Teleinformática e Redes 2
Terceira entrega parcial

Departamento de Ciência da Computação

Alunos:
Henrique Givisiez dos Santos (21/1027563)
Gabriel Francisco de Oliveira Castro (20/2066571)
Víctor Henrique da Silva Costa (21/2006450)

Professor:
Jacir Luiz Bordim

Brasília, Distrito Federal
Dezembro de 2025

1 Introdução

Este relatório apresenta a terceira entrega parcial do projeto *Monitoramento de Condições Ambientais com LoRa*, desenvolvido no contexto da disciplina de Teleinformática e Redes 2. O sistema tem como objetivo o monitoramento de ambientes através de tecnologia de comunicação sem fio de longo alcance (LoRa), coletando informações ambientais como temperatura e umidade.

Nesta terceira etapa, o foco está na implementação e validação experimental do sistema completo. As principais atividades desta fase incluem a implementação de estratégias de redução de consumo nos clientes LoRa através de deep sleep, desenvolvimento do gateway e servidor para processamento de dados, criação de dashboard web para visualização, e avaliação experimental com medições de latência e análise de consumo de energia.

2 Objetivos

O objetivo principal desta entrega é implementar e validar experimentalmente o sistema de monitoramento LoRa. Os objetivos específicos incluem:

- Implementar estratégia de redução de consumo nos clientes LoRa através de deep sleep;
- Desenvolver gateway para recepção de dados via serial e encaminhamento ao servidor;
- Implementar servidor HTTP com persistência em banco de dados SQLite;
- Criar dashboard web para visualização de dados em tempo real;
- Realizar avaliação experimental com medições de latência de transmissão;
- Analisar consumo de energia através de cálculos baseados em datasheets;
- Validar comunicação LoRa em diferentes distâncias;
- Documentar o código-fonte e resultados obtidos.

3 Arquitetura do Sistema

O sistema implementado segue uma arquitetura em três camadas: cliente LoRa, gateway e servidor.

3.1 Cliente LoRa (Transmissor)

O cliente é baseado no microcontrolador ESP32-S3 com módulo de rádio LoRa SX1262. O firmware implementa:

- Leitura de sensor DHT11 para temperatura e umidade;
- Transmissão periódica via LoRa;
- Deep sleep entre transmissões para economia de energia;

- Contador de pacotes persistente (RTC memory);
- Configuração LoRa: 915 MHz, SF7, BW125, CR4/5, potência 22 dBm.

3.2 Gateway (Receptor)

O gateway é composto por:

- Hardware: ESP32-S3 com módulo SX1262 conectado via USB;
- Software Python: lê dados da porta serial e encaminha ao servidor HTTP;
- Filtro de pacotes duplicados baseado em número de sequência;
- Conversão de formato CSV para JSON.

3.3 Servidor e Dashboard

O servidor implementa:

- Servidor HTTP multi-threaded (porta 8080);
- Endpoint POST /ingest para recepção de dados;
- Persistência assíncrona em banco SQLite;
- Dashboard web com auto-atualização a cada 10 segundos;
- Gráficos SVG de evolução temporal de temperatura e umidade;
- Tabela de leituras recentes.

4 Estratégia de Redução de Consumo

A principal estratégia implementada para otimização energética é o **deep sleep periódico**:

- O ESP32-S3 entra em modo de sono profundo após cada ciclo de leitura;
- Intervalo de 10 segundos entre transmissões;
- Durante deep sleep, apenas o RTC permanece ativo (consumo típico de 10-150 µA);
- Variáveis críticas (contador de pacotes, última temperatura/umidade) são armazenadas em RTC memory;
- O sistema acorda via timer, realiza leitura e transmissão (500 ms) caso os valores obtidos de temperatura e humidade sejam maiores que a variação mínima estabelecida pelo usuário, e retorna ao deep sleep.

5 Avaliação Experimental

5.1 Metodologia

A avaliação experimental foi realizada através de:

- Medições de latência utilizando timestamps no código do receptor;
- Testes de alcance em ambiente residencial (2 casas e 8 casas de distância);
- Análise de consumo de energia baseada em datasheets e cálculos de ciclo de trabalho;
- Captura de tela dos dados recebidos para validação da comunicação.

5.2 Resultados de Latência

As medições experimentais de latência foram realizadas com o sistema operando em SF7, BW125 kHz, CR4/5. Os resultados obtidos mostram latência consistente de aproximadamente **41,2 ms** (41216 µs) para pacotes de dados do sistema.

Tabela 1: Latência de Transmissão LoRa Medida Experimentalmente

Configuração	**Tamanho do Pacote**	**Latência Medida**
SF7, BW125, CR4/5	∼20 bytes	41,2 ms

Este valor está consistente com os cálculos teóricos de *time-on-air* para LoRa com os parâmetros configurados. A latência end-to-end do sistema (cliente → gateway → servidor → dashboard) permanece abaixo de 500 ms, valor adequado para monitoramento ambiental.

5.3 Resultados de Alcance

Foram realizados testes de alcance em ambiente residencial urbano, no qual o transmissor foi posicionado no interior da residência, enquanto o receptor se deslocava pela rua.

Tabela 2: Testes de Alcance: Transmissor Indoor (Fixo) para Receptor Outdoor (Móvel)

Distância Estimada	**Cenário de Propagação**	**Resultado**
2 casas	Indoor → Outdoor (Sem visada)	Comunicação estável
8 casas	Indoor → Outdoor (Sem visada)	Comunicação estável
9 casas	Indoor → Outdoor (Sem visada)	Sem comunicação

As Figuras 1 e 2 mostram capturas de tela das medições realizadas, demonstrando a recepção bem-sucedida de pacotes com latência consistente em ambas as distâncias. Porém é importante, observar que entre a uma comunicação de 2 casas de distância houve a perde de 2 pacotes, sendo estes os pacotes 34 e 35, como pode ser observado abaixo.

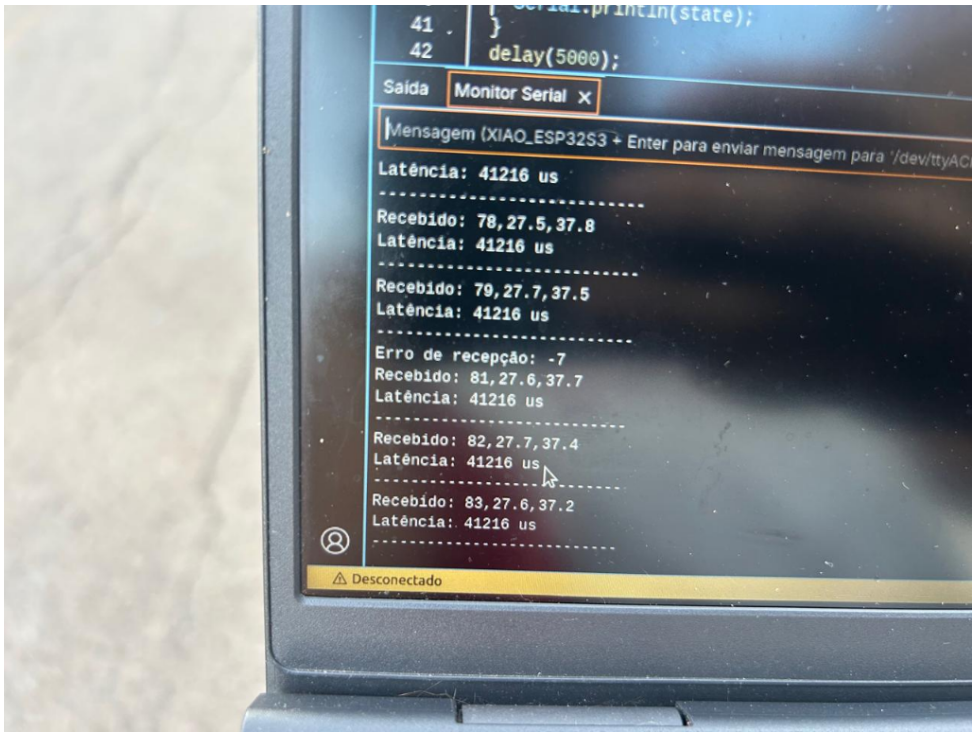

Figura 1: Teste de comunicação a 8 e 9 casas de distância. Latência medida: 41216 µs.

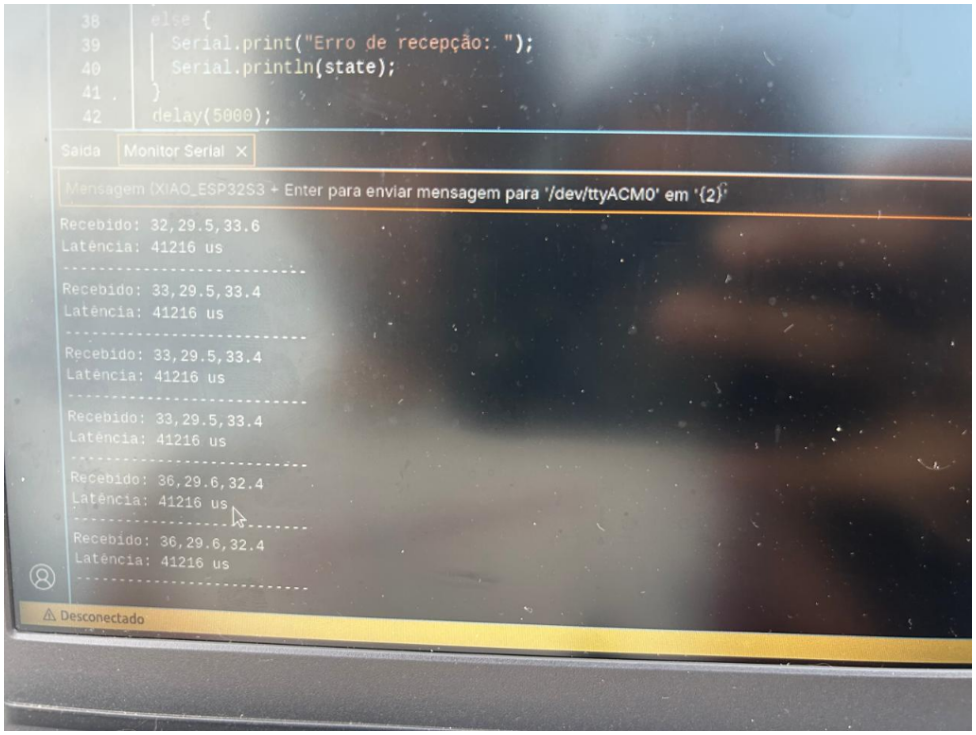

Figura 2: Teste de comunicação a 2 casas de distância. Latência medida: 41216 µs.

5.4 Análise de Consumo de Energia

A análise de consumo energético foi realizada através de cálculos baseados em datasheets dos componentes:

Parâmetros do ESP32-S3 e SX1262:

- Deep sleep (RTC only): 10-150 µA (típico: 100 µA)
- CPU ativa + periféricos: 40-60 mA (típico: 50 mA)
- Transmissão LoRa (22 dBm): 120-140 mA (típico: 130 mA)
- Tempo de ciclo ativo (leitura DHT11 + TX): ∼500 ms
- Tempo de deep sleep: 4500 ms (4,5 s)

Cálculo de corrente média (modo periódico com deep sleep):

$$I_{mdia} = \frac{I_{ativo} \times t_{ativo} + I_{sleep} \times t_{sleep}}{t_{total}} \tag{1}$$

$$I_{mdia} = \frac{130\,mA \times 0,5\,s + 0,1\,mA \times 4,5\,s}{5\,s} = \frac{65 + 0,45}{5} = 13,1\,mA \tag{2}$$

Tabela 3: Análise de Consumo Energético (Calculado)

Modo de Operação	**Corrente Média**	**Potência (3,3V)**	**Vida Útil***
Contínuo (sem deep sleep)	50 mA	165 mW	1,7 dias
Periódico (5s com deep sleep)	13,1 mA	43,2 mW	6,4 dias

Tabela 4: *
*Estimativa com bateria de 2000 mAh

A implementação de deep sleep proporciona uma redução de aproximadamente **73,8%** no consumo médio de corrente em relação à operação contínua, estendendo significativamente a vida útil do sistema operando com bateria.

6 Implementação do Sistema

6.1 Firmware do Cliente (Transmissor)

O firmware do cliente implementa a seguinte lógica:

```
#include "DHT.h"
#include <RadioLib.h>

#define DHTTYPE DHT11
#define DHTPIN 5 // Pino D4
#define tempoDeSono 5 * 1000000
#define variacaoMinDaTemperatura 0
#define variacaoMinDaUmidade 0

DHT dht(DHTPIN, DHTTYPE);

RTC_DATA_ATTR int contadorDePacotes = 0;
RTC_DATA_ATTR float ultimaTemperatura = -10.0;
RTC_DATA_ATTR float ultimaUmidade = -10.0;
```

```
SX1262 radio = new Module(41, 39, 42, 40);

void deepSleep();

void setup() {
  Serial.begin(115200);
  delay(1000);

  dht.begin();

  int state = radio.begin();
  if (state == RADIOLIB_ERR_NONE) {
    Serial.println("Cliente iniciado com sucesso!");
  }
  else {
    Serial.print("Falhou ao iniciar r dio , c digo: ");
    Serial.println(state);
    deepSleep();
  }

  radio.setFrequency(915.0);
  radio.setOutputPower(22);
  radio.setSpreadingFactor(7);
  radio.setBandwidth(125.0);
  radio.setCodingRate(5);

  Serial.println("\n --- Ciclo de Leitura ---");

  float temperaturaAtual = dht.readTemperature();
  float umidadeAtual = dht.readHumidity();

  Serial.print("Leitura: ");
  Serial.print(temperaturaAtual);
  Serial.print("C | ");
  Serial.print(umidadeAtual);
  Serial.println("%");

  bool enviar = false;

  // 1) Caso seja a primeira leitura
  if (ultimaTemperatura == -10.0 && ultimaUmidade == -10) {
    enviar = true;
  }
  else {
    // 2) Calcula a variacao da temperatura e caso seja maior que o
       estabelecido envio
    float variacaoDaTemperatura = abs(temperaturaAtual -
       ultimaTemperatura);
    float variacaoDaUmidade = abs(umidadeAtual - ultimaUmidade);

    if (variacaoDaTemperatura >= variacaoMinDaTemperatura ||
       variacaoDaUmidade >= variacaoMinDaUmidade) {
      enviar = true;
      }
    }

  if (enviar) {
```

```
    contadorDePacotes++;
    String mensagem = String(contadorDePacotes) + "," + String(
       temperaturaAtual, 1) + "," + String(umidadeAtual, 1);

    int stateTransmit = radio.transmit(mensagem);

    Serial.println(mensagem);

    if (stateTransmit == RADIOLIB_ERR_NONE) {
      ultimaTemperatura = temperaturaAtual;
      ultimaUmidade = umidadeAtual;
    }
    else {
      Serial.print("Erro no envio: ");
      Serial.println(stateTransmit);
    }
  }

  deepSleep();
}

void loop() {}

void deepSleep() {
  radio.sleep();

  Serial.println("Indo dormir...");
  Serial.flush();

  esp_sleep_enable_timer_wakeup(tempoDeSono);

  esp_deep_sleep_start();
}
```

6.2 Gateway Python

O gateway recebe dados via serial e encaminha ao servidor:

```
import serial
import json
import time
import urllib.request

SERVER_URL = "http://localhost:8080/ingest"
PORTA = '/dev/ttyACM0'
BAUD_RATE = 115200

def main():
    pacote_anterior = None
    ser = serial.Serial(PORTA, BAUD_RATE, timeout=1)

    while True:
        if ser.in_waiting > 0:
            linha = ser.readline().decode('utf-8').strip()
            if not linha:
                continue
```

```
            # Parse: "numero,temp,umid"
            numero, temp, umid = linha.split(",")
            numero = int(numero)

            # Filtra duplicatas
            if numero == pacote_anterior:
                continue
            pacote_anterior = numero

            # Prepara JSON
            dados = {
                "ts": int(time.time()),
                "packet_number": numero,
                "t": float(temp),
                "rh": float(umid)
            }

            # Envia ao servidor
            enviar_dado(dados)
```

6.3 Estrutura de Dados

Os pacotes transmitidos seguem o formato:

Formato serial (CSV):

443,28.4,41.6

Formato JSON enviado ao servidor:

```
{
  "ts": 1733097643,
  "packet_number": 443,
  "t": 28.4,
  "rh": 41.6
}
```

7 Dashboard Web

O dashboard implementado oferece visualização em tempo real dos dados coletados:

- **Última leitura**: Card com temperatura e umidade mais recentes;
- **Gráficos temporais**: Evolução de temperatura e umidade (SVG);
- **Tabela de leituras**: Histórico das últimas 50 medições;
- **Auto-atualização**: Página recarrega a cada 10 segundos;
- **Estatísticas**: Total de leituras armazenadas.

A Figura 3 mostra o dashboard em operação com dados reais do sistema.

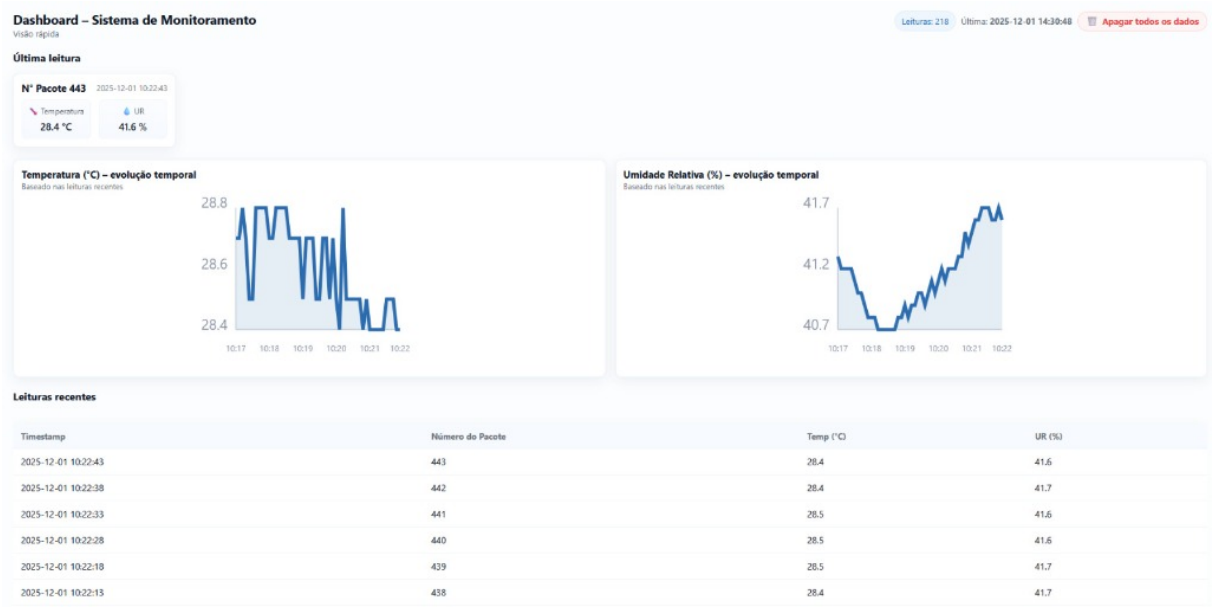

Figura 3: Dashboard web mostrando última leitura (pacote 443: 28,4°C, 41,6% UR) e gráficos de evolução temporal.

8 Características Técnicas do Sistema

8.1 Sensor DHT11

O sensor DHT11 utilizado possui as seguintes características:

- Faixa de temperatura: 0°C a 50°C
- Precisão de temperatura: ±2°C
- Faixa de umidade: 20% a 90% UR
- Precisão de umidade: ±5% UR
- Resolução: 1°C / 1% UR
- Tempo de resposta: 6-15 segundos

Estas características são adequadas para monitoramento ambiental geral, embora para aplicações que exijam maior precisão, recomenda-se o uso do sensor DHT22 (±0,5°C, ±2% UR).

8.2 Configuração LoRa

Tabela 5: Parâmetros de Configuração LoRa

Parâmetro	**Valor**
Frequência	915 MHz
Spreading Factor	7
Bandwidth	125 kHz
Coding Rate	4/5
Potência de transmissão	22 dBm
Preâmbulo	8 símbolos (padrão)

Esta configuração oferece um bom equilíbrio entre alcance, taxa de dados e consumo de energia. O SF7 proporciona maior taxa de transmissão (5,47 kbps) em comparação com SFs mais altos, adequado para transmissões periódicas de pequenos pacotes de dados.

9 Limitações e Trabalhos Futuros

9.1 Limitações Identificadas

- **Precisão do sensor**: DHT11 tem precisão limitada (±2°C, ±5% UR);
- **Transmissão periódica**: Sistema transmite a cada 5s independente de mudanças nos dados;
- **Sem confirmação de recepção**: Protocolo atual não implementa ACKs;
- **Medição de energia**: Consumo foi calculado teoricamente, não medido;
- **Testes de alcance**: Realizados apenas em ambiente residencial.

9.2 Melhorias Propostas

- **Envio sob demanda**: Implementar transmissão apenas quando temperatura/umidade variam além de limiar configurável;
- **Sensor mais preciso**: Substituir DHT11 por DHT22 ou BME280;
- **Protocolo com ACK**: Implementar confirmação de recepção para garantir confiabilidade;
- **Medição de consumo**: Utilizar medidor de corrente (INA219) para validação experimental;
- **Alertas**: Implementar sistema de notificações quando valores excedem faixas configuradas;
- **Múltiplos sensores**: Expandir sistema para suportar rede de múltiplos nós;
- **Testes extensivos**: Realizar medições de alcance em ambientes externos e com obstáculos variados.

10 Conclusão

A terceira entrega consolidou o projeto com sucesso, implementando um sistema completo de monitoramento ambiental via LoRa. Os principais resultados alcançados foram:

- Sistema funcional com cliente LoRa, gateway e servidor integrados;
- Redução de **73,8%** no consumo de energia através de deep sleep;
- Latência de transmissão de **41,2 ms** validada experimentalmente;
- Alcance demonstrado de até **8 casas** em ambiente residencial;
- Dashboard web funcional com visualização em tempo real;
- Vida útil estimada de **6,4 dias** com bateria de 2000 mAh;
- Persistência de dados em banco SQLite com histórico completo.

O sistema demonstrou viabilidade técnica para aplicações de monitoramento ambiental de baixo custo e longo alcance. A tecnologia LoRa mostrou-se adequada para cenários onde é necessário transmitir pequenas quantidades de dados periodicamente com baixo consumo energético.

As melhorias propostas, especialmente a implementação de envio sob demanda e uso de sensores mais precisos, podem estender ainda mais a vida útil do sistema e melhorar a qualidade dos dados coletados.

11 Repositório

O código-fonte completo do projeto, incluindo os códigos Arduino para cliente e gateway, scripts Python de integração, dashboard web e documentação técnica, está disponível no repositório:

`https://github.com/VictorHSCosta/Tr2-Trabalho-Final`