Documentação: Sistema de Monitoramento de Falhas em Redes Elétricas (v1.2)

Lucas Albano Olive Cruz - 2022036209

---

1. Introdução

Infraestruturas críticas modernas dependem, cada vez mais, de redes de comunicação para garantir sua operação segura e contínua. Essa crescente interdependência torna essencial a resiliência e o monitoramento em tempo real desses sistemas. Episódios recentes de interrupções no fornecimento de energia elétrica em países como Portugal e Espanha evidenciam a vulnerabilidade de redes elétricas que operam sem mecanismos adequados de supervisão e resposta automática. Nesse cenário, a Internet das Coisas (IoT), impulsionada pelo avanço de Redes de Sensores Sem Fio (WSNs) de baixo custo, surge como uma solução viável para a implementação de Smart Grids (i.e., redes elétricas inteligentes com capacidade de detecção e reação autônoma a falhas).

Este documento apresenta a arquitetura e a implementação de um protótipo de sistema distribuído para monitoramento de falhas em redes elétricas. O objetivo principal é desenvolver e testar uma infraestrutura de comunicação capaz de mitigar os efeitos de panes inesperadas. O sistema foi desenvolvido integralmente em linguagem C, utilizando a interface POSIX de sockets para comunicação via protocolo TCP. A solução é composta por dois servidores operando em modo peer-to-peer, um responsável pelo gerenciamento do status dos sensores e outro pela localização física dos dispositivos, além de múltiplos sensores clientes simulando os pontos de monitoramento da rede elétrica.

As seções a seguir descrevem em detalhe a arquitetura do sistema, o protocolo de comunicação implementado, as estruturas de dados utilizadas, os fluxos de interação entre os componentes e as decisões de projeto adotadas durante o desenvolvimento.

---

2. Arquitetura do Sistema

O sistema é fundamentado em uma arquitetura distribuída composta por três entidades principais: o Sensor, o Servidor de Status (SS) e o Servidor de Localização (SL). A comunicação entre os sensores e os servidores segue o modelo cliente-servidor, enquanto a comunicação entre os dois servidores é realizada por meio de uma conexão ponto a ponto (peer-to-peer).

- **Sensor**: Atua como o cliente do sistema. Cada sensor é responsável por gerar um identificador único de 10 dígitos, conectar-se simultaneamente aos servidores SS e SL e enviar requisições de acordo com os comandos recebidos do usuário. As funcionalidades incluem a verificação de status de falha, a consulta de localização de outros sensores e o diagnóstico de sensores em uma determinada área.

- **Servidor de Status (SS)**: Este servidor tem como responsabilidade primária gerenciar o estado de risco de cada sensor conectado, armazenando uma flag binária que indica "risco detectado" (1) ou "sem risco" (0). Ao receber uma consulta de um sensor com status de risco, o SS inicia uma comunicação com o SL para obter a localização da potencial falha, agindo como um orquestrador do fluxo de alerta.

- **Servidor de Localização (SL)**: A função deste servidor é manter um registro da localização geográfica de cada sensor, representada por um valor inteiro de 1 a 10. Ele responde a consultas de localização provenientes tanto do servidor SS (no contexto de um alerta) quanto diretamente dos sensores (para funcionalidades de localização e diagnóstico).

A comunicação entre todos os componentes é estabelecida sobre TCP, garantindo a entrega ordenada e confiável das mensagens. Ambos os servidores são projetados para gerenciar até 15 conexões de clientes simultaneamente, utilizando a chamada de sistema select() para multiplexação de E/S, o que permite o tratamento concorrente de requisições de múltiplos clientes, do servidor par e da entrada padrão (teclado) sem o uso de threads.

---

3. Estruturas de Dados e Decisões de Implementação

A implementação foi guiada pela necessidade de um código robusto, modular e eficiente. Um único binário, server, é capaz de operar tanto como SS quanto como SL, determinado por um argumento de linha de comando. Esta decisão de projeto minimiza a redundância de código e simplifica o processo de compilação e implantação.

- **Estruturas de Dados do Servidor**:
    - **ClientInfo:** Uma estrutura fundamental que centraliza todas as informações de um cliente conectado: o descritor de arquivo do socket (socket_fd), o ID de 10 dígitos do sensor (client_id), o slot atribuído pelo servidor (assigned_slot), o identificador de localização (location_id) e o status de risco (risk_status). Um array estático connected_clients[MAX_CLIENTS] armazena o estado de todos os sensores.
    - **P2PState:** Uma máquina de estados finitos (FSM), implementada como um enum, gerencia o ciclo de vida da conexão P2P. Os estados, como P2P_DISCONNECTED, P2P_ACTIVE_CONNECTING, P2P_REQ_SENT e P2P_FULLY_ESTABLISHED, garantem que o complexo handshake de conexão mútua seja tratado de forma ordenada e previsível.
    - **ServerRole:** Um enum (SERVER_TYPE_STATUS, SERVER_TYPE_LOCATION) que define o comportamento do servidor (SS ou SL), permitindo a execução de lógicas condicionais específicas para cada função a partir de um único código-fonte.
- **Identificação de Sensores**: O sistema adota um esquema de dupla identificação. O client_id, uma string de 10 dígitos gerada pelo sensor, serve como um identificador global e persistente, utilizado principalmente na comunicação entre os servidores. O assigned_slot, um inteiro que corresponde ao índice do cliente no array do servidor, é usado como um identificador de sessão na comunicação cliente-servidor, simplificando as operações de busca no lado do servidor. O sensor, após a conexão, valida a consistência do sistema verificando se o assigned_slot recebido de ambos os servidores é o mesmo.

---

4. Protocolo de Comunicação

O protocolo de aplicação define o formato e o significado das mensagens trocadas entre as entidades do sistema. Todas as mensagens são strings de texto ASCII, com no máximo 500 bytes, e seguem uma estrutura simples: <Código> <Payload>. O código é um inteiro que define o tipo da mensagem, e o payload contém dados adicionais. Duas funções de apoio, build_control_message() e parse_message(), foram implementadas para encapsular a construção e a análise sintática dessas mensagens, respectivamente.

- **Mensagens de Controle**: Gerenciam o ciclo de vida das conexões.
    - REQ_CONNPEER (20) / RES_CONNPEER (21): Estabelecem a conexão P2P entre servidores.
    - REQ_DISCPEER (22): Solicita o encerramento da conexão P2P.
    - REQ_CONNSEN (23) / RES_CONNSEN (24): Gerenciam a entrada de um sensor na rede.
    - REQ_DISCSEN (25): Gerencia a saída de um sensor.

- **Mensagens de Dados**: Utilizadas para as funcionalidades principais da aplicação.
    - REQ_SENSSTATUS (40) / RES_SENSSTATUS (41): Comunica a solicitação e a resposta sobre o status de risco de um sensor.
    - REQ_CHECKALERT (36) / RES_CHECKALERT (37): Usadas entre SS e SL para verificar a localização de um sensor em alerta.
    - REQ_SENSLOC (38) / RES_SENSLOC (39): Consultam e retornam a localização de um sensor específico.
    - REQ_LOCLIST (42) / RES_LOCLIST (43): Solicitam e retornam a lista de sensores em uma determinada localização.
- **Mensagens de Confirmação e Erro**:
    - OK_MSG (0): Indica o sucesso de uma operação, com o payload especificando o tipo de sucesso (e.g., 1 para desconexão bem-sucedida).
    - ERROR_MSG (255): Sinaliza uma falha, com o payload indicando o código do erro (e.g., 10 para "Sensor not found").

---

5. Fluxos de Comunicação Detalhados

A interação entre os componentes é orquestrada por fluxos de mensagens bem definidos para cada operação fundamental do sistema.

- **Conexão Peer-to-Peer**: A inicialização da conexão P2P é uma etapa crucial. O primeiro servidor a ser executado tenta uma conexão ativa com o endereço de seu par. Se a conexão falhar, ele assume que é o primeiro na rede, entra em um estado de escuta passiva e aguarda a conexão do outro servidor. Quando o segundo servidor é iniciado, ele consegue se conectar ativamente. A partir daí, um handshake de duas vias com mensagens REQ_CONNPEER e RES_CONNPEER é executado para que ambos os servidores se reconheçam e troquem seus respectivos identificadores (PIdS), culminando no estado P2P_FULLY_ESTABLISHED.

- **Conexão do Sensor**: Um sensor inicia sua operação conectando-se aos servidores SS e SL. Ele envia a mensagem `REQ_CONNSEN`, contendo seu ID de 10 dígitos e uma localização inicial de -1, para ambos os servidores. Cada servidor, ao receber a requisição, valida se o limite de clientes não foi excedido , verifica se o ID já está em uso e, em caso de sucesso, registra o cliente, atribui um `assigned_slot` e responde com `RES_CONNSEN` contendo o slot. Durante este processo de registro, o servidor SL atribui uma localização aleatória entre 1 e 10 (caso a localização recebida seja -1), e o servidor SS atribui aleatoriamente o estado binário `risk_status` (0 ou 1) para o sensor.

- **Funcionalidade: Alerta de Falha (`check failure`)**

Este é o fluxo mais complexo, envolvendo os três componentes para verificar uma potencial pane elétrica.

1. O usuário digita `check failure` no terminal do sensor.
2. O sensor envia `REQ_SENSSTATUS` (código 40) com seu `SlotID` para o SS.
3. O SS verifica o `risk_status` do sensor. Se for 0 (normal), ele responde ao sensor com `RES_SENSSTATUS` (código 41) e um payload de "-1".
4. Se o `risk_status` for 1 (risco) , o SS envia `REQ_CHECKALERT` (código 36) com o `SensorID` de 10 dígitos para o SL.

5. O SL busca a localização (`LocID`) associada ao `SensorID` e responde ao SS com `RES_CHECKALERT` (código 37) e o `LocID` no payload, ou `ERROR` (código 255) com subcódigo 10 se não encontrar.
6. O SS encaminha o resultado para o sensor, enviando `RES_SENSSTATUS` (código 41) com o `LocID` recebido ou a mensagem de erro correspondente.
7. Finalmente, o sensor imprime o alerta formatado com a região correspondente à localização recebida.

- **Funcionalidade: Localização de Sensor (`locate`)**

Este fluxo permite que um sensor consulte a localização de qualquer outro sensor na rede diretamente no Servidor de Localização.

1. O usuário executa o comando `locate <SensID>` no terminal do sensor.
2. O sensor envia a mensagem `REQ_SENSLOC` (código 38) para o SL, contendo o `SensID` alvo como payload.
3. O SL recebe a requisição, busca o `SensID` em sua base de dados de clientes conectados.
4. Se o sensor for encontrado, o SL responde com `RES_SENSLOC` (código 39), contendo a `LocID` correspondente. Caso contrário, envia uma mensagem `ERROR` (código 255) com subcódigo 10 ("Sensor not found").
5. O cliente recebe a resposta e imprime a localização do sensor alvo ou a mensagem de erro.

- **Funcionalidade: Diagnóstico de Localização (`diagnose`)**

Permite a um sensor obter uma lista de todos os sensores ativos em uma determinada localização.

1. O usuário executa o comando `diagnose <LocID>` no terminal.
2. O sensor envia a mensagem `REQ_LOCLIST` (código 42) para o SL. A implementação atual constrói um payload no formato `"<SlotID>,<LocID>"` para esta requisição, identificando quem solicita e qual a localização de interesse.
3. O SL processa a requisição, itera sobre sua lista de clientes e compila uma lista de todos os `SensorID`s que correspondem à `LocID` alvo.
4. Se um ou mais sensores forem encontrados, o SL responde com `RES_LOCLIST` (código 43), cujo payload é uma string contendo os IDs dos sensores separados por vírgula. Se nenhuma localização for encontrada ou for inválida, o servidor envia `ERROR` (código 255) com subcódigo 10.
5. O cliente recebe a resposta e exibe a lista de sensores para o usuário ou a notificação de erro.

- **Desconexão do Sensor (`kill`)**

Este fluxo descreve o processo de encerramento limpo de um sensor.

1. O usuário digita o comando `kill` no terminal do sensor.
2. O sensor envia a mensagem `REQ_DISCSEN` (código 25), contendo seu `SlotID` como payload, para ambos os servidores, SS e SL, de forma independente.
3. Cada servidor, ao receber a mensagem, verifica se o `SlotID` corresponde a um cliente ativo e registrado.
4. Se a verificação for positiva, o servidor remove as informações do cliente de sua base de dados, libera o slot, responde com uma mensagem de confirmação `OK` (código 0) com subcódigo 1 ("Successful disconnect") e fecha a conexão com o socket do cliente. Se o cliente não for encontrado, responde com `ERROR` (código 255) e subcódigo 10.

5. O sensor lê a confirmação de cada servidor, imprime as mensagens de sucesso na desconexão (`"Received disconnect confirmation from SS."` e `"Received disconnect confirmation from SL."`) e encerra sua execução.

---

6. Discussão e Conclusão

A implementação atual atende a todos os requisitos funcionais descritos na especificação do projeto. O executável único para os servidores e o uso de uma máquina de estados para a conexão P2P demonstraram ser boas decisões de projeto, possibilitando a reutilização de código e a robustez do sistema. O tratamento de múltiplas conexões com select() demonstra a aplicação correta de multiplexação de E/S para servidores concorrentes sem a complexidade de programação multithread.

Contudo, uma limitação notável foi identificada no fluxo de "Alerta de Falha". Ao aguardar a resposta do SL, o servidor SS realiza uma chamada bloqueante read(). Em um ambiente real, onde o SL pode estar sobrecarregado ou não responder, isso paralisaria todo o servidor SS, impedindo-o de atender outros clientes. Uma melhoria futura seria refatorar essa lógica para ser totalmente assíncrona: o SS enviaria a requisição ao SL, registraria internamente que está aguardando uma resposta para um cliente específico e continuaria seu loop de eventos. A resposta do SL seria então processada como qualquer outro evento de entrada no select(), permitindo que o servidor permanecesse responsivo.

Em conclusão, o projeto resultou em um sistema funcional e bem-estruturado para o monitoramento de redes elétricas. Ele serviu como um ótimo exercício prático na aplicação de conceitos de programação de redes, como sockets TCP, design de protocolos de aplicação e gerenciamento de estado em sistemas distribuídos. As funcionalidades de conexão, desconexão, alerta, localização e diagnóstico foram todas implementadas com sucesso, fornecendo uma base sólida que poderia ser estendida para sistemas de monitoramento mais complexos e resilientes.

---

7. Compilação e Execução

- **Compilação**: O projeto inclui um Makefile para facilitar a compilação. Navegue até o diretório raiz do projeto e execute o comando make. Isso compilará os arquivos server.c, sensor.c e common.c, gerando os executáveis server e sensor.

- **Execução**:

Servidores: Execute cada servidor em um terminal separado. O primeiro a iniciar aguardará passivamente a conexão P2P. O segundo se conectará ativamente.

```
# Terminal 1: Inicia o Servidor de Status (SS) na porta 61000
./server 127.0.0.1 60000 61000 SS
```

```
# Terminal 2: Inicia o Servidor de Localização (SL) na porta 62000
./server 127.0.0.1 60000 62000 SL
```

Sensor: Execute o cliente em um novo terminal, fornecendo os endereços IP e as portas dos servidores SS e SL.

```
# Terminal 3: Inicia o Sensor
./sensor 127.0.0.1 61000 127.0.0.1 62000
```