U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

1º Trabalho Laboratorial

Protocolo de Ligação de Dados

Redes de Computadores 2021/2022

T2G06

Adriano Soares up201904873@fe.up.pt
Francisco Cerqueira up201905337@fe.up.pt

Índice

Sumário

Este projeto, realizado no âmbito da unidade curricular de Redes de Computadores do 3º ano da L.EIC, visa a elaboração de um protocolo de ligação de dados via porta série e de uma aplicação que integra esta camada.

Este relatório serve para clarificar e detalhar a implementação deste projeto e os conceitos teóricos aplicados no mesmo. Este trabalho prático foi concluído com sucesso e todos os objetivos definidos foram alcançados.

Introdução

Centrado em dois objetivos principais, este trabalho visou a implementação de um protocolo de ligação de dados, de acordo com o enunciado fornecido, e uma aplicação de transferência de ficheiros.

O protocolo utilizado fornece uma conexão estável e de confiança entre dois sistemas ligados pela porta série, assegurando uma correta sincronização, conexão, transferência de dados, terminação, enumeração do número de trama, códigos de confirmação e funcionamento adequado com erros (códigos de rejeição).

Este relatório encontra-se estruturado da seguinte forma:

- Arquitetura
- Estrutura do Código
- Casos de Uso Principais
- Validação
- Eficiência do Protocolo de Ligação de Dados
- Conclusão

Arquitetura

Camadas

Foram criadas duas camadas de protocolo distintas de modo a realizar-se a comunicação de dados.

Camada da Aplicação

A *Camada de Aplicação*, de alto nível, é responsável pela organização de pacotes a enviar e receber. Esta trata de enviar e de receber tanto pacotes de dados como pacotes de controlo iniciais e finais, com o auxílio das funções *llwrite()* e *llread()*. É também encarregue de abrir e fechar a ligação da porta série por meio de chamadas a funções da camada inferior, a *Camada de Ligação de Dados* (através das funções *llopen()* e *llclose()*).

Camada de Ligação de Dados

Esta camada, de baixo nível, é responsável pela gestão da comunicação com a porta série, assegurando que qualquer informação, sendo ela genérica, é enviada e recebida corretamente. É efetuado um controlo de erros (parâmetros BCC nas tramas) e reenvio de pacotes caso estes não estejam satisfatórios (envio de tramas REJ). É também encarregue de iniciar e terminar a conexão da porta série com o envio das tramas SET e DISC, respetivamente.

Execução do Programa

O programa é executado da seguinte forma:

application role port [filepath]

Tal que:

- *role*: 0 no caso do transmissor ou 1 no caso do recetor.
- *port*: número da porta série.
- *filepath*: caminho do ficheiro a transferir (é apenas utilizado no caso do transmissor).

Estrutura do Código

Camada da Aplicação

A aplicação partilha o mesmo executável para ambos os papéis. A partir do seu primeiro argumento, esta altera as suas funcionalidades: como transmissor divide o ficheiro de dados que pretendemos transmitir em pequenos pedaços (se necessário), adiciona informação relevante à cabeça destes, formando um pacote, e envia-os através da *Camada de Ligação de Dados*; como recetor recebe pacotes através da *Camada de Ligação de Dados* e analisa a cabeça destes, verificando se houve algum erro na transmissão, e forma o ficheiro resultante. Esta camada encontra-se dividida por três ficheiros distintos:

- application.c
- interface.c
- auxiliary.c

application.c

```
/* Divide o ficheiro em pequenas parcelas (de 512 bytes, se for necessário) e transmite-as após a
criação de pacotes com estas, utilizando pacotes de controlo para gerir a comunicação inicial e
final */
int transmitFile(int fd, char *filePath, int pathLength);

/* Recebe um ficheiro através do descritor de ficheiro da porta série fornecido, guardando-o na
mesma localização do executável com o nome original do ficheiro */
int receiveFile(int fd);
```

interface.c

```c
/* Enum que dita o estado da aplicação, transmissor ou recetor */
typedef enum { TRANSMITTER, RECEIVER } ApplicationStatus;

/* Estabelece a conexão da porta da porta série fornecida com o estado (transmissor ou recetor) dado
*/
int llopen(int port, ApplicationStatus status);

/* Envia um pacote através do descritor de ficheiro fornecido (da porta série) */
int llwrite(int fd, char *buffer, int length);

/* Recebe um pacote através do descritor de ficheiro fornecido (da porta série) */
int llread(int fd, char *buffer);

/* Termina a conexão da porta série */
int llclose(int fd);
```

auxiliary.c

```c
/* Extrai o nome do ficheiro através de um caminho */
int pathToFilename(char *filename, char *path, int pathLength);

/* Formata um pacote de controlo com os parâmetros fornecidos */
int createControlPacket(char *buffer, char control, int size, char *fileName, char fileNameSize);

/* Formata um pacote de dados com os parâmetros fornecidos */
int createDataPacket(char *buffer, char sequence, char *data, int dataLength);

/* Analisa o pacote de controlo, retornando o tamanho do ficheiro e o nome do ficheiro incluídos
neste através do segundo e terceiro argumento */
int parseControlPacket(char *packet, int packetLength, char *fileName);

/* Imprime na consola o pacote de controlo num formato de fácil compreensão */
void printControlPacket(char *packet);
```

Camada de Ligação de Dados

Esta camada encontra-se dispersa por três ficheiros distintos, dos quais apenas os dois primeiros são de interesse para a camada da aplicação:

- transmitter.c
- receiver.c
- protocol.c

protocol.c

```
/* Obtém a configuração da porta série */
int getConfig(int fd, struct termios *config);

/* Carrega a configuração da porta série */
int loadConfig(int fd, struct termios *config);

/* Retorna a configuração não canónica da porta série para o argumento */
void configNonCanonical(struct termios *config);

/* Aplica a configuração não canónica à porta série, armazena a configuração anterior, limpa a porta
série (através de tcflush) e retorna o descritor de ficheiro da porta série */
int openSerial(int port);

/* Aplica a configuração anterior da porta série e fecha o descritor de ficheiro da mesma */
int closeSerial(int fd);

/* Tenta transmitir uma trama para o dado descritor de ficheiro, realizando a operação de stuffing
antes do envio */
int transmitFrame(int fd, char *frame, int length);

/* Tenta ler uma trama através do descritor de ficheiro dado no tempo fornecido, retornando a
resposta no terceiro argumento */
int receiveFrame(int fd, int timer, char *frame);

/* Máquina de estados byte a byte, útil na confirmação do formato de tramas de informação e de
supervisão/não numeradas. Caso a trama seja de informação o frame irá ser alterado com a versão
destuffed do mesmo e o tamanho deste irá ser alterado também (pode ser reduzido na operação)
*/
FrameState FrameStateMachine(FrameState currentState, char *frame, int *length);

/* Aplica uma operação XOR a todos os elementos de um array */
char calculateBCC(char *data, int length);

/* Converte o array fornecido ao traduzir sequências de ESCAPE+0x5D and ESCAPE+0x5E para ESCAPE e
FLAG, respetivamente */
int destuffing(char *data, int length);

/* Converte o array fornecido ao traduzir qualquer byte ESCAPE e FLAG em ESCAPE+0x5D and
ESCAPE+0x5E, respetivamente */
int stuffing(char *data, int length);

/* Função auxiliar que forma tramas de supervisão/não numeradas */
void createSUFrame(char *frame, char control);

/* Função auxiliar que forma tramas de informação */
void createIFrame(char *frame, char control, char *data, int length);
```

receiver.h

```
/* Estabelece a conexão entre o transmissor e o recetor com o auxílio das funções receiveFrame e
transmitFrame */
int connectReceiver(int port);

/* Desconecta a conexão entre o transmissor e o recetor com o auxílio das funções receiveFrame e
transmitFrame */
int disconnectReceiver(int fd);

/* Recebe uma trama e retorna apenas a informação nela contida, um pacote, através do terceiro
argumento */
int receivePacket(int fd, int attempts, int timer, char *packet);
```

transmitter.c

```
/* Estabelece a conexão entre o transmissor e o recetor com o auxílio das funções receiveFrame e
communicateFrame */
int connectTransmitter(int port);

/* Desconecta a conexão entre o transmissor e o recetor com o auxílio das funções receiveFrame e
transmitFrame */
int disconnectTransmitter(int fd);

/* Tenta enviar uma trama o número de vezes fornecido no segundo argumento. Tenta novamente se nada
for recebido durante o tempo fornecido no terceiro argumento, ou a transmissão falhe. Retorna a
resposta através do quarto argumento */
int communicateFrame(int fd, int attempts, int timer, char *frame, int size);

/* Formata o pacote em tramas e envia-as através da função communicateFrame */
int transmitPacket(int fd, int attempts, int timer, char *packet, int length);
```

Casos de Usos Principais

O projeto tem dois casos de uso principais, o envio e a receção de um ficheiro. Passaremos agora a abordar a sequência de funções da camada superior, a *Camada da Aplicação*.

Transmissor

1. Inicia a aplicação (*main*).
2. A função *main*, após a verificação de argumentos, chama a função *llopen()*, que irá retornar o descritor da porta série (configura porta série, envia SET e espera ACK).
3. Após o retorno da função *llopen()* a *main* chama a função *transmitFile()* com o descritor da porta série e o PATH para o ficheiro a transmitir.

4. A função *transmitFile()* trata de abrir o ficheiro e envia o pacote inicial com as informações deste, nomeadamente o seu tamanho e nome. Divide o ficheiro em parcelas de modo a caber na trama de informação e, após a formatação e numeração correta do pacote, envia o pacote através da chamada à função *llwrite()* (tenta enviar 3 vezes a trama, 3 segundos de timeout, após a realização de stuffing). Finalmente, envia o pacote final e fecha o ficheiro, retornando o código 0, indicador que não houve erro.
5. Após o retorno da função *transmitFile()* a main chama a função *llclose()* que trata de fechar a ligação da porta série (envia DISC, espera por ACK, envia ACK e configura a porta série para a configuração inicial).

Recetor

1. Inicia a aplicação (*main*).
2. A função *main*, após a verificação de argumentos, chama a função *llopen()*, que irá retornar o descritor da porta série (configura porta série, espera SET e envia ACK).
3. Após o retorno da função *llopen()* a *main* chama a função *receiveFile()* com o descritor da porta série.
4. A função *receiveFile()* trata de receber o pacote inicial com as informações do ficheiro a ser transmitido e cria um ficheiro com o nome que recebeu. Recebe pacotes através da chamada à função *llread()* (caso a trama, após destuffing, não esteja corretamente formada envia REJ) e verifica a numeração e a quantidade de pacotes que recebe. Finalmente, recebe o pacote final, verifica a sua integridade, fecha o ficheiro e retorna o código 0, indicando que não ocorreram erros.
5. Após o retorno da função *receiveFile()* a main chama a função *llclose()* que trata de fechar a ligação da porta série (espera DISC, envia ACK, espera ACK e configura a porta série para a configuração inicial).

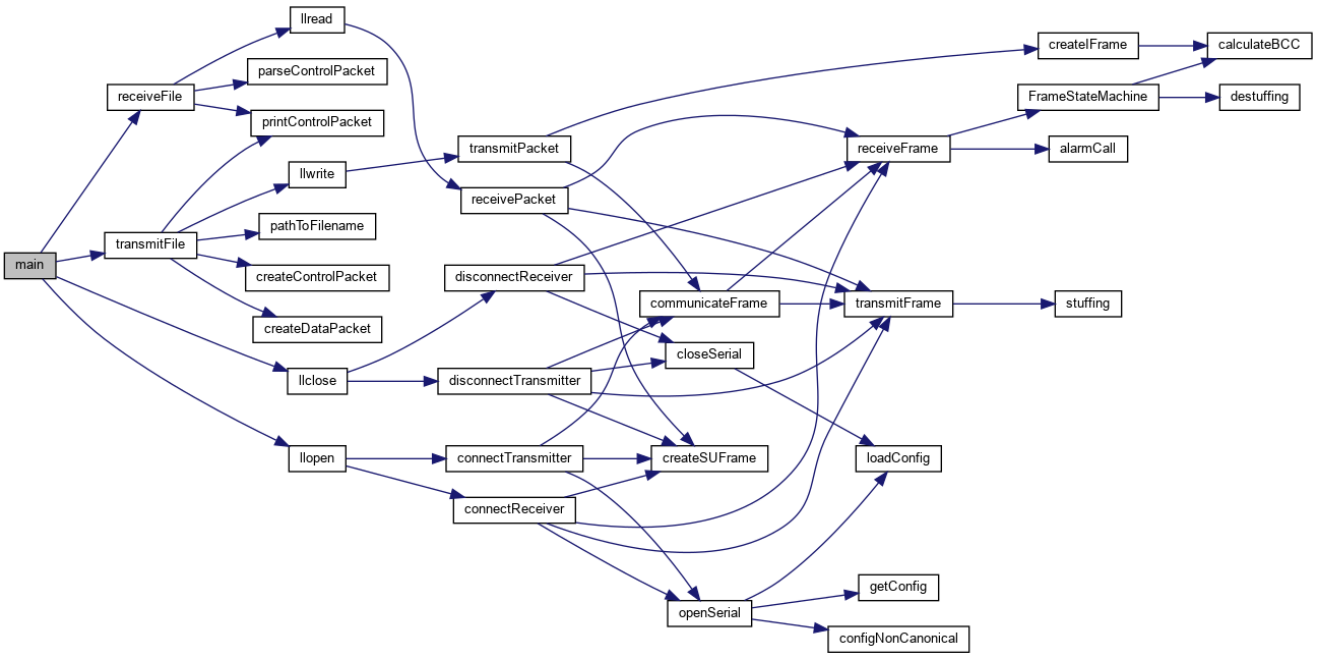


Figura 1: Gráfico de chamada de funções.

Validação

A validação do programa foi efetuada através da realização de diversos testes. Estes foram realizados com sucesso através da análise posterior do ficheiro resultante no lado do recetor. Passaremos agora a enumerá-los:

- Transmissão de ficheiro com diferentes tamanhos: pinguim.gif (11.0 KB), test.txt (30.9 KB), image_used_in_efficiency_tests.jpg (16.9 KB), testimage1.jpg (182.2 KB), testimage2.jpg (2.7 MB);
- Interrupção da transmissão de ficheiros através da desconexão física da porta série;
- Geração de ruído na transmissão de ficheiros de forma a ocorrerem falhas em bits na transmissão;
- Testes de eficiência (utilizando o ficheiro image_used_in_efficiency_tests.jpg de 130 Kb):
    - Transmissão do ficheiro com variação de Baudrate: B4800, B9600, B19200, B38400 e B115200;
    - Transmissão do ficheiro com variação de Frame Error Ratio: 0%, 5%, 10%, 20%, 25%, 30% e 35%;
    - Transmissão do ficheiro com variação do Tempo de Propagação: 0 ms, 50 ms, 100 ms, 150 ms, 200 ms, 200 ms, 250 ms e 500 ms;
    - Transmissão do ficheiro com variação do Comprimento da Trama de Informação: 128 bytes, 256 bytes, 512 bytes, 1024 bytes e 2048 bytes.

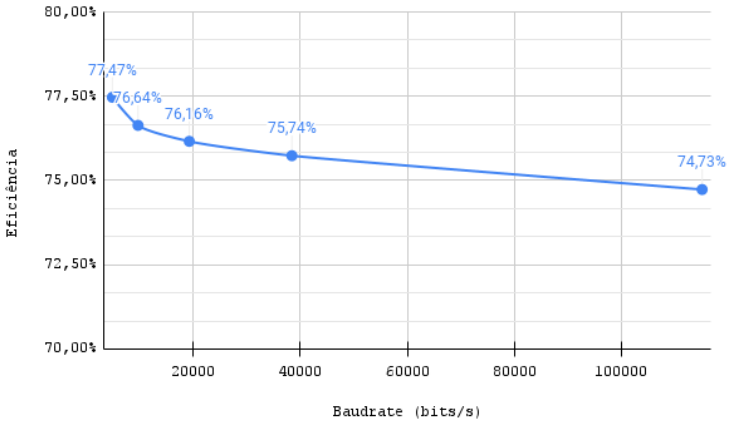
Eficiência do Protocolo de Ligação de Dados

De modo a avaliar a eficiência do protocolo de ligação de dados, foram executados vários testes de eficiência, como listados em seguida, alterando diferentes parâmetros e utilizando um ficheiro de imagem com 130Kb (16.9 KB).

As tabelas utilizadas para gerar os seguintes resultados encontram-se no fim do relatório, nos anexos.

Variação do Baudrate

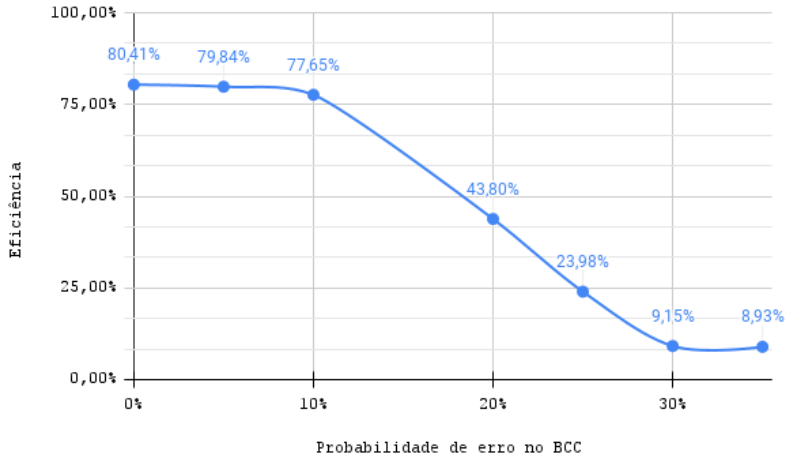
Representado pela letra C (capacidade de ligação), conseguimos concluir que, com o aumento da capacidade de ligação, a eficiência diminui ligeiramente, dado que, como visto nas aulas teóricas, a eficiência é o resultado da divisão do fluxo de informação pelo baudrate.



Variação do Frame Error Ratio

A variação do Frame Error Ratio, representado pela sigla FER, resultou numa diminuição significativa da eficiência. Um erro no BCC1 numa trama de informação ou controlo resulta numa penalização de 3 segundos (timeout), enquanto que um erro apenas no BCC2 resulta num pedido de retransmissão, assim que o erro é detetado, transparecendo uma penalização insignificante. Três timeouts consecutivos originam um término na conexão.

Os testes deste parâmetro seguem um modelo de probabilidade binomial, já que um erro é gerado em média a cada 1 / FER tramas.

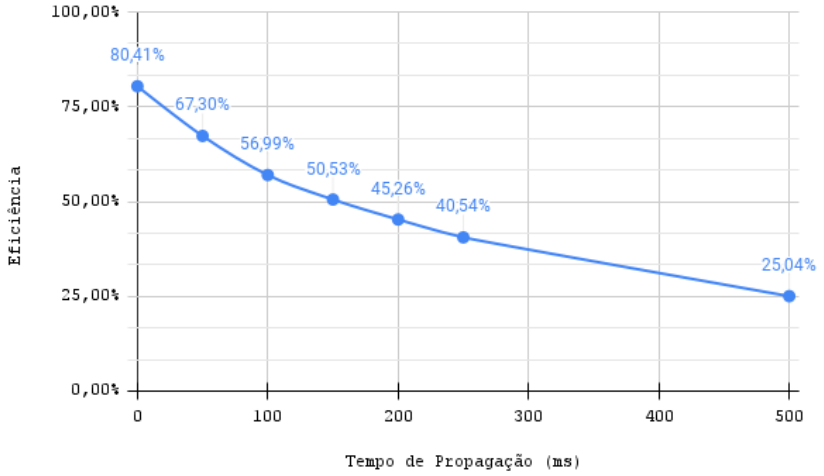


Variação da Distância de Conexão

Este parâmetro foi testado indiretamente, visto que é diretamente proporcional ao tempo de propagação, que pode ser facilmente simulado utilizando a função *usleep()*.

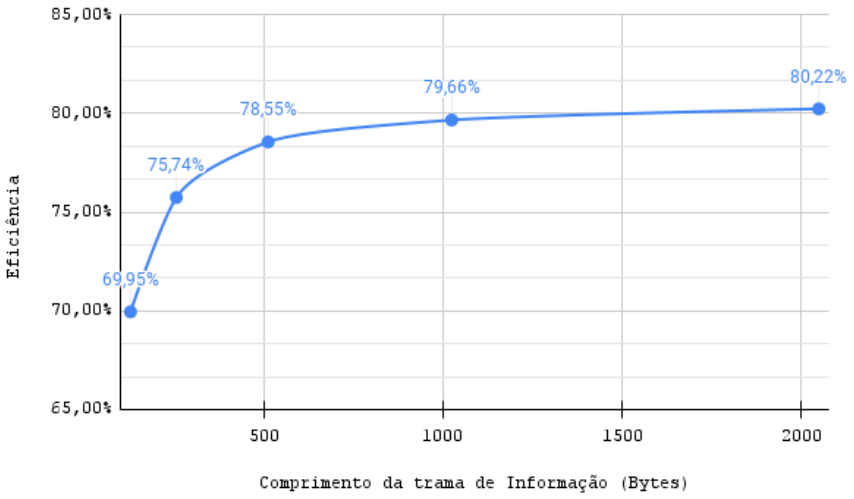
A variação do *t_prop* resultou numa diminuição significativa da eficiência, como seria esperado, visto que, tal como apresentado nas aulas teóricas:

$$\frac{1}{1+2\times\frac{t_{prop}}{t_{transferência}}}$$



Variação do comprimento da Trama de Informação

Os testes que envolveram este parâmetro mostraram-nos que, quanto maior for o pacote de dados, mais eficiente será o protocolo. Isto verifica-se pelo facto do envio de maior quantidade de informação reduzir o número de tramas a enviar.



Conclusão

Para concluir, neste projeto foi fulcral garantir não só a independência entre as camadas da Aplicação e de Ligação de Dados como também assegurar um método bem definido de comunicação entre estas. Deste modo, consideramos que este projeto contribuiu na solidificação dos conhecimentos teóricos das aulas e acreditamos que todos os objetivos propostos pelos professores na realização deste projeto foram alcançados.

Anexo I - Código

O código encontra-se na pasta */code* no .zip submetido.

Anexo II - Tabelas de Medição de Eficiência

Baudrate

Baudrate	t	Fluxo	Eficiência
C = bits / s	s	R = bits / s	S = R / C
4800	34,96	3718,535469	0,7746948894
9600	17,67	7357,102434	0,7663648368
19200	8,89	14623,1721	0,7616235471
38400	4,47	29082,77405	0,7573639075
115200	1,51	86092,71523	0,7473325975

Frame Error Ratio

Prob. BCC	t	Fluxo	Eficiência
%	s	R = bits / s	S = R / C
0%	4,21	30878,85986	0,8041369755
5%	4,24	30660,37736	0,798447327
10%	4,36	29816,51376	0,7764717125
20%	7,73	16817,59379	0,4379581716
25%	14,12	9206,798867	0,2397603872
30%	37,01	3512,564172	0,09147302531
35%	37,91	3429,17436	0,08930141563

Distância de Conexão

t_prop	t	Fluxo	Eficiência
ms	s	R = bits / s	S = R / C
0	4,21	30878,85986	0,8041369755
50	5,03	25844,93042	0,673045063
100	5,94	21885,52189	0,5699354658
150	6,7	19402,98507	0,5052860697
200	7,48	17379,67914	0,4525958111
250	8,35	15568,86228	0,4054391218
500	13,52	9615,384615	0,250400641

Comprimento da Trama de Informação

Comprimento da trama de Informação (bytes)	t	Fluxo	Eficiência
	s	R = bits / s	S = R / C
128	4,84	26859,50413	0,6994662534
256	4,47	29082,77405	0,7573639075
512	4,31	30162,41299	0,785479505
1024	4,25	30588,23529	0,7965686275
2048	4,22	30805,6872	0,8022314376