Trabalho Prático 1 - Redes de Computadores

Priscila Izabelle De-Stefano Santos - 2011054758
Silvana Mara Ribeiro - 2011022147

2 de maio de 2018

1 Introdução

O presente trabalho trata da implementação de um emulador de uma camada de enlace da rede fictícia DCCNET. O emulador trata da codificação em base16 dos dados, do enquadramento, da detecção de erros usando checkSum, do sequenciamento, transmissão usando o algoritmo stop-and-wait e da retransmissão dos dados.

2 Implementação

2.1 Codificação e Decodificação

A codificação e decodificação em base16 são feitas logo que os dados estão para serem enviados e logo que são recebidos, respectivamente. A maior dificuldade encontrada neste momento do trabalho foi a adaptação do código implementado para que realizasse a codificação e decodificação do arquivo .png disponibilizado nos testes, visto que, em um primeiro momento, os dados eram tratados como ASCII até o momento do envio e recebimento, onde eram tratados como base16. Para resolver o problema, o dado passou a ser tratado sempre como byte e não foi mais convertido para ASCII (o que se mostrou um passo desnecessário que estava sendo tomado).

2.2 Enquadramento

Para realização do enquadramento criou-se a classe Frame. Assim que o programa é iniciado os arquivos de input são lidos e objetos da classe Frame são criados com seus respectivos atributos (sync, length, chksum, ID, flags e data). Cada frame foi criado com uma quantidade máxima de 500 caracteres de dados. A única dificuldade encontrada foi a compreensão do que o tamanho do length deveria referenciar: se à quantidade de bytes de dados ou à quantidade de caracteres. uma vez compreendido que o length se refere à quantidade de caracteres no frame, o problema foi sanado.

2.3 Detecção de Erros

A detecção de erros foi feita por meio do algoritmo checksum. Diversas dificuldades foram encontradas nesta parte da implementação. Apesar do algoritmo ter sido provido, não se informou exatamente o que o algoritmo espera como entrada e isto causou divergências entre os valores de checksum calculados pelos testes providos e entre o programa desenvolvido. Foram realizados testes externos ao emulador, primeiramente calculando manualmente o valor do checksum a partir do exemplo de frame disponibilizado no arquivo de teste hello-hex.txt. O valor resultante deste teste foi 44184, mesmo valor calculado pelo nosso emulador e que diverge do checksum 41624 presente no arquivo.Segundamente, foi utilizado um algoritmo de terceiros [1] para realizar os testes abaixo:

1. Foi calculado o checksum do frame disponibilizado no arquivo simple-hex.txt. Visto que este frame já possui os bytes do checksum preenchidos, o resultado deveria ser 0. Contudo, o resultado encontrado foi 2560.
2. Foi calculado o checksum do frame disponibilizado no arquivo simple-hex.txt sem os bytes do checksum preenchidos. O resultado obtido foi 44184.

Infelizmente, o motivo desta divergência não pôde ser identificado em tempo hábil de entrega deste trabalho.

2.4 Sequenciamento

O sequenciamento foi implementado utilizando-se o algoritmo *stop-and-wait*. Cada janela de transmissão e recepção, portanto, armazena apenas um quadro. Um novo quadro só pode ser enviado uma vez que o anterior tenha sido recebido e confirmado pelo outro nó. Caso não haja confirmação em até um segundo a partir do envio, o quadro é reenviado. Para evitar que o mesmo quadro seja recebido duas vezes por um nó, um ID é utilizado (com valores alternando entre 0 e 1) de modo que o receptor saiba que seu ack não chegou ao transmissor e não precisa armazenar o quadro novamente, apenas confirmá-lo novamente.

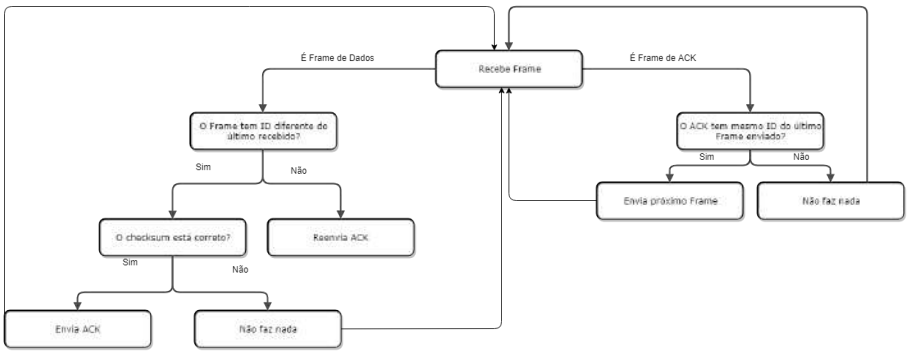

Figura 1: Fluxo de sequenciamento dos nós

A dificuldade encontrada diz respeito ao entendimento de que os enlaces deveriam ambos enviar e receber dados concomitantemente. Uma vez compreendido os papéis de cada nó, a implementação foi simples e direta. Não se fez necessário usar paralelismo para a implementação, já que se utilizou o próprio fluxo do programa (Figura 1] verificando-se se quadros recebidos eram ack's ou quadros de dados para decidir-se qual o próximo passo a ser tomado (envio ou reenvio de dados ou envio e reenvio de ack).

2.5 Retransmissão

A retransmissão acontece caso o ack do frame não chegue em até um segundo de seu envio. O ack pode ter se perdido ou pode não ter sido enviado caso o frame enviado tenha sido corrompido. Não foram encontradas dificuldades nessa parte do trabalho.

3 Testes

Os testes foram realizados de duas formas: utilizando o emulador implementado em ambos os nós e utilizando o emulador disponibilizado para testes em um dos nós. Os testes realizados com o emulador implementado foram executados com sucesso em ambientes Windows e Linux.

3.1 Evidências de Teste

3.1.1 Teste 1

Input servidor: simple.txt
Output servidor: outserver.txt
Input cliente: hello.txt
Output cliente: outcli.txt

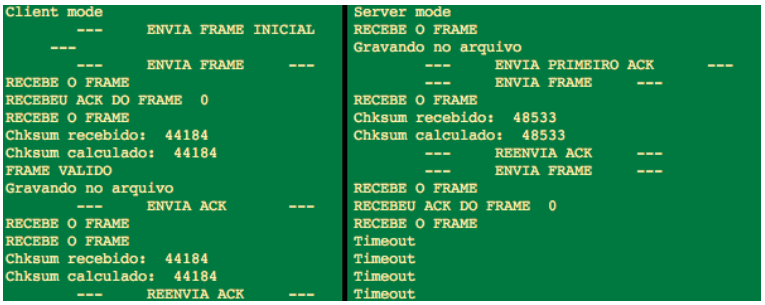

Figura 2: Teste 1

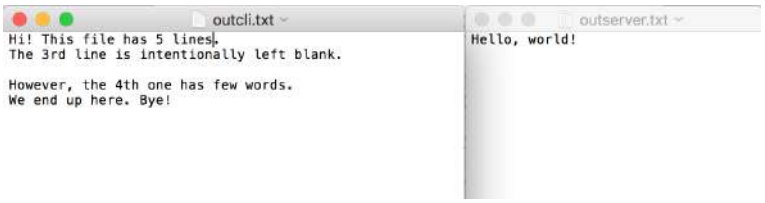

Figura 3: Arquivos Teste 1

3.1.2 Teste 2

Input servidor: hello.txt
Output servidor: outserver.txt
Input cliente: simple.txt
Output cliente: outcli.txt

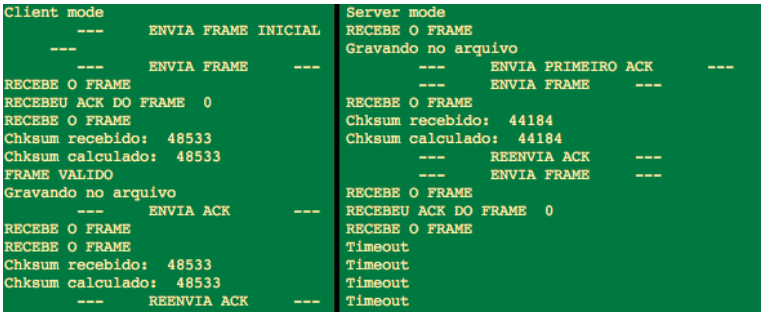

Figura 4: Teste 2

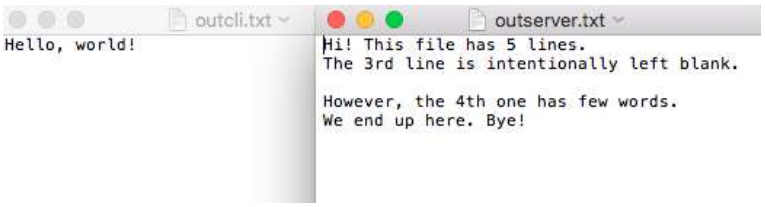

Figura 5: Arquivos Teste 2

3.1.3 Teste 3

Input servidor: lenna.png
Output servidor: outserver.txt
Input cliente: a.txt
Output cliente: outcli.png

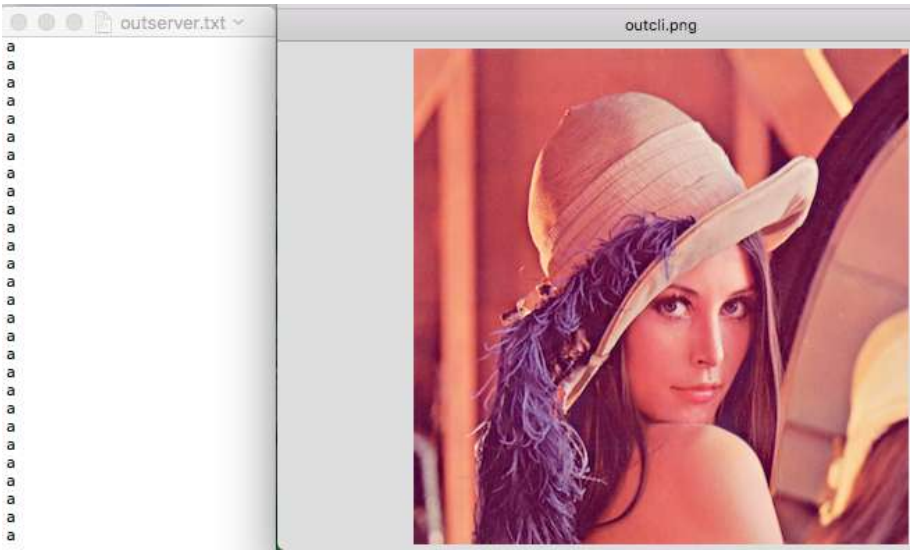

Figura 6: Arquivos Teste 3

4 Conclusão

O presente trabalho prático contribuiu para que conceitos respectivos ao funcionamento de uma camada de enlace fossem colocados em prática, contribuindo para nossa compreensão do assunto. O emulador implementado está de acordo com a especificação, contudo, não foi possível identificar em tempo hábil de entrega do trabalho o motivo das divergências entre nossa implementação e o emulador de testes disponibilizado.

Referências

[1] "Calculate IP Checksum in Python." Network Protocols - Calculate IP Checksum in Python - Stack Overflow, `stackoverflow.com/questions/3949726/calculate-ip-checksum-in-python`.

[2] "ASCII,Hex,Binary,Decimal,Base64 Converter." ASCII Text,Hex,Binary,Decimal,Base64 Converter, `www.rapidtables.com/convert/number/ascii-hex-bin-dec-converter.html`.

[3] "Python 3 : Convert String to Bytes." Python 3 : Convert String to Bytes – Mkyong.com, `www.mkyong.com/python/python-3-convert-string-to-bytes/`.