

A WIRELESS SENSOR NETWORK FOR A CAMPUS WIDE DEPLOYMENT

PROJECT REPORT

TITLE: ARK NETWORK

Supervisor: Dr. Siddhartha Sarma

Authors: Gundu Krishna Sriharsha

Submission date: July 29, 2017

Contents

1	Introduction	4
2	Problem statement	5
3	Deliverables	6
4	Equipments required for this project	6
5	Theory	6
5.1	Getting Started	7
5.1.1	Basic setup	7
5.1.2	Node Red	9
5.2	A Customized network - A simple RF communication	9
5.2.1	Algorithmic View	11
5.3	A Standardized solution - A structured network with MQTT	13
5.3.1	Algorithmic View of ARK network	15
5.3.2	Cross platform Integration - Waspnotes	16
5.4	Node Red - A Graphical Interface Solution	19
6	Timeline	25
7	Division of work	25
8	Lesson learned and challenges faced	25
9	Conclusion and Future scope	26
10	Appendices	27

List of Figures

1	IOT and WSN	5
2	Basic Development boards used	7
3	General Purpose Digital Temperature and Humidity sensor (DHT22)	7
4	Node Red logo	9
5	Customized network, the one on the left is the receiver node and the one on the right is the slave node	10
6	Raspberry Pi to Arduino via USB	11
7	Slave Arduino powered from USB.	11
8	Python input for User Interface	12
9	Standardized network on MQTT protocol	14
10	Output terminal of the RF24SN network	16
11	Waspnote IDE	17
12	Waspnote interfaced with events shield and Xbee (Antenna) radio module	18
13	Raspberry Pi Integrated with all communication radios	18
14	Flow diagram - Input	20
15	Flow diagram - Plotting data	21
16	Flow diagram - Actuation	22
17	Output	23
18	Output	24
19	Output	24

Abstract

The Aim of the project is to build a low-cost Wireless Sensor Network for a campus wide deployment of sensor nodes using the basic development boards like Arduino Uno and SoC computers like Raspberry Pi 3. By building this system, weather and other such parameters in the place of deployment can be monitored from the workplace easily. This gives tremendous flexibility in studying natural phenomenon like weather parameters in a relatively small region. The advancements in the field of, Internet of Things (IoT) and cloud storage, makes it even more flexible for the observer to be located anywhere on the globe, as the wireless sensor network is connected to the Internet at the top most level in the network. Finally, an alternate solution is built using node red layer on top.

1 Introduction

Internet of Things is a field which is rapidly advancing towards to a goal of inter-connectivity of all intelligent systems. A Thing is defined as "an object of the physical world or information world which is capable of being identified and integrated into communication networks" [1]. In other words, it is such a device which has the capacity to connect and compute. Such a technology allows for remote control of a network deployed in a location of examination with reduced human dependency. When such a network is correctly integrated with sensors and actuators, it greatly improves the capabilities and pushes the limit in controlling devices and other physical objects from anywhere on earth.

To enhance this connectivity, wireless technology has improved and come up different modes of communication such as WiFi, Xbee, Bluetooth, etc. In this way, by using an appropriate short range wireless communication, a network can be built. In building one such network, wireless sensor networks play a major role in making the mobile. As there are no wires for connection, there is a flexibility in deciding the place of deployment of the sensor node, to some extent. There are a variety of connections possible such as star, ring, tree, etc.

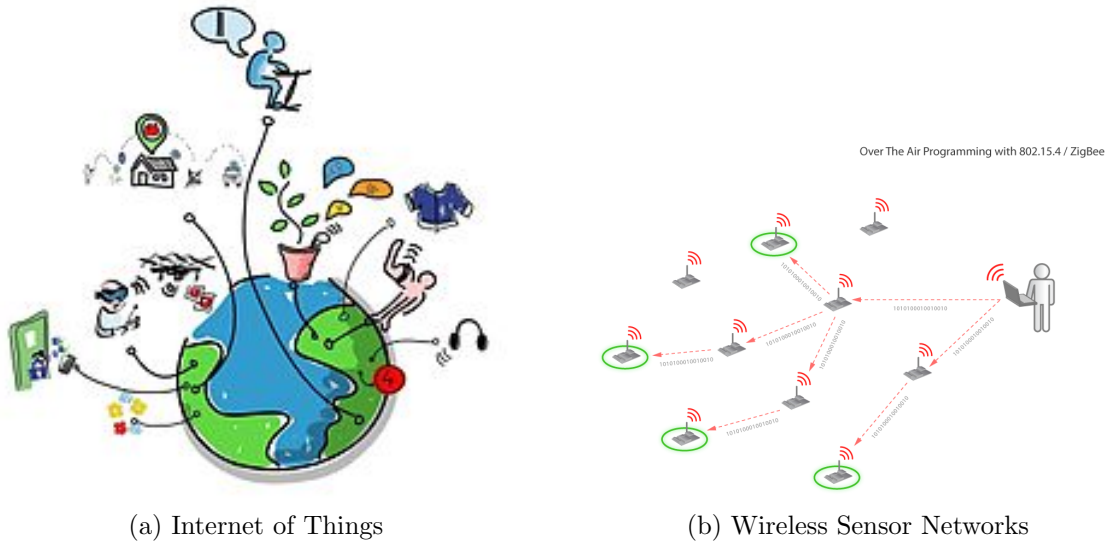


Figure 1: IOT and WSN

Based on the terrain and other parameters this is decided. In such a network, there is a gateway node with a connection to the Internet and thereby connecting the entire network to it indirectly. Due to vast size of the network and machine to machine interaction, a protocol and a message structure is followed to make the system uniform.

2 Problem statement

There is a need for a sensor network which will be deployed across campus. The sensor network should be as power efficient as possible with the help of sleep modes. The network should also be inter-operable, i.e, data measured from one node should be sent to others in the same network irrespective of the mode of communication deployed. The intensity of measurement and updating must be controllable dynamically when the network is up and running. There should be a control in switching individual measurements On or Off. In the case of a restart of an individual node, the network parameters and the sensing parameters should be synced automatically. It could be better if the nodes are made as close to plug and play model as possible.

3 Deliverables

These are the list of deliverables completed in the project

- A Customized model of network
- A Standardized model of network
- A Node red based solution to problem

4 Equipments required for this project

- Raspberry Pi 3
- Arduino Uno or equivalent development boards.
- DHT 22
- NRF24L01 boards
- PIR sensor
- LDR sensor
- Resistors

...and other sensors which would be integrated into the network.

5 Theory

According to the problem definition and the resources available, we need to build a heterogeneous sensor network with Raspberry Pi as the gateway node to collect data from sensor nodes(which could be Arduino uno/nano ,etc.).A customized network solution is built with Arduino uno as the sensor node and NRF24L01 as the board for communicating the sensor data.



(a) Raspberry Pi 3



(b) Arduino Uno Rev3

Figure 2: Basic Development boards used



Figure 3: General Purpose Digital Temperature and Humidity sensor (DHT22)

5.1 Getting Started

5.1.1 Basic setup

Assuming all the initial setup is completed, now the related software for interfacing the NRF24L01 board with the Raspberry Pi should be installed.¹ For functioning of basic point to point (individual) communication, the RF24 library must be installed [3]. Similar to setting up the Raspberry Pi, Arduino or Other sensor node should also be interfaced with this radio module. [4] Once the radio is working with the simple ping pair test, then RF24Network library must be installed. [4]. After installing the RF24Network library, all the nodes are tested with an example code for ping. This is to check if the nodes are working. If the

¹This can be done by using synaptic package manager also.

Raspberry Pi is operating in a network with proxy, the following commands should be used after replacing the IP address and other such fields to the local values.

If Wifi is being used -

```
sudo route add default gw 10.8.0.1
sudo ifconfig wlan0 netmask 255.255.0.0
sudo ifconfig wlan0 broadcast 10.8.255.255
```

If Ethernet is being used -

```
sudo route add default gw 10.8.0.1
sudo ifconfig eth0 netmask 255.255.0.0
sudo ifconfig eth0 broadcast 10.8.255.255
```

Following commands are common to both Wifi and Ethernet -

```
export http_proxy="http://10.8.0.1:8080/"
export https_proxy="https://10.8.0.1:8080/"
```

The following command will install the MQTT software ,named Mosquitto,in the Raspberry Pi. Using Synaptic Package manager or via the terminal, the API for mosquitto (libmosquitto and mosquitto) should be installed. [4]

```
sudo apt-get install mosquitto mosquitto-clients
```

To verify the internet connection, one can use ping google.com command in terminal window. Once all the components are tested, we can proceed to building the network. ²

²Note - If any errors are faced during installation, follow the Home automation link. The entire project is built as a layer to that project. There is a need for makefiles in linux for compilation of C programs. The make file for this project is taken from the home automation [4] project. In the terminal, after changing

5.1.2 Node Red

After setting up for the first time, the related software must be installed. The following commands will install Node Package Manager (NPM) which in turn will install node red on the linux system. After executing the last command, open localhost:1880 in browser(for the same system) or if viewed from another system, use the IP assigned to the node red enabled system at 1880 port. For example, for this project, the IP of the Raspberry Pi was 10.8.17.84, to access Node Red on Pi from my laptop, “10.8.17.84:1880/” will be the address.

```
sudo apt-get update
sudo apt-get install npm
sudo npm i npm -g
sudo npm install -g node-red
sudo npm install -g node-red-dashboard
node-red-start
```

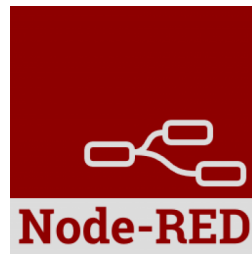


Figure 4: Node Red logo

5.2 A Customized network - A simple RF communication

In this network, there are only two sensor nodes. This is designed as a “two-place” system. One node will be deployed outdoor with a power connection from a battery while the other

the directory to the required location, “make” command must be executed to generate the executable file after every modification is made to the source code. Then start execution “sudo ./MQTTController”. The filename is fixed to “MQTTController” in the makefile which is not edited. On altering the make file this can be changed. In this project the filename “MQTTController” is used.

node is kept inside connected to a wall socket. In this setup, there are two Arduinos and one Raspberry Pi. The Arduino used in the indoor deployment (master Arduino) is connected to Raspberry Pi via USB connection. The data transfer between the master Arduino and the Raspberry Pi is via Serial communication. The other Arduino, deployed outdoor (slave Arduino) communicates using the NRF board to the master Arduino which in turn forwards the data to the Raspberry Pi via USB.

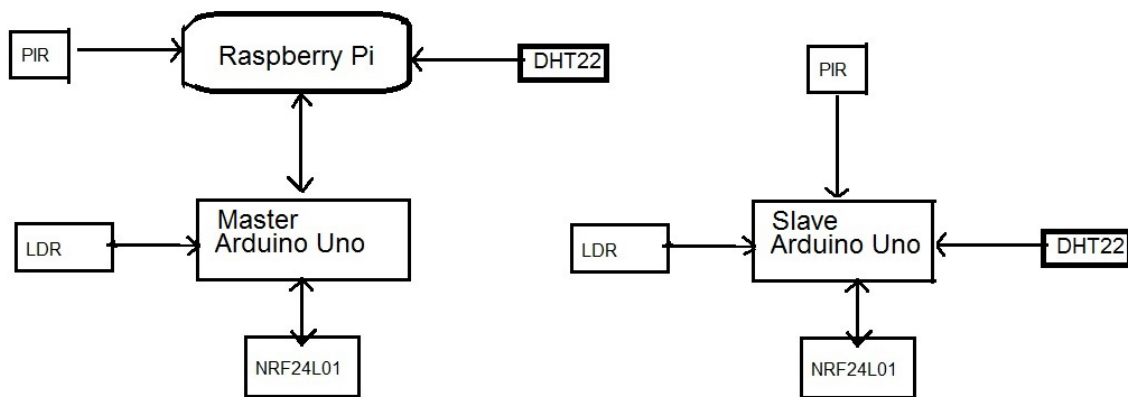


Figure 5: Customized network, the one on the left is the receiver node and the one on the right is the slave node

Since the Raspberry Pi does not have ADC support, LDR and other analog sensors must be connected to master Arduino. The python script is written such that it will signal the master Arduino to measure the analog value and send the data. The rest of the sensors (digital input/output) are connected to Raspberry Pi to reduce the load on Arduino.

The functions of Python script in Raspberry Pi are

- To check, correct and update (if necessary) the timing interval of the slave Arduino
- Signal the master Arduino for the sensor data
- Logging the sensor data into different files according to the type of measurement. (Computational and Non- computational)

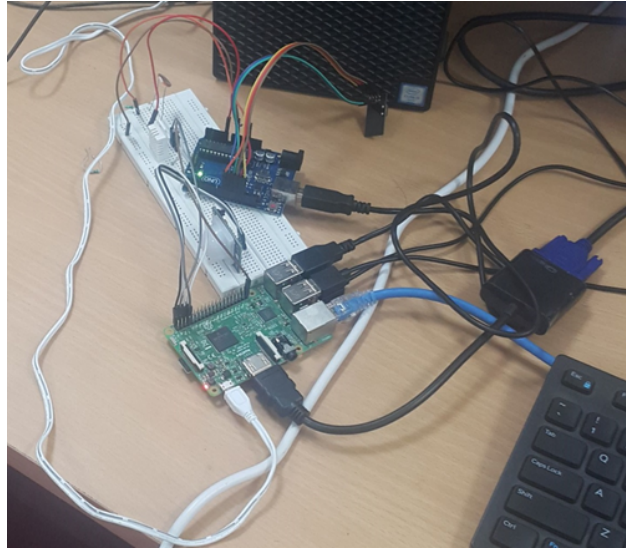


Figure 6: Raspberry Pi to Arduino via USB

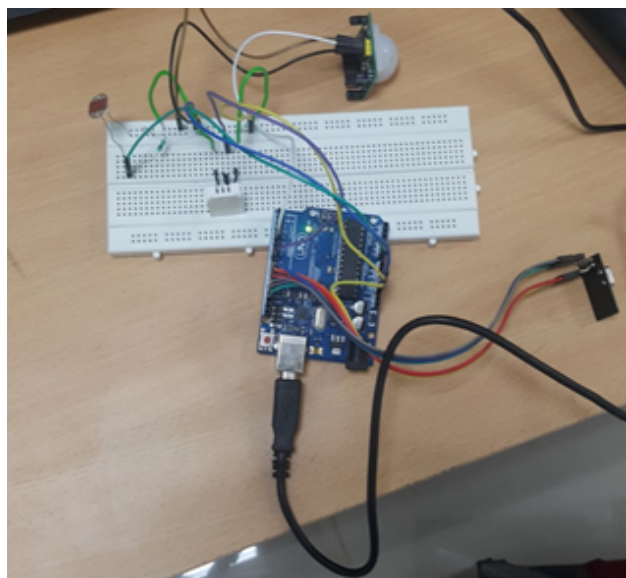


Figure 7: Slave Arduino powered from USB.

5.2.1 Algorithmic View

Raspberry Pi has a Python script, running mainly for the above mentioned reasons. At the start of the execution, it asks user for the input. The user gets to choose among three modes.

- Power Saving Mode - Once every 30 minutes, the sensor data is updated

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ----- RESTART -----
>>>
Select
1.Power Saving Mode(Data (both indoor and outdoor) will be logged once every 30 minutes)
2.Balanced Mode
3.Performance Mode(Data will be logged once every 20 seconds) |
```

Figure 8: Python input for User Interface

- Balanced Mode - The user can set the update intervals of the nodes separately, with a value between 3 seconds to 30 minutes.
- Aggressive Mode - The nodes will be updating the data once every 20 seconds.

Once the selection is done, Python sends the interval of the slave node via serial communication. The master node forwards this to the slave node.

The slave node executes a code which keeps it in one of the two states. It is either listening constantly for an interrupt of its routine interval or it is transmitting the data at the end of the previously said interval. When a command is received, it will be in effect from next measurement.

The Python code has a variable to keep in track of the measurement interval. It is an array of two numbers. The latest time (in milliseconds) is stored in the place of the older of the two times present. The difference of the two blocks gives the delay time of the two successive sensor data. If this is not equal to the interval which the user has set (within a margin of error), then the Python re-instructs the master node to set the selected interval for the slave node. The indoor part of the measurements happen from the Raspberry Pi itself, hence there is no need for keeping it in synchronization. A simple time tracker is enough to keep the measurement interval to a particular interval. Whenever the state of PIR sensor in slave node goes HIGH, then the listening stops and transmits a code which the master Arduino understands as an interrupt and can distinguish this from data. Master Arduino forwards this message to the Raspberry Pi after attaching the slave node address via USB. If there is a HIGH state for PIR attached to Raspberry Pi, then the message is directly logged with

Raspberry Pi's address. If it is time for indoor measurement, the script sends a code which the master Arduino can decipher to be the request for LDR value. After measuring the LDR value, it is transmitted back via the same Serial connection. In this way the sensors are used to get the values for indoor and outdoor weather parameters.

Advantages of such customized system

- It is best for small environments
- Small payload size because there won't be a field for node address ,etc.
- Easy to control and debug the system.

Disadvantages of such customized system

- Interoperability and heterogeneous nature is lacking ,i.e, a board with ESP communication board cannot be a part of the network
- Actuation network cannot be created because of lack of interoperability.
- There is no uniformity in message structure and communication.

5.3 A Standardized solution - A structured network with MQTT

In order to counter the limitations of the previous network, a standardized approach is necessary to level out all the differences in nodes and their various modes of communication. In this solution, the Raspberry Pi is interfaced with NRF radio directly and a network is formed with it as a parent and the other Arduinos as children. This network is called RF24SN.³ This network is responsible for the reception of data from Arduino sensor node to Raspberry Pi. Once the data is received into Raspberry Pi, it then published into an MQTT broker program running in the background. In this setup we have used open source software Mosquitto and its API to publish and subscribe data from the C script running in the Raspberry

³This network drivers are coded by the same person who has coded the libraries for the boards.

Pi. If a message is published in the topic of subscription, the C code is programmed to send an appropriate message to the nodes. For example, if a message is published into "Settings/Knode1/Interval" with 20 , then the interval of Node1 is set to 20 seconds by sending an appropriate message. In a similar way, a switch is programmed such that an individual measurement is turned off/onby publishing the message under the appropriate channel. For data interoperability, before publishing the data, it is converted into JSON format. This is done to all data coming into the MQTT server to maintain uniformity in the network.

From the perspective of the MQTT broker, it is all about data being published and being re-routed to interested nodes. Due to a common JSON format, MQTT broker can directly send the data to the actuator node which can read the information after processing it.To the same MQTT broker,data from other nodes are published. The Raspberry Pi is configured in a way, to receive data from all kinds of radios available. It is at this level, all of the network converges to.

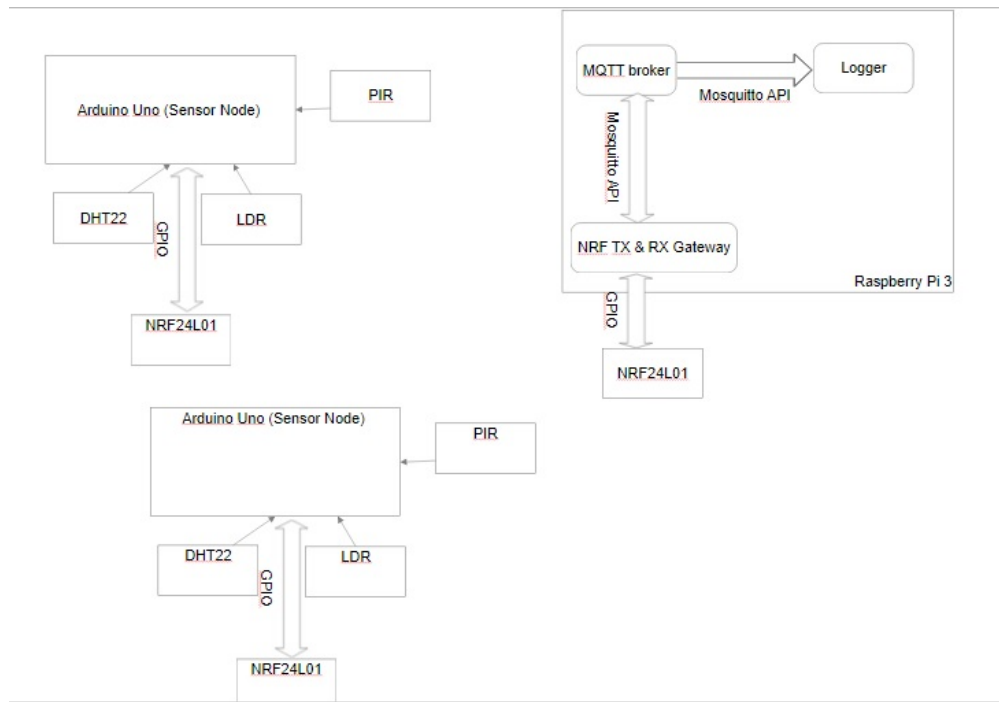


Figure 9: Standardized network on MQTT protocol

5.3.1 Algorithmic View of ARK network

A RF24 network is created with Raspberry Pi as the parent to all the Knodes in ARK network⁴. As of the time of project , there are two child nodes. The Knodes are programmed that they will be delivering the unique message with 5 parameters.

```
Battery level Temperature Light(%) Humidity PIR-State
```

The message is sent with sensor values in the variables. The packet when sent from node contains the source and the destination address. Hence, when the header is examined, the message and the routing details can be known. According to this source address, the data is identified to be from which of the nodes it has been sent. Once the node is identified, then the sensor values are read and posted into respective channels in the MQTT broker through mosquitto API to C++. The same code contains the section where it is connected to few interested channels. The data from these channels is notified to the program. It is coded to be subscribed to all channels under “Settings”, with the help of “#” wild card [5]. Whenever a client publishes data under this channel, it is processed and the equivalent instruction is sent to the sensor node to which it is addressed by the client.

For example , If the requirement is such that the interval of measurement for node1 is to be changed to 3 minutes , instead of the current value, then in the following line must be entered in the terminal.

```
mosquitto_pub -t Settings/Knode1/Interval -m 180
```

Once the above line is executed the data 180 seconds (=3 minutes) is received by the C script and it is sent in its own message structure. The first value is the address to sensor. The second value is the state of the sensor addressed in first value. The third value is the change in interval.

```
{0 0 180}
```

⁴RF24SN + MQTT network is called ARK network and the nodes in it are called Knodes.

If the Temperature measurement must be switched off, in node1, then the following command is executed in terminal.

```
mosquitto_pub -t Settings/Knode1/Temperature -m Off
```

The message from C script to the node will be sent as

```
{1 0 0}
```

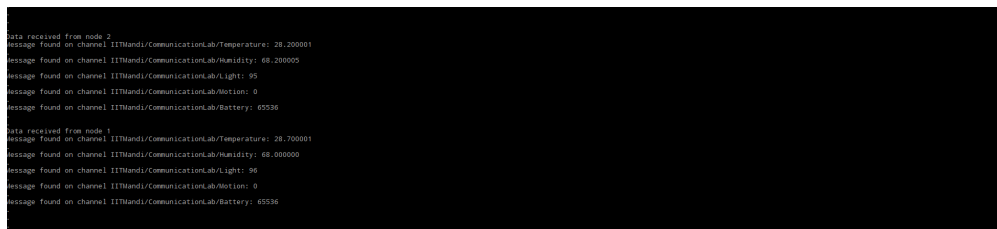
```
1- Temperature 0 - OFF 1 - ON
```

```
2 - Humidity
```

```
3 - PIR
```

```
4 - LDR
```

This particular system is working on push mode. The Knodes are being sent the timing information and hence are pushing the data into the Raspberry Pi. In future, we can also work on pull mode,i.e, data on demand. If monitoring of the system is not required and if the data is needed based on the demand of an external event, this mode can be used. In pull mode, the Knodes will be listening for a command from the Raspberry Pi.It will only measure the required values when instructed. This mode as an option can be integrated into the current system, to make it more versatile.



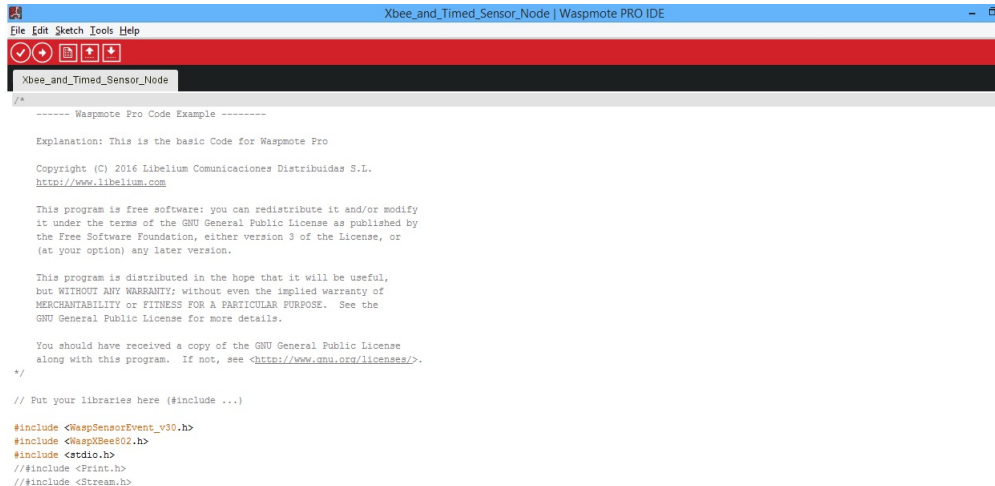
```
data received from node 2
message found on channel IITMandi/CommunicationLab/Temperature: 28.200001
message found on channel IITMandi/CommunicationLab/Humidity: 68.200000
message found on channel IITMandi/CommunicationLab/Light: 95
message found on channel IITMandi/CommunicationLab/Motion: 0
message found on channel IITMandi/CommunicationLab/Battery: 65536

data received from node 1
message found on channel IITMandi/CommunicationLab/Temperature: 28.700001
message found on channel IITMandi/CommunicationLab/Humidity: 68.000000
message found on channel IITMandi/CommunicationLab/Light: 96
message found on channel IITMandi/CommunicationLab/Motion: 0
message found on channel IITMandi/CommunicationLab/Battery: 65536
```

Figure 10: Output terminal of the RF24SN network

5.3.2 Cross platform Integration - Waspnotes

This network, in this configuration supports cross platform nodes. To test this, a development board from Libelium named Waspnote is added into this network.The IDE used by waspmotes are similar to that of Arduino.



```
Xbee_and_Timed_Sensor_Node | Wasmote PRO IDE
File Edit Sketch Tools Help
Xbee_and_Timed_Sensor_Node
/*
----- Wasmote Pro Code Example -----
Explanation: This is the basic Code for Wasmote Pro
Copyright (C) 2016 Libelium Comunicaciones Distribuidas S.L.
http://www.libelium.com
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.
*/
// Put your libraries here (#include ...)
#include <WaspSensorEvent_v30.h>
#include <WaspXbee502.h>
#include <stdio.h>
#include <Print.h>
#include <Stream.h>
```

Figure 11: Wasmote IDE

To increase the heterogeneous nature of the network Xbee modules are used for communicating from the waspmotes to Raspberry Pi. There is a co-ordinator node connected to Raspberry Pi which receives the data and forwards this data to the MQTT broker under appropriate name.

Directory - IITMandi/COMLab/Wasp_01

Now if in the C code of the ARK network, this data is to be accessed, it can be subscribed to this channel and an actuation network on this sensor data can be built elsewhere. This is a key improvement to the previous system. These Wasmotes are pre-programmed with sleep libraries. Hence sleep functions are much easily integrated into the code. The time of sleep should can be set via command similar to the ARK network. In the waspmotes, the sensors are not separate like that of those of Knodes where DHT22 sensor is used separately. An events board is purchased separately and is stacked on top of the main board. There is a highly integrated sensor, which measures temperature, humidity and pressure which fits into a slot onto the events board.

This node sends a string of data in the form of JSON format which makes it easier if an actuation network is being built. Due to its UDP network connection type and its data per transmission being capped to 100 bytes. The data is sent parameter wise i.e, in parts. The



Figure 12: Wasmote interfaced with events shield and Xbee (Antenna) radio module

coordinator node receives the data and forwards them to the common MQTT broker.

Advantages of such Standardized system -

- It is a scalable network.
- Interoperability between different kinds of nodes is possible.
- Unified structure and a protocol is being followed.
- An actuation network can be built anywhere in the network.

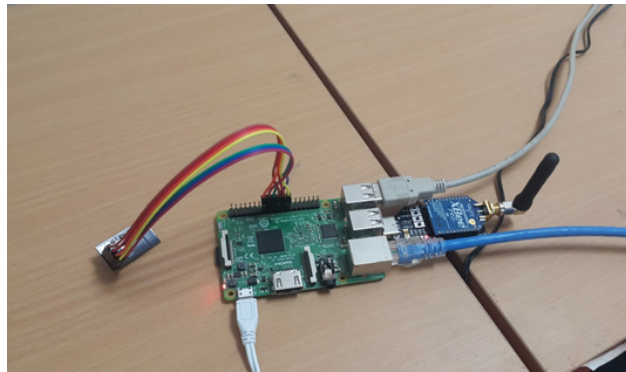


Figure 13: Raspberry Pi Integrated with all communication radios

5.4 Node Red - A Graphical Interface Solution

Node red is an application which is pre-installed in Raspberry Pi. It is a graphical programming tool, which works on Node.js. In Node Red, there are separate nodes modules which have some function associated with it. As soon as a node is executed, the control moves onto the next node module in the flow(the connected diagram in Node red is called a flow). In this way, a complex system can be designed with the help of individual small node modules connected in the way which completes the task.All the nodes start working on being triggered and output the desired function, this data can be used to trigger the next node module.The underlying architecture and communication protocol is similar to that of the previous model.

A C script is written such that it receives the data from the NRF radio and prints it onto the command line, in the following format

```
Node_Number Temperature Humidity Light PIR_State Battery_level
```

After this execution , Node Red gets call back to the flow with the result. The flow is created such that immediately after receiving the message, it is re-triggered to listen for the next message. In this way, it is in an infinite loop in listening to sensor data.

In the Input flow, there is an inject node module which injects a dummy message into the Radio node.On double clicking the node,an option can be seen to inject a message at the start of the deployment.It should be checked for the flow to start immediately after deploying it,else one should click the inject button to start the flow.Radio node is an execution node module in advanced section of the left pane. In this execution node module the following command should be given or a similar one based on the location of the executable script.

```
sudo ./MQTTController (if the file is present in /home/pi directory)
```

OR

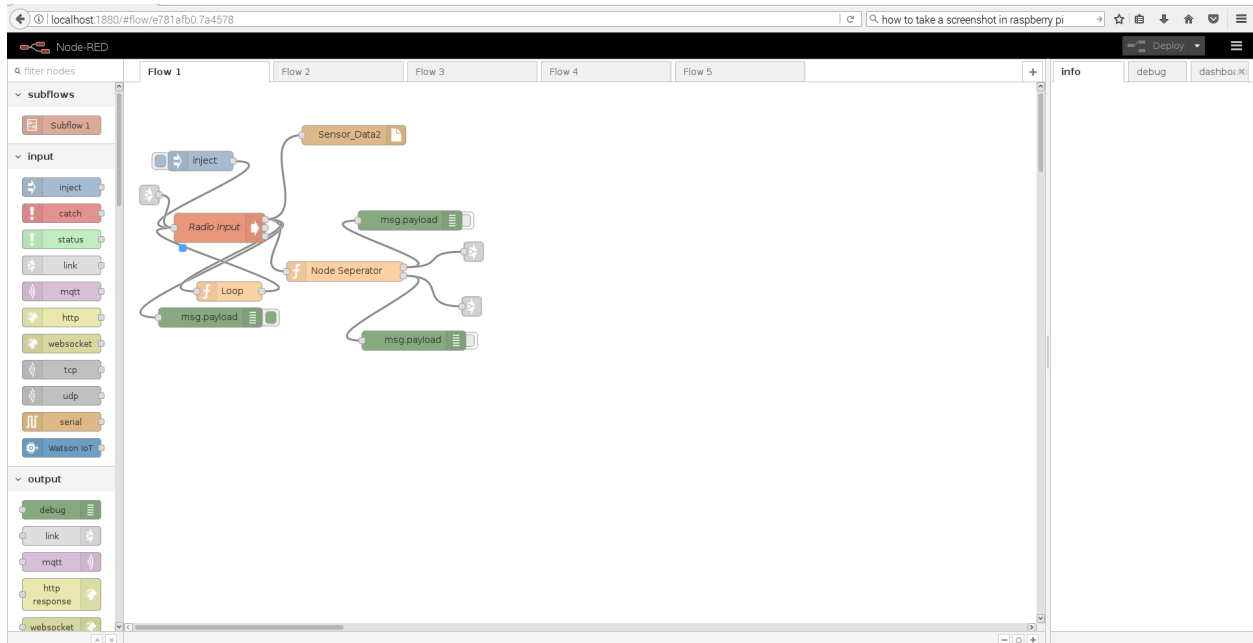


Figure 14: Flow diagram - Input

```
sudo ./filepath/MQTTController
```

Every time , the exec node module is triggered, it opens a radio listening channel and listens for data to be sent in the network. As soon as data is received, it is formatted into tab spaced values and is printed on the terminal. On printing the data, the file finishes its execution and with a call back this data is transferred to node red domain and is sent to the next node module(s). In this case, there are three node modules attached to the output. So all the three node modules receive the data.

The loop node module is a function module found in the left pane. It is a place where a custom script can be executed and the data could be sent as output. The loop node will trigger the Radio node module as soon as receiving the data, thereby keep the forever loop in listening to the data.

The green node in the flow is called debug node, its function is to show the message being passed through the connections in the flows. It is useful for when examining the system in detailed. The debug mode is disabled by clicking the green button attached to it. The

”Node separator” is also a custom function node module. This node module is responsible for dividing the incoming data into two paths according to the node that has sent the data. The grey node modules are linking flows across pages. This is done for aesthetic purposes and to maintain the clarity in the flow diagram.

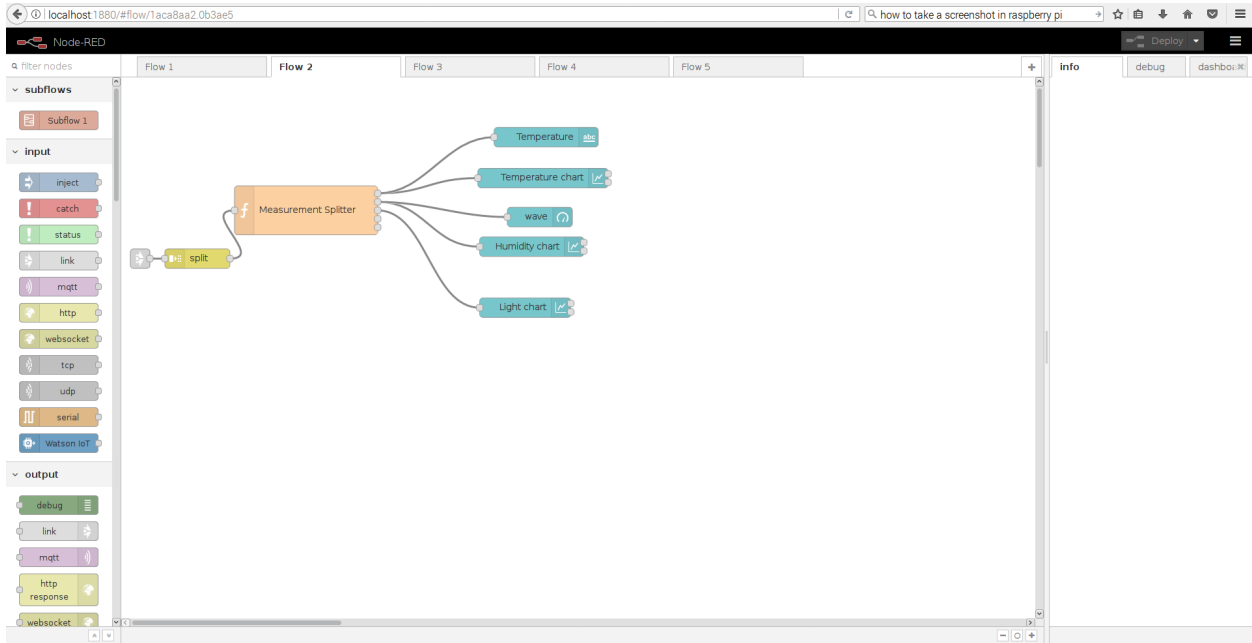


Figure 15: Flow diagram - Plotting data

After splitting the data according to the base address, it is sent to the above flow. For explanation purposes, the flow for one sensor node (Arduino board) is shown but exactly similar flow is designed for other sensor nodes (Arduino board) too.

The data is sent from the grey linking nodes. The split node is responsible for splitting the tab spaced data into individual values. For example -

2 23.4523 66.9822 89 0 43

into

2

23.4523

66.9822

89

0

43

Once the entire data is split into respective measurements, a custom function is written to redirect them into their respective output channels. To these output channels, the required charts can be linked. These values will be shown in "localhost:1880/ui" or "IP-Address:1880/ui" (in my case 10.8.17.84:1880/ui). The wave node is under the name gauge in the dashboard section of the left pane. This displays the data as a level indicator in the same UI page.

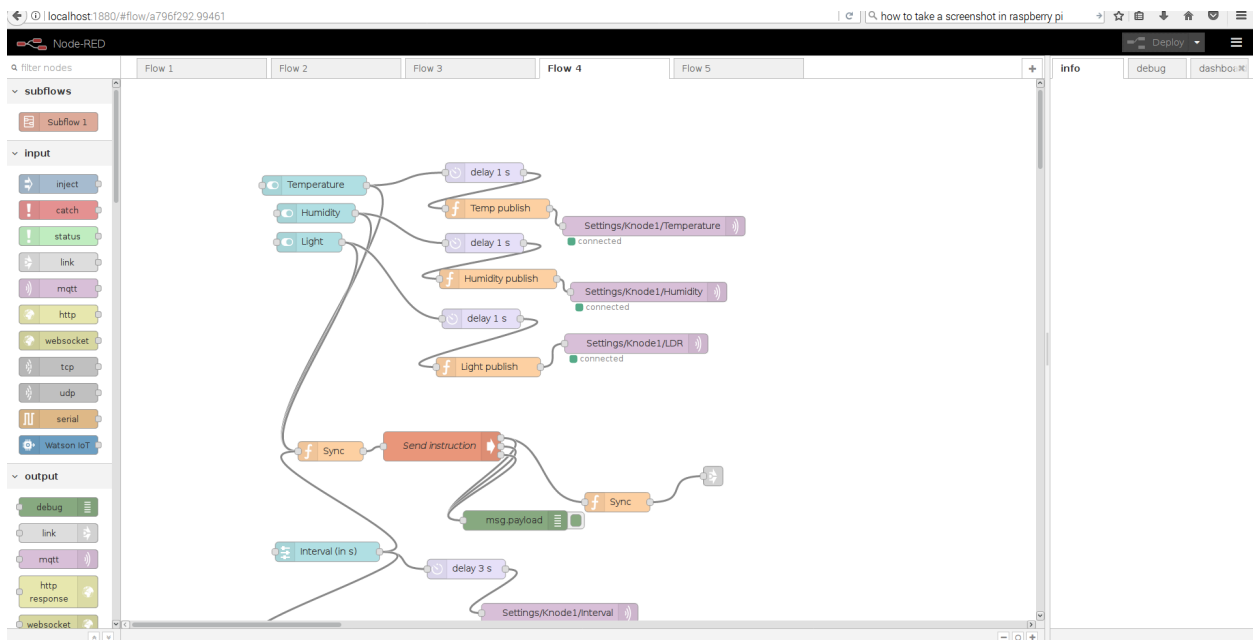


Figure 16: Flow diagram - Actuation

This part of the flow has input from the UI. There are switches integrated into UI for switching an individual measurement between On and Off. Once the state is toggled or the interval of measurement is changed, the node triggers the next node module with the payload. The node module named Sync sets the Radio variable to Output, in this state, the Radio

node in the first flow stops working. It, then starts executing the transmission script which is connected to MQTT broker.

In parallel to this, the code will be delayed by 1 second in the other branch to initiate the transmission script and initialize the radio for it. Then the publish node changes the payload value into an appropriate message for the script to understand. It is then published under the topic in the MQTT broker

The C script will be waiting for a value to be received via MQTT. Once the value is received, it changes the payload into a format the sensor node understands and send it to the addressed node. The code for the sensor node (Knode) is written such that after every update, it checks if any data is transmitted from the Pi. On receiving the data to the sensor node, it alters the measurement parameters and continues in the new state. The C script in Raspberry Pi will terminate after sending the data. The Sync function is executed which sets the radio value to read mode. This linking node will trigger the reading exec node module in 1st flow. A gauge is connected to the slider to show the interval in seconds. The final output in the UI page is

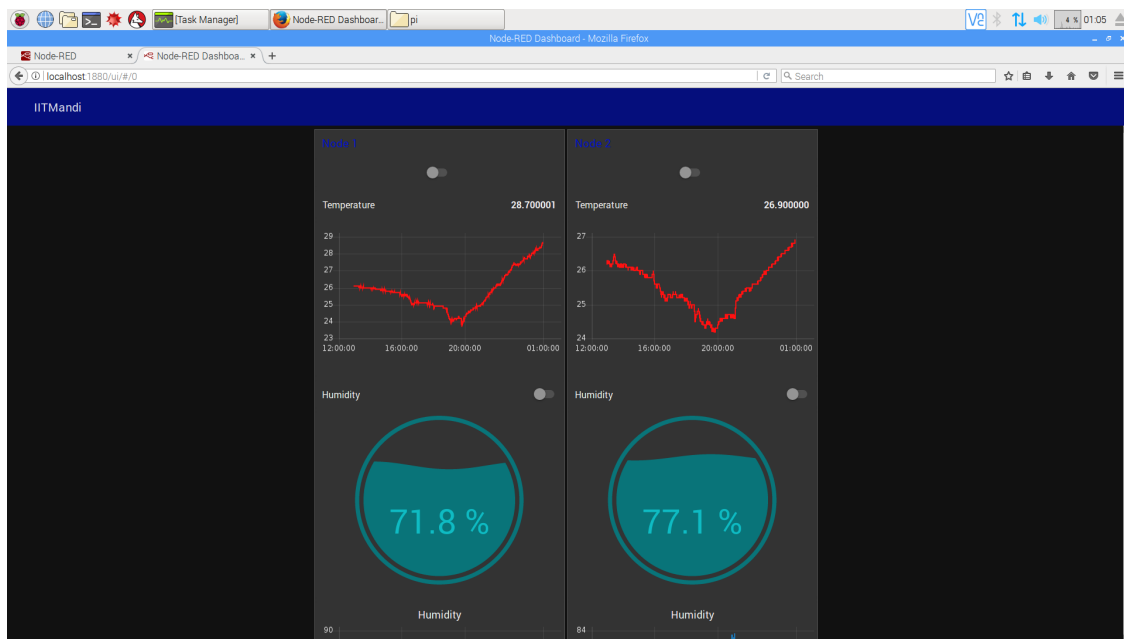


Figure 17: Output

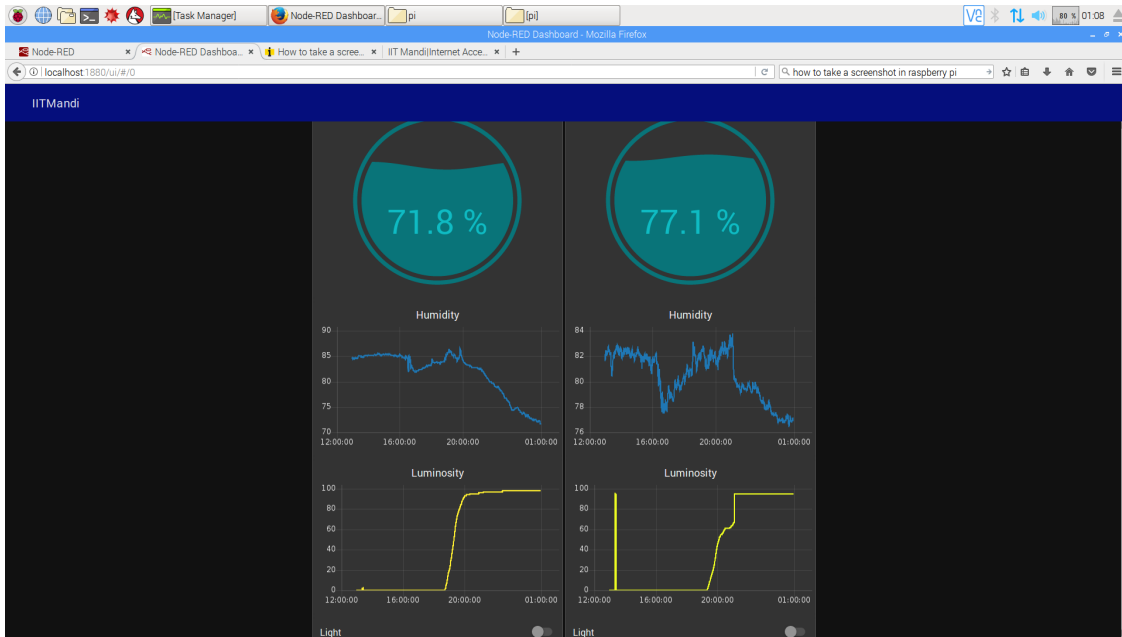


Figure 18: Output

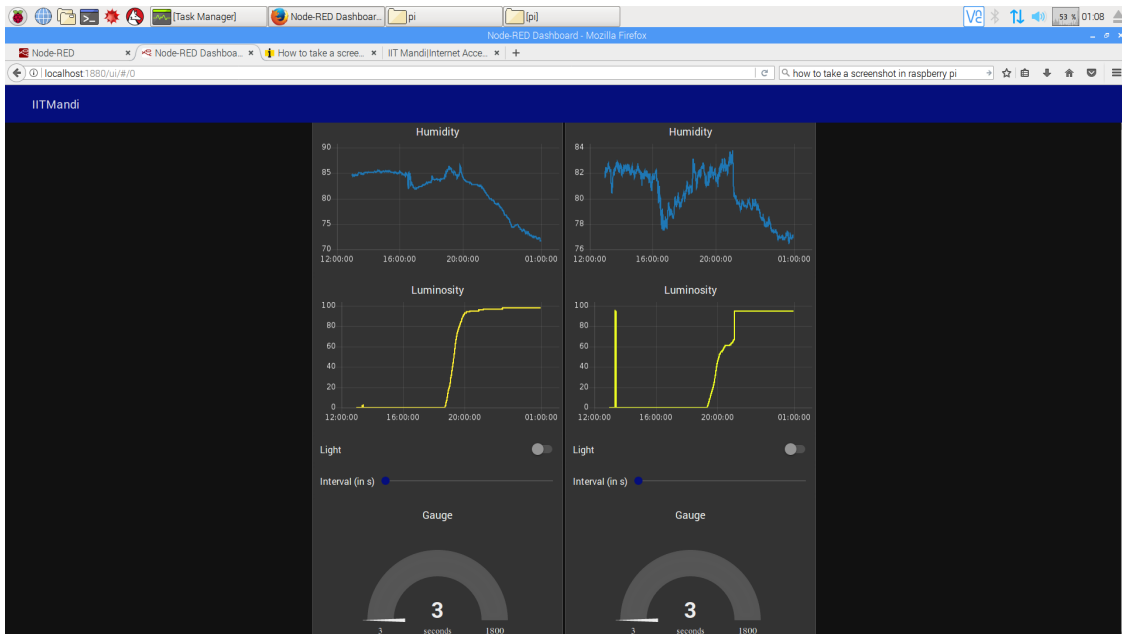


Figure 19: Output

Advantages of using Node Red -

- Very good GUI is present, which can be used to visualize the data.
- Easily understandable due to its graphical nature.

- Little to no programming experience required.
- If an improvement is required, it can be highly localized to replacing one to two nodes which carry out the specified functionality.
- It can be used to run bits of code from other languages with the help of exec node which instructs from command line.

6 Timeline

Overall project duration - *June 12th to July 29, 2017*

From 12th June to 30th June, I was working on developing a customized network.

From 30th June to 15th July, I was working on developing a Standardized network.

The presentation on the above topics was given on 18th July,2017.

From 18th July to 23rd July, I was working on a solution on Node Red.

7 Division of work

I have worked on these sets of solution, meanwhile T.Prateek has worked on setting up of the MQTT broker and a platform for other nodes apart from NRF communication. He has used ESP8266 and Zigbee communication boards to integrate it into the MQTT network. I have integrated ARK network to the MQTT broker, at the last.

8 Lesson learned and challenges faced

There are quite a few challenges faced while setting up the network. Some of them are -

- Time in the Raspberry Pi was not synced.

```
sudo date -s "Thu Aug 9 21:31:26 UTC 2012"
```

- Raspberry Pi was connected to a network which has proxy configured.
- Int data type, is not of the same size in Raspberry Pi and Arduino and hence when any data is transferred, it was being corrupted at the other end. - Instead uint32t as a datatype.

9 Conclusion and Future scope

In the above ways, a wireless sensor network can be setup for sensing the required parameters. These are first few steps in building a wireless sensor network which is extremely, optimized and streamlined for its use. There are still a few flaws with the current model of network. Working on them will be the future scope of this project.

Some of the flaws in the system are

- The sleep modes are not effective due to power wastage from power and other LEDs
- The payload can be reduced by using MQTT-SN structure than MQTT itself.
- The GUI and communication in Node Red can be further optimized by using a HTTP socket or Serial port communication instead of a call back from terminal.

References

- [1] International Telecommunication Union, Overview of the Internet of things, Recommendation ITU-T Y.2060, June 2012
- [2] Pictorial representation of RF24SN network <https://github.com/VaclavSynacek/RF24SN/blob/master/RF24SN.png>

- [3] RF24SN network setup <https://www.element14.com/community/community/raspberry-pi/raspberrypi2/blog/2015/04/07/raspberry-pi-2-gpio-usage-with-nrf24l01-arduino>
- [4] A Similar project for home automation. <http://homeautomationforgeeks.com/rf24hardware.shtml>
- [5] API for Mosquitto broker for C and C++ language. <https://mosquitto.org/api/files/mosquitto-h.html>
- [6] Adafruit <http://www.learn.adafruit.com>
- [7] Github repository for Codes <http://github.com>
- [8] Resources for separate node modules. <http://www.npmjs.com>

10 Appendices

Knode - It is the name of the node which is working in the standardized network with NRF24L01 board and the network is called ARK network.