Relatório Técnico: Sistema de Monitoramento de Falhas em Redes Elétricas

João Correia Costa
Matrícula: 2019029027

13 de Junho de 2025

1 Introdução

A estabilidade das redes de fornecimento de energia elétrica é um pilar fundamental para a sociedade moderna. Interrupções, mesmo que breves, podem desencadear uma cascata de falhas em serviços essenciais, desde hospitais a sistemas financeiros, resultando em prejuízos significativos. Inspirado por eventos reais de apagões em grande escala, este projeto, desenvolvido no âmbito da disciplina de Redes de Computadores, aborda o desafio de criar um sistema de monitoramento preditivo para estas infraestruturas críticas.

O objetivo central foi projetar, implementar e testar uma arquitetura de comunicação distribuída, em linguagem C e utilizando a interface de Sockets POSIX, que simula uma rede de sensores (Internet das Coisas - IoT) para a deteção de anomalias. A solução visa não apenas coletar dados, mas também fornecer um diagnóstico coeso através da colaboração entre múltiplos componentes de software, demonstrando na prática os desafios de consistência, concorrência e comunicação em sistemas de rede.

2 Arquitetura do Sistema

A arquitetura adotada segue o princípio da **separação de preocupações** (Separation of Concerns - SoC), uma decisão de projeto fundamental para garantir a modularidade, manutenibilidade e escalabilidade do sistema. Em vez de um único servidor monolítico, a lógica foi distribuída entre dois servidores especializados, cada um com uma responsabilidade única e bem definida.

2.1 Componentes e Responsabilidades

- **Sensor (Cliente):** Simula o dispositivo de campo. A sua única responsabilidade é interagir com o sistema, enviando os seus dados (localização e risco) e realizando consultas. Ele atua como o orquestrador do seu próprio registo, garantindo a sua presença em ambos os servidores.
- **Servidor de Localização (SL):** O "especialista em geografia"do sistema. A sua única responsabilidade é manter o mapeamento entre o ID de um sensor e a sua localização física. Foi designado como a autoridade para a criação de novos IDs de sensores, centralizando esta função para evitar colisões.
- **Servidor de Status (SS):** O "especialista em diagnóstico"do sistema. A sua responsabilidade é exclusivamente manter o registo do estado de risco (0 ou 1) de cada sensor.

2.2 Comunicação P2P e Fluxos de Dados

A comunicação entre o SL e o SS é realizada através de um canal TCP direto (Peer-to-Peer). Este canal não é utilizado para a sincronização constante, mas sim para **consultas assíncronas** sob demanda. Esta decisão de design evita o overhead de manter as bases de dados perfeitamente sincronizadas em tempo real, optando por uma abordagem mais leve onde um servidor consulta o outro apenas quando necessita de uma informação que não possui. Por exemplo, para responder ao comando `diagnose`, o

SS (que conhece os riscos) precisa de consultar o SL (que conhece as localizações), demonstrando uma colaboração P2P para compor uma resposta completa.

2.3 Modelo de Concorrência

A gestão da concorrência é um dos pilares deste projeto. Foi adotado um modelo de I/O não-bloqueante, centrado na chamada de sistema `select()`. Esta abordagem permite que um único processo servidor, sem o uso de múltiplas threads, monitore eficientemente um conjunto de descritores de ficheiro (sockets e a entrada padrão). O servidor fica "adormecido"até que haja atividade, e ao "acordar", itera sobre os sockets prontos, tratando cada evento de forma sequencial. Este modelo é altamente eficiente para aplicações de rede que são primariamente limitadas por I/O.

3 Protocolo de Comunicação

Para garantir uma comunicação estruturada, foi implementado um protocolo de aplicação textual sobre TCP. A escolha de um protocolo textual em vez de binário foi deliberada, visando facilitar a depuração e a inspeção manual das mensagens trocadas durante o desenvolvimento.

Todas as mensagens seguem um formato delimitado por dois pontos (":"), permitindo um parsing simples e robusto com funções como `strtok_r`.

`CÓDIGO:ID_SENSOR:PAYLOAD\n`

Um módulo dedicado, composto pelos ficheiros `common.c` e `common.h`, foi criado para encapsular esta lógica. Ele fornece uma abstração que permite que o resto da aplicação trabalhe com uma estrutura `ProtocolMessage`, em vez de manipular strings diretamente.

Tabela 1: Tabela de Códigos do Protocolo Implementado.

Código	Nome	Descrição Detalhada
	Mensagens de Controle	
20	REQ_CONN_PEER	Enviada por um servidor ao iniciar para se apresentar ao seu par.
21	RES_CONN_PEER	Resposta de confirmação da conexão P2P.
23	REQ_CONN_SEN	Pedido de um sensor para obter um ID. Enviado para o SL.
24	RES_CONN_SEN	Resposta do SL com o novo ID do sensor.
25	REQ_DISC_SEN	Pedido de um sensor para se desconectar da rede.
	Mensagens de Dados	
36	REQ_CHECK_ALERT	Pedido P2P do SS ao SL pela localização de um sensor em risco.
37	RES_CHECK_ALERT	Resposta P2P do SL para o SS com a localização.
38	REQ_SENS_LOC	Requisição de consulta ou atualização de localização.
39	RES_SENS_LOC	Resposta com a localização de um sensor.
40	REQ_SENS_STATUS	Requisição de consulta ou atualização de status.
41	RES_SENS_STATUS	Resposta com o status de risco de um sensor.
42	REQ_LOC_LIST	Pedido P2P do SS ao SL pela lista de sensores numa área.
43	RES_LOC_LIST	Resposta P2P do SL para o SS com a lista de IDs.
	Mensagens de Sistema e Erro	
0	MSG_OK	Confirmação genérica (Payload indica o sub-tipo).
100	P2P_SYNC_NEW_SENSOR	Comando do sensor para o SS se registar (Fase 2).
101	P2P_SYNC_ACK	Confirmação do SS para o sensor, completando o registo.
255	MSG_ERROR	Mensagem de erro (Payload indica o código do erro).

4 Estruturas de Dados

A gestão do estado do sistema é realizada através de estruturas de dados simples em memória, adequadas para a escala do projeto (até 15 clientes).

4.1 Bases de Dados em Memória

Para armazenar a informação dos sensores, foram utilizados arrays estáticos de estruturas, uma solução simples e eficiente para o número limitado de clientes.

```
// Estrutura para a base de dados do Servidor de Localizacao
typedef struct {
    char id[11];
    int location;
} SensorLocation;

// Estrutura para a base de dados do Servidor de Status
typedef struct {
    char id[11];
    int risk_detected;
} SensorStatus;

// Arrays que funcionam como as bases de dados
SensorLocation location_database[MAX_CLIENTS];
SensorStatus status_database[MAX_CLIENTS];
```

Listing 1: Estruturas de dados principais

4.2 Gestão de Consultas Assíncronas

O maior desafio da arquitetura P2P é lidar com consultas assíncronas. Quando o SS envia uma pergunta ao SL, ele não pode ficar bloqueado à espera da resposta. Para gerir isto, foi implementada a estrutura `PendingQuery`. Ela funciona como uma "lista de lembretes".

1. Quando o SS recebe um comando complexo de um cliente (ex: `diagnose`), ele cria uma entrada em `pending_queries`, guardando o tipo de consulta e o socket do cliente original.
2. Envia a consulta P2P para o SL.
3. Quando a resposta do SL chega, o SS percorre o array `pending_queries` para encontrar o pedido original, processa a informação e envia a resposta final para o cliente correto.

```
typedef struct {
    QueryType type; // O tipo de consulta (CHECK_FAILURE ou DIAGNOSE)
    int original_client_socket; // "De quem era a pergunta original?"
    char sensor_id[11];         // A que sensor se refere?
    char data[256];             // Dados extra para a resposta final
} PendingQuery;
```

Listing 2: Estrutura para consultas pendentes

5 Detalhes da Implementação

5.1 Implementação do Servidor

A lógica do servidor foi modularizada em funções com responsabilidades claras: `handle_new_connection`, `handle_client_command`, e `handle_peer_message`. Uma decisão de projeto crucial foi a gestão do *handshake* P2P. A primeira mensagem recebida num socket recém-aceite é analisada na função `handle_client_command`. Se o código for `REQ_CONN_PEER`, o servidor sabe que se trata do seu par. Ele então move o socket da lista de clientes para a variável dedicada `peer_socket` e responde com `RES_CONN_PEER`, estabelecendo o canal P2P. Todas as outras mensagens são tratadas como comandos de cliente normais.

5.2 Implementação do Sensor

O sensor foi desenhado para ser o orquestrador do seu próprio registo, uma decisão que se provou fundamental para a estabilidade do sistema.

Registo em Duas Fases

Para resolver a "condição de corrida"onde um sensor podia enviar um comando para o SS antes de este ter sido notificado do seu registo, foi implementado o seguinte fluxo:

1. **Fase 1 (Obtenção de ID):** O sensor envia `REQ_CONN_SEN` para o SL. Ao receber a resposta `RES_CONN_SEN` com o seu novo ID, ele guarda-o internamente.
2. **Fase 2 (Sincronização):** De imediato, o sensor envia uma mensagem de sincronização (`P2P_SYNC_NEW_SENSOR`) para o SS.
3. **Conclusão:** O sensor só se considera "totalmente registado"(`is_fully_registered = 1`) e apto a enviar outros comandos após receber a confirmação `P2P_SYNC_ACK` do SS.

Esta abordagem garante a consistência do estado em toda a arquitetura distribuída antes de o sistema se tornar totalmente operacional para o cliente.

6 Discussão Técnica

O desafio mais significativo do projeto foi, sem dúvida, a gestão da consistência de estado num ambiente distribuído. A tentativa inicial de fazer com que o SL notificasse o SS sobre novos sensores revelou uma clássica **condição de corrida**: o cliente podia receber a confirmação do SL e enviar um comando para o SS antes de a mensagem de notificação P2P chegar, resultando em erros de "sensor não encontrado". A solução de um registo em duas fases, orquestrado pelo cliente, eliminou esta ambiguidade, tornando a sequência de eventos explícita e fiável.

A gestão de consultas assíncronas P2P através do array `pending_queries` é uma solução funcional para a escala deste projeto. No entanto, ela tem limitações. Como o array é percorrido linearmente, o seu desempenho degradaria num sistema com um grande número de consultas concorrentes. Uma **melhoria futura** seria substituir este array por uma estrutura de dados mais eficiente, como uma tabela hash, usando o descritor do socket como chave para um acesso em tempo O(1).

Finalmente, o protocolo de erros, embora funcional, poderia ser expandido para fornecer feedback mais granular ao cliente, melhorando a robustez geral do sistema.

7 Conclusão

O projeto implementado desenvolveu e testou uma arquitetura distribuída funcional, composta por dois servidores especializados e múltiplos clientes, capaz de gerir o estado de uma rede de sensores e de responder a consultas que exigem colaboração P2P.

A utilização da interface de Sockets POSIX e do multiplexador de I/O `select()` permitiu criar uma solução concorrente e eficiente, que serve como demonstração prática dos desafios de programação de rede. Os problemas encontrados, principalmente relacionados com sincronização e consistência, foram superados com soluções de design que refletem práticas comuns em sistemas distribuídos, consolidando os conhecimentos teóricos e práticos da disciplina.