Universidade do Porto
FEUP Faculdade de Engenharia

RCOM - Primeiro Trabalho Prático
Protocolo de ligação de dados
Turma 1 Grupo 5

Afonso Carvalho Pereira de Sá - up201604605
Nelson Alexandre Saraiva Gregório - up200900303
Nuno Duarte Ferreira Neves Mourinha Gonçalves - up201706864

Sumário

Este trabalho surge no contexto da disciplina de redes de computadores, como tal desenvolvemos um projeto que consiste na comunicação de portas série entre dois computadores. Esta comunicação é efetuada através da porta série e segue os requisitos dos protocolos lecionados nas aulas téoricas.

Um dos objetivos deste trabalho foi a comparação dos valores obtidos em relação aos valores téoricos. De tal modo testamos a eficiência do protocolo e obtivemos valores dentro do que era esperado em termos téoricos. Aproximando-nos de uma eficiência de 73%.

1 - Introdução

Neste trabalho os objetivos consistem em implementar um protocolo de ligação de dados, através da porta série, capaz de transferência de ficheiros. Neste relatório são descritas as diversas decisões de aquitetura e estrutura de código, são dados casos de uso, é descrita a lógica das diversas camadas do protocolo e são descritos os testes realizados e os valores experimentais obtidos.

De tal modo o programa tem as seguintes funcionalidades:

- Garante a transmissão de dados independente de códigos (transparência);
- Transmissão é organizada em tramas, que podem ser de informação, supervisão e não numeradas;
- Delimitação de tramas feita por meio de uma sequência de oito bits (flag) e a transparência é assegurada pela técnica de byte stuffing;
- Tramas protegidas por código detetor de erros;
- Uso da variante Stop and Wait.

2 - Arquitetura

Na implementação do trabalho decidimos dividir o código em três camadas. A camada tty_layer realiza as principais funções de ligação de dados e comunicação, por sua vez a camada link_layer trata das mensagens e do modo como a comunicação é sincronizada. Finalmente, a camada que chamamos de rcom (app_layer) organiza as tramas de informação, supervisão e não numeradas.

3 - Estrutura do código

Funções contidas na tty_layer:

- **open_port**: abre a porta série;
- **check_fd**: verifica se o estado do atributo fd (file descriptor) é válido;
- **set_port_attr**: muda os atributos da porta série, guardando os atributos anteriores;
- **restore_port_attr**: restaura os atributos da porta série para os valores previamente guardados;
- **close_port**: fecha a porta série;
- **write_msg**: escreve uma mensagem de acordo com o seu tamanho para o fd indicado;
- **read_msg**: lê uma mensagem até atingir o fim desta, delimitado por uma flag e o tamanho da mesma.

Funções contidas na link_layer:

- **BCC2_generator**: gera o campo de proteção independente de dados (bcc2) para uma trama de informação.
- As funções de **check_frame** verificam se o bcc/bcc2/address/control de uma trama são os esperados e retorna um erro apropriado se não for.
- **byte_stuffing**: função que realiza byte stuffing caso enconte no interior de uma trama o octeto *0x7e*, substituindo-o pela sequência *0x7d 0x5e*, e caso encontre o octeto *0x7d* este é também substituido por uma sequência, neste caso *0x7d 0x5d*.
- **byte_destuffing**: função que realiza o processo inverso da função anterior, ou seja, substitui sequências *0x7d 0x5e* por octetos *0x7e* e sequências *0x7d 0x5d* por octetos *0x7d*.
- **timeout_handler**: sempre que é detetado um erro de timeout a enviar ou receber dados esta função mostra que este foi detetado, limpa o erro e volta a tentar enviar/receber os dados.
- **return_on_timeout**: se o número de erros de timeout ocorridos for maior do que o número máximo de tentativas (MAX_RETRIES), o processo termina com um erro.
- **frame_set_request**: esta função envia sinais ao recetor até este se conectar ao emissor.
- **frame_set_reply**: envia-se uma mensagem de volta para o emissor para saber-se que este está conectado com o outro.
- **frame_i_reply**: recebe-se informação do buffer e imprime-se uma mensagem com a trama recebida ou com erro se um destes ocorrer.
- **frame_disc_reply**: envia-se uma mensagem para o emissor que se chegou ao final do ficheiro e espera-se que oemissor se desconecte para desconectar também o recetor.

- **llopen**: abre-se o protocolo de ligação de dados para estabelecer uma ligação entre os dipositivos.
- **llclose**: fecha-se o protocolo de ligação de dados.
- **llwrite**: envia-se os dados do ficheiro para o recetor.
- **llread**: recebe-se os dados do ficheiro enviados pelo emissor.

Funções contidas na camada de aplicação em rcom.c:

- **calculate_size_length**: calcula o número de bytes que é preciso alocar para o tamanho do ficheiro na trama.
- **build_control_packet**: constrói um pacote de controlo.
- **build_data_packet**: constrói um pacote de dados.
- **unpack_control**: obtém-se o nome do ficheiro a abrir e do seu tamanho de um pacote de controlo
- **main**: função principal do programa que tem como função transmitir ou receber os dados do ficheiro especificado usando o port especificado.

4 - Casos de uso principais

Dentro dos casos de uso temos a divisão entre modo de recetor e emissor, sendo que ambos partilham código e certos argumentos. O programa pode ser corrido em modo de recetor com o seguinte protótipo de chamada:

```
./rcom PORT R BAUD BCC_ER_PRCNT BCC2_ER_PRCNT PROP_TIME
PORT - Número identificador da porta série
R - Modo de reciever
BAUD - Valor da baud rate
BCC_ER_PRCNT - Percentagem de probabilidade de erros no bcc
BCC2_ER_PRCNT - Percentagem de probabilidade de erros no bcc2
PROP_TIME - Tempo de propagação (ms)
```

E o emissor (transmissor) segue o seguinte protótipo:

```
./rcom PORT T PATH_FILE PACKET_SIZE BAUD
PORT - Número identificador da porta série
T - Modo de transmitter
PATH_FILE - Path do ficheiro a ser enviado
PACKET_SIZE - Tamanho dos pacotes a enviar
BAUD - Valor da baud rate
```

O recetor ao ser executado vai vai usar as funções acima referidas para se abrir uma ligação (llopen) à porta indicada, recebe o pacote de controlo, abre o ficheiro a ser escrito e começa a ler os pacotes de dados que são enviados pelo emissor e de seguida escreve os bytes de dados no ficheiro.

O emissor inicialmente divide o ficheiro a ser enviada em pacotes que têm o tamanho, em bytes, definido na chamada, de seguida abre a ligação com a porta indicada na sua chamada e constrói e envia o pacote de controlo. Finalizados estes passos o emissor começa a enviar pacotes de dados.

Quando as funções acima descritas estão finalizadas, ambos os processos vão fechar o ficheiro e fazer flush da memória que foi usada previamente. Finalmente, ambos fecham a sua ligação à porta de série.

5 - Protocolo de ligação de lógica

No protocolo de ligação de lógica seguimos todos os parâmetros requeridos tendo efetuado o envio de tramas I, SET, DISC, RR (reciever ready), REJ (reject) e UA (user acknolgement), efetuando sistemas de stop-and-wait e protegendo as tramas.

O seguinte excerto de código demonstra algumas dessas funcionalidades implementadas:

```
void frame_set_reply()
{
    uint8_t err, bcc;
    if((err = check_frame_address(frame, A_SENDER)) != OK)
        DEBUG_PRINT(("Error SET | ADDRESS\n"));
    else if((err = check_frame_control(frame, C_SET)) != OK)
        DEBUG_PRINT(("Error SET | CTRL\n"));
    else if((err = check_frame_bcc(frame, &bcc)) != OK)
        DEBUG_PRINT(("Error UA | BCC CALC: %02x | BCC RCV: %02x\n", bcc, frame[FR
AME_POS_BCC]));

    if(err != OK)
        return;

    DEBUG_PRINT(("Got SET\n"));

    frame_SU[FRAME_POS_A] = A_SENDER;
    frame_SU[FRAME_POS_C] = C_UA;
    frame_SU[FRAME_POS_BCC] = frame_SU[FRAME_POS_A] ^ frame_SU[FRAME_POS_C];

    write_msg(llfd, frame_SU, FRAME_SU_LEN, &bytes_written);
    DEBUG_PRINT(("Sent UA\n"));

    printf("Incoming data...\n\n");
}
```

6 - Protocolo de aplicação

O protocolo de aplicação segue os parâmetros requeridos, realizando a transmissão/escrita da imagem, construindo pacotes de controlo e dados.

O seguinte excerto demonstra a construção dos pacotes de dados:

```
void build_data_packet(uint8_t** packet, size_t* packet_len, size_t sequence_num,
 uint8_t* data, size_t data_len)
{
    // Allocate space for control packet
    *packet = (uint8_t*)malloc(1 + 3 + data_len);

    // Build data packet
    (*packet_len) = 0;
    (*packet)[(*packet_len)++] = TLV_DATA;
    (*packet)[(*packet_len)++] = sequence_num % 255;
    (*packet)[(*packet_len)++] = data_len / 256;
    (*packet)[(*packet_len)++] = data_len % 256;

    // Add data bytes
    for (size_t i = 0; i < data_len; i++)
        (*packet)[(*packet_len)++] = data[i];
}
```

7 - Validação

Para validação do nosso protocolo foram realizados vários testes. Para tal implementamos funções que adicionam erros ao BCC, BCC2 e que mudam o tempo de propagação. É possível ativar esta funcionalidade na chamada do recetor através das seguintes variáveis, previamente apresentadas no ponto 4:

```
BCC_ER_PRCNT - Percentagem de probabilidade de erros no bcc
BCC2_ER_PRCNT - Percentagem de probabilidade de erros no bcc2
PROP_TIME - Tempo de propagação (ms)
```

Com estes valores fomos capazes de medir o número de vezes que há erros no bcc ou bcc2 e quantos erros são detetados e corrigidos e somos capazes de medir o tempo que um certo ficheiro demora a ser enviado. Esta informação foi automaticamente guardada em ficheiros de texto com o seguinte aspeto:

```
Total bytes: 10968 | packet size: 256 | Baud rate: 38400
BCC     : 0 | BCC2   : 0
Retries : 0 | Resend : 0 | Misses : 0
Time interval first - last data packets: 268819 us | 0.268819 s
```

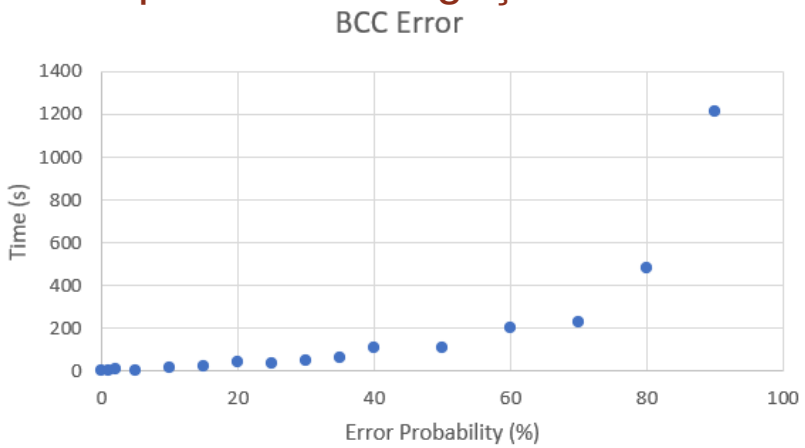
De tal modo, conseguimos obter várias medições com valores facilmente interpretáveis e que nos permitiram obter tabelas bem organizadas, tais com a seguinte:

Total Size (B)	Packet Size (B)	Baud Rate (b/s)	Propagation (µs)	Time (s)	Tf (s)	S
10968	256	38400	1000	3.144134	0.053333333	0.963855422
10968	256	38400	10000	3.539834	0.053333333	0.727272727
10968	256	38400	100000	7.501358	0.053333333	0.210526316
10968	256	38400	200000	11.901369	0.053333333	0.117647059
10968	256	38400	300000	16.301156	0.053333333	0.081632653
10968	256	38400	400000	20.701113	0.053333333	0.0625
10968	256	19200	100000	10.590281	0.106666667	0.347826087
10968	256	19200	200000	14.990098	0.106666667	0.210526316
10968	256	19200	500000	28.190228	0.106666667	0.096385542
10968	256	19200	999999	50.19022	0.106666667	0.050632959
10968	256	57600	100000	6.469721	0.035555556	0.150943396
10968	256	57600	500000	24.071022	0.035555556	0.034334764
10968	256	57600	999999	46.071395	0.035555556	0.017467266
10968	256	115200	100000	5.44106	0.017777778	0.081632653
10968	256	115200	500000	23.041562	0.017777778	0.017467249
10968	256	115200	999999	45.040089	0.017777778	0.008810581

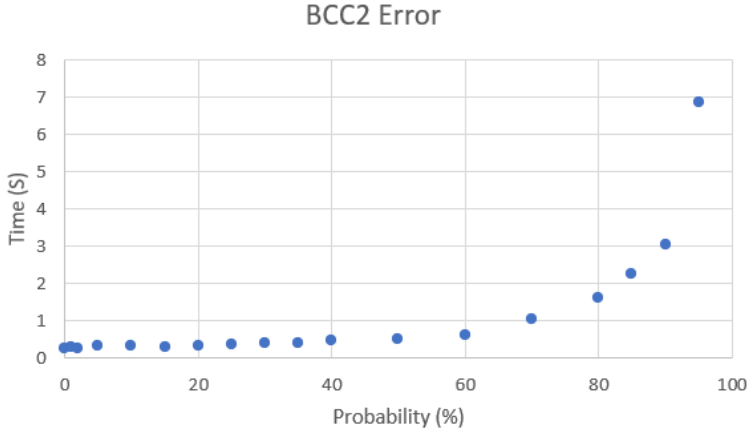
Todos os testes foram realizados com um valor máximo de 3000 de tentativas.

8 - Eficiência do protocolo de ligação de dados

Erros no BCC:



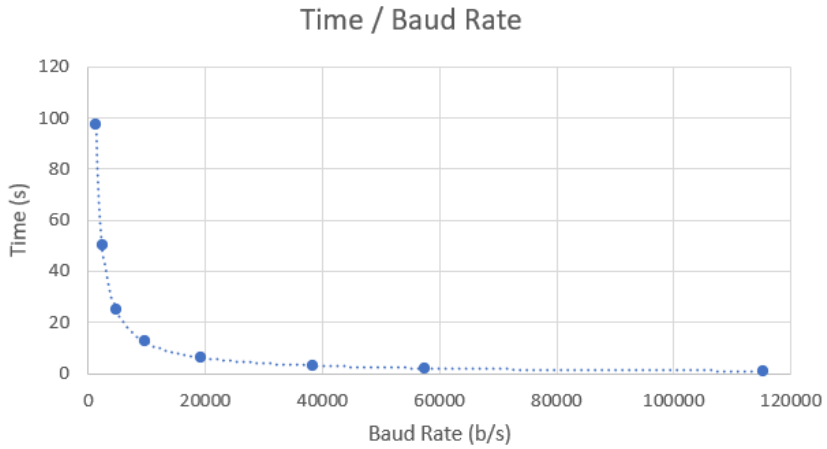
Erros no BCC2:



Tempo de propagação:

Total Size (B)	Packet Size (B)	Baud Rate (b/S)	Propagation (µS)	Time (S)	Tf (S)	S
10968	256	19200	100000	10.590281	0.106666667	0.347826087
10968	256	19200	200000	14.990098	0.106666667	0.210526316
10968	256	19200	500000	28.190228	0.106666667	0.096385542
10968	256	19200	999999	50.19022	0.106666667	0.050632959
10968	256	38400	1000	3.144134	0.053333333	0.963855422
10968	256	38400	10000	3.539834	0.053333333	0.727272727
10968	256	38400	100000	7.501358	0.053333333	0.210526316
10968	256	38400	200000	11.901369	0.053333333	0.117647059
10968	256	38400	300000	16.301156	0.053333333	0.081632653
10968	256	38400	400000	20.701113	0.053333333	0.0625
10968	256	57600	100000	6.469721	0.035555556	0.150943396
10968	256	57600	500000	24.071022	0.035555556	0.034334764
10968	256	57600	999999	46.071395	0.035555556	0.017467266
10968	256	115200	100000	5.44106	0.017777778	0.081632653
10968	256	115200	500000	23.041562	0.017777778	0.017467249
10968	256	115200	999999	45.040089	0.017777778	0.008810581

Baud Rate variável:



Total Size (B)	Packet Size (B)	Baud Rate (b/s)	Time (s)	R (b/s)	S = R / C (%)
10968	256	1200	97.462157	900.2878933	75.02399111
10968	256	2400	50.178409	1748.640536	72.86002233
10968	256	4800	25.090371	3497.118476	72.85663492
10968	256	9600	12.550926	6991.037952	72.823312
10968	256	19200	6.2815	13968.63806	72.75332325
10968	256	38400	3.146274	27888.22588	72.62558824
10968	256	57600	2.100483	41773.2493	72.52300225
10968	256	115200	1.056057	83086.42431	72.12363222

Com estes valores podemos observar os tempos médios de tentativa de resolver erros de BCC sendo este 2.90720159 s, BCC2 sendo o valor 0.007744 s. E a eficiência do protocolo, aproximadamente 73 %, que é um valor próximo dos valores teóricos expectáveis.

9 – Conclusão

A divisão do protocolo em tty_layer, link_layer e app_layer (rcom.c) delimita bem as diferentes camadas dos diferentes níveis e permite que o nosso código seja legível. De tal modo, fomos capazes de obter valores de eficiência próximos aos teóricos (73%).

Os maiores desafios na realização do projeto foram a lógica da construção da link_layer, nomeadamente a resolução de erros nas transmissões e toda a lógica associada a estes e a construção de delimitação e reparação de tramas.