Universidade Federal de Minas Gerais

Departamento de Ciência da Computação

Emulador de camada de enlace

**Disciplina:** DCC023 – Redes de Computadores

**Alunos:** Bruno Henrique Oliveira – 2018070708

Jhonny Bravo

**Belo Horizonte, 30 de junho de 2021**

**Introdução**

Neste trabalho iremos desenvolver um emulador de camada de enlace para uma rede fictícia chamada DCCNET. O emulador tratará da codificação, enquadramento, detecção de erros, sequenciamento e retransmissão de dados.

**Desafios**

Para nós, era a primeira vez trabalhando em um código tão complexo em Python, ainda assim escolhemos essa linguagem ao comparar as facilidades que teríamos em relação às outras linguagens. Não houve uma parte em específica que olhamos e dizemos que isso ou aquilo seria um desafio em específico, mas não por pressupor que fosse fácil, e sim por simplesmente não termos o conhecido adequado para saber o que nos esperava em cada parte do projeto.

Dessa forma, podemos considerar que nosso principal desafio foi conseguir compreender em cada etapa o que realmente era para ser feito. Além disso, um desafio posterior a esse seria como traduzir toda essa lógica para a linguagem que utilizamos.

**Dificuldades**

As dificuldades, como já era de se esperar, foram inúmeras. Em cada etapa que seguíamos, encontrávamos obstáculos. Para que possa ficar de uma maneira mais dinâmica esta documentação, iremos expor em partes algumas das dificuldades mais relevantes que encontramos durante o processo.

Uma das primeiras dificuldades que identificamos foi de fato entender o que estava acontecendo na questão de codificar em base 16, decodificar em base 16, transformar em bytes, transformar em string, empacotar etc. As transformações em si não eram complexas, mas entender de fato o que estava acontecendo foi fundamental para prosseguirmos com o projeto.

Outra dificuldade que encontramos, embora esse tenha sido mais fluida de resolver, foi a criação das funções que são chamadas no decorrer do código. Funções essas como: validateSync, validateLength, validateEcho etc. O que deveríamos enviar, qual o tipo de variável que deveríamos enviar, como retornar, o que retornar. Todas essas funções serão explicadas posteriormente no momento oportuno.

Uma dificuldade que percebemos também foi a adequação do programa para a forma como os testes seriam feitos pelo professor. Gastamos um bom tempo adequando o projeto ao que era pedido do que realmente programando a lógica do que devia ser feito. Mas entendemos que isso faz parte do escopo de entrega.

**Imprevistos**

Tivemos o mínimo de imprevistos possível durante a realização do projeto. Nossa dupla se planejou muito bem na questão de trabalho em equipe e distribuição dos afazeres durante a semana. Estabelecíamos o que devia ser feito durante um período, pensando a longo prazo na entrega do projeto.

Os únicos dois acontecimentos que podemos considerar como um “imprevisto” destacaremos o erro de CheckSum e a generalização para leitura de qualquer tipo de arquivo no input.

O imprevisto em relação ao CheckSum foi que pensávamos que ele era de uma maneira bem mais simples do que realmente era, a solução adotada foi pesquisar na internet como de fato funcionava o CheckSum e adaptá-lo ao nosso código.

O imprevisto em relação a generalização para leitura de qualquer tipo de arquivo no input era que pensávamos que deveríamos ler apenas arquivos .txt. Mas com o esclarecimento do professor percebemos que o projeto deveria ler qualquer tipo de arquivo. O maior impedimento era que nosso programa era baseado no envio de 1 byte de cada vez. Para arquivos do tipo .txt é totalmente aplicável, mas para imagens, por exemplo, o tempo de transmissão era consideravelmente maior. Além disso, se formos considerar também o print no log do servidor, o tempo era maior ainda. Dessa forma precisamos alterar também essa parte do código.

**Impedimentos**

Os dois maiores impedimentos que encontramos foram ajustar o projeto para permitir o envio e recebimento de arquivos simultaneamente e ressincronizar o enlace buscando pelas sequências de sincronização nos bytes transmitidos. Investimentos muito tempo principalmente no checkSum e na leitura de arquivos genéricos. Quando começamos a ler e tentar entender do que se tratava esses dois tópicos de funcionalidades, não encontramos tempo hábil para implementar no nosso projeto.

Dessa forma, fica compreendido que, apesar de termos executado a grande maioria das solicitações, não abordamos esses dois tópicos.

**Funções**

A seguir descreveremos o papel de cada função dentro do nosso código.

**printInputError:** Imprime o erro de chamada e explica como deve como o programa deve ser utilizado.

Texto

Descrição gerada automaticamente

**changeId:** Retorna o novo ID para o próximo pacote que será enviado. Atendendo às especificações do programa, alterna entre 0 e 1.

Tela preta com letras brancas

Descrição gerada automaticamente

**messageClose:** Envia uma mensagem de erro genérica para fechar a conexão quando necessário.

Texto

Descrição gerada automaticamente com confiança média

**validateSync:** Valida tanto o SYNC1 quanto o SYNC2 do pacote quando ele é recebido no servidor.

Texto

Descrição gerada automaticamente

**validateLength:** Valida o tamanho do pacote quando ele é recebido no servidor.

Texto

Descrição gerada automaticamente

**validateEcho:** Valida o ID e verifica a flag do pacote ecoado para o cliente.

Tela preta com letras brancas

Descrição gerada automaticamente

**validateSum:** Valida o checksum do pacote quando ele é recebido no servidor.

Texto

Descrição gerada automaticamente

**calculateCheckSum:** Calcula o checksum para um pacote em específico.

Tela preta com letras brancas

Descrição gerada automaticamente

**createChecked:** Cria um novo pacote com o checksum já calculado.

Texto

Descrição gerada automaticamente