ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"SOFTWARE DE MONITOREO CARDIACO EN LA NUBE"

INFORME DE PROYECTO DE GRADUACIÓN

Previa a la obtención del Título de:

INGENIERO EN CIENCIAS COMPUTACIONALES
ESPECIALIZACIÓN SISTEMAS DE INFORMACIÓN

Presentada por:

GUILLERMO ANDRÉS CASTILLO ALVARADO

Guayaquil - Ecuador
2015

AGRADECIMIENTO

A Dios.

A mi esposa, por quién perdí tantas materias.

A mis padres y abuelos, Quienes me forjaron.

A Miguel Yapur, quién siempre me apoyó.

DEDICATORIA

Con amor,

A mi familia y mis hijos,

A mi alma mater,

Y Ecuador mi patria.

TRIBUNAL DE SUSTENTACIÓN

_____
M.Sc. Sara Rios O.
SUBDECANA DE LA FIEC

_____
M.Sc. Miguel Yapur
Director del Proyecto de Graduación

_____
M.Sc. Víctor Asanza
Miembro del Tribunal

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este Informe, me corresponde exclusivamente; y el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral".

(Reglamento de exámenes y títulos profesionales de la ESPOL)

Guillermo Andrés Castillo Alvarado

RESUMEN

El presente trabajo expone el desarrollo de una herramienta para transmitir una señal eléctrica de un electrocardiógrafo, a través de la red, en tiempo real, utilizando *sockets(punto de comunicación de datos entre dos procesos)* en *python(lenguaje de programación de alto nivel)*. El electrocardiógrafo se conecta a una tarjeta BeagleBone Black(tarjeta controladora) que tiene instalada Linux como su sistema operativo, y que es capaz de leer una señal analógica, convertirla en una señal digital, y exponerla como un número real con una precisión de dos decimales en *python*.

El cliente que tiene conectado el electrocardiógrafo, toma la señal y la transmite a un servidor utilizando *sockets*. El servidor, a su vez, toma esta señal y la retransmite a una lista de consumidores. Esta lista de consumidores consiste en un *Array (Arreglo)* en memoria compartida que contiene *pipes*(tuberías para el intercambio de información entre procesos) de entrada, los cuáles son colocados dinámicamente por consumidores que deseen leer la señal. De esta manera es posible retransmitir la señal a varios clientes en tiempo real, eficientemente. La señal es luego retransmitida a uno o más clientes Web que soporten el protocolo de Web Sockets (punto de comunicación de datos entre dos procesos).

ÍNDICE GENERAL

ABREVIATURAS Y SIMBOLOGÍA

Adafruit_BBIO, librería de python que facilita el acceso al hardware de la tarjeta BeagleBone Black.

API, interfaz de programación de aplicaciones. Es una interfaz que permite la abstracción de funcionalidades de alguna librería en particular y que es a menudamente utilizada por algún otro programa con el objetivo de utilizar dichas funcionalidades.

BSD, Berkeley Software Distribution, fue un sistema operativo desarrollado y distribuido por la Universidad de Berkeley.

Bower, manejador de dependencias para el desarrollo de aplicaciones Web.

EKG, electrocardiograma o electrocardiografía. Es el registro de la actividad eléctrica del corazón, representada por un voltaje a través del tiempo.

Full-duplex, sistema de comunicación punto-a-punto que tiene la capacidad de comunicación simultánea entre éstos.

GET, método utilizado por el protocolo HTTP para obtener datos de un servidor.

Git, herramienta de control de versiones.

Github, servicio de hosting para repositorios git.

Gulp, herramienta de automatización de tareas para el

desarrollo de aplicaciones Web.

Handshake, en computación, se refiere al intercambio de mensajes para inicializar una sesión.

Host, computadora que se encuentra conectada a una red. El término se usa generalmente para indicar el nombre de dominio, o la dirección ip de una computadora.

HTTP, Hypertext Transfer Protocol, protocolo usado para la distribución de páginas Web a través de navegadores.

Javascript, lenguaje de programación que sirve para desarrollar aplicaciones Web.https://www.facebook.com/

POSIX, Portable Operating System Interface, es una familia de estándares especificados por la IEEE. Cumplen el objetivo de mantener compatibilidad entre sistemas operativos.

POST, método utilizado por el protocolo HTTP para enviar datos a un servidor.

Python Tornado, framework Web escrito en python.

SASS, precompilador de css.

Socket, punto de comunicación de datos entre dos procesos.

UNIX, es una familia de sistemas operativos multiusuario y multitarea, que se derivaron originalmente del sistema UNIX de AT&T.

Yeoman, herramienta de *scaffolding* para el desarrollo de aplicaciones Web.

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Un electrocardiógrafo es un dispositivo electrónico no invasivo, que permite leer la actividad eléctrica del corazón. Dicha actividad está representada por un voltaje a través del tiempo, y muchas veces es visualizado en un monitor electrónico, o impreso en papel para electrocardiogramas.

En nuestro trabajo indicamos el problema y nuestra propuesta de solución del mismo en el Capítulo 1 que consiste en encontrar la manera de distribuir la señal producida por un electrocardiógrafo, a través de la red, en tiempo real y que pueda se accedida por múltiples clientes desde un navegador Web o móvil.

Para lo cual nuestra propuesta es que se debe crear un servidor que reciba la señal de uno o más electrocardiógrafos, y que sea capaz de distribuirla a uno o más clientes Web.

Hemos hecho una descripción en el Capítulo 2 del marco teórico en donde se detalla los conceptos y la teoría detrás de los componentes utilizados en el proyecto. Las especificaciones de la tarjeta BeagleBone Black, la utilización de tecnologías como python, sockets, Websockets y pipes. Luego en capítulo 3 se describe el análisis realizado, para el desarrollo del software encargado de manipular el voltaje generado por un electrocardiógrafo y transmitirlo a través de la red estableciendo los requerimientos funcionales y

no funcionales.

En el capítulo 4 hacemos el desarrollo de nuestro diseño de la solución presentada de todo el sistema en el que está basado el proyecto, tanto en el hardware y software.

Finalmente en el Capítulo 5 hacemos una descripción de la implementación en donde se describe los detalles de la librería encargada de la transmisión de la señal eléctrica, el servidor de distribución de la señal, el servidor de WebSocket, y el cliente Web.

CAPÍTULO 1

1 GENERALIDADES

1.1 Antecedentes

Muchos médicos de cabecera alrededor del mundo trabajan y atienden pacientes en más de un hospital. En muchas ocasiones estos pacientes tienen la necesidad de que sus signos vitales sean monitoreados de manera continua.

En la actualidad, la mayoría de hospitales tienen personal de guardia que están activamente monitoreando los signos vitales de sus pacientes; y si ocurriera una emergencia, éstos llamarían a sus médicos de cabecera, alertándolos de lo ocurrido.

1.2 Descripción del problema

El problema que se presenta consiste en encontrar la manera de distribuir la señal producida por un electrocardiógrafo, a través de la red, en tiempo real y que pueda ser accedida por múltiples clientes

desde un navegador Web o móvil.

Se debe crear un servidor que reciba la señal de uno o más electrocardiógrafos, y que sea capaz de distribuirla a uno o más clientes Web.

1.3 Solución Propuesta

Producto de este trabajo de graduación se espera obtener los siguientes componentes de software:

- Programa escrito en python que pueda ser ejecutado desde una tarjeta BeagleBoneBlack, que tome la señal producida por un electrocardiógrafo y la transmita a través de la red a un servidor central para ser distribuida.
- Servidor central que reciba las señales de varios electrocardiógrafos a través de la red y que tenga la capacidad de distribuir dicha señal a uno o más clientes conectados.

- Aplicación Web que pueda ser vista desde un navegador, en una PC, un teléfono móvil o tablet, con la capacidad de conectarse a un servidor central, obtener la señal en tiempo real y mostrarla en pantalla.
- El alcance de este proyecto abarca la redistribución y visualización de una señal emitida por uno o más electrocardiógrafos, en uno o

más dispositivos con navegadores que soporten el protocolo WebSockets.

- Elementos como el manejo de usuarios y de seguridad informática, así como el desarrollo del electrocardiógrafo, están fuera del alcance de este proyecto.
- Para la generación de la señal, se tomará de un electrocardiógrafo desarrollado y construido en el laboratorio de Electrónica Mde la FIEC.

1.4 Alcance y Justificación

El proyecto se justifica con los siguientes puntos:

- La necesidad de un médico de cabecera de monitorear el ritmo cardiaco de uno o más pacientes, en uno o más hospitales.
- La posibilidad de monitorear el ritmo cardiaco de un paciente de manera remota, a través de una aplicación Web, desde una computadora o un teléfono inteligente.

Esto ayudaría a médicos y pacientes de la siguiente manera:

- Poder estar al tanto del ritmo cardiaco de los pacientes más críticos.
- La posibilidad de analizar activamente el estado de los pacientes y posiblemente prever el empeoramiento de el estado de salud actual.

- La capacidad de que el ritmo cardiaco sea monitoreado por uno o más médicos, en lugares remotos.
- La capacidad de que los familiares del paciente puedan monitorear el estado de sus seres queridos.

1.5 Objetivos

Los objetivos del presente trabajo son:

- Construir una aplicación para monitorear de forma remota y en tiempo real el estado del ritmo cardiaco de un paciente, con la finalidad de supervisar su evolución.
- Diseñar una aplicación que de forma colaborativa, permita realizar el monitoreo del ritmo cardiaco de un paciente por varios médicos.

CAPÍTULO 2

2 MARCO TEÓRICO

El siguiente capítulo detalla los conceptos y la teoría detrás de los

componentes utilizados en el proyecto. Las especificaciones de la tarjeta BeagleBone Black, la utilización de tecnologías como python, sockets, Websockets y pipes son algunos de los elementos que serán descritos a continuación.

2.1 Fundamentos de Operación del Electrocardiógrafo

El electrocardiógrafo es un aparato electrónico, capaz de registrar la actividad eléctrica del corazón a través del tiempo. El término para referirse a estos registros se denomina electrocardiograma. La lectura se realiza colocando electrodos en puntos estratégicos del cuerpo, con el objetivo de obtener los impulsos eléctricos generados por el corazón. El electrocardiógrafo detecta y amplifica los pequeños cambios eléctricos en la piel generados por el corazón, y registra los valores a través del tiempo.

El patrón generado, es registrado en lo que se conoce como papel de electrocardiograma, y es utilizado por especialistas para monitorear el corazón y detectar posibles enfermedades cardiacas.

Figura 2.1: Ritmo sinusal normal

En la figura 2.1 podemos observar el voltaje registrado a través del

tiempo, generado por un electrocardiógrafo.

2.2 BeagleBone Black

El BeagleBone Black es una plataforma de bajo costo, y soportada por la comunidad, hecha para desarrolladores y entusiastas, con las siguientes características mostradas en la tabla [[[[[[[1]]]]]]]:

Tabla 1: *Especificaciones Técnicas BeagleBone Black*

Procesador	AM335x 1GHz ARM® Cortex-A8
RAM	512MB DDR3 RAM
Almacenamiento	4GB 8-bit eMMC on-board flash storage
Conectividad	USB host, Ethernet, HDMI

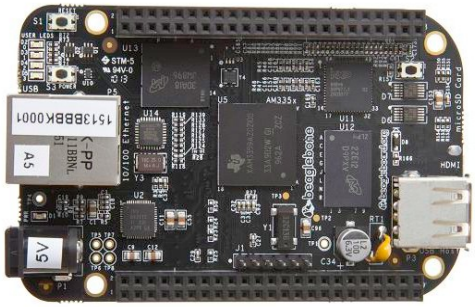

Figura 2.2: Tarjeta BeagleBone Black

En la figura 2.1 se muestra el diseño electrónico de la plataforma BeagleBone, la cual permite el desarrollo de componentes electrónicos que requieren control con entradas y salidas de voltaje. Éstas pueden ser manejadas en un ambiente de alto nivel en lenguajes como python o c (Orange County's Python Community Group).

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETn
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_50
GPIO_48	15	16	GPIO_51
GPIO_5	17	18	GPIO_4
I2C2_SCL	19	20	I2C2_SDA
GPIO_3	21	22	GPIO_2
GPIO_49	23	24	GPIO_15
GPIO_117	25	26	GPIO_14
GPIO_115	27	28	GPIO_113
GPIO_111	29	30	GPIO_112
GPIO_110	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	GPIO_7
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
GPIO_70	45	46	GPIO_71

Figura 2.3: Entradas y salidas eléctricas BeagleBone Black

Como se muestra en la figura 2.3, la tarjeta posee 92 pines con diferentes configuraciones para recibir y enviar señales eléctricas. Las entradas AINx(entrada analógica) en la sección de pines P9 son entradas analógicas capaces de leer voltajes hasta un máximo de 1.8v.

2.3 Python

Python es un lenguaje de programación de alto nivel, multiparadigma e interpretado, que utiliza el concepto de *dynamic typing,* que quiere decir que una variable puede tomar valores de distintos tipos.

El lenguaje fue inventado por Guido van Rossum y su filosofía se resume en los siguientes puntos [[[[[[[2]]]]]]]:

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.

- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Aunque lo práctico gana a la pureza.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que ya mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres (*namespaces*) son una gran idea ¡Hagamos más de esas cosas!

2.4 Sockets

Un socket es un punto de comunicación que sirve para la transferencia de datos entre dos procesos.

Berkeley sockets(o BSD sockets) es una librería que brinda una interfaz de programación de aplicaciones (API), para el uso de sockets de red y sockets de dominio UNIX, que permiten crear puntos de comunicación de datos entre procesos (IPC). La librería es actualmente parte del estándar POSIX[[[[[[[3]]]]]]].

Existen dos tipos de sockets: los sockets de red y los sockets de dominio UNIX. Un socket de dominio UNIX, es un punto de comunicación que sirve para intercambiar información entre uno o más procesos que se ejecutan en el mismo computador.

Un socket de red permite la comunicación de datos, entre procesos que se ejecutan en computadores diferentes a través de la red.

En python existe un módulo llamado *socket* que provee acceso a la interfaz de BSD sockets del sistema operativo.[[[[[[[4]]]]]]]

Esta interfaz es una transliteración directa de las llamadas al sistema UNIX, así como las funciones de la librería de interfaz de sockets del sistema operativo, a un estilo orientado a objetos en python: La función *socket()* retorna un objeto de tipo socket, cuyos métodos implementan varias de las funciones socket del sistema.

2.5 WebSockets

El protocolo WebSocket permite la comunicación de dos vías entre un cliente que corre código no confiable, en un ambiente controlado, y un *usuario* remoto. El protocolo consiste en un *handshake (intercambio de mensajes de inicialización),* seguido por un mensaje básico de encuadrado, a través de la capa TCP (Protocolo de Control de la Transmisión).

El objetivo de esta tecnología es de proveer un mecanismo de comunicación de dos vías, para aplicaciones basadas en navegadores Web, con servidores que no deseen utilizar múltiples conexiones HTTP (Protocolo de Transferencia de Hipertexto) [[[[[[[5]]]]]]].

Esta tecnología hace posible la comunicación interactiva entre un navegador Web y un servidor. El intercambio de datos ocurre de una manera eficaz y sin la sobrecarga de llamadas HTTP, como GET y POST, lo que permite el intercambio de datos en tiempo real.

En la figura 2.4 se muestra el ejemplo, crea y abre una conexión con el servidor: ws://www.example.com/socketserver[6]

```
1 var exampleSocket = new WebSocket("ws://www.example.com/socketserver", "protocolOne");
```

Figura 2.4: Ejemplo de creación de un websocket en un navegador web

```
exampleSocket.send("Here's some text that the server is urgently awaitir
```

Figura 2.5: Ejemplo de envío de datos a través de un websocket

Un servidor de WebSocket es una aplicación TCP que recibe conexiones en cualquier puerto específico, siguiendo el protocolo

```
exampleSocket.onmessage = function (event) {
  console.log(event.data);
}
```

Figura 2.5: Ejemplo de un socket recibiendo datos del servidor

descrito por la especificación.

El servidor recibe un *handshake* de parte del cliente que contiene, entre otras cosas, una clave secreta que es usada para la generación de una respuesta. Una vez establecida la conexión, el cliente y el servidor están listos para enviar datos del uno al otro en un modo *full-duplex*.

Tornado es un Web framework(marco) y librería de red escrita en python. Tornado posee una implementación del protocolo WebSocket, que permite la abstracción del desarrollador sobre el establecimiento de la conexión, permitiéndole concentrarse en el desarrollo de la

comunicación entre su servidor y el cliente Web.

```
class EchoWebSocket(websocket.WebSocketHandler):
    def open(self):
        print "WebSocket opened"

    def on_message(self, message):
        self.write_message(u"You said: " + message)

    def on_close(self):
        print "WebSocket closed"
```

Figura 2.6: Implementación de un servidor de WebSockets en python utilizando Tornado

2.6 Pipeline

Un pipeline consiste en una cadena de entradas y salidas, en donde las salidas de un proceso se convierten en la entrada que recibe el siguiente elemento en la cadena.

Para programas que requieren comunicación entre hilos del mismo proceso, un pipeline puede ser utilizado para que el intercambio de datos sea rápido y efectivo.

En python el método *os.pipe()* retorna un par de descriptores de archivo, uno de lectura, y otro de escritura. De esta manera, un proceso puede tomar el descriptor de lectura y esperar a que algún

otro proceso ingrese datos en el descriptor de escritura.

CAPÍTULO 3.

3 ANÁLISIS

El siguiente capítulo describe el análisis realizado, para el desarrollo del software encargado de manipular el voltaje generado por un electrocardiógrafo y transmitirlo a través de la red.

3.1 Requerimientos funcionales

Producto de este proyecto de graduación, y tomando en cuenta el alcance antes descrito, se detallan a continuación los requerimientos para implementar este sistema de monitoreo.

Cliente EKG

Es necesario desarrollar un cliente, que obtenga la señal eléctrica generada por un electrocardiógrafo y la transmita a través de la red

utilizando sockets. Este cliente debe ser ejecutado en la tarjeta BeagleBone Black.

Servidor de distribución de señales

Se necesita un servidor que reciba la señal enviada por el cliente EKG, y que tenga la capacidad de distribuirla a https://mail.google.com/mail/u/0/#inboxotros clientes de una manera efectiva y con poca sobrecarga, para que la señal pueda ser obtenida en tiempo real.

Servidor de WebSockets

El servidor de Websockets debe correr en el mismo espacio de memoria que el servidor de distribución de señales. De esta manera, es posible utilizar pipes para la redistribución de las señales obtenidas por el servidor de distribución.

Cliente Web

El cliente Web debe ser capaz de conectarse al servidor de Websockets y hacer un requerimiento para monitorear un cliente en específico. El cliente envía el ID de un cliente EKG al que se desea monitorear. El servidor de Websockets analiza el requerimiento y busca el cliente EKG requerido, de entre la lista de clientes conectados. Si el cliente es encontrado, el servidor de Websockets se encargará de crear el canal a través del cual el cliente Web reciba los voltajes al tiempo que son generados. Si el cliente no es encontrado,

el servidor no retornará ningún dato.

El cliente debe presentar la señal recibida en tiempo real en un gráfico en pantalla.

3.2 Requerimientos no funcionales

El sistema debe soportar a varios clientes EKG conectados al servidor de distribución.

Una misma señal debe poder ser vista en varios clientes Web conectados al servidor de Websockets.

La señal debe ser presentada en tiempo real a cada uno de los clientes Web.

CAPÍTULO 4.

4 DISEÑO

En la figura 4.1 se muestra la representación gráfica de todo el sistema en el que está basado el proyecto. Para su comprensión, este capítulo se dividirá en dos grandes partes vistas desde el cliente tanto como en hardware y software.

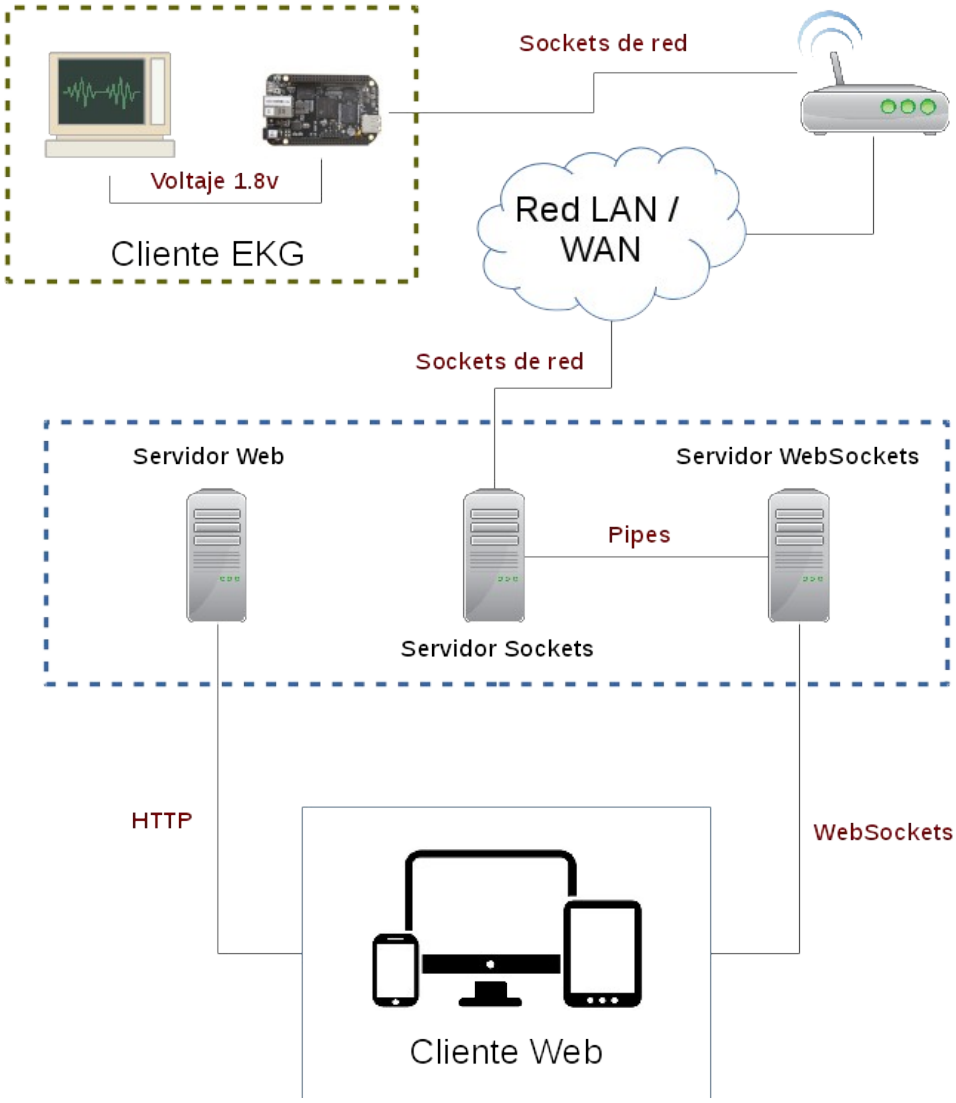

Figura 4.1: Representación gráfica del sistema

4.1 El Cliente EKG: Hardware

El cliente EKG, que se compone de dos partes:

- El electrocardiógrafo: Es el dispositivo electrónico encargado de leer los impulsos eléctricos del corazón. Este dispositivo tiene como entradas los electrodos que se conectan al cuerpo y produce

como salida un voltaje que representa la actividad cardiaca.

- Tarjeta Beaglebone Black: Es la micro-computadora encargada de capturar el voltaje y transmitirlo a través de la red. Tiene conectada la salida de voltaje del electrocardiógrafo en una de las entradas analógicas, capaces de leer un voltaje y manipularlo en un ambiente de alto nivel, como un número real de dos dígitos de precisión. Existen dos maneras de representar dicho voltaje: como un numero entre 0 y 1, y como un voltaje entre 0 y 1800 mV. También posee la capacidad de conectarse a la red a través de LAN o algún dispositivo inalámbrico. De esta manera, la señal de voltaje emitida por el electrocardiógrafo puede ser capturada y transmitida a través de la red.

4.2 El Cliente EKG: Software

El software en el cliente está constituido por una aplicación escrita en python que utiliza sockets y el módulo de Adafruit_BBIO. El cliente debe leer el voltaje utilizando la librería Adafruit_BBIO, con uno de sus métodos de la siguiente manera:

```
1. import Adafruit_BBIO.ADC as ADC
2.         ADC.setup()
3.         voltaje = ADC.read('P9_38')
```

La línea número cuatro, en el código anterior, retorna el voltaje entre 0 y 1, del pin ubicado en la posición P9 número 38.

Este voltaje será leído para ser enviado a través de sockets de red, con una frecuencia de 50 Hz.

Para iniciar el envío del valor del voltaje, se establece una conexión a través de sockets en el puerto escogido, utilizando la librería *sephiroth*, producto de este trabajo de graduación y descrita más adelante.

El cliente funciona de la siguiente manera:

```
1.          defstart(host, port):
2.      # Este id puede ser cualquier string
3.          client_id ='000000000000000'
4.          client = sephiroth.Client(client_id)
5.          client.connect(host=host, port=port)
6.      print'Client %s connected'% client_id
7.      count =0.00
8.          while1:
9.          count  = count + RFQ
10.         signal = get_signal(count)
11.         send_data(client, count, signal)
12.         time.sleep(RFQ) # Read frequency
```

El programa entra en una iteración infinita, tan solo para detener su

ejecución cada tantos segundos, establecidos por la variable RFQ, que es la frecuencia de lectura. Lo que se envía consiste en una cadena de 22 caracteres de longitud como se muestra en la tabla 2:

Tabla 2 Ejemplo de cadena de caracteres enviada por el cliente EKG

“00000000004.21;0000.12”	
00000000004.21	0000.12
Tiempo transcurrido	Voltaje

4.3 Sephiroth: Señales eléctricas a través de la red

Sephiroth es una pequeña librería escrita en python, que tiene como objetivo abstraer las funcionalidades necesarias para transmitir una señal a través de la red. La librería implementa lo siguiente:

- Clases
  - **Client**: Clase que representa el cliente EKG para comunicación de datos a través de sockets. Al momento de instanciar esta clase, se debe recibir un **ID** con el cuál se va a identificar al cliente al momento de conectarse a un servidor. También recibe los parámetros opcionales de **sock** y **msg_len.** La variable **sock** es un socket opcional que puede ser pasado como referencia para ser usado por el cliente al momento de

enviar y recibir datos. La variable **msg_len**, es el tamaño del mensaje que va a ser enviado, y tiene un valor predefinido de 22. La clase tiene un método **connect**, que recibe la dirección IP y el puerto del servidor. Una vez conectados, se puede empezar a enviar datos al servidor.

- **Server**: Clase que representa el servidor de redistribución de señales. Extiende de la clase **SocketServer.TCPServer.** Al momento de ser instanciada, esta clase recibe una función como referencia. Esta función es ejecutada cada vez que un mensaje es recibido. También puede recibir los siguientes parámetros opcionales: El tamaño del mensaje, el tamaño de la identificación del cliente, el host y el puerto en dónde debe escuchar el servidor.
- La clase **handler** encargada de la manipulación de los mensajes de entrada se crea dinámicamente con el método **get_handler** descrito más adelante.
- **ClientNotFound:** Es una clase que extiende de **Exception** y que sirve como elemento de control de flujo.

- Métodos
  - **get_handler:** Este método retorna una clase de tipo SocketServer.BaseRequestHandler, necesaria para manejar los mensajes de entrada en el servidor. Dentro de la

implementación de esta clase se encuentra el método **handle,** que será el encargado de tomar la señal y redistribuirla. Para lograr esto, se recorre un arreglo de pipes de escritura, y por cada pipe encontrado, se escribe el mensaje recibido del cliente EKG. Este arreglo es una variable del módulo sephiroth y que puede ser accedida desde cualquier parte del programa. De esta manera es posible crear un par de pipes de lectura y escritura, colocar el pipe de escritura en el arreglo y esperar por datos en el pipe de lectura.

- **get_pipes(id_client)**: Obtiene el arreglo de pipes de escritura de un cliente en específico. Levanta una excepción de tipo **sephiroth.ClientNotFound,** si el cliente no se encuentra conectado.
- **add_pipe(id_client, pipe)**: Añade un pipe de escritura en el arreglo de pipes del cliente con id_client.
- **remove_pipe(id_client, pipe)**: Remueve un pipe de la lista de pipes de escritura.
- **is_client_connected(id_client)**: Indica si el cliente con ID id_client se encuentra actualmente conectado al servidor.
- **add_client(id_client)**: Añade el cliente con ID id_client a la lista de clientes conectados.
- r**emove(id_client)**: Remueve el cliente con ID id_client de la

lista de clientes conectados.

4.4 Servidor de distribución de señal

El servidor de distribución de señales maneja todo a través de la clase **sephiroth.Server**. Cada vez que un mensaje es recibido, el servidor recorre un arreglo de *pipes* de escritura de un cliente en específico y escribe el mensaje recibido en cada *pipe* encontrado. Este arreglo es una variable del módulo **sephiroth** y puede ser accedida desde cualquier parte del programa.

El servidor se ejecuta en un hilo propio.

4.5 Servidor de WebSockets

La funcionalidad del servidor de Websockets, se implementa al extender de la clase **tornado.Websocket.WebSocketHandler**. Esta clase contiene métodos como open, on_message y on_close, que manejan la funcionalidad del servidor. El servidor crea un hilo de ejecución por cada cliente que hace una petición. Típicamente, un cliente Web envía un primer mensaje con el ID del cliente EKG que se desea monitorear. El servidor crea un hilo nuevo de ejecución y, si el cliente es encontrado, se crea a su vez un par de *pipes* de escritura y lectura. El *pipe* de escritura es colocado en el arreglo específico del cliente que se desea monitorear. Por último, el hilo del servidor entra en una iteración infinita, la cual está constantemente tratando de leer el pipe de lectura. De esta manera, se actúa en conjunto con el

servidor de distribución de señales, pues cada vez que éste recibe un dato, lo escribe inmediatamente en cada pipe de escritura.

Existe otro servidor corriendo en el mismo proceso, pero en un hilo diferente, que se encarga de retornar el ritmo cardiaco.

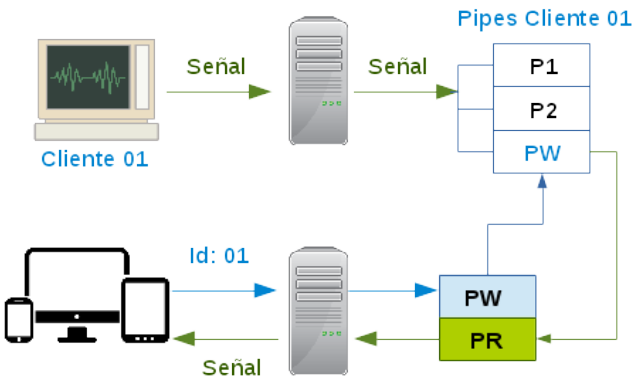

Figura 4.2: Distribución de la señal generada

4.6 Servidor Web

El servidor Web consta simplemente de un servidor de archivos estáticos que se ejecuta utilizando nodejs (Lenguaje abierto que usa JavaScript). El servidor puede ser ejecutado utilizando la herramienta *gulp* de la siguiente manera:

```
gulp serve
```

4.7 Cliente Web

La estructura general de esta aplicación fue creada utilizando las

siguientes herramientas: yeoman, bower y gulp. Los estilos se generan utilizando SASS (Syntactically Awesome Stylesheet) que es es un lenguaje de estilo de hoja diseñado inicialmente por Hampton Catlin. El gráfico para visualizar la señal es generado utilizando una librería de javascript llamada smoothiecharts.

El cliente consiste en una aplicación Web con los siguientes componentes:

- Una caja de texto para ingresar la dirección IP del servidor WebSockets
- Una caja de texto para ingresar la identificación del cliente EKG
- Controles +/- para subir y bajar la velocidad del muestreo
- Un botón para reiniciar la conexión y el muestreo.
- Un botón para pausar el muestreo
- Una etiqueta que muestra el ritmo cardiaco

El cliente utiliza javascript para crear una conexión a través del protocolo WebSocket en el servidor indicado. Una vez establecida la conexión, el servidor comienza el envío de datos, los cuáles son mostrados en pantalla inmediatamente.

El cliente establece una conexión en un puerto separado, para consultar periódicamente el ritmo cardiaco.

4.8 Diseño de pruebas

A continuación se detallan las pruebas realizadas que aseguran el correcto funcionamiento del sistema:

En el cliente EKG

Para acelerar las pruebas de funcionamiento se creó un script que envía una señal de prueba(en este caso, variaciones de la función seno). El script puede ser ejecutado desde una PC o desde el dispositivo BeagleboneBlack.

Para las pruebas con el EKG, se conectó el dispositivo a una persona, y la señal fue transmitida a través de la red.

En el servidor

El servidor se ejecuta desde una PC y debe estar conectado en la misma red que los clientes EKG así como los clientes Web. Las pruebas fueron realizadas desde una PC con las siguientes características:

Procesador: Intel Core i7 i7-3517U

6GB de memoria RAM

Sistema Operativo: Debian GNU/Linux 8 (jessie) 64-bit

Las pruebas aseguran lo siguiente:

La señal emitida por un cliente EKG, es recibida y transmitida en tiempo real.

La misma señal es transmitida por lo menos a 20 clientes Web, sin sufrir deterioro, dentro de una red LAN.

Al menos 30 clientes EKG pueden ser conectados concurrentemente

El ritmo cardiaco es calculado y transmitido correctamente.

En el cliente Web

En el cliente Web se realizaron las siguientes pruebas de funcionamiento:

El cliente Web es capaz de conectarse a un servidor y recibir la señal de un cliente escogido.

El ritmo cardiaco se presenta en pantalla y se actualiza constantemente

CAPÍTULO 5.

5 IMPLEMENTACIÓN

Este capítulo describe los detalles de implementación de la librería encargada de la transmisión de la señal eléctrica, el servidor de distribución de la señal, el servidor de WebSocket y el cliente Web.

5.1 Hardware

El cliente EKG consiste en un electrocardiógrafo implementado por estudiantes de la ESPOL. Los detalles de la implementación electrónica están fuera del alcance de este trabajo de graduación. Sin embargo, es importante mencionar que el dispositivo es capaz de leer la actividad cardiaca de una persona y presentarla como un voltaje entre 0 y 1.8 voltios.

Esta salida de voltaje es conectada a la tarjeta BeagleBone Black, dónde es automáticamente transformada en digital, y hecha disponible

para el usuario a través del sistema operativo.

5.2 Software

A continuación las dependencias de software del sistema:

- **En el cliente EKG:**
  - Debian GNU/Linux como sistema operativo
  - Python 2.7.9
  - Módulo de sockets
  - Módulo Adafruit_BBIO
- **En el servidor de distribución**
  - Python 2.7.9
  - Módulo de sockets
  - Pipes
- En el servidor de WebSockets
  - Python 2.7.9
  - Módulo de sockets
  - Framework Tornado para servidor WebSocket
  - Pipes

- Para el cliente Web
  - Yeoman
  - Bower
  - Gulp
  - HTML5
  - CSS3
  - Javascript
  - WebSockets
  - SmoothieCharts.js

5.3 Instalación

A continuación se detallan los pasos para instalar todos los elementos del sistema. El primer paso es clonar el repositorio del proyecto en github, de la siguiente manera:

git clone https://github.com/phasnox/sephiroth.git

Cliente EKG

Lo primero que se debe hacer es conectar la tarjeta BeagleBone

Black, a la computadora a través del cable USB de debugging.

Una vez conectada, la tarjeta se enciende automáticamente y se debe esperar a que arranque el sistema operativo. En el computador, la tarjeta se muestra como un dispositivo de red con la dirección IP 192.168.7.1. Dentro de esta red, se va a encontrar al dispositivo con una dirección ip de 192.168.7.2. El dispositivo tiene un servidor ssh corriendo y puede ser accedido de la siguiente manera:

```
ssh root@192.168.7.2
```

No se requiere contraseña.

Una vez conectado se puede empezar a ejecutar comandos en consola, creando un directorio llamado **sephiroth**

```
mkdir sephiroth
```

Antes de esto se deben copiar todos los archivos necesarios para ejecutar el cliente:

```
cd sephiroth
```

```
scp bbb_client.py sephiroth.py root@192.168.7.2:/sephiroth/
```

Una vez copiados los archivos, se ejecuta el cliente con el siguiente comando:

```
python bbb_client -o 192.168.7.1
```

De esta manera puede empezar la transmisión de los datos una vez que se encuentre levantado el servidor de distribución.

El servidor de distribución y WebSockets

Para instalar el servidor de distribución, tan solo son necesarias la carpeta del repositorio y las dependencias del sistema. Para ejecutar el servidor se hace lo siguiente:

```
cd sephiroth

python start_server.py
```

Este comando inicia el servidor de sockets y Websockets.

El cliente Web

Para instalar y ejecutar el cliente Web se hace lo siguiente:

Se ingresa a la carpeta frontend y se instala las dependencias de node y bower.

```
cd sephiroth/frontend

npm install

bower install
```

Se utiliza gulp para correr el servidor de prueba:

```
gulp serve
```

Se puede acceder al servidor desde el navegador a través de

http://localhost:9000/

5.4 Pruebas y Análisis de Resultados

A continuación se detallan los resultados de las pruebas realizadas que aseguran el correcto funcionamiento del sistema:

En el cliente EKG

Una vez iniciado el servidor, se ejecuta el script del cliente. El cliente se conecta sin problemas, a menos que exista un problema con el servidor o la red.

Se detiene la ejecución del servidor para determinar el correcto funcionamiento de conexión del cliente. El cliente intenta reconectarse cada 5 segundos, en caso de que exista un error en la comunicación. También se desconecta el dispositivo de la red, con resultados positivos en el comportamiento de reconexión.

En el servidor

Se pudo cerciorar que la señal emitida por un cliente EKG, es recibida y transmitida en tiempo real. La señal fue visualizada en el cliente Web.

La misma señal fue transmitida a 20 clientes Web, sin sufrir deterioro, dentro de una red LAN.

Se conectan 30 clientes EKG concurrentemente y ninguna de las

señales se vio afectada.

El ritmo cardiaco, calculado en el servidor, fue transmitido correctamente a los clientes Web.

En el cliente Web

En el cliente Web se realizaron las siguientes pruebas de funcionamiento:

El cliente Web fue capaz de conectarse a un servidor y recibir la señal de un cliente escogido. Se hicieron pruebas con varios clientes EKG conectados, transmitiendo una señal que consistía en una función seno a través del tiempo.

El ritmo cardiaco de cada cliente EKG fue presentado en pantalla y actualizado constantemente.

CONCLUSIONES

1. La introducción de tarjetas como BeagleBone Black, permiten a muchos desarrolladores, no electrónicos, involucrarse en proyectos que requieren la manipulación de señales eléctricas. Podríamos concluir que gracias a este tipo de dispositivos, la brecha que separa el desarrollo de software y el desarrollo de soluciones que requieren automatización y manejo de señales eléctricas ha sido reducido.
2. Es posible transmitir y monitorear señales eléctricas a través de la red en tiempo real utilizando sockets en un lenguaje de alto nivel.
3. El protocolo WebSocket es una tecnología Web que permite la transmisión de datos en tiempo real entre un navegador y un servidor de datos de manera segura y confiable.

RECOMENDACIONES

1. Se recomienda utilizar sockets en todo escenario en el cual se necesite transmitir datos en tiempo real.
2. Se recomienda utilizar el protocolo WebSocket en cualquier aplicación Web que necesite comunicación de dos pares en tiempo real. Esta tecnología puede ser utilizada para crear aplicaciones de chat y video, así como otros escenarios en los que se requiera visualizar datos en tiempo real.
3. El cliente Web producto de este proyecto de graduación, debe ser accedido en un navegador que soporte la tecnología de Websockets. A la fecha, las últimas versiones de los navegadores Chrome, Firefox, Opera, Safari e Internet Explorer, así como las últimas versiones de los navegadores Web de Android y Apple, soportan esta tecnología.
4. Se recomienda utilizar un servidor con una alta capacidad de procesamiento, de preferencia un CPU de más de 2 GHz.
5. La clave del cliente EKG debe ser generada con un algoritmo de generación de claves seguro.

BIBLIOGRAFÍA

[1] , BeagleBone Black, 2014, http://beagleboard.org/black

[2] Tim Peters, The Zen of Python, 2004

[3] IEEE and The Open Group, The Open Group Base Specifications, 2013

[4] Python Software Foundation, socket - Low-level networking interface,

[5] Fette & Melnikov, The WebSocket Protocol, 2011

[6] Sheppy et al, Writing WebSocket client applications, 2014, https://developer.mozilla.org/enUS/docs/WebSockets/Writing_WebSocket_client_applic