Simulador de Camada de Enlace

Teleinformática e Redes 1 - 2022/2

Integrantes

David Herbert de Souza Brito - 200057405

João Marcos Melo Monteiro - 130143031

Luigi Paschoal Westphal de Oliveira - 190062894

DEPARTAMENTO: Departamento de Ciência da Computação – CiC/UnB
DISCIPLINA: Teleinformática e Redes I
CÓDIGO: 204315
PROFESSOR: Marcelo Antônio Marotta

Introdução

A fonte do sinal de comunicação possui uma sequência de bits (mensagem) a ser transmitida; esta Camada de Enlace Para a segunda etapa deste trabalho tem-se o desenvolvimento que deve ocorrer para a camada de enlace, ela deve realizar a continuação da camada de protocolo. Esta camada tem como objetivo permitir que a comunicação de forma eficiente e confiável entre as unidades de informação inteira, esta unidade são os quadros. Diante disso a camada de enlace de dados deve usar os serviços da camada física para que seja feito o envio e o recebimento de bits através do canal de comunicação. Sendo assim a camada de enlace recebe os pacotes da camada de rede e os encapsula em quadros para que seja feita a transmissão. Cada quadro possui o cabeçalho (header), um campo de carga útil que deve conter o pacote e um final (trailer) de quadro. Como a camada física recebe o fluxo de bits brutos e deve enviar esse fluxo para a camada de enlace, esta deve verificar se existe algum erro no fluxo de bits para que após isso é que ele seja enviado para a camada de rede. Caso exista algum erro a camada de enlace deve ser capaz de corrigi-los. Para que isso seja feito, a camada de enlace divide o fluxo de bits que é recebido pela camada física, essa divisão é feita em quadros. Logo, nota-se que a string é o quadro para a implementação. Diante disso, para a segunda etapa deve ser desenvolvido os protocolos para a implementação do enquadramento na camada de enlace.

Especificações do trabalho

Camadas implementadas

```
g++ -o simulador simulador.cpp
```

```
./simulador
```

- simulador.cpp
- camadaenlace.cpp
- camadaenlace.h

Essa parte do trabalho consiste em criar um simulador em c++ que desenvolva o funcionamento da camada de enlace, sendo implementada algumas funções:

- Enquadramento por contagem de caractere
- Enquadramento por inserção de bytes
- Desenquadramento por contagem de caractere
- Desenquadramento por inserção de bytes
- Correção de Erro usando Bit de paridade par
- Correção de Erro usando CRC
- Código Hamming

Implementação da Camada de enlace

Contagem de caracteres Como esse método utiliza um campo no cabeçalho para que seja possível determinar o número de caracteres que o quadro total possui. Dessa forma, quando a

camada de enlace do receptor sabe quantos caracteres devem ser recebidos e onde está o fim do quadro. Para essa implementação foi considerado que o quadro tivesse um tamanho de 32 e a quantidade de divisões fossem o valor do tamanho dividido por 8. Dessa forma, enquanto a quantidade de sub quadros fosse maior do que zero o enquadramento continuaria para valores menores do que 8. E para realizar o desenquadramento foi feito um laço de repetição que transfira o valor do novo quadro para o quadro inicial, dessa forma tem-se o quadro desenquadrado.

Inserção de bytes ou caracteres

Para realizar a implementação da inserção de bytes foi necessário inicialmente como é a teoria para essa implementação, logo, o que se tem é que cada quadro deve começar com um byte especial, este byte é o byte de flag. Esse byte de flag que é inserido auxilia para resolver o problema de ressincronização após um erro, fazendo dessa forma que cada quadro comece e termine com bytes especiais. Quando o receptor perder a sincronização ele procura a mensagem perdida através do byte de flag a fim de descobrir o fim do quadro atual. Quando encontrado dois bytes de flag consecutivos isso indica o fim do quadro atual e início do quadro seguinte. Na sua implementação foi declarada um byte de flag e um byte de escape, eles são inseridos no momento do enquadramento e são retirados do quadro no momento do desenquadramento.

Para a implementação do tipo de controle de erro na camada de enlace foram desenvolvidas as implementações para o bit de paridade e o CRC. Na implementação foi adicionado o código de controle de erro depois do enquadramento.

Bit de Paridade Par

Seguindo a base teórica da definição do bit de paridade, tem-se que esse tipo de controle de erro permite a detecção de erros individuais, logo, tem-se como princípio desse método a inserção do bit de paridade ao final do fluxo de bits. Para o trabalho foi implementada a função de bit de paridade par para controle de erro. O bit de paridade deve ser escolhido de forma que o número de bits transmitidos seja par ou seja ímpar. Logo, caso o seguinte fluxo de bits seja 1010101, tem-se 4 bits 1 para que o resultado possua paridade ímpar, deve ser adicionado o bit 1 ao final do fluxo de bits, logo seria: 10101011 e para que ele tenha paridade par deve ser inserido o bit 0, assim o fluxo seria 10101010, sendo assim teríamos 4 bits 1, sendo esta a paridade par. Logo na implementação foi considerado que caso seja de paridade par, nada seria alterado na codificação e caso o bit fosse 0 e caso o bit fosse 1 o bit alterna entre 0 e 1. A implementação foi feita considerando a combinação lógica xor. Para a implementação da decodificação foi feita a função inversa que foi feita na codificação de bit de paridade par. No momento que foi desenvolvida a decodificação não foi criado um erro no meio de comunicação, logo o esperado é que não haja erro na transmissão da mensagem. Vale ressaltar que a implementação da decodificação foi feita na camada de enlace receptora.

CRC

O CRC, refere-se a verificação cíclica de redundância, também conhecida como código polinomial, sendo este um método utilizado para a detecção de erro para identificar quando existe alteração em um fluxo de dados. Quando o CRC é empregado o transmissor e o receptor devem concordar em relação ao grau do polinômio gerador. Para realizar um envio de mensagem

como o CRC é necessário considerar que existe uma mensagem a ser enviada e existe uma flag que é apontada como o gerador. Para encontrar o CRC é necessário dividir a mensagem a ser enviada pelo gerador, no processo da divisão é utilizada a operação xor para realizar a subtração dos termos. Ao final da operação, quando no resto tem como resultado 000 isso indica que não há erros na execução do código, quando diferente de 000 isso indica que ocorre erro no código. A mensagem final a ser transmitida corresponde ao fluxo de bits acompanhado do CRC ao final do código. Para a implementação foi definido um gerador , sendo ele de 1001. A mensagem transmitida como 00000000. Ao realizar a divisão o CRC é de 0000, logo a mensagem transmitida é a mensagem de origem concatenada ao CRC, logo 00000000 – 0000. Nesse caso é indicado que não tenha nenhum erro na transmissão da mensagem. Para esse caso o meio de comunicação é apresentado como na implementação da parte 1 do trabalho, ou seja, nenhum erro foi provocado no momento dessa implementação.

Para verificar se a mensagem recebida tinha sido obtida com ou sem erro, foi feito o processo inverso da codificação, sendo assim , a decodificação para que fosse verificado se a mensagem foi recebida e qual foi e se ela possuía algum erro ou não. Até este ponto não foi modificado o meio de comunicação para obter a decodificação do CRC. A parte de decodificação foi implementada na camada de enlace receptora.

Código de Hamming

O código de Hamming é um modelo de correção de erro, ele permite que seja detectado e corrigido o erro do quadro que foi enviado. O quadro é composto por bits de paridade e bits de verificação que são utilizados seguindo regras especiais. Um quadro que possui 11 bits possui 3

bits de paridade, para definir quais são esses bits de paridade são utilizados os bits que são potência de dois, logo os bits 1 , 2 , 4 e 8 são os bits de verificação. Os demais bits, 3 , 5 , 6, 7, 9, 10 e 11 são referentes aos bits de dados. No momento de determinar qual bit ocupará o lugar dos bits de paridade, é feita uma soma de bits. Logo, por padrão, segue como deve ser feita a soma de bits que devem ocupar a posição no bit de paridade. Logo, para: P1 = M3 + M5 + M7 + M9 + M11 P2 = M3 + M6 + M7 + M10 + M11 P4 = M5 + M6 + M7 P8 = M9 + M10 + M11 Ao realizar a soma dos bits o que é avaliado é a paridade da soma dos bits analisados, dessa forma, caso o bit inserido possua número par de bits 1, a paridade é par, logo a soma é 0. Caso contrário, a paridade é ímpar, logo a soma é 0. Esses valores são preenchidos e formam a codificação de hamming. Isso ocorre quando a codificação ocorre para a codificação de Hamming. Para os demais casos, segue a tabela:

Combinações de parâmetros do códigos de Hamming		
n	***k***	***N* = n + k**
Bits de dados	**Bits de paridade**	**Total da mensagem**
1	2	3
4	3	7
11	4	15
26	5	31
57	6	63
120	7	127
247	8	255

Figura 1. Código de Hamming.

O total da mensagem corresponde ao comprimento da mensagem que deve ser utilizado, sendo ele de no máximo 255 bits. Para implementação foi considerada apenas a codificação de Hamming, logo foram implementadas as somas que devem ser feitas no bit de paridade e o quadro que deve ser retornado. E para a decodificação foi implementado o código que extrai os valores que devem preencher os demais bits, sendo eles o M3, M5, M6, M7, M9, M10 e M11.

Simulador

No simulador foi deixada a execução mais básica, de forma que ao compilar o arquivo do simulador.cpp tem-se a união de todos os arquivos, sendo eles a camadafisicaprotocolo.cpp e a camadafisicaprotocolo.h. Foram inseridas os arquivos com camadaenlaceprotocolo.cpp e camadaenlaceprotocolo.h, no simulador.cpp ou simuladorenlace.cpp a chamada foi alterada para trabalhar diretamente com a implementação da camada de enlace.

No simulador para os testes iniciais de implementação foi feita a chamada direta da CamadaEnlaceDadosTransmissora, ela realiza a chamada para as demais funcionalidades da camada de enlace, logo ela realiza a chamada para a CamadaEnlaceDadosTransmissoraEnquadramento que conforme o usuário seleciona o tipo de enquadramento o código segue seu fluxo para enquadrar e desenquadrar a mensagem. Para a compilação, é necessário estar no diretório com os arquivos e abrir a chamada g++ -o teste simulador.cpp e sequencialmente chamar ./teste.

Especificação do SO (Sistema Operacional)

Nome do dispositivo	DELL-459451
Processador	Intel(R) Core(TM) i3-7100 CPU @ 3.90GHz 3.91 GHz
RAM instalada	4,00 GB (utilizável: 3,87 GB)
ID do dispositivo	CA2C0D7A-9CCC-48A4-B3C4-34162E7EE57D

ID do Produto	00330-50900-48364-AAOEM
Tipo de sistema	Sistema operacional de 64 bits, processador baseado em x64
Caneta e toque	Nenhuma entrada à caneta ou por toque disponível para este vídeo

Implementação

A implementação foram feitas algumas alterações para que fossem chamadas as funções seguindo a ordem da camada de aplicação com a camada de enlace e a camada física. Logo na função simulador.cpp ou simuladorenlace.cpp ele chama a camada de AplicacaoTransmissora() que agora chama diretamente a camada de enlace para que seja escolhido o tipo de enquadramento e sequencialmente a correção de erro é demonstrada e posteriormente é feita a chamada da camada física que foi implementada no trabalho 2 e este é o trabalho 3 de TR1, dessa forma é seguida a sequência de implementação das camadas que estão sendo trabalhadas.

Para compilação foi toda feita via terminal, sendo necessário apenas estar dentro do diretório que contém todos os arquivos do simulador:

- camadafisicaprotocolo.cpp

- camadafisicaprotocolo.h

- simulador.cpp

- simuladorenlace.cpp

- camadaenlaceprotocolo.cpp

- camadaenlaceprotocolo.h

Para compilar, siga o seguinte comando:

>g++ -o simulador simulador.cpp

>./simulador

>g++ -o simulador simuladorenlace.cpp

>./simulador

Nome do simulador:

Simulador de Camada De Enlace

O projeto foi criado no VScode e compilado no wsl.

Para compilar, basta rodar:

→make

Para executar:

→make run

Para limpar o diretório:

→make clean

```
#COMPILER
CPP=g++

# Nome do projeto
SIM=simulador

#FLAGS G++
FLAGS= -c -Wall -pedantic

all: $(SIM)
```

```
$(SIM): simuladorenlace.o
 camadaenlaceprotocolo.o
	$(CPP) -o $@ $^

simuladorenlace.o: simuladorenlace.cpp
camadaenlaceprotocolo.hpp
	$(CPP) -o $@ $< $(FLAGS)
camadaenlaceprotocolo.o: camadaenlaceprotocolo.cpp
camadaenlaceprotocolo.hpp
	$(CPP) -o $@ $< $(FLAGS)


run:
	./$(SIM)

clean:
	rm -rf *.o $(SIM)
```

Atividades dos Membros

Descrição das atividades desenvolvidas por cada membro do grupo:

David Herbert

Desenvolveu o código .h e ajudou na implementação do relatório.

João Marcos

Desenvolveu códigos .cpp, .h e ajudou na implementação do relatório.

Luigi Paschoal

Desenvolveu o código .cpp, .h e ajudou na implementação do relatório.

TODOS

- Implementação de todas as camadas
- Para a camada física:

  CamadaFisicaReceptoraDecodificacaoBinaria,CamadaDeAplicacaoReceptoraDecodificacaoManchester,CamadaDeAplicacaoReceptoraDecodificacaoBipolar,CamadaFisicaReceptora,MeioDeComunicacao,CamadaFisicaTrasnmissoraCodificacaoBinaria,CamadaDeAplicacaoTransmissoraCodificacaoManchester,CamadaDeAplicacaoTransmissoraCodificacaoBipolar,CamadaFisicaTransmissora,CamadaDeAplicacaoTransmissora,AplicacaoTransmissora,AplicacaoReceptora e CamadaDeAplicacaoReceptora.
- Para a camada de enlace:

  CamadaDeEnlaceTransmissoraEnquadramentoContagemDeCaracteres,CamadaEnlaceDadosTransmissora,CamadaEnlaceDadosTransmissoraEnquadramento,CamadaEnlaceDadosReceptoraDesenquadramento,CamadaEnlaceDadosReceptora,CamadaEnlaceDadosReceptoraDesenquadramentoContagemDeCaracteres,CamadaDeEnlaceTransmissoraEnqua

dramentoInsercaoDeBytes,CamadaEnlaceDadosReceptoraDesenquadramentoInsercaoDebytes,CamadaEnlaceDadosTransmissoraControleDeErroBitParidadePar,CamadaEnlaceDadosTransmissoraControleDeErroCRC, CamadaDeEnlaceTransmissoraControleDeErro,CamadaEnlaceDadosReceptoraControleDeErroBitParidadePar,CamadaEnlaceDadosReceptoraControleDeErroCRC,CamadaEnlaceDadosTransmissoraControleDeErroHamming e CamadaEnlaceDadosReceptoraControleDeErroDeHamming.

Conclusão

Este trabalho teve o objetivo de mostrar o desenvolvimento que atingiu grande parte dos seus objetivos, tornando possível realizar codificação e decodificação de mensagens utilizando o simulador desenvolvido. E a partir deste trabalho foi possível entender melhor como funciona a camada física e o link como se deve comportar. Desta forma, é possível iniciar um sistema de comunicação entre sistemas finais, além de como a comunicação deve ocorrer na camada de dados. A aplicação, com uma camada de linha e uma camada física, permite criar simulador devido ao fato de que diferentes protocolos devem ser seguidos para execução do sistema. Entre as dificuldades que encontraremos durante o desenvolvimento do trabalho estão: compreensão e como obter a codificação o resultado esperado e determinar como seria o resultado esperado e determinar o que deveria ser corrigido o bug durante a implementação.

Além disso, foi possível fortalecer e aprimorar o entendimento da camada física e link e seus respectivos protocolos que podem ser implementados e como comunicação do transmissor ao receptor, sem perda de informações do e se a transmissão for perdida, a camada de enlace implementou alguns tática para verificar o erro e, se possível, corrigi-lo usando o código Hamming.

Referências bibliográficas

- https://edisciplinas.usp.br/pluginfile.php/5228477/mod_resource/content/1/2020_Aula_05_Camada_Fisica_Parte_3.pdf
- https://acervolima.com/diferenca-entre-esquemas-de-codificacao-de-linha-unipolar-polar-e-bipolar/
- http://www.decom.ufop.br/reinaldo/site_media/uploads/2012-01-bcc361/parte3_camada_enlace_6ppf.pdf
- https://www.inf.pucrs.br/~cnunes/redes_si/aulas/Enlace3.pdf
- https://www.gta.ufrj.br/~miguel/docs/redes1/aula4.pdf