CALIFORNIA STATE UNIVERSITY, FRESNO

Electrical and Computer Engineering Department



ECE 298: PROJECT

**Real-time Applications of Wireless Sensor Networking for Smart Irrigation Monitoring and Control**

Student Name: Manasa Thatipamula

Student ID: 109551721

Project Advisor: Dr. Hovannes Kulhandjian

A Project Report Submitted In Partial

Fulfillment of the Requirements for the Degree of Master of Science

in the Lyles College of Engineering

California State University, Fresno

December 2017

**Abstract**

Wireless sensor networking (WSN) has many useful applications in many different areas. WSNs proliferated due to the availability of inexpensive, low powered, and small components like processors, radios and sensors that were often integrated on a single chip (SOC). The small size and low power consumption of wireless sensor nodes (motes) allows creation of multi-hop message transmission and provides capability of ad-hoc networking. The data transfer occurs within low power RF transmissions in a 2.4 GHz radio spectrum. Motes are applicable of long term autonomous operations in military, health monitoring as well as smart irrigation applications. One of the significant feature of motes is that it goes to the sleep mode in low power mode for long periods of time helps in achieving power management.

The main objective of this project is to wirelessly collect data from the sensors on the XM1000 mote and transmit the data to the base station, which is connected to a PC. The collected data range from temperature, humidity, and light, additional sensor can also be incorporated e.g., the salinity of the soil. In this work we implemented a multi-hop networking in which an intermediate sensor node is used to relay the measured sensed data to the destination node. A GUI is created to display the received data on a PC. The gathered data is then uploaded onto on the web by creating a remote web application, which can be accessed from anywhere in the world having Internet connectivity. After gathering the data and analyzing one can remotely control in a smart way the irrigation of the crops. We have also implemented the Dijkstra's algorithm in multi-hop networking to route the packets through the shortest path from the source node to the destination, the PC. Analytical results of the implementation of the codes on the XM1000 motes are also presented in this work.

**Acknowledgements**

I would like to thank Dr. Hovannes Kulhandjian for assigning me this project and providing me the hardware used in the project. He constantly supported and guided me to overcome the difficulties faced throughout the project. Under his supervision, project was completed successfully.

I would also like to thank my parents and friends who helped and encouraged me all the times in any possible way.

**Contents**

[1. Introduction](#_Toc500630205) 7

[2. Applications of Wireless Sensor Networks 8](#_Toc500630206)

[3. Wireless Sensor Node 10](#_Toc500630207)

[4. Technologies 12](#_Toc500630208)

[4.1 Sensor Nodes 12](#_Toc500630209)

[4.2 TinyOS 13](#_Toc500630210)

[4.3 NESC 14](#_Toc500630211)

[4.4 Python 15](#_Toc500630212)

[4.5 TKINTER Python 15](#_Toc500630213)

[4.6 Adafruit IO 15](#_Toc500630214)

[4.7 MQTT (Message Queue Telemetry Transport) 16](#_Toc500630215)

[5. Architecture and Design 16](#_Toc500630216)

[5.1 Requirements 16](#_Toc500630217)

[5.2 Architecture 16](#_Toc500630218)

[6. Transporting data from Mobile Mote to Base Station Mote. 17](#_Toc500630219)

[7. Multi Hop Routing in Wireless Sensor Networks 18](#_Toc500630220)

[7.1 Experiment set up in a closed room: 18](#_Toc500630221)

[7.2 Experiment setup in field: 19](#_Toc500630222)

[8. Web Application 20](#_Toc500630223)

[8.1 Adafruit Io 20](#_Toc500630224)

[8.2 MQTT 20](#_Toc500630225)

[8.3 Web Application Server 21](#_Toc500630226)

[8.4 Results: 22](#_Toc500630227)

[9. Dijkstra’s Algorithm 22](#_Toc500630228)

[9.1 Algorithm Procedure [15]: 23](#_Toc500630229)

[9.2 Experiment setup Representation: 24](#_Toc500630230)

[9.3 Experiment Set up: 25](#_Toc500630231)

[9.4 Experiment setup in field: 26](#_Toc500630232)

[10. Summary 27](#_Toc500630233)

[10.1 Analysis 27](#_Toc500630234)

[10.2 Future Scope 28](#_Toc500630235)

[11. References 29](#_Toc500630236)

**List of Figure**

|  |  |  |
| --- | --- | --- |
| Fig. No. | Title | Page No. |
| 1 | Wireless Sensor Network | 7 |
| 2 | Wireless Sensor Network Architecture | 8 |
| 3 | Sensor Node Architecture | 10 |
| 4 | XM1000 Mote Module | 12 |
| 5 | Tiny OS application Architecture | 13 |
| 6 | Component Model of NesC | 14 |
| 7 | NesC Components | 14 |
| 8 | Architecture of WSN | 17 |
| 9 | GUI of data collected at PC through Base Station Mote | 17 |
| 10 | Experiment set up of multi-hop routing in a closed room | 18 |
| 11 | GUI of Multi-hop routing in a closed room | 19 |
| 12 | Experiment setup of multi-hop routing in field.… | 19 |
| 13 | GUI of multi-hop routing in field | 20 |
| 14 | Dashboard for the temperature, humidity and light sensed through multi-hop routing | 22 |
| 15 | Dijkstra Algorithm Procedure | 23 |
| 16 | Algorithm setup Representation | 24 |
| 17 | Algorithm set up in closed room | 25 |
| 18 | Results of Algorithm in closed room | 26 |
| 19 | Algorithm set up in field | 26 |
| 20 | Results of Algorithm in Field | 27 |

[Appendix 1 Implementation of Base station Mote Program(Appc.nc) 31](#_Toc500976984)

[A 1.1 Implementation of Base station Mote Program(c.nc) 31](#_Toc500976985)

[A 1.2 Implementation of Mobile Motes Program (Appc.nc) 32](#_Toc500976986)

[A 1.3 Implementation of Mobile Motes Program(c.nc) 33](#_Toc500976987)

[A 1.4 Implementation of Python Program for GUI 35](#_Toc500976988)

[Appendix 2 Base station program for bob in multi-hop relay(Appc.nc) 38](#_Toc500976989)

[A 2.1 Base station program for bob in multi-hop relay(c.nc) 39](#_Toc500976990)

[A 2.2 Mediator Mote program in multi-hop relay(APPc.nc) 40](#_Toc500976991)

[A 2.3 Mediator Mote program in multi-hop relay(c.nc) 41](#_Toc500976992)

[A 2.4 Alice Mote program in multi-hop relay(Appc.nc) 42](#_Toc500976993)

[A 2.5 Alice Mote program in multi-hop relay(c.nc) 43](#_Toc500976994)

[A 2.6 Implementation of Python Program for GUI 45](#_Toc500976995)

[Appendix 3 Alice Mote program for Algorithm (Appc.nc) 49](#_Toc500976996)

[A 3.1 Alice Mote program for Algorithm (c.nc) 49](#_Toc500976997)

[A 3.2 Base station program for Algorithm (Appc.nc) 52](#_Toc500976998)

[A 3.3 Base station program for Algorithm (c.nc) 52](#_Toc500976999)

[A 3.4 Python program for getting the data from Base station 54](#_Toc500977000)

[A 3.5 Python program for Algorithm to calculate shortest Distance after getting data. 54](#_Toc500977001)

1. Introduction

A wireless sensor network (WSN) is a formation of networks by which sensor data can be transported wirelessly. WSN are built of sensor nodes with small units consisting of three parts namely a radio transceiver connected with an internal or external antenna, a microcontroller and power sources. These nodes also have modest on-board computing capabilities. WSNs are used to monitor the changes in physical or environmental conditions like sound, pressure, temperature and humidity co-operatively pass data through the network to the main base station.

Wireless sensor nodes require low power and are software configurable. Sensor nodes communicate between themselves mainly with radio signals. The individual nodes in WSN are naturally resource constrained: they have restricted processing speed, storage capacity, and communication bandwidth. After the sensor nodes are deployed they are responsible for self-organizing a suitable network communication often with multi-hop communication with them then sensors of onboard starts collecting the information from the other nodes.

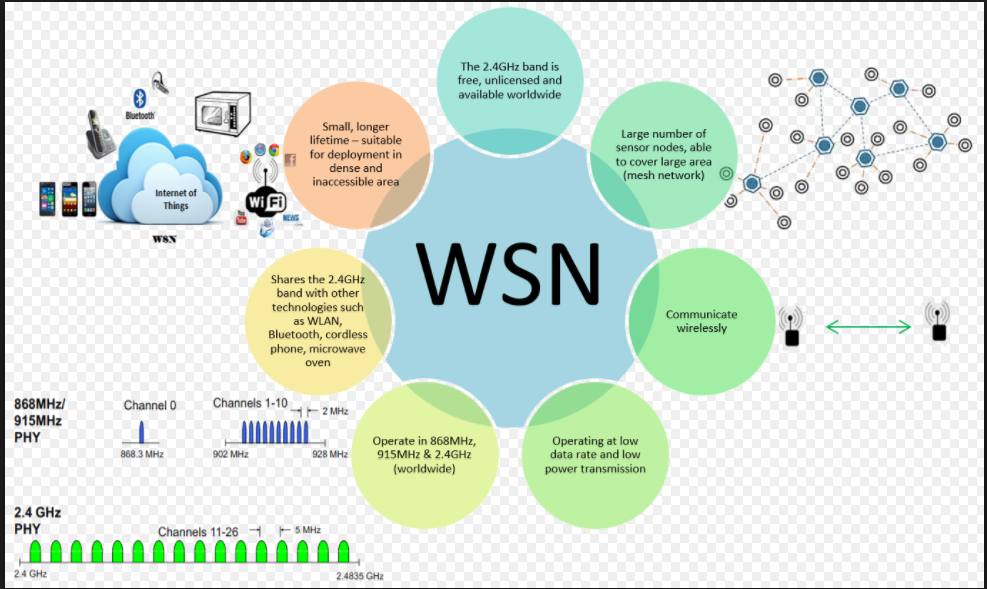


Fig. 1: Wireless Sensor Networking [1]

Wireless sensor devices give quick response to queries sent from a “control site” to perform specific task or supply sensing samples. Working modes of sensor nodes are continuous and sometime or event driven. Global Positioning System (GPS) and local positioning algorithms can be used to acquire location and positioning information.

WSN is a network consisting of dispersedly distributed autonomous devices which uses sensor to monitor physical or environmental conditions such as temperature, pressure, vibration, or pollutants [1]. Military applications motivated the development of wireless sensor networks. In Fig. 2, wireless sensor network architecture is depicted which consist of sensor nodes, Base-Station, and Gateway. The sensor nodes which are also called motes collect data from the environment. The data collected is in the form of packets is transmitted to the base station mote which is attached to the computer. The base station and motes are programmed with TinyOS/NesC.

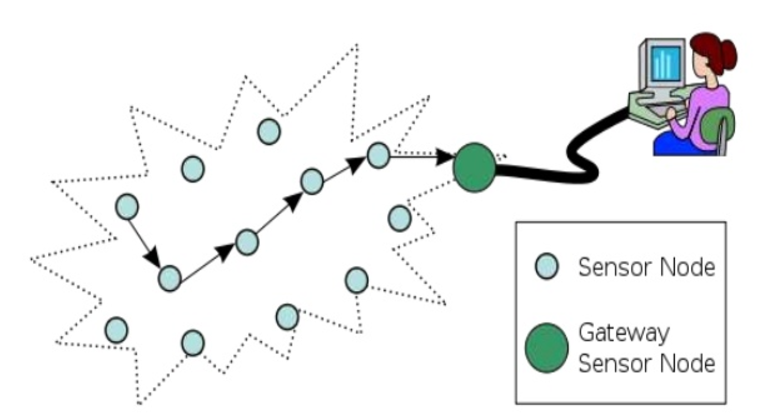


Fig. 2: Wireless Sensor Network Architecture [2]

# **2. Applications of Wireless Sensor Networks**

There are wide range of applications of WSNs. WSNs are used both commercially and in industries to monitor the data that would be difficult to monitor using wired sensors. These sensor motes can be deployed in remote areas without the need to recharge/ replace their power supplies [6]. There are various areas where WSNs are used. WSNs are applied for-

* Medical monitoring
* Agriculture Monitoring
* Military surveillance
* Environmental monitoring
* Structural Health Monitoring

**Medical Monitoring**

Wireless sensor network has major applications in health sector. There are various sensors deployed in hospitals to track and monitor patients and all medical resources. There are various sensors which can measure blood pressure, electrocardiograph (ECG) and body temperature. These days body sensor network (BSN) is used in health monitoring system [3].

**Agriculture Monitoring**

Nowadays, agriculture monitoring field is a significant field of control. A smart WSNs monitor air quality, soil conditions and weather situations. WSNs collect data and process it significantly. The temperature and humidity sensors helps in monitoring the growth of crops, thus increasing their productivity. This real -time monitoring of crops helps in maintaining the precision agriculture.

**Military surveillance**

The original motivation behind the research of WSNs is the military application. As the time elapsed the cost of communication network and sensor nodes decreased and thus application area increased with this phase. WSNs deployed for surveillance used to provide battlefield intelligence regarding the detection of chemical, nuclear weapons, identity of troops and vehicles. US Defense Advanced Research Project Agency (DARPA) started several programs [4].

**Environmental Monitoring**

WSN is significant in monitoring environmental conditions. The network of sensor nodes helps in monitoring several environmental parameters such as underground water level, wind direction and speed, atmosphere humidity and rainfall etc. WSN introduced inexpensive and low power sensor technology to monitor the environmental conditions. The weather forecasting uses this advanced technology. As the sensors are resource constrained, so we can use multiple sensors to monitor the environment conditions. These sensors can be deployed in different environments such as remote or hostile at the affordable cost. And we can use multi hop communication in the areas where direct communication with the base station is difficult [5].

**Structural Health Monitoring**

The process to determine the damage in structures of buildings or engineering structures is called structural health monitoring. To have the structural monitoring to work in efficient way, we deploy the sensor nodes in the structures [6]. This will monitor the damages in easy way. For instance, the Golden Gate Bridge uses the sensor nodes, this helps in the maintenance of the bridge.

# **3. Wireless Sensor Node**

There are various nodes/ sensors deployed in a field that respond to the message sent by the base station. In this network, message is broadcasted throughout the network and the node that is supposed to read it and sends a reply message back to the base station. If the node is not able to make any direct communication with base station, then the message is routed through the nodes to the base station. The data is collected and transferred through motes in the four stages which are:

* Collecting the data
* Processing the data
* Packaging the data
* Communicating the data

Various types of sensors on motes collects the data. After collecting the data, using its electronic brain the mote processes the data. Once the data has been processed, the brain packages the data into the form which is handled easily. This process is called enveloping. Once the data has been collected and processed. To this point, the data has been collected and processed. After this point, the mote begins to interact with other motes. These networks can use different types of wireless technologies such as IEEE 802.11 wireless LANs, Bluetooth, and radio frequency identification (RFID). But nowadays, most of action is with low- power radios that have arrange of about 30 to 200 feet and data of up to around 300 K bit/sec [8].



Fig. 3: Sensor Node Architecture [8]

**Embedded Processor**

Embedded processor plays a great role in a sensor node. The processor controls the functionality of the other hardware components in the network. It processes the data and program the tasks. There are numerous types of embedded processors that can be used in a sensor node include Microcontroller, Digital Signal Processor (DSP), Field Programmable Gate Array (FPGA) etc. Microcontroller is the most widely used processor because of its flexibility to connect to other devices [8]. For example, the most advanced CC2531 development board provided by Chipcon uses 8051 microcontrollers, and the Mica2 Mote platform provided by Crossbow uses ATMega128L microcontroller [8]. In my project, I used XM1000 mote based on TelosB platform which uses TI MSP430F2618 Microcontroller.

**Transceiver**

The functionality of transmitter and receiver are combined into a single device called transceiver. Transmit, receive, idle and sleep are operational states of transceivers. Transceivers have built in state machines that perform operations automatically. The common wireless transmission media are RF (Radio Frequency), Optical communication (Laser) and infrared [8].

**Memory**

The most applicable kinds of memory are the on-chip memory of microcontroller and off chip memory i.e. flash memory. The reason for using flash memories is due to their low cost and low storage capacity.

**Power**

The significant aspect of development of sensor node is ensuring that there is enough energy to power the system. The wireless sensor node consume power in communicating, sensing and data processing. The major part of energy is consumed in communicating. The batteries both rechargeable and non-rechargeable are used for sensor nodes. The most common electromechanical material used for node batteries are Nickel-cadmium (NiCd), Nickel-Zinc (NiZ), NIMH (nickel-metal hydride), and lithium-ion [9].

**Sensors**

Wireless sensor nodes use sensors to capture data from the environment. These hardware devices produce a significant response to the change in the physical condition like temperature and pressure. Due to restricted bandwidth and power, Wireless Sensor Network devices reinforce only low-data-rate sensing. There are several applications which require multi modal sensing, as a result each device may have several sensors on board. The sensors are used specifically as per the requirement of the application. For example, they may include temperature sensors, light sensors, humidity sensors, pressure sensors, accelerometers, magnetometers, chemical sensors, acoustic sensors, or even low-resolution images.

**GPS**

It is significant for all sensor measurements to be location inscribed in numerous WSN applications. To obtain positioning of the system, we need to pre-configure sensor locations at time of deployment, but this is possible in less number of deployments. The operations which are outdoor and where the network is ad hoc, there we require the most easily obtained Global Positioning Systems (GPS) [9].

# **4. Technologies**

## **4.1 Sensor Nodes**

The project has been implemented using XM1000 motes produced by Advanticsys Company [10]. The XM1000 is the new generation mote module which is based on TelosB platform [10]. The specifications of XM1000 are discussed below:

* This is upgraded to 116 Kb-EEPROM and 8Kb-RAM.
* It has TI MSP430F2618 Microcontroller.
* Radio module is CC2420, IEEE 802.15.4 2.4 GHz.
* Integrated with temperature, humidity and light sensors.
* 3X LEDs representing sensors.
* User and Reset Buttons.
* It is compatible with TinyOS 2.x and ContikiOS.
* Two AA Batteries holder.



Fig. 4: XM1000 Mote Module

## **4.2 TinyOS**

TinyOS is a component based embedded operating system developed for low power wireless devices such as mote which are basically used in wireless sensor networks. TinyOS shifted the event-based model which was developed for clusters into a very light-weight form for Motes. The communication network with TinyOS usage scale thousands or millions of nodes within a single network and is flexible to dynamic topology changes. It must also be tolerant of failures due to lossy links or failed nodes and it supports concurrent intensive operations required by networked sensors with minimal hardware requirements. TinyOS is tolerant to lossy links and failed nodes [13].

The three TinyOS features are as follows:

* Component Model: In the component model small reusable codes are compared with the larger abstractions.
* Concurrent Execution Model: TinyOS enables the components to perform the different tasks at the same time consuming less RAM for that.
* Application Programming Interface: In this interface application code make the timers to act periodically, after the node get booted up, the sensors start sensing the data

Application

Timer

Radio

Routing

Sensors

Fig. 5: TinyOS Application Architecture [13]

TinyOS introduced the simple method of developing and implementing wireless sensor networks. The programming language used for TinyOS is special C that is compiled using a special compiler called NesC. Thus, component based operating system uses event based programming with efficient component model [14].

The TinyOS group has currently demonstrated the ability to deploy a self-configuring network of devices, but this needs to be increased to support larger collections of devices. This can be done by improving networking protocols and by decreasing the total amount of data being communicated back to a central location through automatic data aggregation inside the networks. As the Motes, will be widely distributed, system power consumption will be limited, and software can play a major role in controlling the efficient use of the various hardware resources [14]. Thus, future revisions of the software will focus on reducing power consumption as their major goal.

## **4.3 NESC**

NesC (Network Embedded Systems C) is also called "NES-see", and it is dialect of C language. NesC is a component based [event-driven](http://en.wikipedia.org/wiki/Event-driven_programming) programming language. And it is used to build applications like wireless sensor networks [16]. NesC is an extension to C programming language.

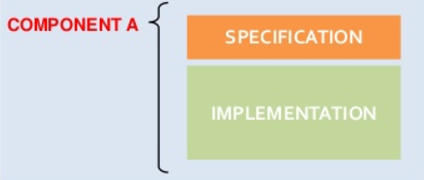


Fig. 6: Component Model in NesC [16]

***The separation of construction and composition****:* Programs of NesC are assembled out of components. The components internally maintain concurrency in the form of tasks or processes. The control threads may be passing into a component via its provided interfaces.

Component

Configurations

Modules

Provides, uses, and implements interfaces

A collection of other components put together by ‘wiring’ syntax.

Fig. 7: NesC Components

***The Interfaces are bidirectional in NesC programming language:*** Interfaces describe a set of functions to be implemented through interface’s provider (commands) and a set to be implemented by the interface's user [17]. This bidirectional feature of interfaces allows a single interface to illustrate a complex interaction between them. It is crucial because all lengthy commands in TinyOS are non-blocking stage; their completion is signaled through an event.

***The components of NesC programming language are statically linked to each other through their******interfaces****:* This allows for better static analysis of the programs and this increases the runtime efficiency. By linking through interfaces, the design becomes more robust [17].

***NesC programming language is modelled under the expectation of code:*** This will be produced by whole program compilers. This should also allow for better code analysis and generation.

## **4.4 Python**

Python was first released in 1991 and was created by “Guido van Rossum”. Python is a high-level programming language which is widely used for general-purpose programming. Python has a design philosophy which mainly emphasizes on code readability. The syntax of python allows code to be expressed in fewer lines by programmers unlike in languages such as C++ or JAVA. Both on small and large scales, python provides constructs that enable clear programming. In this project Python is used for monitoring the received packets from the Base Station on the PC.

## **4.5 TKINTER Python**

The name *Tkinter* comes from *Tk interface*. Tkinter was written by Fredrik Lundh. Tkinter is Python's de-facto standard GUI (Graphical User Interface) package. It is a thin object-oriented layer on top of Tcl (Tool Command Language). Tkinter is implemented as a Python wrapper around a complete Tcl interpreter which is embedded in the Python interpreter. In order to fed to this interpreter which is embedded, Tkinter calls are translated into Tcl commands and thus making it possible to mix Python and Tcl in a single application [21].

## **4.6 Adafruit IO**

Adafruit IO is a system that makes data useful. It mainly focuses on ease of use and allows data connections that are simple thereby requiring little programming. IO includes client libraries that wrap REST and MQTT APIs. IO is built on Ruby on Rails, and Node.js. Adafruit IO is currently in beta. Interaction with the server is significantly simplified by the Adafruit IO client libraries listed below

* Ardino
* Ruby
* Python
* Node.js

## **4.7 MQTT (Message Queue Telemetry Transport)**

MQTT is a protocol supported by Adafruit IO for communication of devices. By using a MQTT library or client one can publish and subscribe to a feed for sending and receiving feed data. MQTT is a Client Server publish/subscribe messaging transport protocol. MQTT was invented in 1999 by Arlen Nipper (Arcom, now Cirrus Link) and Andy Stanford-Clark (IBM). It has considerable characteristics such as it is designed in such a way that it is implemented with ease. It is simple, open and light weight. These characteristics of MQTT make it more suitable for using it in many situations which includes constrained environments such as for communication in Internet of Things (IoT) and Machine to Machine (M2M) contexts where bandwidth of network is at a premium or a small code footprint is required.

MQTT is a binary protocol and very light weight and **excels while transferring data over the wire when compared to protocols like HTTP**. It is because it has a minimum packet overhead. Another important aspect of MQTT is that on the client side, it can be extremely easy to implement. This fits perfectly for limited resources constrained devices. When MQTT was invented this was one of the goals. **MQTT is an OASIS Standard**. The newest version of the protocol is MQTT 3.1.1.

# **5. Architecture and Design**

## **5.1 Requirements**

* The objective of this application is to monitor the temperature, humidity and light of XM1000 motes and provide the intrinsic details to the user remotely. In my project, I have taken two remote motes and one base station mote. The temperature, humidity and light of remote motes is broadcasted by the base station and is displayed on PC with python application
* The sensed data of motes is depicted on web which is given to a remote server and clients can access the data from anywhere with internet connectivity. The web based application is scalable and user friendly.
* The multi hop network of motes is created, in which two remote motes and one base station mote is taken and temperature, humidity and light of mote is measured.
* To come up with a routing algorithm (Dijkstra’s Algorithm) with the best way to route the packets from the sensor nodes to the destination that is the PC using multi-hop communication protocols.

## **5.2 Architecture**

The gateway mote (Base Station) is attached to typically a USB port that communicates to the rest of the motes in the field. All the sensors send data to the base station mote that further sends it serially through python application that deciphers the information to yield meaningful data. Web server is accessed using Internet, the user can access and control temperature, humidity and light readings for analysis through a web interface. The server used in this project is Adafruit IO.

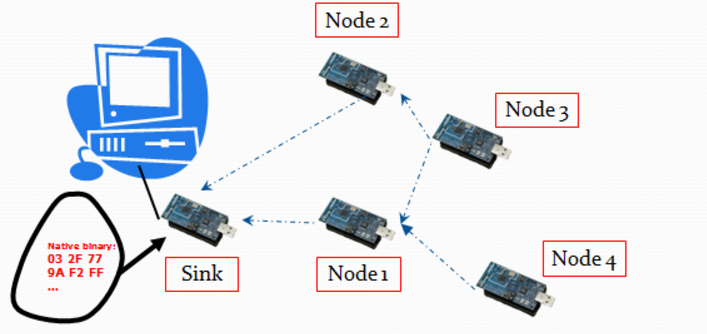


Fig. 8: Architecture of WSN

# **6. Transporting data from Mobile Mote to Base Station Mote.**

Firstly, one mote is programmed as base station and the other mote is programmed as the mobile mote. The mobile mote and base station mote are programmed in such a way that mobile mote has to sense the temperature, humidity and light from the environment and transport this data to the base station. The data collected at the base station is displayed on the PC by creating a GUI. Appendix 1 represents the codes used to program the motes and the code used to create a GUI that monitors the temperature, humidity and light sensed by the mobile mote. Results of GUI created are shown below.

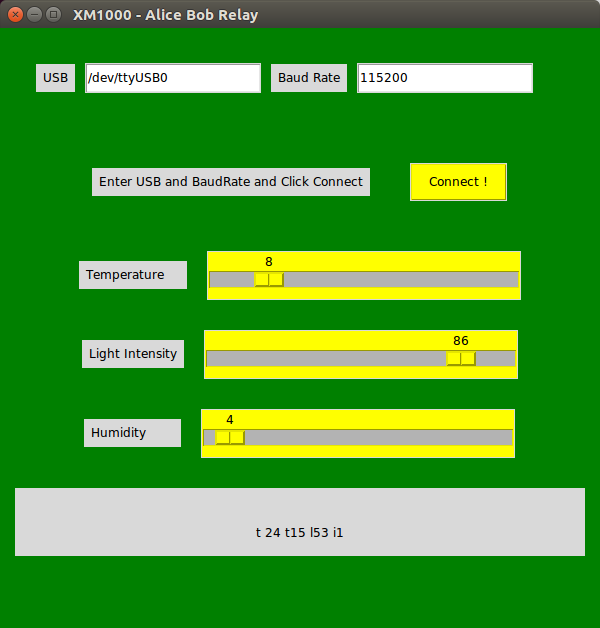


Fig. 9: GUI for data collected at base station

# **7. Multi Hop Routing in Wireless Sensor Networks**

WSNs do not have a fixed communication infrastructure and the nodes connect to their neighbors in an ad hoc manner. In a WSN each node may use other nodes to send a message to a remote target node. Two nodes can establish a communication link between themselves if they are in the radio range of each other. Due to different radio ranges of nodes, the communication channels can be unidirectional. The nodes that are not in the radio range of the sink node can send their messages to sink by multi hop data transmission [20]. In this part of project, I implemented a multi hop temperature, humidity and light monitoring in wireless sensor networks using XM1000 motes.

Three motes have been used in this application. The base station mote is considered to be bob and is connected to the USB port. Alice is placed at a distance and mediator is placed between the Alice and bob. When any of the temperature, humidity or light parameters is varied at Alice, the data is passed to bob which is displaced on the PC using GUI. In order to make sure that data is passed from Alice to bob via the mediator mote, the leds on motes are programmed accordingly. This programming can be seen in appendix 2.2. Three leds of the Alice mote start blinking only when it sends data to the mediator. The mediator starts blinking only when it receives the data from Alice and the received data is sent to bob. Bob which is connected to PC starts blinking only when it receives the data from mediator and the data displayed on the GUI changes accordingly. This application is carried out in a room and in the crop fields. The maximum range for which the Alice bob relay works is for 10meters distance from Alice to mediator and mediator to bob.

Relay technologies have the potential to offer extended cell coverage and improved capacity over the Next Generation Wireless Broadband Radio Access Networks. Standards development organizations are considering how to incorporate relay technologies into new standards.

## **7.1 Experiment set up in a closed room:**

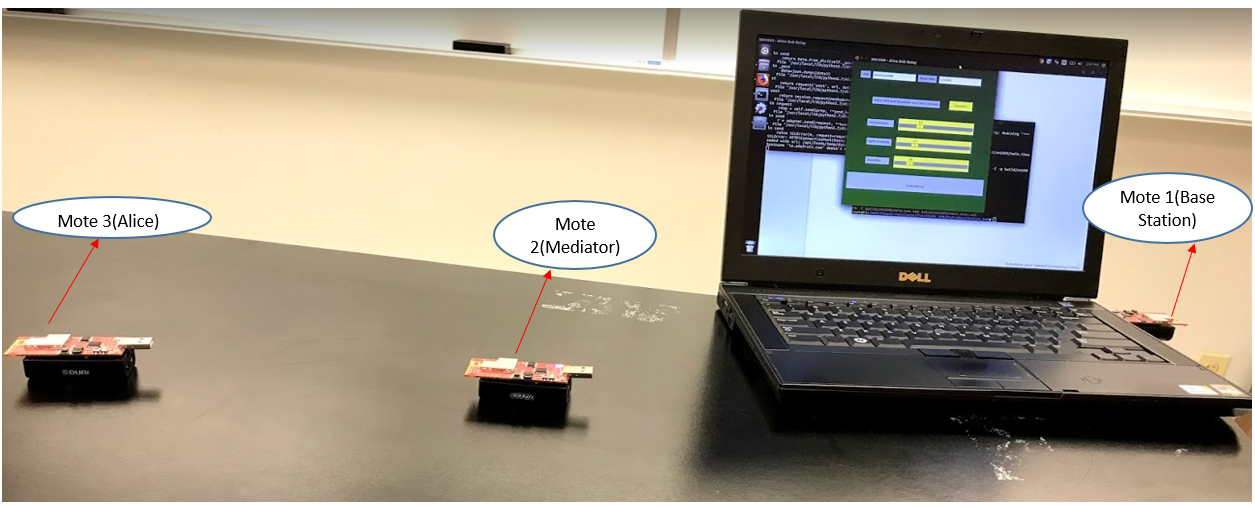


Fig. 10: Experiment set up of multi-hop routing in a closed room.

**Results of GUI:**

****

Fig. 11: GUI for Multi-hop Routing in Closed room.

## **7.2 Experiment setup in field:**



Fig. 12: Experiment setup in field.

**Results of GUI in field:**



Fig. 13: GUI for Multi-hop Routing in Field.

# **8. Web Application**

The web application is basically client- server software application. This software application uses web browser and web technology to run tasks over the internet. Web Application is coded in languages such as JavaScript and HTML, as these languages are browser supported.

## **8.1 Adafruit Io**

The web server is required by the Web Application to manage the requests from the client. The application server is used to complete the tasks which are requested and we can store the information through database. Adafruit IO is a system that makes data useful. It mainly focuses on ease of use and allows data connections that are simple thereby requiring little programming. IO includes client libraries that wrap REST and MQTT APIs. IO is built on Ruby on Rails, and Node.js. Adafruit IO is currently in beta. Interaction with the server is significantly simplified by the Adafruit IO client libraries.

## **8.2 MQTT**

While everyone is more familiar with HTTP protocol, since it is used for almost most of the websites, for each data transfer in HTTP, it requires each connection. To be a bit clear, it requires each connection for data to be written or data to be read. However, it is useful whenever there is a requirement of huge data like websites and IOT connections. It is not so fast and light weight. It is not fast because it is pull only i.e. it can always send data to server when it wants but has to wait in order to pull data from the server which consumes a lot of time. For these reasons MQTT turns out to be a great protocol. It is immensely simple and considerably light weight. It has no tear down or build up overhead since it takes only almost 80 bytes to connect to server and data publication and subscription takes place at a time. We can very easily stream data in and out of multiple topics very quickly. MQTT runs on top of any kind of network, it can either be a TCP/IP, mesh network, Bluetooth, etc. The MQTT style used in adafruit.io is the TCP/IP connection. TCP/IP is one to which WiFi, Cellular and Ethernet can be connected pretty easily and this makes it simple to get connected to adafruit.io.

## **8.3 Web Application Server**

In order to start with adafruit.io one should create an account using an Account username and key this is nothing but password. One can login in anytime with this login details. The core part of Adafruit IO is the “feed”- Basically feed is a set of data that can be read and written from a sequential file. One can add the data to feed or receive the data that is added at the latest. The metadata of the data pushed to Adafruit Io is held by the feeds. It contains the info of the data such as if it is private or public and to which license does the stored data falls into. For each source of data, on feed is to be created. For example, in this project I used a temperature sensor, a humidity sensor and a light sensor. So I had to create three feeds. One can always edit and delete feeds according to one’s convenience.

Once the feeds are created, u can plot graphs as needed. We can plot all the feeds on a single screen using a dashboard. This works as a remote server. Anyone with the username and password can control the data from anywhere. This feature is mainly useful in irrigation or farming. This data from remote server can be accessed and controlled by clients from anywhere with internet connectivity thus, providing ease in communication.

**8.4 Results:**

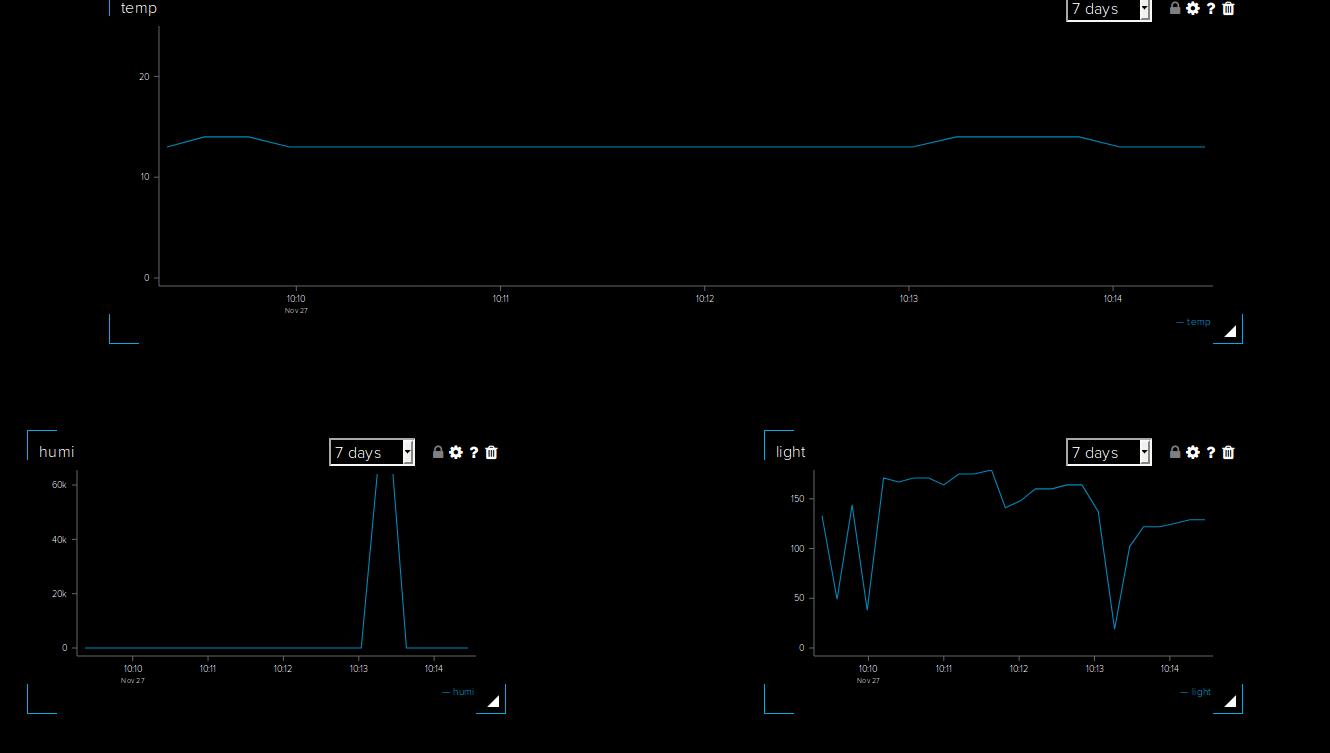
The results of my feeds and the dashboard created for my sensor data is as shown in Fig. 14.

Fig. 14: Dashboard for the temperature, humidity and light sensed through multi-hop routing.

# **9. Dijkstra’s Algorithm**

One algorithm used to find the shortest path to a target node from a starting node in a weighted graph is Dijkstra’s algorithm. Dijkstra's algorithm has many variants but the most common one is to find the shortest paths from the source vertex to all other vertices in the graph. The [algorithm](https://brilliant.org/wiki/algorithms/?wiki_title=algorithm) creates a [tree](https://brilliant.org/wiki/trees-basic/) of shortest paths from the starting vertex, the source, to all other points in the graph. Three values are to be initialized in the graph [22].

dist- an array of distances from source node to target nodes and is initialized as dist(s) = 0 and to all other nodes as dist (v)=α. This should be done while starting the process because as the algorithm proceeds, distance from source node to each other node is recalculated and when the shortest distance is found it is recorded.

Q- Q represents the queue of all nodes in the graph and this Q will get empty when the algorithm ends.

S- S represents an empty set and is filled by each node that the algorithm has visited. By the end of the algorithm, it contains all the nodes.

## **9.1 Algorithm Procedure:**

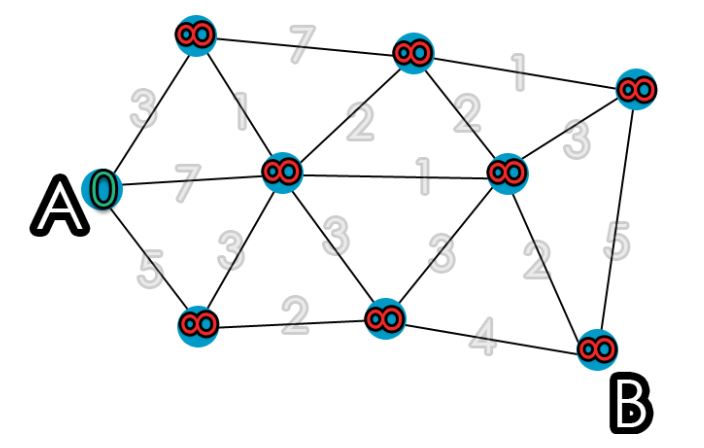


Fig. 15: Dijkstra Algorithm Procedure [22].

* While ‘Q’ is not empty, pop the node ‘v’, that is not already in ‘S’, from with the smallest dist (v). In the first run, source node will be chosen because dist (v) was initialized to 0. In the next run, the next node with the smallest value is chosen.
* Add node ‘v’ to ‘S’, to indicate that ‘v’ has been visited
* Update ‘dist’ values of adjacent nodes of the current node ‘v’ as follows: for each new adjacent node ‘u’,
* If dist (v) = w (u, v) < dist (u), there is a new minimal distance found for ‘u’, so update dist(u) to the new minimal distance value.
* Otherwise, no updates are made to dist (u).

The algorithm has visited all nodes in the graph and found the smallest distance to each node. ‘dist’ now contains the shortest path tree from source.

## **9.2 Experiment setup Representation:**

I have used four motes in this project to find the shortest path from source to destination. The motes are placed as shown in Figure15. The paths from mote ’2’ to mote ’M’ can be either 2-3-M or 2-1-M. there is no direct path from mote ’2’ to mote ‘M’. Therefore in order to find the shortest distance from mote ‘2’ to mote ‘M’, we used the Dijkstra’s Shortest Path Algorithm.

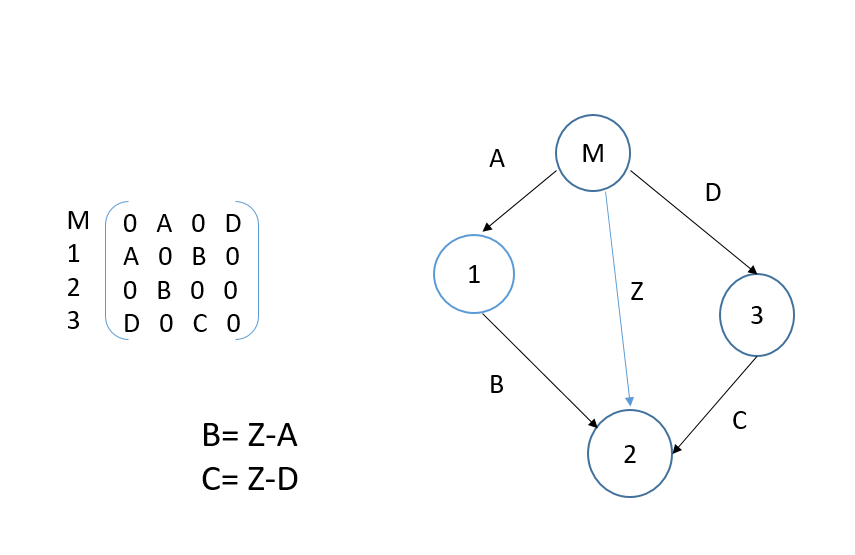


Fig. 16: Experiment setup representation and matrix.

To find the shortest path, the distance is to be calculated from source node to each other node. Calculating distance directly from one node to other in real time is really not so easy. So, the motes are placed in respective positions and the RSSI (Received Signal Strength Indicator) value is calculated from each node to the source node. Though RSSI is greatly impacted by the scatterings, reflections and so many other physical properties, it is believed that RSSI can be used to evaluate distance. Although RSSI is a poor distance estimator, it is used to estimate distance in several applications [24].

Firstly each mote is programmed as per requirement. That means the mote connected to PC is programmed as base station or bob. The other three motes are programmed as the sending motes and may be named as alice1, alice2, alice3. They are programmed using the following command. Base station is programmed by the command:

/opt/tiny-os/apps/tutorials/Rssi/RssiBase/----🡪Computer mote

make xm1000 install bsl,/dev/ttyUSB0

All the other three motes are programmed by giving the command as: /opt/tiny-os/apps/tutorials/Rssi/SendingMote

make xm1000 install.1 bsl,/dev/ttyUSB0

The RSSI values are received from the three motes and the corresponding distance is estimated from the below table.

|  |  |
| --- | --- |
| Distance in cm | RSSI Value |
| 0 | 30 |
| 15 | 21 |
| 30 | 8 |
| 45 | 6 |
| 60 | 4 |
| 75 | 0 |
| 90 | -2 |
| 110 | -12 |
| 150 | -14 |
| 300 | -17 |
| 450 | -29 |
| 800 | -37 |
| 1000 | -42 |

Table1: RSSI value versus Distance in cm

This table is used in the code and different distance values are assigned to different RSSI values. It is to be noted that there can be some error and the values received are not accurate due to variations in the environment. A weighted matrix is created for these values and the shortest distance is calculated accordingly. This can be clearly understood in appendix 3.1. If any of the motes distance is changed, the algorithm immediately checks for the newest formed shortest path and the new shortest path is displayed on the PC. This is shown in appendix 3.2 and 3.3.

## **9.3 Experiment Set up:**

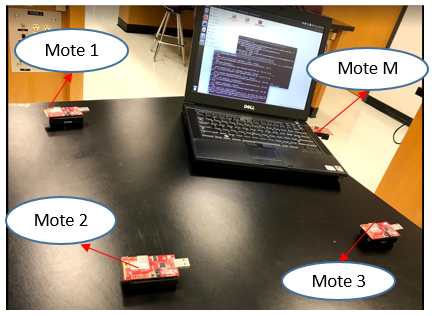


Fig. 17: Experiment set up for Algorithm.

**Results:**

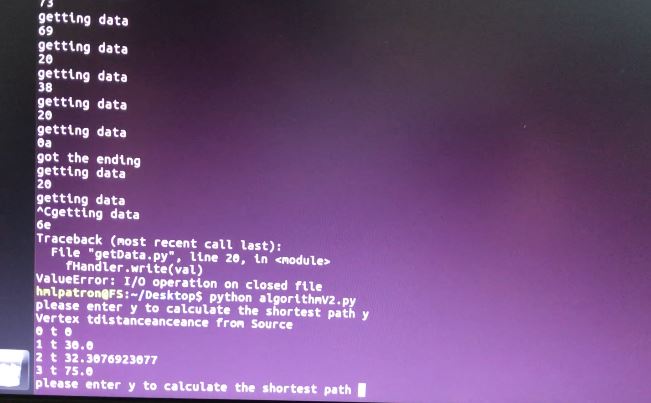


Fig. 18: Results of Dijkstra Algorithm.

## 

## **9.4 Experiment setup in field:**

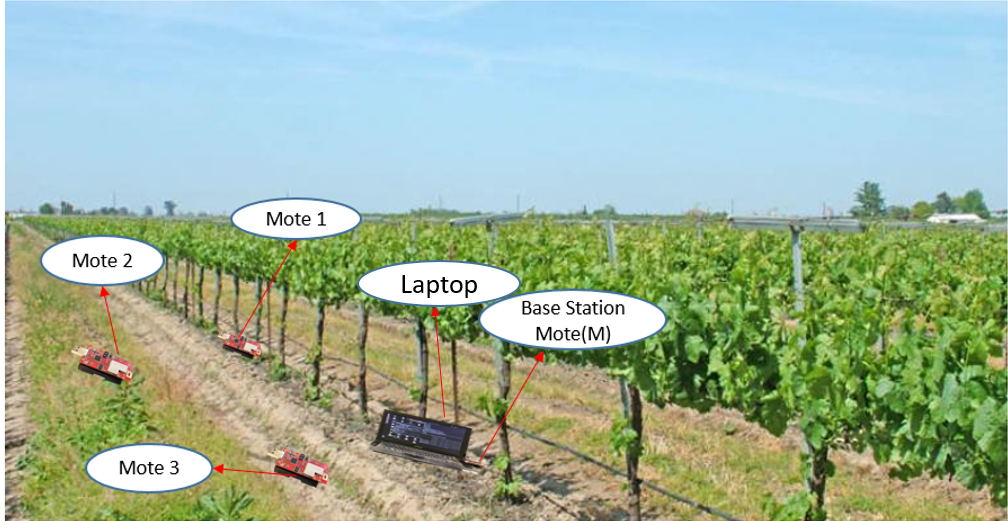


Fig. 19: Experiment set up in field.

**Results:**

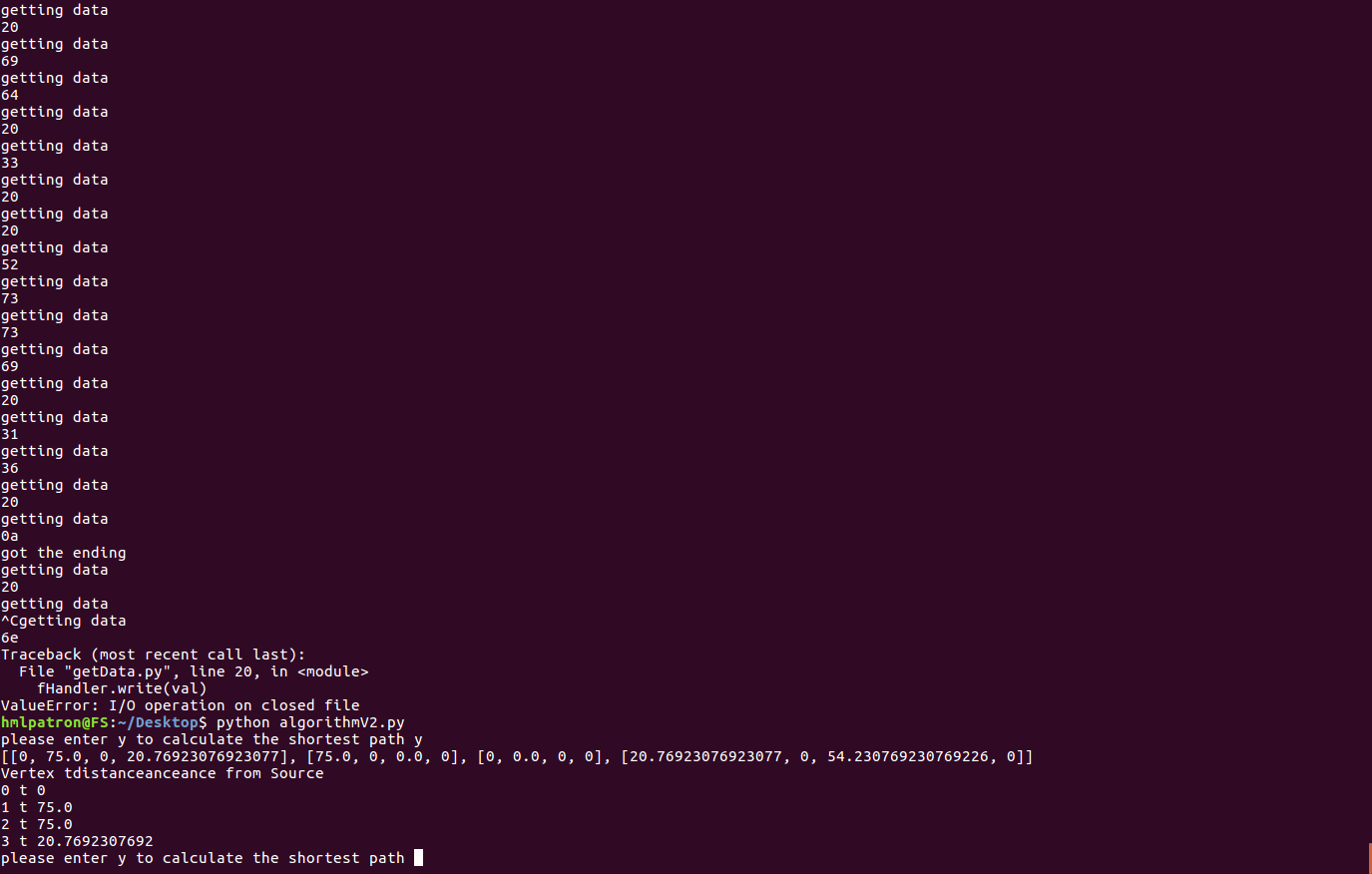
****

Fig. 20: Results of Dijkstra Algorithm in field.

Hence the shortest path algorithm is used in such a way to find the shortest path from source node to the destination node.

# **10. Summary**

## **10.1 Analysis**

* In this project, I was successful to achieve the following goals.
* Successfully installed TinyOS platform on PC. First, I checked the operation of XM1000 motes by testing the blinking program on it and then Null program on it.
* To monitor the temperature, humidity and light from the motes, I programmed two motes one base station and another mobile mote.
* I have implemented the multi-hop routing, and used one mote as the base station mote one as the relay and the other as sending mote. The python program reads sensed data from the base station through serial port and displays this data on PC
* The Web Application is created to show the temperature, humidity and light of mote. The web application uses the MQTT protocol. The web application created is for a remote server and the data from this server can be controlled by clients from anywhere.
* After gathering the data and analyzing one can remotely control in a smart way the irrigation of the crops.
* I have come up with an algorithm that can best route the packets from source to destination in the shortest distance possible.

**10.2 Future Scope**

Given below are some possible improvements which can be done in the project-

* We can implement more data analysis and statistical algorithms.
* We can create some mobile applications that can receive the data.

**11. References**

[1] [Online Source] Accessed on 08/25/2017

<https://ceastech.unimap.edu.my/research/wsn>

[2] [Online Source] Accessed on 08/28/2017

<http://www-bsac.eecs.berkeley.edu/smartdust/>

[3] [Online Source] Accessed on 08/30/2017

[www.uonbi.ac.ke/conferences/WSN/day1/Applications.pdf](http://www.uonbi.ac.ke/conferences/WSN/day1/Applications.pdf)

[4] [Online Source] Accessed on 08/30/2017

“Web based application for wireless sensor network”

<https://pdfs.semanticscholar.org/50fd/b06446fca2538d719d162543e49296fb6145.pdf>

[5] Narayanan, Sarath. T. and Vineeth. V. (2016) Survey on Motes Used in Wireless Sensor Networks: Performance & Parametric Analysis. Wireless Sensor Network, **8**, 51-60. doi: [10.4236/wsn.2016.84005](http://dx.doi.org/10.4236/wsn.2016.84005).

[6] Mahmood-Ul-Hassan, Muhammad, Khan, Ansar& Shah, Ansar, “A Review on Wireless Sensor Networks: An Application Perspective” International Journal of Computer Science andInformation Security, 2016 14(11), 858-867.

[7]Hanfi, Rabiya, & Rai, Yogesh, “Wireless Sensor Network” International Journal of Engineering and Computer Science, International Journal of Engineering and Computer Science, 2016

[8] Luo RC, Chen O. Mobile sensor node deployment and asynchronous power management for wireless sensor networks. IEEE Transactions on Industrial Electronics. 2012 May; 59(5):2377-85

[9] Overview of Wireless Sensor Network [Online Source] Accessed on 08/31/2017

<https://www.intechopen.com/books/wireless-sensor-networks-technology-and-protocols/overview-of-wireless-sensor-network>

[10] [Online Source] Accessed on 09/01/2017

<https://www.advanticsys.com/shop/asxm1000-p-24.html>

[11] Chelloug SA. Impact of the temperature and humidity variations on link quality of xm1000 mote sensors. arXiv preprint arXiv:1501.01073. 2015 Jan 6.

[12] [Online Source] Accessed on 09/03/2017

http://www.sciencedirect.com/science/article/pii/S0950061812009233

[13] [Online Source] Accessed on 09/15/2017

<http://www.tinyos.net>

[14] [Online Source] Accessed on 09/15/2017

csl.stanford.edu/~pal/pubs/tos-programming-web.pdf

[15] [Online Source] Accessed on 09/22/2017

<https://shazkhan.wordpress.com/2011/03/11/basics-of-tinyos-and-nesc>

[16] Gay D, Levis P, Von Behren R, Welsh M, Brewer E, Culler D. The nesC language: A holistic approach to networked embedded systems. InAcm Sigplan Notices 2003 Jun 9 (Vol. 38, No. 5, pp. 1-11). ACM.

[17] Gay D, Levis P, Culler D, Brewer E. nesC 1.1 language reference manual.

[18] Introduction to Multi-hop networks [Online Source] Accessed on 10/10/2017

<https://link.springer.com/chapter/10.1007%2F978-1-4614-6357-3_1>

[19] Spachos, Petros, Liang Song, Dimitrios, & Hatzinakos, Dimitrios,” Wireless multihop networking for a scalable indoor temperature monitoring system,” *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th,* 1132-1133.

[20] Boutaba, Raouf, Murphy, Seán, & Banchs, Albert, “Special issue on Wireless Multi-Hop Networking for Infrastructure Access". *Computer Networks, 2010 54*(2), 175-177.

[21] [Online Source] Accessed on 10/20/2017

<https://docs.python.org/2/library/tkinter.html>

[22] [Online Source] Accessed on 11/01/2017

https://brilliant.org/wiki/dijkstras-short-path-finder/

[23] [Online Source] Accessed on 11/05/2017

<http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/>

[24] Benkic K, Malajner M, Planinsic P, Cucej Z. Using RSSI value for distance estimation in wireless sensor networks based on ZigBee. In Systems, signals and image processing, 2008. IWSSIP 2008. 15th international conference on 2008 Jun 25 (pp. 303-306). IEEE

[25] [Online Source] Accessed on 11/10/2017

<https://www.hackerearth.com/practice/notes/dijkstras-algorithm/>

**Appendix 1 Implementation of Base Station Mote Program(Appc.nc)**

#include "RadioCountToLeds.h"

configuration RadioCountToLedsAppC {}

implementation {

components MainC, RadioCountToLedsC as App, LedsC;

components new AMSenderC(AM\_RADIO\_COUNT\_MSG);

components new AMReceiverC(AM\_RADIO\_COUNT\_MSG);

components new TimerMilliC();

components ActiveMessageC;

components SerialPrintfC;

components new TimerMilliC() as T1;

components new TimerMilliC() as T3;

App.Boot -> MainC.Boot;

App.Receive -> AMReceiverC;

App.AMSend -> AMSenderC;

App.AMControl -> ActiveMessageC;

App.Leds -> LedsC;

App.MilliTimer -> TimerMilliC;

App.Packet -> AMSenderC;

}

## **A 1.1 Implementation of Base station Mote Program(c.nc)**

#include "Timer.h"

#include "RadioCountToLeds.h"

#include "string.h"

#include "stdio.h"

module RadioCountToLedsC @safe() {

uses {

interface Leds;

interface Boot;

interface Receive;

interface AMSend;

interface Timer<TMilli> as MilliTimer;

interface SplitControl as AMControl;

interface Packet;

}

}

implementation {

message\_t packet;

bool locked;

uint16\_t counter = 0;

uint16\_t current\_temp;

event void Boot.booted() {

call AMControl.start();

}

event void AMControl.startDone(error\_t err) {

if (err == SUCCESS) {

//call MilliTimer.startPeriodic(250);

}

else {

call AMControl.start();

}

}

event void AMControl.stopDone(error\_t err) {

// do nothing

}

event message\_t\* Receive.receive(message\_t\* bufPtr, void\* payload, uint8\_t len) {

if (len != sizeof(radio\_count\_msg\_t))

{

return bufPtr;

}

else

{

radio\_count\_msg\_t\* rcm = (radio\_count\_msg\_t\*)payload;

printf("received temp is %u humi is %u light is %u \n ",rcm->temp,rcm->humi,rcm->light);

return bufPtr;

}

}

event void AMSend.sendDone(message\_t\* bufPtr, error\_t error) {

if (&packet == bufPtr) {

locked = FALSE;

}

}

}

## **A 1.2 Implementation of Mobile Motes Program(Appc.nc)**

#include "RadioCountToLeds.h"

configuration RadioCountToLedsAppC {}

implementation {

components MainC, RadioCountToLedsC as App, LedsC;

components new AMSenderC(AM\_RADIO\_COUNT\_MSG);

components new AMReceiverC(AM\_RADIO\_COUNT\_MSG);

components new TimerMilliC();

components ActiveMessageC;

components SerialPrintfC;

//components new DemoSensorC() as Sensor;

components new TimerMilliC() as T1;

components new TimerMilliC() as T3;

components new SensirionSht11ReaderP() as Sht11;

components new AdcReadClientC() as lightSensor;

App.Boot -> MainC.Boot;

App.Receive -> AMReceiverC;

App.AMSend -> AMSenderC;

App.AMControl -> ActiveMessageC;

App.Leds -> LedsC;

App.MilliTimer -> TimerMilliC;

App.Packet -> AMSenderC;

App.TempReader -> Sht11.Temperature;

App.HumiReader -> Sht11.Humidity;

App.LightReader -> lightSensor;

}

## **A 1.3 Implementation of Mobile Motes Program(c.nc)**

#include "Timer.h"

#include "RadioCountToLeds.h"

#include "string.h"

#include "stdio.h"

module RadioCountToLedsC @safe() {

uses {

interface Leds;

interface Boot;

interface Receive;

interface AMSend;

interface Timer<TMilli> as MilliTimer;

interface SplitControl as AMControl;

interface Packet;

interface Read<uint16\_t> as TempReader;

interface Read<uint16\_t> as LightReader;

interface Read<uint16\_t> as HumiReader;

}

}

implementation {

message\_t packet;

bool locked;

uint16\_t current\_temp;

uint16\_t current\_light;

uint16\_t current\_humi;

event void Boot.booted() {

call AMControl.start();

}

event void AMControl.startDone(error\_t err) {

if (err == SUCCESS) {

call MilliTimer.startPeriodic(250);

}

else {

call AMControl.start();

}

}

event void AMControl.stopDone(error\_t err) {

// do nothing

}

event void TempReader.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

current\_temp = val;

}

//temp = -39.6 + (0.04 \* val)

}

event void LightReader.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

current\_light = val;

}

// lumi = 2.5 \* (val/4096.0)\*6250.0

}

event void HumiReader.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

current\_humi = val;

}

//humi = -2.0468 + (0.0367 \* val) + ((-1.5955/100000) \* val \* val)

//ref : https://www.advanticsys.com/wiki/index.php?title=Sensirion%C2%AE\_SHT11

}

event void MilliTimer.fired() {

call TempReader.read();

call LightReader.read();

call HumiReader.read();

dbg("RadioCountToLedsC", "RadioCountToLedsC: timer fired, counter is %hu.\n", counter);

if (locked) {

return;

}

else {

radio\_count\_msg\_t\* rcm = (radio\_count\_msg\_t\*)call Packet.getPayload(&packet, sizeof(radio\_count\_msg\_t));

if (rcm == NULL) {

return;

}

rcm->nodeId = (uint8\_t)1;

rcm->temp = current\_temp;

rcm->humi = current\_humi;

rcm->light = current\_light;

if (call AMSend.send(AM\_BROADCAST\_ADDR, &packet, sizeof(radio\_count\_msg\_t)) == SUCCESS) {

locked = TRUE;

}

}

}

event message\_t\* Receive.receive(message\_t\* bufPtr,

void\* payload, uint8\_t len) {

dbg("RadioCountToLedsC", "Received packet of length %hhu.\n", len);

if (len != sizeof(radio\_count\_msg\_t)) {return bufPtr;}

else {

radio\_count\_msg\_t\* rcm = (radio\_count\_msg\_t\*)payload;

//printf("received msg is %u",rcm->counter);

return bufPtr;

}

}

event void AMSend.sendDone(message\_t\* bufPtr, error\_t error) {

if (&packet == bufPtr) {

locked = FALSE;

}

}

}

## **A 1.4 Implementation of Python Program for GUI**

from Tkinter import \*

from tkMessageBox import \*

#from ttk import \*

import serial

import binascii

import re

def updateVal():

#scaleValue.set(float(ser.readline()))

#tempValue.set()

#humiValue.set()

#lightValue.set()

recvLine = ser.readline()

print(recvLine.split(" "))

liveStreamValue.set(recvLine)

values = recvLine.split(" ")

try :

tempValue.set(int(filter(str.isdigit, values[1])))

humiValue.set(int(filter(str.isdigit, values[2])))

lightValue.set(int(filter(str.isdigit, values[3])))

except :

pass

root.after(10,updateVal)

#liveStreamValue.set(ser.readLine())

def connectToBaseStation():

try:

global ser

ser = serial.Serial(usbEntry.get(),baudRateEntry.get())

liveStreamValue.set("connecion established sucessfully ")

except:

showwarning("Unable to Connect", "Check USB and Baud rate or else Reconnect Base Station")

updateVal()

def startLeds():

ser.write('C')

def stopLeds():

ser.write('D')

def changeLeds():

ser.write('E')

def skipLeds():

ser.write('F')

root = Tk()

root.title("XM1000 - Alice Bob Relay")

root["bg"] = "green"

root.geometry("600x600+300+300")

#variable for live stream

liveStreamValue = StringVar()

liveStreamValueOld2 = StringVar()

liveStreamValueOld1 = StringVar()

liveStreamValue.set(" ")

frameBaudRate = Frame(root,width=300,height=300,bg="green")

frameSettings = Frame(root, width=300, height=300,bg="green")

frameTempValue = Frame(root,width = 300,height=200,bg="green")

frameLightValue = Frame(root,width = 300,height=200,bg="green")

frameHumiValue = Frame(root,width = 300,height=200,bg="green")

frameStream = Frame(root,width = 300,height=300,bg="green")

frameMyName = Frame(root,width = 300,height=300,bg="yellow")

buttonConnect = Button(frameSettings, text="Connect !",bg="yellow",command = connectToBaseStation)

#buttonDisconnect = Button(frameSettings,text="Disconnect !",bg="red",command = disconnectToBaseStation)

usbEntryLabel = Label(frameBaudRate,text="USB")

usbEntry = Entry(frameBaudRate)

baudRateEntryLabel = Label(frameBaudRate,text="Baud Rate")

baudRateEntry = Entry(frameBaudRate)

buttonLabel = Label(frameSettings,text="Enter USB and BaudRate and Click Connect")

tempValue = Scale(frameTempValue,from\_=0, to=50,orient=HORIZONTAL,bg="yellow",length = 300,resolution=1)

tempLabel = Label(frameTempValue,text="Temperature ")

tempValue.set(10)

lightValue = Scale(frameLightValue,from\_=0, to=100,orient=HORIZONTAL,bg="yellow",length = 300,resolution=1)

lightLabel = Label(frameLightValue,text="Light Intensity")

lightValue.set(10)

humiValue = Scale(frameHumiValue,from\_=0, to=100,orient=HORIZONTAL,bg="yellow",length = 300,resolution=1)

humiLabel = Label(frameHumiValue,text="Humidity ")

humiValue.set(10)

liveStreamLabel = Label(frameStream,textvariable=liveStreamValue,height=2,width=600)

liveStreamOld1Label = Label(frameStream,textvariable=liveStreamValueOld1,height=1,width=600)

liveStreamOld2Label = Label(frameStream,textvariable=liveStreamValueOld2,height=1,width=600)

infoLabel = Label(frameMyName,text="yada yada yada .......",height=6,width=600,bg="white");

usbEntryLabel.pack(side=LEFT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

usbEntry.pack(side=LEFT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

baudRateEntryLabel.pack(side=LEFT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

baudRateEntry.pack(side=LEFT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

buttonLabel.pack(side=LEFT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

#buttonDisconnect.pack(side=RIGHT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

buttonConnect.pack(side=RIGHT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

tempLabel.pack(side=LEFT,ipadx=5,ipady=5,padx=5,pady=5)

tempValue.pack(side=RIGHT,ipadx=5,ipady=5,padx=5,pady=5)

lightLabel.pack(side=LEFT,ipadx=5,ipady=5,padx=5,pady=5)

lightValue.pack(side=RIGHT,ipadx=5,ipady=5,padx=5,pady=5)

humiLabel.pack(side=LEFT,ipadx=5,ipady=5,padx=5,pady=5)

humiValue.pack(side=RIGHT,ipadx=5,ipady=5,padx=5,pady=5)

liveStreamOld2Label.pack(side=TOP)

liveStreamOld1Label.pack(side=TOP)

liveStreamLabel.pack(side=TOP)

infoLabel.pack(side=LEFT,ipadx=5,ipady=5,padx=5,pady=5)

frameBaudRate.pack(ipadx = 15,ipady = 15, padx = 15,pady = 15)

frameSettings.pack(ipadx = 15,ipady = 15, padx = 15,pady = 15)

frameTempValue.pack(ipadx = 5,ipady= 5,padx=5,pady=5)

frameLightValue.pack(ipadx = 5,ipady= 5,padx=5,pady=5)

frameHumiValue.pack(ipadx = 5,ipady= 5,padx=5,pady=5)

frameStream.pack(ipadx = 15,ipady= 15,padx=15,pady=15)

frameMyName.pack(ipadx = 15,ipady= 15,padx=15,pady=15)

usbEntry.insert(0,"/dev/ttyUSB0")

baudRateEntry.insert(0,"115200")

#updateVal()

root.mainloop()

# **Appendix 2 Base station program for bob in multi-hop relay(Appc.nc)**

#include "RadioCountToLeds.h"

configuration RadioCountToLedsAppC {}

implementation {

components MainC, RadioCountToLedsC as App, LedsC;

components new AMSenderC(AM\_RADIO\_COUNT\_MSG);

components new AMReceiverC(AM\_RADIO\_COUNT\_MSG);

//components new TimerMilliC();

components ActiveMessageC;

components SerialPrintfC;

//components new TimerMilliC() as T1;

//components new TimerMilliC() as T3;

App.Boot -> MainC.Boot;

App.Receive -> AMReceiverC;

App.AMSend -> AMSenderC;

App.AMControl -> ActiveMessageC;

App.Leds -> LedsC;

// App.MilliTimer -> TimerMilliC;

App.Packet -> AMSenderC;

}

## **A 2.1 Base station program for bob in multi-hop relay(c.nc)**

#include "Timer.h"

#include "RadioCountToLeds.h"

#include "string.h"

#include "stdio.h"

module RadioCountToLedsC @safe() {

uses {

interface Leds;

interface Boot;

interface Receive;

interface AMSend;

interface SplitControl as AMControl;

interface Packet;

}

}

implementation {

message\_t packet;

bool locked;

uint16\_t counter = 0;

uint16\_t current\_temp;

event void Boot.booted() {

call AMControl.start();

}

event void AMControl.startDone(error\_t err) {

if (err == SUCCESS) {

//call MilliTimer.startPeriodic(250);

}

else {

call AMControl.start();

}

}

event void AMControl.stopDone(error\_t err) {

// do nothing

}

event message\_t\* Receive.receive(message\_t\* bufPtr, void\* payload, uint8\_t len) {

if (len != sizeof(radio\_count\_msg\_t))

{

return bufPtr;

}

else

{

radio\_count\_msg\_t\* rcm = (radio\_count\_msg\_t\*)payload;

printf("t%u h%u l%u i%u \n ",rcm->temp,rcm->humi,rcm->light,rcm->nodeId);

call Leds.led0Toggle();

call Leds.led1Toggle();

return bufPtr;

}

}

event void AMSend.sendDone(message\_t\* bufPtr, error\_t error) {

if (&packet == bufPtr) {

locked = FALSE;

}

}

}

## **A 2.2 Mediator Mote program in multi-hop relay(APPc.nc)**

#include "RadioCountToLeds.h"

configuration RadioCountToLedsAppC {}

implementation {

components MainC, RadioCountToLedsC as App, LedsC;

components new AMSenderC(AM\_RADIO\_COUNT\_MSG);

components new AMReceiverC(AM\_RADIO\_COUNT\_MSG);

//components new TimerMilliC();

components ActiveMessageC;

components SerialPrintfC;

//components new TimerMilliC() as T1;

//components new TimerMilliC() as T3;

App.Boot -> MainC.Boot;

App.Receive -> AMReceiverC;

App.AMSend -> AMSenderC;

App.AMControl -> ActiveMessageC;

App.Leds -> LedsC;

// App.MilliTimer -> TimerMilliC;

App.Packet -> AMSenderC;

## **A 2.3 Mediator Mote program in multi-hop relay(c.nc)**

#include "Timer.h"

#include "RadioCountToLeds.h"

#include "string.h"

#include "stdio.h"

#define AliceNodeId 1

#define mNodeId 2

module RadioCountToLedsC @safe() {

uses {

interface Leds;

interface Boot;

interface Receive;

interface AMSend;

interface SplitControl as AMControl;

interface Packet;

}

}

implementation {

message\_t packet;

bool locked;

uint16\_t counter = 0;

uint16\_t current\_temp;

event void Boot.booted() {

call AMControl.start();

}

event void AMControl.startDone(error\_t err) {

if (err == SUCCESS) {

//call MilliTimer.startPeriodic(250);

}

else {

call AMControl.start();

}

}

event void AMControl.stopDone(error\_t err) {

// do nothing

}

event message\_t\* Receive.receive(message\_t\* bufPtr, void\* payload, uint8\_t len) {

if (len != sizeof(radio\_count\_msg\_t))

{

return bufPtr;

}

else

{

radio\_count\_msg\_t\* rcm = (radio\_count\_msg\_t\*)payload;

//printf("t %u t%u l%u i%u \n ",rcm->temp,rcm->humi,rcm->light,rcm->nodeId);

if (locked) {

return rcm;

}

else {

radio\_count\_msg\_t\* rcmS = (radio\_count\_msg\_t\*)call Packet.getPayload(&packet, sizeof(radio\_count\_msg\_t));

if (rcmS == NULL) {

return rcmS;

}

call Leds.led2Toggle();

call Leds.led1Toggle();

call Leds.led0Toggle();

rcmS->nodeId = (uint8\_t)mNodeId;

rcmS->temp = rcm->temp;

rcmS->humi = rcm->humi;

rcmS->light = rcm->light;

if (call AMSend.send(AM\_BROADCAST\_ADDR, &packet, sizeof(radio\_count\_msg\_t)) == SUCCESS) {

locked = TRUE;

}

}

return bufPtr;

}

}

event void AMSend.sendDone(message\_t\* bufPtr, error\_t error) {

if (&packet == bufPtr) {

locked = FALSE;

}

}

}

## **A 2.4 Alice Mote program in multi-hop relay(Appc.nc)**

include "RadioCountToLeds.h"

configuration RadioCountToLedsAppC {}

implementation {

components MainC, RadioCountToLedsC as App, LedsC;

components new AMSenderC(AM\_RADIO\_COUNT\_MSG);

components new AMReceiverC(AM\_RADIO\_COUNT\_MSG);

components new TimerMilliC();

components ActiveMessageC;

components SerialPrintfC;

//components new DemoSensorC() as Sensor;

components new TimerMilliC() as T1;

components new TimerMilliC() as T3;

components new SensirionSht11C() as Sht11;

components new HamamatsuS1087ParC() as lightSensor;

App.Boot -> MainC.Boot;

App.Receive -> AMReceiverC;

App.AMSend -> AMSenderC;

App.AMControl -> ActiveMessageC;

App.Leds -> LedsC;

App.MilliTimer -> TimerMilliC;

App.Packet -> AMSenderC;

App.TempReader -> Sht11.Temperature;

App.HumiReader -> Sht11.Humidity;

App.LightReader -> lightSensor;

}

## **A 2.5 Alice Mote program in multi-hop relay(c.nc)**

#include "Timer.h"

#include "RadioCountToLeds.h"

#include "string.h"

#include "stdio.h"

module RadioCountToLedsC @safe() {

uses {

interface Leds;

interface Boot;

interface Receive;

interface AMSend;

interface Timer<TMilli> as MilliTimer;

interface SplitControl as AMControl;

interface Packet;

interface Read<uint16\_t> as TempReader;

interface Read<uint16\_t> as LightReader;

interface Read<uint16\_t> as HumiReader;

}

}

implementation {

message\_t packet;

bool locked;

uint16\_t current\_temp;

uint16\_t current\_light;

uint16\_t current\_humi;

event void Boot.booted() {

call AMControl.start();

}

event void AMControl.startDone(error\_t err) {

if (err == SUCCESS) {

call MilliTimer.startPeriodic(250);

}

else {

call AMControl.start();

}

}

event void AMControl.stopDone(error\_t err) {

// do nothing

}

event void TempReader.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

current\_temp = -39.6 + (0.01\*val);

}

//temp = -39.6 + (0.04 \* val)

}

event void LightReader.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

current\_light = 2.5 \*(val/4096.0)\*6250.0;

}

// lumi = 2.5 \* (val/4096.0)\*6250.0

}

event void HumiReader.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

current\_humi = -2.0468 + (0.0367 \* val) + ((-1.5955/100000) \* val \* val);

}

//humi = -2.0468 + (0.0367 \* val) + ((-1.5955/100000) \* val \* val)

//ref : https://www.advanticsys.com/wiki/index.php?title=Sensirion%C2%AE\_SHT11

}

event void MilliTimer.fired() {

call TempReader.read();

call LightReader.read();

call HumiReader.read();

call Leds.led0Toggle()

dbg("RadioCountToLedsC", "RadioCountToLedsC: timer fired, counter is %hu.\n", counter);

if (locked) {

return;

}

else {

radio\_count\_msg\_t\* rcm = (radio\_count\_msg\_t\*)call Packet.getPayload(&packet, sizeof(radio\_count\_msg\_t));

if (rcm == NULL) {

return;

}

rcm->nodeId = (uint8\_t)1;

rcm->temp = current\_temp;

rcm->humi = current\_humi;

rcm->light = current\_light;

if (call AMSend.send(AM\_BROADCAST\_ADDR, &packet, sizeof(radio\_count\_msg\_t)) == SUCCESS) {

locked = TRUE;

}

}

}

event message\_t\* Receive.receive(message\_t\* bufPtr,

void\* payload, uint8\_t len) {

dbg("RadioCountToLedsC", "Received packet of length %hhu.\n", len);

if (len != sizeof(radio\_count\_msg\_t)) {return bufPtr;}

else {

radio\_count\_msg\_t\* rcm = (radio\_count\_msg\_t\*)payload;

//printf("received msg is %u",rcm->counter);

return bufPtr;

}

}

event void AMSend.sendDone(message\_t\* bufPtr, error\_t error) {

if (&packet == bufPtr) {

locked = FALSE;

}

}

}

## **A 2.6 Implementation of Python Program for GUI**

from Tkinter import \*

from tkMessageBox import \*

#from ttk import \*

import serial

import binascii

import re

## adafruit io code

# Import library and create instance of REST client.

from Adafruit\_IO import Client

aio = Client('0406da2916a54c3b88e0a87969bd9fe5')

aioCount = 0

def updateVal():

#scaleValue.set(float(ser.readline()))

#tempValue.set()

#humiValue.set()

#lightValue.set()

recvLine = ser.readline()

recvLine = recvLine.replace(" ","")

matches = re.finditer("\D\d+",recvLine)

values = dict()

for i,match in enumerate(matches):

key = re.findall("\D",match.group())

value = re.findall("\d+",match.group())

values[key[0]] = value[0]

liveStreamValue.set(recvLine)

print values['i']

print values

if values['i'] == "2":

#liveStreamValue.set(recvLine)

tempValue.set(int(values['t']))

humiValue.set(int(values['h']))

lightValue.set(int(values['l']))

global aioCount

aioCount = aioCount + 1

if aioCount == 50 :

aio.send('temp',int(values['t']))

aio.send('humi',int(values['h']))

aio.send('light',int(values['l']))

aioCount = 0

root.after(10,updateVal)

#liveStreamValue.set(ser.readLine())

def connectToBaseStation():

try:

global ser

ser = serial.Serial(usbEntry.get(),baudRateEntry.get())

liveStreamValue.set("connecion established sucessfully ")

except:

showwarning("Unable to Connect", "Check USB and Baud rate or else Reconnect Base Station")

updateVal()

def startLeds():

ser.write('C')

def stopLeds():

ser.write('D')

def changeLeds():

ser.write('E')

def skipLeds():

ser.write('F')

root = Tk()

root.title("XM1000 - Alice Bob Relay")

root["bg"] = "green"

root.geometry("600x600+300+300")

#variable for live stream

liveStreamValue = StringVar()

liveStreamValueOld2 = StringVar()

liveStreamValueOld1 = StringVar()

liveStreamValue.set(" ")

frameBaudRate = Frame(root,width=300,height=300,bg="green")

frameSettings = Frame(root, width=300, height=300,bg="green")

frameTempValue = Frame(root,width = 300,height=200,bg="green")

frameLightValue = Frame(root,width = 300,height=200,bg="green")

frameHumiValue = Frame(root,width = 300,height=200,bg="green")

frameStream = Frame(root,width = 300,height=300,bg="green")

frameMyName = Frame(root,width = 300,height=300,bg="yellow")

buttonConnect = Button(frameSettings, text="Connect !",bg="yellow",command = connectToBaseStation)

#buttonDisconnect = Button(frameSettings,text="Disconnect !",bg="red",command = disconnectToBaseStation)

usbEntryLabel = Label(frameBaudRate,text="USB")

usbEntry = Entry(frameBaudRate)

baudRateEntryLabel = Label(frameBaudRate,text="Baud Rate")

baudRateEntry = Entry(frameBaudRate)

buttonLabel = Label(frameSettings,text="Enter USB and BaudRate and Click Connect")

tempValue = Scale(frameTempValue,from\_=0, to=50,orient=HORIZONTAL,bg="yellow",length = 300,resolution=1)

tempLabel = Label(frameTempValue,text="Temperature ")

tempValue.set(10)

lightValue = Scale(frameLightValue,from\_=0, to=100,orient=HORIZONTAL,bg="yellow",length = 300,resolution=1)

lightLabel = Label(frameLightValue,text="Light Intensity")

lightValue.set(10)

humiValue = Scale(frameHumiValue,from\_=0, to=100,orient=HORIZONTAL,bg="yellow",length = 300,resolution=1)

humiLabel = Label(frameHumiValue,text="Humidity ")

humiValue.set(10)

liveStreamLabel = Label(frameStream,textvariable=liveStreamValue,height=2,width=600)

liveStreamOld1Label = Label(frameStream,textvariable=liveStreamValueOld1,height=1,width=600)

liveStreamOld2Label = Label(frameStream,textvariable=liveStreamValueOld2,height=1,width=600)

infoLabel = Label(frameMyName,text="yada yada yada .......",height=6,width=600,bg="white");

usbEntryLabel.pack(side=LEFT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

usbEntry.pack(side=LEFT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

baudRateEntryLabel.pack(side=LEFT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

baudRateEntry.pack(side=LEFT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

buttonLabel.pack(side=LEFT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

#buttonDisconnect.pack(side=RIGHT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

buttonConnect.pack(side=RIGHT,ipadx = 5,ipady = 5, padx = 5,pady = 5)

tempLabel.pack(side=LEFT,ipadx=5,ipady=5,padx=5,pady=5)

tempValue.pack(side=RIGHT,ipadx=5,ipady=5,padx=5,pady=5)

lightLabel.pack(side=LEFT,ipadx=5,ipady=5,padx=5,pady=5)

lightValue.pack(side=RIGHT,ipadx=5,ipady=5,padx=5,pady=5)

humiLabel.pack(side=LEFT,ipadx=5,ipady=5,padx=5,pady=5)

humiValue.pack(side=RIGHT,ipadx=5,ipady=5,padx=5,pady=5)

liveStreamOld2Label.pack(side=TOP)

liveStreamOld1Label.pack(side=TOP)

liveStreamLabel.pack(side=TOP)

infoLabel.pack(side=LEFT,ipadx=5,ipady=5,padx=5,pady=5)

frameBaudRate.pack(ipadx = 15,ipady = 15, padx = 15,pady = 15)

frameSettings.pack(ipadx = 15,ipady = 15, padx = 15,pady = 15)

frameTempValue.pack(ipadx = 5,ipady= 5,padx=5,pady=5)

frameLightValue.pack(ipadx = 5,ipady= 5,padx=5,pady=5)

frameHumiValue.pack(ipadx = 5,ipady= 5,padx=5,pady=5)

frameStream.pack(ipadx = 15,ipady= 15,padx=15,pady=15)

frameMyName.pack(ipadx = 15,ipady= 15,padx=15,pady=15)

usbEntry.insert(0,"/dev/ttyUSB0")

baudRateEntry.insert(0,"115200")

#updateVal()

root.mainloop()

# **Appendix 3 Alice Mote program for Algorithm (Appc.nc)**

#include "RadioCountToLeds.h"

configuration RadioCountToLedsAppC {}

implementation {

components MainC, RadioCountToLedsC as App, LedsC;

components new AMSenderC(AM\_RADIO\_COUNT\_MSG);

components new AMReceiverC(AM\_RADIO\_COUNT\_MSG);

components new TimerMilliC();

components ActiveMessageC;

components SerialPrintfC;

//components new DemoSensorC() as Sensor;

components new TimerMilliC() as T1;

components new TimerMilliC() as T3;

components new SensirionSht11C() as Sht11;

components new HamamatsuS1087ParC() as lightSensor;

App.Boot -> MainC.Boot;

App.Receive -> AMReceiverC;

App.AMSend -> AMSenderC;

App.AMControl -> ActiveMessageC;

App.Leds -> LedsC;

App.MilliTimer -> TimerMilliC;

App.Packet -> AMSenderC;

App.TempReader -> Sht11.Temperature;

App.HumiReader -> Sht11.Humidity;

App.LightReader -> lightSensor;

}

## **A 3.1 Alice Mote program for Algorithm (c.nc)**

#include "Timer.h"

#include "RadioCountToLeds.h"

#include "string.h"

#include "stdio.h"

module RadioCountToLedsC @safe() {

uses {

interface Leds;

interface Boot;

interface Receive;

interface AMSend;

interface Timer<TMilli> as MilliTimer;

interface SplitControl as AMControl;

interface Packet;

interface Read<uint16\_t> as TempReader;

interface Read<uint16\_t> as LightReader;

interface Read<uint16\_t> as HumiReader;

}

}

implementation {

message\_t packet;

bool locked;

uint16\_t current\_temp;

uint16\_t current\_light;

uint16\_t current\_humi;

event void Boot.booted() {

call AMControl.start();

}

event void AMControl.startDone(error\_t err) {

if (err == SUCCESS) {

call MilliTimer.startPeriodic(450);

}

else {

call AMControl.start();

}

}

event void AMControl.stopDone(error\_t err) {

// do nothing

}

event void TempReader.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

//float valF = val;

current\_temp = -39.6 + 0.01\*val - 10;

}

//temp = -39.6 + (0.04 \* val)

}

event void LightReader.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

current\_light = 2.5 \*(val/4096.0)\*6250.0;

}

// lumi = 2.5 \* (val/4096.0)\*6250.0

}

event void HumiReader.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

current\_humi = -2.0468 + (0.0367 \* val) + ((-1.5955/100000) \* val \* val);

}

//humi = -2.0468 + (0.0367 \* val) + ((-1.5955/100000) \* val \* val)

//ref : https://www.advanticsys.com/wiki/index.php?title=Sensirion%C2%AE\_SHT11

}

event void MilliTimer.fired() {

call TempReader.read();

call LightReader.read();

call HumiReader.read();

call Leds.led0Toggle();

printf("temp is %u humi %u light %u",current\_temp,current\_humi,current\_light);

if (locked) {

return;

}

else {

radio\_count\_msg\_t\* rcm = (radio\_count\_msg\_t\*)call Packet.getPayload(&packet, sizeof(radio\_count\_msg\_t));

if (rcm == NULL) {

return;

}

rcm->nodeId = (uint8\_t)3;

rcm->temp = current\_temp;

rcm->humi = current\_humi;

rcm->light = current\_light;

if (call AMSend.send(AM\_BROADCAST\_ADDR, &packet, sizeof(radio\_count\_msg\_t)) == SUCCESS) {

locked = TRUE;

}

}

}

event message\_t\* Receive.receive(message\_t\* bufPtr,

void\* payload, uint8\_t len) {

dbg("RadioCountToLedsC", "Received packet of length %hhu.\n", len);

if (len != sizeof(radio\_count\_msg\_t)) {return bufPtr;}

else {

radio\_count\_msg\_t\* rcm = (radio\_count\_msg\_t\*)payload;

//printf("received msg is %u",rcm->counter);

return bufPtr;

}

}

event void AMSend.sendDone(message\_t\* bufPtr, error\_t error) {

if (&packet == bufPtr) {

locked = FALSE;

}

}

}

## **A 3.2 Base station program for Algorithm (Appc.nc)**

#include "RadioCountToLeds.h"

configuration RadioCountToLedsAppC {}

implementation {

components MainC, RadioCountToLedsC as App, LedsC;

components new AMSenderC(AM\_RADIO\_COUNT\_MSG);

components new AMReceiverC(AM\_RADIO\_COUNT\_MSG);

//components new TimerMilliC();

components ActiveMessageC;

components SerialPrintfC;

//components new TimerMilliC() as T1;

//components new TimerMilliC() as T3;

components CC2420ActiveMessageC;

App -> CC2420ActiveMessageC.CC2420Packet;

App.Boot -> MainC.Boot;

App.Receive -> AMReceiverC;

App.AMSend -> AMSenderC;

App.AMControl -> ActiveMessageC;

App.Leds -> LedsC;

// App.MilliTimer -> TimerMilliC;

App.Packet -> AMSenderC;

}

## **A 3.3 Base station program for Algorithm (c.nc)**

#include "Timer.h"

#include "RadioCountToLeds.h"

#include "string.h"

#include "stdio.h"

module RadioCountToLedsC @safe() {

uses {

interface Leds;

interface Boot;

interface Receive;

interface AMSend;

interface SplitControl as AMControl;

interface Packet;

interface CC2420Packet;

}

}

implementation {

message\_t packet;

bool locked;

uint16\_t counter = 0;

uint16\_t current\_temp;

uint16\_t rssi;

event void Boot.booted() {

call AMControl.start();

}

event void AMControl.startDone(error\_t err) {

if (err == SUCCESS) {

//call MilliTimer.startPeriodic(250);

}

else {

call AMControl.start();

}

}

event void AMControl.stopDone(error\_t err) {

// do nothing

}

event message\_t\* Receive.receive(message\_t\* bufPtr, void\* payload, uint8\_t len) {

if (len != sizeof(radio\_count\_msg\_t))

{

return bufPtr;

}

else

{

radio\_count\_msg\_t\* rcm = (radio\_count\_msg\_t\*)payload;

rssi = (int16\_t)call CC2420Packet.getRssi(bufPtr);

printf("nodeId id %u Rssi %u \n ",rcm->nodeId,rssi);

call Leds.led0Toggle();

call Leds.led1Toggle();

return bufPtr;

}

}

event void AMSend.sendDone(message\_t\* bufPtr, error\_t error) {

if (&packet == bufPtr) {

locked = FALSE;

}

}

}

## **A 3.4 Python program for getting the data from Base station**

# Python Script to get the da#ta from base station

moteUsbPort = '/dev/ttyUSB0' #change this according to the mote usb port

moteBaudRate = 115200 # chage this according to the mote baud rate

import serial

import time

fHandler = open("data.txt",'w')

ser = serial.Serial(moteUsbPort,moteBaudRate)

print "hi there"

while True :

try:

print "getting data"

reading = ser.read()

val = reading

readingHex = reading.encode('hex')

print(readingHex)

fHandler.write(val)

if(int(readingHex,16) == int('A',16)) :

print "got the ending"

#fHandler.write('\n')

except KeyboardInterrupt:

fHandler.close()

## **A 3.5 Python program for Algorithm to calculate shortest Distance after getting data.**

class InterpolatedArray(object):

"""An array-like object that provides

interpolated values between set ps."""

def \_\_init\_\_(self, ps):

self.ps = sorted(ps)

def \_\_getitem\_\_(self, x):

if x < self.ps[0][0] or x > self.ps[-1][0]:

raise ValueError

lower\_p, upper\_p = self.\_GetBoundingps(x)

return self.\_Interpolate(x, lower\_p, upper\_p)

def \_GetBoundingps(self, x):

"""Get the lower/upper ps that bound x."""

lower\_p = None

upper\_p = self.ps[0]

for p in self.ps[1:]:

lower\_p = upper\_p

upper\_p = p

if x <= upper\_p[0]:

break

return lower\_p, upper\_p

def \_Interpolate(self, x, lower\_p, upper\_p):

"""Interpolate a Y value for x given lower & upper

bounding ps."""

slope = (float(upper\_p[1] - lower\_p[1]) /

(upper\_p[0] - lower\_p[0]))

return lower\_p[1] + (slope \* (x - lower\_p[0]))

from dj import \*

from time import sleep

import re

import serial

a=0

d=0

z=0

b=z-a

c=z-d

#cmd = 'java RssiDemo -comm serial@/dev/ttyUSB0:telos'

#cmd = 'python ./output.py'

ser = serial.Serial('/dev/ttyUSB0',9600,timeout=5)

if \_\_name\_\_ == '\_\_main\_\_':

ps = ((30,0), (21, 15), (8,30),(6,45),

(4,60),(0,75),(-2,90),(-12,110),(-14,150),(-17,300),

(-29,450),(-37,800),(-42,1000))

table = InterpolatedArray(ps)

# weightMatrix = [

# [0,a,0,d],

# [a,0,b,0],

# [0,b,0,0],

# [d,0,c,0]

# ]

#print len(weightMatrix)

while True:

user\_input = raw\_input("please enter y to calculate the shortest path ")

if user\_input == 'y':

# fHandle = open('alData.txt','w')

# for i in range(0,1000):

# recvData = ser.read()

# fHandle.write(recvData)

# val = re.findall('\d+',recvData)

# fHandle.close()

fHandleR = open('data.txt','r')

for line in fHandleR :

#print line

#print '--'

try :

val = re.findall('\d+',line)

# if int(val[1]) > 100:

# val[1]=0

if int(val[1]) > 100:

val[1] = 0

if int(val[0]) == 1 :

a = table[int(val[1])]

#print "received rssi val for node 1"

elif int(val[0]) == 2:

z = table[int(val[1])]

#print "received rssi for node 2"

elif int(val[0]) == 3:

d = table[int(val[1])]

#print "received rssi for node 3"

except :

pass

b=z-a

if b < 0 :

b = -1\*b

c=z-d

if c < 0 :

c = -1\*c

weightMatrix = [

[0,a,0,d],

[a,0,b,0],

[0,b,0,0],

[d,0,c,0]

]

print weightMatrix

g = Graph(4)

g.graph = weightMatrix

g.dijkstra(0)

else :

print "please enter y to calulate the shortest path"

#print weightMatrix