Universidade Federal de Minas Gerais
EE - Engenharia de Sistemas
DCC023 - Redes de Computadores

Trabalho Prático I - DCNET

Humberto Monteiro Fialho - 2013430811
Rafael Carneiro de Castro - 2013030210

2 de maio de 2018

1 Introdução

O trabalho aplica os conceitos de codificação, enquadramento, detecção de erro, sequenciamento e retransmissão de dados. A partir de uma entrada aleatória o software deve ser capaz de identificar o padrão referência de sincronização para realizar envio e recebimento de informações entre dois componentes conectados entre si. A técnica utilizada é o enquadramento por contagem para que o dado seja lido de forma correta. Pelo mesmo código é possível rodar uma instância de conexão passiva (servidor) e uma de conexão ativa (cliente). Ambos enviam os dados lidos a partir de um arquivo e escrevem os dados recebidos em outro arquivo. A linguagem utilizada para desenvolver o programa é *Python*, pela facilidade de implementação e robustez.

2 Desenvolvimento

De maneira inicial, foram criadas duas funções para realizar a inicialização do servidor e do cliente. O servidor começa com o endereço 127.0.0.1 e os outros dados (porta e arquivos) são coletados no comando do terminal como foi especificado para o projeto. Para abordar o enlace *full-duplex* tanto no cliente quanto no servidor foram criadas duas *threads* para trabalharem simultaneamente, uma para enviar e outra para receber as informações. A *threads* de recebimento dos dados recebe tanto dados do arquivo quanto confirmações de envio (ACK).

Os dados lidos a partir do arquivo de entrada são armazenados em um objeto de uma classe, para facilitar o tráfego de informações e construção do enquadramento. Vale ressaltar que o quantidade de dados lida por vez do arquivo é limitada, já que o campo de tamanho de dados no enquadramento também tem um limite. Os dados recebidos também são armazenados numa instância dessa classe, para se aproveitar de métodos comuns, como por exemplo o de cálculo de *checksum*.

No envio dos dados, ocorre o envio dos dois SYNCs necessários para identificar o início do quadro DCCNET. Também, o dado é coletado do arquivo, os outros campos do enquadramento são construídos, e tudo é enviado. No recebimento dos dados é realizada a sincronização pelos SYNCs e as outras informações são recebidas uma a uma para facilitar as verificações dos parâmetros *checksum*, *id*, *flag* e *data*, porque eles são utilizados para validação das informações enviadas.

Uma das dificuldades durante a contrução da solução foi encontrar uma saída para o *timeout*, pois não era permitido sair da execução quando não encontrasse o SYNC e sim dar uma sequência repetitiva à busca até o limite de tempo determinado. Para solucionar isso, foi utilizado *threading.Timer()*, que possui a capacidade de realizar a medição do tempo e tomar uma ação definida quando chegar ao valor de referência. Sendo assim, a conexão não é fechada e o gerenciamento do timeout é feito manualmente.

Outra decisão diz respeito a como realizar a leitura dos arquivos. Inicialmente a tentativa era pela função *readline* para facilitar a coleta e o processamento, porém para uma arquivo de dados muito grande, a leitura por completo se torna muito lenta, e a quantidade de dados enviada para a construção de *headers* se torna muito maior. Assim, a função padrão *read* foi adotada, para se ler e enviar a maior quantidade possível de dados.

Um outro detalhe encontrado durante os testes foi a conversão de alguns caracteres especiais da tabela *ASCII* para bytes, pois ao utilizar a função *len()* a partir de uma *string*, a quantidade de bytes poderia

ser diferente. Para caracteres como *Ç*, chamando a função *len()* a partir da *string* obtém-se 1, quando na verdade a quantidade de *bytes* desse carater especial é 2. Para solucionar isso, bastou utilizar o tipo primitivo *byte* do *Python* em todos os dados. Assim, a função *len()* retorna a quantidade de *bytes* do dado, como o desejado.

Outro *trade-off* identificado foi no algoritmo do *checksum*, pois mesmo com ele dando os resultados esperados não era possível acompanhar a velocidade de leitura dos dados. Isso pode provocar algumas dessincronias na leitura de arquivos, já que o recebedor dos dados poderia ainda estar calculando o *checksum* quando o *timeout* do fornecedor já tivesse esgotado. Para arquivos muito grande, isso pode provocar certa lentidão para a conclusão de todo o processo, cerca de 3 segundos para arquivos de 1MB.

3 Conclusão

Todas as tecnicas e conceitos disponíveis foram aplicados. O código desenvolvido foi testado para todos os casos de teste disponibilizados, e ainda outros além destes, mostrando ser capaz de realizar aquilo que foi especificado. Apesar dos desafios, conclui-se que o objetivo final foi atingido.