

Redes de Computadores

**1º Trabalho Laboratorial**

6 de novembro de 2017

Carlos Freitas

David Falcão – up201506571

Luís Martins – up201503344

\*completar com UPs\*

**Índice**

**No index entries found.**

**Sumário**

No âmbito do primeiro trabalho laboratorial de Redes de Computadores, este relatório tem como objetivo realizar uma análise estatística ao comportamento do protocolo implementado para a transmissão e receção de ficheiros através de uma porta de série assíncrona.

O trabalho foi realizado com sucesso, uma vez que todos os objetivos propostos foram atingidos, verificando-se que o protocolo é capaz de assegurar uma transmissão correta dos dados, mesmo que ocorram alterações nos pacotes durante a mesma, como por exemplo interrupções ou curtos-circuitos na ligação, alterando os dados.

**Introdução**

O objetivo do trabalho realizado era implementar um protocolo de ligação através da porta de série, utilizando a técnica *Stop and Wait*, de modo que fosse resistente a alguns fatores de perturbação, como interrupções e curtos-circuitos durante a sua execução. Para tal, foi utilizada a abstração do envio de um ficheiro de teste, para verificar a funcionalidade e resistência do protocolo implementado.

Para garantir coesão nos dados enviados e recebidos, foram utilizadas algumas técnicas de proteção, nomeadamente adição de bytes de controlo e técnica de transparência (*framming*), permitindo assim também fazer um controlo ao fluxo do programa.

O presente relatório tem como objetivo analisar e interpretar o resultado obtido em relação à implementação do protocolo, analisando os dados obtidos através de testes.

Assim, este relatório está organizado da seguinte forma:

* **Arquitetura e estrutura de código:** Demonstração dos blocos funcionais e interfaces. Principais estruturas de dados, principais funções e a sua relação com a arquitetura.
* **Casos de Uso Principais:** Identificação dos principais casos de uso e sequência da chamada de funções.
* **Protocolo de ligação lógica:** Identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes.
* **Protocolo de aplicação:** Identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes.
* **Validação:** Apresentação dos testes efetuados e apresentação quantificada dos resultados obtidos.
* **Eficiência do protocolo de ligação de dados:** Comparação do programa com mecanismos *Stop and Wait* de *ARQ* e apresentação da eficiência do mesmo através dos testes apresentados.
* **Conclusão:** Síntese do projeto realizado e reflexão sobre os principais objetivos de aprendizagem alcançados.

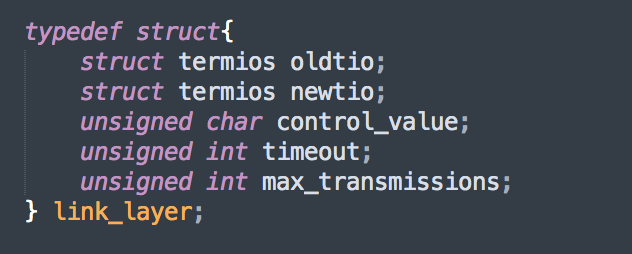
**Arquitetura e Estrutura do código**

O trabalho está organizado em duas principais camadas lógicas, que permitem uma correta organização e funcionamento do protocolo, tornando-o modular.

**Camada de Ligação (Data Link Layer)**

A camada de Ligação é a camada de software “mais baixa” e é responsável pela comunicação e verificação da coesão dos resultados provenientes da porta de série.

Deste modo, esta camada contém todas as funções relativas ao estabelecimento da ligação entre os terminais (*LLOPEN*), escrita e leitura da informação utilizando a porta série (*LLWRITE/LLREAD*), e respetiva análise em relação à sua coesão (*state\_machine* e *verify\_bcc2*). Esta camada é também responsável pelo fecho da ligação no fim da comunicação (*LLCLOSE*).

Para uma melhor estruturação, esta camada contém uma estrutura de dados (struct), por forma a manter algumas informações importantes para a execução das funções de leitura e escrita de e para a porta de série, nomeadamente:

* o tempo de timeout
* o número máximo de retransmissões
* o valor de controlo da última trama, para que se possa verificar se se trata de um duplicado

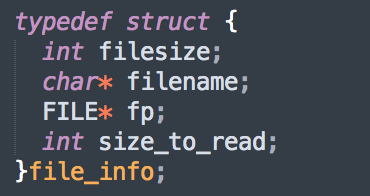
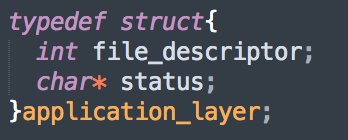
**Camada de Aplicação (Application Layer)**

A camada de Aplicação é a camada “mais alta” e é responsável pela inicialização do programa (*main*). É a partir desta que se desenrola o processo de transferência de dados através da porta de série. Esta camada é também responsável pela criação dos pacotes que contêm os dados a enviar, tanto as tramas de START e END (*create\_STARTEND\_packet*) como as tramas de Informação (*data\_package\_constructor*).

Assim, esta camada contém, entre outras funções, as funções de leitura (*handle\_readfile*) e de criação/escrita (*handle\_writefile*) do ficheiro a transmitir, no caso de o programa ser executado no modo de escritor ou no modo de leitor, respetivamente.

Para uma melhor estruturação, esta camada contém duas estruturas de dados essenciais para guardar informação:

* a primeira relativa ao ficheiro de leitura/escrita (nome, tamanho, ...)
* a segunda relativa aos dados a enviar para a camada de ligação, que se referem à comunicação com a porta de série (descritor da porta)



**Casos de Usos Principais**

A aplicação resultante possui vários casos de uso, nomeadamente os que são responsáveis pela transferência do ficheiro e pela inserção de dados por parte do utilizador. Assim, a transferência do ficheiro dá-se da seguinte forma:

**No caso do emissor:**

O emissor preenche a estrutura file com os dados do ficheiro necessários à abertura e leitura do mesmo. O restante funcionamento do programa divide-se em 3 partes fundamentais, e sequenciais:

* Criação e envio da trama START utilizando as funções *create\_STARTEND\_packet()* e *send\_message()*, sendo que esta última função envia a trama para a camada de ligação através da chamada à função *LLWRITE*;
* Leitura e envio dos vários pacotes de dados provenientes do ficheiro. Para tal, é chamada a função *handle\_readfile()* que lê o ficheiro e chama a função *send\_message()* para envio da informação lida;
* Por fim, é criada e enviada, de forma idêntica à START, a trama END.

De referir que a função *send\_message(),* no caso das tramas de Informação, invoca a função *data\_package\_constructor()* que adiciona o cabeçalho ao pacote, nomeadamente os valores do número do pacote e o tamanho de dados a enviar.

É também importante referir que a função *LLWRITE*, adiciona a restante informação necessária à trama através da chamada à função *create\_frame(),* que por sua vez calcula o valor de BCC2 e invoca as funções *byte\_stuffing()* e *add\_frame\_header(),* para que seja realizada a função de transparência e sejam adicionados os cabeçalhos. No fim, a função *LLWRITE* envia a mensagem através da chamada de sistema *read()*.

**No caso do recetor:**

É invocada a função *get\_message()* que espera uma mensagem proveniente do emissor, através da chamada à função *LLREAD*. A função *LLREAD* lê e retorna o pacote recebido após verificar se existe a coesão dos dados recebidos (BCC1, BCC2, e duplicados). Posteriormente, esta trama é analisada e verifica-se de que tipo de trama se trata. Caso se trate de um pacote de informação, é invocada a função *get\_only\_data()* que verifica de qual dos seguintes tipos de pacotes se trata, retornando o pacote lido:

* Mensagem START: chama a função *start\_message(),* que preenche os dados da estrutura file com os respetivos dados para a criação do ficheiro, criando-o;
* Pacotes de Informação: chama a função *get\_only\_data(),* que remove o cabeçalho do pacote e posteriormente chama a função *handle\_writefile(),* que escreve os dados para o ficheiro de destino;
* Mensagem END: chama a função *verify\_end(),* que verifica se o tamanho do ficheiro recebido corresponde ao tamanho recebido na trama final;
* Se a trama recebida for um DISC ou nula, nada é analisado.

O recetor executa a função *get\_message()* enquanto esta não retornar uma trama DISC, sinal de que a execução do programa terminou e a porta de série já foi fechada.

Em ambos os casos vai ser executada no início a função *LLOPEN,* para abrir e configurar a porta de série.

Quanto à interação com o utilizador, este apenas deve iniciar o programa executando através do terminal um comando com a seguinte estrutura:

* **Emissor:** {Nome da aplicação} {nome da porta de série (ex: /dev/ttyS0)} {modo de abertura (w)} *{nome do ficheiro a enviar}* {tempo de timeout} {número de tentativas máximas de reenvio}
* **Recetor:** {Nome da aplicação} {nome da porta de série (ex: /dev/ttyS0)} {modo de abertura (r)} {tempo de timeout} {número de tentativas máximas de reenvio}

**Protocolo de Ligação Lógica**

A camada de ligação é uma das camadas implementadas neste projeto. Esta camada é responsável pela comunicação e coesão da mesma através de verificações feitas aos valores BCC1, BCC2 e C1. Assim, esta camada tem as seguintes funcionalidades:

* Estabelecer e terminar uma ligação através da porta de série (Envio e receção das tramas SET e UA);
* Escrever e ler mensagens através da porta série (Envio e receção das tramas RR, REJ e I);
* Adicionar e remover os cabeçalhos necessários para o envio das tramas S e I referentes ao envio dos pacotes de informação, e respetivo *stuff/destuff*;
* Analisar os cabeçalhos das mensagens recebidas enviando a respetiva trama de resposta.

Deste modo, as principais funções desta camada são as seguintes:

**LLOPEN**

***int*** *LLOPEN(****char\**** *port,* ***char\**** *mode,* ***char\**** *timeout,* ***char\**** *max\_transmissions) ;*

A função *llopen* é responsável por estabelecer a ligação através da porta de série.

Quando o emissor invoca esta função é feito o envio do comando SET e aguarda a respetiva resposta UA. Em caso de sucesso, a ligação fica estabelecida e a função retorna o valor do descritor da ligação através da porta de série. Caso contrário, existem as seguintes possibilidades:

* se UA não for recebido: a trama SET é reenviada passados timeout segundos até um máximo de max\_transmissions tentativas. Caso seja atingido o número máximo de tentativas, a função retorna -1, indicando que a ligação não foi estabelecida.
* caso exista resposta por parte do recetor mas esta não seja afirmativa, o emissor reenvia a trama SET até que a resposta seja afirmativa.

Já quando o recetor invoca esta função, fica a aguardar a chegada de um comando SET e quando o receber, envia como resposta UA. Deste modo, a ligação fica estabelecida.

**LLWRITE**

***int*** *LLWRITE(****int*** *fd, unsigned* ***char\**** *msg,* ***int\**** *length);*

A função *llwrite* é responsável pelo envio dos pacotes de Informação.

Assim esta função recebe como argumento, entre outros, a mensagem a ser enviada (pacotes de controlo + dados) e o respetivo tamanho. Deste modo, com recurso à função *create\_package()* esta cria a trama com os cabeçalhos necessário (flags, campo de endereço, campo de controlo(c1), BCC1, BCC2 e respetivo stuffing) e envia a mesma através da porta de série.

O envio das tramas de informação é similar ao estabelecimento da ligação por parte do emissor, ou seja, a mensagem é enviada e fica-se a aguardar resposta sendo reenviada a mesma quando for atingido o tempo de timeout. Caso a resposta recebida seja o comando RR (C1 = 0x05/0x85), a mensagem foi recebida corretamente e pode-se enviar o próximo pacote; caso a resposta seja o comando REJ(C1 = 0x01/0x81), o último pacote é reenviado. A função retorna TRUE em caso de sucesso e FALSE caso contrário.

**LLREAD**

***unsigned char\**** *LLREAD(****int*** *fd,* ***int\**** *length);*

A função *llread* é responsável pela leitura dos pacotes de informação provenientes da porta de série. Como tal, esta função, ao ser invocada lê a informação, um char de cada vez, verificando o início e fim da trama(FLAG) e coesão do cabeçalho (BCC1), através da chamada a função *state\_machine().* São também invocadas aqui as funções de introdução de erros aleatórios no cabeçalho e dados. Caso não ocorra nenhum erro no BCC1 a função prossegue.

Nesta mesma função analisa-se a trama recebida verificando os seguintes parâmetros

* se a trama recebida se trata de um DISC (Sinal de terminação de envio das tramas de informação), é invocado LLCLOSE para se proceder ao protocolo de fecho da porta de série;
* se não, é analisada a presença de um duplicado (verificando o valor do campo de controlo), é feito o *destuffing* e a respetiva verificação do valor de BCC2(*verify\_bcc2()*)

Posteriormente é enviada, através da função *send\_response(),* a resposta de acordo com os dados recebidos, segundo a seguinte lógica:

* Trata-se de um duplicado: é enviado RR;
* Não é duplicado e BCC2 está correto: é enviado RR;
* Não é duplicado e BCC2 está errado: é enviado REJ.

Caso seja enviado RR, será o valor correspondente a RR do complemento do tipo de trama recebida, ou seja, caso se receba uma trama 1 (C = 0x40) envia-se RR correspondente a RR0(0x05). Já no caso de ser enviado REJ, será o valor correspondente a REJ do tipo de trama recebida, ou seja, se se receber uma trama 1, envia-se REJ correspondente a REJ(0x81).

**LLCLOSE**

***void*** *LLCLOSE(****int*** *fd,* ***int*** *type);*

A função LLCLOSE é responsável por terminar a ligação através da porta de série. Esta função tem comportamentos distintos caso se trate do recetor ou do emissor.

Assim, no caso do recetor, esta função é chamada na função LLREAD ao ser encontrado um comando DISC. Após isso, o recetor através da chamada à função *send\_disc()* reenvia o comando DISC, aguardando uma resposta UA final.

Já no caso do emissor, esta função é chamada após o envio e receção da respetiva resposta da trama END. É chamada a função *send\_disc()* que envia uma trama DISC e espera uma resposta idêntica.

Em caso de sucesso, a ligação é também terminada no lado do emissor.

**Protocolo de Aplicação**

A camada de aplicação é a camada de mais alto nível implementada neste projeto. Esta camada é responsável pelas seguintes funcionalidades:

* Envio/receção dos pacotes de dados e de controlo
* Leitura/escrita do ficheiro a enviar/recebido.

Uma vez que é nesta camada que se inicia todo o processo, a função *main* tem dois blocos e escolhe qual executar, dependendo se se trata do emissor ou do recetor, parâmetro este que é recebido como argumento da função.

Desta forma, existem funções separadas para ambos os modos, emissor e recetor.

**Emissor**

Na parte do emissor, para garantir a existência de coesão nos dados recebidos são feitas algumas verificações.

Assim, para o envio da informação, o emissor prepara as tramas de início(START) e de fim (END) invocando a função *create\_STARTEND\_packet().* O envio destas tramas permite ao recetor saber qual o nome do ficheiro a criar e, aquando da receção da trama END, verificar se o tamanho de dados recebidos está correto.

É também no protocolo de aplicação que é adicionado o cabeçalho do pacote (*packet header*) no qual vai a informação relativa ao número do pacote e tamanho de dados que o pacote transporta. Este cabeçalho é útil pois, no final, caso o ficheiro esteja corrompido é possível verificar qual(is) os pacotes que falharam, através do ficheiro de logs criado.

**Recetor**

No recetor apenas é chamada a função *get\_message()* que, após executar a função LLREAD, analisa os dados obtidos.

No caso dos Pacotes de Informação (tramas I) é invocada a função *get\_only\_data().* Esta função remove o cabeçalho e, considerando os valores de L1 e L2, lê os dados existentes na trama.

Já no caso de se tratar de uma trama END, é invocada a função *verify\_end()*, que verifica se o tamanho do ficheiro recebido corresponde ao tamanho recebido na trama final, de modo a saber se o ficheiro foi recebido na totalidade.

**Validação**

\*falta dizer que testaram com curtos circuitos e parar a ligação\*

Nesta secção são abordados os diversos testes realizados. Foram utilizados no total três tipos de teste para o mesmo ficheiro “pinguim.gif” usado na apresentação do trabalho com um tamanho total de 10968 kilobytes.

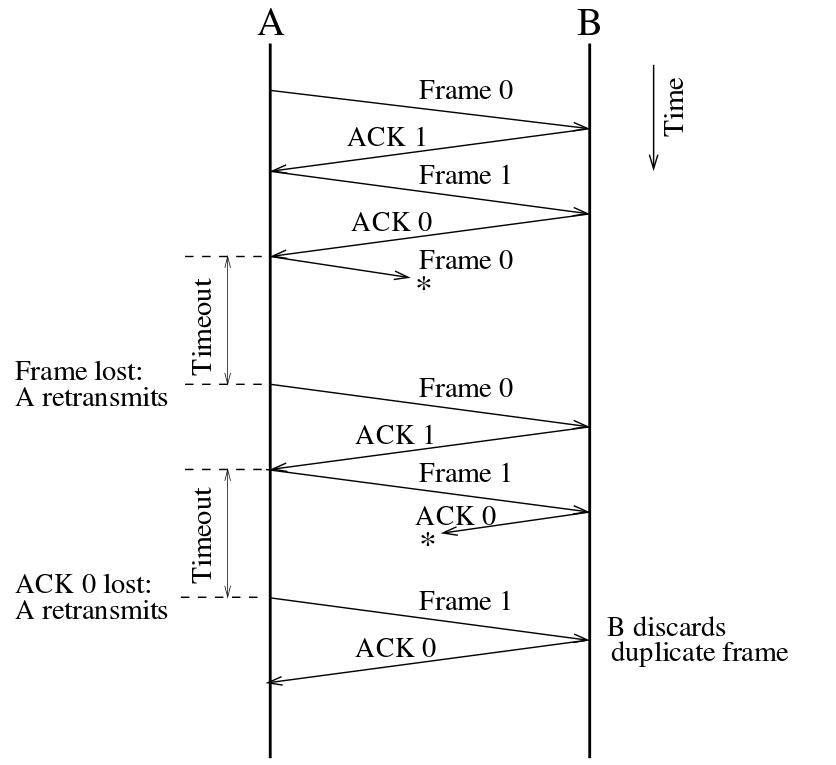
Os primeiros tipos de testes realizados tiveram em conta o aumento da percentagem de erros do BCC1 e do BCC2. Os resultados destes testes podem ser verificados [anexo](#tabela1).

O segundo tipo de testes efetuados teve em conta diferentes tamanhos dos pacotes enviados. Os resultados destes testes podem ser verificados no [anexo](#tabela2).

E por fim, o terceiro tipo de testes realizados foram aplicados a diferentes valores de BAUDRATE. Os resultados destes testes podem ser verificados no [anexo](#tabela3).

**Eficiência do Protocolo de Ligação de Dados**

No mecanismo *Stop and Wait* de ARQ (Automatic Repeat Request), o emissor após enviar uma trama I espera sempre por uma resposta do recetor, se o recetor confirmar a trama enviada o emissor procede e envia uma nova trama, caso contrário o emissor é obrigado a reenviar a trama I até que esta seja aceite pelo recetor. O recetor ao receber uma trama I verifica se deteta algum erro, caso haja erros envia ao emissor mensagem de rejeição NACK, caso contrário manda confirmação ACK ao emissor.

 De forma o programa reconhecer se a trama recebida é uma nova ou apenas uma retransmissão, as tramas I e ACK devem ser numeradas do estilo I(0), I(1) e ACK(0), ACK(1), em que ACK(i) indica ao emissor que o recetor está à espera de uma trama I(i).

Este mecanismo necessita do uso de um alarme com timeout de forma a resolver problemas como a perda das tramas I, ACK ou NACK.

O projeto realizado usa este mecanismo de ARQ, sempre que o emissor manda uma trama espera sempre uma resposta com utilização de um alarme de três segundos. Por exemplo, se o emissor enviar uma trama de informação (quer seja Start, End ou normal) espera sempre por um RR (com numeração 0 ou 1 dependendo da numeração da trama enviada) de resposta positiva ou REJ de resposta negativa, isto não se aplica somente as tramas de informação (tramas de supervisão de certa forma usam também este mecanismo) embora apenas estas necessitam necessariamente de serem numeradas.

Tendo em conta os testes efetuados, após os cálculos do débito recebido bits/s e da eficiência de cada teste podemos concluir o impacto que as variáveis de teste têm na eficiência da aplicação.

No gráfico 1 (em [anexo](#chart1)) podemos verificar que a percentagem de erros do BCC1 e do BCC2 têm um impacto significativo na eficiência da aplicação, por isso vemos um decréscimo de eficiência à medida que as percentagens de erros aumentam. Neste gráfico também podemos comparar e concluir que os erros do BCC1 causam maior impacto na eficiência, como podemos ver na passagem do ponto ‘0+0’ para o ponto ‘2+0’ em comparação com o ponto ‘0+2’.

No gráfico 2 (em [anexo](#chart2)) podemos concluir que quanto maior forem as tramas de informação mandadas maior a eficiência da aplicação pois menos tramas têm de ser mandadas. Existe um ponto em que o aumento de eficiência estagna, pois, as tramas passam a ter capacidade maior que o tamanho do ficheiro a mandar.

No gráfico 3 (em [anexo](#chart3))) verificamos que quanto maior for BAUDRATE, embora seja possível enviar um maior numero de bits pela porta series por segundo, a eficiência diminui.

**Conclusão**

Os objetivos propostos para este trabalho foram atingidos, tendo sido estabelecido o protocolo com sucesso.

Para tal, foi mantida a independência de camadas entre a parte da aplicação e a parte da ligação, permitindo assim que, alterando uma das camadas, a outra nada ou quase nada tenha que alterar. Deste modo, a camada de aplicação trata apenas da criação e interpretação dos dados de informação e respetivo cabeçalho do pacote de dados, enquanto que a camada de ligação apenas se preocupa com a coesão das tramas recebidas e envio das mesmas com os valores nos cabeçalhos necessários.

A realização deste trabalho permitiu a todos os membros do grupo um melhor entendimento sobre a independência de camadas em sistemas e estruturação e funcionamento de um protocolo de comunicação utilizando a porta de série.

**Anexo I**

\*Código tem que ser obrigatoriamente no anexo I\*

**Anexo II**

**Tabelas dos testes efetuados**

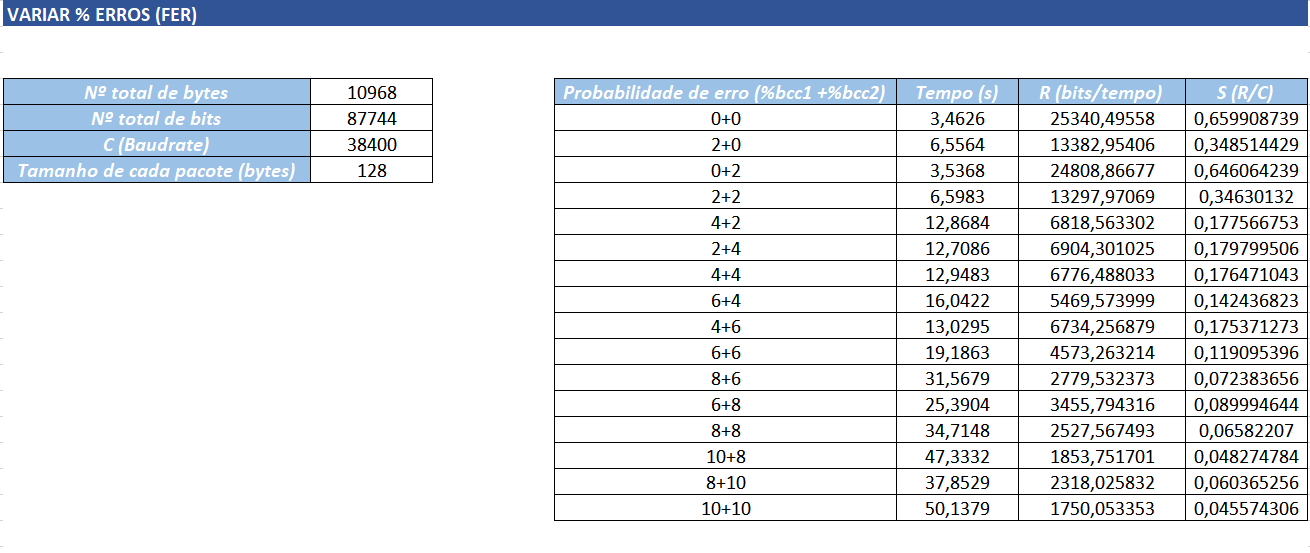


Tabela 1- Resultados dos testes de variação da % erros dos BCCs

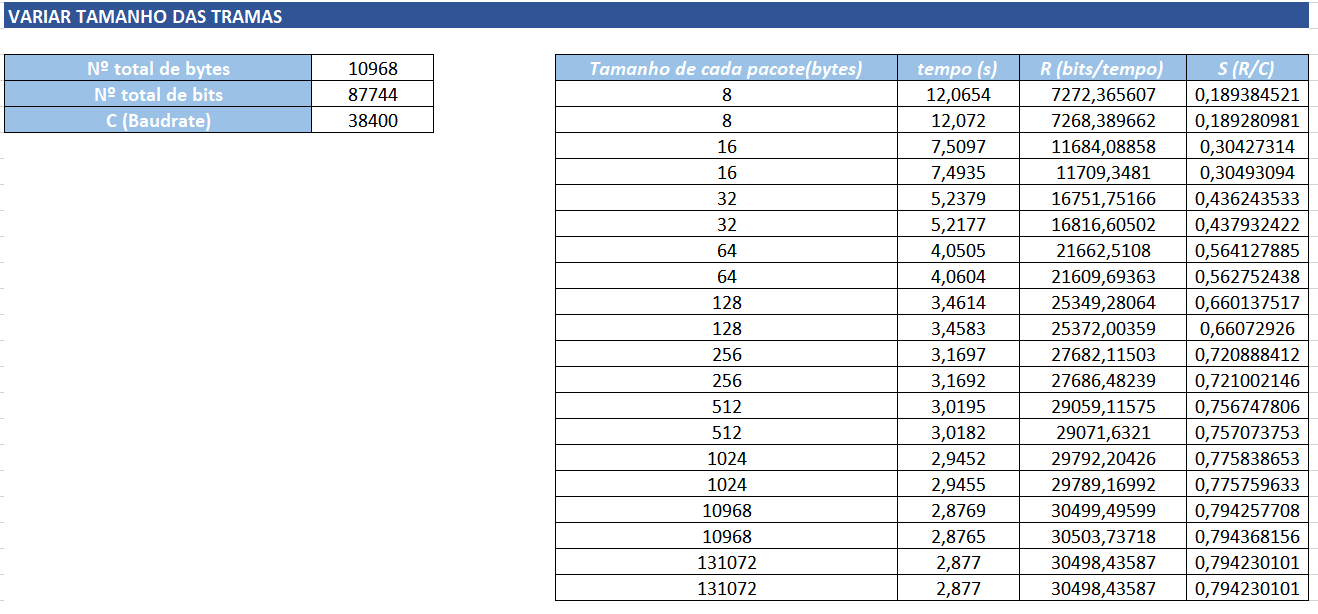


Tabela 2 - Resultados dos testes de diferentes tamanhos de tramas

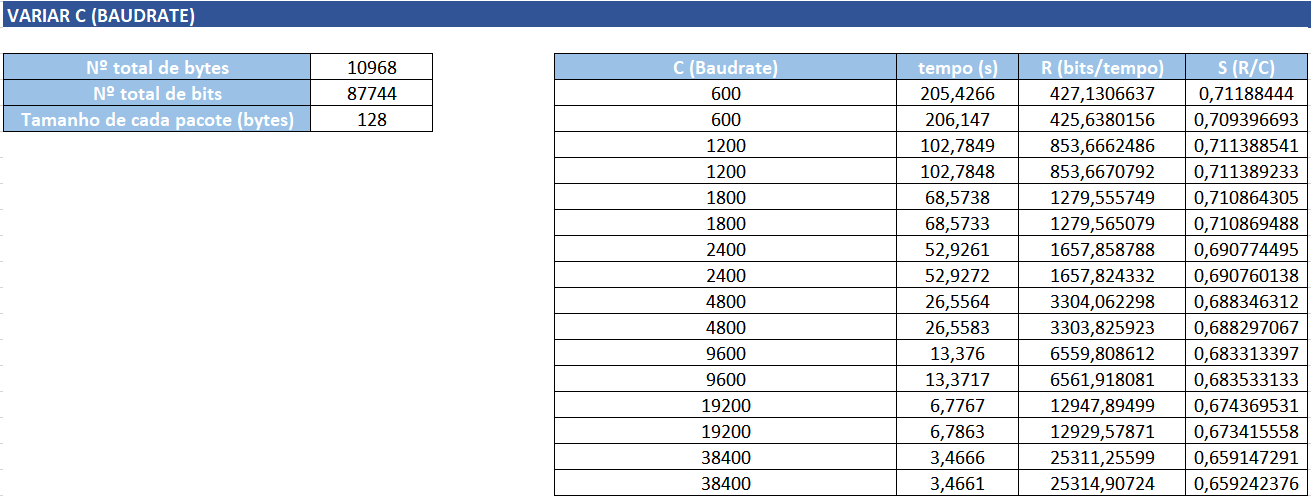


Tabela 3 - Resultados dos testes com diferentes valores de BAUDRATE

**Gráficos dos testes efetuados**

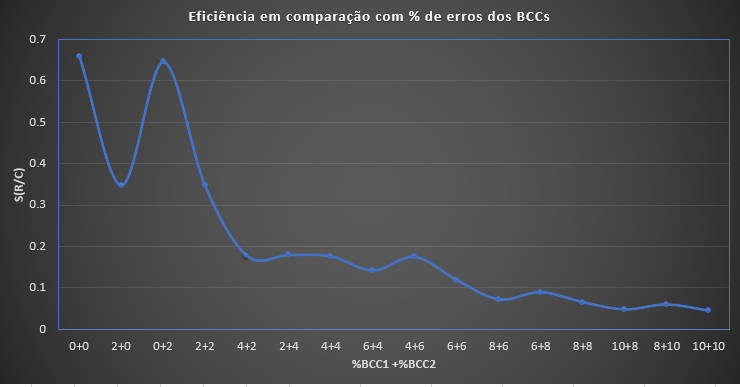


Figura 1- Gráfico eficiência, percentagem de erros obtido dos dados da [tabela 1](#tabela1).

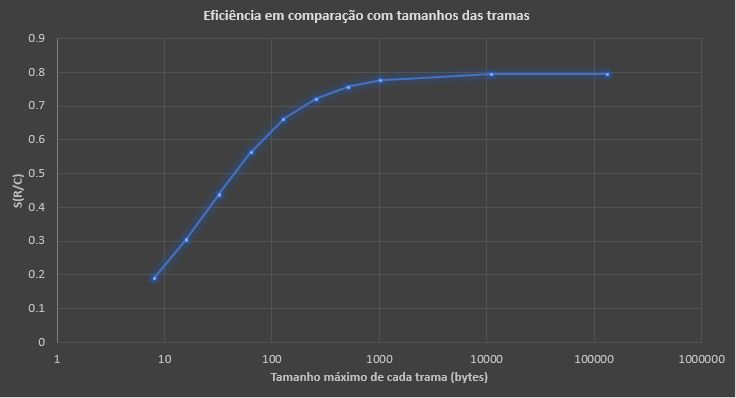


Figura 2- Gráfico de eficiência em relação ao tamanho das tramas obtido do dados da [tabela 2](#tabela2).

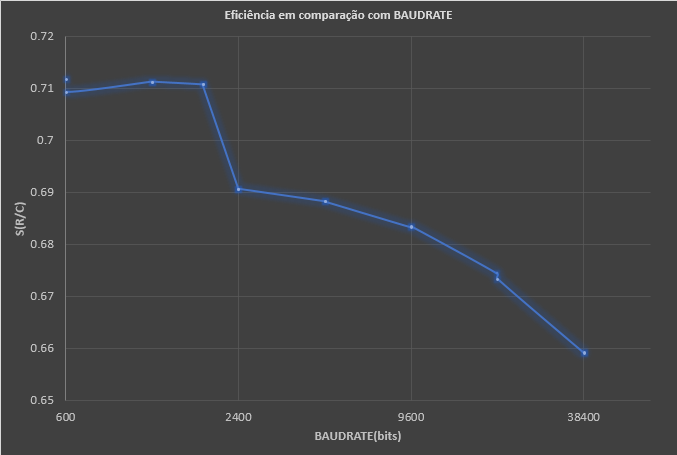


Figura 3 - Gráfico de eficiência em comparação com BAUDRATE retirado dos dados da [tabela 3](#tabela3)