Universidade de Brasília
Departamento de Ciência da Computação

TELEINFORMÁTICA E REDES 1:
TRABALHO FINAL

Hércules Natan Dantas de Almeida Medeiros
Leonardo Tome Sampaio
Vinicíus Gonçalves Duarte

Brasília/DF
2025

1. Introdução

Neste trabalho, o objetivo é apresentar a implementação/simulação das camadas de enlace e física, com o lembrete dos seguintes tópicos: modulação digital, portadora modulada, protocolos de enquadramento de dados, detecção de erros, correção de erros. Usamos NRZ-polar, Manchester e Bipolar para a transmissão do sinal; usando CRC-32, código de hamming e bit de paridade. Para isso usamos NRZ-polar, manchester and bipolare, CRC-32, Hamming code e Parity bit par, para garantir a confiabilidade de transmissão de informações.
Implementamos utilizando a linguagem Python, significa que somente as ferramentas como o Linux, bibliotecas usadas como sendo Numpy, Matplotlib, Python GTK, socket e threading.Este trabalho tem como objetivo apresentar a implementação e simulação das camadas de enlace e física. Para tanto, relembramos os seguintes tópicos: modulação digital, portadora modulada, protocolos de enquadramento de dados, detecção de erros e correção de erros.

Utilizamos os métodos de transmissão de sinal NRZ-polar, Manchester e Bipolar, combinados com CRC-32, código de Hamming e bit de paridade, visando garantir a confiabilidade na transmissão de informações.

A implementação foi realizada em linguagem Python, utilizando ferramentas como Linux e bibliotecas como Numpy, Matplotlib, Python GTK, socket e threading.

2. Implementação

2.1 Camada Física

Modulação digital

Non-return to Zero Polar

Como bits “1” são tensões positivas e bits “0” são negativas, no protocolo NRZ-Polar o modulador não retorna à linha base entre os pulse. Isso torna a transmissão mais fácil, mas limita a sincronização porque uma sequência muito longa de “1” ou “0” não possui transições claras. Para resolver isso, foi implementada a função nrz_modulation(data: int) -> np ndarray.

Manchester

Como o protocolo, um bit é modulado pela transição no sinal: oscilação e divisa para “1” e divisa para oscilação e divisa “2”. Logo, a sincronização é mais fácil. Função implementada anteriormente era manchester_modulation(data: list[int]) -> np.ndarray.

Bipolar

O protocolo Bipolar, bits “1” são alternados entre tensões positivas e negativas enquanto os bits “0” são uma linha base. Decida-se para a função bipolar_modulation(data: int) -> np.ndarray.

2.1 Camada FísicaModulação Digital

Non-return to Zero Polar (NRZ-Polar)

No protocolo NRZ-Polar, bits "1" são representados por tensões positivas e bits "0" por tensões negativas. O modulador não retorna à linha de base entre os pulsos, o que facilita a transmissão, mas limita a sincronização em sequências longas de "1" ou "0" devido à ausência de transições claras. Para contornar essa limitação, foi implementada a função `nrz_modulation(data: int) -> np.ndarray`.

Manchester

No protocolo Manchester, a modulação de um bit ocorre através de uma transição no sinal: uma oscilação seguida de uma divisão para "1", e uma divisão seguida de uma oscilação para "2". Essa abordagem facilita a sincronização. A função implementada anteriormente para este protocolo era `manchester_modulation(data: list[int]) -> np.ndarray`.

Bipolar

O protocolo Bipolar representa bits "1" alternando entre tensões positivas e negativas, enquanto bits "0" são representados por uma linha de base. Para este protocolo, foi implementada a função `bipolar_modulation(data: int) -> np.ndarray`.

Modulação Portadora

Amplitude Shift Keying (ASK)

O protocolo ASK modula a amplitude da portadora para os bits: amplitudes altas para “1” e baixa amplitude ou zero para “0”. Aqui, a função implementada foi ask_modulation(data: int) -> np ndarray.

Frequency Shift Keying (FSK)

No domínio FSK, extremos de frequência que transmitem “1”, alta para “1” e mais baixa para “0”. Para isso, foi usado a função fsk_modulation(data: int) -> np.ndarray.

8-Quadrature Amplitude Modulation

O 8-QAM usam amplitude e fase afim de 8 diferentes estados, sendo isso possível devemos que a codificação mandatia 3 bits por símbolo. a fim de implementar essa função foi utilizada: QAM_modulation(data: int) -> list[int].

Modulação por Chaveamento de Amplitude (ASK)

O protocolo ASK realiza a modulação da amplitude da onda portadora. Amplitudes elevadas representam o bit "1", enquanto amplitudes baixas ou nulas indicam o bit "0". A função `ask_modulation(data: int) -> np.ndarray` foi implementada para este propósito.

Modulação por Chaveamento de Frequência (FSK)

No FSK, a informação é transmitida através da variação da frequência da portadora. Frequências mais altas correspondem ao bit "1" e frequências mais baixas ao bit "0". A função `fsk_modulation(data: int) -> np.ndarray` foi empregada para a implementação.

Modulação por Amplitude em Quadratura de 8 Níveis (8-QAM)

O 8-QAM utiliza combinações de amplitude e fase para gerar 8 estados distintos, permitindo a codificação de 3 bits por símbolo. A função `QAM_modulation(data: int) -> list[int]` foi utilizada para implementar essa modulação.

2.2 Camada de Enlace
Camada de Enlace
Contagem de Caracteres
No entanto, na contagem de caracteres, o primeiro byte de todos os quadros indica quantos caracteres contém em todo o quadro. Portanto, defini a seguinte função como: enquadramento_mensagem(mensagem, tamanho_quadro=8).
Inserção de Bytes
Caracteres especiais para indicar o início e fim do quadro no fluxo de dados. definitivamente def enquadramento_bytes(mensagem, in_icon=False, delimitador_final="ESC", tamanho_quadro=8).
Detecção de Erros
Bit de Paridade Par
Adicionar um bit extra ao final de cada quadro para garantir que o número total de "1" seja par. Destinada somente à função bit paridade(quadro).
CRC-32
Aplicar um cálculo polinomial para gerar um valor de validação é muito bom em erros. Como resultado, implementou-se def equadramento_crc(quadro).
Correção de Erros
Código de Hamming
Utiliza bits extra para corrigir erros nos dados enviados, detecta e corrige o erro na informação de quebra de um dígito.

Camada de Enlace

Contagem de Caracteres

O primeiro byte de cada quadro especifica o número total de caracteres contidos nele. Para isso, foi definida a função `enquadramento_mensagem(mensagem, tamanho_quadro=8)`.

Inserção de Bytes

Caracteres especiais são utilizados para marcar o início e o fim de um quadro no fluxo de dados. A função implementada para isso é `enquadramento_bytes(mensagem, in_icon=False, delimitador_final="ESC", tamanho_quadro=8)`.

Detecção de Erros

- **Bit de Paridade Par:** Um bit adicional é inserido ao final de cada quadro, garantindo que o número total de bits "1" seja par. Esta funcionalidade é exclusiva da função `bit_paridade(quadro)`.
- **CRC-32:** Um cálculo polinomial é aplicado para gerar um valor de validação, eficaz

na detecção de erros. Para isso, foi implementada a função `enquadramento_crc(quadro)`.

Correção de Erros

- **Código de Hamming:** Utiliza bits extras para corrigir erros nos dados transmitidos, detectando e corrigindo falhas de um único dígito na informação.

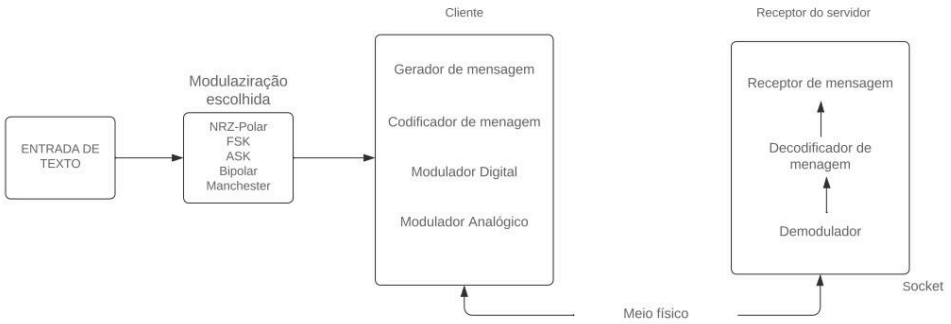

Figura 1: Fluxograma geral

O fluxograma ilustra o processo de codificação, transmissão, verificação e correção de erros.

Entrada dos Dados: Os bits de dados são fornecidos como entrada, representando a informação a ser transmitida.

1. **Cálculo e Inserção dos Bits de Paridade**:
   - Determina-se as posições dos bits de paridade.
   - Os bits de paridade são calculados para garantir que cada subconjunto de bits obedeça às condições de paridade.
2. **Formação do Quadro Codificado**:
   - O quadro é montado com os bits de dados e de paridade combinados.
3. **Transmissão do Quadro**:
   - O quadro é enviado ao receptor.
4. **Recepção e Verificação de Erros**:
   - Os bits de paridade são recalculados no receptor para verificar discrepâncias.
   - Caso os cálculos indiquem erros, a posição do erro é identificada com base nos bits de verificação.
5. **Correção ou Aceitação dos Dados**:
   - Se um erro for identificado, o bit correspondente é corrigido.
   - Caso não haja erros, o quadro é aceito como válido.
6. **Saída dos Dados**:
   - Após a correção (se necessária), os dados originais são recuperados e apresentados como saída.

O fluxograma apresenta o processo de codificação, transmissão, verificação e correção de erros.

Entrada de Dados: Os dados são fornecidos em formato de bits, representando a informação a ser transmitida.

1. **Cálculo e Inserção dos Bits de Paridade**:
    - As posições dos bits de paridade são determinadas.
    - Os bits de paridade são calculados para assegurar que cada subconjunto de bits esteja em conformidade com as condições de paridade.
2. **Formação do Quadro Codificado**:
    - O quadro é montado através da combinação dos bits de dados e de paridade.
3. **Transmissão do Quadro**:
    - O quadro é enviado ao receptor.
4. **Recepção e Verificação de Erros**:
    - Os bits de paridade são recalculados no receptor para identificar discrepâncias.
    - Se os cálculos indicarem erros, a posição do erro é identificada com base nos bits de verificação.
5. **Correção ou Aceitação dos Dados**:
    - Caso um erro seja identificado, o bit correspondente é corrigido.
    - Na ausência de erros, o quadro é aceito como válido.
6. **Saída dos Dados**:
    - Após a correção (se aplicável), os dados originais são recuperados e apresentados como saída.

Membros

- [Hércules] - Implementação da camada de enlace, 8-QAM, interface e integração entre as camadas
- [Leonardo Sampaio] - Implementação da camada física e relatório
- [Vinicius Gonçalves] - Implementação da camada física e camada de enlace

Conclusão

Este projeto proporcionou experiência prática na conexão entre o cliente e o servidor usando sockets e threads. Aprendemos como criar um servidor que suporta conexões simultâneas e como combinar essa funcionalidade com a representação visual das mensagens recebidas.

O maior desafio é manter o servidor funcionando enquanto gerencia múltiplas redes e configura corretamente a troca de mensagens. Apesar das dificuldades, conseguimos um bom resultado: conseguimos construir um servidor que recebia constantemente mensagens dos clientes.Este projeto proporcionou uma experiência prática significativa na conexão entre cliente e servidor, utilizando sockets e threads. Adquirimos conhecimento sobre a criação de um servidor capaz de gerenciar conexões simultâneas e integrar essa funcionalidade à representação visual das mensagens recebidas.

O principal desafio residiu em manter a operação contínua do servidor, enquanto se gerenciavam múltiplas redes e se configurava adequadamente a troca de mensagens. Não obstante as dificuldades, alcançamos um resultado satisfatório: o

desenvolvimento de um servidor apto a receber mensagens de clientes de forma ininterrupta.

Brasília/DF
2025