NSF NeTs Small RUI: Wireless Sensor Network Project

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:

10/01/2018 - 09/30/2019

Student Assistant:

Russel Schellenberg
Calvin Jarrod Smith
Department of Electrical and Computer Engineering
Lyles College of Engineering

Contents

1 Purpose **3**

2 Current Projects **3**
2.1 Bulldog Mote . . . 3
2.2 Wireless Gateway Implementation . . . 3
2.2.1 Tmote Sky Mote . . . 3
2.2.2 Gateway Website . . . 7
2.3 MANET Routing Protocols and Software . . . 8
2.3.1 Contiki OS . . . 8

3 HM-10 BLE Communication Module **10**
3.1 Overview . . . 10
3.2 HM-10 Module Configuration . . . 16
3.3 Connecting Two HM-10 Devices . . . 17
3.3.1 Setting up Slave Device . . . 18
3.3.2 Setting up Master Device . . . 18
3.3.3 Discovery Connection . . . 19
3.4 Connecting to HM-10 from Android Phone . . . 23

4 Wireless Gateway Implementation **30**
4.1 Gateway Operating System and Environment . . . 30
4.2 Required hardware: . . . 30
4.3 Installation . . . 30
4.4 Contiki Program . . . 31
4.5 Gateway Program . . . 32
4.6 Web Server . . . 32

5 One-Hop, Ad-Hoc Sensor Network with Cooja **36**
5.1 Overview . . . 36
5.2 Peripheral Node Gateway Detection . . . 36
5.3 Simulation . . . 37
5.4 Simulation Results . . . 39

A Appendix **42**
A.1 Wireless Gateway, Contiki: example-broadcast.c . . . 42
A.2 Wireless Gateway: gatewayProg.py . . . 45
A.3 Wireless Gateway: index.php . . . 46
A.4 Wireless Gateway, Web Server: fetch.php . . . 51
A.5 Wireless Gateway, Web Server: databaseaccess.php . . . 52
A.6 Wireless Gateway, Web Server: getid.php . . . 53
A.7 One-Hop Algorithm: sensor-network-leaf.c . . . 55
A.8 One-Hop Algorithm: sensor-network-root.c . . . 58

List of Figures

1 Tmote Sky from ADVANTICSYS®. 4
2 Gateway bridging multiple networks together. 4
3 Raspberry Pi 3B+ Model. 5
4 Huawei 4G USB Dongle. 6
5 SORACOM SIM Card. 6
6 Completed Gateway Configuration. 7
7 Dynamic Website Displaying Mote's Sensor Data. 8
8 HM-10 Module from DSD Tech 10
9 HM-10 Pins 10
10 SH-U09F FTDI Module 11
11 UART Communication Configuration 12
12 Selecting Device Manager from the Start Menu 13
13 Device Manager 14
14 Realterm: Serial Monitor when it first starts. 15
15 Realterm Port Settings for FTDI Device. 15
16 Successful FTDI Communication in Realterm. 16
17 Master Discovery Readings. 19
18 Master (Bulldogs2) and Slave (Bulldogs) Devices Connected. 20
19 Master and Slave Sending Strings. 21
20 Master Connecting to Slave Based on MAC Address. 22
21 Bluetooth Adapter Disabled. 24
22 Scanning for BLE Devices. 25
23 HM-10 Services. 26
24 HM-10 ESS Characteristics. 27
25 Sending Temperature Data. 28
26 Temperature Reading in nRF Connect. 29
27 Database Structure. 33
28 Drop down list of motes within the network. 34
29 Site displaying selected mote's data. 35
30 Example of a One-Hop, Ad-Hoc Sensor Network. 36
31 Node Placement in One-Hop Algorithm Simulation. 37
32 Node Communication in One-Hop Algorithm Simulation. 38
33 Message Log of One-Hop Algorithm Simulation. 39

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). 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 Wireless Sensor Networks (WSNs).

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

2 Current Projects

2.1 Bulldog Mote

One project currently being engaged in is the development of a new wireless sensor mote to be used by California State University, Fresno. This new mote will be a brand-new device using existing and up-and-coming technologies to be used by students to further research efficient routing algorithms, device power consumption, network security, etc.

To build the new Bulldog Mote, several wireless protocols have been researched to facilitate communication such as ZigBee, Bluetooth® Low Energy (BLE), LoRa®, etc. These different protocols have similar but varying capabilities that make each one conducive under different circumstances. For the case of the Bulldog Mote, the first design will likely have BLE wireless communication due to the prevalence of Bluetooth devices and the recently released Bluetooth Mesh protocol which is compatible with BLE.

This protocol was tested and experimented using the HM-10 BLE module and the results are documented in the section titled: *HM-10 BLE Communication Module*.

2.2 Wireless Gateway Implementation

2.2.1 Tmote Sky Mote

A previously existing mote on the market is the Tmote Sky developed by the University of California, Berkeley and produced by ADVANTICSYS. This device contains IEEE 802.15.4 compliant wireless sensor for mote to mote communication as well as different sensors to measure light, humidity, and temperature all on one board. The mote can be powered by two AA batteries, which can be inserted into the attached battery pack, or may be powered

through a USB connection.

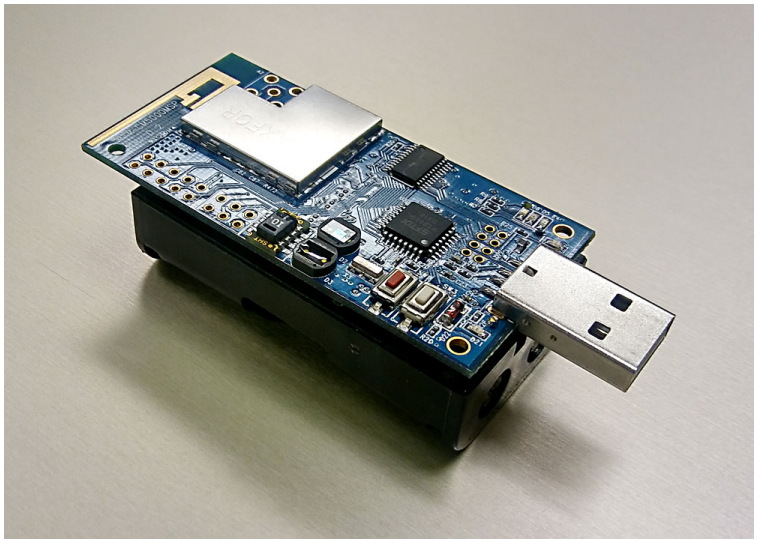


Figure 1: Tmote Sky from ADVANTICSYS®.

As versatile as the Tmote is, it lacks the capacity to display gathered data and TCP/IP without being properly hosted. Implementing a gateway into a network allows all data to be collected by the motes in a central location. It can then be transferred to another network, such as an online database. A gateway is a portal between multiple networks in which they are able to connect together. The image of a how a gateway bridges a wireless network containing motes and a TCP/IP network is shown in Figure 2.

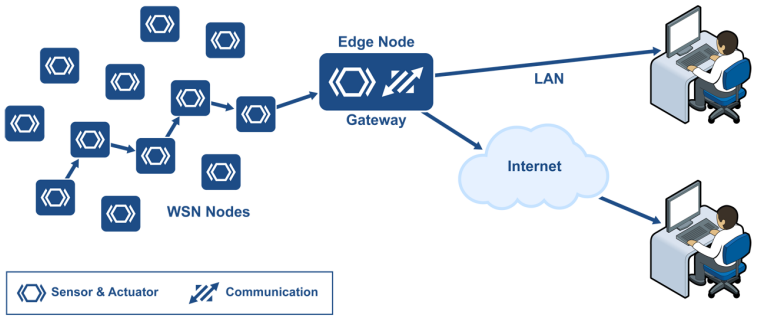


Figure 2: Gateway bridging multiple networks together.

For this project, a small single-board computer, the Raspberry Pi 3b+ (RPi) was implemented as the Gateway. This low cost, low energy, versatile computer was ideal for creating a gateway necessary to connect motes to another network. With the proper operating system configured on the Raspberry Pi (RPi), the Contiki program can be run to communicate with the connected mote. This would function as a "sink" to collect the data transmitted by

neighboring motes. The RPi is equipped with Wi-Fi and Bluetooth capabilities, though is unable to wirelessly communicate with the motes. By connecting a Mote directly into the RPi via USB, the RPi is able to read the data that is being collected.

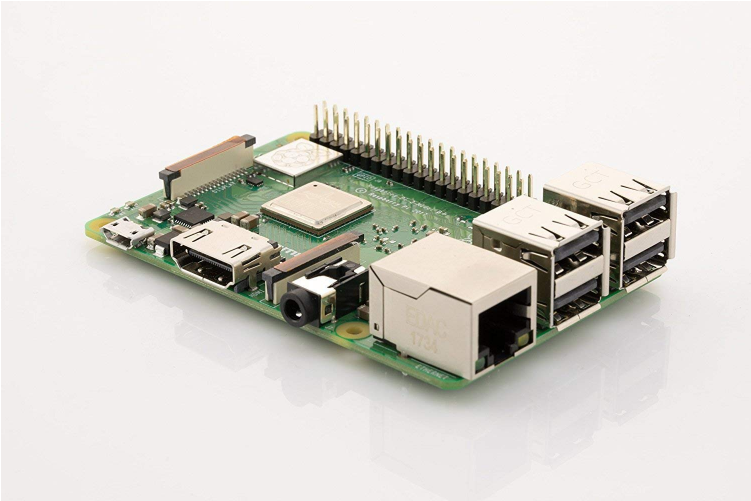


Figure 3: Raspberry Pi 3B+ Model.

The RPi is a miniature computer capable of utilizing other additional components. A wireless 4G USB dongle Huawei E3372 was implemented to allow for wireless mobile telecommunication as another means for the gateway to communicate with a server. By removing the restraints of the limited range of a Wi-Fi signal, the gateway and its mote network can now be adapted to a much wider range of environments. The Huawei 4G USB dongle used is shown in Figure 4.

Figure 4: Huawei 4G USB Dongle.

To acquire a cellular connection, a SIM card and service provider is needed to be purchased. Since the gateway will be transmitting very small packets of data to the web server, data size could be minimal. It was decided to use the company SORACOM. Their service charges a low upkeep cost of $0.06 a day, and $0.073 per megabyte of data. This service is ideal for development to keep costs to a minimum. An image of the SORACOM SIM card that was used is shown in Figure 5.

Figure 5: SORACOM SIM Card.

The completed gateway consists of the RPi with a Tmote Sky connected via USB as the

gateway mote. The Huawei USB dongle was used to allow for the mote network to function in a remote area. A stable power supply was needed to power the gateway, though once configured, a keyboard, mouse, and display was found not necessary. The finished gateway is shown below in Figure 6.

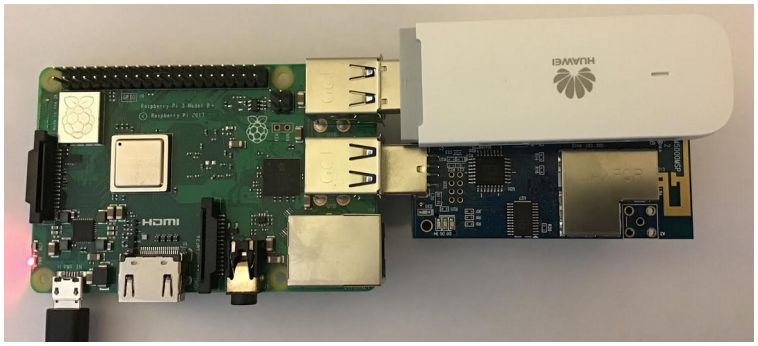
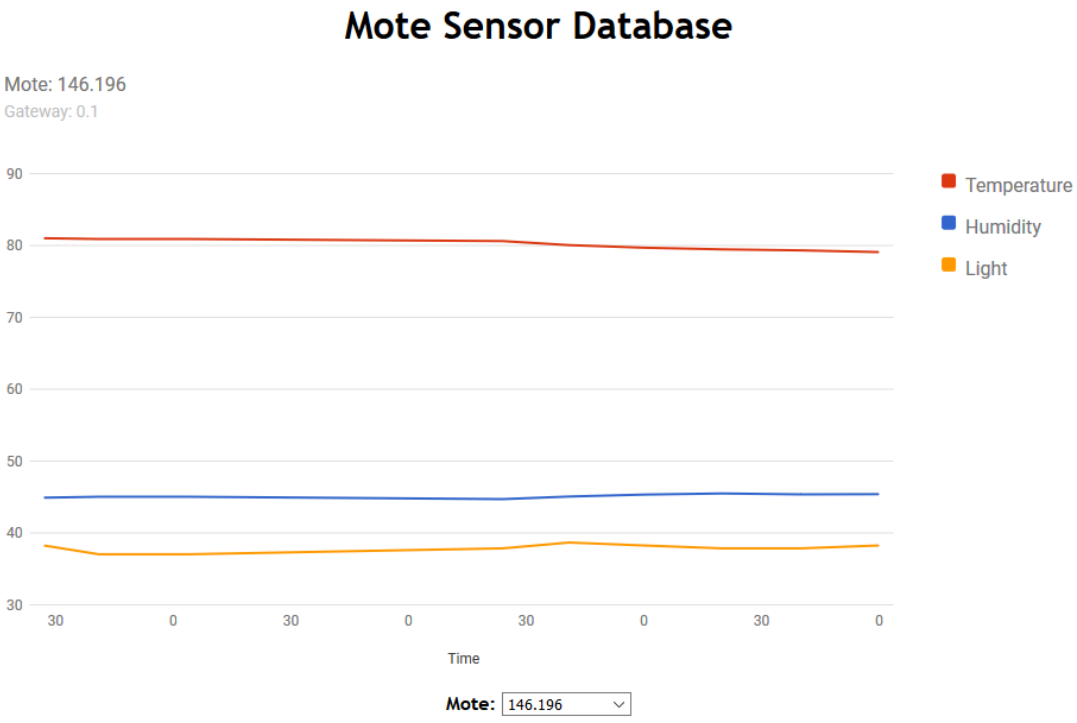


Figure 6: Completed Gateway Configuration.

2.2.2 Gateway Website

To properly view data being collected, the gateway sends data to a web database. A dynamic website was created to display the mote's sensor data, which maps it out over time. Cloud services were considered, but dismissed with the high cost of features that were not necessary, and could be carried out with a proper database. A snapshot of the website is displayed in Figure 7 below, showing the data gathered from mote 146.196.



Entries:10

Mote ID	Date	Temperature	Humidity	Light
146.196	2019-08-05 16:18:00	79.08	45.4	38.26
146.196	2019-08-05 16:17:40	79.32	45.37	37.86
146.196	2019-08-05 16:17:20	79.46	45.5	37.86
146.196	2019-08-05 16:17:00	79.69	45.34	38.26
146.196	2019-08-05 16:16:41	80.05	45.07	38.67
146.196	2019-08-05 16:16:24	80.61	44.71	37.86
146.196	2019-08-05 16:15:04	80.9	45.04	37.04
146.196	2019-08-05 16:15:00	80.9	45.04	37.04
146.196	2019-08-05 16:14:41	80.9	45.04	37.04
146.196	2019-08-05 16:14:27	81.01	44.91	38.26

Figure 7: Dynamic Website Displaying Mote's Sensor Data.

2.3 MANET Routing Protocols and Software

For WSNs to be effective in the field, many need to collect data a distance away from a central node that connects to the internet. This makes data and packet routing essential. For testing these algorithms, two software packages can be used: Contiki OS and NS-3.

2.3.1 Contiki OS

Contiki OS is a software package used to program the Tmote Sky, but in addition has a visual simulator called Cooja as part of its installation. This operating system is an important step to developing a new Bulldog Mote as the software is quite extensive and a great foundation for developing new algorithms. This software however is specific to the Tmote Sky and a few

other motes, so an entirely new operating system will need to be written for the Bulldog Mote.

A basic routing algorithm was developed in the Contiki OS that was used as a starting point to using the system. This new algorithm was a single-hop, ad-hoc wireless sensor network made to facilitate any number of sensor nodes that can connect directly to a gateway device. This program was simulated in Cooja and is shown in the section *One-Hop, Ad-Hoc Sensor Network in Cooja*.

Despite the completeness of Contiki OS, there are several improvements that can be implemented such as different structures of routing tables and additional features within current routing protocols. Such improvements are in the process of being put in place and tested.

3 HM-10 BLE Communication Module

Figure 8: HM-10 Module from DSD Tech

The HM-10 Module is a Bluetooth 4.0 Module produced by DSD Tech that includes Bluetooth Low Energy (BLE). This module transmits over the 2.4 GHz ISM band like traditional Bluetooth, however BLE uses considerably less power while still maintaining its effective communication range. This makes BLE a viable option for IoT communication between devices.

3.1 Overview

The BLE device communicates with a host microcontroller using UART. Depending on the supplier the device is purchased through will determine the number of pins. For this tutorial, the four middle pins will be used: `RXD`, `TXD`, `GND` and `VDD`.

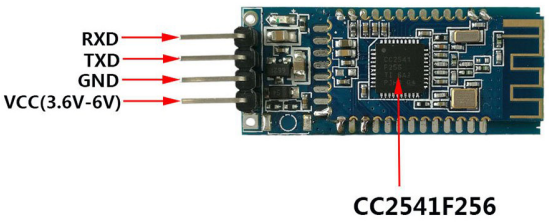


Figure 9: HM-10 Pins

The commands to control this device have to be understood before it is plugged into a microcontroller. To do this, an FTDI module can be used to connect the UART pins of the HM-10 directly to a computer. There, a serial connection can be established with the module and will give the user an easier time testing and debugging commands.

Any FTDI module will work, but be sure to purchase one with a genuine FTDI chip. Most computers have firmware installed already that will make these devices plug-and-play ready. However if the FTDI chip is not a genuine one, the modern firmware will be able to tell and break the counterfeit chip, rendering it unusable.

The FTDI component used here is the SH-U09F, produced by DSD Tech as well.

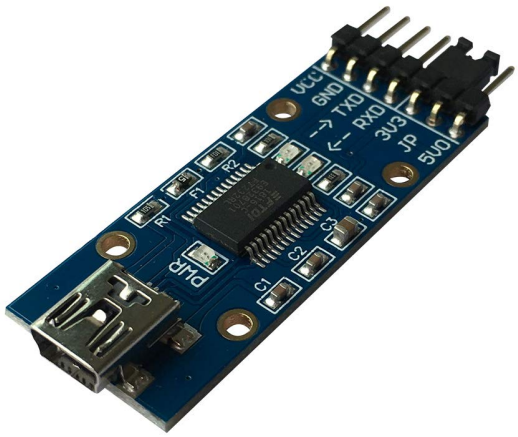


Figure 10: SH-U09F FTDI Module

The current drivers should be downloaded and installed before plugging the FTDI component into the computer. The current FTDI drivers can be found at `http://www.ftdichip.com/Drivers/VCP.htm`. There you can select the correct firmware for your operating system. Once installed plug the device into your computer. The SH-U09F module here uses a micro USB cable to connect to the host computer.

After the computer finishes configuring the FTDI module, unplug the device. Next the FTDI module will be connected to the BLE module. The HM-10 device uses 3.3 V logic and power, so on the SH-U09F module the jumper between the 5V0 pin and JP pin should be changed to connect the 3V3 and JP pins instead. This changes the logic and VCC voltages to 3.3 V. Next, the pins of the devices can be connected with jumper wire. The following diagram should be followed:

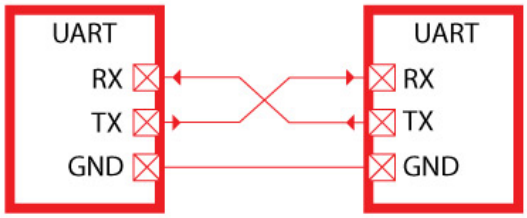


Figure 11: UART Communication Configuration

In UART communication, the receive line (Rx) on one device is connected to the transmit (Tx) of the other, and visa-versa. VCC and GND should also be connected.

Once connected, a serial communication program can be used to send and receive data from the user's computer through a connected USB to the end device on the other side of the FTDI module. The program used here to send and receive the UART data is *Realterm: Serial Terminal*: `https://realterm.sourceforge.io/index.html#downloads_Download`. Go to the link and click the download option at the top of the page to get the install executable.

Once installed and opened, the program will connect automatically to a device with a COM port. So plug in the USB of the FTDI module again, then open the **Device Manager** by clicking the *Start* menu, then type *Device Manager* in the *Search* text box. Open the Device Manager once it pops up.

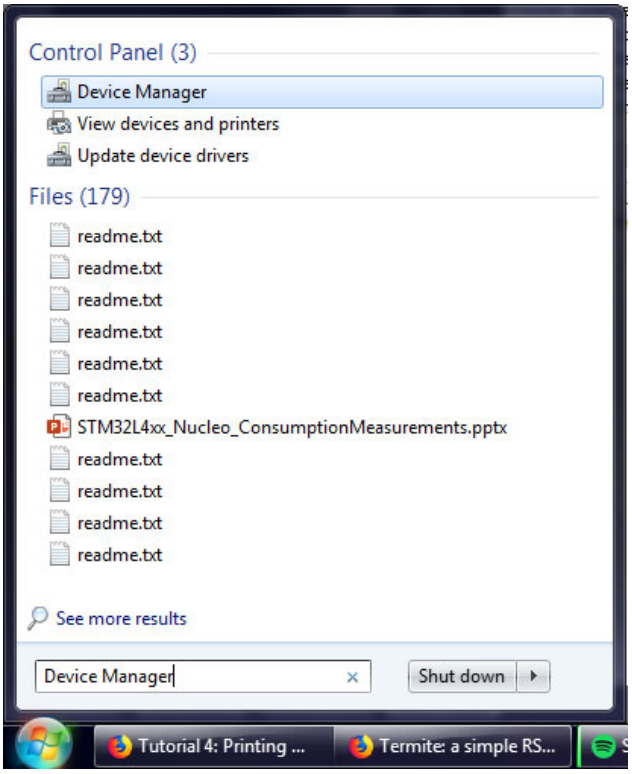


Figure 12: Selecting Device Manager from the Start Menu

Once opened, go to the *Ports* list to identify the COM number of the FTDI module. In this case it is *COM11*.

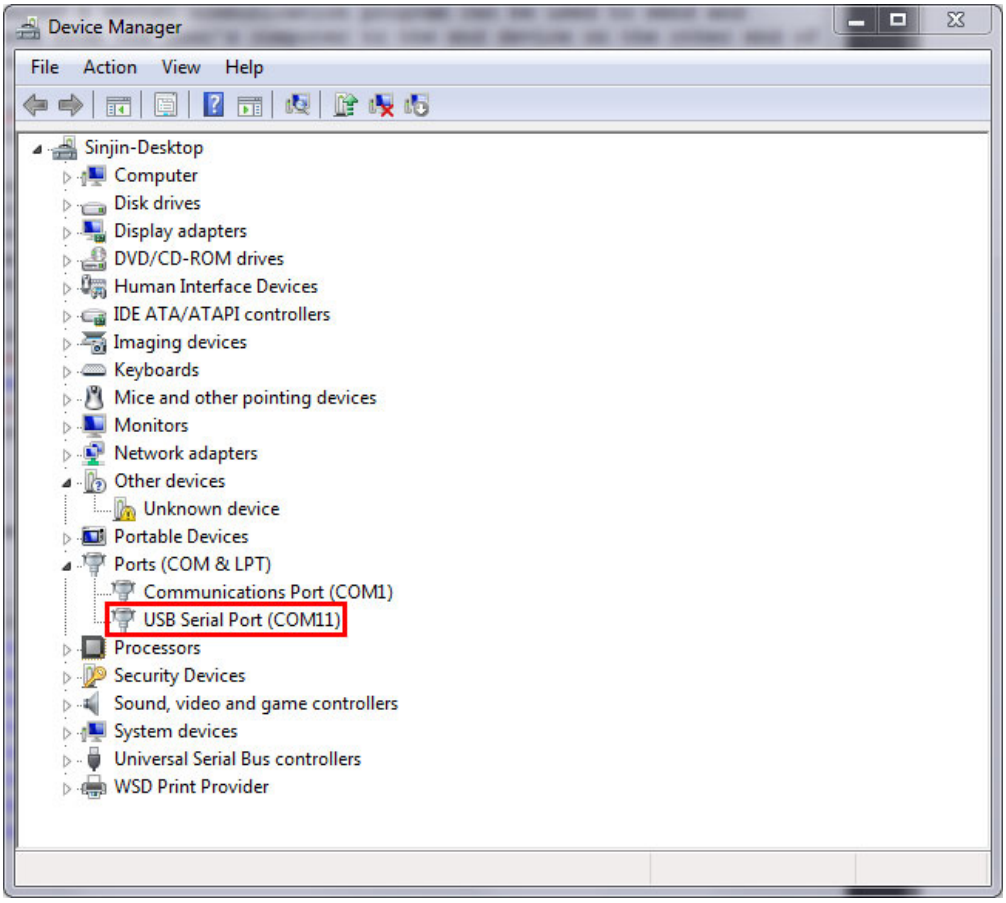


Figure 13: Device Manager

If there are many devices connected and it is difficult to tell which is the correct one, then right click on them one at a time and go to **Properties**. Under *Manufacturer* should be *FTDI* for the correct module.

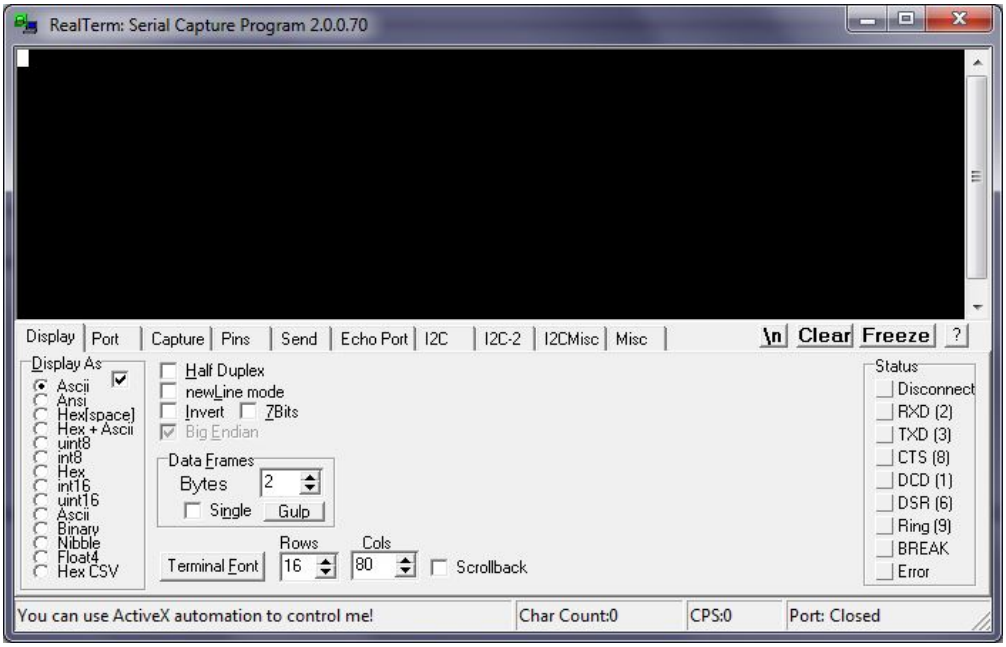


Figure 14: Realterm: Serial Monitor when it first starts.

The first tab displayed in Realterm is just how the serial information is displayed. Leave the *Display As* option as *ASCII*. Next, under the *Port* tab, select the correct port number for the FTDI device and a Baud of 9600. Then click on the *Open* button to connect to the module.

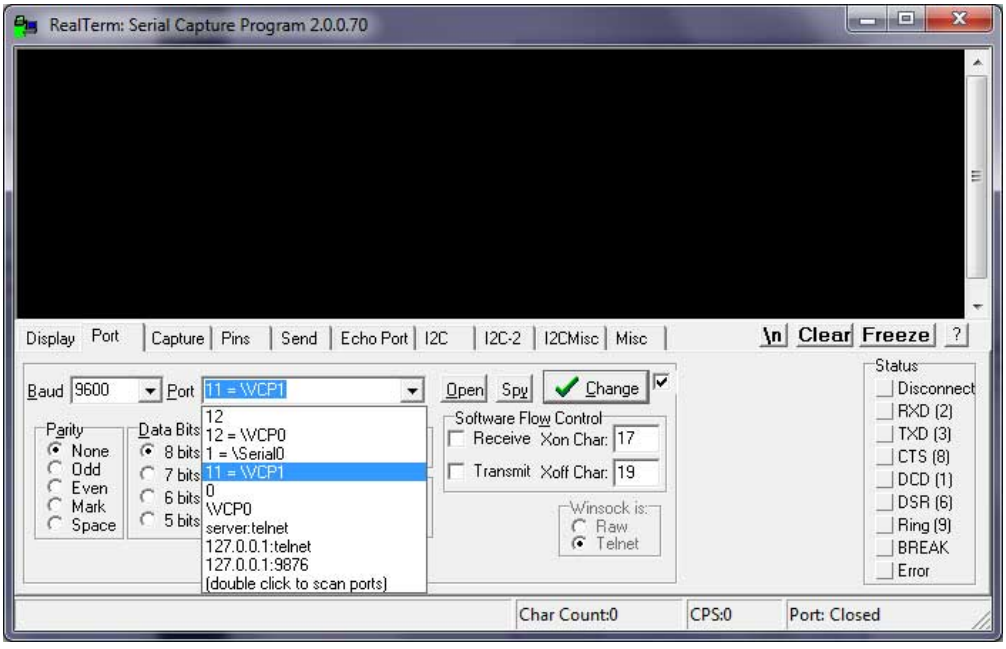


Figure 15: Realterm Port Settings for FTDI Device.

The FTDI device will now be ready to talk to. To start, click on the *Send* tab. There

will be two identical lines with a text input and buttons reading *Send Numbers* and *Send ASCII*. Either of the two lines can be used to send data. There will be cases where you'll want to send raw binary/hexadecimal data to the device where the *Send Numbers* will be useful. But for configuring the FTDI module ASCII encoded characters are what we need. Also, to have FTDI responses on separate lines, click the *After* check box in the $\backslash n$ section. Otherwise all of the responses will run on the same line.

Now the connection can be tested. In the text box type **"AT"** (without quotes) and click *Send ASCII*. If the connection is successful there should be a response of **"OK"** in the window.

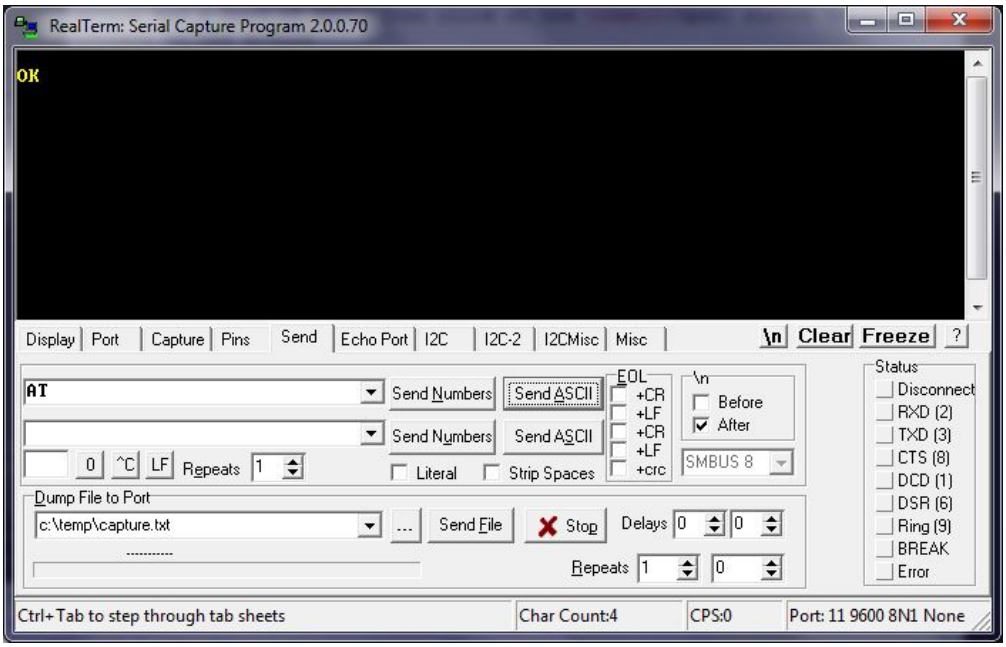


Figure 16: Successful FTDI Communication in Realterm.

3.2 HM-10 Module Configuration

Some default settings in the HM-10 module need to be changed after the module is wired up and communication is verified in Realterm.

While in Realterm, type: **"AT+IMME?"**. If the response is **"OK+Get:0"**, then type **"AT+IMME1"** to change the settings. This setting is used to tell the module to start in *command mode* or *work mode* when the device starts up. In command mode (changed by sending *"AT+IMME1"*), the component will only perform what the user asks based on the **AT** commands. In work mode (*"AT+IMME1"*), the device will automatically perform actions like connecting to neighboring devices, enabling discovery, etc. based on its current settings on start up.

Next, we'll want to see the messages that the device displays, so the next command will turn on notifications: **"AT+NOTI1"**. Then, type **"AT+NOTP1"** to show the MAC

addresses related to the notification.

The HM-10 device likely has a default name (not the same as the MAC address). So give the device a unique name by typing **"AT+NAME[name]"** where "[name]" is a string. For example, if you want to name the device *Bulldogs*, then the command would be **"AT+NAMEBulldogs"**.

Next, all Bluetooth Low Energy devices use some UUID value to identify what type of information to expect in a standardized manner. To change the UUID value sent the command: **"AT+UUID[uuid]"** where "[uuid]" is a valid UUID value in hexadecimal with a hexadecimal prefix. For example, the UUID value 0x181A is the Bluetooth assigned number for the *Environment Sensing* service. For more information on the Bluetooth protocol, read the *Bulldog BLE Handbook* (coming soon).

Similarly, the service characteristic value can also be set on the HM-10. Under *Environmental Sensing Service* there are several characteristics, one of which is *Temperature* with a UUID of 0x2A6E. Use the command **"AT+CHAR[char]"** where "[char]" is the UUID value of the characteristic to set the device's characteristic.

Something to keep in mind is that when two HM-10 modules are connected to each other, no AT-commands will be valid until the connection is broken. This can be done manually by simply typing **"AT"**. The module should return **"OK+LOST"** when successfully disconnected.

3.3 Connecting Two HM-10 Devices

Two HM-10 devices can be connected together once the settings listed previously are changed. Actually the devices *could* be connected prior to running those commands, but to save a head-ache or two its advisable to run above commands first.

Multiple instances of Realterm can be run on the same computer so this process can be completed on a single computer with two HM-10 devices connected to it. Whether you decide to connect the other HM-10 to the same computer or not, perform the same steps in the *Overview* and *HM-10 Module Configuration* sections of this document to set up the second FTDI and HM-10 device separately.

Once both devices are setup and connected to the computer, then they can be paired and bonded. To do so takes a few steps.

First, one device must be selected as the master and the other the slave device. In Bluetooth Point-to-Point protocols, the slave device is the one that advertises its presence and the master device is the one that makes the connection between the two devices.

To set the master/slave roles for each device use the command **"AT+ROLE"**.

3.3.1 Setting up Slave Device

To change the role of the module to a slave use the command: **"AT+ROLE0"**. Doing this starts the module advertising its presence to BLE master devices. As a note, some datasheets call the slave device the Peripheral Node.

Next, to change the interval in which the slave sends an advertisement use the command: **"AT+ADVI[P1]"**, where **[P1]** is a value 0 through F and represents different time intervals in Table 1:

[P1]	Advertising Interval (ms)	**[P1]**	Advertising Interval (ms)
0	100	8	1022.5
1	152.5	9	1285
2	211.25	A	2000
3	318.75	B	3000
4	417.5	C	4000
5	546.25	D	5000
6	760	E	6000
7	852.5	F	7000

Table 1: ADVI Values

The default value of 1285 ms is fine for our purposes.

Also to keep in mind, there are different advertising types under the **"AT+ADTY"** command. The default allows advertising, scanning response and makes the device openly connectable. To set the device back to its default type use: **"AT+ADTY0"**.

Lastly, some device won't advertise unless the advertising data flag is set. To do this the *AT+FLAG* command is used. Type **"AT+FLAG0"** and click *Send ASCII*. This command can use values from 0 through FF, however using any value will get the device to advertise.

Now the slave device is advertising its presence and ready to be connected to.

3.3.2 Setting up Master Device

Setting up the master is a little simpler than the slave device. The command to set the device to master is **"AT+ROLE1"**. The master is also called the Central Node in some datasheets.

Once set the device can immediately begin scanning for slave devices. Type **"AT+DISC?"** to do so. If the slave is configured correctly the terminal for the master should read: **"OK+DISCS"** (starting discovery), then **"OK+DIS0:[MACADDR]"** (where **[MACADDR]** is the MAC address of the slave module), and finally **"OK+DISCE"** (for ending discovery process). Likely all of these messages will be on the same line.

If there were multiple BLE slave devices available and in range there will be a **"OK+DIS1:[MACADDR]"**. The important thing here is the value after *OK+DIS*. This value represents the array index of the connectable BLE devices.

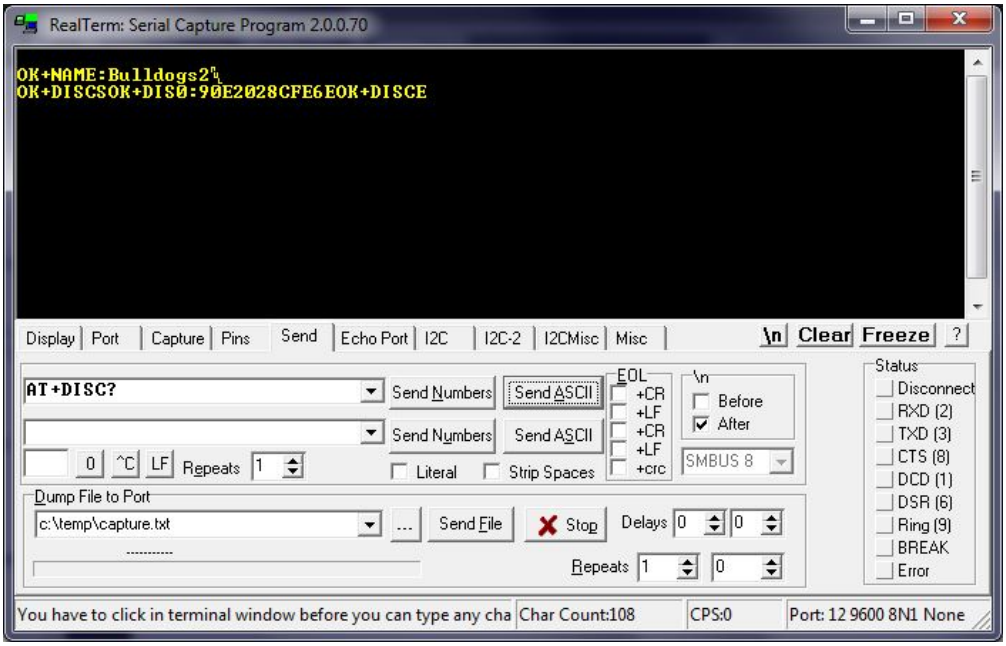


Figure 17: Master Discovery Readings.

3.3.3 Discovery Connection

From Figure 17, the slave device is identified by **"OK+DIS0:90E2028CFE6E"** meaning that the device with MAC address 90:E2:02:8C:FE:6E can be connected to using index 0. So now the master can connect to this discovery device using the command **"AT+CONN[DISC]"** where **[DISC]** is the array index. So in the case of Figure 17 the command would be **"AT+CONN0"**. Doing this connects to the slave device and shows in both devices terminals that they are connected to each others MAC addresses.

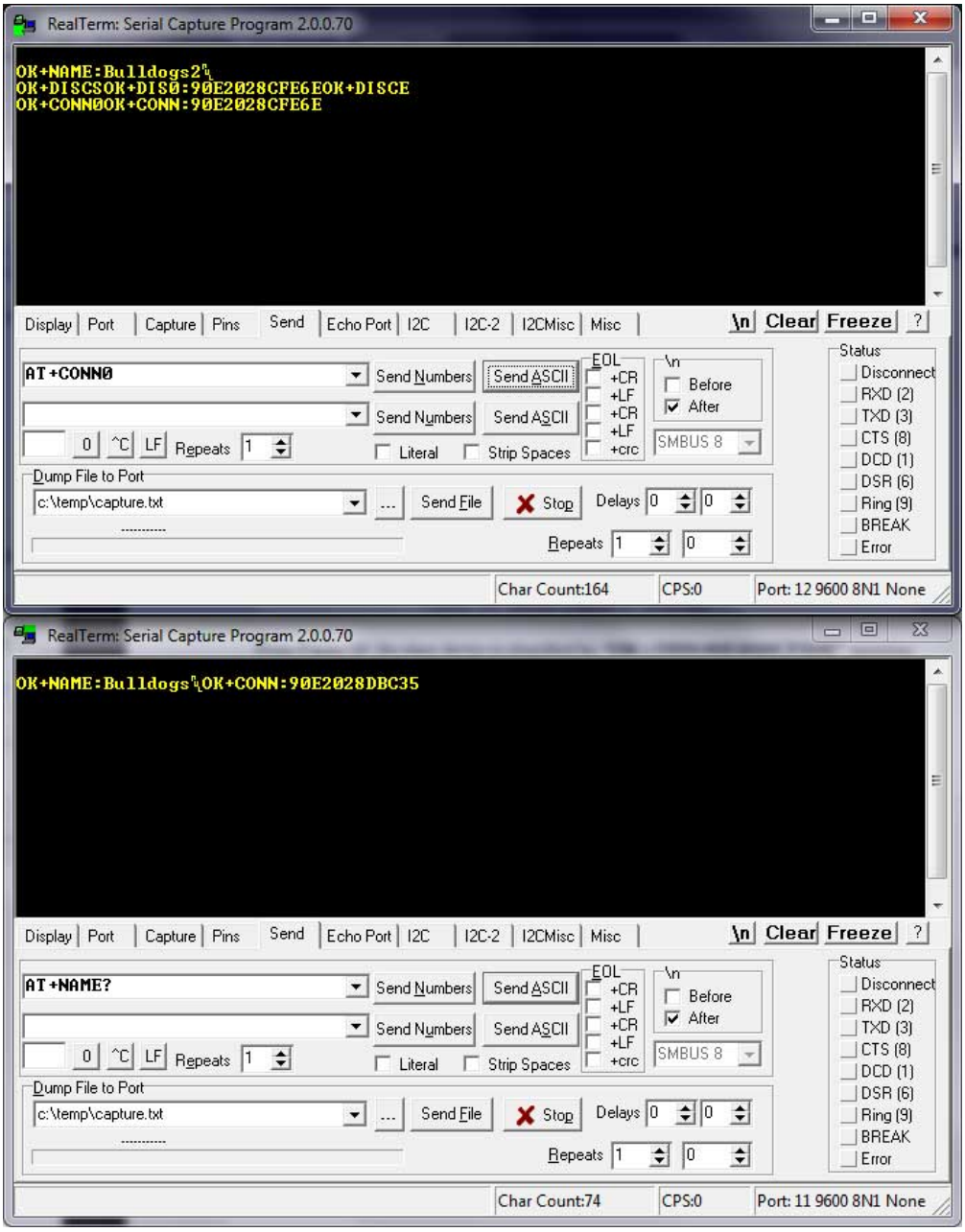


Figure 18: Master (Bulldogs2) and Slave (Bulldogs) Devices Connected.

Now simple strings or numbers can be transmitted from slave to master and visa-versa. In transmit mode, none of the *AT* commands will be recognized except the **"AT"** command, which disconnects both devices.

Note: Once the slave device is disconnected the advertising flag needs to be set again before it can be discovered.

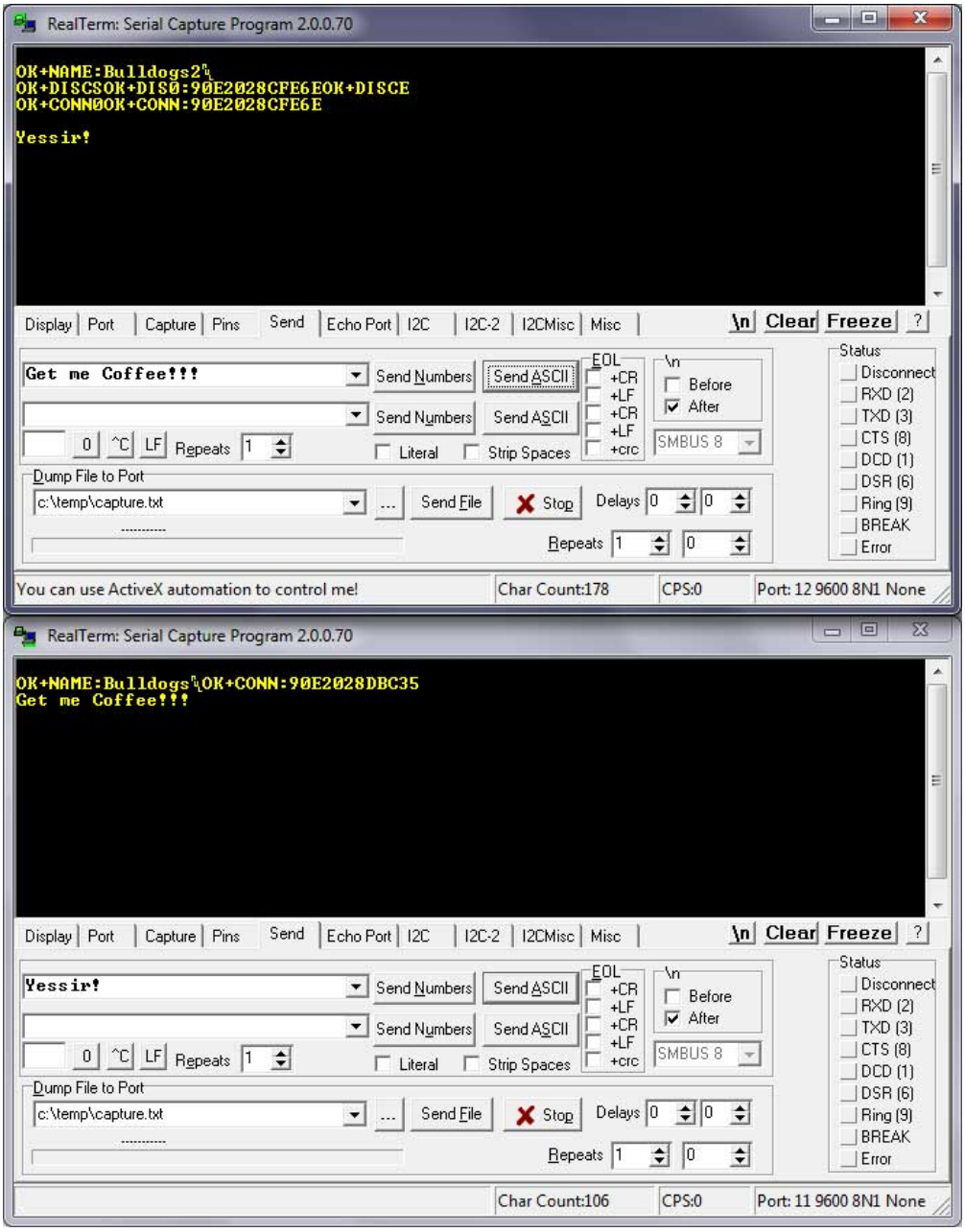


Figure 19: Master and Slave Sending Strings.

When connecting to a device its sometimes more desirable to connected to a device based on its MAC address instead of discovery array index. The command **"AT+CON[MACADDR]"** can be used to so this from the master. The discovery command should still be run first to know if the slave device is available to be connected to first.

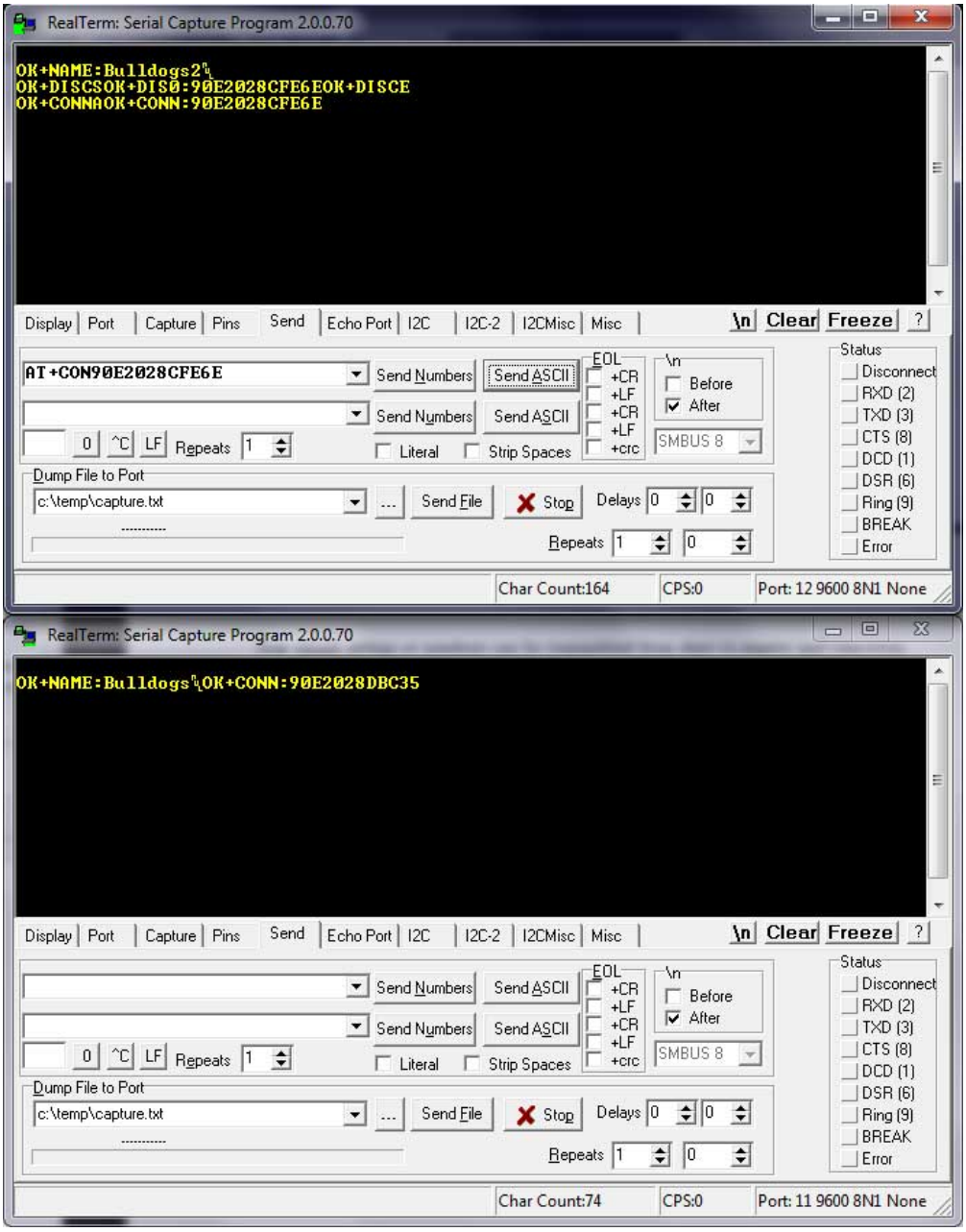


Figure 20: Master Connecting to Slave Based on MAC Address.

3.4 Connecting to HM-10 from Android Phone

In the last part, two HM-10 devices were connected, one being the master and the other the slave. In this part, one HM-10 module will be used as a slave device and a smart phone will be used as the master device. This part completely depends on the capabilities of your smart phone. Make sure your phone has BLE capabilities to be able to do this.

If the phone does have BLE capabilities, the *nRF Connect* can be downloaded from the Google Play Store. nRF Connect is an app by Nordic Semiconductor used to connect and communication to any BLE device.

First however, one of the HM-10 modules should be set up as a slave device as shown in the last section. In addition, the UUID for the slave device's service and characteristic should be configured as described in the *HM-10 Module Configuration*. Make sure to reset the module after making changes to apply them using the **"AT+RESET"** command.

Once the slave device is set up and advertising its presence, the Android device can detect and connect to it. To do so, install and open the nRF Connect app. If the Bluetooth adapter is disabled it there will be a red bar instructing you to enable it.

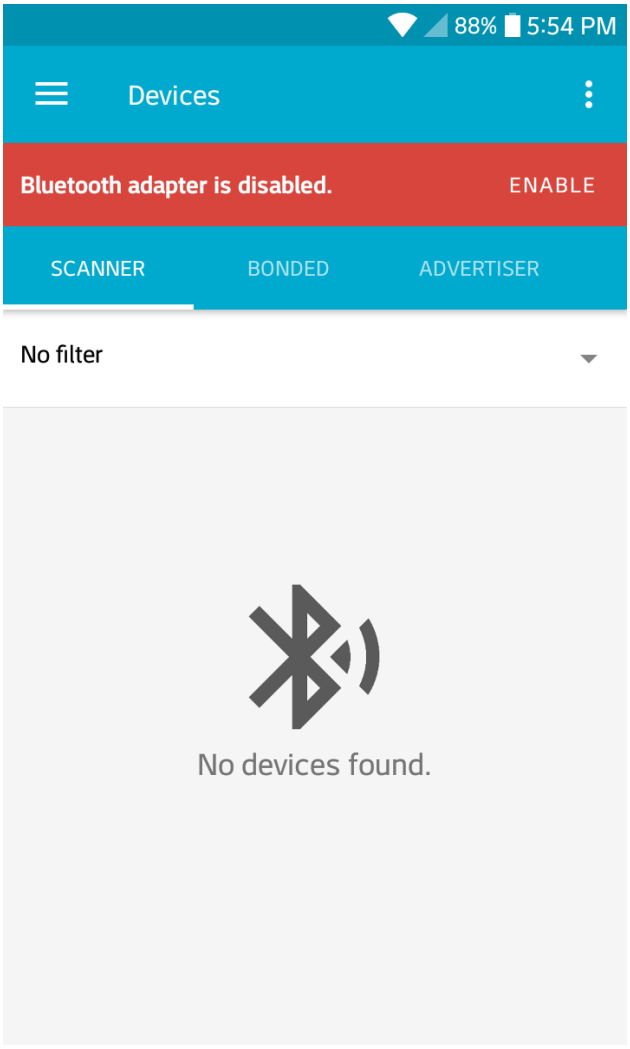


Figure 21: Bluetooth Adapter Disabled.

Once enabled, click the *Scan* button on the top right. After a moment, the HM-10 device should show up with its name.

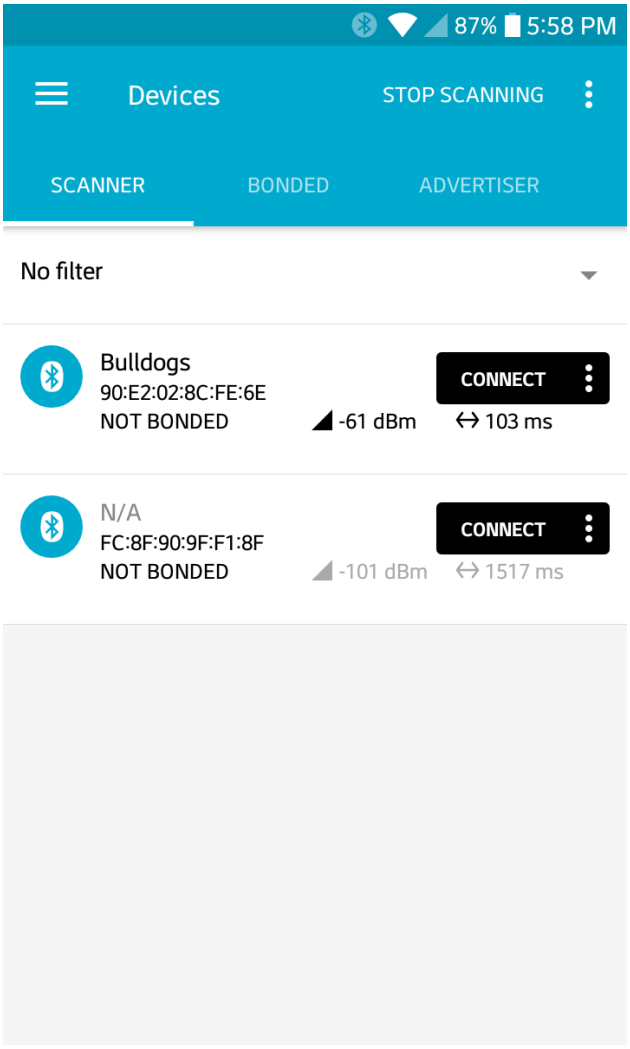


Figure 22: Scanning for BLE Devices.

Click on the *Connect* button on the HM-10 device. Doing this will open a new tab with the name of the module. In this tab there are three main services, two generic services and one *Environmental Sensing*. Click on the Environmental Sensing service. If the UUIDs were set up correctly, the values used in the configuration were for the Bluetooth Environment Sensing Service (ESS) and Temperature Characteristic within the ESS.

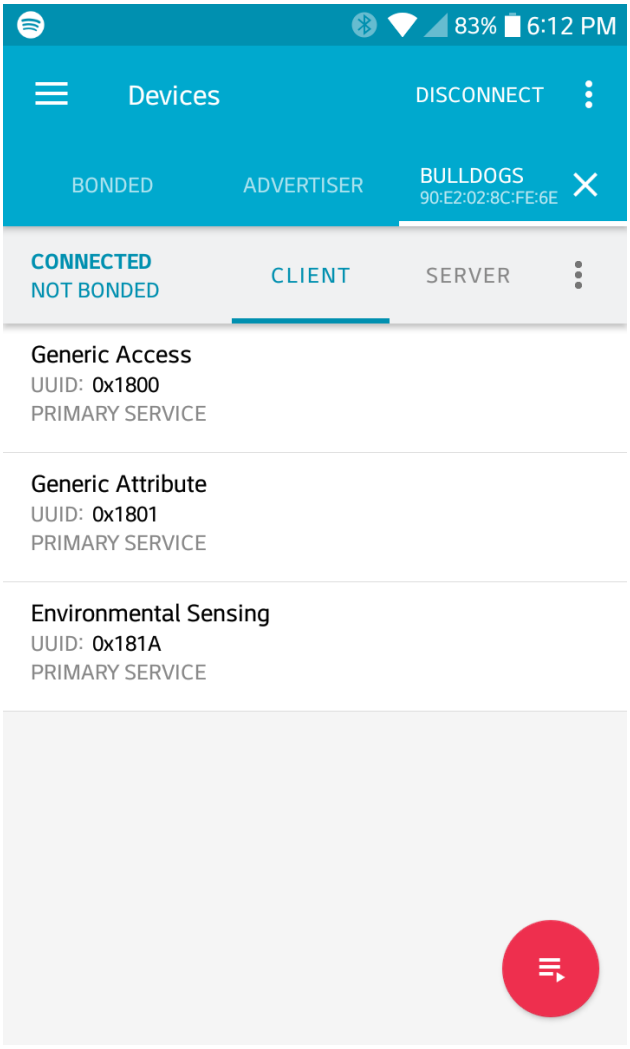


Figure 23: HM-10 Services.

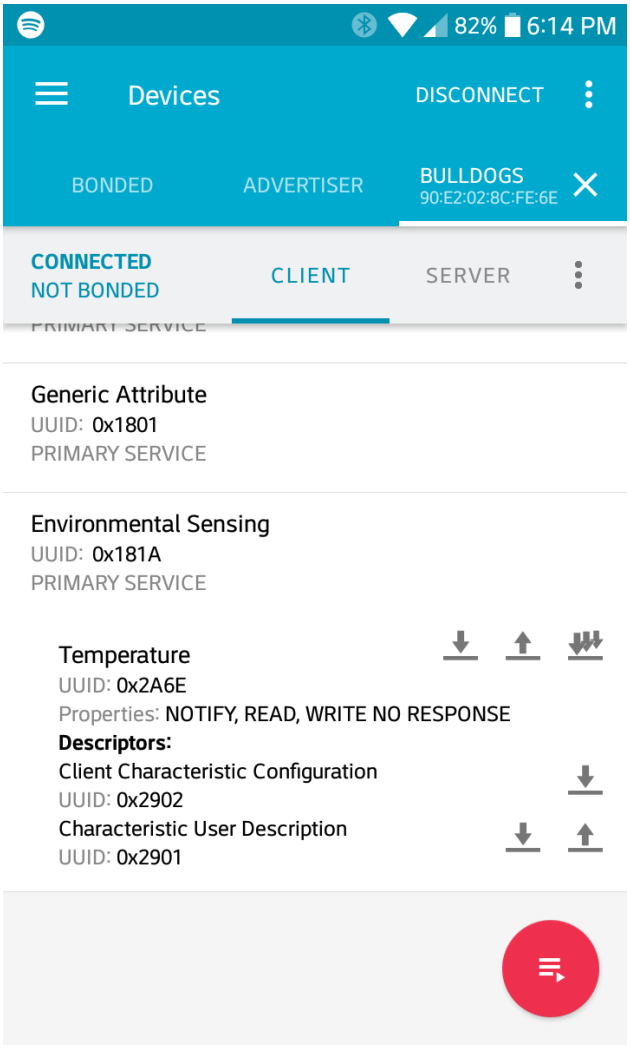


Figure 24: HM-10 ESS Characteristics.

However, there is no data for the temperature reading yet. We don't have an actual temperature sensor connected to the BLE module, so we'll have to fake it for now.

The temperature value can be transmitted from Realterm in the form of raw hexadecimal values. Bluetooth gives specifications for formatting data for all characteristics. The ESS Specification for the Temperature Characteristic allows a degree resolution down to 0.01°C. Also, the data type of the value is an `sint16`, or 16-bit signed-integer. To fix the decimal point problem that would be lost in using a integer the temperature value needs to be multiplied by 10^2 before being converted into signed-integer. So a temperature reading of 34.56°C would be 3456 and 0x0D80 in hexadecimal. Also, signed-integers used Two's Compliment to represent negative values. Thus, a temperature reading of -10.35°C would be multiplied by 10^2 and converted to Two's Compliment to get a result of `0xFBF5`.

So lets use the temperature value of 34.56°C to send to the Android device connected to the HM-10. In Realterm we can send these values in their signed-integer, hexadecimal form

with a couple things to consider. During the UART transmission, the Endianness of the bytes of the numbers are reversed, so the byte order will need to be fixed manually. Also in Realterm, each byte has to be expressed individually with a prefix, so 0x0D80 would need to be sent as 0x0D and 0x80. And if we take into consideration the byte reversal then 0x0D80 becomes 0x80 and 0x0D.

So with this in mind, go to Realterm and in the *Send* tab, type **"0x80 0x0D"** in the input box. Then instead of clicking *Send ASCII*, click *Send Numbers* instead to send the raw hexadecimal values.

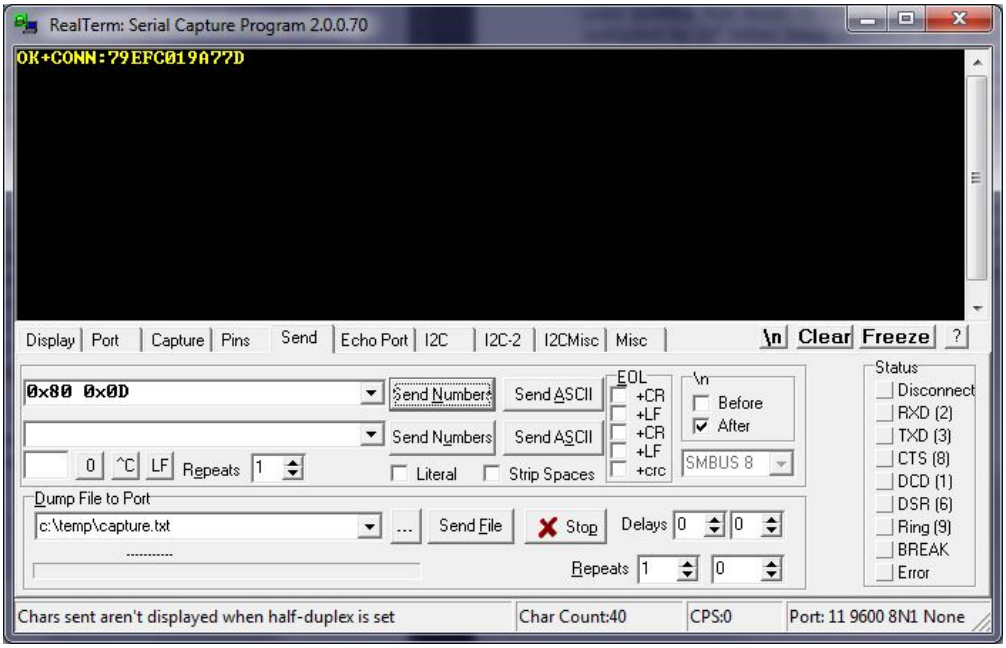


Figure 25: Sending Temperature Data.

Now checking the Andriod device, it can be seen that a *Value* line appears with the temperature reading of 34.56°C:

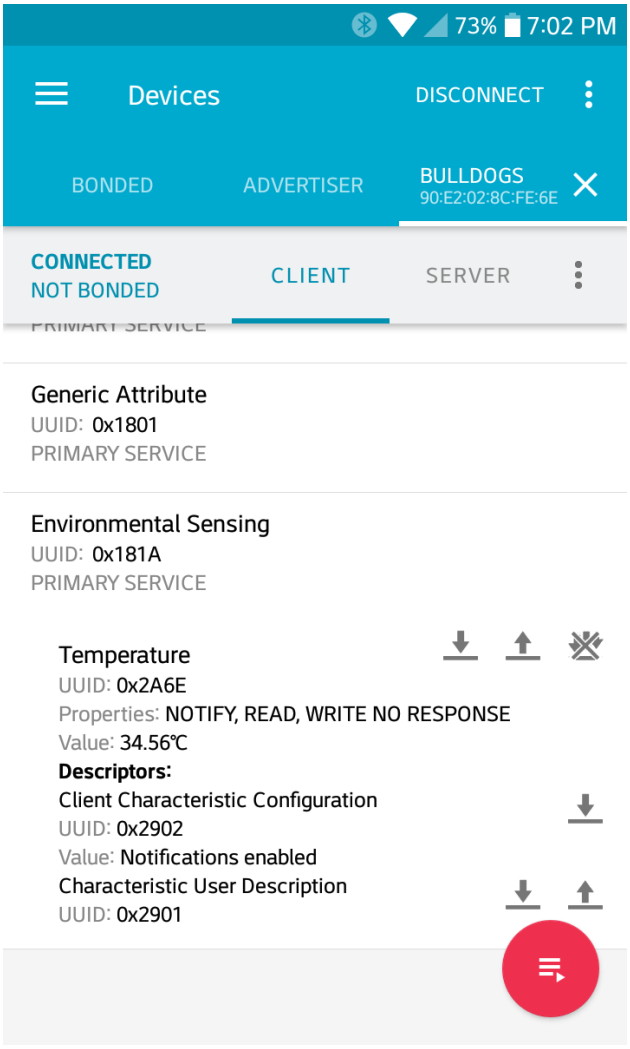


Figure 26: Temperature Reading in nRF Connect.

4 Wireless Gateway Implementation

4.1 Gateway Operating System and Environment

Using the RPi required a different environment than the default Contiki OS. The Linux distribution, Ubuntu-Mate was selected since it was compatible with the Contiki system, and the architecture of the RPi. The operating system allowed for a desktop environment which was user friendly, while also having the option to removing the Graphic User Interface (GUI), for an even lighter system to preserve resources.

The Operating system was saved as a custom operating system so that the installation does not need to be performed for each of the gateways. The following steps were taken to properly install a fresh operating system and environment for the gateway.

4.2 Required hardware:

- Raspberry Pi 3 B or Raspberry Pi 3 B+ (minor testing shows that a 2B model may work)
- Micro SD card (At least 8 gigabytes, 16+ gigabytes recommended)
- Separate computer with the ability to write to a Micro SD card
- Mouse, keyboard, and display for RPi

4.3 Installation

1. Download the Ubuntu Mate operating system image for the Raspberry pi aarch32 architecture (NOT aarch64) from `https://ubuntu-mate.org/download/`.
2. After downloading the image, the program "Rufus" can be downloaded and used to burn the image to a micro SD card. Putting the micro SD card into the pi and powering it on begins the installation process.
3. Once the OS was installed, the installation of Contiki can be installed following the guide at: `http://anrg.usc.edu/contiki/index.php/Installation`

   While installing the packages for the OS, one of them was not found therefore the others were not installed. Following the error provided and the command should be:

   ```
   sudo apt-get install build-essential binutils-msp430 gcc-msp430
      msp430-libc msp430mcu mspdebug gcc-arm-none-eabi openjdk-8-jdk
      openjdk-8-jre ant libncurses5-dev
   ```

4. Once completed, the hello-world program was implemented to test for other errors: `http://anrg.usc.edu/contiki/index.php/Hello_World`

   There should not be errors during the first half, though the second half will give errors. If there is a permission issue, put ***sudo*** in front of the code.

   Instead of following the instructions provided about putting in a new "serialdump-linux", do the following commands using the terminal.

   (a) Navigate to the following directory *contiki/tools/sky* using: `cd contiki/tools/sky`

   (b) Delete the file *serialdump-linux* using: `sudo rm serialdump-linux`

   (c) Rebuild the file using: `make serialdump`

   (d) Check to see if the file created is named *serialdump-linux* using: `ls`

   (e) If it's not, then rename it to that by using: `mv serialdump serialdump-linux`

   (f) Change the privilege of the file using: `sudo chmod 775 serialdump-linux`

   (g) Use the command: `sed -ie 's/\(ifdef O_\)SYNC/\1DIRECT/' serialdump.c`

   (h) Navigate to the following directory *contiki/examples/hello-world* using: `cd && cd contiki/examples/hello-world`

   (i) Retry running the program using: `make TARGET=sky hello-world.upload login`

5. The program should now run properly off of the Sky Mote, as well as the other programs.

4.4 Contiki Program

The Contiki program created is saved and ran on all of the motes including the gateway mote. This program was based off of Contiki's broadcast example, though was altered to collect data from the sensors.

The program example-broadcast.c can be found in Appendix A, which can replace the file located in the directory: contiki/examples/rime/example-broadcast.c. Once the file is in the proper location, the following steps should be completed to save the file to each of the motes, using a computer with Contiki and its new program installed, or the gateway.

1. Plug the mote into the USB port.
2. Open up the terminal (Ctrl+Alt+T).
3. Navigate to the directory where the file was saved using the command: `cd contiki/examples/rime`
4. Save file to the mote: `sudo make TARGET=sky example-broadcast.upload`

5. Once the process is completed and no errors have appeared, the mote can be disconnected and these steps can be repeated for all of the motes in the network.

6. The last mote that needs the program should be the gateway mote (the one that stays connected to the RPi gateway). To test the mote network and view the data in the present terminal, the following command can be used: `sudo make TARGET=sky example-broadcast.upload login`

This programs main functionality was to establish and test communication between the mote network and a web server. More complex programs, implementing more advanced protocols can be constructed based off of this example.

4.5 Gateway Program

The purpose of the gateway program is to automatically make the connection from the gateway to the web server, and run the Contiki program on the mote. Since the motes lack the capability to store their own collected data, the RPi must read what the mote is collecting, writing it to its own location, while simultaneously reading from the location to send the web server. The python program written can be viewed in Appendix B. The file location does not matter, though to keep things simple, should be saved to the desktop. The desktop is where the motes data will be written in a file created by the program.

Inside of the program, there is a comment stating where the database information needs to go. For security reasons, the database used to develop this project was not included, and replaced with "x". There are two ways about this, the web server can be created on the users localhost, or a website and hosting can be purchased and configured.

4.6 Web Server

While mote networks have the opportunity to be implemented in remote areas, being able to view data in these remote areas are not ideal. A web server was created to view the data collected at the gateway, so it can be viewed in another location. Data was collected and transmitted from the gateway, then sent and organized within a web server's database. The database was designed with the structure shown in Figure 27.

#	Name	Type
1	id	double
2	date	datetime
3	temperature	double
4	humidity	double
5	light	double

Figure 27: Database Structure.

The *id* column is where the motes address identification is stored, each mote has its own individual ID which it can be identified inside of a network. The next column is *data*, which contains the year, month, day, hour, minute, and second that the gateway received its data from a mote. The last three columns are *Temperature*, *Humidity*, and *Light*, which is the sensor data that the motes have collected. Although the data may not vary over time depending on how often the motes transmit, the date will always be incremented in time, allowing each post to the database to be unique and avoid errors.

An important feature the web server needed was that it was dynamic, to allow for motes to be added or removed to the network without needing to change any code. When viewing the web server, the data shown is from the most recent updated mote. Its mote ID is displayed at the top of the page, along with its designated gateway ID (which was a static address given). The chart displays the sensor data stored in the database from that mote ID. A snapshot of the initial home page is shown in Figure 7 above.

Towards the bottom of the page there is a drop down list, which asks the user to "Select a Mote". Selecting the drop down list displays all of the motes address IDs in the database as shown in Figure 28.

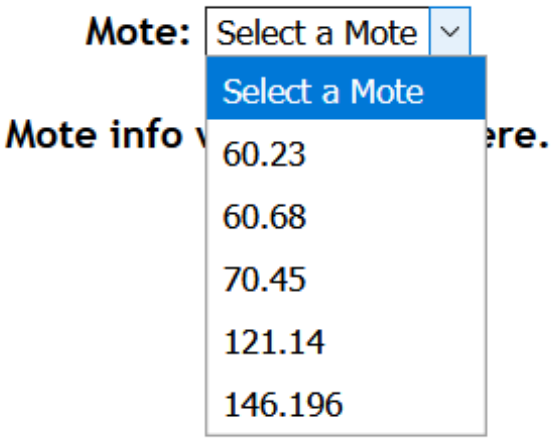


Figure 28: Drop down list of motes within the network.

Once a mote is selected from the list, the chart is updated to display its sensor data, as well as a list of all entries being displayed at the bottom as shown in Figure 29.

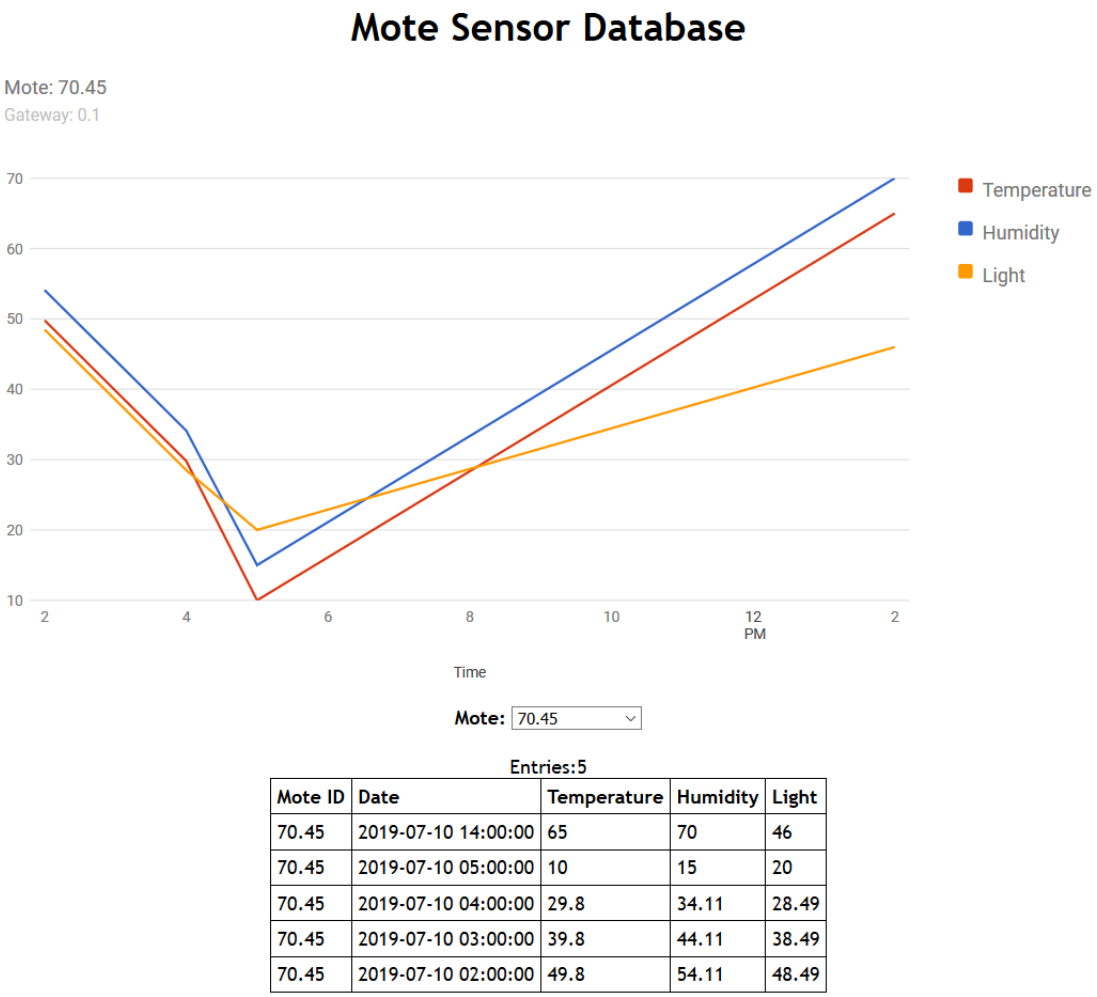


Figure 29: Site displaying selected mote's data.

5 One-Hop, Ad-Hoc Sensor Network with Cooja

5.1 Overview

In some instances when routing data in a Mobile Ad-Hoc Network (MANET), all relevant peripheral nodes are a single hop away from the gateway device. In such instances, algorithms like Ad-Hoc, On-Demand Distance Vector (AODV) and Dynamic Source Routing (DSR) can be more complex and computationally intensive than necessary.

Data routing in a single-hop network is far too simple to require such algorithms, so this single-hop sensor network was developed as a simple solution. This program utilized the *broadcast* and *unicast* features of the *Contiki OS* on the *Tmote Sky*.

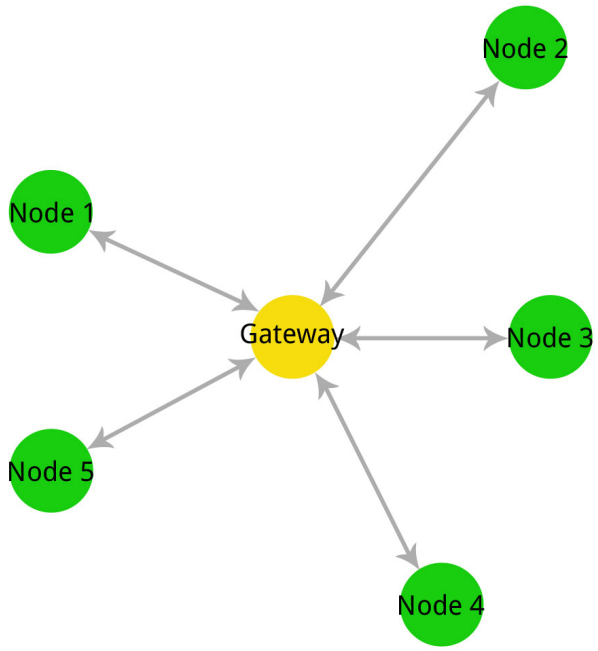


Figure 30: Example of a One-Hop, Ad-Hoc Sensor Network.

5.2 Peripheral Node Gateway Detection

Each peripheral sensor node needs to be able to transmit its sensor reading upstream to a gateway device. This allows the network to publish its data to some useful end whether that be on the internet, server or other device to read and analyze the data. In this case, since all the sensor nodes are within range of the gateway they can communicate directly.

However the devices doesn't know which of their neighbors is the gateway. So to determine where to transmit its data, each node will broadcast a message identifying itself as a *leaf* node (or peripheral node). The broadcast message will be assigned a type of

`BROADCAST_TYPE_LEAF_TO_PARENT`. This will help other nodes identify the type of broadcast messages being sent out.

All other leaf nodes will ignore these types of messages, since they are specifically for the *parent* node. When the parent or gateway node receives this message however, it will keep track of address of the node and unicast an acknowledgement (ACK) back to the node that sent the parent request. This message is then received by the leaf node and the parent's address is recorded. Then the leaf node can freely transmit data to the parent as frequently as it needs to.

5.3 Simulation

Cooja was used to simulate this algorithm after it was implemented using Contiki OS. Figure 31 shows the node placement in the simulation. Nodes 1, 2 and 3 are the leaf nodes (green) and node 4 is the gateway node (yellow).

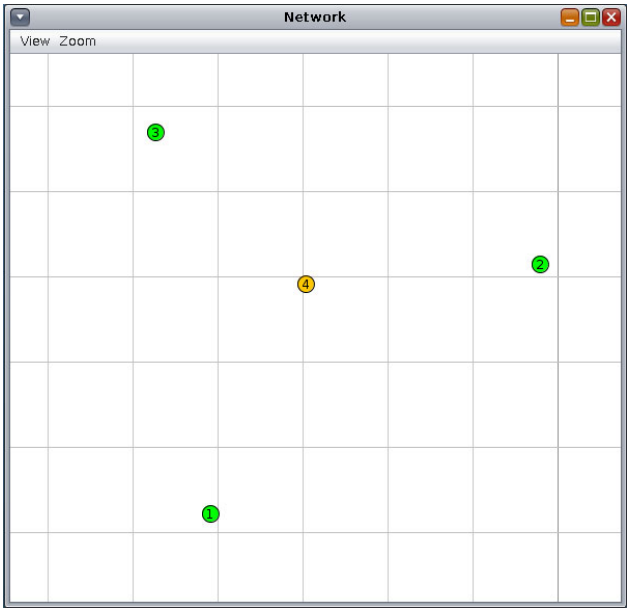


Figure 31: Node Placement in One-Hop Algorithm Simulation.

Once the simulation begins, a message log displays the activity of messages of each node and the *Network* window begins showing messages being sent and received by each node (Figure 32). The message log is shown in Figure 33.

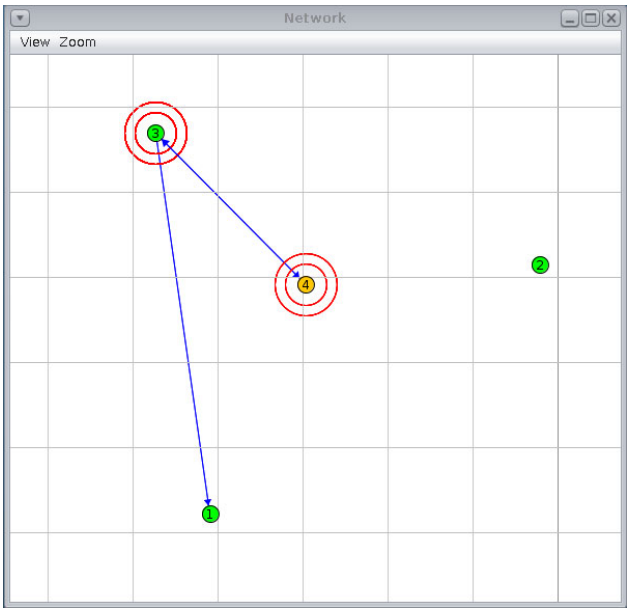


Figure 32: Node Communication in One-Hop Algorithm Simulation.

Time	Mote	Message
00:00.5	ID:2	Rime started with address 2.0
00:00.5	ID:2	MAC 02:00:00:00:00:00:00:00 Contiki 3.0 started. Node id is set to 2.
00:00.5	ID:2	nullsec CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
00:00.5	ID:2	Starting 'Broadcast process' 'Unicast process'
00:00.6	ID:4	Rime started with address 4.0
00:00.6	ID:4	MAC 04:00:00:00:00:00:00:00 Contiki 3.0 started. Node id is set to 4.
00:00.6	ID:4	nullsec CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
00:00.6	ID:4	Starting 'Broadcast process' 'Unicast process'
00:00.6	ID:1	Rime started with address 1.0
00:00.7	ID:1	MAC 01:00:00:00:00:00:00:00 Contiki 3.0 started. Node id is set to 1.
00:00.7	ID:1	nullsec CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
00:00.7	ID:1	Starting 'Broadcast process' 'Unicast process'
00:01.2	ID:3	Rime started with address 3.0
00:01.2	ID:3	MAC 03:00:00:00:00:00:00:00 Contiki 3.0 started. Node id is set to 3.
00:01.2	ID:3	nullsec CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
00:01.2	ID:3	Starting 'Broadcast process' 'Unicast process'
00:24.2	ID:3	Sending broadcast for parent
00:24.3	ID:4	broadcast message received from 3.0 with seqno 0, RSSI 65484, LQI 37, avg seqno gap 1.00
00:24.6	ID:3	unicast parent ACK received from 4.0
00:24.6	ID:3	Parent 4.0 added to list
00:26.7	ID:2	Sending broadcast for parent
00:26.8	ID:4	broadcast message received from 2.0 with seqno 0, RSSI 65480, LQI 37, avg seqno gap 1.00
00:27.0	ID:2	unicast parent ACK received from 4.0
00:27.0	ID:2	Parent 4.0 added to list
00:28.6	ID:3	sending unicast to 4.0
00:28.7	ID:4	unicast ping received from 3.0
00:28.8	ID:3	unicast parent ACK received from 4.0
00:30.0	ID:1	Sending broadcast for parent
00:30.0	ID:4	broadcast message received from 1.0 with seqno 0, RSSI 65477, LQI 37, avg seqno gap 1.00
00:30.3	ID:1	unicast parent ACK received from 4.0
00:30.3	ID:1	Parent 4.0 added to list
00:41.7	ID:2	sending unicast to 4.0
00:41.7	ID:4	unicast ping received from 2.0
00:41.9	ID:2	unicast parent ACK received from 4.0
00:43.6	ID:1	sending unicast to 4.0
00:43.7	ID:4	unicast ping received from 1.0
00:43.8	ID:1	unicast parent ACK received from 4.0
00:53.5	ID:3	sending unicast to 4.0
00:53.6	ID:4	unicast ping received from 3.0
00:53.7	ID:3	unicast parent ACK received from 4.0

Figure 33: Message Log of One-Hop Algorithm Simulation.

5.4 Simulation Results

Referring to Figure 33, whenever a node broadcasted a message for a parent, the parent node would send an ACK in response. Broadcast parent requests can be seen at times 00:24.2 (node 3), 00:26.7 (node 2) and 00:30.0 (node 1). Each of these messages were acknowledged at times 00:24.6 (node 3), 00:27.0 (node 2) and 00:30.3 (node 1). Then after each ACK was received, the leaf nodes stored the address of the parent, which in the simulation is 4.0. After each node logs the parent's address, it waits a random period of time to send a unicast message back to the parent (at times 00:28.6, 00:41.7 and 00:43.6). These message are also

acknowledged by the parent at times 00:28.8, 00:41.9 and 00:53.7.

These results show that the leaf and parent nodes successfully demonstrate this algorithm. The leaf nodes always start by attempting to find the parent through broadcasting requests and successfully ignore other parent requests from other nodes. The parent nodes correctly identifies these requests and acknowledges them, allowing leaf nodes to continuously send data (in the case of the simulation, the leaf only sent *PINGs* to the parent). In turn each ping is acknowledged by the parent.

References

- B. Krishnamachari, "Autonomous Networks Research Group", *Anrg.usc.edu*, 2016. [Online]. Available: https://anrg.usc.edu/www/. [Accessed: 18- Aug- 2019].
- JNHuaMao Technology Company, "HM Bluetooth module datasheet," Bluetooth 4.0 BLE module Datasheet, Mar. 2015.
- "GATT Services", *Bluetooth.com*, 2019. [Online]. Available: https://www.bluetooth.com/specifications/gatt/services/.
- M. Currey, "HM-10 Bluetooth 4 BLE Modules", *Martyncurrey.com*, 2017. [Online]. Available: http://www.martyncurrey.com/hm-10-bluetooth-4ble-modules/.
- Micrium Embedded Software, *https://www.micrium.com/wp-content/uploads/2014/03/wireless-sensor-network.png.* 2019.

A Appendix

A.1 Wireless Gateway, Contiki: example-broadcast.c

```
/*
 * Copyright (c) 2007, Swedish Institute of Computer Science.
 * All rights reserved.
 *
 * 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.
 *
 */

/**
 * \file
 *         Testing the broadcast layer in Rime
 * \author
 *         Adam Dunkels <adam@sics.se>
 */


#include <stdlib.h>
#include <stdio.h>
#include "dev/light-sensor.h"
#include "dev/sht11/sht11-sensor.h"
#include <math.h>
#include "contiki.h"
#include "net/rime/rime.h"
#include "random.h"
#include "dev/button-sensor.h"
#include "dev/leds.h"
#include "cfs/cfs.h"

char message [90]; // Global message, going from sensor to transmit
int timer_duration = 20; // Seconds

// Header is located in contiki/core/sys/process.h
/*---------------------------------------------------------------------------*/
PROCESS(example_broadcast_process, "Broadcast example");
PROCESS(sensor_acq_process,"Sensor Acquisition"); // Adding sensor Process
AUTOSTART_PROCESSES(&example_broadcast_process, &sensor_acq_process); // Adding sensor Process
/*---------------------------------------------------------------------------*/
```

```
static void
broadcast_recv(struct broadcast_conn *c, const linkaddr_t *from)
{

// using the terminal command "sudo make login > filename.txt" to write output to file
  printf("%d.%d %s\n", from->u8[0], from->u8[1], (char *)packetbuf_dataptr());

}
static const struct broadcast_callbacks broadcast_call = {broadcast_recv};
static struct broadcast_conn broadcast;
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(example_broadcast_process, ev, data)
{
  static struct etimer et;

  PROCESS_EXITHANDLER(broadcast_close(&broadcast);)

  PROCESS_BEGIN();

  broadcast_open(&broadcast, 129, &broadcast_call);

  while(1) {

    /* Delay 2-4 seconds */
    //etimer_set(&et, CLOCK_SECOND * 4 + random_rand() % (CLOCK_SECOND * 4));
    etimer_set(&et, CLOCK_SECOND * timer_duration);

    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

    packetbuf_copyfrom(message, 91); // making size one more than message
    broadcast_send(&broadcast);
  }

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

PROCESS_THREAD(sensor_acq_process,ev,data)
{
      static struct etimer et;
      static int val;
      static float s = 0;
      static int dec;
      static float frac;

    // Placeholders to gather data before going into message
      char message_Temp [30];
      char message_Hum [30];
      char message_Light [30];

      PROCESS_BEGIN();

      printf("Starting Sensor Example.\n");

      while(1)
      {
       etimer_set(&et, CLOCK_SECOND * 2);
       SENSORS_ACTIVATE(light_sensor);
           SENSORS_ACTIVATE(sht11_sensor);

       PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

       // Temperature
           val = sht11_sensor.value(SHT11_SENSOR_TEMP);
       if(val != -1)
           {
        // s= ((0.01*val) - 39.60); // This one is for Celsius
        s= (((0.01*val) - 39.60)*1.8)+32; // This one is for Fahrenheit
```

```
            dec = s;
            frac = s - dec;

         // Stores the data in the placeholder
         sprintf(message_Temp,"%d.%02u", dec, (unsigned int)(frac * 100));

            }

        // Humidity
        val=sht11_sensor.value(SHT11_SENSOR_HUMIDITY);
        if(val != -1)
            {
         s= (((0.0405*val) - 4) + ((-2.8 * 0.000001)*(pow(val,2))));
            dec = s;
            frac = s - dec;

         // Stores the data in the placeholder
         sprintf(message_Hum,"%d.%02u", dec, (unsigned int)(frac * 100));

            }

        // Light
            val = light_sensor.value(LIGHT_SENSOR_TOTAL_SOLAR);
            if(val != -1)
            {
             s = (float)(val * 0.4071);
             dec = s;
             frac = s - dec;
         // Stores the data in the placeholder
         sprintf(message_Light,"%d.%02u", dec, (unsigned int)(frac * 100));

            }

         // Concatenates the placeholders into the message
        sprintf(message, "%s %s %s", message_Temp, message_Hum, message_Light);

        etimer_reset(&et);
            SENSORS_DEACTIVATE(light_sensor);
            SENSORS_DEACTIVATE(sht11_sensor);


      } //end of while

      PROCESS_END();
}
```

A.2 Wireless Gateway: gatewayProg.py

```
# Implenting the gateway programs in a single program using threading

import threading
import time
import os
import mysql.connector
import datetime

def firstProg():
    # Changes to proper directory
    os.chdir("contiki/examples/rime")

    # Normal command to run program, though saving the output to a text file
    os.system("sudo make TARGET=sky example-broadcast.upload login > /home/pi/Desktop/Gatewaydata.txt")

def secondProg():
    # Need to wait 30 seconds for firstProg to get set up
    time.sleep(30)
    # Needed to change directory where text file is
    #os.system("pwd") # Use for debugging
    os.chdir("/home/pi/Desktop/")

    # Connect to Database
    # Fill in database information below
    mydb = mysql.connector.connect(
        host ="xxxx",
        user="xxxx",
        passwd="xxxx",
        database="xxxx"
    )

    mycursor = mydb.cursor()


    while(1):
        if (os.path.getsize("Gatewaydata.txt") > 0):
            x = datetime.datetime.now() # Get the time
            with open("Gatewaydata.txt", "r+") as file:

                for line in file:
                    if line.strip():
                        # print line
                        split_line = line.split()
                        if len(split_line) == 4:
                            mote_id, temp, hum, light = line.split(" ", 4)
                            print ("id: " + str(mote_id) + ", Temp=: " + str(temp) + ", Hum: " + str(hum) \\
                                + ",Light: " + str(light))
                file.truncate(0) # clears file
            time.sleep(3)
            # sql= "DELETE FROM pi WHERE id = %s"
            # val = (mote_id,)
            # mycursor.execute(sql, val)
            sql2 = "INSERT INTO pi (id, date, temperature, humidity, light) VALUES (%s, %s, %s, %s, %s)"
            val2 = (mote_id, x.strftime("%Y-%m-%d %H:%M:%S"), temp, hum, light)
            mycursor.execute(sql2, val2)
            mydb.commit()
            print(mycursor.rowcount, "record inserted...")
            print(x.strftime("%Y-%m-%d %H:%M:%S"))


t1=threading.Thread(target = firstProg, name = 'thread1')
t2=threading.Thread(target = secondProg, name = 'thread2')

t1.start()
t2.start()
```

A.3 Wireless Gateway: index.php

```
<?php
    session_start();
    ?>
<!DOCTYPE HTML>
<html>
<head>
   <title> Mote Sensor Database</title>
   <center><h1>Mote Sensor Database</h1></center>
   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
</head>
<body>

<!--PHP code section-->
<?php

    require 'databaseaccess.php';
    include 'getid.php';
    $entries3;
    $v_temp3;


    # For the initial Graph
    $sql_init = "SELECT id FROM pi ORDER BY date DESC LIMIT 1";
    $result_init = $mysqli->query($sql_init);
    $row_init = $result_init->fetch_assoc(); // Just need to get one

    $sql_init2 = "SELECT * FROM pi WHERE id = '".$row_init["id"] ."'";

    $result_init2 = $mysqli->query($sql_init2);

    $entries_init = $result_init2->num_rows;

    $date_init = $temp_init = $hum_init = $light_init = array();
    if ($entries_init > 0) {
        // output data of each row
        while($row = $result_init2->fetch_assoc()) {
            array_push($date_init, $row["date"]);
            array_push($temp_init, $row["temperature"]);
            array_push($hum_init, $row["humidity"]);
            array_push($light_init, $row["light"]);

        }

    }
    else{
        echo "<br>NO RESULTS<br>";
    }



    # Gathers the mote IDs for the dropdown list
    $sql = "SELECT * FROM pi";
    $result = $mysqli->query($sql);

    $v_date2 = $v_temp2 = $v_hum2 = $v_light2 = $v_id = array();
    $entries = $result->num_rows;

    if ($entries > 0) {
        // output data of each row
        while($row = $result->fetch_assoc()) {
            if(!in_array($row["id"], $v_id))
                {array_push($v_id, $row["id"]);}
        }

    }
```

```
    else {
        echo "<br>0 results<br>";
    }



?>
<!---End PHP code section--->

    <style type="text/css">
        .table_titles, .table_cells_odd, .table_cells_even {
                padding-right: 20px;
                padding-left: 20px;
                color: #000;
        }
        .table_titles {
            color: #FFF;
            background-color: #666;
        }
        .table_cells_odd {
            background-color: #CCC;
        }
        .table_cells_even {
            background-color: #FAFAFA;
        }
        table {
            border: 2px solid #333;
        }
        body { font-family: "Trebuchet MS", Arial; }
    </style>

    <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
    <script type="text/javascript">
      google.charts.load('current', {'packages':['line']});
      google.charts.setOnLoadCallback(load_data);


      // Moving array data from PHP to JS
        var chart;

        function load_data(id){
            if(id == null){
              $.ajax({
                method:"POST",
                dataType:"text",
                success:function()
                {
                    drawChartInitial();
                }
                });
            }
            else {
                $.ajax({
                    url:"fetch.php",
                    method:"POST",
                    data:{id:id},
                    dataType:"json",
                    success:function(data)
                    {
                        drawChart(data);
                    }
                });
            }
        }

        function drawChart(chart_data){
```

```
            var jsonData = chart_data ;
            var data = new google.visualization.DataTable();
                data.addColumn('datetime', 'Time');
                data.addColumn('number', 'Temperature');
                data.addColumn('number', 'Humidity');
                data.addColumn('number', 'Light');

                var Mote_string ='Mote: '; // add id when working
                var options = {
                chart: {
                  title: 'Gateway: 0.1'
                  ,subtitle: Mote_string},
                width: 900,
                height: 500
              };

                var limit = jsonData[0].length;
                var counter =0;
                while (counter < limit){
          data.addRows([
            [new Date(jsonData[0][counter]), Number(jsonData[1][counter]), Number(jsonData[2][counter]), Number(jsonData[3][cour

            counter = counter +1;
            }



                var chart = new google.charts.Line(document.getElementById('chart_loc'));

                chart.draw(data, google.charts.Line.convertOptions(options));
        }

    function drawChartInitial(){
        var counter = 0;
              // Initial values for chart
      var limit = <?php echo $entries_init; ?>;
      var temp = <?php echo json_encode($temp_init); ?>;
      var hum =  <?php echo json_encode($hum_init); ?>;
      var light = <?php echo json_encode($light_init); ?>;
      var Entry_Dates = <?php echo json_encode($date_init); ?>;

        var data = new google.visualization.DataTable();
            data.addColumn('datetime', 'Time');
            data.addColumn('number', 'Temperature');
            data.addColumn('number', 'Humidity');
            data.addColumn('number', 'Light');
            var Mote_string ='Mote: '+ <?php echo $row_init["id"]; ?>;
            var options = {
            chart: {
              title: 'Gateway: 0.1'
              ,subtitle: Mote_string
            },
            width: 900,
            height: 500
          };

        while (counter < limit){
          data.addRows([
            [new Date(Entry_Dates[counter]), Number(temp[counter]), Number(hum[counter]), Number(light[counter])]]);

            counter = counter +1;
            }
            var chart = new google.charts.Line(document.getElementById('chart_loc'));

            chart.draw(data, google.charts.Line.convertOptions(options));
    }

</script>
```

```
<script>

$(document).ready(function(){

    $('#id').change(function(){
        var id = $(this).val();
        if(id != '')
        {
            load_data(id);
        }
    });

});

</script>

<!--Div that will hold the Line chart-->
<div id="chart_loc"></div>

</center>

<!-----------Drop down list of Motes------------>
<!-- Code from w3-->
<script>
function showId(str) {
  if (str=="") {
    document.getElementById("txtHint").innerHTML="";
    return;
  }
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
  } else { // code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (this.readyState==4 && this.status==200) {
      document.getElementById("txtHint").innerHTML=this.responseText;
    }
  }


// Multiply str, DOUBLE->INT, before passing
  xmlhttp.open("GET","getid.php?q="+(str*1000),true);
  xmlhttp.send();

}
</script>
<!-- END Code from w3-->


<!-- Try to draw chart here -->
<center>
<h4>Mote: </h4>
<select id="select" onchange="showId(this.value); load_data(this.value)">
    <option value=0>Select a Mote</option>
</select>

<script>
    var select = document.getElementById("select"),
        id = <?php echo json_encode($v_id); ?>;

    for(var i =0; i<id.length; i++) {
        var option = document.createElement("OPTION"),
            txt = document.createTextNode(id[i]);
        option.appendChild(txt);
        option.setAttribute("value",id[i]);
        select.insertBefore(option,select.lastChild);
```

```
    }

</script>

<div id="txtHint"><b>Mote info will be listed here.</b></div>

<input type="button" value="Redraw Chart" onClick="load_data()"> </input>

</center>
<!--END Drop down list of Motes-->

</body>

</html>
```

A.4 Wireless Gateway, Web Server: fetch.php

```
<?php

//fetch.php

include('databaseaccess.php');


if(isset($_POST["id"])){

 $query = "SELECT * FROM pi WHERE id = '".$_POST["id"]."' ";
 // ORDER BY date DESC

$statement = $mysqli->query($query);

$date = $temp = $hum = $light = array();

while($row = $statement->fetch_assoc()) {
    array_push($date, $row["date"]);
    array_push($temp, $row["temperature"]);
    array_push($hum, $row["humidity"]);
    array_push($light, $row["light"]);
}

 $output = array($date, $temp, $hum, $light);


}
 echo json_encode($output);

?>
```

A.5 Wireless Gateway, Web Server: databaseaccess.php

```
<?php
    # Insert database information below
    $host = "localhost";
    $username = "xxxx";
    $user_pass = "xxxx";
    $database_in_use = "xxxx";

    $mysqli = mysqli_connect($host, $username, $user_pass, $database_in_use);
    if ($mysqli->connect_errno) {
        echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
    }

?>
```

A.6 Wireless Gateway, Web Server: getid.php

```
<?php
    session_start();
    ?>
<!DOCTYPE html>
<html>
<head>
<style>
table {
    width: 900;
    border-collapse: collapse;
}

table, td, th {
    border: 1px solid black;
    padding: 5px;
}

th {text-align: left;}
</style>
</head>
<body>
<center>
<?php

require 'databaseaccess.php'; // Connects to database

//Divide $q, INT->DOUBLE, to get the correct mote ID
$q = intval($_GET['q'])/1000;

if($q != 0){

    # The name of the columns and the table theyre from
    $sql = "SELECT * FROM pi WHERE id = '".$q."'";

    $result2 = $mysqli->query($sql);

    $entries2 = $result2->num_rows;
    echo "Entries:" . $entries2 . "<br>";


    echo "<table>
    <tr>
    <th>Mote ID</th>
    <th>Date</th>
    <th>Temperature</th>
    <th>Humidity</th>
    <th>Light</th>
    </tr>";


    if ($entries2 > 0) {
        // output data of each row
        while($row = $result2->fetch_assoc()) {
            echo "<tr>";
            echo "<td>" . $row['id'] . "</td>";
            echo "<td>" . $row['date'] . "</td>";
            echo "<td>" . $row['temperature'] . "</td>";
            echo "<td>" . $row['humidity'] . "</td>";
            echo "<td>" . $row['light'] . "</td>";
            echo "</tr>";
            array_push($v_date2, $row["date"]);
            array_push($v_temp2, $row["temperature"]);
            array_push($v_hum2, $row["humidity"]);
            array_push($v_light2, $row["light"]);
        }
```

```
        echo "</table>";
        }

    $_SESSION['entries3'] = $entries2;
    $_SESSION['v_date3'] = $v_date2;
    $_SESSION['v_temp3'] = $v_temp2;
    $_SESSION['v_hum3'] = $v_hum2;
    $_SESSION['v_light3'] = $v_light2;

} // end of if statement

?>


</center>

</body>
</html>
```

A.7 One-Hop Algorithm: sensor-network-leaf.c

```
#include "contiki.h"
#include "lib/list.h"
#include "lib/memb.h"
#include "lib/random.h"
#include "net/rime/rime.h"
#include "dev/light-sensor.h"
#include "dev/sht11/sht11-sensor.h"
#include <stdio.h>
#include <math.h>

// Struct used to hold the broadcast message
struct broadcast_message {
    uint8_t seqno;
    uint8_t type;
};

// Struct used to hold the type of unicast message
struct unicast_message {
    uint8_t type;
};

// defines the types of unicast messages
enum {
    UNICAST_TYPE_PING,
    UNICAST_TYPE_PONG,
    UNICAST_PARENT_ACK
};

enum {
    BROADCAST_TYPE_LEAF_TO_PARENT
};

struct neighbor {
    // next pointer needed for Contiki lists
    struct neighbor *next;
    // address of the neighbor
    linkaddr_t addr;
    // RSSI = Received Signal Strength Indicator
    // LQI = CC2420 Link Quality Indicator
    uint16_t last_rssi,last_lqi;
    // last sequence number we saw from this neighbor
    uint8_t last_seqno;
    // contains the average seqno gap from this neighbor
    uint32_t avg_seqno_gap;
};

struct parent {
    // next pointer needed for Contiki lists
    struct parent *next;
    // address of the neighbor
    linkaddr_t addr;
    // RSSI = Received Signal Strength Indicator
    // LQI = CC2420 Link Quality Indicator
    uint16_t last_rssi,last_lqi;
    // last sequence number we saw from this neighbor
    uint8_t last_seqno;
    // contains the average seqno gap from this neighbor
    uint32_t avg_seqno_gap;
};


#define MAX_PARENTS 16
MEMB(parents_memb, struct parent, MAX_PARENTS);
LIST(parents_list);
```

```
static struct broadcast_conn broadcast;
static struct unicast_conn unicast;

#define SEQNO_EWMA_UNITY 0X100
#define SEQNO_EWMA_ALPHA 0X040

PROCESS(broadcast_process,"Broadcast process");
PROCESS(unicast_process,"Unicast process");

AUTOSTART_PROCESSES(&broadcast_process,&unicast_process);

static void broadcast_recv(struct broadcast_conn *c, const linkaddr_t *from) {
    //neighbor *n;
    struct broadcast_message *m;
    //uint8_t seqno_gap;

    m=packetbuf_dataptr();
    // ignore leaf to parent broadcasts from other leaf nodes
    if (m->type == BROADCAST_TYPE_LEAF_TO_PARENT) { return; }
    else {;}
}

static const struct broadcast_callbacks broadcast_call = {broadcast_recv};

static void recv_uc(struct unicast_conn *c, const linkaddr_t *from) {
    struct unicast_message *msg;
    struct parent *p;
    msg = packetbuf_dataptr();

    if (msg->type == UNICAST_PARENT_ACK) {
        printf("unicast parent ACK received from %d.%d\n",
                from->u8[0],from->u8[1]);
        for (p=list_head(parents_list); p != NULL; p = list_item_next(p)) {
            if (linkaddr_cmp(&p->addr,from)) {
                return;
            }
        }
        if (p == NULL) {
            p = memb_alloc(&parents_memb);
            if (p == NULL) {
                return; // cant allocate new neighbor in memory so give up
            }
        }
        linkaddr_copy(&p->addr,from);

        list_add(parents_list,p);
        printf("Parent %d.%d added to list\n",from->u8[0],from->u8[1]);
    }
}

static const struct unicast_callbacks unicast_call = {recv_uc};

PROCESS_THREAD(broadcast_process, ev, data) {
    static struct etimer et;
    static uint8_t seqno;
    struct broadcast_message msg;

    PROCESS_EXITHANDLER(broadcast_close(&broadcast););

    PROCESS_BEGIN();

    broadcast_open(&broadcast,129,&broadcast_call);

    // only use broadcast to find this node's parent node
    while (1) {
        // send message every 1-2 seconds
        etimer_set(&et,CLOCK_SECOND*16 + random_rand() % (CLOCK_SECOND*16));
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
```

```
        if (list_length(parents_list) == 0) {
            msg.seqno = seqno;
            msg.type = BROADCAST_TYPE_LEAF_TO_PARENT;
            packetbuf_copyfrom(&msg,sizeof(struct broadcast_message));
            broadcast_send(&broadcast);
            printf("Sending broadcast for parent\n");
            seqno++;
        }
    }
    PROCESS_END();
}

PROCESS_THREAD(unicast_process,ev,data) {
    PROCESS_EXITHANDLER(unicast_close(&unicast);)
        PROCESS_BEGIN();
    unicast_open(&unicast,146,&unicast_call);

    while(1) {
        static struct etimer et;
        struct unicast_message msg;
        struct parent *p;

        etimer_set(&et,CLOCK_SECOND*16 + random_rand() % (CLOCK_SECOND*16));

        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

        if(list_length(parents_list) > 0) {
            p = list_head(parents_list);
            printf("sending unicast to %d.%d\n",p->addr.u8[0],p->addr.u8[1]);

            msg.type = UNICAST_TYPE_PING;
            packetbuf_copyfrom(&msg,sizeof(msg));
            unicast_send(&unicast,&p->addr);
        }
    }
    PROCESS_END();
}
```

A.8 One-Hop Algorithm: sensor-network-root.c

```
#include "contiki.h"
#include "lib/list.h"
#include "lib/memb.h"
#include "lib/random.h"
#include "net/rime/rime.h"

#include <stdio.h>

// Struct used to hold the broadcast message
struct broadcast_message {
    uint8_t seqno;
    uint8_t type;
};

// Struct used to hold the type of unicast message
struct unicast_message {
    uint8_t type;
};

// defines the types of unicast messages
enum {
    UNICAST_TYPE_PING,
    UNICAST_TYPE_PONG,
    UNICAST_TYPE_ACK
};

enum {
    BROADCAST_TYPE_LEAF_TO_PARENT
};

struct child {
    // next pointer needed for Contiki lists
    struct child *c;
    // address of the neighbor
    linkaddr_t addr;
    // RSSI = Received Signal Strength Indicator
    // LQI = CC2420 Link Quality Indicator
    uint16_t last_rssi,last_lqi;
    // last sequence number we saw from this neighbor
    uint8_t last_seqno;
    // contains the average seqno gap from this neighbor
    uint32_t avg_seqno_gap;
};

#define MAX_CHILDREN 16

MEMB(children_memb, struct child, MAX_CHILDREN);

LIST(children_list);

static struct broadcast_conn broadcast;
static struct unicast_conn unicast;

#define SEQNO_EWMA_UNITY 0X100
#define SEQNO_EWMA_ALPHA 0X040

PROCESS(broadcast_process,"Broadcast process");
PROCESS(unicast_process,"Unicast process");

AUTOSTART_PROCESSES(&broadcast_process,&unicast_process);

static void broadcast_recv(struct broadcast_conn *con, const linkaddr_t *from) {
    struct child *c;
    struct broadcast_message *m;
    uint8_t seqno_gap;
```

```
    m=packetbuf_dataptr();
    for (c=list_head(children_list); c != NULL; c = list_item_next(c)) {
        if (linkaddr_cmp(&c->addr,from)) {
            break;
        }
    }
    if (c == NULL) {
        c = memb_alloc(&children_memb);
        if (c == NULL) {
            return; // cant allocate new neighbor in memory so give up
        }
    }
    linkaddr_copy(&c->addr,from);
    c->last_seqno = m->seqno - 1;
    c->avg_seqno_gap = SEQNO_EWMA_UNITY;

    list_add(children_list,c);

    c->last_rssi = packetbuf_attr(PACKETBUF_ATTR_RSSI);
    c->last_lqi = packetbuf_attr(PACKETBUF_ATTR_LINK_QUALITY);

    seqno_gap = m->seqno - c->last_seqno;
    c->avg_seqno_gap = (((uint32_t)seqno_gap * SEQNO_EWMA_UNITY) *
            SEQNO_EWMA_ALPHA)/ SEQNO_EWMA_UNITY +
        ((uint32_t)c->avg_seqno_gap * (SEQNO_EWMA_UNITY -
            SEQNO_EWMA_ALPHA)) / SEQNO_EWMA_UNITY;

    c->last_seqno = m->seqno;
    printf("broadcast message received from %d.%d with seqno %d, RSSI %u, LQI %u, avg seqno gap %d.%02d\n",
            from->u8[0],from->u8[1],
            m->seqno,
            packetbuf_attr(PACKETBUF_ATTR_RSSI),
            packetbuf_attr(PACKETBUF_ATTR_LINK_QUALITY),
            (int)(c->avg_seqno_gap / SEQNO_EWMA_UNITY),
            (int)(((100UL * c->avg_seqno_gap)/SEQNO_EWMA_UNITY)%100));

    struct unicast_message *ack;
    ack = packetbuf_dataptr();
    ack->type = UNICAST_TYPE_ACK;
    packetbuf_copyfrom(ack,sizeof(struct unicast_message));
    unicast_send(&unicast,from);
}

static const struct broadcast_callbacks broadcast_call = {broadcast_recv};

static void recv_uc(struct unicast_conn *con, const linkaddr_t *from) {
    struct unicast_message *msg;

    msg = packetbuf_dataptr();

    if (msg->type == UNICAST_TYPE_PING) {
        printf("unicast ping received from %d.%d\n",
                from->u8[0],from->u8[1]);
        msg->type = UNICAST_TYPE_ACK;
        packetbuf_copyfrom(msg,sizeof(struct unicast_message));
        unicast_send(con,from);
    }
}

static const struct unicast_callbacks unicast_call = {recv_uc};

PROCESS_THREAD(broadcast_process,ev,data) {
    static struct etimer et;
    PROCESS_EXITHANDLER(broadcast_close(&broadcast);)

        PROCESS_BEGIN();
```

```
    broadcast_open(&broadcast,129,&broadcast_call);

    while(1) {
        etimer_set(&et,CLOCK_SECOND*8 + random_rand() % (CLOCK_SECOND*8));
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
    }
    PROCESS_END();
}

PROCESS_THREAD(unicast_process,ev,data) {
    static struct etimer et;
    //struct unicast_message msg;
    //struct child *c;
    PROCESS_EXITHANDLER(unicast_close(&unicast);)
        PROCESS_BEGIN();
    unicast_open(&unicast,146,&unicast_call);

    while(1) {
        etimer_set(&et,CLOCK_SECOND*8 + random_rand() % (CLOCK_SECOND*8));

        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
    }
    PROCESS_END();
}
```