Internship Report

Building campus wide heterogeneous Wireless networks :An MQTT implementation

Author: T Pratik

Submission date: July 15, 2017

Contents

List of Figures

1 Introduction

Internet of things(IoT) is expected to offer advanced connectivity of devices, systems, and services that covers a variety of protocols, domains, and applications for technologies such as smart grids, virtual power plants, smart homes, intelligent transportation and smart cities.

The Internet of things (IoT) is the inter-networking of physical devices, vehicles (also referred to as *connected devices* and *smart devices*), buildings, and other items embedded with electronics, software, sensors, actuators, and network connectivity which enable these objects to collect and exchange data.Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure.The interconnection of these embedded devices (including smart objects), is expected to usher in automation in nearly all fields, while also enabling advanced applications like a smart grid, and expanding to areas such as smart cities. “Things”, in the IoT sense, can refer to a wide variety of devices such as heart monitoring implants, biochip transponders on farm animals, electric clams in coastal waters, automobiles with built-in sensors, DNA analysis devices for environmental/food/pathogen monitoring, or field operation devices that assist firefighters in search and rescue operations. Legal scholars suggest regard “Things” as an “inextricable mixture of hardware, software, data and service” [14].

Wireless sensor networks (WSN), sometimes called wireless sensor and actuator networks (WSAN), are spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, pressure, etc. and to cooperatively pass their data through the network to a main location.The more modern networks are bi-directional, also enabling control of sensor activity. The development of wireless sensor networks was motivated by military applications such as battlefield surveillance; today such networks are used in many industrial and consumer applications, such as industrial process monitoring and control, machine health monitoring, and so on.

MQTT (MQ Telemetry Transport or Message Queue Telemetry Transport) is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based “lightweight” messaging protocol for use on top of the TCP/IP protocol. It is designed for connections with remote locations where a “small code footprint” is required or the network bandwidth is limited. The publish-subscribe messaging pattern requires a message broker. The broker is responsible for distributing messages to interested clients based on the topic of a message [2].

2 Goal of the project

Try and build a working autonomous wireless sensor network for envioronment monitoring, with different hardwares that can communicate with eachother using MQTT protocol.

3 Building a heterogeneous MQTT based sensor network

Table 1: overview of nodes

u-controller	radio-type	No. of such endnodes
NodeMCU	WiFi(inbuilt ESP-12e)	2
Arduino Nano	WiFi(ESP8266)	2
Waspmotes	XBee S1(IEEE-802.15.4)	2
Wi-Sense	CC2520(ZigBee,IEEE-802.15.4)	2

MQTT (MQ Telemetry Transport or Message Queue Telemetry Transport) is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based "lightweight" messaging protocol for use on top of the TCP/IP protocol

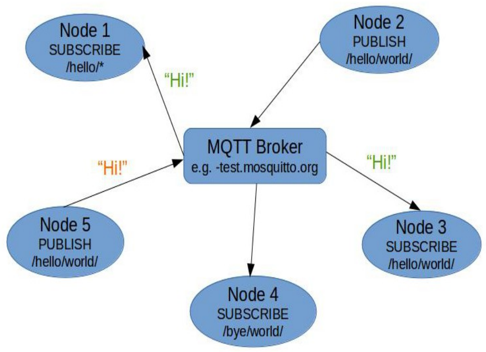


Figure 1: block diagram of MQTT

MQTT defines methods (sometimes referred to as verbs) to indicate the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server [1] [2].

Connect: Waits for a connection to be established with the server.

Disconnect: Waits for the MQTT client to finish any work it must do, and for the TCP/IP session to disconnect.

Subscribe: Waits for completion of the Subscribe or UnSubscribe method.

UnSubscribe: Requests the server unsubscribe the client from one or more topics.

Publish: Returns immediately to the application thread after passing the request to the MQTT client.

3.1 Modules of MQTT

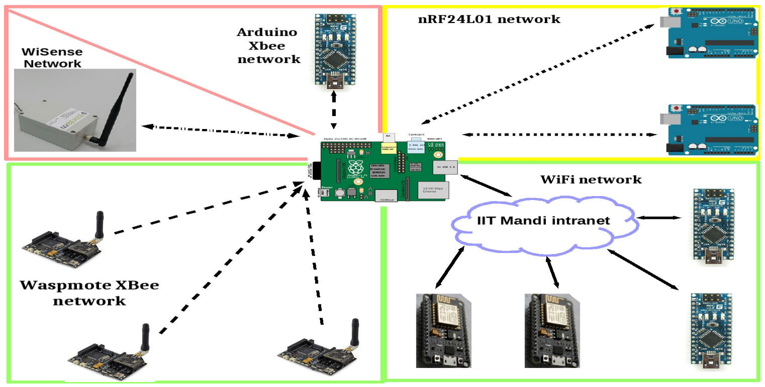


Figure 2: Diagram of the Network Structure

3.1.1 Getting started with MQTT

MQTT is a device to device communication protocol using publish-subscribe model, consists of two main parts “Broker” and “Clients”. The Broker hosts an MQTT server and a Client may be a sensor or an actuator . A client containing data publishes to the broker under a topic name, and a client interessted in the data subscribes to the same topic name to get the data.

A raspberry-Pi(Rpi) is a microprocessor embedded prototyping device. It has many versions, in this project we have used Rpi-3 ,which is powered by a 1.2GHz ARM Cortex A5 processor ,has 1GB RAM, has inbuit Bluetooth & WiFi, 4 USB ports and it can run a variety of operating systems including Windows and Linux. Hence, it can be considered as a small computer. We installed Raspbian Jessie, which is a flavour of Linux [12] [13].

A Raspberry-pi was made the broker by installing mosquitto broker, which i a free MQTT broker for research purpose. The broker always runs in the background and is started automatically on startup.

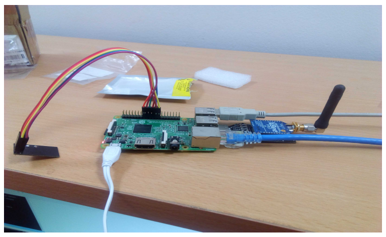


Figure 3: The Raspberry-Pi which runs the Broker

3.2 MQTT on waspmote network:

As mentioned earlier the waspmotes were sending data to the coordinator, the coordinator was attached to the Raspberry-pi and through serial communication, the data sent by the nodes were read.

A python MQTT-forwarder script was written ,which read the incoming packets and forwarded them to the broker under respective topic names.

3.3 WiFi Module

3.3.1 MQTT with Arduino :

For implementing MQTT with Arduino we chose Arduino Nano due to its small size and less power consumption, in comparsion to Arduino UNO. For connectivity we attached ESP8266 [3] to them for WiFi capabilty. Using PubSubClient [8] library which is a light weight MQTT library for Arduino, the arduinos were able to directly publish their data to the broker in SenML [7] format, and the were able to publish their data from anywhere within the campus wherever “IITMandiWiFi” was available, and a node kept at “IITMandi South campus” can also publish data to the broker kept at “IITMandi North campus”.

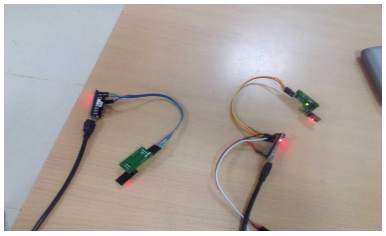


Figure 4: Arduino-Nanos with ESP8266 implementing MQTT

3.3.2 MQTT with NODE-MCU :

Just to add a variety ,two NODE-MCUs were also programmed to publish their data to the broker .They can also publish their data from anywhere within thw campus wherever “IITMandiWiFi” is available.

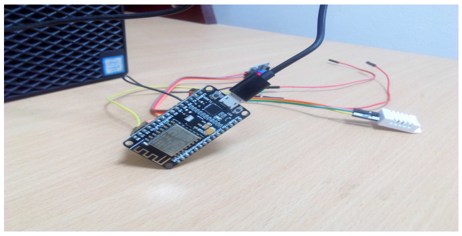
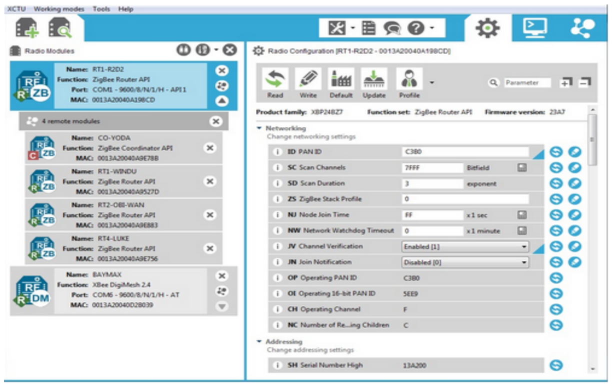


Figure 5: NodeMCU implementing MQTT

3.4 XBee

An arduino Nano along with an XBee S2 radio running 802.15.4 firmware, was programmed with proper configuration to add it to the network of waspmotes [6], since it also sent sensor data to the same coordinator to which waspmote node were sending, it can be said that the Arduino is also now a part of the waspmote network, hence we achieved interoperability with devices manufactured from multiple manufacturers.

First ,the XBee radio was configured with XCTU [11] to set the Channel number, Network PanID ,API mode and Endpoint point, then it was directly attached to the arduino for serial communication using Arduino XBee library.



Name: RT1-R2D2
Function: ZigBee Router API
Port: COM1 - 9600/8/N/1/H - API1
MAC: 0013A20040A198CD

Name: CO-YODA
Function: ZigBee Coordinator API
MAC: 0013A20040A9E78B

Name: RT1-WINDU
Function: ZigBee Router API
MAC: 0013A20040A9527D

Name: RT2-OBI-WAN
Function: ZigBee Router API
MAC: 0013A20040A9E883

Name: RT4-LUKE
Function: ZigBee Router API
MAC: 0013A20040A9E756

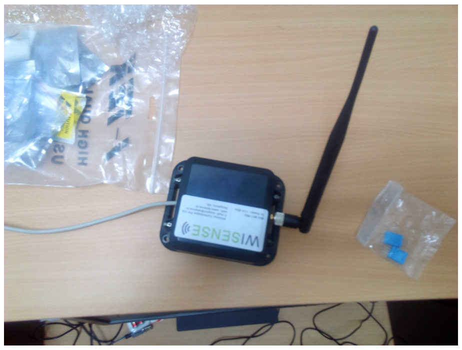
Name: BAYMAX
Function: XBee DigiMesh 2.4
Port: COM6 - 9600/8/N/1/H - AT
MAC: 0013A20040D2B039

Radio Configuration [RT1-R2D2 - 0013A20040A198CD]

Product family: XBP24BZ7 Function set: ZigBee Router API Firmware version: 23A7

ID PAN ID	C3B0	
SC Scan Channels	7FFF	Bitfield
SD Scan Duration	3	exponent
ZS ZigBee Stack Profile	0	
NJ Node Join Time	FF	x 1 sec
NW Network Watchdog Timeout	0	x 1 minute
JV Channel Verification	Enabled [1]	
JN Join Notification	Disabled [0]	
OP Operating PAN ID	C3B0	
OI Operating 16-bit PAN ID	5EE9	
CH Operating Channel	F	
NC Number of Re...ing Children	C	
SH Serial Number High	13A200	

Figure 6: An XCTU interface


(a) Wi-Sense co-ordinator

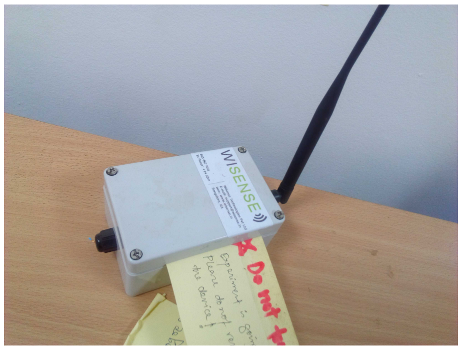

(b) Wi-Sense endnode

Figure 7: Wi-Sense modules

3.5 Wisense

The pressure sensor on a wisense sensor node which was turned off by default , was turned on and it was made a reduced functional device(RFD) ie. Capable of star networking.

Note: wisense nodes are programmed using embedded C, and one must be comfortable with "Code Composer Studio" before Starting to program these nodes otherwise they may stop working.

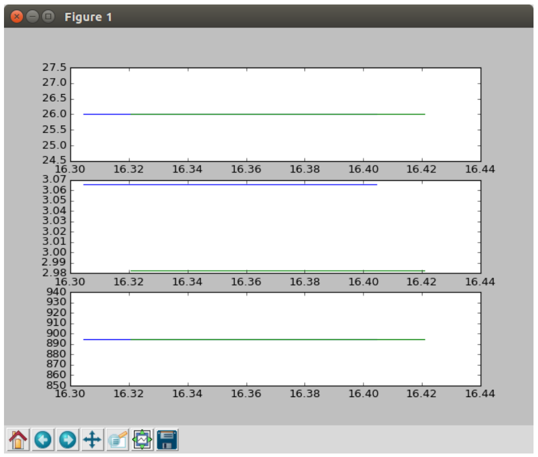
4 Visualization

The aim was to take live sensor readings(from wisense nodes), and plot them graphically as and when the are updated. First a shell script was run which extracted raw data from wisense coordinator ,preprocessed them and wrote them into a ".csv" file.

Second, two python scripts were written to plot the data:

4.1 Offline Liveplot

A local liveplot was created using matplotlib [5], which can be used within the IITMandi campus to view live updates, which uses subgraphs and the common attributes from different sensors are plotted on the same subgaph graph.**For Example:** "Temperature" readings from sensor node-1 and node-2 are plotted on the same subgraph with different colours to make comparision easier.


(a) Offline-Liveplot

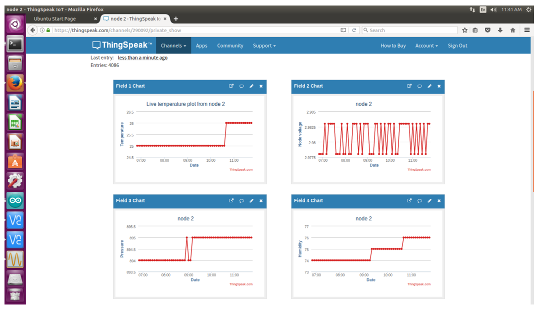

(b) Online-Liveplot

Figure 8: Visualisations

4.2 Online Liveplot

In the first, whenever a new row of data was written into the ".csv" file, it was sent to thingspeak channel(a cloud service), where graphs were plotted automatically. This can be viewed by anyone who is given the read API keys by the channel owner(Dr.Siddhartha Sarma).

Note : Due the proxy settings, initially we were unable to reach the thingspeak server, after adding some lines of code to configure proxy settings, we successfully posted the data to the thingspeak server. Below is the code for setting proxy

```
import urllib2
proxy = urllib2.ProxyHandler({'http':'10.8.0.1','https':'10.8.0.1'})
opener = urllib2.build_opener(proxy)
urllib2.install_opener(opener)
```

5 Integration,Experimental evaluation and Result

The network conists of a central broker(Raspberry-Pi) and 6 clients.

2 waspmotes which use forwarder to publish their data ,2 Arduino Nano with ESP-8266 WiFi radios which publish directly to the broker ,2 Node-MCUs which also publish directly to the broker.

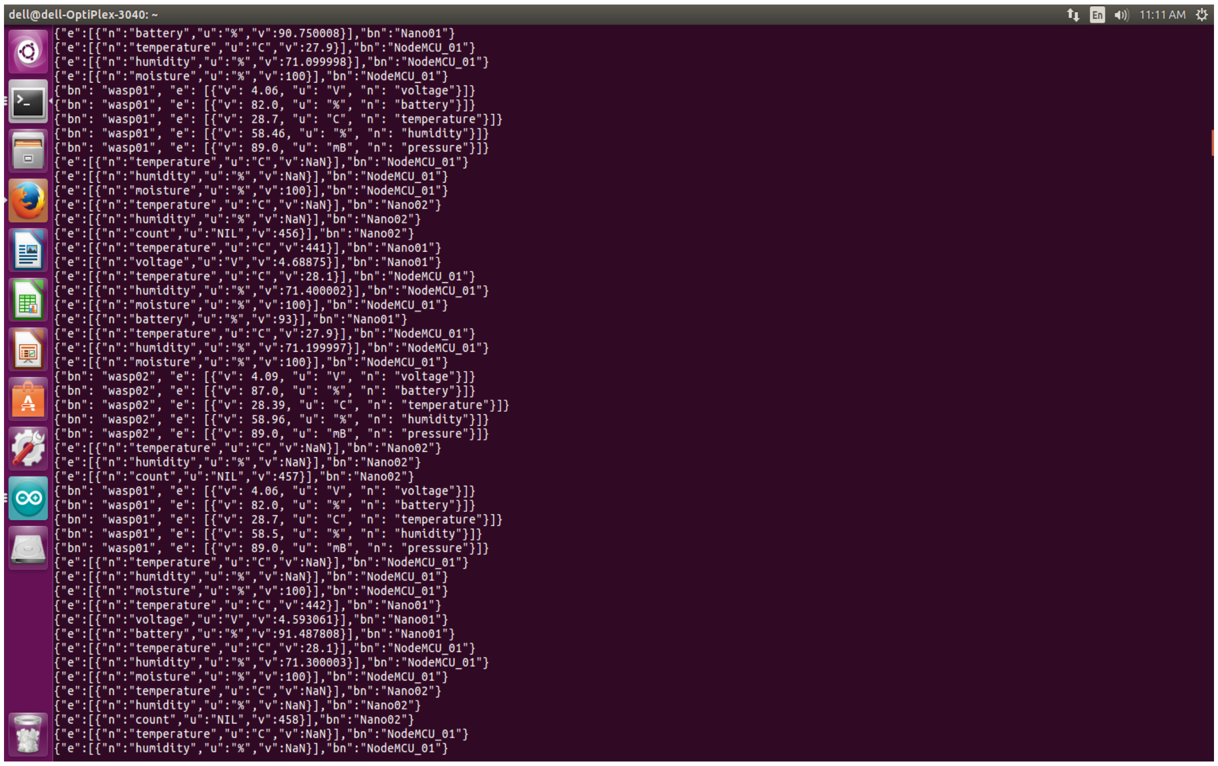
Each of their payload consists of SenML-JSON [7]arduinojson format, as following :

{'bn': 'location/name of node';'e':['n':'parameter name','u':'Parameter unit','v':'value']}

from which **topic** name can be extracted as : String(value(bn) + value(parameter))

Note: Currently substitute “IITMandi/COMLab” in place of “location”

By installing MQTT client software [9](we tested with paho mqtt client), connecting to the Broker and subscribing to this **topic** will give you the live readings of the sensors.



```
dell@dell-OptiPlex-3040: ~
{"e":[{"n":"battery","u":"%","v":90.750008}],"bn":"Nano01"}
{"e":[{"n":"temperature","u":"C","v":27.9}],"bn":"NodeMCU_01"}
{"e":[{"n":"humidity","u":"%","v":71.099998}],"bn":"NodeMCU_01"}
{"e":[{"n":"moisture","u":"%","v":100}],"bn":"NodeMCU_01"}
{"bn": "wasp01", "e": [{"v": 4.06, "u": "V", "n": "voltage"}]}
{"bn": "wasp01", "e": [{"v": 82.0, "u": "%", "n": "battery"}]}
{"bn": "wasp01", "e": [{"v": 28.7, "u": "C", "n": "temperature"}]}
{"bn": "wasp01", "e": [{"v": 58.46, "u": "%", "n": "humidity"}]}
{"bn": "wasp01", "e": [{"v": 89.0, "u": "mB", "n": "pressure"}]}
{"e":[{"n":"temperature","u":"C","v":NaN}],"bn":"NodeMCU_01"}
{"e":[{"n":"humidity","u":"%","v":NaN}],"bn":"NodeMCU_01"}
{"e":[{"n":"moisture","u":"%","v":100}],"bn":"NodeMCU_01"}
{"e":[{"n":"temperature","u":"C","v":NaN}],"bn":"Nano02"}
{"e":[{"n":"humidity","u":"%","v":NaN}],"bn":"Nano02"}
{"e":[{"n":"count","u":"NIL","v":456}],"bn":"Nano02"}
{"e":[{"n":"temperature","u":"C","v":441}],"bn":"Nano01"}
{"e":[{"n":"voltage","u":"V","v":4.68875}],"bn":"Nano01"}
{"e":[{"n":"temperature","u":"C","v":28.1}],"bn":"NodeMCU_01"}
{"e":[{"n":"humidity","u":"%","v":71.400002}],"bn":"NodeMCU_01"}
{"e":[{"n":"moisture","u":"%","v":100}],"bn":"NodeMCU_01"}
{"e":[{"n":"battery","u":"%","v":93}],"bn":"Nano01"}
{"e":[{"n":"temperature","u":"C","v":27.9}],"bn":"NodeMCU_01"}
{"e":[{"n":"humidity","u":"%","v":71.199997}],"bn":"NodeMCU_01"}
{"e":[{"n":"moisture","u":"%","v":100}],"bn":"NodeMCU_01"}
{"bn": "wasp02", "e": [{"v": 4.09, "u": "V", "n": "voltage"}]}
{"bn": "wasp02", "e": [{"v": 87.0, "u": "%", "n": "battery"}]}
{"bn": "wasp02", "e": [{"v": 28.39, "u": "C", "n": "temperature"}]}
{"bn": "wasp02", "e": [{"v": 58.96, "u": "%", "n": "humidity"}]}
{"bn": "wasp02", "e": [{"v": 89.0, "u": "mB", "n": "pressure"}]}
{"e":[{"n":"temperature","u":"C","v":NaN}],"bn":"Nano02"}
{"e":[{"n":"humidity","u":"%","v":NaN}],"bn":"Nano02"}
{"e":[{"n":"count","u":"NIL","v":457}],"bn":"Nano02"}
{"bn": "wasp01", "e": [{"v": 4.06, "u": "V", "n": "voltage"}]}
{"bn": "wasp01", "e": [{"v": 82.0, "u": "%", "n": "battery"}]}
{"bn": "wasp01", "e": [{"v": 28.7, "u": "C", "n": "temperature"}]}
{"bn": "wasp01", "e": [{"v": 58.5, "u": "%", "n": "humidity"}]}
{"bn": "wasp01", "e": [{"v": 89.0, "u": "mB", "n": "pressure"}]}
{"e":[{"n":"temperature","u":"C","v":NaN}],"bn":"NodeMCU_01"}
{"e":[{"n":"humidity","u":"%","v":NaN}],"bn":"NodeMCU_01"}
{"e":[{"n":"moisture","u":"%","v":100}],"bn":"NodeMCU_01"}
{"e":[{"n":"temperature","u":"C","v":442}],"bn":"Nano01"}
{"e":[{"n":"voltage","u":"V","v":4.593061}],"bn":"Nano01"}
{"e":[{"n":"battery","u":"%","v":91.487808}],"bn":"Nano01"}
{"e":[{"n":"temperature","u":"C","v":28.1}],"bn":"NodeMCU_01"}
{"e":[{"n":"humidity","u":"%","v":71.300003}],"bn":"NodeMCU_01"}
{"e":[{"n":"moisture","u":"%","v":100}],"bn":"NodeMCU_01"}
{"e":[{"n":"temperature","u":"C","v":NaN}],"bn":"Nano02"}
{"e":[{"n":"humidity","u":"%","v":NaN}],"bn":"Nano02"}
{"e":[{"n":"count","u":"NIL","v":458}],"bn":"Nano02"}
{"e":[{"n":"temperature","u":"C","v":NaN}],"bn":"NodeMCU_01"}
{"e":[{"n":"humidity","u":"%","v":NaN}],"bn":"NodeMCU_01"}
```

Figure 9: MQTT client displaying data from all sensor nodes

The above readings are of a client which has subscribed to every topic on the broker. This is done using “IITMandi/COMlab/#” .

6 Additional Contribution

6.1 Implementing sleep modes :

- Since the waspmotes are industrial grade sensor nodes , so they have inbuilt codes for implementing the best suited sleepmode for optimal use, hence we just included those codes to implement sleepmodes in them.

- The arduino Nano has a variety of sleep modes to choose from, each having its own advantages and limitations, so we chose the deep-sleep mode, to achieve maximum power saving,, since in this mode everything is turned off and the device can only comeout of sleep mode either by external reset or watchdog timer interrupt .

- The Node-MCUs also have deep sleep modes similar to Arduinos, for them also deep sleep mode was chosen and they automatically turn their WiFi radio off when in sleep mode .
- The nodes implement a cycle in which, they read the sensor values, publish it ,go to sleep, wake-up and continue.

Note : The arduinos along with ESP8266 radios, donot save much power mainly due to the “power” LED which is always on .So in order to take full advantage of sleep mode, it is advisable to get rid of them by desoldering.

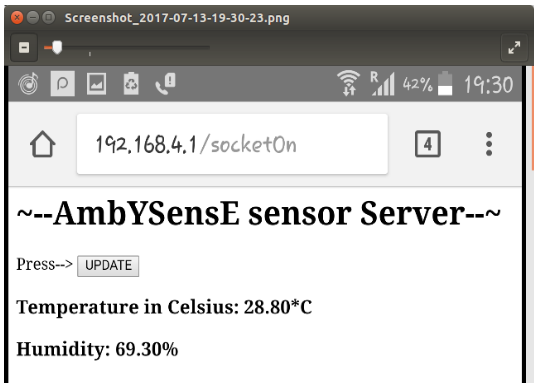
6.2 Remotely Enabling/Disabling sensors

- One of the NodeMCUs was programmed to demonstrate remote enabling/disabling of sensors.
- It subscribes to the topic “settings”, which contains instructions to enable or disable a sensor ,which is published by the administrator
- We couldnot achieve this whilst implementing sleep mode, and the reason for this is not clear, but it is suspected that due to the inability of PubSubClient library to allow a persistent connection with the broker, it may be so.
- In a persistent connection, the broker queues the messages for a client which have not been received and acknowledged by it, due to disconnection or network problems. Upon reconnection, broker tries to resend these messages.

6.3 AmbYsensE

- A NodeMCU [4] may be small, but it is a networking powerhouse. So, we have created “AmbYsensE”.
- To exploit its capabilities a Node-MCU was programmed to start an Access-point, and host a webpage which displayed the temperture and humidity readings of the room in which it was

present. When it is powered up,it sets up an AP currently named as "AmbYSensE" with default password "123456789". Visit the IP 192.168.4.1 to see it working.



~--AmbYSensE sensor Server--~

Press-->

Temperature in Celsius: 28.80*C

Humidity: 69.30%

Figure 10: Screenshot of webpage

7 Conclusion and future work

- Adding actuators to the network.
- Make the nodes power efficient by removing LEDs and implemeting sleep modes
- Field testing and deporting the sensor networks.

References

[1] MQTT https://github.com/mqtt/mqtt.github.io/wiki

[2] MQTT v.3 http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[3] ESP8266 http://esp8266.net/

[4] NODEMCU http://nodemcu.com/indexen

[5] Python-Matplotlib documentation https://matplotlib.org/contents.html

[6] Waspmotes http://www.libelium.com/libeliumworld/waspmote/

[7] SenML https://tools.ietf.org/html/draft-jennings-senml-10

[8] Arduino MQTT library https://github.com/knolleary/pubsubclient

[9] Mosquitto broker https://mosquitto.org/download/

[10] Arduino JSON https://github.com/bblanchon/ArduinoJson

[11] XCTU software https://www.digi.com/resources/documentation/digidocs

[12] Raspberry-Pi documentation https://github.com/raspberrypi/documentatio

[13] Raspberry-Pi hardware https://www.raspberrypi.org/documentation/hardware/raspberrypi/README

[14] Charith PereraA, Chi Harold Liu and Srimal Jayawardena "The Emerging Internet of Things Marketplace From an Industrial Perspective: A Survey" in IEEE Transactions on Emerging Topics in Computing ,vol. 3, no. 4, Dec. 2015 .