Trabalho Prático 2 - Redes de Sensores Sem Fio

1 Introdução

As Redes de Sensores Sem Fio (RSSF) representam uma tecnologia fundamental para a coleta de dados em tempo real em diversos ambientes, com aplicações que vão desde o monitoramento ambiental até a automação industrial e intervenções biomédicas. Essas redes são compostas por pequenos dispositivos sensores distribuídos por uma área, que coletam dados como temperatura, umidade, pressão, entre outros, e se comunicam sem fio com outros dispositivos ou diretamente com um servidor central. Isso permite a criação de sistemas distribuídos e escaláveis.

Este trabalho prático foca na simulação de uma RSSF com três tipos de sensores em uma região delimitada(representada por uma matriz de coordenadas inteiras 10x10), onde os sensores são responsáveis por coletar e reportar informações de forma periódica, utilizando um modelo publish/-subscribe com tópicos para a comunicação com um servidor central.

2 Estrutura de comunicação

No protocolo de comunicação entre clientes e servidor do sistema, as mensagens trocadas são encapsuladas na estrutura `sensor_message`.

2.1 Descrição das propriedades

- `type`: String de caracteres que identifica o tipo do sensor que está enviando a mensagem.
- `coords`: Representa as coordenadas geográficas do sensor.
- `measurement`: Um valor float que carrega a medição atual feita pelo sensor.

3 Implementação do Servidor

3.1 Descrição geral

O servidor implementado em C utiliza sockets POSIX para estabelecer conexões TCP/IP, suportando múltiplas conexões de clientes em um modelo multi-threaded. Ele foi projetado para lidar com mútiplos clientes, onde diferentes tipos de sensores enviam dados para tópicos específicos.

3.2 Estrutura `client_t`

A estrutura `client_t` foi implementada para armazenar informações sobre os clientes conectados ao servidor em um sistema de comunicação baseado em sockets. Cada cliente é representado por uma instância desta estrutura, que armazena o descritor do socket associado ao cliente e o tópico ao qual o cliente está inscrito. Ela permite que o servidor mantenha um registro organizado de todos os clientes conectados, facilitando a gestão de suas subscrições e a comunicação efetiva baseada em tópicos.

3.3 Gerenciamento de clientes e tópicos

Para gerenciar os clientes conetados ao servidor, e suas respectivas inscrições em tópicos, foram criadas as seguintes estruturas globais:

- `*client_t susbcribed_clients[MAX_CLIENTS]`: essa lista gerencia todos os clientes conectados ao servidor.
- `*client_t clients_per_topic[MAX_TOPICS][MAX_CLIENTS_PER_TOPIC]`: já essa matriz gerencia os clientes por tópico, onde os diferentes tópicos são as linhas, e as colunas os cliente inscritos.

3.4 Fluxo do código e principais funções

- **Iniciação do Servidor:** O servidor começa pela configuração do socket, escolhendo IPv4 ou IPv6 com base no argumento da linha de comando. Ele então vincula o socket a um endereço e começa a escutar por conexões entrantes.
- **Aceitação de Novos Clientes:** Quando um cliente tenta conectar, o servidor aceita a conexão e aloca uma estrutura **client_t** para armazenar informações do cliente, como o socket, o endereço, e o tópico ao qual ele eventualmente se inscreverá. Para adicionar o cliente a lista de clientes conectados a função `add_client` é chamada. Ela instancia o mutex necessário para editar a lista de clientes. Cada novo cliente é tratado por uma thread separada, permitindo que o servidor lide com múltiplas conexões simultâneas de forma eficiente.
- **Função `handle_client`:** Esta função é executada em uma nova thread para cada cliente conectado. Ela lida com a leitura das mensagens dos clientes, processando as subscrições aos tópicos e encaminhando as mensagens dos sensores para os tópicos apropriados. A primeira mensagem recebida de um novo cliente é usado para definir a sua subscrição.
- **Gerenciamento de Tópicos:** A função `subscribe_client_to_topic` inscreve um cliente em seu tópico correspondente, enquanto `publish_to_topic` envia mensagens recebidas de um sensor para todos os clientes inscritos no mesmo tópico que ele. Ambas funções são protegidas por mutexes, em `subscribe_client_to_topic` um mutex é instanciado para adicionar o cliente a lista de inscritos ao tópico, de forma a evitar uma condição de corrida para salvar novos clientes a estrutura. Em `publish_to_topic` um mutex também é usado, para garantir que um cliente mande mensagem para todos inscritos no seu tópico por vez, evitando assim conflitos no envio de mensagens.
- **Remoção de Clientes:** Quando um cliente se desconecta ou ocorre um erro de leitura, a conexão é encerrada, o cliente é removido da lista global e sua memória é liberada. Novamente aqui um mutex é utilizado.

4 Implementação do Cliente

4.1 Descrição geral

O programa cliente foi configurado para conectar-se ao servidor, enviar dados de sensoriamento e receber atualizações ou comandos do servidor, seguindo o protocolo TCP/IP, podendo operar com endereços IPv6 e IPv4.

4.2 Fluxo de execução e principais funções

- **Conexão com o Servidor:** O cliente inicializa um socket TCP, configurando-o para o endereço IP e a porta fornecidos nos argumentos de linha de comando. Ele suporta tanto conexões IPv4 quanto IPv6, escolhendo o tipo de endereço baseado na presença de ':' no IP fornecido, indicando um endereço IPv6.
- **Envio de Dados Iniciais:** Após a conexão, o cliente configura e envia uma mensagem inicial para o servidor contendo o tipo do sensor, as coordenadas e uma medição inicial. Esse processo é executado apenas uma vez ao estabelecer a conexão.
- **Manuseio de Sinais:** O cliente implementa um manipulador de sinal para SIGINT, que envia uma última mensagem ao servidor com uma medição negativa, indicando que está se desconectando, antes de fechar o socket e terminar o processo.
- **Recepção e Processamento de Dados:** Em um loop contínuo, o cliente recebe mensagens do servidor, processa qualquer ajuste necessário com base nas medições de outros sensores próximos, e envia novas medições. A lógica inclui a manutenção de uma lista de sensores ordenada de acordo com sua proximidade do cliente, para determinar como as medições devem ser ajustadas.
- **Tratamento de erros:** Antes de propriamente incializar o sensor, o programa verifica se todos comandos necessários foram passados para o terminal, e se os parâmetros tem valores aceitáveis, assim esses erros são tratatdos do lado do cliente.

4.3 Funções de Manipulação da Lista

A estrutura escolhida para realizar o gerenciamento de sensores vizinhos foi uma lista encadeada, onde cada nó da lista é um sensor, com suas respectivas coordenadas e distância do sensor cliente. Essa estrutura foi escolhida por ser simples e efetiva, e como o contexto de teste mais complexo seria de 12 sensores, seu custo temporal O(n) é aceitável. Mais sobre seu funcionamento abaixo:

- **Inserção Ordenada:** Foi implementada a função `insert_sorted` que insere sensores na lista encadeada com base na distância até o sensor principal. Esta inserção ordenada assegura que a lista sempre mantenha os sensores em ordem crescente de distância durante a descoberta dinâmica de sensores, facilitando a rápida identificação dos sensores mais próximos para ajustes de medição.
- **Atualização de Medições:** Ao receber dados novos de sensores vizinhos, o cliente utiliza esta lista para determinar quais sensores afetarão suas medições. Se um sensor vizinho estiver entre os 3 mais próximos, suas medições influenciarão diretamente as medições do sensor principal, permitindo ajustes baseados na proximidade e precisão aumentada das medições gerais.

- **Remoção de Sensores:** Quando um sensor envia uma medição negativa (indicando desconexão), a função de remoção é chamada para retirá-lo da lista encadeada. Isso garante que a lista reflita apenas os sensores ativos e próximos, mantendo a eficiência do processo de cálculo e ajuste das medições.

5 Desafios e Soluções

5.1 Do Lado do Cliente

Durante a implementação do cliente, diversos desafios técnicos foram enfrentados para atender a todas as exigências especificadas no projeto. A principal tarefa foi estabelecer uma conexão estável com o servidor e gerenciar adequadamente a recepção e processamento de mensagens. Erros de entrada fornecidos pelo usuário através do terminal são validados e tratados.

Medições são simuladas automaticamente, gerando dados aleatórios dentro de um intervalo pré-definido para cada tipo de sensor. Dependendo também do tipo de sensor representado, o período de inatividade (sleep) varia, o que exigiu uma implementação flexível e adaptativa. Após a recepção de mensagens, era necessária uma verificação para assegurar que as mensagens pertencessem ao mesmo tipo de sensor, permitindo assim processamentos subsequentes de forma coerente.

Uma lista encadeada dinâmica foi implementada para gerenciar os sensores descobertos, mantendo sempre atualizada uma lista dos três sensores vizinhos mais próximos. Isso foi crucial para a correção e eventual convergência das medidas entre sensores adjacentes. Testes extensivos foram realizados para garantir que a inserção e remoção de sensores na lista fossem executadas corretamente, mantendo a lista ordenada e funcional.

5.2 Do Lado do Servidor

No lado do servidor, o desafio predominante foi o gerenciamento eficaz de múltiplos clientes e a subscrição de clientes a tópicos específicos. A complexidade aumentou com a necessidade de garantir acesso concorrente a recursos compartilhados sem causar deadlocks ou condições de corrida. Mutexes foram utilizados para sincronizar o acesso a esses recursos, um passo essencial para a escalabilidade e estabilidade do servidor.

A arquitetura do servidor foi testada com até 12 sensores ativos simultaneamente, comportando-se conforme o esperado sob carga. O manejo de erros como SIGPIPE, que podem ocorrer quando um sensor é removido do sistema, foi tratado adequadamente. Esse tipo de sinal foi utilizado para identificar a saída de sensores e atualizar a matriz de sensores inscritos por tópico sem interrupções.

Esse rigoroso processo de desenvolvimento e teste garantiu que tanto o cliente quanto o servidor funcionassem de maneira eficiente e robusta, adequando-se às necessidades específicas do projeto e atendendo as expectativas de desempenho e confiabilidade.