Aplicaciones de Internet de las cosas usando Mosquitto y MQTT

IoT applications based on Mosquitto and MQTT

Edward P Guillen P[1], Carlos Omar Ramos L[1], Leonardo J Ramirez L[1]

[1] Grupo de Investigación GISSIC, Universidad Militar Nueva Granada, Bogotá, Colombia
gissic@unimilitar.edu.co

Resumen. El internet de las cosas –IoT es una nueva tecnología que logra reunir una amplia cantidad de sensores, procesadores y protocolos. La presente investigación muestra paso a paso la implementación de un sistema IoT para medir el gradiente de temperatura en un ambiente abierto. Se inicia con la selección de los sensores y la programación para el pre y procesamiento usando Arduino. Se comparan las medidas de los sensores para realizar un análisis estadístico. Luego se realiza un segundo experimento con el monitoreo de la frecuencia cardiaca y se analiza el proceso de captura de datos y envío a la tarjeta Raspberry, la instalación del bróker Mosquitto, la conexión de este a la Raspberry. Por último, re realizan las pruebas de interconexión entre un Publisher y el suscriptor de la topología propuesta usando el protocolo MQTT. Los montajes y sus resultados pueden ser reproducidos para nuevos experimentos.

Palabras clave: Internet, MQTT, Mosquitto, Arduino, Temperatura, Frecuencia cardiaca.

Abstract. The Internet of Things - IoT is a new technology that manages to bring together a wide number of sensors, processors and protocols. The present investigation shows step by step the implementation of an IoT system to measure the temperature gradient in an open environment. It starts with the selection of sensors and programming for pre and processing using Arduino. Sensor measurements are compared to perform statistical analysis. Then a second experiment is carried out with the monitoring of heart rate and the process of data capture and sending to the Raspberry card, the installation of the Mosquitto broker, its connection to the Raspberry are analyzed. Finally, they perform the interconnection tests between a Publisher and the subscriber of the proposed topology using the MQTT protocol. The assemblies and their results can be reproduced for further experiments.

Keywords: Internet, MQTT, Mosquitto, Arduino, Temperature, Heart rate.

1 Introducción

IoT (internet of things) define uno de los procesos de cambio más grande de internet, a través de la reunión de cuatro pilares fundamentales :datos, procesos equipos y personas; Debido a esta naturaleza, es necesario entender cómo es el funcionamiento de cada uno de estos ítems ,que permitirán aumentar la capacidad para predecir actuar

y ejecutar acciones que busquen el mejoramiento de la calidad de vida de la humanidad, internet de las cosas permitirá que para el año 2020 según estimaciones alrededor de 50.000 millones de dispositivos se encuentren suministrando información a la red [1],pero para que esto sea una realidad; para esto, en esta experiencia, se realiza la implementación de un escenario que relacione los pilares de IoT de forma práctica a través de la comunicación exitosa de un "Publisher" (conformado por el conjunto definido por actuador y sensorica) y "Suscriber" (usuario conectado mediante aplicación Java) mediante un bróker MQTT Mosquitto montado en la nube. Esta temática es de suma relevancia, debido a que permite generar un marco conceptual y práctico para entender la evolución de internet y como el uso de nuevas plataformas renovará la manera de conectarnos con los elementos de nuestro entorno.

2 Método

Message Queue Telemetry Transport –MQTT: Es un protocolo de conexión usado en M2M (machine to machine) el cual envía datos telemétricos como si fueran mensajes. Tiene la forma de la mensajería que es Publish y Subscribe, es decir quien hace petición y quien escucha y responde. Este presenta dos características fundamentales para su funcionamiento, las cuales son:

- SERVIDORES
- CLIENTES
- HERRAMIENTAS

En servidores, el más utilizado es Mosquitto, que es de código abierto, soporta lenguajes de programación C, C++ y Python. En cliente, Eclipse Paho se usa y se comporta como "cliente Java" Por ultimo en herramientas se tiene Eclipse Paho que sirve también para comunicarse con el brocker y mqtt.io que da un cliente "web" para interacción con los brocker públicos que tengan el protocolo MQTT [2].

Phyton: Es unos de los lenguajes más utilizados en conexiones MQTT y uso de la tarjeta programable raspberry pi. Python es un lenguaje de programación orientado a objetos, que se usa para elaborar aplicaciones o servicios para la red. Ha sido una herramienta importante debido a lo que ofrece, es decir las librerías y funciones que son varias, los sistemas operativos que soporta cuando se desarrolla tales como Unix, Windows, Mac, entre otros y, por último, la facilidad en manejar el programa por primera vez para la creación de códigos [3].

3 Método

Primero se desarrollará la programación que se ejecutara para el sensado con el sensor lm35, una vez compilado y adquirirlo sobre la tarjeta Arduino se ejecutarán las pruebas de funcionamiento. Tras finalizar este proceso, se comparará los datos obtenidos mediante el sensor con datos tomados de un multímetro con termocupla. En la figura 1 se observa la arquitectura del sistema IoT propuesto:

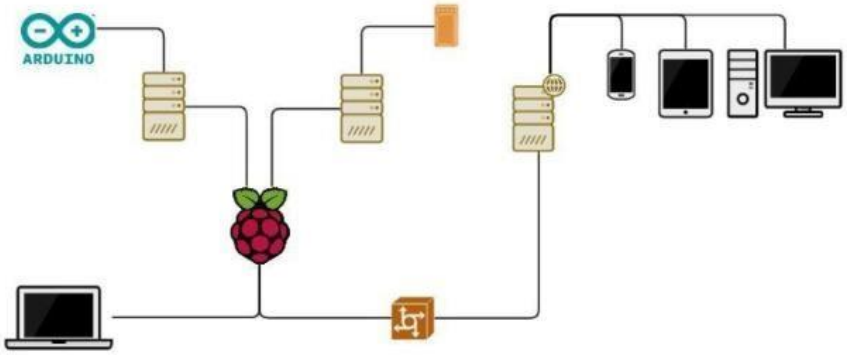

Fig. 1. Arquitectura de conexión del sistema IoT desarrollado

El sensor usado es LM335 para capturar los datos de temperatura que serán enviados al bróker MQTT, el código de programación se encuentra a continuación:

```
float tempC; int led=10; int tempPin = 0; // Definimos la
entrada en pin A0 void setup()
{
// Abre puerto serial y lo configura a 9600 bps
Serial.begin(9600); pinMode(led,OUTPUT);
} void loop() {
// Lee el valor desde el sensor tempC =
analogRead(tempPin); // Convierte el valor a temperatura
tempC = (5.0 * tempC * 100.0)/1024.0; // Envia el dato al
puerto serial Serial.print(tempC); Serial.print(" grados
Celsius\n");
// Espera medio segundo para repetir el loop delay(500);
// Enciende el led a temperaturas mayores a
28° if (tempC > 28.00){ digitalWrite(led,HIGH);
delay(100);}
// Enciende el led a temperaturas menores a
25° else if(tempC < 25.00){ digitalWrite(led,HIGH);
delay(100);}
// apaga el led a temperaturas entre 25° y
28° else{ digitalWrite(led,LOW); delay(100);}

}
```

4 Resultados

4.1 Aplicación IoT para temperatura

Primero se observa en la tabla 1 los resultados del terminal o monitor serie que se obtiene en la herramienta sketch, luego de esto se realiza la comparación con los resultados obtenidos en el multímetro fluke con termocoupla incorporada.

Tabla 1. Datos experimentales de temperatura ambiente.

Termocoupla	LM35	Diferencia
25.6°C	23.93°C	1.67
25.6°C	23.44°C	2.16
25.5°C	23.93°C	1.57
25.6°C	23.93°C	1.67
25.6°C	23.93°C	1.67
25.6°C	23.44°C	2.16
25.7°C	23.93°C	1.77
25.7°C	23.41°C	1.29
25.8°C	24.90°C	0.9
25.8°C	24.90°C	0.9
25.8°C	24.41°C	1.39
25.9°C	24.41°C	1.19
25.8°C	24.90°C	0.9
25.8°C	24.90°C	0.9
25.8°C	24.90°C	0.9
25.9°C	24.41°C	1.49

Para verificar las diferencias entre los datos obtenidos en el sensor de temperatura y aquellos obtenidos con herramientas de alta precisión se hace uso de mediciones como la media y la desviación estándar.

Media Termocoupla = 25,81290323

Media Sensor LM335 = 24,28709677

La diferencia es 1.51 en el valor de temperatura de la media. Se obtiene el valor de la desviación estándar que permite identificar el margen de eficacia del sensor con respecto a mediciones de alto margen de exactitud que se obtienen con la termocoupla. La diferencia es 0.3552 en el valor de desviación estándar.

Desviación estándar Termocoupla = 0.1384243

Desviación estándar Sensor LM335 = 0.40458929

Es necesario analizar las características técnicas del sensor lm 35, para identificar si este cumple con los requerimientos técnicos. Estos datos son obtenidos de la ficha lm35 de la empresa Texas Instruments [4].

Los valores de las variables que juegan un papel fundamental que corresponden a aquellos puntos que para nuestra aplicación son críticos. Con los anteriores datos de la ficha técnica se observó que aunque para aplicaciones en las que se requiere un nivel de exactitud medio este tipo de sensor es ideal, para aplicaciones médicas se requiere

un alto grado de precisión que permita realizar censado de forma ideal, se identifica además que es susceptible a variaciones de flujos de aire, debido a que este sensor utiliza estos flujos para establecer la escala de voltaje que utiliza para la conversión a grados centígrados ,otro punto importante es la alimentación de este sensor que lo hace útil para Arduino que proporciona de 0 a 5V en la salida analógica justo en el margen de operación de este sensor.

4.2 Aplicación IoT para muestreo de la Frecuencia Cardiaca

A continuación, se presenta el código de ejecución del sensor de pulso usado como segundo dentro del sistema IoT planteado, este sensor permitirá realizar un monitoreo constante al estado del paciente en tiempo real, generando en intervalos de 2ms muestras.

Los datos que de este se arrojan serán enviados al bróker y por ultimo recibidos por el cliente Java.

En las siguientes líneas de comando se realiza la declaración de los pines usados para el control y ejecución del sensor de pulso:

```
  int pulsePin = 0; // indicación de pin usado para la toma
de los datos por parte del sensor .se usa el pin analogo 0
int blinkPin = 13; int fadePin = 5; int fadeRate = 0;
//variables volatiles
  volatile int BPM; // declaracion de valor variables
declaradas para los tiempos de muestreo y tiempos de
interrupcion volatile int Signal; volatile int IBI = 600;
volatile boolean Pulse = false; volatile boolean QS =
false;
  // configuración de metódos de visualización de los datos
,para este caso se habilita el uso del monitor ASCCI del
sketch de Arduino static boolean serialVisual = true; //
Set to 'false' by Default. Re-set to 'true' to see Arduino
Serial Monitor ASCII Visual Pulse void setup(){
pinMode(blinkPin,OUTPUT);         pinMode(fadePin,OUTPUT);
Serial.begin(115200);
  interruptSetup();  //configuración de la rutina de
interrupción
  }
  // ejecución de rutina para detectar void loop(){

  serialOutput() ;

  if (QS == true){ // A Heartbeat Was Found fadeRate = 255
serialOutputWhenBeatHappens();     .     QS     =     false
ledFadeToBeat(); delay(20); }
```

```
  void ledFadeToBeat(){ fadeRate -= 15; // set LED fade
value      fadeRate      =      constrain(fadeRate,0,255);
analogWrite(fadePin,fadeRate);
  }
```

Esta función es declarada dentro del archivo principal y es usada para la representación de los datos que son tomados del sensor:

```
  void   serialOutput(){   if   (serialVisual   ==   true){
arduinoSerialMonitorVisual('-', Signal);
  } else{
  sendDataToSerial('S', Signal);
  }
  }
  //funcion que permite determiner la prevalencia o no de
pulso cardiaco en el sensor, en caso de existir; seran
leidos los datos para su posterior procesamiento void
serialOutputWhenBeatHappens(){ if (serialVisual == true){
  Serial.print("*** Heart-Beat Happened *** ");
  Serial.print("BPM: ");
  Serial.print(BPM);
  Serial.print(" ");
  } else{
  sendDataToSerial('B',BPM); sendDataToSerial('Q',IBI);
  }
  }
  //envia los datos al Puerto serial ,"conexión con la
tarjeta raspberry PI 3"
  void sendDataToSerial(char symbol, int data ){
  Serial.print(symbol);
  Serial.println(data);
  }
  void arduinoSerialMonitorVisual(char symbol, int data ){
const int sensorMin = 0; // sensor minimum, discovered
through experiment const int sensorMax = 1024; // sensor
maximum, discovered through experiment int sensorReading =
data; int range = map(sensorReading, sensorMin, sensorMax,
0, 11);
  // estructura switch case para determinar el nivel de la
señal del sensor y la acción concecuente a esta toma de la
muestra.
  switch (range) { case 0:
  Serial.println(""); break; case 1:
  Serial.println("---"); break; case 2:
  Serial.println("------"); break; case 3:
  Serial.println("---------"); break; case 4:
```

```
  Serial.println("------------"); break; case 5:
  Serial.println("--------------|-"); break; case 6:
  Serial.println("--------------|---"); break; case 7:
  Serial.println("--------------|-------"); break; case 8:
  Serial.println("--------------|----------"); break; case
9:
  Serial.println("--------------|----------------");
break; case 10:
  Serial.println("--------------|-------------------");
break; case 11:
  Serial.println("--------------|----------------------
"); break;
  }
  }
```

Código que presenta los tiempos de muestreo e interrupción de la señal entrante del sensor:

```
  volatile int rate[10]; volatile unsigned long
sampleCounter = 0; volatile unsigned long lastBeatTime =
0; volatile int P =512; volatile int T = 512; volatile int
thresh = 525; volatile boolean firstBeat = true; volatile
boolean secondBeat = false; void interruptSetup(){
  //inicialización del timer 2 y configuración del mismo
para funcionar a intervalos de 2 mS TCCR2A = 0x02;
  TCCR2B = 0x06; OCR2A = 0X7C; TIMSK2 = 0x02; sei(); }
  ISR(TIMER2_COMPA_vect){ cli();
  Signal = analogRead(pulsePin); sampleCounter +=10; int N
= sampleCounter - lastBeatTime; if(Signal < thresh && N >
(IBI/5)*3){
  if (Signal < T){
  T = Signal;
  } }
  if(Signal > thresh && Signal > P){
  P = Signal;
  } if (N > 250){ if ( (Signal > thresh) && (Pulse == false)
&& (N > (IBI/5)*3) ){ Pulse = true;
  digitalWrite(blinkPin,HIGH); IBI = sampleCounter -
lastBeatTime; lastBeatTime = sampleCounter;
if(secondBeat){ secondBeat = false; for(int i=0; i<=9;
i++){ rate[i] = IBI;
  } } if(firstBeat){ firstBeat = false; secondBeat = true;
sei(); return; }
  word runningTotal = 0; for(int i=0; i<=8; i++){ rate[i]
= rate[i+1]; runningTotal += rate[i];
  } rate[9] = IBI; runningTotal += rate[9]; runningTotal
/= 10;
```

```
  BPM = 60000/runningTotal;
  QS = true;
  } }
  if (Signal < thresh && Pulse == true){
  digitalWrite(blinkPin,LOW); Pulse = false; amp = P - T;
thresh = amp/2 + T;
  P = thresh;
  T = thresh; } if (N > 2500){ thresh = 512 P = 512; T =
512;
  lastBeatTime = sampleCounter; firstBeat = true;
secondBeat = false;
  } sei();
  }// end isr
```

Conexión de tarjeta Arduino a Raspberry Pi 3: Una vez capturados los datos en la tarjeta Arduino es necesario que estos sean recibidos por la tarjeta Raspberry Pi 3 que hará la labor de conectarlos al bróker MQTT que posteriormente será implementado en la nube mediante Microsoft Azure.

```
  Conexión lógica: En la raspberry Pi 3 se ejecuta la
siguiente línea de comandos que permiten esta conexión.
  import serial // importa fichero de configuración serial
para conexión con dispositivo
  import time // importa archivo para modificar tiempo y
hora en los archivos de Python para realizar la
sincronización entre los dispositivos arduino
  = serial.Serial( '/dev/ttyACM0',baudrate =9600)
//Especifica el puerto serial por el cual escucha el
dispositivo Raspberry Pi 3.y la tasa de baudios para la
sincronización entre el origen y destino arduino.
  setDTR(False)
  time.sleep( 1)
  arduino.flushInput()
  arduino.setDTR(True) //configuración de parámetros para
la conexión entre la tarjeta // generación de loop infinito
para la captura de datos del sensor de temperatura y
recibidos por la tarjeta Arduino
  while True: comando
  = arduino.readline() //lectura de datos
  print(comando) //impresión de los datos al usuario (puede
ser local o como posteriormente se hará ,ser enviados
arduino.
  close()
```

Instalación y configuración de bróker Mosquitto en Microsoft Azure: El siguiente paso para el proceso descrito en este informe es la instalación de la máquina virtual

Ubuntu server 14.04 LTS sobre el cual se instalará el broker que permite la comunicación entre un publisher y un suscriber.

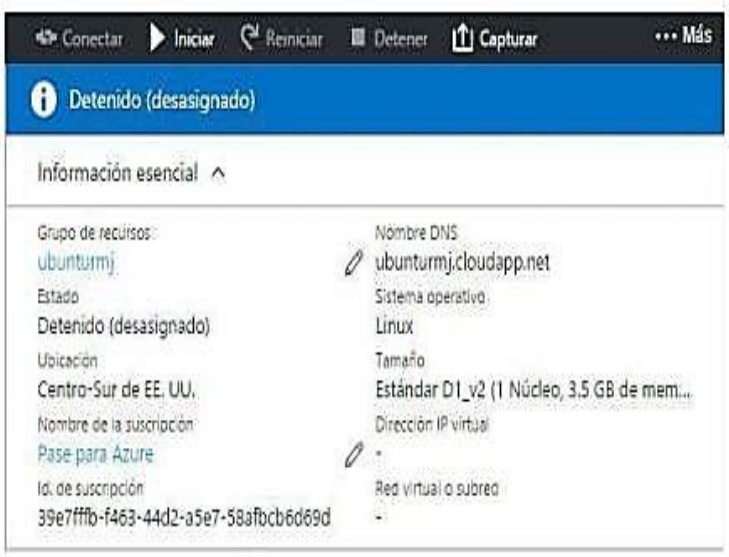

Fig. 2. Información general de máquina virtual Ubuntu server 14.04 LTS

En la tabla 2 se presenta la información relacionada con el servidor virtual Ubuntu 14.04 LTS comprado e instalado en Azure.

Tabla 2. Características generales de servidor virtual clásico instalado.

Elemento	Característica
Dirección IP	Dinámica
Puertos habilitados	1883 TCP para MQTT 22 TCP para SSH
Núcleos	1
Memoria	3.5 GB
Ubicación del servidor	Centro-Sur de EEUU
Dirección DNS	Ubunturmj.cloudapp.net

Instalación de Broker Mosquitto: Para la instalación del broker mosquitto sobre la máquina virtual de Ubuntu server 14.04 LTS en la terminal (usando Putty) del servidor.

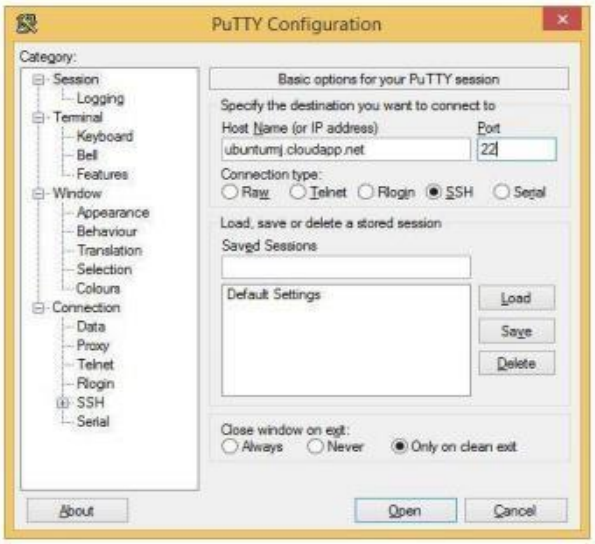

Fig. 3. Información general de máquina virtual Ubuntu server 14.04 LTS

En la figura 3 se presenta el método de conexión usando Putty;ssh permite el establecimiento seguro de la conexión entre el servidor Ubuntu y la terminal de configuración, al generar encriptación de los datos enviados entre los dos miembros de la comunicación. Luego de esto se escriben los siguientes comandos.

```
sudo wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
sudo apt-key add mosquitto-repo.gpg.key
cd /etc/apt/sources.list.d/
sudo wget http://repo.mosquitto.org/debian/mosquitto-wheezy.list
sudo apt-get update
sudo apt-get install mosquitto
```

Fig. 4. Instalación Mqtt Client

En el proceso de instalación mostrado en la figura 4 se hace uso del repositorio mosquitto Wheezy y utiliza el método de llave pública para el establecimiento seguro de la conexión. La instalación del cliente MQTT se lleva a cabo en la máquina virtual creada en Windows azure (ubuntu server 14.04 por medio de putty) y en la raspberry, permitiendo así la publicación de los mensajes de temperatura en un tópico definido en este caso es llamado “temp” su visualización puede puede verse por los clientes inscritos al tópico. Si bien la librería paho se enfoca en proporcionar un cliente que relaciona las aplicaciones para conectar a el broker MQTT con el fin de publicar los mensajes se presentaron algunos problemas en su instalación por lo tanto no se trabajó con este para esta práctica, ya que esta librería proporciona una ayuda para hacer más

factible la comunicación no significa que se esté ligado obligatoriamente a trabajar con esta.

Configuración de puerto para comunicación de Raspberry Pi 3 con el broker Mosquitto: Para permitir el acceso tanto al publisher como al subscriber al broker Mosquitto se habilita el puerto 1883 de la máquina virtual Ubuntu server 14.04 LTS

Fig. 5. Configuración de puerto en máquina virtual

La figura 5 muestra el estado activo del puerto 1883 que es usado por defecto por Mosquitto para la comunicación entre los publisher y subscriber de este broker. Esta configuración a nivel de seguridad genera falencias debido a que cualquier atacante tiene vía libre al acceso por este, generando serios problemas de confidencialidad de los miembros del bróker.

Implementación código MQTT para la publicación de información en base al DNS: La dirección IP y el puerto usado de la máquina virtual que almacena al bróker especificando el nombre del tópico donde se quiere publicar la información. El siguiente código es uno de los métodos usados para la conexión entre la Raspberry y el broker, Paho también realiza este proceso con la diferencia de que en este caso se ejecuta directamente sobre:

```
  mosquitto sin usar agentes externos.
  import mosquitto, os, urlparse # Define event callbacks
def on_connect(mosq, obj, rc):
  print("rc: " + str(rc)) def on_message(mosq, obj, msg):
  print(msg.topic + " " + str(msg.qos) + " " +
str(msg.payload)) def on_publish(mosq, obj, mid):
  print("mid: " + str(mid)) def on_subscribe(mosq, obj,
mid, granted_qos):
  print("Subscribed: " + str(mid) + " " + str(granted_qos))
def on_log(mosq, obj, level, string):
  print(string) mqttc = mosquitto.Mosquitto() # Assign
event callbacks mqttc.on_message = on_message
mqttc.on_connect = on_connect mqttc.on_publish =
on_publish mqttc.on_subscribe = on_subscribe # Uncomment
to enable debug messages #mqttc.on_log = on_log
```

```
  # Parse CLOUDMQTT_URL (or fallback to localhost)
  url_str = os.environ.get('ubunturmj.cloudapp.net:1883',
'mqtt://40.12.15.60:1883') //Se especifican el del DNS y
dirección IP con el puerto de salida url =
urlparse.urlparse(url_str) # Connect
  mqttc.username_pw_set(url.username, url.password)
mqttc.connect(url.hostname, url.port) # Start subscribe,
with QoS level 0
  mqttc.subscribe("temp", 0) //Suscripción al tópico temp
  # Publish a message
  mqttc.publish("temp", "HOLA_MUNDO") //Se publica en el
tópico temp el mensaje
  HOLA_MUNDO
  # Continue the network loop, exit when an error occurs
rc = 0 while rc == 0: rc = mqttc.loop() print("rc: " +
str(rc))
```

Publicación de los valores de temperatura en el bróker: Tomando como base el código anterior se implementa en este el código que permite la conexión lógica entre los valores en el Arduino para la Raspberry

```
  import mosquitto, os, urlparse def on_connect(mosq, obj,
rc):
  print("rc: " + str(rc)) def on_message(mosq, obj, msg):
  print(msg.topic + " " + str(msg.qos) + " " +
str(msg.payload)) def on_publish(mosq, obj, mid):
  print("mid: " + str(mid)) def on_subscribe(mosq, obj,
mid, granted_qos):
  print("Subscribed: " + str(mid) + " " + str(granted_qos))
def on_log(mosq, obj, level, string):
  print(string) mqttc = mosquitto.Mosquitto()
mqttc.on_message = on_message mqttc.on_connect =
on_connect mqttc.on_publish = on_publish
mqttc.on_subscribe = on_subscribe url_str =
os.environ.get('ubunturmj.cloudapp.net:1883',
'mqtt://40.124.15.60:1883') url =
urlparse.urlparse(url_str) mqttc.connect(url.hostname,
url.port) mqttc.subscribe("temp", 0) import serial import
time arduino = serial.Serial('/dev/ttyACM0',baudrate=9600)
arduino.setDTR(False) time.sleep(1) arduino.flushInput()
arduino.setDTR(True)
  //ciclo para la publicación en el broker en el tópico
“temp” while True: comando = arduino.readline()
print(comando) mqttc.publish("temp", comando + " ")
arduino.close() rc = 0 while rc == 0:
  rc = mqttc.loop() print("rc: " + str(rc))
```

En este se implementan los dos códigos de tal manera que el ciclo que se presenta para la lectura del valor de la temperatura se realice con la publicación en el tópico determinado hacia el Broker, en este caso la variable “comando” almacena el valor de temperatura por ende este es el valor que se publica continuamente.

Pruebas de la implementación broker y presentación del envío de datos de un publisher a un subscriber: Tras generar los procesos anteriormente descritos en este informe, el último proceso a ejecutar son las pruebas de funcionamiento, para estas se utiliza el complemento de Google Chrome Mqtt Lens que actuará como subscriber al broker Mosquitto configurado en la máquina virtual.

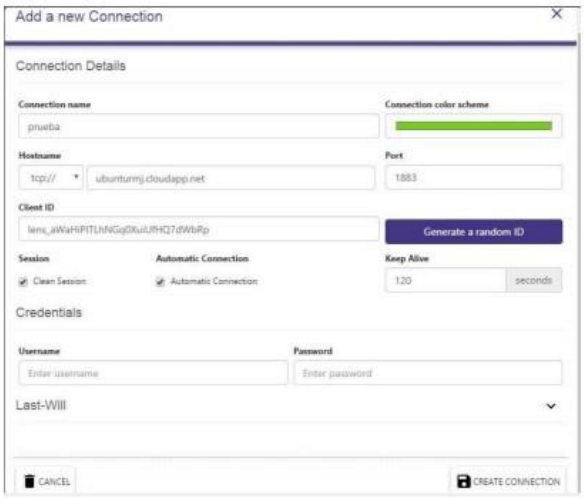

Fig. 6. Parámetros de configuración de la conexión del subscriber al bróker

La figura 6 presenta los parámetros de configuración de la conexión del subscriber al broker, dentro de la ventana se ingresa la URL o dirección DNS del servidor ubuntu 14.04 LTS alojado en Azure, el puerto 1883 usado para MQTT y el nombre de la conexión. Una vez puesto los valores correctos, se realiza el establecimiento de la conexión.

4.3 Pruebas de funcionamiento del código python del sensor de pulso

La información se transmite al bróker en intervalos de 2 mm entre muestra y muestra como se observa en la figura 9. Es importante mencionar que los datos están fuera de rango ya que surgió un problema técnico con los sensores y por esta razón la información no se puede utilizar para un previo análisis. Posteriormente se puede ver un menú principal que cuenta con tres opciones:

1. Sensor de temperatura
2. Sensor de pulso
3. Salir

Para realizar un análisis eficiente es necesario tomar un rango considerable de muestras el cual permita obtener información para lograr concluir en este caso acerca de los parámetros clínicos de un paciente con la finalidad de diagnosticar cualquier tipo de anomalía del paciente, este análisis es mucho más fácil de concluir si se puede visualizar a través de gráficos.

Pruebas de funcionamiento del sensor de temperatura: Por medio de la aplicación de MQTT lens podemos verificar la conexión cuando se reciben los mensajes, en la siguiente imagen se puede ver que se está subscrito al tópico de temp y los datos que están siendo recibidos del sensor, nuevamente se ve el rango erróneo de los datos debido al daño que se proporcionó en los sensores.

Implementación de la aplicación Cliente Java: Por medio del dominio del bróker denominado ubunturmj.cloudapp.net se lograr hacer la conexión con la aplicación de Java, es necesario crear dos tópicos que corresponden a los dos sensores uno de temperatura y el otro de pulso. Para lograr la comunicación es preciso subscribirse al tópico temp o pulso, dando click sobre la que se desee, esto depende de los requerimientos del usuario. En la segunda imagen se puede ver que ya se ha hecho el proceso de subscripción y también se puede verificar la comunicación bidireccional debido a los datos recibidos mostrados en pantalla.

Código de conexión al bróker MQTT por parte del cliente Java: Es importante agregar la librería org.eclipse.paho.client.mqttv3 ya que esta contiene un conjuntos de clases, las cuales poseen una seria de métodos y atributos que facilitan las operaciones ya que permiten reutilizar código y se puede hacer uso de dichos métodos, clases y atributos que componen a la librería, con la finalidad de evitar que el usuario tenga que implementar las funcionalidades. A continuación, el código que se adaptó.

```
  package iot_umng; import java.io.BufferedWriter; import
java.io.File;     import     java.io.FileWriter;     import
java.io.PrintWriter;    import    java.util.Date;    import
javax.swing.JOptionPane;                                  import
org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;  import
org.eclipse.paho.client.mqttv3.MqttCallback;        import
org.eclipse.paho.client.mqttv3.MqttClient;          import
org.eclipse.paho.client.mqttv3.MqttConnectOptions;  import
org.eclipse.paho.client.mqttv3.MqttException;       import
org.eclipse.paho.client.mqttv3.MqttMessage;         import
org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;
public class conec {

  public void pulso(String m){

  String topic = "pulso"; String content = m; int qos = 1;
  String broker = "tcp://ubunturmj.cloudapp.net:1883";
  String clientId = "pc";
  MemoryPersistence persistence = new MemoryPersistence();
try {
```

```
  MqttClient mqttClient = new MqttClient(broker, clientId,
persistence); mqttClient.setCallback(new MqttCallback() {
public void messageArrived(String topic, MqttMessage msg)
  throws Exception {
  System.out.println("Recived:" + topic);
  System.out.println("Recived:"           +           new
String(msg.getPayload()));
  String m=new String(msg.getPayload());

  iotwind.mpulso.setText(m);
  // text.setText(new String(msg.getPayload()));
  }
  public void deliveryComplete(IMqttDeliveryToken arg0) {
  System.out.println("Delivary complete");
  }
  public void connectionLost(Throwable arg0) {
  }
  });
  MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setCleanSession(true);
  // connOpts.setUserName("pc");
  //    connOpts.setPassword(new    char[]{'p',    'c'});
mqttClient.connect(connOpts);
  MqttMessage           message           =           new
MqttMessage(content.getBytes()); message.setQos(qos);
  System.out.println("Publish  message:  "  +  message);
mqttClient.subscribe(topic,                          qos);
mqttClient.publish(topic, message);
  } catch(MqttException me) {
  System.out.println("reason "+me.getReasonCode());
  System.out.println("msg "+me.getMessage());
  System.out.println("loc "+me.getLocalizedMessage());
  System.out.println("cause             "+me.getCause());
System.out.println("excep "+me); me.printStackTrace();
  JOptionPane.showMessageDialog(null,"Sin    conexión    a
internet ");
  }
  }
   public void temp(String m){

  String topic = "temp"; String content = m; int qos = 1;
  String broker = "tcp://ubunturmj.cloudapp.net:1883";
  String clientId = "pc";
  MemoryPersistence persistence = new MemoryPersistence();
try {
  MqttClient mqttClient = new MqttClient(broker, clientId,
persistence); mqttClient.setCallback(new MqttCallback() {
```

```
public void messageArrived(String topic, MqttMessage msg)
throws Exception {
  System.out.println("Recived:" + topic);
  System.out.println("Recived:"           +           new
String(msg.getPayload()));
  String m=new String(msg.getPayload());

  iotwind.mtemp.setText(m);
   }
  public void deliveryComplete(IMqttDeliveryToken arg0) {
  System.out.println("Delivary complete");
  }
  public void connectionLost(Throwable arg0) {
  }
  });
  MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setCleanSession(true);
  // connOpts.setUserName("pc");
  // connOpts.setPassword(new char[]{'p', 'c'});
  mqttClient.connect(connOpts);
  MqttMessage           message           =           new
MqttMessage(content.getBytes()); message.setQos(qos);
  System.out.println("Publish  message:  "  +  message);
mqttClient.subscribe(topic,                          qos);
mqttClient.publish(topic, message);
  } catch(MqttException me) {
  System.out.println("reason "+me.getReasonCode());
  System.out.println("msg "+me.getMessage());
  System.out.println("loc "+me.getLocalizedMessage());
  System.out.println("cause             "+me.getCause());
System.out.println("excep "+me);
  me.printStackTrace();
  JOptionPane.showMessageDialog(null,"Sin   conexión   a
internet ");
  }
  }
  }
```

Código para llamar bloques: Esta parte de la interfaz es sencilla pues consta de implementar dos botones que corresponden a los sensores.

```
  package iot_umng; public class Iot_umng { public static
void main(String[] args) {

  iotwind            iotwindow=new            iotwind();
iotwindow.setVisible(true);
  iotwindow.setLocationRelativeTo(null);
```

```
    }
}
```

Interfaz en Java

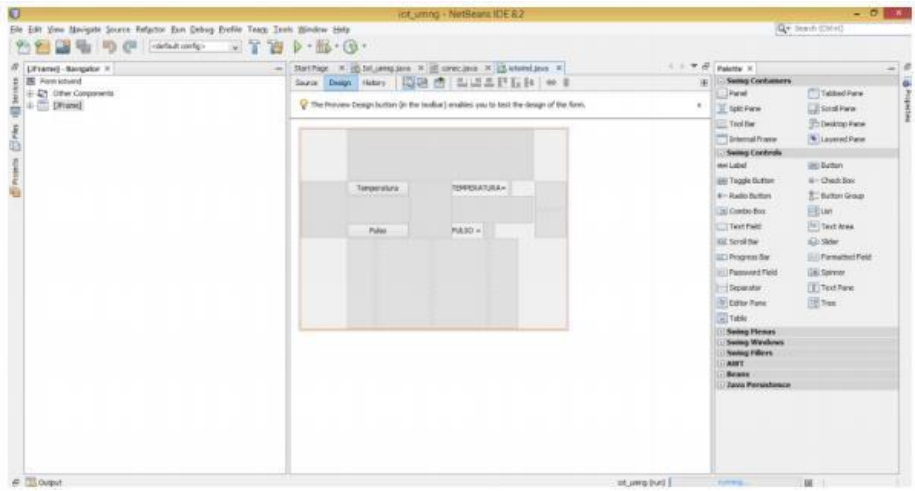

Fig. 7. Diseño de la aplicación cliente Java

Implementación de Seguridad: En este instante del proceso ya se ha realizado la configuración adecuada, por otro lado el bróker debe estar ejecutándose preferiblemente sin ningún tipo de seguridad por el puerto 1883, si no se tiene seguridad en esto igualmente se puede comprobar con el comando:

Sudo service mosquitto status

Posteriormente se debe visualizar que 643 es el ID del proceso que es aleatorio para cada máquina y también cuando se reinicie el servicio. Gracias al siguiente comando se genera un archivo que al ejecutarse genera los correspondientes certificados y claves para el bróker:

Wget:http://github.com/ownntracks/tools/raw/master/TLS/generateCa.shbash/geneate-CA.sh

Luego de que se ejecuta el anterior comando se generan 6 archivos con extensiones de:

.crt que son certificados
.key que son las claves
.csr de las solicitudes
.srl para el proceso de firma

Los archivos MQTT tendrán el nombre de la máquina y serán generados automáticamente. Dichos archivos deberán guardarse en el directorio de mosquitto.

Dentro de los archivos de configuración del mosquitto es necesario agregar las siguientes líneas de comando para la configuración del puerto 8883 puerto que soporta

TLS, la adjudicación de los certificados que se generan y la configuración de seguridad sobre el bróker para esto, se hace lo siguiente:

Se copian los archivos de certificados digitales generados en apartes anteriores, a las carpetas de mosquitto

```
  ca_certificates       y     certs       sudo  cp
  ca.crt /etc/mosquitto/ca_certificates/ sudo cp MQTT.crt
MQTT.key /etc/mosquitto/certs/
```

Los archivos MQTT.crt y MQTT.key para futuras implementaciones son renombrados con el nombre de la máquina que se realizó para la implementación Luego de esto se es necesario la configuración del puerto y certificados ,para esto se ingresa a la carpeta mosquitto.conf

```
cd /etc/mosquitto/mosquitto.conf.
```

Dentro de esta carpeta incluir

```
  listener 8883 pid_file /var/run/mosquitto.pid
  persistence true persistence_location
  /var/lib/mosquitto/ log_dest file
  /var/log/mosquitto/mosquitto.log cafile
  /etc/mosquitto/ca_certificates/ca.crt certfile
  /etc/mosquitto/certs/MQTT.crt keyfile
  /etc/mosquitto/certs/MQTT.key require_certificate
  true
```

Tras esto reiniciar el servidor mosquitto para guardar los cambios ejecutados Sudo service mosquitto status. Para lograr la comunicación con los clientes es necesario que estos obtengan también sus certificados, para este proceso es necesario ejecutar los siguientes comandos:

```
  Openssl genrsa –out client.key 2048
  Openssl  req  –new  –out  client.csr\-key  client.key  –
subj”/CN=client/O=example.com”
  Openssl x509 –req –in client.csr –CA ca.crt\-CAkey ca.key
–Caserial -/ca.srl –out client.crt\- days 3650 –addtrust
clientAuth
```

Los certificados y las claves del cliente deben ser instalados en este, para lograr una comunicación segura del cliente con el bróker.

5 Conclusiones

- Todos los códigos fueron validados y son completamente reproducibles

- Las aplicaciones para IoT son cada día más amplias y con este trabajo se puede iniciar en su implementación.

Agradecimientos. Los autores agradecen a la Universidad Militar Nueva Granada por el soporte con los laboratorios necesarios para los procesos experimentales proyecto código INV-ING-2365 y al ingeniero Harold Hoyos por las pruebas en algunos procedimientos del Broker en el servidor del Grupo de Investigación GISSIC de la Universidad Militar Nueva Granada.

Referencias

1. Código Fuente: http://www.cisco.com/c/es_mx/solutions/internet-of-things/overview.html. Fecha de consulta:
2. Código Fuente: https://www.nociones.de/introduccion-paho-mqtt-iot/
3. Código Fuente: http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf
4. Texas Instruments, (2014), Ficha técnica sensor, Texas Instruments [en linea], Disponible en: http://www.ti.com/lit/ds/symlink/lm35.pdf
5. Daniel Gallardo García, Apuntes de arduino nivel pardillo, [en linea], disponible en: http://educacionadistancia.juntadeandalucia.es/profesorado/pluginfile.php/2882/mod_resource/content/1/Apuntes_ARDUINO_nivel_PARDILLO.pdf
6. Ross M Sheldon, Probabilidad y Estadistica para Ingenieros (2001), Segunda edición,McGrawHill SwitchDoc, Tutorial: IOT / Installing and Testing Mosquitto MQTT on the Raspberry Pi Disponible en: http://www.switchdoc.com/2016/02/tutorial-installing-and-testing mosquitto-mqtt-on-raspberry-pi/
7. CloudMQTT Documentation - Python
8. Ramirez L, Guillen E and Ramos C. Effective Validation Model and Use of Mobile-Health Applications for the Elderly Healthc Inform Res. 2018;24(4):276-282. DOI: https://doi.org/10.4258/hir.2018.24.4.276