UNIVERSIDAD DE EL SALVADOR

FACULTAD DE INGENIERIA Y ARQUITECTURA

ESCUELA DE INGENIERIA ELÉCTRICA

IMPLEMENTACIÓN XML-RPC PARA LA MONITORIZACIÓN DE MEDIDORES DE ENERGÍA ELÉCTRICA

PRESENTADO POR:

SAMUEL ANTONIO RENDEROS ALFARO

PARA OPTAR AL TITULO DE:

INGENIERO ELECTRICISTA

CIUDAD UNIVERSITARIA, FEBRERO 2017

UNIVERSIDAD DE EL SALVADOR

RECTOR INTERINO :

LIC. JOSÉ LUIS ARGUETA ANTILLÓN

SECRETARIA GENERAL :

DRA. ANA LETICIA ZAVALETA DE AMAYA

FACULTAD DE INGENIERIA Y ARQUITECTURA

DECANO :

ING. FRANCISCO ANTONIO ALARCÓN SANDOVAL

SECRETARIO :

ING. JULIO ALBERTO PORTILLO

ESCUELA DE INGENIERIA ELÉCTRICA

DIRECTOR :

ING. ARMANDO MARTÍNEZ CALDERÓN

UNIVERSIDAD DE EL SALVADOR

FACULTAD DE INGENIERIA Y ARQUITECTURA

ESCUELA DE INGENIERIA ELÉCTRICA

Trabajo de Graduación previo a la opción al Grado de:

INGENIERO ELECTRICISTA

Título :

IMPLEMENTACIÓN XML-RPC PARA LA MONITORIZACIÓN DE MEDIDORES DE ENERGÍA ELÉCTRICA

Presentado por :

SAMUEL ANTONIO RENDEROS ALFARO

Trabajo de Graduación Aprobado por:

Docente Asesor :

Dr. CARLOS EUGENIO MARTÍNEZ CRUZ

San Salvador, Febrero 2017

Trabajo de Graduación Aprobado por:

Docente Asesor :

DR. CARLOS EUGENIO MARTÍNEZ CRUZ

ACTA DE CONSTANCIA DE NOTA Y DEFENSA FINAL

En esta fecha, jueves 2 de febrero de 2017, en la Sala de Lectura de la Escuela de Ingeniería Eléctrica, a las 4:00 p.m. horas, en presencia de las siguientes autoridades de la Escuela de Ingeniería Eléctrica de la Universidad de El Salvador:

1. Ing. Armando Martínez Calderón
   Director

Firma: _____

ESCUELA DE INGENIERÍA ELÉCTRICA
DIRECCIÓN
FACULTAD DE INGENIERÍA Y ARQUITECTURA

2. MSc. José Wilber Calderón Urrutia
   Secretario

Firma: Wilber Calderón

Y, con el Honorable Jurado de Evaluación integrado por las personas siguientes:

Firma:

1- Dr. Carlos Eugenio Martínez Cruz
(Docente-Asesor)

2- MSc. Jorge Alberto Zetino Chicas

3- Ing. Walter Leopoldo Zelaya Chicas

Se efectuó la defensa final reglamentaria del Trabajo de Graduación:

IMPLEMENTACIÓN XML-RPC PARA LA MONITORIZACIÓN DE MEDIDORES DE ENERGÍA ELÉCTRICA

A cargo del Bachiller:

- RENDEROS ALFARO SAMUEL ANTONIO

Habiendo obtenido en el presente Trabajo una nota promedio de la defensa final: 9.5

(NUEVE CINCO)

Agradecimientos

Agradezco el apoyo incondicional de mi madre Teresa de Jesús Alfaro y de mi padre José Efraín Renderos, quienes desde un principio han sido mi mayor motivación en esta aventura académica. Es a ellos a quienes les debo este logro, por su constante apoyo y motivación durante toda mi carrera universitaria. A mi hermano Efraín Alexander Renderos quien me ha apoyado y ha sido un ejemplo a seguir a lo largo de toda la carrera.

Agradezco de una manera muy especial a mis hermanas Claudia Yesenia Renderos y Meybel Soraira Alfaro quienes siempre estuvieron pendiente y me brindaron todo su apoyo.

Agradezco a Dios por darme la oportunidad de culminar mis estudios universitarios, guiarme a lo largo de mi carrera, darme sabiduría, entendimiento y paciencia para alcanzar mis objetivos.

También quiero agradecer al Dr. Carlos Martínez por la oportunidad de trabajar en este proyecto y permitirme aplicar mi conocimiento y habilidades adquiridas durante toda la carrera.

A todos mis amigos y compañeros de estudio durante toda la carrera. A todos los docentes de la escuela de eléctrica. Gracias por ser parte de mi formación académica.

Gracias totales.

Samuel Antonio Renderos Alfaro

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1: Introducción

La Escuela de Ingeniería Eléctrica de la Universidad de El Salvador ha realizado importantes contribuciones en el tema de monitorización remota de medidores de energía eléctrica. La contribución más reciente se realizó desarrollando mejoras al sistema de medición que la Universidad de El Salvador tiene instalado en su campus central, incorporando persistencia, actualizando la manera de interrogar a los medidores y desarrollando una aplicación de visualización. Sin embargo, a pesar de todas esas contribuciones el sistema aún adolece de robustez y por lo tanto es necesario encontrar una técnica que permita mantener la integridad de los datos.

En el año 2012 la Universidad de El Salvador implementó una red de medición en treinta de sus subestaciones, para lo cual se instalaron treinta medidores que monitorean diferentes tipos de variables eléctricas. Además a cada medidor se le instaló un router inalámbrico tipo WiFi. Todos los router forman parte de una red malla o mesh [1]. En el año 2014 se realizaron los primeros trabajos de graduación utilizando la red de medidores, que consistieron en dotar a la red con la capacidad de almacenamiento, mediante un mecanismo de interrogación continúa. Para ello se aprovechó que los medidores permiten la comunicación mediante el protocolo MODBUS [1] [2].

Este trabajo de graduación tiene como finalidad mejorar la integridad de los datos del sistema actual de monitorización de medidores de energía eléctrica, actualizando la arquitectura de interrogación. Por una parte realizando y almacenando las mediciones *in situ* a través de los router instalados en cada medidor, utilizando el protocolo Modbus TCP/IP. Por otra parte enviando los datos, mediante el protocolo XML-RPC, a un servidor central configurado en una raspberry pi, para finalmente visualizarlos en una aplicación web utilizando la plataforma Google App Engine.

El trabajo se divide en tres partes principales. La primera corresponde a la interrogación y almacenamiento de las mediciones *in situ*, en la cual se requiere de la técnica de compilación cruzada para generar el programa de consulta y ejecutarlo dentro del router. Se implementa un método diferente de interrogar a la red de medidores, mediante una aplicación en C que se ejecuta dentro de cada router y realiza consultas periódicas a cada medidor, almacenando los resultados en una base de datos SQLite. La segunda parte tiene que ver con la implementación del protocolo XML-RPC para el envió de los datos, desde el router hacia el servidor, mediante una aplicación cliente XML-RPC programada en LUA que será capaz de consultar la base de datos SQLite y cada vez que se tengan nuevos datos, intentará enviarlos a una aplicación servidor XML-RPC programada en PERL. La última parte consiste en el almacenamiento y visualización de los datos utilizando la plataforma Google App Engine en una Raspberry Pi.

1.1 Interés de la investigación

Todo sistema de monitorización remota es medido por la integridad de sus datos. Solo aquellos sistemas que demuestren confiabilidad podrán ser catalogados como sistemas robustos dignos de ser adoptados por la industria. Este trabajo de graduación busca dar ese paso de calidad. Hasta la fecha el sistema de monitorización adolece del problema de pérdida de datos. Este trabajo de graduación se encargará de mejorar dicha situación.

1.2 Antecedentes

La Escuela de Ingeniería Eléctrica de la Universidad de El Salvador ha desarrollado varias aplicaciones en materia de monitorización remota de medidores de energía eléctrica. A lo largo de los años han sido varias las mejoras realizadas en este campo. Sin embargo, aún hay espacio para seguir introduciendo nuevos métodos que permitan mejorar la integridad del sistema.

1.2.1 Esquema Red de medidores

En el primer trabajo de graduación sobre la red de monitoreo remoto de medidores, se estableció la arquitectura de la red [1]. En la Figura 1, se muestra el diagrama de la red malla o mesh implementada.

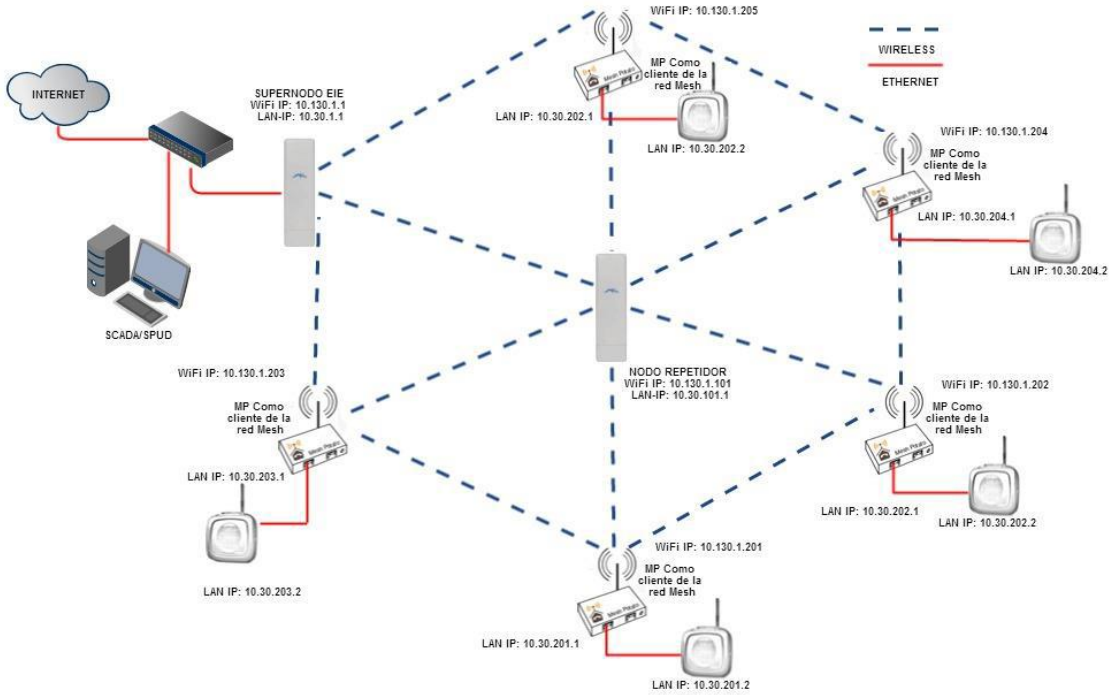

Figura 1. Red de medidores campus central Universidad de El Salvador [1].

La red de medidores de la energía eléctrica está compuesta por nodos de medición, nodo repetidor en el edificio de la escuela de Arquitectura y un supernodo instalado en el techo de la Escuela de Ingeniería Eléctrica en la FIA.

Cada nodo de medición consiste en un medidor y un router, conectados mediante cable de red ethernet. El router utilizado es el MP01 [3]. Originalmente este router fue pensado para dotar de telefonía y acceso a internet a zonas rurales. Sin embargo, mediante la adecuada configuración se utiliza como nodo de conexión de los medidores. Es un router inalámbrico diseñado por Village Telco, el propósito principal es servir como punto VoIP y de internet. Tiene instalado de fábrica el firmware VT de Village Telco basado en OpenWrt, el paquete para VoIP asterisk y el paquete del protocolo de enrutamiento B.A.T.M.A.N. En el año 2013 se actualizó el firmware a todos los router MP01 de la red, se cambió a la versión SECN 1.1 la cual incluye el protocolo de enrutamiento B.A.T.M.A.N. Adv.

Los medidores son tipo SHARK, la red cuenta con 18 medidores Shark 100S, 6 medidores Shark 200S y 2 medidor Shark 200, distribuidos de acuerdo a la Tabla 1. Desde el año 2015 cuatro medidores se encuentran fuera de servicio [4].

MEDIDOR	IP	IP MP01	TIPO (SHARK)
Agronomia	10.30.217.2	10.130.3.217	100S
AgronomiaDecanato	10.30.218.2	10.130.3.218	100S
AgronomiaGalera	10.30.219.2	10.130.3.219	100S
AgronomiaQuimica*	10.30.220.2	10.130.3.220	200S
AuditoriumMarmol	10.30.215.2	10.130.3.215	100S
Cafetines	10.30.212.2	10.130.3.212	100S
ComedorUES	10.30.211.2	10.130.3.211	100S
Derecho	10.30.201.2	10.130.3.201	200S
Economia1	10.30.202.2	10.130.3.202	100S
Economia2	10.30.203.2	10.130.3.203	100S
Economia3	10.30.204.2	10.130.3.204	100S
Economia4	10.30.205.2	10.130.3.205	100S
Economia6	10.30.207.2	10.130.3.207	100S
Humanidades2	10.30.209.2	10.130.3.209	100S
Humanidades3	10.30.210.2	10.130.3.210	100S
Humanidades4	10.30.231.2	10.130.3.231	200
MecanicaComplejo	10.30.216.2	10.130.3.216	100S
Medicina	10.30.227.2	10.130.3.227	200S
Odontologia1	10.30.224.2	10.130.3.224	200S
Odontologia2	10.30.225.2	10.130.3.225	200S
Odontologia3	10.30.226.2	10.130.3.226	200S
OdontologiaImprenta	10.30.223.2	10.130.3.223	100S
Periodismo	10.30.213.2	10.130.3.213	100S

PrimarioFIA	10.30.214.2	10.130.3.214	200
Psicologia	10.30.228.2	10.130.3.228	100S
Quimica	10.30.221.2	10.130.3.221	200S
QuimicaImprenta	10.30.222.2	10.130.3.222	100S

Tabla 1. Distribución de nodos de medición

1.2.2 Red mesh y protocolo B.A.T.M.A.N.

Se denomina red mesh a aquella donde existen al menos 2 caminos a cada nodo. Los router mesh se encargan de realizar funciones de reenvío de tráfico de otros nodos y funciones de gateway para interconectar con otras redes existentes.

Un aspecto importante del funcionamiento de las redes mesh es que un nodo puede transmitir a otro nodo que no esté en su mismo rango de cobertura, ya que se realiza un enrutamiento multi-salto y la información se transmite a través nodos intermedios hasta que alcanza el destino [5].

En el trabajo de graduación, se utilizó la versión del firmware VT de Village Telco en cada uno de los router [1]. Este firmware trae configurado por defecto el protocolo B.A.T.M.A.N. para redes mesh. Todos los router forman parte la red mesh como se observa en la Figura 2. Posteriormente en el año 2015 se realizaron mejoras a la red de medidores, migrando a la versión del firmware SECN 1.1, el cual por defecto tiene configurado el protocolo B.A.T.M.A.N. Advance [6].

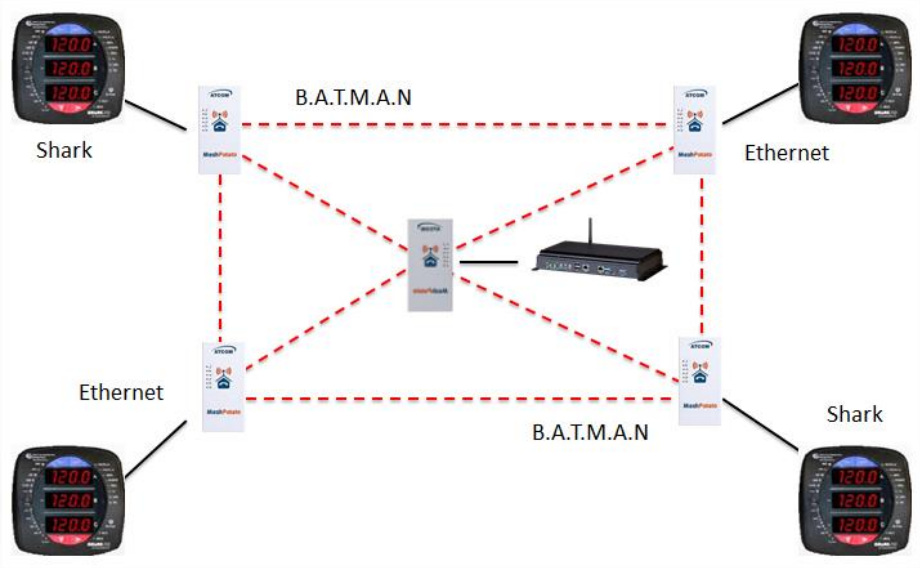

Figura 2. Red mesh mediante protocolo B.A.T.M.A.N.

B.A.T.M.A.N. es un protocolo de enrutamiento dinámico y proactivo para redes mesh ad-hoc que utiliza las tablas de enrutamiento para las decisiones de tráfico de paquetes. Este

protocolo no calcula rutas completas entre un nodo origen y destino sino que selecciona un nodo de salto para utilizarlo como gateway hacia el destino [7].

La función de B.A.T.M.A.N. es encontrar otros nodos B.A.T.M.A.N. y definir el mejor vecino para llegar a ellos. Además hace un seguimiento de los nuevos nodos e informa a sus vecinos de su existencia.

Es decir, cuando un nodo se incorpora a la red envía un paquete broadcast para avisar de su existencia. Este mensaje se va distribuyendo por toda la red. Cuando un nodo recibe este mensaje registra la dirección por donde le ha llegado la información más rápido, y así si tiene que enviarle información utilizará ese nodo para transmitir. El protocolo mantiene la información sobre la existencia de los nodos mientras sean accesibles. Por lo tanto el protocolo B.A.T.M.A.N. busca un nodo vecino hacia el destino para ser utilizado como gateway y conseguir la comunicación, es decir, se encarga de elegir el mejor salto para cada destino. Esto provoca que la implementación sea rápida y eficiente, ya que no es necesario calcular la ruta completa.

Cada nodo percibe y mantiene la información sobre el mejor salto hacia el resto de nodos. Por este motivo es innecesario el conocimiento global de la red.

Resumiendo, cada nodo transmite mensajes broadcast para indicar a los nodos vecinos de su existencia. Y los vecinos lo retransmiten a sus vecinos, y así sucesivamente. De este modo el mensaje llega a todos los nodos de la red. Estos mensajes son pequeños, el tamaño típico es de 52 bytes. Los campos que contienen estos mensajes son la dirección de origen, la dirección del nodo que retransmite, el TTL y el número de secuencia.

Es posible que se reciban varias veces el mismo mensaje, ya que hay rutas donde se propagan con retardo o con pérdidas debido a la baja calidad del enlace, y habrá otras donde se propagan de manera rápida y fiable. Para solucionar este problema se etiqueta cada mensaje con un número de secuencia. De este modo se hace un estudio de que nodo vecino es el que le ha transmitido la información primero y lo selecciona como el vecino de salto para enviar datos al origen del mensaje y lo configura en su tabla de enrutamiento.

B.A.T.M.A.N. Advance es un protocolo de enrutamiento de capa 2, es decir, las tablas de enrutamiento almacenan las direcciones físicas (MAC), de cada uno de los nodos dentro de la red mesh. A diferencia de B.A.T.M.A.N. en el que las tablas de enrutamiento almacenan las direcciones lógicas (IP) de cada nodo.

1.2.3 Arquitectura de interrogación

En la Figura 3, se muestra la arquitectura actual de interrogación.

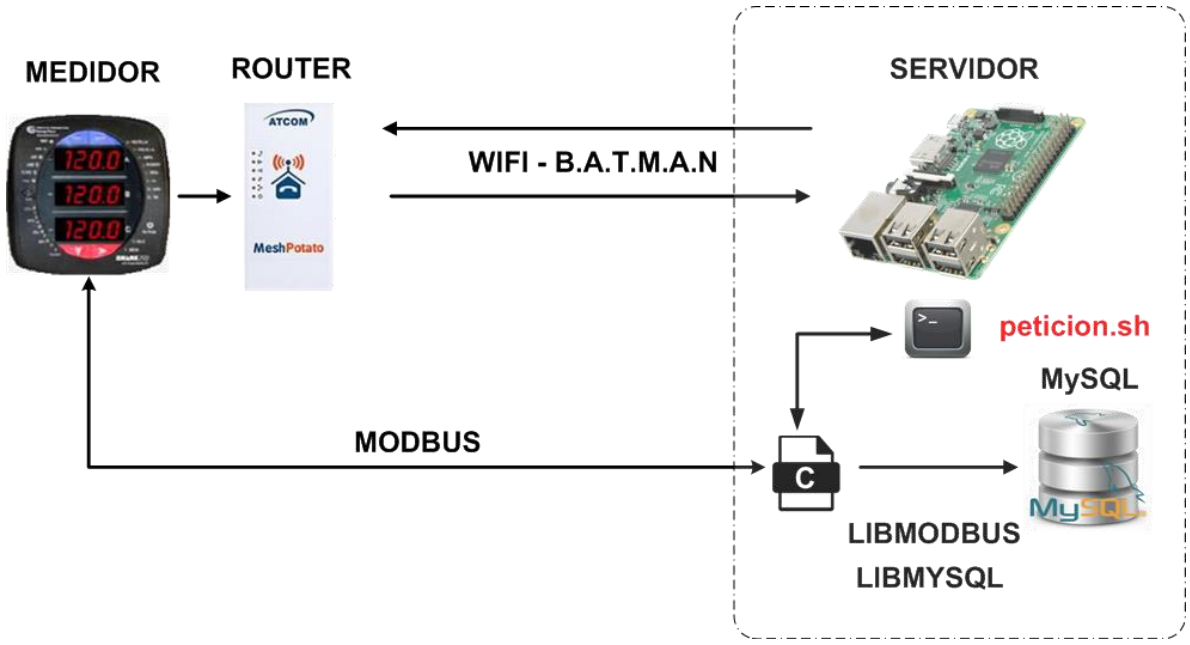

Figura 3. Arquitectura actual de interrogación de medidores

La estructura actual de interrogación se realiza a través de un único dispositivo central. Éste interroga de forma continua toda la red de medidores, mediante consultas realizadas utilizando el protocolo Modbus TCP/IP. Las consultas son realizadas en intervalos de 15 minutos y se almacenan en una base de datos MySQL [2].

```
pi@raspberrypi ~ $ sudo crontab -l | tail -2
*/15 * * * * cd /home/pi/tesisduque/medidores && ./peticion.sh >> /home/pi/tesisduque/logs/medidores.log 2>&1
*/50 * * * * cd /home/pi/tesisduque && ./updater.sh >> /home/pi/tesisduque/logs/actualizacion.log 2>&1
pi@raspberrypi ~ $
```

Figura 4. Tareas crontab programadas en el servidor

El script *peticion.sh*, programado en una tarea crontab dentro de la raspberry pi como se muestra en la Figura 4, es el encargado de realizar la ejecución de un programa escrito en C, que se encarga de consultar a los medidores y almacenar los datos en una tabla de MySQL. Luego estos datos son subidos a la Web mediante el script *updater.sh*, que se encarga de consultar la tabla MySQL y enviar los datos al servidor de la plataforma Google App Engine, scripts desarrollados en [2].

La Figura 5 muestra que al principio, la interrogación se realizaba de forma secuencial, un medidor a la vez. Sin preocuparse de si la interrogación era o no exitosa. Sin embargo, los enlaces de radio mediante WiFi mostraron poca robustez. Conduciendo a la pérdida de datos. En el trabajo de graduación desarrollado en [2], se identificaron dos tipos de

errores, todos atribuibles a la variación en la calidad de los enlaces de radio. El primer error identificado no permitía el establecimiento de una conexión TCP. El segundo error si permitía el establecimiento de una conexión TCP, pero nunca se llegaba a finalizar con satisfacción la transferencia de los datos debido a que no existían ninguna persistencia por parte del protocolo Modbus.

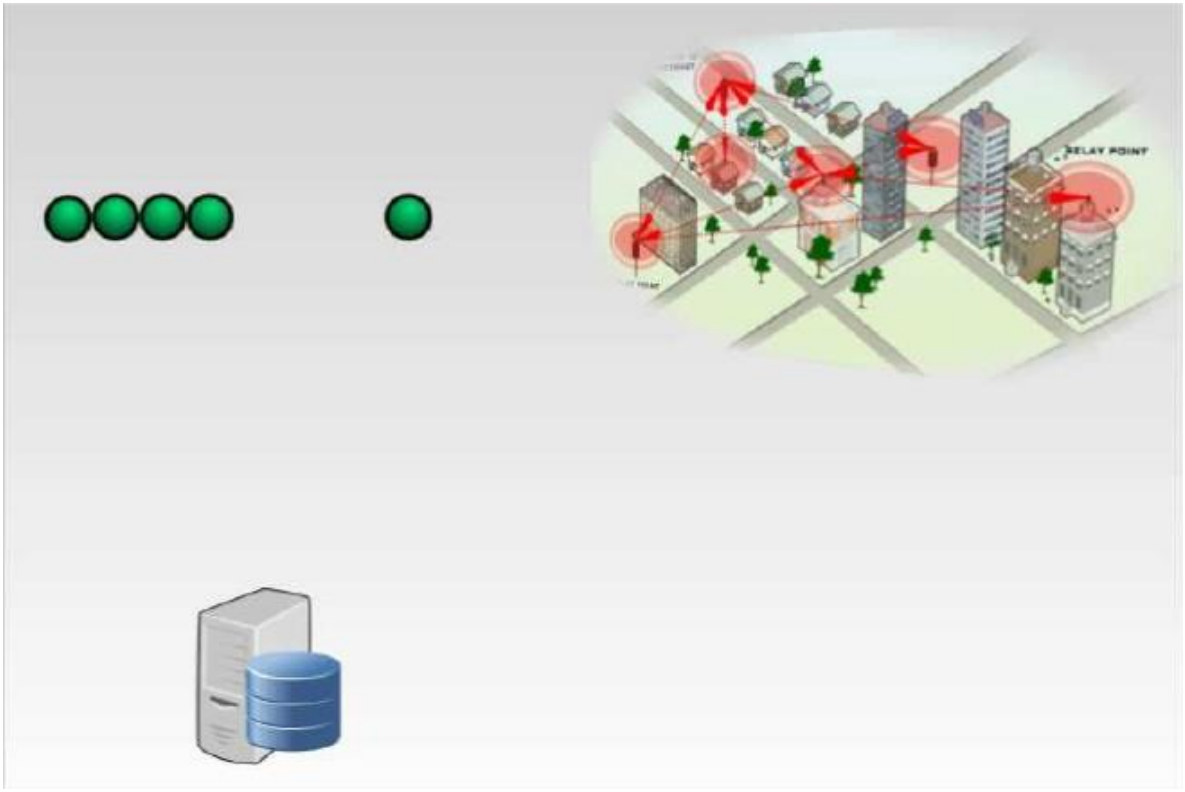

Figura 5. Lógica de interrogación implementada en [1]

Buscando soluciones a la perdida de paquetes, se cambió la lógica de interrogación, realizándola tipo Broadcast como se muestra en la Figura 6. Para ello, se creó un script en el cual si por alguna razón hay un fallo en el establecimiento de una conexión TCP, el script persiste tantas veces como se haya programado, buscando el establecer la conexión cliente-servidor.

En la Figura 7 se muestra el flujograma de la interrogación donde se interroga a los medidores, todos al mismo tiempo. Esta idea se denomina interrogación tipo broadcast. Y permite tener control sobre el tiempo de persistencia en la interrogación de cada medidor.

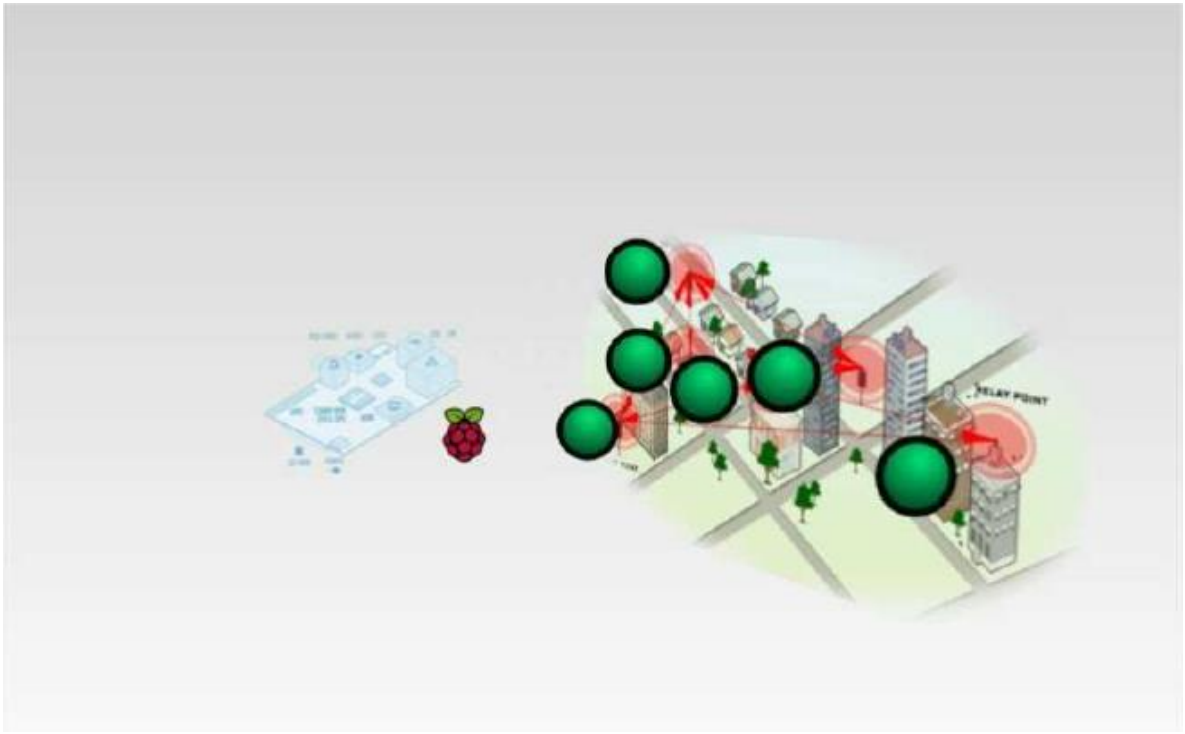

Figura 6. Lógica de interrogación implementada en [2]

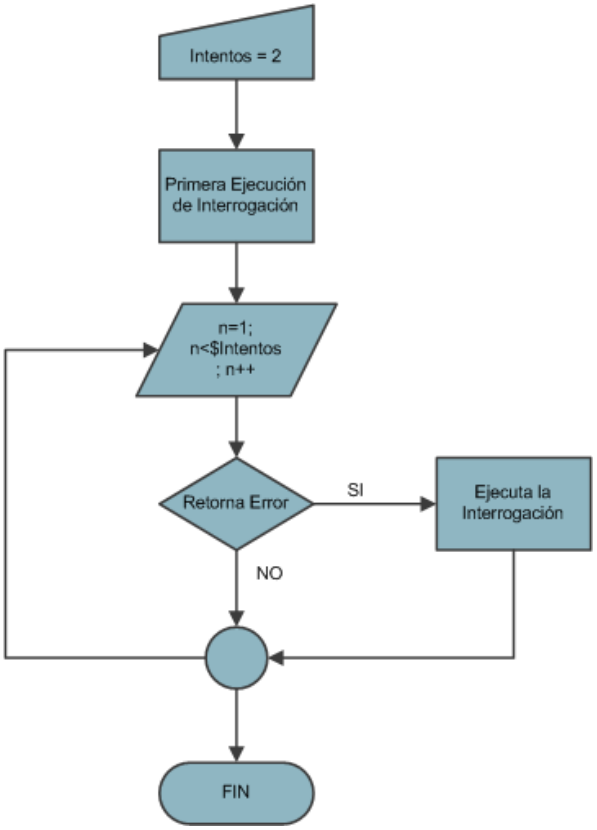

Figura 7. Flujograma del script desarrollado en [2].

1.2.4 Actualización de firmware y reconfiguración de la red MESH

En el año 2013 se realizaron mejoras a la red de medidores, se actualizó a la versión del firmware SECN 1.1 en cada Router MP01, el cual por defecto tiene configurado el protocolo B.A.T.M.A.N. Adv.

El trabajo consistió en configurar e instalar nuevos nodos a la red de medidores, el montaje de los router se realizó en ventanas y techos de los diferentes locales y edificios donde se alcanzaba una altura considerable para que se estableciera un enlace de comunicación con los demás router. También se configuró un sistema de monitoreo basado en un servidor web instalado en una computadora del laboratorio de comunicaciones de la Escuela de Ingeniería Eléctrica para mantener funcional la red inalámbrica y realizar las optimizaciones necesarias según su comportamiento.

Se implementó el software de monitoreo SPUD, un panel de control basado en PHP que se comunica con el servidor de visualización de B.A.T.M.A.N. Adv y muestra el estado de la red inalámbrica en tiempo real. El software está escrito en CakePHP (Aplicación basada en PHP framework MVC) y utiliza API de Google Maps 1.3 para la visualización. Este sistema de monitoreo ya no se pudo restablecer a partir de enero de 2016.

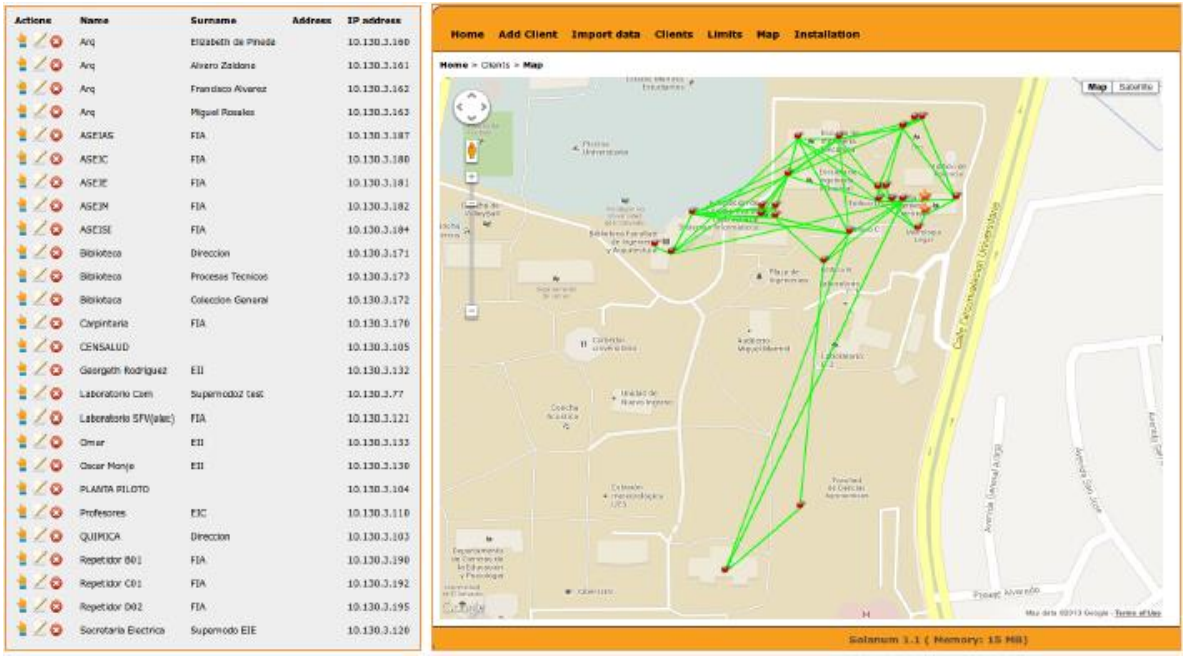

Figura 8. Software de monitoreo de la red de medidores de energía eléctrica.

Es importante mencionar que con la actualización del firmware SECN 1.1, se realizó reconfiguración de las IPs asignadas a cada router MP01, utilizando IPs dentro de la red 10.130.3.0/24.

1.2.5 Problemas con la red actual

Hasta la fecha el sistema de monitorización aún adolece de robustez, frecuentemente se presenta el problema de pérdida de datos. El problema se debe principalmente a la calidad de los enlaces por las distancias en las que se encuentran los medidores del servidor central.

A continuación se muestran las gráficas de los medidores más problemáticos con los que se presenta perdida de datos. Medidor Humanidades4 tipo Shark 200S y medidor Economia6 tipo Shark 100S.

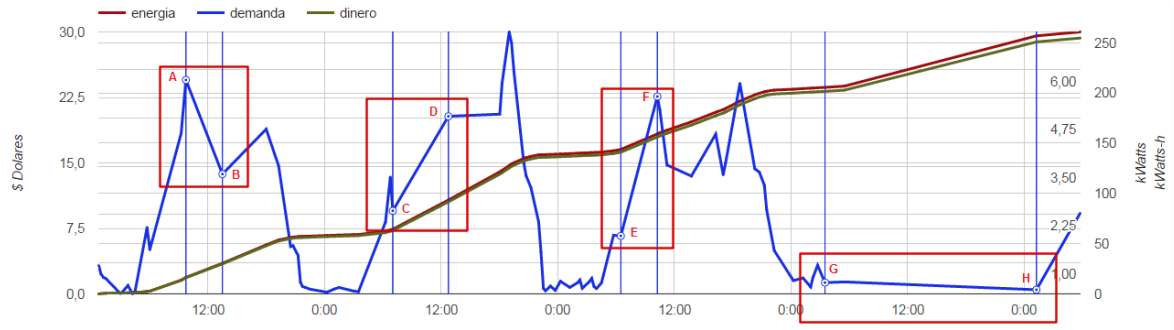

Figura 9. Grafica de lecturas del medidor Humanidades4 [23].

El primer bloque en la Figura 9 corresponde a una lectura A con fecha 28/11/2016 09:45 y una lectura B con fecha 28/11/2016 13:30, reflejando una pérdida de datos durante 3 horas aproximadamente. De igual manera el segundo bloque, una lectura C con fecha 29/11/2016 07:00 y una lectura D con fecha 29/11/2016 12:45, dando como resultado perdida de datos en un lapso de 5 horas aproximadamente. El problema se repite en el tercer bloque. Existen ocasiones en las que el intervalo de tiempo sin conexión al medidor es más prolongado, como se observa en el cuarto bloque, donde la lectura G tiene fecha 01/12/2016 03:30 y la siguiente lectura H se realizó en la fecha 02/12/2016 01:15, presentándose una pérdida de datos de aproximadamente un día. Comportamiento similar mostrado en la Figura 10.

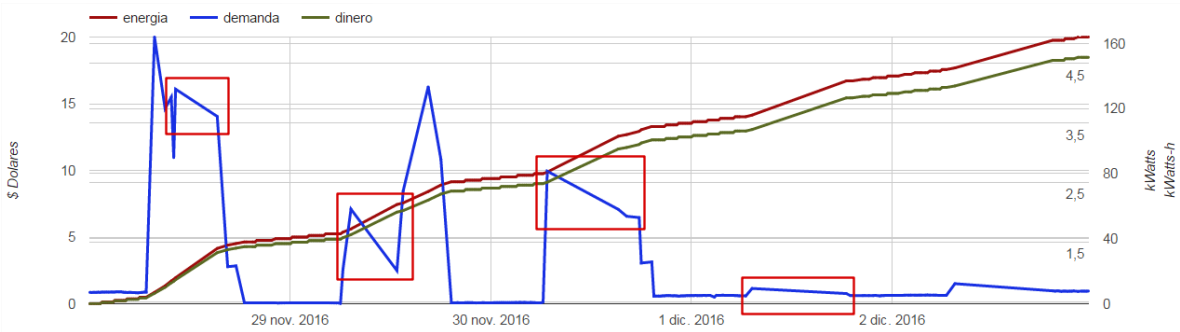

Figura 10. Grafica de lecturas del medidor Economia6 [23].

1.3 Motivación para realizar el proyecto

A pesar de todas la contribuciones realizadas a la red de monitorización de medidores de energía eléctrica. El sistema aún adolece de robustez. Es necesario encontrar una técnica que permita mantener la integridad de los datos.

Este trabajo de graduación cambia por completo la arquitectura actual de interrogación. Cada router realizará una consulta directa a su respectivo medidor. Los datos leídos serán almacenados en una base de datos SQLite dentro del mismo router. El router estará continuamente esperando el momento adecuado para enviar los datos al servidor central. Los datos serán transferidos mediante la ejecución de un procedimiento remoto basado en el protocolo XML-RPC. Una vez ejecutado el procedimiento los datos quedarán guardados en una base de datos local.

1.4 Objetivos

1.4.1 Objetivo general

Desarrollar una aplicación que permita mejorar la integridad de los datos, realizando la interrogación y almacenamiento *en situ* a cada medidor, para luego enviar los datos al servidor mediante la implementación del protocolo XML-RPC.

1.4.2 Objetivos específicos

- ✓ Implementar el protocolo Modbus TCP/IP en un router MP01, utilizando la librería libmodbus del lenguaje de programación C, mediante la técnica de compilación cruzada.
- ✓ Configurar la comunicación Modbus entre los router MP01 y los medidores de energía tipo SHARK 100 y SHARK 200.
- ✓ Instalar y configurar el gestor de bases de datos SQLite en un router MP01.
- ✓ Almacenar la información en el router MP01 mediante el uso de base de datos.
- ✓ Investigar el funcionamiento del protocolo XML-RPC en sistema empotrados.
- ✓ Implementar y configurar un cliente XML-RPC en un router MP01.
- ✓ Realizar rutinas mediante script de Linux, para la consulta y envió de los datos.
- ✓ Implementar y configurar un servidor XML-RPC en una raspberry pi.
- ✓ Mejorar la integridad de los datos en el sistema actual de monitorización de medidores de energía eléctrica.
- ✓ Implementar un sistema de respaldo basado en micro-computadoras Raspberry Pi.

1.5 Organización

El trabajo se divide en tres partes principales. La primera corresponde a la interrogación y almacenamiento de las mediciones in situ, en la cual se requiere de la técnica de compilación cruzada para generar el programa de consulta y ejecutarlo dentro del router. Se implementa un método diferente de interrogar a la red de medidores, mediante una aplicación en C que se ejecuta dentro de cada router y realiza consultas periódicas a cada medidor, almacenando los resultados en una base de datos SQLite. La segunda parte tiene que ver con la implementación del protocolo XML-RPC para el envió de los datos, desde el router hacia el servidor, mediante una aplicación cliente XML-RPC programada en LUA que será capaz de consultar la base de datos SQLite y cada vez que se tengan nuevos datos, intentará enviarlos a una aplicación servidor XML-RPC programada en PERL. La última parte consiste en el almacenamiento y visualización de los datos utilizando la plataforma Google App Engine en una Raspberry Pi.

Capítulo 2: Interrogación de los medidores in situ

2.1 Nueva arquitectura de interrogación

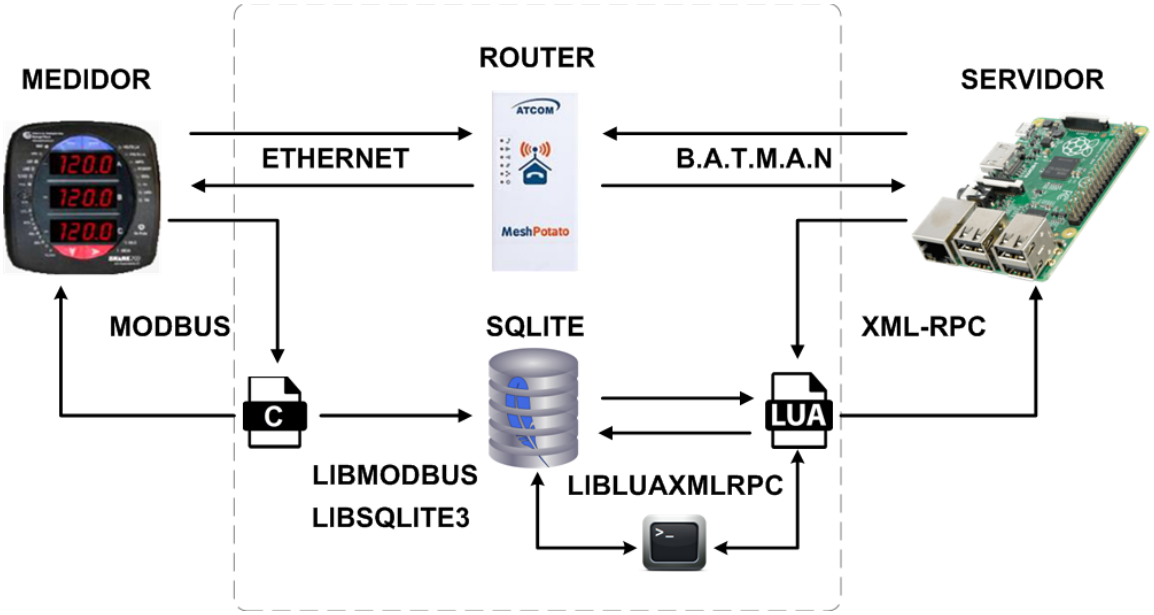

Figura 11. Nueva arquitectura de interrogación

La nueva arquitectura de interrogación mostrada en la Figura 11 consiste en crear una aplicación en C para comunicar medidor y router implementando el protocolo Modbus TCP/IP. La aplicación deberá correr en el router, por lo tanto se implementará la técnica de compilación cruzada para compilar la librería libmodbus y que el programa pueda ser ejecuta en el MP01. Por una parte, el sistema huésped es una computadora personal. Por otra parte, el sistema objetivo será el router MP01. El método utilizado para realizar la compilación cruzada será mediante la cadena de herramientas o toolchain que proporcionan los desarrolladores de OpenWRT [8]. Una vez compilada la librería Libmodbus se desarrollará un programa que funcione como cliente modbus, capaz de solicitar datos al medidor. Los datos leídos por el router se almacenarán en una base de datos SQLite.

De manera permanente el router estará intentando enviar los datos al servidor, para lograrlo se desarrollara otra aplicación en LUA basada en el protocolo XML-RPC. La aplicación será capaz de consultar la base de datos SQLite y cada vez que se tengan nuevos datos, intentará enviarlos, mediante la ejecución de un procedimiento remoto, al servidor.

También se desarrollara una aplicación CGI en el servidor basada en el protocolo XML-RPC. La aplicación será capaz de recibir los datos enviados por el router y almacenarlos en una base de datos local MySQL.

2.2 Protocolo Modbus en medidores tipo SHARK

2.2.1 Protocolo Modbus

El protocolo Modbus es una estructura de mensajería desarrollado por Schneider Electric en 1979 [9]. Sirve para establecer la comunicación entre dispositivos inteligentes maestro-esclavo/cliente-servidor. Es un estándar abierto y es el protocolo de red más utilizado en el ambiente de la fabricación industrial. El protocolo Modbus proporciona un método industrial estándar que los dispositivos Modbus utilizan para analizar los mensajes.

La transferencia de datos se organizó en términos de registros de 16 bits (formato de entero) o como información de estado en términos de bytes de datos. Con los años el protocolo fue extendido y ha sido adoptado por otros fabricantes también. Se han añadido nuevos tipos de datos, en especial para lograr una resolución más alta para los valores que se transmiten. Se aprobó el protocolo para la transferencia de los nuevos medios, tales como Modbus Plus o Modbus TCP/IP [9].

2.2.2 Orientado a la conexión

Modbus serial es un protocolo de comunicación sin estado, es decir, cada solicitud del maestro es tratada independientemente por el esclavo y es considerada una nueva solicitud no relacionada a las anteriores, de esta forma haciendo a las transferencias de datos altamente resistentes a rupturas debido a ruido y además requiriendo mínima información de recuperación para ser mantenida la transferencia en cualquiera de los dos terminales [10].

Las operaciones de programación esperan una comunicación orientada a la conexión, es decir, las máquinas de origen y de destino establecen un canal de comunicaciones antes de transferir datos. Este tipo de operaciones son implementadas de diferentes maneras por las diversas variantes de Modbus serial (Modbus RTU, Modbus ASCII, Modbus PLUS).

Modbus TCP/IP maneja una conexión inicialmente establecida en esta capa de protocolo (nivel de aplicación), y esa conexión única puede llevar múltiples transferencias independientes.

También, TCP permite establecer un gran número de conexiones concurrentes. De este modo el cliente (maestro) puede ya sea usar una conexión previamente establecida ó crear una nueva, en el momento de realizar una transferencia de datos.

2.2.3 Modbus TCP/IP

TCP/IP se refiere al protocolo de control de transmisión y el protocolo de Internet, que proporciona el medio de transmisión para los mensajes de Modbus TCP/IP. Es simplemente el protocolo Modbus RTU con una interfaz TCP que se ejecuta sobre Ethernet.

Modbus TCP/IP utiliza TCP/IP y Ethernet para transportar los datos de la estructura de los mensajes de Modbus entre dispositivos compatibles. Es decir, combina una red física (Ethernet) con una red de trabajo estándar (TCP/IP), y un método estándar de representación de datos.

La estructura de los mensajes de Modbus definen las reglas para la organización y la interpretación de los datos independientemente de los datos del medio de transmisión.

A continuación se presenta un análisis de establecimiento y finalización de una conexión Modbus TCP/IP, donde se implementan las configuraciones que se muestran en la Figura 12. El objetivo principal consiste en establecer una comunicación entre el host y el medidor mediante una aplicación en C, la cual hace una petición al medidor.

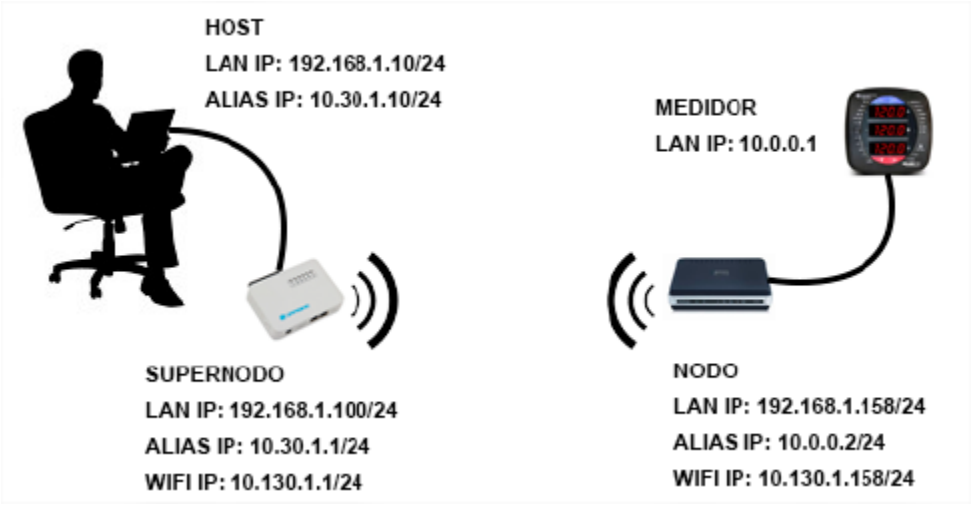

Figura 12. Diagrama de red para el análisis de una trama Modbus TCP/IP.

Cuando dos dispositivos se comunican utilizando TCP, se establece una conexión antes de que puedan intercambiarse los datos. Luego de que se completa el intercambio de datos, se finaliza la sesión entre los dispositivos.

Cada conexión representa dos medios de comunicación de una vía o sesiones. Para establecer la conexión los dispositivos realizan un intercambio de señales de tres pasos. Los bits de control en el encabezado TCP indican el progreso y estado de la conexión.

En conexiones TCP, el dispositivo que brinde el servicio como cliente inicia la sesión al servidor. Los tres pasos para el establecimiento de una conexión TCP son:

1) El cliente que inicia la conexión envía un segmento que contiene un valor de secuencia inicial, que actúa como solicitud para el servidor para comenzar una sesión de comunicación.

2) El servidor responde con un segmento que contiene un valor de reconocimiento igual al valor de secuencia recibido más 1, además de su propio valor de secuencia de sincronización. El valor es uno mayor que el número de secuencia porque el ACK es siempre el próximo Byte u Octeto esperado. Este valor de reconocimiento permite al cliente unir la respuesta al segmento original que fue enviado al servidor.

3) El cliente que inicia la conexión responde con un valor de reconocimiento igual al valor de secuencia que recibió más uno. Esto completa el proceso de establecimiento de la conexión.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.30.1.10	10.0.0.1	TCP	74	50019 > asa-appl-proto [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=632951 TSecr=0 WS=16
2	0.752211	10.0.0.1	10.30.1.10	TCP	60	asa-appl-proto > 50019 [SYN, ACK] Seq=0 Ack=1 Win=511 Len=0 MSS=1024
3	0.752240	10.30.1.10	10.0.0.1	TCP	54	50019 > asa-appl-proto [ACK] Seq=1 Ack=1 Win=14600 Len=0
4	0.752300	10.30.1.10	10.0.0.1	Modbus/	66	query [1 pkt(s)]: trans: 1; unit: 1, func: 3: Read multiple registers.
5	0.765125	10.0.0.1	10.30.1.10	TCP	60	asa-appl-proto > 50019 [ACK] Seq=1 Ack=13 Win=511 Len=0
6	0.786131	10.0.0.1	10.30.1.10	Modbus/	67	response [1 pkt(s)]: trans: 1; unit: 1, func: 3: Read multiple registers.
7	0.786145	10.30.1.10	10.0.0.1	TCP	54	50019 > asa-appl-proto [ACK] Seq=13 Ack=14 Win=14600 Len=0
8	0.786355	10.30.1.10	10.0.0.1	TCP	54	50019 > asa-appl-proto [FIN, ACK] Seq=13 Ack=14 Win=14600 Len=0
9	0.789081	10.0.0.1	10.30.1.10	TCP	60	asa-appl-proto > 50019 [FIN, ACK] Seq=14 Ack=14 Win=511 Len=0
10	0.789100	10.30.1.10	10.0.0.1	TCP	54	50019 > asa-appl-proto [ACK] Seq=14 Ack=15 Win=14600 Len=0

Figura 13. Trama capturada con el programa wireshark.

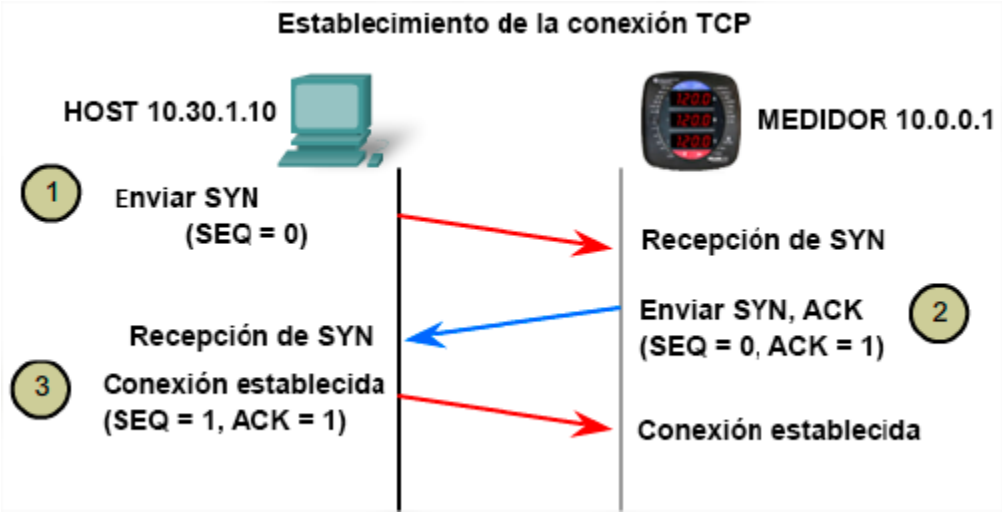

Figura 14. Esquema de los tres pasos en el inicio de una conexión TCP.

En el paso 1 de la Figura 14, el dispositivo que actúa como cliente TCP comienza el enlace enviando un segmento con el señalizador de control SYN, indicando un valor inicial en el campo de número de secuencia del encabezado. Este valor para el número de secuenciase elige de manera aleatoria y se utiliza para comenzar a rastrear el flujo de datos desde el cliente al servidor para esta sesión. Como se observa en la Figura 13 y en la Figura 14, se establece el señalizador de control SYN y el número de secuencia relativa en 0. Hay que mencionar que wireshark asigna los valores para los números de secuencia y de acuse de recibo, los valores reales son números binarios de 32 bits.

Para el paso 2, el dispositivo servidor TCP necesita reconocer la recepción del segmento SYN del cliente para establecer la sesión de cliente a servidor. Para hacerlo, el servidor envía un segmento al cliente con el señalizador ACK establecido indicando que el número de acuse de recibo es significativo. Con este señalizador establecido en el segmento, el cliente interpreta esto como acuse de recibo de que el servidor ha recibido el SYN del cliente TCP. El valor del número de campo del acuse de recibo es igual al número de secuencia inicial del cliente más 1. Esto establece una sesión desde el cliente al servidor. El señalizador ACK permanecerá establecido para mantener el equilibrio de la sesión.

En el último paso, el cliente TCP responde con un segmento que contiene un ACK que actúa como respuesta al SYN de TCP enviado por el servidor. No existen datos de usuario en este segmento. El valor del campo número de acuse de recibo contiene uno más que el número de secuencia inicial recibido del servidor. Una vez establecidas ambas sesiones entre el cliente y el servidor, todos los segmentos adicionales que se intercambien en la comunicación tendrán establecido el señalizador ACK.

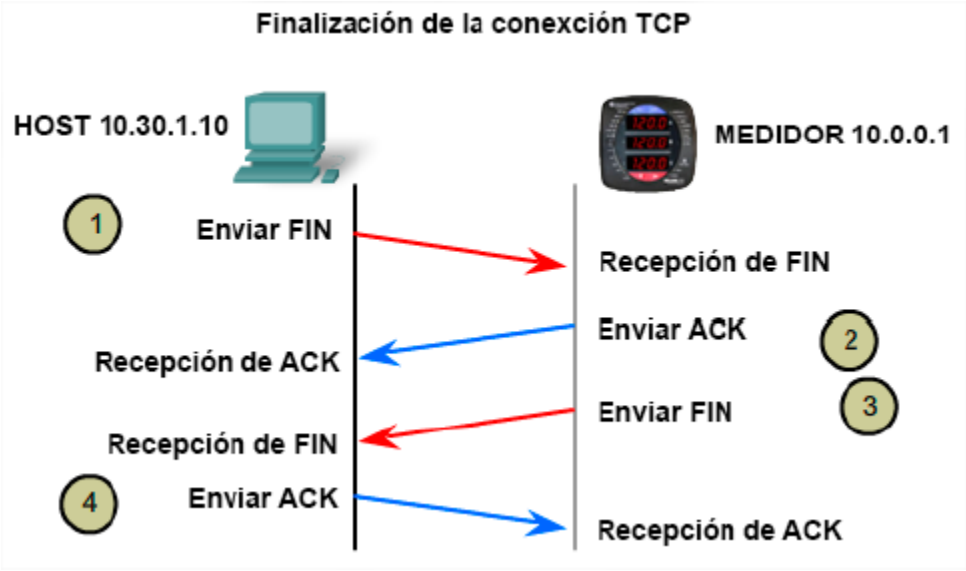

Figura 15. Esquema de finalización de una conexión TCP.

Para cerrar la conexión se debe establecer el señalizador de control FIN (Finalizar) en el encabezado del segmento. Para finalizar todas las sesiones TCP de una vía, se utiliza un enlace de dos vías, que consta de un segmento FIN y un segmento ACK.

Cuando la finalización de sesión del cliente no tiene más datos para transferir, establece el señalizador FIN en el encabezado de un segmento. Luego, el servidor finaliza la conexión y envía un segmento normal que contiene datos con el señalizador ACK establecido utilizando el número de acuse de recibo, confirmando así que se han recibido todos los bytes de datos. Cuando se produce el acuse de recibo de todos los segmentos, se cierra la sesión.

La sesión en la otra dirección se cierra mediante el mismo proceso. El receptor indica que no existen más datos para enviar estableciendo el señalizador FIN en el encabezado del segmento enviado al origen. Un acuse de recibo de retorno confirma que todos los bytes de datos han sido recibidos y, por lo tanto, se ha cerrado la sesión.

También es posible terminar la conexión mediante un enlace de tres pasos. Cuando el cliente no posee más datos para enviar, envía un señalizador FIN al servidor. Si el servidor tampoco tiene más datos para enviar, puede responder con los señalizadores FIN y ACK, combinando dos pasos en uno. El cliente responde con un ACK.

2.2.4 Modelo Cliente/Servidor

Los mensajes de Modbus proporcionan una comunicación cliente/servidor entre dispositivos conectados en una red Ethernet TCP /IP. Este modelo de cliente servidor está basado en cuatro tipos de mensajes: Petición Modbus, Confirmación Modbus, Indicación Modbus, Respuesta Modbus.

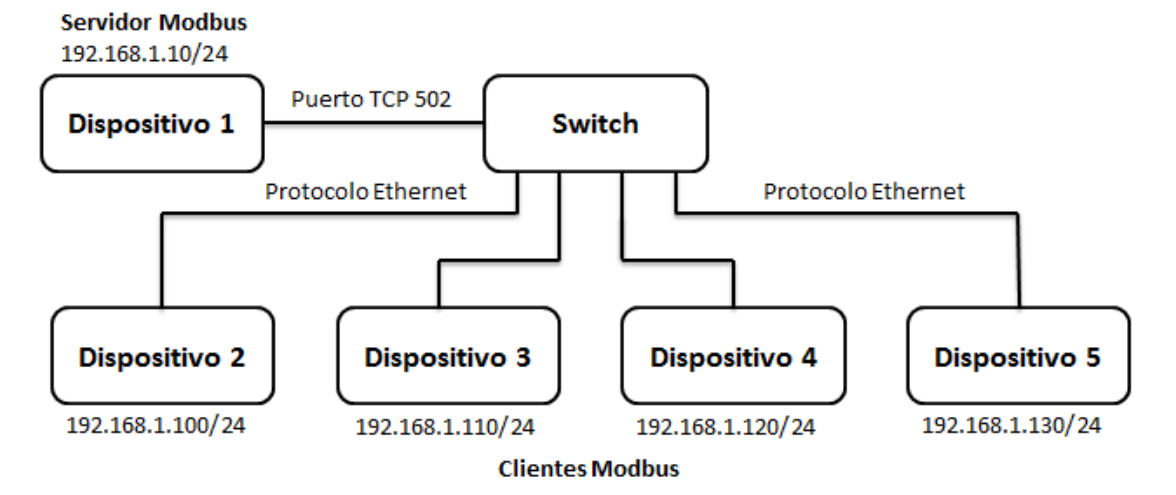

Figura 16. Arquitectura Cliente/Servidor

El servicio de mensajes Modbus es usado para el intercambio de información en tiempo real:

- ✓ Entre aplicaciones de dos dispositivos
- ✓ Entre la aplicación de un dispositivo y otro dispositivo.
- ✓ Entre aplicaciones de SCADA/Sistemas de control y dispositivos.
- ✓ Entre una PC y un programa de dispositivo proporcionando servicios en línea.

2.2.5 Estructura de un paquete de datos Modbus TCP/IP

Modbus TCP/IP incluye una trama de datos estándar dentro de una trama TCP, sin la comprobación que incluye Modbus RTU, como se muestra en la Figura 17.

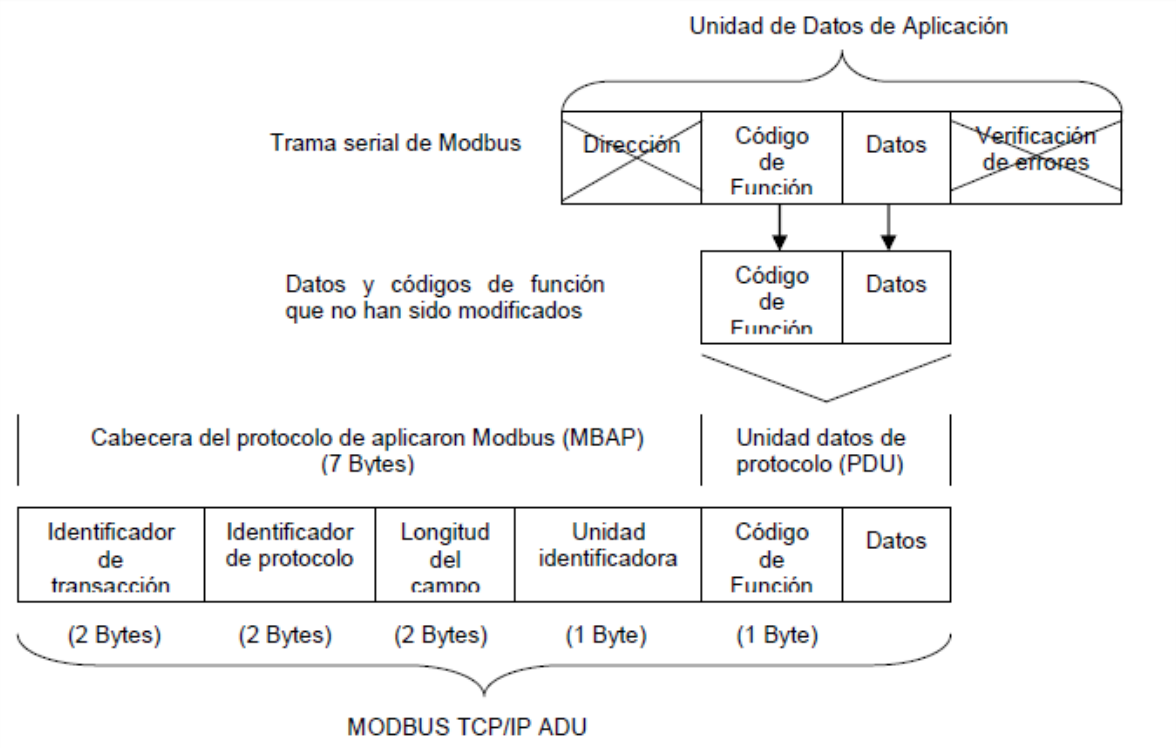

Figura 17. Estructura de un paquete de datos Modbus TCP/IP

Los comandos de Modbus y los datos de usuario son encapsulados en un paquete de datos de una red TCP/IP sin ser modificados de ninguna manera. Sin embargo, el campo de comprobación de errores de Modbus RTU no se utiliza como la capa de enlace estándar de Ethernet TCP/IP por los métodos utilizados, para proporcionar garantía de la integridad de los datos.

En la Figura 17 se observa que el código de la función y los datos de los campos son absorbidos en su forma original. De esta forma, una unidad de datos de aplicación

Modbus TCP/IP adopta la forma de una cabecera de 7 bytes (Identificador de transacción + identificador de protocolo + longitud del campo + unidad identificadora) y la unidad de protocolo de datos.

La cabecera del protocolo de aplicación Modbus es de 7 bytes de longitud donde se utilizan los siguientes campos:

- ✓ **Identificador de transacción (2 bytes):** Este campo de identificación se utiliza para la operación cuando varios mensajes son enviados a través de la misma conexión TCP por un cliente sin esperar a una previa respuesta.
- ✓ **Identificador de protocolo (2 bytes):** Este campo es siempre 0 para los servicios Modbus y otros valores están reservados para futuras ampliaciones.
- ✓ **Longitud del campo (2 byes):** Este campo es un contador de bytes de los campos restantes e incluye la unidad de identificación de bytes, la función de código byte y los campos de los datos.
- ✓ **Unidad Identificadora (1 byte):** Este campo se utiliza para identificar un servidor remoto situado en una red que no es TCP/IP. En un típico servidor de aplicaciones Mosbus TCP/IP, la unidad de identificación está establecido en 00 o FF, ignorada por el servidor.

2.2.6 Modbus en medidores de energía tipo SHARK

Shark es una marca de un medidor de energía multifunción diseñado para ser utilizado en subestaciones eléctricas, tableros y como medidor de energía para equipos de OEM's, Fabricante de Equipos Originales. La unidad proporciona la medición de múltiples funciones de todos los parámetros eléctricos.

La unidad está diseñada con capacidades avanzadas de medición, permitiendo que alcance Exactitud del alto desempeño. El medidor está especificado como medidor de energía clase 0.2% para uso de facturación así como un medidor altamente exacto para la indicación en tablero [11].

Específicamente la versión Shark 200S tiene hasta 2 MB de memoria para el registro y grabación de datos. Esta ofrece 3 localidades de memoria para tendencias históricas, 1 localidad para Límites y Alarmas, y 1 localidad para Eventos del Sistema [12].

Los medidores de energía Shark 100-S y Shark 200-S cuentan con la interfaz RS 485 para utilizarse con el modo Modbus RTU y también con un puerto Ethernet para utilizar el modo Modbus TCP/IP.

La Tabla 2 describe los registros de memoria utilizados para este trabajo de graduación y su respectivo tamaño en bytes para cada tipo de medidor.

Cantidad de Medidores	Tipo	Lectura de registros	Tamaño	Dirección
18	SHARK 100S	KW-h Total	2 bytes	1105
		KVA-h Total	2 bytes	1115
		Watts Positivos, Trifásico, Promedio	2 bytes	2005
6	SHARK 200S	KW-h Total	2 bytes	1505
		KVA-h Total	2 bytes	1515
		Watts Positivos, Trifásico, Promedio	2 bytes	2005
2	SHARK 200	KW-h Total	2 bytes	1505
		KVA-h Total	2 bytes	1515
		Watts Positivos, Trifásico, Promedio	2 bytes	2005

Tabla 2. Registros leídos de los medidores tipo SHARK

Es de vital importancia conocer las direcciones del mapa de memoria donde se encuentran ubicados los registros que nos interesan. La aplicación de consulta hará uso de esas direcciones, mediante programación implementando Modbus TCP/IP se leerán cada una de las variables utilizando las direcciones correspondientes del mapa de memoria.

En la Tabla 2 se observan las direcciones de las variables del bloque de energía y demanda correspondientes a los medidores SHARK. En el caso del tipo SHARK 200S y SHARK 200, las direcciones son las mismas. Debido a la variación de las direcciones del mapa de memoria se crearan dos aplicaciones de consulta con la misma lógica de programación, diferenciándolas por las direcciones utilizadas para realizar las lecturas.

2.3 Linux en sistemas embebidos

Un sistema GNU/Linux embebido simplemente hace referencia a un sistema embebido basado en el kernel de Linux. Es importante destacar que no existe un kernel específico para sistemas embebidos, es decir, no se necesita crear un kernel especial para sistemas embebidos. A menudo se utilizan las versiones oficiales del kernel de Linux para construir un sistema, por supuesto, es necesario configurar al mismo para dar soporte especial al hardware de un determinado equipo. Fundamentalmente, un kernel utilizado en un sistema embebido difiere de un kernel usado en una computadora de escritorio o servidor en la configuración del mismo al momento de compilarlo.

2.3.1 Arquitectura

La arquitectura de un sistema GNU/Linux involucra diferentes componentes. Desde un punto de vista genérico, un sistema GNU/Linux se puede describir mediante diferentes

capas que van desde el hardware hasta las aplicaciones. En la Figura 18 observamos estos niveles de abstracción [13].

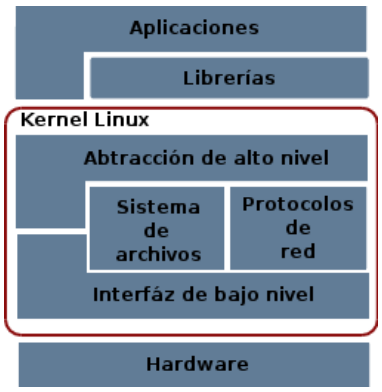

Figura 18. Arquitectura de un sistema GNU/Linux embebido [10].

Inmediatamente sobre el hardware se sitúa el kernel. El kernel es el componente central del sistema operativo. Sus funciones son principalmente administrar el hardware de manera coherente y justa mientras se le otorga un nivel de abstracción familiar, a través de las APIs, a las aplicaciones de nivel de usuario.

Entre otras tareas relevantes de un sistema operativo, el kernel Linux maneja dispositivos, administra los accesos de E/S, controla los procesos y administra el uso compartido de memoria.

Dentro del kernel, la interfaz de bajo nivel es específica para cada configuración de hardware, sobre la cual, el kernel ejecuta y provee control directo de los recursos hardware. Típicamente, los servicios de bajo nivel manejan operaciones específicas de la CPU, operaciones de memoria específicas a la arquitectura, y provee interfaces básicas para dispositivos.

La capa de alto nivel provee abstracciones comunes a todos los sistemas Unix, incluyendo procesos, archivos, sockets y señales. Este nivel de abstracción se mantiene constante aunque difiera el hardware.

Entre estos dos niveles de abstracción, el kernel necesita lo que se denomina componentes de interpretación para comprender e interactuar con datos estructurados provenientes de, o hacia ciertos dispositivos.

Los diferentes tipos de sistemas de archivos y los protocolos de red son ejemplos de fuentes de datos estructurados. El kernel necesita interpretarlos e interactuar a fin de proveer acceso a los datos provenientes desde estas fuentes o hacia las mismas.

Los servicios brindados por el kernel no son soporte suficiente para cargar y ejecutar las aplicaciones. Es necesario contar con librerías, éstas proveen APIs familiares y abstracciones de servicios que interactúan con el kernel en nombre de las aplicaciones para obtener la funcionalidad deseada.

La librería principal, utilizada en la mayoría de las aplicaciones Linux, es la librería C GNU [14]. Típicamente las librerías son enlazadas dinámicamente en el momento en el que se ejecutan las aplicaciones. Es decir, no son parte de la aplicación ejecutable generada luego de la compilación, sino que se cargan dentro del espacio de memoria de las aplicaciones durante la ejecución de las mismas. Esto permite a varias aplicaciones utilizar una misma instancia de una librería en vez de realizar una copia en memoria por cada aplicación que se ejecuta.

2.4 Compilación cruzada para router MP01

La compilación de código fuente que, realizada bajo una determinada arquitectura genera código ejecutable para una arquitectura diferente se denomina compilación cruzada. Para realizar este tipo de compilación es necesario contar con una serie de programas y librerías que establezcan un ambiente propicio para llevar a cabo esta tarea. Este ambiente se denomina entorno de compilación cruzada [15].

Para implementar un entorno de compilación cruzada es necesario un conjunto de librerías y archivos binarios. A este conjunto de componentes se les denomina toolchain.

Estos componentes son:

- ✓ Compilador C: compilador de C básico, generador de código objeto.
- ✓ Librería C: implementa las llamadas al sistema mediante APIs.
- ✓ Binutils: conjunto de programas necesarios para la compilación, enlazado, ensamblado y depuración de código.

En uno de los trabajos de graduación previos, se implementó compilación cruzada utilizando la herramienta Buildroot, la cual brinda una interfaz amigable para seleccionar las versiones de los toolchains y configuraciones extra para implementar el entorno de compilación cruzada [16]. Este software nos permite seleccionar a través de diferentes menús las características que son necesarias de implementar.

El problema con Buildroot es que para realizar el proceso de configuración e instalación, se demora varias horas, dependiendo del procesador de la PC y de la velocidad de internet. Esto es una limitante a la hora modificar librerías y configuraciones del entorno debido a que se tiene que realizar el proceso de nuevo.

En el presente trabajo de graduación se utiliza una alternativa para implementar el entorno de compilación cruzada mediante el toolchain proporcionado por los desarrolladores del firmware OpenWrt. OpenWrt proporciona en su página oficial el toolchain para cada una de las versiones del firmware [17].

2.4.1 Entorno de compilación cruzada

En el entorno de compilación se definen dos tipos de sistemas, el Huesped, en donde se realiza la compilación del código fuente, y el Objetivo donde se ejecuta la aplicación [13].

Para el sistema Huesped se utilizó una minilaptop y el sistema Objetivo es el router MP01 como se describe a continuación.

Sistema Huesped

La implementación de un entorno de compilación nos brinda la posibilidad de aprovechar los recursos que disponemos en una computadora tipo PC: Procesador Intel Atom x86 N455 de 1,66 Ghz, Disco duro de 320 GB, 1024 MB DDR3 SDRAM, Acelerador Intel 3150, LAN inalámbrica,Ubuntu 14.04.3 LTS.

Sistema Objetivo

El dispositivo embebido es un Router MP01. Este equipo se basa en un sistema Atheros SoC (System on chip): Arquitectura MIPS 4K, Bootloader RedBoot, System-On-Chip Atheros AR2317, Velocidad CPU 180 Mhz, Memoria Flash 8 MByte EEPROM, RAM 16 Mbyte, 1 Puerto 10/100Mbit LAN, Linux kernel 2.26.3, OpenWRT.

Comunicación entre sistema Huesped y sistema Objetivo

Las configuraciones del router MP01 se realizaron mediante una consola terminal en la Mini Laptop, utilizando el protocolo ssh. De igual manera la transferencia de archivos, se realizó utilizando el protocolo scp a través de ssh.

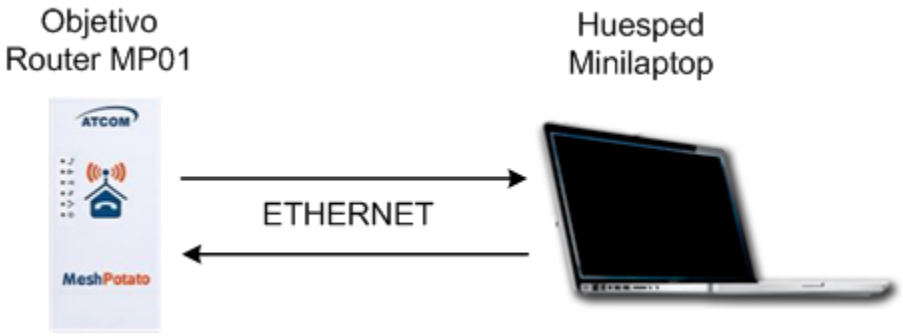

Figura 19. Comunicación Huesped - Objetivo

2.4.2 Compilación cruzada

Se descargan los fuentes proporcionados por OpenWrt para poder generar el toolchain correspondiente a la versión Kamikaze, la cual se encuentra instalada en los router MP01.

```
:~$ mkdir mp01
:~$ cd mp01
:~/mp01$wget
http://downloads.openwrt.org/kamikaze/8.09.2/kamikaze_8.09.2_source.tar.bz2
:~/mp01$ tar -xjf kamikaze_8.09.2_source.tar.bz2
:~/mp01$
```

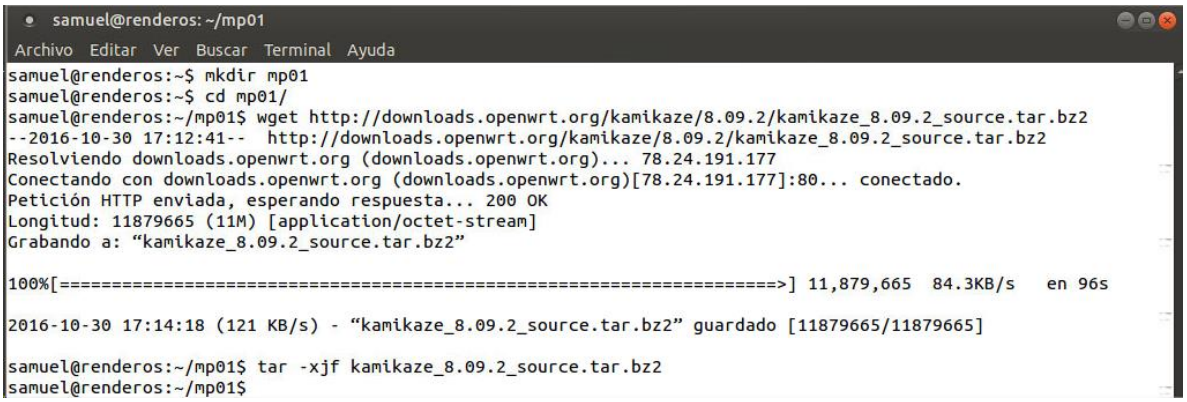

Figura 20. Descarga del SDK de OpenWRT para la versión Kamikaze

Se compilan los fuentes para generar el toolchain de la versión Kamikaze.

```
:~/mp01$ cd 8.09.2
:~/mp01/8.09.2$ make prereq
:~/mp01/8.09.2$ make
```

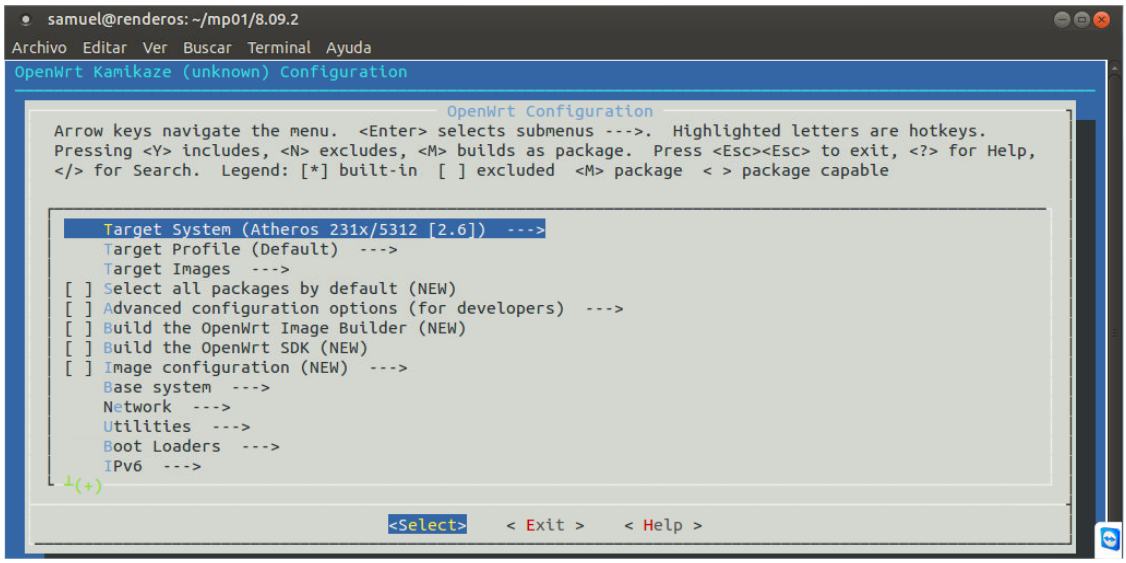

Figura 21. Compilación de los fuentes de OpenWRT para la versión Kamikaze

En la Figura 21 solamente se configura el chip del sistema objetivo, Atheros 2317 para el router MP01.

Se copia el toolchain generado y también se descargan las librerías implementadas, libmodbus y sqlite3.

```
:~/mp01$ cp -r 8.09.2/staging_dir/toolchain-mips_gcc4.1.2/ .
:~/mp01$ wget http://libmodbus.org/releases/libmodbus-3.1.2.tar.gz
:~/mp01$ wget https://sqlite.org/2016/sqlite-amalgamation-3150000.zip
:~/mp01$ tar -xzf libmodbus-3.1.2.tar.gz
:~/mp01$ unzip sqlite-amalgamation-3150000.zip
```

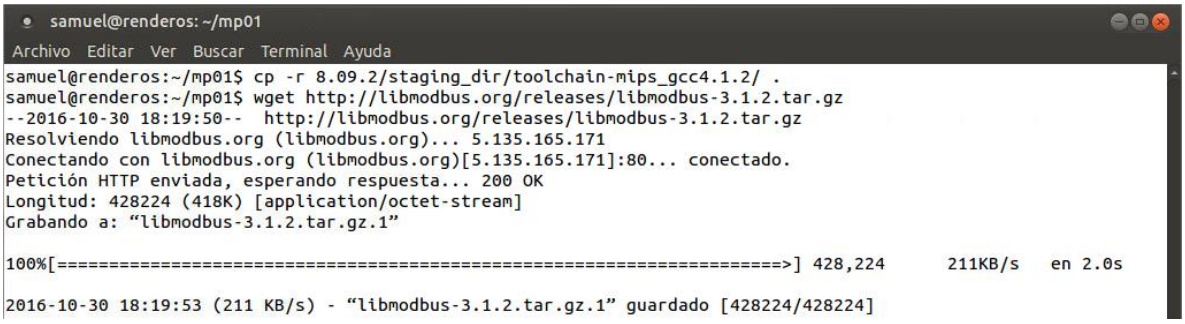

Figura 22. Descarga de la librería Libmodbus

Luego se exportan las siguientes variables de entorno, teniendo en cuenta la ruta del directorio toolchain previamente descargado. Este paso necesario para que el compilador enlace los archivos binarios correspondientes al toolchain en el proceso de compilar la librería libmodbus.

```
:~/mp01$ export TOOLCHAIN=/home/samuel/mp01/toolchain-mips_gcc4.1.2
:~/mp01$ export PATH=$PATH:$TOOLCHAIN/bin
:~/mp01$ export AR=$TOOLCHAIN/bin/mips-linux-uclibc-ar
:~/mp01$ export AS=$TOOLCHAIN/bin/mips-linux-uclibc-as
:~/mp01$ export LD=$TOOLCHAIN/bin/mips-linux-uclibc-ld
:~/mp01$ export NM=$TOOLCHAIN/bin/mips-linux-uclibc-nm
:~/mp01$ export CC=$TOOLCHAIN/bin/mips-linux-uclibc-gcc
:~/mp01$ export CPP=$TOOLCHAIN/bin/mips-linux-uclibc-cpp
:~/mp01$ export GCC=$TOOLCHAIN/bin/mips-linux-uclibc-gcc
:~/mp01$ export RANLIB=$TOOLCHAIN/bin/mips-linux-uclibc-ranlib
```

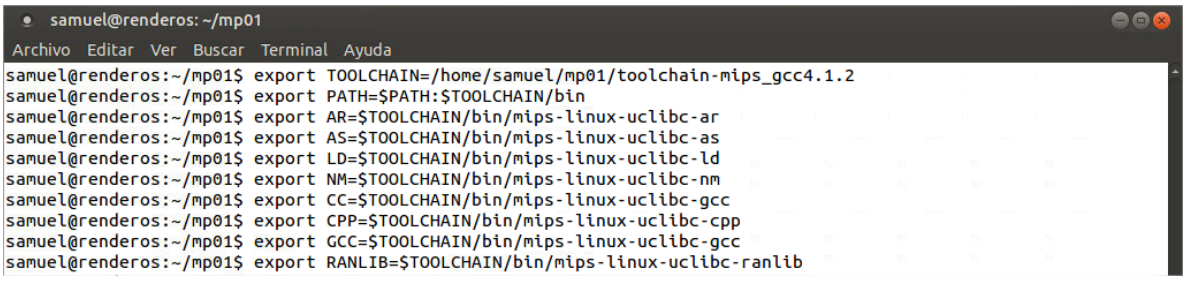

Figura 23. Variables de entorno para los script ejecutables del toolchain

Se generan los scripts de la librería libmodbus y se enlazan el compilador del toolchain mediante el script *configure*, colocándole como parámetro la arquitectura para la cual se compilara la librería libmodbus.

```
:~/mp01$ cd libmodbus-3.1.2
:~/mp01/libmodbus-3.1.2$./configure --host=mips-linux-uclibc
```

Procedemos a compilar la librería completa con la herramienta make.

```
:~/mp01/libmodbus-3.1.2$ make
```

Figura 24. Resultado de la compilación de libmodbus

Para finalizar se edita el archivo libmodbus.pc, de tal manera que el compilador enlace las librerías estáticas y dinámicas desde el mismo directorio donde se generaron los *scripts* como se muestra en la Figura 25. Libmodbus.pc es fichero generado por la librería libmodbus, se utiliza para configurar los directorios donde se encuentran los archivos fuentes de la librería.

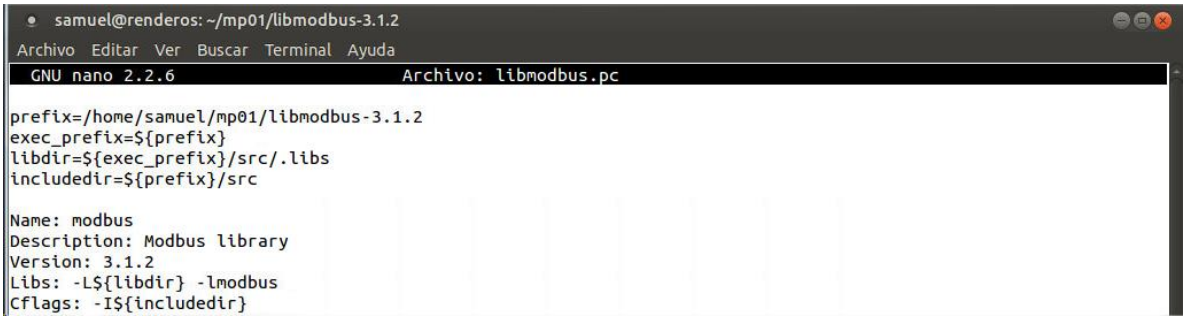

Figura 25. Modificación del archivo libmodbus.pc

Antes de ejecutar aplicaciones libmodbus en el sistema objetivo, es necesario copiar una de las librerías dinámicas (libmodbus.so.5) al directorio de librerías principal del sistema objetivo. Para el caso del router MP01, el directorio mencionado es /usr/lib. En la Figura 26 se traslada la librería libmodbus.so.5 al router MP01.

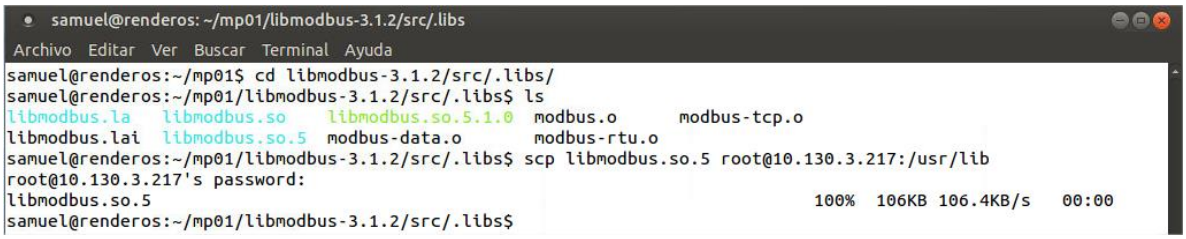

Figura 26. Envió de la librería dinámica libmodbus al router MP01

La compilación de cualquier aplicación para un sistema objetivo, se realiza mediante la herramienta pkg-config, la cual provee una interfaz unificada para llamar librerías cuando se está compilando un programa a partir del código fuente.

Figura 27. Compilación de aplicación de consulta

En la Figura 27 se ha compilado la aplicación de consulta, el primer paso consiste en definir la variable de entorno PKG_CONFIG_PATH con la ruta del directorio donde se encuentra el archivo libmodbus.pc, luego se compila mediante la siguiente estructura.

Segmento de código	Descripción
/home/samuel/mp01/toolchain-mips_gcc4.1.2/bin/mips-openwrt-linux-uclibc-gcc	Compilador para C del toolchain
clientetcp-mp100.c	Archivo fuente de la aplicación de consulta
sqlite-amalgamation-3150000/sqlite3.c	Archivo fuente de la librería SQLite
-o clientetcp-mp100	Nombre del archivo ejecutable
`pkg-config --libs --cflags libmodbus`	Enlace de la librería libmodbus mediante pkg-config
-I sqlite-amalgamation-3150000 -lpthread -ldl -w	Enlace del directorio de los archivos de encabezado de sqlite.

Tabla 3. Descripción de la estructura de compilación.

2.5 Aplicación de consulta y almacenamiento

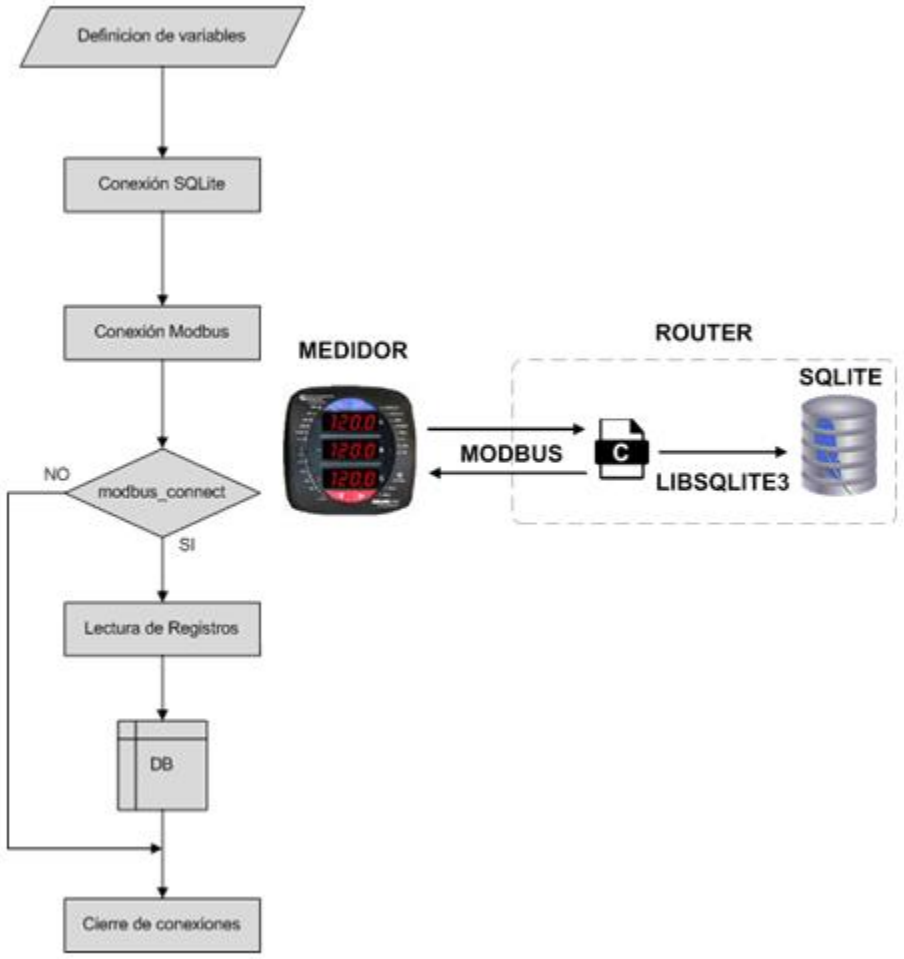

Figura 28. Esquema y flujograma de la aplicación de consulta

En la Figura 28 se muestra el esquema de comunicación que realiza la aplicación de consulta y el flujograma del funcionamiento general.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <modbus.h>
#include <time.h>
#include "sqlite3.h"

#define MEDIDOR 0x01    //Direccion asignada al servidor

int main(int argc, char *argv[])
{
  modbus_t *ctx;      //Crea puntero
  sqlite3 *db;
  uint16_t tab_reg_wh[2];
```

```
  uint16_t tab_reg_vah[2];
  uint16_t tab_reg_demanda[2];
  uint16_t toFloat[2];
  uint16_t toInt[2];
  int rc, rc_sql;
  float VAN, VAB, voltajeBC;
  char *zErrMsg = 0;
  char query[1024];
```

Figura 29. Segmento de definición de variables de la aplicación de consulta

En la Figura 29 se muestra el segmento inicial del programa de consulta, donde se declara el identificador de la dirección asignada al servidor. La función principal se configura para que la aplicación reciba parámetros. Luego se declara un puntero *ctx a la estructura modbus_t donde se almacenara toda la información de conexión, también se declaran las variables de lectura, conversión y de control de flujo.

```
/* Establecer un contexto MODBUS */
  ctx = modbus_new_tcp(argv[1], 502); //Usando contexto TCP, ip y puerto
  modbus_set_slave(ctx, MEDIDOR);

  if (ctx == NULL) {
    fprintf(stderr, "No pudo ubicar el contexto libmodbus\n");
    return -1;
    }
/* Establece la conexion SQLITE*/
  rc_sql = sqlite3_open("db", &db);
  if( rc_sql ){
    fprintf(stderr, "Error en la apertura de la base de datos: %s\n",
sqlite3_errmsg(db));
    sqlite3_close(db);
    exit(1);
  }

/* Establece la conexion MODBUS*/
  if (modbus_connect(ctx) == -1) {
    fprintf(stderr, "La conexion fallo: %s\n", modbus_strerror(errno));
    modbus_free(ctx);
    return -1;
  }
```

Figura 30. Segmento de conexión a SQLite y Modbus de la aplicación de consulta

Luego sigue el segmento de conexiones como se observa en la Figura 30. La función modbus_new_tcp establece el contexto Modbus para TCP/IP. El primer argumento de la función es la IP del medidor y corresponde al primer parámetro enviado desde línea de comandos, el segundo argumento es el puerto de comunicación, 502 por defecto.

La función modbus_set_slave establece el número de identificador de dispositivo en el contexto TCP/IP. Luego se conecta a la base de datos SQLite mediante la función

sqlite3_open. Se establece la conexión con el Medidor utilizando la función modbus_connect, devolviendo cero si tuvo éxito. De lo contrario devuelve -1 y produce un error.

La Figura 31 muestra el segmento de lecturas utilizando la función modbus_read_registers. El resultado es guardado en tab_red_wh para W-hours, tab_reg_vah para VA-hours y tab_red_demanda para Watts. La función devuelve el número de los registros leídos si tuvo éxito, de lo contrario devolverá -1 y produce un error.

```
/*Lectura de W-hours, Total. Registro 1106 en el mapa modbus */
  rc = modbus_read_registers(ctx, 1105, 2, tab_reg_wh);
  if (rc == -1) {
    fprintf(stderr, "Fallo lectura W-hours, Total de Agronomia: %s",
modbus_strerror(errno));
    return -1;
  }
/*Lectura de VA-hours, Total. Registro 1116 en el mapa modbus*/
  rc = modbus_read_registers(ctx, 1115, 2, tab_reg_vah);
  if (rc == -1) {
    fprintf(stderr, "Intento Fallo Lectura VA-hours, Total en Agronomia:
%s", modbus_strerror(errno));
    return -1;
  }
/* Lectura de Positive Watts, 3-Ph, Average, Registro 2006 en el mapa
modbus */
  rc = modbus_read_registers(ctx, 2005, 2, tab_reg_demanda);
  if (rc == -1) {
    fprintf(stderr, "Intento Fallo lectura de Positive Watts, 3-Ph,
Average en Agronomia: %s", modbus_strerror(errno));
    return -1;
  }
```

Figura 31. Segmento de lecturas de la aplicación de consulta

En la Figura 32 se muestra el segmento de código donde se concatenan los registros de tab_reg_wh y tab_reg_vah convirtiéndolos a un entero de 32 bits mediante la función MODBUS_GET_INT32_FROM_INT16. Para el caso de tab_reg_demanda, primero se invierten los registros y luego se convierten a formato Float utilizando la función modbus_get_float.

```
/*Convirtiendo a entero los valores de W-hours y VA-hours a formato
SINT32 (registros de 16 bits no invertidos)*/

  int Wh_Tot=0;
  toInt[0]=tab_reg_wh[0];
  toInt[1]=tab_reg_wh[1];
  Wh_Tot=MODBUS_GET_INT32_FROM_INT16(toInt,0);

  int VAh_Tot=0;
```

```
  toInt[0]=tab_reg_vah[0];
  toInt[1]=tab_reg_vah[1];
  VAh_Tot=MODBUS_GET_INT32_FROM_INT16(toInt,0);

/*Convirtiendo a flotante los valores de Demanda*/

  float Pos_Watts_3ph_Average = 0.0;
  toFloat[0]=tab_reg_demanda[1];
  toFloat[1]=tab_reg_demanda[0];
  Pos_Watts_3ph_Average=modbus_get_float(toFloat);
```

Figura 32. Segmento de conversión de variables de la aplicación de consulta

En la Figura 33 se muestra el bloque de código donde los datos leídos se insertan en la tabla Medidor de la base de datos SQLite. El primer paso consiste en formatear la query con las variables resultantes de la conversión para luego ejecutarla utilizando la función sqlite3_exec.

Para finalizar se utiliza la función modbus_close, la cual cierra la conexión establecida con el medidor. La función modbus_free libera la información de la estructura modbus_t. La función sqlite_close cierra la conexión con la base de datos SQLite.

```
  sprintf(query, "insert into Medidor(WhTot,VAhTot,Pos_Watts_3ph_Av)
values ('%d','%d','%f')",Wh_Tot,VAh_Tot,Pos_Watts_3ph_Average);

  rc_sql = sqlite3_exec(db, query, 0, 0, &zErrMsg);
  if( rc sql!=SQLITE OK ){
    fprintf(stderr, "SQL error: %s\n", zErrMsg);
    /* This will free zErrMsg if assigned */
    if (zErrMsg)
      free(zErrMsg);
  }
 /* Cierra la conexion MODBUS */
  modbus_close(ctx);
  modbus_free(ctx);

  sqlite3_close(db);

  return 0;
}
```

Figura 33. Segmento de almacenamiento y cierre de conexiones.

2.5.1 Configuración de SQLite en router MP01

En el Anexo A se detalla una guía de configuraciones que se deben de realizar en el router MP01 con la finalidad de correr SQLite y XML-RPC. El protocolo XML-RPC permite enviar datos por medio de HTTP en formato XML. En la sección 3.1 se detallará su funcionalidad.

Para que la aplicación de consulta pueda ejecutarse en el router MP01 sin ningún problema, se debe crear la tabla Medidor con los campos mostrados en la Tabla 6.

Medidor
Id (INT)
Fecha_hora (TIMESTAMP)
WhTot (INT)
VAhTot (INT)
Pos_Watts_3ph_Av (FLOAT)

Tabla 4. Estructura tabla Medidor

Id: Identificador único de cada registro. Fecha_hora: Fecha y hora de cada registro. WhTot: Valor de Watt-Horas Totales. VAhTot: Valor de Voltio-Amperio-Horas Totales. Pos_Watts_3ph_Av: Demanda máxima.

Nos conectamos al router MP01 vía ssh y creamos la tabla Medidor mediante el comando sqlite3 siguiendo los pasos descritos a continuación. Sqlite3 es una aplicación para OpenWRT que realiza operaciones sobre una base de datos SQLite.

```
:/# sqlite3 db "CREATE TABLE Medidor (Id INTEGER PRIMARY KEY
AUTOINCREMENT, Fecha_hora TIMESTAMP DATE DEFAULT
(datetime('now','localtime')), WhTot INTEGER (11), VAhTot INTEGER (11),
Pos_Watts_3ph_Av FLOAT)"
```

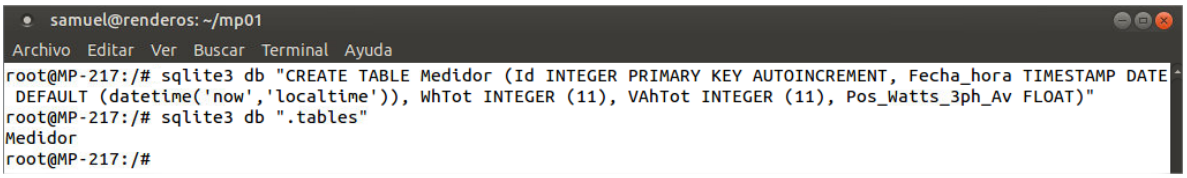

Figura 34. Creación de tabla Medidor en Router MP01

2.5.2 Pruebas y resultados

Debido a la variación de las direcciones del mapa de memoria en los medidores tipo SHARK 100S y SHARK 200-200S, se crearon dos aplicaciones de consulta mostradas en la Tabla 7, con la misma lógica de programación, diferenciándolas por las direcciones utilizadas para realizar las lecturas como se observa en la Figura 35 y Figura 36.

```
/*Lectura de W-hours, Total. Registro 1106 en el mapa modbus */
  rc = modbus_read_registers(ctx, 1105, 2, tab_reg_wh);
  if (rc == -1) {
    fprintf(stderr, "Fallo lectura W-hours, Total de Agronomia: %s",
modbus_strerror(errno));
    return -1;
  }

/*Lectura de VA-hours, Total. Registro 1116 en el mapa modbus*/
  rc = modbus_read_registers(ctx, 1115, 2, tab_reg_vah);
  if (rc == -1) {
    fprintf(stderr, "Intento Fallo Lectura VA-hours, Total en Agronomia:
```

```
%s", modbus_strerror(errno));
    return -1;
  }

/* Lectura de Positive Watts, 3-Ph, Average, Registro 2006 en el mapa
modbus */
  rc = modbus_read_registers(ctx, 2005, 2, tab_reg_demanda);
  if (rc == -1) {
    fprintf(stderr, "Intento Fallo lectura de Positive Watts, 3-Ph,
Average en Agronomia: %s", modbus_strerror(errno));
    return -1;
  }
```

Figura 35. Segmento de lectura de la aplicación clientetcp-mp100

```
/*Lectura de W-hours, Total. Registro 1506 en el mapa modbus */
  rc = modbus_read_registers(ctx, 1505, 2, tab_reg_wh);
  if (rc == -1) {
    fprintf(stderr, "Fallo lectura W-hours, Total de Agronomia: %s",
modbus_strerror(errno));
    return -1;
  }

/*Lectura de VA-hours, Total. Registro 1516 en el mapa modbus*/
  rc = modbus_read_registers(ctx, 1515, 2, tab_reg_vah);
  if (rc == -1) {
    fprintf(stderr, "Intento Fallo Lectura VA-hours, Total en Agronomia:
%s", modbus_strerror(errno));
    return -1;
  }

/* Lectura de Positive Watts, 3-Ph, Average, Registro 2006 en el mapa
modbus */
  rc = modbus_read_registers(ctx, 2005, 2, tab_reg_demanda);
  if (rc == -1) {
    fprintf(stderr, "Intento Fallo lectura de Positive Watts, 3-Ph,
Average en Agronomia: %s", modbus_strerror(errno));
    return -1;
  }
```

Figura 36. Segmento de lectura de la aplicación clientetcp-mp200

Compilación de las aplicaciones de consulta.

```
:~/mp01$ export PKG_CONFIG_PATH=/home/samuel/mp01/libmodbus-3.1.2
:~/mp01$ /home/samuel/mp01/toolchain-mips_gcc4.1.2/bin/mips-linux-
uclibc-gcc clientetcp-mp100.c sqlite-amalgamation-3150000/sqlite3.c -o
clientetcp-mp100 `pkg-config --libs --cflags libmodbus` -I sqlite-
amalgamation-3150000 -lpthread -ldl -w
:~/mp01$       /home/samuel/mp01/toolchain-mips_gcc4.1.2/bin/mips-linux-
uclibc-gcc  clientetcp-mp200.c  sqlite-amalgamation-3150000/sqlite3.c  -o
clientetcp-mp200  `pkg-config  --libs  --cflags  libmodbus`  -I  sqlite-
amalgamation-3150000 -lpthread -ldl -w
```

Figura 37. Compilación de las aplicaciones clientetcp-mp100 y clientetcp-mp200.

Envió de la aplicación clientetcp-mp100 al router MP01 del medidor de Agronomía.

```
:~/mp01$ scp clientetcp-mp100 root@10.130.3.217:/
```

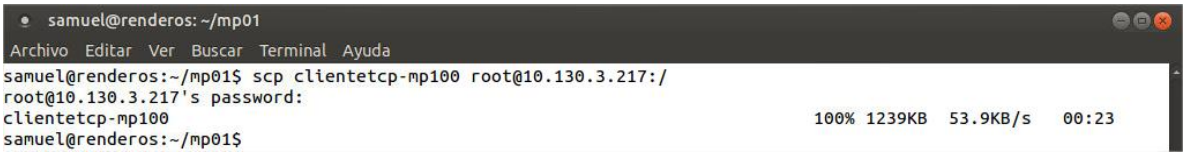

Figura 38. Envió de la aplicación al router mp01 mediante scp

Corremos el programa de consulta, pasando como argumento la IP del medidor de Agronomía.

```
:/# ./clientetcp-mp100 10.30.217.2
:/# sqlite3 db 'select * from Medidor'
```

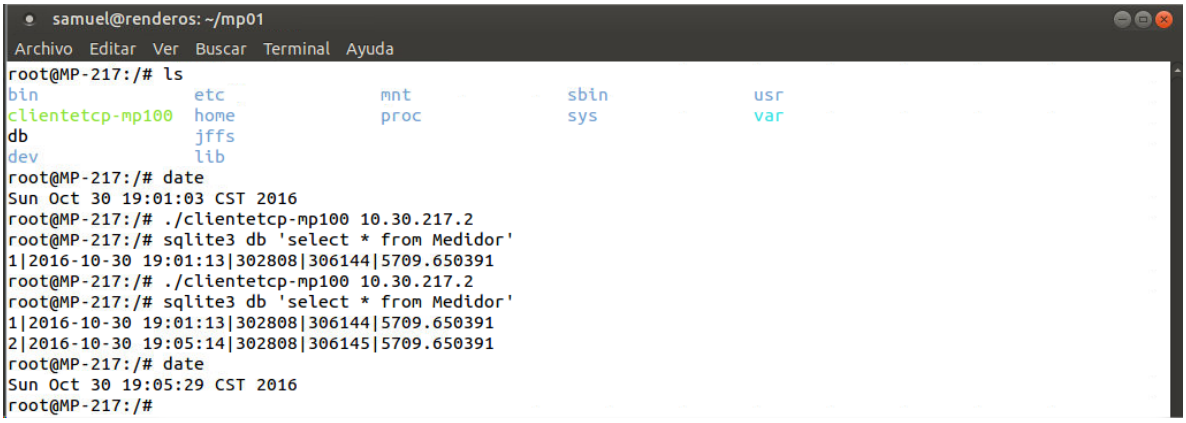

Figura 39. Resultado de la aplicación de consulta

Para finalizar programamos una tarea en el router para ejecutar la aplicación de consulta cada 5 minutos, utilizando la herramienta crontab.

```
:/# crontab -e
*/5 * * * * cd / && ./clientetcp-mp100 10.30.217.2
```

Figura 40. Tarea crontab programada en el router mp01

Los campos que son parte de la estructura de crontab están formados de la siguiente manera:

```
Minuto Hora DiaDelMes Mes DiaDeLaSemana Comando
```

Minuto: Controla el minuto de la hora en que el comando será ejecutado.

Hora: Controla la hora en que el comando será ejecutado, se especifica en un formato de 24 horas, los valores deben estar entre 0 y 23, 0 es medianoche.

DiaDelMes: Día del mes en que se quiere ejecutar el comando. Por ejemplo se indicaría 20, para ejecutar el comando el día 20 del mes.

Mes: Mes en que el comando se ejecutará, puede ser indicado numéricamente (1-12), o por el nombre del mes en inglés, solo las tres primeras letras.

DiaDeLaSemana: Día en la semana en que se ejecutará el comando, puede ser numérico (0-7) o por el nombre del día en inglés, solo las tres primeras letras. (0 y 7 = domingo).

Comando: Comando, script o programa que se desea ejecutar. Este campo puede contener múltiples palabras y espacios.

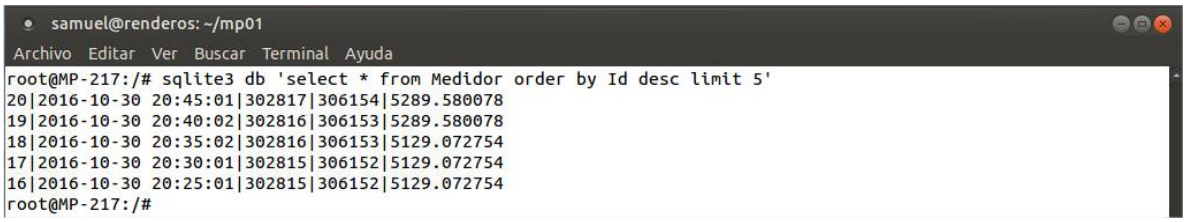

Figura 41. Resultados de la tarea programada sobre la aplicación de consulta.

De esta forma la aplicación de consulta interrogara constantemente al medidor y almacenará el resultado en la tabla Medidor dentro del router MP01. En la Figura 40 se programa una tarea crontab para ejecutar la aplicación de consulta cada 5 minutos y en la Figura 41 se muestra el resultado luego de 20 minutos aproximadamente, donde se observa las capturas realizadas en ese rango de tiempo.

Capítulo 3: Envió de los datos al servidor mediante XLM-RPC

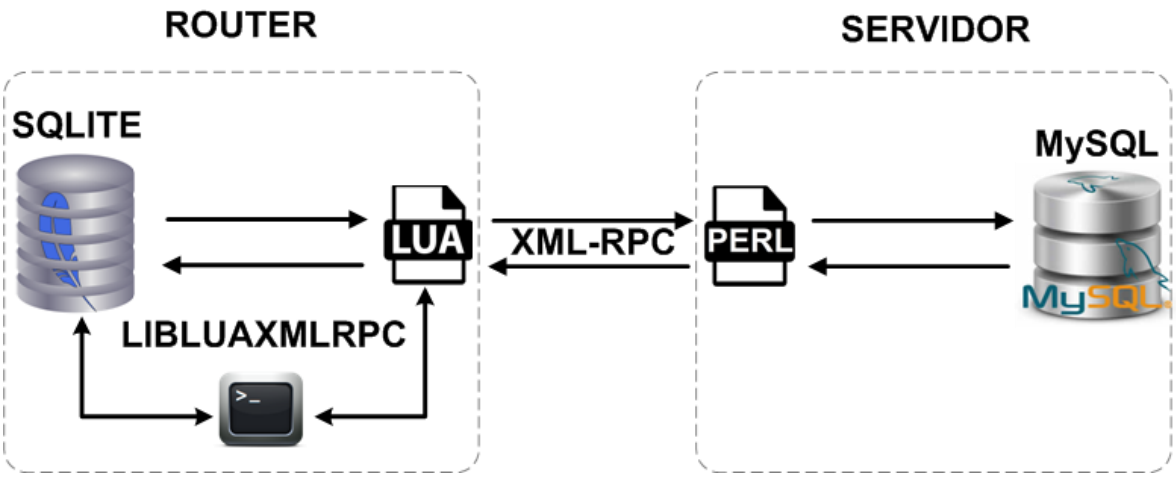

Figura 42. Esquema de comunicación router – servidor

De manera permanente el router estará intentando enviar los datos al servidor. Para lograrlo se desarrollará una aplicación en LUA basada en el protocolo XML-RPC. LUA es un lenguaje de programación extensible diseñado para una programación procedimental y viene instalado en el firmware del router MP01. La aplicación será capaz de consultar la base de datos SQLite y cada vez que se tengan nuevos datos, intentará enviarlos, mediante la ejecución de un procedimiento remoto, al servidor.

El servidor será capaz de recibir los datos y guardarlos en una base de datos MySQL. Para ello se desarrollará otra aplicación en PERL basada en el protocolo XML-RPC. La aplicación estará en sincronización con la aplicación del router.

3.1 Protocolo XML-RPC

XML-RPC Se trata de una especificación que permite enviar datos por medio de HTTP en formato XML [18]. También se pueden realizar llamadas de procedimiento remotos a través de Internet. Está llamadas a procedimientos remotos se realiza a través de HTTP como el transporte y XML como la codificación. XML-RPC está diseñado para ser tan simple como sea posible, permitiendo al mismo tiempo las estructuras de datos complejas para ser transmitida, procesada y devuelta.

En la Figura 43 se muestra la lógica que utiliza XML-RPC para trasportar los datos. Los datos a enviar se codifican en XML según la especificación. Posteriormente se trasportan por medio de HTTP. Además de los datos se envía el nombre del procedimiento remoto a ejecutar.

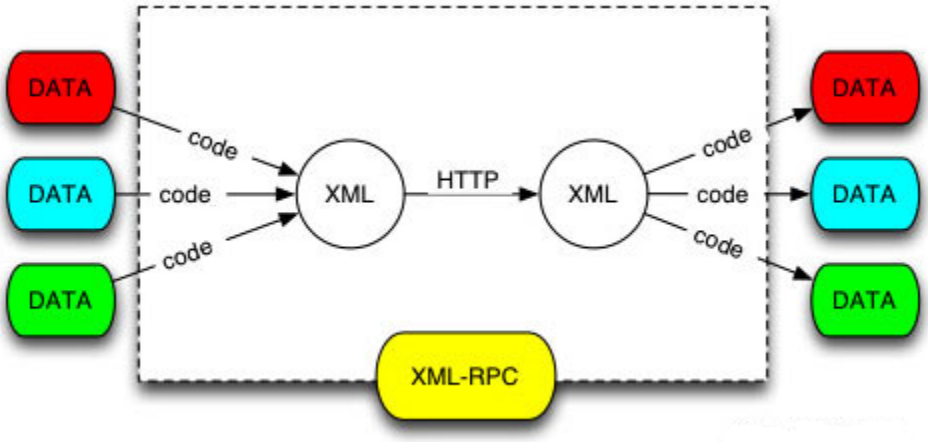

Figura 43. Esquema de comunicación XML-RPC

Una de las ventajas de utilizar XML-RPC es que la comunicación se puede realizar en sistemas operativos diferentes y en entornos diferentes como lo son PERL, LUA, PHP, JAVA etc [19]. Los lenguajes de programación mencionados poseen librerías XML-RPC para hacer más flexible la programación de aplicaciones que utilicen llamadas a procedimientos remotos. El entorno o lenguaje de programación utilizado por el cliente y el servidor pueden ser diferentes. Para el presente trabajo de graduación realizamos la comunicación entre un cliente utilizando LUA y un servidor que utiliza PERL.

3.2 Implementación y configuración de XML-RPC en router MP01

3.2.1 Lua Socket

Lua Socket es una librería de LUA. Se compone de dos partes: un núcleo C que proporciona soporte para los protocolos TCP y UDP y un conjunto de módulos de LUA que añaden soporte para las aplicaciones que tienen que ver con la Internet [20].

En 2011 se desarrolló el trabajo de graduación sobre un medidor inalámbrico de consumo de energía eléctrica de bajo costo, mediante la modificación del router DIR-300. Se investigó e implementó la librería LuaSocket [21].

En nuestro caso también utilizamos la librería lua-socket. Ésta viene en el paquete luasocket_2.0.2-1_atheros.ipk. Y lo tomamos de los paquetes de OpenWrt para Flukso. Además se utiliza la librería xmlrpc.http que viene en el paquete: liblua-xmlrpc_1.0x-1_atheros.ipk. Al igual que lua-socket, se encuentra en los paquetes de OpenWrt para Flukso. También, se pueden encontrar estas librerías en los repositorios de OpenWrt.

En el Anexo A se detalla una guía de configuraciones que se deben de realizar en el router MP01 con la finalidad de correr SQLite y XMLRPC.

3.3 Aplicación cliente XLM-RPC en router MP01

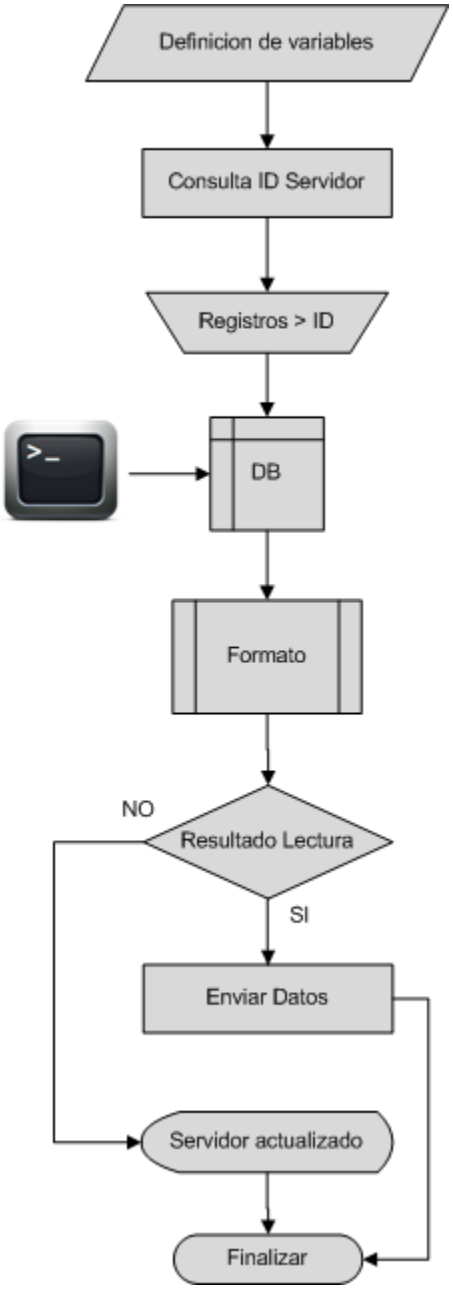

Figura 44. Flujograma de la aplicación cliente XML-RPC

En la Figura 45 se muestra el esquema de comunicación que realiza la aplicación cliente XML-RPC. En la Figura 44 se muestra el flujograma del funcionamiento general. Se utiliza un comando del Shell de Linux para interactuar con la base de datos SQLite.

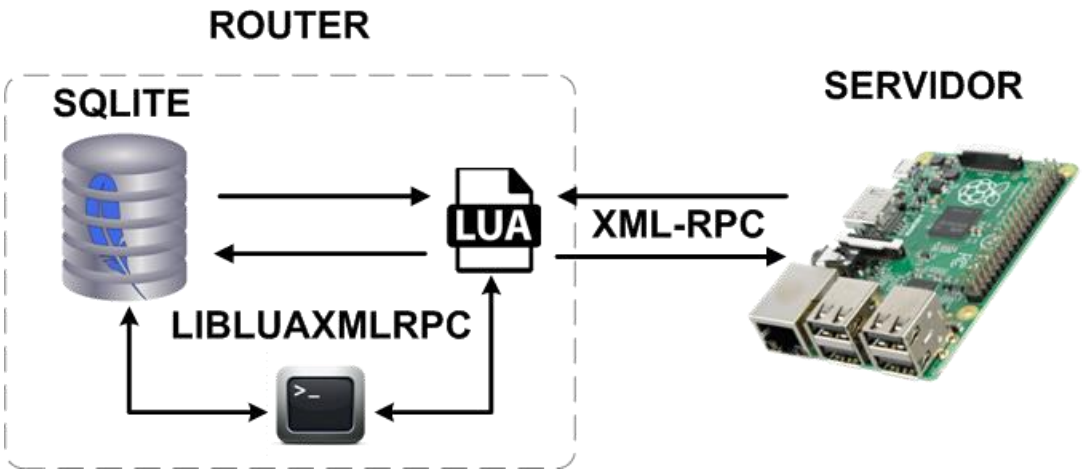

Figura 45. Esquema de aplicación cliente XML-RPC

En la Figura 46 se muestra el segmento de consulta al servidor sobre el último ID almacenado en su base de datos local MySQL. La función xmlrpc.http.call ejecuta un procedimiento remoto llamado serverXMLRPV2, alojado en la dirección: http://10.130.3.150/cgi-bin/serverXMLRPCV2.cgi. La dirección del servidor donde se envían los datos. Se especifica que es una consulta y se envía el nombre del medidor como argumento. El resultado se guarda en la variable "id" si la consulta tuvo éxito.

```
require("socket.http")
require("xmlrpc.http")

id = 0

local ok, res = xmlrpc.http.call("http://10.130.3.150/cgi-
bin/serverXMLRPCV2.cgi", "serverXMLRPCV2", "Consulta", arg[1])
assert(ok, string.format("Fallo la conexión XML-RPC: %s",
tostring(res)))
for i, v in pairs(res) do
      id = v
      print (i, v)
end
```

Figura 46. Segmento de código de xmlpiV2.lua que solicita el último ID al servidor

Luego se realiza la consulta de todos los registros mayores a la variable "id", devolviendo los registros mayores al último dato almacenado en el servidor. Para lograr esto se utiliza la herramienta sqlite3, ejecutando la línea de comando mostrada en la Figura 47.

```
local handle = io.popen("sqlite3 -separator $',' db 'select * from
Medidor where Id > " .. id .. " order by Id limit "..arg[2].."'")
local result = handle:read("*a")
handle:close()
```

Figura 47. Segmento de condigo de xmlpiV2.lua que consulta SQLite

Debido a que no se encontró una forma de comunicar directamente LUA con SQLite, se optó por utilizar la funcion io.popen, la cual ejecuta una linea de comando para el shell de Linux y el resultado lo almacena como fichero temporal.

El resultado de la consulta se formatea para poderlo enviar como sentencia SQL, en el segmento de codigo de la Figura 48, primero cada registro se filtra por el carácter de nueva linea(\n) y cada campo se filtra por coma(,). Luego se construye el bloque de registros a enviar de acuerdo al siguiente formato:

(Id1,'AA:MM:DD HH:mm:SS', WhTot1, VAhTot1, Pos_Watts_3ph_Av1),(Id2,'AA:MM:DD HH:mm:SS', WhTot2, VAhTot2, Pos_Watts_3ph_Av2),(Id3,'AA:MM:DD HH:mm:SS', WhTot3, VAhTot3, Pos_Watts_3ph_Av3)…

```
r={ }
consulta=""
i=1
j=0

for x in string.gmatch(result, "[^\n]+") do
      for y in string.gmatch(x, "[^,]+") do
            r[i] = y
            i = i+1
      end
      consulta = consulta.."("..r[i-5]..",'"..r[i-4].."',"..r[i-
3].."),"..r[i-2].."),"..r[i-1].."),"
      j = j+1
end

consulta = string.sub(consulta,1,-2)
```

Figura 48. Segmento de código de xmlpiV2.lua para formatear los registros a enviar

Para finalizar se vuelve a utilizar la función xmlrpc.http.call para enviar el bloque de datos con el formato preestablecido. Se llama al mismo procedimiento remoto a diferencia que el primer parámetro corresponde al nombre del medidor seguido de los datos a enviar.

```
if result then
      print("Datos enviados: " .. consulta)

      local ok, res = xmlrpc.http.call("http://10.130.3.150/cgi-
bin/serverXMLRPCV2.cgi", "serverXMLRPCV2", arg[1], consulta, j)
      assert(ok, string.format("XML-RPC call failed on client: %s",
tostring(res)))

      for i, v in pairs(res) do
            print (i, v)
            if v == 'OK' then
                  os.execute("sqlite3 db 'DELETE FROM Medidor WHERE Id <
"..id.."'")
```

```
            end
      end
else
      print("Servidor actualizado!")
end
```

Figura 49. Segmento de código de xmlpiV2.lua para enviar los datos al servidor

En resumen, el primer paso de la aplicación cliente XML-RPC consiste en consultar al servidor el último dato almacenado en su base MySQL. Luego se consultan todos los registros mayores al dato recibido en SQLite. Si hay nuevos datos, estos se formatean en una sintaxis SQL y se envían al servidor. La aplicación servidor XML-RPC se encarga de insertarlos en la tabla correspondiente del medidor de la base de datos MySQL.

3.4 Implementación y configuración de XML-RPC en Raspberry Pi 2

3.4.1 Raspberry Pi 2

Raspberry Pi 2 es una micro computadora de bajo coste desarrollada en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas [22].

El diseño incluye un System-on-a-chip Broadcom BCM2836, que contiene un procesador ARM11 ARMv7 ARM Cortex-A7 4 núcleos a 900 MHz, un procesador gráfico (GPU) VideoCore IV, y 1 GB de memoria RAM aunque originalmente en sus primeras versiones fue de 256 MB y 512 MB de memoria RAM. El diseño no incluye un disco duro o una unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente; tampoco incluye fuente de alimentación o carcasa.

La fundación Raspberry Pi da soporte para descargas de las distribuciones para arquitectura ARM, Raspbian (derivada de Debian), RISC OS y Arch Linux; y promueve principalmente el aprendizaje del lenguaje de programación Python y otros lenguajes como Tiny BASIC, C y Perl.

Raspbian es un sistema operativo basado en Debian 7 Wheezy el cual se ha convertido en el sistema operativo más utilizado para la Raspberry Pi, La distribución Debian 7 tiene soporte para microprocesadores ARM, por esa razón se adaptó con facilidad.

3.4.2 Configuración de CGI y XML-RPC en Raspberry Pi 2

El primer paso es instalar el servidor MySQL y las librerías necesarias de XML-RPC y MySQL para Perl.

```
pi@raspberrypi:~$ sudo apt-get install mysql-server
pi@raspberrypi:~$ sudo apt-get install libdbd-mysql-perl
pi@raspberrypi:~$ sudo apt-get install libdbi-perl
pi@raspberrypi:~$ sudo apt-get install libfrontier-rpc-perl
```

En la Figura 50 se muestra la versión instalada de MySQL Server.

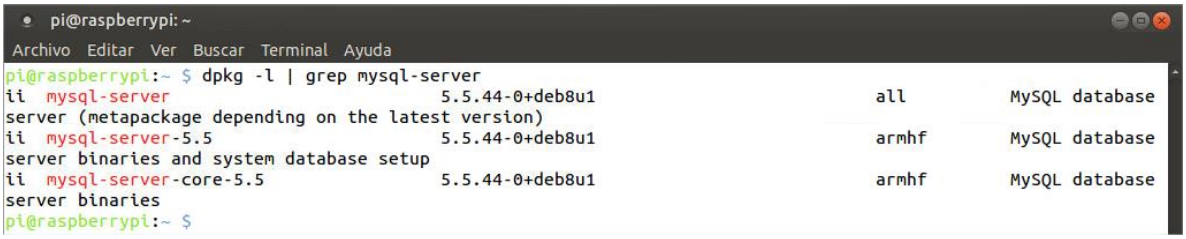

Figura 50. Versión instalada de MySQL Server

El la Figura 51 se muestran las versiones instaladas de las librerías XML-RPC y MySQL para Perl.

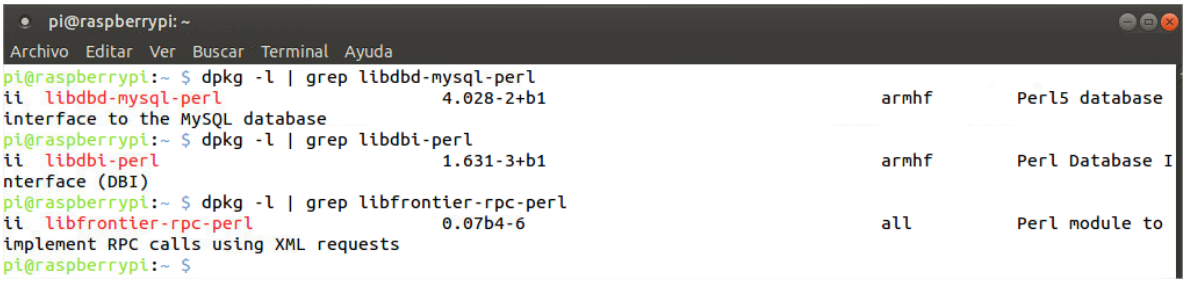

Figura 51. Versiones instaladas de las librerías XML-RPC y MySQL para Perl

Creamos el directorio donde ubicaremos la aplicación servidor XML-RPC.

```
pi@raspberrypi:~$ sudo mkdir /var/cgi-bin
pi@raspberrypi:~$ chmod +x /var/cgi-bin/serverXMLRPCV2.cgi
```

Figura 52. Aplicación servidor XML-RPC

Configuramos Apache para habilitar el servidor CGI agregando el siguiente bloque, en el archivo /etc/apache2/sites-enabled/000-default.conf.

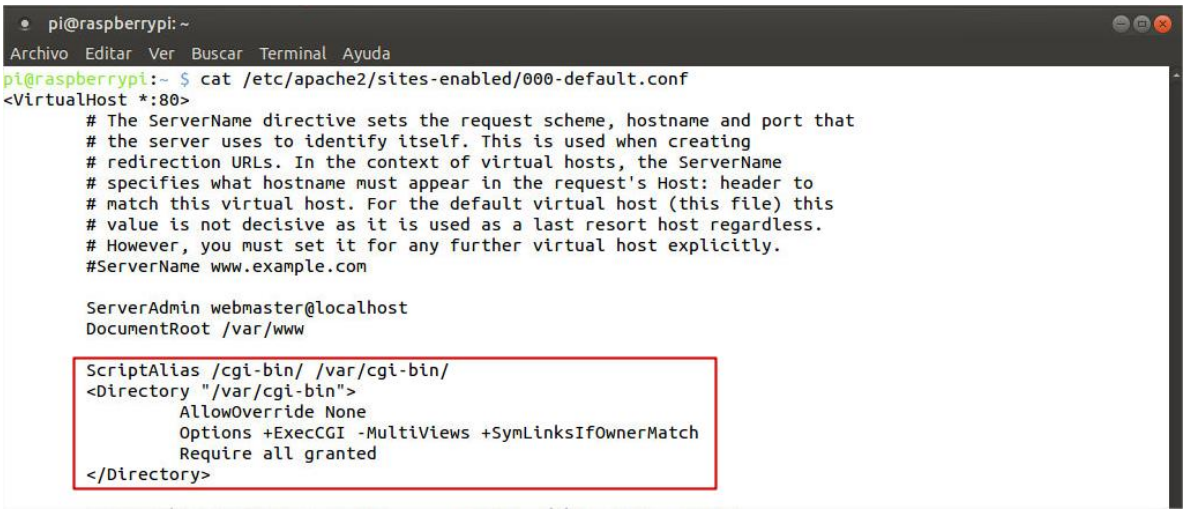

Figura 53. Configuración de Apache para habilitar CGI

Para finalizar se agregan los enlaces simbólicos siguientes para habilitar aplicaciones CGI.

```
pi@raspberrypi:~$ sudo ln -s /etc/apache2/mods-available/cgid.load
/etc/apache2/mods-enabled/
pi@raspberrypi:~$ sudo ln -s /etc/apache2/mods-available/cgid.conf
/etc/apache2/mods-enabled/
pi@raspberrypi:~$ sudo service apache2 reload
```

3.5 Aplicación servidor XML-RPC en Raspberry Pi 2

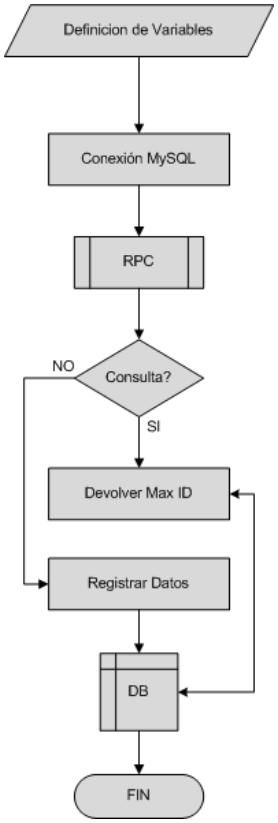

Figura 54. Flujograma de la aplicación servidor XML-RPC

La Figura 54 muestra el flujograma de la aplicación servidor XML-RPC, la cual se encarga de recibir los datos de la aplicación cliente XML-RPC y los guarda en una tabla local MySQL.

En el primer segmento de código mostrado en la Figura 55, se definen las variables para establecer la conexión con la base de datos MySQL. Además el procedimiento serverXMLRPCV2 recibe tres parámetros, c1, c2 y c3.

```
#!/usr/bin/perl -w

use strict;
use Frontier::RPC2;
use DBI;

sub serverXMLRPC {
    my ($c1, $c2, $c3) = @_;

    my $driver = "mysql";
    my $database = "testdb";
    my $dsn = "DBI:$driver:database=$database";
    my $userid = "test";
```

```
    my $password = "test";
```

Figura 55. Segmento de código para la definición de conexión a MySQL

Luego se establece la conexión con la base de datos MySQL. También se declaran las Querys a utilizar, “query1” corresponde a la operación de consulta realizada por la aplicación cliente XML-RPC, la cual devuelve el último registro almacenado en la BD. La Query “query2” corresponde a insertar los datos recibidos desde la aplicación cliente XML-RPC.

```
    my $dbh = DBI->connect($dsn, $userid, $password ) or die
$DBI::errstr;

    my $query1 = "SELECT MAX(Id) FROM $c2";
    my $query2 = "INSERT INTO $c1 VALUES $c2";
```

Figura 56. Segmento de conexión con MySQL y declaración de las Querys.

En el segmento de código mostrado en la Figura 57 se ejecutan las Querys dependiendo del contenido de la variable “c1”, si es “Consulta” ejecuta la Query “query1” para devolver el último registro almacenado en la BD, caso contrario ejecuta la Query “query2” para insertar los registros recibidos en la BD.

```
    my $sth = ($c1 eq "Consulta") ? $dbh->prepare("$query1") : $dbh-
>prepare("$query2");

    my $status = $sth->execute() or die $DBI::errstr;

    my $result = ($c1 eq "Consulta") ? $sth->fetch()->[0] : ($status ==
$c3) ? "OK!" : "Error";

    $sth->finish();
```

Figura 57. Segmento de lógica de operaciones realizada por el servidor.

Para finalizar se envía el resultado de la operación realizada al cliente XML-RPC. Si la operación es de consulta, se envía la etiqueta “Ultimo Registro” seguido del resultado de la ejecución de la “query1”. Caso contrario envía el resultado de la ejecución de la “query2” junto con la etiqueta “Resultado”.

```
    my $etiqueta = ($c1 eq "Consulta") ? "Ultimo Registro" :
"Resultado";

    return {$etiqueta => $result};
}
```

Figura 58. Segmento de código de los resultados enviados al cliente XML-RPC

3.6 Script de sincronización Router MP01 – Raspberry Pi 2

Uno de los aspectos más importantes en sistemas de monitoreo es la fecha y hora en la que se realiza cada lectura o medición. Los router MP01 no poseen reloj de tiempo real y pierden la fecha cada vez que se reinician. Es por ello que se desarrollo un script de sincronización para el Shell de Linux, el cual tiene la función de actualizar la fecha y ejecutar la aplicación cliente XML-RPC para enviar los datos al servidor. El flujograma del script se muestra en la Figura 59.

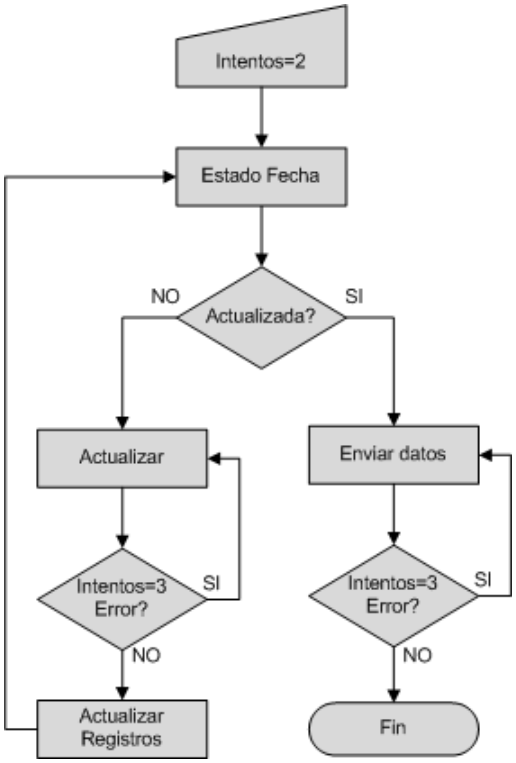

Figura 59. Flujograma del script de sincronización entre Router MP01 y Raspberry Pi 2

Para que el script de sincronización pueda ejecutarse en el router MP01 sin ningún problema, se debe crear la tabla Fecha con los campos mostrados en la Tabla 8.

Fecha
FechaHora (TIMESTAMP)
Estado (TEXT)

Tabla 5. Estructura tabla Fecha

Nos conectamos al router MP01 del medidor de Agronomía vía ssh y creamos la tabla Fecha mediante el comando sqlite3 siguiendo los pasos descritos a continuación.

```
:/# sqlite3 db " CREATE TABLE Fecha(FechaHora TIMESTAMP DATE DEFAULT
(datetime('now','localtime')), Estado TEXT)"
:/# sqlite3 db "INSERT INTO Fecha(Estado) VALUES('Pendiente')"
```

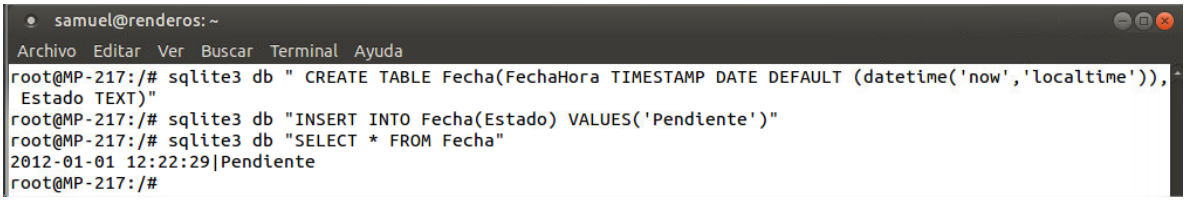

Figura 60. Creación de tabla Fecha en Router MP01

El propósito de la tabla Fecha es llevar un control del estado de la fecha y hora del router.

El script de sincronización inicia definiendo una lista de servidores NTP como se muestra en la Figura 61. Luego consulta el estado de la fecha mediante una sentencia SQLite hacia la tabla Fecha.

```
#!/bin/sh

SERVIDORES="10.130.3.198 10.130.3.199 10.130.3.200"
i="0"

while [ $i -lt 2 ]
do
      j="0"
      if [ "$(sqlite3 db 'SELECT Estado FROM Fecha')" = "Actualizada" ]
      then
```

Figura 61. Segmento de código de definición de servidores NTP

Si la fecha esta actualizada realiza el primero de tres intentos de enviar los datos al servidor, ejecutando la aplicación cliente XML-RPC (xmlpiV2.lua). Si el primer intento devuelve error, el script espera diez segundos para realizar el segundo intento. El primer parámetro de la aplicación cliente XML-RPC es el nombre del medidor, el segundo corresponde a la máxima cantidad de datos a enviar. De pruebas se estableció enviar 30 registros por envió.

```
      if [ "$(sqlite3 db 'SELECT Estado FROM Fecha')" = "Actualizada" ]
      then
            echo "Enviando datos al servidor.."
            while [ $j -lt 3 ]
            do
                  echo "Intento $((j+1)).."
                  ping -c5 10.130.3.150 > /dev/null 2>&1
                  /usr/bin/lua xmlpiV2.lua $1 30 > /dev/null 2>&1
                  if [ $? -eq 0 ]
                  then
                        echo "OK!"
                        break
                  else
```

```
                        j=$((j+1))
                        sleep 10
                fi
        done
        break
```

Figura 62. Segmento de código de envió de datos al servidor

Si la fecha no se encuentra actualizada se intenta actualizar utilizando el comando ntpdate. El bucle for recorre la lista de servidores NTP definidos al inicio del script.

```
        else
        echo "Actualizando Fecha.."
        while [ $j -lt 3 ]
        do
                echo "Intento $((j+1)).."
                for s in $SERVIDORES ; do
                        fechaPrevActualizar=$(date +'%Y-%m-%d %H:%M:%S')

                        /usr/sbin/ntpdate -s -b -u -t "5" "$s"
```

Figura 63. Segmento de código de actualización de fecha y hora

En el escenario donde el router pierde la fecha y no se logra actualizar mediante el servidor NTP, la aplicación de consulta clientetcp-mp continua interrogando sin importar que la fecha no se encuentre actualizada. Cuando el script de sincronización logra actualizar la fecha, todos los registros de la tabla Medidor con la fecha incorrecta son actualizados con la fecha aproximada real de lectura mediante el siguiente algoritmo.

```
        if [ $? -eq 0 ]
                        then
                                deltaFecha=$(($(date +%s) - $(date -d
"$fechaPrevActualizar" +%s)))
                                $(sqlite3 db "UPDATE Medidor SET
Fecha_hora=datetime(strftime('%s', Fecha_hora)+$deltaFecha,'unixepoch')
where Fecha_hora < '2016-01-01 00:00:00'")
                                $(sqlite3 db "UPDATE Fecha SET FechaHora =
datetime('now','localtime'), Estado = 'Actualizada'")
                                echo "OK!"
                                break 2
                        else
                                $(sqlite3 db "UPDATE Fecha SET
Estado='Pendiente'")
                                j=$((j+1))
                        fi
                done
        done
```

Figura 64. Segmento de código de ajuste de fecha y hora

Fecha Aproximada Real de Lectura = Fecha Incorrecta + deltaFecha

Dónde:

deltaFecha = fechaActualizada - fechaPrevActualizar

fechaActualizada: Valor de la fecha un instante después de actualización

fechaPrevActualizar: Valor de la fecha un instante antes de actualización

De esta manera se evita la perdida de datos aun cuando el router MP01 pierde la fecha.

3.7 Pruebas y Resultados

Previo a ejecutar la aplicación cliente XML-RPC consultamos los últimos 5 registros almacenados en la tabla Medidor en el router MP01. En la Figura 65 notamos que se ha presentado el escenario mencionado en la sección anterior. El router MP01 ha perdido la fecha y los registros se han guardado con la fecha incorrecta.

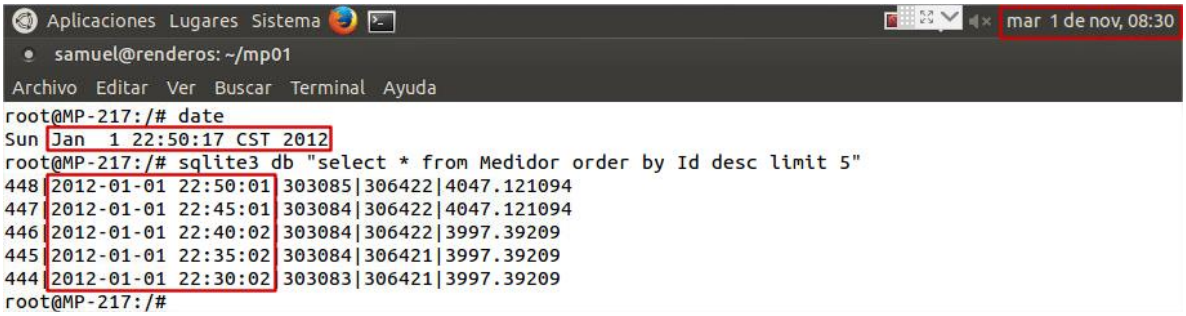

Figura 65. Consulta de la tabla Medidor en el Router MP01

En la Figura 65 observamos que el último registro 448 se guardó con fecha incorrecta de 2012-01-01 22:50:01 y la fecha real aproximada debe ser 2016-11-01 08:30:00. Luego ejecutamos el script de sincronización y el resultado se muestra en la Figura 66.

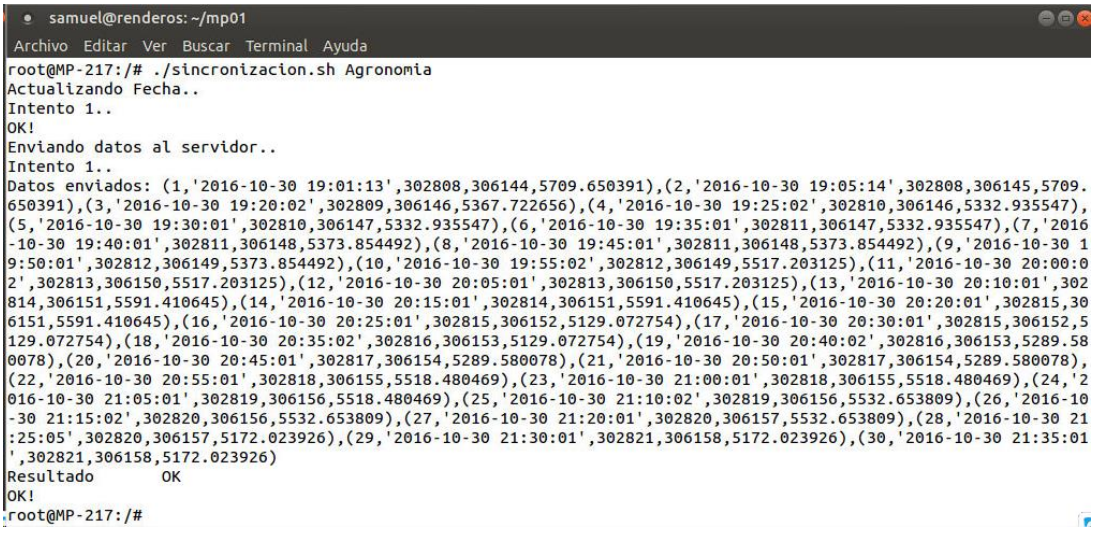

Figura 66. Resultado del script de sincronización

El script de sincronización detecta que la fecha no se encuentra actualizada y realiza el primer intento de actualizarla. Luego de actualizarla realiza el primer intento de envió de datos. Al detectar que el servidor no tiene ningún registro en la tabla Agronomía, la aplicación cliente XML-RPC envía los primeros 30 registros almacenados en la tabla Medidor.

Al actualizar la fecha, el script de sincronización actualizo los registros con la fecha incorrecta mostrados en la Figura 65. En la Figura 67 se muestran los últimos 5 registros actualizados con la fecha real aproximada de cada lectura. De esta manera es como el script de sincronización se encarga de actualizar los registros en el caso que el router se encuentre con la fecha desactualizada.

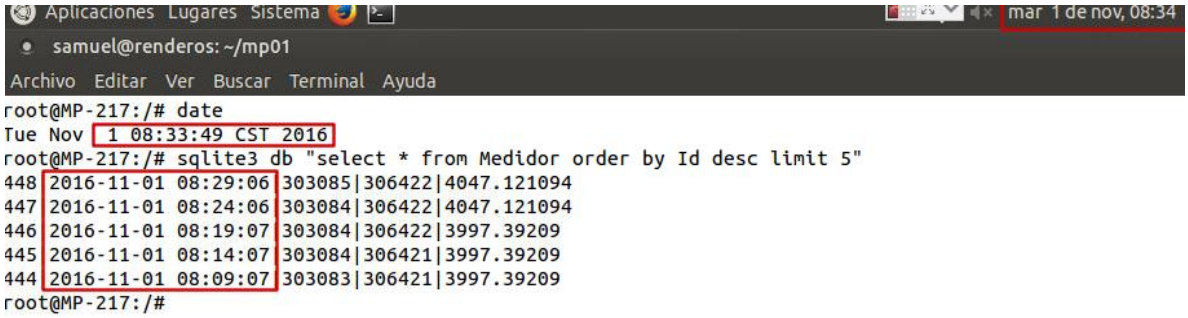

Figura 67. Actualización de registros con fecha real aproximada

En la Figura 68 se muestran los registros almacenados en la tabla Agronomía.

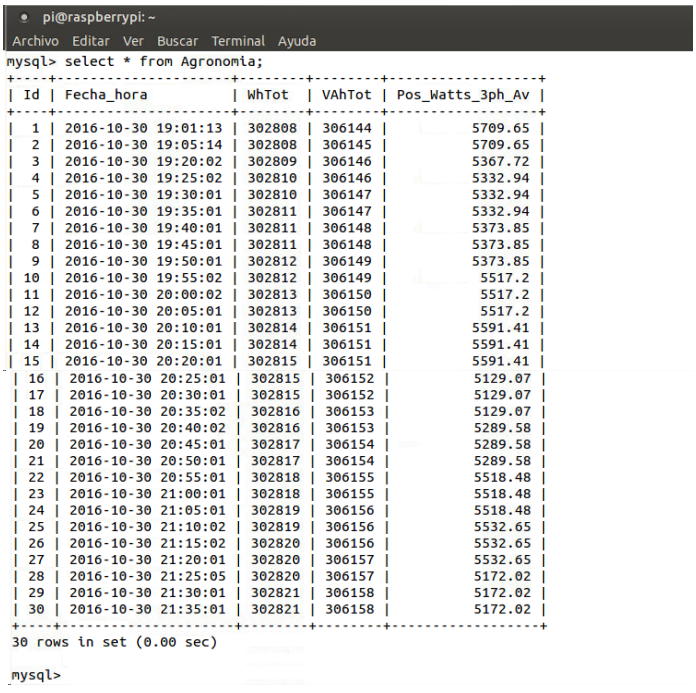

Figura 68. Registros almacenados en el servidor tabla Agronomía

En la Figura 68 se observa el resultado de la aplicación de consulta para el medidor de Agronomía. La aplicación cliente XML-RPC traslada los registros a la aplicación servidor XML-RPC en segmentos de 30 registros como se explicó previamente. De esta forma el router constantemente consultará la base de datos SQLite y cada vez que se tengan nuevos datos, intentara enviarlos al servidor.

A continuación se presentan resultados de la nueva arquitectura de interrogación comparada con la arquitectura actual.

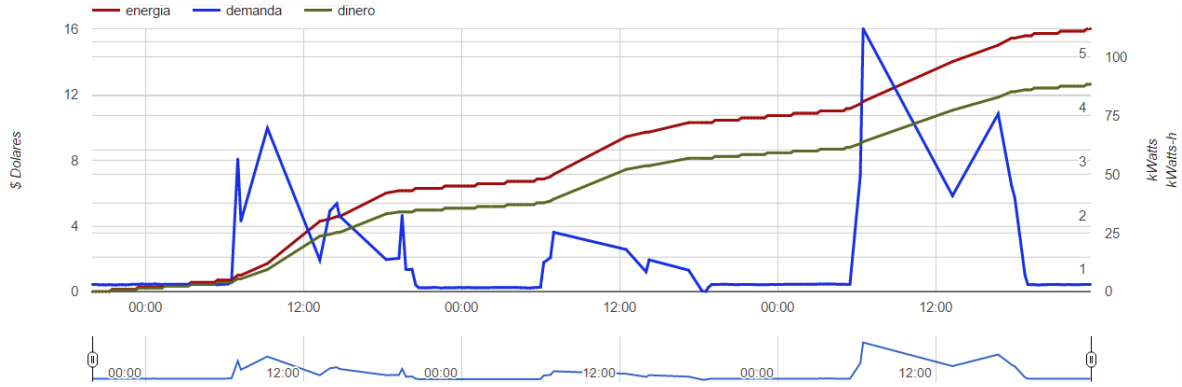

Figura 69. Lecturas del medidor Economia6 – Arquitectura actual de interrogación [23].

La Figura 69 corresponde al grafico de las lecturas realizas por el medidor de Economia6, comprendidas entre las fechas 24-10-2016 20:00 y 27-10-2016 00:00. Como se explico en la sección 1.2.5 el método actual de interrogación presenta perdida de datos debido a la calidad de los enlaces o perdida de la conexión del servidor actual en la Raspberry Pi. El medidor de Economia6 es uno de los más problemáticos en este aspecto. Este problema se intenta resolver con la implementación de la nueva arquitectura de interrogación.

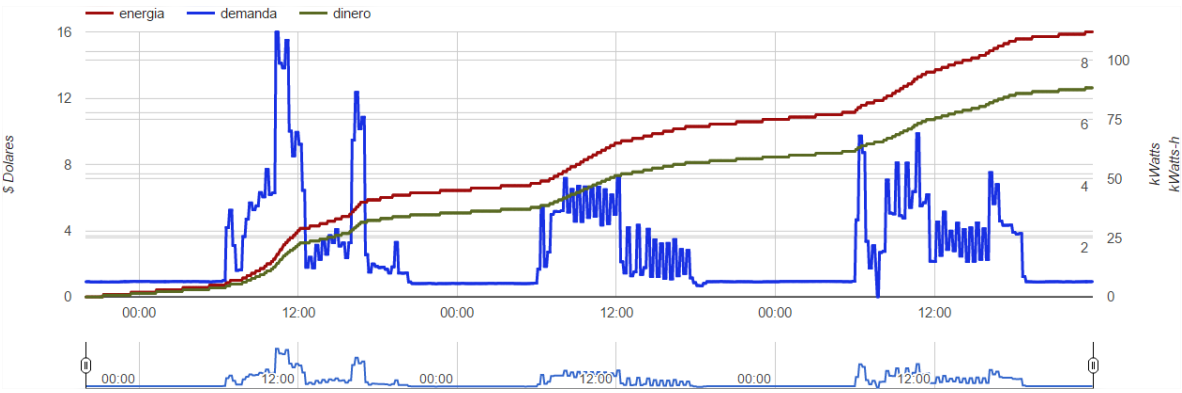

Figura 70. Lecturas del medidor Economia6 – Nueva arquitectura de interrogación [24].

La Figura 70 corresponde al grafico de lecturas realizadas por el medidor de Economia6 en el mismo rango de fechas, pero con la implementación de la nueva arquitectura de interrogación. Del grafico se observa que ha mejorado notablemente la integridad de los datos y se ha reducido la perdida de los mismos.

Humanidades4 es otro de los medidores que presenta el problema de pérdida de datos.

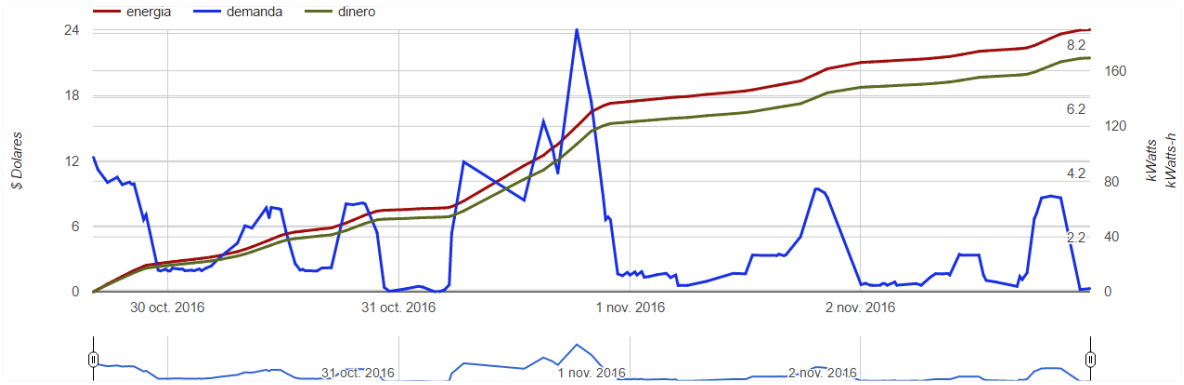

Figura 71. Lecturas del medidor Humanidades4 – Arquitectura actual de interrogación [23].

La Figura 71 corresponde al grafico de las lecturas realizas por el medidor de Humanidades4, comprendidas entre las fechas 29-10-2016 07:00 y 02-11-2016 00:00. En la Figura 72 se muestra el mismo resultado con la implementación de la nueva arquitectura de interrogación mostrando mejoras respecto a la perdida de datos.

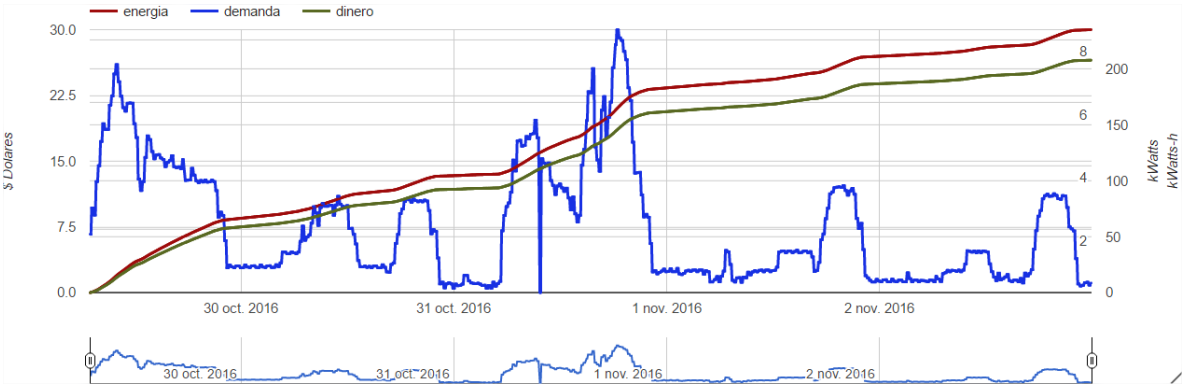

Figura 72. Lecturas del medidor Humanidades4 – Nueva arquitectura de interrogación [24].

También se presentan casos donde el problema de la conexión es más prolongado, como se muestra en la Figura 73, que corresponde a lecturas realizadas por el medidor de AuditoriumMarmol entre las fechas 29-10-2016 07:00 y 02-11-2016 00:00.

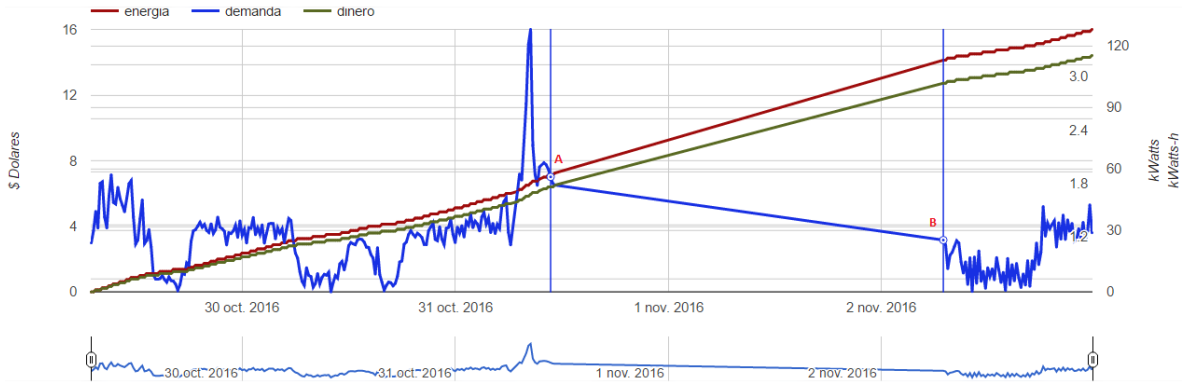

Figura 73. Lecturas medidor AuditoriumMarmol – Arquitectura actual de interrogación [23].

En la Figura 74 se muestra como la nueva arquitectura de interrogación resuelve la perdida de datos para el segmento de lecturas comprendidas entre el punto A y B.

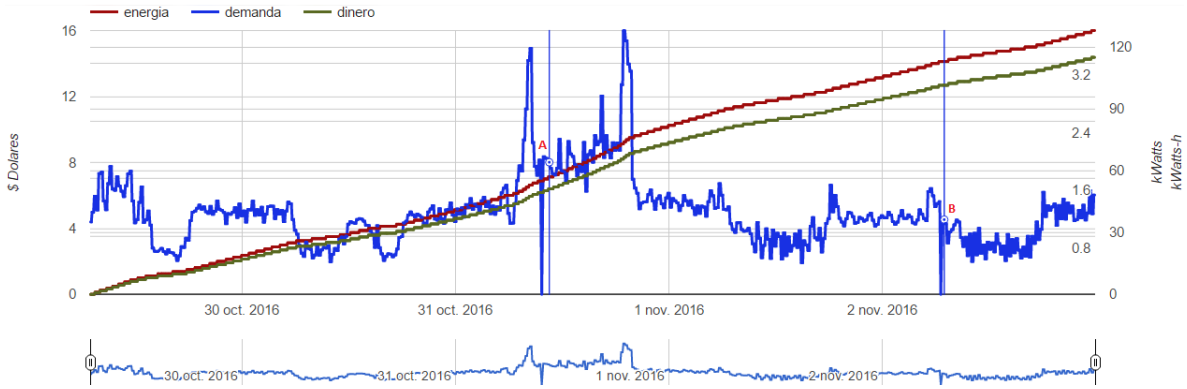

Figura 74. Lecturas medidor AuditoriumMarmol – Nueva arquitectura de interrogación [24].

4. Conclusiones y líneas futuras

4.1 Conclusiones

- ✓ La nueva arquitectura de interrogación, basada en la implementación del protocolo XML-RPC para el monitoreo de la red de medidores de energía eléctrica en el campus central de la Universdad de El Salvador, permite mejorar la integridad de los datos mediante la interrogación continua y almacenamiento *in situ* de cada router de la red.

- ✓ La aplicación de consulta y el sistema de base de datos SQLite en el router MP01 genero excelentes resultados de rendimiento y estabilidad, logrando interrogar cada 5 minutos a cada medidor y almacenando la información en la memoria interna del router.

- ✓ La técnica de compilación cruzada empleada resulto exitosa, con la que se pueden desarrollar aplicaciones en C y ser ejecutadas en dispositivos embebidos como el router MP01. Técnica en la que se aprovecha la facilidad que brindan los desarrolladores de OpenWRT al proporcionar las herramientas de compilación cruzada (toolchain).

- ✓ Las aplicaciones de cliente XML-RPC programada LUA y servidor XML-RPC programada PERL combinadas con el script de sincronización y actualización de fecha, generaron los resultados esperados, consultando la base de datos SQLite en el router MP01 y enviando un máximo de 30 registros al servidor en cada intento de actualización.

4.2 Líneas Futuras

- ✓ Optimizar los script de actualización para que se permita enviar, desde el router MP01 hacia el medidor, más de 30 registros y de esta manera reducir el tiempo de actualización del servidor en la Raspberry Pi.

- ✓ Implementar un sistema de actualización de fecha y hora para la red de medidores y no depender solamente del servidor NTP configurado en la Raspberry Pi.

- ✓ Incorporar más nodos y medidores de energía para las subestaciones no incluidas en la red. Medidores que cuenten con el protocolo de comunicación Modbus/TCP podrían incorporarse al sistema con facilidad.

Anexo A. Configuración de router MP01

A continuación se presenta una guía de configuración de los router MP01 para la implementación de la nueva arquitectura de interrogación.

En la etapa de pruebas del presente trabajo de graduación se presentaron problemas con la instalación de nuevos paquetes dentro del router MP01. Esto debido a que la partición */dev/root,* donde por defecto se instalan las nuevas aplicaciones, se encuentra con el 100% de uso como se observa en la Figura A1.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
root@MP-20:/# df -h
Filesystem                Size      Used Available Use% Mounted on
/dev/root                 3.0M      3.0M         0 100% /rom
tmpfs                     6.8M     52.0K      6.7M   1% /tmp
tmpfs                   512.0K         0    512.0K   0% /dev
/dev/mtdblock3            3.9M    292.0K      3.7M   7% /jffs
mini_fo:/jffs             3.0M      3.0M         0 100% /
root@MP-20:/#
```

Figura A1. Detalle del espacio de memoria en el router MP01.

Para solucionar este inconveniente se modificó el archivo de configuración del gestor de paquetes de OpenWrt, OPKG. Se agregó la opción de permitir instalar aplicaciones en la partición */jffs* como se muestra en la figura A2.

```
src/gz snapshots
http://downloads.openwrt.org/kamikaze/8.09.2/atheros/packages
dest root /
dest ram /tmp
lists_dir ext /var/opkg-lists
option overlay_root /jffs
```

Figura A2. Contenido del archivo opkg.conf

El primer paso de la configuración consiste en enviar los archivos de configuración y aplicaciones hacia el router MP01. La aplicación de consulta enviada dependerá del tipo de medidor, para este caso es un tipo SHARK 100S, por lo tanto se envía la aplicación *clientetcp-mp100*.

```
:~/ConfiguracionMPV2$ scp opkg.conf root@10.130.3.217:/etc
:~/ConfiguracionMPV2$ scp libmodbus.so.5 root@10.130.3.217:/usr/lib
:~/ConfiguracionMPV2$ scp ntpdate root@10.130.3.217:/etc/init.d
:~/ConfiguracionMPV2$ scp clientetcp-mp100 root@10.130.3.217:/
:~/ConfiguracionMPV2$ scp xmlpiV2.lua root@10.130.3.217:/
:~/ConfiguracionMPV2$ scp sincronizacion.sh root@10.130.3.217:/
```

Figura A3. Envió de archivos de configuración y aplicaciones al router MP01

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2$ sudo ifconfig eth3:0 10.130.3.240/24 up
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2$ scp opkg.conf root@10.130.3.217:/etc
root@10.130.3.217's password:
opkg.conf                                     100%  161     0.2KB/s   00:00
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2$ scp libmodbus.so.5 root@10.130.3.217:/usr/lib
root@10.130.3.217's password:
libmodbus.so.5                                100%  105KB 104.7KB/s   00:00
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2$ scp ntpdate root@10.130.3.217:/etc/init.d
root@10.130.3.217's password:
ntpdate                                       100%  751     0.7KB/s   00:00
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2$ scp clientetcp-mp100 root@10.130.3.217:/
root@10.130.3.217's password:
clientetcp-mp100                              100% 1019KB  67.9KB/s   00:15
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2$ scp xmlpiV2.lua root@10.130.3.217:/
root@10.130.3.217's password:
xmlpiV2.lua                                   100% 1367     1.3KB/s   00:00
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2$ scp sincronizacion.sh root@10.130.3.217:/
root@10.130.3.217's password:
sincronizacion.sh                             100% 1242     1.2KB/s   00:00
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2$
```

Figura A4. Resultado del envió de los archivos de configuración y aplicaciones.

Luego se envían archivos de configuración personalizados de cada router MP01.

```
:~/ConfiguracionMPV2/Agronomia$ scp network root@10.130.3.217:/etc/config
:~/ConfiguracionMPV2/Agronomia$ scp system root@10.130.3.217:/etc/config
:~/ConfiguracionMPV2/Agronomia$ scp tareas root@10.130.3.217:/
```

Figura A5. Envió de los archivos de configuración del sistema y crontab.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2$ cd Agronomia
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2/Agronomia$ scp network root@10.130.3.217:/etc/config
root@10.130.3.217's password:
network                                       100%  749     0.7KB/s   00:00
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2/Agronomia$ scp system root@10.130.3.217:/etc/config
root@10.130.3.217's password:
system                                        100%  362     0.4KB/s   00:00
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2/Agronomia$ scp tareas root@10.130.3.217:/
root@10.130.3.217's password:
tareas                                        100%  108     0.1KB/s   00:00
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2/Agronomia$
```

Figura A6. Resultado del envió de los archivos del sistema y crontab.

El archivo *network* contiene la configuración de red del router. Es importante mencionar que con la actualización del firmware SENC1.1, se perdió la comunicación lógica entre los router MP01 y los medidores. Por lo que fue necesario configurar una *interface alias* para volver a tener comunicación entre ambos dispositivos.

```
config 'alias'
      option 'interface' 'lan'
      option 'proto' 'static'
      option 'ipaddr' '10.30.217.1'
      option 'netmask' '255.255.255.0'
```

Figura A7. Segmento de configuración del archivo network.

En el archivo *system* se configura el formato de la zona horaria y el nombre del dispositivo.

```
config 'system'
      option 'timezone' 'CST6'
      option 'hostname' 'MP-217'
```

Figura A8. Segmento de configuración del archivo system.

El archivo tareas se utiliza para programar el crontab con las siguientes operaciones. Cada 5 minutos se ejecuta la aplicación de consulta y cada 10 minutos se realiza el intento de enviar los datos al servidor.

```
*/5 * * * * cd / && ./clientetcp-mp100 10.30.217.2
*/10 * * * * cd / && /bin/sh sincronizacion.sh Agronomia
```

Figura A9. Contenido del archivo tareas.

Luego se envían las aplicaciones necesarias para implementar XML-RPC y SQLite dentro del router. También se envía el script de instalación de las aplicaciones a ejecutarse el router MP01.

```
scp opkg_4564-3_mips.ipk liblua_5.1.4-2_mips.ipk lua_5.1.4-2_mips.ipk
liblua-socket_2.0.2-1_mips.ipk libexpat_1.95.8-1_mips.ipk liblua-
expat_1.0.2-2_mips.ipk liblua-xmlrpc_1.0x-1_mips.ipk libsqlite3_3.4.2-
1_mips.ipk libreadline_5.2-1_mips.ipk sqlite3-cli_3.5.9-1_mips.ipk
script_config.sh root@10.130.3.217:/tmp
```

Figura A10. Envió de las aplicaciones requeridas para correr XML-RPC y SQLite.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2$ scp opkg_4564-3_mips.ipk liblua_5.1.4-2_mips.ip
k lua_5.1.4-2_mips.ipk liblua-socket_2.0.2-1_mips.ipk libexpat_1.95.8-1_mips.ipk liblua-e
xpat_1.0.2-2_mips.ipk liblua-xmlrpc_1.0x-1_mips.ipk libsqlite3_3.4.2-1_mips.ipk libreadli
ne_5.2-1_mips.ipk sqlite3-cli_3.5.9-1_mips.ipk script_config.sh root@10.130.3.217:/tmp
root@10.130.3.217's password:
opkg_4564-3_mips.ipk                                  100%   65KB  65.3KB/s   00:00
liblua_5.1.4-2_mips.ipk                               100%   74KB  74.2KB/s   00:00
lua_5.1.4-2_mips.ipk                                  100% 5411     5.3KB/s   00:00
liblua-socket_2.0.2-1_mips.ipk                        100%   40KB  39.5KB/s   00:00
libexpat_1.95.8-1_mips.ipk                            100%   49KB  49.5KB/s   00:00
liblua-expat_1.0.2-2_mips.ipk                         100% 7041     6.9KB/s   00:00
liblua-xmlrpc_1.0x-1_mips.ipk                         100% 5977     5.8KB/s   00:00
libsqlite3_3.4.2-1_mips.ipk                           100%  176KB 175.8KB/s   00:00
libreadline_5.2-1_mips.ipk                            100%  107KB 107.1KB/s   00:00
sqlite3-cli_3.5.9-1_mips.ipk                          100%   14KB  14.0KB/s   00:00
script_config.sh                                      100%  896     0.9KB/s   00:00
ubuntu@ubuntu:~/TBE115/ConfiguracionMPV2$
```

Figura A11. Resultado del envió de las aplicaciones requeridas para XML-RPC y SQLite.

Una vez finalizado el envió de todos los archivos y aplicaciones necesarias, se ingresa al router MP01 por medio de SSH y se ejecuta el script de instalación y configuración.

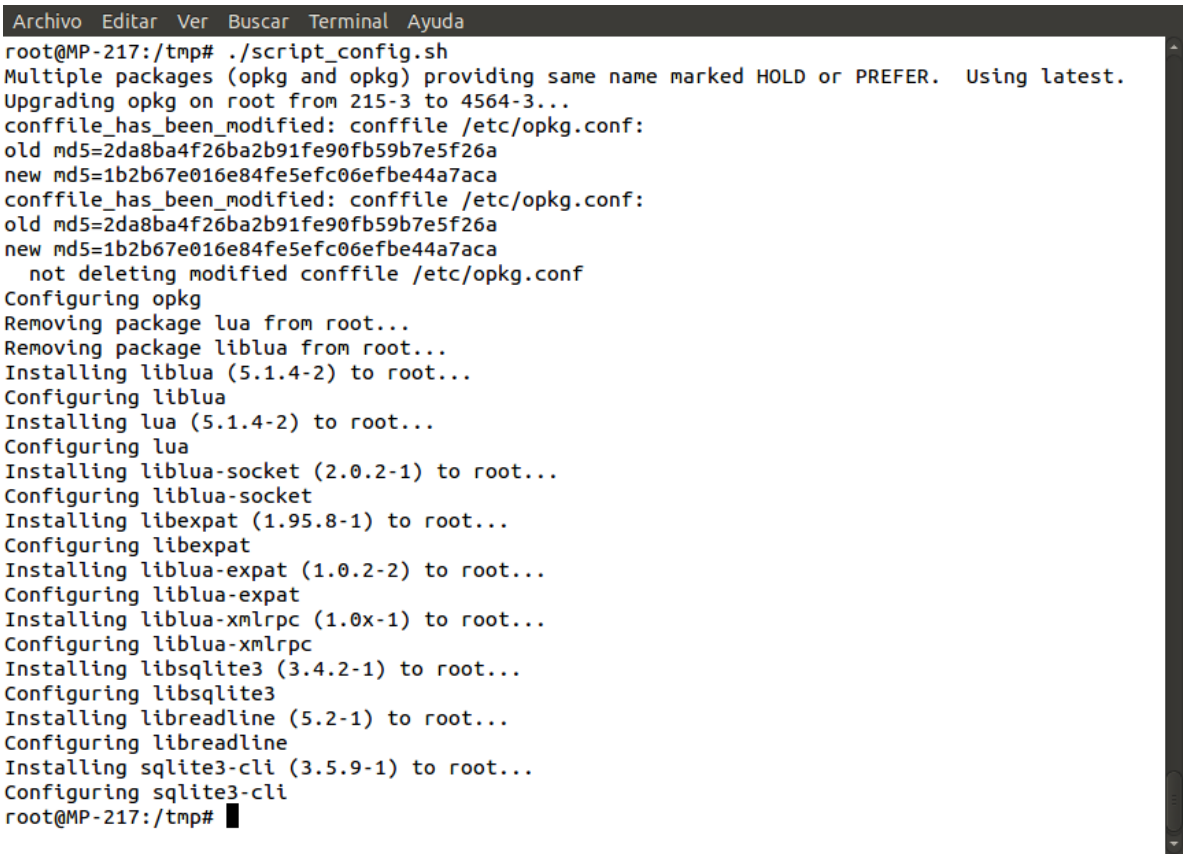

Figura A12. Ejecución y resultado del script de instalación y configuración.

El script de instalación y configuración instala las aplicaciones necesarias para correr el protocolo XML-RPC y SQLite dentro del router. También se encarga de crear y configurar las tablas Fecha y Medidor por medio de la aplicación *sqlite3*.

```
#/bin/bash
opkg install opkg_4564-3_mips.ipk -force-defaults
opkg remove lua liblua libexpat libsqlite3 libreadline -force-removal-
of-dependent-packages
opkg install liblua_5.1.4-2_mips.ipk
opkg install lua_5.1.4-2_mips.ipk
opkg install liblua-socket_2.0.2-1_mips.ipk
opkg install libexpat_1.95.8-1_mips.ipk
opkg install liblua-expat_1.0.2-2_mips.ipk
opkg install liblua-xmlrpc_1.0x-1_mips.ipk
opkg install libsqlite3_3.4.2-1_mips.ipk
opkg install libreadline_5.2-1_mips.ipk
opkg install sqlite3-cli 3.5.9-1 mips.ipk
cd /
sqlite3 db "CREATE TABLE Fecha(FechaHora TIMESTAMP DATE DEFAULT
(datetime('now','localtime')), Estado TEXT)"
sqlite3 db "INSERT INTO Fecha(Estado) VALUES('Pendiente')"
sqlite3 db "CREATE TABLE Medidor (Id INTEGER PRIMARY KEY AUTOINCREMENT,
Fecha_hora TIMESTAMP DATE DEFAULT (datetime('now','localtime')), WhTot
INTEGER (11), VAhTot INTEGER (11), Pos_Watts_3ph_Av FLOAT)"
```

Figura A13. Contenido del script de instalación y configuración.

Para finalizar se programan las tareas por medio de la aplicación Crontab. Se habilita para que al reiniciar el router, cargue la configuración programada.

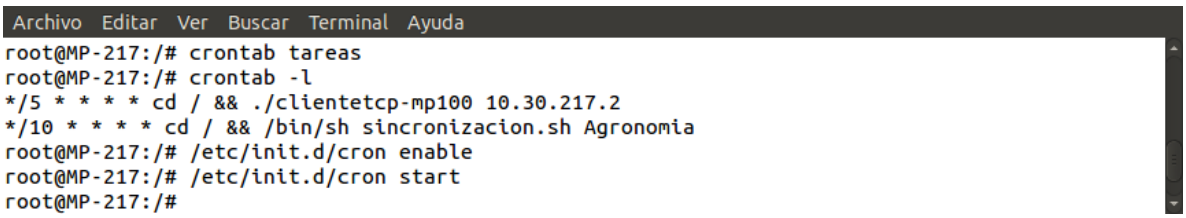

Figura A14. Configuración de ejecución de tareas programadas.

Anexo B. Compilación cruzada para router Dragino

B.1 Compilación de aplicaciones C para router Dragino

Arquitectura MIPS

Con el nombre de MIPS (siglas de Microprocessor without Interlocked Pipeline Stages) se conoce a toda una familia de microprocesadores de arquitectura RISC desarrollados por MIPS Technologies.

Los diseños del MIPS son utilizados en la línea de productos informáticos de SGI; en muchos sistemas embebidos; en dispositivos para Windows CE; routers Cisco; y videoconsolas como la Nintendo 64 o las Sony PlayStation, PlayStation 2 y PlayStation Portable. Más recientemente, la NASA usó uno de ellos en la sonda New Horizons. Las primeras arquitecturas MIPS fueron fsfs en 32 bits (generalmente rutas de datos y registros de 32 bits de ancho), si bien versiones posteriores fueron implementadas en 64 bits. Existen cinco revisiones compatibles hacia atrás del conjunto de instrucciones del MIPS, llamadas MIPS I, MIPS II, MIPS III, MIPS IV y MIPS 32/64.

OpenWrt

OpenWrt es un firmware basado en una distribución de Linux empotrada en dispositivos tales como routers personales. El desarrollo de OpenWrt fue impulsado inicialmente gracias a la licencia GPL, que obligaba a todos aquellos fabricantes que modificaban y mejoraban el código, a liberar éste y contribuir cada vez más al proyecto en general. Poco a poco el software ha ido creciendo y se encuentran características implementadas que no tienen muchos otros fabricantes de dispositivos comerciales para el sector no profesional, tales como QoS, VPN y otras características que dotan a OpenWrt de un dispositivo realmente potente y versátil, apto para utilizar los hardware donde corre OpenWrt no sólo para utilizarlos como routers, sino como servidores de archivo, nodos P2P, servidores de WEBcams, firewall o puertas de acceso VPN.

Compilación cruzada

La compilación de código fuente que, realizada bajo una determinada arquitectura genera código ejecutable para una arquitectura diferente se denomina compilación cruzada. Para realizar este tipo de compilación es necesario contar con una serie de programas y librerías que establezcan un ambiente propicio para llevar a cabo esta tarea. Este ambiente se denomina entorno de compilación cruzada.

En el entorno de compilación se definen dos tipos de sistemas, el **Huesped**, es en donde se realiza la compilación del sistema, y el **Objetivo** donde se ejecuta el código.

Sistema Huesped

Como se ha descrito en la definición de compilación cruzada, la implementación de un entorno de compilación nos brinda la posibilidad de aprovechar los recursos que disponemos en una computadora tipo PC: Procesador Intel Atom N455 de 1,66 Ghz, Disco duro de 320 GB, 1024 MB DDR3 SDRAM, Acelerador Intel 3150, LAN inalámbrica, Ubuntu 14.04.3 LTS.

Sistema Objetivo

El dispositivo embebido es un Router Dragino MS12. Este equipo se basa en un sistema Atheros SoC (System on chip). A diferencia de la mayoría de router domesticos, también tiene soporte para la conexión de cualquier tipo de sensor, un puerto serie UART, GPIOs que se pueden utilizar como un puerto SPI y viene con firmware OpenWRT preinstalado: Arquitectura MIPS 4K, Bootloader RedBoot, System-On-Chip Atheros AR2317, Velocidad CPU 180 Mhz, Memoria Flash 8 MiB, RAM 16 MiB / 32 MB v1.0, Ethernet 1xRJ45, Serial UART, SPI simulado por GPIO, OpenWRT Backfire 10.03.1.

Comunicación entre sistema huésped y objetivo

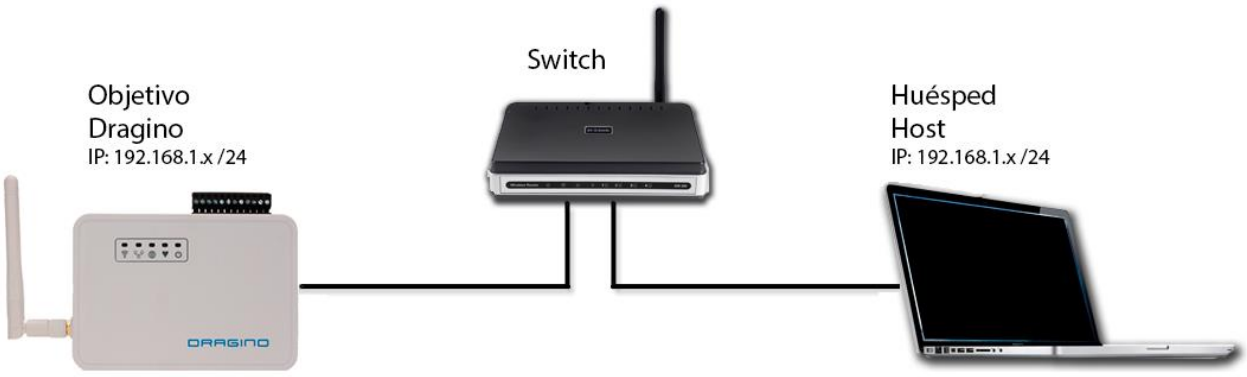

Figura B1. Esquema de comunicación entre sistema huésped y objetivo.

Las configuraciones del router Dragino se realizan mediante una consola terminal en la Mini Laptop, utilizando el protocolo ssh. De igual manera la transferencia de archivos, se realiza utilizando el protocolo scp a través de ssh.

Para esta guía el Router Dragino utiliza la IP 192.168.1.236 y el Host 192.168.1.9, ambos sistemas con mascarad de Red 255.255.255.0.

Accedemos al router con la siguiente línea de comando:

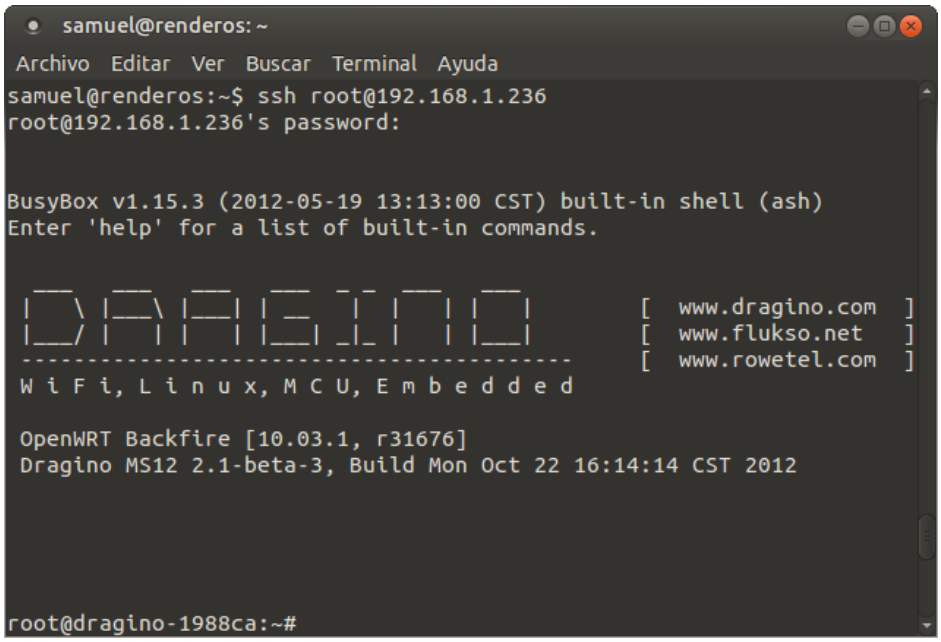

Figura B2. Acceso al router Dragino mediante SSH.

De la consola de bienvenida del Router Dragino se observa:
Firmware: OpenWRT Backfire 10.03.1
Imagen: Dragino MS12 2.1-beta-3

Entorno de compilación cruzada en sistema huésped

Los componentes necesarios para implementar un entorno de compilación cruzada consisten básicamente en tres elementos: Compilador C, Librería C, Binutils.

Cada una de estas herramientas funciona en dependencia de las otras, es por esto que configurar y compilarlas es una operación delicada. La complejidad radica en que no existe una compatibilidad asegurada entre las diferentes versiones de los componentes.

Hay diferentes maneras de implementar compilación cruzada, en esta guía se realiza mediante la cadena de herramientas que proporcionan los desarrolladores de OpenWRT. La cadena de herramientas o toolchain es un término que agrupa a una serie de proyectos que contienen las herramientas de programación producidas por el proyecto GNU. En el sitio oficial de OpenWRT se proporciona el toolchain para cada una de las versiónes del firmware.

Se descarga el toolchain para la versión Backfire 10.03.1 y se renombran los directorios con las siguientes líneas:

```
:~$  mkdir draginocc
:~$  cd draginocc
:~/draginocc$  wget
https://downloads.openwrt.org/backfire/10.03.1/atheros/OpenWrt-
Toolchain-atheros-for-mips-gcc-4.3.3%2bcs_uClibc-0.9.30.1.tar.bz2
:~/draginocc$  tar -xjvf OpenWrt-Toolchain-atheros-for-mips-gcc-
4.3.3%2bcs_uClibc-0.9.30.1.tar.bz2
:~/draginocc$  mv OpenWrt-Toolchain-atheros-for-mips-gcc-
4.3.3%2bcs_uClibc-0.9.30.1  openwrt
:~/draginocc$  cd openwrt
:~/draginocc/openwrt$  mv toolchain-mips_gcc-4.3.3%2bcs_uClibc-0.9.30.1
toolchain
```

Figura B3. Descarga de toolchain y renombre de directorios.

Luego de lo anterior debería de quedar un orden y nombres de directorios como se muestra en la Figura B4.

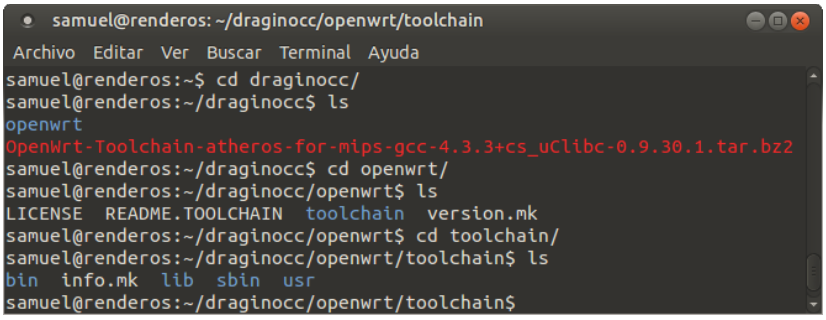

Figura B4. Resultado de la descarga del toolchain y renombre de directorios.

Estas operaciones también se pueden realizar desde el entorno grafico de Ubuntu, accediendo al sitio web https://downloads.openwrt.org/ y descargando el toolchain correspondiente a la versión Backfire 10.03.1 sistema Atheros.

En la ubicación *usr/bin/* del directorio *toolchain* se encuentran los scripts ejecutables:

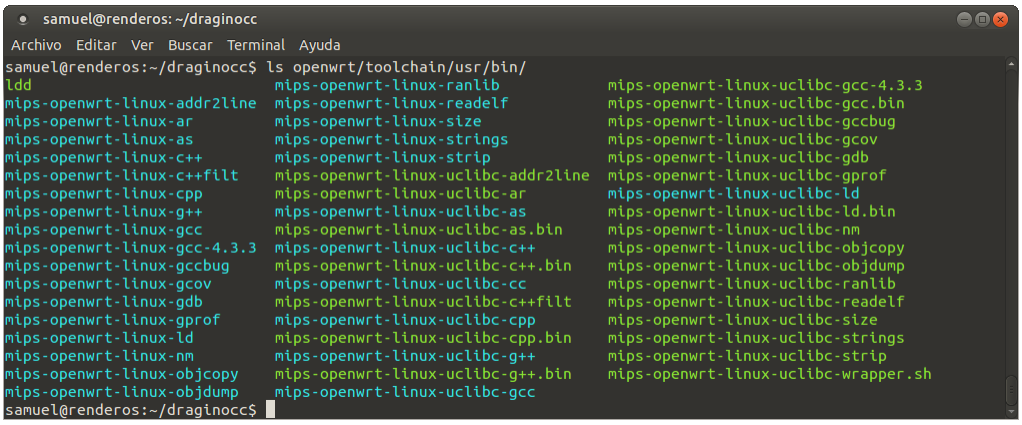

Figura B5. Scripts ejecutables del toolchain.

En un editor de texto se crea un archivo con código de lenguaje C, por ejemplo *holamundo.c* y se realiza la compilación con el script *mips-openwrt-linux-uclibc-gcc.*

```
:~/draginocc$  vi holamundo.c
#include <stdio.h>

int main() {
     printf("\nHola mundo – Compilacion cruzada\n\n");
     return 0;
}
:~/draginocc$  openwrt/toolchain/usr/bin/mips-openwrt-linux-uclibc-gcc
holamundo.c –o holamundo
:~/draginocc$  file holamundo
```

Figura B6. Programa de ejemplo en C.

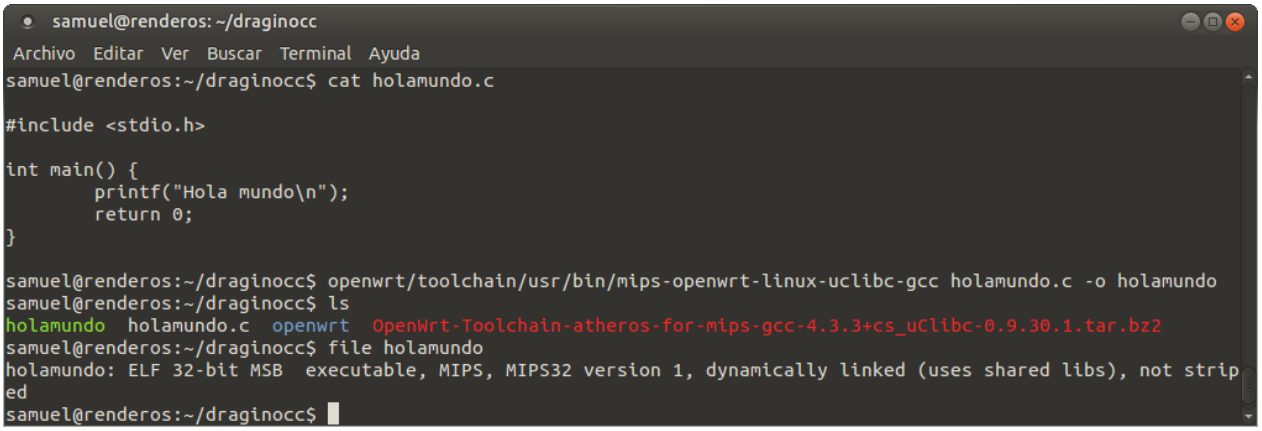

Figura B7. Compilación de programa de ejemplo en C.

Con el comando *file* se observan los detalles del ejecutable creado, como se muestra en la imagen. Luego lo enviamos al Router Dragino por SCP con las siguientes líneas:

```
:~/draginocc$  scp holamundo root@192.168.1.236:/tmp
root@192.168.1.236's password: root
```

Accedemos al Router por medio de SSH, nos ubicamos en el directorio */tpm/* y ejecutamos la aplicación *holamundo.*

Figura B8. Ejecución del programa de ejemplo en router Dragino.

B.2 Compilación de libmodbus para router Dragino

Libmodbus

Libmodbus es una librería para enviar y recibir datos con un dispositivo que implemente el protocolo Modbus. Esta librería contiene lógicas de programación para comunicarse a través de diferentes medios (Modo RTU para serial y TCP/IP para Ethernet). Libmodbus proporciona una abstraccion de las capas mas bajas de comunicación y ofrece la misma API en todas las plataformas soportadas.

El protocolo modbus contiene dos variantes (Serial RTU - Ethernet TCP), para facilitar la implementación de una variante, la librería fue diseñada para usar una lógica de programación para cada variante. Estas lógicas de programación son ademas un conveniente camino para cumplir otros requerimientos (Operaciones en tiempo real).

Utilizando el mismo esquema de comunicación, procedemos a configurar el entorno de compilación en el sistema huésped. A continuación se detallan las dos formas de configurar la librería libmodbus, para ejucutar aplicaciones en router Dragino y en el mismo sistema huésped:

Preparando entorno para realizar y ejecutar aplicaciones libmodbus en Router Dragino:

Se descarga la última versión estable desde el sitio oficial de la librería:

```
:~/draginocc$ wgethttp://libmodbus.org/releases/libmodbus-3.1.2.tar.gz
:~/draginocc$ tar -xzvf libmodbus-3.1.2.tar.gz
```

Figura B9. Descarga de la librería libmodbus.

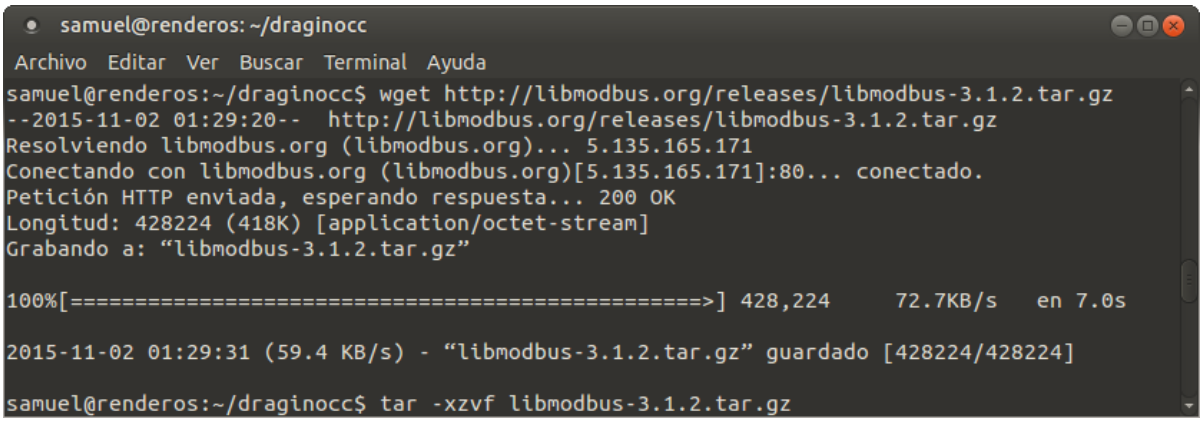

Figura B10. Descarga y extracción de la librería libmodbus.

Luego se exportan las siguientes variables de entorno, teniendo en cuenta la ruta del directorio toolchain descargado previamente (coloco de ejemplo los directorios de mi computadora).

```
:~/draginocc$  export
TOOLCHAIN_PATH=/home/samuel/draginogcc/openwrt/toolchain
:~/draginocc$  export PATH=$PATH:$TOOLCHAIN_PATH/usr/bin/
:~/draginocc$  export AR=$TOOLCHAIN_PATH/usr/bin/mips-openwrt-linux-
uclibc-ar
:~/draginocc$  export AS=$TOOLCHAIN_PATH/usr/bin/mips-openwrt-linux-
uclibc-as
:~/draginocc$  export LD=$TOOLCHAIN_PATH/usr/bin/mips-openwrt-linux-
uclibc-ld
:~/draginocc$  export NM=$TOOLCHAIN_PATH/usr/bin/mips-openwrt-linux-
uclibc-nm
:~/draginocc$  export CC=$TOOLCHAIN_PATH/usr/bin/mips-openwrt-linux-
uclibc-gcc
:~/draginocc$  export CPP=$TOOLCHAIN_PATH/usr/bin/mips-openwrt-linux-
uclibc-cpp
:~/draginocc$  export GCC=$TOOLCHAIN_PATH/usr/bin/mips-openwrt-linux-
uclibc-gcc
:~/draginocc$  export CXX=$TOOLCHAIN_PATH/usr/bin/mips-openwrt-linux-
uclibc-g++
:~/draginocc$  export RANLIB=$TOOLCHAIN_PATH/usr/bin/mips-openwrt-linux-
uclibc-ranlib
```

Figura B11. Variables de entorno para enlazar los script del toolchain.

Ahora si, se generan los scripts necesarios de la librería y enlazamos el compilador del toolchain.

```
:~/draginocc$ cd libmodbus-3.1.2
:~/draginocc/libmodbus-3.1.2$ ./configure --host=mips-openwrt-linux-
uclibc
```

Figura B12. Ejecución del script de configuración de la librería.

Luego se edita el archivo libmodbus.pc, de tal manera que el compilador enlace las librerías estáticas y dinámicas desde el mismo directorio donde se generaron los scripts:

Figura B13. Edición del archivo libmodbus.pc.

Se procede a compilar la librería completa con la herramienta make. En el caso de no tener la herramienta make, la instalamos mediante el comando sudo apt-get install make.

```
:~/draginocc/libmodbus-3.1.2$ make
```

Figura B14. Compilación de la librería libmodbus.

Preparando entorno para realizar y ejecutar aplicaciones en el sistema huésped.

De igual manera, descargamos la última versión estable desde el sitio oficial de la librería. Abrimos una nueva ventana de consola (necesario para borrar variables de entorno) y la descarga la realizamos en un nuevo directorio como se especifica a continuación:

```
:~/draginocc$ wget http://libmodbus.org/releases/libmodbus-3.1.2.tar.gz
-P libmodbusx86
:~/draginocc$ cd libmodbusx86/
:~/draginocc/libmodbusx86$ tar -xzvf libmodbus-3.1.2.tar.gz
```

Figura B15. Descarga de libmodbus para sistema huésped.

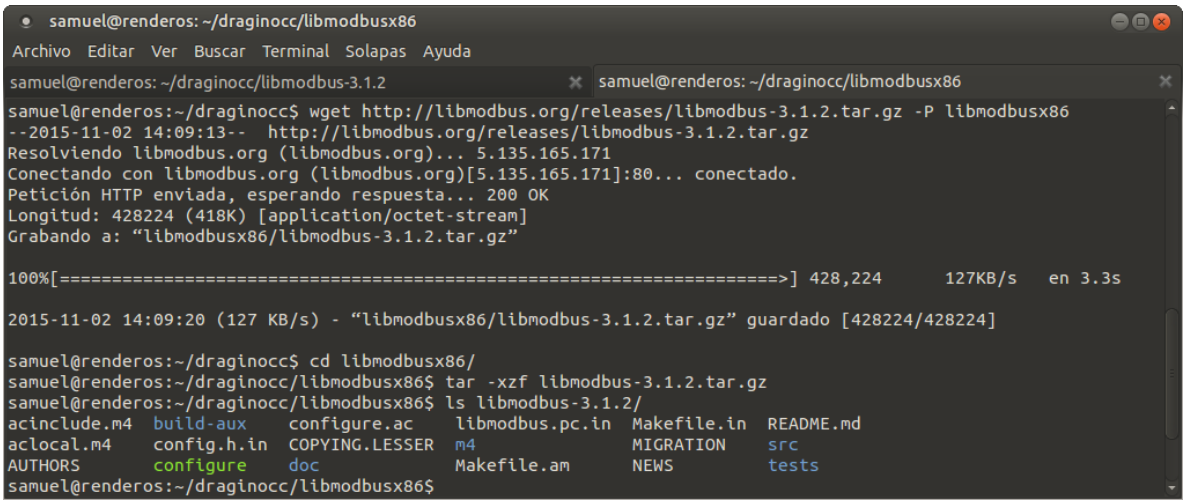

Figura B16. Resultado de la descarga y extracción de la librería libmodbus.

Luego se generan los scripts necesarios de la librería y enlazamos el compilador del sistema huesped:

```
:~/draginocc$ cd libmodbus-3.1.2
:~/draginocc/libmodbusx86/libmodbus-3.1.2$ ./configure
```

Figura B18. Generación de scrips de la librería libmodbus.

A diferencia de la configuración para el sistema objetivo, el compilador enlazado es el del sistema huésped.

Se edita el archivo libmodbus.pc, de tal manera que el compilador enlace las librerías estáticas y dinámicas desde el mismo directorio donde se generaron los scripts:

Se compila la librería completa con la herramienta make:

```
:~/draginocc/libmodbusx86/libmodbus-3.1.2$ make
```

Figura B19. Compilación de la librería libmodbus.

Se ha finalizado de configurar el entorno para compilar aplicaciones libmodbus para un sistema huésped. A continuación se describe el procedimiento para compilar programas libmodbus. La compilación tanto para un sistema huésped como para un sistema objetivo, se realiza mediante la herramienta pkg-config, la cual provee una interfaz unificada para llamar librerias cuando se está compilando un programa a partir del código fuente.

Antes de ejecutar aplicaciones libmodbus en un sistema objetivo, es necesario copiar una de las librerías dinámicas (libmodbus.so.5) al directorio de librerías principal. Para esta guía se utiliza el directorio /usr/lib del Router Dragino:

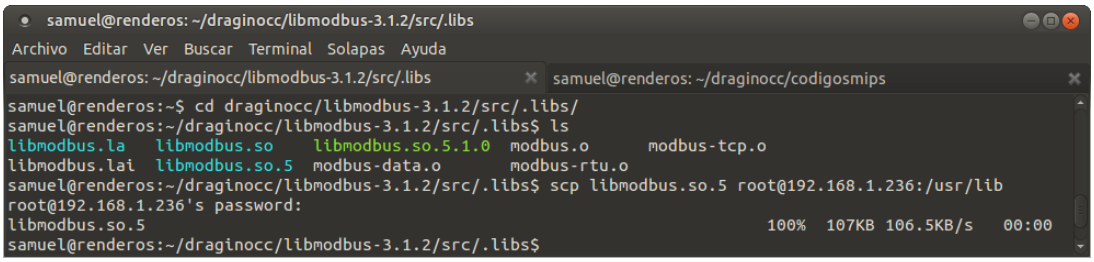

Figura B20. Envió de la librería dinámica de libmodbus hacia el router Dragino.

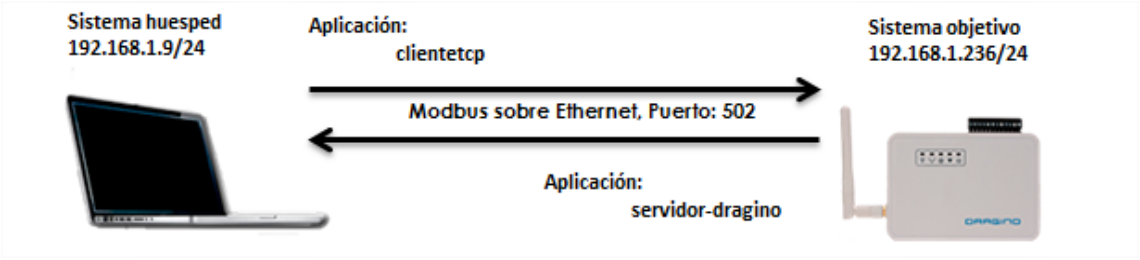

Figura B21. Esquema de comunicación empleado.

Programa de ejemplo

Compilación del programa de ejemplo servidor-dragino.c (sistema objetivo):

```
samuel@renderos:~/draginocc$ cd codigosmips/
samuel@renderos:~/draginocc/codigosmips$ export PKG_CONFIG_PATH=/home/samuel/draginocc/libmodbus-3.1.2
samuel@renderos:~/draginocc/codigosmips$ /home/samuel/draginocc/openwrt/toolchain/usr/bin/mips-openwrt-linux-u
libc-gcc servidor-dragino.c -o servidor-dragino `pkg-config --libs --cflags libmodbus`
samuel@renderos:~/draginocc/codigosmips$ file servidor-dragino
servidor-dragino: ELF 32-bit MSB  executable, MIPS, MIPS32 version 1, dynamically linked (uses shared libs), n
t stripped
samuel@renderos:~/draginocc/codigosmips$ scp servidor-dragino root@192.168.1.236:/tmp
root@192.168.1.236's password:
servidor-dragino                                      100%   10KB   9.9KB/s   00:00
samuel@renderos:~/draginocc/codigosmips$
```

Figura B22. Compilación del programa de ejemplo servidor-dragino.

Compilación del programa de ejemplo clientetcp.c (sistema huesped):

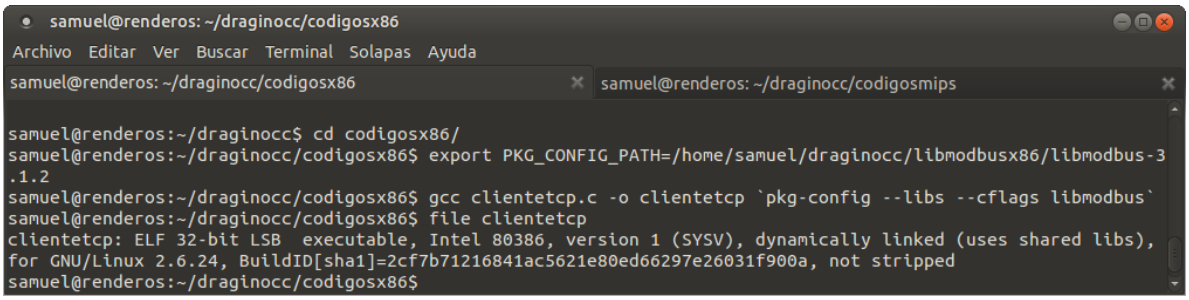

Figura B23. Compilacion del programa de ejemplo clientetcp.c

Ejecución del programa servidor-dragino en el Router Dragino:

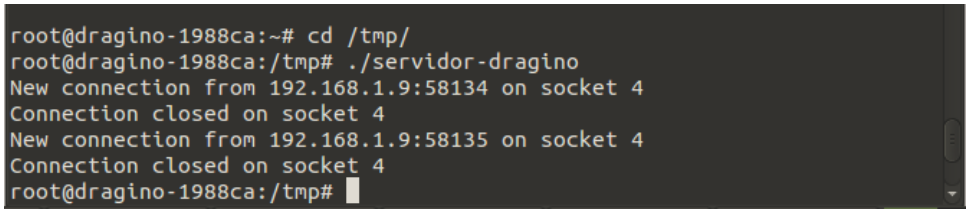

Figura B24. Ejecución del programa servidor-dragino.

Ejecución del programa clientetcp en el sistema huésped:

Hay que tomar en cuenta que la librería libmodbus no fue instalada en el sistema, solo configurada dentro del directorio *libmodbusx86/libmodbus-3.1.2,* por lo tanto hay que enlazar las librerías dinámicas utilizando la variable de entorno ***LD_LIBRARY_PATH***.

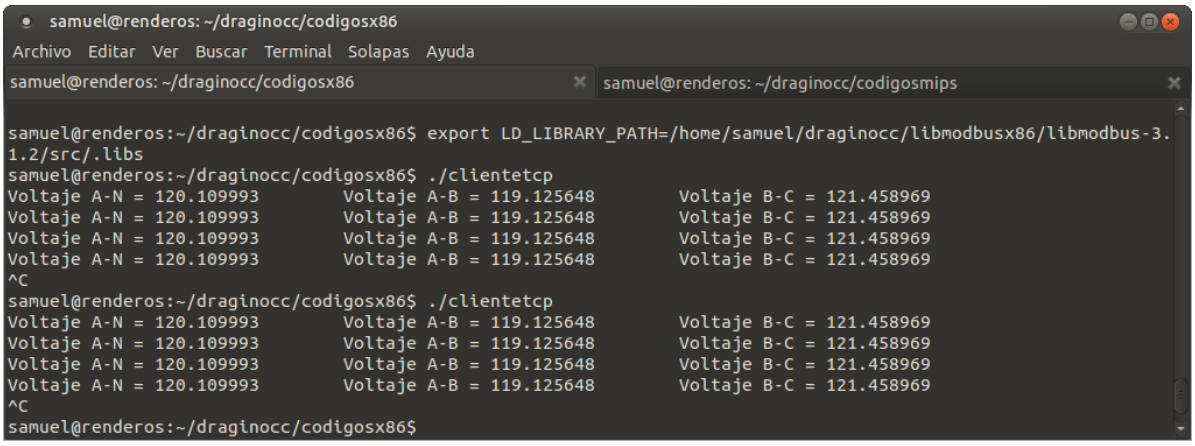

Figura B25. Ejecución del programa clientetcp.

Bibliografía

[1] Bonilla Perla, Juan José (2014). Diseño, configuración y supervisión de la red de medidores de energía eléctrica del campus central de la Universidad de El Salvador. http://ri.ues.edu.sv/5584/

[2] Duque Alas, Alexander Omar (2014). Optimización del sistema de monitorización remota de medidores de energía eléctrica. Tesis Ingeniería, Universidad de El Salvador. http://ri.ues.edu.sv/6534/

[3] Página oficial del proyecto Mesh Potato (MP01) de Village Telco.
http://wiki.villagetelco.org/Mesh_Potato_(MP01)

[4] Arteaga Hernández, Camilo Ernesto (2016). Protocolo de mantenimiento para redes inalámbricas de medidores de energía eléctrica. Tesis Ingenieria, Universidad de El Salvador.
http://ri.ues.edu.sv/11056/

[5] Redes Mesh.
http://wiki.ead.pucv.cl/index.php/Red_Mesh

[6] Guía de referencia sobre firmware SECN 1.1 y Batman-Adv.
http://wiki.villagetelco.org/SECN_1.1_User_Guide

[7] Redes inalámbricas de uso comunitario: un análisis comparativo de protocolos
http://bibliotecadigital.uca.edu.ar/greenstone/cgi-bin/library.cgi?a=d&c=Revistas&d=redes-inalambricas-comunitario

[8] Toolchain OpenWrt.
https://wiki.openwrt.org/es/about/toolchain

[9] Protocolo Modbus.
http://www.tolaemon.com/docs/modbus.htm

[10] Implementación de una red Modbus/TCP.
http://objetos.univalle.edu.co/files/Implementacion_de_una_red_MODBUS.pdf

[11] Manual de usuario Medidor SHARK 100
https://www.electroind.com/pdf/sp/SP_ES145701_100_100T_UserMan.pdf

[12] Manual de usuario Medidor SHARK 200
https://www.electroind.com/pdf/sp/ES149701_SP_Shark200_manual.pdf

[13] Arquitectura de un sistema embebido.
http://linuxemb.wikidot.com/tesis-c2

[14] Librería C GNU
http://linuxemb.wikidot.com/libc

[15] Compilación cruzada – GNU/Linux embebido.
http://linuxemb.wikidot.com/tesis-c3

[16] Cornejo Mejía, Marvin Andrés (2014). Aplicación del router Dragino como dispositivo de almacenamiento masivo.
http://ri.ues.edu.sv/5734/

[17] Página oficial del proyecto OpenWrt.
https://openwrt.org/

[18] Especificación del XML-RPC
http://www.lab.dit.upm.es/~labscom/practicas/xml_rpc.htm

[19] XML-RPC
http://xmlrpc.scripting.com/

[20] LuaSocket
http://w3.impa.br/~diego/software/luasocket/

[21] Zaldaña, Jonathan Alberto (2011). Medidor inalámbrico de consumo de energía eléctrica de bajo costo. Tesis Ingeniería, Universidad de El Salvador. http://ri.ues.edu.sv/2029/

[22] Raspberry Pi 2 Model B
https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

[23] Aplicación Web Mi consumo de energía – Arquitectura actual de interrogación
http://ues.miconsumodeenergia.com/

[24] Aplicación Web Mi consumo de energía – Nueva arquitectura de interrogación
http://uesmiconsumodeenergia.appspot.com/