Protocolo de Ligação de Dados

Mestrado Integrado Em Engenharia Informática e Computação

Redes de Computadores

Turma 4, Grupo

Duarte Manuel Ribeiro Pinto - 201304777

José Miguel Botelho Mendes - 201304828

Edgar Duarte Ramos - 201305973

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

04 de Novembro de 2015

Sumário

No âmbito da unidade curricular de Redes de Computadores, foi-nos proposto que elaborássemos um Protocolo de Ligação de Dados entre dois computadores através da porta de série. Este protocolo foi conseguido através de duas camadas fulcrais, a camada de Aplicação (Application Layer), que a camada mais a alto nível que trata do envio e receção dos ficheiro e a camada da Ligação de Dados (Data Link Layer), que trata de enviar e receber apenas tramas. Todos os conceitos utilizados foram lecionados nas aulas teóricas e práticas.

No final do trabalho o envio e receção dos ficheiros foram concretizados, sem erros. Se ocorrer alguma falha técnica durante a transferência, é possível voltar a reenviar o ficheiro, com sucesso.

Todo o código foi comentado de forma a poder-se correr Doxygen. Todas as funções e estruturas de dados foram devidamente comentadas.

Índice

1. Introdução

O objetivo deste trabalho foi implementar um protocolo de ligação de dados em Linux, utilizando a porta de série. Com a aplicação feita, é possível enviar ficheiros de qualquer tipo para um computador, desde que estejam ligados por uma porta de série e ambos tenham a aplicação a correr.

Nos próximos pontos serão abordados com mais pormenor vários pontos fulcrais da aplicação desenvolvida, assim como a forma que abordamos o trabalho proposto, os protocolos implementados e a estruturação do código.

2. Arquitetura

O projeto foi desenvolvido em ambiente Linux, através do uso de uma porta de série em modo não canónico. A arquitetura baseia-se em duas camadas chave: a camada de Ligação de Dados e a camada da Aplicação. A camada de ligação de dados trata do byte stuffing e destuffing, dos protocolos implementados e do envio e receção de tramas. A camada da Aplicação, sendo mais de alto nível, é responsável por dividir um ficheiro em determinado número de *packets*, enviando e recebendo cada um.

3. Estrutura do Código

3.1 Camada da Aplicação

A camada da Aplicação encontra-se nos ficheiros *app_layer.c*, *app_layer.h*. Estes ficheiros além da camada da Aplicação, também contêm algumas funções auxiliares para melhor compreensão do código, como *al_sendFile(LinkLayer *link_layer, char * file_buffer, int size)*, *al_readFile(LinkLayer * link_layer, FileInfo * file)*, *al_sendPacket(LinkLayer * link_layer, char * packet, int size, int i)* e mais algumas que tratam dos *Control Packets* e do tamanho e nome do ficheiro. A principal estrutura de dados é:

typedef struct {

int size;

char * name;

int fd;

int sequenceNumber;

}**FileInfo**; (toda a informação relativa ao ficheiro)

As principais funções são:

void app_layer_receiver(**LinkLayer*** link_layer);

void app_layer_transmitter(**LinkLayer*** link_layer, **char*** file_name);

3.2 Camada da Ligação de Dados

Esta camada, de mais baixo nível, está nos ficheiros *link_layer.h*, *link_layer.c*, *state.h*, *state.c*, *alarm.h, alarm.c, bstuffing.h* e *bstuffing.c*. Nos ficheiro state encontram-se todos os envios e receções de tramas, e as suas máquinas de estados. Os ficheiros alarm têm apenas a rotina que trata de todos os alarmes usados no projeto (muda a flag e soma +1 a um contador). Por fim, existe o link_layer que contém as funções mais fulcrais a esta camada:

int ll_open(**LinkLayer** *link_layer);

int ll_close(**LinkLayer** *link_layer);

int ll_write(**LinkLayer** *link_layer, int size);

int ll_read(**LinkLayer** *link_layer);

void ll_init(**LinkLayer** * newLinkLayer, **char** port[20], **int** baudRate, **unsigned int** timeout, **unsigned int** maxTries, **unsigned int** maxFrameSize, **int** status);

void ll_end(**LinkLayer** * linkLayer);

As primeiras quatro funções foram sugeridas nas aulas teóricas e foram implementadas corretamente. No entanto, as últimas duas são diferentes. Apesar de não existir, na nossa aplicação, a possibilidade de alterar a baudrate, o timeout, ou outros atributos, é possível alterá-los todos chamando a função *ll_init* que inicia a principal estrutura de dados desta camada, que será vista de seguida. A função ll_end termina o caminho da porta de série.

A principal estrutura de dados desta camada:

```
typedef struct {
    char * port;                    //Dispositivo /dev/ttySx, x = 0, 1
    int fd;
    int baudRate;                   //Velocidade de transmissão
    unsigned int timeout;           //Valor do temporizador: 1 s
    unsigned int maxTries;          //Número de tentativas em caso de falha
    unsigned int maxFrameSize;
    unsigned int maxPacketSize;
    int status;
    struct termios * oldtio;
    char * dataPacket;
```

} **LinkLayer**;

3.3 Outros ficheiros

Além de todos os ficheiros mencionados anteriormente, existe também um ficheiro *utils.h*, que contém todas as constantes usadas no projeto, e também um *Makefile*, para poder compilar mais facilmente o programa.

4. Casos de Uso

Correndo a aplicação normalmente, surgem dois ficheiros executáveis, o *writenoncanonical* (emissor) e o *noncanonical* (recetor). O *writenoncanonical* recebe a porta de série como primeiro argumento e o nome do ficheiro a transmitir como segundo. O *noncanonical* apenas recebe a porta de série como argumento.

Ambos os executáveis seguem uma ordem predeterminada, isto é:

- A main inicia o programa e cria a **link_layer** com a função *ll_init.*
- É chamada a função *app_layer*.
- É chamada a função *ll_end* para dar *close* ao caminho da porta de série.

No entanto, a função app_layer varia de acordo com o programa que a chama, correndo 2 funções diferentes, dependendo se é o emissor ou o transmissor.

5. Protocolo de ligação lógica

A camada da Ligação de Dados trata de:

- Abrir e fechar a ligação entre os dois computadores (emissor e recetor)
- Enviar e receber *packets*
- Fazer o byte stuffing e destuffing
- Verificar as tramas recebidas e enviar novas

Existem 6 funções que a camada da Aplicação pode usar livremente:

5.1 ll_open

A função ll_open trata de abrir a ligação da porta de série.

Da parte do emissor, envia uma trama SET, aguardando pela resposta do recetor de uma trama UA. Se passar o tempo definido em **timeout** na *struct* **LinkLayer**, então o emissor trata de reenviar a trama SET, recorrendo ao uso de alarmes para passar a espera bloqueante da função *read*. Se após **maxTries** definido na *struct* **LinkLayer** ainda não tiver recebido a trama UA, o programa fecha e a aplicação termina, pois não conseguiu estabelecer ligação.

Da parte do recetor, é bastante similar, esperando por uma trama SET e, após a receção desta, envia a trama UA. De acordo com as mesmas condições em acima referidas, o programa fecha e a aplicação termina se não conseguir receber a trama SET.

5.2 ll_close

Esta função é praticamente igual à *ll_open*, sendo as condições do máximo número de tentativas e tempo de espera iguais.

Da parte do emissor, envia uma trama DISC, e aguarda uma trama DISC também. Após a receção, envia uma trama UA.

Da parte do recetor, espera por uma trama DISC, envia uma nova trama DISC e espera até receber a trama UA.

5.3 ll_write

A função *ll_write* é de utilização única do emissor, servindo para enviar *data* para a porta. É aqui que ocorre o *bytestuffing* da *data* recebida e após o envio do *packet* espera pela receção de uma trama RR, com as mesmas condições referidas em cima.

5.4 ll_read

A função *ll_read* é também exclusiva do recetor, recebendo *data*, fazendo o *bytedestuffing* e validando essa mesma *data.* Dependendo do tamanho do *data Packet,* pode enviar uma trama RR, uma trama UA, ou falhar a receção dos dados.

5.5 ll_init

O construtor da *struct* **LinkLayer** que inicializa com todos os dados pretendidos.

5.6 ll_end

Dá *close* ao caminho da porta de série.

6. Protocolo de aplicação

A camada da aplicação trata de dois aspetos apenas: o envio de dois pacotes de controlo e o envio de um ficheiro separado em vários *data packets*.

6.1 *Packets*

Existem 2 tipos de packets enviados e recebidos, os *control packets* e os *data packets.*

6.1.1 *Control Packets*

As funções que tratam do envio e receção dos *Control Packets* são: *app_layer_receiver, app_layer_transmitter, al_readInitControlPacket* e *al_checkEndCtrlPacket.* Estas funções preenchem, enviam e recebem os *Control Packets*, o primeiro que indica o tamanho do ficheiro e o segundo que indica o nome do ficheiro. No final, voltam a enviar o mesmo *Control Packet,* mudando apenas o primeiro byte.

6.1.2 *Data Packets*

As funções que tratam do envio e receção dos *Data Packets* são: *app_layer_receiver*, *app_layer_transmitter, readInformationPacket, al_readFile, al_sendFile* e *al_sendPacket.* Estas funções dividem o ficheiro em *Data Packets* e enviando-os e recebendo-os.

7. Validação

Para garantir que o projeto estava bem implementado em termos de teste de erros e da sua validação, foram feitos vários testes durante a apresentação.

Durante a transferência dos *Data Packets* foi removido o cabo da porta de série durante breves segundos. Depois de se reconectar o cabo, a aplicação terminou sem erros e o ficheiro transferido não teve falhas.

O segundo teste foi bastante parecido com o primeiro, mas enquanto o cabo estava desconectado, passou-se uma chave metálica para enviar lixo durante a transferência. Novamente, após a conexão do cabo, a aplicação terminou sem erros e o ficheiro transferido não teve falhas.

8. Elementos de valorização

8.1 Verificação do número de bytes do ficheiro

Do lado do emissor, é possível ver a quantidade de bytes recebidos e é comparado com a quantidade de bytes que o ficheiro deveria ter.

8.2 Documentação Doxygen

Todo o código foi comentado de forma a poder-se correr o Doxygen e gerar documentação automática.

9. Conclusões

O grupo reagiu bem ao guião inicial e conseguiu compreender o que era pedido em cada fase do projeto e em cada camada, implementando todas as funcionalidades pedidas, apenas não criando um menu.

O grupo quer também agradecer aos docentes pelo apoio prestado nas aulas em termos de dúvidas e de compreensão de que tramas enviar em que situações. Além disso, os slides disponíveis no moodle da Unidade Curricular foram fulcrais para compreender o trabalho rapidamente.

10. Anexo I - Código Fonte

Ficheiro alarm.c

```
#include "alarm.h"


static int flag = -1;

static int tries = 1;


void atende()  {

        flag=1;

        tries++;

         // instala  rotina que atende interrupcao

}


int getFlag() {

  return flag;

}


int getTries() {

  return tries;

}


void setFlag(int f) {

  flag = f;

}

void setTries(int t) {

  tries = t;

}
```

Ficheiro app_layer.c

```
#include "app_layer.h"



void app_layer(LinkLayer *link_layer, char* file_name) {
	if (link_layer->status == TRANSMITTER)
		app_layer_transmitter(link_layer, file_name);
	else if (link_layer->status == RECEIVER)
		app_layer_receiver(link_layer);
}


void app_layer_receiver(LinkLayer *link_layer) {
	fprintf(stderr, "Abrir Ligação\n");
	ll_open(link_layer);
	fprintf(stderr, "Ligação Estabelecida\n");
	FileInfo file;


	al_readFile(link_layer, &file);


	//write(fd, file.file, bytesRead);


	setTries(0);
	if(ll_close(link_layer) < 0)
		fprintf(stderr, "Ligação terminada com sucesso\n");
	else
		fprintf(stderr, "Não foi possivel terminar a ligação correctamente\n");
}
```

```
int al_readFile(LinkLayer * link_layer, FileInfo * file){
	fprintf(stderr, "À espera de packet inicio\n");

	while(!al_readInitControlPacket(link_layer, file)){
		fprintf(stderr, "Continua à espera de packet inicio\n");
	}

	int received = FALSE;
	int bytesRead = 0;
	fprintf(stderr, "A recever dados\n");
	do{
		fprintf(stderr, "Started reading packet %d\n",file->sequenceNumber);
		int packetSize = ll_read(link_layer);

		if(al_checkEndCtrlPacket(link_layer,file, packetSize) > 0){
			received = TRUE;
			fprintf(stderr, "Recevido pacote controlo 2\n");
			break;
		}

		int bytes = readInformationPacket(link_layer,file,packetSize,bytesRead);

		if(bytes < 0)
			fprintf(stderr, "Failed reading packet %d\n", file->sequenceNumber);
		else{
			fprintf(stderr, "Success reading packet %d\n", file->sequenceNumber);
```

```
                bytesRead += bytes;
            }
            (file->sequenceNumber)++;

        }while(!received);

        fprintf(stderr, "Expected %d bytes. Received %d bytes!!\n", file->size,bytesRead);

        return bytesRead;
}

int readInformationPacket(LinkLayer * link_layer, FileInfo * file, int packetSize, int bytesRead){

        char * dataPacket = link_layer->dataPacket;
        if(dataPacket[0] != C_DATA)
            return -1;

        int seqNum = dataPacket[1];
        int size = ((unsigned int)dataPacket[2]<<4) +((unsigned int) dataPacket[3]); // FOI AQUI

        if(size != packetSize -4)
            return -1;

        if(seqNum != file->sequenceNumber)
            return -1;

        write(file->fd, &dataPacket[4], size);
```

```
	return size;
}

int al_readInitControlPacket(LinkLayer * link_layer, FileInfo * file){
	int dataPacketSize = ll_read(link_layer);
	if(dataPacketSize < 7)
		return -1;
	char * dataPacket = link_layer->dataPacket;
	if(dataPacket[0] != C_START)
		return -1;

	int fieldLength=0;
	int fileSize = readFileSize(&dataPacket[1], &fieldLength);
	if(fileSize < 0)
		return -1;

	char * fileName = readFileName(&dataPacket[1+2+fieldLength], &fieldLength);
	if(fileName == NULL)
		return -1;

	file->size = fileSize;
	file->name = fileName;

	int fd = open(fileName, O_WRONLY | O_TRUNC | O_CREAT, 0660);
	file->fd = fd;
	file->sequenceNumber = 0;
```

```
	return 1;
}

int al_checkEndCtrlPacket(LinkLayer * link_layer, FileInfo * file, int packetSize){
	char * dataPacket = link_layer->dataPacket;
	fprintf(stderr, "Ctrl check packet %d\n packetSize %d, dataPacket %d\n", file->sequenceNumber, packetSize, dataPacket[0]);
	if(packetSize < 7)
		return -1;
	if(dataPacket[0] != C_END)
		return -1;

	int fieldLength=0;
	int fileSize = readFileSize(&dataPacket[1], &fieldLength);
	fprintf(stderr, "Packet %d fileSize %d\n", file->sequenceNumber,fileSize);
	if(fileSize < 0)
		return -1;

	char * fileName = readFileName(&dataPacket[1+2+fieldLength], &fieldLength);
	if(fileName == NULL)
		return -1;

	if(file->size != fileSize)
		return -1;

	if(strcmp(file->name,fileName) != 0)
		return -1;
	return 1;
```

```
}

int readFileSize(char * dataPacket, int * fieldLength){
	if(dataPacket[0] != F_SIZE)
		return -1;
	*fieldLength = (unsigned int) dataPacket[1];
	int fileSize=*((uint32_t *) &dataPacket[2]);
	//int i = 0;









	return fileSize;
}

char * readFileName(char * dataPacket, int * fieldLength){
	if(dataPacket[0] != F_NAME)
		return NULL;


	*fieldLength = (unsigned int) dataPacket[1];
	char * fileName = malloc(*fieldLength);
	memcpy(fileName, &dataPacket[2], *fieldLength);
	return fileName;
}

void app_layer_transmitter(LinkLayer *link_layer, char * file_name) {
```

```
int file;
char * file_buffer;
struct stat * file_stat = malloc(sizeof(struct stat));
fprintf(stderr, "Abrir ficheiro %s\n",file_name);
file = open(file_name, O_RDONLY);

fstat(file, file_stat);

file_buffer = malloc(file_stat->st_size);

read(file, file_buffer, file_stat->st_size);

printf("tamanho!: %lld\n", (long long)file_stat->st_size);
        fprintf(stderr, "Abrir ligação\n");
        ll_open(link_layer);
        fprintf(stderr, "Ligação estabelecida\n");
        char controlPacket[3 + 4 + 3 + strlen(file_name) + 1];

        fprintf(stderr, "A enviar packet controlo 1\n");
        controlPacket[0] = C_START;
        controlPacket_size(file_stat, controlPacket);
        controlPacket_name(file_stat, &controlPacket[3 + controlPacket[2]], file_name);
        memcpy(link_layer->dataPacket, controlPacket, sizeof(controlPacket));
        ll_write(link_layer,sizeof(controlPacket));

        fprintf(stderr, "A começar envio de dados\n");
        int sentBytes = al_sendFile(link_layer, file_buffer, file_stat->st_size);
```

```
        printf("Sent %d bytes from %d\n", sentBytes,(int) file_stat->st_size);


        fprintf(stderr, "A enviar pacote controlo 2\n");
        controlPacket[0] = C_END;
        memcpy(link_layer->dataPacket, controlPacket, sizeof(controlPacket));
        ll_write(link_layer,sizeof(controlPacket));



        setTries(1);
        fprintf(stderr, "A terminar ligação\n");
        if(ll_close(link_layer) < 0)
                fprintf(stderr, "Ligação terminada com sucesso\n");
        else
                fprintf(stderr, "Não foi possivel terminar a ligação correctamente\n");

}

int al_sendFile(LinkLayer * link_layer, char * file_buffer, int size){
        int defautlPacketSize = getPacketSize(link_layer->maxFrameSize);

        int sentBytes = 0;
        int i = 0;
        while(sentBytes < size){
                int packetSize;
                if((size - sentBytes) >= defautlPacketSize )
                        packetSize = defautlPacketSize;
                else
```

```
                packetSize = size - sentBytes;
            fprintf(stderr, "Sending dataPacket i = %d with size %d\n", i, packetSize);
            if(al_sendPacket(link_layer, &file_buffer[sentBytes], packetSize,i) < 0){ //ve
isto
                fprintf(stderr, "Error sending dataPacket i=%d\n",i );
                exit(-1);
            }
            fprintf(stderr, "dataPacket sent i = %d, with size %d\n", i, packetSize);
            i++;
            sentBytes += packetSize;
        }

        return sentBytes;
}

int getPacketSize(int maxFrameSize){
        return (maxFrameSize - NUMBER_FLAGS)/2 - FRAME_HEADER_SIZE -
PACKET_HEADER_SIZE;
}

int al_sendPacket(LinkLayer * link_layer, char * packet, int size,int i){
        link_layer->dataPacket[0] = C_DATA;
        link_layer->dataPacket[1] = i;
        link_layer->dataPacket[2] = (size >> 4); //aqui
        link_layer->dataPacket[3] = size % 16; // aqui

        memcpy(&(link_layer->dataPacket[4]), packet, size);
```

```
        //fprintf(stderr, "%x %x\n", (unsigned char)packet[0], (unsigned) packet[1]);
        sleep(1);
        return ll_write(link_layer, size + PACKET_HEADER_SIZE);
}

void controlPacket_size(struct stat * file_stat, char * controlPacket){
        controlPacket[1] = F_SIZE; //tamanho ficheiro
        controlPacket[2] = 4; // 4 bytes é o máximo
        int i = 0;
        for(;i < controlPacket[2]; i++){
                controlPacket[i+3] = 0xFF & (file_stat->st_size >> (8 * i));
        }
}

void controlPacket_name(struct stat * file_stat, char * controlPacket, char * name){
        controlPacket[0] = F_NAME; //nome do ficheiro
        controlPacket[1] = strlen(name) + 1;      //+ 1 porque /0

        memcpy(&controlPacket[2], name, strlen(name) +1);
}
```

Ficheiro bstuffing.c

```
#include "bstuffing.h"


int bytestuffing(char * dataPacket, int size, char * stuffedPacket){
	int i;
	int j;
	for(i = 0, j = 0;i < size; i++, j++){
		char data = dataPacket[i];
		if(data == FLAG || data == ESC){
			//fprintf(stderr, "Stuffing byte i=%d, j=%d\n",i,j);
			stuffedPacket[j] = ESC;
			j++;
			stuffedPacket[j] = data^SUBS;
		}else{
			stuffedPacket[j] = dataPacket[i];
		}
	}

	return j;
}


int bytedestuffing(char * stuffedPacket, int size, char * dataPacket){
	int i;
	int j;
	for(i = 0, j = 0; j < size; i++, j++){
		if(stuffedPacket[j] == ESC){
			//fprintf(stderr, "destuffing byte i = %d, j = %d \n",i,j);
```

```
                j++;
                dataPacket[i] = stuffedPacket[j]^SUBS;
            }else
                dataPacket[i] = stuffedPacket[j];
        }
        return i;
}
```

Ficheiro link_layer.c

```
#include "link_layer.h"



static int s = 0;


int ll_open(LinkLayer *link_layer) {
   if (link_layer->status == RECEIVER)
                    return ll_open_receiver(link_layer);
   else
                    return ll_open_transmitter(link_layer);
}


int ll_close(LinkLayer *link_layer) {
   if (link_layer->status == RECEIVER)
                    return ll_close_receiver(link_layer);
   else
                    return ll_close_transmitter(link_layer);
}


int ll_open_receiver(LinkLayer *link_layer) {
   char SET[5];
   char UA[5];


   UA[0] = F;
   UA[1] = A;
   UA[2] = C_UA;
```

```
    UA[3] = A^C_UA;
    UA[4] = F;

    receive_SET(link_layer->fd, SET);

    //printf("FLAGS READ FROM SET: %x, %x, %x, %x, %x\n", SET[0], SET[1], SET[2], SET[3],
SET[4]);

    send_UA(link_layer->fd, UA);
    return -1;

}

int ll_open_transmitter(LinkLayer *link_layer) {
        int tries = getTries();
        char UA[5];
        char SET[5];
        SET[0] = F;
        SET[1] = A;
        SET[2] = C_SET;
        SET[3] = BCC_SET;
        SET[4] = F;

        /* WRITE SET */

        while(tries <= ATTEMPTS){
                printf("Attempt %d\n", tries);
                tries = getTries();
```

```
                send_SET(link_layer->fd, SET);


                setStopUA(FALSE);


                receive_UA(link_layer->fd, UA);


                if(!(check_UA(UA))){
                    //printf("FLAGS READ FROM UA: %x, %x, %x, %x, %x\n\n", UA[0], UA[1],
UA[2], UA[3], UA[4]);
                    tries=-1;
                    break;
                }else{
                    tries++;
                }
        }
        return tries;
}


int ll_close_receiver(LinkLayer *link_layer) {


   char DISC[link_layer->maxFrameSize];
   char UA[5];
   char DISC_send[5];


   DISC_send[0] = F;
   DISC_send[1] = A;
   DISC_send[2] = C_DISC;
```

```
DISC_send[3] = A^C_DISC;

DISC_send[4] = F;


int diskSize = 0;

char dataPacket[link_layer->maxFrameSize];


do {

 setStopFRAME(FALSE);

 diskSize = receive_FRAME(link_layer->fd, DISC, link_layer->maxFrameSize);

 int result = check_I(dataPacket, s, DISC, diskSize);

 if(result == RE_SEND_RR){

       send_RR(link_layer->fd,s);

 }

}while(check_DISC(DISC));



 printf("FLAGS READ WITH SUCCESS FROM DISC: %x, %x, %x, %x, %x\n", DISC[0], DISC[1],
DISC[2], DISC[3], DISC[4]);


int tries = getTries();

fprintf(stderr, "Tries = %d\n", tries);


while(tries <= ATTEMPTS){

                printf("Attempt %d\n", tries);

                tries = getTries();


                send_DISC(link_layer->fd, DISC_send);
```

```
                setStopUA(FALSE);
                receive_UA(link_layer->fd, UA);
                if(!(check_UA(UA))){
                        printf("FLAGS READ FROM UA: %x, %x, %x, %x, %x\n\n", UA[0], UA[1],
UA[2], UA[3], UA[4]);
                        tries=-1;
                        break;
                }
                else{
                  tries++;
                }
        }
        return tries;
}

int ll_close_transmitter(LinkLayer *link_layer) {

   char DISC[5];
   char DISC_rec[5];
   char UA[5];

   DISC[0] = F;
   DISC[1] = A;
   DISC[2] = C_DISC;
   DISC[3] = DISC[1]^DISC[2];
   DISC[4] = F;
```

```
    UA[0] = F;
    UA[1] = A;
    UA[2] = C_UA;
    UA[3] = A^C_UA;
    UA[4] = F;

    int tries = getTries();

        while(tries <= ATTEMPTS){
            printf("Attempt %d\n", tries);

            send_DISC(link_layer->fd, DISC);

            setStopDISC(FALSE);

            receive_DISC(link_layer->fd, DISC_rec);

            if(!(check_DISC(DISC_rec)))
            {
                //printf("FLAGS READ FROM DISC: %x, %x, %x, %x, %x\n\n",
DISC_rec[0], DISC_rec[1], DISC_rec[2], DISC_rec[3], DISC_rec[4]);
                tries = -1;
                break;
            }
            else
            {
                if (tries == getTries())
```

```
                        {
                            setTries(tries++);
                        }
                }

                tries = getTries();

        }
        if(tries < 0)
                send_UA(link_layer->fd, UA);
        return tries;
}

int ll_write(LinkLayer *link_layer, int size) {
        char * data_packet = link_layer->dataPacket;
        char frameAdder[3 + 1 + size];

        int current_s = s;

        char C = 0;
        C = C|(s << 5);
        frameAdder[0] = A;
        frameAdder[1] = C;
        frameAdder[2] = frameAdder[0] ^ frameAdder[1];

        int i=0;
        frameAdder[3] = data_packet[0];
```

```
        //fprintf(stderr, "data_packet[%d]=%x\n",i, data_packet[i]);
        char bcc_2 = frameAdder[3];
        for (i = 1; i < size; i++) {
                frameAdder[i+3]=data_packet[i];
                //fprintf(stderr, "data_packet[%d]=%x\n",i, (unsigned char) data_packet[i]);
                bcc_2^=data_packet[i];
        }
        frameAdder[size+3] = bcc_2;
        //fprintf(stderr, "bcc_2 %x\n", (unsigned char) bcc_2);
        char frame[(size + 4) * 2 + 2];
        char * stuffedPacket = frame + 1;
        int size_stuffed_packet = bytestuffing(frameAdder, size + 4, stuffedPacket);
        //fprintf(stderr, "size_stuffed_packet=%d size%d\n", size_stuffed_packet, size);
        int frameSize = size_stuffed_packet + 2;
        //fprintf(stderr, "bcc_2 stuffed %x\n",(unsigned char)
stuffedPacket[size_stuffed_packet - 1]);



        frame[0] = FLAG;
        frame[frameSize - 1] = FLAG;


        setTries(1);
        int tries = getTries();


        while (tries < link_layer->maxTries) {


                write(link_layer->fd, frame, frameSize); //enviar packet
                alarm(3);
```

```
                setFlag(0);


                char answerRR[5];

                setStopRR(FALSE);

                int ans = receive_RR(link_layer->fd, answerRR, current_s); //receber RR

                //printf("FLAGS READ FROM RR: %x, %x, %x, %x, %x\n\n", answerRR[0],
answerRR[1], answerRR[2], answerRR[3], answerRR[4]);

                tries = getTries();

                fprintf(stderr, "%d\n", ans);

                if (ans < 0) {

                        continue;

                }

                s = ans;

                break;

        }



        if (tries == link_layer->maxTries)

                return -1;


        return frameSize;

}


int ll_read(LinkLayer * link_layer) {

        char *dataPacket = link_layer->dataPacket;

        int dataPacketSize = 0;

        int validated = FALSE;

        char currC;
```

```
	char UA[5];



	while(!validated){
		char frame[link_layer->maxFrameSize];


		setStopFRAME(FALSE);
		int frameSize = receive_FRAME(link_layer->fd, frame, link_layer-
>maxFrameSize);


		dataPacketSize = check_I(dataPacket, s, frame, frameSize);
		fprintf(stderr, "check_I %d\n", dataPacketSize);
		switch(dataPacketSize){
			case FAILED:
				break;
			case RE_SEND_RR:
				currC = (s<<5);
				send_RR(link_layer->fd,currC);
				break;
			case RE_SEND_SET:
				UA[0] = F;
				UA[1] = A;
				UA[2] = C_UA;
				UA[3] = A^C_UA;
			    UA[4] = F;


				send_UA(link_layer->fd,UA);
				break;
```

```
                        default:
                                validated = TRUE;
                }
        }
        fprintf(stderr, "Vai enviar o RR com s = %d\n", s ^ 0x1);
        send_RR(link_layer->fd,(s ^ 0x1));
        s ^= 0x01;

        return dataPacketSize;
}



void ll_init(LinkLayer * newLinkLayer, char * port, int baudRate, unsigned int timeout, unsigned int maxTries, unsigned int maxFrameSize, int status){
   struct termios * oldtio = malloc(sizeof(struct termios));
   struct termios newtio;

   newLinkLayer->port = port;

        int fd = open(port, O_RDWR | O_NOCTTY );
        if (fd < 0) {
                fprintf(stderr, "Error opening port %s\n", port);
                exit (-1);
        }

        if ( tcgetattr(fd,oldtio) == -1) { /* save current port settings */
```

```
  perror("tcgetattr");
  exit(-1);
}


bzero(&newtio, sizeof(newtio));

newtio.c_cflag = baudRate | CS8 | CLOCAL | CREAD;
newtio.c_iflag = IGNPAR;
newtio.c_oflag = 0;

/* set input mode (non-canonical, no echo,...) */
newtio.c_lflag = 0;

newtio.c_cc[VTIME]    = 0;   /* inter-character timer unused */
newtio.c_cc[VMIN]     = 1;   /* blocking read until 5 chars received */

        /*
        VTIME e VMIN devem ser alterados de forma a proteger com um temporizador a
        leitura do(s) próximo(s) caracter(es)
        */


tcflush(fd, TCIOFLUSH);

if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
  perror("tcsetattr");
```

```
    exit(-1);
  }

  newLinkLayer->fd = fd;
  newLinkLayer->baudRate = baudRate;
  newLinkLayer->port = port;
  newLinkLayer->timeout = timeout;
  newLinkLayer->maxTries = maxTries;
  newLinkLayer->maxFrameSize = maxFrameSize;
  newLinkLayer->status = status;
  newLinkLayer->oldtio = oldtio;

  newLinkLayer->dataPacket = malloc( (maxFrameSize - 2)/2 - (3 + 1));

  fprintf(stderr,"New termios structure set\n");

}



void ll_end(LinkLayer * linkLayer){
        if ( tcsetattr(linkLayer->fd,TCSANOW,linkLayer->oldtio) == -1) {
    perror("tcsetattr");
    exit(-1);
  }
 close(linkLayer->fd);
}
```

Ficheiro state.c

```
#include "state.h"

#include "alarm.h"

#include "utils.h"

#include "bstuffing.h"


#include <unistd.h>

#include <stdio.h>


#define FAILED -1

#define RE_SEND_RR -2

#define RE_SEND_SET -3


static volatile int STOP_UA=FALSE;


static volatile int STOP_SET=FALSE;


static volatile int STOP_DISC=FALSE;


static volatile int STOP_RR=FALSE;


static volatile int STOP_FRAME=FALSE;



int send_SET(int fd, char *SET) {


        int res;
```

```
        int flag = getFlag();

        if (flag){
                alarm(3);
                setFlag(0);
        }

        res = write(fd, SET, 5);

        //printf("FLAGS SENT FROM SET: %x, %x, %x, %x, %x\n\n", SET[0], SET[1], SET[2],
SET[3], SET[4]);

        return res;
}

int send_UA(int fd, char *UA) {
  int res;

  res = write(fd, UA, 5);

  //printf("FLAGS SENT FROM UA: %x, %x, %x, %x, %x\n", UA[0], UA[1], UA[2], UA[3], UA[4]);

  return res;
}

int send_DISC(int fd, char *DISC) {

  int res;
```

```
  int flag = getFlag();


        if (flag){
                alarm(3);
                setFlag(0);
        }


  res = write(fd, DISC, 5);


  //printf("FLAGS SENT FROM DISC: %x, %x, %x, %x, %x\n", DISC[0], DISC[1], DISC[2], DISC[3],
DISC[4]);


  return res;
}


int send_RR(int fd, int r){
        char RR[5];


        RR[0] = FLAG;
        RR[1] = A;
        RR[2] = (r << 5) | 0x01;
        RR[3] = RR[1] ^ RR[2];
        RR[4] = FLAG;


        write(fd, RR, 5);
        printf("FLAGS SENT FROM RR: %x, %x, %x, %x, %x\n", RR[0], RR[1], RR[2], RR[3], RR[4]);
        return 0;
```

```
}


void receive_UA(int fd, char *UA) {

        int option = START;
        char flag_ST;

        while(!(STOP_UA)) {
                //fprintf(stderr, "flag %d\n", getFlag());
                read(fd, &flag_ST, 1);
                int flag = getFlag();
                //fprintf(stderr, "option %d, flag_ST %x flag %d\n",option, flag_ST,flag);
                if(flag && flag != -1){
                    alarm(0);
                    setFlag(-1);
                    STOP_UA = TRUE;
                  }

                switch (option){
                        case START:
                                if (flag_ST == F){
                                        option = FLAG_RCV;
                                        UA[0] = flag_ST;
                                }
                                else
                                        option = START;
```

```
            break;


case FLAG_RCV:
            if (flag_ST == F){
                        option = FLAG_RCV;
                        UA[0] = flag_ST;
                }
            else if (flag_ST == A){
                        option = A_RCV;
                        UA[1] = flag_ST;
                }
            else
                option = START;
            break;


case A_RCV:
            if (flag_ST == F){
                        option = FLAG_RCV;
                        UA[0] = flag_ST;
                }
            else if (flag_ST == C_UA){
                        option = C_RCV;
                        UA[2] = flag_ST;
                }
            else
                option = START;
            break;
```

```
case C_RCV:
        if (flag_ST == F){
                option = FLAG_RCV;
                UA[0] = flag_ST;
        }
        else if (flag_ST == BCC_UA){
                option = BCC_OK;
                UA[3] = flag_ST;
        }
        else
                option = START;
        break;

case BCC_OK:
        if (flag_ST == F){
                option = STOP_ST;
                STOP_UA = TRUE;
                UA[4] = flag_ST;
        }
        else
                option = START;
        break;

case STOP_ST:
        STOP_UA = TRUE;
        break;
```

```
                default:
                        break;
                }
        }
}

void receive_SET(int fd, char *SET) {

  char flag_ST;
  int option = START;

  while(!(STOP_SET)){
        read(fd, &flag_ST, 1);

        switch (option){
                case START:
                        if (flag_ST == F){
                                option = FLAG_RCV;
                                SET[0] = flag_ST;
                        }
                        else
                                option = START;
                        break;

                case FLAG_RCV:
                        if (flag_ST == F){
```

```
                        option = FLAG_RCV;
                        SET[0] = flag_ST;
                }
        else if (flag_ST == A){
                        option = A_RCV;
                        SET[1] = flag_ST;
                }
        else
                option = START;
        break;

case A_RCV:
        if (flag_ST == F){
                        option = FLAG_RCV;
                        SET[0] = flag_ST;
                }
        else if (flag_ST == C_SET){
                        option = C_RCV;
                        SET[2] = flag_ST;
                }
        else
                option = START;
        break;

case C_RCV:
        if (flag_ST == F){
                        option = FLAG_RCV;
```

```
                        SET[0] = flag_ST;
                    }
                else if (flag_ST == BCC_SET){
                        option = BCC_OK;
                        SET[3] = flag_ST;
                    }
                else
                    option = START;
                break;

        case BCC_OK:
                if (flag_ST == F){
                        option = STOP_ST;
                        SET[4] = flag_ST;
                        STOP_SET = TRUE;
                    }
                else
                    option = START;
                break;

        case STOP_ST:
                STOP_SET = TRUE;
                break;

        default:
                break;
    }
```

```
  }
}

void receive_DISC(int fd, char *DISC_rec) {

  char flag_ST;
  int option = START;

  while(!(STOP_DISC)){

          read(fd, &flag_ST, 1);

          int flag = getFlag();
             if(flag && flag != -1){
                    alarm(0);
                    setFlag(-1);
                    STOP_DISC = TRUE;
                   }
          switch (option){
                    case START:
                              if (flag_ST == F){
                                        option = FLAG_RCV;
                                        DISC_rec[0] = flag_ST;
                                        }
                              else
                                        option = START;
                              break;
```

```
case FLAG_RCV:
        if (flag_ST == F) {
                option = FLAG_RCV;
                DISC_rec[0] = flag_ST;
                }
        else if (flag_ST == A){
                option = A_RCV;
                DISC_rec[1] = flag_ST;
                }
        else
                option = START;
        break;

case A_RCV:
        if (flag_ST == F){
                        option = FLAG_RCV;
                        DISC_rec[0] = flag_ST;
                }
        else if (flag_ST == C_DISC){
                        option = C_RCV;
                        DISC_rec[2] = flag_ST;
                }
        else
                option = START;
        break;
```

```
case C_RCV:
        if (flag_ST == F){
                option = FLAG_RCV;
                DISC_rec[0] = flag_ST;
        }
        else if (flag_ST == BCC_DISC){
                option = BCC_OK;
                DISC_rec[3] = flag_ST;
        }
        else
                option = START;
        break;

case BCC_OK:
        if (flag_ST == F){
                option = STOP_ST;
                STOP_DISC = TRUE;
                DISC_rec[4] = flag_ST;
        }
        else
                option = START;
        break;

case STOP_ST:
        STOP_DISC = TRUE;
        break;
```

```
                default:
                        break;
        }
 }
}

int receive_RR(int fd, char *RR, int s) {

        char flag_ST;
        int option = START;
        int r = s ? 0 : 1;
        int c_rr = 1 | (r << 5);

        while(!(STOP_RR)){

                read(fd, &flag_ST, 1);
                //fprintf(stderr, "option %d flag_ST 0x%x flag %d r %d c_rr %x\n", option,
flag_ST, getFlag(), r, c_rr);
                int flag = getFlag();
                if(flag && flag != -1){
                   alarm(0);
                   setFlag(-1);
                   STOP_RR = TRUE;
                   return -1;
                }

                switch (option) {
```

```
case START:
        if (flag_ST == F){
                option = FLAG_RCV;
                RR[0] = flag_ST;
        }
        else
                option = START;
        break;
case FLAG_RCV:
        if (flag_ST == F){
                option = FLAG_RCV;
                RR[0] = flag_ST;
        }
        else if (flag_ST == A) {
                option = A_RCV;
                RR[1] = flag_ST;
        }
        else
                option = START;
        break;
case A_RCV:
        if (flag_ST == F){
                option = FLAG_RCV;
                RR[0] = flag_ST;
        }
        else if (flag_ST == c_rr) {
                option = C_RCV;
```

```
                        RR[2] = flag_ST;
                }
        else
                option = START;
        break;
case C_RCV:
        if (flag_ST == F) {
                        option = FLAG_RCV;
                        RR[0] = flag_ST;
                }
        else if (flag_ST == (c_rr^A)){
                        option = BCC_OK;
                        RR[3] = flag_ST;
                }
        else
                option = START;
        break;
case BCC_OK:
        if (flag_ST == F){
                        option = STOP_ST;
                        STOP_RR = TRUE;
                        RR[4] = flag_ST;
                }
        else
                option = START;
        break;
```

```
                        case STOP_ST:
                                STOP_RR = TRUE;
                                break;
                        default:
                                break;
                }
        }
        return r;
}

int receive_I(int fd, char *I, int maxFrameSize) {

        char flag_ST;
        int data = 0;
        int option = START;

        while(!(STOP_FRAME)){
                read(fd, &flag_ST, 1);
                switch (option) {
                        case START:
                                data = 0;
                                if (flag_ST == F){
                                        option = FLAG_RCV;
                                        I[0] = flag_ST;
                                        data++;
                                }
                                else
```

```
                option = START;
        break;
case FLAG_RCV:
        data = 1;
        if (flag_ST == F) {
                option = FLAG_RCV;
                I[0] = flag_ST;
        }
        else{
                I[data] = flag_ST;
                data++;
                option = A_RCV;
        }
        break;
case A_RCV:
        if (data >= 5 && flag_ST == F){
                I[data] = flag_ST;
                option = STOP_ST;
        }
        else if (data > maxFrameSize){
                option = START;
        }
        else if (data < 6 && flag_ST == F){
                option = A_RCV;
        }
        else{
                I[data] = flag_ST;
```

```
                                        data++;
                                        option = A_RCV;
                                }
                                break;
                        case STOP_ST:
                                data++;
                                STOP_FRAME = TRUE;
                                break;
                        default:
                                break;
                }
        }
        return data;
}

int receive_FRAME(int fd, char *FRAME, int maxFrameSize){
        char flag_ST;
        int data = 0;
        int option = START;

        while(!(STOP_FRAME)){
                read(fd, &flag_ST, 1);
                //fprintf(stderr, "option %d, flag_ST %x data %d\n",option, (unsigned
char)flag_ST,data);

                switch (option) {
                        case START:
                                data = 0;
```

```
            if (flag_ST == F){
                        option = FLAG_RCV;
                        FRAME[0] = flag_ST;
                        data++;
            }
            else
                        option = START;
            break;
case FLAG_RCV:
            data = 1;
            if (flag_ST == F) {
                        option = FLAG_RCV;
                        FRAME[0] = flag_ST;
            }
            else{
                        FRAME[data] = flag_ST;
                        data++;
                        option = A_RCV;
            }
            break;
case A_RCV:
            if (data >= 4 && flag_ST == F){
                        FRAME[data] = flag_ST;
                        option = STOP_ST;
                        data++;
                        STOP_FRAME = TRUE;
            }
```

```
                else if (data > maxFrameSize){
                        option = START;
                }
                else if (data < 4 && flag_ST == F){
                        option = A_RCV;
                }
                else{
                        FRAME[data] = flag_ST;
                        data++;
                        option = A_RCV;
                }
                break;
        case STOP_ST:
                data++;
                STOP_FRAME = TRUE;
                //fprintf(stderr, "Vai sair\n");
                break;
        default:
                break;
        }
    }
    return data;
}


int check_UA(char *sent) {
    int error = 0;
```

```
	if (sent[0] != F || sent[1] != A || sent[2] != C_UA || sent[3] != (sent[1]^sent[2]) ||
sent[4] != F)

		error = 1;

	return error;

}


int check_SET(char *sent) {

	int error = 0;

	if (sent[0] != F || sent[1] != A || sent[2] != C_SET || sent[3] != (sent[1]^sent[2]) ||
sent[4] != F)

		error = 1;

	return error;

}


int check_I(char * dataPacket, int s, char *frame, int frameSize){


	int stuffedPacketSize  = frameSize - 2;


	if(stuffedPacketSize < 6)

		return FAILED;




	char * stuffedPacket = frame + sizeof(*frame);

	//Fazer destuff ao packet

	char framedPacket[stuffedPacketSize];

	int framedPacketSize = bytedestuffing(stuffedPacket, stuffedPacketSize,
framedPacket);
```

```
	//fprintf(stderr, "framedPacketSize=%d, byte = %x\n",(unsigned char) framedPacketSize,(unsigned char) framedPacket[284]);


	//Verificar o A
	if(framedPacket[0] != A)
		return FAILED;


	//Verificar o C
	char currC = (s<<5);


	if(framedPacket[1] != (s<<5)){
		if(currC == (s ^ 0x1)<<5)
			return RE_SEND_RR;
		if(check_SET(framedPacket) && framedPacketSize == 5)
			return RE_SEND_SET;
		return FAILED;
	}
	//Verificar BCC1
	if(framedPacket[2] != (A ^ currC))
		return FAILED;



	//Verificar BCC2 e ao mesmo tempo passar para o array onde é suposto guardar o dataPacket
	int i = 3;
	char bcc_2 = framedPacket[i];
	dataPacket[i-3] = framedPacket[i];
	//fprintf(stderr, "framedPacket[3] = %x\n", (unsigned char)framedPacket[3]);
```

```
	for(i = 4; i < framedPacketSize - 1; i++){
		dataPacket[i-3] = framedPacket[i];
		bcc_2 ^= framedPacket[i];
	}
	fprintf(stderr, "Verificar BCC2 bcc_2 %x, Esperado %x, i = %d\n", (unsigned char)
bcc_2,(unsigned char)framedPacket[framedPacketSize - 1],i);
	if(bcc_2 != framedPacket[framedPacketSize - 1] )
		return FAILED;

	fprintf(stderr, "Success, returning %d\n", framedPacketSize-4);
	return framedPacketSize-4;
}

int check_DISC(char *DISC_rec) {
	int error = 0;
	if (DISC_rec[0] != F || DISC_rec[1] != A || DISC_rec[2] != C_DISC || DISC_rec[3] !=
(DISC_rec[1]^DISC_rec[2]) || DISC_rec[4] != F)
		error = 1;
	return error;
}

int getStopUA(){
  return STOP_UA;
}

void setStopUA(int st) {
  STOP_UA = st;
}
```

```
int getStopSET() {
  return STOP_SET;
}

void setStopSET(int st) {
  STOP_SET = st;
}

int getStopDISC() {
  return STOP_DISC;
}

void setStopDISC(int st) {
  STOP_DISC = st;
}

void setStopRR(int st) {
        STOP_RR = st;
}

int getStopRR() {
        return STOP_RR;
}

void setStopFRAME(int st){
        STOP_FRAME = st;
```

```
}


int getStopFRAME(){
        return STOP_FRAME;
}
```

Ficheiro noncanonical.c (recetor)

```
/*Non-Canonical Input Processing*/


#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#include "alarm.h"
#include "link_layer.h"
#include "utils.h"
#include "app_layer.h"

int main(int argc, char** argv) {



        LinkLayer *link_layer = malloc(sizeof(LinkLayer));


  struct sigaction sa;
  sa.sa_flags = 0;
  sa.sa_handler = atende;
  if (sigaction(SIGALRM, &sa, NULL) == -1) {
```

```
  perror("Error: cannot handle SIGALRM");
  return 0;
 }

 if ( (argc < 2) ||
              (strcmp("/dev/ttyS0", argv[1])!=0 &&
              (strcmp("/dev/ttyS4", argv[1])!=0))) {
  printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n");
  exit(1);
 }

 ll_init(link_layer, argv[1], BAUDRATE, 1, 5, 1000, RECEIVER);

 app_layer(link_layer, 0);

 fprintf(stderr, "Saiu da app_layer. Encerrar a link_layer\n");

 ll_end(link_layer);

        return 0;
}
```

Ficheiro writenoncanonical.c (emissor)

```
/*Non-Canonical Input Processing*/


#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#include "utils.h"
#include "alarm.h"
#include "link_layer.h"
#include "app_layer.h"

int main(int argc, char** argv) {
  LinkLayer *link_layer = malloc(sizeof(LinkLayer));

  struct sigaction sa;
  sa.sa_flags = 0;
  sa.sa_handler = atende;
  if (sigaction(SIGALRM, &sa, NULL) == -1) {
    perror("Error: cannot handle SIGALRM");
    return 0;
```

```
  }

  if ( (argc < 3) ||
                (strcmp("/dev/ttyS0", argv[1])!=0 &&
                (strcmp("/dev/ttyS4", argv[1])!=0))) {
    printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n");
    exit(1);
  }

  ll_init(link_layer, argv[1], BAUDRATE, 1, 5, 1000, TRANSMITTER);

  app_layer(link_layer, argv[2]);

  ll_end(link_layer);

  return 0;
}
```

Ficheiro utils.h

```
#ifndef ___UTILS

#define ___UTILS


#define BAUDRATE B9600

#define MODEMDEVICE "/dev/ttyS1"

#define _POSIX_SOURCE 1 /* POSIX compliant source */

#define FALSE 0

#define TRUE 1


#define FLAG 0x7E

#define ESC 0x7D

#define SUBS 0x20


#define F 0x7E

#define FLAG 0x7E

#define A 0x03

#define C_SET 0x07

#define BCC_SET (A^C_SET)

#define BCC_UA (A^C_UA)

#define BCC_DISC (A^C_DISC)

#define A_RECEPTOR 0x01


#define START 0

#define FLAG_RCV 1

#define A_RCV 2

#define C_RCV 3
```

```
#define BCC_OK 4

#define STOP_ST 5

#define ESC 0x7D

#define SUBS 0x20


#define ATTEMPTS 4

#define TIME_OUT 3


#define C_UA 0x03


#define C_DISC 0x0B


#define RECEIVER 0

#define TRANSMITTER 1


#define FAILED -1

#define RE_SEND_RR -2

#define RE_SEND_SET -3



#endif /* UTILS */
```