Projeto RC 2024/2025
Protocolo "PowerUDP"

Redes de Comunicação — LEI 2024/2025

Nome	Nº Estudante	Contacto
Vasco Guilherme Alves	2022228207	960399272
Rodrigo Faria	2023234032	964653531

Conteúdo

1 Introdução

Este projecto foi feito no ambito da disciplina de Redes de Comunicação com o objectivo de implementar um protocolo

2 Arquitetura do projeto

Neste projeto usamos dois ficheiros : cliente.c e servidor.c.

Para autenticação entre o servidor e os clientes e o envio da configuração inicial, usamos protocolo TCP.

A configuração pode ser alterada dinamicamente via multicast, enviada pelo servidor.

A comunicação entre clientes é feita por UDP, com os mecanismos de confiabilidade implementados no PowerUDP.

3 Implementação

Aqui diremos como implementámos cada funcionalidade do PowerUDP, e as escolhas que fizemos para a realizaçao de cada funcionalidade. No projeto, para podermos executar varias funções simultaneamente, usamos threads. Decidimos usar threads em vez de processos para evitar ter de criar memória partilhada para todos os processos acederem. Deste modo, mantemos todas as variaveis na mesma stack. Para sockets e threads, usamos POSIX como a nossa escolha. Com esta escolha, evitamos também o uso de pipes para comunicação.

Sempre que um cliente mandam uma mensagem UDP para outro cliente, começamos um temporizador lógico. Em vez de usar select(), implementámos um controlo de timeout, que aguarda a resposta dentro de um loop. Se não for recebido um ACK dentro do tempo, a mensagem é retransmitida.

Quando enable_sequence está ligado, o emissor incrementa a sequência a cada mensagem enviada e o recetor mantém um expected_seq localmente, que o faz aceitar apenas mensagens com o número esperado. Mensagens fora de ordem são rejeitadas.

Para validar a integridade da mensagem, o cliente calcula um checksum do cabeçalho (pu_checksum_helper), ao receber a mensagem, o recetor calcula o checksum e compara com o recebido. Se não coincidir, a mensagem é rejeitada.

Para todas as mensagens, se a mensagem for aceite mandamos um ACK, caso contrário,

é mandado um NAK

Para testar a resiliência do protocolo, usamos ums injeção de perda de pacotes, que permite nos simular um caso não desejado. Enquanto recebe, com base num valor aleatório, a mensagem é descartada para testar.

3.1 Servidor

O servidor esctura por ligações TCP. Ao se ligar, cria uma thread para tratar desse cliente. O servidor valida a PSK e envia a configuração atual do protocolo.

A configuração atual do PowerUDP pode ser atualizada no servidor. Quando isso acontece, entrega a configuração nova para todos os clientes com UDP multicast.

Para poder comunicar com cada cliente simultaneamente, o servidor cria uma thread para cada ligação TCP recebida.

3.2 Cliente

Ao inicarmos o cliente, estabelecemos uma ligação TCP com o servidor. Esta primeira ligação serve para autenticar o cliente (mandando uma mensagem de registo com a PSK) e receber a config atual.

Após o cliente ficar registado no servidor, guarda a configuração recebida (ConfigMessage), que irá ditar o comportamento do protocolo.

Após isso, o cliente junta se ao grupo multicast do servidor. Assim, sempre que o servidor enviar uma nova configuração, o cliente receberá através deste canal.

Para enviar mensagens UDP confiaveis, criamos um pacote UDP com um cabeçalho PU_header, que inclui o número de sequência, timestamp, flags e checksum. Como esperado, incluimos também um payload com os dados.

O uso de checksum e uma sequência é crucial, sendo o checksum necessário para garantir a integridade dos dados e a sequência para garantir a ordenação correta se enable_sequence estiver ligado.

4 Conclusão

Este projeto permitiu nos aplicar e consolidar conceitos fundamentais de redes de comunicação como comunicação entre clientes e servidores, a usar um protocolo fiável sobre UDP, que usa sockets TCP e UDP.

Foram implementados com sucesso mecanismos de retransmissão com backoff exponencial, controlo de sequência, validação de checksum e resposta ACK/NAK, garantindo a

entrega fiável de mensagens.

A utilização de threads para execução simultanea de funções correu como esperado, evitando criar memória partilhada ou pipes desnecessários, fazendo este programa resource efficient.