

Setup LoRa com Arduino, Raspberry Pi e shield Dragino



ÍNDICE DE CONTEÚDO

- Introdução
- Desenvolvimento
- Resultados
- Conclusões
- Saiba mais

Introdução

A Internet das Coisas (*Internet of Things* ou IoT) é um conceito que existe desde 1985 e pode ser definido como a integração de pessoas, processos e tecnologia com dispositivos e sensores conectados capazes de habilitar monitoramento remoto, *status*, manipulação e avaliação de tendências desses dispositivos. Sendo assim, IoT é uma das novas tecnologias com maior potencial de impacto social e econômico nos próximos anos. A integração de objetos à Internet permite a criação de aplicações inovadoras, capazes de monitorar e atuar em sistemas complexos e, com isso, possibilitar uma nova evolução tecnológica no que diz respeito à facilitação de controles que necessitam de transmissão sem fio.

As tecnologias de IoT, como por exemplo LoRa, Sigfox, RPMA e Weightless, utilizam protocolos de comunicação para conexão com a Internet. Estes protocolos visam implementar redes de *Low Power Wide Area Network* (LPWAN), onde o baixo consumo de energia e a escalabilidade, em termos do número de nós, são requisitos fundamentais.

LoRa (**Long Range** ou longo alcance) é uma técnica de modulação patenteada pela empresa francesa Semtech. A técnica baseia-se em espalhamento espectral Chirp e opera nas faixas sub-GHz, podendo ser utilizada na banda *Industrial Scientific and Medical* (ISM). LoRa oferece taxas de transmissão de 290 bps a 50 kbps, o que pode ser suficiente para a transmissão de dados de monitoramento ou de imagens de baixa resolução com pacotes de até 255 bytes e operando em larguras de bandas de 125, 250 ou 500 kHz. O alcance da tecnologia é de aproximadamente até 5 km em áreas urbanas, 15 km em áreas rurais e 30 km na superfície da água.

LoRaWAN, por sua vez, é a especificação da pilha de protocolos em que LoRa é utilizado na camada física.

Neste artigo será abordado um método de comunicação configurável pelo LoRaWAN capaz de avaliar a funcionalidade prática da tecnologia por um baixo custo. A Figura 1 mostra o *setup* que será utilizado e detalhado mais adiante.

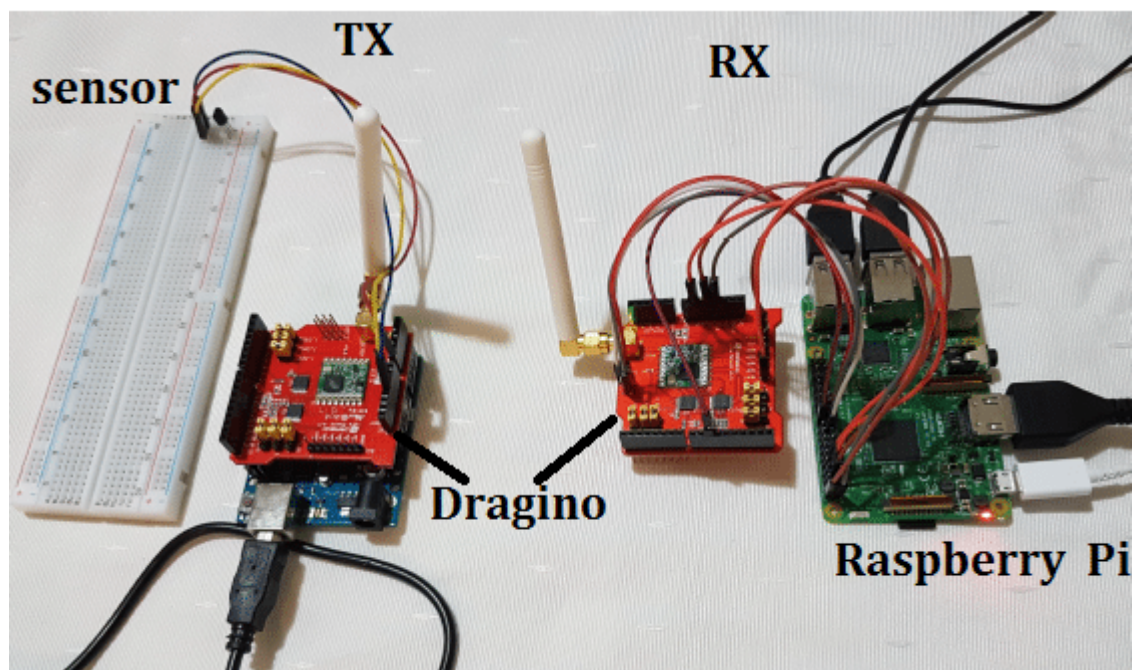


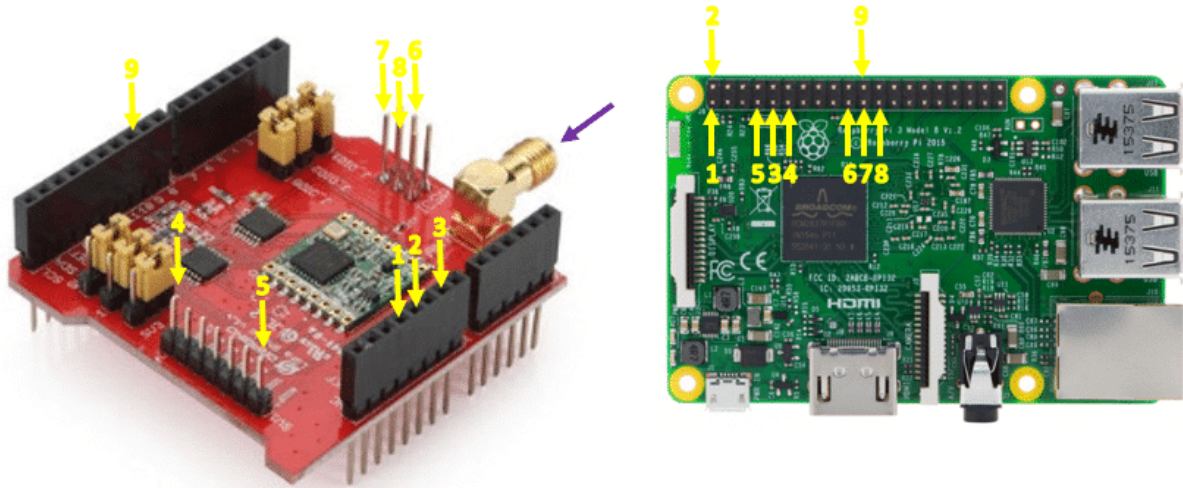
Figura 1 – Setup de baixo custo para avaliação do LoRa.

Desenvolvimento

O desenvolvimento para prototipação consiste em integrar a *shield* [Dragino](#) com o módulo nó sensor, responsável pela captura dos dados ambientais (que neste caso foi utilizado um Arduino com um sensor de temperatura LM35) e também com um nó *gateway* montado com outra *shield* Dragino e um [Raspberry Pi 3](#).

O passo a passo da configuração segue abaixo.

1) Plugar os conectores tipo macho-fêmea entre o Dragino e o RPi conforme descreve a imagem e a tabela abaixo.



Figuras 2 e 3 – Conexões do Shield Dragino e da Raspberry Pi.

Tabela 1 – Designação de pinos entre o Dragino e o Raspberry Pi

Mapa	Dragino	RPi	Pino Rpi
1	3V3	3.3V PWR	1
2	5V	5V	2
3	GND	GND	9
4	RESET	GPIO17	11
5	DIO0	GPIO4	7
6	ICSP4	GPIO10 (MOSI)	19
7	ICSP1	GPIO9 (MISO)	21
8	ICSP3	GPIO11 (CLK)	23
9	D10	GPIO25	22

Nas Figuras 2 e 3, em amarelo não consta a pinagem do RPi, mas sim o diagrama conforme feita a conexão no Dragino. A pinagem correta está descrita na Tabela 1 pela coluna “Pino RPi”.

Após as conexões, basta rosquear a antena no local indicado pela seta roxa.

No RPi, plugar via USB um teclado e um mouse, assim como um cabo HDMI a um monitor para amostragem dos dados. Se não foi feita a conexão cabeada pela rede LAN, configurar o RPi para utilizar o Wi-Fi.

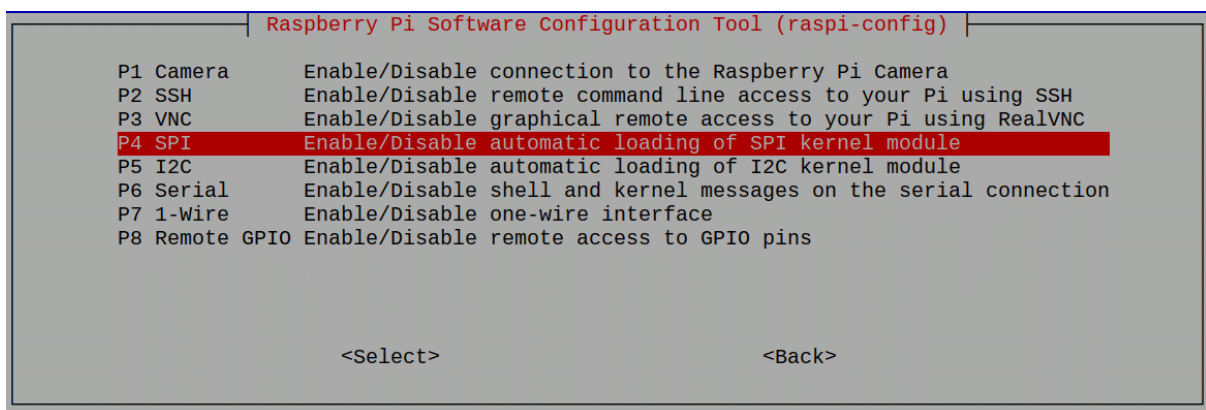
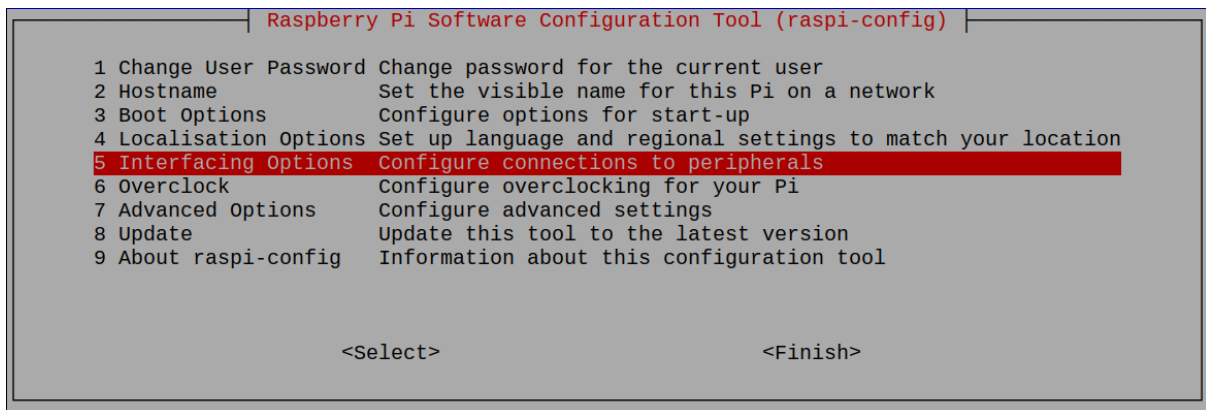
2) Após todas as conexões entre o Dragino e o RPi, deve-se realizar a programação do RPi via terminal de comandos.

A primeira tarefa baseia-se em clonar o diretório do *software* que o *Gateway* irá executar:

```
git clone https://github.com/tftelkamp/single_chan_pkt_fwd
```

Depois, uma vez que necessitamos da comunicação síncrona serial entre os *hardwares*, é necessário habilitar o SPI e reiniciar o RPi:

```
pi@raspberrypi:~ $ sudo raspi-config
```




Would you like the SPI interface to be enabled?

<Yes>

<No>

The SPI interface is enabled

<Ok>



The SPI interface is enabled

<Ok>

```
pi@raspberrypi:~ $ sudo shutdown -r now
```

Para que se consiga controlar os pinos GPIO do RPi é necessário executar o comando a seguir:

Hoje, em 2022, a biblioteca do WiringPi, que é utilizada para controlar os pinos GPIO de RPi foi descontinuada, e não se encontra mais disponível oficialmente, para contornar essa situação foi encontrado um repositório no github que contém os códigos relativos a essa biblioteca. Esse código pode ser instalado conforme os comandos a seguir:

\$ git clone <https://github.com/WiringPi/WiringPi>

para clonar o repositório para o raspberry

após esse passo nós devemos dar um build nos arquivos com o seguinte código no terminal:

\$ cd ~/wiringPi

\$./build

No arquivo principal que foi clonado, alguns ajustes devem ser feitos para melhor adequação ao cenário Brasileiro. Assim, o arquivo fonte principal deve ser acessado e então alterado conforme abaixo:

```
pi@raspberrypi:~ $ cd ~/single_chan_pkt_fwd
pi@raspberrypi:~/single_chan_pkt_fwd $ nano main.cpp
```

```
// SX1272 - Raspberry connections
int ssPin = 6;
int dio0 = 7;
int RST = 0;

// Set spreading factor (SF7 - SF12)
sf_t sf = SF10;

// Set center frequency
uint32_t freq = 915000000; // in Mhz! (915.0)
```

As configurações que precisam ser feitas são no fator de espalhamento, que varia entre 7 e 12 (com limitações de 7 a 10 para 915 MHz devido ao *dwell time*) e a frequência de operação. Por padrão, os pinos de habilitação do transdutor são clonados já com as definições corretas para o uso no RPi e, portanto, a sequência para ssPin, dio0 e RST mantêm-se em 6, 7 e 0.

A localização geográfica que o *Gateway* irá situar-se também pode ser parametrizada nesta etapa:

```
// Set location
float lat=-22.89253001;
float lon=-46.407123430;
int alt=350;
```

Após, deve ser direcionado um servidor e uma porta válida para conexão do RPi com a rede. Este servidor varia de acordo com o arquivo principal clonado e o próprio Dragino estabelece alguns servidores disponíveis.


```
// define servers
// TODO: use host names and dns
#define SERVER1 "13.76.168.68"
//#define SERVER2 "192.168.1.10"
#define PORT 1700
```

Por fim, caso alguma modificação foi feita no arquivo principal, o comando abaixo deve ser realizado para executar o *software*.

```
pi@raspberrypi:~/single_chan_pkt_fwd $ make
```

Caso tudo esteja configurado de acordo com as características do *hardware* utilizado, ao executar o programa, o *gateway* estará apto a receber pacotes de qualquer nó que estiver enviando, além de mostrar a frequência de operação que está trabalhando e o fator de espalhamento. Outras informações, como o endereço identificador do *gateway* e dados de recebimento de pacotes, também podem ser consultados pelo RPi, embora as informações sejam melhor visualizadas no servidor em nuvem que o sistema irá interagir.

```
pi@raspberrypi:~/single_chan_pkt_fwd $ ./single_chan_pkt_fwd
SX1276 detected, starting.
Gateway ID: b8:27:eb:ff:ff:33:fa:b6
Listening at SF10 on 915.000000 Mhz.
-----
stat update: {"stat":{"time":"2018-02-07 17:39:26 GMT","lati":-22.89253,"long":-46.40712,"alti":350,"rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":"","desc":""}}
```

● Programação no Arduino:

3) No que diz respeito à conexão do Dragino ao Arduino, basta que se encaixe cuidadosamente a *shield* de modo que todos os pinos fiquem conectados nos respectivos locais. Após, o cabo USB deve ser plugado no Arduino e conectado ao computador para que possa ser realizada a programação pela interface de desenvolvimento. Para comprovação dos testes de sensoriamento remoto com o LoRa, inseriu-se o sensor de temperatura LM35, alimentado com 5 V e programado pelo pino analógico 0 do Dragino. Deve-se ajustar as configurações de fator de espalhamento no nó sensor da mesma forma que foi realizada para o *gateway*.

As linhas de código utilizadas para o nó sensor podem ser conferidas no APÊNDICE (créditos: Thomas Telkamp e Matthijs Kooijman) mas algumas observações devem ser levadas em consideração para o nó sensor:

- No código fonte, o LM35 foi estabelecido como ponto de exemplo para auferir a temperatura, entretanto qualquer outro tipo de sensor cuja informação captada seja de baixa taxa de bits também poderia ser utilizado. A limitação de canal, contudo, foi feita para que o nó sensor só canalize os dados obtidos e transmita em uma única frequência.

```
for (int i = 1; i < 64; i++)
{
    LMIC_disableChannel(i);
}
```

- A biblioteca utilizada é a <lmic.h> e nos subdiretórios desta biblioteca, o arquivo config.h deve ser alterado para a padronização brasileira.

```
//#define CFG_eu868 1
#define CFG_us915 1
```

- No arquivo lorabase.h, a frequência de operação correta também deve ser ajustada.

```
// Default frequency plan for US 915MHz
enum { US915_125kHz_UPFBASE = 915000000
```

- Na linha 760 do arquivo limic.c deve ser ajustada a divisão de canais para usos futuros.

```
760 void LMIC_disableChannel (ul_t channel) {
761     if( channel < 72+MAX_XCHANNELS )
762         LMIC.channelMap[channel/16] &= ~(1<<(channel&0xF));
763 }
```

- Deve ser realizada a limitação no número de canais de transmissão caso não haja muitos *uplinks* ou *downlinks*. Para o simples sensoriamento remoto de um nó, um canal é suficiente para averiguar as condições ambientais e enviar a um *gateway* conectado à nuvem. Logo, é preciso desabilitar todos os canais que não estão em uso uma vez que se isso não for feito, o transmissor tentará enviar pacotes em todos os canais existentes enquanto que o receptor só estará apto a operar em um canal.

- **Configuração do gateway a plataforma The things Network:**

/* A conexão não foi possível em 2022 pois a plataforma em sua versão v3 não permite que transmissores LoRa com 1 único canal de transmissão funcionem como gateway */

4) Para configurar o servidor em nuvem que irá fazer a amostragem dos pacotes recebidos, os passos a seguir devem ser considerados. É importante salientar que o servidor não faz nenhuma atuação de resposta, mas pode ser associado a algum atuador em outras aplicações capaz de assim fazê-lo.

Primeiramente, é preciso acessar a plataforma [The Things Network](#) e efetuar um cadastro e *login*.



Please log in

EMAIL OR USERNAME



l.restivo93@gmail.com|

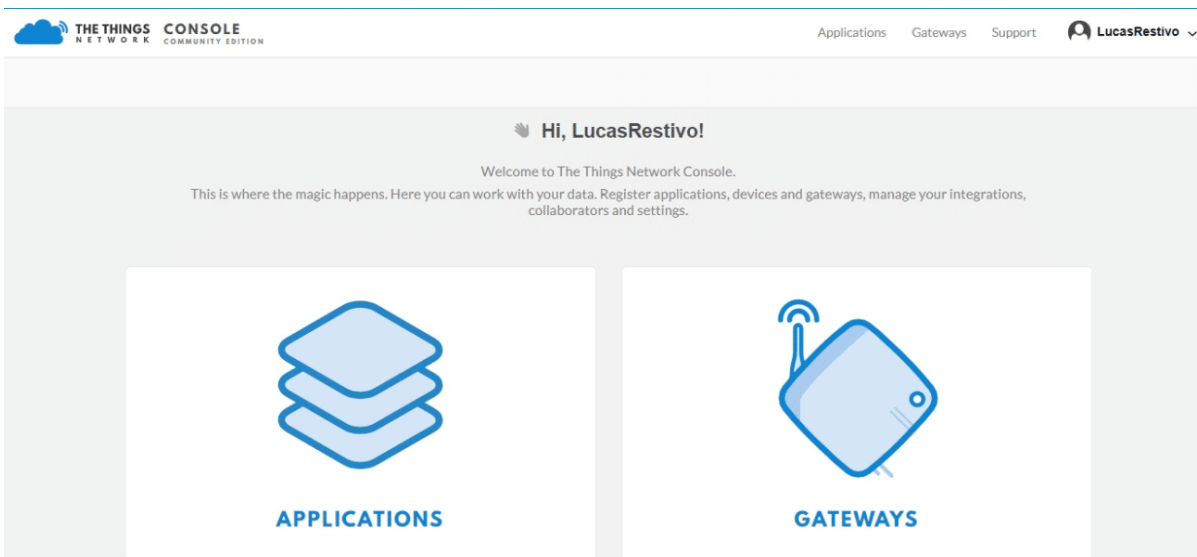
PASSWORD



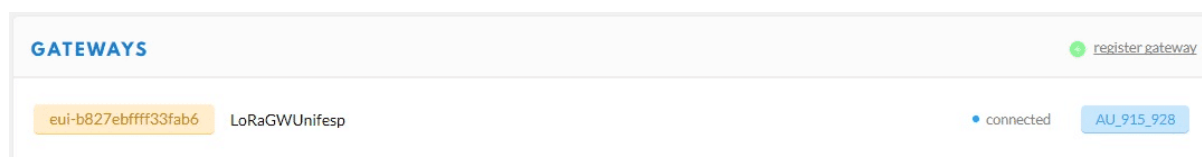
••••••••

Log in

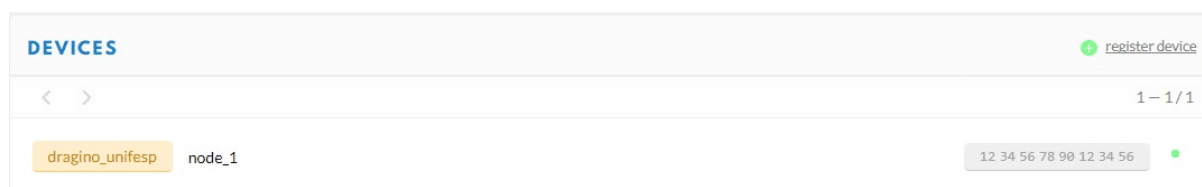
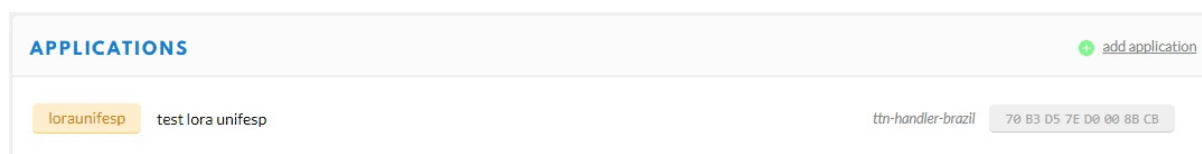
Na tela inicial, são mostradas duas abas que precisam ser ajustadas: *Applications* e *Gateways*.



Em *Gateways*, deve-se registrar o RPi com o módulo Dragino para torná-lo integrado no servidor. Algumas informações básicas como nome do *gateway*, região e frequência de operação devem ser configuradas de acordo com o identificador do RPi programado previamente.



De maneira similar, em *Applications* é feito o registro dos nós que constituem todo o sistema, de modo que cada um tenha um registro único e atrelado ao seu respectivo código fonte no Arduino. Este registro é composto por três chaves geradas pelo servidor: Device Address, Network Session Key e App Session Key.



Applications > loraunifesp > Devices > dragino_unifesp

Application ID loraunifesp

Device ID dragino_unifesp

Description node_1

Activation Method ABP

Device EUI <> 12 34 56 78 90 12 34 56

Application EUI <> 70 B3 D5 7E D0 00 8B CB

Device Address <> 26 03 14 33

Network Session Key <> A3 FB B7 18 A9 CB 4E B1 2A 3C EE 54 BA F8 F2 D9

App Session Key <> AE 41 F3 1B 03 3C 53 0D 94 22 A8 E0 89 D1 B4 79

Status 6 minutes ago

Os pacotes recebidos, enviados e seus respectivos detalhes de transmissão podem ser consultados nas abas de *Traffic* no servidor. O tempo de duração no ar que um pacote levou para ser enviado de um nó sensor até um *gateway*, a frequência de operação que este sistema está operando, o *coding rate* (CR), fator de espalhamento, largura de banda, identificador do nó e tamanho do pacote podem ser verificado nas abas de gerenciamento do servidor.

Gateways > eui-b827ebffff33fab6 > Traffic beta

GATEWAY TRAFFIC beta

uplink downlink join 0 bytes X

|| pause trash clear

time	frequency	mod.	CR	data rate	airtime (ms)	cnt	
▲ 16:08:54	915	lora	4/5	SF 10 BW 125	329.7	43	dev addr: 26 03 14 33 payload size: 18 bytes
▲ 16:08:29	915	lora	4/5	SF 10 BW 125	329.7	42	dev addr: 26 03 14 33 payload size: 18 bytes
▲ 16:08:05	915	lora	4/5	SF 10 BW 125	329.7	41	dev addr: 26 03 14 33 payload size: 18 bytes
▲ 16:07:40	915	lora	4/5	SF 10 BW 125	329.7	40	dev addr: 26 03 14 33 payload size: 18 bytes
▲ 16:07:16	915	lora	4/5	SF 10 BW 125	329.7	39	dev addr: 26 03 14 33 payload size: 18 bytes
▲ 16:06:52	915	lora	4/5	SF 10 BW 125	329.7	38	dev addr: 26 03 14 33 payload size: 18 bytes
▲ 16:06:29	915	lora	4/5	SF 10 BW 125	329.7	37	dev addr: 26 03 14 33 payload size: 18 bytes
▲ 16:00:04	915	lora	4/5	SF 10 BW 125	329.7	21	dev addr: 26 03 14 33 payload size: 18 bytes

Também é possível visualizar os detalhes da transmissão de cada pacote que foi recebido pelo *gateway*. Neste teste foi criado um nó sensor da aba *Applications* e gerado um endereço de dispositivo que foi descrito no código do Arduino. Como exemplo, o contador de transmissões completas de número 43 identifica uma carga

codificada que foi enviada de um nó sensor com registro de número 26031433 ao gateway eui-b827ebffff33fab6.

```
2  {"gw_id": "eui-b827ebffff33fab6",
3  "payload": "QDRUyaAKuA8BUm00zhKebqx",
4  "f_cnt": 43,
5  "lora": {
6    "spreading_factor": 10,
7    "bandwidth": 125,
8    "air_time": 329720000
9  },
10 "coding_rate": "4/5",
11 "timestamp": "2018-02-07T18:08:54.398Z",
12 "rssi": -49,
13 "snr": 13,
14 "dev_addr": "26031433",
15 "frequency": 915.0,
16 "data_rate": "SF 10 BW 125"
}
```

Na análise de tráfego de dados do gateway pode então ser mostrado de qual nó foi enviado o pacote, visualizado o tamanho total da carga e também só a codificação em HEX da informação.

time	frequency	mod	CR	data rate	airtime (ms)	cnt
16:08:54	915	lor	4/5	SF 10 BW 125	329.7	43

dev addr: 26 03 14 33 payload size: 18 bytes

Uplink

Dev Address

26 03 14 33

Network: The Things Network
Net ID: 0x13
Region: World

Physical Payload

40 33 14 03 26 00 2B 00 01 05 49 8E 0F 38 64 69 BA B1

Na tela de dados do RPi, pode-se constatar que os dados criptografados para as transmissões correspondem aos dados recebidos no servidor. No exemplo citado da transmissão 43, os dados do pacote são registrados conforme são enviados.

```

Packet RSSI: -47, RSSI: -104, SNR: 10, Length: 18
rxpk update: {"rxpk":{"tmst":783273403,"chan":0,"rfch":0,"freq":915.000000,"stat":1,"modu":"LORA",
,"datr":"SF10BW125","codr":"4/5","lsnr":10,"rssi":-47,"size":18,"data":"QDMUAYaAKgABWd7Mc8FBbBAS
"}}}
Packet RSSI: -49, RSSI: -104, SNR: 13, Length: 18
rxpk update: {"rxpk":{"tmst":808074861,"chan":0,"rfch":0,"freq":915.000000,"stat":1,"modu":"LORA",
,"datr":"SF10BW125","codr":"4/5","lsnr":13,"rssi":-49,"size":18,"data":"QDMUAYaAKwABBUM0Dzhkabqx
"}}}
stat update: {"stat":{"time":"2018-02-07 18:08:55 GMT","lati":-22.89253,"long":-46.40712,"alti":3
50,"rxnb":2,"rxok":2,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":
":"","desc":""}}
Packet RSSI: -49, RSSI: -102, SNR: 13, Length: 18
rxpk update: {"rxpk":{"tmst":832971213,"chan":0,"rfch":0,"freq":915.000000,"stat":1,"modu":"LORA",
,"datr":"SF10BW125","codr":"4/5","lsnr":13,"rssi":-49,"size":18,"data":"QDMUAYaALAA80Y/qjHhQrwm9
"}}}
stat update: {"stat":{"time":"2018-02-07 18:09:25 GMT","lati":-22.89253,"long":-46.40712,"alti":3
50,"rxnb":1,"rxok":1,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0,"pfrm":"Single Channel Gateway","mail":
":"","desc":""}}
Packet RSSI: -52, RSSI: -104, SNR: 10, Length: 18
rxpk update: {"rxpk":{"tmst":857262195,"chan":0,"rfch":0,"freq":915.000000,"stat":1,"modu":"LORA",
,"datr":"SF10BW125","codr":"4/5","lsnr":10,"rssi":-52,"size":18,"data":"QDMUAYaALQABFV0p4Px48j
"}}}
stat update: {"stat":{"time":"2018-02-07 18:09:55 GMT","lati":-22.89253,"long":-46.40712,"alti":3

```

Nota-se que no Monitor Serial do Arduino os dados obtidos pelo sensor são informados e enfileirados para a transmissão sem fio junto a um número de contagem para a análise quantitativa dos pacotes que estão sendo enviados, além de outras informações de controle. Na tela de amostragem de pacotes recebidos do servidor *The Things Network*, é possível comprovar a correta transferência de dados ao converter os valores em DEC dos sensores para HEX conforme mostra o servidor.

The screenshot shows the Arduino Serial Monitor window for COM5 (Arduino/Genuino Uno). It displays the following data being transmitted:

```

Packet queued
TX n°: 43
915000000
Temperatura = 28.06 °C
64768043: 10
EV_TXCOMPLETE (includes waiting for RX windows)

Packet queued
TX n°: 44
915000000
Temperatura = 27.96 °C
66285611: 10
EV_TXCOMPLETE (includes waiting for RX windows)

Packet queued
TX n°: 45
915000000
Temperatura = 27.96 °C

```

Below the Serial Monitor, the The Things Network packet capture interface is visible, showing a list of received packets. The packet with ID 43 is highlighted, showing its payload in hexadecimal: 32 38 2E 30 36.

Time	ID	Size	Payload (Hex)
16:09:43	45	1	payload: 32 37 2E 39 36
16:09:19	44	1	payload: 32 37 2E 39 36
16:08:54	43	1	payload: 32 38 2E 30 36
16:08:29	42	1	payload: 32 38 2E 30 36
16:08:05	41	1	payload: 32 38 2E 30 36
16:07:41	40	1	payload: 32 37 2E 39 36
16:07:17	39	1	payload: 32 37 2E 38 35

É possível verificar a redução do tempo no ar que um pacote transmitido com fator de espalhamento 7 sofre em relação ao mesmo pacote transmitido com fator de espalhamento 10. A distância na qual os testes foram realizados não sofreriam

consequência significativas de perdas de pacotes mesmo com fator de espalhamento mais elevado, entretanto, se o mesmo teste for executado para separações maiores entre o nó sensor e o *gateway* é possível notar diferenciações em termos de perdas de pacotes.

Esta característica é intrínseca à tecnologia LoRa e se deve ao fato de haver proporcionalidade inversa entre o fator de espalhamento e alcance ou tempo no ar, deixando padrão todos os outros parâmetros.

GATEWAY TRAFFIC

beta

uplink

downlink

join

0 bytes

✕

⏏ pause

🗑 clear

time	frequency	mod.	CR	data rate	airtime (ms)	cnt	
▲ 16:26:08	915	lor	4/5	SF 7 BW 125	51.5	1	dev addr: 26 03 14 33 payload size: 18 bytes
▲ 16:25:45	915	lor	4/5	SF 7 BW 125	51.5	0	dev addr: 26 03 14 33 payload size: 18 bytes
▲ 16:23:26	915	lor	4/5	SF 10 BW 125	329.7	79	dev addr: 26 03 14 33 payload size: 18 bytes
▲ 16:23:02	915	lor	4/5	SF 10 BW 125	329.7	78	dev addr: 26 03 14 33 payload size: 18 bytes

Pode-se ainda associar estas informações a um banco de dados e em um servidor de aplicação para melhor quantificação dos dados, assunto este que pode ser abordado em um artigo posterior.

Resultados

Nos testes realizados, foi possível alcançar 3 km entre o nó sensor e o nó *gateway* em uma área urbana com antena não direcional, mas com perdas de pacotes consideráveis.

SF7	
Distância (m)	Tx Receb. (%)
1000	81,3
500	96,4

Tabela 2 – Fator de espalhamento 7

SF12	
Distância (m)	Tx Receb. (%)
3000	*
2500	65,0
2000	72,2
1500	100,0

Tabela 3 – Fator de espalhamento 12

Conforme mostra as Tabelas 2 e 3, a transferência sem perdas foi possível obter para distâncias de até 1,5 km, que já é o suficiente para cobrir uma boa área e capacitar o monitoramento remoto em diversas aplicações, como número de veículos, poluição do ar, fluxo de pedestres e gerenciamento de lixo, por exemplo.

Na Figura 4 é possível exemplificar a cobertura de 1,5 km de raio quando utilizados dois *gateways* recebendo dados de nós LoRa espalhados na cidade de São José dos Campos, São Paulo. Pode-se verificar que vários bairros podem ser cobertos com esta configuração.

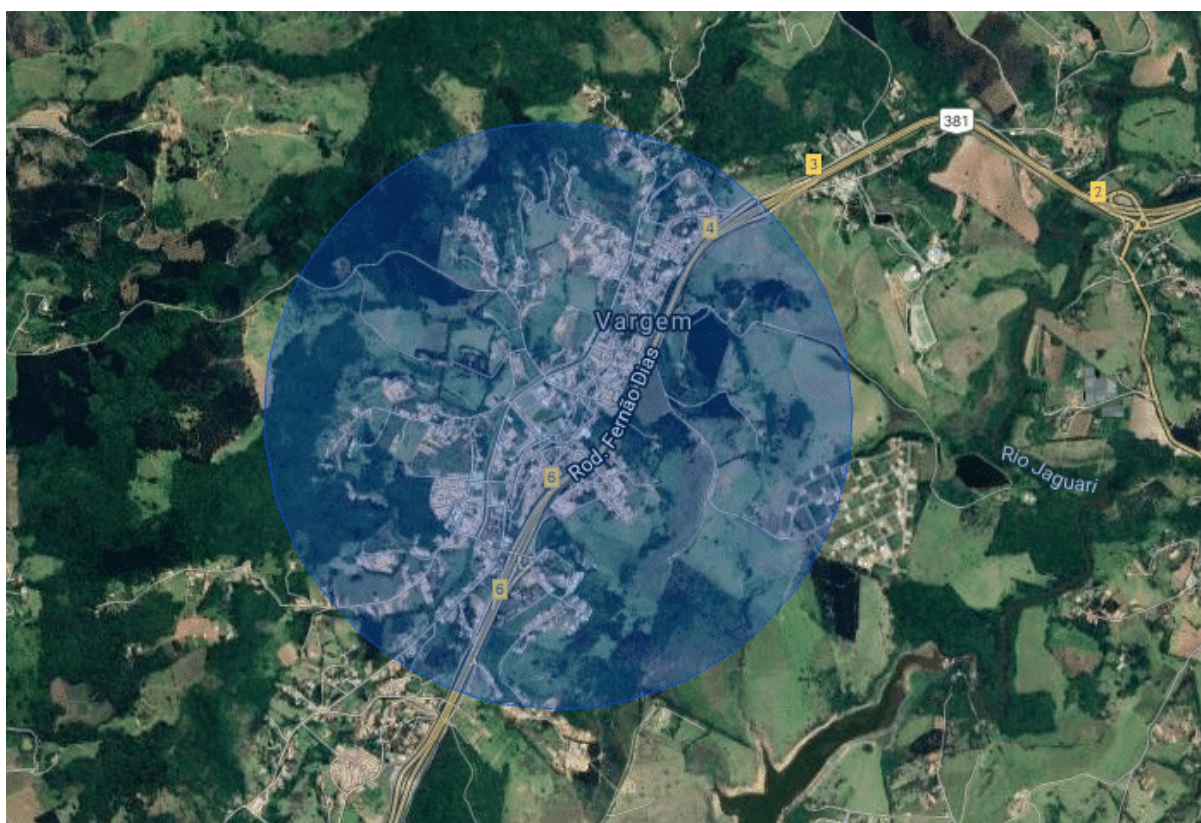


Figura 5 – Aplicação de um gateway LoRa em cidades do interior.

Conclusões

As aplicações de LoRa para IoT são inúmeras. *Smart City*, *Smart Grid*, *Smart Farm*, *Health Care*, campus universitários e até tropas militares já se beneficiam com esta tecnologia. Esses resultados expressam que a tecnologia é funcional para *Smart City*, mas que pode ser aperfeiçoada com ajustes de potência de transmissão, ganho de antena, fator de espalhamento e gerenciamento de pacotes recebidos. Mesmo assim, foi possível avaliar o bom desempenho do protocolo LoRaWAN e sua atuação no sensoriamento remoto urbano utilizando *hardwares* de custo relativamente baixo e acessível à população.

Referências

- Adelantado, F., Vilajosana, X., TusetPeiro, P., Martinez, B., Melià-Seguí, J., and Watteyne, T. (2017). Understanding the limits of lorawan. *IEEE Communications Magazine*, pages 34–40.
- Petäjäjärvi, J., Mikhaylov, K., Roivainen, A., Hänninen, T., and Pettissalo, M. (2015). On the coverage of lpwans: Range evaluation and channel attenuation model for lora technology. *ITS Telecommunications (ITST)*, pages 1–5.

Petäjärvi, J., Mikhaylov, K., Yasmin, R., Hämäläinen, M., and Linatti, J. (2017). Evaluation of LoRa LPWAN Technology for Indoor Remote Health and Wellbeing Monitoring. International Journal of Wireless Information Networks, 24(2):153–165.

Battle, S., Gaster, B. (2017) LoRaWAN Bristol. IEEE. Proceedings of the 21st International Database Engineering and Applications Symposium, pages 287-290.

Semtech (2017). What is LoRa?

<https://www.semtech.com/wireless-rf/internet-of-things/what-is-lora/>. Acessado: 18/09/2017

LoRa Alliance, Inc. (2017a). LoRaWAN 1.1 Regional Parameters. Version 1.1, published in October 11, 2017.

[Dragino](#)

[Raspberry Pi](#)

[Conheça a tecnologia LoRa® e o protocolo LoRaWAN™](#)

[Plataformas de desenvolvimento baseadas em LoRa](#)

[Código fonte do nó sensor](#)

[Código fonte do gateway](#)

Apêndice

É o mesmo [código fonte](#) do nó sensor descrito nas referências.

● Código do arduino:

```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
//ttn
static const PROGMEM u1_t NWKSKEY[16] = { 0xA3, 0xFB, 0xB7, 0x18, 0xA9, 0xCB,
0x4E, 0xB1, 0x2A, 0x3C, 0xEE, 0x54, 0xBA, 0xF8, 0xF2, 0xD9 };
// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
//ttn
static const u1_t PROGMEM APPSKEY[16] = { 0xAE, 0x41, 0xF3, 0x1B, 0x03, 0x3C,
0x53, 0x0D, 0x94, 0x22, 0xA8, 0xE0, 0x89, 0xD1, 0xB4, 0x79 };

//
// LoRaWAN end-device address (DevAddr)
// See https://thethingsnetwork.org/wiki/AddressSpace
// ttn
static const u4_t DEVADDR = 0x26031433;
static const int LM35 = 0;
int temp_lida = 0;
float temperatura;
int contador = 0;

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static uint8_t mydata[] = { 0, 0, 0, 0, 0, 0, 0, 0 };
static osjob_t initjob, sendjob, blinkjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 20;

// Pin mapping
const lmic_pinmap lmic_pins = {
```

```

.nss = 10,
.rxtx = LMIC_UNUSED_PIN,
.rst = 9,
.dio = {2, 6, 7},
};
void do_send(osjob_t* j) {

    dtostrf(temperatura, 5, 2, (char*)mydata);

    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        // Prepare transmission at the next possible time.
        LMIC_setTxData2(1, mydata, strlen((char*) mydata), 0);
        Serial.println();
        Serial.println("Packet queued");
        Serial.print("TX n°: ");
        Serial.println(contador);
        contador++;
        Serial.println(LMIC.freq);
        Serial.print("Temperatura = ");
        Serial.print(temperatura);
        Serial.println(" *C");
    }
}

void onEvent (ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(": ");
    Serial.println(ev);
    switch (ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println("EV_SCAN_TIMEOUT");
            break;
        case EV_BEACON_FOUND:
            Serial.println("EV_BEACON_FOUND");
            break;
        case EV_BEACON_MISSED:
            Serial.println("EV_BEACON_MISSED");
            break;
        case EV_BEACON_TRACKED:
            Serial.println("EV_BEACON_TRACKED");
            break;
        case EV_JOINING:
            Serial.println("EV_JOINING");
            break;
        case EV_JOINED:

```

```

    Serial.println("EV_JOINED");
    break;
case EV_RFU1:
    Serial.println("EV_RFU1");
    break;
case EV_JOIN_FAILED:
    Serial.println("EV_JOIN_FAILED");
    break;
case EV_REJOIN_FAILED:
    Serial.println("EV_REJOIN_FAILED");
    break;
case EV_TXCOMPLETE:
    Serial.println("EV_TXCOMPLETE (includes waiting for RX windows)");
    if (LMIC.dataLen) {
        // data received in rx slot after tx
        Serial.print("Data Received: ");
        Serial.write(LMIC.frame + LMIC.dataBeg, LMIC.dataLen);
        Serial.println();
    }
    // Schedule next transmission
    os_setTimedCallback(&sendjob, os_getTime() + sec2osticks(TX_INTERVAL),
do_send);
    break;
case EV_LOST_TSYNC:
    Serial.println("EV_LOST_TSYNC");
    break;
case EV_RESET:
    Serial.println("EV_RESET");
    break;
case EV_RXCOMPLETE:
    // data received in ping slot
    Serial.println("EV_RXCOMPLETE");
    break;
case EV_LINK_DEAD:
    Serial.println("EV_LINK_DEAD");
    break;
case EV_LINK_ALIVE:
    Serial.println("EV_LINK_ALIVE");
    break;
default:
    Serial.println("Unknown event");
    break;
}
}

void setup() {
    Serial.begin(9600);

```



```

analogReference(INTERNAL);
while (!Serial);
Serial.println("Starting");
#ifdef VCC_ENABLE
    // For Pinoccio Scout boards
    pinMode(VCC_ENABLE, OUTPUT);
    digitalWrite(VCC_ENABLE, HIGH);
    delay(1000);
#endif

    // LMIC init
    os_init();
    // Reset the MAC state. Session and pending data transfers will be discarded.
    LMIC_reset();
    //LMIC_setClockError(MAX_CLOCK_ERROR * 1/100);
    // Set static session parameters. Instead of dynamically establishing a session
    // by joining the network, precomputed session parameters are be provided.
#ifdef PROGMEM
    // On AVR, these values are stored in flash and only copied to RAM
    // once. Copy them to a temporary buffer here, LMIC_setSession will
    // copy them into a buffer of its own again.
    uint8_t appskey[sizeof(APPSKEY)];
    uint8_t nwkskey[sizeof(NWKSKEY)];
    memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
    memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
    LMIC_setSession (0x1, DEVADDR, nwkskey, appskey);
#else
    // If not running an AVR with PROGMEM, just use the arrays directly
    LMIC_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);
#endif

    // Disable link check validation
    LMIC_setLinkCheckMode(0);

    // TTN uses SF9 for its RX2 window.
    LMIC.dn2Dr = DR_SF9;

    // Set data rate and transmit power (note: txpow seems to be ignored by the library)
    LMIC_setDrTxpow(DR_SF10, 14);

    for (int i = 1; i < 64; i++)
    {
        LMIC_disableChannel(i); // only the first channel 902.3Mhz works now.
    }

    // Start job
    do_send(&sendjob);

```

```
}  
  
void loop() {  
  os_runloop_once();  
  temp_lida = analogRead(LM35);  
  temperatura = temp_lida * 0.1075268817204301;  
}
```

Ato nº 14448 da ANATEL. Lá especifica-se para 250 KHz, que é uma possibilidade de configuração. Os padrões americanos e australianos situam-se em casos similares (no fórum da TTN também há algumas pessoas comentando sobre isso).