2º Trabalho Prático de TERC - Lora

Andrew Martins, Daniel Saldanha, Davi Simite, Hilton Costa, Laísa Paiva, Letícia Balbi e Ligia Marcia

Instituto de Computação – Universidade Federal do Amazonas (UFAM)
Av. General Rodrigo Otávio, 6200 – Coroado I – CEP: 69077-000 – Manaus – AM – Brasil

1. Introdução

A internet das coisas (**IoT**) é um conceito o qual se refere à interconexão digital de objetos cotidianos com a internet. Entre os dispositivos conectados estão presentes sensores de temperatura, umidade, *ph*, proximidade e etc. Dependendo do ambiente e da finalidade dos sensores, a distância entre eles e a área de cobertura de dispositivos os quais os gerenciam pode ser maior que o limite, desse modo, impossibilitando a comunicação entre os dispositivos que utilizam protocolos de alta frequência e curto alcance, como o 802.11. Redes de Longa Distância e Baixa Potência (LPWAN) são ideais para esse cenário onde é preciso enviar mensagens com pouca informação por uma distância grande na qual outras tecnologias não conseguem oferecer um bom serviço. LPWAN contorna essa situação através da utilização de baixas frequências para conectar os dispositivos. A comunicação realizada utilizando poucos *Hertz* na frequência sofre menos atenuação pelo ambiente SNR(*Signal Noise Relationship*) devido a sua baixa amplitude de onda, porém, a quantidade de informação que pode ser transmitida é menor. Manipulando esse tipo de rede é possível configurar dispositivos *gateway* para mandar ou receber mensagens de grandes distâncias, e depois utilizar outra rede como a *ethernet* para mandar ou receber mensagens de outro dispositivo como um computador ou tablet.

2. Experimento

2.1. Objetivo

Neste experimento, vamos caracterizar o canal entre um sensor e o *gateway* usando *LoRa* . O Sensor será representado pelo medidor de *ph* da água conectado a um Arduíno que fará a comunicação com o *gateway*, o qual será um *raspbery pi* com *Lora Shield* conectado a ele. O objetivo é verificar questões relacionadas às parametrizações da comunicação *LoRa*, efeitos na potência de sinal recebida, a relação sinal-ruído, tempo de comunicação, taxa de perda e distância de alcance entre os nodos em relação a *Coding Rate - CR*.

2.2. Materiais

- 1 Raspberry Pi 3 Model B v1.2
- 1 Arduino Uno R3
- 2 LoRa Shield
- Conectores P1
- 2 Antenas
- Sensor de *ph* da água
- periféricos (Monitor, teclado e etc...)

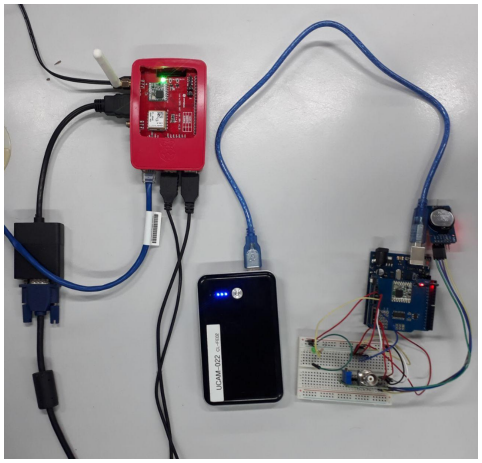

Figura 1. Cenário do experimento de teste usando os equipamentos

O *raspbery pi* usado no experimento já está conectado com o *LoRa Shield*. Foram utilizadas 11 pinos do *raspbery pi* para o reconhecimento da biblioteca *pyRadioHead* que foi utilizada no experimento. Elas são:

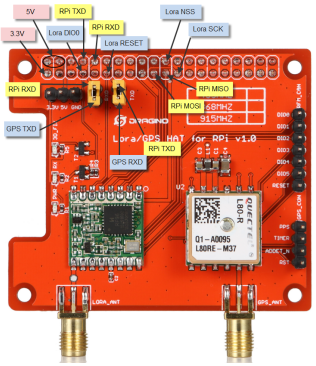

Figura 2. *Lora/GPS HAT*[Dragino 2019]

BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
		3.3v			1 || 2			5v		
2	8	SDA.1	ALT0	1	3 || 4			5V		
3	9	SCL.1	ALT0	1	5 || 6			0v		
4	7	GPIO. 7	IN	1	7 || 8	0	ALT0	TxD	15	14
		0v			9 || 10	1	ALT0	RxD	16	15
17	0	GPIO. 0	IN	0	11 || 12	0	IN	GPIO. 1	1	18
27	2	GPIO. 2	IN	0	13 || 14			0v		
22	3	GPIO. 3	IN	0	15 || 16	0	IN	GPIO. 4	4	23
		3.3v			17 || 18	1	OUT	GPIO. 5	5	24
10	12	MOSI	ALT0	0	19 || 20			0v		
9	13	MISO	ALT0	1	21 || 22	1	OUT	GPIO. 6	6	25
11	14	SCLK	ALT0	1	23 || 24	1	ALT0	CE0	10	8
		0v			25 || 26	1	ALT0	CE1	11	7
0	30	SDA.0	ALT0	1	27 || 28	1	ALT0	SCL.0	31	1
5	21	GPIO.21	IN	1	29 || 30			0v		
6	22	GPIO.22	IN	1	31 || 32	0	IN	GPIO.26	26	12
13	23	GPIO.23	IN	0	33 || 34			0v		
19	24	GPIO.24	IN	0	35 || 36	0	IN	GPIO.27	27	16
26	25	GPIO.25	IN	0	37 || 38	0	IN	GPIO.28	28	20
		0v			39 || 40	0	IN	GPIO.29	29	21
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM

Lora/GPS HAT	RaspberryPi 2B
3.3V	3.3V
5V	5V
GND	GND
DIO0	GPIO7
RX	GPIO15/TX
TX	GPIO16/RX
RESET	GPIO0
NSS	GPIO6
MISO	GPIO13/MISO
MOSI	GPIO12/MOSI
SCK	GPIO14/SCLK

Figura 3. Pinagem do *Raspberry PI 3* e *LoRa Shield* [Dragino 2019]

2.3. Configuração

Servidor

O servidor será responsável por receber os pacotes durante a transmissão de dados. Através dele, é possível verificar as características da comunicação após o sinal percorrer grandes distâncias com parâmetros diferentes. As principais funções realizadas pelo servidor são:

1. Estabelecer parâmetros de comunicação
   Antes de escutar o canal o servidor determina quais fatores serão levados em consideração para a recepção do sinal. Esses parâmetros são relativos a frequência utilizada, taxa de codificação, fator de espalhamento, modulação entre outros. Dependendo de quais valores sejam, ondas emitidas fora dessa configuração não serão interpretadas, e portanto, serão ignoradas.
2. Escutar o canal
   Neste período o servidor verifica se o seu *buffer* recebeu algum conteúdo de acordo com os parâmetros determinados, e passa para o próximo estado quando recebe esse conteúdo.
3. Tratar pacotes identificados
   Ao identificar que seu *buffer* tem pacotes, o servidor manipula o seu conteúdo de acordo como foi programado. No nosso experimento, ele está responsável por identificar características do canal de acordo com as características da mensagem recebida, Tais como: tempo de emissão e recepção, qualidade do sinal recebida, relação sinal ruído, tamanho da mensagem e outros.

Código do Servidor(em linguagem Python):

```
#!/usr/bin/python

import sys, os

# Add path to pyRadioHeadiRF95 module
sys.path.append(os.path.dirname(__file__) + "/../")
import pyRadioHeadRF95 as radio
rf95 = radio.RF95()
rf95.init()
rf95.setTxPower(20, False)
rf95.setFrequency(915)
rf95.setSignalBandwidth(rf95.Bandwidth500KHZ)
rf95.setSpreadingFactor(rf95.SpreadingFactor12)
rf95.setCodingRate4(rf95.CodingRate4_5)

print ("StartUp Done!")
print ("Receiving...")

while True:
        if rf95.available():
```

```
                (sensorRead, l) = rf95.recv()
                msg = sensorRead
                if (len(msg) < 5):
                        print(int(msg[3]))
                        if(int(msg[3]) == 5):
                                rf95.setCodingRate4(rf95.
                                   CodingRate4_5)
                                print("Coding Rate = 4/5")
                        if(int(msg[3]) == 6):
                                rf95.setCodingRate4(rf95.
                                   CodingRate4_6)
                                print("Coding Rate = 4/6")
                        if(int(msg[3]) == 7):
                                rf95.setCodingRate4(rf95.
                                   CodingRate4_7)
                                print("Coding Rate = 4/7")
                        if(int(msg[3]) == 8):
                                rf95.setCodingRate4(rf95.
                                   CodingRate4_8)
                                print("Coding Rate = 4/8")
                else:
                        print(msg)
                sys.stdout.flush()
```

Cliente

Tem a tarefa de enviar mensagens para o servidor. As mensagens terão o conteúdo gerado por um sensor de *ph* da água conectado a um Arduíno Uno.

As principais funções realizadas pelo cliente são:

1. Estabelecer parâmetros da comunicação
   Nesta etapa o cliente cria um canal de comunicação e estabelece parâmetros para o servidor interpretar as mensagem. Os parâmetros variam de acordo com o roteiro para a nossa equipe.
2. Receber o valor do sensor de *ph*
   O sensor de *ph* está constantemente retornando valores para o Arduíno, e portanto, é possível ler o pino de entrada do sensor e manipular os seus valores.
3. Receber o valor do relógio
   Além do sensor de *ph* existe um sensor que captura a hora atual [Oliveira 2019]. Com ele é possível determinar o tempo de emissão do pacote.
4. Gerar e enviar mensagem
   Utilizando os valores de *ph* e o tempo dos sensores, é formada a mensagem a ser enviada para o servidor. Nesse experimento o valor da mensagem segue o seguinte formato: id-value-time-conf.
   - Id: um número que identifica o pacote. Com ele é possível determinar a ordem de recepção de cada pacote.
   - Value: valor do sensor de *ph*.
   - Time: tempo de emissão da mensagem.

- Conf: um número de 1 a 5 o qual identifica o grupo de configuração que o pacote pertence. As configurações variam de acordo com os parâmetros passados no roteiro do trabalho para nossa equipe.

5. Alterar parâmetros
   Ao decorrer da comunicação é necessário alterar os valores possíveis para o *Coding Rate* das mensagens. Portanto, o cliente é responsável por enviar mensagens de controle para que o servidor identifique a troca de parâmetros.

Código do Cliente(em Arduino):

```
#include <RH_RF95.h>
#include<DS3232RTC.h>
#include<Streaming.h>
RH_RF95 rf95;
DS3232RTC omnitrix;
#define DEBUG true
void setup(){
  Serial.begin(9600);
  //tempo stuff
  setSyncProvider(RTC.get);
  if(timeStatus()!=timeSet){
    Serial.println("FAIL!");
  }
  Serial.println("Setup finished");
  // end time stuff
  // Lora Stuff
  while(!Serial); //Wait for serial port to be avaliable
  if(!rf95.init())
    Serial.println("RF95 init failed");
  rf95.setTxPower(20,false); //seta potencia de transmissao
  rf95.setFrequency(915.0);
  rf95.setSignalBandwidth(500000) ;
  rf95.setSpreadingFactor(12) ;
  //end Lora Stuff
}

void loop(){
  int valor_do_sensor;
  String tempo;
  int parametro;
  for(parametro = 5; parametro < 9; parametro++){
    if(parametro == 5) {rf95.setCodingRate4(5);}
    if(parametro == 6) {rf95.setCodingRate4(6);}
    if(parametro == 7) {rf95.setCodingRate4(7);}
    if(parametro == 8) {rf95.setCodingRate4(8);}
    //Mensagem de controle
    for(int controle = 0; controle < 20; controle++){
     //   delay(500);
      String control_mensage = ("CR:"+String(parametro));
      Serial.println(control_mensage);
```

```
      uint8_t valueToSend[control_mensage.length()+1];
      control_mensage.toCharArray(valueToSend,control_mensage.
         length()+1);
      rf95.send ( valueToSend ,sizeof(valueToSend));
      rf95.waitPacketSent();
    }
    //Pacotes enviados
    for(int id = 0; id < 100; id++){
      delay(100);
      valor_do_sensor = getSensorValue().toInt();
      tempo = pegaTempo();
      String payload = (String(id) + '-' + String(
         valor_do_sensor) + '-' + String(tempo) + '-' + String(
         parametro));
      //String payload = "teste";
      Serial.println(payload);
      uint8_t valueToSend[payload.length()+1];
      payload.toCharArray(valueToSend,payload.length()+1);
      rf95.send ( valueToSend ,sizeof(valueToSend));
      rf95.waitPacketSent();
    }
  }
  //Reset
  parametro = 5;
  delay(1000);
}

String getTime(){
  time_t t = omnitrix.get();
}

String  getSensorValue(){
  float sensorValue = analogRead(0);
  String aux = String(sensorValue,1);
  return aux;
}

String pegaTempo(){
  time_t t;
  t = now();
  String horario = "";
  horario.concat(printI00 (hour(t),':')) ;
  horario.concat(printI00 (minute(t),':'));
  horario.concat(printI00 (second(t),"")) ;
  return horario;
}

String printI00 ( int val , char delim ){
  String t_unit = "";
```

```
  if (val < 10 ) t_unit.concat('0') ;
  int decimal = val;
  t_unit.concat(String(decimal)) ;
  if ( delim>0) t_unit.concat(delim) ;
  return t_unit;
}
```

3. Considerações sobre os parâmetros RC

Para melhorar a robustez do link, o *LoRa* emprega código de erro cíclico para executar o erro de encaminhamento de detecção e correção. Essa configuração de codificação incorre em um *transmission overhead*. O Overhead adicional de dados por transmissão resultante é mostrado abaixo:

CodingRate **(RegTxCfg1)**	**Cyclic Coding Rate**	**Overhead Ratio**
1	4/5	1.25
2	4/6	1.5
3	4/7	1.75
4	4/8	2

Figura 4. Cyclic Coding Overhead [HOPE MICROELECTRONICS CO. 2006]

A correção direta de erros é eficiente na melhoria da confiabilidade do link na presença de interferência. De modo que a taxa de codificação (e, portanto, robustez à interferência) possa ser alterada em resposta às condições do canal. A taxa de codificação pode ser incluída, opcionalmente, no cabeçalho do pacote para uso do receptor.

A modulação LoRa também adiciona uma correção de erro direta (FEC) em todas as transmissões de dados. Essa implementação é feita codificando dados de 4 bits com redundâncias em 5 bits, 6 bits, 7 bits ou até 8 bits. O uso dessa redundância permitirá que o sinal LoRa suporte interferências curtas. O valor da taxa de codificação (CR) precisa ser ajustado de acordo com as condições do canal usado para transmissão de dados. Se houver muita interferência no canal, é recomendável aumentar o valor de CR. Contudo, o aumento do valor de CR também aumentará a duração da transmissão.

4. Resultados

O procedimento para a apuração dos pacotes enviados foi feita dentro do ambiente da UFAM. Utilizamos o corredor que se estende do estacionamendo da Faculdade de Tecnologia e o estacionamento do Instituto de Filosofia, Ciências Humanas e Sociais da Universidade Federal do Amazonas mostrado na figura a seguir:

Figura 5. Cyclic Coding Overhead [HOPE MICROELECTRONICS CO. 2006]

Foram feitas medições das taxas de erros em 5 distâncias diferentes variando em aproximadamente 80 metros entre o ponto de origem e o ponto final da reta. Os dados coletados estão na tabela abaixo:

Distancia	Coding Rate	Perda de Pacote (%)	RSSI
80 m	4/5	58%	-85 db
	4/6	56%	
	4/7	54%	
	4/8	59%	
160 m	4/5	64%	-88 db
	4/6	57%	
	4/7	51%	
	4/8	47%	
240 m	4/5	64%	-82 db
	4/6	57%	
	4/7	59%	
	4/8	57%	
320 m	4/5	70%	-90 db
	4/6	62%	
	4/7	91%	
	4/8	65%	
400 m	4/5	?	-95 db
	4/6	82%	
	4/7	76%	
	4/8	?	

Para capturar o RSSI foi utilizado o RFExplorer ao lado do dispositivo receptor.

Figura 6. RFExplorer e Raspiberry utilizados durante o experimento

5. Conclusão

Com os resultados obtidos é possível perceber que quanto mair a distancia do receptor do emissor maior é a taxa de perda de pacotes e a Potência do sinal recebido se degrada. A variação do Coding Rate quanto maior for menor será a perda de pacotes. Porém isso influencia na demora da transmissão do sinal pelo ambiente devido a maior quantidade de informação colocada na onda eletromagnética. Esses dados são colocados no cabeçalho do pacote no campo chamado ”Payload CR”o qual contém informação redundante sobre a mensagem para o tratamento de erros quando for recebido. A quantidade de bits desse campo varia de acordo com o parametro definido na comunicação.

6. Problemas durante o experimento

6.1. *LoRa Shield* não reconhecido

Quando começamos a verificar as configurações iniciais para conectar o *raspberry pi* com o LoRa shield tivemos diversos erros de reconhecimento. No inicio, seguimos os procedimentos descritos por [de Oliveira 2019], porém ele se baseava em uma versão diferente do Lora e por isso gerava erros de reconhecimento do hardware. Devido não acharmos uma documentação que especificasse a pinagem correta para a versão especifica do hardware do Lora da nossa equipe com um *raspberry pi*, tivemos que procurar a pinagem no próprio código fonte da biblioteca RadioHead para *raspberry* e resolvemos o problema.

6.2. *RadioHead* não reconhece pacotes (biblioteca bcm)

Durante o período que utilizamos a versão antiga da biblioteca [Suhanko 2019], nós não conseguimos capturar pacotes pelo servidor. Isso aconteceu devido as funções específicas da biblioteca e contornamos a situação optando por não usa-las. Comentamos parte do código o qual gerava falhas na execução do programa e escolhemos outras funções similares para substituir as antigas.

6.3. Largura de Banda utilizada não pode ser setada

A biblioteca *RadioHead* [McCauley 2019] é então utilizada nos permite escolher apenas 4 opções de parâmetros pré-estabelecidos, desse modo, não atende aos requisitos do experimento. Quando descobrimos isso, mudamos a versão do *RadioHead* utilizada para a mais atual, a qual possibilita a definição de parâmetros de modulação mais flexíveis e precisos.

6.4. LED Travando

Durante a execução do programa do servidor, o *script* pisca o LED para sinalizar o recebimentos de pacotes. Por algum motivo (problema no hardware) o *Lora shield* trava com o LED ligado e não para de executar a função de desligar o LED. Portanto, optamos por não utilizar funções que envolvam os LED's.

6.5. SSH não funciona

Para fazer o experimento, decidimos usar um laptop para controlar o *raspberry* a distância. Assim poderíamos visualizar os pacotes pelo display do notebook e facilitaria a execução do experimento. Porém, ao estabelecer uma conexão *ad-hoc* entre o computador e o *raspberry* os dispositivos tendem a desconectar, o que impossibilitou a utilização de SSH para controle remoto.

6.6. Servidor para de receber pacotes

Após todos esses problemas conseguimos estabelecer comunicação entre o cliente e servidor, porém, após alguns pacotes recebidos o servidor para de identificar novos pacotes transmitidos pelo cliente. Achamos ter relação com a falha no envio, mas ao reiniciar o servidor a captura de pacotes acontecia novamente, até travar logo em seguida.

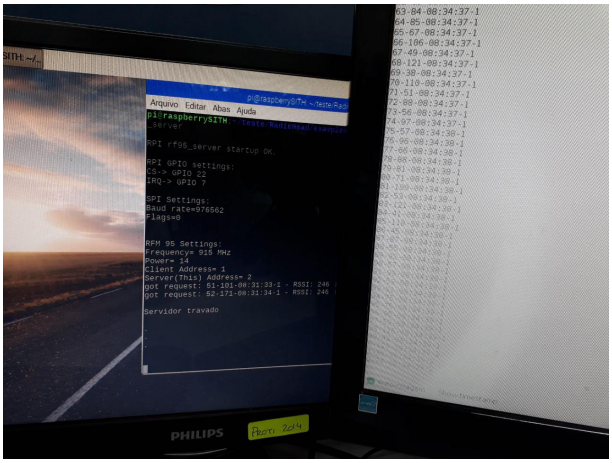

Figura 7. A direita o servidor travado e na direita o cliente enviando mensagens

Achamos ter relação com o buffer que não esvaziava ao receber novos pacotes e utilizamos uma função específica para limpar esse buffer, porém o problema não foi resolvido. Tentamos colocar um intervalo de tempo entre o recebimento de pacotes, isso acarreta em um tempo muito maior de recepção de pacotes, mas poderia resolver o problema, mas não tivemos êxito. Devido a esse impasse, não conseguimos estabelecer conexão por tempo suficiente para caracterizar o canal, e portanto, não fizemos o experimento completo.

Referências

de Oliveira, L. R. (2019). Setup LoRa com Arduino, Raspberry Pi e shield Dragino. Disponível em `https://www.embarcados.com.br/lora-arduino-raspberry-pi-shield-dragino/`. Acesso em 21/11/2019.

Dragino (2019). Lora/GPS HAT. Disponível em https://wiki.dragino.com/index.php?title=Lora/GPS_HAT. Acesso em 25/11/2019.

HOPE MICROELECTRONICS CO., L. (2006). RFM95/96/97/98(W) - Low Power Long Range Transceiver Module V1.0. Disponível em https://cdn.sparkfun.com/assets/learn_tutorials/8/0/4/RFM95_96_97_98W.pdf. Acesso em 25/11/2019.

McCauley, M. (2019). RadioHead . Disponível em https://www.airspayce.com/mikem/arduino/RadioHead/index.html. Acesso em 21/11/2019.

Oliveira, E. (2019). Como usar com Arduino – Módulo Real Time Clock RTC DS3231. Disponível em https://blogmasterwalkershop.com.br/arduino/como-usar-com-arduino-modulo-real-time-clock-rtc-ds3231/. Acesso em 21/11/2019.

Suhanko, D. (2019). LoRa com Raspberry e Arduino. Disponível em https://www.dobitaobyte.com.br/lora-com-raspberry-e-arduino/. Acesso em 21/11/2019.