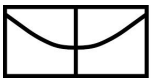


Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

Trabalho 1

Teleinformática e Redes 1

Simulador Manchesto

André Filipe da Conceição | 150005547
Gabriel Matheus da Rocha de Oliveira | 170103498
Guilherme Braga Pinto | 170162290

Brasília
2020

Introdução

O objetivo deste projeto é simular, de forma prática, como seria a implementação de uma transmissão de dados através da simulação de uma simples camadas de aplicação e uma camada física mais elaborada junto de um meio de comunicação simbólico.

O simulador é composto apenas pela camada de aplicação e física dividido em camadas menores sendo elas: Aplicação Transmissora, Camada de Aplicação Transmissora, Camada física Transmissora, Meio de Transmissão, Camada Física Receptora, Camada de Aplicação Receptora e Aplicação Receptora.

O simulador implementado funciona da seguinte forma: Primeiramente, o simulador mostra uma mensagem ao usuário, a partir da camada de aplicação, solicitando que o mesmo digite a mensagem que deseja transmitir. A mensagem é transmitida para a camada de Aplicação Transmissora que codifica a mensagem recebida a transformando em um fluxo de bits 0s e 1s e a envia para a camada Física Transmissora. Ao receber os dados, a camada realiza uma das três diferentes codificações possíveis, sendo elas Codificação Binária, Codificação Manchester e Codificação Manchester Diferencial e a envia para o Meio de Transmissão que representa um canal de comunicação simbólico.

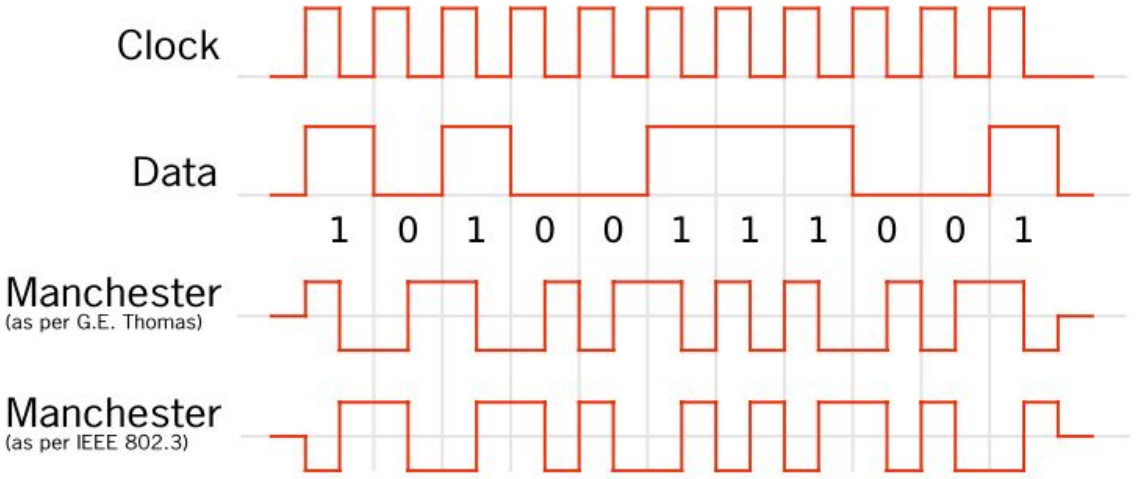
Após o envio simbólico pelo meio de comunicação, o fluxo de bits codificado é recebido pela camada Física Receptora que decodifica a mensagem de acordo com o método de codificação utilizado anteriormente e a envia para a camada de Aplicação Receptora que converte o fluxo de bits para uma sequência de caracteres ASCII. Por fim, a mensagem é exibida ao usuário por meio da camada de aplicação, completando assim, a simulação de um processo de transmissão de dados.

Descrição dos métodos de codificação utilizados:

- **Codificação Binária:** nessa codificação, a mensagem é transformada em bytes da tabela ASCII (através da função *bitset* da linguagem utilizada, C++), e enviada sem alterações, apenas contendo 0s e 1s respeitando a sequência da mensagem original. A decodificação, portanto, é feita apenas transformando os intervalos de 8 bits em caracteres através da mesma tabela ASCII.

- **Codificação Manchester:** nessa codificação, é feita uma comparação entre a mensagem codificada em bits e um clock definido previamente (neste projeto, foi utilizado o clock base 01). Na comparação, é utilizada uma função XOR (ou seja, a codificação será nível alto quando o bit analisado do fluxo é 1 e o clock é 0, ou quando o quando o bit analisado é 0 e o clock é 1). A terceira linha da imagem abaixo ilustra este processo, mas é importante ressaltar que, na imagem, é utilizado o clock 10 e não o clock 01 como neste projeto.

- **Codificação Manchester Diferencial:** nessa codificação, é preciso saber qual era o estado anterior do sinal, então quando o bit analisado é 1, há mudança no par codificado (ou seja: na mensagem a ser enviada, o par da codificação Manchester Diferencial será invertido quando o bit analisado for 1, e continuará da mesma forma caso o bit for 0). No exemplo da imagem abaixo utilizando o clock 01, a codificação ficaria (note na inversão dos pares em negrito): **10**-10-**01**-01-01-**10**-**01**-**10**-10-10-**01**.



(Fonte: File:Manchester encoding both conventions.svg)

Exemplo de resultado de execução do simulador:

```
Aplicacao Transmissora:
Digite uma mensagem:
TR1
Mensagem a ser transmitida: TR1

Camada de Aplicacao Transmissora
010101000101001000110001

Camada Fisica Transmissora:
Codificacao Manchester:
011001100110010101100110010110010101101001010110

Meio de Transmissao

Camada Fisica Receptora
Decodificacao Manchester:
001100110011000000110011000011000000111100000011

Camada de Aplicação Receptora
010101000101001000110001

Aplicacao Receptora
A mensagem recebida foi: TR1
```

Implementação

Decisões relativas ao desenvolvimento

- **Técnica de Desenvolvimento:** Foi utilizada a técnica de desenvolvimento *pair-programming*, para que erros fossem corrigidos assim que identificados e para que o desenvolvimento fosse mais ágil e completo. Assim, todos os integrantes do grupo desenvolveram todas as funções e codificações em um revezamento entre controlador e observador. Essa contribuição em grupo ocorreu de maneira remota, onde um programador controlador compartilha o trabalho em seu computador ao vivo para que todos os participantes participem. Utilizou-se a plataforma Github para manter o repositório comum entre todos os participantes do trabalho.
    - Mais informações em: Como programação pareada funciona.

- **Função auxiliar relativa ao uso da função bitset:** ao se passar um caractere para sua representação em ASCII, procurou-se utilizar uma função pronta da linguagem C++. A função de bitset acabou por ser a escolhida, porém o seu uso não facilitou em nada o uso de loops. Os dados armazenados no tipo declarado com o bitset usam menos espaço de memória, porém dificultaram o seu uso em loops como ocorre em qualquer variável de outro tipo. O resultado disso foi a necessidade da criação de uma função que recebe um endereço e o array que contém a mensagem a ser traduzida, além do endereço do fluxo de bits bruto. A cada chamada da função, cria-se como variável local o tipo bitset que traduzirá um caractere para sua representação binária de 8 bits em ASCII, e esse conjunto de bits é alocado na variável do fluxo bruto. Ao final da execução da função, o array com os bits foi atualizado e o tipo bitset foi descartado, sendo criado novamente na próxima chamada da função. Perda de performance por esta decisão de implementação não foi perceptível, pelo contrário, a performance vista foi aceitável.

- **Passando de um conjunto de bits para um caractere:** ao se codificar a passagem de um conjunto de 8 valores do tipo inteiro (sendo 0 ou 1) para seu valor equivalente em caractere do tipo ASCII, percebeu-se que uma função nativa ao C++ do tipo que seja exatamente a contrária do bitset não existia. A implementação escolhida que faria a tradução de forma mais direta de binário para caractere seria primeiro transformando o número binário em questão para apenas 1 número em uma variável do tipo inteiro, para que então esta variável passe por um cast e seja realocada em uma variável do tipo string. Itera-se então os 8 bits se acumulando em 2^x, onde x é a posição que se está analisando no momento, indo da posição 0 até a posição 7, de 8 em 8 bits no array da mensagem em ASCII. No final a string corresponde ao equivalente passado em ASCII.

- **Identificando o tamanho dos vetores:** ao implementar o vetor para transmitir o quadro para a camada física receptora, utilizou-se uma formalidade própria para identificar o final do vetor: ao final do fluxo de bits (0s e 1s), há um número 2. Esse método, que foi o mais prático para alcançar o objetivo, foi tão utilizado que se decidiu-se transformá-lo em uma função utilizada em todo o projeto.

- **Escolha da implementação a ser utilizada pelo simulador:** no arquivo de cabeçalho (.h), foi implementado um *define* para escolher de forma mais simples qual implementação (binária, Manchester ou Manchester Diferencial) será utilizada. A diretiva que define a codificação que será utilizada é conhecida ao se compilar o projeto, e por consequência atualiza também o tipo de decodificação. Dessa forma o programa nunca fará uma codificação Binária na transmissão para que se execute então uma decodificação Manchester na recepção. A definição de codificação e decodificação obedece a seguinte convenção:

Codificação/Decodificação	**Código atrelado**
Binária	0
Manchester	1
Manchester Diferencial	2

- **Codificação com XOR:** nas codificações Manchester e Manchester Diferencial, não foi utilizada uma função XOR própria da linguagem C++, e sim uma comparação "simbólica" bit-a-bit entre variáveis do tipo inteiro em um vetor. O limite de possibilidades de operações do tipo XOR facilita a codificação e decodificação Ou seja:

A (fluxo de bits)	**B (clock)**	**A XOR B**
00	01	01
11	01	10

- **Escolha do *clock* base:** na codificação Manchester e Manchester Diferencial, utilizou-se o clock base como 01. Ou seja, na codificação Manchester Diferencial se o fluxo de bits a ser codificado começa com 1, então o fluxo codificado começa com 10 (que é o contrário do clock). Dessa forma se obedece a codificação Manchester Diferencial, onde se grava apenas as mudanças de 0 para 1 e de 1 para 1. A primeira aparição de um 1 nesse caso no fluxo a ser codificado fará a "mudança" do padrão anterior tendo como referência o clock. A primeira aparição sendo a de um 0 no fluxo a ser codificado apenas repetirá o padrão definido pelo clock.

Membros

- **André Filipe:** implementação prática das codificações binária, Manchester e Manchester Diferencial; elaboração do relatório; identificação e correção de erros de desenvolvimento.

- **Gabriel Matheus:** implementação prática das codificações binária, Manchester e Manchester Diferencial; modularização e codificação da interface (terminal) e desenvolvimento do código.

- **Guilherme Braga:** implementação prática das codificações binária, Manchester e Manchester Diferencial; decisões de desenvolvimento; elaboração do código.

Conclusão

A implementação do simulador nos permitiu obter um conhecimento mais prático do funcionamento da transmissão de dados por meio das camadas de aplicação e física além do funcionamento da codificação Binária, Manchester e Diferencial. Além disso, as decisões de implementação adotadas pelo grupo permitiram que diferentes opiniões e pontos de vista fossem levados em conta para alcançar um objetivo em comum e resolver alguns problemas encontrados durante o desenvolvimento do trabalho.

Durante o desenvolvimento do simulador não houve dificuldades de compreensão ou elaboração dos algoritmos necessários para realizar as diferentes formas de codificação. Dessa forma, o maior obstáculo no projeto não consistiu no entendimento da base teórica do problema, mas na manipulação e conversões entre caracteres ASCII e Bits necessárias durante o processo, destacando principalmente, a conversão do fluxo de bits para caracteres ASCII na camada Física Receptora.

Em suma, o simulador alcançou resultados satisfatórios e por meio dos resultados gerados a partir de sua execução se tornou uma ferramenta útil para entender e analisar o funcionamento simplificado de cada etapa presente em um processo de transmissão de dados utilizando apenas a camada física e de aplicação.

O resultado do projeto pode ser encontrado no Github por meio deste link. Instruções de uso se encontram no arquivo README.md .