DCCNET: Implementação e Desafios

Resumo do Projeto

O TP1 DCCNET teve como objetivo criar uma rede de comunicação entre cliente e servidor, utilizando um protocolo de comunicação customizado. A implementação foi feita utilizando a linguagem Python com uma comunicação baseada em frames de dados. Essa documentação descreve os principais desafios encontrados, as dificuldades enfrentadas durante a implementação e as soluções adotadas.

Aplicações

O foco da primeira aplicação foi a construção de um sistema(cliente) capaz de se comunicar com o sistema do professor(servidor), seguindo todos os padrões estabelecidos no trabalho. Já a segunda aplicação, teve como objetivo criar um sistema capaz ser utilizado tanto como cliente quanto como servidor, a fim de realizar a troca de dados.

Mecanismo de Recuperação de Framing Após Erros

Um dos maiores desafios do projeto foi garantir a integridade da comunicação, especialmente quando quadros de dados podem ser corrompidos durante a transmissão. A recuperação de framing após erros foi tratada por um mecanismo de verificação, que utiliza o checksum para validar os dados transmitidos. Quando há uma exceção durante o recebimento ou o processamento do frame, geralmente porque o ACK esperado não chegou, realizamos a retransmissão.

A estrutura de framing foi projetada com um cabeçalho contendo bytes de sincronização (SYNC_BYTES), o comprimento do quadro, o identificador e a flag. Para cada quadro transmitido, a soma de verificação é recalculada e verificada antes do envio.

Tratamento de Exceções e Mecanismo de Retransmissão

Para garantir comunicação confiável mesmo em cenários de erro, foram implementadas **exceções específicas** que ajudam a isolar e tratar falhas em diferentes etapas do protocolo. As principais exceções utilizadas são:

- `InvalidChecksumError`: lançada quando o checksum do quadro recebido é inválido.
- `MissingAckError`: lançada quando um ACK esperado não é recebido dentro do esperado.

Essas exceções permitem uma **recuperação controlada**: ao capturar falhas previsíveis, a aplicação pode tomar ações como re**transmitir o frame**, garantindo que a comunicação continue estável mesmo diante de perdas ou corrupção de pacotes.

A lógica de retransmissão é integrada à função de comunicação principal (`make_comunication`), que encapsula o controle de envio, espera por resposta, verificação de integridade e retransmissão automática em caso de falhas. Esse mecanismo é essencial para simular a resiliência exigida por protocolos da camada de enlace.

Interface de Aplicações com a Implementação do DCCNET

A interface com as aplicações foi projetada para ser simples e eficiente. A principal interface exposta para a aplicação é uma função que realiza a autenticação com o servidor e envia dados através de quadros específicos, como mensagens de "autenticação completa".

As aplicações podem interagir com o DCCNET utilizando funções que abstraem os detalhes do protocolo de comunicação. Por exemplo, a função `make_autentication()` envia os quadros necessários para autenticação e aguarda a resposta do servidor. O cliente simula o envio e a recepção de quadros de dados, com verificação da integridade e recuperação de quadros corrompidos.

Contribuição das Ferramentas de IA na Solução

As ferramentas de Inteligência Artificial desempenharam um papel fundamental para a avaliação de erros durante o desenvolvimento. A IA foi utilizada para auxiliar no aprendizado do uso de socket e assincronismo. Ademais, a IA ajudou a documentar de maneira eficiente e organizada, garantindo uma visão clara e acessível sobre a estrutura do trabalho proposto, facilitando a compreensão.

Implementação do Cliente-Servidor para Transferência de Arquivos

Na segunda parte do projeto DCCNET, o foco foi expandido para implementar um sistema cliente-servidor assíncrono, capaz de realizar transferência de arquivos utilizando o protocolo desenvolvido. Essa etapa foi fundamental para validar a aplicação prática do protocolo de comunicação e para lidar com desafios reais de transmissão de dados em rede.

Foram desenvolvidos dois módulos principais: o cliente (`run_client`) e o servidor (`run_server_async`). Ambos foram implementados em Python utilizando `asyncio`, garantindo operações não bloqueantes para envio e recepção de dados. O cliente é responsável por se conectar a um servidor especificado (suportando tanto IPv4 quanto IPv6) e enviar arquivos definidos pela linha de comando, enquanto o servidor escuta conexões em uma porta específica, processando as transmissões recebidas e salvando os dados no destino especificado.

Tratamento de Transmissão e Recepção em Paralelo

A comunicação do DCCNET foi projetada para permitir a transmissão e a recepção de dados em paralelo, aproveitando-se do assincronismo(segunda aplicação). A abordagem foi implementada garantindo que a transmissão de dados ocorra enquanto o sistema ainda pode receber quadros de resposta.

A estratégia adotada para gerenciar a recepção e transmissão em paralelo envolveu o uso de um protocolo simples de ACK, onde cada quadro transmitido é aguardado até que um ACK correspondente seja recebido, antes de continuar com a próxima transmissão. Esse processo foi automatizado dentro de uma função que aguarda as respostas enquanto retransmite dados se necessário.

Integração com o Protocolo DCCNET

O módulo `handle_connection_async` foi o elo entre a lógica de rede e o protocolo de comunicação. Ele cuidava de empacotar e desempacotar quadros, garantindo que as mensagens de controle (como ACKs e retransmissões) e os dados dos arquivos fossem corretamente transmitidos e avaliados. Assim, a segunda parte consolidou a aplicação prática do protocolo, provando que ele não

só funcionava em cenários simulados, mas também em situações reais de troca de arquivos entre sistemas distintos.

Dificuldades e Soluções Adotadas

Durante o desenvolvimento, diversos desafios foram enfrentados:

1. **Erro de Frames**: Identificar erros de frames foi um desafio nas duas aplicações feitas. Na primeira, em decorrência da falta de explicações para as falhas e na segunda, o problema foi o envio de frames com o tamanho dos dados de 4096 bytes, o que se mostrou inviável. Assim, para contornar isso, resolvemos enviar em cada pacote apenas 1000 bytes de dados;
2. **Sincronização de Envio e Recepção**: A sincronização entre o envio e a recepção de dados foi um dos grandes desafios, já que a recepção do ACK está intrinsecamente ligada ao envio de dados;
3. **Uso do assincronismo:** foi um desafio no início da implementação da segunda parte.

Testes

Os principais testes feitos foram na segunda parte, em que além de realizarmos a transmissão local de dados entre cliente e servidor com arquivos de diferentes tamanho, como, o cliente enviado um arquivo maior que o servidor e depois o contrário, e também o envio de arquivos grandes(os pdfs) e pequenos(os txts), nós buscamos testar com um colega de sala, chamado Thiago SIlva(que fez o seu código em uma linguagem diferente), em que conseguimos transmitir e receber arquivos seguindo a mesma lógica dos testes locais. Para tal, nós utilizamos o Hamachi.

Por fim, vale ressaltar que foi no teste feito com nosso colega que descobrimos que quando enviássemos em torno de 4096 bytes de dados nos frames, o checksum falhava algumas vezes(nós conferimos se nosso método de calcular o checksum estava igual). Logo, decidimos diminuir o número de bytes enviados.

Conclusão

O projeto DCCNET foi esclarecedor devido às complexidades da comunicação em rede, tratamento dos frames, erros de transmissão e a necessidade de retransmissões.