**Faculdade de Engenharia da Universidade do Porto**



1º Trabalho Laboratorial de Redes de Computadores:

Protocolo de Ligação de dados

**9 de novembro de 2016**

**Turma 7, Grupo 5:**

Andreia Rodrigues - [up201404691@fe.up.pt](mailto:up201404691@fe.up.pt)

Eduardo Leite - [gei12068@fe.up.pt](mailto:gei12068@fe.up.pt)

Gonçalo Leão - [up201406036@fe.up.pt](mailto:up201406036@fe.up.pt)

Francisco Queirós - [up201404326@fe.up.pt](mailto:up201404326@fe.up.pt)

# 

# 

# 

# **Sumário**

No âmbito da unidade curricular de Redes de Computadores, foi-nos proposto como primeiro trabalho laboratorial o desenvolvimento de uma aplicação que corre em duas máquinas, ligadas por cabo pelas suas portas série, e que seja capaz de transferir ficheiros. Para isso, o nosso programa deve implementar, a mais baixo nível, um protocolo de ligação de dados, e, a mais alto nível, um protocolo de aplicação. As especificações de ambos os protocolos são descritas no guião do trabalho fornecido pelos docentes da unidade curricular.

Como principais conclusões, é do entendimento dos membros do grupo que este trabalho permitiu-nos expandir os nossos horizontes enquanto engenheiros informáticos, ao pôr em prática diversos conceitos abordados nas aulas teóricas desta disciplina, nomeadamente, a implementação de uma pilha de protocolos, e a construção e processamento de tramas e pacotes.

# **Índice**

[**Sumário**](#_bl62uwdgp9z5)

[**Índice**](#_pavhkiez8t0c)

[**Introdução**](#_xx07k0wmuh35)

[**Arquitetura**](#_oqb3abpues2w)

[**Estrutura do código**](#_ozk9l9wxlk3f)

[**Casos de uso principais**](#_jiwwc9v6kp6z)

[**Protocolo de ligação lógica**](#_51j0wfgj2u63)

[**Protocolo de aplicação**](#_rwkkcof7lonf)

[**Elementos de valorização**](#_lun20uq9b7l)

[**Seleção de parâmetros pelos utilizador**](#_ntirhae77on)

[**Geração aleatória de erros nas tramas de Informação**](#_ep86pm24eh2y)

[**Implementação de REJ**](#_llej9pg7okkk)

[**Verificação da integridade de dados pela Aplicação**](#_dwsdlmlx4x8k)

[**Registo de ocorrências**](#_lvab09uqtqz5)

[**Validação**](#_ghrq8bew1gec)

[**Conclusões**](#_w38uzdmr9xoe)

[**Anexo 1: Grafo de chamada de funções**](#_5reijx3mk1ug)

# 

# **Introdução**

O primeiro trabalho laboratorial da unidade curricular de Redes de Computadores, do primeiro semestre do terceiro ano do Mestrado Integrado em Engenharia Informática e Computação, teve como objetivo a elaboração de uma programa, em linguagem C e em ambiente de Linux, capaz de transferir ficheiros entre duas máquinas. Para isso, o programa devia correr em duas máquinas, ligadas por cabo pelas portas série (de modelo RS-232) e comunicando usando transmissão serial assíncrona.

O programa deve distinguir duas principais camadas, ligação de dados e aplicação, cujas especificações estão detalhadas no guião do trabalho.

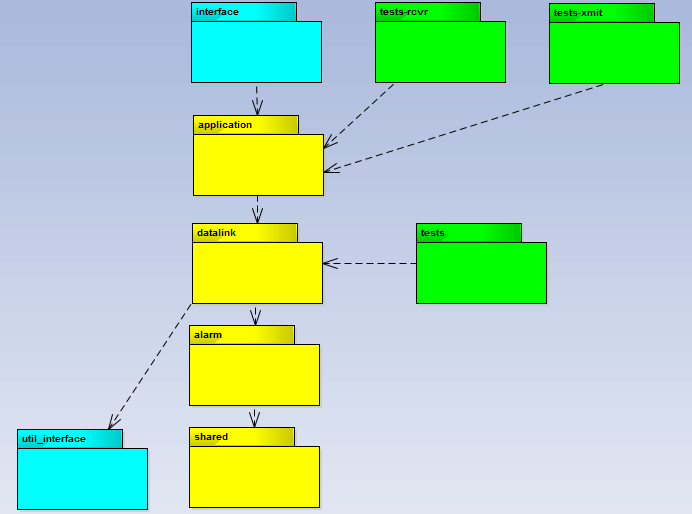
O objetivo deste relatório é apresentar, de forma sintetizada, a abordagem adotada para o desenvolvimento do programa. Assim, este documento também tem como objetivo mostrar que o grupo domina tanto a parte prática como a parte de cariz mais teórica deste trabalho laboratorial.

O seguimento deste relatório subdivide-se nas seguintes partes:

* **Arquitetura**, onde são identificados os blocos funcionais do programa, e as suas interações e dependências.
* **Estrutura do código**, onde é descrito a *Application Programming Interface* (API), as principais estruturas de dados e funções, e as sequências de chamada de funções.
* **Casos de uso principais**.
* **Protocolo de ligação lógica**, onde são identificados os seus principais aspetos funcionais e a sua estratégia de implementação.
* **Protocolo de aplicação**, onde são identificados os seus principais aspetos funcionais e a sua estratégia de implementação.
* **Elementos de valorização**, onde é descrito a sua estratégia de implementação.
* **Validação**, onde são apresentados os testes efetuados.
* **Conclusões**.

# **Arquitetura**

No diagrama abaixo, pode-se ver o diagrama UML de *packages* da programa, onde cada *package* representa um módulo/bloco funcional (associado a um ficheiro .c e, eventualmente, a um ficheiro .h). As setas representam as dependências entre módulos.



*Figura 1: Diagrama UML de “packages” do programa*

Os módulos a amarelo representam os blocos de mais baixo nível que permitem a transmissão de ficheiros.

Os módulos a azul representam os ficheiros usados para o interface com a linha de comandos.

Os módulos a verde representam os ficheiros de testes.

A tabela seguinte descreve a responsabilidade de cada módulo.

|  |  |
| --- | --- |
| **Módulo** | **Descrição** |
| shared | Módulo elementar com certas *macros* e funções usados pelos restantes módulos, como a *macro* “DEBUG\_MSGS”, um booleano que indica se o programa deve correr em modo *debug* (e imprimir linhas usadas para testar o código) ou em modo de utilizador (com o *interface* completo). |
| alarm | Módulo usado para a gestão do alarme, essencial para a implementação do mecanismo de *time-out*. |
| datalink | Módulo com a camada de ligação de dados, que fornece um serviço de comunicação fiável entre as duas máquinas, ligadas usando a porta série. |
| application | Módulo com a camada de aplicação, que fornece um serviço de comunicação de mais alto nível para o envio de ficheiros completos. |
| util\_interface | Módulo com *macros* e funções utilitárias para leitura e escrita na consola, nomeadamente as *macros* que permitem mudar a cor do texto e de fundo. |
| interface | Módulo de *user interface*, que permite a interação do utilizador com a aplicação. |
| tests | Módulo de testes de mais baixo nível usado para exercitar certas funções que não envolvem a comunicação entre duas máquinas, como o *stuffing*/*destuffing* de tramas. |
| tests\_xmit | Módulo de testes de mais alto nível usado para exercitar certas funções dos módulos “datalink” e “application” que envolvem a comunicação entre as duas máquinas, nomeadamente para a transferência de uma *string* e de um ficheiro. Este módulo testa a comunicação do lado do transmissor. |
| tests\_rcvr | Módulo semelhante a “tests\_xmit” mas que testa a comunicação do lado do recetor. |

*Tabela 1 - Responsabilidade de cada módulo da aplicação*

# **Estrutura do código**

A camada de ligação lógica dispõe de uma API composta pelas funções “initDataLinkLayer”, “llopen”, “llclose”, “llread” e “llwrite”. A camada de ligação lógica é centrada à volta de uma struct, “LinkLayerInfo”, com as configurações globais da camada:

**typedef struct**

**{**

**char port[PORT\_NAME\_SIZE];**

**tcflag\_t baudrate;**

**size\_t frameSize;**

**unsigned char seqNum;**

**unsigned int timeout;**

**unsigned int maxTries;**

**ConnectionMode mode;**

**double byteErrorRatio;**

**} LinkLayerInfo;**

A camada da aplicação dispõe, similarmente, de uma API composta pelas funções “initApplicationLayer”, “receiveFile” e “sendFile”. Esta camada é centrada à volta de uma struct “ApplicationLayerInfo”:

**typedef struct**

**{**

**int fd;**

**int port;**

**char fileName[256];**

**unsigned int fileSize;**

**ConnectionMode mode;**

**unsigned int seqNum;**

**size\_t packetSize;**

**} ApplicationLayerInfo;**

As funções referidas nesta secção são faladas em mais detalhe nas secções respectivas.

É disponibilizada uma imagem nos anexos com o grafo de chamada de funções.

# **Casos de uso principais**

A aplicação inclui como principais casos de uso:

* Definir os parâmetros de comunicação: a partir do menu principal, o utilizador pode alterar os seguintes campos:
  + a porta série a usar (/dev/ttySx, onde x é um inteiro de 0 a 2).
  + *o baudrate*,
  + o tamanho máximo para as tramas de tipo I (informação),
  + o intervalo de *time-out* em segundos,
  + o número máximo de tentativas de retransmissão de tramas,
  + o *byte error ratio* a usar para simular erros nas tramas recebidas.

Ao correr o programa, cada um destes parâmetros é inicializado com um valor por omissão.

* Enviar um ficheiro: o utilizador indica o nome do ficheiro a enviar, e, quando a máquina estabelecer comunicação com outra máquina que esteja pronta para receber os dados, o ficheiro é enviado.
* Receber um ficheiro: o programa fica à espera que seja estabelecida comunicação com outra máquina e guarda o ficheiro que lhe está a ser enviado.

# **Protocolo de ligação lógica**

Na especificação desta camada, destaca-se:

* o envio/receção de diferentes tipos de trama (tramas de tipo informação, supervisão e não-numeradas)
* o mecanismo de detecção de erros nas tramas usando bits de paridade
* o mecanismo de *Stop and Wait* para retransmissão automática de tramas decidida pelo emissor (*time-out*) ou recetor (tramas REJ)
* o *stuffing* ou *destuffing* das tramas enviadas ou recebidas, respectivamente

Apresenta-se abaixo os cabeçalhos das funções mais relevantes desta camada:

**int initDatalinkLayer(tcflag\_t baudrate, size\_t frameSize, unsigned int timeout, unsigned int maxTries, double byteErrorRatio);**

**int llopen(int port, ConnectionMode mode);**

**int llwrite(int fd, char \*buffer, size\_t length);**

**ssize\_t llread(int fd, char \*\*buffer);**

**int llclose(int fd);**

**unsigned int stuff(char \*\*frame, unsigned int frameSize);**

**unsigned int destuff(char \*\*frame, unsigned int frameSize);**

# **Protocolo de aplicação**

Na especificação desta camada, destaca-se:

* o envio/receção de diferentes tipos de pacote (pacote de controlo e de dados)
* o envio/receção de ficheiros completos

Apresenta-se abaixo os cabeçalhos das funções mais relevantes desta camada:

**int initApplicationLayer(int port, tcflag\_t baudrate, size\_t frameSize, unsigned int timeout, unsigned int maxTries, double byteErrorRatio);**

**int receiveFile();**

**int sendFile(char \*fileName);**

# **Elementos de valorização**

## Seleção de parâmetros pelos utilizador

O utilizador tem a capacidade de configurar vários aspectos do programa, tal como já referido na secção “Casos de uso principais”.

## Geração aleatória de erros nas tramas de Informação

Na camada de ligação lógica, ao receber tramas, o programa insere bytes aleatórios nos dados recebidos com uma probabilidade definida pelo utilizador, usando a interface.

**void insertRandomErrors(char \*buffer, size\_t length)**

**{**

**unsigned char \*bufferToUnsigned = (unsigned char \*) buffer;**

**size\_t i;**

**for (i = 0; i < length; i++)**

**{**

**int randomResult;**

**randomResult = rand();**

**if (randomResult <= (llInfo.byteErrorRatio \* RAND\_MAX))**

**{**

**randomResult = rand() % 256;**

**bufferToUnsigned[i] = (unsigned char) randomResult;**

**}**

**}**

**}**

## Implementação de REJ

O receptor responde ao emissor com uma trama do tipo REJ quando ocorre um erro na validação da informação que uma trama I carrega usando o byte BCC2, sendo a trama não duplicada.

**case BAD:**

**if (msg.error == BAD\_BCC2)**

**{**

**llInfo.seqNum = msg.seqNum;**

**if (sendSUFrame(fd, REJ) == -1)**

**{**

**if (DEBUG\_MSGS)**

**{**

**printf("llread: I frame had BCC2 error, failed to send REJ frame.\n");**

**}**

**return -1;**

**}**

**...**

**}**

**break;**

## Verificação da integridade de dados pela Aplicação

É feita a verificação do tamanho do ficheiro por parte do receptor:

**while (nWrote < alInfo.fileSize)**

**{**

**if (-1 == receiveDPacket(&nReceived, &fileBuf))**

**{**

**...**

**}**

**...**

**}**

É também feito o controle de pacotes duplicados ou perdidos:

**unsigned char N = packet[1];**

**if (N > alInfo.seqNum)**

**{**

**return -1;**

**}**

**else if (N < alInfo.seqNum)**

**{**

**return -2;**

**}**

## Registo de ocorrências

É feito o registo do número de tramas/pacotes enviados e recebidos, dos seus tipos e de quando ocorrem erros (por exemplo, no campo BCC1 de tramas).

A título de exemplo, apresenta-se abaixo a *struct* usada para guardar as estatísticas da camada da aplicação (existe uma *struct* equivalente para a camada de ligação lógica):

**typedef struct**

**{**

**unsigned int CPacketsSent;**

**unsigned int CPacketsReceived;**

**unsigned int DPacketsSent;**

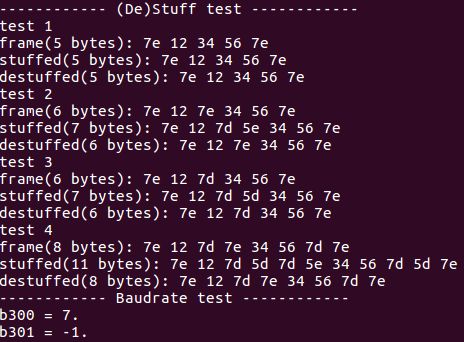
**unsigned int DPacketsReceived;**

**} ApplicationLayerStats;**

# **Validação**

Para começar, foram feitos testes mais elementares que não envolvem a comunicação entre as duas máquinas. No ficheiro tests.c, foram exercitadas dois funcionalidades: o *stuffing*/*destuffing* de tramas e a conversão de um inteiro (“unsigned int”) com o valor do *baudrate* na *flag* de controle (“tcflag\_t”) correspondente (usado para configurar a porta série). Estas duas funcionalidades foram implementadas com as funções “stuff”/”destuff” e “getBaudrate” (que devolve -1, caso não exista uma *flag* de controle para o valor de *baudrate* especificado) do módulo “datalink”, respetivamente.

A figura abaixo apresenta o resultado de alguns dos testes realizados. A impressão do conteúdo de tramas (e pacotes) foi conseguida através da função “printByteByByte” do módulo “shared”.



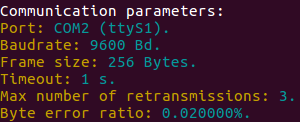
*Figura 2: Alguns resultados dos testes realizados no módulo “tests”*

De seguida, os módulos “tests-xmit” e “tests-rcvr” foram criados para exercitar, de forma incremental, a comunicação entre duas máquinas, do lado do emissor e recetor, respetivamente. Primeiro, testou-se apenas a chamada de “llopen” seguido de “llclose” do módulo “datalink”. Depois, testou-se o envio de apenas uma *string*. Por fim, testou-se a invocação de “sendFile” e “receiveFile” do módulo “application” para o envio do ficheiro de imagem de um pinguim, fornecido pelos docentes.

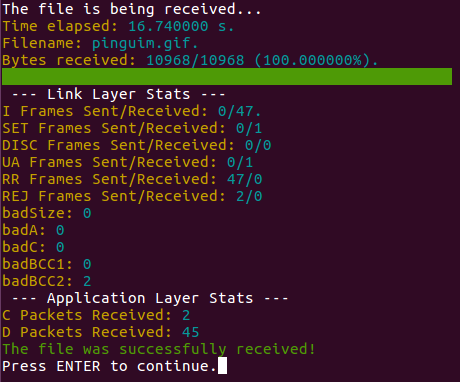
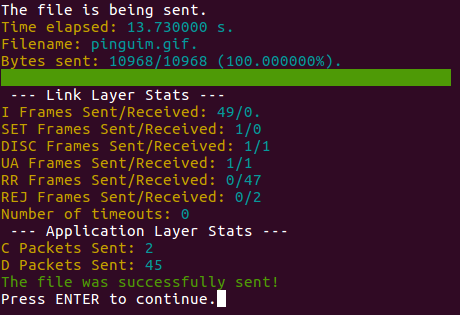
Para terminar, testou-se a aplicação diretamente pelo módulo “interface” onde foram enviados com sucesso ficheiros de diferentes tipos (imagem, música, vídeo…).

Para a demonstração final deste trabalho, conseguiu-se enviar o ficheiro da imagem do pinguim através do cabo série, tanto quando se cortava a comunicação a meio do envio como quando se interferia com o sinal com a ajuda de um dispositivo presente nos laboratórios de redes de computadores do edifício I da FEUP.

Na figura abaixo, apresentam-se os parâmetros de comunicação e o resultado da transferência do ficheiro da imagem do pinguim, sem o uso do dispositivo para corte ou interferência com o sinal, mas usando o gerador de erros aleatórios nas tramas.



*Figura 3: Parâmetros de comunicação usados no teste de envio do ficheiro do pinguim*



*Figuras 4/5: Resultados do teste de envio/receção do ficheiro do pinguim*

# **Conclusões**

Este primeiro trabalho prático de Redes de Computadores exigiu bastante esforço, tempo, paciência e trabalho de equipa para cumprir com as suas metas. Houve partes do projeto que se revelaram particularmente desafiantes, nomeadamente, a construção e processamento de tramas e pacotes, e a implementação de uma camada de ligação de dados com a robustez necessária para garantir à camada da aplicação fiabilidade na transmissão de dados.

Relativamente ao enunciado, o grupo acredita que todos as metas e especificações do guião foram bem entendidas e implementadas.

O programa está dividido em duas principais camadas: ligação de dados e aplicação, sendo que a primeira camada, de mais baixo nível, fornece serviços à segunda camada.

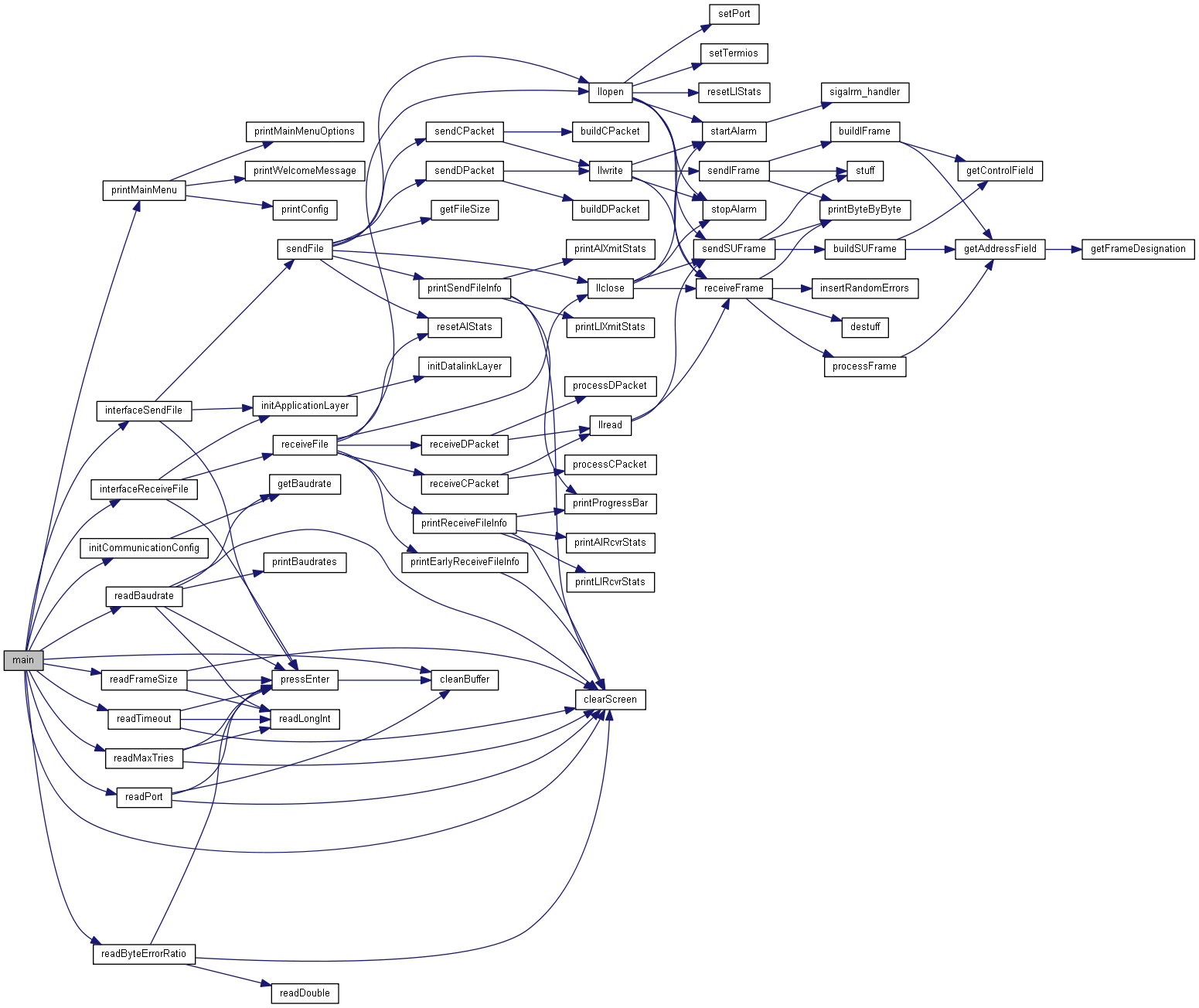
O programa cumpre o seu caso de uso fundamental ao permitir a transferência de ficheiros. A interface implementada é intuitiva e informativa, contribuindo assim para a simplicidade da aplicação.

Em termos de elementos de valorização, todos os pontos foram implementados: registo de ocorrências, uso de tramas do tipo REJ, seleção de parâmetros pelo utilizador, geração aleatória de erros nas tramas e verificação da integridade dos dados por parte da camada de aplicação.

Os testes efetuados para a verificação e validação do programa foram bastante úteis pois permitiram identificar alguns *bugs*, e, de forma mais geral, avaliar a robustez do programa.

O grupo considera que foram cumpridos os objetivos deste trabalho, dado que os membros da equipa estão agora mais familiarizados, tanto a nível teórico como prático, com diversos aspetos essenciais de redes de computadores como a construção de uma pilha de protocolos e o uso de tramas e pacotes, para transferir os dados relevantes para cada camada do programa.

# **Anexo 1: Grafo de chamada de funções**



*Figura 6: Grafo de chamada de funções*