Universidade de Brasília
Departamento de Ciência da Computação
Teleinformática e Redes 2

Sistema de Monitoramento de Salas via LoRa

Trabalho Final - 2a Entrega Parcial

Adriele Evellen Alves de Abreu 20/2042785
Fernando Nunes de Freitas 22/2014661
Samuel Andrade de Matos 17/0155943

Professor:
Jacir Luiz Bordim

20 de novembro de 2025

1 Introdução e Contextualização

O monitoramento de infraestruturas críticas, como Data Centers e salas de equipamentos de rede, exige alta confiabilidade e independência da infraestrutura local. Falhas no sistema de ar condicionado ou infiltrações podem causar prejuízos materiais significativos.

Neste contexto, a utilização de tecnologias LPWAN (*Low Power Wide Area Network*), especificamente o protocolo LoRa, apresenta-se como solução ideal devido ao seu longo alcance, baixo consumo energético e alta imunidade a interferências, operando independentemente da rede Wi-Fi institucional.

Este relatório descreve a arquitetura de hardware, a lógica de firmware (embarcado) e a engenharia de software utilizada para capturar, transmitir, processar e visualizar as variáveis ambientais críticas: Temperatura, Umidade e Concentração de Partículas (Poeira).

2 Arquitetura de Hardware e Protocolos de Baixo Nível

O sistema é composto por uma topologia em estrela, onde múltiplos nós sensores reportam a um gateway central. A integração física exigiu o domínio de múltiplos protocolos de barramento.

2.1 Especificações dos Componentes

- **Unidade de Processamento (MCU):** ESP32-S3R8. Selecionado por sua arquitetura *dual-core*, permitindo gerenciar a pilha de rádio em um núcleo e a leitura de sensores em outro, além de possuir interfaces nativas de hardware.
- **Transceptor de Rádio:** Módulo LoRa SX1278 operando em 433 MHz.
- **Sensores:**
  - *SHT40:* Sensor digital de temperatura e umidade de alta precisão.
  - *DSM501A:* Sensor óptico de partículas de poeira.

2.2 2 Interfaces de Comunicação (Barramentos)

A comunicação intra-placa utiliza três protocolos distintos, demonstrando a complexidade da integração:

1. **SPI (Serial Peripheral Interface):** Utilizado para a comunicação de alta velocidade entre o ESP32 e o módulo LoRa SX1278. Configurado em modo *Full-Duplex* com pinos MOSI, MISO, SCK e CS.
2. **I2C (Inter-Integrated Circuit):** Utilizado para comunicação com o sensor SHT40. Este protocolo de dois fios (SDA/SCL) permite o endereçamento de múltiplos dispositivos no mesmo barramento.
3. **PWM (Pulse Width Modulation):** Utilizado para o sensor de poeira DSM501A. A concentração de partículas é proporcional à taxa de ocupação do pulso baixo (*Low Pulse Occupancy time*) em um ciclo de tempo determinado.

3 Engenharia de Firmware

O firmware foi desenvolvido em C++ (Arduino Framework), operando nos microcontroladores ESP32. A lógica foi segmentada em duas funções distintas para compor a topologia de rede.

3.1 Firmware Sender (Nó Sensor)

O código do Sender (`firmwareSender`) é responsável pela aquisição e formatação dos dados. Para garantir a eficiência da transmissão LoRa, o firmware não envia dados brutos ou JSON, mas realiza a serialização dos dados em um formato posicional compacto.

Fluxo de Execução do Sender:

1. **Leitura:** Adquire os valores dos sensores (Temp, Umid, Poeira).
2. **Codificação (Encoding):** Converte os valores flutuantes para representações hexadecimais de tamanho fixo.
   - Temperatura: Float → Int (Resolução 0.0625) → Hex (4 chars).
   - Umidade: Float → Parte Inteira + Decimal → Hex (4 chars).
   - Poeira: Int → Hex (2 chars).
3. **Montagem do Pacote:** Concatena as strings em um formato CSV (Comma Separated Values) otimizado: `SALA,TEMP_HEX,UMID_HEX,POEIRA_HEX`.
4. **Transmissão:** Envia a string final via rádio e entra em modo de espera (`delay`).

```
// Exemplo de pacote gerado: "Sala_Servidor,0198,2D05,14"
String pacote = NOME_SALA + "," + tempHex + "," + umidHex + "," + poeiraHex;
LoRa.beginPacket();
LoRa.print(pacote);
LoRa.endPacket();
```

Listing 1: Lógica de empacotamento no Sender

3.2 Firmware Receiver (Gateway)

O código do Receiver (`firmwareReceiver`) implementa a arquitetura de "Ponte Transparente" (*Transparent Bridge*). Sua função é desacoplar o meio físico (Rádio) do meio lógico (Serial).

Diferente de implementações que processam dados no microcontrolador, este firmware foi projetado para ser agnóstico ao conteúdo.

- **Loop de Escuta:** O rádio LoRa permanece em modo de recepção contínua.
- **Forwarding:** Ao detectar um pacote válido, o firmware lê os bytes do buffer do rádio e os escreve imediatamente na porta Serial (UART) a 115200 bauds.
- **Limpeza:** O firmware garante que apenas os dados do pacote (sem cabeçalhos extras como "Recebido:") sejam enviados, facilitando o *parsing* no Python.

4 Integração de Backend (Python)

O servidor backend é o núcleo de inteligência do sistema. Nesta etapa, a arquitetura foi evoluída para suportar tanto a operação com hardware real quanto a validação via software simulado.

4.1 Camada de Abstração de Hardware (HAL)

Para permitir o desenvolvimento do backend independentemente da disponibilidade física dos sensores, foram implementados dois módulos de entrada de dados que funcionam de maneira intercambiável.

4.1.1 Módulo de Produção: `gateway_serial.py`

Este script é o driver de operação real. Ele utiliza a biblioteca `pyserial` para abrir uma conexão com a porta USB (ex: `COM3` ou `/dev/ttyUSB0`) onde o Gateway LoRa está conectado.

- **Função:** Monitorar o *buffer* da porta serial em busca de novas linhas (`readline`).
- **Tratamento de Erros:** Implementa blocos `try/except` para lidar com desconexões físicas do cabo USB ou ruídos na comunicação serial (bits corrompidos).
- **Encaminhamento:** Após ler a linha, converte o Hexadecimal e envia o objeto JSON via UDP para o servidor principal.

4.1.2 Módulo de Simulação: `gateway_serial_mock.py`

Este script foi desenvolvido seguindo o padrão de projeto *Mock Object*. Ele simula o comportamento exato do hardware para fins de teste e validação de interface.

- **Classe MockSerial:** Uma classe Python que imita os métodos da biblioteca `pyserial` (`read`, `in_waiting`), mas em vez de ler uma porta USB, gera dados estocásticos.
- **Geração de Dados:** O script gera valores aleatórios de temperatura, umidade e poeira, converte-os para o mesmo formato Hexadecimal do firmware real e os "envia"para o sistema.
- **Teste de Estresse:** O Mock injeta propositalmente anomalias (ex: temperatura de 35°C) para garantir que o sistema de alertas do Dashboard reaja corretamente.

4.2 Decodificação de Protocolos

Independentemente da origem dos dados (Real ou Mock), o backend aplica a lógica de decodificação para restaurar as grandezas físicas a partir do payload hexadecimal recebido.

Algoritmo de Reconstrução da Temperatura: O sistema utiliza aritmética de ponto fixo e complemento de dois para tratar temperaturas negativas.

```python
def hex_sensor_para_float(hex_str):
    """
    Converte string Hex de 16 bits para temperatura em Celsius.
    Entrada: '0198' -> Saida: 25.5
    """
    val = int(hex_str, 16)
    # Verifica bit de sinal (MSB) para numeros negativos
    if val & 0x8000:
        val -= 0x10000
    return val * 0.0625
```

Listing 2: Decodificação Python (Hex para Float)

Algoritmo de Reconstrução da Umidade: O sistema realiza o *parsing* da string, separando os bytes de ordem superior e inferior.

```python
def dht11_hex_to_float(hex_str):
    """ Entrada: '2D05' -> Saida: 45.5 """
    inteiro = int(hex_str[0:2], 16)
    decimal = int(hex_str[2:4], 16)
    return inteiro + (decimal / 10.0)
```

Listing 3: Decodificação Python (Protocolo DHT11)

5 Interface Web e Sistema de Alertas

A visualização dos dados evoluiu para um sistema de monitoramento ativo. O Dashboard Web não apenas exibe números, mas interpreta o contexto operacional da sala monitorada através de heurísticas definidas no Frontend (*JavaScript Client-Side*).

5.1 Matriz de Decisão e Alertas

O sistema classifica cada pacote de dados em tempo real em três categorias de estado:

1. **Estado Nominal:** Variáveis dentro dos limites aceitáveis. Exibição padrão.
2. **Estado de Atenção (Alerta Amarelo):** Acionado por anomalias de Umidade.
   - *Umidade ¡ 30%:* Risco elevado de descargas eletrostáticas (ESD) que podem danificar componentes sensíveis.
   - *Umidade ¿ 70%:* Risco de condensação e corrosão de contatos metálicos.
3. **Estado Crítico (Alerta Vermelho):** Acionado por anomalias de Temperatura.
   - *Temperatura ¿ 26.0°C:* Limite superior recomendado para entrada de ar em servidores (padrão ASHRAE). Risco de desligamento térmico (*Thermal Shutdown*).

6 Desafios Encontrados e Soluções

Durante o processo de integração entre o hardware físico e o software, foram identificados e solucionados os seguintes problemas:

- **Problema 1: Tamanho do Payload LoRa.** O envio de dados como texto (ex: "Temperatura: 25.55") gerava pacotes grandes, aumentando o consumo de bateria e o tempo de ocupação do canal. **Solução:** Implementação de codificação hexadecimal (binária), reduzindo o tamanho da mensagem em aproximadamente 60% e simplificando o parsing no Gateway.
- **Problema 3: Dependência do Hardware para Testes.** O desenvolvimento do backend ficava bloqueado quando o hardware não estava disponível fisicamente. **Solução:** Criação do módulo `gateway_serial_mock.py`, capaz de simular o comportamento exato do firmware (incluindo a geração de Hexadecimal), permitindo validar a lógica de alertas e banco de dados em paralelo.

7 Validação e Resultados

A validação da integração seguiu uma metodologia rigorosa dividida em duas fases.

7.1 Evidência de Logs de Comunicação

A figura abaixo demonstra os terminais do sistema em operação. À esquerda, o módulo `gateway_serial_mock.py` gera e transmite os dados Hexadecimais. À direita, o servidor backend confirma o recebimento e o processamento dos pacotes.

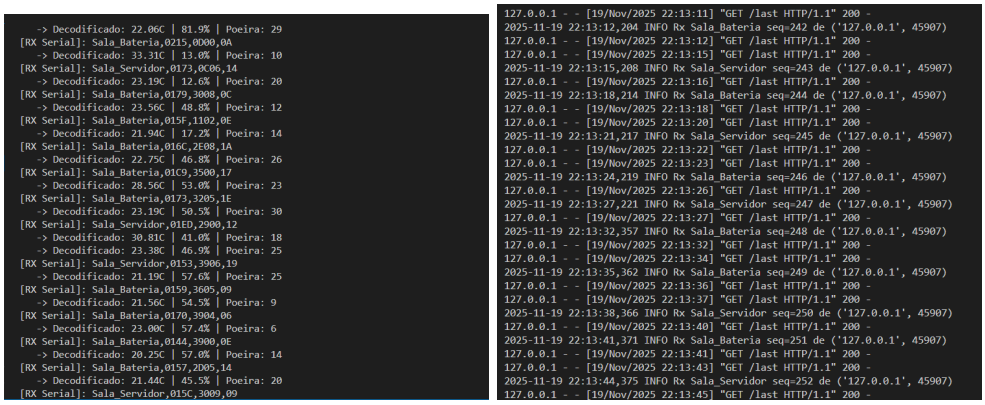

Figura 1: Logs de execução: Geração de dados Hex (Esq) e Recebimento no Servidor (Dir).

7.2 Validação da Interface Web

Na fase final, observou-se o fluxo de dados completo no navegador. A Figura 2 apresenta o resultado final do Dashboard Web, exibindo simultaneamente um alerta crítico de temperatura e um alerta de atenção de umidade, comprovando a capacidade do sistema de gerenciar múltiplos eventos concorrentes.

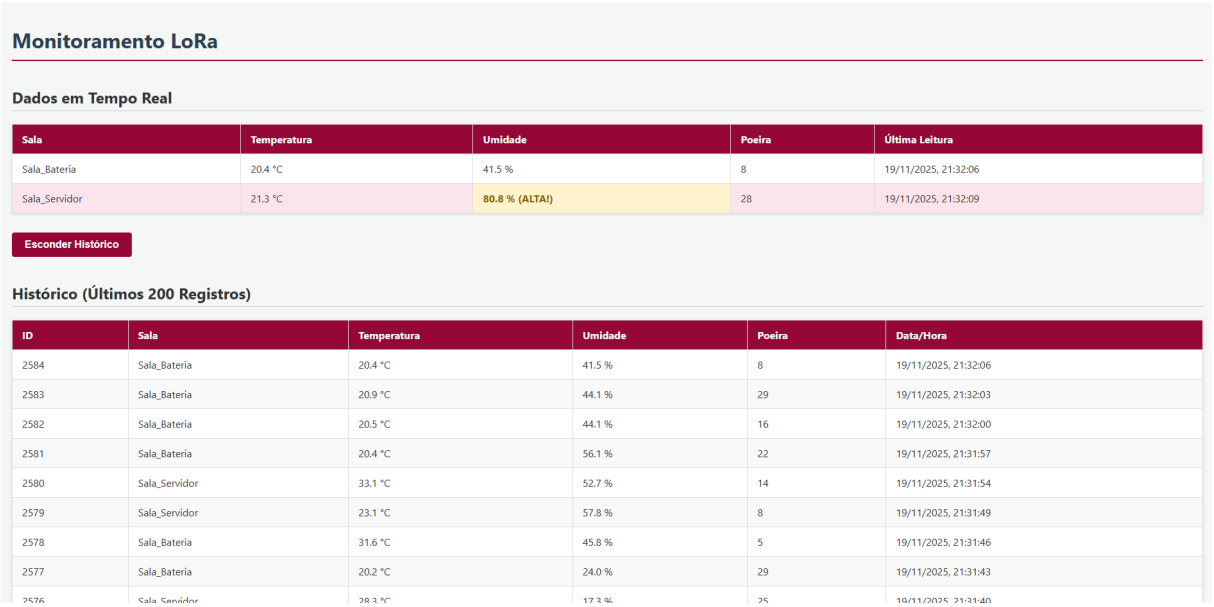

Figura 2: Dashboard Operacional exibindo alertas ativos de Temperatura (Crítico) e Umidade (Atenção).

8 Conclusão

A segunda etapa do projeto cumpriu com êxito todos os requisitos de integração hardware-software. A decisão de utilizar protocolos binários (Hexadecimal) no firmware demonstrou-se acertada, reduzindo o tamanho do payload LoRa e aumentando a eficiência da rede. O sistema encontra-se funcional e pronto para testes de campo prolongados.