Universidade de Brasília

Simulador de Camada Física

Teleinformática e Redes 1

Prof. Doutor 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 sejam encaminhadas informações digitais através da modulação digital que por sua vez utiliza conceitos de canal de banda base e canal de banda passante.

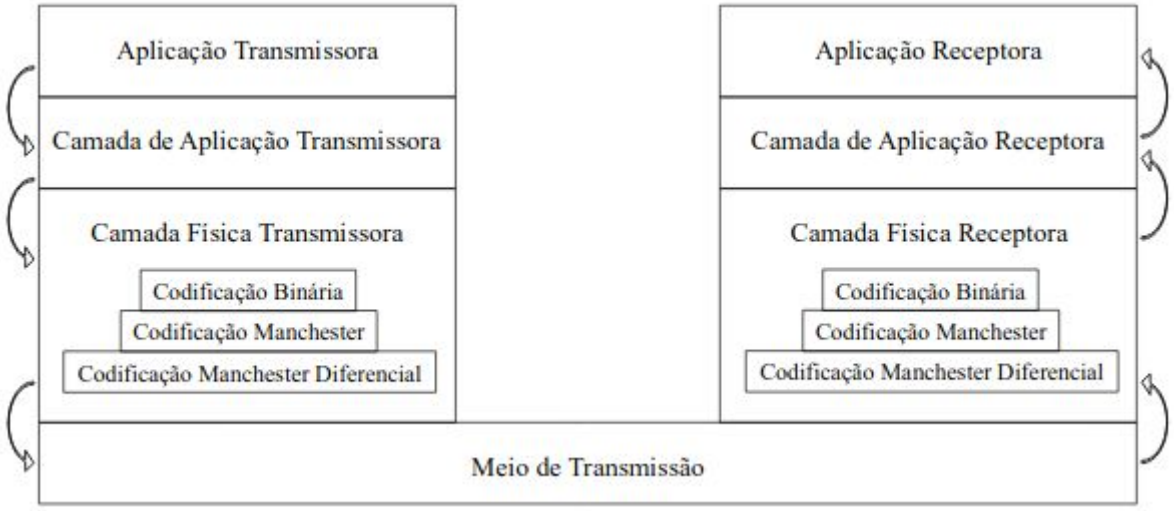
No primeiro trabalho da disciplina Teleinformática e Redes 1 iremos abordar os conceitos de canal de banda base. O canal recebe esse nome pois a própria frequência está simétrica ao eixo de frequência 0, ou seja, a base do espectro de frequência será $f = 0$ Hz. A base por sua vez vai desde $f = 0$ até f_c (uma frequência de corte) (no qual todo o sinal estará contido nessa faixa de frequência).

Temos então que a banda base é a banda de frequência responsável pelos sinais transmitidos usados para modular uma porta para transmissão. Nos aparelhos celulares, por exemplo, é responsável por administrar as comunicações de rádio e trocas de informações com a antena dos dispositivos; no iPhone, também comanda o wi-fi e bluetooth.

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. Neste trabalho prático iremos desenvolver o funcionamento do enlace físico através da implementação das seguintes codificações: Binária, Manchester e Manchester Diferencial.

Seguindo a implementação da camada de aplicação física como descrita na imagem



Implementação

A implementação do projeto utilizou práticas comuns no design orientado a objetos, explorando funcionalidades do C++ que iriam agilizar o desenvolvimento e torná-lo 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 guarda ponteiros para duas interfaces, a **CamadaFisicaTransmissora** e a **CamadaFisicaReceptora**. Durante sua inicialização, o construtor recebe o tipo de codificação que será realizada e, com base nisso, inicializa as interfaces de acordo com o método desejado.

```
Simulacao::Simulacao(int tipo_codificacao) {
   switch(tipo_codificacao) {
       case 1:
           transmissora = new CFTManchester();
           receptora = new CFRManchester();
           break;
       case 2:
           transmissora = new CFTManchesterDiferencial();
           receptora = new CFRManchesterDiferencial();
           break;
       default:
           cout << "Tipo de codificação inválido (Utilizamos o tipo de
codificação binária)" << endl;
       case 0:
           transmissora = new CFTBinaria();
           receptora = new CFRBinaria();
           break;
   }
}
```

A partir dessa inicialização, podemos perceber que os atributos **transmissora** e **receptora** recebem diferentes tipos de codificação a depender da opção fornecida. Para que isso seja possível, a seguinte arquitetura foi utilizada (tanto para a transmissão quanto recepção):

```
class CamadaFisicaTransmissora {
public:
   virtual BitArray *execute(BitArray *) = 0;
};

class CFTBinaria : public CamadaFisicaTransmissora {
public:
   BitArray* execute(BitArray *) override;
};

class CFTManchester : public CamadaFisicaTransmissora {
public:
   BitArray* execute(BitArray *) override;
};

class CFTManchesterDiferencial : public CamadaFisicaTransmissora{
public:
   BitArray* execute(BitArray *) override;
};
```

A classe **CamadaFisicaTransmissora** 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.

É possível verificar que todos os método trabalham com a estrutura (resumida) **BitArray**, descrita abaixo:

```
class BitArray {
private:
    u_int8_t *container;
    unsigned int lenght;
public:
    BitArray(const std::string &);
    ~BitArray();
    unsigned int tam();
    void setBit(unsigned int);
    void clearBit(unsigned int);
    int operator[](unsigned int);
    void operator = (const BitArray&);
}
```

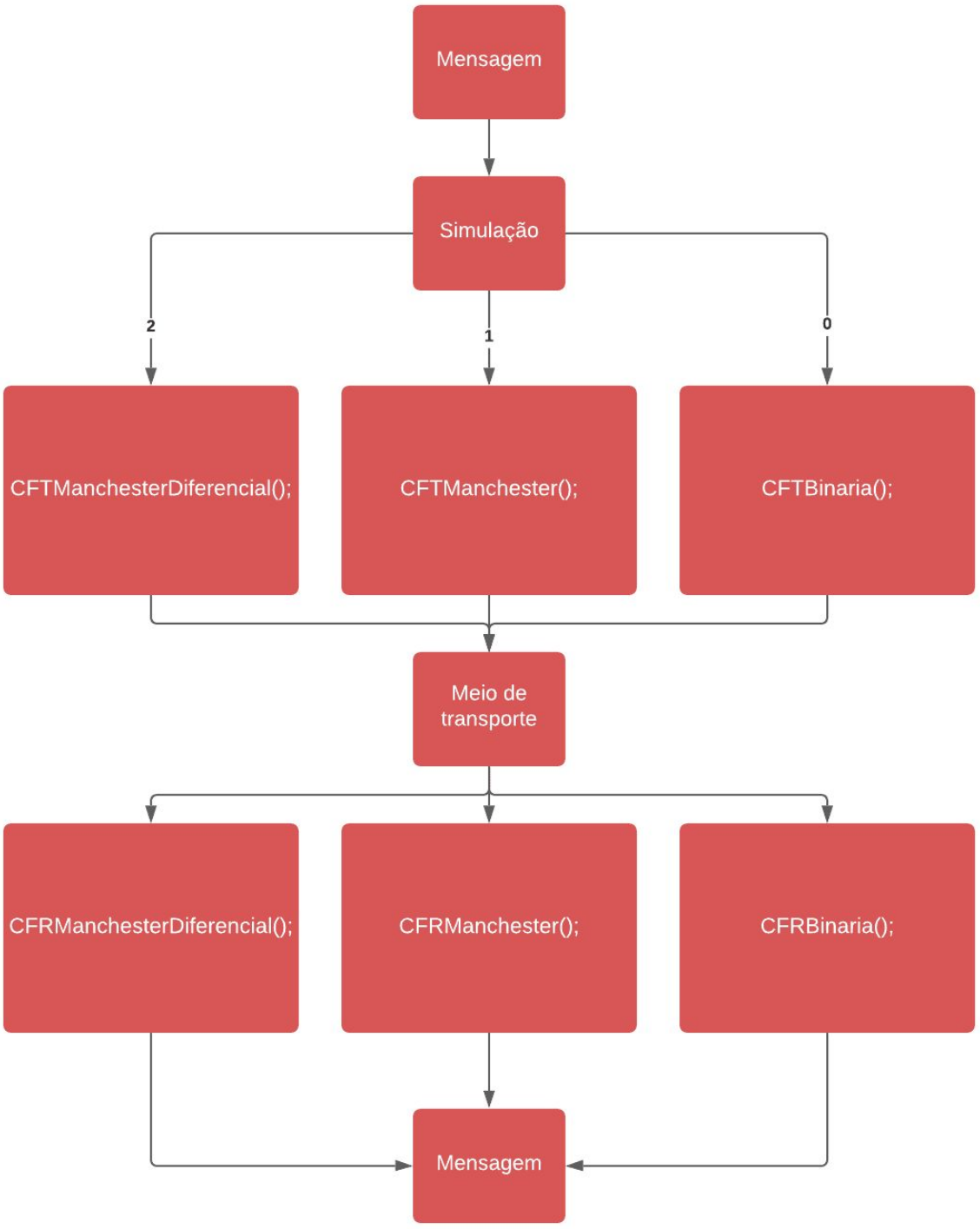
A classe **BitArray** utiliza um vetor de *u_int8_t* (onde cada índice equivale a um byte), para armazenar o código ASCII de cada letra presente na mensagem. Dessa maneira, utilizamos as funções descritas acima para realizar o acesso bit a bit de cada um desses bytes presentes no vetor.

Essas funções utilizam operações booleanas para acessar os bits. Para facilitar a utilização da classe foi utilizado o método de *operator overloading*, que sobrescreve o uso padrão de operadores do C++. A classe realiza o overloading do operador [ ], com o intuito de tornar o acesso bit a bit semelhante ao de um array ou vector<>, com os quais já estamos habituados.

O grupo optou por utilizar essa estrutura de dados visando dar uma implementação mais profissional para o problema, visto que o C++ não fornece uma maneira nativa de utilizar um vetor de bits. Com a BitArray, o acesso é abstraído e se economiza uma enorme quantidade de memória em relação a um array ou vector<> de inteiros (que desperdiçam por volta de 96% de cada endereço).

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

Fluxograma


Mensagem
Simulação
2
1
0
CFTManchesterDiferencial();
CFTManchester();
CFTBinaria();
Meio de transporte
CFRManchesterDiferencial();
CFRManchester();
CFRBinaria();
Mensagem

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 camada de enlace físico e implementação da codificação binária, o Pedro Augusto realizou a refatoração de todo o código, 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.

Após o horário de envio do projeto, o repositório ficará público para consulta através do repositório remoto:

https://github.com/SwitchDreams/simulador-camada-fisica.

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.

O desenvolvimento do projeto utilizando a linguagem de programação C++ em um paradigma de POO, utilizando o design pattern Façade 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 duas interfaces, a CamadaFisicaTransmissora e a CamadaFisicaReceptora, utilizando conceitos de herança e override para uma implementação menos verbosa e mais eficiente.

Referências

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

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.