Mestrado Integrado de Engenharia Informática

Redes de Computador

Protocolo de Ligação de Dados

(1º Trabalho Laboratorial)

Gustavo Nunes Ribeiro de Magalhães up201705072@fe.up.pt

In Young Jang up201604450@fe.up.pt

Paulo Roberto Dias Mourato up201705616@fe.up.pt

Índice

Sumário

O presente relatório realizado no âmbito da unidade curricular de Redes de Computadores tem por objetivo a consolidação e explanação do primeiro trabalho laboratorial que consistiu na criação de um programa capaz de transferir dados através uma porta de série RS-232, respeitando protocolos de comunicação descritos no guião.
Em suma, foram atingidos os objetivos propostos tornado possível a transmissão de ficheiros sem perdas de informação, até mesmo com introdução de efemérides na transmissão.

Introdução

O trabalho teve por objetivo a implementação de um programa capaz de fazer transferência de dados entre duas máquinas conectadas por uma porta de série respeitando o protocolo de ligação descrito no guião que nos foi fornecido. Por outro lado, o relatório serve para toda a exposição teórica da implementação de código feita e devidos esclarecimentos.
Posto isto, o relatório divide-se em:

Arquitetura

Blocos funcionais

Estrutura do código

API’s, principais estruturas de dados, principais funções e sua relação com a arquitetura

Casos de uso principais

Identificação dos principais aspectos funcionais e descrição da estratégia de implementação.

Protocolo de ligação lógica

Principais estruturas de dados, principais funções e sua relação com a arquitetura do nível da ligação de dados

Protocolo de aplicação

Principais estruturas de dados, principais funções e sua relação com a arquitetura do nível da camada da aplicação.

Validação

Descrição dos testes efetuados.

Eficiência do protocolo de ligação de dados

Caracterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido.

Arquitetura

O código encontra-se dividido essencialmente em 2 blocos:

- bloco do nível da aplicação (application.c) responsável, pelas funcionalidades do nível da aplicação (como por exemplo a leitura de ficheiros e formação de tramas de informação, através das chamadas às funções do nível da ligação de dados) e também pela impressão e manipulação da UI;
- bloco do nível da ligação de dados (datalayer.c) responsável por todas as funcionalidades contidas a este nível sob a forma das funções requeridas no guião. Esta camada é, portanto, capaz de controlar timeouts, controlar erros, fazer stuffing a tramas...

De notar que se desenvolveu uma UI, invés de uso de argumentos na chamada do programa, com a finalidade de se testarem timeouts e demonstrar cada uma das funções objetivo do trabalho individualmente.

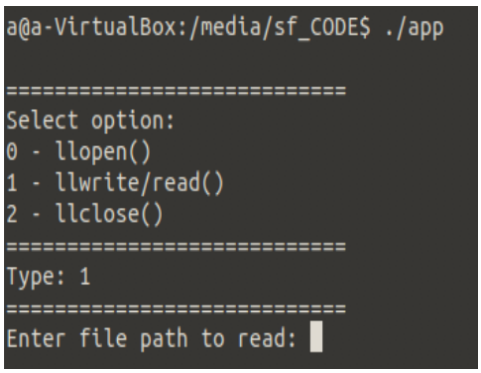


Figura 1- User interface

Estrutura do código

Utils.c - contém todas as macros do programa estruturas e funções utilitárias.

- Macros relevantes:
    - BAUDRATE - define valor da baudrate atual;
    - TIMEOUT - define timeout a usar no alarme;
    - ATTEMPTS - número de tentativas que se voltam a pedir tramas;
    - MAX_SIZE - tamanho de cada trama a de dados;
    - TRANSMITTER/RECEIVER - define o tipo de programa;
    - Macros restantes usadas para definir Bytes usados na criação das tramas.

- Estruturas:
    - *applicationLayer* - guarda descritor de ficheiro em que a porta foi aberta, e o tipo de abertura (transmissor/recetor);
    - *linkLayer* - guarda variáveis da que caracterizam a ligação, nome da porta, baurate, número de sequência, timeout, número de transmissões e tamanhos das tramas;
    - *supervision_instance_data_t - guarda e*stado atual da máquina de estados que vai processar as tramas de supervisão, para abertura e fecho de portas;
    - supervision_stat_t (enum) - contém todos os estados da máquina de estados que processa as tramas de supervisão.

- Funções:
    - setLinkLayer- cria a estrutura linklayer;
    - makeControlPacket - cria pacote de controlo;
    - sendDataPacket - cria e envia pacotes de dados;
    - set_reception, ua_reception, disc_reception - definem máquinas de estado de aceitação para os pacotes de supervisao;
    - BCC_make - gera BCC através da trama de dados;
    - read_control_field - ler a resposta do llread() e extrair a parte do campo de controlo;
    - setThingsFromStart - coloca na devidas variáveis a informação referente ao nome do ficheiro a receber, tanto como o seu nome. Esta informação é recebida através do pacote de controlo inicial;
    - endReached - verifica se acabou a transmissão do ficheiro, nomeadamente verificando se a mensagem recebida se trata do pacote de controlo final;

- headerRemoval - retorna cada mensagem, neste caso sem cabeçalho;
- checkBcc2 - verifica a integridade do BCC2;
- sendControlMessage - usada para enviar mensagens de controlo do tipo RR ou REJ para o TRANSMITTER;

Application.c - Contém o nível de aplicação assim como desenho da UI

- Guarda como variáveis globais

```
applicationLayer app;
extern linkLayer layerPressets;
```

- int interface () - responsável por desenhar interface e controlo de estado atual do programa. Através do tipo definido em linkLayer, decide os comportamentos que a camada de aplicação irá tomar, por exemplo criar pacotes de dados caso transmissor, ler pacote de dados com controlo de flags, caso recetor.

Datalayer.c- define bloco da ligação de dados do programa, onde apenas contém as funções específicas a esta camada impostas pelo guião.

- *Int llopen(int port, int type)* - Abre a porta de série com nome resultante da concatenação da string "/dev/ttyS com o número de porta recebido (port)". Ajustando o comportamento da função ao tipo de abertura que recebe como argumento (type)(transmissora/leitura);

- *Int llwrite(int fd,unsigned *buffer, int length)* – após realização de byte stuffing e adição de tramas de supervisão ao buffer recebido, com tamanho length, a função escreve o mesmo na porta com descritor de ficheiro fd.

- *Int llread(int fd, unsigned char *buffer)* – lê da porta com descritor de ficheiro fd, para o buffer a trama enviada pelo transmissor, encarrega-se de verificar fiabilidade de dados e fazer destuffing reescrevendo o buffer, por referência.

- *Int llclose(int fd)* – fecha a porta de série aberta pelo programa recorrendo ao descritor de ficheiro da porta (fd).

Alarm.h - define handler do alarme a usar nos timeouts, assim como função para desligar o alarme com finalidade de resetar a variável das tentativas de receções com sucesso.

Error.h – usada com a finalidade de definir macros de erros a usar;

Casos de uso principais

Os casos de uso do programa resumem-se à UI que tem por finalidade fazer chamada diretas às funções da camada de ligação de dados. Quando o utilizador corre o programa sem uso de argumentos é lhe presenteada uma interface onde deverá por esta ordem fazer execuções do llopen(), llwrite()/llread() e llclose(), selecionando os números que indexam as mesmas.

Na escolha de llopen() será perguntado o tipo de abertura (0-transmissão/1-recepção) devendo ser opostas em cada máquina, seguido do número da porta de série que será concatenado com a string “/dev/ttyS”. Posto isto executa a função llopen, guardado em cada um o tipo de abertura feito.

Aquando da seleção da opção seguinte llwrite()/llread() o programa encarrega-se de selecionar a função correta para o tipo de abertura feita, transmissor envia dados através de llwrite enquanto lê ficheiro, recetor recebe através de llread guardando-os e escrevendo um ficheiro no fim. No caso de escrita é perguntado o nome do ficheiro a passar, onde se usou primariamente “pinguim.gif”.

Por fim na seleção da última opção apenas se dá o fecho de portas.

Protocolo de ligação lógica

LLWRITE

*int llwrite(int fd, unsigned char * buffer, int length):*

Argumentos:

–fd: identificador da ligação de dados

–*Buffer*: pacote de dados formatados no nível de aplicação;

Retorno:

– Número de caracteres escritos, -1 em caso de erro.

O parâmetro buffer vai ser inserido no campo de informação, D1 até DN, das tramas de informação do nível de dados.

Figura 2 - tramas de informação

Cada vez que se envia informação, esta vai ter o formato indicado na imagem acima.

- “F” seria o flag, com byte 0x7E, que indica o início e o fim das tramas de informação.
- “A” é o campo de endereço, 0x03 se forem comandos enviados pelo emissor e respostas pelo recetor, que é o caso para esta função.
- “C” é o campo de controlo, alterando o valor, 0x00 (NS=0) e 0x40 (NS=1), dependendo da sequência em que a informação vai ser enviado. Por exemplo, na primeira trama de informação enviado terá 0x00, já na segunda enviará com 0x40 neste campo, a terceira voltará a enviar com 0x00 no campo de controlo e assim sucessivamente (considerando inexistência de erros), como ilustrado na imagem seguinte:

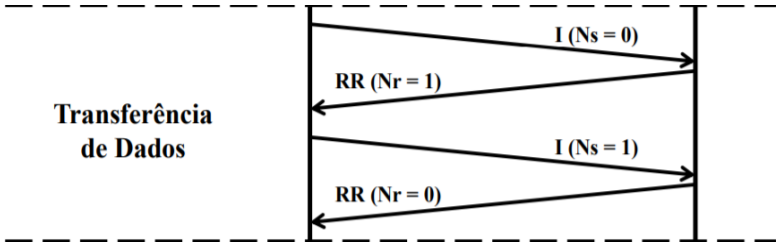


Figura 3 - transferência de dados

- Continuando com a trama da informação, o BCC1 (*block check character)* é o campo de proteção. Este campo é necessário para efetuar o parity check, um método para verificar a existência de erros quando lidos pela função llread(). O BCC1 é gerado para que o número de bits ‘1’ em “A”, “C” e BCC1, sejam pares. Por exemplo, se “A” em termos de bits for 101 e “C” for 010, existe um número ímpar de 1’s, então o BCC1 tem de introduzir ímpar de “1” s para que sejam pares. Para isto, é feito um XOR entre “A” e “C”. Com o exemplo referido acima, o BCC1 seria 111. O conjunto de “1” s no “A”, “C” e BCC1, nesse exemplo, seria 6, ou seja, seria par. Em suma, a função de XOR foi utilizado para gerir o BCC1.

- O BCC2 é gerado usando o mesmo método quem em BCC1, mas neste caso com XOR entre “D1” até “DN”.

- Posteriormente como as tramas de dados podem ter codificação igual à flag F, “0x7E”, substituímos esse valor por “0x7D” e adicionamos “0x5E”, caso sejam iguais ao ESC, “0xFD”, adicionamos um byte “0x5D” seguidamente. Este procedimento chama-se *byte stuffing,* tem por objetivo remover bytes iguais às flags que delimitam o pacote, para que na leitura os dados não sejam comprometidos.

No término deste processo envia as tramas de dados para o recetor, escrevendo os dados no descritor de ficheiro da porta.
Irá aguardar a leitura da resposta do recetor, extraindo o campo de controlo.
Dependendo do valor do campo de controlo, que podem ser RR/REJ (Nr = 0, 1) - valores principais - tomará o seguinte procedimento:

- Se REJ, repete envio incrementando tentativas.
- Se enviar um pacote de dados, I(NS = 0), e receber uma resposta pela llread(), o RR(NR = 0), isso quer dizer que o llread está a pedir mais uma vez um I(NS = 0). Se o llread() der um RR(NR = 1), este quer dizer que na próxima vez em que a função llwrite() é chamado, enviará um pacote de dados, I(NS = 1).
- Se enviar um pacote de dados, I(NS = 1), e receber uma resposta pela llread(), o RR(NR = 1), isso quer dizer que o llread está a pedir mais uma vez um I(NS = 1). Se o llread() der um RR(NR = 0), este quer dizer que llwrite() pode mandar um novo pacote de dados na próxima llwrite(), que será I(NS = 0).

Todo este procedimento é protegido através de tentativas, onde reenviará o pacote e caso de erro ou timeout dado por um alarme instalado. Caso excedam as tentativas os programas para.

LLREAD
int llread(int fd, unsigned char *buffer);

Argumentos:
–fd: identificador da ligação de dados
–Buffer: array de caracteres recebidos por referência
Retorno:
–Comprimento do array (número de caracteres lidos)

Função fundamental ao receiver e efetua a receção das tramas.
O formato das tramas é verificado através de uma máquina de estados.
Tramas com cabeçalho errado são imediatamente rejeitadas.
Para garantia da integridade dos dados verifica-se o BCC2 usando função checkBcc2() que gera um BCC como acima descrito através dos dados recebidos, comparando o resultado com o BCC recebido, caso esteja correto é enviado RR, caso contrário REJ através da função sendControlMessage().
Efetua-se também o destuffing de modo a obter-se a mensagem correta, e por último são tratados os duplicados, verificando-se a correspondência entre o número da trama e número esperado.

Protocolo de aplicação

Todo o código deste protocolo encontra-se em application.c, que eventualmente será evocado por seleção da opção 1 na UI e quando a porta se encontra aberta. Através de um switch case definido, tomará diferentes comportamentos de acordo com o tipo declarado:

- TRANSMITTER
    1. Pede um input do nome do ficheiro, a transmitir, guardando o nome;
    2. Abre ficheiro guardando tamanho deste em Bytes;
    3. Com auxílio da função makeControlPacket, irá formar um pacote de controlo de dados sob formato TLV, onde C codifica START, primeira trama TLV o tamanho do ficheiro e a segunda o nome do ficheiro. Posteriormente evoca llwrite com a mesma trama, passando-a à camada de ligação de dados.

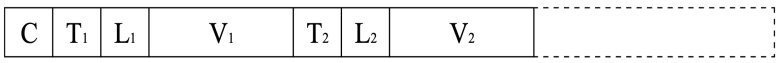


Figura 4 - Pacote de controlo

    4. Recorrendo a um ciclo o ficheiro será lido iterativamente com dimensões iguais às de MAX_SIZE, até que se chegue ao fim do ficheiro.
    5. Por cada iteração feita, será enviando um pacote à camada de ligação de dados através da função sendDataPacket, que recebendo os dados do ficheiro, irá acrescentar à cabeça desses dados flags, construindo o data packet. Estes pacotes terão esta estrutura:

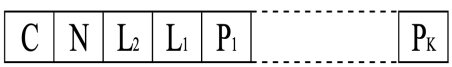


Figura 5 - Pacote de Dados

Onde C codifica o tipo de trama como sendo de dados, N o número de sequência da trama, L1 e L2 número de octetos no campo de dados e P1...Pk os dados a enviar divididos em bytes.

6. Por último, repete o passo 3 com a mudança de C para END, terminando com o fecho do ficheiro a escrever na porta.
   Caso a escrita retorne erro no llwrite, o programa para.

- RECEIVER
  1. Inicialmente o llread() é invocado para receber o pacote de controlo inicial;
  2. Através do pacote inicial e da função setThingsFromStart() descobrem-se o tamanho do ficheiro que vai ser transmitido, tanto como o seu nome;
  3. Segue-se então o loop infinito de receção de todas as tramas de informação. Onde se fazem sucessivas chamadas da função llread(), tentando receber sempre a proxima trama. Quando a trama é recebida verifica-se sempre se esta possui o byte de controlo END, para que em caso afirmativo se possa terminar o loop.

     Antes dos dados serem armazenados na estrutura que contém a informação do ficheiro a criar, o cabeçalho da trama de informação é retirado através da função headerRemoval().

  4. Finalmente é criado o ficheiro final com toda a informação recebida através deste protocolo de ligação de dados.

Validação

Para efeitos de validação do código desenvolvido foram realizados os seguintes testes:

1. Transmissões de ficheiros em vários formatos, sendo estes os formatos testados *.txt,*.js,*gif,*png,*jpg,*mp3;
2. Transmissões de ficheiros de diferentes tamanhos, tendo o maior 1,8MB;
3. Cortes de ligação, tanto como abertura do ficheiro como abertura e fecho de portas;
4. Geração de curto circuitos durante a transmissão;
5. Transmissões com variações nos tamanhos das tramas de informação e larguras de banda;
6. Envio de tramas com erros simulados;

Aos quais todos se realizaram com sucesso.

Eficiência do protocolo de ligação de dados

Todos os testes foram feitos através de transferências do ficheiro ‘pinguim.gif’ que possui um tamanho de 10968 Bytes o que constitui assim um tamanho de 87 744 bit considerado nos cálculos que foram feitos. (Tabelas usadas, presentes em anexo).

Variação do tamanho das tramas

Teste feito com variações da macro MAX_SIZE mantendo baudrate constante a 38400.

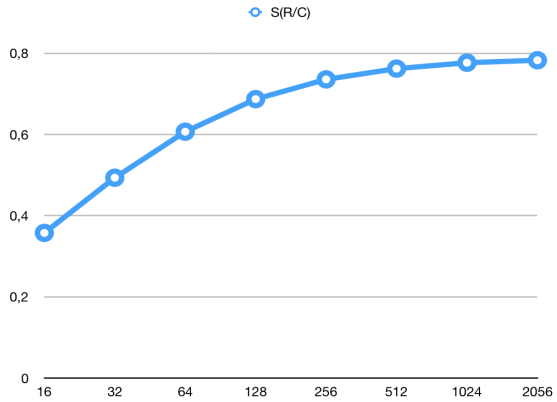


Figura 6 - grafico variação do tamanho de tramas

Como representado no gráfico após medições temporais foi possível concluir que que quanto mais baixo for o tamanho da trama mais tempo irá demorar a transmissão logo será menos eficiente, com o seu aumento de tamanho constata-se um aumento significativo de que estabiliza por volta dos 512B por trama.

Variação da capacidade de ligação

Teste feito com variações da macro BAUDRATE mantendo tamanho de pacotes a 256B constantes.

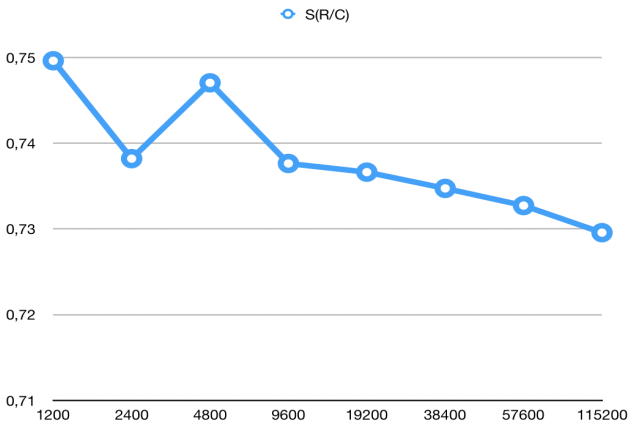


Figura 7 - Gráfico com variação da capacidade de ligação

Observa-se que com redução de capacidades de ligação a eficiência diminui, contudo na ordem das centésimas, dado os resultados de eficiência serem muito próximos uns dos outros quase se podendo dizer serem constantes.

Variação no atraso de transmissão

Teste feito com baudrate constante de 38400 e de pacotes a 256B, por adição de usleep(time) no código.

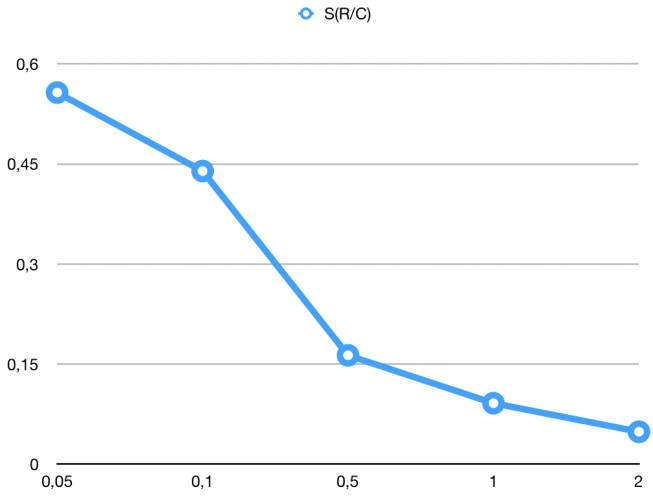


Figura 8 - Gráfico com variação no atraso de transmissão

Como será expectável por introdução de atrasos na transmissão a eficiência desde bastante aproximando-se de 0.

Variação do FER

Teste feito com baudrate constante de 38400 e de pacotes a 256B, com introdução de erros nos BCC das tramas com base numa probabilidade definida por macro variável.

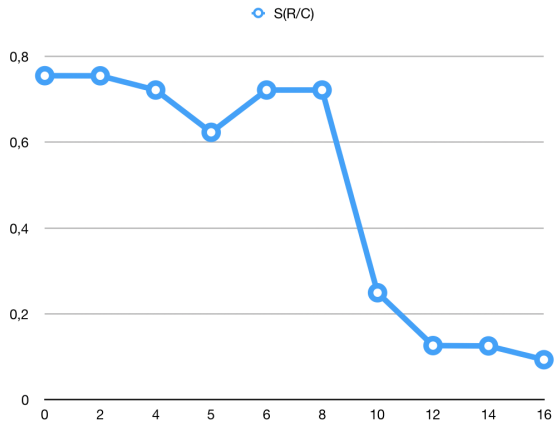


Figura 9 - Gráfico com variação de FER

Conclui-se que o programa apresentou ótimas eficiências até probabilidades de erro acima dos 8%, onde desde aí sofre perdas de eficiência muito significativas. Testes realizaram-se até 16% probabilidades de erro, não se aumentando por obtenção de eficiências próximas de 0, com aumentos de erros.

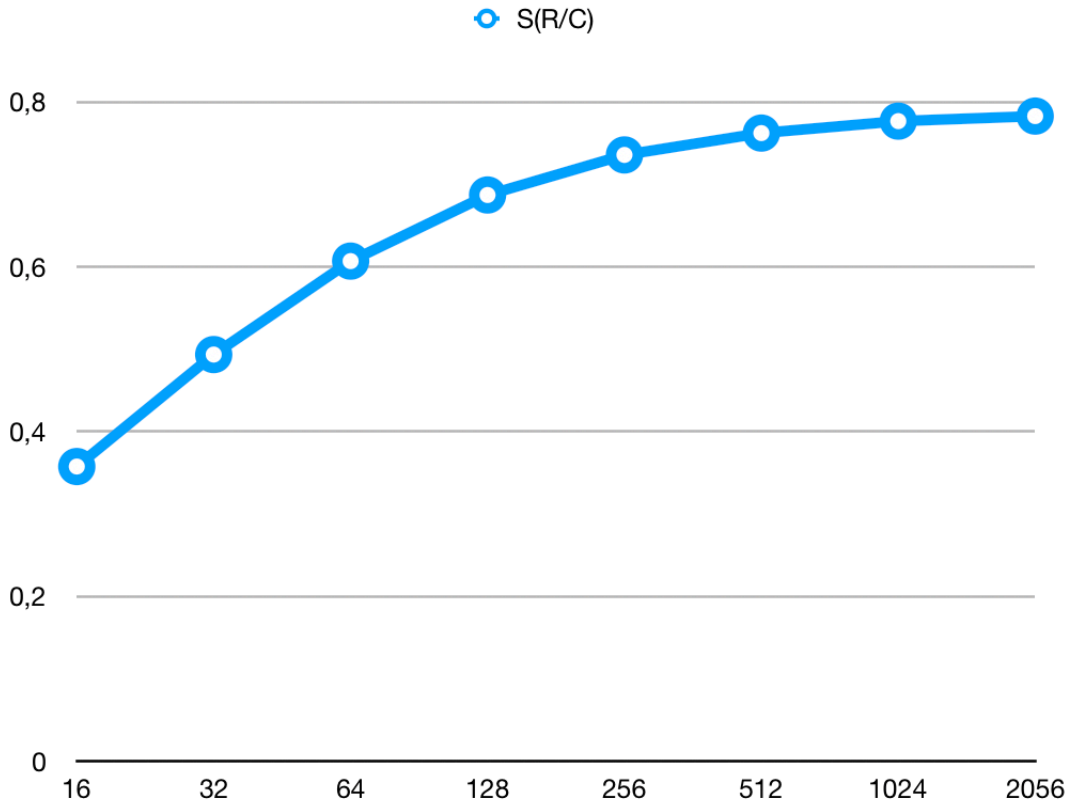
Conclusões

Neste projeto foi-nos possível estabelecer um protocolo de ligação de dados, que visa a partilha de informação entre dois sistemas ligados por uma porta de série.
A informação a transmitir foi então fragmentada e encapsulada em pacotes de dados, posteriormente transportados.
Baseamo-nos numa arquitetura em camadas, nomeadamente mantendo a independência destas. A informação dos cabeçalhos dos pacotes a transportar em tramas de Informação foi considerada inacessível ao protocolo de ligação de dados, e por outro lado a camada de aplicação não conhece os detalhes do protocolo de ligação de dados, apenas a forma como acede ao serviço.
De modo geral, o trabalho foi concluído com sucesso e com participação equilibrada de todos os membros do grupo. Será apenas de realçar que em termos teóricos foi fácil a interpretação do protocolo de transmissão a implementar, porém as dificuldades surgiram essencialmente no uso de strings de c e a sua necessidade de manipulação de memória.
Com isto, consideram-se alcançadas as metas definidas e adquirido o conhecimento pressuposto.

ANEXO 1 – Tabelas

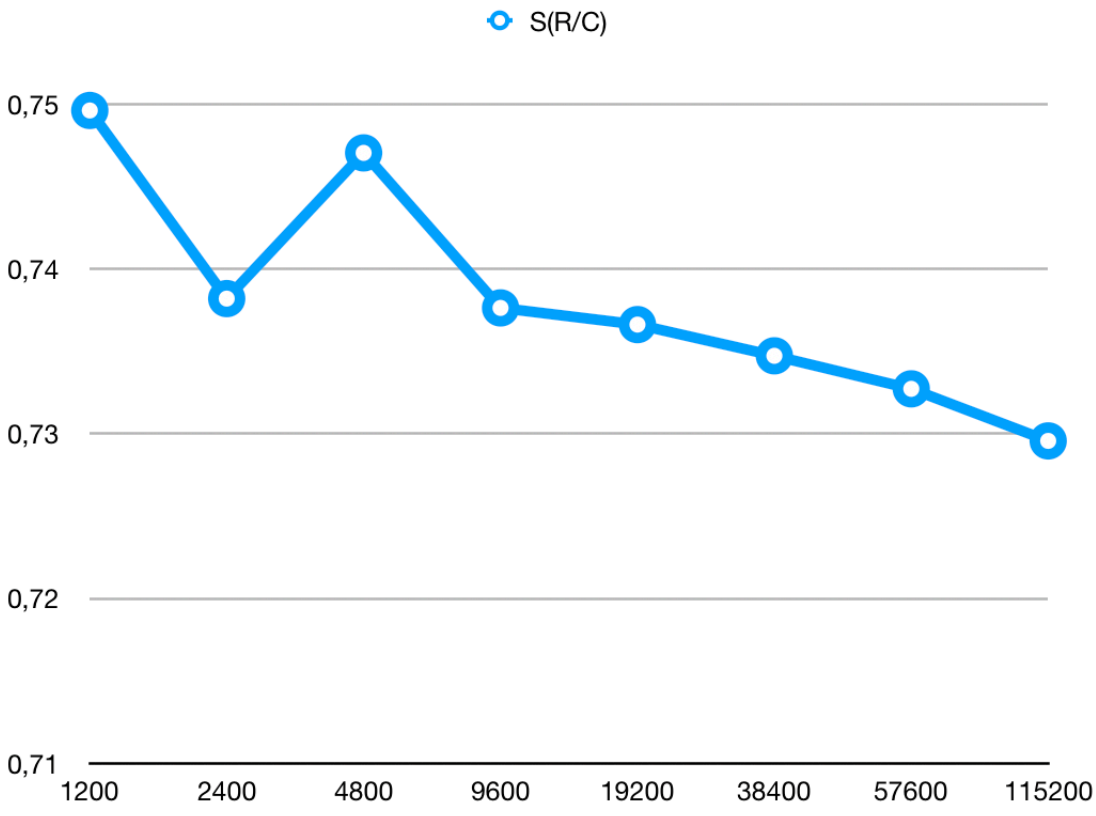
Variação de packet

Trama	tempo(s)	R(bit/tempo)	S(R/C)
16	6,394	13722,865186112	0,357366280888333
32	4,632	18943,0051813472	0,493307426597583
64	3,766	23298,9909718534	0,606744556558682
128	3,326	26381,2387251954	0,687011425135297
256	3,106	28249,8390212492	0,735672891178365
512	2,998	29267,5116744496	0,762174783188792
1024	2,942	29824,6091094494	0,776682528891911
2056	2,919	30059,6094552929	0,782802329564919



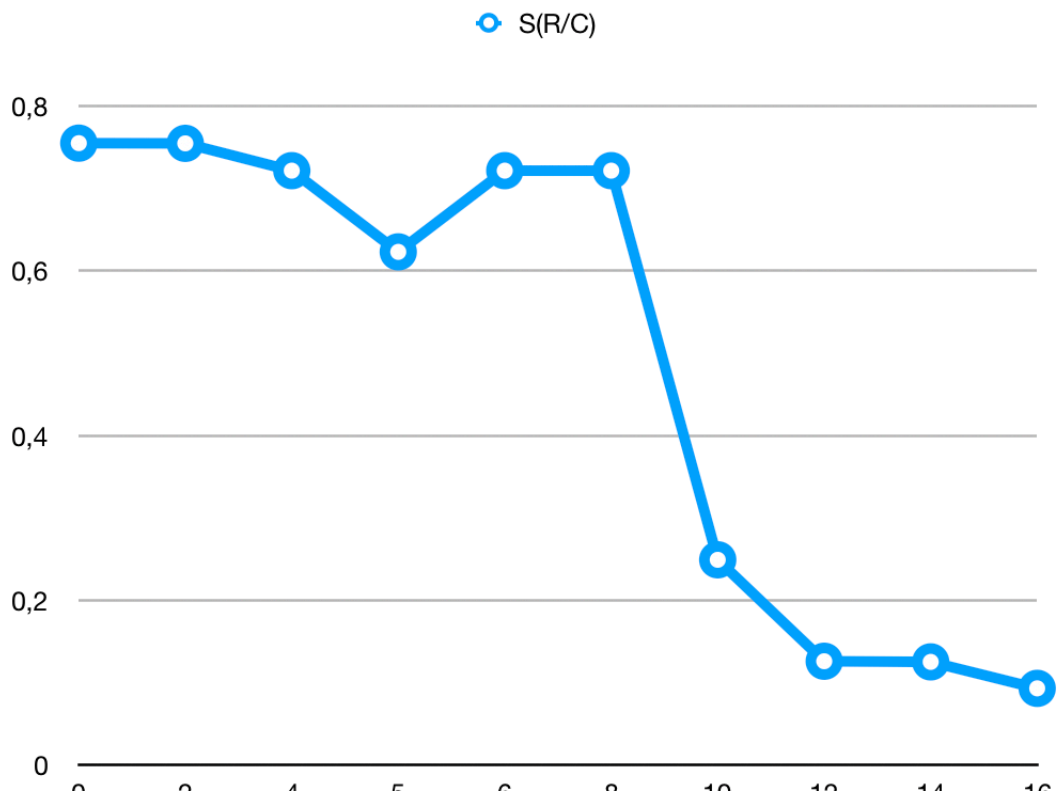
Variação da capacidade de ligação

Baud	Tempo (s)	R(bit/tempo)	S(R/C)
1200	97,544	899,532518658247	0,749610432215206
2400	49,526	1771,67548358438	0,738198118160158
4800	24,470	3585,779	0,747037188393952
9600	12,391	7081,26866273908	0,737632152368654
19200	6,204	14143,1334622824	0,736621534493875
38400	3,110	28213,505	0,734726688102893
57600	2,079	42204,9062049062	0,732724066057399
115200	1,044	84045,9770114943	0,729565772669221



Variação de FER

FER	Tempo(s)	R(bit/tempo)	S(R/C)
0	3,026	28996,6953073364	0,755122273628552
2	3,027	28987,1159563925	0,754872811364388
4	3,166	27714,4662034112	0,721730890713833
5	3,667	23928,0065448596	0,623125170439052
6	3,166	27714,4662034112	0,721730890713833
8	3,166	27714,4662034112	0,721730890713833
10	9,166	9572,76892864936	0,24929085751691
12	18,116	4843,45330094944	0,126131596378892
14	18,229	4813,42915135224	0,125349717483131
16	24,509	3580,07262638215	0,093231057978701



Variação de tempo propagação

Atraso(s)	Tempo(s)	R(bit/tempo)	S(R/C)
0,05	4,102	21390,5411994149	0,557045343734763
0,1	5,201	16870,6018073447	0,439338588732935
0,5	14,001	6266,98092993358	0,163202628383687
1	25,000	3509,760	0,091
2	47,001	1866,8538967256	0,048615986893895