Universidade de Brasília
Departamento de Ciência da Computação
Disciplina: Teleinformática e Redes 1

Simulador da Camada Física

Integrantes:

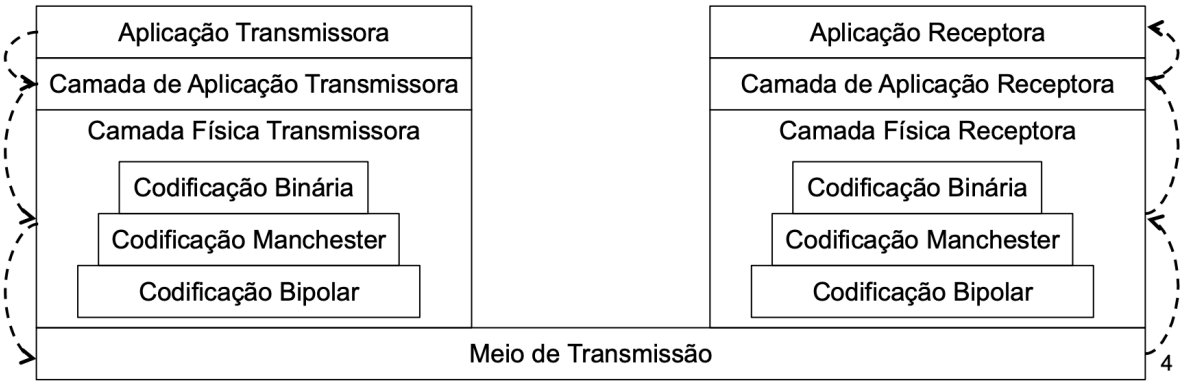
Bruno Couto Mariño - 19/0011106
Caio de Oliveira Mendes - 19/0011211
Igor Laranja Borges Taquary - 18/0122231
Pablo Arruda Araujo - 19/0049901

I. Introdução

No presente relatório, iremos construir um simulador capaz de representar o funcionamento da camada física num projeto de transmissão de dados. A camada física é responsável por transportar pelo meio físico da rede os bits constituintes da informação propagada.

Para isso, construímos nosso simulador com base no seguinte diagrama:

Figura 1 - Diagrama de funcionamento do simulador



Para cada quadrado do diagrama acima, iremos criar uma sub-rotina em C++ responsável por implementar o funcionamento específico de cada protocolo. Além disso, iremos utilizar a biblioteca NCurses para criar uma representação visual da mensagem codificada, seja ela na codificação binária, manchester ou bipolar.

II. Implementação

Como vimos na Figura 1, a criação do simulador consistiu na representação de diversas sub-rotinas interconectadas para simular a transferência de uma mensagem entre dois pontos por meio da camada física. A seguir, iremos explicar de forma detalhada o funcionamento de cada uma das subrotinas.

Figura 2 - Aplicação transmissora

```
/*****************************************************************************
Funcao responsavel por receber a mensagem do usuario e repassar para as
proximas camadas
*****************************************************************************/
void AplicacaoTransmissora(void) {
  string mensagem;

  cout << "Digite uma mensagem:" << endl;

  cin >> mensagem;

  CamadaDeAplicacaoTransmissora(mensagem);
}
```

A Aplicação transmissora é a primeira sub-rotina chamada no nosso simulador. Ela é responsável por receber uma mensagem informada pelo usuário e transmitir para a camada seguinte, conforme ilustra a figura a seguir:

Figura 3 - Camada de Aplicação Transmissora

```
/*****************************************************************************
Funcao responsavel por transformar os bytes em bits da mensagem original
*****************************************************************************/
void CamadaDeAplicacaoTransmissora(string mensagem) {
  vector<int> quadro;

  for (std::size_t i = 0; i < mensagem.size(); ++i) {
    bitset<8> x = mensagem.c_str()[i];

    for (int j = 7; j >= 0; j--)
      quadro.push_back(x[j]);
  }

  // chama a proxima camada
  CamadaFisicaTransmissora(quadro);
}
```

A Camada de Aplicação Transmissora é responsável por receber a mensagem da camada anterior e quebrá-la em bits. Cada caracter da mensagem é representado por um

byte. Com isso, pegamos cada um dos bits e separamos em oito bits, utilizando a estrutura bitset do C++. Como o bitset representa os bits invertidos, invertemos ele novamente e os adicionamos dentro de um vetor chamado quadro, e passamos para a camada seguinte.

Figura 4 - Camada Física Transmissora

```
/********************************************************************************
Funcao responsavel por codificar a mensagem original
********************************************************************************/
void CamadaFisicaTransmissora(vector<int> quadro) {
  vector<int> fluxoBrutoDeBits;

  switch (tipoDeCodificacao) {
    case BINARIA:
      fluxoBrutoDeBits = CamadaFisicaTransmissoraCodificacaoBinaria(quadro);
      break;
    case MANCHESTER:
      fluxoBrutoDeBits = CamadaFisicaTransmissoraCodificacaoManchester(quadro);
      break;
    case BIPOLAR:
      fluxoBrutoDeBits = CamadaFisicaTransmissoraCodificacaoBipolar(quadro);
      break;
  }

  MeioDeComunicacao(fluxoBrutoDeBits);
}
```

A Camada Física Transmissora recebe o quadro de bits da camada anterior e, dependendo do tipo de codificação informado pelo usuário, seja ela binária, manchester ou bipolar, a codificamos e enviamos para o meio de comunicação.

Figura 5 - Codificação Binária

```
/********************************************************************************
Funcao responsavel por utilizar a codificacao binaria
********************************************************************************/
vector<int> CamadaFisicaTransmissoraCodificacaoBinaria(vector<int> quadro) {
  vector<int> result;

  for (int i = 0; i < quadro.size(); i++) {
    result.push_back(quadro[i] == 0 ? 0 : 5);
  }

  return result;
}
```

Para a implementação da codificação binária, representamos cada bit 1 do quadro como sendo um sinal de amplitude de 5 V. Cada bit 0 do quadro será representado por um sinal de amplitude de 0 V.

Figura 6 - Codificação Manchester

```
/*****************************************************************************
Funcao responsavel por utilizar a codificacao manchester
*****************************************************************************/
vector<int> CamadaFisicaTransmissoraCodificacaoManchester(vector<int> quadro) {
  vector<int> result;

  for (int i = 0; i < quadro.size(); i++) {
    result.push_back(quadro[i] ^ 0); // XOR com amplitude baixa do clock
    result.push_back(quadro[i] ^ 1); // XOR com amplitude alta do clock
  }

  return result;
}
```

Para a implementação da codificação Manchester, simulamos um sinal de clock com a frequência duas vezes mais rápida que a do sinal original. Em termos de código, o clock é representado por um vetor consistindo de bits 0 e 1 intercalados, com tamanho sendo o dobro do tamanho do quadro. Assim, para cada bit do quadro, aplicamos a operação XOR para os bits 0 e 1 do clock, gerando, como resultado, um fluxo de bits com o dobro do tamanho original,

Figura 7 - Codificação Bipolar

```
/*****************************************************************************
Funcao responsavel por utilizar a codificacao bipolar
*****************************************************************************/
vector<int> CamadaFisicaTransmissoraCodificacaoBipolar(vector<int> quadro) {
  bool up = true;

  vector<int> result;

  for(int i = 0; i < quadro.size(); i++){
    result.push_back(quadro[i]==0 ? 0 : (up ? 1 : -1));

    if(quadro[i]) up = !up;
  }

  return result;
}
```

Para cada bit do quadro, o bit 0 é sempre representado como um sinal de amplitude nula. Os bits 1, no entanto, possuem uma representação intercalada entre um sinal de amplitude de 1 V e um sinal de amplitude de -1 V.

Figura 8 - Meio de Comunicação

```
/* Este metodo simula a transmissao da informacao no meio de
 * comunicacao, passando de um pontoA (transmissor) para um
 * ponto B (receptor)
 */
void MeioDeComunicacao(vector<int> fluxoBrutoDeBits) {
  iniciarGUI(); // Inicializar interface grafica

  vector<int> fluxoBrutoBitsPontoA, fluxoBrutoBitsPontoB;

  fluxoBrutoBitsPontoA = fluxoBrutoDeBits;

  for(int i = 0 ; i < fluxoBrutoBitsPontoA.size() ; i++) {
    fluxoBrutoBitsPontoB.push_back( fluxoBrutoBitsPontoA[i] ); // BITS! Sendo tr
  }

  transmitirInformacao(fluxoBrutoBitsPontoB); // Imprimir informacao codificada

  // chama proxima camada
  CamadaFisicaReceptora(fluxoBrutoBitsPontoB);
}
```

A figura acima representa a camada do meio de comunicação do nosso simulador. Essa camada é responsável por receber o fluxo de bits codificado das camadas anteriores e, os transmitir para o ponto B do fluxo de comunicação. É nessa camada que exibimos na interface gráfica o fluxo de informação sendo transmitida para o usuário.

Figura 9 - Camada Física Receptora

```
/*************************************************************************
Funcao responsavel por codificar a mensagem original
*************************************************************************/
void CamadaFisicaReceptora(vector<int> quadro) {
  vector<int> fluxoBrutoDeBits;

  switch (tipoDeCodificacao) {
    case BINARIA:
      fluxoBrutoDeBits = CamadaFisicaReceptoraDecodificacaoBinaria(quadro);
      break;
    case MANCHESTER:
      fluxoBrutoDeBits = CamadaFisicaReceptoraDecodificacaoManchester(quadro);
      break;
    case BIPOLAR:
      fluxoBrutoDeBits = CamadaFisicaReceptoraDecodificacaoBipolar(quadro);
      break;
  }

  // chama proxima camada
  CamadaDeAplicacaoReceptora(fluxoBrutoDeBits);
}
```

A Camada Física Receptora presente na figura acima é responsável por receber a informação codificada e realizar a decodificação do sinal antes de passar para a próxima camada.

Figura 10 - Decodificação Binária

```
/*************************************************************************
Funcao responsavel por utilizar a decodificacao binaria
*************************************************************************/
vector<int> CamadaFisicaReceptoraDecodificacaoBinaria(vector<int> quadro){
  // implementacao do algoritmo para DECODIFICAR
  vector<int> result;

  for (int i = 0; i < quadro.size(); i++) {
    result.push_back(quadro[i] == 0 ? 0 : 1);
  }

  return result;
}
```

Para realizar a decodificação binária, apenas observamos os bits codificados do quadro. Se eles forem nulos, o sinal decodificado é nulo. Caso contrário, o bit decodificado é 1.

Figura 11 - Decodificação Manchester

```
/*****************************************************************************
Funcao responsavel por utilizar a decodificacao manchester
*****************************************************************************/
vector<int> CamadaFisicaReceptoraDecodificacaoManchester(vector<int> quadro) {
  // implementacao do algoritmo para DECODIFICAR
  vector<int> result;

  for (int i = 0; i < (int) quadro.size(); i += 2) {
    if (quadro[i] == 0 && quadro[i+1] == 1) {
      result.push_back(0);
    } else {
      result.push_back(1);
    }
  }

  return result;
}
```

Para realizar a decodificação Manchester, é bem simples. Para isso, iteramos o sinal codificado de dois em dois elementos. O bit decodificado será 0 sempre que o primeiro elemento for nulo e o segundo for 1. Caso contrário, o bit decodificado será 1.

Figura 12 - Decodificação Bipolar

```
/*****************************************************************************
Funcao responsavel por utilizar a decodificacao bipolar
*****************************************************************************/
vector<int> CamadaFisicaReceptoraDecodificacaoBipolar(vector<int> quadro) {
  // implementacao do algoritmo para DECODIFICAR
  vector<int> result;
  for(int i = 0; i < quadro.size(); i++){
    result.push_back(quadro[i] == 0 ? 0 : 1);
  }

  return result;
}
```

Também é simples realizar a decodificação bipolar. Para isso, se o elemento do quadro for nulo, o bit decodificado será 0. Caso contrário, será 1.

Figura 13 - Camada de Aplicação Receptora

```
/*************************************************************************
Funcao responsavel por transformar os bits na mensagem original
*************************************************************************/
void CamadaDeAplicacaoReceptora(vector<int> quadro) {
  string mensagem;

  for(int i = 0; i < quadro.size(); i+=8){
    bitset<8> aux;
    for(int j = i; j < i + 8; j++){
      aux.set(7-(j-i), quadro[j]);
    }

    mensagem.push_back(char(aux.to_ulong()));
  }

  // chama proxima camada
  AplicacaoReceptora(mensagem);
}
```

A Camada de Aplicação Receptora representada pela figura acima é responsável por receber o sinal decodificado em bits e transformar os bits em bytes a fim de formar a mensagem final. Novamente, utilizamos a estrutura bitset do C++. Posteriormente, a mensagem final é transmitida para a próxima camada.

Figura 14 - Camada de Aplicação Receptora

```
/*************************************************************************
Funcao responsavel por exibir a mensagem decodificada ao usuario
*************************************************************************/
void AplicacaoReceptora(string mensagem) {
  endwin(); // Fecha interface grafica
  cout << "A mensagem recebida foi:\n" << mensagem << endl;
}
```

Na camada acima, encerramos a interface gráfica e exibimos a mensagem recebida final ao usuário.

Para compilar o código, é necessário compilar com:

```
g++ -std=c++17 Simulador.cpp CamadaFisica.cpp -lncurses
```

III. Membros

Todos os membros integrantes do presente trabalho participaram por igual de todas as atividades desenvolvidas, auxiliando no desenvolvimento do presente relatório, na elaboração das sub-rotinas em C++ e da pesquisa e implementação da interface gráfica presente no projeto.

IV. Conclusão

Com a realização do trabalho, podemos aprender de forma prática como se dá o funcionamento da camada física dentro de um projeto de redes, bem como o funcionamento mais detalhado da codificação binária, mancheste e bipolar.

Tivemos duas dificuldades principais durante o desenvolvimento do simulador: a primeira delas é em relação à implementação da interface gráfica do projeto. Há uma grande variedade de bibliotecas para isso, no entanto, muitas delas não apresentam uma forma de instalação muito clara e outras não apresentam uma documentação muito intuitiva. A segunda dificuldade encontrada foi a realização da abstração em código dos conceitos da camada física em si, tenho dificuldade em transformar em código a abstração das ondas transmitidas dentro da camada.