2 DE NOVEMBRO DE 2022

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

1º TRABALHO LABORATORIAL – LIGAÇÃO DE DADOS

DANIEL GOMES SILVA - 201909935
LEANDRO SILVA - 202008061
3LEIC01

Sumário

No âmbito da unidade curricular de Redes de Computadores (RC), desenvolveu-se um protocolo de ligação de dados como 1º trabalho laboratorial, de acordo com a especificação descrita no guião. Este protocolo foi testado com uma aplicação simples de transferência de ficheiros, igualmente especificada.

O trabalho desenvolvido cumpriu os objetivos estabelecidos podendo até dizer-se que foi finalizado com sucesso, capaz de transferir dados mesmo com a adversidade da existência de ruído ou a interrupção da ligação.

1. Introdução

A principal finalidade deste projeto é implementar um protocolo de ligação de dados, cujo objetivo é fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio de transmissão, neste caso, um cabo série.

Deste modo, neste relatório, explica-se como é feita a transmissão entre as duas máquinas – quais os procedimentos, todo o desenrolar da ação, bem como uma especificação e explicação das estruturas do código desenvolvido.

Por fim, é explicado quais os testes a que o código desenvolvido foi submetido e é apresentada uma caracterização estatística da eficiência do protocolo, de forma a ser possível ter uma perceção concreta do desempenho do mesmo.

2. Arquitetura

No âmbito deste trabalho existem, principalmente, duas camadas independentes: ligação de dados e aplicação.

Na camada de ligação de dados, o objetivo é fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio (canal) de transmissão, neste caso, um cabo série. Sincronismo (delimitação) de trama, estabelecimento/terminação da ligação, numeração de tramas, confirmação positiva, controlo de erros (por exemplo *Stop-and-Wait*) e controlo de fluxo são funções genéricas de protocolos de ligação de dados. Esta camada comunica com a porta série utilizando a API do *driver* da porta série.

A camada de aplicação usa o serviço fiável oferecido pelo protocolo de ligação de dados através da Interface protocolo-aplicação (para transferir um ficheiro) e não conhece os detalhes do protocolo de ligação de dados, mas apenas a forma como acede ao serviço.

3. Estrutura do código

Na camada de aplicação existem duas funções principais – ***alwrite*** e ***alread*** – que são utilizadas pelo Emissor e o Recetor respetivamente. No caso do Emissor a função de escrita começa pelo envio de um pacote de controlo (*start*) que sinaliza o início da transmissão do ficheiro, em seguida envia os pacotes de dados (contendo fragmentos do ficheiro a transmitir) e termina com o envio do pacote de controlo (*end*) que sinaliza o fim da transmissão. A outra máquina, o Recetor, executa a função de leitura que irá receber os pacotes de controlo e de dados, processando-os devidamente.

A fim de existir uma comunicação entre a camada de aplicação e o protocolo de ligação de dados, foram criadas quatro funções - ***llopen, llwrite, llread, llclose*** - que servem de interface ao lidar com os pedidos da camada superior para a camada inferior.

A função que é executada antes de qualquer operação, ***llopen***, é responsável pela configuração da porta série e estabelecimento da ligação entre os dois computadores que estão a correr o programa. Como o nome indica, a função ***llwrite*** é usada para fazer

envios de pacotes de informação, já a ***llread*** tem como objetivo fazer a leitura dos mesmos. Ambas funções retornam o número de caracteres escritos e lidos, respetivamente, ou um valor negativo em caso de erro. Por fim, a função ***llclose*** fecha a ligação entre os dois computadores e repõe as configurações da porta série.

A estrutura de dados, ***LinkLayer***, guarda a porta série utilizada, o *LinkLayerRole* (Emissor ou Recetor), a *baud rate*, o número de tentativas máximo de retransmissões e o valor do temporizador (*time-out*).

4. Casos de uso principais

O programa começa por chamar a função ***llopen***, quer do ponto de vista do Recetor como do Emissor, embora o comportamento da mesma seja diferente. Isto é, no caso do Emissor, após a configuração da porta série e estabelecimento da ligação, é enviada uma trama não numerada (U) do tipo **SET** (*set up*), e espera outra do tipo **UA** (*unnumbered acknowledgment*), através das funções genéricas para o envio e receção de tramas U e S - ***sendSUFrame*** e ***receiveSUFrame*** - cujos argumentos especificam o *file descriptor* da porta série e os campos de endereço e de controlo. No caso do Recetor é feito o inverso, ou seja, espera uma trama de **SET** e envia uma trama **UA**. No envio da trama **SET** é ativado um temporizador (*alarm*) que será desativado após a receção de uma reposta válida. Na receção das tramas **SET** e **UA** é utilizada uma máquina de estados própria para este tipo de tramas – ***setUaDiscStateMachine*** – que é chamada a partir da ***receiveSUFrame***.

Tendo configurado a porta série e por sua vez a fase de **estabelecimento** ter terminado com sucesso, começa a fase de **transferência de dados**. Assim, no que diz respeito ao Emissor é feita a seguinte sequência de chamada de funções em ***alwrite***:

1) Envio do pacote de controlo (*start*) com uma chamada à ***send_control_packet***, que por sua vez, chama ***llwrite***. Em ***llwrite*** é criada uma trama de informação (I) e de modo a garantir a transparência assegurada pela técnica de *byte stuffing*, é feita uma chamada à ***byteStuffing*** antes de ser enviada através da função ***sendIFrame***. Deste modo, evita-se o falso reconhecimento de uma *flag* no interior da trama.
2) Para validar confirmações (positivas ou negativas) enviadas pelo Recetor, são feitas chamadas à ***supervisionStateMachine*** na ***sendIFrame***. Esta máquina de estados é utilizada na leitura das tramas de supervisão (S) dos tipos *receiver ready* (**RR**) e *reject* (**REJ**).
3) O procedimento para o envio dos pacotes de dados (contendo fragmentos do ficheiro a transmitir) é idêntico aos passos 1 e 2, uma vez que, são feitas chamadas à função ***send_data_packet*** que também chama ***llwrite***.
4) Envio do pacote de controlo (*end*) com uma chamada à ***send_control_packet,*** idêntico ao passo 1.

No Recetor é feita a seguinte sequência de chamada de funções em ***alread***:

1) Leitura do pacote de controlo (*start*) com uma chamada à ***llread***. Em ***llread*** é chamada a função ***receiveIFrame***, que por sua vez, faz chamadas à ***informationStateMachine***. Esta máquina de estados é utilizada na leitura de tramas de informação (I).
2) Em ***llread***, depois de ler a trama I, é feito o *byte destuffing* (operação inversa ao *byte stuffing*) com uma chamada à ***byteDestuffing*** e a trama é analisada. Existem dois procedimentos (explicados na secção 5): ignorar, sem qualquer ação ou enviar uma trama S (RR ou REJ) através da ***sendSUFrame***. Se se tratar duma nova trama, sem erros, o campo de dados é aceite (e passado à camada de aplicação).
3) Para processar o pacote lido é chamada ***receive_control_packet*** onde são guardados os campos com o tamanho e o nome do ficheiro.
4) O procedimento para a leitura dos pacotes de dados é idêntico aos passos 1 e 2, uma vez que, são feitas chamadas à função ***llread***. A diferença é no

passo 3, para processar o pacote lido é chamada ***receive_data_packet***, onde é guardado o campo de dados do pacote para posteriormente escrever para o novo ficheiro.

5) Leitura do pacote de controlo (*end*) com uma chamada à ***llread***, idêntico aos passos 1 e 2.

Após a fase de transferência de dados ter terminado, começa a última fase – **terminação**. É feita uma chamada à função ***llclose***, que fecha a ligação entre as duas máquinas e repõe as configurações da porta série. O Emissor envia uma trama U do tipo **DISC** (*disconnect*), que será recebida pelo Recetor que também irá enviar uma trama **DISC**. Por último, o Emissor envia uma trama **UA**. São utilizadas as funções genéricas para o envio e receção de tramas U e S - ***sendSUFrame*** e ***receiveSUFrame***. No envio da trama **DISC** é ativado um temporizador (*alarm*) que será desativado após a receção de uma reposta válida. Na receção das tramas **DISC** e **UA** é utilizada uma máquina de estados própria para este tipo de tramas – ***setUaDiscStateMachine*** – que é chamada a partir da ***receiveSUFrame***.

5. Protocolo de ligação lógica

Na camada de ligação de dados não é feito qualquer processamento que incida sobre o cabeçalho dos pacotes a transportar em tramas I, esta informação é considerada inacessível. Também não existe qualquer distinção entre pacotes de controlo e de dados, nem é relevante (nem tida em conta) a numeração dos pacotes de dados. Em particular, os mecanismos de numeração de tramas I e de pacotes de dados são totalmente independentes e não há nenhuma relação entre eles.

Nesta camada, destacam-se as três máquinas de estados:

- ***setUaDiscStateMachine*** – utilizada pelo Emissor e Recetor para garantir uma leitura correta de comandos **SET** e **DISC** e respostas **UA**.
- ***supervisionStateMachine*** – utilizada pelo Emissor para garantir uma leitura correta de tramas S dos tipos *receiver ready* (RR) e *reject* (REJ) enviadas pelo Recetor. O campo de controlo da trama é guardado na variável ***controlField***, que será usado para o Emissor decidir se deve retransmitir a trama (caso REJ) ou atualizar o número de sequência (caso RR).
- ***informationStateMachine*** – utilizada pelo Recetor na leitura de tramas I para garantir a existência de pelo menos um campo entre duas *flags*. Devido ao mecanismo de *byte stuffing* não existem tramas com uma *flag* no seu interior, logo não existem duas *flags* seguidas.

Na receção de tramas, após a análise, o Recetor deverá enviar uma trama S (confirmação positiva/negativa) ou ignorar a trama recebida. Procedimentos:

- **RR** – se se tratar de uma nova trama, sem erros detetados no cabeçalho e no campo de dados, o campo de dados é aceite (e passado à aplicação) e a trama deve ser confirmada com RR (com o próximo número de sequência). Caso seja um duplicado, sem erros ou com erro detetado apenas no campo de dados, também deve ser confirmada com RR.
- **REJ** – se se tratar de uma nova trama, sem erro detetado no cabeçalho, mas com erro detetado no campo de dados, este é descartado e é feito um pedido de retransmissão com REJ, o que permite antecipar a ocorrência de *time-out* no Emissor. O erro é detetado caso o BCC2 (campo de proteção), penúltimo *byte* da trama, seja diferente da operação ***xor*** entre todos os *bytes* do campo de informação.
- **Ignorar** – se se tratar de uma trama com cabeçalho errado, esta será ignorada, sem qualquer ação e haverá ocorrência de *time-out* no Emissor.

6. Protocolo de aplicação

A camada de aplicação não conhece os detalhes do protocolo de ligação de dados, mas apenas a forma como acede ao serviço – utilizando as funções ***llopen, llwrite, llread, llclose.*** O protocolo de aplicação desconhece a estrutura das tramas e o respetivo mecanismo de delineação, a existência de *stuffing* (e qual a opção adotada), o mecanismo de proteção das tramas, eventuais retransmissões de tramas de informação, etc. Todas estas funções são exclusivamente realizadas na camada de ligação de dados

6.1 Função *alwrite*

A função ***alwrite*** recebe o nome do ficheiro como parâmetro e realiza as seguintes operações: abre o *file* para leitura e obtém o seu tamanho em *bytes*. Depois utiliza-se a função ***send_control_packet*** para enviar o pacote de controlo inicial com o nome do ficheiro e o seu tamanho. Após estes passos, implementou-se um ciclo que percorre todos os *bytes* do ficheiro e a cada **MAX_PAYLOAD_SIZE** *bytes* lidos envia-se um pacote de dados com esses *bytes* através da função ***send_data_packet***. Os bytes finais do ficheiro (que não atingem o MAX_PAYLOAD_SIZE) também são enviados depois do ciclo. Por fim envia-se o pacote de controlo final, novamente com o nome do ficheiro e o seu tamanho.

6.2 Função *alread*

A função **alread** vai essencialmente receber o ficheiro enviado pela função ***alwrite*** realizando as seguintes operações: espera até receber o pacote de controlo inicial e quando o recebe a partir da função ***llread***, dá *parse* do mesmo através da função ***receive_control_packet***. Depois abre um ficheiro novo para escrita com o nome dado no parâmetro da função e entra-se num ciclo que vai receber todos os pacotes de dados e até receber o pacote de controlo final. A cada um destes pacotes de dados é efetuado o seu *parsing* usando a função ***receive_data_packet*** e os dados são escritos no novo ficheiro, recentemente aberto, usando ***fwrite***. Por fim, basta dar *parse* do pacote de controlo final, comparar os campos recebidos neste pacote com os campos recebidos no pacote de controlo inicial e comparar o número de *bytes* enviados com o número de *bytes* de lidos. Caso todas estas igualdades se verificarem, então a escrita do novo ficheiro foi realizada sem erros e pode-se proceder à libertação da memória utilizada em *arrays* e ao fecho do *file* recentemente escrito.

6.3 Funções auxiliares

Para auxiliar na implementação deste protocolo foram implementadas as seguintes funções auxiliares:

• ***send_control_packet****(const unsigned char control_field, unsigned long file_size, const char *filename)*
Recebe como parâmetros um *control field* (que pode ser **C_START** ou **C_END**), o tamanho do ficheiro e o nome do ficheiro. Esta função envia para a camada de ligação, usando ***llwrite***, o pacote de controlo inicial ou final dependendo do valor do *control field*.

• ***send_data_packet****(unsigned char *data, unsigned long size_data)*
Recebe como parâmetros um *array* de caracteres (dados a enviar) e o tamanho desse *array*. Esta função envia um pacote de dados (fragmentos do ficheiro) para a camada de ligação.

• ***receive_control_packet****(unsigned char *packet, unsigned long *file_size, char *file_name)*

Recebe como parâmetros um *array* de caracteres com um pacote de controlo. Esta função vai realizar o *parsing* do pacote de controlo fornecido e colocar nos parâmetros de *output* os valores presentes neste pacote, nomeadamente o tamanho e nome do ficheiro.

• ***receive_data_packet****(unsigned char *packet, unsigned char *packet_data_field, unsigned needed_sequence_number, unsigned *total_bytes_read)*

Recebe como parâmetros um *array* de caracteres com um pacote de dados e um número de sequência. Esta função vai fazer *parse* do pacote de dados fornecido, vai comparar o número de sequência presente no pacote de dados com o valor dado no parâmetro ***needed_sequence_number*** (se forem iguais não ocorreram erros), vai colocar no parâmetro de *output* ***packet_data_field*** todos os dados presentes no pacote de dados e o valor do parâmetro ***total_bytes_read*** será incrementado com número de *bytes* de dados lidos.

7. Validação

De forma a validar o protocolo desenvolvido foram efetuados os seguintes testes:

- Envio de ficheiros de vários tamanhos;
- Interrupção da ligação durante o envio do ficheiro até o número de tentativas máximo de retransmissões não ser excedido e até ser excedido;
- Geração de ruído na ligação durante o envio do ficheiro;
- Envio de ficheiro com a variação do tamanho dos pacotes (MAX_PAYLOAD_SIZE**),** T_PROP, BAUDRATE, número de tentativas máximo de retransmissões (N_TRIES) e o valor do temporizador (TIMEOUT);
- Envio de ficheiro com geração de erros, no cabeçalho e no campo de dados, em tramas I, com probabilidades predefinidas (e independentes) – HEADER_FER e DATA_FIELD_FER

Todos os testes foram concluídos com sucesso.

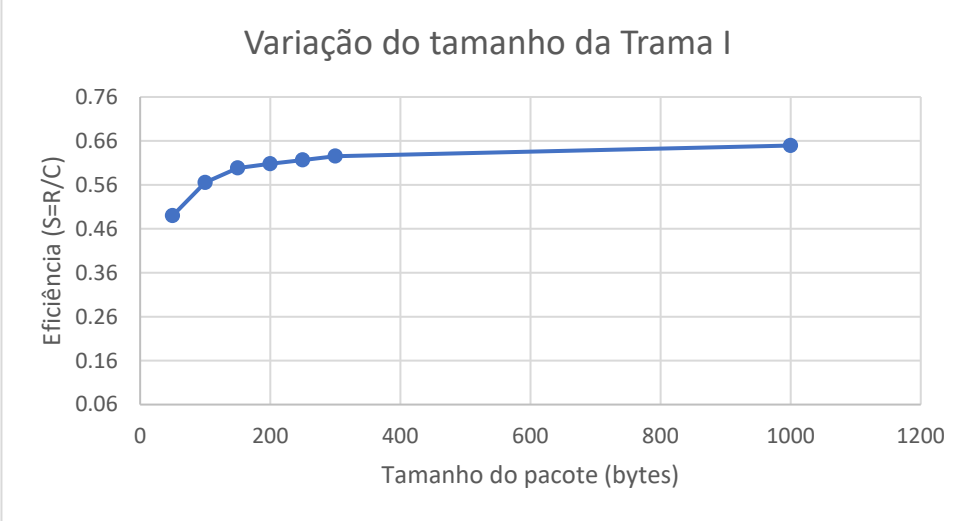
8. Eficiência do protocolo de ligação de dados

De forma a realizar a caracterização estatística da eficiência do protocolo, foram efetuadas medidas no laboratório recorrendo ao código desenvolvido. Foram testados quatro parâmetros – variação do tamanho da trama I, FER, T_prop e C (capacidade de ligação).

O ficheiro enviado foi o *penguin.gif* com 10968 *bytes* e os valores por defeito (quando não são variados) do baud rate e MAX_PAYLOAD_SIZE (tamanho do pacote) são 9600 e 1000 respetivamente. O *time-out* é 4 segundos. Para as mesmas condições foram feitas sempre duas medições e a sua média para diminuir o desvio dos resultados.

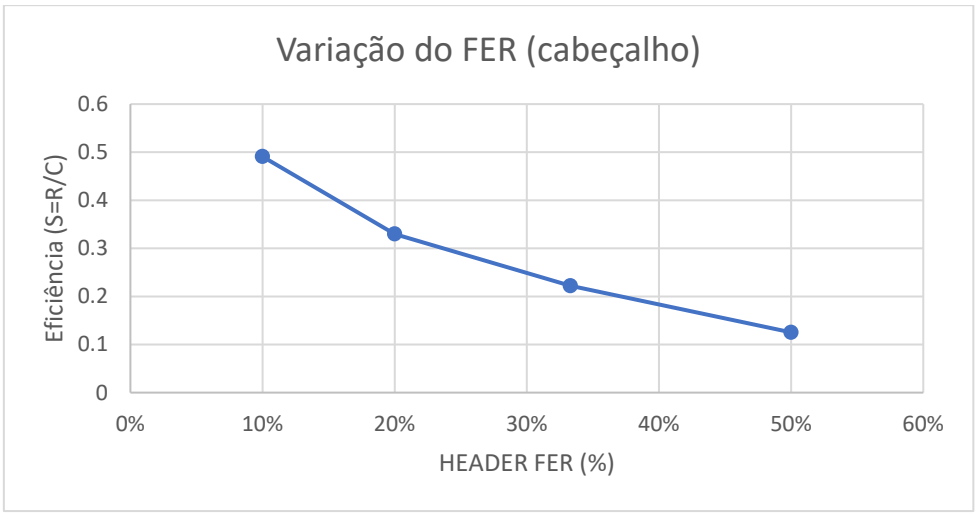
8.1 Variação do tamanho da trama I

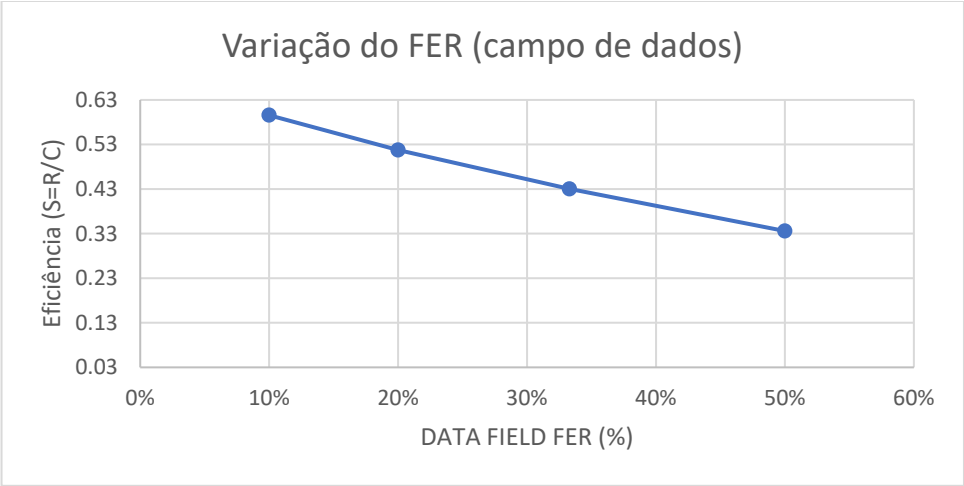
Notou-se um aumento da eficiência com o aumento do tamanho dos pacotes (consequente aumento do tamanho das tramas I) a serem enviados. Isto acontece, uma vez que, como são enviados mais dados por pacote são enviados menos pacotes. O tempo de transmissão irá ser menor para a mesma quantidade de dados enviados. Tamanhos dos pacotes usados – 50, 100, 150, 200, 250, 300 e 1000.



8.2 Variação do FER

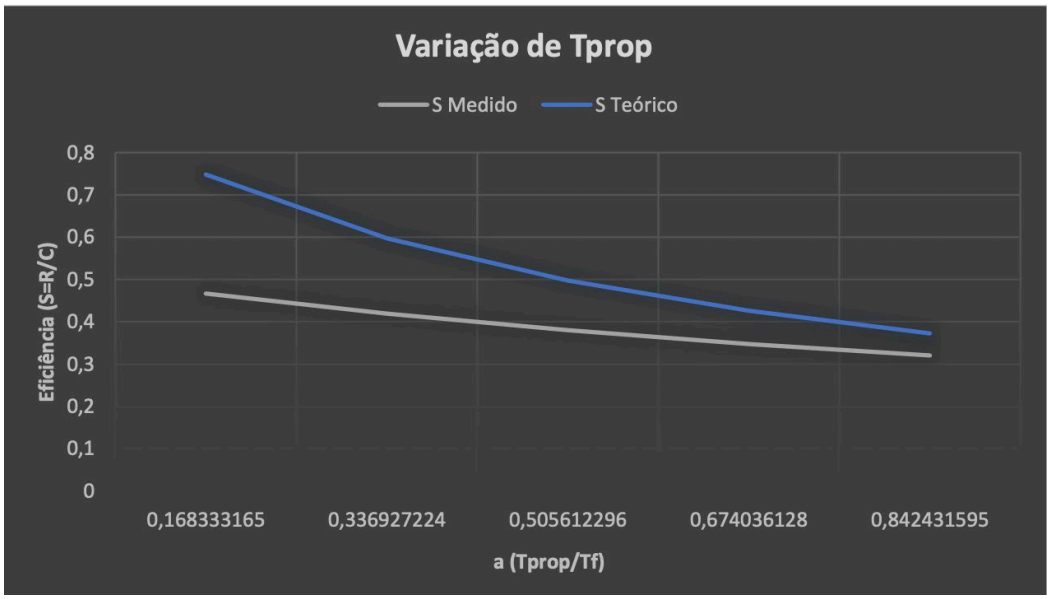
Como era esperado, é possível confirmar que com o aumento da probabilidade de ocorrência de erros, quer no cabeçalho quer no campo de dados, a eficiência diminui. No entanto, o erro no cabeçalho tem maior impacto na eficiência, porque as tramas com esse tipo de erro são ignoradas pelo Recetor, sem qualquer ação e há sempre ocorrência de time-out no Emissor, obrigando a pelo menos uma retransmissão.





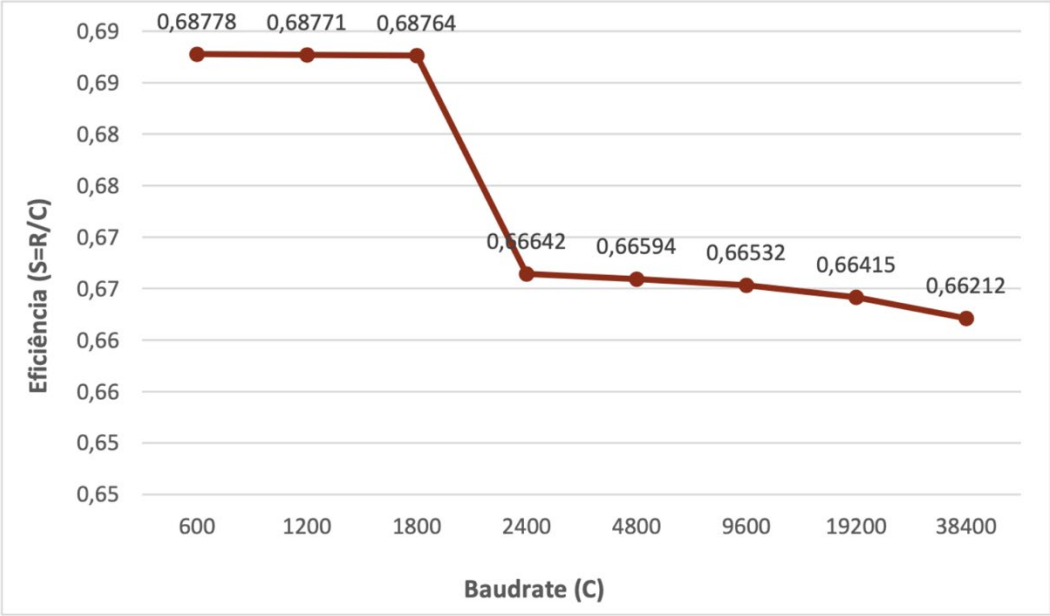
8.3 Variação do T_prop

Como é possível analisar através do gráfico seguinte, percebe-se que o valor da eficiência diminui com o aumento de T_prop e consequentemente o aumento de a. Pode-se também verificar que os resultados obtidos foram bastante positivos pelo que pouco se diferenciam dos valores teóricos.



8.4 Variação da C (capacidade da ligação)

A eficiência diminui ligeiramente com o aumento da capacidade da ligação. Teoricamente faz sentido, uma vez que, a eficiência é o resultado da divisão do débito recebido pela capacidade de ligação. O débito recebido é diretamente proporcional à capacidade de ligação, logo a eficiência deverá manter-se constante.



9. Conclusões

Com o desenvolvimento deste projeto, colmatamos certos conhecimentos teóricos lecionados nas aulas teóricas. Deste modo, conseguimos ter uma melhor perceção sobre o princípio de **independência entre camadas**, pois acabamos por ter de desenvolver diferentes camadas (*application layer* e *link layer*) que em nada dependiam uma da outra.

Por outro lado, com o desenvolvimento do protocolo, conseguimos compreender alguns dos cuidados que é necessário ter para se conseguir desenvolver um meio que proporciona uma comunicação de dados fiável entre dois sistemas.

Assim, consideramos que este desenvolvimento trouxe bastantes benefícios, pois para além de ter sido uma excelente forma de entrar em contacto com os conteúdos da unidade curricular, também nos ajudou a entender melhor o funcionamento e o conceito de muitos temas desta área.

Anexo I – Código fonte

O código fonte encontra-se disponível no repositório partilhado com o docente e na pasta *code*.