena data por si acaso
    clearData();
    // Actualizamos el mensaje de KEEPALIVE con el nuevo valor de num de Saltos
    sprintf(data,"##%s, %s,%s,%s,%s,%s,%s,%s,%ld####", dia_aux, fecha_hora_aux, tipoMens_aux, NAOrigen_aux, NADestino_aux,
temp_aux, batt_aux, dc_aux, num_salt_longint);
}

/* ----------------------------------------------------------------------
 *   Funcion que genera un mensaje de KEEPALIVE en el nodo
 * ----------------------------------------------------------------------
 */
void crearMensajeKeepAlive(uint8_t tipo_MSG_KA)
{
    // MENSAJE KEEPALIVE. Tipo NODE
    // + -- + ----------- + --------- + -------- + --------- + ------ + ------ + ------ + ---------- + ---- +
    // | ## | Fecha/Hora  | KEEPALIVE | NAOrigen | NaDestino |  Temp  |  Batt  |   DC   | Num Saltos | #### |
    // + -- + ----------- + --------- + -------- + --------- + ------ + ------ + ------ + ---------- + ---- +

    // MENSAJE KEEPALIVE. Tipo LINK  (Se envía por Broadcast a los nodos adyacentes)
    // + -- + ---- + --------- + -------- + ------------ + --- + --- + --- + --- + ---- +
    // | ## |      | KEEPALIVE | NAOrigen | NA_Broadcast |     |     |     |     | #### |
    // + -- + ---- + --------- + -------- + ------------ + --- + --- + --- + --- + ---- +

    // Si el mensaje de KEEPALIVE es de tipo NODO, se rellenan los campos.
    if (tipo_MSG_KA == KEEPALIVE_NODE)
    {
      // Obtenemos el "timestamp": Fecha/Hora
      fecha_hora = RTC.getTime();

      // Obtenemos la temperatura del xBee
      temp = RTC.getTemperature();

      // Obtenemos el nivel de bateria
      batt_level = PWR.getBatteryLevel();

      // Obtenemos el porcentaje de duty cycle disponible que ha sido usado
      xbee868.getDutyCicle();
      dc_usado = xbee868.dutyCicle;

      sprintf(data,"##%s,%s,%s,%s,%d,%d,%d,%d####", fecha_hora, MSG_KEEPALIVE, NAOrigen, NADestino, temp, batt_level, dc_usado,
NUM_SALTOS);
    }
    // Si el mensaje de KEEPALIVE es de tipo LINK, los campos se dejan vacíos y la dirección de destino se pone a FFFF.
    else if (tipo_MSG_KA == KEEPALIVE_LINK)
    {
      sprintf(data,"##,,%s,%s,%s,,,,####", MSG_KEEPALIVE, NAOrigen, NABroadcast);
    }
}

/* ----------------------------------------------------------------------------
 *   Funcion que genera un mensaje de ESTADO DE ENLACE en el nodo de recepcion
 * ----------------------------------------------------------------------------
 */
void crearMensajeEstEnlace()
{
    // MENSAJE ESTADO_ENLACE
    // + -- + ----------- + --------------- + -------- + --------- + ------ + ---------- + ----------- + ---- +
    // | ## | Fecha/Hora  |  ESTADO_ENLACE  | NAOrigen | NaDestino |  RSSI  | MacRetries | Radioenlace | #### |
```

```
    // + -- + ----------- + --------------- + -------- + --------- + ------ + ---------- + ----------- + ---- +

    // Obtenemos el "timestamp": Fecha/Hora
    fecha_hora = RTC.getTime();

    // Guardamos el valor de RSSI
    xbee868.getRSSI();
    valorRSSI = xbee868.valueRSSI[0];

    // Guardamos el numero de reintentos MAC
    reintentos = xbee868.packet_finished[xbee868.pos-1]->retries;

    sprintf(data,"##%s,%s,%s,%s,%d,%d,%s,%s####", fecha_hora, MSG_EST_ENL, NAOrigen, NADestino, valorRSSI, reintentos,
NAOrigen_aux, NAOrigen);
}

/* ------------------------------------------------------------------------
 *   Funcion que genera un mensaje de ALARMA en el nodo
 * ------------------------------------------------------------------------
 */
void crearMensajeAlarma(uint8_t tipoAlarma)
{
    // MENSAJE ALARMA.                                                    { Campo opcional }
    // + -- + ----------- + -------- + -------- + --------- + ------------- + ----------- + ---- +
    // | ## | Fecha/Hora  |  ALARMA  | NAOrigen | NaDestino |  TIPO_ALARMA  | Radioenlace | #### |
    // + -- + ----------- + -------- + -------- + --------- + ------------- + ----------- + ---- +

    /* tipoAlarma:
       0 = CAIDA_ENLACE
       1 = TEMP_ALTA
       2 = BATERIA_BAJA
       3 = DUTY_CYCLE
    */

    // Obtenemos el "timestamp": Fecha/Hora
    fecha_hora = RTC.getTime();

    switch(tipoAlarma)
    {
      case 0:
             // ALARMA por CAIDA_ENLACE
             // Este caso NO se va a dar ya que si cae un enlace directo al nodo de oficina, no puede informar a otro nodo
             // de que ha caido, para ello, se ocupa Waspmonitor, con unos temporizadores que marcan si un nodo ha caido y
             // sus radioenlaces adyacentes.
             break;
      case 1:
             // ALARMA por TEMP_ALTA
             sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_TEMP);
             break;
      case 2:
             // ALARMA por BATERIA_BAJA
             sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_BATT);
             break;
      case 3:
             // ALARMA por DUTY_CYCLE
             sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_DC);
             break;
      default:
             // Opcion no valida...
             break;
    }
}

/* ------------------------------------------------------------------------------------------
 *  Limpiar el mensaje de datos
 * ------------------------------------------------------------------------------------------
 */

void clearData()
{
  uint8_t k;

  for(k=0;k<200;k++)
    data[k] = '\0';
}

/* ------------------------------------------------------------------------------------------
 *  Configuración inicial del Waspmote
 * ------------------------------------------------------------------------------------------
 */

void setup()
{
  // Inicializa la libreria XBee 868
```

```
  xbee868.init(XBEE_868,FREQ868M,NORMAL);

  // Enciende el XBee
  xbee868.ON();

  // Se configura la potencia al máximo (25 dBm)
  xbee868.setPowerLevel(4);

  // Inicializa el RTC
  RTC.ON();

  // Meter todo el código de setup y dejar para el final el
  // cálculo de los instantes de tiempo para evitar que la llamada
  // a la funcion loop meta mucho retardo
  T = millis();
  T2 = millis();
}

/* ------------------------------------------------------------------------------------
 *  Bucle infinito
 * ------------------------------------------------------------------------------------
 */

 void loop()
{
  // Obtenemos el instante actual.
  t = millis();
  t2 = millis();

  if ( t2 >= T2+120000 )
  {
     // Actualizamos el instante de inicio de periodo de 2 min.
     T2 = t2;

     // Enviamos un KEEPALIVE de broadcast a todos los nodos adyacentes informando de que el nodo esta vivo
     paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
     paq_sent->mode=BROADCAST;
     paq_sent->MY_known=0;
     paq_sent->packetID=0x52;
     paq_sent->opt=0;
     xbee868.hops=0;
     xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

     // Creamos un mensaje de KeepAlive de tipo LINK
     crearMensajeKeepAlive(KEEPALIVE_LINK);

     // Retardo antes de enviar de 1 s
     delay(1000);

     xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
     xbee868.sendXBee(paq_sent);

     // Liberar memoria de paquete API
     free(paq_sent);
     paq_sent=NULL;
     // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
     clearData();
  }

  // Lo primero que se hace es enviar el mensaje de KEEPALIVE
  // (Si han pasado 30 segundos desde el ultimo envio de KEEPALIVE)
  if ( t >= T+30000)
  {
    // Actualizamos el instante de inicio de periodo de 30 seg.
    T = t;

    // ------------------------------------------------------------------
    // PASO 1. Siempre se envia el paquete de KEEPALIVE, haya alarma o no.
    // ------------------------------------------------------------------

    // Set params to send en paquete API (se reserva memoria para el paquete primero)
    paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
    paq_sent->mode=UNICAST;
    paq_sent->MY_known=0;
    paq_sent->packetID=0x52;
    paq_sent->opt=0;
    xbee868.hops=0;
    xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

    // Creamos un mensaje de KeepAlive
    crearMensajeKeepAlive(KEEPALIVE_NODE);

    // Retardo antes de enviar de 1 s
    delay(1000);
```

```
xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
xbee868.sendXBee(paq_sent);
// Liberar memoria de paquete API
free(paq_sent);
paq_sent=NULL;
// "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
clearData();

// ------------------------------------------------------------------
// PASO 2. En caso de que alguno de los valores susceptibles de generar
// alarmas supere un determinado umbral, mandar alarma.
// ------------------------------------------------------------------

// ALARMA POR TEMP_ALTA
if(temp >= 55)
{
  // Primero esperamos 1 s antes de enviar la alarma para que
  // todo se procese bien.
  delay(1000);

  // Set params to send en paquete API (se reserva memoria para el paquete primero)
  paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
  paq_sent->mode=UNICAST;
  paq_sent->MY_known=0;
  paq_sent->packetID=0x52;
  paq_sent->opt=0;
  xbee868.hops=0;
  xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

  // Creamos un mensaje de ALARMA por TEMP_ALTA
  crearMensajeAlarma(1);

  xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
  xbee868.sendXBee(paq_sent);
  // Liberar memoria de paquete API
  free(paq_sent);
  paq_sent=NULL;
  // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
  clearData();

}
// ALARMA POR BATERIA_BAJA
if(batt_level <= 20)
{
  // Primero esperamos 1 s antes de enviar la alarma para que
  // todo se procese bien.
  delay(1000);

  // Set params to send en paquete API (se reserva memoria para el paquete primero)
  paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
  paq_sent->mode=UNICAST;
  paq_sent->MY_known=0;
  paq_sent->packetID=0x52;
  paq_sent->opt=0;
  xbee868.hops=0;
  xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

  // Creamos un mensaje de ALARMA por BATERIA_BAJA
  crearMensajeAlarma(2);

  xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
  xbee868.sendXBee(paq_sent);
  // Liberar memoria de paquete API
  free(paq_sent);
  paq_sent=NULL;
  // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
  clearData();

}
// ALARMA POR DUTY CYCLE
if(dc_usado >= 90)
{
  // Primero esperamos 1 s antes de enviar la alarma para que
  // todo se procese bien.
  delay(1000);

  // Set params to send en paquete API (se reserva memoria para el paquete primero)
  paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
  paq_sent->mode=UNICAST;
  paq_sent->MY_known=0;
  paq_sent->packetID=0x52;
  paq_sent->opt=0;
  xbee868.hops=0;
```

```
      xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

      // Creamos un mensaje de ALARMA por DUTY_CYCLE
      crearMensajeAlarma(3);

      xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
      xbee868.sendXBee(paq_sent);
      // Liberar memoria de paquete API
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();
    }

  }

  // ---------------------------------------------------------------------------
  // PASO 3. Se comprueba si hay paquetes recibidos.
  // En caso afirmativo, se van guardando los paquetes, se crean mensajes
  // de ESTADO_ENLACE a partir de ellos y por último se guardan estos mensajes
  // en la tarjeta SD del nodo.
  // ---------------------------------------------------------------------------
  if( XBee.available() )
  {
    // Se extraen los datos desde la UART (del xBee al ATMega)
    xbee868.treatData();
    if( !xbee868.error_RX )
    {
      // Guardamos los parametros del paquete recibido
      while(xbee868.pos>0)
      {
        // Copiamos los datos recibidos en la cadena "data"
        Utils.strCp( xbee868.packet_finished[xbee868.pos-1]->data , data);

        // Liberamos memoria del paquete
        free(xbee868.packet_finished[xbee868.pos-1]);
        xbee868.packet_finished[xbee868.pos-1]=NULL;
        xbee868.pos--;

        // Extraemos los campos del mensaje
        extraerCampos(dia_aux, fecha_hora_aux, tipoMens_aux, NAOrigen_aux, NADestino_aux, temp_aux, batt_aux, dc_aux,
num_saltos_aux);

        // Si el mensaje es de KEEPALIVE de tipo NODE, se actualiza el numero de saltos y se sobreescribe el mensaje "data"
        if( (Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) && (Utils.strCmp(NADestino_aux, "FFFF", 4)!=0))
        {
          actualizarNumSaltos();
        }

        // Si el mensaje NO es un mensaje de KEEPALIVE de tipo LINK, se reenvía
        if( (Utils.strCmp(NADestino_aux, "FFFF", 4)!=0) )
        {
          // Reenviamos el paquete
          // Set params to send en paquete API (se reserva memoria para el paquete primero)
          paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
          paq_sent->mode=UNICAST;
          paq_sent->MY_known=0;
          paq_sent->packetID=0x52;
          paq_sent->opt=0;
          xbee868.hops=0;
          xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

          // Retardo antes de enviar de 1 s
          delay(1000);

          // Se reenvia
          xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
          xbee868.sendXBee(paq_sent);
          // Liberar memoria de paquete API
          free(paq_sent);
          paq_sent=NULL;
          // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
          clearData();
        }

        // Si el mensaje recibido es un KEEPALIVE de tipo NODE, proviene de uno de los nodos adyacentes (en este caso el 0041 y
el 0042)
        // y el numero de saltos es 1 solo (enrutamiento directo) -> GENERAR MENSAJE DE ESTADO_ENLACE
        if( ( ( (Utils.strCmp(NAOrigen_aux, "0041", 4)==0) || (Utils.strCmp(NAOrigen_aux, "0042", 4)==0) )
           && ((Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) && (Utils.strCmp(NADestino_aux, "FFFF", 4)!=0 )))
           && (num_salt_longint == 1))
        {
            // Creamos un mensaje de Estado de Enlace
            crearMensajeEstEnlace();
```

```
            // Enviamos el paquete de Estado de Enlace
            // Set params to send en paquete API (se reserva memoria para el paquete primero)
            paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
            paq_sent->mode=UNICAST;
            paq_sent->MY_known=0;
            paq_sent->packetID=0x52;
            paq_sent->opt=0;
            xbee868.hops=0;
            xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

            // Retardo antes de enviar de 1 s
            delay(1000);

            xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
            xbee868.sendXBee(paq_sent);
            // Liberar memoria de paquete API
            free(paq_sent);
            paq_sent=NULL;
            // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy)
            clearData();
        }

      }
    }
  }

}
```

A1.4. NODO 0000: Oficina.

```
/*-----------------------------------------------------------------------------
 * - Nombre: NODO 0000 - OFICINA
 * - Autor: Manuel Jesús Cano Moreno
 *-----------------------------------------------------------------------------
*/

// -------------------------------
// MAC del nodo: 0013A200403AB314
// -------------------------------

// MAC del nodo siguiente. (Esto en realidad seria la tabla de encaminamiento..."
#define MAC_DESTINO "0013A200403AB313"

// Tipos de MENSAJE
#define MSG_KEEPALIVE  "KEEPALIVE"
#define MSG_ALARMA     "ALARMA"
#define MSG_EST_ENL    "ESTADO_ENLACE"

// Tipos de ALARMA
#define ALM_TEMP  "TEMP_ALTA"
#define ALM_BATT  "BATERIA_BAJA"
#define ALM_DC    "DUTY_CYCLE"

// Tipos de KEEPALIVE
#define KEEPALIVE_NODE 0
#define KEEPALIVE_LINK 1

#define NUM_SALTOS 0

packetXBee* paq_sent;

// Datos de tiempo
unsigned long t; // Instante actual (valor absoluto)
unsigned long T; // Instante de inicio de periodo (valor absoluto)

// Datos del tiempo para KEEPALIVES de tipo 2 (informan de que un nodo se ha vuelto a activar)
unsigned long t2; // Instante actual (valor absoluto)
unsigned long T2; // Instante de inicio de periodo (valor absoluto)

// Mensaje a enviar
char data[200];

// -- DIRECCIONAMIENTO Y ENCAMINAMIENTO ----------------------------------------

// NOTA: En este caso no hay tabla de encaminamiento, puesto que sólo hay una opción
//       como siguiente salto, el nodo 0000 (Oficina).

char NAOrigen[6] = "0000";
```

```
char NADestino[6] = "0000";
char NABroadcast[6] = "FFFF";

// ------------------------------------------------------------------------------

// Campo comun a todos (KEEPALIVE, ESTADO_ENLACE, ALARMA)
char *fecha_hora;

// Campos del mensaje de KEEPALIVE
uint8_t temp;
char aux_temp[6];
uint8_t batt_level;
uint8_t dc_usado;

// Campos del mensaje de ESTADO_ENLACE
uint8_t valorRSSI;
char valorRSSI_hex[6];
uint8_t reintentos = 0;

// -- CADENAS AUXILIARES ------------------------------------------------------
// Cadenas auxiliares para guardar los campos del mensaje
char dia_aux[15] = {0};
char fecha_hora_aux[40] = {0};
char NAOrigen_aux[6] = {0};
char NADestino_aux[6] = {0};
char tipoMens_aux[15] = {0};
char temp_aux[6] = {0};
char batt_aux[6] = {0};
char dc_aux[6] = {0};
char num_saltos_aux[6] = {0};
long int num_salt_longint;

/* ---------------------------------------------------------------------------------
 *  Extrae los campos del mensaje. (NAOrigen, NADestino y tipoMensaje (KEEPALIVE, ...)
 * ---------------------------------------------------------------------------------
 */

//extraerCampos(dia_aux, fecha_hora_aux, NAOrigen_aux, NADestino_aux, tipoMens_aux, temp_aux, batt_aux, dc_aux, num_saltos_aux);

void extraerCampos(char *dia, char *fechaHora, char *tipoMens, char *NAOrig, char *NADest,
                   char *temper, char *bater, char *duty, char *numSaltos)
{
  uint8_t i;
  char *p; // Puntero que recorre el mensaje para extraer los datos
  p = data;

  for (i=0; i<9; i++)
  {
    // Este bucle extrae los campos sin ponerle el '\0' al final de cada uno
    while(*p != ',')
    {
      if(i==0)
      {
         if (*p != '#')
         {
           *dia = *p;
           dia++;
         }
      }
      if (i == 1)
      {  *fechaHora = *p;
          fechaHora++;
      }
      if (i == 2)
      {   *tipoMens = *p;
          tipoMens++;
      }
      if (i == 3)
      {   *NAOrig = *p;
          NAOrig++;
      }
      if (i == 4)
      {   *NADest = *p;
          NADest++;
      }

      // En el caso de que sea un mensaje de KEEPALIVE, extraemos los campos que correspondan.
      if ( (i > 4) && (Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) )
      {
          if (i == 5)
          {   *temper = *p;
              temper++;
          }
          if (i == 6)
```

```
          {   *bater = *p;
              bater++;
          }
          if (i == 7)
          {   *duty = *p;
              duty++;
          }
          if (i == 8)
          {
              if (*p != '#')
              {
                *numSaltos = *p;
                numSaltos++;
              }
              else break; // Si el caracter es '#', quiere decir, que hemos llegado al final del mensaje
                          // salimos del bucle while
          }
      }
      p++; // Se mueve un caracter el puntero que recorre el mensaje
    }
    p++; // Esto sirve para que "salte" la coma y pase al siguiente campo

    // Añadimos los '\0' de finalizacion del campo
    if(i==0)
    {  *dia = '\0';
    }
    if (i == 1)
    {  *fechaHora = '\0';
    }
    if (i == 2)
    {  *tipoMens = '\0';
    }
    if (i == 3)
    {  *NAOrig = '\0';
    }
    if (i == 4)
    {  *NADest = '\0';
    }
    if ( (i > 4) && (Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) )
    {
        if (i == 5)
        {   *temper = '\0';
        }
        if (i == 6)
        {   *bater = '\0';
        }
        if (i == 7)
        {   *duty = '\0';
        }
        if (i == 8)
        {   *numSaltos = '\0';
        }
    }
  }
}

/* ----------------------------------------------------------------------
 *   Funcion que actualiza el numero de saltos del campo correspondiente
 *   en los mensajes de KEEPALIVE tipo Node.
 * ----------------------------------------------------------------------
 */

void actualizarNumSaltos()
{
    // Convertimos a entero la cadena num saltos
    num_salt_longint = strtol(num_saltos_aux, NULL, 10);
    // Incrementamos en 1
    num_salt_longint++;
    // Limpiamos la cadena data por si acaso
    clearData();
    // Actualizamos el mensaje de KEEPALIVE con el nuevo valor de num de Saltos
    sprintf(data,"##%s, %s,%s,%s,%s,%s,%s,%s,%ld####", dia_aux, fecha_hora_aux, tipoMens_aux, NAOrigen_aux, NADestino_aux,
temp_aux, batt_aux, dc_aux, num_salt_longint);
}

/* ----------------------------------------------------------------------
 *   Funcion que genera un mensaje de KEEPALIVE en el nodo
 * ----------------------------------------------------------------------
 */
void crearMensajeKeepAlive(uint8_t tipo_MSG_KA)
{
    // MENSAJE KEEPALIVE. Tipo NODE
    // + -- + ---------- + --------- + -------- + --------- + ------ + ------ + ------ + ---------- + ---- +
    // | ## | Fecha/Hora  | KEEPALIVE | NAOrigen | NaDestino |  Temp  |  Batt  |   DC   | Num Saltos | #### |
```

```
    // + -- + ----------- + --------- + -------- + --------- + ------ + ------ + ------ + ---------- + ---- +

    // MENSAJE KEEPALIVE. Tipo LINK  (Se envía por Broadcast a los nodos adyacentes)
    // + -- + ---- + --------- + -------- + ------------ + --- + --- + --- + --- + ---- +
    // | ## |      | KEEPALIVE | NAOrigen | NA_Broadcast |     |     |     |     | #### |
    // + -- + ---- + --------- + -------- + ------------ + --- + --- + --- + --- + ---- +

    // Si el mensaje de KEEPALIVE es de tipo NODO, se rellenan los campos.
    if (tipo_MSG_KA == KEEPALIVE_NODE)
    {
      // Obtenemos el "timestamp": Fecha/Hora
      fecha_hora = RTC.getTime();

      // Obtenemos la temperatura del xBee
      temp = RTC.getTemperature();

      // Obtenemos el nivel de bateria
      batt_level = PWR.getBatteryLevel();

      // Obtenemos el porcentaje de duty cycle disponible que ha sido usado
      xbee868.getDutyCicle();
      dc_usado = xbee868.dutyCicle;

      sprintf(data,"##%s,%s,%s,%s,%d,%d,%d,%d####", fecha_hora, MSG_KEEPALIVE, NAOrigen, NADestino, temp, batt_level, dc_usado,
NUM_SALTOS);
    }
    // Si el mensaje de KEEPALIVE es de tipo LINK, los campos se dejan vacíos y la dirección de destino se pone a FFFF.
    else if (tipo_MSG_KA == KEEPALIVE_LINK)
    {
      sprintf(data,"##,,%s,%s,%s,,,,####", MSG_KEEPALIVE, NAOrigen, NABroadcast);
    }
}

/* ------------------------------------------------------------------------
 *   Funcion que genera un mensaje de ESTADO DE ENLACE en el nodo de recepcion
 * ------------------------------------------------------------------------
 */
void crearMensajeEstEnlace()
{
    // MENSAJE ESTADO_ENLACE
    // + -- + ----------- + ---------------- + -------- + --------- + ------ + ---------- + ----------- + ---- +
    // | ## | Fecha/Hora  |  ESTADO_ENLACE  | NAOrigen | NaDestino |  RSSI  | MacRetries | Radioenlace | #### |
    // + -- + ----------- + ---------------- + -------- + --------- + ------ + ---------- + ----------- + ---- +

    // Obtenemos el "timestamp": Fecha/Hora
    fecha_hora = RTC.getTime();

    // Guardamos el valor de RSSI
    xbee868.getRSSI();
    valorRSSI = xbee868.valueRSSI[0];

    // Guardamos el numero de reintentos MAC
    reintentos = xbee868.packet_finished[xbee868.pos-1]->retries;

    sprintf(data,"##%s,%s,%s,%s,%d,%d,%s,%s####", fecha_hora, MSG_EST_ENL, NAOrigen, NADestino, valorRSSI, reintentos,
NAOrigen_aux, NAOrigen);
}

/* ------------------------------------------------------------------------
 *   Funcion que genera un mensaje de ALARMA en el nodo
 * ------------------------------------------------------------------------
 */
void crearMensajeAlarma(uint8_t tipoAlarma)
{
    // MENSAJE ALARMA.                                                    { Campo opcional }
    // + -- + ----------- + -------- + -------- + --------- + ------------- + ----------- + ---- +
    // | ## | Fecha/Hora  |  ALARMA  | NAOrigen | NaDestino |  TIPO_ALARMA  | Radioenlace | #### |
    // + -- + ----------- + -------- + -------- + --------- + ------------- + ----------- + ---- +

    /* tipoAlarma:
       0 = CAIDA_ENLACE
       1 = TEMP_ALTA
       2 = BATERIA_BAJA
       3 = DUTY_CYCLE
    */

    // Obtenemos el "timestamp": Fecha/Hora
    fecha_hora = RTC.getTime();

    switch(tipoAlarma)
    {
      case 0:
             // ALARMA por CAIDA_ENLACE
             // Este caso NO se va a dar ya que si cae un enlace directo al nodo de oficina, no puede informar a otro nodo
```

```
            // de que ha caido, para ello, se ocupa Waspmonitor, con unos temporizadores que marcan si un nodo ha caido y
            // sus radioenlaces adyacentes.
            break;
      case 1:
            // ALARMA por TEMP_ALTA
            sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_TEMP);
            break;
      case 2:
            // ALARMA por BATERIA_BAJA
            sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_BATT);
            break;
      case 3:
            // ALARMA por DUTY_CYCLE
            sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_DC);
            break;
      default:
            // Opcion no valida...
            break;
    }
}

/* ----------------------------------------------------------------------------------
 *  Limpiar el mensaje de datos
 * ----------------------------------------------------------------------------------
 */

void clearData()
{
  uint8_t k;

  for(k=0;k<200;k++)
    data[k] = '\0';
}

/* ----------------------------------------------------------------------------------
 *  Configuración inicial del Waspmote
 * ----------------------------------------------------------------------------------
 */

void setup()
{
  // Inicializa la libreria XBee 868
  xbee868.init(XBEE_868,FREQ868M,NORMAL);

  // Enciende el XBee
  xbee868.ON();

  // Ponemos la potencia de transmisión al máximo
  xbee868.setPowerLevel(4);

  // Inicializa el RTC
  RTC.ON();

  // Meter todo el código de setup y dejar para el final el
  // cálculo de los instantes de tiempo para evitar que la llamada
  // a la funcion loop meta mucho retardo
  T = millis();
  T2 = millis();
}

/* ----------------------------------------------------------------------------------
 *  Bucle infinito
 * ----------------------------------------------------------------------------------
 */

 void loop()
{
  // Obtenemos el instante actual.
  t = millis();
  t2 = millis();

  if ( t2 >= T2+120000 )
  {
     // Actualizamos el instante de inicio de periodo de 2 min.
     T2 = t2;

     // Enviamos un KEEPALIVE de broadcast a todos los nodos adyacentes informando de que el nodo esta vivo
     paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
     paq_sent->mode=BROADCAST;
     paq_sent->MY_known=0;
     paq_sent->packetID=0x52;
     paq_sent->opt=0;
     xbee868.hops=0;
     xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);
```

```
   // Creamos un mensaje de KeepAlive de tipo LINK
   crearMensajeKeepAlive(KEEPALIVE_LINK);

   // Retardo antes de enviar de 1 s
   delay(1000);

   xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
   xbee868.sendXBee(paq_sent);

   // Liberar memoria de paquete API
   free(paq_sent);
   paq_sent=NULL;
   // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
   clearData();
}

// Lo primero que se hace es enviar el mensaje de KEEPALIVE de tipo Node
// (Si han pasado 30 segundos desde el ultimo envio de KEEPALIVE)
if ( t >= T+30000)
{
  // Actualizamos el instante de inicio de periodo de 30 seg.
  T = t;

  // ------------------------------------------------------------------
  // PASO 1. Siempre se envia el paquete de KEEPALIVE de tipo Node, haya alarma o no.
  // ------------------------------------------------------------------

  // Set params to send en paquete API (se reserva memoria para el paquete primero)
  paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
  paq_sent->mode=UNICAST;
  paq_sent->MY_known=0;
  paq_sent->packetID=0x52;
  paq_sent->opt=0;
  xbee868.hops=0;
  xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

  // Creamos un mensaje de KeepAlive
  crearMensajeKeepAlive(KEEPALIVE_NODE);

  // Retardo antes de enviar de 1 s
  delay(1000);

  xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
  xbee868.sendXBee(paq_sent);
  // Liberar memoria de paquete API
  free(paq_sent);
  paq_sent=NULL;
  // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
  clearData();

  // ------------------------------------------------------------------
  // PASO 2. En caso de que alguno de los valores susceptibles de generar
  // alarmas supere un determinado umbral, mandar alarma.
  // ------------------------------------------------------------------

  // ALARMA POR TEMP_ALTA
  if(temp >= 55)
  {
    // Primero esperamos 1 s antes de enviar la alarma para que
    // todo se procese bien.
    delay(1000);

    // Set params to send en paquete API (se reserva memoria para el paquete primero)
    paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
    paq_sent->mode=UNICAST;
    paq_sent->MY_known=0;
    paq_sent->packetID=0x52;
    paq_sent->opt=0;
    xbee868.hops=0;
    xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

    // Creamos un mensaje de ALARMA por TEMP_ALTA
    crearMensajeAlarma(1);

    xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
    xbee868.sendXBee(paq_sent);
    // Liberar memoria de paquete API
    free(paq_sent);
    paq_sent=NULL;
    // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
    clearData();

  }
```

```
    // ALARMA POR BATERIA_BAJA
    if(batt_level <= 20)
    {
      // Primero esperamos 1 s antes de enviar la alarma para que
      // todo se procese bien.
      delay(1000);

      // Set params to send en paquete API (se reserva memoria para el paquete primero)
      paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
      paq_sent->mode=UNICAST;
      paq_sent->MY_known=0;
      paq_sent->packetID=0x52;
      paq_sent->opt=0;
      xbee868.hops=0;
      xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

      // Creamos un mensaje de ALARMA por BATERIA_BAJA
      crearMensajeAlarma(2);

      xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
      xbee868.sendXBee(paq_sent);
      // Liberar memoria de paquete API
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();

    }
    // ALARMA POR DUTY CYCLE
    if(dc_usado >= 90)
    {
      // Primero esperamos 1 s antes de enviar la alarma para que
      // todo se procese bien.
      delay(1000);

      // Set params to send en paquete API (se reserva memoria para el paquete primero)
      paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
      paq_sent->mode=UNICAST;
      paq_sent->MY_known=0;
      paq_sent->packetID=0x52;
      paq_sent->opt=0;
      xbee868.hops=0;
      xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

      // Creamos un mensaje de ALARMA por DUTY_CYCLE
      crearMensajeAlarma(3);

      xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
      xbee868.sendXBee(paq_sent);
      // Liberar memoria de paquete API
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();
    }

  }

  // ----------------------------------------------------------------------------
  // PASO 3. Se comprueba si hay paquetes recibidos.
  // En caso afirmativo, se van guardando los paquetes, se crean mensajes
  // de ESTADO_ENLACE a partir de ellos y por último se guardan estos mensajes
  // en la tarjeta SD del nodo.
  // ----------------------------------------------------------------------------
  if( XBee.available() )
  {
    // Se extraen los datos desde la UART (del xBee al ATMega)
    xbee868.treatData();
    if( !xbee868.error_RX )
    {
      // Guardamos los parametros del paquete recibido
      while(xbee868.pos>0)
      {
        // Copiamos los datos recibidos en la cadena "data"
        Utils.strCp( xbee868.packet_finished[xbee868.pos-1]->data , data);

        // Liberamos memoria del paquete
        free(xbee868.packet_finished[xbee868.pos-1]);
        xbee868.packet_finished[xbee868.pos-1]=NULL;
        xbee868.pos--;

        // Extraemos los campos del mensaje
        extraerCampos(dia_aux, fecha_hora_aux, tipoMens_aux, NAOrigen_aux, NADestino_aux, temp_aux, batt_aux, dc_aux,
num_saltos_aux);
```

```
        // Si el mensaje es de KEEPALIVE de tipo NODE, se actualiza el numero de saltos y se sobreescribe el mensaje "data"
        if( (Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) && (Utils.strCmp(NADestino_aux, "FFFF", 4)!=0) )
        {
          actualizarNumSaltos();
        }

        // Si el mensaje NO es un mensaje de KEEPALIVE de tipo LINK, se reenvía
        if( (Utils.strCmp(NADestino_aux, "FFFF", 4)!=0) )
        {
          // Reenviamos el paquete
          // Set params to send en paquete API (se reserva memoria para el paquete primero)
          paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
          paq_sent->mode=UNICAST;
          paq_sent->MY_known=0;
          paq_sent->packetID=0x52;
          paq_sent->opt=0;
          xbee868.hops=0;
          xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

          // Retardo antes de enviar de 1 s
          delay(1000);

          // Se reenvia
          xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
          xbee868.sendXBee(paq_sent);
          // Liberar memoria de paquete API
          free(paq_sent);
          paq_sent=NULL;
          // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
          clearData();
        }

        // Si el mensaje recibido es un KEEPALIVE de tipo NODE, proviene de uno de los nodos adyacentes (0041,0042,0043 o 0044)
        // y el numero de saltos es 1 solo (enrutamiento directo) -> GENERAR MENSAJE DE ESTADO_ENLACE
        if( ( ((Utils.strCmp(NAOrigen_aux, "0041", 4)==0) || (Utils.strCmp(NAOrigen_aux, "0042", 4)==0) ||
               (Utils.strCmp(NAOrigen_aux, "0043", 4)==0) || (Utils.strCmp(NAOrigen_aux, "0044", 4)==0))
             && ((Utils.strCmp(tipoMens_aux, "KEEPALIVE", 9)==0) && (Utils.strCmp(NADestino_aux, "FFFF", 4)!=0 )))
             && (num_salt_longint == 1) )
        {
            // Creamos un mensaje de Estado de Enlace
            crearMensajeEstEnlace();

            // Enviamos el paquete de Estado de Enlace
            // Set params to send en paquete API (se reserva memoria para el paquete primero)
            paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
            paq_sent->mode=UNICAST;
            paq_sent->MY_known=0;
            paq_sent->packetID=0x52;
            paq_sent->opt=0;
            xbee868.hops=0;
            xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

            // Retardo antes de enviar de 1 s
            delay(1000);

            xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
            xbee868.sendXBee(paq_sent);
            // Liberar memoria de paquete API
            free(paq_sent);
            paq_sent=NULL;
            // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
            clearData();
        }
    }
  }
 }
}
```

A1.5. NODO 0044: Alcantarillas.

```
/*-------------------------------------------------------------------------------
 * - Nombre: NODO 0044 - ALCANTARILLAS
 * - Autor: Manuel Jesús Cano Moreno
 *-------------------------------------------------------------------------------
*/

// MAC del nodo siguiente. (Esto en realidad seria la tabla de encaminamiento..."
#define MAC_DESTINO "0013A200403AB314"
```

```
// Tipos de MENSAJE
#define MSG_KEEPALIVE  "KEEPALIVE"
#define MSG_ALARMA     "ALARMA"
#define MSG_EST_ENL    "ESTADO_ENLACE"

// Tipos de ALARMA
#define ALM_CAIDA "CAIDA_ENLACE"
#define ALM_TEMP  "TEMP_ALTA"
#define ALM_BATT  "BATERIA_BAJA"
#define ALM_DC    "DUTY_CYCLE"

// Tipos de KEEPALIVE
#define KEEPALIVE_NODE 0
#define KEEPALIVE_LINK 1

#define NUM_SALTOS 0

packetXBee* paq_sent;

// Datos de tiempo
unsigned long t; // Instante actual (valor absoluto)
unsigned long T; // Instante de inicio de periodo (valor absoluto)

// Mensaje a enviar
char data[200];

// -- DIRECCIONAMIENTO Y ENCAMINAMIENTO -----------------------------------------

char NAOrigen[6] = "0044";
char NADestino[6] = "0000";
char NABroadcast[6] = "FFFF";

// ------------------------------------------------------------------------------

// Campos del mensaje de KEEPALIVE
char *fecha_hora;

uint8_t temp;
char aux_temp[6];
uint8_t batt_level;
uint8_t dc_usado;

/* ---------------------------------------------------------------------
 *   Funcion que genera un mensaje de KEEPALIVE en el nodo
 * ---------------------------------------------------------------------
 */
void crearMensajeKeepAlive(uint8_t tipo_MSG_KA)
{
    // MENSAJE KEEPALIVE. Tipo NODE
    // + -- + ----------- + --------- + -------- + --------- + ------ + ------ + ------ + ---------- + ---- +
    // | ## | Fecha/Hora  | KEEPALIVE | NAOrigen | NaDestino |  Temp  |  Batt  |   DC   | Num Saltos | #### |
    // + -- + ----------- + --------- + -------- + --------- + ------ + ------ + ------ + ---------- + ---- +

    // MENSAJE KEEPALIVE. Tipo LINK  (Se envía por Broadcast a los nodos adyacentes)
    // + -- + ---- + --------- + -------- + ------------ + --- + --- + --- + --- + ---- +
    // | ## |      | KEEPALIVE | NAOrigen | NA_Broadcast |     |     |     |     | #### |
    // + -- + ---- + --------- + -------- + ------------ + --- + --- + --- + --- + ---- +

    // Si el mensaje de KEEPALIVE es de tipo NODO, se rellenan los campos.
    if (tipo_MSG_KA == KEEPALIVE_NODE)
    {
      // Obtenemos el "timestamp": Fecha/Hora
      fecha_hora = RTC.getTime();

      // Obtenemos la temperatura del xBee
      temp = RTC.getTemperature();

      // Obtenemos el nivel de bateria
      batt_level = PWR.getBatteryLevel();

      // Obtenemos el porcentaje de duty cycle disponible que ha sido usado
      xbee868.getDutyCicle();
      dc_usado = xbee868.dutyCicle;

      sprintf(data,"##%s,%s,%s,%s,%d,%d,%d,%d####", fecha_hora, MSG_KEEPALIVE, NAOrigen, NADestino, temp, batt_level, dc_usado,
NUM_SALTOS);
    }
    // Si el mensaje de KEEPALIVE es de tipo LINK, los campos se dejan vacíos y la dirección de destino se pone a FFFF.
    else if (tipo_MSG_KA == KEEPALIVE_LINK)
    {
      sprintf(data,"##,,%s,%s,%s,,,,####", MSG_KEEPALIVE, NAOrigen, NABroadcast);
    }
}
```

```
/* ------------------------------------------------------------------------
 *   Funcion que genera un mensaje de ALARMA en el nodo
 * ------------------------------------------------------------------------
 */
void crearMensajeAlarma(uint8_t tipoAlarma)
{
    // MENSAJE ALARMA.                                                  { Campo opcional }
    // + -- + ----------- + -------- + -------- + --------- + ------------- + ----------- + ---- +
    // | ## | Fecha/Hora  |  ALARMA  | NAOrigen | NaDestino |  TIPO_ALARMA  | Radioenlace | #### |
    // + -- + ----------- + -------- + -------- + --------- + ------------- + ----------- + ---- +

    /* tipoAlarma:
       0 = CAIDA_ENLACE
       1 = TEMP_ALTA
       2 = BATERIA_BAJA
       3 = DUTY_CYCLE
    */

    // Obtenemos el "timestamp": Fecha/Hora
    fecha_hora = RTC.getTime();

    switch(tipoAlarma)
    {
      case 0:
             // ALARMA por CAIDA_ENLACE
             sprintf(data,"##%s,%s,%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_CAIDA, NAOrigen,
NADestino);
             break;
      case 1:
             // ALARMA por TEMP_ALTA
             sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_TEMP);
             break;
      case 2:
             // ALARMA por BATERIA_BAJA
             sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_BATT);
             break;
      case 3:
             // ALARMA por DUTY_CYCLE
             sprintf(data,"##%s,%s,%s,%s,%s####", fecha_hora, MSG_ALARMA, NAOrigen, NADestino, ALM_DC);
             break;
      default:
             // Opcion no valida...
             break;
    }
}

/* ------------------------------------------------------------------------------------------
 *  Limpiar el mensaje de datos
 * ------------------------------------------------------------------------------------------
 */

void clearData()
{
  uint8_t k;

  for(k=0;k<200;k++)
    data[k] = '\0';
}

/* ------------------------------------------------------------------------------------------
 *  Configuración inicial del Waspmote
 * ------------------------------------------------------------------------------------------
 */

void setup()
{
  // Inits the XBee 868 library
  xbee868.init(XBEE_868,FREQ868M,NORMAL);

  // Powers XBee
  xbee868.ON();

  // Poner el xBee a potencia máxima
  xbee868.setPowerLevel(4);

  // Inicializamos el RTC
  RTC.ON();

  // Meter todo el código de setup y dejar para el final el
  // cálculo de los instantes de tiempo para evitar que la llamada
  // a la funcion loop meta mucho retardo
  T = millis();
}
```

```
void loop()
{
  // Obtenemos el instante actual.
  t = millis();

  // Lo primero que se hace es enviar el mensaje de KEEPALIVE
  // (Si han pasado 30 segundos desde el ultimo envio de KEEPALIVE)
  if ( t >= T+30000)
  {
    // Actualizamos el instante de inicio de periodo de 30 seg.
    T = t;

    // ------------------------------------------------------------------
    // PASO 1. Siempre se envia el paquete de KEEPALIVE de tipo Node, haya alarma o no.
    // ------------------------------------------------------------------

    // Set params to send en paquete API (se reserva memoria para el paquete primero)
    paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
    paq_sent->mode=UNICAST;
    paq_sent->MY_known=0;
    paq_sent->packetID=0x52;
    paq_sent->opt=0;
    xbee868.hops=0;
    xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

    // Creamos un mensaje de KeepAlive
    crearMensajeKeepAlive(KEEPALIVE_NODE);

    // Retardo antes de enviar de 1 s
    delay(1000);

    xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
    xbee868.sendXBee(paq_sent);

    // Liberar memoria de paquete API
    free(paq_sent);
    paq_sent=NULL;
    // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
    clearData();


    // ------------------------------------------------------------------
    // PASO 2. En caso de que alguno de los valores susceptibles de generar
    // alarmas supere un determinado umbral, mandar alarma.
    // Luego se considerara el caso de los mensajes de ESTADO_ENLACE
    // ------------------------------------------------------------------

    // ALARMA POR TEMP_ALTA
    if(temp >= 55)
    {
      // Primero esperamos 1 s antes de enviar la alarma para que
      // todo se procese bien.
      delay(1000);

      // Set params to send en paquete API (se reserva memoria para el paquete primero)
      paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
      paq_sent->mode=UNICAST;
      paq_sent->MY_known=0;
      paq_sent->packetID=0x52;
      paq_sent->opt=0;
      xbee868.hops=0;
      xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

      // Creamos un mensaje de ALARMA por TEMP_ALTA
      crearMensajeAlarma(1);

      xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
      xbee868.sendXBee(paq_sent);
      // Liberar memoria de paquete API
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();

    }
    // ALARMA POR BATERIA_BAJA
    if(batt_level <= 20)
    {
      // Primero esperamos 1 s antes de enviar la alarma para que
      // todo se procese bien.
      delay(1000);

      // Set params to send en paquete API (se reserva memoria para el paquete primero)
```

```
      paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
      paq_sent->mode=UNICAST;
      paq_sent->MY_known=0;
      paq_sent->packetID=0x52;
      paq_sent->opt=0;
      xbee868.hops=0;
      xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

      // Creamos un mensaje de ALARMA por BATERIA_BAJA
      crearMensajeAlarma(2);

      xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
      xbee868.sendXBee(paq_sent);
      // Liberar memoria de paquete API
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();

    }
    // ALARMA POR DUTY CYCLE
    if(dc_usado >= 90)
    {
      // Primero esperamos 1 s antes de enviar la alarma para que
      // todo se procese bien.
      delay(1000);

      // Set params to send en paquete API (se reserva memoria para el paquete primero)
      paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
      paq_sent->mode=UNICAST;
      paq_sent->MY_known=0;
      paq_sent->packetID=0x52;
      paq_sent->opt=0;
      xbee868.hops=0;
      xbee868.setOriginParams(paq_sent, "5678", MY_TYPE);

      // Creamos un mensaje de ALARMA por DUTY_CYCLE
      crearMensajeAlarma(3);

      xbee868.setDestinationParams(paq_sent, MAC_DESTINO, data, MAC_TYPE, DATA_ABSOLUTE);
      xbee868.sendXBee(paq_sent);
      // Liberar memoria de paquete API
      free(paq_sent);
      paq_sent=NULL;
      // "Limpiar" mensaje de datos (para evitar errores extraños al usar strCpy
      clearData();
    }

  }
}
```