DOCUMENTAÇÃO TÉCNICA: SISTEMA DE RASTREAMENTO E BILHETAGEM IOT (GSM/GPS/RFID)

1. VISÃO GERAL DO PROJETO

Este documento descreve o firmware desenvolvido para um sistema embarcado de gestão de transporte (ônibus). O sistema possui duas funções principais:

1. **Rastreamento:** Monitoramento da localização geográfica do veículo em tempo real via GPS.
2. **Bilhetagem:** Leitura de cartões de passageiros via RFID e envio dos dados para validação.

A comunicação de dados é realizada via rede móvel (GPRS/2G) utilizando um modem SIM800L, garantindo conectividade em movimento sem dependência de redes Wi-Fi locais.

---

2. ARQUITETURA DE HARDWARE E BIBLIOTECAS

2.1 Hardware Alvo

- **Microcontrolador:** ESP32 (integrado na placa TTGO T-Call).
- **Modem GSM:** SIM800L (integrado, comunicação Serial).
- **Módulo GPS:** Genérico (comunicação Serial UART).
- **Leitor RFID:** MFRC522 (comunicação SPI).

2.2 Bibliotecas e Dependências

O código utiliza bibliotecas específicas para abstrair a complexidade dos periféricos:

- `TinyGsmClient.h` : Gerencia a comunicação AT com o modem SIM800.
- `TinyGPSPlus.h` : Realiza o parsing (interpretação) das sentenças NMEA vindas do GPS.
- `MFRC522.h` : Controla o leitor de cartões RFID.
- `ArduinoJson.h` : Formata os dados no padrão JSON para envio à API.
- `SPI.h` & `Wire.h` : Protocolos de comunicação base.

---

3. ANÁLISE DA LÓGICA E DECISÕES DE PROGRAMAÇÃO

3.1 Configurações de Rede e Servidor

O sistema define explicitamente as credenciais de APN (Access Point Name) da operadora Vivo.

Decisão de Design (IP Local vs. Rede Móvel): O código aponta para o servidor `192.168.16.121` .

> **Nota Crítica:** IPs que começam com `192.168...` são IPs locais (LAN). Um dispositivo conectado via 4G/GPRS (que está na "internet pública") **não conseguirá acessar** esse IP local, a menos que haja uma VPN configurada. Para produção, este IP deve ser substituído por um IP Público ou Domínio (DNS).

3.2 Gerenciamento de Hardware (Pinagem)

O código define macros (`#define`) para todos os pinos para facilitar a manutenção.

Conflito de Hardware (SPI vs Modem): Há um tratamento especial para o pino `23` .

- O pino 23 é usado para ligar o Modem (`MODEM_POWER_ON`).
- O barramento SPI padrão do ESP32 geralmente usa o pino 23 para MOSI.
- **Solução Lógica:** No `setup()` , o comando `SPI.begin(18, 19, 23, ...)` tenta forçar o mapeamento dos pinos SPI. Isso indica que o hardware físico exige essa configuração específica para que o RFID e o Modem coexistam.

3.3 Inicialização (Setup)

A função `setup` segue uma ordem estrita para garantir estabilidade:

1. **Power-Cycle do Modem:** O código realiza uma sequência de `HIGH` -> `LOW` -> `HIGH` no pino `MODEM_PWRKEY` . Isso simula o pressionar físico do botão de ligar, garantindo que o modem reinicie limpo.
2. **Conexão GPRS:** O sistema tenta conectar à internet imediatamente. Se falhar, ele apenas avisa, mas não trava o sistema (permitindo que o `loop` tente novamente).
3. **GPS e RFID:** Iniciados em interfaces separadas (Serial2 para GPS e SPI para RFID) para evitar colisão de dados.

3.4 O Loop Principal (Non-blocking)

A função `loop` foi projetada para não ser bloqueante (não usar `delay` longos), permitindo multitarefa simulada:

1. **Verificação de Conexão:** A cada ciclo, checa se o GPRS caiu (`!modem.isGprsConnected()`) e tenta reconectar automaticamente. Isso garante resiliência em

áreas de sombra de sinal.
2. **Polling de Sensores:** Chama `checkRFID()` e `readGPS()` sequencialmente.
3. **Manutenção:** Chama `modem.maintain()` para lidar com tarefas de fundo da rede GSM.

3.5 Lógica do RFID (`checkRFID`)

1. Verifica se há cartão presente.
2. Lê o UID (Identificador Único) do cartão.
3. **Tratamento de Dados:** Converte os bytes hexadecimais em uma `String` legível e maiúscula (ex: "E2A4F5...").
4. **Ação:** Serializa um objeto JSON contendo o UID e o ID do Ônibus e envia imediatamente para a rota `/v1/bus/fare`.

3.6 Lógica do GPS (`readGPS` e `sendGPSIfReady`)

Diferente do RFID (que envia dados por evento), o GPS envia dados baseados em tempo.

Decisão de Programação (Timer com `millis()`): O código utiliza a lógica: `if (millis() - lastGpsPost < 10000) return;`

- **Motivo:** Enviar dados GPS a cada milissegundo congestionaria a rede GPRS (que é lenta) e sobrecarregaria o servidor. Um intervalo de 10 segundos foi escolhido como ideal para rastreamento de ônibus.
- **Validação:** Só envia se `gps.location.isValid()` for verdadeiro, evitando enviar coordenadas (0,0) antes do GPS triangular os satélites.

3.7 Comunicação HTTP Manual (`sendPostGPRS`)

Esta é a parte mais complexa e customizada do código. Em vez de usar uma biblioteca `HTTPClient` de alto nível, o código constrói a requisição HTTP manualmente via comandos TCP brutos.

Por que essa decisão? A biblioteca `TinyGsm` trabalha como um fluxo de dados (*Stream*). Construir o cabeçalho HTTP manualmente (`POST ... HTTP/1.1`) oferece:

1. **Controle total:** Permite definir *Headers* específicos (`Content-Type: application/json`).
2. **Leveza:** Evita carregar bibliotecas HTTP pesadas que muitas vezes são incompatíveis com o cliente GSM.
3. **Compatibilidade:** Funciona diretamente sobre o Socket TCP aberto pelo modem SIM800L.

Fluxo da Função:

1. Verifica a conexão GPRS.
2. Abre conexão TCP (`client.connect`).
3. Envia linhas do cabeçalho HTTP (Método, Host, Tamanho do conteúdo, Connection Close).
4. Envia o corpo (JSON).
5. Aguarda resposta (com timeout de 10s) para evitar o travamento do código caso o servidor não responda.

---

4. PONTOS DE ATENÇÃO E MELHORIAS FUTURAS

1. **Segurança:** As credenciais e URLs estão fixas no código ("hardcoded"). Recomendação: mover para variáveis de ambiente ou arquivo de configuração.
2. **Gerenciamento de Memória:** O uso extensivo da classe `String` no Arduino pode causar fragmentação de memória em longos períodos de operação. Para sistemas 24/7, recomenda-se refatorar para usar `char arrays` (C-Strings).
3. **Tratamento de Erros HTTP:** Atualmente, se o servidor retornar erro 500 ou 404, o código apenas imprime no Serial. Uma melhoria seria implementar uma fila (buffer) para tentar reenviar os dados posteriormente.

5. CONCLUSÃO

O código apresenta uma solução robusta para rastreamento veicular utilizando hardware de baixo custo. A lógica demonstra maturidade ao lidar com reconexão automática de rede e temporização não bloqueante, essenciais para aplicações IoT móveis. A implementação manual do protocolo HTTP sobre TCP demonstra um controle avançado sobre o funcionamento do modem GSM.