

Cover

Federal Agency and Organization Element to Which Report is Submitted:

4900

Federal Grant or Other Identifying Number Assigned by Agency:

1816197

Project Title:

NeTS: Small: RUI: Bulldog Mote- Low Power Sensor Node and design Methodologies
for Wireless Sensor Networks

PD/PI Name:

- Nan Wang, Principal Investigator
- Woonki Na, Co-Principal Investigator

Recipient Organization:

California State University-Fresno Foundation

Project/Grant Period:

10/01/2018 - 09/30/2021

Reporting Period:

10/01/2019 - 09/30/2020

Prepared by: Calvin Jarrod Smith

National Science Foundation Bulldog Mote Project

Progress as of September 28, 2020

Calvin Jarrod Smith

Department of Electrical and Computer Engineering
California State University, Fresno
NSF Bulldog Mote Team

Contents

1	Purpose	3
2	Current Projects	3
2.1	TelosB Prototype Gateway Design	3
2.1.1	Overview	3
2.1.2	TelosB Network	5
2.1.3	STM32 Master Controller	6
2.1.4	ESP32 Wi-Fi Sub-System	11
2.1.5	Raspberry Pi and TelosB Receiver	13
2.1.6	Results	14
2.2	PCB Design of CC2630 Embedded System	15
2.2.1	Chip Selection	16
2.2.2	Basic Components and Circuit Design	16
2.2.3	Antenna Circuit	17
2.2.4	Decoupling Capacitors	19
2.2.5	PCB Design Software	19
2.2.6	Results	20
3	Next Steps of Project	21
4	Project Code Files	22
4.1	TelosB Client Code	22
4.2	STM32 Main Controller Code	24
4.3	ESP32 Sub-System Code	35
4.4	TelosB Server Code	38

List of Figures

1	TelosB Gateway Prototype Diagram.	4
2	TelosB Gateway Prototype.	5
3	TelosB Wireless Sensor Mote.	6
4	STM32 L432KC Development Board.	7
5	STM32CubeMX Software User Interface.	8
6	STM32CubeMX Project Settings.	9
7	Keil μ Vision IDE.	9
8	STM32 Controller LCD Indicator for Network Status.	11
9	ESP32 Node-MCU Device used for Wi-Fi Communication in Gateway Prototype.	11
10	The TelosB Server and Raspberry Pi Sub-System.	13
11	List of Feeds from Devices in Network.	14
12	Graphical Feed of Mote with Address CD6ED.	14
13	PCB Prototype of the CC2630 Embedded System.	15
14	Circuit Design of PCB Prototype System.	17
15	MIFA Antenna, IFA Antenna, Folded-Dipole Antenna and YAGI Antenna PCB Designs.	18
16	PCB IFA, Balun and LC Filter connected to MCU.	19
17	Front and Back of PCB Prototype Design.	20
18	TelosB and CC2630 PCB Prototype System Size Comparison.	20

1 Purpose

The National Science Foundation (NSF) Bulldog Mote Team is part of California State University, Fresno Department of Electrical and Computer Engineering. The purpose of this group is to discover, research and develop wireless communication protocols and hardware used in Mobile Ad-Hoc Networks (MANETs) and Wireless Sensor Networks (WSNs). This team's purpose is to research current wireless technologies and algorithms and create new hardware implementations to support current and future protocols for both MANETs and WSNs.

To accomplish this goal, the team has researched new a developing wireless schemes, hardware to support the processing of data within these wireless networks and various routing algorithms used to direct data through MANETs and sensor networks.

2 Current Projects

2.1 TelosB Prototype Gateway Design

The TelosB wireless sensor nodes are equipped with a Texas Instruments CC2420 wireless transceiver. This ZigBee and 6LoWPAN-ready chip is compliant with the IEEE 802.15.4 standard for low-rate, personal wireless area networks. This allows the TelosB motes to be able to form a wireless network mesh topology in which collected sensor data can be sent through collection of motes to some predetermined destination for data collection. However, these devices are not equipped for internet connectivity. Their lack of Ethernet and Wi-Fi capabilities keep them isolated from other internet-connected devices. To remedy this, a WSN gateway has been developed to bridge the two types of networks, allowing the collected data to be published to the internet for some further purpose such as data forecasting, visualization, monitoring or surveillance, among other applications.

2.1.1 Overview

The TelosB Prototype Gateway system consists of several sub-systems that allow the handling of both 6LoWPAN and Wi-Fi communication.

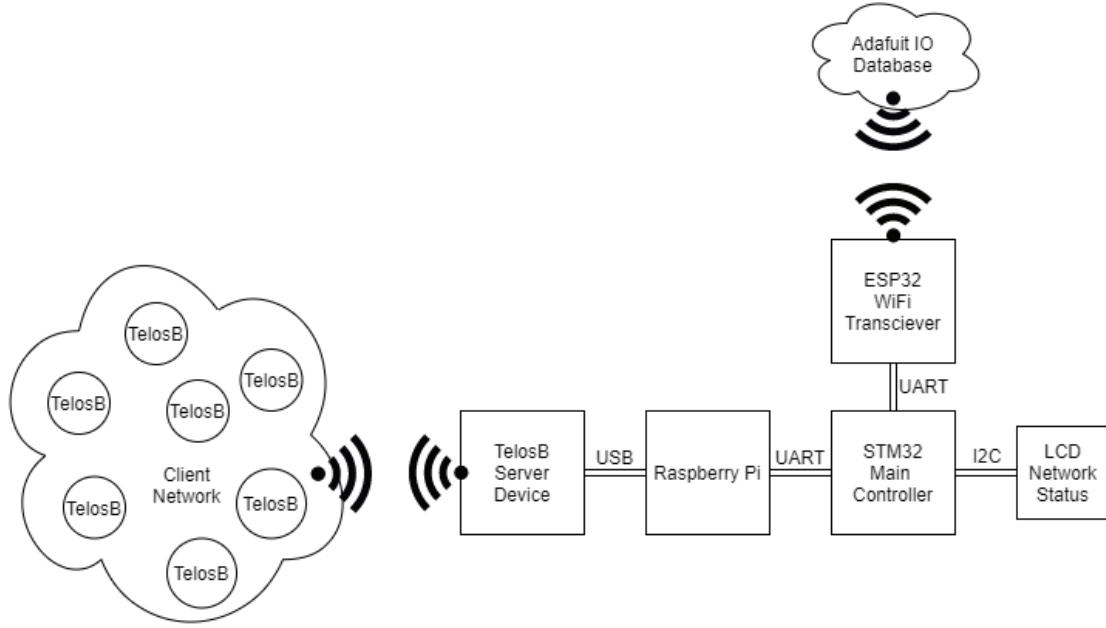


Figure 1: TelosB Gateway Prototype Diagram.

This design consists of an STM32 L432KC device used as a master controller, an ESP32 Node-MCU used as a Wi-Fi transceiver, an LCD screen for network status indication, and a TelosB/Raspberry Pi sub-system used for 6LoWPAN interconnection for the TelosB network. The working prototype is shown in Figure 2.

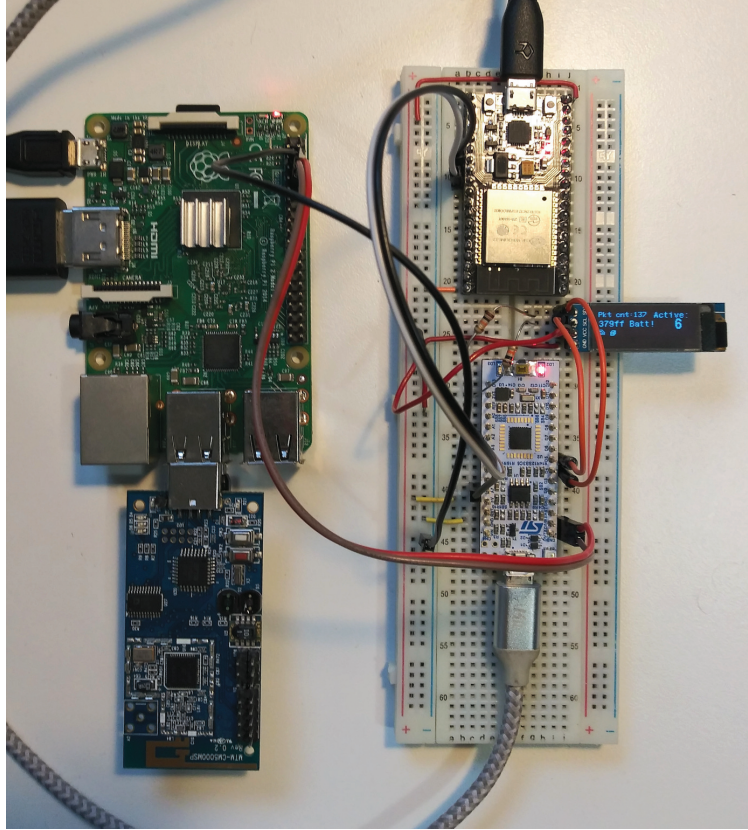


Figure 2: TelosB Gateway Prototype.

2.1.2 TelosB Network

The TelosB mote is a wireless sensor device equipped with temperature, light, infrared and humidity sensors. This device is compatible with TinyOS and ContikiOS operating systems (OS) and is capable of using both ZigBee and 6LoWPAN wireless transmission protocols.

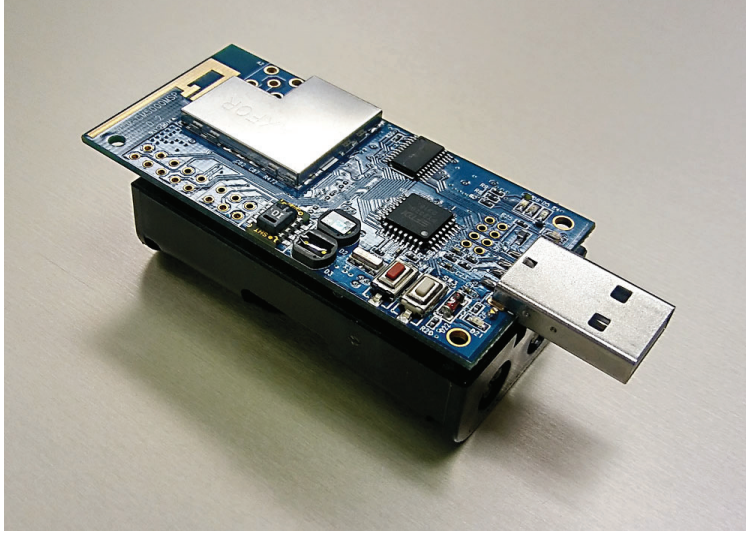


Figure 3: TelosB Wireless Sensor Mote.

Six of these motes were used to perform temperature data collection. This collection of motes was placed in a distributed fashion around the laboratory to purposely form a mesh topology where multiple hops would be necessary for distance motes to transmit sensed data to the gateway device. Each device was programmed using the Contiki NG OS as a UDP client that performed a temperature reading, then used RPL routing to send the temperature data to the UDP server, which was programmed to be the TelosB device connected to the gateway.

The program running on each mote is shown in Section 4.1. Each mote is configured to read its temperature sensor and battery level every 20 ± 2 seconds. This slight *jitter* is added to the timer to avoid consistent transmission collisions in case two or more nodes happen to transmit periodically at the same time. Also, to make transmissions easier to identify, the green on-board LED was momentarily flashed every time a UDP packet was created and sent from an originating node.

2.1.3 STM32 Master Controller

The device chosen for the main system controller was an STM32 Low-power Cortex-M4 microcontroller. The L432KC model was decided upon for its small form factor, low cost and functionality. The hardware can be programmed to support up to 2 UART and 1 I2C communication ports at the same time, so this chip was an ideal choice.

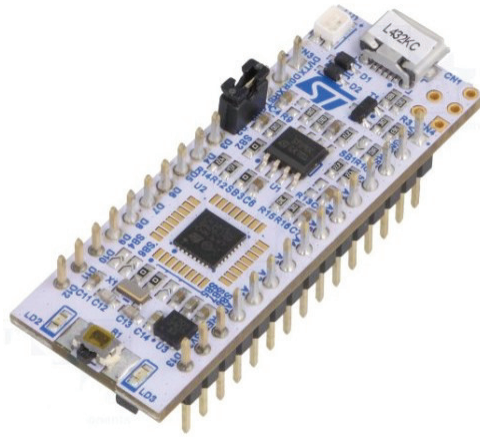


Figure 4: STM32 L432KC Development Board.

In addition, the controller needed a real-time operating system to handle multiple tasks and interrupt events, so the chip needed to have enough memory in terms of program size (ROM or Flash) and stack and data memory (RAM). The STM32 L432KC contains 256 KB of Flash memory for programs and 64 KB of RAM for program memory, which is sufficient for running a small real-time operating system like FreeRTOS.

FreeRTOS was chosen as the OS used in the main controller system because of extensive community support, small program size, and the ability to handle interrupts while running non-preempting (or cooperative) multiple tasks. The latter allows tasks to run to completion without the risk of interrupt each other, but allows external events to be handled, like UART packets, through hardware interrupts.



The first step to begin programming the STM32 development board with FreeRTOS, was to setup the hardware devices necessary to communicate with all sub-systems. For this, 2 UART ports were needed for communication with the TelosB network through the Raspberry Pi/TelosB Server mote and another for the ESP32 to connect to the internet and upload temperature readings to a database. In addition, an I2C port was needed for the LCD screen for network status indication. To set up the hardware and create all of the necessary peripherals and middleware, STM32CubeMX software from STMicroelectronics was used.

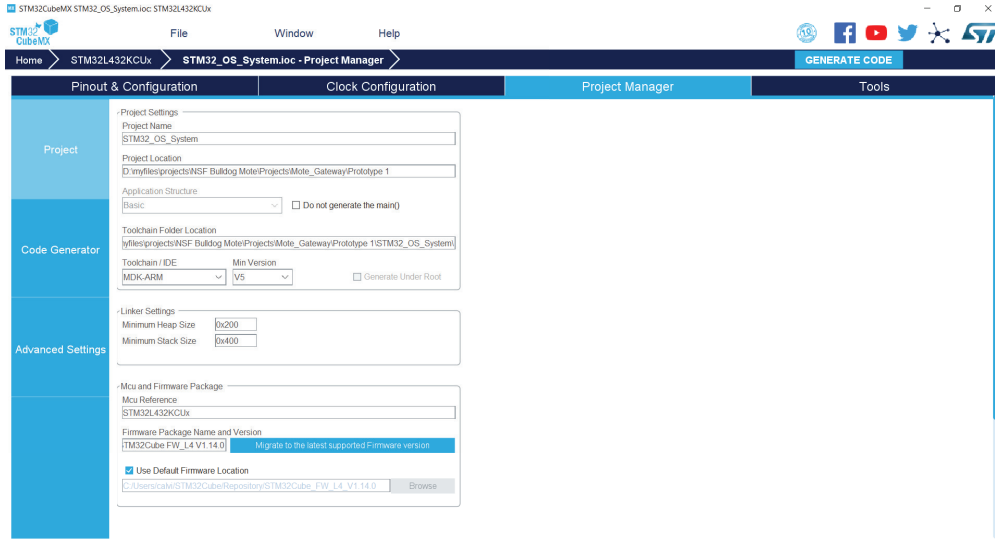


Figure 6: STM32CubeMX Project Settings.

The project was then ready to begin software development. To open working project code, the directory *MDK-ARM* inside the project folder was accessed and the *.uvprojx* file was selected to open the generated μ Vision project.

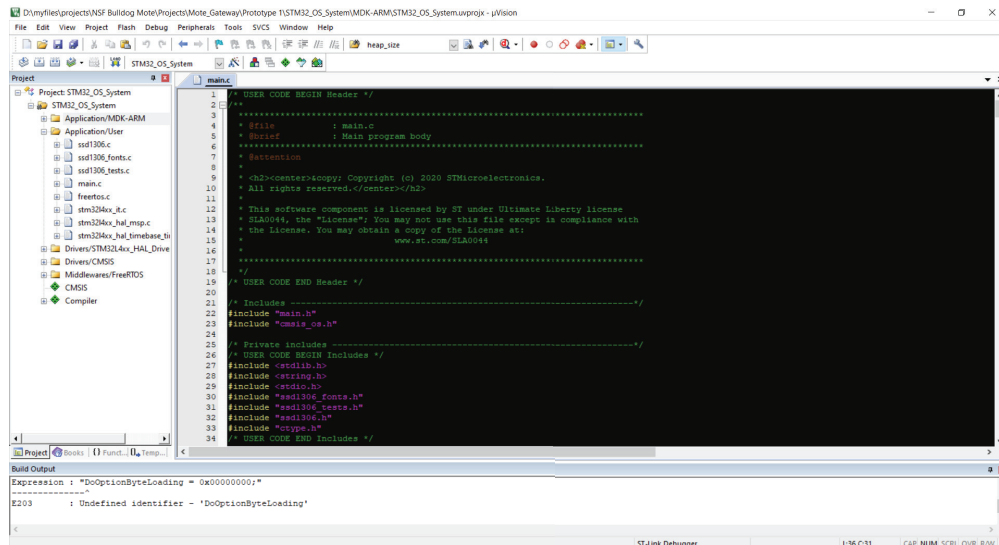


Figure 7: Keil μ Vision IDE.

The generated code contains one default task, so other tasks have to be created. As well, the default heap size needs to be increased for dynamically allocated data within each task. This was done by opening the file *startup_stm32l432xx.s* and scrolling to line 43. There the *Heap_Size* declaration was changed to *0x10000*, which changes the heap size to 65 KB. This size gave enough space for all of the tasks needed for the gateway system.

Next, the main program file was opened and code was written for the main controller of the gateway system. The complete code is shown in Section 4.2.

To handle UART transmissions and a network status display, new structures for packet buffers, mote addresses, and active network motes were created. As well, transmit and receive UART buffers were instantiated as 8-bit character buffers to hold the UART packets for both the TelosB mote receiver and the ESP32 systems. The LCD screen drivers were also installed in included with the project files.

Two new tasks were created in addition to the default task: `UpdateMotes()` and `LCDTask()`, plus handlers for each task. Within the `main()` function, the packet buffer, and mote list were initialized, as well as the hardware peripherals. The ESP32 is initialized by sending WiFi and server connection requests to the ESP32 device over UART2, which is retransmitted until the ESP32 acknowledges those requests. Also, the LCD screen is initialized with configuration data to properly display the network status. The main function also initializes the task handlers and starts the OS kernel. After that, the default task begins execution.

Within the default task, the system sets up the UART1 connected to the Raspberry Pi/TelosB server to trigger an interrupt for the next incoming UART transmission. The task then stays in an infinite loop waiting for incoming packets, which is determined by checking if the size of the packet buffer is greater than 0. If a packet is present, a critical section is entered to ensure mutual exclusion of the packet buffer structure, as the UART1 interrupt handler also modifies the structure. A packet is popped off and its address, temperature reading and battery reading are extracted. If the address is not present in the list of currently active motes, it stores the new address and sets its active value to `MOTE_ACTIVE`, which is a constant that is decayed over time to determine which nodes may have died or become disconnected from the network. If the battery value is lower than the threshold set by `MOTE_BATTERY_THRES`, this mote is marked as having a low battery and the variable `MOTE_BATT_LOW` is set to one. The address of the device is also saved so that the `LCDTask()` can indicate this mote as having a low battery. Once the LCD variables are updated, the data can then be sent to the ESP32 device to be uploaded to the Adafruit IO database. Various commands are used to tell the ESP32 system what operations to perform and are shown in Table 1. The STM32 device sends a data request command byte of 5, then sends the posting data of 11 bytes containing the mote address and temperature reading, and finally sends a data request completion byte of 0xF. The controller continuously resends this sequence until it receives a data request acknowledgement byte from the ESP32 system of 7. Once it completes the packet handling, the task is manually yielded to allow other tasks to run.

Within the `UpdateMotes()` task, the number of active motes is modified for the LCD indicator based on the current `active` value held in the active mote list. This function works to decay every active mote value by 1 every 10 seconds. Once the value reaches 0, the mote is marked as inactive and the mote count displayed on the LCD will decrease. In addition the `LCDTask()` is run periodically to update the network status on the LCD screen. The display is shown in Figure ?? with the network status values.



Figure 8: STM32 Controller LCD Indicator for Network Status.

The last custom written function in the main program file is the UART receive callback: `HAL_UART_RxCpltCallback()`. This function is triggered when a UART module completes a reception of a certain number of bytes of data into a receive buffer. In this case, it is triggered when 17 bytes of the receive buffer are filled, which is the size of the expected packet from the TelosB server mote. This function takes the received packet in the `Rx_buff1` and copies it into a space in the packet buffer as long as there is room. It then increments the size, head pointer and current packet count used for the LCD. Finally, the function triggers another UART1 reception via interrupt so that the next packet can be collected and the callback triggered again.

The remainder of the code in `main.cc` was the generated code from the STM32CubeMX software that sets up the hardware peripherals and includes the correct library components to link the entire project during compilation.

2.1.4 ESP32 Wi-Fi Sub-System

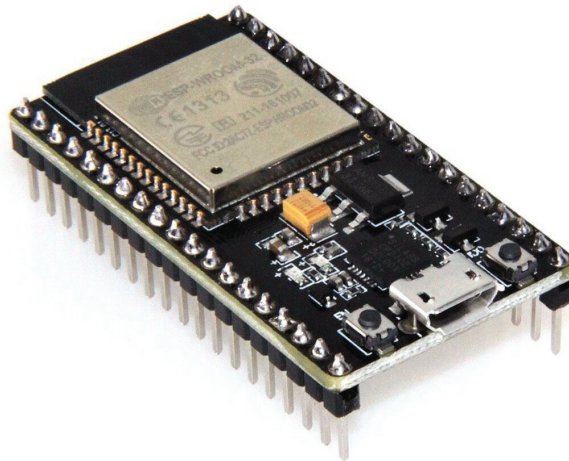


Figure 9: ESP32 Node-MCU Device used for Wi-Fi Communication in Gateway Prototype.

The ESP32 device was used to take UART commands from the STM32 controller and publish received data to an Adafruit IO database for displaying the data graphically. To do this, the

Code (Hex)	UART Command
0x1	Request connection to Wi-Fi
0x2	Acknowledge Wi-Fi connection
0x3	Request connection to Adafruit IO Server
0x4	Acknowledge Server connection
0x5	Begin Data Transmission
0x6	End Data Transmission
0x7	Data Transmission Acknowledgement
0xF	Transmit Error

Table 1: ESP32 and STM32 UART Command and Acknowledgment Codes

UART2 port of the STM32 device was connected to the UART2 of the ESP32.

The list of commands used between the STM32 and ESP32 devices are shown in Table 1

The ESP32 development board was programmed using the Arduino IDE because of the simple Wi-Fi and Adafruit IO libraries that are already implemented for the system. Since this part of the project deals mostly with hardware, the software on the systems matters less.

Within the program running on the ESP32, a new class was developed to handle all of the feeds and motes listed to Adafruit IO. First, each encountered mote was given its own feed so that data from each mote could be isolated in the database. Then functions for inserting and checking mote addresses into the `AIO_Feed_Class` and functions to send data to the mote’s particular feed in the database were written along with the default class constructor and destructor.

In the Arduino `setup()` function, the serial ports were opened for the serial console display and the serial UART connection to the STM32 using 115200 as the baud rate. Then a connection was established to the Adafruit IO database using `io.connect()`. This establishes both the Wi-Fi connection and database connection. Then, in the `loop()` function, UART commands are read into the ESP32 system. Each command is broken in to determine what the requested operation is. The command code is saved into the `oper` variable which is then compared to available operations. A command of 1 tells the ESP32 system to check Wi-Fi connectivity and confirm by transmitting back a command code of 2. Command code 3 requests a connection to the Adafruit IO server, which the ESP32 will reply with a 4 code for acknowledgement. When the system receives a 5, it will wait for a brief period to allow the requested data to be transmitted into the UART hardware buffer completely. This data will then be copied into the software variable `inData`, which the lower 5-bytes of the IPv6 address and the temperature data are extracted separately and inserted into the feed class using the `sendData()` method. This method then checks if the address is current part of the list of known nodes, add it to the list and creates its own feed if its not and then publishes the requested data to that feed. Once this data transmission is complete, the ESP32 then transmits a 7 back to the STM32 controller to indicate the data publication was successful. The full working code for the ESP32 system is shown in Section 4.3.

2.1.5 Raspberry Pi and TelosB Receiver

This part of the gateway system was used to collect temperature and battery data from the motes in the TelosB mesh network. The TelosB device acting as the UDP server was connected to the Raspberry Pi through USB. This configuration was necessary as there is only two serial communication ports already designated for the CC2420 Radio Transceiver and UART via USB port. As there were no other ports available to route to the TelosB's GPIO pins, the USB had to be used. The Raspberry Pi was selected as the USB-to-UART translator because it has the hardware USB input ports, an OS with all the necessary firmware and drivers for the USB ports and accessible UART GPIO pins to connect to the STM32 device.

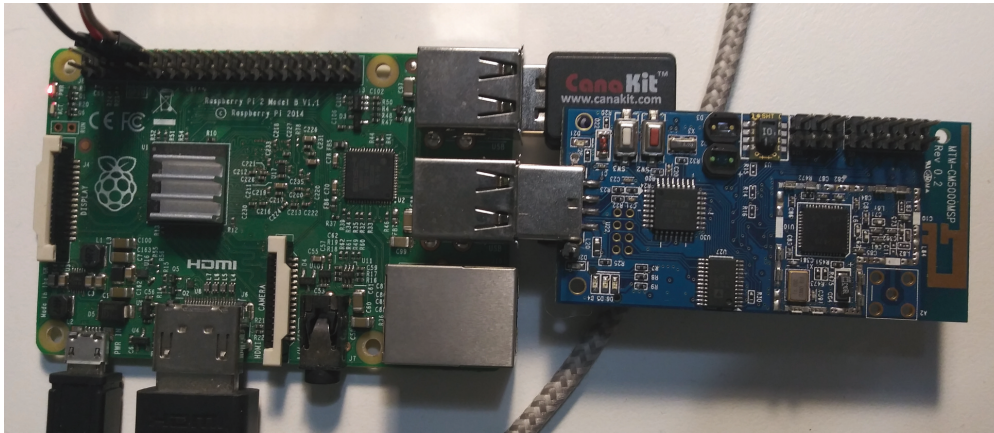


Figure 10: The TelosB Server and Raspberry Pi Sub-System.

The TelosB device was programmed using Contiki NG as a UDP server to collect the data being transmitted in the TelosB network. In the receive callback function of the server, the device prints the packet data and originating address of the client device to the USB. This acts as a UART communication over the USB to the Raspberry Pi. The code for the server program is shown in Section 4.4.

On the Raspberry Pi, the kernel control of the serial communication pins had to be disabled so that a user program could use the UART pins. This was done using the `raspi-config` command in Raspbian OS. once disabled the system was rebooted and a bash script was written to take incoming USB serial data and send over the UART GPIO pins located at `/dev/ttyAMA0`:

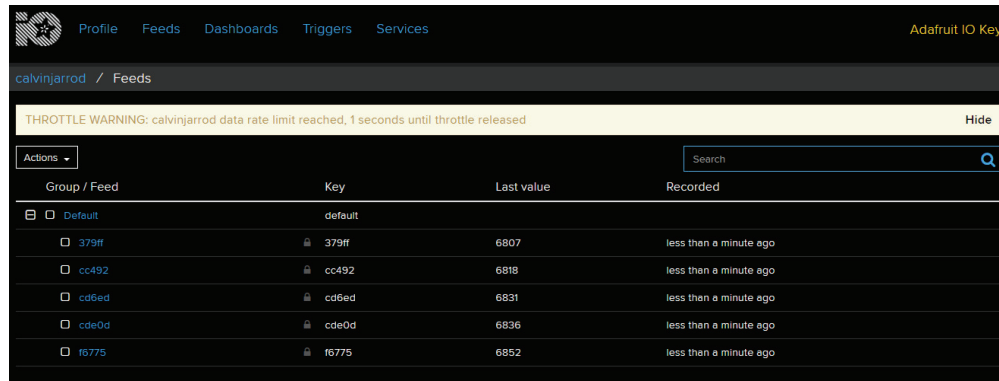
```
#!/usr/bin/env bash

stty 115200 sane -echo < /dev/ttyUSB0
while (true); do cat /dev/ttyUSB0; done > /dev/ttyAMA0
```

This script sets the baud rate for the transmissions to 115200 and repeatedly checks the serial input of the USB at `/dev/ttyUSB0` for any data. Every received byte of data from the USB is sent over the

2.1.6 Results

When all the sub-systems are wired together correctly and executed, the TelosB network data is successfully uploaded to the Adafruit IO database and graphical data is shown for the six different motes in the network.



Group / Feed	Key	Last value	Recorded
Default	default		
<input type="checkbox"/> 379ff	379ff	6807	less than a minute ago
<input type="checkbox"/> cc492	cc492	6818	less than a minute ago
<input type="checkbox"/> cd6ed	cd6ed	6831	less than a minute ago
<input type="checkbox"/> cde0d	cde0d	6836	less than a minute ago
<input type="checkbox"/> f6775	f6775	6852	less than a minute ago

Figure 11: List of Feeds from Devices in Network.

Figure 11 shows the six active motes that are running in the network. Each mote collects and transmits temperature and battery readings every 20 seconds, so this network transmits 18 unique pieces of data every minute.



Figure 12: Graphical Feed of Mote with Address CD6ED.

Mote CD6ED shows a graphical representation of data in Figure 12.

2.2 PCB Design of CC2630 Embedded System

Once the gateway prototype for the TelosB network was complete, the natural next step was to begin the design of a PCB system in which to migrate the development of a new gateway onto. This new PCB design will help with the development of both a new TelosB gateway device and a new wireless sensor mote.

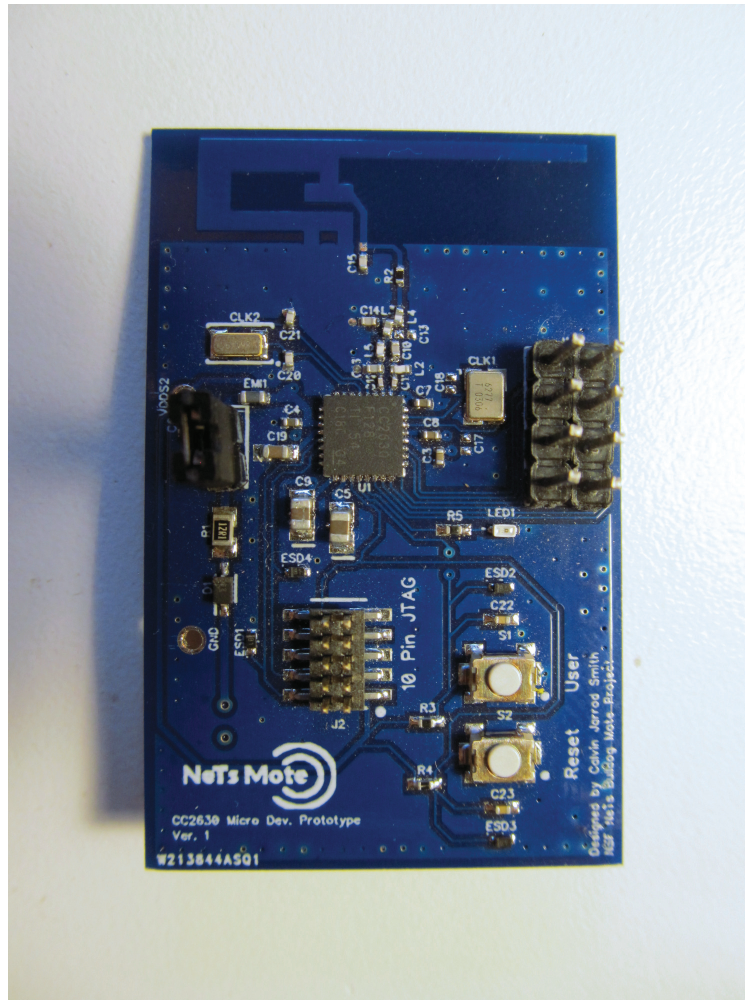


Figure 13: PCB Prototype of the CC2630 Embedded System.

This new development board contains few components, including a Texas Instruments CC2630 Wireless MCU, JTAG header, GPIO, push buttons and a green LED. When designing this new PCB prototype, there were several objectives:

1. The design needed to be as small as possible.
2. The components should consume as little power as possible.
3. The design needed to be as low-cost as possible.
4. The design needed to be very simple, so that testing and debugging could be done more easily.

5. The design needed to be battery operated so it could act like a wireless sensor node.
6. The system needed to have wireless capabilities and be able to communicate with a 6LoWPAN, IEEE 802.15.4 standard wireless network.

To be able to meet these objectives several design components needed to be considered: chip selection, basic components and circuit design, antenna circuit and decoupling capacitors.

2.2.1 Chip Selection

The Texas Instruments CC2630 chip was chosen out of a several other considerations. This chip is an all-in-one wireless microcontroller with ZigBee and 6LoWPAN capabilities. Having a single chip on-board saves space when compared to the TelosB mote where the MSP430 microcontroller and CC2420 Radio transceiver are separated and require many passive components [1].

Other all-in-one wireless microcontroller units (MCUs) were also considered like the Texas Instruments CC2538 and CC2650. These two and the CC2630 chip are all supported by Contiki OS, are 6LoWPAN-ready, are very low-power. Out of these three the CC2630 is the lowest cost while also consuming the least power [2] [3] [4]. The RHB IC package of the CC2630 was also chosen due to its incredibly compact size (5 mm × 5 mm) while still having 32 pins.

2.2.2 Basic Components and Circuit Design

Texas Instruments has published several *Application Reports* that help embedded designers create effective systems using their ICs. The report *CC13xx/CC26xx Hardware Configuration and PCB Design Considerations* [5] was used to aid in the design of the PCB design in this project. In addition, several reference design were used to determine correct component placement and system circuit design such as the TI SensorTag [6], the TI Humidity and Temperature Sensor Node [7], and the CC2538EM system [8].

The reference design stated above were used to create the system circuit design shown in Figure 15.

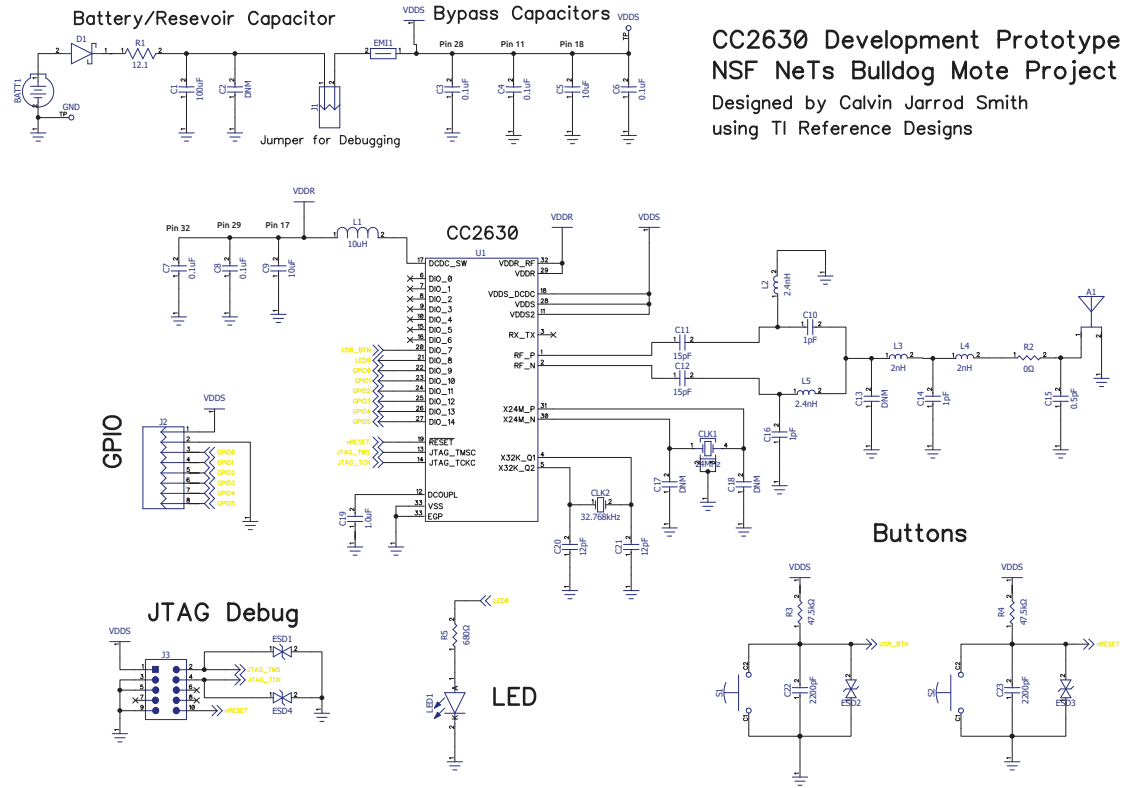


Figure 14: Circuit Design of PCB Prototype System.

Several basic components in this PCB design are necessary for correct functionality of the system. First, two crystal clocks are needed. A 24 MHz crystal is required for the radio transceiver as a frequency reference, and the 32.768 kHz crystal is used to improve sleep clock accuracy as opposed to using the internal RC oscillator. Next, decoupling capacitors are used to decouple several voltage inputs from the battery via VDD5 and VDDR. To program the chip, a 10-pin JTAG port was used to connect to the chip's JTAG_TMSC and JTAG_TCKC lines. These input also included electro-static discharging diodes to help reduce any noise on the input lines. To allow other serialized devices to interact with the development board, GPIO pins were included as well. Also, for indication and user interaction, a green LED and two push buttons were included in the design.

2.2.3 Antenna Circuit

The antenna circuit is a critical component of a wireless design, not just for its application, but also for its complexity level of difficulty in designing a custom one. There are four typical PCB antenna designs for 2.4 GHz applications: a Meandering Inverted-F Antenna (MIFA), an Inverted-F Antenna (IFA), a Folded Dipole Antenna, and a YAGI Antenna.

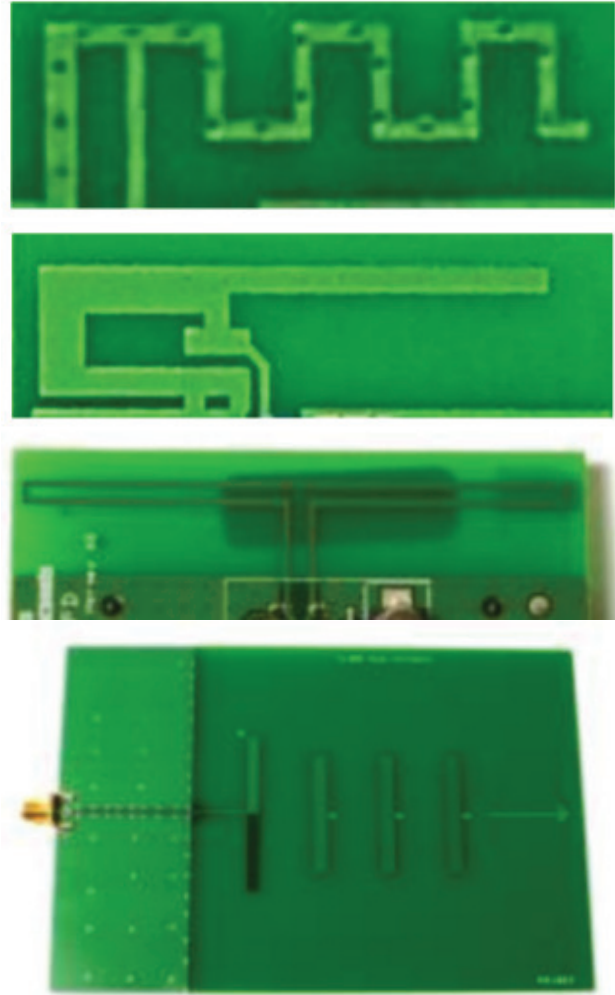


Figure 15: MIFA Antenna, IFA Antenna, Folded-Dipole Antenna and YAGI Antenna PCB Designs.

Out of all of these designs the IFA and MIFA designs are the smallest and work best for a compact PCB design. Of these two however, the IFA design has a more omni-directional radiation pattern than the MIFA, so for this project the IFA was the antenna of choice.

To balance the RF signal being received on the antenna, a *Balun* is used as part of the antenna circuit. The layout of a Balun needs to be as symmetrical as possible. This circuit plus the LC filter connected between the microcontroller and the Balun to attenuate signal harmonics and to function as an impedance transformation for $50\ \Omega$. This circuit is also tuned using specialized, expensive equipment and for this reason most manufacturers recommend copying the antenna circuit from one of their reference design. In the case of this project, the Texas Instruments SensorTag antenna circuit was used as it featured an IFA and the SensorTag device has a CC2650 MCU, which has an identical pin layout the CC2630 in this project. The PCB antenna for this design is shown in Figure 16.

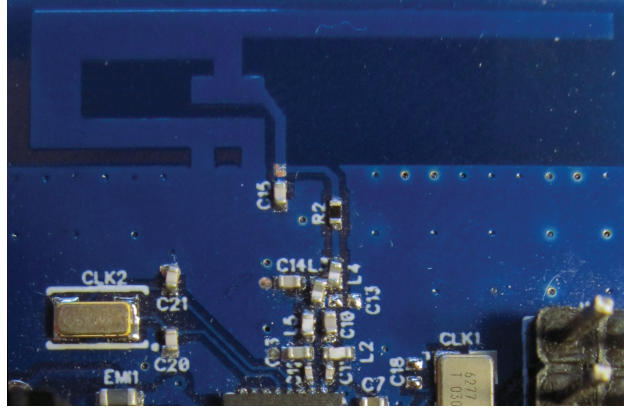


Figure 16: PCB IFA, Balun and LC Filter connected to MCU.

2.2.4 Decoupling Capacitors

The decoupling capacitors are necessary in the design to act as a energy reservoir in case the system draws a large amount of current from the battery causing a voltage levels to be pulled down. There are general rules that should be followed when placing decoupling capacitors. First, they should be placed on the same side of the PCB as the active component, in this case the CC2630 chip. Each capacitor should also have its own via to ground to reduce noise coupling. Current return paths should be short, generally using a large ground pad on the back of the PCB. Decoupling capacitors should also be as close as possible to the pin they are supposed to decouple. These considerations will help with reducing jittery power lines and voltage drop during times of high current demand.

In this design, decoupling capacitors were placed very close to the IC, each with their own via to ground on the back of the PCB, which acted as a large ground pad for the entire board.

2.2.5 PCB Design Software

DipTrace was the design software used to create the PCB layout. The models for each components were either obtained from the manufacturer website or created based on the footprints given in the device's datasheet. The completed PCB designs are shown in Figures ?? and ??

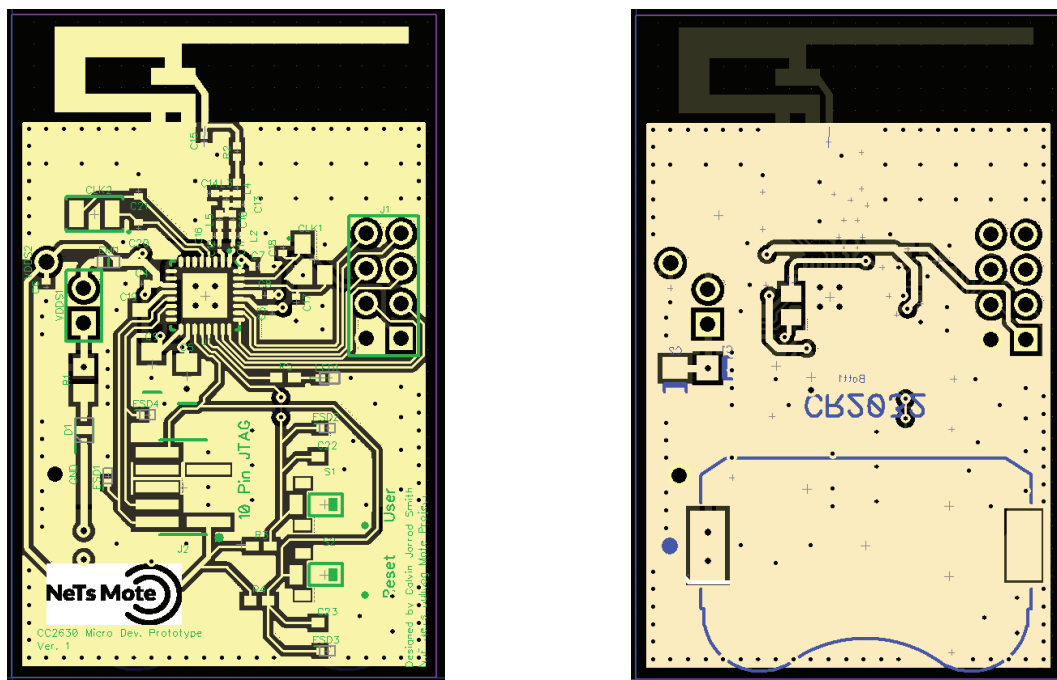


Figure 17: Front and Back of PCB Prototype Design.

2.2.6 Results

The resulting PCB system was successfully design, fabricated and components soldered in place. The complete PCB implementation is shown in Figure 13. A size comparison between the TelosB and the new CC2630 PCB prototype board are shown in Figure 18.

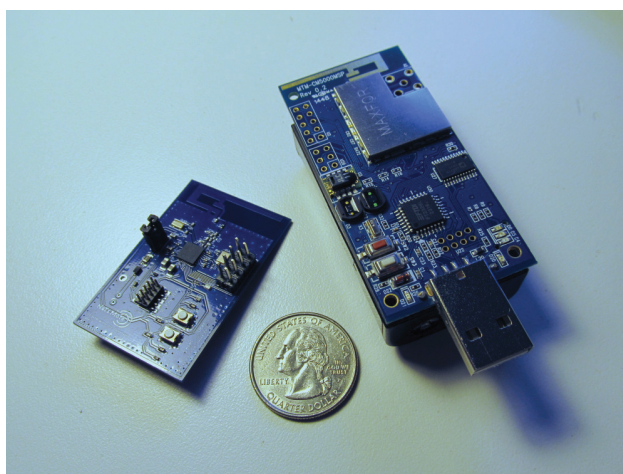


Figure 18: TelosB and CC2630 PCB Prototype System Size Comparison.

3 Next Steps of Project

The next steps in for the NeTs Bulldog Mote project are threefold. After through testing of the PCB prototype design, a new wireless sensor mote will be design using the PCB design in this report as a starting point. As well, a PCB design for a new gateway device will be developed using the designs in both projects in this report. Finally, software routing algorithms to make traditional routing more energy-efficient are currently being developed.

4 Project Code Files

4.1 TelosB Client Code

```
#include "contiki.h"
#include "net/routing/routing.h"
#include "random.h"
#include "net/netstack.h"
#include "net/ipv6/simple-udp.h"
#include "dev/sht11/sht11-sensor.h"
#include "dev/battery-sensor.h"
#include <msp430.h>

#include "sys/log.h"
#define LOG_MODULE "App"
#define LOG_LEVEL LOG_LEVEL_INFO

#define WITH_SERVER_REPLY 1
#define UDP_CLIENT_PORT 8765
#define UDP_SERVER_PORT 5678

#define SEND_INTERVAL (10 * CLOCK_SECOND)

static struct simple_udp_connection udp_conn;
/*-----*/
PROCESS(udp_client_process, "UDP client");
AUTOSTART_PROCESSES(&udp_client_process);
/*-----*/

static void
udp_rx_callback(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)
{
    LOG_INFO("Received response '%.*s' from ", datalen, (char *) data);
    LOG_INFO_6ADDR(sender_addr);
    #if LLSEC802154_CONF_ENABLED
        LOG_INFO_(" LLSEC LV:%d", uipbuf_get_attr(UIPBUF_ATTR_LLSEC_LEVEL));
    #endif
    LOG_INFO_("\n");
}
/*-----*/
PROCESS_THREAD(udp_client_process, ev, data)
{
    static struct etimer periodic_timer;
    static unsigned count;
    static char str[32];
    uip_ipaddr_t dest_ipaddr;

    static int temp;
    static int volt;

    // P5DIR |= (0x16 | 0x32 | 0x64); // All three LEDs
    P5DIR |= 0x32;
    PROCESS_BEGIN();

    /* Initialize UDP connection */
    simple_udp_register(&udp_conn, UDP_CLIENT_PORT, NULL,
                      UDP_SERVER_PORT, udp_rx_callback);
```

```

etimer_set(&periodic_timer , random_rand() % SEND_INTERVAL);
while(1) {
    SENSORS_ACTIVATE(sht11_sensor);
    SENSORS_ACTIVATE(battery_sensor);

    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));

    //P5OUE
    P5OUT &= ~(1<<5); // bit-clear LED3 to sink current

    //leds_on(LED3);
    temp = sht11_sensor.value(SHT11_SENSOR_TEMP);
    volt = battery_sensor.value(0);
    if(NETSTACK_ROUTING.node_is_reachable() && NETSTACK_ROUTING.get_root_ipaddr(&dest_ipaddr))
        /* Send to DAG root */
        printf("Sending request %u to \n",count);
        LOG_INFO_6ADDR(&dest_ipaddr);
        LOG_INFO("\n");

        snprintf(str , sizeof(str) , "%d %d" , temp,volt);
        simple_udp_sendto(&udp_conn , str , strlen(str) , &dest_ipaddr);
        count++;
    } else {
        LOG_INFO("Not reachable yet\n");
    }

    /* Add some jitter */
    etimer_set(&periodic_timer , SEND_INTERVAL
        - CLOCK_SECOND + (random_rand() % (2 * CLOCK_SECOND)));

    //P5OUT |= (0X16 | 0x32 | 0x64);
    P5OUT |= (1<<5);
    SENSORS_DEACTIVATE(sht11_sensor);
    SENSORS_DEACTIVATE(battery_sensor);
}

PROCESS_END();
}
/*-----*/

```

4.2 STM32 Main Controller Code

```
/* USER CODE BEGIN Header */
/**
 * *****
 * @file           : main.c
 * @brief          : Main program body
 * *****
 * @attention
 *
 * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
 * All rights reserved.</center></h2>
 *
 * This software component is licensed by ST under Ultimate Liberty license
 * SLA0044, the "License"; You may not use this file except in compliance with
 * the License. You may obtain a copy of the License at:
 *
 *                                     www.st.com/SLA0044
 *
 * *****
 */
/* USER CODE END Header */

/* Includes -----*/
#include "main.h"
#include "cmsis_os.h"

/* Private includes -----*/
/* USER CODE BEGIN Includes */
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "ssd1306_fonts.h"
#include "ssd1306_tests.h"
#include "ssd1306.h"
#include "ctype.h"
/* USER CODE END Includes */

/* Private typedef -----*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----*/
/* USER CODE BEGIN PD */
#define MOTE_PKT_SIZE 17
#define MAX_MOTES 64
#define PKT_BUFF_MAX 8
#define LCD_ROW_1 1
#define LCD_ROW_2 17
#define LCD_ROW_3 33
#define LCD_ROW_4 49
#define MOTE_BATTERY_THRES 3000
#define MOTE_ACTIVE 3
/* battery threshold determined by ADC value
   Minimum battery voltage to operate is 2.1V
   Measured Voltage = (ADCval/4096)*Vref*2
   Vref of 2.4 used as worst case, so the ADCval
   that would represent 2.1V is 1792, round up to get
   1800 which gives some time before mote will die
 */
/* USER CODE END PD */

/* Private macro -----*/
/* USER CODE BEGIN PM */
struct MOTE_PKT {
    char pkt[MOTE_PKT_SIZE];
};
```

```

/* USER CODE END PM */

/* Private variables _____*/
I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;

osThreadId defaultTaskHandle;
/* USER CODE BEGIN PV */

osThreadId UpdateMotesHandle;
osThreadId LCDTaskHandle;

// buffer of received packets
struct PKT_BUFF {
    unsigned int head;
    unsigned int tail;
    unsigned int size;
    struct MOTE_PKT buff[PKT_BUFF_MAX];
}PKT_BUFF;

struct MOTE_ADDR {
    char addr[5];
} MOTE_ADDR;

struct NETWORK_MOTES {
    struct MOTE_ADDR moteAddr[MAX_MOTES];
    int active[MAX_MOTES];
    int size;
} NETWORK_MOTES;

// UART buffers
// modified by UART1 and UART2 RX callbacks
// and PacketHandler for the Tx_buff
char Rx_buff1[MOTE_PKT_SIZE];
uint8_t Rx_buff2[8];
char Tx_buff[11];

// packet buffer
// modified by UART1 RX callback and PacketHandler
struct PKT_BUFF pktBuff;

// current motes in network
// modified by PacketHandler
int ACTIVE_MOTES = 0;
struct NETWORK_MOTES networkMotes;

// current number of received packets
// modified by UART1 RX Callback
int CURRENT_PKT_COUNT = 0;

// current network conditions from ESP32
// modified by UART2 RX Callback
int CONNECTED_WIFI = 0;
int CONNECTED_SERVER = 0;

// low battery status
// modified by PacketHandler
int MOTE_BATTERY_LOW = 0;
char MOTE_BATTERY_LOW_ADDR[5];

// LCDTask Globals _____
char display[20];
// _____

/* USER CODE END PV */

```

```

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);

// STM32 Gateway Tasks
void StartDefaultTask(void const * argument);
void UpdateMotes(void const * argument);
void LCDTask(void const * argument);

int stringCmp(char* a, char* b, size_t size);
int int2StringCpy(char** dest, int src, size_t size);
int stringCpy(char** dest, char* src, size_t size);
void ESP32_Init(void);

/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{
    /* USER CODE BEGIN 1 */
    pktBuff.head = 0;
    pktBuff.tail = 0;
    pktBuff.size = 0;

    // initialize all places in network motes to 0
    for (int i = 0; i < MAXMOTES; i++) {
        strncpy(networkMotes.moteAddr[i].addr, "00000", 5);
        networkMotes.active[i] = 0;
    }
    /* USER CODE END 1 */

    /* MCU Configuration -----*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */
    //HAL_UART_Receive_IT(&huart2, (uint8_t *) Rx_buff2, 6);
    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART1_UART_Init();
    MX_USART2_UART_Init();
    MX_I2C1_Init();
    /* USER CODE BEGIN 2 */

```



```

    ssd1306_Init();
    ESP32_Init();
    //HAL_UART_Receive_IT(&huart1,(uint8_t *) Rx_buff1,MOTE_PKT_SIZE);
/* USER CODE END 2 */

/* USER CODE BEGIN RTOS_MUTEX */
/* add mutexes, ... */
/* USER CODE END RTOS_MUTEX */

/* USER CODE BEGIN RTOS_SEMAPHORES */
/* add semaphores, ... */
/* USER CODE END RTOS_SEMAPHORES */

/* USER CODE BEGIN RTOS_TIMERS */
/* start timers, add new ones, ... */
/* USER CODE END RTOS_TIMERS */

/* USER CODE BEGIN RTOS_QUEUES */
/* add queues, ... */
/* USER CODE END RTOS_QUEUES */

/* Create the thread(s) */
/* definition and creation of defaultTask */
osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 1, 128);
defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
    osThreadDef(LCDTask1, LCDTask, osPriorityNormal, 1, 512);
LCDTaskHandle = osThreadCreate(osThread(LCDTask1), NULL);
    osThreadDef(UpdateMotes1, UpdateMotes, osPriorityNormal, 1, 128);
UpdateMotesHandle = osThreadCreate(osThread(UpdateMotes1), NULL);
    //osThreadDef(UpdateMoteActivity1, UpdateMoteActivity, osPriorityNormal, 1, 128);
    //UpdateMoteActivityHandle = osThreadCreate(osThread(UpdateMoteActivity1), NULL);
    //HAL_UART_Receive_IT(&huart1,(uint8_t *) Rx_buff1,MOTE_PKT_SIZE);
    //HAL_UART_Receive_IT(&huart2,(uint8_t *) Rx_buff2,6);
/* USER CODE END RTOS_THREADS */

/* Start scheduler */
osKernelStart();

/* We should never get here as control is now taken by the scheduler */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
    RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

    /** Initializes the CPU, AHB and APB busses clocks
     */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
    RCC_OscInitStruct.MSISState = RCC_MSILON;

```

```

RCC_OscInitStruct.MSICalibrationValue = 0;
RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
    Error_Handler();
}
/** Initializes the CPU, AHB and APB busses clocks
 */
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                               |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
{
    Error_Handler();
}
PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_USART2
                                     |RCC_PERIPHCLK_I2C1;
PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_PCLK1;
if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
{
    Error_Handler();
}
/** Configure the main internal regulator output voltage
 */
if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
{
    Error_Handler();
}
}

/**
 * @brief I2C1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_I2C1_Init(void)
{
    /* USER CODE BEGIN I2C1_Init 0 */

    /* USER CODE END I2C1_Init 0 */

    /* USER CODE BEGIN I2C1_Init 1 */

    /* USER CODE END I2C1_Init 1 */
    hi2c1.Instance = I2C1;
    hi2c1.Init.Timing = 0x00000E14;
    hi2c1.Init.OwnAddress1 = 0;
    hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
    hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
    hi2c1.Init.OwnAddress2 = 0;
    hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
    hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
    hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
    if (HAL_I2C_Init(&hi2c1) != HAL_OK)
    {
        Error_Handler();
    }
    /** Configure Analogue filter
     */
    if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)

```

```

    {
        Error_Handler();
    }
    /** Configure Digital filter
    */
    if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN I2C1_Init 2 */

    /* USER CODE END I2C1_Init 2 */

}

/**
 * @brief USART1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART1_UART_Init(void)
{

    /* USER CODE BEGIN USART1_Init 0 */

    /* USER CODE END USART1_Init 0 */

    /* USER CODE BEGIN USART1_Init 1 */

    /* USER CODE END USART1_Init 1 */
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 115200;
    huart1.Init.WordLength = UART_WORDLENGTH_8B;
    huart1.Init.StopBits = UART_STOPBITS_1;
    huart1.Init.Parity = UART_PARITY_NONE;
    huart1.Init.Mode = UART_MODE_TX_RX;
    huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart1.Init.OverSampling = UART_OVERSAMPLING_16;
    huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
    huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
    if (HAL_UART_Init(&huart1) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN USART1_Init 2 */

    /* USER CODE END USART1_Init 2 */

}

/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART2_UART_Init(void)
{

    /* USER CODE BEGIN USART2_Init 0 */

    /* USER CODE END USART2_Init 0 */

    /* USER CODE BEGIN USART2_Init 1 */

    /* USER CODE END USART2_Init 1 */
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 115200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;

```

```

huart2.Init.StopBits = UART_STOPBITS_1;
huart2.Init.Parity = UART_PARITY_NONE;
huart2.Init.Mode = UART_MODE_TX_RX;
huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart2.Init.OverSampling = UART_OVERSAMPLING_16;
huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
if (HAL_UART_Init(&huart2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN USART2_Init 2 */

/* USER CODE END USART2_Init 2 */

}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);

    /*Configure GPIO pin : PB3 */
    GPIO_InitStruct.Pin = GPIO_PIN_3;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

void ESP32_Init(void) {
    Tx_buff[0]=1;
    Tx_buff[1]='\n';
    HAL_UART_Transmit_IT(&huart2,(uint8_t*)Tx_buff,2);
    HAL_UART_Receive(&huart2,Rx_buff2,1,100);
    HAL_Delay(50);
    if (Rx_buff2[0] == '2') CONNECTED_WIFI = 1;
    while (CONNECTED_WIFI == 0) {
        HAL_UART_Transmit_IT(&huart2,(uint8_t*)Tx_buff,2);
        HAL_UART_Receive(&huart2,Rx_buff2,1,100);
        if (Rx_buff2[0] == '2') CONNECTED_WIFI = 1;
        HAL_Delay(50);
    }

    Tx_buff[0]=3;
    HAL_UART_Transmit_IT(&huart2,(uint8_t*)Tx_buff,2);
    HAL_UART_Receive(&huart2,Rx_buff2,1,100);
    if (Rx_buff2[0] == '4') CONNECTED_SERVER = 1;
    HAL_Delay(50);
    while (CONNECTED_SERVER == 0) {
        HAL_UART_Transmit_IT(&huart2,(uint8_t*)Tx_buff,2);
        HAL_UART_Receive(&huart2,Rx_buff2,1,100);
        if (Rx_buff2[0] == '4') CONNECTED_SERVER = 1;
        HAL_Delay(50);
    }
}

```

```

    }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    //if (&huart1 == huart) { /* UART1 */
    if (pktBuff.size != PKT_BUFF_MAX) {
        strncpy(pktBuff.buff[pktBuff.head].pkt, (char *)Rx_buff1, MOTE_PKT_SIZE);
        pktBuff.size++;
        pktBuff.head++;
        if (pktBuff.head == PKT_BUFF_MAX) pktBuff.head = 0;
        CURRENT_PKT_COUNT++;
    }
    HAL_UART_Receive_IT(&huart1, (uint8_t *) Rx_buff1, MOTE_PKT_SIZE);
    //}
}

void UpdateMotes(void const * arguments) {
    int loopCount = 0;
    for (;;) {
        int newTotal = 0;
        for (int i = 0; i < MAX_MOTES; i++) {
            if (networkMotes.active[i] > 0) newTotal++;
        }
        ACTIVE_MOTES = newTotal;
        loopCount++;
        if (loopCount == 10) {
            uint32_t waitOnVal;
            osSignalWait(0x0001, 1000);
            for (int i = 0; i < MAX_MOTES; i++) {
                // mark all as inactive, motes that send packets within 10 seconds will be
                // remarked as active
                networkMotes.active[i] <= 0 ? networkMotes.active[i] = 0 : networkMotes.active[i]--;
            }
            loopCount = 0;
        }
        osDelay(1000);
    }
}

void LCDTask(void const * argument)
{
    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    // int loopCount = 0;
    //char write2Uart2[20];
    int timesThrough = 1;
    for (;;)
    {

        snprintf(display, sizeof(display), "%s", "Pkt cnt: ");

        snprintf(display+8, sizeof(display)-8, "%d", CURRENT_PKT_COUNT);
        ssd1306_Fill(Black);
        ssd1306_SetCursor(0, LCD_ROW_1);
        ssd1306_WriteString(display, Font_6x8, White);

        if (MOTE_BATTERY_LOW) {
            ssd1306_SetCursor(0, (LCD_ROW_2+LCD_ROW_3)/2);
            strncpy(display, MOTE_BATTERY_LOW_ADDR, 5);
            snprintf(display+5, sizeof(display)-5, "%s", " Batt!");
            ssd1306_WriteString(display, Font_7x10, White);
        }

        ssd1306_SetCursor(0, LCD_ROW_4);
        if (CONNECTED_WIFI) {
            ssd1306_WriteSymbol(Sym_Wifi, White);
        } else {
            ssd1306_WriteSymbolSpace();
        }
    }
}

```

```

    }
    ssd1306_WriteSymbolSpace();
    if (CONNECTED_SERVER) {
        ssd1306_WriteSymbol(Sym_Server, White);
    }
    ssd1306_SetCursor(75, LCD_ROW_1);
    snprintf(display, sizeof(display), "%s", "Active:");
    ssd1306_WriteString(display, Font_7x10, White);
    ssd1306_SetCursor(93, LCD_ROW_2);
    snprintf(display, sizeof(display), "%2d", ACTIVE_MOTES);
    ssd1306_WriteString(display, Font_11x18, White);
    ssd1306_UpdateScreen();

    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3);
    //display++;
    osDelay(1000);
}
/* USER CODE END 5 */
}

/* USER CODE END 4 */

/* USER CODE BEGIN Header_StartDefaultTask */
/**
 * @brief Function implementing the defaultTask thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void const * argument)
{
    HAL_UART_AbortReceive(&huart1);
    HAL_UART_Receive_IT(&huart1, (uint8_t *) Rx_buff1, MOTE_PKT_SIZE);

    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    char currentPkt[MOTE_PKT_SIZE];
    char currentAddr[5];
    osStatus status;
    for(;;)
    {
        while (pktBuff.size > 0) {
            strncpy(currentPkt, pktBuff.buff[pktBuff.tail].pkt, MOTE_PKT_SIZE);
            // cannot interrupt when this function modifies the tail pointer and
            // number of currently held elements, since the UART RX callback can
            // interrupt at any time
            taskENTER_CRITICAL();
            pktBuff.tail++;
            if (pktBuff.tail == PKT_BUFF_MAX) pktBuff.tail = 0;
            pktBuff.size--;
            taskEXIT_CRITICAL();

            if (isalnum(currentPkt[0])) {
                // obtained packet, so now its okay to allow callback function to
                // this element's spot
                strncpy(currentAddr, currentPkt+10, 5);
                int listedMote = 0;

                // update active motes
                // { MAKE THIS A HASH MAPPING WITH SINGLE STEP OFFSET
                for (int i = 0; i < networkMotes.size; i++) {
                    if (strcmp(networkMotes.moteAddr[i].addr, currentAddr, 5) == 0) {
                        networkMotes.active[i] = MOTE_ACTIVE;
                        listedMote = 1;
                    }
                }
            }
        }
    }
}

```

```

        if (listedMote == 0 && networkMotes.size < MAXMOTES) {
            strncpy(networkMotes.moteAddr[networkMotes.size].addr, currentAddr, 5);
            networkMotes.active[networkMotes.size] = MOTE_ACTIVE;
            networkMotes.size++;
        }
    // }

    // update low battery
    char batteryStat[4];
    strncpy(batteryStat, currentPkt+5, 4);
    if (atoi(batteryStat) < MOTE_BATTERY_THRES) {
        MOTE_BATTERY_LOW = 1;
        strncpy(MOTE_BATTERY_LOW_ADDR, currentAddr, 5);
    }

    // send new data to server
    Tx_buff[0]=5;
    Tx_buff[1]='\n';
    HAL_UART_Transmit_IT(&huart2, (uint8_t *)Tx_buff, 2);
    HAL_Delay(50);
    strncpy(Tx_buff, currentPkt, 5);
    strncpy(Tx_buff+5, currentAddr, 5);
    Tx_buff[10]='\n';
    HAL_UART_Transmit_IT(&huart2, (uint8_t *)Tx_buff, 11);
    HAL_Delay(50);
    Tx_buff[0]=6;
    Tx_buff[1]='\n';
    HAL_UART_Transmit_IT(&huart2, (uint8_t *)Tx_buff, 2);
    HAL_UART_Receive(&huart2, (uint8_t *)Rx_buff2, 1, 100);
    HAL_Delay(1000);
    while (Rx_buff2[0] != '7') {
        Tx_buff[0]=5;
        Tx_buff[1]='\n';
        HAL_UART_Transmit_IT(&huart2, (uint8_t *)Tx_buff, 2);
        HAL_Delay(50);
        strncpy(Tx_buff, currentPkt, 5);
        strncpy(Tx_buff+5, currentAddr, 5);
        Tx_buff[10]='\n';
        HAL_UART_Transmit_IT(&huart2, (uint8_t *)Tx_buff, 10);
        HAL_Delay(50);
        Tx_buff[0]=6;
        Tx_buff[1]='\n';
        HAL_UART_Transmit_IT(&huart2, (uint8_t *)Tx_buff, 2);
        HAL_UART_Receive(&huart2, (uint8_t *)Rx_buff2, 1, 100);
        HAL_Delay(1000);
    }
}

}

status = osThreadYield();
configASSERT(status == osOK);
}
/* USER CODE END 5 */
}

/**
 * @brief Period elapsed callback in non blocking mode
 * @note This function is called when TIM1 interrupt took place, inside
 * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
 * a global variable "uwTick" used as application time base.
 * @param htim : TIM handle
 * @retval None
 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    /* USER CODE BEGIN Callback 0 */

```

```

/* USER CODE END Callback 0 */
if (htim->Instance == TIM1) {
    HAL_IncTick();
}
/* USER CODE BEGIN Callback 1 */

/* USER CODE END Callback 1 */
}

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */

    /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 * where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(char *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line number,
    tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/***** (C) COPYRIGHT STMicroelectronics *****END OF FILE*****/

```


4.3 ESP32 Sub-System Code

```
// for AdafruitIO
#include "config.h"
#include <string.h>

//AdafruitIO_Feed *mote1 = io.feed("f6775");
//AdafruitIO_Feed *mote2 = io.feed("cd6c7");
#define MAXMOTES 64

/*
    Next: Implement hash table with linear probing for mote addresses
    for quicker lookup
*/
class AIO_Feed_Class {
private:
    size_t currentMotes;
    String motes[MAXMOTES];
    AdafruitIO_Feed *AIOfeeds[MAXMOTES];
public:
    AIO_Feed_Class();
    ~AIO_Feed_Class();
    bool insertAddr(String addr);
    bool checkAddr(String addr);
    bool sendData(String addr, String Data);
    int getIndx(String addr);
};

AIO_Feed_Class::AIO_Feed_Class() {
    for (int i = 0; i < MAXMOTES; i++) {
        motes[i] = "00000";
    }
    currentMotes = 0;
}

AIO_Feed_Class::~~AIO_Feed_Class() {}

bool AIO_Feed_Class::insertAddr(String addr) {
    if (currentMotes == MAXMOTES) return false;
    uint32_t indx = ((0x0F & (addr[4])) | (0x0F & (addr[3] << 4)) | (0xF00 & (addr[2] << 8))\
        | (0xF000 & (addr[1] << 12)) | (0xF0000 & (addr[0] << 16)));
    indx = indx % MAXMOTES;
    while (motes[indx] != "00000") {
        indx++;
        if (indx == MAXMOTES) indx = 0;
    }
    motes[indx] = addr;
    char addrCharStr[6];
    addr.toCharArray(addrCharStr, 6);
    AIOfeeds[indx] = io.feed(addrCharStr);
    currentMotes++;
    return true;
}

bool AIO_Feed_Class::checkAddr(String addr) {
    if (currentMotes == 0) return false;
    uint16_t indx = ((0x0F & (addr[4])) | (0x0F & (addr[3] << 4)) | (0xF00 & (addr[2] << 8))\
        | (0xF000 & (addr[1] << 12)));
    indx = indx % MAXMOTES;
    int timesThrough = 0;
    while (motes[indx] != addr) {
        if (motes[indx] == NULL) return false;
        indx++;
        if (indx == MAXMOTES) indx = 0;
        timesThrough++;
        if (timesThrough == MAXMOTES) return false;
    }
}
```

```

    }
    return true;
}

int AIO_Feed_Class::getIndx(String addr) {
    if (currentMotes == 0) return -1;
    uint16_t indx = ((0x0F & (addr[4])) | (0x0F & (addr[3] << 4)) | (0xF00 & (addr[2] << 8))\
        | (0xF000 & (addr[1] << 12)));
    indx = indx % MAXMOTES;
    int timesThrough = 0;
    while (motes[indx] != addr) {
        indx++;
        if (indx == MAXMOTES) indx = 0;
        timesThrough++;
        if (timesThrough == MAXMOTES) return -1;
    }
    return indx;
}

bool AIO_Feed_Class::sendData(String addr, String Data) {
    if (checkAddr(addr) == false) insertAddr(addr);
    int indx = getIndx(addr);
    AIOfeeds[indx]->save(Data.toInt());
    return true;
}

HardwareSerial Serial3(2);
String inOperation;
String inData;

void setup() {
    // put your setup code here, to run once:
    Serial3.begin(115200);
    Serial.begin(9600);

    // connect to io.adafruit.com
    io.connect();

    // wait for a connection
    while (io.status() < AIO.CONNECTED) {
        Serial.print(".");
        delay(500);
    }
    Serial.print("Connected!");
}

void loop() {
    Serial3.flush();
    int incomingByte;
    AIO_Feed_Class aioFeeds;
    while (true) {
        io.run();
        if (Serial3.available() > 0) {
            if (Serial3.available() > 20) {
                Serial3.flush();
                continue;
            }
        }
        inOperation = Serial3.readStringUntil('\n');
        inOperation.trim();
        int oper=inOperation.charAt(0);
        Serial.print("Reveived: ");
        Serial.println(oper);
        //if (inOperation.toInt()) {
        //    oper = inOperation.toInt();
        //} else {continue;}

        // mote system requests wifi connection

```

```

if (oper == 1) {
  if (io.status() >= AIO.CONNECTED) {
    Serial.println("Sending Wifi Ack");
    Serial3.print((int)2);
  } else {
    io.connect();
    while (io.status() < AIO.CONNECTED) {
      delay(500);
    }
    Serial.print("Sending Wifi Ack");
    Serial3.print((int)2);
  }
}

// mote system requests server connection
} else if (oper == 3) {
  if (io.status() >= AIO.CONNECTED) {
    Serial.println("Sending Server Ack");
    Serial3.print(4);
  } else {
    io.connect();
    while (io.status() < AIO.CONNECTED) {
      delay(500);
    }
    Serial.println("Sending Server Ack");
    Serial3.print(4);
  }
}

// mote system requests data transfer to server
} else if (oper == 5) {
  Serial.println("Request to send data");
  delay(5);
  delay(5);
  inData = Serial3.readStringUntil('\n');
  String Data = inData.substring(0, 4), Addr = inData.substring(5, 11);
  Data.trim(), Addr.trim();
  Serial.print("Addr:");
  Serial.print(Addr);
  Serial.print(" Data:");
  Serial.println(Data);
  aioFeeds.sendData(Addr, Data);
  Serial3.print(7);
}
}
delay(50);
}
}

```

4.4 TelosB Server Code

```
/*
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the Institute nor the names of its contributors
 * may be used to endorse or promote products derived from this software
 * without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ‘‘AS IS’’ AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * This file is part of the Contiki operating system.
 */

#include "contiki.h"
#include "net/routing/routing.h"
#include "net/netstack.h"
#include "net/ipv6/simple-udp.h"

#include "sys/log.h"
#define LOG_MODULE "App"
#define LOG_LEVEL LOG_LEVEL_INFO

#define WITHSERVER_REPLY 1
#define UDP_CLIENT_PORT 8765
#define UDP_SERVER_PORT 5678

static struct simple_udp_connection udp_conn;

PROCESS(udp_server_process, "UDP server");
AUTOSTART_PROCESSES(&udp_server_process);
/*-----*/
static void
udp_rx_callback(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)
{
    /*char SIG_ADDR[5] = {sender_addr->u8[11], sender_addr->u8[12], sender_addr->u8[13], \
                        sender_addr->u8[14], sender_addr->u8[15]};*/
    printf("%.5s ", datalen, (char *) data);
    uint8_t recv_addr[5];
    int i = 0;
    while (i < 3) {
        //for (int i = 0; i < 4; i++) {
            recv_addr[i*2] = 0xF & (sender_addr->u8[15-i]);
            if (i < 2)
```

```

        recv_addr[i*2 + 1] = sender_addr->u8[15-i] >> 4;
        i++;
    }
    i = 4;
    while (i >= 0) {
        if (recv_addr[i] <= 9) printf("%d",recv_addr[i]);
        else if (recv_addr[i] == 10) printf("a");
        else if (recv_addr[i] == 11) printf("b");
        else if (recv_addr[i] == 12) printf("c");
        else if (recv_addr[i] == 13) printf("d");
        else if (recv_addr[i] == 14) printf("e");
        else if (recv_addr[i] == 15) printf("f");
        i--;
    }
    printf("\n");
    //printf("%d %d %d %d %d",recv_addr[0],recv_addr[1],recv_addr[2],recv_addr[3],
    //recv_addr[4]);
    //LOG_INFO_6ADDR(sender_addr);
    //LOG_INFO_6ADDR(sender_addr);
#ifdef WITH_SERVER_REPLY
    /* send back the same string to the client as an echo reply */
    //LOG_INFO("Sending response.\n");
    simple_udp_sendto(&udp_conn, data, datalen, sender_addr);
#endif /* WITH_SERVER_REPLY */
}
/*-----*/
PROCESS_THREAD(udp_server_process, ev, data)
{
    PROCESS_BEGIN();

    /* Initialize DAG root */
    NETSTACK_ROUTING.root_start();

    /* Initialize UDP connection */
    simple_udp_register(&udp_conn, UDP_SERVER_PORT, NULL,
        UDP_CLIENT_PORT, udp_rx_callback);

    PROCESS_END();
}
/*-----*/

```

References

- [1] Advantic Sistemas Y Servicios S.L., “CM5000 DATASHEET,” TelosB datasheet, Oct., 2010.
- [2] Texas Instruments, “CC2630 SimpleLink™ 6LoWPAN, ZigBee® Wireless MCU,” CC2630 datasheet, July, 2016.
- [3] Texas Instruments, “CC2538 Powerful Wireless Microcontroller System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN, and ZigBee® Applications,” CC2538 datasheet, April, 2015.
- [4] Texas Instruments, “CC2650 SimpleLink™ Multistandard Wireless MCU,” CC2650 datasheet, July, 2016.
- [5] “CC13xx/CC26xx Hardware Configuration and PCB Design Considerations,” Texas Instruments Incorporated, May, 2020.
- [6] “CC2650 SimpleLink™ Multistandard Wireless MCU,” Texas Instruments Incorporated, Mar., 2015.
- [7] “Humidity and Temperature Sensor Node for Star Networks Enabling 10+ Year Coin Cell Battery Life,” Texas Instruments Incorporated, May, 2016.
- [8] Texas Instruments Incorporated, “CC2538EM Reference Design,” 2020. [Online]. Available: <https://www.ti.com/tool/CC2538EM-RD>. [Accessed Sept. 28, 2020].