U.PORTO

FEUP FACULTY OF ENGINEERING
UNIVERSITY OF PORTO

3.LEIC09 - Redes de Computadores

Data Link Layer Protocol

Novembro 2024

Grupo

Trabalho Realizado Por:

- André David Aires de Freitas up202007189@fe.up.pt
- Jerson Narciso Amone up202109252@fe.up.pt

Resumo

Este projeto foi desenvolvido no âmbito da disciplina de Redes de Computadores e tem como objetivo a criação de um protocolo de ligação de dados para a transmissão de ficheiros, utilizando o ARQ Stop and Wait. O projeto proporcionou uma oportunidade valiosa para implementar e consolidar, na prática, o conhecimento teórico adquirido nas aulas, num contexto técnico. Adicionalmente, o desenvolvimento do protocolo sublinhou a importância do adequado isolamento das camadas. Esta abordagem não só simplificou a estrutura e a codificação do projeto, mas também aumentou a sua robustez, modularidade e reduziu a sua suscetibilidade a erros.

Conteúdo

Introdução

O objetivo deste trabalho de laboratório é implementar um protocolo de camada de enlace de dados que permita a transferência confiável de arquivos por cabo serial, utilizando um protocolo "Stop-and-Wait"para controle de fluxo e tratamento de erros. Este relatório descreve a arquitetura, estrutura de código e funcionalidade do protocolo, e valida sua eficiência através de testes. Cada secção explora aspetos específicos do design, incluindo enquadramento, controle de erros e a API fornecida para a camada de aplicação.

Arquitetura

A arquitetura do sistema é modular, apresentando uma clara distinção entre a camada de aplicação, camada de ligação, funções utilitárias e o gerenciamento da porta serial, o que permite uma fácil manutenção e substituição de componentes.

- **Camada de Aplicação:** Esta camada é responsável pela gestão da transmissão de ficheiros e utiliza a API da camada de ligação, com funções como `llwrite` e `llread`, para enviar e receber pacotes de forma estruturada e eficiente. Além disso, verifica a integridade do ficheiro transferido, garantindo que todos os dados foram corretamente enviados e recebidos.
- **Camada de Ligação:** Esta camada implementa funcionalidades fundamentais para a comunicação confiável entre dois sistemas, como enquadramento, controlo de erros e controlo de fluxo. As funções principais incluem `llopen` (iniciar conexão), `llwrite` (enviar quadros), `llread` (receber quadros) e `llclose` (encerrar conexão). A camada de ligação abstrai a complexidade de transmissão de dados e garante a confiabilidade na troca de informação.
- **Utilitários:** São funções de suporte que asseguram o enquadramento correto dos dados através da inserção de bytes e conversões específicas, mantendo a estrutura do pacote para evitar ambiguidades na interpretação dos dados.
- **Gerenciamento da Porta Serial:** Responsável pelas operações de baixo nível na conexão serial, esta camada efetua a leitura e escrita de bytes diretamente, assegurando uma comunicação eficiente e sem erros através da porta serial.

Esta organização modular permite a substituição e a atualização de componentes individuais sem comprometer o sistema como um todo, garantindo flexibilidade e facilidade de adaptação a diferentes necessidades de comunicação.

Estrutura do Código

3.0.1 Serial Port API

```c
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <termios.h>
#include <unistd.h>

// MISC
#define _POSIX_SOURCE 1 // POSIX compliant source

int fd = -1;              // Global file descriptor for open serial port
struct termios oldtio; // Serial port settings to restore on closing
```

Funções Principais

```c
// Abre e configura a porta serial.
// Retorna -1 em caso de erro.
int openSerialPort(const char *serialPort, int baudRate);

// Restaura as configurações originais da porta e fecha a porta serial.
// Retorna -1 em caso de erro.
int closeSerialPort();

// Aguarda por um byte recebido da porta serial e o lê (deve verificar
// se um byte foi realmente recebido com o valor de retorno).
// Retorna -1 em caso de erro, 0 se nenhum byte for recebido, 1 se um byte for recebido.
int readByte(char *byte);

// Escreve até numBytes na porta serial (deve verificar quantos foram
// realmente escritos pelo valor de retorno).
// Retorna -1 em caso de erro, caso contrário, o número de bytes escritos.
int writeBytes(const char *bytes, int numBytes);
```

3.0.2 Application Layer

Estruturas de Dados Principais

```c
// Estados do receptor durante a transmissão
enum ReceiveState {
    RECEIVE_STATE_START,
    RECEIVE_STATE_CONTINUE,
    RECEIVE_STATE_END
};

// Estrutura de metadados do ficheiro
typedef struct {
    size_t file_size;   // Tamanho do ficheiro
    char * file_name;   // Nome do ficheiro
    size_t bytesRead;   // Contagem de bytes lidos
} FileMetaData;
```

Funções Principais

```c
// Configura a camada de aplicação para transmissão ou recepção de ficheiro
void applicationLayer(const char *serialPort, const char *role, int baudRate,
                      int nTries, int timeout, const char *filename);

// Envia um pacote de dados com o número de bytes especificado
int sendPacketData(size_t nBytes, unsigned char *data);

// Envia um pacote de controlo com o nome do ficheiro e tamanho
int sendPacketControl(unsigned char C, const char * filename, size_t file_size);

// Lê e processa dados de um pacote, atualizando o tamanho lido
unsigned char * readPacketData(unsigned char *buff, size_t *newSize);

// Lê e processa um pacote de controlo
int readPacketControl(unsigned char * buff);

// Função do transmissor para enviar o ficheiro
void applicationLayerTransmitter(const char *filename);

// Função do receptor para receber o ficheiro
void applicationLayerReceiver(const char *filename);
```

3.0.3 Link Layer

Estruturas de Dados Principais

```c
typedef struct
{
    char serialPort[50];
    LinkLayerRole role;
    int baudRate;
    int nRetransmissions;
```

```
    int timeout;
} LinkLayer;
```

State Machine

```
enum CommState {
    STATE_START,            /**< Estado inicial, aguardando o flag de início. */
    STATE_FLAG_RECEIVED,    /**< Flag de início recebida, aguardando o byte de endereço. */
    STATE_ADDRESS_RECEIVED, /**< Byte de endereço recebido, aguardando o byte de controle. *
    STATE_CONTROL_RECEIVED, /**< Byte de controle recebido, aguardando o BCC. */
    STATE_BCC_OK,           /**< BCC correto, aguardando o flag de fim. */
    STATE_DATA,             /**< Bytes de dados sendo recebidos. */
    STATE_STOP              /**< Flag de fim recebida, pacote processado com sucesso. */
};
```

Funções Principais

```
// Abre a conexão de enlace com os parâmetros fornecidos.
// Retorna -1 em caso de erro.
int llopen(LinkLayer connectionParameters);

// Envia um buffer de dados através da camada de enlace.
// Retorna o número de bytes enviados ou -1 em caso de erro.
int llwrite(const unsigned char *buf, int bufSize);

// Lê um pacote da camada de enlace e preenche o buffer.
// Retorna o número de bytes lidos ou -1 em caso de erro.
int llread(unsigned char *packet);

// Fecha a conexão de enlace e exibe estatísticas, se necessário.
// Retorna -1 em caso de erro.
int llclose(int showStatistics);

// Estabelece a conexão serial com os parâmetros fornecidos.
// Retorna 0 em caso de sucesso, -1 em caso de erro.
int establishSerialConnection(LinkLayer connectionParametersApp);

// Termina a conexão serial.
// Retorna 0 em caso de sucesso, -1 em caso de erro.
int terminateSerialConnection();

// Manipulador de alarme para sinal de timeout.
void alarmHandler(int signal);

// Envia um pacote de comando com os parâmetros de endereço e controle.
// Retorna -1 em caso de erro, 0 em caso de sucesso.
int sendCommandPacket(unsigned char address, unsigned char control);

// Recebe um pacote com retransmissões, aguardando um pacote
// esperado com o endereço e controle especificados.
```

```
34 // Retorna -1 em caso de erro, número de bytes recebidos em caso de sucesso.
35 int receivePacketWithRetransmission(unsigned char expectedAddress,
36                                     unsigned char expectedControl,
37                                     unsigned char sendAddress,
38                                     unsigned char sendControl);
39
40 // Aplica o stuffing de bytes no buffer e ajusta o tamanho do novo buffer.
41 // Retorna o novo tamanho do buffer.
42 const unsigned char * byteStuffing(const unsigned char *buffer, int bufSize, int *newSize);
43
44 // Realiza o destuffing de bytes, processando e ajustando o tamanho do buffer.
45 // Retorna o número de bytes processados.
46 int byteDestuffing(unsigned char *buffer, int bufferSize,
47                                     int *processedSize, unsigned char *bcc2Received);
48
49 // Processa um único byte dentro da máquina de estados
50 // para recepção de pacotes. Esta função recebe o
51 // estado atual, o byte de entrada e os bytes de
52 // endereço e controle esperados. Ela avalia o byte
53 // de acordo com a lógica da máquina de estados e
54 // atualiza o estado atual conforme necessário.
55 enum CommState processByte(enum CommState currentState, unsigned char byte,
56                                    unsigned char expectedAddress, unsigned char expectedControl);
```

Principais Casos de Uso

4.0.1 Modo de Transmissor

No modo de **transmissor**, o objetivo é enviar um ficheiro para outra máquina. Para isso, o transmissor começa invocando a função `llopen`, que estabelece a comunicação e configura a porta série. Em seguida, o transmissor utiliza repetidamente a função `llwrite` para enviar pacotes de controlo (indicando o início e fim da transmissão) e pacotes de dados (contendo as partes do ficheiro). Os pacotes são criados com as funções `sendPacketControl` e `sendPacketData`.

4.0.2 Modo de Recetor

No modo de **recetor**, a máquina está configurada para receber e armazenar o ficheiro. A primeira ação do recetor também é invocar a função `llopen` para estabelecer a conexão. Depois, o recetor chama a função `llread` repetidamente até receber um pacote que indica o fim da transmissão. Durante esse processo, o recetor utiliza as funções `readPacketControl` e `readPacketData` para interpretar os pacotes e recuperar o conteúdo do ficheiro.

4.0.3 Finalização da Conexão

Após a transmissão ser concluída, ambas as máquinas (transmissor e recetor) devem encerrar a conexão corretamente. Para isso, ambas invocam a função `llclose`, que termina a comunicação e imprime as estatísticas relevantes da transmissão, como o número de pacotes enviados e a taxa de erro, fornecendo uma visão geral da performance da comunicação.

Protocolo de Ligação Lógico

O **Protocolo de Ligação Lógica** (LLP) é responsável pela comunicação entre duas máquinas através de uma série de funções específicas. Essas funções garantem o estabelecimento, envio, receção e encerramento de uma ligação, permitindo a troca de dados de forma fiável. As principais funcionalidades do protocolo incluem:

- Estabelecer a ligação com a função `llopen`
- Enviar uma trama através da função `llwrite`
- Receber uma trama através da função `llread`
- Fechar a ligação com a função `llclose`

A seguir, detalharemos o funcionamento de cada uma dessas funções.

5.0.1 Estabelecimento de Ligação: Função `llopen`

A função `llopen` tem a seguinte assinatura:

```
int llopen(LinkLayer connectionParameters);
```

Antes de estabelecer a ligação entre o transmissor e o recetor, ambos devem configurar a porta série e guardar uma cópia da estrutura `LinkLayer`, que será utilizada ao longo do programa. O processo de estabelecimento da ligação é iniciado pelo transmissor, que envia uma trama de supervisão do tipo `SET`. O recetor, ao receber a trama `SET`, responde com uma trama `UA` (Unnumbered Acknowledgement), indicando que está pronto para iniciar a comunicação. Quando o transmissor recebe a trama `UA`, a conexão é considerada estabelecida com sucesso. Caso o recetor não envie a trama `UA` dentro do tempo de espera (timeout) definido na estrutura `LinkLayer`, o transmissor tenta reenviar a trama `SET` um número pré-definido de vezes. Se após esses reenvios a trama `UA` não for recebida, a tentativa de ligação é considerada falhada.

5.0.2 Envio de Trama: Função `llwrite`

A função `llwrite` é utilizada para enviar dados ao recetor e tem a seguinte assinatura:

```
int llwrite(const unsigned char *buf, int bufSize);
```

Para transmitir os dados, o transmissor prepara a trama de forma a evitar a interpretação errada dos dados como uma flag. Isso é feito através de uma técnica chamada `stuffing`, que adiciona caracteres especiais aos dados e ao campo `BCC2` (campo de controlo), para que os valores hexadecimais idênticos ao valor da flag não sejam confundidos com o início ou fim da trama. Após o `stuffing`, a trama é montada com todos os campos necessários, incluindo os campos `BCC1` e `BCC2`, que garantem a integridade da transmissão. O transmissor envia a trama para o recetor e aguarda uma resposta. Caso a resposta seja do tipo `RR` (Receive Ready), a trama foi recebida com sucesso. Se a resposta for `REJ` (Reject), isso indica que houve um erro na trama, que será reenviada. Tal como na função `llopen`, o transmissor tentará reenviar a trama um número máximo de vezes caso não receba uma resposta dentro do intervalo de tempo especificado.

5.0.3 Leitura de Trama: Função `llread`

A função `llread` tem a seguinte assinatura:

```
int llread(unsigned char *packet);
```

No processo de receção de dados, o recetor utiliza uma máquina de estados para ler a trama. Se o cabeçalho da trama estiver corrompido, o recetor não enviará qualquer resposta e aguardará a retransmissão da trama. Se a trama for recebida corretamente, o recetor realiza a operação de `destuffing` nos campos de dados e no `BCC2`, removendo os caracteres especiais inseridos durante o `stuffing`. Em seguida, verifica a integridade dos dados. Se houver algum erro, o recetor envia uma trama `REJ` solicitando a retransmissão. Caso contrário, os dados são extraídos e enviados para a camada de aplicação, juntamente com o número de bytes lidos.

5.0.4 Fecho da Ligação: Função `llclose`

A função `llclose` tem a seguinte assinatura:

```
int llclose(int showStatistics);
```

O fecho da ligação segue um processo semelhante ao de estabelecimento. O transmissor envia uma trama de supervisão `DISC` (Disconnect), e o recetor responde com a mesma trama `DISC`, aguardando uma resposta do transmissor com uma trama `UA`. Após o recetor receber a trama `UA`, a ligação é considerada encerrada com sucesso. Tanto o transmissor como o recetor então reconfiguram a porta série, restabelecendo o seu estado original antes da execução do programa. Se o parâmetro `showStatistics` for verdadeiro, são apresentadas estatísticas relacionadas com o processo de comunicação.

Protocolo de Aplicação

No nível da aplicação, a interação direta com o ficheiro a ser transferido é essencial para o correto funcionamento do protocolo. Após a conexão ser estabelecida por ambos os lados, utilizando a função `llopen`, o recetor entra num ciclo contínuo de chamadas à função `llread`, até que um pacote sinalize o fim da transmissão do ficheiro. Cada vez que um pacote é lido, o recetor invoca a função `readPacketControl` ou `readPacketData` para processar o conteúdo do pacote recebido. O recetor então grava os dados extraídos dos pacotes num novo ficheiro local.

```
// Lê e processa um pacote de controlo
int readPacketControl(unsigned char * buff);
```

Por outro lado, o transmissor, após estabelecer a ligação com o recetor, inicia o processo de envio enviando primeiro um pacote de controlo que indica o início da transferência do ficheiro. Este pacote, codificado no formato **TLV** (Tipo, Comprimento, Valor), contém informações como o tamanho total do ficheiro em bytes e o nome do ficheiro a ser transferido. Este processo é realizado através da função `sendPacketControl`, que serve como uma abstração para o transmissor enviar as informações iniciais do ficheiro. Após o envio do pacote de controlo inicial, o transmissor começa a enviar o ficheiro em pacotes de dados de tamanho pré-determinado. Cada pacote é enviado utilizando a função `sendPacketControl` para cada pacote de dados. Quando todos os pacotes de dados forem transmitidos, o transmissor envia um último pacote de controlo para sinalizar o fim da transferência. Este último pacote contém os mesmos dados que o primeiro, indicando que a transmissão foi concluída.

```
// Envia um pacote de controlo com o nome do ficheiro e tamanho
int sendPacketControl(unsigned char C, const char * filename,
                      size_t file_size);
```

Estas funções, `sendPacketControl` e `readPacketData`, atuam como uma abstração, permitindo que o transmissor (TX) e o recetor (RX) interajam de forma eficiente e simplificada, sem a necessidade de manipulação direta dos dados de baixo nível. O uso dessas abstrações permite que o código seja mais modular e fácil de compreender, mantendo a complexidade reduzida.

Quando o recetor termina de receber os pacotes de dados, ele verifica se o número total de bytes lidos é igual ao tamanho do ficheiro que estava originalmente previsto para ser

recebido, confirmando assim que a transferência foi bem-sucedida. Após essa verificação, ambos os lados procedem com o encerramento da ligação, utilizando a função `llclose`, e o programa é concluído.

```
// Lê e processa dados de um pacote, atualizando o tamanho lido
unsigned char * readPacketData(unsigned char *buff, size_t *newSize);
// Função do transmissor para enviar o ficheiro
void applicationLayerTransmitter(const char *filename);
// Função do receptor para receber o ficheiro
void applicationLayerReceiver(const char *filename);
```

6.0.1 Funcionalidades adicionais

Adicionalmente, no nível da aplicação, é implementada uma barra de progresso no recetor para fornecer feedback visual sobre o estado da transferência do ficheiro. Esta barra é atualizada a cada pacote de dados recebido, permitindo ao recetor e ao utilizador monitorizar o progresso da transferência de forma contínua até a conclusão da transmissão. Esta funcionalidade oferece uma experiência de utilizador mais interativa e eficiente, especialmente em transmissões de ficheiros de maior dimensão.

```
Serial port connection established
Connection established
Bytes received:[18]
[INFO] Started receiving file: 'penguin.gif'
Bytes received:[1003]
[####                                              ] 9.12%, 0.87 KB/s, Bytes received:[1003]
[#########                                         ] 18.23%, 0.89 KB/s, Bytes received:[1003]
[#############                                     ] 27.35%, 0.90 KB/s, Bytes received:[1003]
[##################                                ] 36.47%, 0.91 KB/s, Bytes received:[1003]
[######################                            ] 45.59%, 0.91 KB/s, Bytes received:[1003]
[###########################                       ] 54.70%, 0.91 KB/s, Bytes received:[1003]
[###############################                   ] 63.82%, 0.91 KB/s, Bytes received:[1003]
[####################################              ] 72.94%, 0.91 KB/s, Bytes received:[1003]
[########################################          ] 82.06%, 0.91 KB/s, Bytes received:[1003]
[############################################      ] 91.17%, 0.91 KB/s, Bytes received:[971]
[################################################# ] 100.00%, 0.91 KB/s, Bytes received:[18]
[INFO] Finished receiving file: 'penguin.gif'
```

Figura 6.1: Exemplo da barra de progresso no recetor durante a transferência de ficheiro.

Validação

A validação do sistema foi realizada através de uma série de testes que visaram avaliar o seu desempenho em diversas condições operacionais. Os testes cobriram cenários típicos de utilização, bem como situações extremas que poderiam comprometer a integridade da transferência de dados. A seguir, são apresentados os procedimentos utilizados e os resultados obtidos.

7.0.1 Procedimento de Teste

Foram efectuados vários testes para verificar o comportamento do sistema em condições reais e adversas. As situações testadas incluíram:

- Transferência de ficheiros em condições de operação **normais**;
- Transferência de ficheiros com a presença de **ruído** no meio físico de transmissão;
- Transferência interrompida devido à desconexão do **cabo**, seguida de reconexão;
- Transferência de ficheiros após uma tentativa de transmissão anterior ter sido **cancelada**;
- Transferência com **taxas de transmissão** e **tempos de espera** (timeouts) ajustados, mas mantidos dentro dos valores adequados para cada situação;
- Transferência com indução artificial de uma **taxa de erro de trama (FER)** não nula;
- Transferência de ficheiros com **diferentes tamanhos**.

Cada teste foi realizado em condições controladas, variando os parâmetros de transmissão e simulando diferentes tipos de falhas na rede, de modo a assegurar uma avaliação completa do sistema.

7.0.2 Resultados

Os testes efectuados demonstraram que, em todos os cenários com os parâmetros devidamente configurados, o sistema conseguiu completar a transmissão do ficheiro com sucesso. No entanto, quando o protocolo **Stop & Wait** foi aplicado, observou-se uma diminuição na eficiência da transmissão em alguns casos específicos. Os cenários em que a transmissão foi mais impactada incluíram:

- **Ruído no meio físico de transmissão**: A presença de ruído causou falhas na integridade das tramas, o que resultou num aumento significativo no número de retransmissões, prolongando assim o tempo total de transferência.
- **Interrupção do cabo** e reconexão subsequente: A desconexão temporária do cabo causou a perda momentânea da ligação, exigindo retransmissões e aumentando o tempo de transmissão.
- **Taxa de erro de trama (FER) não nula**: A indução artificial de erros nas tramas também resultou em falhas nas verificações de integridade, forçando o sistema a reenviar as tramas afetadas.

Em todos estes casos, o protocolo **Stop & Wait** revelou-se menos eficiente devido à sua natureza, que exige a confirmação de cada trama antes de enviar a seguinte, o que aumenta substancialmente o tempo de transmissão quando são necessárias retransmissões.

Eficiência do Protocolo de Ligação de Dados

Neste capítulo, analisamos a eficiência do protocolo de ligação de dados sob diferentes condições e parâmetros. Para cada uma das variações testadas, mantivemos os outros fatores constantes, de forma a não influenciar os resultados. Os gráficos correspondentes a estes testes podem ser consultados no Anexo II.

8.0.1 Variação do Tempo de Propagação

Uma das variáveis testadas foi o tempo de propagação, o qual foi ajustado utilizando a função `usleep`, simulando assim o atraso na transmissão dos dados através do meio físico. Com o aumento do tempo de propagação, observou-se uma redução na eficiência do protocolo. Este comportamento é esperado teoricamente, pois a eficiência de comunicação está inversamente relacionada ao tempo de propagação. Em cenários com longos tempos de propagação, o tempo total necessário para a transmissão de dados aumenta, resultando em uma maior sobrecarga e, consequentemente, numa diminuição da eficiência do sistema.

8.0.2 Variação da Probabilidade de Erro no Cabeçalho da Trama

Para simular a ocorrência de erros no cabeçalho das tramas de informação, introduziu-se uma variável `FAKE_BCC1_ERR`, que define a probabilidade de ocorrer um erro no campo BCC1 (código de verificação do cabeçalho). A cada falha simulada, o receptor não envia qualquer resposta, como se o erro fosse real. Os testes mostraram que, tal como previsto teoricamente, a eficiência do protocolo diminui à medida que aumenta a probabilidade de erro no cabeçalho. Este tipo de erro tem um impacto mais significativo na eficiência em comparação com os erros no campo de dados, uma vez que resulta diretamente em time-outs no transmissor, dado que o receptor não envia qualquer resposta ao erro detetado.

8.0.3 Variação da Probabilidade de Erro no Campo de Dados

A probabilidade de erro no campo de dados das tramas foi ajustada utilizando uma variável `FAKE_BCC2_ERR`, que determina a probabilidade de ocorrência de erros no campo BCC2 (código de verificação dos dados). Neste caso, a simulação de erros no campo de dados resultou na não receção de uma resposta `REJ` pelo receptor, tratando o erro de forma semelhante ao erro real. A eficiência também diminui com o aumento da probabilidade de erro no campo de dados, como seria de esperar, uma vez que a detecção de erros requer retransmissões que sobrecarregam o sistema e afetam o desempenho.

8.0.4 Variação da Taxa de Transmissão (Baudrate)

A taxa de transmissão, ou baudrate, foi ajustada modificando o valor da taxa de baud no início da execução do programa. Nos testes realizados, verificou-se que, em geral, a eficiência diminui com o aumento do baudrate. Isto pode parecer contra-intuitivo, mas está relacionado ao fato de que, em taxas de transmissão mais elevadas, a largura de banda do meio físico é menos eficiente. Com baudrates mais elevados, o canal de comunicação é subutilizado, o que resulta em maior tempo de inatividade durante a transmissão, afetando a eficiência global do sistema.

8.0.5 Variação do Tamanho das Tramas

O tamanho das tramas foi modificado alterando o valor da variável `MAX_PAYLOAD_SIZE` no arquivo `link_layer.h`. Durante os testes com valores moderados para o tamanho das tramas, verificou-se que a eficiência se manteve estável. Isto ocorreu porque os testes foram realizados com uma taxa de erro nula. No entanto, em condições reais, com um maior tamanho de trama, aumenta a probabilidade de ocorrerem erros durante a transmissão, o que afetaria negativamente a eficiência. Ou seja, apesar de um aumento no tamanho das tramas não ter mostrado impacto significativo nos testes realizados, em situações reais o desempenho poderia ser comprometido devido à maior probabilidade de falhas nas transmissões.

8.0.6 Conclusão sobre a Eficiência do Protocolo

Através das diferentes variações realizadas, foi possível observar os principais fatores que afetam a eficiência do protocolo de ligação de dados. A redução da eficiência com o aumento do tempo de propagação, o impacto da probabilidade de erros e a variação da taxa de transmissão demonstraram como os parâmetros do sistema influenciam diretamente o desempenho. Em particular, a presença de erros, especialmente no cabeçalho, tem um efeito mais acentuado na eficiência, dado que provoca timeouts e retransmissões adicionais. Assim, é essencial otimizar tanto os parâmetros de transmissão quanto a resistência a erros para garantir o funcionamento eficiente do protocolo em condições variadas.

Conclusões

Este projeto permitiu alcançar todos os objetivos propostos, com a implementação bem-sucedida do protocolo de comunicação. O protocolo foi desenvolvido conforme as especificações estabelecidas e os testes realizados confirmaram o seu correto funcionamento.

Os resultados dos testes validaram o funcionamento do sistema, permitindo uma melhor compreensão dos conceitos de inserção de bytes, enquadramento e o funcionamento do protocolo **Stop & Wait**. Além disso, foi possível internalizar os mecanismos de detecção e tratamento de erros, como a retransmissão de tramas em caso de falha.

Em resumo, este projeto proporcionou uma experiência prática enriquecedora, consolidando os conhecimentos adquiridos nas aulas e ampliando a compreensão sobre os desafios e soluções em protocolos de comunicação.

Anexo I - Source Code

Anexo II - Analise de Eficiência(Gráficos)

Anexo III - Estatísticas TX & RX

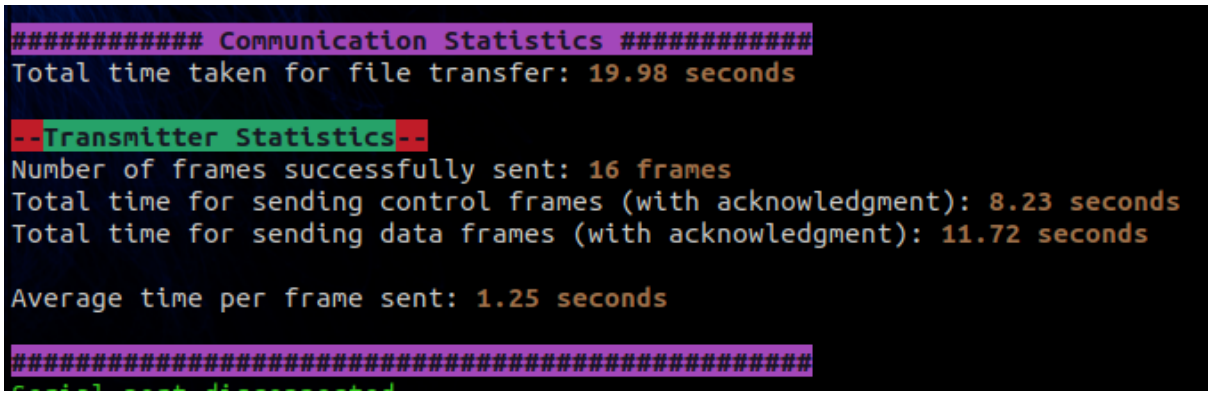

Figura 12.1: Exemplo de estatísticas no transmissor no final do enviou do ficheiro.

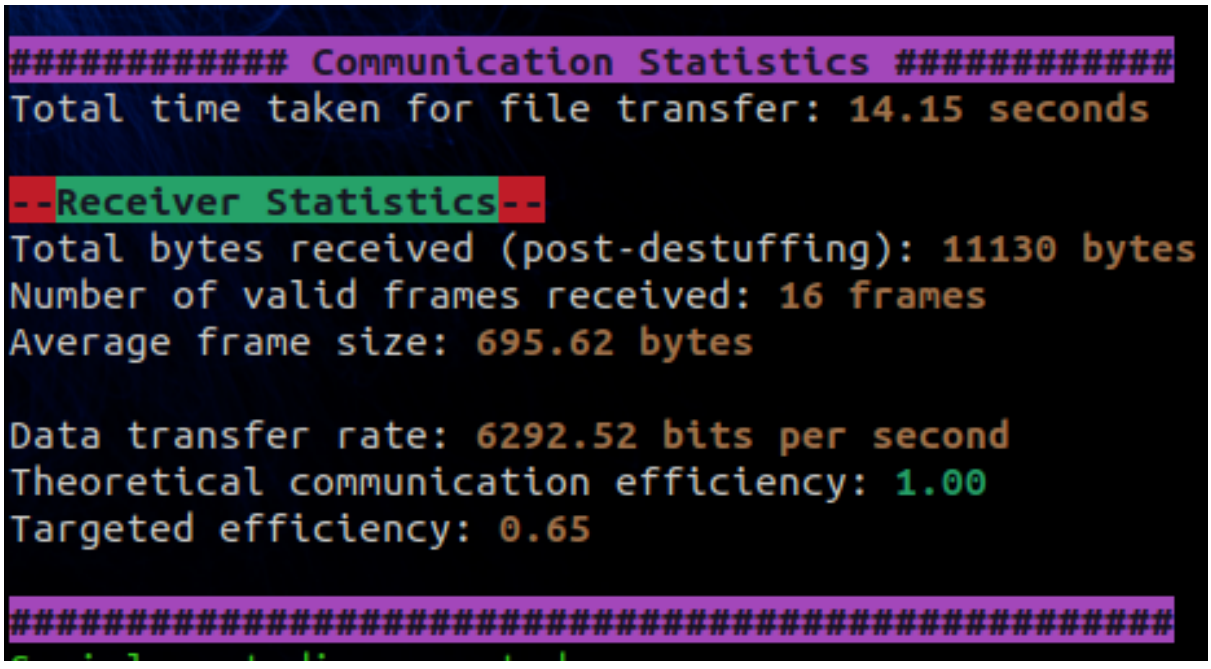

Figura 12.2: Exemplo de estatísticas no recetor no final da tranferiência do ficheiro.