

# Comunicações por Computadores

TRABALHO PRÁTICO Nº2 – TRANSFERÊNCIA RÁPIDA E FIÁVEL DE MÚLTIPLOS SERVIDORES  
EM SIMULTÂNEO

FÁBIO LEITE, RUI RIBEIRO, RUI ALVES - GRUPO 06(G16)  
UNIVERSIDADE DO MINHO, DEPARTAMENTO DE INFORMÁTICA  
E-MAIL: {A100902, A100816, A100699}@ALUNOS.UMINHO.PT

# 1. Introdução

O objetivo deste trabalho foi desenhar, implementar e testar um serviço de partilha de ficheiros peer-to-peer, sendo a transferência baseada num protocolo de transferência a correr sobre o protocolo de transporte UDP, enquanto que a comunicação cliente-servidor tem por base o protocolo de transporte TCP.

Foi com este objetivo que o grupo partiu para a elaboração do trabalho prático em causa. Dada a liberdade da plataforma sobre o qual o trabalho seria executado, o grupo decidiu trabalhar em Java, visto que já tínhamos alguma experiência a trabalhar com esta linguagem, e os módulos que esta plataforma fornece simplificou o trabalho em certos aspetos.

Nos capítulos seguintes vamos explorar mais a fundo as escolhas feitas pela nossa parte, assim como explicar o funcionamento do nosso protocolo.

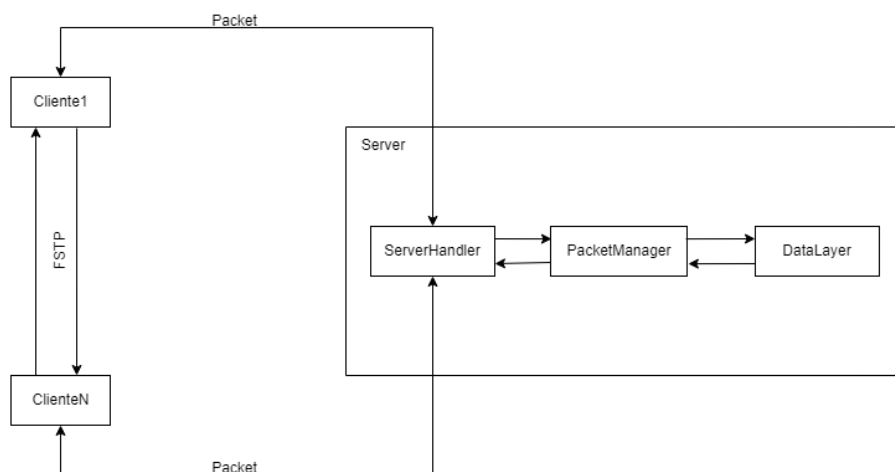
## 2. Arquitetura da solução

Quando é registado um novo cliente(FSNode) o mesmo envia uma mensagem ao servidor o servidor o registar na sua base de dados. Este packet enviado leva consigo o tipo de query que vai ser executada e as informações necessárias para o registo do cliente. Relativamente à atualização da informação de um dado nodo, o mesmo processo acontece, embora as informações enviadas no packet sejam diferentes.

Dentro do servidor, decidimos dividir o mesmo em várias partes independentes, que executam a função do servidor. Dividimos então o Servidor em:

- **ServerHandler** : trata de receber as mensagens e trata-as de forma a que o PacketManager consiga tratar da mensagem na forma de um packet.
- **PacketManager**: recebe um packet, chama os métodos da DataLayer e trata de devolver um packet com a resposta a query pedida.
- **DataLayer**: funciona como a base de dados do servidor, guardando a informação tanto dos nodes como dos ficheiros em HashMap's.

Para a obtenção dos ficheiros, a comunicação é feita entre os diferentes Clientes, após obterem do Servidor a localização dos ficheiros que eles pretendiam encontrar. A comunicação entre os Clientes é feita através de FstpPackets sendo as ligações feitas através do protocolo UDP.



**Figura 2.1:** Arquitetura da Solução

## 3. Especificação do protocolo

### 3.1 Formato das mensagens protocolares

Como apresentamos dois protocolos de transferência de dados(TCP e UDP), consequentemente também vamos apresentar dois formatos distintos das mensagens protocolares.

#### FS Track Protocol

- **ID:** identificador unico do pacote, atribuido aleatoriamente.
- **TYPE (Request, Response):** define se o pacote é uma resposta ou um pedido.
- **QUERY(Get, Register, Update, FileInfo):** indica a natureza da operação a ser realizada pelo pacote.
- **CONTENT:** possui o conteudo do pacote.

```
Packet register = new Packet(Packet.Type.REQUEST, Packet.Query.REGISTER, node.toString());
```

Figura 3.1: Exemplo da criação de um pacote de registro.

No diagrama temporal abaixo, é possível observarmos um exemplo de uma comunicação entre o cliente e o servidor, quando o cliente deseja efetuar o seu registro.

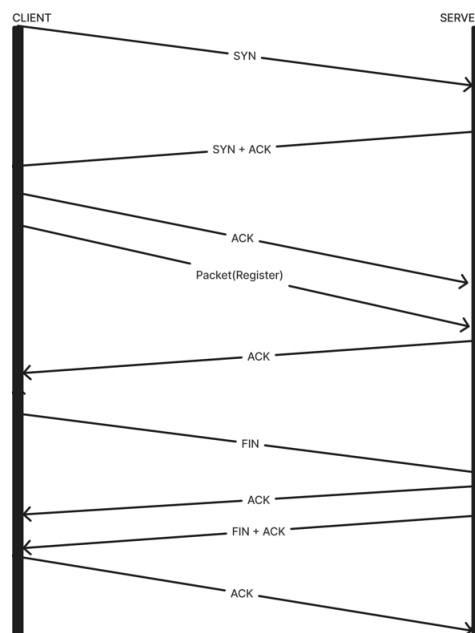


Figura 3.2: Diagrama temporal de um pacote de registro.

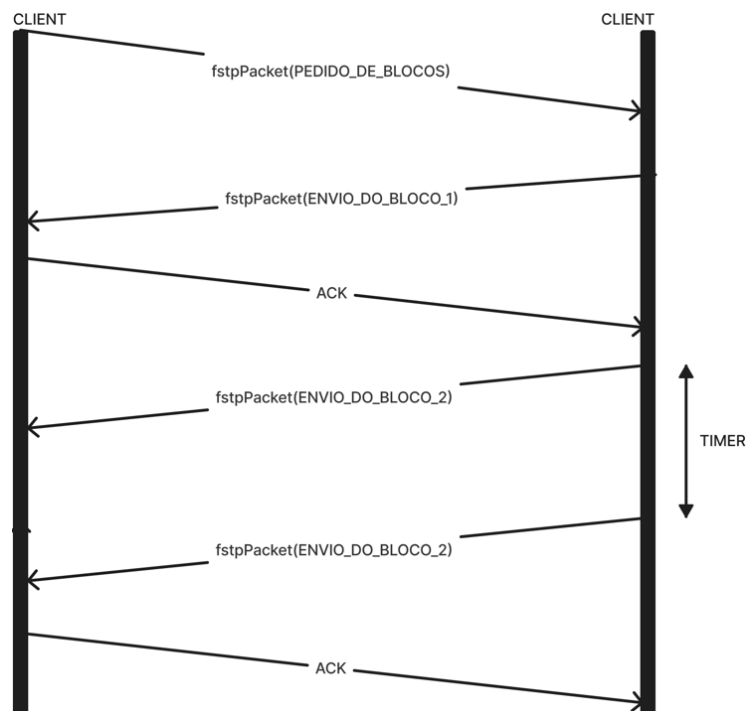
## FS Transfer Protocol

- **Header** - contém informação que permite a comunicação entre FSNodes. Possui 32 bytes e é representado por:
  - **TYPE (1,2,3)**: equivalente a BLOCK\_REQUEST, BLOCK\_SEND e BLOCK-CONFIRMATION. (4 bytes)
  - **ID**: identificador unico do pacote. (4 bytes)
  - **Client IP**: indica quem enviou o pacote. (4 bytes)
  - **DataSize**: tamanho da DATA enviada. (4 bytes)
  - **FileName**: nome do ficheiro que está a ser tratado no pacote. (12 bytes)
  - **BlockNumber**: número do bloco do ficheiro enviado. (4 bytes)
  - **TotalBlocks**: número total de blocos do ficheiro a ser transferido. (4 bytes)
- **Data** - representa o conteúdo que o pacote possui, com 256 bytes no seu limite.

```
Fstp responsePacket = new Fstp(blockContent, type 2, node.getClient().toString(), fileName, blockNumber, totalBlocks);
```

**Figura 3.3:** Exemplo da criação de um pacote de envio de um bloco de um ficheiro.

No diagrama temporal abaixo, é possível observarmos um exemplo de uma comunicação entre dois clientes, quando um cliente efetua um pedido de um bloco a outro cliente. De realçar também, temos a perda do bloco 2 do ficheiro e, conseqüentemente, o reenvio do mesmo após um tempo de espera, com um limite de 3 reenvios por bloco.



**Figura 3.4:** Diagrama temporal de um pacote de envio de um bloco de um ficheiro.

## 3.2 Interações

No nosso projeto são possíveis observar dois tipos de interações, sendo elas:

### 3.2.1 FSTracker entre FSNode

Relativamente à interseção entre o Servidor e o Cliente, o FSNode registra-se ao FSTracker através do envio de um pacote TCP com o valor da query sendo *REGISTER*. Posteriormente, o Servidor armazena na base de dados as informações básicas do cliente.

```
root@Portatil1:/home/core/Desktop# java -jar client.jar fs file 10.4.4.1 10.1.>
register
FSNode registado com sucesso!
```

**Figura 3.5:** Registo de um cliente.

O Cliente pode ainda, através do envio de um pacote TCP com o valor da query sendo *UPDATE*, enviar uma atualização da sua pasta, com os ficheiros que possui, ao Servidor, para este armazenar essa informação na sua base de dados.

```
update
FSTracker atualizado com sucesso!
```

**Figura 3.6:** Update do FSTracker.

É possível ainda, o Cliente, através do envio de um pacote TCP com o valor da query sendo *GET*, pedir ao Servidor a localização de um determinado ficheiro. Ao invocarmos esta query e caso obtenhamos uma resposta positiva, é enviado ao Servidor um pacote com o valor da query sendo *FileInfo*, onde o Cliente vai receber o tamanho do ficheiro que está a pedir para posteriormente efetuar a divisão desse ficheiro em blocos.

```
get test.txt
Arquivo completo recebido e salvo em: files2/test.txt
```

**Figura 3.7:** Get de um ficheiro.

### 3.2.2 FSNode entre FSNode

Relativamente à interação entre dois clientes, esta acontece apenas quando um cliente tem um ficheiro que outro cliente pediu. O cliente que pede o ficheiro, envia ao cliente que possui esse ficheiro uma lista de blocos que deseja receber, através do envio de um pacote UDP com o valor do type sendo *BLOCK\_REQUEST*.

Posteriormente, o cliente que recebe esse pedido, envia através um pacote UDP com o valor do type sendo *BLOCK\_SEND*, os blocos para o cliente que efetuou o pedido. De realçar que o cliente que está a efetuar o envio dos blocos, só envia um próximo quando receber um pacote UDP com o valor do type sendo *BLOCK\_CONFIRMATION*., do cliente que efetuou o pedido em questão.

```
Enviado Block Number: 0 ao node: /10.1.1.3.
Enviado Block Number: 1 ao node: /10.1.1.3.
```

**Figura 3.8:** Envio de blocos de um ficheiro a um cliente.

## 4. Implementação

### 4.1 Files

Ao longo da realização do projeto, adotamos a estratégia de criar uma classe chamada *FileInfo* que vai guardar, toda a informação que um ficheiro possui. Essa classe é representada pelo nome do ficheiro, pelo tamanho do ficheiro e por uma lista de *FileBlocks*. A classe *FileBlocks* é representada pelo número do bloco em questão, o tamanho desse bloco e o conteúdo que ele possui.

### 4.2 Client

#### 4.2.1 ClientInfo

Ao iniciar o Client, uma instância do objeto *ClientInfo* é criada com base nas suas informações, sendo elas:

- **ID:** identificador unico do cliente, atribuído aleatoriamente.
- **FileFolder:** pasta onde vão ficar guardados todos os ficheiros que o cliente possui.
- **pathFolder:** o caminho para a pasta do cliente.
- **ipClient:** Endereço IP do cliente.

#### 4.2.2 ClientHandler e UDPClientHandler

As classes ClientHandler e UDPClientHandler são responsáveis pela comunicação entre servidor-cliente e cliente-cliente, respetivamente. O UDPClientHandler utiliza o protocolo de comunicação UDP. Esta classe opera em uma thread, aguardando a receção de pacotes UDP e gestão das operações associadas à transferência de ficheiros. Já o ClientHandler utiliza o protocolo de comunicação TCP. Esta classe também opera em uma thread, tratando de toda as mensagens trocadas entre o cliente e o servidor.

### 4.3 Server

O Server desempenha um papel fundamental no projeto. É nele que são tratadas todas as mensagens enviadas pelos clientes, através do packetManager. Possui ainda uma base de dados, o DataLayer, onde fica guardada toda a informação de nodes e ficheiros.

### 4.4 Algoritmo de divisão em blocos

Primeiramente é perguntado ao FSTracker(Servidor) o tamanho do ficheiro. Após a obtenção desse valor, iremos dividi-lo por 256 (tendo em conta que definimos que 256 Bytes é tamanho máximo que cada bloco pode ter) para saber a quantidade de blocos que esse ficheiro possui.

Após isso é criado 'HashMap <IP, LIST(INT)>' em que atribuímos a cada IP uma lista com os id's dos blocos do ficheiro que lhe vão ser pedidos. Essa atribuição é feita de forma sequencial em que, por exemplo, se existem dois clientes com o ficheiro(4 blocos) pedido, é atribuído ao primeiro cliente os dois primeiros blocos e ao segundo cliente os últimos dois blocos. Em caso de numero ímpar de blocos, é atribuído da mesma forma esses mesmos blocos.

### 4.5 Mecanismo Stop & wait

O mecanismo Stop-and-Wait é uma técnica fundamental na comunicação entre nós de uma rede para garantir a entrega fiável dos dados. No contexto do nosso sistema P2P, implementamos esse mecanismo para a transferência de blocos de ficheiros entre nós clientes.

O funcionamento do mecanismo envolve a transmissão de um bloco de dados seguido de uma espera pela confirmação (ACK) do recetor antes que o próximo bloco seja enviado. Essa abordagem garante que cada bloco seja recebido e reconhecido antes do envio do próximo, proporcionando um controle eficaz do fluxo de dados.

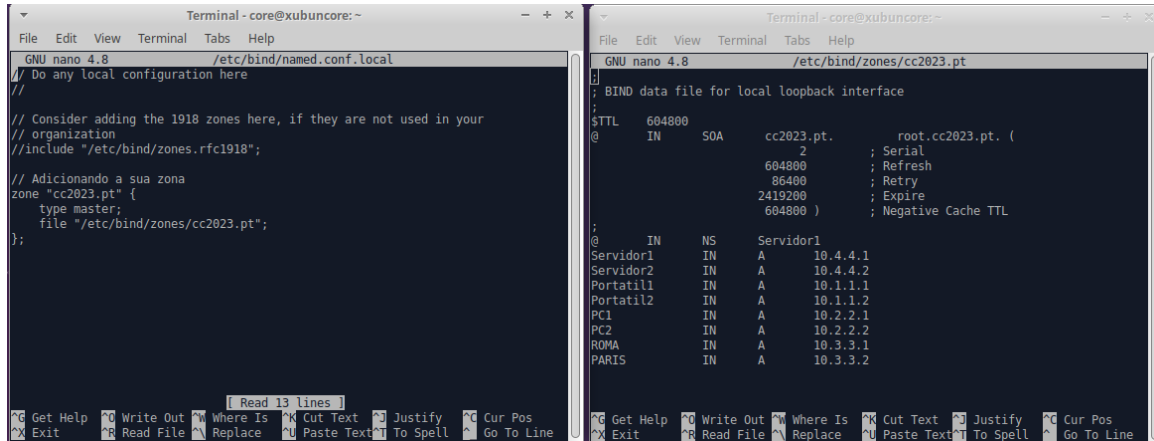
```
Enviado Block Number: 10 ao node: /10.1.1.1.  
Enviado Block Number: 11 ao node: /10.1.1.1.  
Falha ao enviar o bloco 12 para o node: /10.1.1.1.  
Retransmitindo...  
Enviado Block Number: 12 ao node: /10.1.1.1.  
Enviado Block Number: 13 ao node: /10.1.1.1.  
Enviado Block Number: 14 ao node: /10.1.1.1.  
Enviado Block Number: 15 ao node: /10.1.1.1.  
Enviado Block Number: 16 ao node: /10.1.1.1.  
Falha ao enviar o bloco 17 para o node: /10.1.1.1.  
Retransmitindo...  
Enviado Block Number: 17 ao node: /10.1.1.1.
```

**Figura 4.1:** Exemplo do funcionamento do mecanismo.



## 4.6 DNS

O DNS desempenha um papel crucial no sentido de traduzir nomes de domínio em endereços IP, simplificando a navegação e a comunicação na rede. As configurações foram realizadas no arquivo principal da configuração, localizado em `/etc/bind/named.conf.local`, e as informações específicas do domínio foram organizadas no arquivo `/etc/bind/zones/cc2023.pt`.



```
Terminal - core@xubuncore: ~
GNU nano 4.8 /etc/bind/named.conf.local
// Do any local configuration here
//
// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";
// Adicionando a sua zona
zone "cc2023.pt" {
    type master;
    file "/etc/bind/zones/cc2023.pt";
};

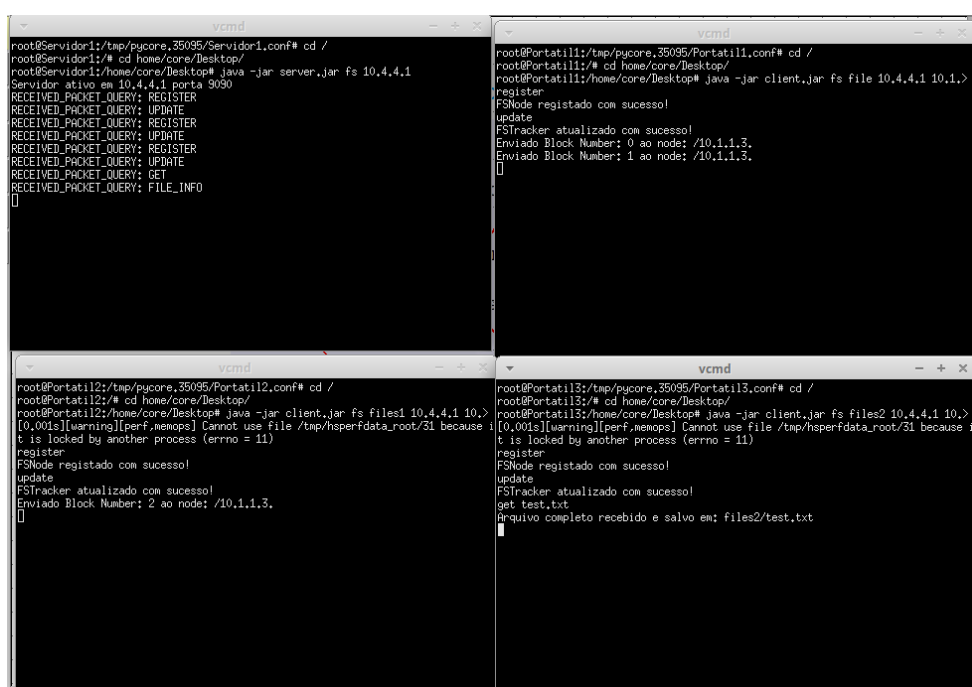
Terminal - core@xubuncore: ~
GNU nano 4.8 /etc/bind/zones/cc2023.pt
; BIND data file for local loopback interface
;
$TTL 604800
@ IN SOA cc2023.pt. root.cc2023.pt. (
    2 ; Serial
    604800 ; Refresh
    86400 ; Retry
    2419200 ; Expire
    604800 ) ; Negative Cache TTL
;
@ IN NS Servidor1
Servidor1 IN A 10.4.4.1
Servidor2 IN A 10.4.4.2
Portatil1 IN A 10.1.1.1
Portatil2 IN A 10.1.1.2
PC1 IN A 10.2.2.1
PC2 IN A 10.2.2.2
ROMA IN A 10.3.3.1
PARIS IN A 10.3.3.2
```

**Figura 4.2:** Configuração do `named.conf.local` e `cc2023.pt`.

A redundância do DNS é um aspecto de grande importância. É importante a existência de mais do que um servidor DNS para cada domínio, garantindo que o serviço DNS seja confiável e esteja disponível em caso de falha de um dos servidores. O servidor DNS primário atua como um DNS autoritativo, sendo a partir dele que os administradores gerem os arquivos de zona e realizam alterações no DNS, como adicionar, excluir e atualizar registros de DNS. O DNS secundário guarda uma cópia idêntica das informações presentes no DNS primário. Os arquivos presentes no DNS primário são sincronizados com o DNS secundário por meio de uma transferência de zona.

## 5. Testes e resultados

Na figura seguinte é representado um teste com 3 clientes (FSNodes) e um servidor (FSTracker). Como é possível observar na figura, cada nodo é inicializado com o **register**, e de seguida é feito um **update**, o que permite ao servidor ter conhecimento dos ficheiros que cada cliente tem. Por fim, no Portátil1 é feito um **get** que faz com que receba o ficheiro **text.txt** dividido em 3 blocos (recebendo 2 blocos do Portátil2 e 1 bloco do Portátil3).



```
vcmd
root@Servidor1:/tmp/pycore.35095/Servidor1.conf# cd /
root@Servidor1:/# cd home/core/Desktop/
root@Servidor1:/home/core/Desktop# java -jar server.jar fs 10.4.4.1
Servidor ativo em 10.4.4.1 porta 9090
RECEIVED_PACKET_QUERY: REGISTER
RECEIVED_PACKET_QUERY: UPDATE
RECEIVED_PACKET_QUERY: UPDATE
RECEIVED_PACKET_QUERY: REGISTER
RECEIVED_PACKET_QUERY: UPDATE
RECEIVED_PACKET_QUERY: GET
RECEIVED_PACKET_QUERY: FILE_INFO
[]

vcmd
root@Portatil1:/tmp/pycore.35095/Portatil1.conf# cd /
root@Portatil1:/# cd home/core/Desktop/
root@Portatil1:/home/core/Desktop# java -jar client.jar fs file 10.4.4.1 10.1.>
register
FSNode registado com sucesso!
update
FSTracker atualizado com sucesso!
Enviado Block Number: 0 ao node: /10.1.1.3.
Enviado Block Number: 1 ao node: /10.1.1.3.
[]

vcmd
root@Portatil2:/tmp/pycore.35095/Portatil2.conf# cd /
root@Portatil2:/# cd home/core/Desktop/
root@Portatil2:/home/core/Desktop# java -jar client.jar fs files1 10.4.4.1 10.>
[0.001s][warning][perf.memops] Cannot use file /tmp/hyperfdata_root/31 because i
t is locked by another process (errno = 11)
register
FSNode registado com sucesso!
update
FSTracker atualizado com sucesso!
Enviado Block Number: 2 ao node: /10.1.1.3.
[]

vcmd
root@Portatil3:/tmp/pycore.35095/Portatil3.conf# cd /
root@Portatil3:/# cd home/core/Desktop/
root@Portatil3:/home/core/Desktop# java -jar client.jar fs files2 10.4.4.1 10.>
[0.001s][warning][perf.memops] Cannot use file /tmp/hyperfdata_root/31 because i
t is locked by another process (errno = 11)
register
FSNode registado com sucesso!
update
FSTracker atualizado com sucesso!
get test.txt
Arquivo completo recebido e salvo em: files2/test.txt
[]
```

Figura 5.1: Teste com 3 nodos e um servidor

Na figura seguinte é apresentado o resultado do comando **nslookup**. Este resultado permite-nos ter conhecimento do funcionamento do DNS uma vez que, segundo o teste, ao Portátil1 corresponde o IP 10.1.1.1, como esperado.



```
root@Servidor1:/tmp/pycore.35095/Servidor1.conf# nslookup Portatil1.cc2023.pt
Server:      10.4.4.1
Address:     10.4.4.1#53
Name:       Portatil1.cc2023.pt
Address:    10.1.1.1
```

Figura 5.2: Teste com comando nslookup.

## 6. Conclusões e trabalho futuro

Ao longo do desenvolvimento deste projeto de partilha de arquivos P2P, enfrentamos uma série de desafios e implementamos soluções para abordar essas complexidades. Projetamos e implementamos um sistema que permite a transferência eficiente de arquivos entre nós em uma rede peer-to-peer. Utilizamos o protocolo FS Transfer Protocol (Fstp) para facilitar a comunicação entre os nós e garantir uma transferência confiável e eficiente de blocos de arquivos.

Uma das características fundamentais do nosso projeto é o mecanismo Stop-and-Wait, que implementamos para garantir que a transferência de blocos ocorra de maneira confiável. Este mecanismo permite que um cliente aguarde a confirmação (ACK) do recetor antes de enviar o próximo bloco.

Outro ponto importante do nosso projeto é a integração de um serviço de resolução de nomes (DNS), permitindo que os nós se identifiquem por nomes significativos em vez de endereços IP. Isso torna o sistema mais amigável e escalável, facilitando a identificação dos nós na rede. Relativo ao mesmo, conseguimos somente a configuração do sistema através do **bind9**, não conseguindo fazer a implementação do mesmo com as restantes funcionalidades projeto. Da mesma forma, embora tendo também noção da necessidade de implementação do DNS, também não foi possível a sua implementação.

Em suma, consideramos este projeto de grande importância, uma vez que o desenvolvimento do mesmo proporcionando insights valiosos sobre as complexidades e desafios associados à construção de sistemas P2P distribuídos.