UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Master oficial en Ingeniería Informática

TRABAJO FIN DE MASTER

SOFTWARE DE CONTROL SOBRE ELEMENTOS DE MEDIACIÓN.

Guillermo Carniado Cartas

Tutor: Luis Miguel Redondo

Ponente: Iván González Martínez

ENERO_2017

RESUMEN DEL TRABAJO FIN DE MASTER

El trabajo realizado se centra en el desarrollo del software de control utilizado por elementos de mediación pertenecientes a un sistema de monitorización sobre elementos de red.

El sistema permite la interacción remota entre el cliente y los diferentes elementos finales mediante la intervención del elemento de mediación dentro de una red local.

Gracias al elemento de mediación, el cliente puede conectar con diferentes elementos finales sin necesidad de conocer sus direcciones particulares dentro de la red local. En el sentido inverso, se encarga de interceptar las señales trap generadas por cada elemento final y hacerlas llegar al cliente.

Adicionalmente el elemento de mediación realiza una serie de operaciones internas de mantenimiento (señales periódicas de latido, comprobaciones de conexión, reseteo remoto, etc).

En conclusión, el trabajo realizado se ha centrado en el desarrollo del software necesario para manejar la conexión entre los elementos de un sistema que permite aislar al cliente de las particularidades de cada elemento final.

PALABRAS CLAVE

Elemento de mediación, Elemento final.

GLOSARIO

Tabla 1: Glosario

EM	Elemento de mediación.
EF	Elementos heterogéneos finales de red.

ABSTRACT

The current Works is focused on the control software development used by mediation elements that belongs to a network elements monitoring system.

The system allows the remote interaction between a client and different final elements through a mediation element inside a local network.

Thanks to the mediation element the client can connect with different final elements without need to know their particular addresses inside the local network. In the other way, it takes care of intercept the trap signals generated by each final element and retrieve it to the client.

In addition, the mediation element implements a set of maintenance internal operations (health checks, connectivity checks, remote resetting, etc).

In conclusion the works has been focused in the software development needed to handle the connection between the final elements inside a system that allows to isolate the client from the particularities of each final element.

KEYWORDS

Mediation element, final element.

GLOSSARY

Tabla 2: Glossary

EM	Mediation element.
EF	Heterogeneous network elements.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

En este apartado se pretende acotar el proyecto, señalando las motivaciones de las que parte, los objetivos que pretende cumplir y presentando la estructura del documento.

1.1. Motivación

La motivación del proyecto surge de la necesidad de actualizar las tecnologías referentes a los elementos de mediación.

La empresa colaboradora tiene una larga tradición de producción de distintos elementos de mediación para compañías con múltiples elementos de red. Estos dispositivos se conectaban a los elementos finales mediante bus serie, y permitían la conexión de los clientes mediante RTC (red telefónica conmutada).

Estos elementos contaban con las siguientes desventajas:

- Velocidad de red lenta (33Kbps), algo que era suficiente en su momento, pero que cambió debido al avance tecnológico, especialmente las interfaces de control de los elementos de red (pasaron de ser consolas a páginas web)
- Necesidad de una red telefónica conmutada, lo que suponía limitaciones a la hora de dar servicio a plantas donde era difícil implantar dichas redes (zonas remotas, por ejemplo, montañas).
- Precio relativamente alto.

Por estos motivos se buscaba un sistema capaz de actuar como mediador solventando los problemas de conectividad sin suponer un gasto excesivo.

Con el auge de las placas de bajo coste y al abaratamiento de los modem LTE surgió la alternativa de desarrollar un nuevo EM con unas prestaciones mejores dentro de un coste aceptable.

En conclusión, el proyecto pretende solventar la problemática existente en los antiguos EM gracias a las prestaciones del hardware actual.

1.2. Objetivos y alcance

El objetivo del trabajo es el análisis, diseño y desarrollo del software que utilizarán los elementos de mediación para permitir a un determinado cliente la gestión remota de distintos elementos de red.

Para abordarlo es necesario estudiar con detalle los siguientes elementos:

- Hardware del elemento de mediación: Sistema operativo de la placa y especificaciones del fabricante del módem.
- Bibliotecas de manejo de la placa en Python.
- Estructura de IpTables para la comunicación con los EFs.

Tras el conocimiento general del sistema y de los elementos que lo conforman se deberá detallar un análisis detallado de las características que debe cumplir el software a desarrollar.

Una vez realizado el análisis del software, se determinará el diseño de la solución propuesta para cumplir los requisitos determinados.

Tras el diseño de la solución, se desarrollará el código que ejecutará el elemento de mediación, y se realizarán pruebas para asegurar su funcionamiento.

De tal forma que el objetivo del trabajo, puede sintetizarse en los siguientes puntos:

- Estudio del sistema general en el que se engloba el elemento de mediación que utilizará el software a desarrollar.
- Investigación de los componentes que conforman el sistema general.
- Análisis de los problemas a resolver por el software.
- Diseño de la solución propuesta.
- Desarrollo del software del EM.
- Pruebas.

1.3. Estructura del documento

Este documento refleja todas las fases por las que ha atravesado el proyecto. Se ha dedicado un capítulo para describir todos sus puntos clave y características.

Capítulo 1 (***Introducción***): Se introducen los objetivos y alcance del proyecto.

- Capítulo 2 (***Estado del arte***): Se realiza un estudio acerca de las plataformas involucradas en el proyecto.
- Capítulo 3 (***Análisis***): Se documenta el análisis e investigación del sistema y los componentes que lo forman. Adicionalmente se formaliza la lista de requisitos del proyecto.
- Capítulo 4 (***Diseño del sistema***): Se muestra el diseño y desarrollo del sistema y en concreto del elemento de mediación.
- Capítulo 5 (***Pruebas***): Se mostrarán las pruebas realizadas sobre el software desarrollado, diferenciando el ámbito de las mismas.
- Por último, el capítulo 6 (***Conclusiones y trabajo futuro***) incluirá las conclusiones obtenidas de la realización completa del software y se comentarán posibles trabajos futuros que pudieran aportar mejoras al proyecto software.
- ***Anexos****:* información para una mejor comprensión del documento que, por su extensión, se ha decidido no incluir como contenido de la memoria.

2. ESTADO DEL ARTE

En este capítulo se pretenden mostrar las características y estado actual de las diferentes tecnologías utilizadas en el proyecto, de manera que pueda esclarecer las motivaciones para realizarlo, así como el contexto previo a su implementación.

DYCEC, la empresa colaboradora en el trabajo, tiene un largo recorrido en el desarrollo de soluciones profesionales para grandes clientes de las telecomunicaciones, como Orange o Telefónica.

Entre las soluciones desarrolladas por DYCEC, conviene centrarse en el ámbito de los elementos de mediación, por ser el que se abarca en el trabajo.

Todas estas soluciones se implementaron con éxito en el pasado, pero las nuevas tecnologías, y el surgimiento del mercado de placas de bajo coste suponían una oportunidad ideal para desarrollar nuevos elementos de mediación con mayores prestaciones y con un menor gasto.

De manera que el contexto en el que se sitúa el trabajo, es en el del desarrollo de un nuevo producto en un ámbito en el que se ha trabajado con anterioridad, pero que sin embargo supone una necesidad analítica debido al cambio en el hardware de estos productos.

Conociendo esto, se muestra un pequeño resumen para conocer el estado de las diferentes posibilidades hardware existentes en el mercado, y los motivos de la elección realizada por DYCEC para este producto en concreto.

2.1. Placas de bajo coste

En los últimos años ha despegado el mercado de las placas de bajo coste, gracias a sus muchas aplicaciones y reducido tamaño.

Dentro de este tipo de placas existen distintos fabricantes que ofrecen diferentes placas.

2.3.1. Placas Raspberry pi

Las placas desarrolladas por la fundación Raspberry Pi quizás sean las que más popularidad han obtenido.

Desde la aparición del modelo A en febrero de 2012, han surgido diversas versiones de esta placa, los modelos más habituales son los siguientes:

- Raspberry pi model A: La primera placa Raspberry en ser comercializada, contaba con 256 Mb de memoria RAM y no disponía de ethernet.
- Raspberry pi model B: Placa basada en el modelo A, con el mismo procesador, pero con mejores prestaciones (512 Mb de memoria RAM y ethernet).
- Raspberry pi model 2B: Siguiente generación de placas, mejor procesador y el doble de memoria RAM que el modelo B original.
- Raspberry pi model 3B: El último modelo comercializado. Presenta un procesador más potente e incorpora conexión Wifi y bluetooth.

Debido a las necesidades del proyecto resulta imprescindible la conexión ethernet, de manera que tanto el modelo A, como el resto de modelos sin ethernet (como Raspberry Pi Zero) resultan descartables como opciones a tener en cuenta.

De manera que, dentro de las placas Raspberry, la que resulta más atractiva es la Raspberry Pi 3B, por su conectividad y mayor capacidad de procesamiento.

2.1.2. Placas Arduino

Las placas arduino también han gozado de bastante popularidad, gracias sobre todo a la gran cantidad de proyectos de aprendizaje existentes en internet.

Estas placas están más enfocadas a proyectos hardware, por lo que disponen de una mayor variedad de puertos I/O que las placas Raspberry, sin embargo, tienen menor memoria RAM.

Debido a sus limitaciones de procesamiento, el uso de estas placas para afrontar el proyecto ha sido descartado.

2.1.3 Placas BeagleBone

Las placas BeagleBone producidas por Texas Instruments destacan por su hardware de código abierto y por su bajo consumo.

Además, estas placas presentan un punto intermedio entre la potencia de las placas Raspberry Pi, y la cantidad de puertos de las placas Arduino.

Entre las placas BeagleBone destacan las siguientes:

- BeagleBone: Comercializada desde octubre de 2010, esta placa destaca por disponer de 256 MB de memoria RAM, un procesador ARM a 720 MHz y un buen número de conectores (ethernet, USB, GPIO de 46 pin y puerto serie).

- BeagleBone Black: Aumenta la memoria RAM de Beaglebone hasta 512 Mb, además de disponer de 4GB de memoria flash eMMC en su última revisión.

Estas placas proporcionan un rendimiento adecuado para el proyecto, además de proporcionar más conectores que las placas Raspberry.

Debido a la memoria interna de la BeagleBone Black en su última revisión, resulta ser la placa más adecuada para el propósito de este trabajo de las que oferta este fabricante.

2.2. Comparativa de placas

En esta sección se realiza una comparativa entre las placas seleccionadas de cada fabricante, para terminar de justificar la elección.

Tabla 3: Comparativa de placas

Placa	**BeagleBone Black**	**Raspberry Pi 3B**
USB	1 (2.0)	**4(2.0)**
Almacenamiento	**Integrada: 4GB eMMC**	Integrada: No
Red	Ethernet: 10/100 Wifi: No	Ethernet: 10/100 **Wifi: Si**
Comunicaciones	Bluetooth: No	**Bluetooth: 4.1**
I/O	**GPIO: 66 pines** **Analógico: ADC 12 bit**	GPIO: 17 pines Analógico: No

Como se puede ver en la tabla comparativa, donde la placa Raspberry es superior es en el apartado de comunicaciones, al incluir wifi y bluetooth, sin embargo, debido a lo crítico de la comunicación inalámbrica en este proyecto, y que además se va a requerir comunicación 4G, esta funcionalidad se cubre mediante un modem externo.

Por otra parte, la placa Beaglebone incorpora un mayor número de conectores, y además existen librerías de manejo en Python que resultan de gran utilidad en el desarrollo de las funcionalidades del proyecto, de manera que la placa que se ha elegido finalmente es la Beaglebone Black.

Ilustración 1: BeagleBone Black

3. ANÁLISIS

En este apartado se pretende ofrecer una visibilidad global del sistema, correspondiente a la fase de análisis y estudio del problema a resolver.

3.1. Componentes del sistema

El sistema final está formado por los siguientes componentes:

- Aplicación cliente.
- Elemento de mediación (EM).
- Elementos finales.

El trabajo se enfoca en el software de control sobre el EM.

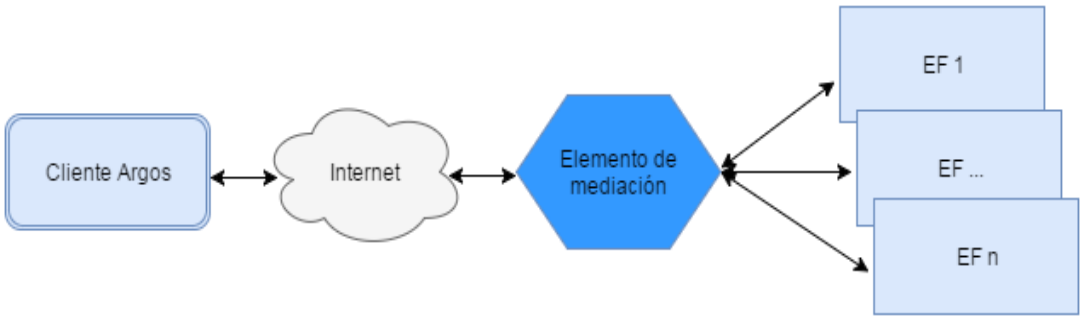

Ilustración 2: componentes del sistema

3.1.1. Clientes

El sistema está pensado para que un determinado cliente pueda gestionar de manera remota los elementos de red.

Actualmente existe ARGOS, una aplicación desarrollada por DYCEC para Orange que proporciona a los usuarios una interfaz web desde la que operar y recibir información sobre los elementos de red, a través de un EM.

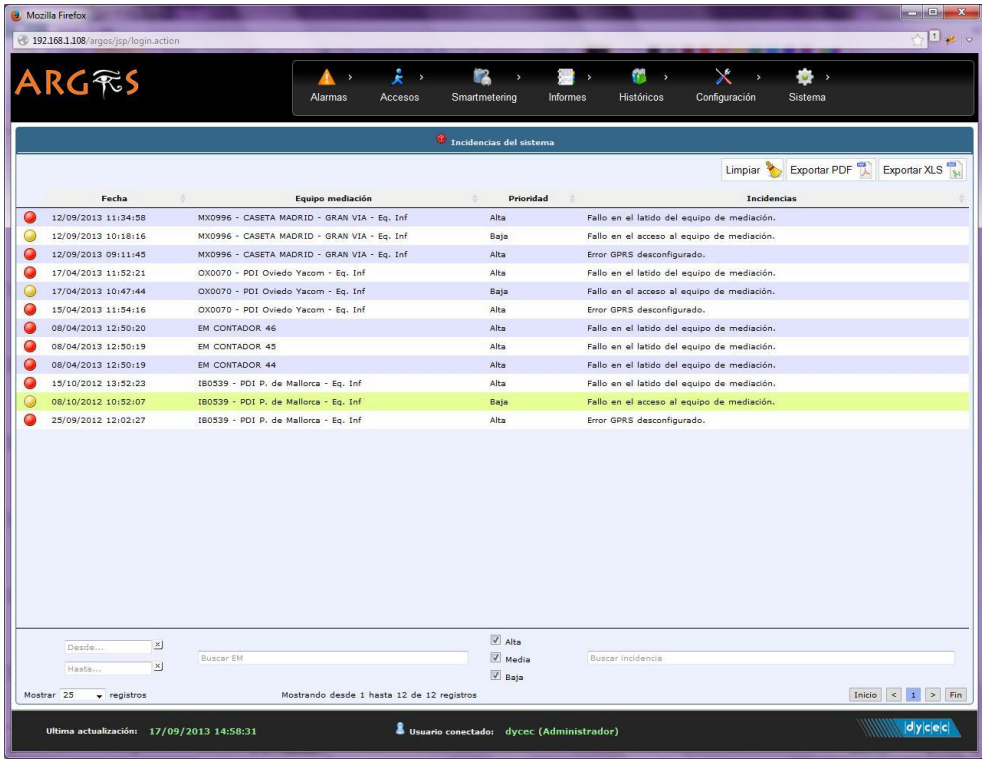

Ilustración 3: Cliente ARGOS

El desarrollo de esta aplicación, o de cualquier aplicación cliente que pueda utilizar los EM para comunicarse con elementos de red está fuera del ámbito de desarrollo de este trabajo, pero es importante conocerlas y explicarlas para tener una visión global del sistema.

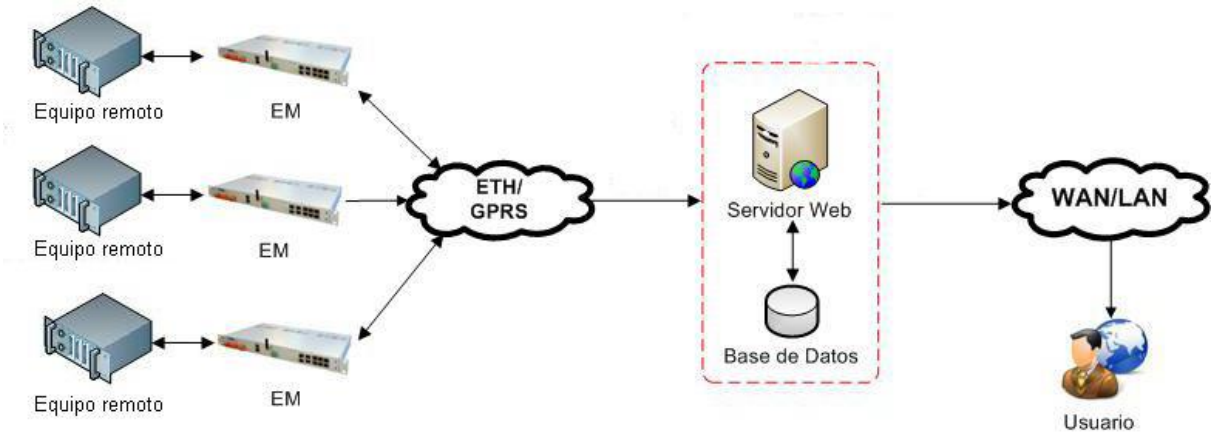

Ilustración 4: Esquema del sistema completo

En conclusión, el EM debe permitir la conexión a distintos clientes para gestionar elementos de red.

3.1.2. EM

El elemento de mediación consiste en una plataforma hardware que permite gestiones y operaciones remotas mediante acceso 4G sobre elementos finales (generalmente ubicados en centrales y lugares de comunicación) conectados a dicha plataforma mediante Ethernet.

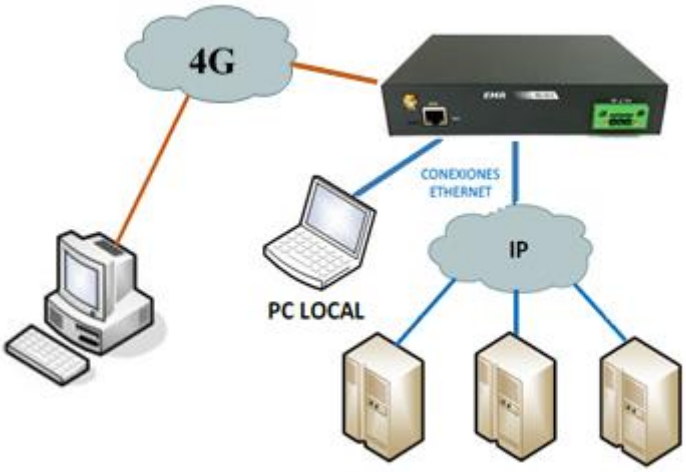

Ilustración 5: Conexiones con el EM

Las funcionalidades implementadas por el elemento de mediación están detalladas en el apartado 3.2 de este documento.

Ilustración 6: Elemento de mediación

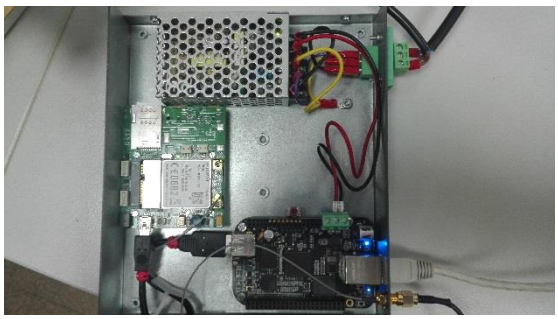

Ilustración 7: Elemento de mediación abierto

El EM está compuesto por una placa Beaglebone Black, conectada a un modem Huawei ME909s.

La placa BeagleBone es capaz de soportar la distribución Linux Debian, desde la que podrá configurarse el modem Huawei y el resto de funcionalidades necesarias.

El modem Huawei permite el estándar de comunicaciones LTE (categoría 4) para poder conectarse al EM vía 4G, con una velocidad de bajada de 150 Mbps, suficiente para el volumen de datos que maneja el sistema.

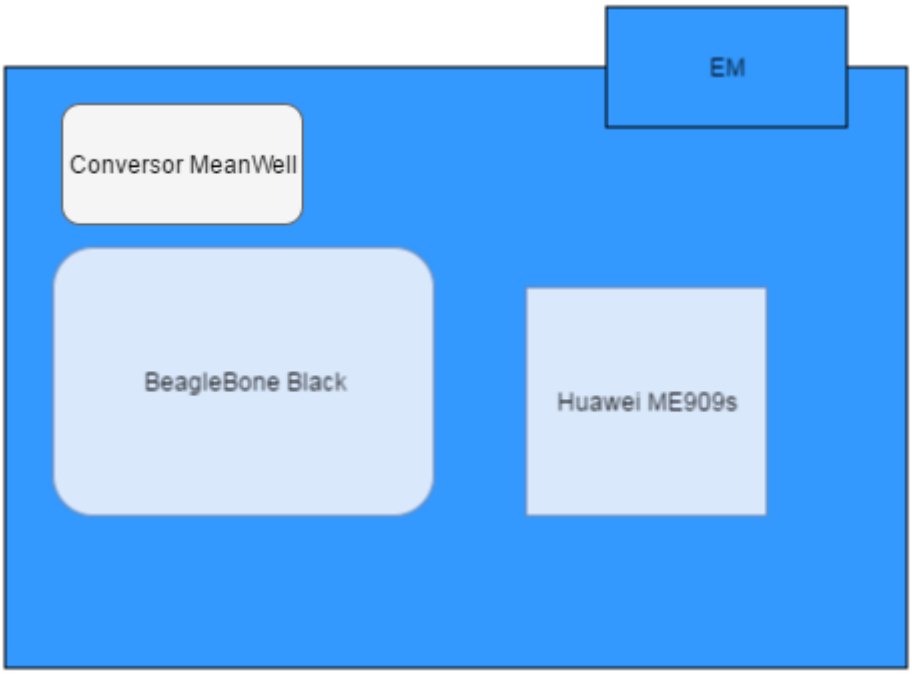

Ilustración 8: Esquema de EM

La fuente de alimentación MeanWell es necesaria debido a que generalmente los lugares de red utilizan una alimentación a -48VCC, mediante dicha fuente se hace la conversión necesaria para alimentar la placa BeagleBone con 5V.

3.1.3. EF

El objetivo final del EM es permitir a un determinado cliente monitorizar una serie de elementos finales.

Normalmente se trata de elementos de red situados en planta, en particular rectificadores de red de distintos fabricantes. Estos elementos pueden estar situados en lugares con mala conectividad.

Ilustración 9: Rectificador I

Ilustración 10: Rectificador II

3.2. Funcionalidades del EM

En este apartado se especifican los requisitos que definen el software utilizado por el EM.

3.2.1. Requisitos funcionales

A continuación, se especifican los módulos de funcionalidades soportados por el EM, detallando en cada uno de ellos los requisitos funcionales que los definen.

3.2.1.1. Configuración

- RF1: Inicializar el módem mediante parámetros definidos en un archivo de configuración.
    - RF1.1: Permitir la configuración de usuario, contraseña y APN del módem.
- RF2: Modificar la dirección Ethernet y máscara de red del equipo.
- RF3: Permitir restablecer la configuración inicial del equipo (configuración de fábrica).

3.2.1.2. Interfaz gráfica

- RF4: Permitir la configuración del EM mediante una interfaz de usuario.

3.2.1.3. Generación de informes

- RF5: Mantener un registro con los reinicios del sistema
- RF6: Mantener un registro periódico con los datos de cobertura del módem del EM

3.2.1.4. Comunicación

- RF7: Permitir redirección de los puertos TCP del EM con pares de IP y puerto pertenecientes al EF destino.

- RF8: Permitir el reenvío de alarmas (TRAP SNMP) de los EFs a diferentes direcciones IP de cliente.
- RF9: Enviar una señal de latido a un determinado cliente cada hora, indicando el id del equipo y la dirección IP asignada en la conexión módem.

3.2.2. Requisitos no funcionales

- RNF1: El Sistema debe asegurar la disponibilidad.
    - RNF1.1: El sistema debe responder correctamente ante fallos o bloqueos de la placa
    - RNF1.2: El sistema debe responder correctamente ante fallos de conexión con el módem del EM
    - RNF1.4: El sistema debe mantener las conexiones con todos los EF y clientes, aunque su dirección IP cambie (mantener funcionalidad en entornos con IP dinámica)
    - RNF1.5: El sistema debe mantener la hora, aunque sufra bloqueos o pérdida de conexión.

4. DISEÑO Y DESARROLLO DEL SISTEMA

En este apartado se definen los módulos funcionales necesarios para abordar los requisitos funcionales especificados en el apartado de análisis.

Para el desarrollo de las funcionalidades del EM se ha optado por el lenguaje de programación Python, los motivos de esta elección son los siguientes:

- Gran variedad de bibliotecas, incluidas bibliotecas de manejo de la Beaglebone.
- Conocimiento técnico por parte del equipo de DYCEC.

El principal apoyo para el desarrollo de los scripts de la placa consiste en la biblioteca *PyBBIO*, esta biblioteca proporciona herramientas para el manejo de la Beaglebone además de herramientas para generar una web sencilla para controlar la placa.

El desarrollo consiste en distintos scripts en Python para cumplir con los requisitos especificados.

Para lanzar estos scripts, se ha optado por archivo *crontab* del sistema.

Crontab [1] es un archivo del sistema Linux que permite especificar ciertos comandos que se ejecutarán cuando se cumplan determinadas condiciones, de esta manera se puede indicar que se ejecuten al arranque del sistema, que se repitan "n" veces o que se ejecuten a una hora determinada.

Para la interfaz de usuario, se utiliza la herramienta BBIOServer de la biblioteca PyBBIO, esta herramienta permite arrancar un servidor web simple para poder interactuar con la placa.

En resumen, el desarrollo del sistema consiste en la creación de los scripts en Python, que se ejecutarán mediante *crontab* o mediante llamadas desde la interfaz web cuando sea necesario, además de la creación de la propia web.

4.1. Configuración previa del EM

En este bloque se documenta la configuración previa para poder comenzar a trabajar con el equipo.

El primer paso es instalar el sistema operativo que más se ajuste a las necesidades. En la página de Beaglebone se proporcionan diferentes imágenes [2] de distribuciones Linux para la placa, como Ångström o Debian.

Finalmente se ha optado por Debian debido a que es la distribución con la que más familiarizado está el equipo de DYCEC.

4.2. Módulo de interfaz de usuario

Para permitir la gestión del EM es necesario proveer al sistema de un punto de entrada amigable con el usuario. En este caso se ha optado por una interfaz web, accesible mediante conexión 4G o mediante conexión local con el EM.

Hay que tener en cuenta que la interfaz web supone el único punto de entrada para la configuración del resto de parámetros del EM por parte del usuario, y que se incluyen capturas de la interfaz web en los bloques documentales de otros módulos.

Para generar la interfaz, se ha utilizado *BBIOServer* [2], una biblioteca de *PyBBIO* para generar un servidor web sencillo, este servidor amplía las funciones del servidor de Python *simpleHTTPServer*.

Esta biblioteca tiene implementada una clase llamada *Pages* que permite la implementación sencilla de páginas web que implementará el servidor.

Los controles que proporciona la clase *Pages* son limitados (cabeceras, textos, botones, campos de entrada…) y se han tenido que ampliar para añadir otro tipo de controles como las tablas.

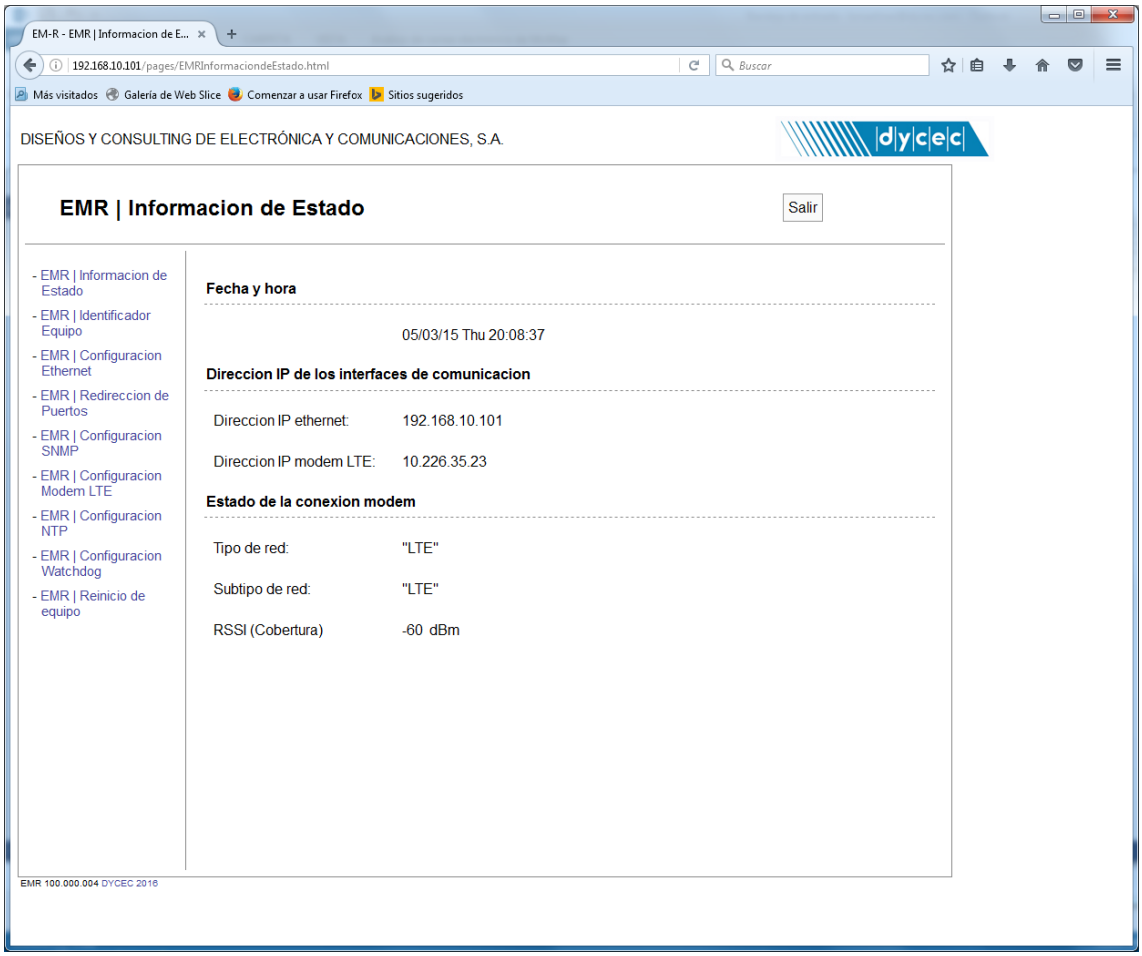

Ilustración 11: Punto de entrada de la interfaz web

4.3. Módulo de configuración

A continuación, se documenta el diseño y desarrollo del módulo de configuración, encargado de preparar al sistema para su correcto funcionamiento.

4.3.1. Configuración del modem

Este apartado se centra en la configuración del módem Huawei para su correcto funcionamiento dentro del equipo EM.

La configuración del módem incluye las siguientes tareas:

- Integración entre la placa y el módem.
- Conexión del módem y parametrización de usuario, contraseña y APN.

4.3.1.1 Integración entre la placa y el módem

El módem Huawei ME909s proporciona los drivers para su funcionamiento dentro de la misma placa, en el caso de Linux como en módulos para el kernel.

Sin embargo, el sistema lo reconoce como un dispositivo de almacenamiento en vez de como un módem; para solventarlo, la herramienta *usb_modeswitch* permite cambiar esta configuración para reconocer el dispositivo como un módem.

Una vez configurado el dispositivo como módem, se podrá comunicar con él mediante las direcciones */dev ttyUSB0* y */dev ttyUSB2*.

Por último, es recomendable desactivar el mecanismo de control de flujo RTS/CTS para asegurar una conexión correcta. Al utilizarlo, la conexión se puede bloquear.

De manera que para realizar la configuración hardware del módem, es necesario desarrollar un mecanismo capaz de forzar el reconocimiento del dispositivo como módem y de indicarle que no implemente el mecanismo RTS/CTS.

Como el sistema debe ser robusto ante bloqueos y permitir el reseteo de fábrica, las instrucciones para realizar estas tareas se incluirán en un script lanzado desde el *crontab* (Ver Anexo B).

Para que la placa reconozca el módem, se utiliza la herramienta *usb_modeswitch*, esta herramienta permite enviar comandos al dispositivo para que se inicie en el modo deseado.

En este caso se utiliza de la siguiente forma:

```
/usr/sbin/usb_modeswitch -v VENDOR-NUM -p PRODUCT-NUM -H -W
```

Donde se encuentran los siguientes parámetros:

- -v VENDOR-NUM: Indica que VENDOR-NUM es el id del fabricante.
- -p PRODUCT-NUM: Indica que PRODUCT-NUM es el id del producto.
- -H: Indica que se trata de un modelo huawei.
- -W: Indica que se debe imprimir la configuración antes de ejecutar el comando (para pruebas).

Una vez se ha ejecutado correctamente el cambio de modo del dispositivo, y se han creado las entradas *ttyUSB0* y *ttyUSB2* para comunicarse con él, es recomendable desactivar el mecanismo RTS/CTS para asegurar el correcto funcionamiento del módem.

Para realizarlo, se utiliza la herramienta *stty* [2], esta herramienta permite configurar ciertos parámetros de los puertos serie, en el caso actual servirá para desactivar el mecanismo RTS/CTS, añadiendo las siguientes instrucciones:

```
/bin/stty -F /dev/ttyUSB0 -crtscts
/bin/stty -F /dev/ttyUSB2 -crtscts
```

Donde se indica lo siguiente:

- -F DISPOSITIVO: Abre el dispositivo indicado.
- -crtcts: Desactiva el mecanismo RTS/CTS

Hecho esto. el dispositivo está listo para usarse.

4.3.1.2 Configuración de la conexión del módem

Una vez se ha configurado correctamente el módem, es necesario lanzar la conexión con los parámetros necesarios (configurables desde la web).

Para realizar la conexión a través del módem Huawei, se utilizará la herramienta *wvdial*. Esta herramienta permite la conexión mediante implementación de protocolo punto a punto, generando un demonio pppd para el sistema.

Para comenzar a funcionar, w*vdial* [3] necesita cargar la configuración del fichero *wvdialconf*.

Para autogenerarlo basta con ejecutar la siguiente línea:

```
wvdialconf /etc/wvdialconf
```

De esta manera la herramienta detecta el módem y genera el fichero de configuración por defecto para el móvil.

```
[Dialer Defaults]
Init1 = ATZ
Init2 = ATQ0 V1 E1 S0=0
Init4 = AT+CGDCONT=1,"IP","movistar.es";
Password = MOVISTAR
Phone = *99***1#
Idle seconds = 0
Modem Type = Analog Modem
Stupid Mode = 1
Baud = 9600
New PPPD = yes
Dial Command = ATD
Modem = /dev/ttyUSB0
ISDN = 0
Username = MOVISTAR
```

Este fichero es configurable, tanto por si se ha generado incorrectamente, faltan campos o se desea parametrizar algún campo.

Para poder parametrizar el usuario, contraseña y APN, se incluye la siguiente pantalla en la web de configuración:

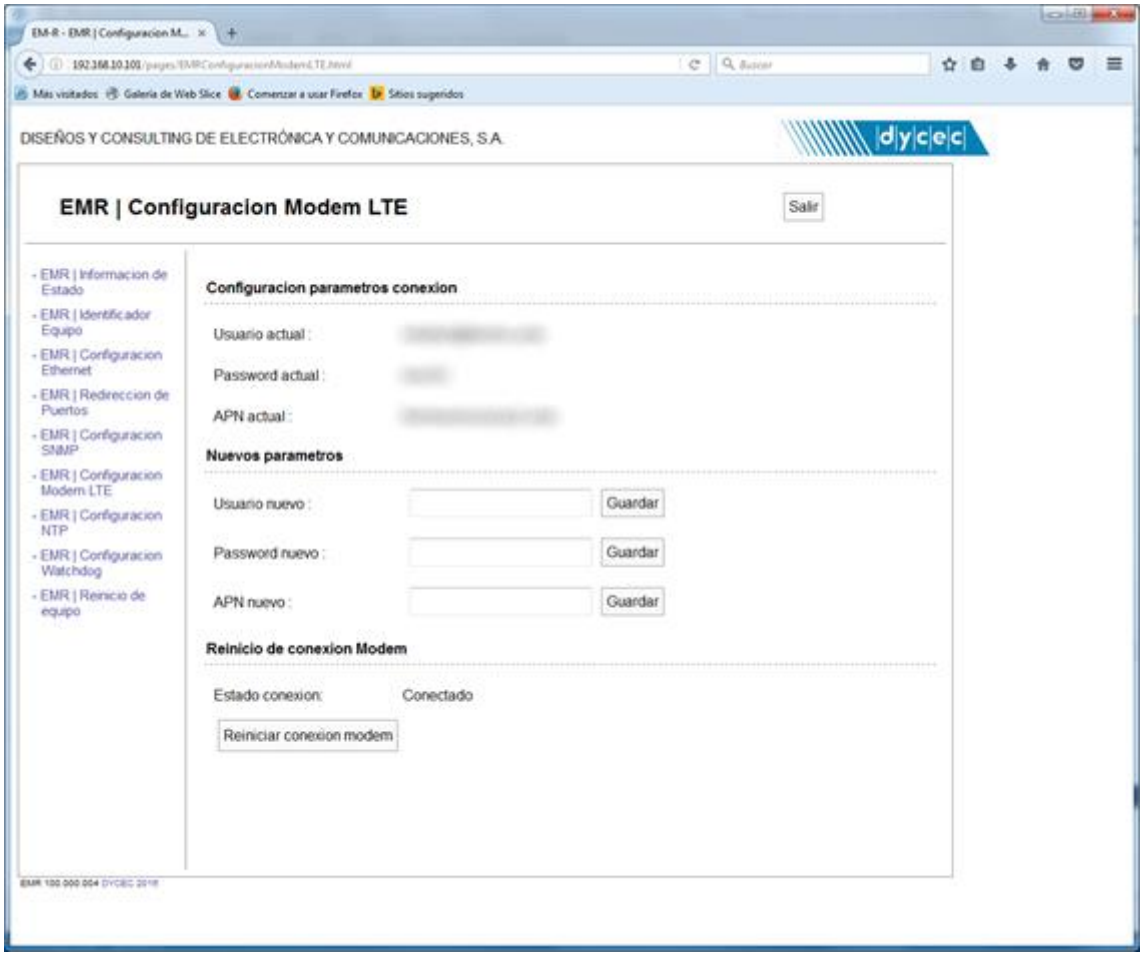

Ilustración 12: Configuración del modem

En esta pantalla, se permitirá cambiar cualquiera de los parámetros y reiniciar la conexión del módem.

Al guardar los parámetros se modificará el fichero *wvdialconf*, sin embargo, no se aplicarán los cambios (cargando esa configuración en wvdial) hasta que no se reinicie la conexión del módem. Algo que se pedirá bajo demanda mediante el botón de reinicio de módem ejecutando el script (Ver apartado 4.3.1.1), o se ejecutará automáticamente en el inicio mediante orden del *crontab*.

4.3.2. Configuración de acceso ethernet

En este apartado se explican los procedimientos necesarios para poder comunicar el acceso Ethernet para la red local en la que se ubicarán los distintos EFs.

Conviene recordar que los EFs pueden ser elementos de red de cualquier tipo, o por ejemplo ordenadores personales. El caso de los ordenadores personales es habitual, ya que los técnicos se pueden conectar al EM en el caso de tener que realizar alguna operación de mantenimiento.

De este modo se permite al usuario configurar la dirección y máscara de la red local, para adecuarse a las particularidades necesarias en cada planta.

Como en casos anteriores, para permitir la configuración Ethernet del módem se proporciona una página web desde la que configurar los parámetros necesarios.

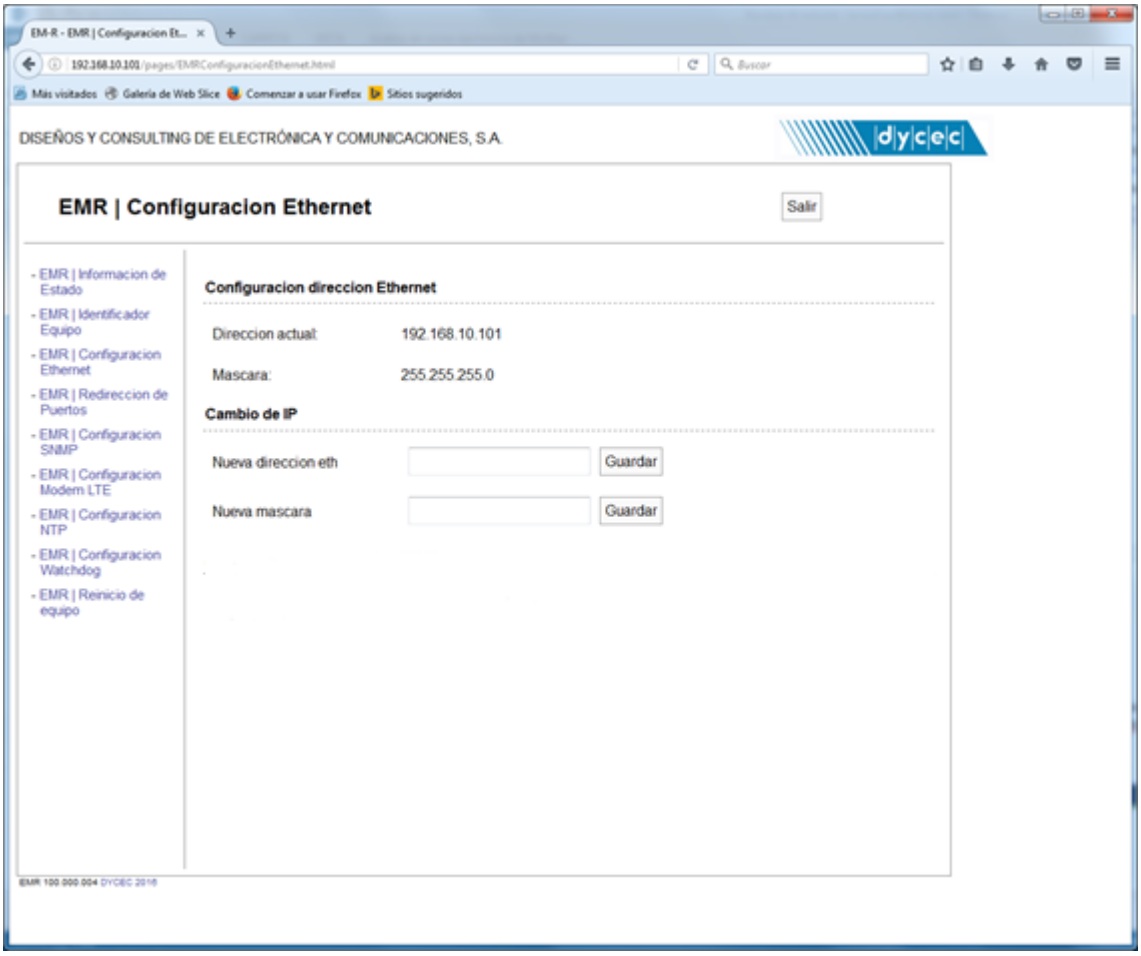

Ilustración 13: Configuración ethernet

Al guardar los parámetros se configura la interfaz Ethernet de la placa [6] modificando el fichero */etc/network/interfaces.*

```
def setMask(netmask,ip):
    fichero = "/etc/network/interfaces"
    patron = "iface eth0"
    patron2 = "auto eth0"
    reemplazo = "auto eth0\niface eth0 inet static\n\taddress " + ip + "\n\tnetmask " +
netmask + "\n"
    linea_ifconfig = "sudo /sbin/ifconfig eth0 " + ip + " netmask " + netmask
    borra_lineas(fichero, patron, 3)
    reemplaza_linea(fichero, patron2, reemplazo)
    os.system(linea_ifconfig)
    lanza_tablas()
```

Además, se relanza la creación de tablas *iptables* (Ver apartado 4.4.1).

4.3.3. Configuración NTP

Teniendo en cuenta que el EM carece de una batería interna, puede darse la situación de que en una circunstancia en la que se pierda la alimentación se pierda también la hora del sistema.

Para protegerle de este tipo de situaciones conviene dotar al sistema de la funcionalidad necesaria para recuperar la hora cuando se reestablezca el funcionamiento de la placa.

Para realizarlo, es posible obtener la hora actual de un servidor NTP. Dentro de las operaciones de configuración del EM se incluye la posibilidad de seleccionar el servidor NTP del que se recuperará la hora del sistema.

Para ello se proporcionará al usuario una página donde éste pueda indicar el servidor deseado.

Como el equipo tiene configurado el DNS, en este campo se permite poner nombres en vez de direcciones. Por ejemplo: pool.ntp.org. Sin embargo, en redes privadas a veces sólo se pueden poner direcciones.

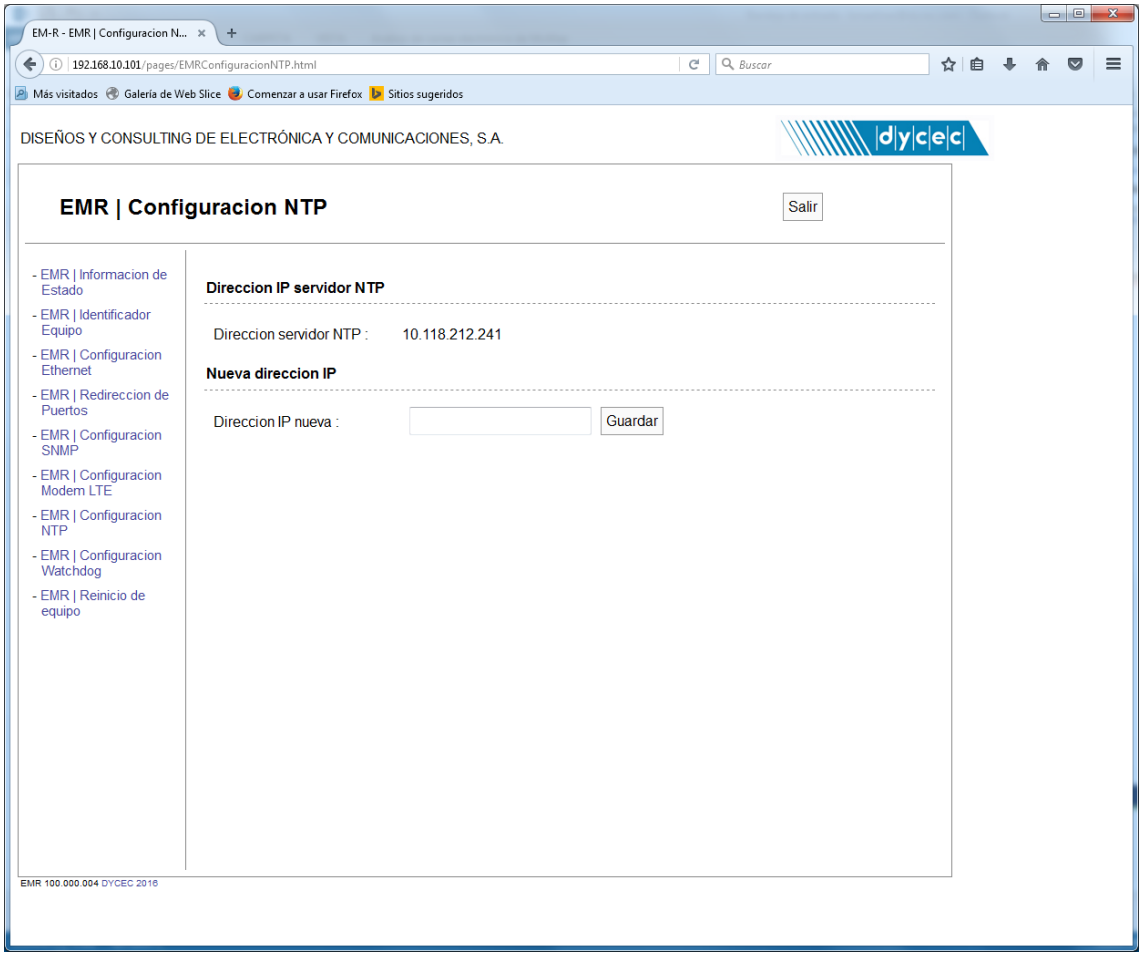

Ilustración 14: Configuración NTP

La dirección seleccionada en esta pantalla, se almacenará en un fichero de configuración que se utilizará cada vez que se desee actualizar la hora del sistema.

Para realizarlo se necesita un pequeño script encargado de recuperar la hora del servidor [2] definido y asignársela al reloj del sistema RTC de la BeagleBone mediante *hwclock*, que puede haber perdido la hora al quedarse sin alimentación [3].

```
/usr/sbin/ntpdate [DIRECCION_NTP]

sleep 10

/sbin/hwclock -w
```

Para que este script se ejecute de manera periódica y asegurar tener la conexión correcta en el sistema, se añade la siguiente instrucción en el *crontab*:

```
1 * * * * /usr/local/bin/lanza_ntp.sh
```

4.4. Módulo de comunicación

En este apartado se explica la elección de las tecnologías elegidas para abarcar los requisitos referentes a la comunicación entre el EM y el resto de componentes del sistema (Clientes y EFs).

4.4.1. Asignación de direcciones de elementos finales

Este apartado se centra en la resolución del requisito referente a la configuración del EM para redireccionar la entrada de uno de sus puertos con un determinado EF.

De esta manera los clientes podrán conectarse remotamente a los elementos finales a partir del EM sin conocer su IP.

La tecnología seleccionada para este propósito es *Iptables* [8] [9].

Iptables es una aplicación que maneja las bibliotecas TCP/IP del sistema (filtrado, intercepción y manipulado de paquetes de red).

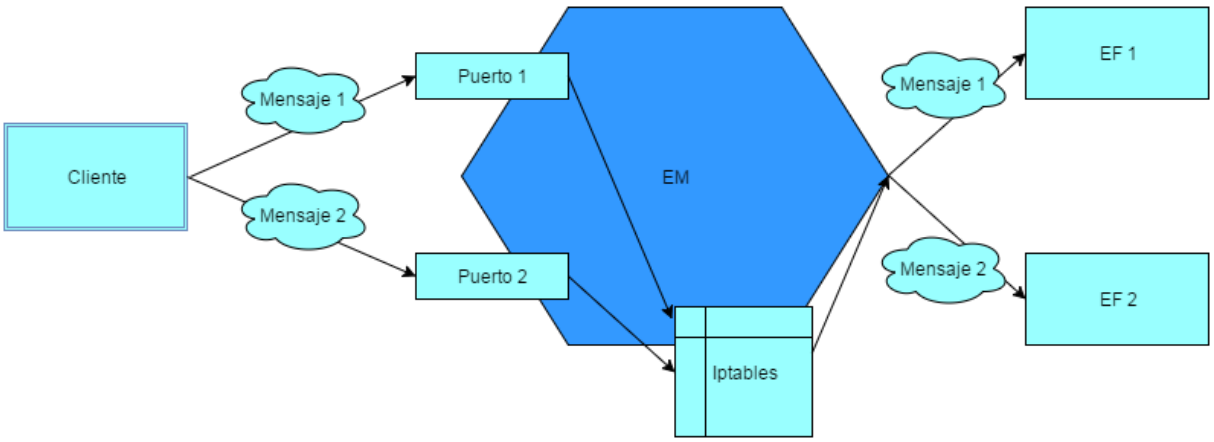

Ilustración 15: Esquema IpTables

De esta forma es posible interceptar los paquetes que entren a un determinado puerto, y reenviarlos hacia el EF deseado. Del mismo modo, permite interceptar y tratar el retorno del paquete enviado al EF.

Así que, gracias a *Iptables* [6] podremos configurar el módem como un NAT estático que permita redireccionar hacia los diversos EFs.

Para poder configurarlo, la interfaz web proporciona un formulario por cada puerto a redireccionar, dónde se requiere rellenar los siguientes campos:

- Puerto de acceso: Correspondiente al puerto del EM que será redireccionado a un determinado EF.
- Dirección de destino: Espera la dirección IP del EF.
- Puerto de destino: Corresponde al puerto destino del EF.
- Protocolo: Tipo de tráfico a redireccionar (TCP, UDP o ambos)

A continuación, se muestra la pantalla de la interfaz encargada de la configuración de los puertos:

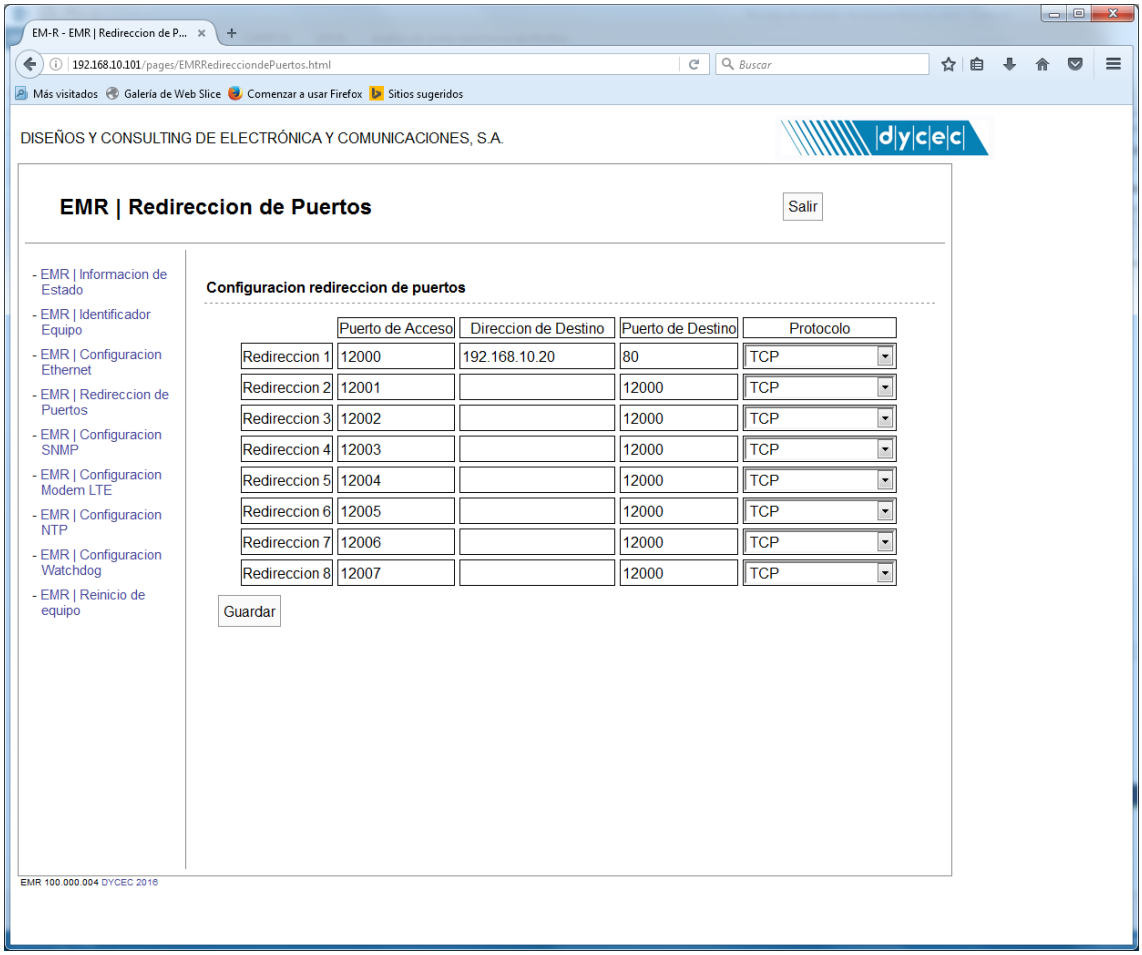

Ilustración 16: Pagina de redirección de puertos

Para cada línea editada por el usuario, se ejecutan las siguientes reglas

/sbin/iptables -t nat -A PREROUTING -p **PROTOCOLO** --dport ***PUERTO_ACCESO*** -j DNAT --to-destination ***DIRECCION_DESTINO***: ***PUERTO_DESTINO***

/sbin/iptables -t nat -A POSTROUTING -d ***DIRECCION_DESTINO*** -j SNAT --to-source **DIRECCION_ORIGEN**

Los primeros parámetros son comunes en ambos comandos:

- -t nat: Indica que el comando de iptables operará sobre la tabla NAT.
- -A: Indica que el comando se añadirá al final de la lista de operaciones de iptables.

A continuación se disecciona el primer mensaje, encargado de redireccionar los paquetes que entran al EM hacia el EF deseado:

- PREROUTING: Fuerza a iptables a alterar el paquete según entra al EM.
- -p [PROTOCOLO]: Marca el protocolo del paquete que será modificado, en el caso de que se hayan seleccionado ambos, se duplica esta línea con ambos protocolos.
- --dport [Puerto]: Indica el puerto del EM en el que se está escuchando dicha regla.
- -j DNAT: Especifica que se va a actuar sobre la tabla Destination NAT (que contiene las direcciones de destino de las conexiones).
- --to-destination: Especifica la nueva dirección y puerto de destino.

La segunda regla modifica la dirección de origen del paquete por la dirección del EM, de esta manera la respuesta del mensaje llegará al EM:

- POSTROUTING: Fuerza a iptables a alterar el paquete cuando está a punto de abandonar el EM, de modo que cualquier servicio de la máquina podrá ver la dirección de origen del cliente antes de que la regla la modifique.
- -d [dirección origen]: Indica la dirección destino de los paquetes que serán modificados por esta regla.
- -j SNAT: Especifica que se va a actuar sobre la tabla Source NAT (que contiene las direcciones de origen de las conexiones).
- --to-source: Especifica la nueva dirección y puerto de origen.

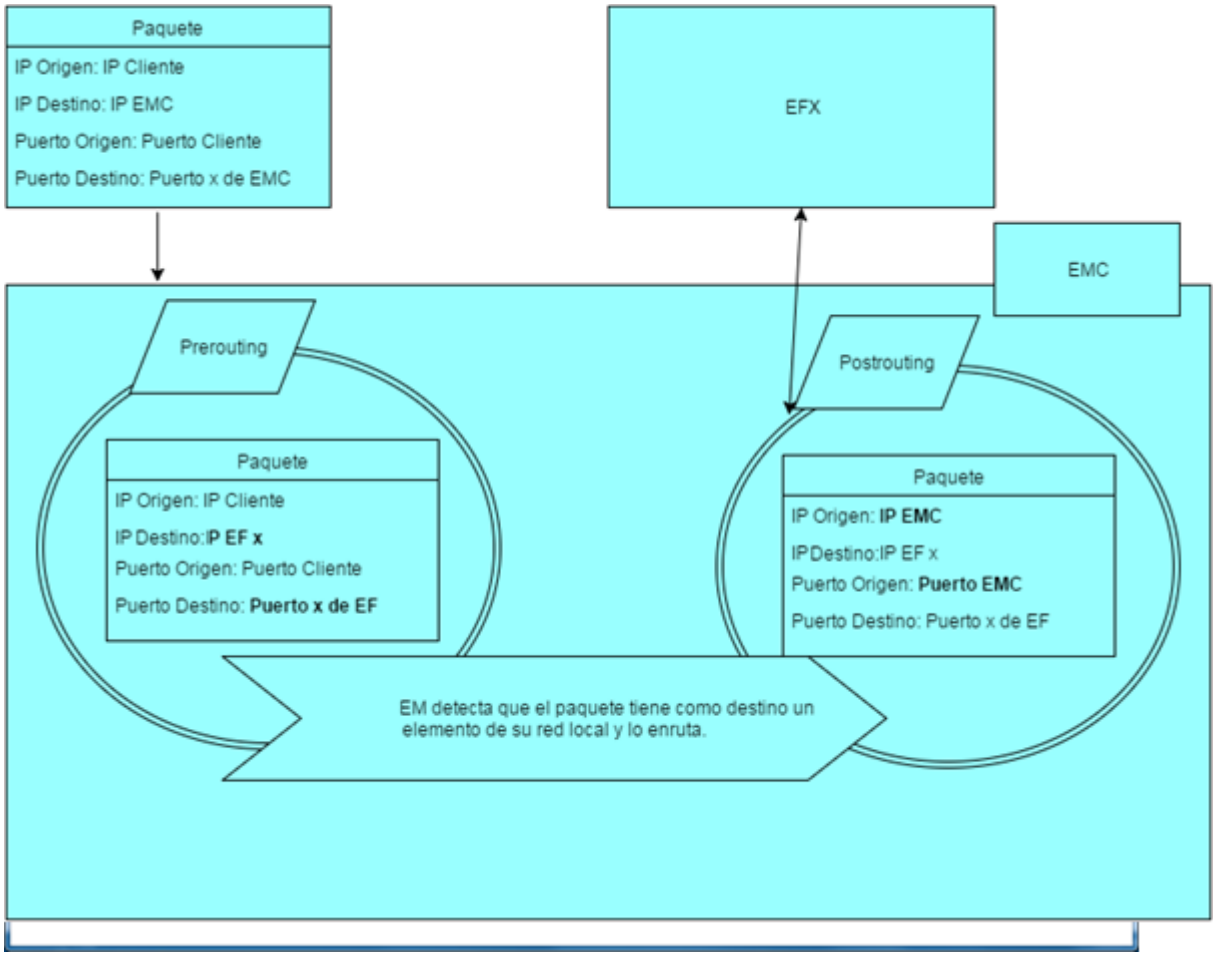

Ilustración 17: Modificaciones en el paquete

Tras lanzar el comando, se almacena en un fichero de configuración para mantener la persistencia de las tablas de mapeo en caso de que el EM tenga que reiniciarse.

4.4.2. Detección y envió de paquetes TRAP a cliente

Uno de los requisitos de la aplicación consiste en la recepción de las señales trap SNMP de los EFs por parte del EM y su posterior reenvío a los clientes.

Las señales trap, son mensajes (generalmente alertas) generados por los EFs.

Para realizarlo se ha optado por la biblioteca de código abierto *Net-SNMP*, en concreto de su función *snmptrapd* [5] [6] que permite la recepción de traps y su posterior reenvío a la dirección deseada.

Gracias a esta función el EM podrá escuchar los traps en el puerto estándar definido para ello, y reenviarlos a las direcciones configuradas previamente.

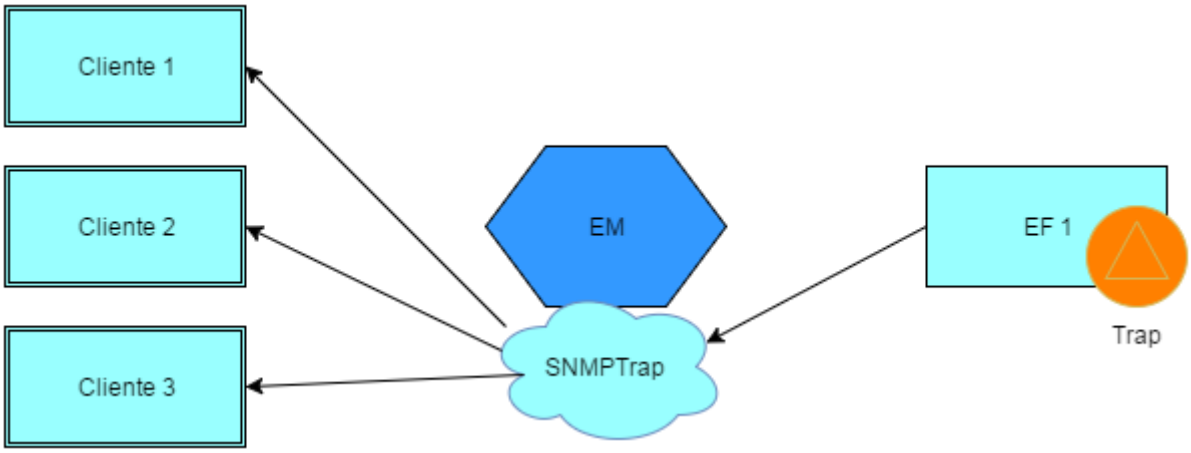

Ilustración 18: Esquema redirección Trap

Los EFs implementan trap SNMP para enviar sus alarmas, mediante la configuración de estos EF, la dirección de estas tramas corresponderá a la dirección del EM, el cual se encargará de reenviar a sus distintos clientes.

Para realizar esto, se utiliza la biblioteca *Net-SNMP*.

En este caso, desde la interfaz web se permite la configuración de tres direcciones a las que se reenviarán dichas tramas, cada dirección corresponde a un cliente.

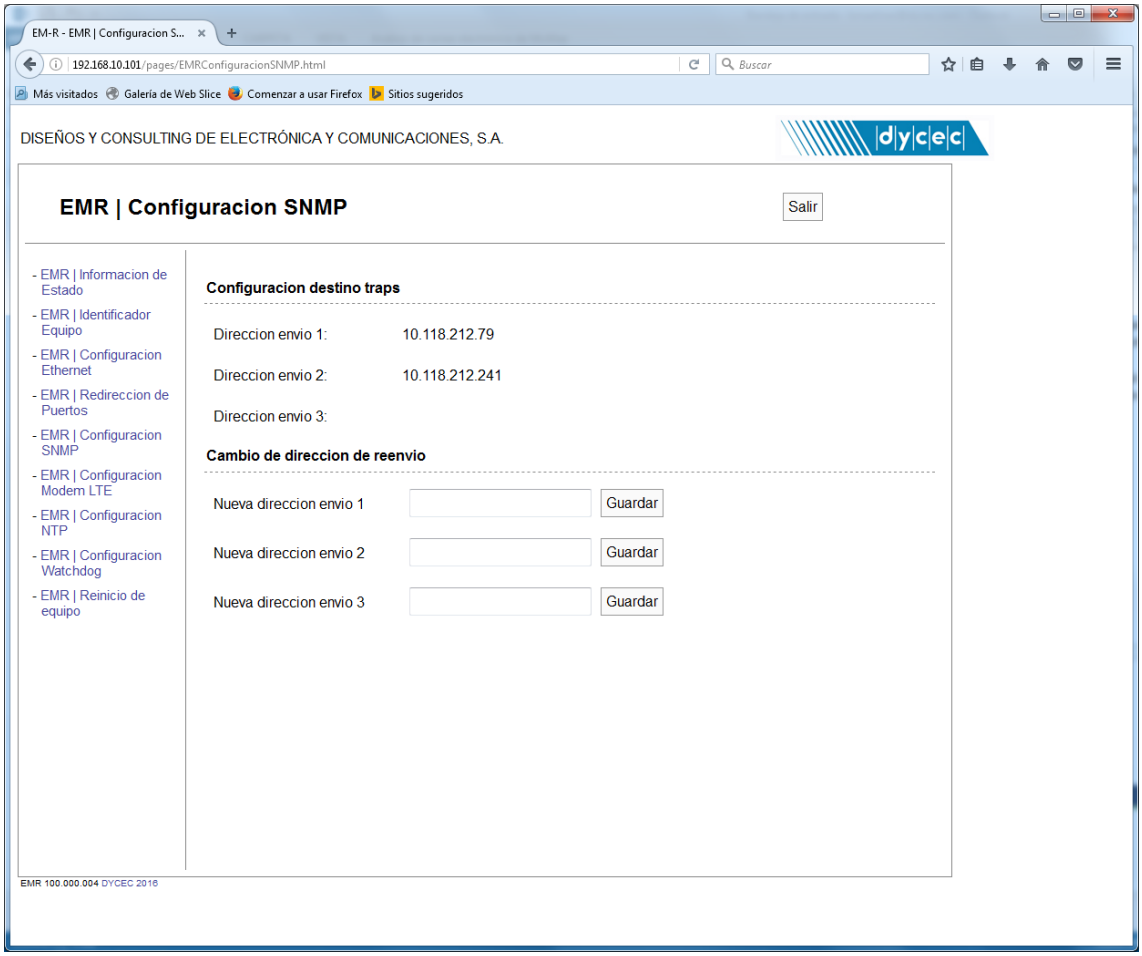

Ilustración 19: Configuración de direcciones de reenvío

Para cada dirección se debe modificar el fichero de configuración de *snmptrapd* (/etc/snmp/snmptrapd.conf) con la siguiente línea:

```
forward default direccion
```

Adicionalmente, es necesario incluir la siguiente línea, que fuerza a snmptrapd a aceptar todos los mensajes entrantes.

```
disableAuthorization yes
```

```
def escribe_ip_reenvio():
  reemplazo = ''
  sennal_hup_snmptrapd = "killall -1 snmptrapd" –fuerza a snmptrapd a releer su fichero de conf
  for i in range(1,4):
   if (lee_direccion_envio_traps(i) != '\n'):
    reemplazo += "forward default " + lee_direccion_envio_traps(i)
  reemplazo += "\ndisableAuthorization yes\n"
  f = open(FILE_SNMPTRAPD_CONF, 'w')
  f.write(reemplazo)
  f.close()
  os.system("/bin/sync")  –fuerza a escribir el fichero en disco duro
  os.system(sennal_hup_snmptrapd)
```

De este modo, un fichero de configuración *snmptrapd* quedaría así:

```
forward default dirección1
forward default dirección2
disableAuthorization yes
```

4.4.3. Envío de latido periódico

Uno de los requisitos del sistema consiste en la necesidad de enviar un latido periódico con información del sistema a un determinado cliente.

Deben enviarse los siguientes datos:

- Id del sistema (configurable).
- IP asignada en la conexión módem.

Para realizarlo se genera un mensaje trap SNMP propio del sistema.

Esta información se enviará mediante *snmptrap*. En un primer acercamiento se trató de realizar la comunicación mediante funciones de la biblioteca estándar de Python, sin embargo, esto generaba problemas con algunos mensajes, posiblemente por conflictos con la biblioteca net-snmp (necesaria para reenviar los mensajes de los EFs al cliente).

Para abordar este módulo se utiliza la función snmptrap de la biblioteca *net-snmp*, esta función es similar a *snmptrapd*, sin embargo, no se ocupa de interceptar mensajes, porque en este caso se generarán en el propio equipo.

En primer lugar, es necesario obtener la dirección IP de los clientes, que se corresponden con las configuradas en la tabla de reenvíos trap.

A continuación, se debe permitir al usuario configurar el identificador del equipo, que se mostrará en el cliente con el trap asociado. Para hacerlo se le proporciona la siguiente pantalla en la interfaz web.

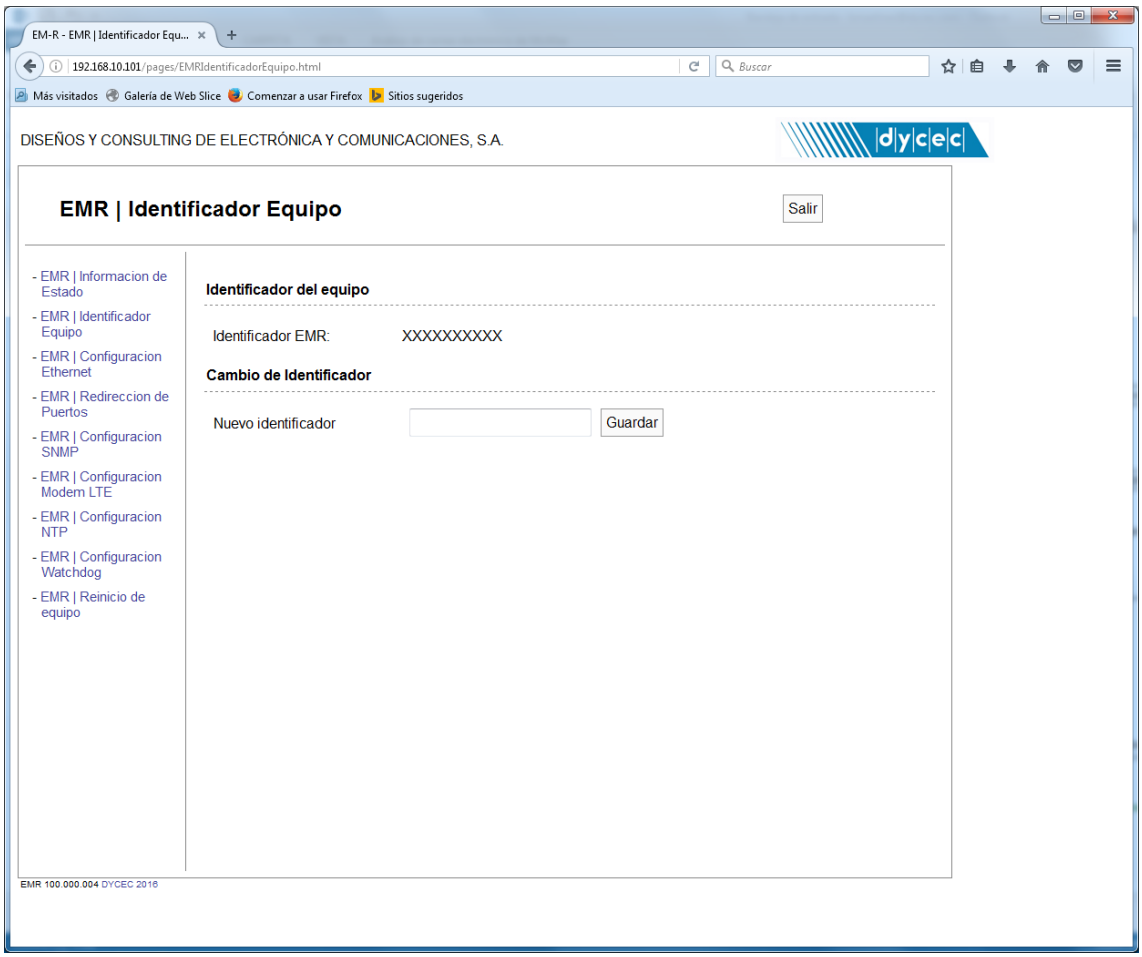

Ilustración 20: Identificador del EM

Cuando se guarde el identificador, se escribirá en el archivo /etc/ID_EMR.conf

Por otra parte, las direcciones donde se enviarán los latidos periódicos serán las mismas que se configuraron para el reenvío de traps de los EFs.

```
def escribe_direccion_envio_traps(nueva_dir, num):
  patron = "envio_traps_direccion_" + str(num)
  reemplazo = patron + ' ' + nueva_dir.split(' ')[0] + '\n'
  reemplaza_linea(FILE_SNMP_CONF, patron, reemplazo)
  escribe_ip_reenvio()
```

De manera similar al apartado anterior (Ver 4.4.2), el envío de traps mediante snmp requiere modificar el archivo FILE_SNMP_CONF, un archivo de configuración con dos direcciones quedaría así.

```
envio_traps_direccion_1 192.168.10.100
envio_traps_direccion_2 10.118.212.241
envio_traps_direccion_3
```

Por último, el script desarrollado y lanzado desde crontab se encarga de lanzar el latido periódicamente, informando de la situación del equipo periódicamente.

```
def latido():
  i = 0
  num_envios_max = 3
  ip_reenvio = ["","","]
  while i < num_envios_max:
    ip_reenvio[i] = lee_direccion_envio_traps(i+1)[:-1]
    i += 1
  j = 1
  for direccion in ip_reenvio:
    if direccion == '':
      continue
    nombre=getID()[:-1].ljust(32)
    trap_parte_1 = time.strftime("%d/%m/%y,%H:%M:%S") + ',' + nombre
    trap_parte_3 = ',EM,A/A/-/-/-/-/-/-,Mod.Telem:-,Mod.TyH:-/-/-/-/-/-/-/-
,R/R/R/R/R/R/R/-/-/-/-/-/-/-/-,V. APLICACION: ' + lee_version()
    command = "/usr/local/bin/snmptrap -v 2c -c public '" + direccion + "' "
1.3.6.1.6.3.1.1.5.1 1.3.6.1.4.1.12592.6.1.1.0 i 25 1.3.6.1.4.1.12592.6.3.1.0 s '" +\
        time.strftime("%d/%m/%y %a %H:%M:%S") + "' 1.3.6.1.4.1.12592.6.4.1.0 i
11 1.3.6.1.4.1.12592.6.5.1.0 s 'XXX' 1.3.6.1.4.1.12592.1.2.1.0 s '" +\
        trap_parte_1 + "," + getIP('ppp0') + trap_parte_3 + """
    os.system(command)
    time.sleep(1)
t = perpetualTimer(3600,latido)
t.start()
```

4.5. Módulo de generación de informes

Para resolver los requisitos funcionales de generación de informes, se ha decidido mantener dos ficheros de log:

- reinicios.log: Almacena los reinicios del sistema.
- estado_modem_conexion.info: Almacena periódicamente los datos de cobertura del módem.

Para gestionar el registro de reinicios del sistema, se ha desarrollado un script encargado de escribir la fecha y hora del sistema:

```
import time


fichero = "/etc/RMD/log/reinicios.log"

f = open(fichero, 'a')

f.write(time.strftime("%d/%m/%y,%H:%M:%S") + ' : Inicio sistema\n')

f.close()

os.system("/bin/sync")
```

Para que este script se ejecute en cada reinicio del EM, se añade la siguiente instrucción en el *crontab*:

```
@reboot /usr/bin/python /usr/local/bin/log_init.py
```

Para la generación de informes periódicos con la cobertura del módem se escriben periódicamente en un fichero los datos de conexión del módem.

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-

# First we must import PyBBIO:
from bbio import *
# Then we can import BBIOServer:
from bbio.libraries.DycecLib import *
from threading import Timer,Thread,Event

t = perpetualTimer(4, pide_info_modem_conexion)
t.start()
```

Este script pide información del módem y la escribe en un fichero periódicamente (mediante la función pide_info_modem_conexion).

Los resultados los guarda en /var/tmp/estado_modem_conexion.info. Por ejemplo:

```
28/12/16 Wed 16:58:42
tipo_conexion "WCDMA"
rssi (cobertura) -70 dBm
rscp -77 dBm
ecio -7.0 dB
subtipo "DC-HSPA+"
```

4.6. Módulo de monitorización

Este módulo está enfocado en la resolución en los requisitos no funcionales, relacionados con la disponibilidad.

Las funcionalidades pertenecientes a este módulo pretenden dotar al sistema de robustez y capacidad para responder ante fallos para mantener el servicio. Para realizarlo es posible distinguir entre problemas ocasionados por la misma placa y problemas ocasionados por la conexión o módem.

4.6.1. Monitorización de la placa

Este módulo aporta las funcionalidades necesarias para garantizar la respuesta del EM ante fallos relacionados con su placa BeagleBone.

4.6.1.1 Monitorización del correcto funcionamiento de la placa

El sistema debe continuar proporcionando servicio ante cualquier bloqueo de la placa, además debe mantener la configuración que asegura su funcionamiento como EM cuando se fuerce un reseteo de fábrica.

Para asegurar el correcto funcionamiento de la placa, se utiliza el watchdog del kernel Linux.

De manera que gracias al watchdog del sistema se pretende:

- Asegurar la disponibilidad de la placa en todo momento.
- Asegurarlo de manera que cause el menor impacto en el sistema.

Es posible escribir en el watchdog del kernel de linux [4], este watchdog se mantiene a la espera de escritura, de manera que si en un minuto no se ha podido escribir reinicia el sistema.

A continuación se muestra el código del script indicado para realizar esta tarea:

```
import time
import os


os.nice(20)
wd = open("/dev/watchdog", "w+")
while 1:
    wd.write("\n")
    wd.flush()
    time.sleep(5)
```

Este script abre el watchdog del kernel y escribe cada 5 segundos. Además, indica que el proceso es de baja prioridad mediante la función *nice* [5].

4.6.1.2. Monitorización del forzado del reseteo de la placa

El sistema debe seguir dando servicio en el caso en el que se resetee la placa mediante hardware, para eso se ha desarrollado un script que observa este comportamiento en la placa y toma las medidas necesarias para mantener la funcionalidad.

Para realizarlo, se definen las operaciones que se realizarán con los pines de la placa:

```
def setup():
  pinMode(USR0, OUTPUT)
  pinMode(USR1, OUTPUT)
  pinMode(USR2, OUTPUT)
  pinMode(USR3, OUTPUT)
  pinMode(GPIO1_17, OUTPUT)
  pinMode(GPIO3_21, INPUT)
  pinMode(GPIO3_19, INPUT)
```

En este caso se han definido el pin GPIO3_19 como pin de entrada, y los pines USR0,1,2 y 3 como pines de salida.

Los PINES USR0,1,2 y 3 son los LEDs de la placa. GPIO1_17 se usa a modo de señal continuamente a nivel alto. El pulsador cruza GPIO1_17 con los otros GPIO definidos como entrada, que son los que se leen para saber si se ha pulsado el botón de configuración de fábrica. Si es así, se encienden los 4 LEDs

Para después lanzar una operación periódica que comprueba la operación a realizar si se resetea el dispositivo.

```
def loop():
  state = digitalRead(GPIO3_19)
  if (state == 1):
    while i < 3:
      digitalWrite(USR0, HIGH)
      digitalWrite(USR1, HIGH)
      digitalWrite(USR2, HIGH)
      digitalWrite(USR3, HIGH)
      delay(50)
      i += 1
    os.system("/usr/local/bin/copia_ficheros_ori.sh")
    os.system("/usr/bin/killall wvdial")
    os.system("/bin/sleep 3; /usr/local/bin/devmem2 0x47401c60 b 0x00")
    os.system("(/bin/sleep 17; /sbin/reboot -f)&")
    return
  delay(1000)
run(setup, loop)
```

Cuando se detecta que se pulsa el reseteo del equipo, se encienden los pines de salida para advertir al usuario, después se realizan las operaciones necesarias para resetear el sistema con las funcionalidades añadidas y se reinicia la placa.

4.6.2. Monitorización de la red

El sistema debe saber reponerse correctamente ante cualquier problema de red, para asegurarlo conviene comprobar los siguientes puntos:

- El EM tiene conexión a una dirección IP y puerto configurados. con la idea de reiniciar el EM si no tiene conexión con un determinado cliente.
- La conexión entre el módem y la placa es correcta.
- El sistema mantiene su configuración tras un cambio en su dirección IP (algo que sucede con IP dinámicas).

4.6.2.1. Comprobación de conexión con el cliente

Con esta funcionalidad se pretende comprobar que la conexión con un determinado cliente sea satisfactoria.

Para realizarlo, se permite la configuración del servicio que realizará las conexiones mediante la interfaz web. Los campos configurables serán los siguientes:

- Dirección conectividad: Dirección IP del destinatario con el que se desea monitorizar la conexión. Si el campo se guarda vacío la funcionalidad se desactiva.
- Puerto TCP conectividad: El puerto en el que se realizará la conexión.
- Reintentos: Número de reintentos de conexión.
- Tiempo entre reintentos.

Una vez guardados los parámetros, se almacenan en el archivo */etc/RMD/watchdog.conf*.

Por otra parte, mediante *crontab* se lanza el servicio: un script que comprueba la conexión con los parámetros configurados, y que reinicia el sistema si se superan los reintentos fallidos (Véase el anexo D).

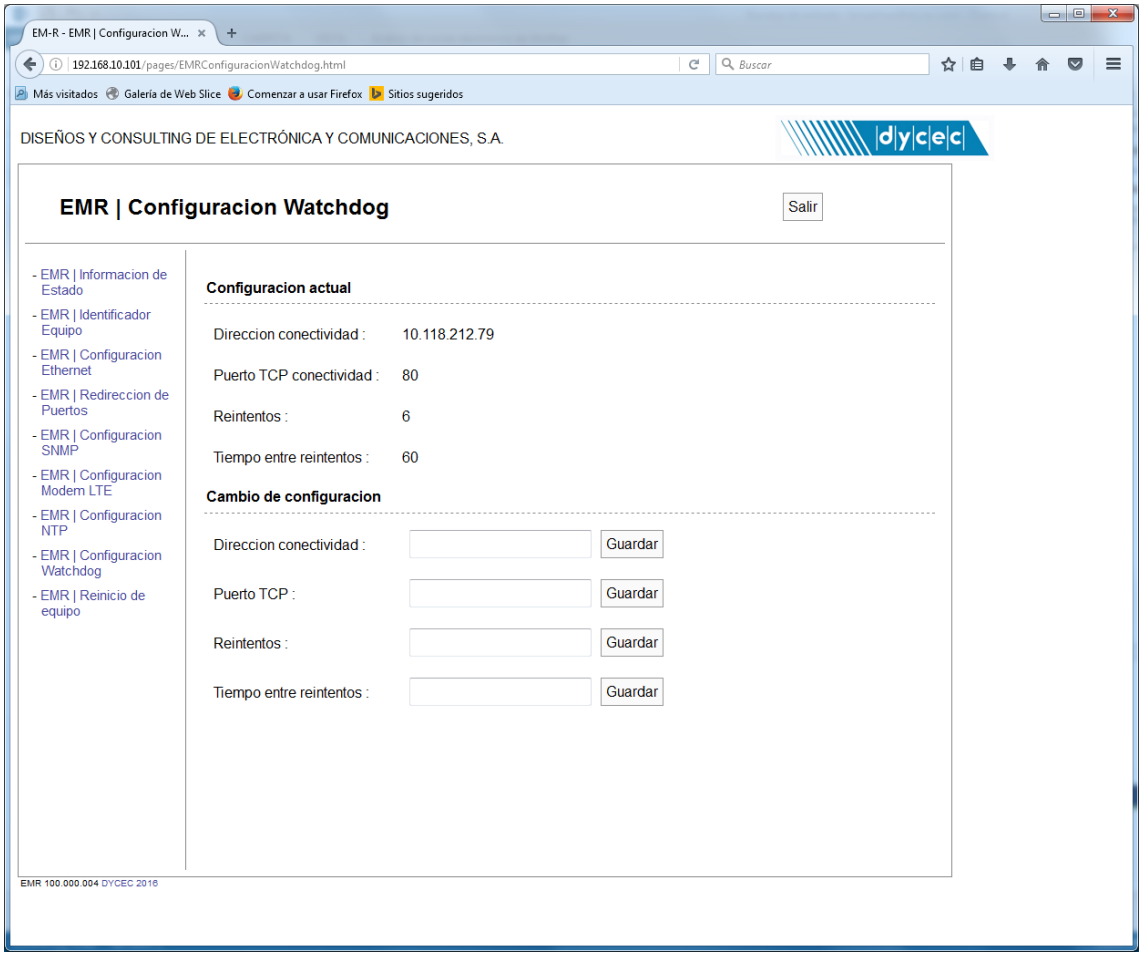

Ilustración 21: Configuración del watchdog

Este script, recupera los parámetros almacenados en fichero y trata de realizar conexión con la dirección y puerto indicados mediante un socket de conexión.

Si hay algún fallo en la conexión, aumenta el número de reintentos fallidos (en un fichero de configuración), y en el caso que se haya superado el máximo, reinicia el módem.

En el caso de que la conexión sea correcta se cierra el socket y se reinicia el contador de reintentos.

4.6.2.2. Comprobación de bus USB

Puede darse el caso de que el módem externo no se haya encendido correctamente después de una situación en el que se haya reiniciado el sistema de forma forzosa (como, por ejemplo, en situaciones de perdida de alimentación).

Para estos casos es necesario un mecanismo capaz de detectar el problema y reiniciar el equipo para solucionarlo.

Para realizarlo, primero es necesario comprobar si existe conexión con la dirección de comunicación del módem */dev ttyUSB0* (Ver 4.3.1.1).

```
def exists(path):
    try:
        os.stat(path)
    except OSError:
        return False
    return True
```

Mediante la función *os.stat* (ruta) [4] es posible comprobar si existe la dirección de configuración del módem, y de este modo saber si esta encendido correctamente.

En este caso se han definido 10 reintentos para realizar esta comprobación, en el caso de que se superen se reiniciará el equipo.

```
def check_ttyUSB0():
  global cont
  path = "/dev/ttyUSB0"
  if exists(path):
    cont = 0
  else:
    cont = cont+1
  if (cont > 10):
    os.system("/usr/local/bin/devmem2 0x47401c60 b 0x00")
    os.system("sleep 17;/sbin/reboot -f")
t = perpetualTimer(5,check_ttyUSB0)
t.start()
```

5. PRUEBAS

En este apartado se documentan las pruebas realizadas sobre el software del EM.

5.1. Pasos previos: carga del firmware

En este apartado se explica cómo configurar el EM para pasar el proceso de pruebas, para realizarlo hay que seguir los siguientes pasos:

1. **Generar la tarjeta microSD para la copia del Firmware (sólo la primera vez):**
    - Descomprimir el fichero con la versión del Firmware registrada: **emr-bbb-xxx.xxx.xxx.rar.**
    - Copiar la imagen **emr-bbb-xxx.xxx.xxx.img** a la tarjeta microSD con el programa **Win32DiskImager:**
        - Seleccionar la imagen a copiar.
        - Seleccionar la unidad en la que copiar.
        - Pulsar opción "**write**".

- *Nota: La imagen a cargar está preparada para tarjetas de 4GB o superior. En este documento se indica que se tiene que usar una tarjeta de 8GB para garantizar que la imagen entra correctamente. Las tarjetas de 4GB podrían dar problemas, ya que no todas son exactamente del mismo tamaño y al hacer la copia la aplicación podría indicar que no tiene suficiente espacio.*

2. **Copia del Firmware de tarjeta microSD en tarjeta Beaglebone (carga del Firmware):**
    - Introducir tarjeta microSD con la imagen cargable en la tarjeta Beaglebone, en el conector "**microSD Card**" próximo al conector **USB**.
    - Conectar la alimentación del equipo y pulsar **S2 hasta que los leds de indicación de la tarjeta comiencen a iluminarse con un patrón continuo de vaivén.**
    - El equipo copiará la imagen de la tarjeta. Durante el proceso, que tarda unos 15 minutos, los leds de indicación de la tarjeta seguirán iluminándose con el patrón continuo de vaivén. La copia acaba cuando se detienen y se apagan los leds.
    - Sacar la tarjeta microSD (si se deja puesta se repite el proceso de copia).
    - Quitar alimentación y volver a conectarla.

- *Nota: En algunas tarjetas se han detectado problemas en la copia de la imagen del Firmware. Si al realizar el proceso de copia como se describe en este punto, el patrón de vaivén de los leds de indicación no ocurre, es que la tarjeta está mal copiada o no funciona correctamente. En tal caso, repetir el proceso de carga desde el punto 1 usando otra tarjeta microSD de 8GB.*

3. Desactivación del watchdog de conexión:

Para las pruebas es necesario desactivar el watchdog de conexión para que no se reinicie constantemente el equipo. Para ello, en el primer arranque del equipo entrar por telnet:

telnet 192.168.10.36

user: admin

password: dycec

Pasar a modo superusuario:

su

Password: dycec@1234

Desactivar el watchdog con el comando:

cp /etc/RMD/watchdog.conf.off /etc/RMD/watchdog.conf

4. Copia del loader:

El loader que se debe copiar está incluido en la propia imagen del Firmware, en una carpeta específica. Para la copia del loader hay que seguir los siguientes pasos:

Ejecutar los siguientes comandos para la copia del loader y verificar la respuesta:

cd /root/u-boot-2016.05

dd if=MLO of=/dev/mmcblk0 bs=512 seek=256 count=256 conv=notrunc

- 152+1 records in
- 152+1 records out
- 77928 bytes (78 kB) copied, 0.0183518 s, 4.2 MB/s

(El tiempo y la tasa de transferencia pueden variar).

dd if=u-boot.img of=/dev/mmcblk0 bs=512 seek=768 count=1024 conv=notrunc

- 761+1 records in
- 761+1 records out
- 389860 bytes (390 kB) copied, 0.0253786 s, 15.4 MB/s

(El tiempo y la tasa de transferencia pueden variar).

5. Desconectar la alimentación y conectarla de nuevo.

5.2. Pruebas a realizar

En este apartado se documenta las pruebas realizadas.

6.2.1. Verificación de la instalación del firmware

Para comprobar que el firmware se está ejecutando correctamente abrir una conexión telnet a la dirección IP configurada por defecto en el EM y entrar con el usuario "**admin**" y password "**dycec**". Ejecutar el comando ***ps -ax***.

Comprobar que se están ejecutando los procesos que se muestran en la pantalla a continuación:

```
1001 ?        S1     0:21 /usr/bin/python /usr/local/bin/comprueba_inactividad_conectados.py
1003 ?        S1     0:21 /usr/bin/python /usr/local/bin/watchdog-modem-en-bus-usb.py
1007 ?        S1     0:18 /usr/bin/python /usr/local/bin/watchdog_conexion.py
1008 ?        S      0:00 [file-storage]
1013 ?        S1     0:23 /usr/bin/python /usr/local/bin/trap-v2c-latido-periodico.py
1015 ?        S      0:04 /usr/bin/python /usr/local/bin/check_configuracion_fabrica.py
1023 ?        SN     0:00 /usr/bin/python /usr/local/bin/watchdog_sistema.py
1037 ?        S1     0:13 python /usr/local/lib/PyBBIO/examples/Confweb_dycec.py
```

Ilustración 22: Listado de procesos

En la lista de procesos deben aparecer 7 procesos PYTHON. En el ejemplo son los que están entre el número 1001 y el 1037.

6.2.2. Pruebas del equipo

A continuación, se enumeran las pruebas que hay que realizar sobre el equipo.

6.2.2.1. Prueba interfaz ethernet

Probar a hacer un ping al equipo y ver que responde. Abrir un navegador y acceder a la dirección http://192.168.10.36. Comprobar que se abre la página Web del equipo y se pueden modificar los parámetros.

6.2.2.2. Prueba módem 4G

Por defecto el equipo EMR lleva configurado el APN para conectarse a la red privada de ORANGE. En caso de introducir para las pruebas una tarjeta diferente a una tarjeta ORANGE

habría que cambiar el APN en la página web del EMR en la opción de **EMR/Configuración Modem LTE**. En la imagen se muestra la configuración para el operador MOVISTAR.

El APN/usuario/contraseña para la red de ORANGE privada con IP fija es:

- APN: ---
- Usuario: ---
- Contraseña: ---

Desde la conexión Ethernet, acceder por http al menú de configuración del equipo, utilizando como usuario: *admin* y como contraseña: *dycec.*

Ir al menú **EMR | Información de Estado** y comprobar que se obtiene IP desde la conexión del modem LTE.

6.2.2.3. Verificación de carga configuración de fábrica

Desde la conexión Ethernet, acceder al menú del equipo y modificar los datos de algún menú de configuración (por ejemplo, **EMR | Configuración Modem LTE**). Guardar los cambios y comprobar apagando y encendiendo el equipo, que los cambios quedan guardados.

Una vez verificado que los cambios se mantienen, hay que pulsar el botón situado en el exterior de la tarjeta procesadora, en la zona central del conector **P9**. En este momento se enciende el led **D5** (o **USER3**). El equipo tarda 20 segundos en realizar el reinicio para asegurar que el módem se apaga correctamente, más el tiempo necesario para que arranquen todos los procesos.

Esperar un minuto desde que se pulsó el botón y comprobar que los cambios realizados sobre la configuración del equipo vuelven a quedar con los valores de fábrica.

Al realizar la prueba del botón de fábrica también se activa el watchdog de conexión, que está configurado por defecto para la red de ORANGE.

6. CONCLUSIONES Y TRABAJO FUTURO

Como conclusión cabe destacar que se ha realizado un trabajo profesional, que cubre las funcionalidades requeridas al inicio del proyecto.

En el transcurso del trabajo se han realizado tareas de investigación en campos como el control de módems mediante Python y el kernel linux, el uso de *iptables* [10] para la redirección de paquetes y de snmp para la intercepción y reenvío de traps.

Además, se ha analizado las posibles debilidades de un sistema que requiere estar dotado de la máxima autonomía, diseñando y desarrollando las herramientas que permitan detectar situaciones inconvenientes y poniéndoles solución.

Por último, se ha dotado al sistema de una interfaz simple, pero efectiva, para poder configurar los parámetros que se han considerado más apropiados.

El proyecto finaliza con la funcionalidad básica del EM completada, y presenta varias áreas susceptibles de mejora como trabajo futuro.

Una de las funcionalidades que se planean incorporar en un trabajo futuro, es la comunicación de los EFs con el EM mediante puertos serie.

Se ha considerado la entrada de datos de EFs mediante RS-232 (V.24) [5] y mediante RS-422 [6], debido a que algunos EFs lo utilizan y ampliaría el espectro de uso del EM.

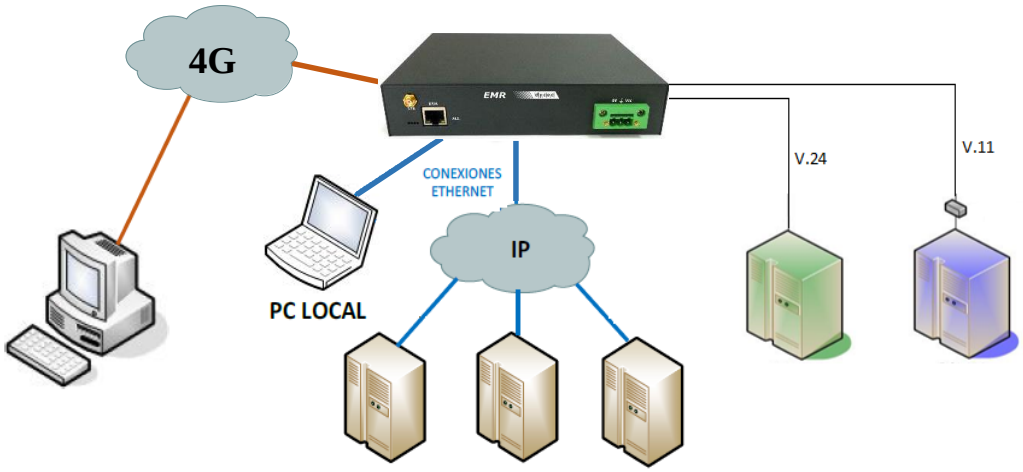

Ilustración 23: Esquema de sistema en el futuro

También se considera la incorporación de entradas digitales, para poder recoger las medidas de los sensores disponibles en los emplazamientos de red.

Del mismo modo sería interesante la incorporación de Telemandos (salidas digitales que actúan como relés) para actuar sobre el EF (corte de alimentación, botón de encendido, etc.)

Por último, se considera la opción de integrar el módulo encargado de la conversión de la alimentación del EM dentro del circuito interno del sistema.

De manera, que el sistema queda finalizado en una primera versión completamente funcional que podrá ser mejorada dependiendo de las necesidades futuras.

BIBLIOGRAFÍA

[1] Ubuntu. [En línea]. Available: https://help.ubuntu.com/community/CronHowto.

[2] BeagleBoard. [En línea]. Available: http://beagleboard.org/latest-images.

[3] GrayCatLabs. [En línea]. Available: https://github.com/graycatlabs/PyBBIO/wiki/BBIOServer.

[4] https://linux.die.net. [En línea]. Available: https://linux.die.net/man/1/stty.

[5] wiki.archlinux.org. [En línea]. Available: https://wiki.archlinux.org/index.php/Wvdial.

[6] https://wiki.debian.org. [En línea]. Available: https://wiki.debian.org/es/NetworkConfiguration.

[7] «https://www.freebsd.org,» [En línea]. Available: https://www.freebsd.org/cgi/man.cgi?query=ntpdate&sektion=8.

[8] https://linux.die.net. [En línea]. Available: https://linux.die.net/man/8/hwclock.

[9] netfilter.org. [En línea]. Available: http://www.netfilter.org/documentation/HOWTO/es/NAT-HOWTO-6.html.

[10] netfilter.org. [En línea]. Available: http://www.netfilter.org/documentation/HOWTO/es/NAT-HOWTO-7.html.

[11] wiki.archlinux.org. [En línea]. Available: https://wiki.archlinux.org/index.php/Iptables_(Espa%C3%B1ol).

[12] http://net-snmp.sourceforge.net. [En línea]. Available: http://net-snmp.sourceforge.net/docs/man/snmptrapd.conf.html.

[13] http://net-snmp.sourceforge.net. [En línea]. Available: http://net-snmp.sourceforge.net/docs/man/snmptrapd.html.

[14] https://www.kernel.org. [En línea]. Available: https://www.kernel.org/doc/Documentation/watchdog/watchdog-api.txt.

[15] http://bencane.com. [En línea]. Available: http://bencane.com/2013/09/09/setting-process-cpu-priority-with-nice-and-renice/.

[16] https://docs.python.org. [En línea]. Available: https://docs.python.org/2/library/os.html#os.stat.

[17] Wikipedia. [En línea]. Available: https://es.wikipedia.org/wiki/RS-232.

[18] Wikipedia. [En línea]. Available: https://es.wikipedia.org/wiki/RS-422.

[19] GrayCatLabs, «https://github.com/graycatlabs/PyBBIO/wiki,» [En línea].

[20] Wikipedia. [En línea]. Available: https://en.wikipedia.org/wiki/Comparison_of_single-board_computers.

[21] ipset.netfilter.org. [En línea]. Available: http://ipset.netfilter.org/iptables.man.html.

[22] ipset.netfilter.org. [En línea]. Available: http://ipset.netfilter.org/iptables-extensions.man.html.

[23] Wikipedia. [En línea]. Available: https://en.wikipedia.org/wiki/IEEE_802.11_RTS/CTS.

[24] Wikipedia. [En línea]. Available: https://es.wikipedia.org/wiki/Point-to-Point_Protocol.

[25] https://embeddedfreak.wordpress.com. [En línea]. Available: https://embeddedfreak.wordpress.com/2010/08/23/howto-use-linux-watchdog/.

[26] http://www.tldp.org. [En línea]. Available: http://www.tldp.org/HOWTO/Modem-HOWTO-7.html.

ANEXOS

Anexo A: Esquemas de placas Raspberry

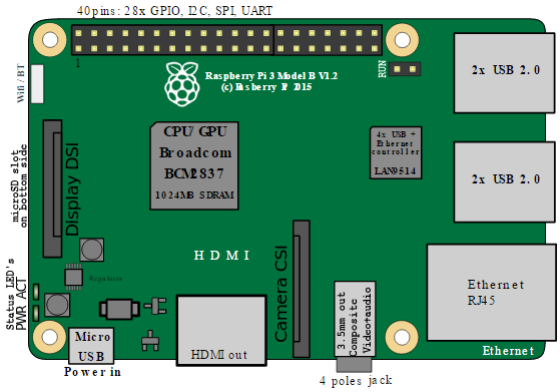

Ilustración 24: Esquema Raspberry Pi 3 B

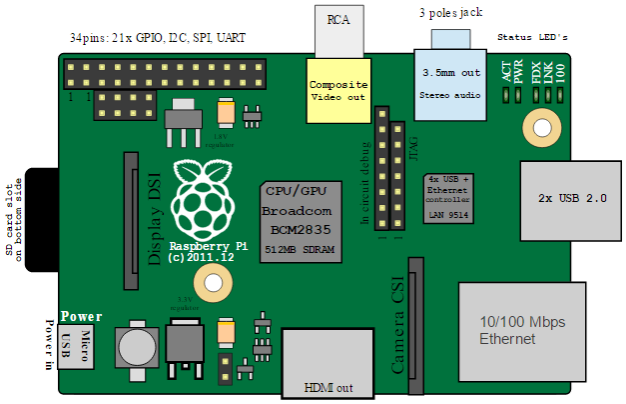

Ilustración 25: Esquema Raspberry Pi B

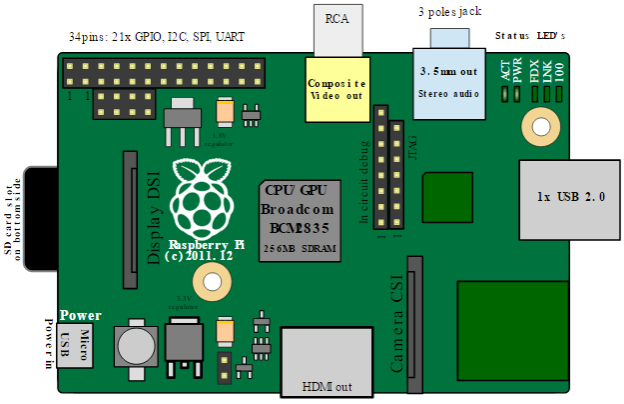

Ilustración 26: Esquema Raspberry Pi A

Anexo B: Crontab

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
*/3 * * * * /usr/local/bin/check_restart_ppp0.sh
1 * * * * /usr/local/bin/lanza_ntp.sh
@reboot /usr/bin/python /usr/local/bin/watchdog_sistema.py
@reboot /usr/bin/python /usr/local/bin/log_init.py
@reboot (sleep 40;/usr/local/bin/initmodem.sh)
@reboot /etc/init.d/confweb start
@reboot (sleep 10;/usr/bin/python /usr/local/bin/comprueba_inactividad_conectados.py)
@reboot /usr/bin/python /usr/local/bin/trap-v2c-latido-periodico.py
@reboot /usr/bin/snmptrapd -Le&
@reboot /usr/bin/python /usr/local/bin/watchdog_conexion.py
@reboot /usr/bin/python /usr/local/bin/check_configuracion_fabrica.py
@reboot (sleep 15; /usr/bin/python /usr/local/bin/watchdog-modem-en-bus-usb.py)
@reboot /usr/bin/python /usr/local/bin/get_datos_modem.py
@reboot /usr/bin/python /usr/local/bin/reload_net_options.py
```

Anexo C: Problema boot loader

Durante la fase de pruebas se detectó un bug en el u-boot que venía por defecto en Debian. El fallo consistía en que en los reinicios se producía ruido en el puerto de consola de la placa. Este ruido provocaba que el u-boot se parara debido a que la consola detectaba que se estaba pulsando una tecla, algo que por defecto detiene el arranque.

El u-boot se cambió para que sólo se pare si se escribe por consola la palabra "stop". Y da de margen 2 segundos para hacerlo.

Para conseguir ésto se bajó a la propia Beaglebone la última versión de u-boot.

Pasos para cambio de u-boot:

Lanzar los siguientes comandos para bajar el u-boot y adaptarlo a la BBB:

- `wget` ftp://ftp.denx.de/pub/u-boot/u-boot-latest.tar.bz2
- `tar -xjf u-boot-latest.tar.bz2`
- `cd u-boot-2015.10 (la fecha cambia)`
- `wget` https://rcn-ee.com/repos/git/u-boot-patches/v2015.10/0001-am335x_evm-uEnv.txt-bootz-n-fixes.patch `(la fecha depende del u-boot bajado)`
- `patch -p1 < 0001-am335x_evm-uEnv.txt-bootz-n-fixes.patch`

En nuestro caso se ha añadido:

```
En include/configs/am335x_evm.h

+#define CONFIG_AUTOBOOT_KEYED           1
+#define CONFIG_AUTOBOOT_STOP_STR        "stop"
```

La razón es para que la consola de arranque sólo se pare con la "s" y evitar los reinicios fallidos.

Una vez con el código bien configurado, se compila:

- `make distclean`
- `make am335x_boneblack_config`
- `make`

Una vez compilado se Copian el MLO y el u-boot.bin en el lugar adecuado de la MMC

- **`dd if=MLO of=/dev/mmcblk0 bs=512 seek=256 count=256 conv=notrunc`**
- `dd if=u-boot.img of=/dev/mmcblk0 bs=512 seek=768 count=1024 conv=notrunc`

Además, para que el loader modificado sea el que se copie en las tarjetas SD con la imagen completa, estos ficheros MLO y u-boot.img se han de copiar en /opt/backup/uboot/.

Anexo D: Script de monitorización de red

```
def check_conexion():
  host = str(lee_wd_ip())
  port = int(lee_wd_puerto())
  fallos_max = int(lee_wd_intentos())
  if ((host != '\n') and (host != '')):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
      s.settimeout(5)
      s.connect((host, port))
      s.shutdown(2)
      escribe_contador_fallos_conexion('0')
    except socket.error as e:
      fallos = int(lee_contador_fallos_conexion())
      if (fallos >= fallos_max):
        escribe_contador_fallos_conexion('0')
          os.system("/usr/bin/python /usr/local/bin/reinicia_modem.py")
          os.system("/usr/sbin/killall wvdial")
          os.system("sleep 17;/sbin/reboot -f")
      else:
        fallos += 1
        escribe_contador_fallos_conexion(str(fallos))
  else:
    escribe_contador_fallos_conexion('0')
tiempo = lee_wd_tiempo()
t = perpetualTimer(int(tiempo),check_conexion)
t.start()
```

Anexo E: Creación de imagen

A continuación, se detalla un manual para cargar el software generado en cualquier otra BeagleBone Black.

- Se introduce una SD en la ranura y se ejecuta el script disponible en Debian para BBB: */opt/scripts/tolos/eMMC/beaglebone-black- make-microSD- flasher-from- eMMC.sh*
- Este script genera una tarjeta SD autoejecutable en cualquier Beaglebone que copia el loader y el Sistema operativo que contienen.
- A partir de la tarjeta generada se crea una imagen de la misma con ayuda del programa **Win32DiskImager:**
    - Seleccionar la imagen donde copiar.
    - Seleccionar la unidad de la que copiar.
    - Pulsar opción "**read**".