Universidade de Brasília

Simulador de Camada de Enlace (Controle de Erros)

Teleinformática e Redes 1

Prof. Geraldo Pereira

João Pedro Assis dos Santos (17/0146367)
Pedro Augusto Ramalho Duarte (17/0163717)
Waliff Cordeiro Bandeira (17/0115810)

Brasília
2020

Introdução

Os meios de comunicação com e sem fio precisam ter a capacidade de propagar sinais analógicos e por meio desses sinais analógicos encaminhar informações digitais através da modulação digital que por sua vez utiliza conceitos de canal de banda base e canal de banda passante.

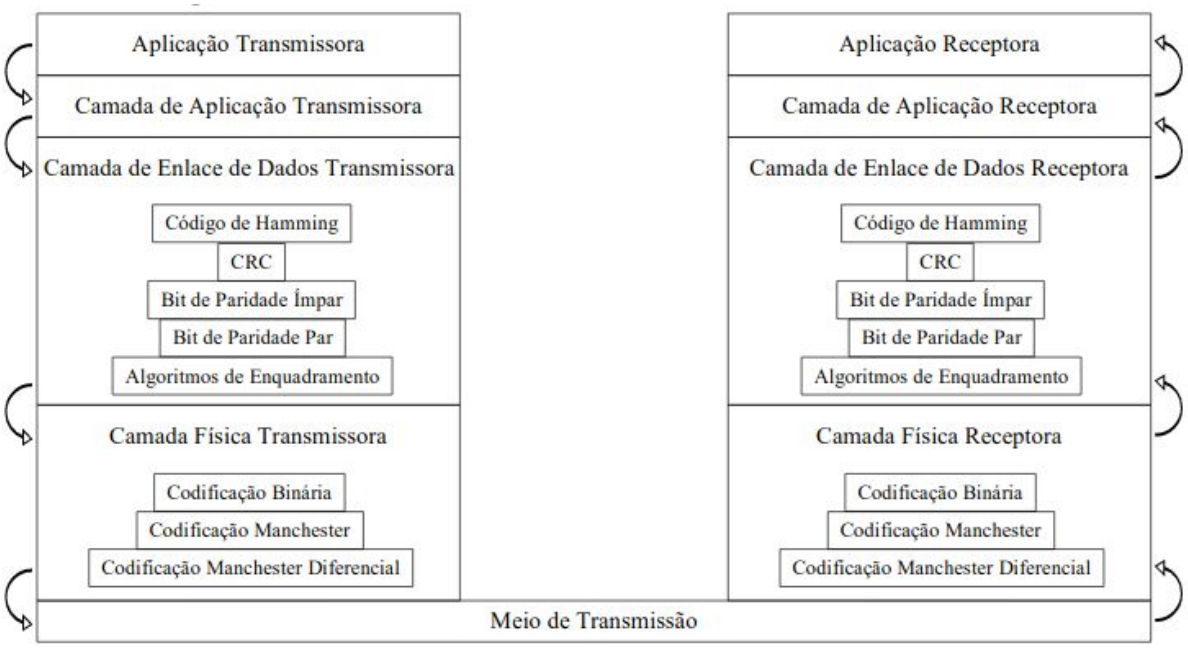
A transmissão banda base é conhecida também como transmissão sem modulação, pois utiliza apenas um processo de codificação banda base (codificação de linha). A codificação de banda base é baseada na temporização (clocking), que determina o início e fim da transmissão dos bits para que possam ser decodificados, ou seja, determina o sinal e o nível do sinal (0 ou 1). Elas podem ser categorizadas em três distintivos níveis: curta distância (sensível ao sinal), médias distâncias (sensível à fase) e longas distâncias (códigos de blocos).

Para codificação de bits em símbolos é de extrema importância que o receptor consiga definir bem o momento que um símbolo termina e outro começa para que os bits sejam decodificados coerentemente. No primeiro trabalho prático foi desenvolvido o funcionamento do enlace físico através da implementação das seguintes codificações: Binária, Manchester e Manchester Diferencial.

Para a Camada de Enlace de Dados Transmissora serão explorados três diferentes algoritmos: Contagem de caracteres, inserção por bytes e inserção por bits. A contagem de caracteres consiste em colocar a quantidade de bytes que contém a mensagem no cabeçalho (primeiro byte) do quadro. Note que por definirmos o tamanho do quadro como um byte, o cabeçalho sempre será igual a 1. A inserção por bytes conta com a inserção de um byte de flag antes e após o campo de carga útil, caso essa flag esteja no campo de carga útil (seja um dado), deve-se colocar um byte de esc (escape) para indicar que aquela flag é um dado da mensagem, o mesmo ocorre caso o esc seja um dado do campo de carga útil, outro esc deve ser precedido. No caso da inserção de bits, quando temos um bit ‘0’ e cinco bits ‘1’ consecutivos no campo de carga útil, acrescenta-se um bit ‘0’ para sabermos que trata-se do campo de carga útil. Esses conceitos e sua implementação serão melhor abordados nos tópicos abaixo.

Os canais de comunicação possuem peculiaridades no quesito taxa de erros, independente do canal de comunicação, ocorrem erros. Tendo esse motivador, precisamos lidar com os erros, as formas mais comuns são inserção de dados redundantes para detecção e/ou correção. No terceiro trabalho exercitamos esses conceitos através de controle de erros baseados no **CRC**, **bit de paridade** e **código de Hamming**.

Para o terceiro trabalho, seguiremos a implementação do controle de erros na camada de enlace como descrita na imagem a seguir



Implementação

A implementação do projeto utilizou práticas comuns no design orientado a objetos, explorando funcionalidades do C++ que agilizaram o desenvolvimento e tornaram-o mais dinâmico. A arquitetura do sistema seguiu o padrão Façade, que foi de alta utilidade para uma implementação mais limpa e modular.

O sistema tem como base a classe **Simulação,** que anteriormente guardava ponteiros para quatro interfaces, a **CamadaFisicaTransmissora** e a **CamadaFisicaReceptora, CamadaEnlaceTransmissora** e **CamadaEnlaceReceptora** e nesse trabalho foram adicionadas mais quatro interfaces relacionadas ao **controle de erros**. Durante sua inicialização, o construtor recebe o tipo de codificação, o tipo de enlace e o método de controle de erro que será realizado e, com base nisso, inicializa as interfaces.

Camada de Enlace

Contagem de caracteres

Nesse algoritmo o tamanho do quadro é o tamanho da mensagem que é recebida, com objetivo do método

Nele é colocado um cabeçalho com o tamanho da mensagem que no máximo pode ser de 255, representando o tamanho da mensagem em bytes.

Inserção por Bytes

Na inserção por byte definimos o quadro com o tamanho fixo de **1 byte** (8 bits). A **flag** foi escolhida sendo **01000000** (**@**) e o **esc** foi escolhida sendo **00100011** (**#**). A implementação do algoritmo consiste na inserção de um byte de flag antes e após o campo de carga útil, caso essa flag esteja no campo de carga útil (seja um dado), deve-se colocar um byte de esc (escape) para indicar que aquela flag é um dado da mensagem, o mesmo ocorre caso o esc seja um dado no campo de carga útil, outro esc deve ser precedido para indicar isso.

Inserção por Bits

Na inserção de bits foi definido o tamanho do quadro como fixo em 1 byte. Quando temos um bit ‘0’ e cinco bits ‘1’ consecutivos no campo de carga útil (011111), acrescenta-se um bit ‘0’ (0111110) para sabermos que trata-se do campo de carga útil. Antes e após cada quadro (que consiste em 1 byte) temos um byte (8 bits) de flag (01111110), sendo assim, fica possível diferenciar a flag do campo de carga útil e fazer a recepção detectando as possíveis falhas de transmissão.

Controle de Erros

Bit de paridade (par e ímpar)

A estratégia mais simples que permite a detecção de erros é o bit de paridade, ele consiste em apenas um bit (o último) para fazer essa identificação de erro. Ainda no bit de paridade temos o conceito de paridade par ou paridade ímpar. No método de bit de paridade par, a soma dos bits ‘1’ sempre precisam dar 0, ou seja, se temos 11 o bit de paridade será 0 e se temos 01 o bit de paridade será 1 para, ao somar com os demais bits ‘1’ termos uma quantidade par números 1. A paridade ímpar é o exato oposto dessa estratégia, ao somar os números ‘1’ precisamos obter um número ímpar, de forma que se tivermos 11 o bit de paridade será 1 e se tivermos 01 o bit de paridade será 0.

Código de Redundância Cíclica (CRC)

Também conhecido como código polinomial. Quando pensa-se no CRC, temos que na camada de enlace, o transmissor e o receptor devem concordar em relação a um polinômio gerador G(x). O bit de mais alta e mais baixa ordem devem ser iguais a ‘1’ e a quantidade de bits do quadro deve ser maior que a quantidade de bits do gerador. Na divisão dos bits, uma peculiaridade, é que este método utiliza o a operação XOR ao invés da subtração, ou seja, ao fazer a divisão do receptor pelo gerador, aplicando operações XOR ao invés de subtrações, se o resultado for zero, não houve nenhum erro na transmissão ou recepção. Nesse trabalho, utilizamos o polinômio 0xEDB88320, conhecido como polinômio CRC-32 reverso. Isso foi necessário, pois utilizamos uma arquitetura *MSB first*, ou seja, o bit menos significativo primeiro. Tendo isso em vista, não seria possível utilizar o polinômio convencional.

Código de Hamming

O código de Hamming é o único dos métodos que além de conseguir detectar, também consegue corrigir o erro. Vários bits de paridade / verificação são acrescidos, temos então um quadro com x bits de dados e y bits redundantes (os de verificação), ou seja, o tamanho de bits total do quadro é a quantidade de bits de dados somada à quantidade de bits redundantes. O total do quadro é chamado de palavra de código, os bits da palavra de código são enumerados em ordem crescente da esquerda para a direita. Temos então que um bit de verificação contribui em mais de um tipo de dado e isso é feito pela base binária, ou seja, um bit k = 5 irá contribuir para posição 4 e 1. Assim sendo, cada bit de verificação força a paridade do conjunto de bits. Como o exemplo mostrado em aula:

P1 = M3 + M5 + M7 + M9 + M11
P2 = M3 + M6 + M7 + M10 + M11
P4 = M5 + M6 + M7
P8 = M9 + M10 + M11

1	2	3	4	5	6	7	8	9	10	11
P1	P2	M3	P4	M5	M6	M7	P8	M9	M10	M11

Percebemos que o bit de verificação P1 tem influência sobre os bits de dado M3, M5, M7, M9 e M11 e são dados por potência de 2. Então soma-se os bits que o bit de validação indica e verifica-se a paridade dessa soma. O resultado das verificações de paridade indicam exatamente qual foi o bit que ocorreu o erro.

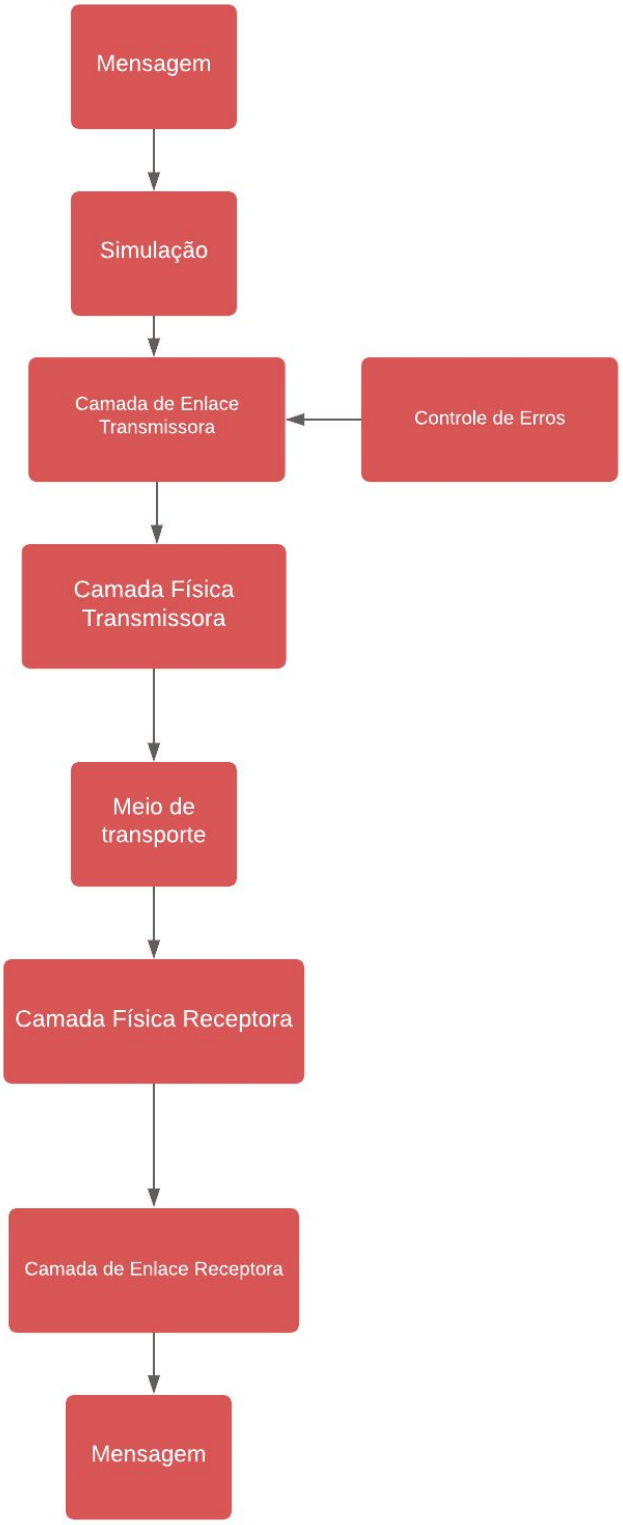
No caso de nossa implementação, estendemos o caso base para o tamanho de bits necessário (variável) para transmitir e receber a mensagem.

As classes feitas neste trabalho foram desenvolvidas pelo mesmo modelo do anterior e vamos exemplificar novamente utilizando a **CamadaTransmissoraControleErro** que é uma classe virtual pura, ou seja, não pode ser instanciada diretamente e serve apenas como uma interface para classes filhas. Por sua vez, as classes filhas herdam o método **execute()** e o sobrescrevem, fazendo com que a implementação desse método seja dependente da classe que o invoca.

Dessa maneira, após instanciar a classe **Simulação**, sempre chamaremos **transmissora->execute()** e o compilador irá decidir, em tempo de execução, qual método executar. Essa prática é interessante, pois abstrai o método de codificação utilizado. E é aplicada essa mesma hierarquia para as receptora e transmissora, das camadas de enlace e física.

Utilizando todas as classes supracitadas, o fluxo do código é dado por:

Fluxograma



Membros

Ao início do projeto foi utilizada uma técnica do Extreme Programming (XP) conhecida como pair programming, a diferença é que ao invés de desenvolver com dois, o desenvolvimento ocorreu com três desenvolvedores de forma simultânea utilizando a plataforma discord para conversa e compartilhamento de tela.

O momento inicial citado foi utilizado para definição da arquitetura, técnicas e tecnologias que seriam utilizadas para o desenvolvimento do projeto. Foi feita então as sessões de pair programming necessárias para a implementação de toda a arquitetura do projeto. A linguagem utilizada foi C++ com paradigma orientado à objetos e durante o desenvolvimento de todas as classes e a conclusão da arquitetura principal (a interface que consistiria em todo o enlace da camada física, o esqueleto do projeto) foi feito de forma conjunta.

Após a sessão inicial, na qual havia sido trabalhado os conceitos da disciplina para implementação da arquitetura da camada de enlace físico, o Pedro Augusto fez os métodos de acesso bit a bit, codificação binária e a refatoração/documentação de todo o código realizado, o João Pedro ficou responsável por terminar as codificações restantes (Manchester e Manchester Diferencial) e o Waliff pela confecção do artefato relatório.

Apesar de termos algumas tarefas atribuídas a desenvolvedores específicos, o trabalho foi feito de forma bem conjunta. Inicialmente iríamos utilizar a metodologia Scrum para organizar e distribuir melhor as tarefas de cada iteração, mas ao fazer as sessões iniciais de pair programming e com o auxílio das facilidades de gerência e versionamento que o GitHub proporciona, achamos que não seria tão necessário uma definição tão engessada da gerência para a demanda desse projeto.

Na implementação referente à camada de enlace, foi dividido o trabalho previamente, sendo que o Pedro Augusto ficou responsável pela contagem de caracteres, o João Pedro pela inserção por bits e o Waliff Cordeiro pela inserção por bytes. Sempre que necessário, fazíamos sessões de pair programming para que todos soubessem o que estava sendo tratado em todo o código, tendo um entendimento completo do desenvolvimento e do conteúdo tratado, além de ajudar no desenvolvimento de todas as partes.

Ao desenvolver o controle de dados distribuímos as tarefas entre os membros da equipe após uma reunião inicial, na qual todos entenderam o conceito de cada controle de erro proposto e qual seria o caminho seguido para o desenvolvimento. Foram feitas, assim como nas implementações do trabalho 1 e 2, sessões de pair programming para que todos tivessem acesso à todo conteúdo da disciplina e contribuíssem na implementação de todos os tipos de controle de erros implementados.

Conclusão

O desenvolvimento nos proporcionou exercitar conceitos relevantes sobre redes e o funcionamento do enlace físico. Através da implementação pôde-se perceber na prática alguns conceitos e dificuldades de transmissão e implementação de um enlace de camada.

Tivemos a oportunidade de exercitar os conceitos de transmissão banda base que também é conhecida como transmissão sem modulação. Verificamos na prática a importância de que a codificação de bits em símbolos aconteça com a melhor precisão possível pelo receptor, pois é de nosso interesse saber o momento que um símbolo termina e outro começa para que os bits sejam decodificados com maior assertividade.

Tentando garantir uma maior precisão, várias codificações e protocolos foram desenvolvidos ao longo dos anos. O exercício para trabalhar esses conceitos foi desenvolvido através da implementação das codificações Binária, Manchester e Manchester Diferencial.

A camada de enlace é responsável pela transmissão e recepção de quadros, assim como delimitá-los, e pelo controle de fluxo, ela detecta e pode corrigir eventuais erros que possam aparecer na camada física. Os conceitos de camada de enlace foram trabalhados com base em três conceitos: contagem de caracteres, inserção por bytes e inserção por bits.

O desenvolvimento do projeto utilizando a linguagem de programação C++ em um paradigma de POO, utilizando o design pattern Facade e conceitos de Teleinformática e Redes I proporcionou também a revisão de conceitos importantes de programação. O projeto foi desenvolvido com base na classe Simulação, que guarda ponteiros para quatro interfaces, a CamadaFisicaTransmissora, CamadaFisicaReceptora, CamadaEnlaceTransmissora, CamadaEnlaceReceptora e controle de erros através do **bit de paridade**, **código de hamming** e **CRC** utilizando conceitos de herança e override para uma implementação menos verbosa e mais eficiente, permitindo no futuro que sejam adicionadas outros tipos de camadas mudando poucas partes do código.

Referências

Materiais disponibilizados ao decorrer da disciplina: livro texto e vídeo aulas ministradas pelo professor Marcelo Antonio Marotta e Geraldo Pereira.

Wikipedia. Banda base. Disponível em: <https://pt.wikipedia.org/wiki/Banda_base>. Acesso em: 30 de outubro de 2020.

Até o momento. Design Pattern Facade. Disponível em: <https://www.ateomomento.com.br/facade-padrao-de-projeto/>. Acesso em 30 de outubro de 2020.

Wikipedia. Camada de enlace de dados. Disponível em: <https://pt.wikipedia.org/wiki/Camada_de_enlace_de_dados>. Acesso em 29 de novembro de 2020.