U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Faculdade de Engenharia da Universidade do Porto

Redes de Computadores

3º ANO - **MIEIC**

Protocolo de Ligação de Dados

Trabalho Laboratorial 1

Estudantes & Autores

Diogo Silva, up201706892

up201706892@fe.up.pt

João Henrique Luz, up201703782

up201703782@fe.up.pt

04 de **novembro** de **2019**

Índice

Tabela de Figuras

Sumário

Este trabalho, desenvolvido no âmbito da unidade curricular de Redes de Computadores, tem como objetivo a elaboração de uma aplicação capaz de enviar ficheiros entre dois computadores conectados por uma **porta de série**.

Posto isto, este relatório tem por objetivo esclarecer as diferentes etapas procedimentais deste projeto. É ainda de salientar que todos os requisitos foram cumpridos e ainda foram implementadas outras funcionalidades para facilitar a utilização do programa, tendo resultado numa aplicação viável para o uso pretendido.

A. Introdução

Este trabalho foi desenvolvido com o objetivo de criar uma aplicação de transferência de ficheiros entre dois computadores através de uma **porta de série**, conseguida através da implementação de um **protocolo de ligação de dados** capaz de garantir a integridade dos dados transferidos, estando preparado para eventuais falhas de ligação e corrupção de dados.

O presente relatório procura examinar as diferentes componentes do projeto e a forma como se complementam para o bom funcionamento do mesmo. Assim, este tem a seguinte estrutura:

- **Arquitetura**
  Demonstração dos blocos funcionais e interfaces.
- **Estrutura do Código**

Análise das APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura.

- **Casos de Usos Principais**

  Identificação dos casos de utilização e dos consequentes mecanismos despoletados.

- **Protocolo de Ligação Lógica**

  Identificação dos principais aspetos funcionais e descrição da estratégia de implementação dos mesmos com apresentação de extratos de código.

- **Protocolo de Aplicação**

  Identificação dos principais aspetos funcionais e descrição da estratégia de implementação dos mesmos com apresentação de extratos de código.

- **Validação**

  Descrição dos testes efetuados com apresentação quantificada dos resultados, se possível.

- **Eficiência do Protocolo de Ligação de Dados**

  Caracterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido. Caracterização teórica de um protocolo Stop&Wait, que deverá ser usada como termo de comparação.

- **Conclusões**

  Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

B. Arquitetura

Neste projeto existem dois blocos funcionais, pensados com vista na independência entre camadas, sendo assim responsáveis por desempenhar tarefas específicas garantindo o encapsulamento e, por isso, não dependendo do funcionamento interno umas das outras. As camadas são as seguintes:

Protocolo de Ligação de Dados

Este grupo está encarregado de estabelecer e terminar a ligação. Além disto, garante a transmissão integral da informação pretendida, mesmo na eventualidade da ocorrência de erros, desconexão inesperada ou corrupção de algum dos dados.

Aplicação

Este módulo está responsável pela interface do utilizador, possibilitando a escolha entre a

receção ou o envio de um ficheiro (adotando, respetivamente, o papel de **receiver** ou **transmitter**) sendo, neste último caso, necessário indicar qual o ficheiro a enviar.

A aplicação faz uso das funções da camada do protocolo de ligação para, enquanto **transmitter**, enviar dados e informações relevantes do ficheiro ou, no caso do **receiver**, para os receber.

C. Estrutura do Código

Cada um dos blocos está dividido num ficheiro de **código fonte** (`.c`) e num respetivo **cabeçalho** (`.h`). Também é fornecido um `Makefile` de forma a facilitar a compilação do programa.

Assim sendo, existe o ficheiro `ll.c` onde se encontram as funções necessárias para o protocolo de ligação de dados (data link layer) e o `application.c`, responsável pelo correto funcionamento do programa no que diz respeito à interface do utilizador e ao protocolo da aplicação. No cabeçalho de cada um destes módulos, existem **macros**, **protótipos** e algumas **variáveis globais** ou **outras estruturas de dados** indispensáveis ao bom funcionamento do programa.

Protocolo de Ligação de Dados – ll.c

Funções Principais

- **`llopen`**, estabelece ligação entre transmissor e recetor.
- **`llwrite`**, transmite dados em pacotes.
- **`llread`**, recebe pacotes e preenche um *buffer* com dados recebidos.
- **`llclose`**, termina ligação entre transmissor e recetor.

Macros Importantes

- **`BUFFER_SIZE 256`**, tamanho de um *buffer*.
- **`RECEIVER 0`**, usada para identificar o papel do computador a correr a aplicação como recetor.
- **`TRANSMITTER 1`**, usada para identificar o papel do computador a correr a aplicação como transmissor.

Foram também definidas outras `macros` cujos valores são os indicados no guião.

Variáveis Globais e Outras Estruturas de Importância

- **`states_t state`**, enumeração usada para controlar as máquinas de estados que aceitam ou rejeitam pacotes enviados e recebidos.

- `int alarm_flag / conn_attempts`, variáveis necessárias à implementação da funcionalidade `TIMEOUT`.
- `int last_C_received`, variável que guarda o valor do número de sequência da última trama `I` recebida de forma a evitar aceitar pacotes repetidos.

Protocolo da Aplicação – application.c

Funções Principais

- **`start_writting`**, delega quais os pacotes a ser enviados ao longo da transmissão.
- **`start_reading`**, delega quais os pacotes a ser recebidos ao longo da transmissão.
- **`write_data`**, preenche e envia os pacotes de dados com dados do ficheiro.
- **`read_data`**, recebe os pacotes de dados e escreve para um novo ficheiro.
- **`send_start_packet`**, preenche e envia o pacote de controlo do tipo `START`.
- **`send_end_packet`**, preenche e envia o pacote de controlo do tipo `END`.
- **`show_transf_progress`**, mostra o progresso alcançado no envio e receção de pacotes de dados.

Macros Importantes

- `FILE *file`, ficheiro aberto para leitura ou escrita dependendo se a aplicação estiver a desempenhar o papel de transmissor ou recetor respetivamente.
- `int used_bytes`, variável que auxilia o cálculo do número de pacotes de dados necessários.

D. Casos de Uso Principais

O principal caso de utilização deste trabalho trata-se do envio de um ficheiro entre duas máquinas ligadas pela **porta de série**. Tal é possível através da passagem de argumentos pela linha de comandos na execução do programa, sendo possível definir a **porta de série** a utilizar e o **papel** da máquina na transferência – recetor ou transmissor – sendo, neste último, necessário indicar o **nome do ficheiro** a transferir.

```
[RECEIVER]                          [TRANSMITTER]
Usage: program serial_port role     Usage: program serial_port role file_name
e.g.: nserial /dev/ttyS1 receiver   e.g.: nserial /dev/ttyS1 transmitter img.png
```

Figura **1**: Argumentos da Linha de Comandos

Fluxo Normal

As etapas normais de execução seguem a seguinte ordem:

1. Escolha da porta de série, papel e ficheiro a enviar.
2. Estabelecimento da ligação entre as duas máquinas.
3. Leitura do ficheiro e envio dos dados pelo transmissor.
4. Receção e escrita, pelo recetor, dos dados num novo ficheiro.
5. Fecho dos ficheiros abertos anteriormente.
6. Terminação da ligação.

Durante o decorrer da transferência, é apresentado um *log* do que está a decorrer em cada um dos lados da transmissão, como o exemplo da **Figura 2**.

```
*** Connection established. ***
        Start packet sent!
                File name: pinguim.gif
                File size: 10968 bytes
        Sending data...
                Packet size: 128 bytes
                [Packet 85 of 85] 100.00%
        End packet send!
        [Time elapsed: 3.693529 seconds]
*** Connection terminated. ***
```

Figura **2**: Transmitter Log

E. Protocolo de Ligação Lógica

O protocolo de ligação lógica envolve os seguintes passos:

1. Estabelecimento da conexão entre os dois computadores.
2. Envio e receção de tramas `I`.
3. Término da ligação.

```
int llopen(int port, int flag);
```

Esta função é responsável por estabelecer a conexão entre o transmissor e o recetor. O emissor começa por enviar uma trama de controlo `SET` e fica a aguardar por uma trama `UA` de confirmação. Por sua vez, o recetor começa por esperar por uma trama `SET` e assim que a recebe, responde com `UA`. Visto que a troca de tramas pode potencialmente não acontecer

devido a falhas em ambos os intervenientes, o transmissor faz uma retransmissão do SET após um período definido de espera e até um máximo de tentativas indicadas, acabando por fazer TIMEOUT se ultrapassar esse limite.

Ambos utilizam uma *state machine* que verifica a integridade das tramas que recebem. Caso esta operação seja efetuada com sucesso, procede-se à fase seguinte.

```
int llwrite(int fd, unsigned char *buffer, int length);
```

Na função `llwrite`, é feito o envio de tramas `I` e a receção das respetivas tramas `RR`. Após o envio de uma trama `I` é feito um intervalo de espera durante o qual o recetor deve responder com uma trama `RR`. Na enventualidade de tal não acontecer no tempo previsto, tal como anteriormente, sucede-se um `TIMEOUT`. A validação das tramas `RR` recebidas é feita utilizando uma máquina de estados semelhante à da função `llopen`.

```
void send_w(); void send();
```

É com recurso a estas função que são preenchidas e enviadas as tramas `SET` e `I`. Na ocorrência de alguns *bytes* das tramas `I` terem o valor `0x7d` (octeto de `ESCAPE`) ou `0x7e` (octeto `FLAG`), é efetuada a operação de **byte stuffing**, sendo em função disso ajustados os campos necessários da trama, nomeadamente o `BCC2`.

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

A função `llread` trata de receber as tramas enviadas em `llwrite` e de responder adequadamente, quer por tramas `RR`, quer por tramas `REJ`. No início deste método é feita a validação das tramas `I` recorrendo a uma máquina de estados e dependendo dos resultados obtidos é feito o envio de uma trama com `RR` para que seja enviada uma nova trama `I`, ou trama `REJ` para que seja efetuada a retransmissão da última trama.
É tanto nesta função como na `llwrite` que está presente a implementação do método Stop&Wait.
Por último, é importante realçar que é também feita, se necessária, a operação de **destuffing**.

```
int llclose(int fd);
```

Esta função está encarregada de terminar a conexão entre o transmissor e o recetor. Do lado do transmissor é enviada uma trama `DISC`, ficando este à espera de resposta. Uma vez

processada a trama DISC enviada pelo transmissor, é enviada uma trama com o mesmo campo de controlo pelo recetor. Finalmente, o transmissor recebe a resposta do recetor e envia uma trama UA, dando assim por concluída a conexão.

```
void llwrite_alarm_handler(); void llopen_alarm_handler();
```

A funcionalidade do TIMEOUT é efetuada fazendo uso das funções llwrite_alarm_handler e llopen_alarm_handler que tratam das chamadas das funções send_w e send sempre que necessário.

F. Protocolo de Aplicação

Este protocolo consiste em várias etapas:

1. Obtenção de informações sobre o ficheiro como o nome e o tamanho em *bytes*.
2. Preenchimento e envio de pacotes de controlo do tipo START e END com os dados acima mencionados.
3. Receção de pacotes de controlo do tipo START e END.
4. Leitura do ficheiro a ser enviado e preenchimento e envio de pacotes de dados (do tipo DATA) com as informações obtidas.
5. Receção de pacotes de dados (DATA) e escrita do novo ficheiro.

Obtenção de Dados

A struct stat file_info é preenchida com informação relevante ao ficheiro recorrendo à função stat(file_name, &file_info), em que file_name é o nome do ficheiro indicado nos argumentos da linha de comandos.

Envio de Pacotes de Controlo

Para o envio dos pacotes de controlo do tipo START é utilizada a seguinte função:

```
int send_start_packet(int fd, unsigned char *control_packet,
                      off_t size, char *file_name);
```

No pacote enviado estão as informações contidas na struct file_info no formato TLV (Type Length Value).

Por sua vez, o envio do pacote de controlo do tipo END é enviado recorrendo à função send_end_packet, apresentada agora:

```
int send_end_packet(int fd, unsigned char *control_packet,
                    int packet_size;
```

Este último pacote é idêntico ao pacote do tipo `START`, com a exceção do campo de controlo que tem o valor `END`.

Receção dos Pacotes de Controlo

A receção dos pacotes de controlo `START` e `END` é feita nesta função:

```
int start_reading(int fd);
```

As informações obtidas no pacote `START` são utilizadas para abrir o ficheiro com o nome correto.

Leitura do Ficheiro e Envio de Pacotes de Dados

A leitura de dados do ficheiro e a respetiva transmissão é feita na função `write_data`:

```
int write_data(int fd, int used_bytes);
```

À medida que são lidos *bytes* do ficheiro, estes são colocados em pacotes de dados, cujo campo de controlo tem o valor `DATA`. Estes pacotes têm também campos que permitem transmitir ao recetor o número de sequência e o tamanho do campo de dados daquele.

Enquanto são enviados pacotes de dados, são feitas chamadas à função `show_transf_progress`, informando o utilizador do progresso no envio de pacotes.
Na eventualidade de corrupção de dados e/ou interrupção do envio de pacotes, é apresentada uma mensagem indicando a rejeição dos mesmos.

Receção de Pacotes de Dados e Escrita do Ficheiro

Os pacotes de dados são recebidos na função `read_data` que abre um novo ficheiro com o nome recebido no pacote de controlo e escreve nele a informação recebida naqueles. A função `read_data` apresenta o seguinte protótipo:

```
int read_data(int fd, int cycles, char *file_name);
```

À semelhança do que acontece no computador que transmite os dados, aqui também são feitas sucessivas chamadas à função `show_transf_progress` para que seja evidente o progresso feito na receção de pacotes.

Outros Detalhes

São também de salientar as funções `start_writting` e `start_reading` que delegam a ordem do envio e receção dos diferentes tipos de pacotes do início ao fim da transmissão. O protótipo da função `start_writting` é o seguinte:

```
int start_writting(int fd, struct stat *file_info, char *file_name);
```

G. Validação

De forma a validar a correta implementação do protocolo de ligação de dados e da aplicação, foram desenvolvidos testes com o objetivo de comprovar a robustez do programa.

Primeiramente, procedemos à variação do tamanho dos ficheiros transmitidos e do tamanho dos pacotes usados. Desta mesma forma, também modificamos a capacidade de ligação (*baudrate*) para envio dos mesmos ficheiros.

De seguida, foi introduzido ruído durante a transmissão de um ficheiro e, temporária e intermitentemente, interrompida a ligação da porta de série.

Por fim, foram introduzidos erros aleatórios simulados em percentagem variável, tendo o programa resistido a todas estas dificuldades.

H. Eficiência do Protocolo de Ligação de Dados

Os testes referidos anteriormente foram também usados para avaliar a eficiência do programa. Para cada um dos fatores foram feitos três testes e calculada a média dos resultados para que esta pudesse ser comparada com os valores esperados.

Tamanho do Ficheiro Transferido

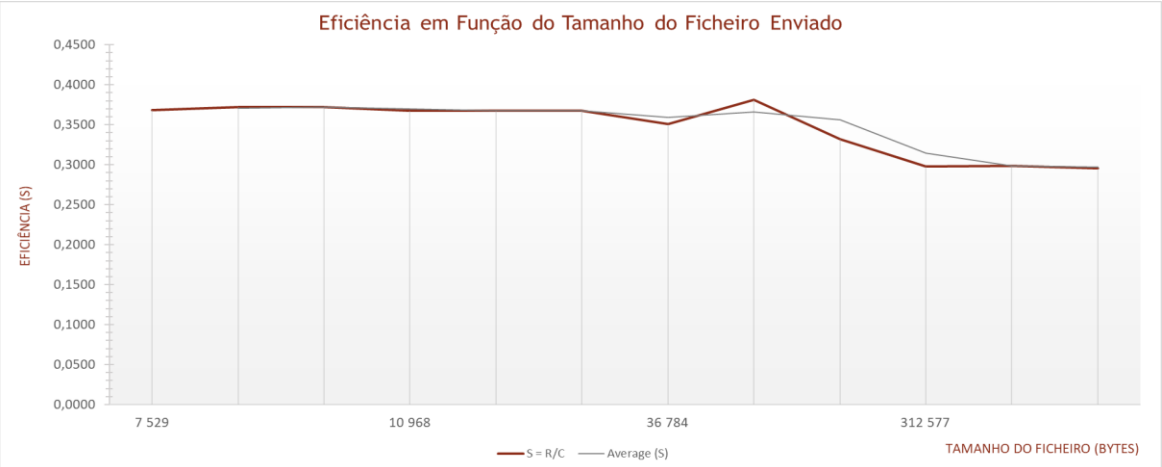

Figura 3: Variação do Tamanho do Ficheiro

A análise do gráfico da **Figura 3**, permite concluir que a eficiência do programa começa a diminuir ligeiramente à medida que o tamanho dos ficheiros aumenta.

Tamanho dos Pacotes de Dados (Trama I)

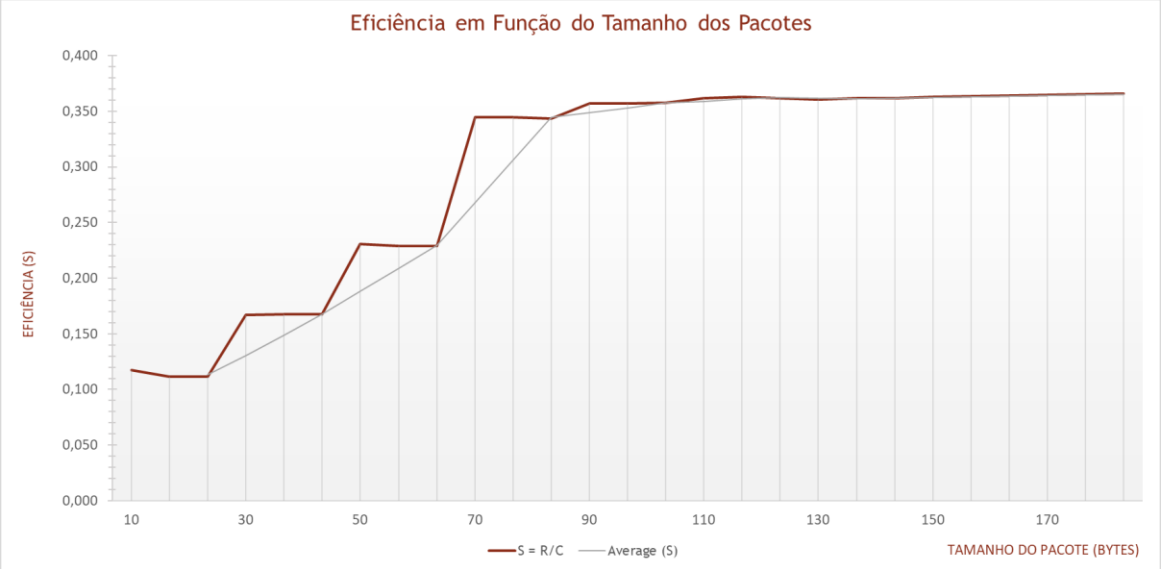

Figura 4: Variação do Tamanho das Tramas de Dados

Pode-se comprovar que apesar de a eficiência aumentar com o aumento do tamanho da trama, esta estabiliza rapidamente, apresentando valores semelhantes para pacotes com tamanho superior a cerca de 100 bytes.

Capacidade de Ligação

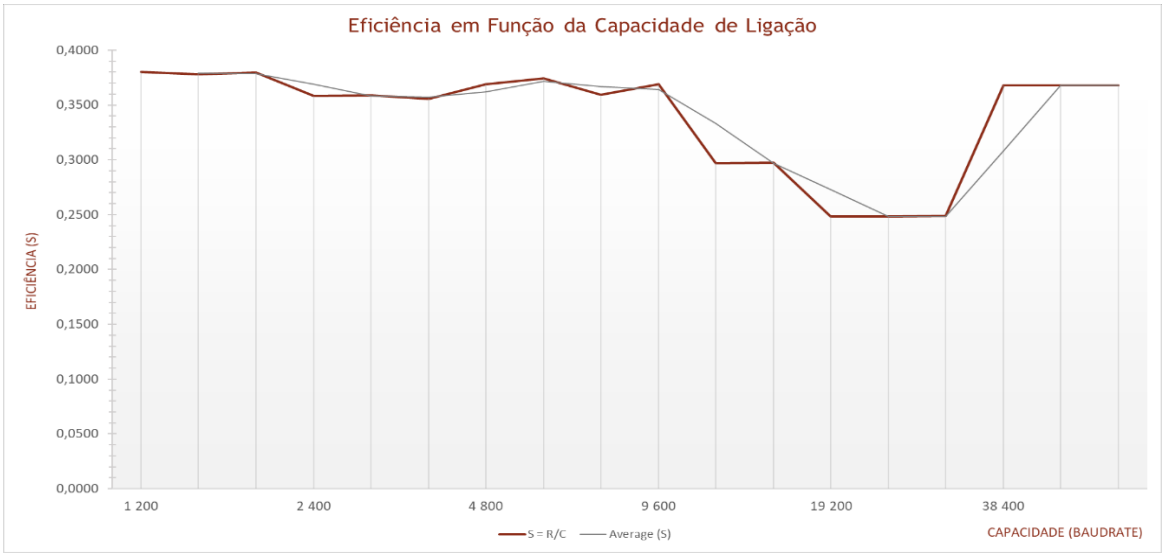

Figura 5: Variação da Capacidade de Ligação

Tal como podemos constatar pela análise do gráfico da **Figura 5**, a eficiência sofre um declínio quando a capacidade apresentada está entre 9600 e 38 400 bits/s, sendo quase constante para os restantes valores.

Ocorrência de Erros Simulados

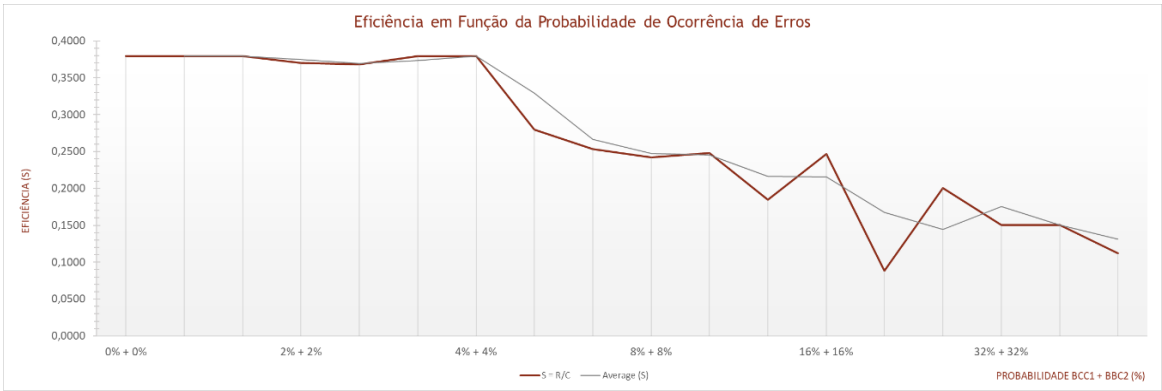

Figura **6**: Variação da Ocorrência de Erros Simulados

Concluímos, pela leitura do gráfico, que a eficiência do programa tende a diminuir com o aumento da probabilidade de ocorrência de erros.

Todos estes resultados foram comparados com o valor teórico de Stop&Wait estando a sua documentação detalhada no **Anexo I**.

Conclusões

Em suma, reiteramos que cumprimos os objetivos propostos para este projeto. A aplicação desenvolvida é capaz de realizar o envio de ficheiros entre computadores através da porta de série, estando apta a reagir de forma adequada na eventualidade de interrupção da conexão ou corrupção de dados.

Salientamos ainda as iniciativas tomadas para garantir que informações relevantes, nomeadamente o nome, tamanho e o estado da transmissão do ficheiro, estão disponíveis ao utilizador da aplicação, procurando assim uma melhor compreensão do processo a decorrer.

Na nossa opinião, este projeto revelou-se muito útil, facilitando, através de uma abordagem prática, a consolidação de conceitos teóricos.

Anexo I

File Size (bytes)	Time (s)	R = bits/time	S = R/C	Average (T)	Average (R)	Average (S)	Expected(S)	Variation (%)
7 529	4,2590	14 142,2537	0,3683	4,2287	14 243,8726	**0,3709**	0,6979	46,85%
	4,2140	14 293,3759	0,3722					
	4,2132	14 295,9881	0,3723					
10 968	6,2122	14 124,4648	0,3678	6,2121	14 124,8058	**0,3678**	0,6997	47,43%
	6,2120	14 125,0332	0,3678					
	6,2120	14 124,9195	0,3678					
36 784	21,8365	13 476,1832	0,3509	21,6784	13 618,7364	**0,3547**	0,7073	49,86%
	20,1002	14 640,2669	0,3813					
	23,0987	12 739,7591	0,3318					
312 577	218,7002	11 433,9920	0,2978	218,9608	11 420,5848	**0,2974**	0,7423	59,94%
	217,9874	11 471,3776	0,2987					
	220,1947	11 356,3849	0,2957					

Figura **1**: Eficiência em Função do Tamanho do Ficheiro Enviado

Packet Size (bytes)	Time (s)	R = bits/time	S = R/C	Average (T)	Average (R)	Average (S)	Expected (S)	Variation (%)
10	19,5082	4 497,8078	0,1171	20,1368	4359,5471	**0,1135**	0,8830	87,14%
	20,4507	4 290,5217	0,1117					
	20,4517	4 290,3119	0,1117					
30	13,6629	6 422,0627	0,1672	13,6576	6424,5443	**0,1673**	0,8366	80,00%
	13,6550	6 425,7828	0,1673					
	13,6550	6 425,7875	0,1673					
50	9,9116	8 852,6664	0,2305	9,9501	8818,4847	**0,2296**	0,7886	70,88%
	9,9697	8 801,0849	0,2292					
	9,9690	8 801,7029	0,2292					
70	6,6247	13 245,0771	0,3449	6,6327	13229,1216	**0,3445**	0,7132	51,70%
	6,6258	13 242,8582	0,3449					
	6,6476	13 199,4296	0,3437					
90	6,3963	13 717,9521	0,3572	6,3929	13725,1853	**0,3574**	0,7057	49,35%
	6,3968	13 716,7727	0,3572					
	6,3856	13 740,8310	0,3578					
110	6,3179	13 888,1371	0,3617	6,3091	13907,5971	**0,3622**	0,7029	48,47%
	6,2919	13 945,6377	0,3632					
	6,3175	13 889,0164	0,3617					
130	6,3353	13 850,0369	0,3607	6,3202	13883,1225	**0,3615**	0,7033	48,59%
	6,3128	13 899,3790	0,3620					
	6,3125	13 899,9515	0,3620					
150	6,2975	13 933,2366	0,3628	6,2861	13958,4800	**0,3635**	0,7021	48,23%
	6,2861	13 958,4495	0,3635					
	6,2747	13 983,7538	0,3642					
170	6,2633	14 009,1501	0,3648	6,2520	14034,6697	**0,3655**	0,7010	47,86%
	6,2520	14 034,6387	0,3655					
	6,2406	14 060,2203	0,3662					

Figura **8**: Eficiência em Função do Tamanho dos Pacotes

Capacity (Baudrate)	Time (s)	R = bits/time	S = R/C	Average (T)	Average (R)	Average (S)	Expected (S)	Variation (%)
1 200	192,4593	455,9095	0,3799	192,8357	455,0219	**0,3792**	0,9864	61,56%
	193,4736	453,5192	0,3779					
	192,5743	455,6371	0,3797					
2 400	102,0031	860,2088	0,3584	147,5508	858,0126	**0,3575**	0,9746	63,32%
	101,9472	860,6805	0,3586					
	102,8473	853,1484	0,3555					
4 800	49,5295	1 771,5506	0,3691	49,7507	1764,1776	**0,3675**	0,9491	61,28%
	48,8472	1 796,2943	0,3742					
	50,8753	1 724,6880	0,3593					
9 600	24,7799	3 540,9344	0,3688	28,7658	3082,0772	**0,3210**	0,9144	64,89%
	30,7654	2 852,0342	0,2971					
	30,7522	2 853,2630	0,2972					
19 200	18,3889	4 771,5840	0,2485	18,3868	4772,1142	**0,2485**	0,8733	71,54%
	18,3922	4 770,7128	0,2485					
	18,3794	4 774,0457	0,2486					
38 400	6,2122	14 124,4648	0,3678	6,2121	14124,8058	**0,3678**	0,6997	47,43%
	6,2120	14 125,0332	0,3678					
	6,2120	14 124,9195	0,3678					

Figura **9**: Eficiência em Função da Capacidade de Ligação

Probability of Error (BCC1 + BCC2)	Time (s)	R = bits/time	S = R/C	Average (T)	Average (R)	Average (S)	Expected(S)	Variation (%)
0% + 0%	6,0270	14 558,4868	0,3791	6,0274	14 557,4764	**0,3791**	0,6933	45,32%
	6,0293	14 552,8463	0,3790					
	6,0259	14 561,0961	0,3792					
2% + 2%	6,1722	14 215,9317	0,3702	6,1376	14 298,4094	**0,3724**	0,6971	46,58%
	6,2120	14 125,0332	0,3678					
	6,0287	14 554,2632	0,3790					
4% + 4%	6,0292	14 553,1456	0,3790	7,7405	11 673,0056	**0,3040**	0,7381	58,82%
	8,1660	10 745,1062	0,2798					
	9,0265	9 720,7651	0,2531					
8% + 8%	9,4415	9 293,4189	0,2420	10,3450	8 634,8586	**0,2249**	0,7921	71,61%
	9,2132	9 523,7165	0,2480					
	12,3802	7 087,4404	0,1846					
16% + 16%	9,2713	9 464,0041	0,2465	15,5165	6 854,0316	**0,1785**	0,8276	78,43%
	25,8976	3 388,1145	0,0882					
	11,3806	7 709,9761	0,2008					
32% + 32%	15,1727	5 783,0220	0,1506	16,9254	5 284,2560	**0,1376**	0,8616	84,03%
	15,2196	5 765,1975	0,1501					
	20,3840	4 304,5484	0,1121					

Figura **10**: Eficiência em Função da Probabilidade de Ocorrência de Erros