LoRa Temperature and Humidity Network

Feb 13, 2018

Goal of project:

Simple, low cost long distance sensor network, Using 915mhz LoRa, Arduino UNO, Wemos D1 Mini, and LoRa32u4 boards.
Create a simple Dashboard using Crouton, and keep sensor data using Google Sheets and Pushingbox.

Components and Controllers Needed:

There are a number of parts to this project, For this demo we will be using the follow:

A) Two (2) Arduino UNO clone boards
B) Two (2) Linksprite 915mhz LoRa Shields for the UNO
    a) http://linksprite.com/wiki/index.php5?title=Lora_Radio_Shield
C) Two (2) DHT-11 Temperature and Humidity Sensors
    a) Used on the Arduino UNOs
    b) https://www.ebay.com/itm/New-DHT11-Temperature-And-Relative-Humidity-Sensor-Module-For-Arduino-Nice-Hot/173084343587?epid=1051342374&hash=item284ca17d23:g:lN0AAOSwWEZaTyYm
D) LoRa32u4 915Mhz Board
    a) https://www.ebay.com/itm/LoRa32u4-II-Lora-Development-Board-LoraWan-Atmega328-SX1276-HPD13-915MHZ/162634101174?hash=item25ddbf71b6:g:0-wAAOSwZ3BaTxRj
        i) * Needs a antenna *
        ii) https://www.ebay.com/itm/868MHz-915MHz-Zigbee-RFID-Antenna-SMA-Aerial-IPX-U-FL-Cable-for-Smart-Home/253271489966?hash=item3af8283dae:g:Oq0AAOSwtVxaKlmz
        iii) Antenna
    b) https://www.ebay.com/itm/868MHz-915MHz-LoRa32u4-Development-Board-Lora-SX1276-HPD13-Wifi-Module-Antenna/332427363771?hash=item4d663705bb:m:mtaCMGiILq_KjzfMwwO2oHA
        i) * Comes with Antenna *
E) Two Wemos D1 Mini (ESP8266 based) controllers
F) Two LoRa RFM95 915mhz Boards

a) https://www.ebay.com/itm/RFM95-RFM95W-SX1276-Wireless-Transceiver-Module-LoRaTM-Wireless-Transceiver-Hot/222763221610?hash=item33ddb9026a:m:mHuwBOW5i6WopL8gUpN05iQ
    i) These need to be soldered to a Shield and require SMD capacitors, and diodes.
    ii) These also require a antenna

G) Two LoRa Shield boards for the D1 Mini
    a) https://github.com/TheNitek/WeMos-Lora/
    b) Order from: https://pcbs.io/share/4XG02
        i) Soldering skills are required

H) At least one (1) OLED Shield for D1 Mini
    a) https://www.ebay.com/itm/OLED-Schild-D1-mini-0-66-zoll-64X48-IIC-I2C-white-display/152692771543?epid=13009500989&hash=item238d32cad7:g:ApkAAOSwo4pYax9w
    b) For my demo I am using two of these. Only one is required.

I) One SHT30 Shield for the D1 Mini
    a) https://www.ebay.com/itm/SHT30-Shield-for-D1-mini-I2C-digital-temperature-and-humidity-sensor-module/152719290227?epid=18009596964&hash=item238ec76f73:g:CQoAAOSwc2FaF7v-

J) At least one Battery Shield for the D1 Mini
    a) * I am using two in this demo, only one is required for the remote node

K) One DS18B20 Temperature sensor
    a) https://www.ebay.com/itm/DS18B20-Temperature-Sensor-Module-Temperature-Measurement-Module-For-Arduino/322649868645?hash=item4b1f6e4965:m:mJHBt_U0889NsU315SoVsMg
    b) * There are few different types of this sensor, the above is like the one I am using

L) Other Things, batteries, antennas, small cases, and ways to power the controllers are all needed as well.

There are a number of components that need to be soldered, take your time!
Here are a few build pictures - most of which are after everything has been soldered.
https://photos.app.goo.gl/jRGbo1zgncJQKluf1

Software:

Link To GITHUB GOES HERE!

Libraries Needed:

A) Temperature_control
https://github.com/milesburton/Arduino-Temperature-Control-Library
(used for the DS18B20)

B) arduino-LoRa (LoRa) https://github.com/sandeepmistry/arduino-LoRa

(used for all LoRa boards, and devices)

C) WEMOS_SHT3x_Arduino_Library
https://github.com/wemos/WEMOS_SHT3x_Arduino_Library
(used for the D1 Mini SHT30 sensor)
D) Adafruit_GFX https://github.com/adafruit/Adafruit-GFX-Library
E) ESP8266 SSD1306 (64x48) Library
Recommended my slightly modified version
https://github.com/kd8bxp/mcauser-64x48-OLED-SSD1306-library
Original version
https://github.com/mcauser/Adafruit_SSD1306/tree/esp8266-64x48
(This is a modified version of the Adafruit SSD1306 library for use with the small OLED and D1 Mini. My slight modification renames the library so I could have the original Adafruit version and this version in the libraries directory at the same time.) If you want to use mcauser's version, you'll have to remove the D1 from the include in the sketches.
F) Adafruit_Sensor https://github.com/adafruit/Adafruit_Sensor
G) Adafruit DHT-sensor-library https://github.com/adafruit/DHT-sensor-library
H) WiFiManager https://github.com/tzapu/WiFiManager
I) PubSubClient https://github.com/knolleary/pubsubclient
(**SPECIAL NOTES:** Need to change MQTT_MAX_PACKET_SIZE in PubSubClient.h to 512)
J) ThingSpeak - https://github.com/mathworks/thingspeak-arduino
(at one point I used thingspeak to display a single node, I've included the sketch for historical and learning reasons.) Thingspeak isn't used for this project now.

Additional Boards for board manager:

The LoRa32U4 II board is a clone of the Adafruit Feather LoRa board and needs to be added to the IDE with the board manager.
https://learn.adafruit.com/adafruit-feather-32u4-radio-with-lora-radio-module/using-with-arduino-ide
https://adafruit.github.io/arduino-board-index/package_adafruit_index.json
Linux Users may need to follow these additional steps to get the board to work:
https://learn.adafruit.com/adafruit-feather-32u4-radio-with-lora-radio-module/using-with-arduino-ide#ubuntu-and-linux-issue-fix

You will also need to add the D1 Mini boards to the IDE using the guide here:
https://github.com/esp8266/Arduino

The Sketches:

There are basically two sketches here: The "SENDER" which will be the remote sensor, it uses LoRa to SEND the sensor data to the "GATEWAY". There are three different "SENDER"

sketches. The base code for all three is the same, the only real changes are because different sensors are used for different boards. The other change is in the pins used by the LoRa device.
The idea is take a sensor reading, phase the data, and send the data, wait for 15 seconds, and do the whole process over again. 15 seconds is pretty fast for the type of data I am collecting, but I wanted to demo something that we could see work, and not wait for a while.
The “SENDER” code is pretty simple and straight forward, for the most part.

- It should be noted that the DS18B20 only reads temperatures. To keep things simple, I am sending a zero for the humidity

The “magic” happens on the “GATEWAY”.

There are 4 gateway sketches, The simplest of which is a display only sketch, which just receives the node data, and displays it on the OLED.

There is also a sketch that uses thingspeak. This works well if you only have one NODE, Thingspeak could be setup to handle more, but there is a limit to the number of data points you can send to thingspeak.

There are two versions that use a service called CROUTON - The Final and the CROUTON, both currently work the same, however the Final was also suppose to update a google sheet with the data collected - In testing this worked fine, in actual usage it didn’t work (I’m not sure why yet, and I’d like to know). CROUTON lets you setup a “Dashboard” from a device, the device tells CROUTON what to display, and how to update it.
CROUTON is open sourced, and can be installed on a local machine (Thou I ran into problems getting it to work on both the Raspberry PI and my desktop running Linux Mint 18.3)
https://github.com/edfungus/Crouton

CROUTON
http://crouton.mybluemix.net/crouton/connections
Crouton connects to a MQTT broker using websockets. The broker I’ve chosen to use is broker.hivemq.com on port 8000 (Websocket port). The GATEWAY sketches also connect to broker.hivemq.com but on the public port of 1883.
To the broker CROUTON appears to be just another device. IT subscribes to what crouton calls “devices” - these have to be unique names (So things like node1, would cause problems)
I’ve chosen the device names “node1LFMIOT, node2LFMIOT, node3LFMIOT, node4LFMIOT”
And I tell crouton to auto-connect to these “devices”. Crouton works by using a inbox and a outbox, and end points. So the first thing we need to do is set these devices up. The endpoint for this is “deviceInfo” The “deviceInfo” is a JSON string of information, that includes endpoints to listen too. The JSON string looks like this:
"{"deviceInfo":{"name":"Temperature","endPoints":{"t":{"title":"Node 2","card-type":"crouton-simple-text","units":"F","values":{"value":0}},"h":{"title":"Node 2 Humidity","card-type":"crouton-simple-text","units":"","values":{"value":0}}},"description":"LoRa Node","status":"ok"}}
So we publish this information to
/outbox/node2LFMIOT/deviceInfo

What does this mean? We are telling Crouton, we have a device named “Temperature”, and to start to listen to /outbox/node2LFMIOT/t and /outbox/node2LFMIOT/h also that we want to display this information as simple-text, and that the starting values are zero.
As far as I can tell, description and status really don’t do anything yet.

*Originally I had setup just one “device” on Crouton, the JSON string was very very long, and PubSubClient library had problems sending it. I opted to setup a “device” for each node. This worked better, and was in the long run easier to manage. Each node has the same endpoints “t” and “h”, but with a different device name. I made a couple of arrays to hold the “device” outbox information, and using the NODE number that each “SENDER” sends I was able to just use the array and one call to the MQTT broker.
These are the endpoint arrays:

```
const char* names[] =
{"/outbox/node1LFMIOT/deviceInfo","/outbox/node2LFMIOT/deviceInfo","/outbox/node3LFMIOT/deviceInfo","/outbox/
node4LFMIOT/deviceInfo"};
const char* nodet[] =
{"NULL","/outbox/node1LFMIOT/t","/outbox/node2LFMIOT/t","/outbox/node3LFMIOT/t","/outbox/node4LFMIOT/t"};
const char* nodeh[] =
{"NULL","/outbox/node1LFMIOT/h","/outbox/node2LFMIOT/h","/outbox/node3LFMIOT/h","/outbox/node4LFMIOT/h"};
```

And this is the code when we publish to each endpoint:

```
void updateCrouton() {
  char tBuff[20];
  char hBuff[20];
  sprintf(tBuff, "{\"value\":%d.%02d}",(int)t,(int)(t * 100) %100);
  sprintf(hBuff, "{\"value\":%d.%02d}",(int)h,(int)(h * 100) %100);
  client.publish(nodet[n], tBuff);
  //delay(500);
  client.publish(nodeh[n], hBuff);
  //delay(500);
}
```

So here we have gotten data from a LoRa node. We have already parsed the data to it’s base components (Node number N, Temperature T, and Humidity H).
Now we made the data a string so we can send it. (It’s also being send as a JSON string) {“value”:72.32} for an example. Notice how we used the float in the sprintf.
  When Crouton sees this change it will display it on the dashboard we setup with the deviceInfo.
Each node now has it’s own “device” that Crouton is listening for, and each one has it’s own endpoints.

Crouton can do more - It has different ways to display the data, and setup devices.
Crouton can be used to also send data from the website to the devices - this is beyond this demo, but it is something I’d like to explore.

One limitation that I see of Crouton is it doesn't store the values, so it doesn't collect data, it just displays it.

Using Google Sheets with Pushingbox

To overcome this limitation, I thought it might be interesting to update a Google Sheet with the data, this would give you a history of each node, and a way to sort the data.
In the past I've used Pushingbox for a few project demos, pushingbox makes it pretty easy to connect a device with a API - so a device can control something or do something that would be hard to do without using a third party service.
There are a few tutorials I've posted for this. And they should be easy enough to find here in my corner.
For this part thou I found a really good tutorial on how to connect Pushingbox to google sheets. (*I am not sure how secure this really is, it appears you are giving a script access to the sheet - it also appears it only has access to that sheet*)
The tutorial can be found here:
https://www.hackster.io/detox/transmit-esp8266-data-to-google-sheets-8fc617
Step 3 is a little hard to follow - What I found is the script provided needs to be outside of the "function myFunction() { } " and function myFunction() {} needs to be left in the editor.
This hung me up for a few minutes (about an hour really).
Once it works you can change the switch (param) to parameters you'd like to use, but remember to also change these on Pushingbox.

So for me, I changed the parameters to
node, t, h
The other is the "A" column to display the time as well as the date. And I added some conditions to change the color of the Node field based on node number.
Otherwise the tutorial is good.

Here is my Google Sheet showing some of my LoRa Data
https://docs.google.com/spreadsheets/d/1-MbSci0T5Q6W3Bun_CTnIg3JhvkkoX1wb2_zNF571v0/edit?usp=sharing

In Testing this worked without any problems, but under "real" conditions, the sheet was getting updated, but Crouton was not. I have not figured out why,
At first I thought it might be because of the floats - but now I kind of suspect it might be the OLED on the gateway. I'll investigate some more and see if I can get both working.

What is LoRa?

LoRa™ (Long Range) is a modulation technique based on spread-spectrum techniques and a variation of Chirp Spread Spectrum (CSS), which provides significantly longer range than the competing **technologies**. The **LoRa** wireless**technology** was developed by Cycleo SAS, which was later acquired by Semtech

How Lora works?

Gateways are connected to the network server via standard IP connections while end-devices use single-hop wireless communication to one or many gateways. ...**LoRa** is also based on chirp spread spectrum modulation which the alliance claims maintains low-power characteristics and significantly increases communication range.

LoRa is also based on chirp spread spectrum modulation which the alliance claims maintains low-power characteristics and significantly increases communication range. Chirp spread spectrum has been used in military and space communication for decades, but LoRa is the first low-cost implementation for commercial usage. Communication between end-devices and gateways is spread out on different frequency channels and data rates. The selection of the data rate is a trade-off between communication range and message duration. Due to the spread spectrum technology, communications with different data rates do not interfere with each other and create a set of “virtual” channels increasing the capacity of the gateway.

Channels 64+8+8
Channel BW Up 125/500kHz
Channel BW Dn 500kHz
TX Power Up +20dBm typ (+30dBm allowed)
TX Power Dn +27dBm
SF UP 7-10
Data rate 980bps - 21.9kpbs
Link Budget Up 154db
Link Budget Down 157dB
https://www.link-labs.com/blog/what-is-lora
https://www.link-labs.com/blog/lora-faqs

LoRa and LoRaWAN are two different things - A lot of documents found on the internet talk about LoRaWAN and are misleading to people just looking for information on LoRa.

https://www.rs-online.com/designspark/5-things-to-know-about-working-with-lora
https://www.semtech.com/uploads/documents/an1200.22.pdf
http://www.thomasclausen.net/wp-content/uploads/2016/09/2016-A-Study-of-LoRa-Long-Range-Low-Power-Networks-for-the-Internet-of-Things.pdf

Link Budget Calculations:

http://www.techplayon.com/lora-link-budget-sensitivity-calculations-example-explained/

What is MQTT?

MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimise network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging "machine-to-machine" (M2M) or "Internet of Things" world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

MQTT has been around since 1999 invented by IBM
http://mqtt.org/faq
https://en.wikipedia.org/wiki/MQTT

What is JSON?

https://www.json.org/
JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language,Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.