U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

# Redes de Computadores
# 1º Trabalho Laboratorial: Ligação de dados

Filipe Campos - up201905609@up.pt

Vasco Alves - up201808031@up.pt

22 de dezembro de 2021

# Conteúdo

# 1 Sumário

Este trabalho foi realizado no âmbito da unidade curricular de Redes de Computadores do 3º ano da Licenciatura em Engenharia Informática e Computação da FEUP. O seu objetivo consiste no desenvolvimento de um programa capaz de transferir ficheiros de um computador para o outro, através da porta série.

Todos os objetivos definidos foram alcançados com sucesso, sendo criada uma aplicação que permite a transferência de ficheiros sem perda de informação.

# 2 Introdução

O objetivo deste trabalho é implementar um protocolo de ligação de dados e testá-lo através da transferência de ficheiros, tendo como recurso uma porta série. O relatório tem como objetivo esclarecer detalhes teóricos, arquiteturais e de implementação deste projeto.

# 3 Arquitetura

O nosso programa divide-se em dois blocos funcionais, o bloco de aplicação e o de ligação de dados. A aplicação interage com a camada de ligação de dados através de funções de interface `llopen`, `llwrite`, `llread`, `llclose` implementadas usando a camada de mais baixo nível.

# 4 Estrutura do código

O programa está dividido ao longo dos seguintes ficheiros:

- `application.c` - Camada de aplicação.
- `interface.c` - Interface utilizada para comunicar com a camada de ligação de dados.
- `common.h` - Ficheiro com estruturas, macros e constantes comuns a todos os ficheiros da camada de ligação de dados.
- `config.c` - Configuração da porta série e alarme.
- `core.c` - Funções essenciais relacionadas com a criação de frames e byte stuffing.

- `datalink_emitter.c` - Emissão de frames.
- `datalink_receiver.c` - Receção de frames.
- `logger.c` - Funções de registo de informação utilizadas pelos dois ficheiros anteriores.

Na Figura 1 [página 12] podemos ver a ordem de chamada de funções, destacando-se as funções `emitter/receiver` e `emit_frame/receive_frame` pertencentes à camada de aplicação e à camada de ligação de dados, respetivamente.

## 5 Casos de uso principais

Os principais casos de uso são a escolha e transferência de um ficheiro a enviar. Para efetuar o envio, o emissor deve especificar qual a porta série que pretende utilizar e o ficheiro a enviar, já o receptor pode apenas especificar a porta série, sendo no entanto possível especificar também o nome do ficheiro para o qual o conteúdo recebido será escrito, dando override ao nome original do ficheiro.

Utilização do programa:

```
./application emitter port_number input_filename
./application receiver port_number
./application receiver port_number output_filename
```

**Exemplo de execução**

**Emissor**

- ./application emitter 0 pinguim.gif
- Chama `llopen` para abrir e configurar /dev/ttyS0.
- Cria e envia control packet com nome do ficheiro e o seu tamanho.
- Lê e emite o conteúdo do ficheiro pinguim.gif através de vários pacotes de informação.
- Envia control packet final.

**Recetor**

- ./application receiver 0
- Chama `llopen` para abrir e configurar /dev/ttyS0.
- Recebe control packet, obtendo o nome do ficheiro.
- Recebe pacotes de informação e guarda-os no ficheiro pinguim.gif
- Recebe control packet com parâmetros idênticos aos iniciais.

- Chama `llclose` para finalizar a conexão.

- Chama `llclose` para finalizar a conexão.

# 6 Camada de Aplicação

## 6.1 Estrutura

As estruturas de dados utilizadas nesta camada representam respetivamente pacotes de dados e pacotes de controlo, e estão abaixo definidas.

```
typedef struct {
    uint8_t sequence_number;
    uint8_t L2;
    uint8_t L1;
    uint8_t data[MAX_PACKET_SIZE - 4];
} data_packet;

...

typedef struct {
    uint8_t control;
    control_parameter parameters[CONTROL_PACKET_PARAMETER_COUNT];
} control_packet;
```

Esta camada está subdividida em duas funções principais, `emitter` e `receiver`.

## 6.2 Emissor

O emissor lê uma string passada como argumento ao programa, que indica o nome do ficheiro a transmitir. Esse nome e o tamanho do ficheiro respetivo irão ser enviados para o recetor através de pacotes de controlo, marcando assim o início da transferência. De seguida serão enviados $\left\lceil \frac{file\_size}{MAX\_PACKET\_DATA\_SIZE} \right\rceil$ pacotes de informação, terminando com o envio de pacotes de controlo.

## 6.3 Recetor

O receptor pode ou não receber uma string que indica o nome do ficheiro. No caso de não receber, o ficheiro terá o seu nome original, caso contrário será dado overwrite ao nome do ficheiro. Este recebe o nome e o tamanho do ficheiro a partir do primeiro pacote de controlo, seguidamente escreve para o ficheiro de destino todos os dados dos pacotes de informação recebidos, terminando a recepção recebendo um último pacote de controlo.

# 7 Interface

## 7.1 Estrutura

A informação da interface série utilizada internamente é representada pela seguinte estrutura, que é criada internamente na chamada `llopen()` e eliminada na chamada `llclose()`

```
typedef enum{
    EMITTER,
    RECEIVER
} flag_t;

typedef struct {
    int fd; // File descriptor associated with the serial port
    int S; // Sequence number of the transmission
    flag_t status; // EMITTER | RECEIVER
    struct termios oldtio; // Old settings to be restored after closing.
    bool open; // States if the serial_interface is still in use or not
} serial_interface;
```

## 7.2 llopen

A função `llopen()` é responsável pela abertura da porta série e também pela configuração do alarme. Seguidamente, no caso do emissor, é enviado o comando SET, recorrendo à função `emit_frame_until_response`, onde a resposta esperada é UA. No lado do recetor será utilizada a função `receive_frame` para receber o frame SET e, de seguida, será enviado um frame com control byte UA. O recetor está também protegido por um timeout idêntico ao utilizado na função llread para evitar o bloqueio do programa no caso de um emissor não se conectar.

## 7.3 llwrite

A função `llwrite()` trata da transmissão do frame de informação, e espera pela trama de resposta RR, tentando realizar a retransmissão do frame um máximo de três vezes no caso de falha, quer seja devido a um timeout, ou devido a uma resposta incorreta como REJ.

## 7.4 llread

A função `llread()` lê um frame de informação e é responsável por responder de acordo com a sua configuração, detetando erros e bytes fora de sequência.

Respostas possíveis para diferentes situações:

- **Ocorreu erro**
  - É um frame novo: Envia REJ.
  - É um frame antigo: Envia RR.
- **Não ocorreu erro**
  - É um frame novo: Dados são retornados,
  - É um frame antigo: Dados são descartados.

Esta função também utiliza um mecanismo de timeout para permitir o recetor se desconectar se por algum motivo o emissor terminar a conexão, este timeout é implementado recorrendo a um alarme de ((número_máximo_retransmissões_do_emissor + 1) * timeout_do_emissor) segundos, isto garante que o recetor apenas dará timeout após o emissor ter esgotado todas as suas retransmissões.

### 7.5 llclose

A função `llclose()` é responsável pelo encerramento da comunicação, com o emissor a enviar o comando DISC e o receptor receber este comando e enviá-lo novamente, acabando com o emissor a enviar o comando UA e o receptor a recebê-lo. Para finalizar são restauradas as definições anteriores da porta série e fechado o seu file descriptor.

## 8 Protocolo de ligação de dados

### 8.1 Estrutura

Para representar internamente um frame, foi criada uma struct framecontent que representa o conteúdo de um frame, excluindo as duas flags que marcam o seu início/fim. O array data tem o tamanho `BUFFER_SIZE = MAX_INFO_SIZE*2`, para permitir realizar byte stuffing sem recorrer à alocação de memória extra mesmo no caso extremo em que todos os bytes de dados mais o bcc são uma flag e necessitam stuffing.

```
// Representation of a frame, after flags have been removed
typedef struct {
        uint8_t address;
        uint8_t control;
        uint8_t data[BUFFER_SIZE];
        size_t data_len;
} framecontent;
```

Para facilitar a criação de frames são utilizadas as funções `create_information_frame` e `create_non_information_frame` que permitem criar frames com / sem campo de dados, respetivamente.

### 8.2 Timeout

Para implementar a funcionalidade de timeout, tanto no emissor como no recetor, recorremos à função `alarm` que foi configurada para acionar um signal handler que altera o valor da variável ALARM_ACTIVATED para true.

### 8.3 Emissor

A função essencial desta secção denomina-se `emit_frame` e converte os conteúdos de um framecontent para um array de bytes e envia-os para o file descriptor fornecido. Para implementar o mecanismo de retransmissões foi desenvolvida a função `emit_frame_until_response` que recebe o framecontent a enviar e a resposta esperada (byte de controlo) e realiza uma primeira chamada à função `emit_frame`, se não for recebida uma resposta dentro de `FRAME_RESEND_TIMEOUT` segundos serão realizadas mais `MAX_EMIT_ATTEMPTS` retransmissões, espaçadas pelo tempo de timeout, se todas as retransmissões falharem será retornado um erro.

### 8.4 Recetor

A função principal do módulo receptor é a função `receive_frame`, que lê os conteúdos do file descriptor fornecido e verifica se estes são válidos. Esta função é responsável pela implementação da máquina de estados que lê os frames byte a byte para um framecontent, fazendo o byte destuffing e também a verificação do BCC.

## 9 Validação

Para validar o funcionamento do programa realizamos os seguintes testes:

- Enviar pinguim.gif, emissor iniciado após o recetor - Ficheiro enviado com sucesso
- Enviar pinguim.gif, emissor antes do recetor - A emissão começa após 3 segundos (tempo de timeout) e o ficheiro é enviado com sucesso

- Enviar ficheiro composto exclusivamente por bytes 0x7E para testar byte stuffing - Byte stuffing realizado sem problemas e ficheiro enviado com sucesso.
- Iniciar emissor sem recetor - Timeout após 3 segundos.
- Iniciar recetor sem emissor - Timeout após 12 segundos.
- Desconectar e voltar a ligar a porta série a meio da transferência - Transferência retoma o seu estado anterior e termina sem erros.
- Causar interferência nos bytes enviados - Transferência concluída com sucesso

# 10 Eficiência do protocolo de ligação de dados

De modo a avaliar a eficiência do protocolo implementado foram realizados 5 testes, para cada teste, os parâmetros não variáveis assumiram os seguintes valores:

$$baud = 38400$$

$$fer = 0$$

$$infosize = 512B$$

$$tprop = 0s$$

Adicionalmente, foi utilizado um timeout de 1 segundo no emissor e aceitamos no máximo 3 retransmissões.

## 10.1 Variar capacidade de ligação

O aumento do baudrate leva a um grande decréscimo da eficiência, como podemos ver na figura 2 [página 13], variando de 75.223%, para um baudrate de 9600, a 12.484% para um baudrate de 230400.

## 10.2 Variar frame error ratio

Para estes testes os erros no BCC1 e no BCC2 são equiprováveis, portanto metade dos erros ocorrem devido a problemas no header e a outra metade devido a problemas nos dados.

O aumento do frame error ratio leva a um decréscimo da eficiência, como podemos ver na figura 3 [página 14]

### 10.3 Variar tamanho de dados da trama de informação

Com o aumento do tamanho do campo de dados da trama de informação notamos um aumento na eficiência do protocolo, como representado na Figura 4 [página 14]. Entre 256B e 2048B ocorreu um aumento de 6.081%.

### 10.4 Variar tempo de propagação

Como expectado um aumento do tempo de propagação leva a um decréscimo da eficiência, figura 5 [página 15]

### 10.5 Variar a

Para calcular o valor de $a$ para cada resultado, primeiro calculamos o valor

$$T_f = \frac{infosize \cdot 8}{C}$$

Tendo esse valor podemos obter o valor de $a$ experimental:

$$a = \frac{tprop}{T_f}$$

O gráfico de S em função de a para diferentes valores de frame error ratio pode ser visualizado na Figura 6 [página 15].

Como previsto o valor de S sofre um decréscimo exponencial com o aumento de $a$, isto coincide com os valores teóricos esperados, cujo gráfico se encontra na Figura 7 [página 16].

Este segundo gráfico foi obtido recorrendo à formula:

$$S = \frac{1 - fer}{1 + 2a}$$

## 11 Conclusões

Com este trabalho conseguimos implementar um protocolo de ligação de dados e utilizar-lo numa aplicação exemplo, permitindo a transferência de ficheiros entre dois computadores via uma porta série. Isto permitiu-nos expandir e reforçar os nossos conhecimentos do protocolo *Stop & Wait* tanto do ponto de vista teórico como de uma perspectiva prática sendo portanto um ponto focal da nossa aprendizagem de Redes de Computadores.

# 12 Anexo I - Grafo de chamada a funções

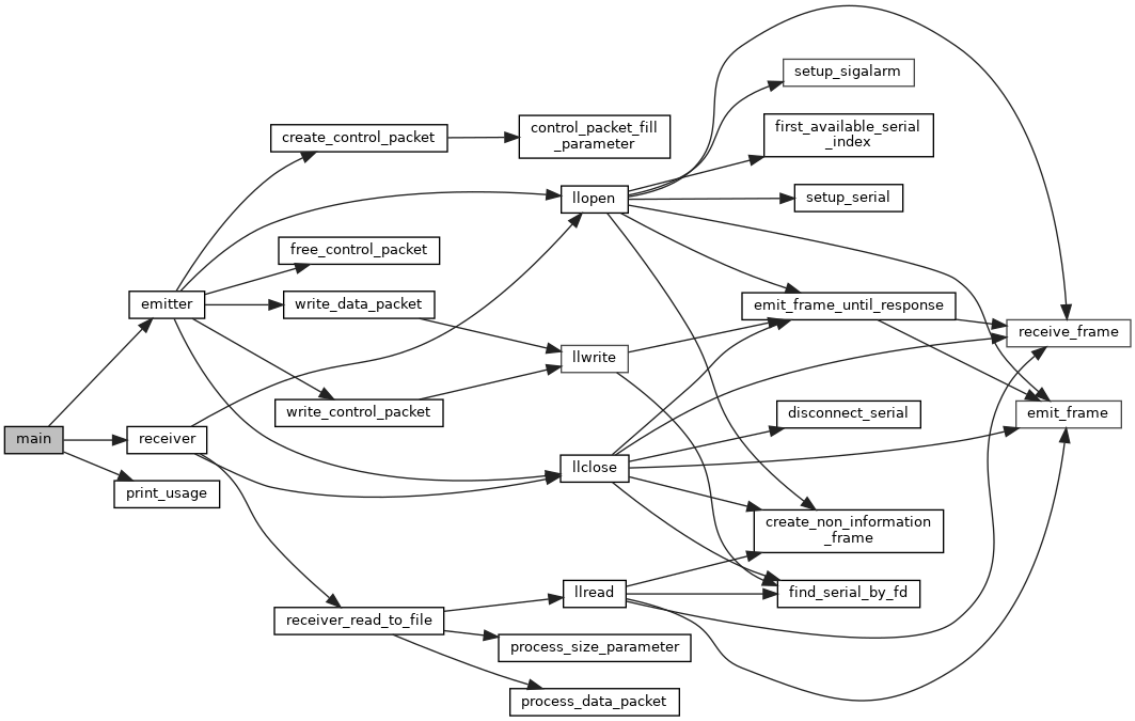


Figura 1: Grafo de chamadas de funções

# 13 Anexo II - Gráficos de Análise de Eficiência

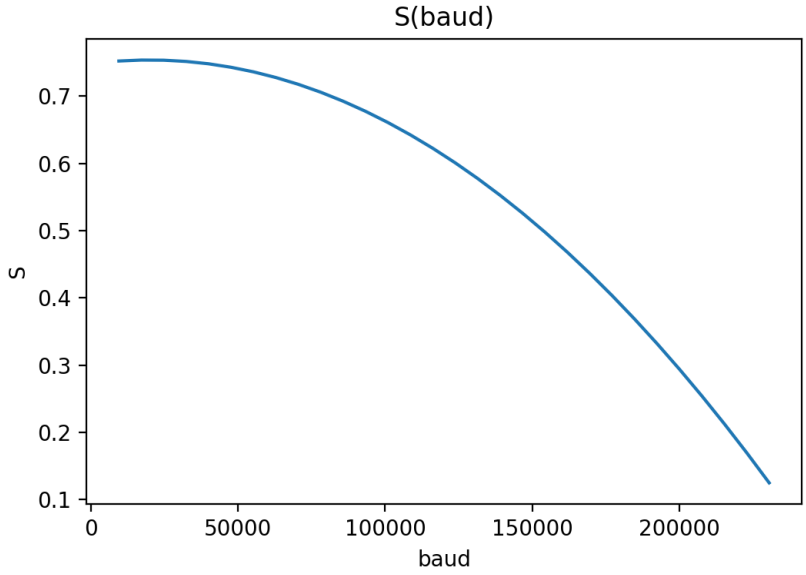


Figura 2: S em função da capacidade da ligação

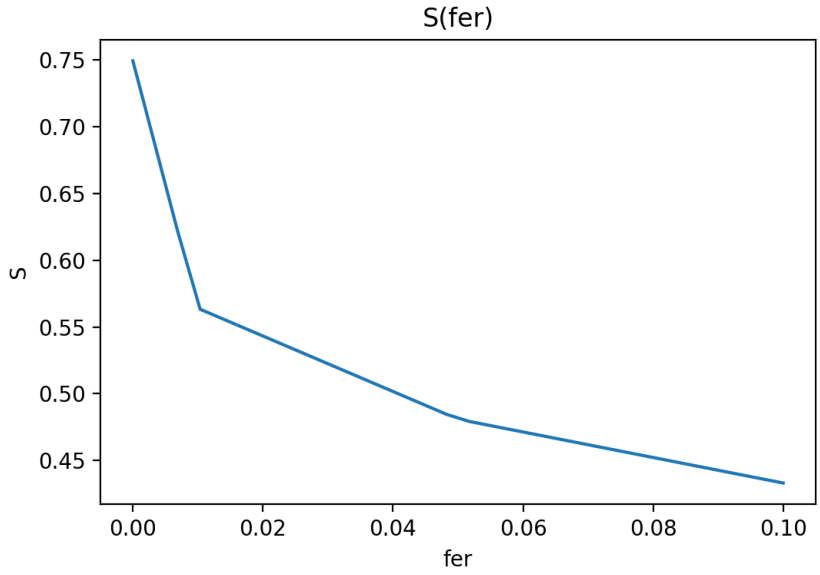


Figura 3: S em função de frame error ratio

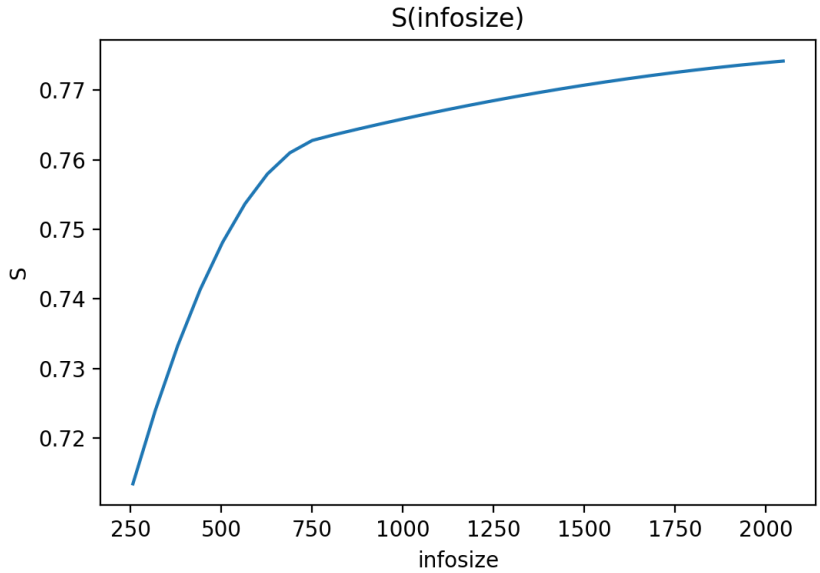


Figura 4: S em função do tamanho de dados da trama de informação

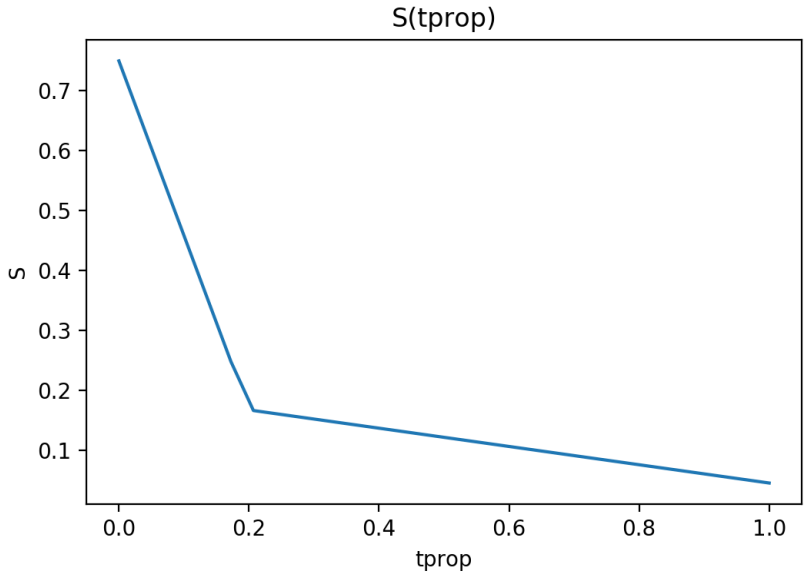


Figura 5: S em função do tempo de propagação

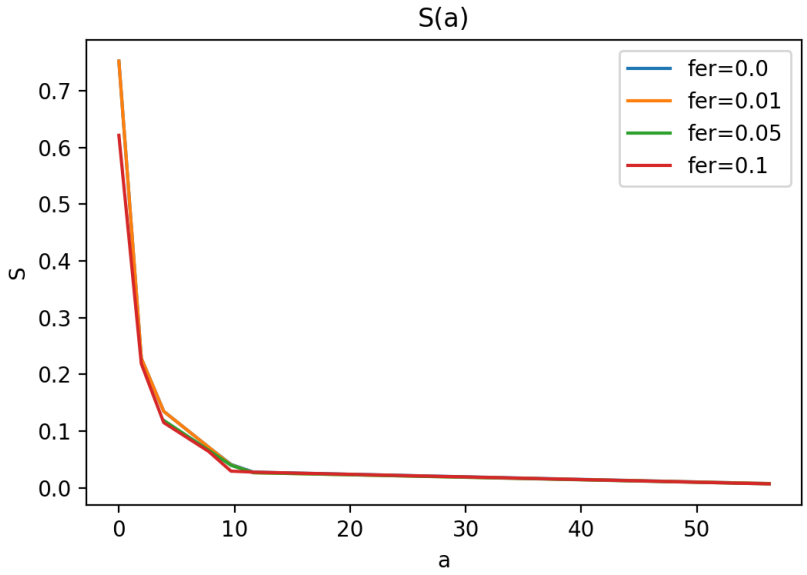


Figura 6: Valor obtido para S em função de a

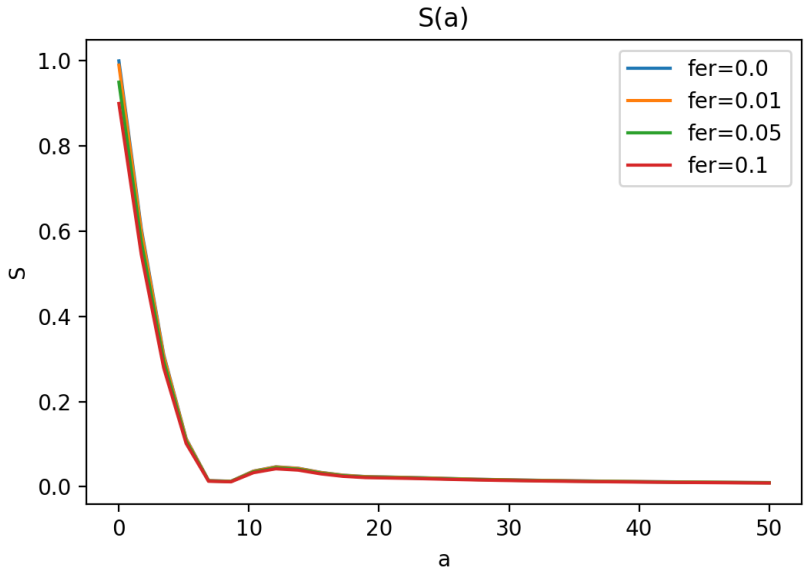


Figura 7: Valor teórico para S em função de a

# 14 Anexo III - Código Fonte

## 14.1 Camada de Aplicação

### 14.1.1 application.h

```
#ifndef __APPLICATION__
#define __APPLICATION__

#include <stdint.h>
#include <stddef.h>
#include "interface.h"

#define MAX_PACKET_SIZE (MAX_WRITE_SIZE)
#define MAX_PACKET_DATA_SIZE MAX_PACKET_SIZE - 4 /*Section of
↪  MAX_PACKET_SIZE that's available for data*/
#define CONTROL_PACKET_PARAMETER_COUNT 2 /* Number of parameters
↪  sent in control packet */

/* Control bytes sent in control packet */
#define CTL_BYTE_DATA 1
#define CTL_BYTE_START 2
#define CTL_BYTE_END 3


typedef struct {
    uint8_t sequence_number;
    uint8_t L2;
    uint8_t L1;
    uint8_t data[MAX_PACKET_SIZE - 4];
} data_packet;

typedef enum {
    SIZE,
    NAME
} parameter_type; /* Control packet parameters type */

/* TLV Control packet parameter */
typedef struct {
    parameter_type type;
    uint8_t length;
    uint8_t *value;
} control_parameter;

typedef struct {
```

```
    uint8_t control;
    control_parameter parameters[CONTROL_PACKET_PARAMETER_COUNT];
} control_packet;

/**
 * @brief Print program usage
 *
 * @param name - program name
 */
void print_usage(char *name);

/**
 * @brief Run receiver
 *
 * @param argc
 * @param argv
 * @param port_number serial port number [0,999]
 * @return int status
 */
int receiver(int argc, char *argv[], int port_number);

/**
 * @brief Read packets from input_fd and write them onto output_fd
 *
 * @param input_fd
 * @param output_fd
 * @param argc
 * @return int
 */
int receiver_read_to_file(int input_fd, int output_fd, int argc);

/**
 * @brief Process data packets, writting them into output_fd
 *
 * @param buffer
 * @param sequence
 * @param output_fd
 * @return int
 */
int process_data_packet(uint8_t *buffer, uint8_t *sequence, int
↪  output_fd);

/**
 * @brief Process the SIZE parameter sent in control packet
 *
```

```
 * @param value
 * @return uint64_t total_packets necessary to send the complete
↪  file
 */
uint64_t process_size_parameter(uint8_t *value);

/**
 * @brief Write a control packet using llwrite
 *
 * @param fd
 * @param packet
 * @return int
 */
int write_control_packet(int fd, control_packet *packet);

/**
 * @brief Write a data packet using llwrite
 *
 * @param fd
 * @param packet
 * @return int status
 */
int write_data_packet(int fd, data_packet *packet);

/**
 * @brief Fill a parameter in control packet
 *
 * @param packet target control packet
 * @param parameter_index index of chosen parameter
 * @param type
 * @param length
 * @param value
 * @return int status
 */
int control_packet_fill_parameter(control_packet *packet, size_t
↪  parameter_index, parameter_type type, size_t length, uint8_t
↪  *value);

/**
 * @brief Run emitter
 *
 * @param argc
 * @param argv
 * @param port_number serial port number [0,999]
 * @return int
```

```
 */
int emitter(int argc, char *argv[], int port_number);

/**
 * @brief Create a complete control packet
 *
 * @param ctl_packet
 * @param fd
 * @param filename
 * @return uint64_t
 */
uint64_t create_control_packet(control_packet *ctl_packet, int fd,
↪  char *filename);

/**
 * @brief Free memory used by a control packet
 *
 * @param packet
 * @return int
 */
int free_control_packet(control_packet *packet);

#endif
```

### 14.1.2 application.c

```c
#include "application.h"
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>

int write_control_packet(int fd, control_packet *packet){
    uint8_t buffer[MAX_PACKET_SIZE];
    buffer[0] = packet->control;

    int buf_pos = 1;
    for(int i = 0; i < CONTROL_PACKET_PARAMETER_COUNT; ++i){
        control_parameter parameter = (packet->parameters)[i];
        if(buf_pos + parameter.length > MAX_PACKET_SIZE){
            return 1;
        }
        buffer[buf_pos++] = parameter.type;
        buffer[buf_pos++] = parameter.length;
        memcpy(buffer+buf_pos, parameter.value, parameter.length);
        buf_pos += parameter.length;
    }
    llwrite(fd, buffer, buf_pos);
    return 0;
}

int write_data_packet(int fd, data_packet *packet){
    uint8_t buffer[MAX_PACKET_SIZE];
    buffer[0] = CTL_BYTE_DATA;
    buffer[1] = packet->sequence_number;
    buffer[2] = packet->L2;
    buffer[3] = packet->L1;
    size_t len = (packet->L2)*256 + (packet->L1);
    memcpy(buffer + 4, packet->data, len);
    return llwrite(fd, buffer, len + 4) >= 0 ? 0 : -1;
}
```

```
int control_packet_fill_parameter(control_packet *packet, size_t
↪  parameter_index, parameter_type type, size_t length, uint8_t
↪  *value){
    if(parameter_index > CONTROL_PACKET_PARAMETER_COUNT){
        return -1;
    }
    packet->parameters[parameter_index].type = type;
    packet->parameters[parameter_index].length = length;
    packet->parameters[parameter_index].value = malloc((sizeof
    ↪  (uint8_t)) * length);
    memcpy(packet->parameters[parameter_index].value, value,
    ↪  length);
    return 0;
}

int free_control_packet(control_packet *packet){
    for(int i = 0; i < CONTROL_PACKET_PARAMETER_COUNT; ++i){
        free(packet->parameters[i].value);
    }
    return 0;
}

uint64_t create_control_packet(control_packet *ctl_packet, int fd,
↪  char *filename){
    ctl_packet->control = CTL_BYTE_START;

    struct stat statbuf;
    if(fstat(fd, &statbuf) < 0){
        return 0;
    }
    uint64_t size_value = statbuf.st_size;
    if(size_value == 0){
        perror("empty file\n");
        return 0;
    }
    uint64_t total_packets = size_value / ((uint64_t)
    ↪  MAX_PACKET_DATA_SIZE);
    total_packets += (size_value % ((uint64_t)
    ↪  MAX_PACKET_DATA_SIZE)) == 0 ? 0 : 1;
    if(control_packet_fill_parameter(ctl_packet, 0, SIZE, 8,
    ↪  (uint8_t *) &size_value) < 0){
        return 0;
    }
    if(control_packet_fill_parameter(ctl_packet, 1, NAME,
    ↪  strlen(filename)+1, filename) < 0){
```

```
        return 0;
    }
    return total_packets;
}

int emitter(int argc, char *argv[], int port_number){
    int output_fd = llopen(port_number, EMITTER);
    if (output_fd < 0){
        return 1;
    }
    int input_fd = open(argv[3], O_RDONLY);
    if(input_fd < 0){
        return 1;
    }

    control_packet ctl_packet;
    uint64_t total_packets = 0;
    if((total_packets = create_control_packet(&ctl_packet,
    ↪  input_fd, argv[3])) <= 0){
        return 1;
    }
    write_control_packet(output_fd, &ctl_packet);
    printf("Sent START packet\n");

    int read_res = 0;
    uint8_t sequence = 0;
    uint64_t current_num_packets = 0;

    data_packet dt_packet;
    while((read_res = read(input_fd, &(dt_packet.data),
    ↪  MAX_PACKET_DATA_SIZE)) > 0){
                dt_packet.sequence_number = sequence++;
        current_num_packets++;
        dt_packet.L2 = read_res / 256;
        dt_packet.L1 = read_res % 256;
        if(write_data_packet(output_fd, &dt_packet) < 0){
            printf("Failed to send DATA packet [%ld/%ld]\n",
            ↪  current_num_packets, total_packets);
            return 1;
        }
        printf("Sent DATA packet [%ld/%ld]\n", current_num_packets,
        ↪  total_packets);
        }

    ctl_packet.control = CTL_BYTE_END;
```

```
    write_control_packet(output_fd, &ctl_packet);
    printf("Sent END packet\n");
    free_control_packet(&ctl_packet);
    llclose(output_fd);
    close(input_fd);
    return 0;
}

int process_data_packet(uint8_t *buffer, uint8_t *sequence, int
↪  output_fd){
    uint8_t packet_sequence_number = buffer[0];
    if((*sequence) != packet_sequence_number){
        return -1;
    }
    (*sequence)++;
    uint8_t L2 = buffer[1];
    uint8_t L1 = buffer[2];

    if(write(output_fd, buffer+3, (L2*256)+L1) < 0){
        return 1;
    }
    return 0;
}

uint64_t process_size_parameter(uint8_t *value){
    uint64_t size_value;
    memcpy(&size_value, value, 8);
    printf("  parameter T=%d L=%d V=%ld\n", SIZE, 8, size_value);
    uint64_t total_packets = size_value / ((uint64_t)
    ↪  MAX_PACKET_DATA_SIZE);
    total_packets += (size_value % ((uint64_t)
    ↪  MAX_PACKET_DATA_SIZE)) == 0 ? 0 : 1;
    return total_packets;
}

int receiver_read_to_file(int input_fd, int output_fd, int argc){
        uint8_t buffer[MAX_PACKET_SIZE];
    int read_res = 0;

    uint8_t sequence = 0;
    uint64_t current_num_packets = 1;
    uint64_t total_packets = 0;

    bool started = false;
    while((read_res = llread(input_fd, buffer)) >= 0){
```

```
        if(read_res == 0){
            continue;
        }
        int buf_pos = 0;
        uint8_t control_byte = buffer[buf_pos++];

        if(control_byte == CTL_BYTE_START || control_byte ==
        ↪  CTL_BYTE_END){
            if(!started){
                printf("Received START packet\n");
            } else {
                printf("Received END packet\n");
            }

            for(int i = 0; i < CONTROL_PACKET_PARAMETER_COUNT;
            ↪  ++i){
                uint8_t type = buffer[buf_pos++];
                uint8_t len = buffer[buf_pos++];
                uint8_t value[len];
                memcpy(value, buffer + buf_pos, len);
                if(type == NAME && argc == 3){
                    output_fd = open(value, O_WRONLY | O_CREAT,
                    ↪  S_IRUSR | S_IWUSR);
                    if(output_fd < 0){
                        return 1;
                    }
                }
                if(len == 8 && type == SIZE){
                    total_packets = process_size_parameter(value);
                } else {
                    printf("  parameter T=%d L=%d V=%s\n", type,
                    ↪  len, value);
                }
                buf_pos += len;
            }
            if(started){ // END packet received
                break;
            }
            started = true;
        } else if (control_byte == CTL_BYTE_DATA){
            process_data_packet(buffer+buf_pos, &sequence,
            ↪  output_fd);
            printf("Received DATA packet [%ld/%ld]\n",
            ↪  current_num_packets, total_packets);
            current_num_packets++;
```

```
        } else {
            printf("error\n");
            return 1;
        }
    }
    return read_res >= 0 ? 0 : -1;
}

int receiver(int argc, char *argv[], int port_number){
    int input_fd = llopen(port_number, RECEIVER);
    if(input_fd < 0){
        return 1;
    }

    int output_fd;
    if(argc == 4){
        output_fd = open(argv[3], O_WRONLY | O_CREAT, S_IRUSR |
        ↪  S_IWUSR);
        if(output_fd < 0){
            return 1;
        }
    }

    if(receiver_read_to_file(input_fd, output_fd, argc) < 0){
        close(output_fd);
        return 1; // Receiver had an error caused by serial port
        ↪  error, llclose will also lead to an error.
    }

    llclose(input_fd);
    close(output_fd);
    return 0;
}

void print_usage(char *name){
    printf("Usage:\n");
    printf("    %s emitter port_number input_filename\n", name);
    printf("    %s receiver port_number\n", name);
    printf("    %s receiver port_number output_filename\n", name);
}

int main(int argc, char *argv[]){
    if(argc < 3){
        print_usage(argv[0]);
        return 1;
```

```
    }

    int port_number;
    if(sscanf(argv[2], "%d", &port_number) != 1){
        printf("Invalid port number\n");
        return 1;
    }

    srand(time(NULL)); // Used for FER analysis.
    if(strcmp(argv[1],"emitter") == 0){
        if(argc != 4){
            print_usage(argv[0]);
            return 1;
        }
        return emitter(argc, argv, port_number);
    }
    if(strcmp(argv[1], "receiver") == 0){
        if(argc > 4){
            print_usage(argv[0]);
        }
        return receiver(argc, argv, port_number);
    }
    printf("Invalid arguments");
    print_usage(argv[0]);
    return 1;
}
```

## 14.2 Interface

### 14.2.1 interface.h

```
#ifndef __INTERFACE__
#define __INTERFACE__

#include <stdbool.h>
#include <termios.h>
#include "common.h"

#define MAX_WRITE_SIZE (MAX_INFO_SIZE)
#define MAX_OPEN_SERIAL_PORTS 8 // Maximum allowed serial ports
↪  open at any given time

typedef enum{
    EMITTER,
    RECEIVER
} flag_t;

typedef struct {
    int fd; // File descriptor associated with the serial port
    int S; // Sequence number of the transmission
    flag_t status; // EMITTER | Receiver
    struct termios oldtio; // Old settings to be restored after
    ↪  closing.
    bool open; // States if the serial_interface is still in use or
    ↪  not
} serial_interface;

/**
 * @brief Connect serial port with respective flag
 *
 * @param port Serial port to be started
 * @param flag Indicator if it is emitter or receiver
 * @return int -1 if failure; file descriptor if success
 */
int llopen(int port, flag_t flag);

/**
 * @brief Create frame and send it
 *
 * @param fd File descriptor
 * @param buffer Buffer with packet
 * @param length Length of packet
```

```
 * @return int -1 if failure, frame size if success
 */
int llwrite(int fd, uint8_t * buffer, int length);

/**
 * @brief Receive frame and read it
 *
 * @param fd File descriptor
 * @param buffer Buffer with packet
 * @return int -1 if failure, packet length if success
 */
int llread(int fd, uint8_t * buffer);

/**
 * @brief Finish transmission and disconnect serial port
 *
 * @param fd File descriptor
 * @return int -1 if failure, 0 if success
 */
int llclose(int fd);

#endif
```

### 14.2.2 interface.c

```
#include "config.h"
#include "datalink_emitter.h"
#include "datalink_receiver.h"
#include "core.h"

#include "interface.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

static int used_serial_ports = 0;
static serial_interface serial_ports[MAX_OPEN_SERIAL_PORTS];

int first_available_serial_index(){
    if(used_serial_ports < MAX_OPEN_SERIAL_PORTS){
        serial_ports[used_serial_ports].open = true;
        return used_serial_ports++;
    }
```

```
    for(int i = 0; i < used_serial_ports; ++i){
        if(serial_ports[i].open == false){ // Reuse serial ports
        ↪  that are not in use
            serial_ports[i].open = true;
            return i;
        }
    }
    return -1;
}

int find_serial_by_fd(int fd){
    for(int i = 0; i < used_serial_ports; ++i){
        if(serial_ports[i].fd == fd){
            return i;
        }
    }
    return -1;
}

int llopen(int port, flag_t flag) {
    if(port < 0 || port > 999){
        return -1;
    }
    int serial_index = first_available_serial_index();
    if(serial_index < 0){ // Too many serial ports already open
        return -1;
    }
    serial_interface *serial = &(serial_ports[serial_index]);
    serial->status = flag;
    setup_sigalarm();

    char device_name[10 + 3];
    snprintf(device_name, 10 + 3, "/dev/ttyS%d", port);
        serial->fd = setup_serial(&(serial->oldtio), device_name);

    if (flag == EMITTER) {
        serial->S = 0;
        framecontent fc = create_non_information_frame(CTL_SET,
        ↪  ADDRESS1);
        if(emit_frame_until_response(serial->fd, &fc, CTL_UA) !=
        ↪  0){
            printf("Maximum emit attempts reached\n");
            return -1;
            }
    }
```

```
    else if (flag == RECEIVER) {
        serial->S = 1;
        bool success = false;
        alarm(READ_TIMEOUT);
        for(int i = 0; i < MAX_EMIT_ATTEMPTS; ++i){
            framecontent received_fc = receive_frame(serial->fd);
            if(received_fc.control == CTL_SET){
                success = true;
                break;
            }
            if(ALARM_ACTIVATED){
                ALARM_ACTIVATED = false;
                return -1;
            }
        }
        alarm(0);
        ALARM_ACTIVATED = false;
        if(success){
            framecontent fc = create_non_information_frame(CTL_UA,
            ↪  ADDRESS1);
                emit_frame(serial->fd, &fc);
        } else {
            return -1;
        }
    } else {
        perror("Invalid flag");
        return -1;
    }
    return serial->fd;
}

int llwrite(int fd, uint8_t * input_buffer, int length) {
    int serial_index = find_serial_by_fd(fd);
    if(serial_index < 0){
        return -1;
    }
    serial_interface *serial = &(serial_ports[serial_index]);

    if(serial->status != EMITTER || length > MAX_INFO_SIZE){
        return -1;
    }
    framecontent fc = create_information_frame(input_buffer,
    ↪  length, serial->S,  ADDRESS1);
    if(fc.data_len == 0){ // Datalen > MAX_INFO_SIZE.
        return -1;
```

```
    }
    if(emit_frame_until_response(fd, &fc,
    ↪  CREATE_RR_FRAME_CTL_BYTE(serial->S)) != 0){
        printf("Maximum emit attempts reached\n");
        return -1;
    }
    serial->S = 1 - (serial->S);
    return INFO_FRAME_SIZE_WITHOUT_DATA + fc.data_len;
}

int llread(int fd, uint8_t * output_buffer) {
    int serial_index = find_serial_by_fd(fd);
    if(serial_index < 0){
        return -1;
    }
    serial_interface *serial = &(serial_ports[serial_index]);

    if (serial->status != RECEIVER) {
        return -1;
    }

    framecontent received_fc;
    bool received = false;
    alarm(READ_TIMEOUT);
    while(!received){
        received_fc = receive_frame(fd);
        if(IS_INFO_CONTROL_BYTE(received_fc.control)){
            uint8_t control = 0;

            bool is_new_frame = serial->S !=
            ↪  GET_INFO_FRAME_CTL_BIT(received_fc.control);
            if(received_fc.data_len > 0){
                if(is_new_frame){
                    serial->S = 1 - (serial->S);
                } else {
                    received_fc.data_len = 0;
                }
                control = CREATE_RR_FRAME_CTL_BYTE(serial->S);
                received = true;
            } else {
                printf("An error has occurred, responding with: ");
                if(is_new_frame){
                    printf("REJ\n");
                    control =
                    ↪  CREATE_REJ_FRAME_CTL_BYTE(1-(serial->S));
```

```
                } else {
                    printf("RR\n");
                    control = CREATE_RR_FRAME_CTL_BYTE(serial->S);
                    received_fc.data_len = 0;
                    received = true;
                }
            }
            framecontent response_fc =
            ↪  create_non_information_frame(control, ADDRESS1);
            emit_frame(fd, &response_fc);
        }
        if(ALARM_ACTIVATED){
            ALARM_ACTIVATED = false;
            return -1;
        }
        // If the frame doesn't have an control_byte an error might
        ↪  have occurred
        // The receiver should try to read again
    }
    alarm(0);
    ALARM_ACTIVATED = false;
    memcpy(output_buffer, received_fc.data, received_fc.data_len);
    return received_fc.data_len;
}

int llclose(int fd){
    int serial_index = find_serial_by_fd(fd);
    if(serial_index < 0){
        return -1;
    }
    serial_interface *serial = &(serial_ports[serial_index]);
    serial->open = false;

    if(serial->status == EMITTER){
        framecontent fc = create_non_information_frame(CTL_DISC,
        ↪  ADDRESS1);
        emit_frame_until_response(fd, &fc, CTL_DISC);
        fc = create_non_information_frame(CTL_UA, ADDRESS2);
        emit_frame(fd, &fc);
    } else if (serial->status == RECEIVER){
        alarm(READ_TIMEOUT);
        framecontent received_fc = receive_frame(fd);
        if(received_fc.control == CTL_DISC){
            framecontent fc =
            ↪  create_non_information_frame(CTL_DISC, ADDRESS2);
```

```
            emit_frame_until_response(fd, &fc, CTL_UA);
        } else {
            alarm(0);
            ALARM_ACTIVATED = false;
            return -1;
        }
        alarm(0);
        ALARM_ACTIVATED = false;
    } else {
        return -1;
    }
        disconnect_serial(fd, &(serial->oldtio));
    return 0;
}
```

## 14.3 Camada de Ligação de Dados

### 14.3.1 common.h

```
#ifndef __COMMON__
#define __COMMON__

#include <stdint.h>
#include <stddef.h>

#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define VERBOSE false /* Controls if logger functions are enabled
↪  or disabled */

/* For efficiency analysis purposes only, should be set to 0 for
↪  real use */
#ifndef FER
#define FER 0 // In percentage. Used to test FER,should be set to 0
↪  for 'real' use.
#endif
#define FER_HEADER (FER/2) // Chance of the error occuring in the
↪  header
#define FER_DATA (FER - FER_HEADER) // Chance of the error occuring
↪  in the data
// FER = FER_HEADER + FER_DATA
#ifndef T_PROP
#define T_PROP 0 // In microseconds
#endif

#ifndef MAX_INFO_SIZE
#define MAX_INFO_SIZE 512
#endif
#define BUFFER_SIZE (MAX_INFO_SIZE*2 + 2) /*Necessary buffer size
↪  to accommodate INFO_FRAME_SIZE,
                                                    in the edge
↪  case where all data is flags (double the size)
                                                    + 2 bytes
↪  for the BCC (if it also needs to be escaped) */

#define FRAME_RESEND_TIMEOUT 3 /* Timeout between frame resends */
#define MAX_EMIT_ATTEMPTS 3 /* Maximum amounts of frame resends
↪  before giving up*/
#define READ_TIMEOUT (FRAME_RESEND_TIMEOUT * (MAX_EMIT_ATTEMPTS+1))

#define FLAG 0x7e /*Frame flag byte*/
```

```
#define ESCAPE 0x7d /*Frame escape byte*/
#define ADDRESS1 0x03 /*Frame address byte*/
#define ADDRESS2 0x01 /*Frame address byte*/

// Control Commands
#define CTL_SET 0x03
#define CTL_UA 0x07
#define CTL_DISC 0X0B
#define CTL_RR 0x05
#define CTL_REJ 0x01
#define CTL_INVALID_FRAME 0xFF

// Representation of a frame, after flags have been removed
typedef struct {
        uint8_t address;
        uint8_t control;
        uint8_t data[BUFFER_SIZE];
        size_t data_len;
} framecontent;

#define CREATE_INFO_FRAME_CTL_BYTE(S) (S << 6) /* Create INFO
↪  Control byte with the chosen S value */
#define GET_INFO_FRAME_CTL_BIT(b) (b >> 6) /*Get the sequence bit
↪  from a INFO control byte*/
#define IS_INFO_CONTROL_BYTE(b) ((b & 0xBF) == 0) /*Check if a
↪  control byte is an INFO control byte*/
#define INFO_FRAME_SIZE_WITHOUT_DATA 5 /*Size of a information
↪  frame without data (FLAG,CTL,ADDR,BYTE,FLAG)*/

#define CREATE_RR_FRAME_CTL_BYTE(R) (((R) << 7) | CTL_RR) /* Create
↪  RR Control byte with chosen R value */
#define CREATE_REJ_FRAME_CTL_BYTE(R) (((R) << 7) | CTL_REJ) /*
↪  Create REJ Control byte with chosen R value */
#define GET_RESPONSE_FRAME_CTL_BIT(b) (b >> 7) /* Get the R bit
↪  from a RR/REJ Control byte */
#define RESPONSE_CTL_MASK 0b00000111 /*Mask that removes the R bit
↪  from RR/REJ*/
#define APPLY_RESPONSE_CTL_MASK(b) (b & RESPONSE_CTL_MASK) /*Short
↪  function to apply the previous mask*/


#endif
```

### 14.3.2 config.h

```
#ifndef __CONFIG__
#define __CONFIG__

#define BAUDRATE B38400
#define DEFAULT_VTIME 5
#define DEFAULT_VMIN 5

#include <termios.h>
#include <stdbool.h>

extern volatile bool ALARM_ACTIVATED;

/**
 * @brief Set the up serial port
 *
 * @param oldtio Old serial port configuration
 * @param serial_device Serial port device
 * @return int -1 if error, file descriptor if success
 */
int setup_serial(struct termios *oldtio, char *serial_device);

/**
 * @brief Disconnect serial port
 *
 * @param fd File descriptor
 * @param oldtio Old serial port configuration
 * @return int 0 if success, -1 if failure
 */
int disconnect_serial(int fd, struct termios *oldtio);

/**
 * @brief Activate alarm
 *
 * @param signum
 */
void sig_handler(int signum);

/**
 * @brief Set up the alarm
 *
 */
void setup_sigalarm();
```

```
#endif
```

### 14.3.3 config.c

```
#include <fcntl.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

#include "config.h"


// ========= [Serial port configuration] ========= //
int setup_serial(struct termios *oldtio, char *serial_device) {
        /*
    Open serial port device for reading and writing and not as
↪   controlling tty
    because we don't want to get killed if linenoise sends CTRL-C.
    */
        int fd = open(serial_device, O_RDWR | O_NOCTTY);
        if (fd < 0) {
                perror(serial_device);
                exit(-1);
        }
        if (tcgetattr(fd, oldtio) == -1) { /* save current port
        ↪  settings */
                perror("tcgetattr");
                exit(-1);
        }

        struct termios newtio;
        bzero(&newtio, sizeof(newtio)); // Set contents of newtio
        ↪  to zero.
        newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD; /*
      BAUDRATE => B38400
      CS8 => Character size mask
      CLOCAL => Ignore modem control lines
```

```
      CREAD => Enable receiver
  */
        newtio.c_iflag = IGNPAR;  // Ignore framing errors and
        ↪  parity errors
        newtio.c_oflag = 0;

        /* set input mode (non-canonical, no echo,...) */
        newtio.c_lflag = 0;
        newtio.c_cc[VTIME] = DEFAULT_VTIME; /* inter-uint8_tacter
        ↪  timer unused (in deciseconds) */
        newtio.c_cc[VMIN] = DEFAULT_VMIN;  /* blocking read until 5
        ↪  uint8_ts received */

        if (tcflush(fd, TCIOFLUSH) == -1) { // Clear the data that
        ↪  might be present in the fd
                perror("tcflush");
                exit(-1);
        }
        if (tcsetattr(fd, TCSANOW, &newtio) == -1) {  // TCSANOW ->
        ↪  set attr takes place immediately
                perror("tcsetattr");
                exit(-1);
        }
        return fd;
}

int disconnect_serial(int fd, struct termios *oldtio) {
        if (tcsetattr(fd, TCSANOW, oldtio) == -1) {
                perror("tcsetattr");
                exit(-1);
        }
        if (close(fd) == -1) {
                perror("close");
                exit(-1);
        }
        return 0;
}

// ========= [Alarm setup] ========= //
volatile bool ALARM_ACTIVATED = false;

void sig_handler(int signum){
        ALARM_ACTIVATED = true;
}
```

```
void setup_sigalarm(){
        struct sigaction a;
        a.sa_handler = sig_handler;
        a.sa_flags = 0;
        sigemptyset(&a.sa_mask);
        sigaction(SIGALRM, &a, NULL);
}
```

### 14.3.4 core.h

```
#ifndef __CORE__
#define __CORE__

#include "common.h"

/**
 * @brief Create a non information frame object (any frame other
↪  than INFO)
 *
 * @param control Frame control byte
 * @param address Address byte
 * @return framecontent filled with the arguments given
 */
framecontent create_non_information_frame(uint8_t control, uint8_t
↪  address);

/**
 * @brief Create a information frame object
 *
 * @param data Pointer to data to be sent
 * @param data_len Data size
 * @param S Sequence bit
 * @param address Address byte
 * @return framecontent filled with the arguments given
 *          If data_len > MAX_INFO_SIZE, an empty fc with
↪  fc.data_len=0 will be returned.
 */
framecontent create_information_frame(uint8_t *data, size_t
↪  data_len, int S, uint8_t address);

/**
 * @brief Apply byte stuffing to data, note that the data buffer
 * must have at least 2*data_len of space to avoid any error.
 * @param data Pointer to data
```

```
 * @param data_len Data size
 * @return size_t New data size
 */
size_t byte_stuffing(uint8_t *data, size_t data_len);

/**
 * @brief Apply byte destuffing to data *
 * @param buffer Pointer to buffer
 * @param buf_size Buffer size
 * @return size_t New buffer size
 */
size_t byte_destuffing(uint8_t* buffer, size_t buf_size);

/**
 * @brief Calculate BCC for an array of bytes
 * @param data
 * @param data_len
 * @return uint8_t BCC value
 */
uint8_t calculate_bcc(uint8_t *data, size_t data_len);

#endif
```

### 14.3.5 core.c

```
#include "core.h"
#include "datalink_receiver.h"
#include "config.h"
#include "logger.h"

#include <string.h>

// ========= [Frame creation] ========= //
framecontent create_non_information_frame(uint8_t control, uint8_t
↪  address){
  framecontent fc;
  fc.control = control;
  fc.address = address;
  fc.data_len = 0;
  return fc;
}

framecontent create_information_frame(uint8_t *data, size_t
↪  data_len, int S, uint8_t address){
```

```
        if(data_len > MAX_INFO_SIZE){
                framecontent fc;
                fc.data_len = 0;
                return fc;
        }
        uint8_t bcc = calculate_bcc(data, data_len);

        framecontent fc;
        fc.control = CREATE_INFO_FRAME_CTL_BYTE(S);
        fc.address = address;
        memcpy(fc.data, data, data_len);
        fc.data[data_len] = bcc;
        size_t stuffed_bytes_size = byte_stuffing(fc.data,
        ↪  data_len+1);
        fc.data_len = stuffed_bytes_size;
        return fc;
}

// ========= [Byte stuffing] ========= //

size_t byte_stuffing(uint8_t *data, size_t data_len) {
        uint8_t aux_buffer[BUFFER_SIZE];
        if(data_len > BUFFER_SIZE){
                return -1;
        }
        memcpy(aux_buffer, data, data_len);

        int k = 0;
        for(int i = 0; i < data_len; ++i){
                if (aux_buffer[i] == FLAG) {
                        data[k++] = ESCAPE;
                        data[k++] = FLAG ^ 0x20;
                }
                else if (aux_buffer[i] == ESCAPE) {
                        data[k++] = ESCAPE;
                        data[k++] = ESCAPE ^ 0x20;
                } else {
                        data[k++] = aux_buffer[i];
                }
        }
        return k;
}

size_t byte_destuffing(uint8_t* buffer, size_t buf_size) {
        int size_dif = 0;
```

```
        int current = 0;
        for (int i = 0; i < buf_size; ++i){
                if( buffer[i] == ESCAPE ){
                        i++;
                        size_dif++;
                        buffer[current] = buffer[i] ^ 0x20;
                } else {
                        buffer[current] = buffer[i];
                }
                current++;
        }
        return buf_size - size_dif;
}

// ========= [BCC] ========= //
uint8_t calculate_bcc(uint8_t *data, size_t data_len){
        uint8_t result = 0;
        for(size_t i = 0; i < data_len; ++i){
                result ^= data[i];
        }
        return result;
}
```

### 14.3.6 datalink_emitter.h

```
#ifndef __DATALINKEMITTER__
#define __DATALINKEMITTER__

#include "common.h"

/**
 * @brief Fill buffer with the content of the frame
 *
 * @param buffer Buffer to be filled
 * @param buffer_size Size of the buffer
 * @param fc Content of the frame
 * @return int 0 if success, -1 if failure
 */
int frame_to_bytes(uint8_t *buffer, size_t buffer_size,
↪  framecontent *fc);

/**
 * @brief Write frame content to file descriptor
 *
```

```
 * @param fd File descriptor
 * @param frame Frame to be written
 * @param frame_size Size of frame
 * @return int 0 if success, -1 if failure
 */
int send_bytes(int fd, uint8_t *frame, size_t frame_size);

/**
 * @brief Send frame
 *
 * @param fd File descriptor
 * @param fc Frame
 * @return int 0 if success, -1 if failure
 */
int emit_frame(int fd, framecontent *fc);

/**
 * @brief Send frame until there is a response
 *
 * @param fd File descriptor
 * @param fc Frame
 * @param expected_response Expected response from receiver
 * @return int 0 if success, 1 if max attempts reached, -1 if
↪  failure
 */
int emit_frame_until_response(int fd, framecontent *fc, uint8_t
↪  expected_response);

#endif
```

### 14.3.7 datalink_emitter.c

```
#include "datalink_emitter.h"
#include "datalink_receiver.h"
#include "config.h"
#include "logger.h"

#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
```

```
int frame_to_bytes(uint8_t *buffer, size_t buffer_size,
↪  framecontent *fc) {
        if (buffer_size < 5 + fc->data_len) {
                return -1;
        }
        buffer[0] = FLAG;
        buffer[1] = fc->address; // Address
        buffer[2] = fc->control; // Control
        buffer[3] = (fc->address) ^ (fc->control); // BCC
        int i = 4;
        if(fc->data_len > 0){
                memcpy(buffer+4, fc->data, fc->data_len);
                i += fc->data_len;
        }
        buffer[i] = FLAG;
        return 0;
}

int send_bytes(int fd, uint8_t *buffer, size_t buffer_size) {
        int res = write(fd, buffer, buffer_size);
        if (res == -1) {
                perror("write");
                exit(-1);
        }
        return 0;
}

int emit_frame(int fd, framecontent *fc) {
        size_t buffer_size = 5 + fc->data_len;
        uint8_t buffer[buffer_size];
        if(frame_to_bytes(buffer, buffer_size, fc) < 0){
                return -1;
        }
        if(send_bytes(fd, buffer, buffer_size) < 0){
                return -1;
        }
        log_emission(fc);
        return 0;
}

int emit_frame_until_response(int fd, framecontent *fc, uint8_t
↪  expected_response){
        if(emit_frame(fd, fc) < 0){
                return -1;
        }
```

```
        int attempts = MAX_EMIT_ATTEMPTS;
        alarm(FRAME_RESEND_TIMEOUT);
        while(true){
                framecontent response_fc = receive_frame(fd);
                if(response_fc.control == expected_response){
                        break;
                } else { // Either receive_frame was interrupted by
                ↪  alarm or control byte is invalid (e.g REJ),
                ↪  therefore requiring reemission.
                        ALARM_ACTIVATED = false;
                        alarm(0);
                        if(attempts == 0){
                                break;
                        }
                        printf("Resending frame, attempt %d/%d\n",
                        ↪  MAX_EMIT_ATTEMPTS-attempts+1,
                        ↪  MAX_EMIT_ATTEMPTS);
                        if(emit_frame(fd, fc) < 0){
                                return -1;
                        }
                        alarm(FRAME_RESEND_TIMEOUT);
                        attempts--;
                }
        }
        alarm(0);
        ALARM_ACTIVATED = false;
        return attempts == 0 ? 1 : 0;
}
```

### 14.3.8 datalink_receiver.h

```
#ifndef __DATALINKRECEIVER__
#define __DATALINKRECEIVER__

#include <stdbool.h>

typedef enum {
        START,
        FLAG_RCV,
        A_RCV,
        C_RCV,
        BCC_OK,
        INFO,
        STOP
```

```
} receiver_state;

/**
 * @brief Process byte when in FLAG_RCV state
 *
 * @param byte
 * @return receiver_state Next state
 */
receiver_state statemachine_flag_received(uint8_t byte);

/**
 * @brief Verify if the given byte is an valid control byte
 *
 * @param byte
 * @return bool
 */
bool is_valid_control_byte(uint8_t byte);

/**
 * @brief Process byte when in A_RCV state
 *
 * @param byte
 * @return receiver_state Next state
 */
receiver_state statemachine_address_received(uint8_t byte);

/**
 * @brief Process byte when in C_RCV state
 *
 * @param byte
 * @param fc
 * @return receiver_state Next state
 */
receiver_state statemachine_control_received(uint8_t byte,
↪  framecontent *fc);

/**
 * @brief Receive a frame from fd
 *
 * @param fd
 * @return framecontent FC filled with the received frame.
 *                 If the function was interrupted by an alarm,
↪  fc.control will be equals to CTL_INVALID_FRAME.
```

```
 *                If an error occurs in a INFO frame data (wrong
↪  bcc), the frame will return with data_len=0 (data is
↪  discarded).
 *                If the received INFO frame data is greater (in
↪  length) than BUFFER_SIZE, the contents will be
 *        discarded and the state machine will reset to START
↪  state
 */
framecontent receive_frame(int fd);

#endif
```

### 14.3.9 datalink_receiver.c

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "common.h"
#include <string.h>
#include "core.h"
#include "datalink_receiver.h"
#include "logger.h"


receiver_state statemachine_flag_received(uint8_t byte) {
        return (byte == ADDRESS1 || byte == ADDRESS2) ? A_RCV :
        ↪  START;
}

bool is_valid_control_byte(uint8_t byte) {
        if (byte == CTL_SET || byte == CTL_UA || byte == CTL_DISC
                        || (APPLY_RESPONSE_CTL_MASK(byte) ==
                        ↪  CTL_RR)
                        || (APPLY_RESPONSE_CTL_MASK(byte) ==
                        ↪  CTL_REJ)
                        || IS_INFO_CONTROL_BYTE(byte)
                ) {
                return true;
        }
        return false;
}

receiver_state statemachine_address_received(uint8_t byte) {
```

```
        return is_valid_control_byte(byte) ? C_RCV : START;
}

receiver_state statemachine_control_received(uint8_t byte,
↪  framecontent *fc) {
        if (((fc->control) ^ (fc->address)) == byte) {
                if(IS_INFO_CONTROL_BYTE(fc->control)) {
                        return INFO;
                }
                return BCC_OK;
        }
        return START;
}

framecontent receive_frame(int fd) {
        framecontent fc;
        size_t buffer_pos = 0;
        uint8_t current_byte;
        receiver_state state = START;

        while (state != STOP) {
                int res = read(fd, &current_byte, 1);
                if (res == -1) {
                        if(errno == EINTR){ // Read was interrupted
                        ↪  by an alarm.
                                fc.control = CTL_INVALID_FRAME;
                                return fc;
                        }
                        perror("read");
                        exit(-1);
                }
                if(current_byte == FLAG){
                        if(state == BCC_OK){ // Marks the end of an
                        ↪  non-information frame
                                state = STOP;
                                break;
                        }
                        if(state == INFO){
                                state = STOP;
                                size_t destuffed_size =
                                ↪  byte_destuffing(fc.data,
                                ↪  buffer_pos);
```

```
                        uint8_t bcc =
                        ↪  fc.data[destuffed_size-1]; //
                        ↪  The last byte of the buffer is
                        ↪  the BCC. We can't distinguish
                        ↪  it from the data until we hit a
                        ↪  flag.
                        if(bcc == calculate_bcc(fc.data,
                        ↪  destuffed_size-1)){
                                fc.data_len =
                                ↪  destuffed_size-1;
                                break;
                        }
                        fc.data_len = 0;
                        break;
                        // If an error occurs, in data
                        ↪  (wrong BCC) the data is
                        ↪  discarded.
                }
                state = FLAG_RCV;
        } else {
                switch (state) {
                        case FLAG_RCV: state =
                        ↪  statemachine_flag_received(
                        current_byte);
                                fc.address = current_byte;
                                ↪  break;
                        case A_RCV:         state =
                            statemachine_address_received(
                            current_byte);
                                fc.control = current_byte;
                                ↪  break;
                        case C_RCV:         state =
                            statemachine_control_received(
                            current_byte, &fc); break;
                        case INFO:
                                if(buffer_pos <
                                ↪  BUFFER_SIZE){
                                        fc.data[buffer_pos++]
                                        = current_byte;
                                        ↪  break;
                                } else {
                                        state = START;
                                        buffer_pos = 0;
                                        break;
                                }
```

```
                                default: state = START;
                        }
                }
        }

        int random_value = rand() % 101;
        if(random_value < FER){
                if(random_value < FER_HEADER){
                        printf("[Efficiency Analysis]: Simulating
                        ↪  an error in the header\n");
                        return receive_frame(fd); // Return to
                        ↪  START as if an BCC error occured.
                } else {
                        printf("[Efficiency Analysis]: Simulating
                        ↪  an error in the data\n");
                        fc.data_len = 0;
                }
        }
        usleep(T_PROP);
        log_receival(&fc);
        return fc;
}
```

#### 14.3.10 logger.h

```
#ifndef __LOGGER__
#define __LOGGER__

#include "common.h"

/**
 * @brief Print the control byte
 *
 * @param byte Control byte
 */
void log_control_byte(uint8_t byte);

/**
 * @brief Print address byte
 *
 * @param byte Address byte
 */
void log_address_byte(uint8_t byte);
```

```
/**
 * @brief Print frame emition info
 *
 * @param fc Frame emitted
 */
void log_emission(framecontent *fc);

/**
 * @brief Print frame receival info
 *
 * @param fc Frame received
 */
void log_receival(framecontent *fc);

#endif
```

### 14.3.11 logger.c

```
#include "logger.h"
#include <stdbool.h>
#include <stdio.h>

void log_control_byte(uint8_t byte){
        switch(byte){
                case CTL_SET : printf("SET"); return;
                case CTL_UA : printf("UA"); return;
                case CTL_DISC : printf("DISC"); return;
        }
        if(IS_INFO_CONTROL_BYTE(byte)){
                printf("INFO");
                return;
        }
        else if(APPLY_RESPONSE_CTL_MASK(byte) == CTL_RR){
                printf("RR");
                return;
        }
        else if(APPLY_RESPONSE_CTL_MASK(byte) == CTL_REJ) {
                printf("REJ");
                return;
        }
        printf("INVALID");
        return;
}
void log_address_byte(uint8_t byte){
```

```
        switch(byte){
                case ADDRESS1 : printf("ADDRESS1"); break;
                case ADDRESS2 : printf("ADDRESS2"); break;
                default: printf("INVALID"); break;
        }
}
void log_emission(framecontent *fc){
    if(VERBOSE == false){
        return;
    }
        printf(" emit: CTL=");
        log_control_byte(fc->control);
        printf(" ADR=");
        log_address_byte(fc->address);
        if(IS_INFO_CONTROL_BYTE(fc->control)){
                printf(" INFO=\n");
                for(int i = 0;  i < fc->data_len; ++i){
                        printf(" %02x ", fc->data[i]);
                }
                printf("\n");
                printf(" S=%d",
                ↪  GET_INFO_FRAME_CTL_BIT(fc->control));
        }
        printf("\n\n");
}
void log_receival(framecontent *fc){
    if(VERBOSE == false){
        return;
    }
        printf(" receive: CTL=");
        log_control_byte(fc->control);
        printf(" ADR=");
        log_address_byte(fc->address);
        if(IS_INFO_CONTROL_BYTE(fc->control)){
                printf(" INFO=\n");
                for(int i = 0;  i < fc->data_len; ++i){
                        printf(" %02x ", fc->data[i]);
                }
                printf("\n");
                printf(" S=%d",
                ↪  GET_INFO_FRAME_CTL_BIT(fc->control));
        }
        printf("\n\n");
}
```

# 15 Anexo IV - Resultados de Testes de Eficiência

| | baud | tprop | fer | infosize | time | R | S | a |
|---|---|---|---|---|---|---|---|---|
| 0 | 9600 | 0.0 | 0.0 | 256 | 12.68207 | 6918.74173 | 0.7207 | 0.0 |
| 1 | 9600 | 0.0 | 0.0 | 512 | 12.15057 | 7221.39207 | 0.75223 | 0.0 |
| 2 | 9600 | 0.0 | 0.0 | 2048 | 39.45494 | 2223.90425 | 0.23166 | 0.0 |
| 3 | 9600 | 0.0 | 0.01 | 256 | 12.68285 | 6918.31968 | 0.72066 | 0.0 |
| 4 | 9600 | 0.0 | 0.01 | 512 | 12.15121 | 7221.01036 | 0.75219 | 0.0 |
| 5 | 9600 | 0.0 | 0.01 | 2048 | 39.45512 | 2223.89412 | 0.23166 | 0.0 |
| 6 | 9600 | 0.0 | 0.05 | 256 | 16.96988 | 5170.57401 | 0.5386 | 0.0 |
| 7 | 9600 | 0.0 | 0.05 | 512 | 12.15098 | 7221.14411 | 0.7522 | 0.0 |
| 8 | 9600 | 0.0 | 0.05 | 2048 | 39.45498 | 2223.90151 | 0.23166 | 0.0 |
| 9 | 9600 | 0.0 | 0.1 | 256 | 18.9691 | 4625.62881 | 0.48184 | 0.0 |
| 10 | 9600 | 0.0 | 0.1 | 512 | 14.7093 | 5965.20451 | 0.62138 | 0.0 |
| 11 | 9600 | 0.2 | 0.0 | 256 | 32.08483 | 2734.75016 | 0.28487 | 0.9375 |
| 12 | 9600 | 0.2 | 0.0 | 512 | 22.75232 | 3856.48608 | 0.40172 | 0.46875 |
| 13 | 9600 | 0.2 | 0.0 | 2048 | 36.05259 | 2433.7779 | 0.25352 | 0.11719 |
| 14 | 9600 | 0.2 | 0.01 | 256 | 32.0846 | 2734.7703 | 0.28487 | 0.9375 |
| 15 | 9600 | 0.2 | 0.01 | 512 | 22.75223 | 3856.50156 | 0.40172 | 0.46875 |
| 16 | 9600 | 0.2 | 0.01 | 2048 | 36.05267 | 2433.77244 | 0.25352 | 0.11719 |
| 17 | 9600 | 0.2 | 0.05 | 256 | 32.7704 | 2677.53854 | 0.27891 | 0.9375 |
| 18 | 9600 | 0.2 | 0.05 | 2048 | 36.05296 | 2433.75294 | 0.25352 | 0.11719 |
| 19 | 9600 | 0.2 | 0.1 | 512 | 25.61344 | 3425.70201 | 0.35684 | 0.46875 |
| 20 | 9600 | 1.0 | 0.0 | 256 | 94.32877 | 930.19342 | 0.0969 | 4.6875 |
| 21 | 9600 | 1.0 | 0.0 | 512 | 50.58649 | 1734.53413 | 0.18068 | 2.34375 |
| 22 | 9600 | 1.0 | 0.0 | 2048 | 44.71909 | 1962.1149 | 0.20439 | 0.58594 |
| 23 | 9600 | 1.0 | 0.01 | 256 | 95.32829 | 920.44033 | 0.09588 | 4.6875 |
| 24 | 9600 | 1.0 | 0.01 | 512 | 50.58734 | 1734.50507 | 0.18068 | 2.34375 |
| 25 | 9600 | 1.0 | 0.01 | 2048 | 44.71957 | 1962.09421 | 0.20438 | 0.58594 |
| 26 | 9600 | 1.0 | 0.05 | 256 | 95.30292 | 920.68533 | 0.0959 | 4.6875 |
| 27 | 9600 | 1.0 | 0.05 | 512 | 50.58745 | 1734.50143 | 0.18068 | 2.34375 |
| 28 | 9600 | 1.0 | 0.1 | 512 | 50.56315 | 1735.33507 | 0.18076 | 2.34375 |
| 29 | 38400 | 0.0 | 0.0 | 256 | 3.20295 | 27394.7108 | 0.7134 | 0.0 |
| 30 | 38400 | 0.0 | 0.0 | 512 | 3.0507 | 28761.92172 | 0.74901 | 0.0 |
| 31 | 38400 | 0.0 | 0.0 | 1024 | 2.98244 | 29420.24866 | 0.76615 | 0.0 |
| 32 | 38400 | 0.0 | 0.0 | 2048 | 2.9514 | 29729.65305 | 0.77421 | 0.0 |
| 33 | 38400 | 0.0 | 0.01 | 256 | 3.33405 | 26317.55538 | 0.68535 | 0.0 |
| 34 | 38400 | 0.0 | 0.01 | 512 | 4.05163 | 21656.47794 | 0.56397 | 0.0 |
| 35 | 38400 | 0.0 | 0.01 | 1024 | 2.98364 | 29408.36235 | 0.76584 | 0.0 |

| | baud | tprop | fer | infosize | time | R | S | a |
|---|---|---|---|---|---|---|---|---|
| 36 | 38400 | 0.0 | 0.01 | 2048 | 2.95161 | 29727.47014 | 0.77415 | 0.0 |
| 37 | 38400 | 0.0 | 0.05 | 256 | 3.26184 | 26900.14972 | 0.70052 | 0.0 |
| 38 | 38400 | 0.0 | 0.05 | 512 | 4.75065 | 18469.87737 | 0.48099 | 0.0 |
| 39 | 38400 | 0.0 | 0.05 | 1024 | 3.25498 | 26956.87063 | 0.702 | 0.0 |
| 40 | 38400 | 0.0 | 0.05 | 2048 | 2.95173 | 29726.26956 | 0.77412 | 0.0 |
| 41 | 38400 | 0.0 | 0.1 | 256 | 11.26348 | 7790.13173 | 0.20287 | 0.0 |
| 42 | 38400 | 0.0 | 0.1 | 512 | 5.27388 | 16637.47742 | 0.43327 | 0.0 |
| 43 | 38400 | 0.0 | 0.1 | 1024 | 2.98264 | 29418.1862 | 0.7661 | 0.0 |
| 44 | 38400 | 0.0 | 0.1 | 2048 | 2.95155 | 29728.1305 | 0.77417 | 0.0 |
| 45 | 38400 | 0.2 | 0.0 | 256 | 22.59242 | 3883.78035 | 0.10114 | 3.75 |
| 46 | 38400 | 0.2 | 0.0 | 512 | 13.65224 | 6427.07614 | 0.16737 | 1.875 |
| 47 | 38400 | 0.2 | 0.0 | 1024 | 9.18328 | 9554.75852 | 0.24882 | 0.9375 |
| 48 | 38400 | 0.2 | 0.0 | 2048 | 7.15214 | 12268.22089 | 0.31948 | 0.46875 |
| 49 | 38400 | 0.2 | 0.01 | 256 | 22.59196 | 3883.85912 | 0.10114 | 3.75 |
| 50 | 38400 | 0.2 | 0.01 | 512 | 13.65225 | 6427.07457 | 0.16737 | 1.875 |
| 51 | 38400 | 0.2 | 0.01 | 1024 | 9.18315 | 9554.89273 | 0.24883 | 0.9375 |
| 52 | 38400 | 0.2 | 0.01 | 2048 | 7.15211 | 12268.26056 | 0.31949 | 0.46875 |
| 53 | 38400 | 0.2 | 0.05 | 256 | 23.06406 | 3804.36004 | 0.09907 | 3.75 |
| 54 | 38400 | 0.2 | 0.05 | 512 | 15.65236 | 5605.79986 | 0.14598 | 1.875 |
| 55 | 38400 | 0.2 | 0.05 | 1024 | 9.85471 | 8903.76009 | 0.23187 | 0.9375 |
| 56 | 38400 | 0.2 | 0.05 | 2048 | 7.75561 | 11313.61443 | 0.29463 | 0.46875 |
| 57 | 38400 | 0.2 | 0.1 | 256 | 29.06525 | 3018.86306 | 0.07862 | 3.75 |
| 58 | 38400 | 0.2 | 0.1 | 512 | 16.19118 | 5419.2467 | 0.14113 | 1.875 |
| 59 | 38400 | 0.2 | 0.1 | 1024 | 13.18362 | 6655.53345 | 0.17332 | 0.9375 |
| 60 | 38400 | 0.2 | 0.1 | 2048 | 9.15209 | 9587.31648 | 0.24967 | 0.46875 |
| 61 | 38400 | 1.0 | 0.0 | 256 | 94.11969 | 932.25973 | 0.02428 | 18.75 |
| 62 | 38400 | 1.0 | 0.0 | 512 | 50.17947 | 1748.60356 | 0.04554 | 9.375 |
| 63 | 38400 | 1.0 | 0.0 | 1024 | 28.30943 | 3099.462 | 0.08072 | 4.6875 |
| 64 | 38400 | 1.0 | 0.0 | 2048 | 18.57479 | 4723.82203 | 0.12302 | 2.34375 |
| 65 | 38400 | 1.0 | 0.01 | 256 | 95.12096 | 922.44656 | 0.02402 | 18.75 |
| 66 | 38400 | 1.0 | 0.01 | 512 | 50.18029 | 1748.57505 | 0.04554 | 9.375 |
| 67 | 38400 | 1.0 | 0.01 | 1024 | 28.30937 | 3099.46795 | 0.08072 | 4.6875 |
| 68 | 38400 | 1.0 | 0.01 | 2048 | 18.57466 | 4723.8538 | 0.12302 | 2.34375 |
| 69 | 38400 | 1.0 | 0.05 | 256 | 99.02225 | 886.10389 | 0.02308 | 18.75 |
| 70 | 38400 | 1.0 | 0.05 | 512 | 51.16872 | 1714.79774 | 0.04466 | 9.375 |
| 71 | 38400 | 1.0 | 0.05 | 1024 | 29.31078 | 2993.57418 | 0.07796 | 4.6875 |
| 72 | 38400 | 1.0 | 0.05 | 2048 | 18.57434 | 4723.93554 | 0.12302 | 2.34375 |
| 73 | 38400 | 1.0 | 0.1 | 256 | 101.08678 | 868.00665 | 0.0226 | 18.75 |
| 74 | 38400 | 1.0 | 0.1 | 512 | 53.0216 | 1654.87272 | 0.0431 | 9.375 |

| | baud | tprop | fer | infosize | time | R | S | a |
|---|---|---|---|---|---|---|---|---|
| 75 | 38400 | 1.0 | 0.1 | 1024 | 31.01141 | 2829.4106 | 0.07368 | 4.6875 |
| 76 | 38400 | 1.0 | 0.1 | 2048 | 19.57821 | 4481.71726 | 0.11671 | 2.34375 |
| 77 | 230400 | 0.0 | 0.0 | 256 | 3.18966 | 27508.88336 | 0.1194 | 0.0 |
| 78 | 230400 | 0.0 | 0.0 | 512 | 3.05066 | 28762.27014 | 0.12484 | 0.0 |
| 79 | 230400 | 0.0 | 0.0 | 1024 | 2.9828 | 29416.6178 | 0.12768 | 0.0 |
| 80 | 230400 | 0.0 | 0.0 | 2048 | 2.95159 | 29727.71267 | 0.12903 | 0.0 |
| 81 | 230400 | 0.0 | 0.01 | 256 | 3.19063 | 27500.513 | 0.11936 | 0.0 |
| 82 | 230400 | 0.0 | 0.01 | 512 | 3.05144 | 28754.90804 | 0.1248 | 0.0 |
| 83 | 230400 | 0.0 | 0.01 | 1024 | 2.9828 | 29416.61545 | 0.12768 | 0.0 |
| 84 | 230400 | 0.0 | 0.01 | 2048 | 2.95181 | 29725.53005 | 0.12902 | 0.0 |
| 85 | 230400 | 0.0 | 0.05 | 256 | 3.19025 | 27503.84244 | 0.11937 | 0.0 |
| 86 | 230400 | 0.0 | 0.05 | 512 | 3.19058 | 27500.95688 | 0.11936 | 0.0 |
| 87 | 230400 | 0.0 | 0.05 | 1024 | 3.25543 | 26953.15905 | 0.11698 | 0.0 |
| 88 | 230400 | 0.0 | 0.05 | 2048 | 2.95185 | 29725.04267 | 0.12901 | 0.0 |
| 89 | 230400 | 0.0 | 0.1 | 256 | 7.33449 | 11963.2035 | 0.05192 | 0.0 |
| 90 | 230400 | 0.0 | 0.1 | 512 | 9.74268 | 9006.14851 | 0.03909 | 0.0 |
| 91 | 230400 | 0.0 | 0.1 | 2048 | 4.95191 | 17719.21404 | 0.07691 | 0.0 |
| 92 | 230400 | 0.2 | 0.0 | 256 | 22.59271 | 3883.72986 | 0.01686 | 22.5 |
| 93 | 230400 | 0.2 | 0.0 | 512 | 13.65237 | 6427.01441 | 0.0279 | 11.25 |
| 94 | 230400 | 0.2 | 0.0 | 1024 | 9.18318 | 9554.85626 | 0.04147 | 5.625 |
| 95 | 230400 | 0.2 | 0.0 | 2048 | 7.15226 | 12268.00783 | 0.05325 | 2.8125 |
| 96 | 230400 | 0.2 | 0.01 | 256 | 23.06419 | 3804.33861 | 0.01651 | 22.5 |
| 97 | 230400 | 0.2 | 0.01 | 512 | 14.19108 | 6183.04023 | 0.02684 | 11.25 |
| 98 | 230400 | 0.2 | 0.01 | 1024 | 9.18351 | 9554.51741 | 0.04147 | 5.625 |
| 99 | 230400 | 0.2 | 0.01 | 2048 | 7.15209 | 12268.3035 | 0.05325 | 2.8125 |
| 100 | 230400 | 0.2 | 0.05 | 256 | 25.53685 | 3435.97544 | 0.01491 | 22.5 |
| 101 | 230400 | 0.2 | 0.05 | 512 | 14.13738 | 6206.52383 | 0.02694 | 11.25 |
| 102 | 230400 | 0.2 | 0.05 | 1024 | 9.183 | 9555.04306 | 0.04147 | 5.625 |
| 103 | 230400 | 0.2 | 0.05 | 2048 | 7.56597 | 11597.18571 | 0.05034 | 2.8125 |
| 104 | 230400 | 0.2 | 0.1 | 256 | 26.01074 | 3373.37527 | 0.01464 | 22.5 |
| 105 | 230400 | 0.2 | 0.1 | 512 | 13.65219 | 6427.09904 | 0.0279 | 11.25 |
| 106 | 230400 | 0.2 | 0.1 | 1024 | 15.79181 | 5556.29826 | 0.02412 | 5.625 |
| 107 | 230400 | 0.2 | 0.1 | 2048 | 10.69676 | 8202.8592 | 0.0356 | 2.8125 |
| 108 | 230400 | 1.0 | 0.0 | 256 | 94.11918 | 932.26484 | 0.00405 | 112.5 |
| 109 | 230400 | 1.0 | 0.0 | 512 | 50.17783 | 1748.66081 | 0.00759 | 56.25 |
| 110 | 230400 | 1.0 | 0.0 | 1024 | 28.30787 | 3099.63218 | 0.01345 | 28.125 |
| 111 | 230400 | 1.0 | 0.0 | 2048 | 18.57265 | 4724.36525 | 0.02051 | 14.0625 |
| 112 | 230400 | 1.0 | 0.01 | 256 | 96.12126 | 912.84693 | 0.00396 | 112.5 |
| 113 | 230400 | 1.0 | 0.01 | 512 | 50.18032 | 1748.57379 | 0.00759 | 56.25 |

| | baud | tprop | fer | infosize | time | R | S | a |
|---|---|---|---|---|---|---|---|---|
| 114 | 230400 | 1.0 | 0.01 | 1024 | 28.3078 | 3099.64087 | 0.01345 | 28.125 |
| 115 | 230400 | 1.0 | 0.01 | 2048 | 18.57445 | 4723.90807 | 0.0205 | 14.0625 |
| 116 | 230400 | 1.0 | 0.05 | 256 | 95.11927 | 922.46297 | 0.004 | 112.5 |
| 117 | 230400 | 1.0 | 0.05 | 512 | 54.18229 | 1619.42208 | 0.00703 | 56.25 |
| 118 | 230400 | 1.0 | 0.05 | 1024 | 29.30936 | 2993.7197 | 0.01299 | 28.125 |
| 119 | 230400 | 1.0 | 0.05 | 2048 | 18.57586 | 4723.54865 | 0.0205 | 14.0625 |
| 120 | 230400 | 1.0 | 0.1 | 512 | 54.18269 | 1619.41022 | 0.00703 | 56.25 |
| 121 | 230400 | 1.0 | 0.1 | 1024 | 28.29877 | 3100.6291 | 0.01346 | 28.125 |
| 122 | 230400 | 1.0 | 0.1 | 2048 | 19.01994 | 4613.26385 | 0.02002 | 14.0625 |