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, portanto não há maneiras de se escapar de possíveis erros.

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.

Logo em sequência, o arquivo é aberto e toda a informação contida nele coletada, começando pela busca dos dois SYNCs de referência para inicializar a coleta. Todos são armazenados através de uma classe para facilitar o armazenamento e troca das informações.

No envio dos dados, ocorre uma busca pelos dois SYNCs necessários para identificar o início do quadro DCCNET. Quando eles são identificados, a *string* é coletada completamente e enviada. No recebimento dos dados, é realizada a sincronização pelos SYNCs e as outras informações separadas uma a uma para facilitar as verificações dos parâmetros *checksum*, *id*, *flag* e *data*, porque eles são utilizados para validação da conexão e 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.

Outra curiosidade foi como realizar a leitura dos arquivos, inicialmente a tentativa era pela função *readline* para facilitar a coleta e o processamento, porém para uma base de dados muito grande foi um empecilho devido a ser lenta em relação à velocidade das tentativas de envio e recebimento dos dados entre o cliente e o servidor. Assim, a função padrão *read* foi adotada.

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()* o retorno era incorreto. Para solucionar isso, bastou utilizar o formato binário em todos os dados, facilitando a identificação de cada um dos caracteres.

Outro *trade-off* encontrado foi no algoritmo do *checksum*, pois mesmo com ele dando os resultados esperados não era possível acompanhar a velocidade de leitura dos dados, portanto ocorriam algumas resincronizações. Isso ocorre somente quando há uma volume alto de dados a serem lidos. Uma das maneiras de se contornar foi aumentar o *timeout*, entretanto isso não foi efetivo.

3 Conclusão

Todas as tecnicas e conceitos disponíveis foram aplicados. O código desenvolvido foi testado para todas as entradas disponibilizadas e possíveis outros casos, mostrando ser capaz de realizar aquilo que foi especificado.