



Comunicações por Computador 2023/24

TP2 – Transferência rápida e fiável de múltiplos servidores em simultâneo

Trabalho realizado por:

João Pedro da Rocha Rodrigues - a100896

João Pedro Mota Baptista - a100705

Índice

Introdução	2
Arquitetura da Solução	2
Especificação dos Protocolos Propostos	3
FS Track Protocol	3
FS Transfer Protocol.....	4
DNS – Domain Name System.....	5
Implementação	6
Testes e Resultados	7
Conclusões e Trabalho Futuro	10

Introdução

A partilha de ficheiros constitui um serviço fundamental em redes, permitindo a transferência fiável de dados entre clientes e servidores. Este serviço exige que os ficheiros sejam descarregados e armazenados sem erros, assegurando que a réplica obtida pelo cliente é uma cópia integral da existente no servidor, cuja integridade pode ser verificada. Este projeto propõe a conceção de um serviço avançado de transferência de ficheiros numa rede peer-to-peer (P2P) composta por vários servidores, cada um atuando simultaneamente como cliente. Estes servidores interagem entre si, permitindo que um ficheiro seja disponibilizado a partir de múltiplas fontes simultaneamente utilizando um protocolo baseado em UDP, otimizando a disponibilidade e o desempenho.

Arquitetura da Solução

A arquitetura proposta pode ser resumida em 5 pontos:

- **FS_Node** : Cada nó da rede executa o FS_Node, que atua como cliente e servidor simultaneamente. Este é responsável por gerenciar a partilha de ficheiros, conectar-se ao FS_Tracker para atualizações de rede e disponibilizar ficheiros para download por outros nós.
- **FS_Tracker** : O FS_Tracker é um servidor centralizado responsável por manter uma visão atualizada da rede P2P. Ele armazena informações sobre cada nó, respetivos ficheiros disponíveis e a sua atividade na rede. Os FS_Nodes consultam o FS_Tracker para obter listas atualizadas de localizações de ficheiros e informações sobre outros pares.
- **FS Track Protocol** : Este protocolo facilita a comunicação entre os FS_Nodes e o FS_Tracker. Baseado em TCP, ele permite a troca de informações sobre a disponibilidade de ficheiros, a manutenção de conexões e a atualização do FS_Tracker sobre a atividade de cada nó na rede.
- **FS Transfer Protocol** : Projetado para operar sobre o protocolo de transporte UDP, o FS Transfer Protocol é responsável pela troca eficiente de dados entre os FS_Nodes. Ele suporta a recuperação de blocos perdidos, simulando a fiabilidade presente em TCP, e permite a transferência paralela de múltiplos blocos de diferentes fontes.
- **Divisão e Monitorização da Rede**: O sistema implementa uma divisão dos ficheiros em blocos identificados, permitindo a transferência paralela de N blocos de N nós simultaneamente.

Especificação dos Protocolos Propostos

FS Track Protocol

Ao executar FSNode num Nodo, este procura conexão recorrendo a um socket TCP. O FSTracker, ao receber um pedido de conexão no seu socket, aceita, obtendo o nome do Nodo, a partir do seu IP, utilizando o servidor DNS desenvolvido. Finalmente, cria uma nova thread com o objetivo de gerir as mensagens enviadas por esse Node.

Depois de estabelecer a conexão com o FSTracker, o FSNode envia, pelo socket TCP, uma mensagem "REGISTER,{files_str}", em que "files_str" se traduz numa String que contém todos os ficheiros na sua pasta partilhada. Assim que o FSTracker recebe essa mensagem, regista esse Nodo na sua base de dados, relacionando-o com os seus ficheiros.

Fica à espera de novas mensagens até receber uma mensagem "EXIT" (que apaga todas as informações desse Nodo da sua base de dados).

Quando o FSTracker recebe uma mensagem "GET", seguido do nome de um ficheiro, envia para o remetente uma mensagem "FILE_FOUND {filename}~{node_ip_list_str}<", caso tenha encontrado o ficheiro completo na rede, ou "FILE_NOT_FOUND {filename}<", caso não exista. Envia também uma mensagem "B_FOUND {filename}~{node_ip_result}<", caso existam Nodes que contenham o ficheiro incompleto (se a transferência ainda não foi terminada), ou "B_NOT_FOUND {filename}<", caso não existam Nodes que contenham o ficheiro incompleto. A partir daí, o Node tem todas as informações necessárias para gerir o download do Ficheiro.

As restantes mensagens que o FSTracker pode receber de um Node são "DONE" e "GOT_BLOCK", que são enviadas pelo Node quando este termina a transferência de um certo ficheiro ou quando o mesmo terminou a transferência de um certo bloco do ficheiro, respetivamente, mantendo sempre assim o FSTracker atualizado.

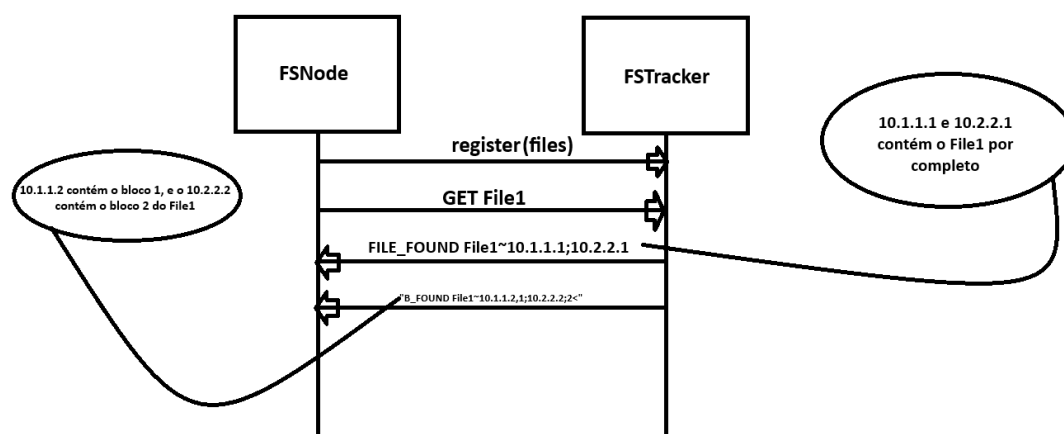


Figura 1. FSNode interroga FSTracker acerca de um Ficheiro

FS Transfer Protocol

Assim que um dado FSNode recebe a lista dos Nodes que contêm o ficheiro, o mesmo calcula o tempo de transferência relativo estimado, enviando uma mensagem pelo seu socket UDP para cada Node, esperando pela resposta (este passo não é executado quando a lista de Nodes apenas contém 1 elemento).

Depois de verificar qual dos Nodes será o mais eficiente, envia uma mensagem "DOWNLOAD_REQUEST,{filename}" para o destinatário pretendido. Após este receber a mensagem, divide o ficheiro em blocos, e envia o conteúdo e o checksum de cada um para o Node que fez a requisição, com uma mensagem "BLOCK~{filename}~{block_number}~{total_blocks}~{checksum}~{block_content.decode('utf-8')}". Posteriormente, quando o Node receber esta mensagem, irá verificar o checksum (caso o checksum não esteja correto, envia uma mensagem "CORRUPTED_BLOCK,{filename},{block_number},{total_blocks}", para que seja reenviado esse bloco) e guarda o conteúdo de cada bloco na sua base de dados (podendo fornecê-lo a outros Nodes, aquando da transferência), organizando, juntando e apagando-os no fim, quando tiver recebido tudo. Para finalizar, apenas é necessário escrever o conteúdo num novo ficheiro.

Não tivemos oportunidade de levar a transferência de blocos individuais a fundo. Ainda assim, essa ideia ficou bem estruturada no código. Apenas seria necessário selecionar, usando um algoritmo, quais blocos seriam transferidos de cada nodo, após o Node receber a lista de Nodes que contêm os blocos individuais (funcionalidade já implementada).

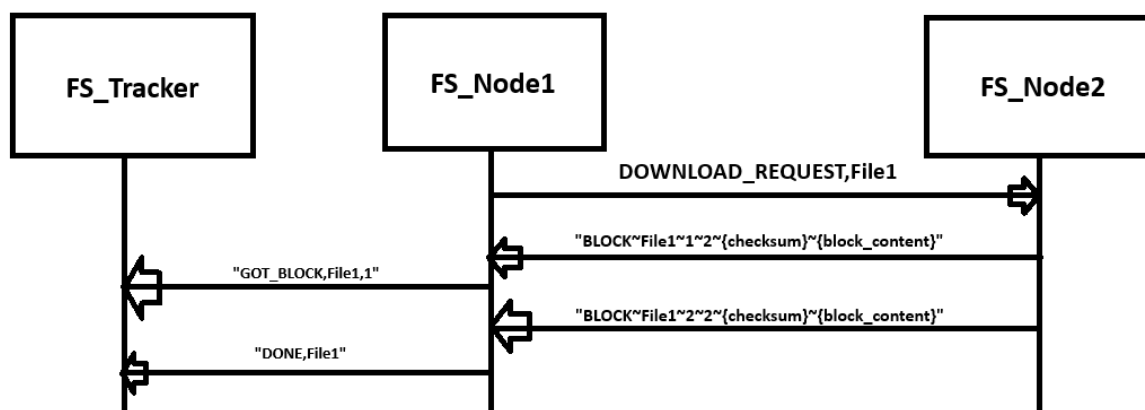


Figura 2. FSNode1 pede um Ficheiro a FSNode2

DNS – Domain Name System

Para a implementação do DNS, recorreremos ao serviço “named” presente na máquina virtual disponibilizada. Foram criadas duas zonas: cc2023 (principal) e cc2023_reverse (para o Reverse DNS).

```
;
; BIND data file for local loopback interface
;
$TTL 604800
@      IN      SOA     cc2023. root.cc2023. (
                                4      ; 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
```

```
;
; BIND reverse data file for local loopback interface
;
$TTL 604800
@      IN      SOA     cc2023. root.cc2023. (
                                4      ; Serial
                                604800 ; Refresh
                                86400  ; Retry
                                2419200 ; Expire
                                604800 ) ; Negative Cache TTL
;
@      IN      NS      Servidor1.cc2023.
1.4.4.10.in-addr.arpa. IN    PTR    Servidor1.cc2023.
2.4.4.10.in-addr.arpa. IN    PTR    Servidor2.cc2023.
1.1.1.10.in-addr.arpa. IN    PTR    Portatil1.cc2023.
2.1.1.10.in-addr.arpa. IN    PTR    Portatil2.cc2023.
1.2.2.10.in-addr.arpa. IN    PTR    PC1.cc2023.
2.2.2.10.in-addr.arpa. IN    PTR    PC2.cc2023.
1.3.3.10.in-addr.arpa. IN    PTR    Roma.cc2023.
2.3.3.10.in-addr.arpa. IN    PTR    Paris.cc2023.
```

Figura 3 e 4. Zonas cc2023 e cc2023_reverse

Adicionalmente, precisamos de adicionar estas duas zonas ao ficheiro “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";

zone "cc2023" IN {
    type master;
    file "/etc/bind/zones/cc2023.zone";
};

zone "10.in-addr.arpa" IN {
    type master;
    file "/etc/bind/zones/cc2023_reverse.zone";
};
```

Figura 4. named_conf_local

Finalmente, apenas foi necessário testar no programa se tudo estava funcional. No Nodo que se abre o FSTracker, necessita-se sempre de escrever “service named start” para iniciar o servidor DNS. Após isso, podemos decodificar os IPs dos Nodos / Tracker.

Implementação

Para a implementação do código necessário a este projeto (posteriormente testado na topologia do Core) optamos por utilizar a linguagem de programação “python”, visto que é bastante intuitiva e inclui bibliotecas que simplificam bastante as necessidades requisitadas.

O código encontra-se dividido em dois ficheiros:

O **FS_Tracker.py**, que contém:

- Funções para receber conexões de FS_Nodes e registar informações sobre cada nó na rede.
- Métodos para lidar com a ativação e desativação dinâmica de nós.
- Funcionalidades para responder a consultas de FS_Nodes e fornecer informações atualizadas sobre a rede, incluindo localizações de ficheiros.
- Lógica para garantir que as informações dadas aos FS_Nodes estejam constantemente atualizadas.

O **FS_Node.py**, que contém:

- Funções para a partilha de ficheiros, incluindo a lógica para dividir os ficheiros em blocos identificados.
- Mecanismos para disponibilizar blocos incompletos a outros nós na rede, mesmo durante o processo de transferência.
- Funcionalidades para a comunicação com o FS_Tracker, como por exemplo o registo do nó e a atualização de informações sobre ficheiros disponíveis.
- Lógica para consultar o FS_Tracker com a finalidade de obter informações atualizadas sobre a rede.

Para estas funcionalidades foram utilizadas as bibliotecas:

1. Socket, para a comunicação via TCP (FSTrackProtocol) ou UDP (FSTransferProtocol);
2. Sys, para obter os argumentos da linha de comando;
3. Threading, para criar Threads para lidar com uma certa função:
 - Quando um Node entra na rede, o FSTracker cria uma Thread para lidar com as mensagens recebidas do mesmo
 - Quando o FSTracker recebe uma mensagem de um dado FSNode, o mesmo cria uma outra Thread para executar o pedido, fazendo com que a outra Thread continue em execução para novos pedidos desse FSNode;
 - Cada FSNode tem inicialmente 3 Threads a decorrer para lidar com as mensagens vindas do FSTracker (ou seja, do seu socket TCP), dos FSNodes (do seu socket UDP), e do input da linha de comandos;
 - Sempre que um Node ou o FSTracker envia uma mensagem para si, uma nova Thread é criada para executar essa mesma tarefa.
5. Time, para a medição do tempo de resposta;
6. OS, para ler e escrever ficheiros do Sistema Operativo;
7. Hashlib, para calcular o checksum dos blocos / ficheiro.

Testes e Resultados

Para verificar o funcionamento da nossa “solução”, optamos por testar a mesma através de uma sequência de partilhas de um mesmo ficheiro (File1). As seguintes imagens mostram a passagem deste ficheiro entre os dispositivos, tendo como origem o “Portatil1”, passando para “Paris” e, finalmente, para “PC1”.

Para tal, iniciamos o FSTracker no Servidor1, tendo posteriormente aberto o FSNode no Portatil1, Paris e PC1 (Figura 1).

A partir de Paris, escreveu-se “GET File1”. Depois de interrogado ao FSTracker, o mesmo forneceu a Paris tanto os Nodes que contêm o ficheiro completo como aqueles com apenas alguns blocos. Como, neste caso, apenas havia 1 Node que tinha o “File1”, Paris não necessitou de calcular e comparar velocidades de download. Fez um pedido de download imediato ao Portatil1, pelo que o mesmo lhe enviou os blocos referentes a esse ficheiro, e os seus checksum’s, informando o FSTracker de cada bloco que transfere com êxito. Depois de Paris recolher todos os blocos e os validar, com o checksum, organiza e junta-os, escrevendo o conteúdo final num novo ficheiro “File1”, na sua pasta de ficheiros. Depois de terminada a transferência, Paris informa o FSTracker que acabou de transferir esse ficheiro, sendo que este vai atualizar a sua base de dados com essa informação. (Figura 2)

Na segunda transferência do File1 (a partir do PC1), o FSTracker já tem na sua base de dados dois Nodes que contêm o File1 (Portatil1 e Paris). Quando PC1 o interroga, são-lhe devolvidos pelo FSTracker esses Nodes, pelo que o mesmo terá de calcular e comparar as velocidades de download, enviando um ping a cada um. Depois de comparadas as velocidades, o PC1 fez o pedido de download a Paris (pois julgou que seria mais rápido, comparativamente com o Portatil1). O resto do processo é semelhante à primeira transferência. (Figura 3)

Finalmente, foi simulado o pedido de transferência do File1 a partir do Portatil1. Como esperado, foi retornado um erro, pois este Node já contém esse mesmo ficheiro. (Figura 4)

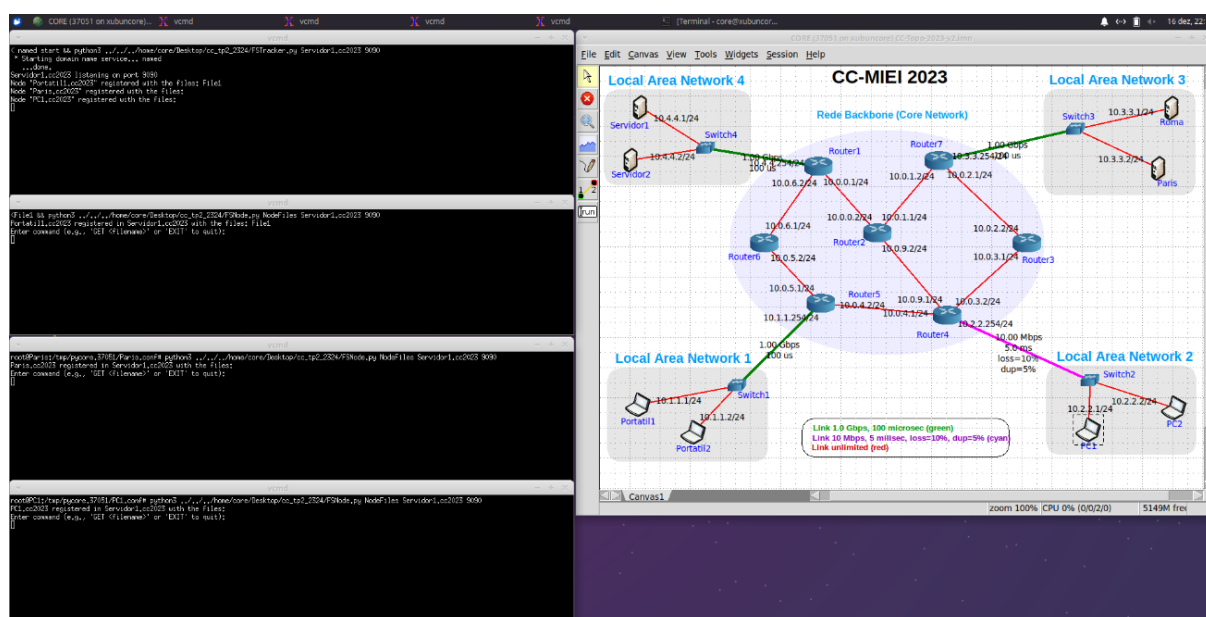
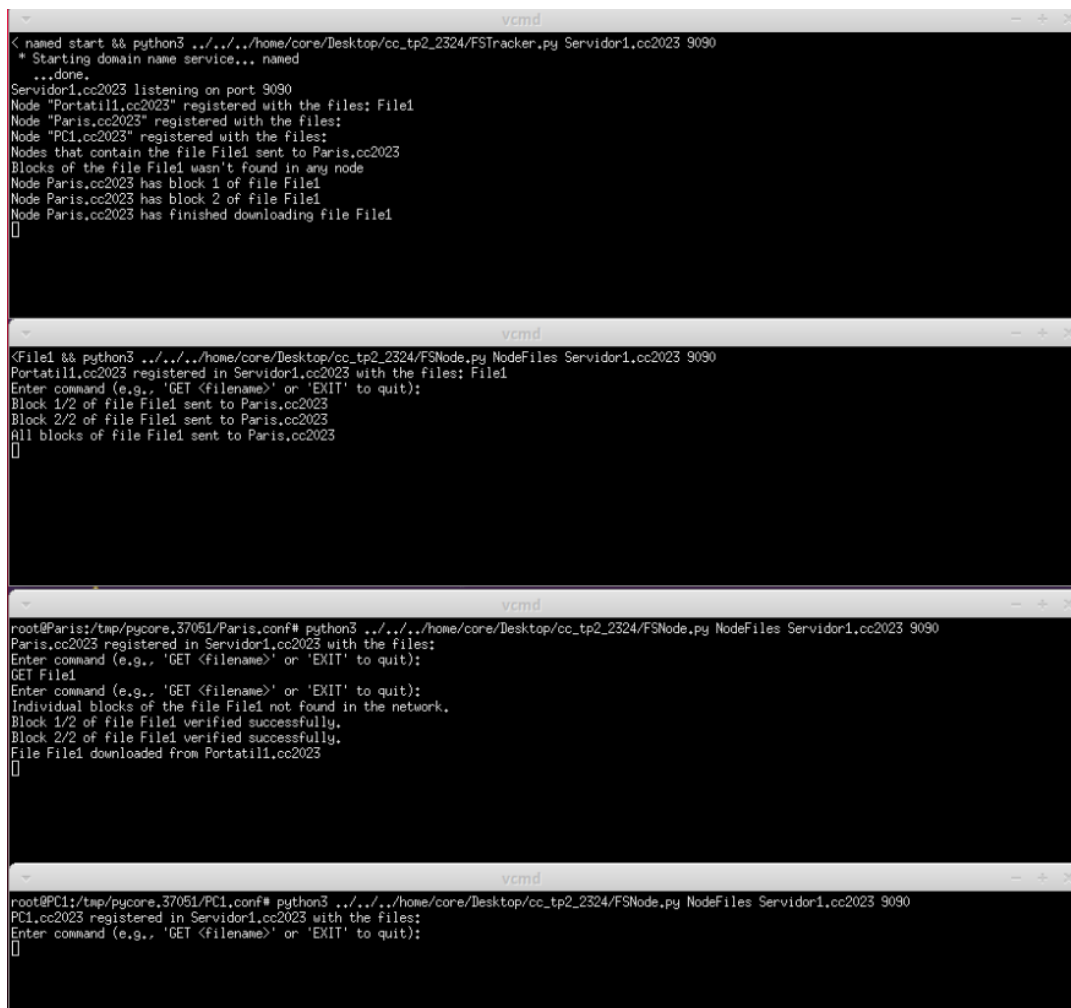


Figura 5. Registo do FSTracker e dos 3 Nodes (Portatil1 com File1).



```
vcmd
< named start && python3 ../../home/core/Desktop/cc_tp2_2324/FSTracker.py Servidor1,cc2023 9090
* Starting domain name service... named
...done.
Servidor1,cc2023 listening on port 9090
Node "Portatil1,cc2023" registered with the files: File1
Node "Paris,cc2023" registered with the files:
Node "PC1,cc2023" registered with the files:
Nodes that contain the file File1 sent to Paris,cc2023
Blocks of the file File1 wasn't found in any node
Node Paris,cc2023 has block 1 of file File1
Node Paris,cc2023 has block 2 of file File1
Node Paris,cc2023 has finished downloading File1
[]

vcmd
<File1 && python3 ../../home/core/Desktop/cc_tp2_2324/FSNode.py NodeFiles Servidor1,cc2023 9090
Portatil1,cc2023 registered in Servidor1,cc2023 with the files: File1
Enter command (e.g., 'GET <Filename>' or 'EXIT' to quit):
Block 1/2 of file File1 sent to Paris,cc2023
Block 2/2 of file File1 sent to Paris,cc2023
All blocks of file File1 sent to Paris,cc2023
[]

vcmd
root@Paris:/tmp/pycore.37051/Paris.conf# python3 ../../home/core/Desktop/cc_tp2_2324/FSNode.py NodeFiles Servidor1,cc2023 9090
Paris,cc2023 registered in Servidor1,cc2023 with the files:
Enter command (e.g., 'GET <Filename>' or 'EXIT' to quit):
GET File1
Enter command (e.g., 'GET <Filename>' or 'EXIT' to quit):
Individual blocks of the file File1 not found in the network.
Block 1/2 of file File1 verified successfully.
Block 2/2 of file File1 verified successfully.
File File1 downloaded from Portatil1,cc2023
[]

vcmd
root@PC1:/tmp/pycore.37051/PC1.conf# python3 ../../home/core/Desktop/cc_tp2_2324/FSNode.py NodeFiles Servidor1,cc2023 9090
PC1,cc2023 registered in Servidor1,cc2023 with the files:
Enter command (e.g., 'GET <Filename>' or 'EXIT' to quit):
[]
```

Figura 6. Paris a fazer download de File1 proveniente do Portatil1

The figure consists of four terminal windows, each titled 'vcmd', showing the execution of a file transfer process. The first window shows the initial setup where 'Servidor1.cc2023' is listening on port 9090 and registers 'Portatil1.cc2023', 'Paris.cc2023', and 'PC1.cc2023'. It then shows the transfer of 'File1' from Paris to PC1, with blocks 1 and 2 being sent and received. The second window shows 'Portatil1.cc2023' registered in 'Servidor1.cc2023' and the transfer of 'File1' from Paris to PC1, with blocks 1 and 2 being sent and received. The third window shows 'Paris.cc2023' registered in 'Servidor1.cc2023' and the transfer of 'File1' from Paris to PC1, with blocks 1 and 2 being sent and received. The fourth window shows 'PC1.cc2023' registered in 'Servidor1.cc2023' and the transfer of 'File1' from Paris to PC1, with blocks 1 and 2 being sent and received.

```
< named start && python3 ../../home/core/Desktop/cc_tp2_2324/FSTracker.py Servidor1.cc2023 9090
* Starting domain name service... named
...done.
Servidor1.cc2023 listening on port 9090
Node "Portatil1.cc2023" registered with the files: File1
Node "Paris.cc2023" registered with the files:
Node "PC1.cc2023" registered with the files:
Nodes that contain the file File1 sent to Paris.cc2023
Blocks of the file File1 wasn't found in any node
Node Paris.cc2023 has block 1 of file File1
Node Paris.cc2023 has block 2 of file File1
Node Paris.cc2023 has finished downloading file File1
Nodes that contain the file File1 sent to PC1.cc2023
Blocks of the file File1 wasn't found in any node
Node PC1.cc2023 has block 1 of file File1
Node PC1.cc2023 has block 2 of file File1
Node PC1.cc2023 has finished downloading file File1
[]

<File1 && python3 ../../home/core/Desktop/cc_tp2_2324/FSNode.py NodeFiles Servidor1.cc2023 9090
Portatil1.cc2023 registered in Servidor1.cc2023 with the files: File1
Enter command (e.g., 'GET <filename>' or 'EXIT' to quit):
Block 1/2 of file File1 sent to Paris.cc2023
Block 2/2 of file File1 sent to Paris.cc2023
All blocks of file File1 sent to Paris.cc2023
Ping response sent to PC1.cc2023
[]

root@Paris:/tmp/pycore.37051/Paris.conf# python3 ../../home/core/Desktop/cc_tp2_2324/FSNode.py NodeFiles Servidor1.cc2023 9090
Paris.cc2023 registered in Servidor1.cc2023 with the files:
Enter command (e.g., 'GET <filename>' or 'EXIT' to quit):
GET File1
Enter command (e.g., 'GET <filename>' or 'EXIT' to quit):
Individual blocks of the file File1 not found in the network.
Block 1/2 of file File1 verified successfully.
Block 2/2 of file File1 verified successfully.
File File1 downloaded from Portatil1.cc2023
Ping response sent to PC1.cc2023
Block 1/2 of file File1 sent to PC1.cc2023
Block 2/2 of file File1 sent to PC1.cc2023
All blocks of file File1 sent to PC1.cc2023
[]

root@PC1:/tmp/pycore.37051/PC1.conf# python3 ../../home/core/Desktop/cc_tp2_2324/FSNode.py NodeFiles Servidor1.cc2023 9090
PC1.cc2023 registered in Servidor1.cc2023 with the files:
Enter command (e.g., 'GET <filename>' or 'EXIT' to quit):
GET File1
Enter command (e.g., 'GET <filename>' or 'EXIT' to quit):
Individual blocks of the file File1 not found in the network.
Block 1/2 of file File1 verified successfully.
Block 2/2 of file File1 verified successfully.
File File1 downloaded from Paris.cc2023
[]
```

Figura 7. PC1 a fazer a transferência de File1 proveniente de Paris

The figure consists of two terminal windows, each titled 'vcmd', showing the execution of a file download process. The first window shows the initial setup where 'Servidor1.cc2023' is listening on port 9090 and registers 'Portatil1.cc2023', 'Paris.cc2023', and 'PC1.cc2023'. It then shows the transfer of 'File1' from Paris to PC1, with blocks 1 and 2 being sent and received. The second window shows 'Portatil1.cc2023' registered in 'Servidor1.cc2023' and the transfer of 'File1' from Paris to PC1, with blocks 1 and 2 being sent and received.

```
* Starting domain name service... named
...done.
Servidor1.cc2023 listening on port 9090
Node "Portatil1.cc2023" registered with the files: File1
Node "Paris.cc2023" registered with the files:
Node "PC1.cc2023" registered with the files:
Nodes that contain the file File1 sent to Paris.cc2023
Blocks of the file File1 wasn't found in any node
Node Paris.cc2023 has block 1 of file File1
Node Paris.cc2023 has block 2 of file File1
Node Paris.cc2023 has finished downloading file File1
Nodes that contain the file File1 sent to PC1.cc2023
Blocks of the file File1 wasn't found in any node
Node PC1.cc2023 has block 1 of file File1
Node PC1.cc2023 has block 2 of file File1
Node PC1.cc2023 has finished downloading file File1
File File1 already exists in node Portatil1.cc2023
[]

<File1 && python3 ../../home/core/Desktop/cc_tp2_2324/FSNode.py NodeFiles Servidor1.cc2023 9090
Portatil1.cc2023 registered in Servidor1.cc2023 with the files: File1
Enter command (e.g., 'GET <filename>' or 'EXIT' to quit):
Block 1/2 of file File1 sent to Paris.cc2023
Block 2/2 of file File1 sent to Paris.cc2023
All blocks of file File1 sent to Paris.cc2023
Ping response sent to PC1.cc2023
GET File1
Enter command (e.g., 'GET <filename>' or 'EXIT' to quit):
File File1 already exists.
[]
```

Figura 8. Portatil1 a tentar fazer download do File1

Conclusões e Trabalho Futuro

Em conclusão, conseguimos desenvolver a maioria das funcionalidades requisitadas, tendo o projeto sucedido na parte de comunicação através de TCP e na transferência dos ficheiros completos através de UDP. No entanto, há aspetos a melhorar, que serão referidos de seguida.

Medidas para implementar futuramente:

1. Caso, no cálculo de tempo de resposta de um certo Node, o mesmo não responda num certo período de tempo, voltar a enviar um “Ping”, removendo-o da rede se não responder na segunda tentativa;
2. Implementar os algoritmos de transferência de blocos de diferentes Nodes;
3. Abranger a transferência de ficheiros, permitindo o download de outros tipos, como por exemplo, MP4, etc.