Relatório Técnico: Sistema de Monitorização de Falhas em Redes Elétricas

Autor: Ramiro Noronha Reis Ribeiro
Disciplina: Redes de Computadores

1. Introdução

Neste relatório, é detalhada a implementação e o desenho de um Sistema de Monitorização de Falhas em Redes Elétricas, desenvolvido como trabalho prático da disciplina de Redes de Computadores. O objetivo do projeto é simular uma solução capaz de prever e, portanto, atenuar os danos causados por panes e apagões por meio de sensores e servidores.

O sistema é composto por dois tipos de servidores que se comunicam de forma P2P: um Servidor de Status (SS), responsável por monitorar o estado de risco dos sensores, e um Servidor de Localização (SL), que gere a posição geográfica de cada sensor. Múltiplos clientes (sensores) conectam-se simultaneamente a ambos os servidores para enviar dados e receber comandos.

Toda a comunicação é implementada na linguagem C, utilizando a interface de sockets POSIX sobre o protocolo TCP, garantindo a entrega ordenada e confiável das mensagens. A gestão de múltiplas conexões concorrentes nos servidores é realizada através da chamada de sistema select().

2. Mensagens

A comunicação entre os componentes do sistema é baseada num protocolo de aplicação textual, operante sobre o TCP. As mensagens são formatadas como strings ASCII, separadas por espaços ("CODIGO PAYLOAD1 PAYLOAD2"), e são analisadas no recetor através de sscanf ou strtok para extrair os seus componentes. A seguir, é detalhada a função de cada código de mensagem implementado.

- **REQ_CONNPEER (20):** Invocada na função initialize_p2p_link() do servidor que consegue estabelecer uma conexão P2P ativa. É a primeira mensagem do "handshake" entre os servidores.
- **RES_CONNPEER (21):** Resposta ao REQ_CONNPEER. O servidor que estava a escutar responde com o seu ID para o iniciador. O iniciador, por sua vez, responde a esta mensagem com o seu próprio ID, resultando numa troca para identificação mútua que conclui o handshake.
- **REQ_DISCPEER (22):** Enviada quando o comando kill é acionado no terminal de um dos servidores. Solicita o encerramento da ligação P2P.
- **REQ_CONNSEN (23):** Na inicialização, o sensor gera o seu próprio ID e localização aleatória. Ele envia esta mensagem, contendo ambos os dados como payloads, para os servidores SS e SL para se registar na rede.
- **REQ_DISCSEN (25):** Acionada pelo comando kill no terminal do sensor, esta mensagem é enviada para ambos os servidores para solicitar a sua remoção da base de dados. O payload contém o ID do sensor que deseja desconectar-se.
- **REQ_CHECKALERT (36):** Mensagem interna P2P. Quando o SS deteta que um sensor tem um estado de risco (risk_status = 1) após um pedido REQ_SENSSTATUS, ele envia esta mensagem ao SL para pedir a localização desse sensor.
- **RES_CHECKALERT (37):** Mensagem interna P2P. Em resposta a REQ_CHECKALERT, o SL encontra a localização do sensor e envia-a de volta para o SS.
- **REQ_SENSLOC (38):** Enviada pelo sensor ao SL após o comando locate. O payload contém o ID

do sensor cuja localização se deseja consultar.
- **REQ_SENSSTATUS (40, 1 payload):** Disparada no sensor pelo comando check failure e enviada ao SS. O payload contém o ID do próprio sensor que está a realizar a verificação.
- **REQ_LOCLIST (40, 2 payloads):** Enviada pelo sensor ao SL após o comando diagnose. Os payloads contêm o ID do sensor solicitante e o ID da localização a ser diagnosticada.
- **MSG_OK (0):** Mensagem genérica de confirmação, usada para confirmar uma desconexão bem-sucedida.
- **MSG_ERROR (255):** Mensagem genérica de erro. O payload contém um código numérico que detalha a natureza do erro (ex: "10" para "Sensor not found").

3. Arquitetura do Sistema

A arquitetura do sistema foi construída de forma distribuída, respeitando a separação de responsabilidades e a escalabilidade do monitoramento.

- **Sensores (sensor.c):** Representam os dispositivos de monitorização. Cada sensor, ao iniciar, gera o seu próprio identificador (persistido em sensor_ids.txt) e estabelece duas conexões de rede distintas e simultâneas: uma com o SS e outra com o SL.
- **Servidores (server.c):** Implementados num único código-fonte, os servidores operam em dois modos distintos, definidos pela sua porta de conexão com os clientes. O SS (porta 61000) mantém o estado de risco, enquanto o SL (porta 60000) mantém a localização.

4. Implementação do Servidor (server.c)

O código do servidor constitui o núcleo do sistema. A sua lógica central é baseada num ciclo de eventos infinito que utiliza select() para gerir todas as fontes de entrada e saída de forma concorrente (entrada do teclado, socket de escuta de clientes, socket de escuta P2P, e todos os sockets de comunicação ativos).

A diferenciação de papéis (SS vs SL) é alcançada através da variável global my_role, definida na inicialização com base no número da porta do cliente. Esta flag é então utilizada na função central process_incoming_message para direcionar o fluxo de execução para a lógica correta, especialmente para mensagens com códigos ambíguos como o 40, o qual pode ser tanto REQ_SENSTATUS ou REQ_LOCLIST.

O servidor utiliza um array em memória (SensorInfo[]) para armazenar as informações dos sensores conectados e um segundo array (PendingRequest[]) para gerir o estado de pedidos P2P assíncronos. A lógica de negócio foi abstraída em funções auxiliares como register_new_sensor, handle_check_failure_command, get_sensor_by_id, etc.

5. Implementação do Sensor (sensor.c)

O programa do sensor simula a operação de um dispositivo individual na rede. Ele funciona como o cliente no sistema, sendo responsável por gerar a sua própria identidade, iniciar a comunicação e responder aos comandos do utilizador.

A primeira ação do equipamento é definir a sua própria identificação através da função generate_unique_sensor_id, que cria um ID único de 10 dígitos e o persiste no ficheiro sensor_ids.txt para garantir a unicidade entre diferentes execuções. Em sequência, o sensor estabelece duas conexões TCP paralelas (para o SS e para o SL) e envia a mensagem REQ_CONNSEN para se registar.

Uma vez registado, o equipamento entra num ciclo de vida interativo gerido por select(), aguardando por comandos do utilizador através da entrada padrão (kill, check failure, diagnose, locate) ou por mensagens vindas de qualquer um dos servidores. A lógica para enviar os pedidos e para processar as respostas está encapsulada, respetivamente, nas funções process_keyboard_input e process_server_message.

6. Decisões de Implementação e Refinamentos

Várias decisões de design foram tomadas para refinar a especificação e garantir a robustez do sistema:

1. **Geração de ID Descentralizada:** A decisão de mover a geração de IDs do servidor para os sensores (generate_unique_sensor_id) torna o sistema mais robusto e elimina um ponto único de falha. A persistência em sensor_ids.txt e a remoção de IDs no kill (remove_id_from_file) garantem a unicidade e a limpeza dos IDs ao longo do tempo.
2. **Dupla Conexão do Sensor:** Foi fundamental para resolver o problema de o Servidor de Localização não ter conhecimento dos sensores registados apenas no Servidor de Status.
3. **Gestão do Handshake P2P:** A introdução da variável de estado p2p_handshake_complete foi a solução para os dois principais bugs da comunicação P2P: o loop infinito de respostas (RES_CONNPEER) e a desconexão prematura.
4. **Diferenciação de Papéis por Porta:** O papel do servidor é determinado elegantemente pela porta de cliente em que ele escuta (60000 para SL, 61000 para SS), simplificando a execução.
5. **Modularização do Código:** Tanto o server.c quanto o sensor.c foram extensivamente refatorados. A lógica de tratamento de eventos foi extraída da função main para funções dedicadas (handle..., process...), tornando o código mais legível e de fácil manutenção.

7. Discussão e Conclusão

A realização deste projeto foi uma grande oportunidade para consolidar os conhecimentos teóricos da disciplina numa aplicação prática e tangível. O principal aprendizado foi a implementação de baixo nível da interface de sockets POSIX em C, desde a configuração inicial dos sockets e endereços, passando pela gestão do ciclo de vida das conexões TCP, até a troca de dados.

Ademais, o projeto permitiu visualizar de maneira clara a importância de desenhar e implementar um protocolo de aplicação bem definido. A padronização dos códigos de operação e erro foi essencial para garantir que o cliente e os dois servidores pudessem se comunicar de maneira previsível. A comunicação P2P dos servidores, em particular, ilustrou como a arquitetura distribuída colaborou para compor uma funcionalidade que um componente não conseguiria oferecer isoladamente.

A implementação de um Sistema de Monitoramento de Falhas em Redes Elétricas foi concluída com sucesso, resultando numa aplicação funcional que valida a eficiência de uma arquitetura distribuída para a deteção de anomalias. O resultado é um sistema robusto que cumpre os requisitos propostos, demonstrando um entendimento sólido dos conceitos de redes de computadores.