WSN APPLICATION FOR MONITORING PATIENT'S BODY TEMPERATURE IN HOSPITALS

FINAL REPORT

TEST-BED PROJECT BY HABIBUL ISLAM & DENESH SAPKOTA

NOVEMBER 4, 2016
SENSOR NETWORKS
UNIVERSITY OF TURKU

Table of Contents

LIST OF ABBRIVIATION AND SYMBOLS

WSN	Wireless Sensor Network
RF	Radio Frequency
ECG	Electrocardiogram
IoT	Internet of Things
IPv6	Internet Protocol version 6
6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
CoAP	Constrained Application Protocol
HTTP	Hypertext Transfer Protocol
LLNs	Low-Power and Lossy Networks
RPL	Routing Protocol for Low-Power and Lossy Networks
IETF	Internet Engineering Task Force
ROLL	Routing Over Low power and Lossy
DAG	Directed Acyclic Graph
DODAG	Destination Oriented Directed Acyclic Graph
UDP	User Datagram Protocol

Project Objective

As the goal of this project, we implemented a wireless sensor network application to monitor patient's body temperature remotely. The main idea is to monitor patient's body temperature without the intervention of patient himself but connecting digital temperature sensor node on the body and automatically send the body temperature value (in °Celsius) to the concerned medical personnel, which can be accessible through internet at any time anywhere in the world.

1. Introduction

In traditional way, patient's body temperature is monitored by the presence of the doctor or other paramedical staffs. In some cases, continuous monitoring of the patient's body temperature is important and it is often laborious for a doctor or paramedical stuff in the hospital. In this WSN system, the digital temperature sensor continuously reads patients body temperature with regular interval and transmits the data to the receiver microcontroller, which then allow to display the data online. For data acquisition, we used two microcontrollers to act as a sender and a receiver in the sensor network system. The sender end connected to the patient's body and the receiver end works as a gateway to the webserver. The sender and receiver nodes communicate over the air by using RF (radio frequency) within their range. The receiver node is connected to the internet and it allows the doctors to see the situation continuously from their chamber.

The project was developed as a basic implementation of wireless network system. In this test-bed project we tried to implement the real life WSN approach in healthcare monitoring system. This project was developed using Contiki OS's RPL protocol for network connection. Both the sender and receiver program was written using C programming language. The complete project was done my modifying the example code files for temperature sensor, unicast sender and unicast receiver available in Contiki OS. For implementing the receiver node as a boarder router a python script was written to read the serial data and to feed that data to the plotly web server.

Throughout the project work we learned the process to use and configure sensors motes in using contiki operating system. Actually the whole course, labs and project was designed to help us to have better understanding about the IoT. The complete project was developed using microcontrollers, writing programs for motes, sensors and data acquisition. By using the similar approach other applications like patients' blood pressure monitoring system, heart rate monitoring system, blood glucose level monitoring system, ECG data monitoring system and motion detection system can be developed. In this report there are some discussion about the theoretical background needed to accomplish the project. In the latter part of the document there is the discussion about the programing approach of the project.

2. Wireless communications

The main idea of wireless communication is to transfer the information between two or more points without prior connection by an electrical conductor or wire. *Radio wave* is the most commonly used technology in wireless communication. The radio transmitter and receiver use radio waves in order to transfer information over the air. In WSN, a few to several hundreds of nodes is connected to each other with internal or external antenna. One of the main benefits of WSN is the scalability according to the deployment needs. Besides these, low power method for radio communication privileged the adaptation of wireless sensor network. The communication between WSN and other network is established through a gateway and it is tagged as bridge between two different networks which primarily serves to process the data to more resources, for instance, a remote web server. A typical wireless sensor network consists three major elements such as hardware, software and operating system.

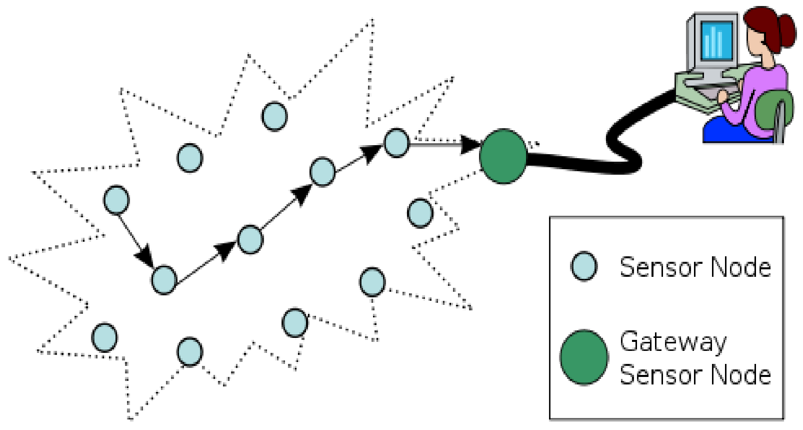


Figure 1: An architecture of typical multi-hop wireless sensor network

3. Application Overview

The system operation follows this following procedures:

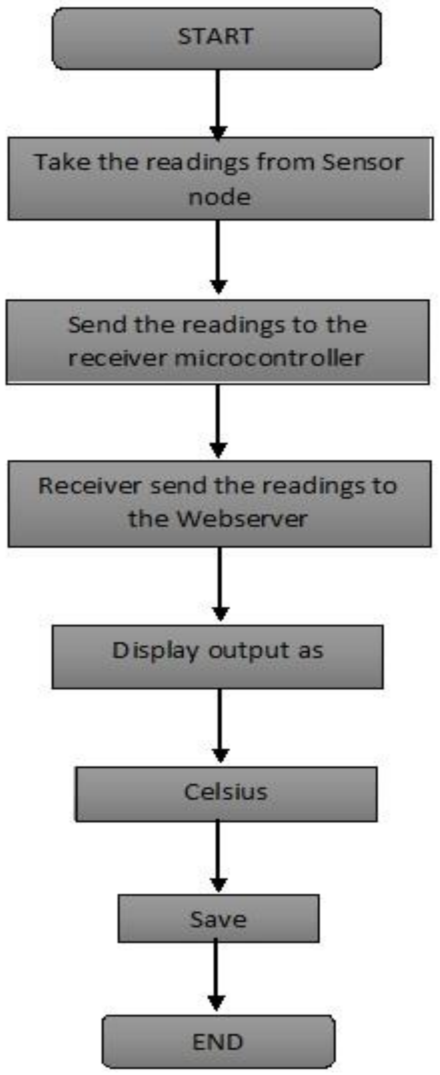

START
Take the readings from Sensor node
Send the readings to the receiver microcontroller
Receiver send the readings to the Webserver
Display output as
Celsius
Save
END

Figure 2: The program flowchart

1. Firstly, we place the digital sensor node to patient's body (i.e., Usually right or left hand)
2. The wireless Zolertia Z1 sender node is turn on where temperature sensor is attached.
3. Another Zolertia Z1 receiver node is placed within the range of the sender node.
4. The receiver node is connected to the computer with micro USB which has python script running on it. The data is then processed through the receiver node and pass it to the webserver.
5. The data is finally available for display on doctor's computer by using Plotly web application from anywhere and anytime

The displayed data is also automatically saved according to the time. As the data is available in the internet, it can be accessible by the authorized medical personnel.

4. System components

Hardware	Software
○ Two Zolertia Z1 motes ○ Tmp102 (Embedded Temperature sensor) ○ Micro USB cable ○ Computers ○ Power supply ○ Display	○ OS- Contiki 2.7 ○ Programming language- C and Python ○ Compiler (gcc-msp430) ○ Plotly Web server

Table 1. List of hardware and software components

In our project, we used only two Zolertia Z1 motes that fulfilled our initial purpose. The temperature sensor was embedded in the board so that we didn’t require any external temperature sensor. Besides, we used Contiki 2.7 operating system in order to program the microcontrollers according to our need. Both of the microcontroller were written using C programming language and gateway implemented by combining the receiver node and the computer through python script. In order to visualize the data we used a third party software application called Plotly that makes the system more user friendly.

Zolertia Z1

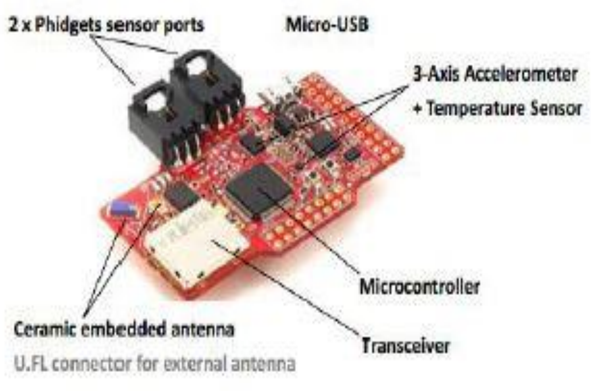


Figure 3: Low power wireless module Z1 (Z1 datasheet)

Zolertia Z1 is designed to develop low-power WSN projects and due to its sensor network module it has been widely used in WSN applications. For example, motion detection system, personal healthcare monitoring, emergency control systems, energy metering etc. It is 2.4GHz IEEE 802.15.4 & 6LowPAN compatible, which supports open source operating system like TinyOS or Contiki to connect the application directly to IoT with IPv6. One of the good features of Z1 is on-board digital sensors- accelerometer (ADXL345) and temperature sensor (TMP102). However, Z1 also allows adding other analogue and digital sensors with its Phidgets compatibility. The low power microcontroller MSP430F2617 features 8KB RAM and a 92KB Flash memory with a powerful 16-bit RISC CPU at 16MHz clock speed [1]. The required power supply is 3V.

TMP102

- Accuracy: 0.5°C (-25°C to +85°C)
- Two-wire serial interface
- 1.4V to 3.6VDC supply range

One-wire serial output temperature sensor TMP102 is available in the Z1 module. One factor we found which may lead to inaccurate reading is the thermal propagation. The temperature sensors reads values biased by up to +2°C when it is connected through the USB port. The fact it often stems from voltage regulator inside the CP2102 USB to RS232 converter [2].

I2C Bus

I2C protocol is used for the connection of low speed IC’s to processors and microcontrollers normally for short distance and intra-board communication. I2c uses bidirectional open-drain lines called serial data line (SDA) and serial clock line (SCL) with resistors. Recent I2C can run with the speed of 400 kbps. These speeds are quite important in the development of the embedded system. It consists two roles for nodes master nodes and slave nodes. Where master node generates the clock and also initiates communication with slaves. Similarly slave node receives the clock and respond to the master. I2C is able facilitate microcontroller to use two general purpose I/O pins and software to control networks of devices chips. In the implementation section of this project we can see that the library for I2C communication have been included while programming the Z1 mote. [12]

Contiki

Contiki, an open source operating system, especially developed to configure low-power microcontrollers to implement IoT [3]. This operating system is best suitable for long time running devices because of its efficient low-power wireless communication. Both IPv6 and IPv4 is supported by Contiki. Besides, other low-power wireless standards such as 6LoWPAN, RPL and CoAP also supported by Contiki. Contiki was best suitable for this project because Z1 libraries and other example files all are inclusive with the operating system. It is implemented in C language and has been ported to a number of microcontroller architectures. In Contiki, the routing modules are separated in directory such as "contiki/core/net/rpl" and allow to use those models for different tasks. For instance, rpl-icmp6.c can be used for packaging ICMP messages. Compared to Windows, Android and Linux, Contiki is still considered a new operating system. But one of the benefits we found is that its examples. It has plenty of examples which help the developers to use them as a guideline of WSN project. Mandatory AES128 is now supported by Contiki, which has solved earlier security issues.

Plotly

In order to visualize the data we used Plotly third party web application in our project. It allows to create informative graph using open source API and visualization library. In our approach, Plotly act as a web-server that processes HTTP requests from doctor's computer. The idea was to display the output in the graph which is supplied through the python program. According to our project requirement we have successfully posted data to the plotly server and graph was created in real time according to the supplied temperature and time values. The *Time Series Plot* with *temperature* and *datetime* objects allow to visualize the data in a graph where the changes can be easily seen.

5. Routing over Low-power and Lossy Networks

By making intelligent routing decision, routing protocol forward the packets from one to other nodes. Embedded devices are limited with the power, memory and processing resources. Low-power and lossy networks are basically the combinations of those embedded devices that are optimized for energy saving. However, the traffic flows in LLNs can be point-to-point, multipoint-to-point, point-to-multipoint. On the other hand, this kind of network can be interconnected up to thousands of nodes thus, it leads challenge to a routing solution. In order to provide proper routing mechanism, the IETF ROLL working group has defined specific requirements for LLNs [4].

RPL

RPL is routing protocol for IP smart object networks. This IPv6 based routing protocol specially designed for LLNs and it has been integrated to Contiki OS in order to connect the sensor devices. Since sensor nodes run on battery, one of the main purposes of using RPL over LLN application domain is to minimize energy and latency to overcome the constraints. In WSN, the communication activities such as transmitting and receiving actually take most of the energy. The purpose has been met by separating packet processing and forwarding from the routing optimization. Moreover, RPL is a distance- vector and proactive protocol and due to a nature of proactiveness, it starts finding the routes immediately after the network is initialized. RPL forms tree like topology called Directed Acyclic Graph (DAG) and Destination Oriented DAG (DODAG) in neighbour discovery mechanism. The figure below shows the difference between DAG and DODAG.

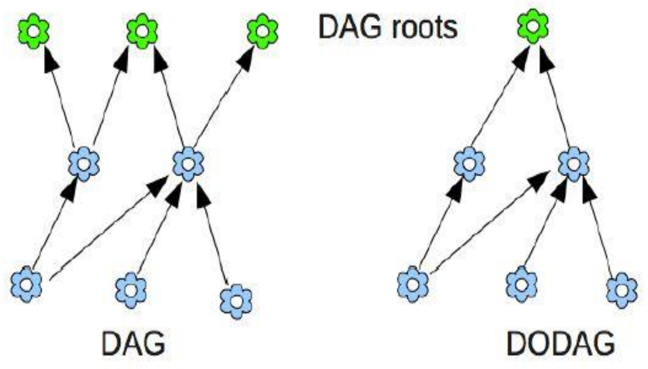


Figure 4: Topology of DAG and DODAG

In DODAG neighbor discovery method, a DAG is rooted at a single destination. In order to get access to the Internet all nodes usually want to connect with a single border router. On the other hand, each node in the network basically act as a gateway for that node because it has a preferred parent. However, unlike a traditional network, RPL uses more factors like routing metrics, objective functions and routing constraints while computing best paths for the network. In order to reduce the losses in the radio medium and then increase the mobility of the nodes in LLN, route aggregation is not recommended in RPL network [5].

UDP

User Datagram Protocol (UDP) is implemented on top of RPL. UDP mainly uses the concept of port to send messages to a specific recipient at one IP address. Prior communication between two ports does not require to set up transmission channel. In practice, UDP is reliable in short LAN link and there is no need for packet retransmission. As UDP is message oriented, it is a good choice for weak networks while dealing with small message and notifications. However, UDP is often a good choice even over weak networks for the applications such as real-time audio or video. For this project UDP protocol is perfect choice because we are sending a packet with complete data so, that the packet loss will not affect the normal functioning of the application.

Why 6LoWPAN?

IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) basically allows a mechanism to send and receive IPv6 packets over IEEE 802.15.4 based networks. The IEEE 802.15.4 standard aims at low-cost and low power communications. Since, IPv6 address alone consists 128 bits space to compress this address space 6LoWPAN was developed. Due to 6LoWPAN it is possible to use IPv6 address also in low power and low processing devices can communicate Internet-based services.

6. Project Implementation

After gathering all the theoretical knowledge explained in the previous section of this report, we started our project implementation by finalizing the project architecture. The main idea of the project was to develop the temperature monitoring system for the patients in hospital using wearable temperature sensor. Due to the hardware limitation we have used two Z1 node for sensor network system and its communication. Similarly, we used inbuilt Tmp102 sensor of Z1 to measure the temperature. Since, the idea of the project was to make a real time application which relays the temperature data to a cloud server. To fulfil this requirement we have used free plotly web service to connect the sensor network to the cloud server. In this implementation we have used Z1 nodes as data transmitter and receiver using RPL protocol for unicast communication between the nodes. Our normal computer was used as a gateway to read the data received by the node and sending it to the plotly web server. Detailed explanation of these implementation will be discussed in the further section of this document. In the following figure the network architecture of the project of the project can be seen;

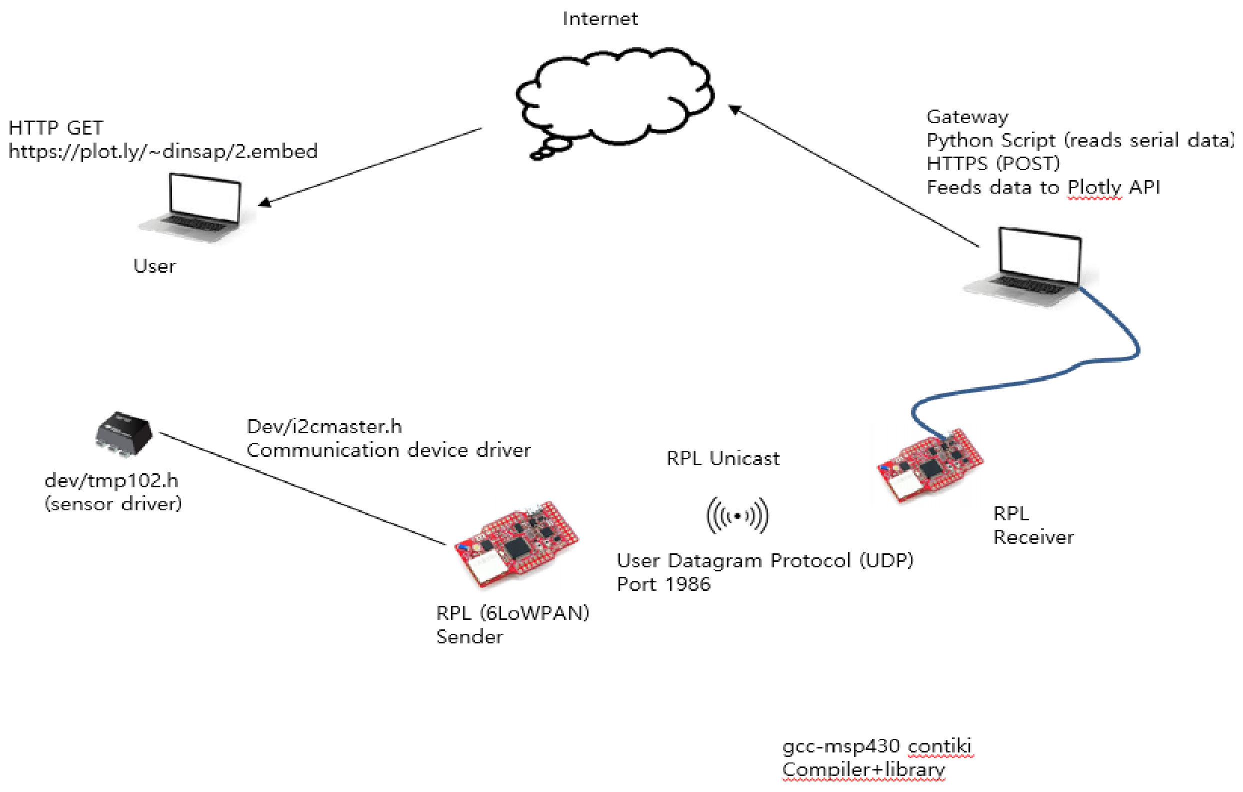


Figure 5: Overall network architecture of the project

In the beginning of the project we have downloaded and installed the gcc-msp430 compiler in contiki to be able to program in the Z1 mote. When the Contiki IDE was ready, we defined the project configuration file to use CSMA driver to be ContikiMAC with the channel check rate to be 8 Hz. ContikiMAC was used because it has lower energy consumption and 8 Hz channel check rate was used because the data was taken every 3 second this speed was fast enough for this project implementation. After this step we started to learn about the inbuilt temperature sensor of the Z1 microcontroller to be able to read the data from the sensor. According to project requirement we were supposed use two nodes for the connection and data transmission, due to that reason RPL unicast communication was used. As, we have learned during this course that in energy constraint devices it is always wise to design the system using less processing power to reduce energy consumption. For that reason, UDP transport protocol was used for data transmission. UDP protocol was used by trading-off the data loss with processing efficiency. In this particular application even after losing some data it will not have any effect because the data is send frequently and each packet consists a complete information of temperature. Then project implementation was continued developing unicast transmitter, unicast receiver and boarder router (gateway) with unlimited resources to transmit data to the web server. In the upcoming section of this document there will the detailed discussion of each of these implementations.

6.1. Unicast Transmitter Node

The implementation of the unicast transmission consists of two individual sections, temperature data acquisition from the sensor and transmitting that data using unicast communication to the receiver. To accomplish this task the sensor driver and Z1 device driver was used. Similarly for the unicast communication UDP was used on top of RPL protocol. In the first part of this document there is a discussion about the sensor data collection and data formatting.

Sensor Data

The built-in temperature sensor of Z1 microcontroller has been designed to measure the environmental temperature from -25°C to +85°C. Due to the limitation of the available devices same sensor was used to get the test data for body temperature. To be able to sense the temperature data and read it in the Z1 mote we need to import sensor driver (i.e. dev/tmp102.h) and microcontroller driver (i.e. dev/i2cmaster.h). This Tmp102 temperature sensor driver helps us to read the digital temperature data and I2C communication device driver for Z1 sensor node is used to transfer data inside the microcontroller. Since it was the test project due to that reason the temperature sensing interval has been set to be 3 sec. Which makes it easy to show that the system is working properly during demo. But for the real life situation the sensor should read data in 60 seconds interval. The raw data gathered from the sensor must be formatted to convert it to human readable °C values. In the following code example the process of getting the temperature data and its formatting is shown; [10]

```
tmp102_init();  /* to set up ports and pins for I2C communication */
sign = 1;
while(1) {
        raw = tmp102_read_temp_raw();  // Reading from the sensor
        absraw = raw;
        if (raw < 0) {        // Perform 2C's if sensor returned negative data
          absraw = (raw ^ 0xFFFF) + 1;
          sign = -1;
        }
       tempint  = (absraw >> 8) * sign;
        tempfrac = ((absraw>>4) % 16) * 625;        // Info in 1/10000 of degree
        minus = ((tempint == 0) & (sign == -1)) ? '-'  : ' ' ;
```

Figure 6: Methods used to read the temperature data

Unicast Sender Implementation

In this section a program was written for Z1 microcontroller that sends the temperature data obtained as explained in the previous section to the receiver node. But to achieve this functionality we have used IPv6 routing protocol for low power and lossy networks (RPL). According to the project requirement the unicast communication was the perfect match to design the efficient application. While writing the program for sender node all the necessary library were imported. But in this discussion most important among them will be discussed. To set run-time parameters like IP address uip.h header file was imported to use uIP configuration function. To support the program for the data structures of IPV6 uip-ds6.h header file was imported. UDP protocol was suitable for data transmission in this project and simple-udp module of Contiki OS was used by importing simple-udp.h header file. Similarly Contiki OS has server hack application which consists service registration and dissemination. These registered services are transmitted to all the neighbour nodes which are running servreg-hack application. At last UDP port number 1986 and service id 190 was defined to start writing the program for sender node [10].

Since the program was using IPv6 routing protocol the method was defined to set global IP address for the node. In the beginning of the implementation an IPv6 address was defined by providing the hexadecimal value. Then the data structure handling function was called for using neighbour discovery and auto configuration of the state machines. After that unicast address structure to the IP address was added. Finally local table was checked and the address was configured according to tentative or preferred address. In the following code snippet we can see the implementation of this method; [10]

```
static void
set_global_address(void)
{
  uip_ipaddr_t ipaddr;
  int i;
  uint8_t state;
  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
  printf("IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(uip_ds6_if.addr_list[i].isused &&
       (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
      uip_debug_ipaddr_print(&uip_ds6_if.addr_list[i].ipaddr);
      printf("\n");
    }
  }
```

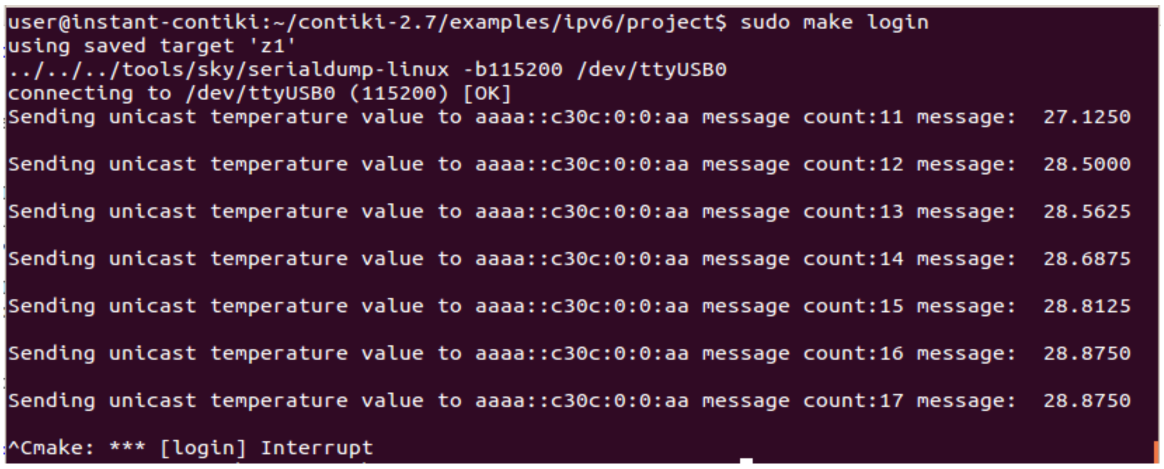
Figure 7: Setting IPv6 address for the sender node

After setting the IP address, port number and service id for the node the implementation proceed further to the process thread part of the program. Where two etimer events were decleared and called periodic timer and send timer to send temperature data when the send timer expires. Inside the thread the method servreg_hack_init() was called to transmits the registered services to all the nodes running the server-hack application. All the services running in the neighbour nodes are stored in the local table which can be accessed and checked according to the requirement. Then the method that we have discussed previously was called to assign the appropriate IP address to the node. After that our program is ready to establish the UDP connection to the receiver. To achieve that UDP connection was registered by providing the arguments for local port, remote address, remote port and call-back function. In the implementation remote IP address was set to null to accept packets from any IP address. When a packet is received from remote node we can all the contents of that packets can be accessed. Which will be discussed in the receiver node section. When the connection between two nodes is established the IP address of the receiver can be accessed by providing service ID to servreg_hack_lookup method. At this point of the implementation all the required information was gathered and now it is ready to send the data packet using the connection. When the data collected from the sensor is formatted then the data is stored in the buffer and checked if there is a remote node running server hack application. After that simple_udp_sendto() method was used providing all the values for the arguments to send the temperature data packet. In the following code snippet this implementation is shown; [10]

```
servreg_hack_init();
  set_global_address();
  simple_udp_register(&unicast_connection, UDP_PORT, NULL, UDP_PORT, receiver);
while(1) {
      PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&send_timer));
      addr = servreg_hack_lookup(SERVICE_ID);
            if(addr != NULL) {
                        static unsigned int message_number;
                        char buf[30];
                         printf("Sending unicast temperature value to ");
                        uip_debug_ipaddr_print(addr);
                        sprintf(buf, "%c%d.%d\n", minus, tempint, tempfrac);
                        printf(" message count:%d message: %s\n", message_number,buf);
                        message_number++;
                        simple_udp_sendto(&unicast_connection, buf, strlen(buf) + 1, addr);
            } else {
                        printf("Service %d not found\n", SERVICE_ID);
             }
  }
```

Figure 8: Unicast sender process thread code

After following the steps explained in the previous steps Z1 microcontroller was programmed as a sensor node which sense temperature data and sends the data to the receiver node using IPv6 UDP protocol. Complete temperature data is send in single packet due to that reason data loss during the transmission will not effects the application and processing power is saved using UDP transport protocol. The new line character was appended in the buffer to make it easier to read the packet contents. In the following figure the result of sender nodes programme execution is shown; [10]



```
user@instant-contiki:~/contiki-2.7/examples/ipv6/project$ sudo make login
using saved target 'z1'
../../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 (115200) [OK]
Sending unicast temperature value to aaaa::c30c:0:0:aa message count:11 message:  27.1250

Sending unicast temperature value to aaaa::c30c:0:0:aa message count:12 message:  28.5000

Sending unicast temperature value to aaaa::c30c:0:0:aa message count:13 message:  28.5625

Sending unicast temperature value to aaaa::c30c:0:0:aa message count:14 message:  28.6875

Sending unicast temperature value to aaaa::c30c:0:0:aa message count:15 message:  28.8125

Sending unicast temperature value to aaaa::c30c:0:0:aa message count:16 message:  28.8750

Sending unicast temperature value to aaaa::c30c:0:0:aa message count:17 message:  28.8750

^Cmake: *** [login] Interrupt
```

Figure 9: UDP message send to the receiver

6.2. Unicast Receiver Implementation

While programming the receiver all the library files which were imported for sender node program were imported. For receiver also same port and same service id was used, and process of setting the IP address was also same. After setting the IP address that address value was passed to create RPL Directed Acyclic Graph (DAG) function, where the IPv6 data related structure was checked. After that checking it if the received value is null or not. If the value is not null then that address is set as root address and set the address of server as the root of dag and set its prefix. In the following code the process of creating RPL DAG is shown; [10]

```
static void
create_rpl_dag(uip_ipaddr_t *ipaddr){
        struct uip_ds6_addr *root_if;
         root_if = uip_ds6_addr_lookup(ipaddr);
         if(root_if != NULL) {
                rpl_dag_t *dag;
                uip_ipaddr_t prefix;
                rpl_set_root(RPL_DEFAULT_INSTANCE, ipaddr);
                dag = rpl_get_any_dag();
                uip_ip6addr(&prefix, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
                rpl_set_prefix(dag, &prefix, 64);
                PRINTF("created a new RPL dag\n");
        } else {
        PRINTF("failed to create a new RPL DAG\n");
```

Figure 10: Creating RPL DAG for receiver

This program is supposed to receive the data packets send by the sender node to achieve it first of all the server hack initialized and register its application with service id and receiver IP address. When the process thread reaches the simple_udp_register method it call-back the receiver function to receive the packets send by the sender by passing all the arguments. Since only the temperature data was send in the packet so that only the temperature value was printed as output. In the following code receiver function implementation can be seen; [10]

```
static void

receiver(struct simple_udp_connection *c,  const uip_ipaddr_t *sender_addr,  uint16_t sender_port, const uip_ipaddr_t *receiver_addr,
uint16_t receiver_port, const uint8_t *data,  uint16_t datalen) {

        printf("%s",data);

}
```

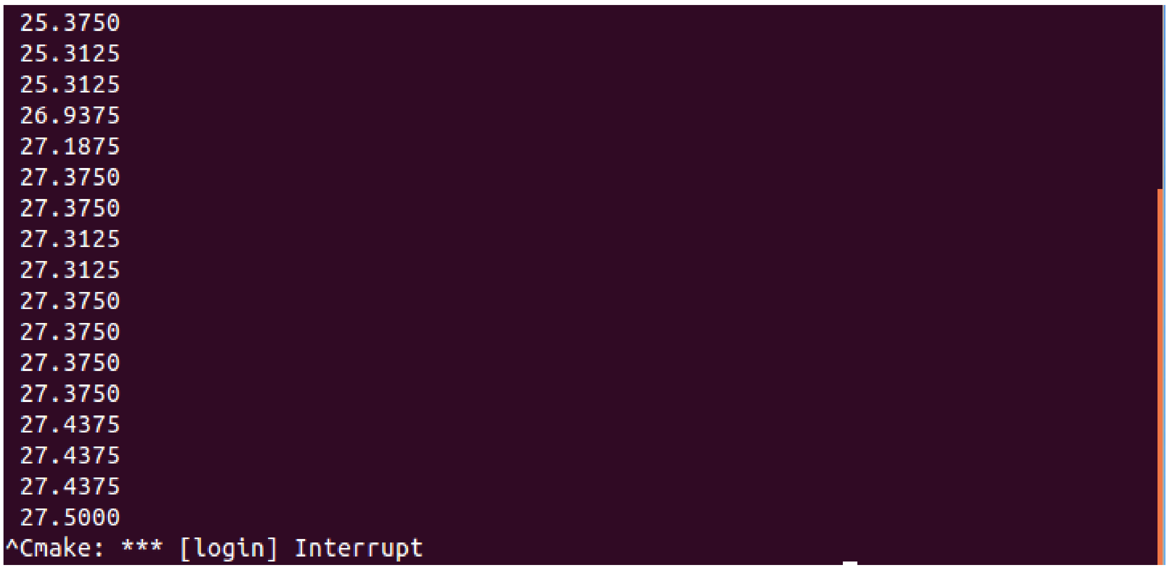
Figure 11: Receiver function in the program.

In the process thread of the receiver server hack, set the global IP address to the node and create RPL DAG was initialized as we have mentioned in the previous section. After that server hack application was registered with service ID and its IP address to have a connection to the clients with same service ID. Then a simple UDP connection was registered using unicast connection and other argument over RPL. In the following code snippet shows the implementation process; [10]

```
PROCESS_THREAD(unicast_receiver_process, ev, data) {

        uip_ipaddr_t *ipaddr;

         PROCESS_BEGIN();

        servreg_hack_init();

         ipaddr = set_global_address();

         create_rpl_dag(ipaddr);

        servreg_hack_register(SERVICE_ID, ipaddr);

        simple_udp_register(&unicast_connection, UDP_PORT, NULL, UDP_PORT, receiver);

        while(1) {

                 PROCESS_WAIT_EVENT();

         }

                PROCESS_END();

        }
```

Figure 12: Receiver process thread implementation

After completing all the above mentioned steps the program was uploaded in the Z1 receiver node. When the connection between the receiver and sender was successful using RPL unicast connection the receiver was able to receive the temperature data send by the sender. According to project requirement only the temperature value was needed due to that reason only the temperature data was printed in the following output figure it is shown as a result; [10]



```
 25.3750
 25.3125
 25.3125
 26.9375
 27.1875
 27.3750
 27.3750
 27.3125
 27.3125
 27.3750
 27.3750
 27.3750
 27.3750
 27.4375
 27.4375
 27.4375
 27.5000
^Cmake: *** [login] Interrupt
```

Figure 13: Temperature data received by the receiver

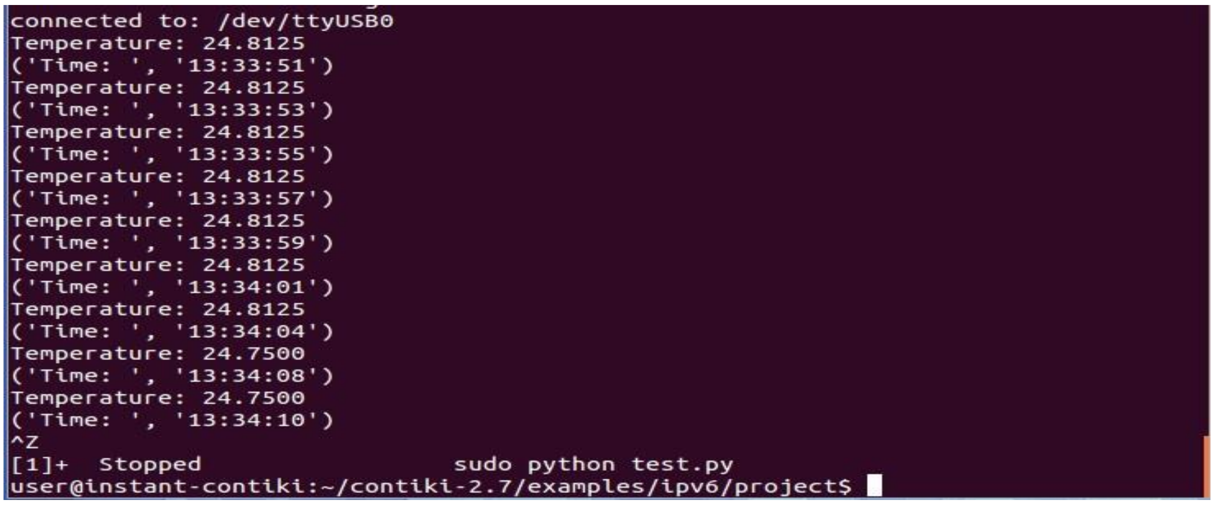
6.3. Data Acquisition from Serial

Until the previous section we were able to program two Z1 as a sender and receiver node. According to our project plan, we were trying to send the data collected by the receiver to web service to make the data remotely available in World Wide Web. Since, we all know that sensor network system are the resource constraint devices. Due to that reason we were trying to implement our project in such a way where the more resource intensive operation was done using our own computer as a boarder router to have a connection to the web service to feed temperature data. To accomplish this task python programming language was used to write a program to read the temperature data received by the receiver node. When the temperature data is received by the node current time stamp was captured which helps to use Plotly API to create a graph of the data obtained during this process in the real time. Receiver node was connected to the computer using the data cable and the python script was written to read the data in the serial. For this implementation serial library and time library of python was imported. After that serial port was defined by providing port address, baudrate to read data, parity, stop bits, byte size and time out value. Byte size was set to be 8 bits and baudrate (i.e. 115200) was defined exactly the same value while flashing the Z1 node. If the value is different than the program will not be able to read the value send by sender node. While writing the script if different value is assigned then the data printed in the serial will not be readable. In this program the seral data was read character by character inside forever loop. When the serial read method finds a new line character then the program knows that the packet data was complete and that value is stored in temperature string. At the same time the current time stamp is also sent to send to the Plotly server. The following script shows the implementation of serial data;

```
import serial
import time
ser = serial.Serial( port='/dev/ttyUSB0',\ baudrate=115200,\  parity=serial.PARITY_NONE,\
    stopbits=serial.STOPBITS_ONE,\ bytesize=serial.EIGHTBITS,\ timeout=0)
print("connected to: " + ser.portstr)
temp_str = ""
while True:
        for c in ser.read():
                if c == ' ':
                        temp_str = temp_str
                elif c == '\n':
                        print("Temperature: ",float(temp_str))
                        print("Time: ",time.strftime("%H:%M:%S"))
                        temp_str = ""
                else:
                        temp_str+=str(c)
ser.close()
```

Figure 14: Script to read the data from serial

The result of this particular part of the implementation can be seen in the following figure;



```
connected to: /dev/ttyUSB0
Temperature: 24.8125
('Time: ', '13:33:51')
Temperature: 24.8125
('Time: ', '13:33:53')
Temperature: 24.8125
('Time: ', '13:33:55')
Temperature: 24.8125
('Time: ', '13:33:57')
Temperature: 24.8125
('Time: ', '13:33:59')
Temperature: 24.8125
('Time: ', '13:34:01')
Temperature: 24.8125
('Time: ', '13:34:04')
Temperature: 24.7500
('Time: ', '13:34:08')
Temperature: 24.7500
('Time: ', '13:34:10')
^Z
[1]+  Stopped                 sudo python test.py
user@instant-contiki:~/contiki-2.7/examples/ipv6/project$
```

Figure 15: Temperature and time values from serial

6.4. Plotly API Implementation

First of all user account was created in plotly website to get the API key and streaming API tokens which are needed to stream the traces. Then plotly python library, numpy, plotly tools and plotly graph objects library was imported. Then the code to get stream id was written, after getting the id an instance of stream id object was made. Using this stream id object trace was initialized for streaming in plotly, where the array for X and y coordinates was defined. Then the title was added for the layout object and figure object was defined to draw the figure. Then after defining the streaming object with stream id the stream connection was opened. Data needed for the two coordinates of layout object are send to the web server using HTTP post. Current data gathered inside the forever loop is appended to the plot in the graph. These implementations can be seen in the following code snippet; [11]

```
import plotly
import certifi
import urllib3
http = urllib3.PoolManager(cert_reqs='CERT_REQUIRED',ca_certs=certifi.where())
plotly.tools.set_credentials_file(username='dinsap', api_key='0l2j9xmxa3')
import numpy as np
import plotly.plotly as py
import plotly.tools as tls
import plotly.graph_objs as go
#set stream_ids
f = []
f.append('ovm08xi3v8')
tls.set_credentials_file(stream_ids=f)
stream_ids = tls.get_credentials_file()['stream_ids']
# Get stream id from stream id list
stream_id = stream_ids[0]
# Make instance of stream id object
stream_1 = go.Stream(
    token=stream_id,  # link stream id to 'token' key
    maxpoints=80      # keep a max of 80 pts on screen
)
# Initialize trace of streaming plot by embedding the unique stream_id
trace1 = go.Scatter(
    x=[],
    y=[],
    mode='lines+markers',
    stream=stream_1         # (!) embed stream id, 1 per trace
)
data = go.Data([trace1])
# Add title to layout object
layout = go.Layout(title='Time series')
# Make a figure object
fig = go.Figure(data=data, layout=layout)
# We will provide the stream link object the same token that's associated with the trace we wish to stream to
s = py.Stream(stream_id)
# We then open a connection
s.open()
```

```
while True:
        for c in ser.read():
                if c == ' ':
                    ……
                elif c == '\n':
                    x = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')
                    y = temp_str
                    print("Temperature: "+ temp_str)
                    print("Time: ",time.strftime("%H:%M:%S"))
                    temp_str = ""
                    # Send data to your plot
                    s.write(dict(x=x, y=y))
                    #     Write numbers to stream to append current data on plot,
                    #     write lists to overwrite existing data on plot
                    time.sleep(2)  # plot a point every second
                else:
                    #line.append(c)
                    temp_str+=str(c)
# Close the stream when done plotting
s.close()
# Embed never-ending time series streaming plot
tls.embed('streaming-demos','12')
```

Figure 16: Python script to use Plotly API

After this implementation our sensor network system was connected to the web service and to draw a real time graph according to the temperature and time values. Networking section of the project was successfully implemented but the key application part where we can set the alarm if the temperature value is above the normal range can be done in the future development. Similarly it is a health care application where the information and network is sensitive so that it is important to implement the security protocols of contiki OS to secure the network and data. The following figure shows the results of our implementation;

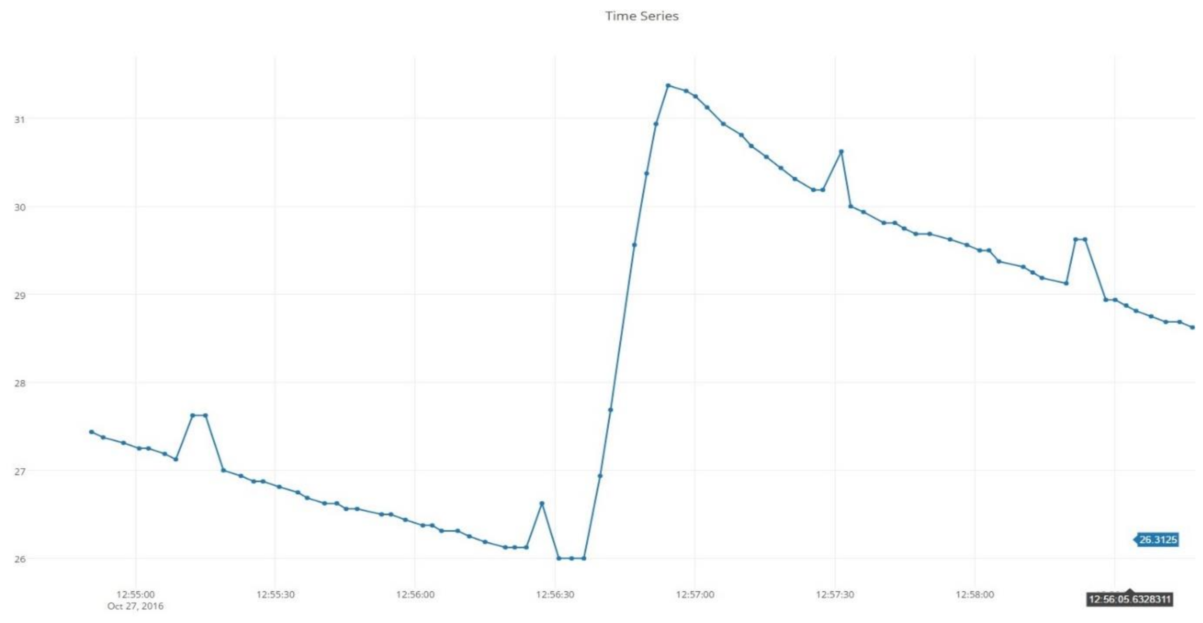


Figure 17: Real time graph obtained in the web server

7. Results and Performance Evaluation

The previous section gave the implementation method of our test-bed project. In the performance evaluation, we used third-party web application to display our test outcome. The figure 17 shows test output information Plotly web page. In the project, we used 3 seconds interval to send the data from the node. The data displayed on the web page is valid. We tested several times to see the accuracy of the data. Every time we heat up the temperature sensor, the graph shows an immediate change in real time. Besides, we also checked the compatibility of Z1 mote for body temperature functionality. For test purpose, we used a regular body temperature measuring thermometer to take our readings. To compare, we also tested with the Z1 mote by placing in elbow. We basically took five readings and every time Z1 mote shows on average of 5°Celsius less reading than the analog thermometer.

However, this was obvious to get an inaccurate reading while using Z1's embedded sensor for measuring body temperature. The test outcome would be accurate if we could use a wearable external sensor with Z1 by using Phidgets port. Nevertheless, the test outcome provides a clear evidence of accurate wireless sensor network communication between the nodes and the PC connected through the Internet. The prototype system has been successfully tested and displayed the result on the web page.

8. Future Development

This project was implemented to use only two Z1 nodes for the network communication and the receiver node was connected to the computer to act as gateway to connect to the cloud services. This project can be further developed using RPL multicast communication to sense multiple patient body temperature using wearable sensors. All the data received by the web server are not important, so that in web server we can only store the information which is important. If the reading is not normal and the patient needs immediate attention then alert message system can be implemented to send the alert message to the responsible physician. Since this implementation consists the sensitive health information of the patients due to that reason patients information and network must be secured. So, that in the further development security of the information and network must be implemented. As Contiki OS has library called ContikiSec that can be used to enforce security in Z1 nodes. Among the three modes of operation ContikiSec-AE is better because it provides confidentiality, integrity and authentication.

9. Conclusion

Wireless sensor network system is significantly useful in healthcare services. In this project work, we tried to find out the way how this kind of system is developed. It was a good learning experience where we were able to learn the basics of sensor network system. We were able to gain some programming experience using Contiki OS. Since, both of us were not from the embedded computing background; due to that reason it was difficult at the beginning of the project but when we understand the concepts and hardware started working it was fun. The most interesting part of this project was to write a program for a real hardware and testing the result by running the actual hardware. It was interesting for us because we never had an opportunity to do similar project with real devices. One of most difficult part of this implementation was to be successful to have network connection between the nodes. This project also provided some opportunity to learn little bit about python programming language which was also a good learning experience. Another difficult part of the implementation was in boarder router implementation where serial data was gathered and those data was send to the cloud service. This project helped us to have a better understanding about the implementation of internet of things (IoT). In conclusion, we can say that the project's outcome was partially achieved. In future, the project can be extended by adding more sensor nodes and mobile application. It can be developed for the doctors to get alerts. Since, body temperature reading may not be that important information but other health information such as blood pressure, heart bit rate, glucose level monitoring system can be developed using same approach. This system might be useful for the health care center to increase the productivity of the employers. Along with all of these things a basic sensor network system was developed with cloud services. Both of the Z1 microcontroller were configured and program was written to develop the complete application.

References

[1] "Z1 - Zolertia", Zolertia.sourceforge.net, 2016. [Online]. Available: http://zolertia.sourceforge.net/wiki/index.php/Z1. [Accessed: 04- Nov- 2016]

[2] 2]2016. [Online]. Available: http://Zolertia, W. S. N. platform, Z1 Datasheet. [Accessed: 04- Nov- 2016].

[3] Dunkels, A., Gronvall, B., & Voigt, T. (2004, November). Contiki-a lightweight and flexible operating system for tiny networked sensors. In Local Computer Networks, 2004. 29th Annual IEEE International Conference on (pp. 455-462). IEEE.

[4]"RFC 6550 - RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", Tools.ietf.org, 2016. [Online]. Available: https://tools.ietf.org/html/rfc6550. [Accessed: 04- Nov- 2016].

[5] Ali, H. (2012). A performance evaluation of rpl in contiki.

[6] Postel, J. (1980). User datagram protocol (No. RFC 768)

[7] Shelby, Z., & Bormann, C. (2011). 6LoWPAN: The wireless embedded Internet (Vol. 43). John Wiley & Sons

[8] Abdullah, A., Ismael, A., Rashid, A., Abou-ElNour, A., & Tarique, M. (2015). Real Time Wireless Health Monitoring Application using Mobile Devices. *International Journal of Computer Networks & Communications (IJCNC)*, *7*(3).

[9] Shelar, M., Singh, J., & Tiwari, M. (2013). Wireless Patient Health Monitoring System. *International Journal of Computer Applications*, *62*(6).

[10] Modules. (n.d.). Retrieved November 04, 2016, from http://contiki.sourceforge.net/docs/2.6/modules.html

[11] Plotly. (2015). Getting started with streaming. Retrieved November 4, 2016, from https://plot.ly/python/streaming-tutorial/

[12] Info, I. 2 C. (2016). I2C Info – I2C bus, interface and protocol. Retrieved November 4, 2016, from http://i2c.info/