Comunicações por Computador

a87983 Pedro Pinto
a100659 Rui Pinto
a100066 Ricardo Jesus
Grupo 3 - PL7

Dezembro 2023

1 Introdução

No âmbito deste projeto, apresentamos a concepção e implementação de um serviço de partilha de ficheiros numa rede peer-to-peer (P2P) composta por um servidor principal (FS_TRACKER) e vários nodos clientes (FS_NODE). Este modelo, inspirado em sistemas como o "BitTorrent", visa proporcionar uma transferência eficiente, fiável e distribuída de ficheiros entre os pares da rede.

A partilha de ficheiros é um elemento fundamental em redes de computadores, permitindo que nodos clientes identifiquem, transfiram e armazenem ficheiros a partir de servidores disponíveis. No entanto, para assegurar a integridade dos dados transferidos, é essencial que a replicação do ficheiro no cliente seja uma cópia fiel do original no servidor, e que a transferência seja realizada de forma fiável, sem erros.

O objetivo principal deste projeto é desenvolver um serviço de partilha de ficheiros peer-to-peer de alto desempenho. Para garantir uma transferência eficiente, implementamos um protocolo de transferência sobre o protocolo de transporte UDP, permitindo a divisão prévia do ficheiro em blocos identificados. Estes blocos podem ser descarregados em paralelo a partir de várias localizações diferentes, otimizando assim a disponibilidade e o desempenho da rede.

O serviço será implementado em três fases distintas. Inicialmente, desenhamos e implementamos o protocolo FS Track Protocol, que funciona sobre TCP e é responsável pelo registo de FS_Nodes, atualização de listas de ficheiros disponíveis e pedidos de localização de ficheiros. Posteriormente, passamos para a segunda fase, onde desenvolvemos e testamos o protocolo FS Transfer Protocol, operando sobre UDP. Este protocolo é responsável por aceitar pedidos de blocos, pedir blocos a outros FS_Nodes e assegurar a recuperação de blocos perdidos.

Finalmente, na terceira fase, exploramos a possibilidade de utilizar um serviço de resolução de nomes, como o DNS, para identificar FS_Nodes e o FS_Tracker pelos seus nomes em vez de endereços IP. Esta alteração visa facilitar a gestão da rede.

Em resumo, este projeto propõe uma solução para a partilha de ficheiros em redes P2P, incorporando elementos de desempenho, fiabilidade e distribuição que são essenciais para um serviço eficiente e robusto.

2 Arquitetura da Solução

Arquitetura Geral: A solução proposta adota uma arquitetura peer-to-peer (P2P) distribuída, na qual múltiplos FS_Node, desempenham simultaneamente os papéis de servidor e cliente. Esta natureza descentralizada permite que os pares interajam diretamente entre si para partilhar ficheiros, promovendo uma distribuição eficiente e descentralizada da carga na rede.

Protocolos Utilizados: Dois protocolos principais são implementados na solução: o FS Track Protocol, que opera sobre o protocolo de transporte TCP, é responsável pelo registo de FS_Nodes e gestão da rede, enquanto o FS Transfer Protocol, que funciona sobre o protocolo de transporte UDP, facilita a transferência de blocos de ficheiros. A escolha de utilizar TCP para o FS Track Protocol visa garantir uma comunicação fiável e orientada à conexão, enquanto o uso de UDP no FS Transfer Protocol otimiza a eficiência da transferência, implementando estratégias de multithreading para aceitar e pedir blocos em paralelo.

Componentes Principais: Os principais componentes do sistema incluem o FS_Tracker e o FS_Node. O FS_Tracker atua como um servidor centralizado, encarregado do registo de FS_Nodes, atualização de listas de ficheiros e gestão da rede. Os FS_Nodes, por sua vez, atuam como clientes e servidores simultaneamente, participando na partilha de ficheiros e colaborando na transferência de blocos, utilizando multithreading para otimizar operações que ocorrem de forma simultânea.

FS_Tracker: O FS_Tracker assume um papel central na rede P2P, registando FS_Nodes, mantendo listas atualizadas de ficheiros e blocos e gerindo a interação entre os pares. Ele fornece uma visão abrangente do estado da rede, permitindo uma localização eficiente de ficheiros e otimização do desempenho da transferência.

FS_Node: Os FS_Nodes são os nodos participantes na rede P2P, desempenhando funções de cliente e servidor. Ao registar-se no FS_Tracker, atualizar informações sobre os ficheiros que possuem e interagir com outros FS_Nodes, eles formam a espinha dorsal da arquitetura, facilitando a partilha distribuída de ficheiros. Além disso, os FS_Nodes implementam multithreading para aceitar pedidos de blocos em paralelo e pedir blocos do mesmo ficheiro a outros FS_Nodes.

FS Track Protocol: O FS Track Protocol que opera sobre o protocolo TCP, desempenha um papel fundamental na comunicação entre os FS_Nodes e o FS_Tracker, estabelecendo uma base para o registo de nós, atualização de informações e pesquisa de ficheiros na rede P2P.

FS Transfer Protocol: Operando sobre o protocolo UDP, o FS Transfer Protocol possibilita a transferência de blocos de ficheiros entre FS_Nodes. Estratégias para lidar com perdas de blocos e garantir entrega fiável são implementadas, utilizando multithreading para processar operações em paralelo e otimizar o desempenho.

Divisão e Transferência de Ficheiros: A solução adota uma abordagem de divisão prévia de ficheiros em blocos de 1024 Bytes devidamente identificados. Esses blocos podem ser transferidos em paralelo de diferentes FS_Nodes, otimizando a disponibilidade e o desempenho da rede.

Considerações de Desempenho: A arquitetura é desenhada para garantir um bom desempenho na transferência de ficheiros, com estratégias que visam maximizar a eficiência na utilização dos recursos da rede, incluindo o uso de multithreading para operações simultâneas e otimização do desempenho.

Cenário de Teste: No cenário de teste proposto, a arquitetura é avaliada em condições simuladas usando o emulador CORE e a topologia CC-Topo-2023-v2.imn.

A interação entre FS_Tracker e vários FS_Nodes é examinada para verificar o funcionamento adequado da solução em diferentes localizações da rede.

3 Especificação do(s) protocolo(s) propostos

3.1 Formato das mensagens protocolares e Interações entre Nodos

O protocolo de comunicação estabelecido entre o FS_Node e o FS_Tracker define as interações essenciais para o registo de nós, pesquisa de ficheiros e atualizações na rede P2P. Este protocolo simplificado visa garantir uma comunicação eficaz e coordenada entre os diversos componentes do sistema.

Protocolo FS Track:

1. Registo de Nó:
   - Comando: **REGISTER File [Blocks]**
   - Descrição: O FS_Node comunica ao FS_Tracker o seu registo, fornecendo o nome do ficheiro que possui e a lista de blocos associados a esse ficheiro. O IP do nodo é encontrado através do socket que permite a ligação ao Fs_tracker
   - Exemplo: **REGISTER File1 [Block1, Block2, Block3]**
   - Resposta: **Nodo 192.168.1.1 registado**
2. Pedido para listar todos os Ficheiros:
   - Comando: **FIND_ALL_FILES**
   - Descrição: O FS_Node solicita ao FS_Tracker uma lista atualizada de todos os ficheiros disponíveis na rede.
   - Exemplo: **FIND_ALL_FILES**
   - Resposta: **[File1, File2, File3]**
3. Pesquisa de um Ficheiro:
   - Comando: **SEARCH File**
   - Descrição: O FS_Node solicita ao FS_Tracker a localização de um ficheiro específico, indicando o nome do ficheiro desejado.
   - Exemplo: **SEARCH File1**
   - Resposta: **File1 List_Nodes Blocks**
4. Saída do Sistema:
   - Comando: **EXIT**
   - Descrição: O FS_Node notifica o FS_Tracker da sua saída do sistema. Após a execução deste comando o FS_Tracker eliminina o registo no nodo que efetuou a saída do sistema.
   - Exemplo: **EXIT**

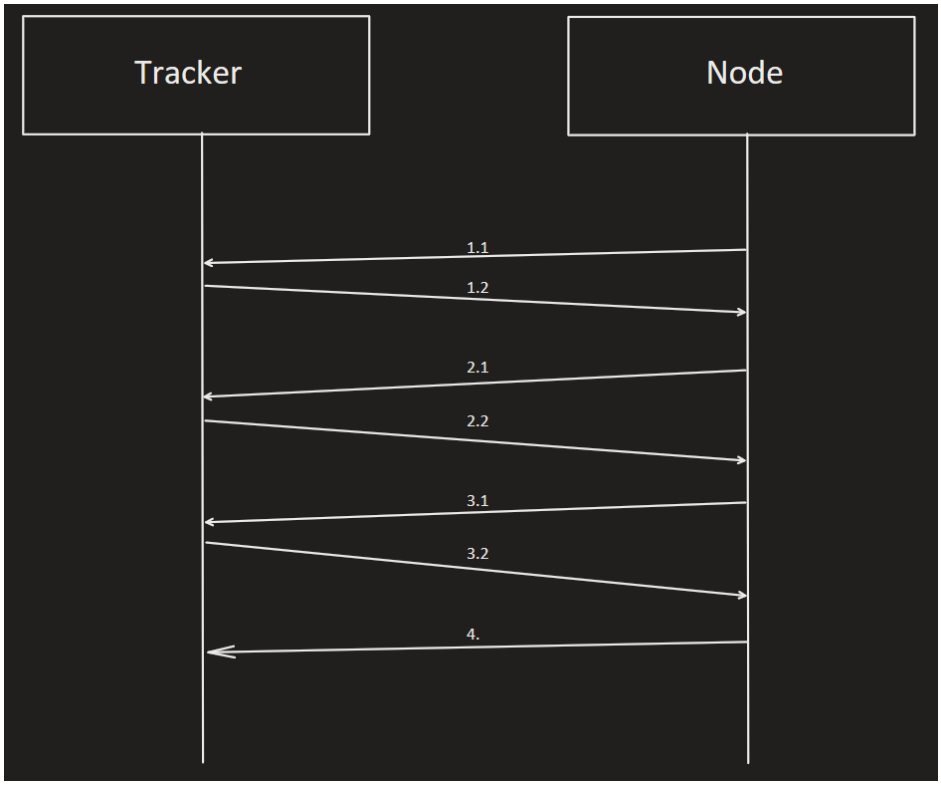

Figura 1: Diagrama de Comunicação Nodo Tracker

- **1.1** - REGISTER File [Blocks]
- **1.2** - Nodo registado
- **2.1** - FIND_ALL_FILES
- **2.2** - List_Files
- **3.1** - SEARCH File
- **3.2** - File List_Nodes Blocks
- **4** - EXIT

Protocolo FS Transfer:

1. Transferência de ficheiro
    - Comando: **GET File**
    - Descrição: O FS_Node comunica ao FS_Tracker que quer um determinado ficheiro, fornecendo o seu nome. Após isso, o FS_Tracker responde com uma lista dos nodos e os respetivos blocos do ficheiro pedido. Ao receber a lista, é chamado o algoritmo de escalonamento e seleção dos blocos que por sua vez seleciona quais blocos de cada nodo tem de selecionar para posterior transferência. Após isso é enviado um array ([”REQUEST”, filename, filename_blockX.txt]) aos nodos para pedir individualmente cada bloco aos nodos selecionados pelo algoritmo de seleção. Para evitar a perda de blocos, o nosso programa faz o REQUEST continuamente de cada bloco até ter a certeza que o mesmo foi transferido.
    - Exemplo: **GET File1**
    - Resposta: blocos tranferidos para pasta dos blocos de um ficheiro e blocos unidos num único ficheiro

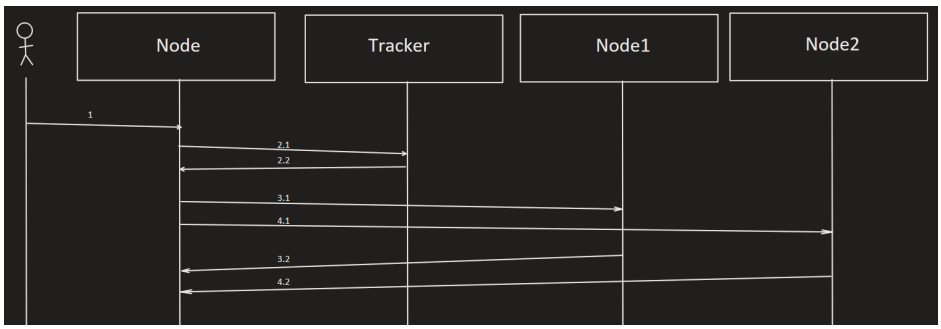

Figura 2:

- **1** - GET File
- **2.1** - GET File
- **2.2** - [Node,[Blocks]]
- **3.1** - [REQUEST, filename, filename_block]
- **4.1** - [REQUEST, filename, filename_block]
- **3.2** - Block
- **4.2** - Block

4 Implementação

4.1 Detalhes, parâmetros, bibliotecas de funções, etc.

Na fase de implementação decidimos dividir o projeto em dois ficheiros python diferentes, fs_node.py e fs_tracker.py.

4.1.1 Fs_node

O ficheiro **Fs_node** em Python é um nodo cliente, quando comunica com o **Fs_tracker** e simultaneamente serve de servidor na conexão a outros **Fs_node**. Começamos por importar os módulos necessários, incluindo json, socket, sys, os, threading, time, ast, random e hashlib. O **Fs_node** é inicializado com os respetivos argumentos na linha de comandos. Estes incluem o caminho da pasta compartilhada (**shared_folder**), o IP do Servidor **Fs_Tracker** (**HOST**), a porta TCP (**PORT**), usada para comunicar entre um **Fs_node** e **Fs_tracker** e a porta do servidor UDP (**PORT_UDP_SERVER**) usada para transferir blocos de dados entre duas instâncias do fs_node . O loop "while"principal recebe comandos de entrada do utilizador. Se o comando for **EXIT**, o nodo envia uma solicitação de saída para o servidor. Se for um comando **GET**, este solicita blocos de outros nodos usando UDP. Se for um outro comando como **REGISTER** ou **LIST_ALL_FILES** OU **SEARCH**, este envia o comando para o **Fs_tracker**. De início, um socket TCP (**client_socket**) é criado e conectado ao servidor usando o host e a porta especificados. Logo a seguir a função **list_files_in_shared_folder** lista os ficheiros das pastas existentes e envia um comando **REGISTER** automaticamente por TCP para o servidor, fornecendo informações sobre os ficheiros disponíveis e os seus blocos. Ao mesmo tempo a função **udp_server** cria um socket de servidor UDP e aguarda mensagens UDP recebidas. Esta inicia uma nova thread para cada mensagem recebida para lidar com a tarefa associada de forma concorrente. Quando o utilizador quer transferir um ficheiro de outro nodo, usa o comando **GET** com o respetivo ficheiro. Este comando é enviado ao **Fs_tracker**, que responde com uma lista dos nodos e os blocos que estes contêm do ficheiro pedido. Logo após, é chamado um algoritmo de seleção que dos nodos com todos os blocos do ficheiro a transferir, escolhe dois aleatoriamente e retira metade dos blocos a cada um, paralelamente usando multithreading. No caso de haver apenas um nodo com blocos do ficheiro transfere todos os blocos desse nodo. De seguida é chamada numa thread diferente a função **get_Node_Blocks** que envia um comando **REQUEST** juntamente com o nome do ficheiro e o bloco que quer transferir, para o nodo que tem o bloco. Nesse nodo é recebido o comando **REQUEST** que é lido na função **deal_node_server_task**. Aqui é calculado um checksum (**calculate_checksum**) e é enviada uma estrutura que contém o **checksum** e o **content** que é o respetivo bloco a transferir. Do lado do nodo que vai receber a informação calcula-se o checksum originalmente calculado na função **deal_node_server_task** com o checksum recebido. Se este for igual significa que o conteúdo não foi alterado e o bloco seguinte é incrementado e procede-se ao seu envio. Caso o checksum seja diferente, o bloco seguinte não é incrementado e procede-se ao envio do mesmo bloco do qual a informação estava comprometida.

4.1.2 Fs_tracker

O **Fs_tracker.py** é responsável por registar e manter informação de ficheiros em varios nodos da rede distribuída. O comando **'REGISTER'** permite o registo de um nodo no sistema. Quando recebido, o código verifica se o nó já está registrado. Se não estiver, cria uma entrada associando um dicionário vazio a esse nó. Se o nó já estiver registado, verifica se o ficheiro também está na lista de ficheiros desse nó e adiciona-o se não estiver. O código também verifica e atualiza a lista geral de ficheiros (**files_folder**) que contém informação sobre os ficheiros existentes na rede e o número de blocos máximo desse ficheiro No final, os dicionários **registered_nodes** e **files_folder** são imprimidos para debug do utilizador. O comando **'SEARCH'** procura por um ficheiro no sistema. A função **search_file** percorre o dicionário **registered_nodes**, procurando o ficheiro desejado. Se encontrado, retorna uma lista de tuplos contendo o ID do nó e a lista de blocos associados ao ficheiro. Além disso,

retorna o número máximo de blocos associados ao ficheiro obtido do dicionário **files_folder**. O comando **'LIST_ALL_FILES'** lista todos os ficheiros registados no sistema. O código percorre os dicionários registered_nodes, colecta todos os ficheiros únicos e retorna uma mensagem com a lista desses ficheiros O comando **'GET'** procura informações sobre um ficheiro no sistema. Utiliza a função **search_file** para obter informações sobre o ficheiro e converte a resposta numa string JSON antes de enviá-la de volta ao cliente. O comando **'EXIT'** encerra a conexão de um nodo no sistema. Remove o nodo correspondente dos dicionários **registered_nodes** e retorna uma mensagem de sucesso ou informa que o nodo não foi encontrado. Além desses comportamentos específicos de comandos, aceita conexões num loop e inicia uma nova thread para cada cliente. A função **process_client** gere as operações do cliente, recebendo comandos, processando-os e enviando as respostas. Os dicionários registered_nodes e **files_folder** são fundamentais para reter informação dos nodos registados e dos ficheiros no sistema.

5 Testes e Resultados

De forma a testar a implementação deste projeto, optamos pelo uso da ferramenta CORE disponibilizada na plataforma elearning. Nesta abordagem, empregamos uma topologia também ela fornecida pelos docentes, que simula uma rede composta por vários nodos. Estes podem ser representados por servidores, portáteis, PCs, etc. Entre estes nodos, existem também vários tipos de ligações. Uma com 1 Gbps de velocidade e outra com 10 Mbps, onde existem também perdas de pacotes (para testar a forma como lidamos com a perda dos mesmos). De seguida podemos observar uma figura que exemplifica a forma como a topologia está organizada.

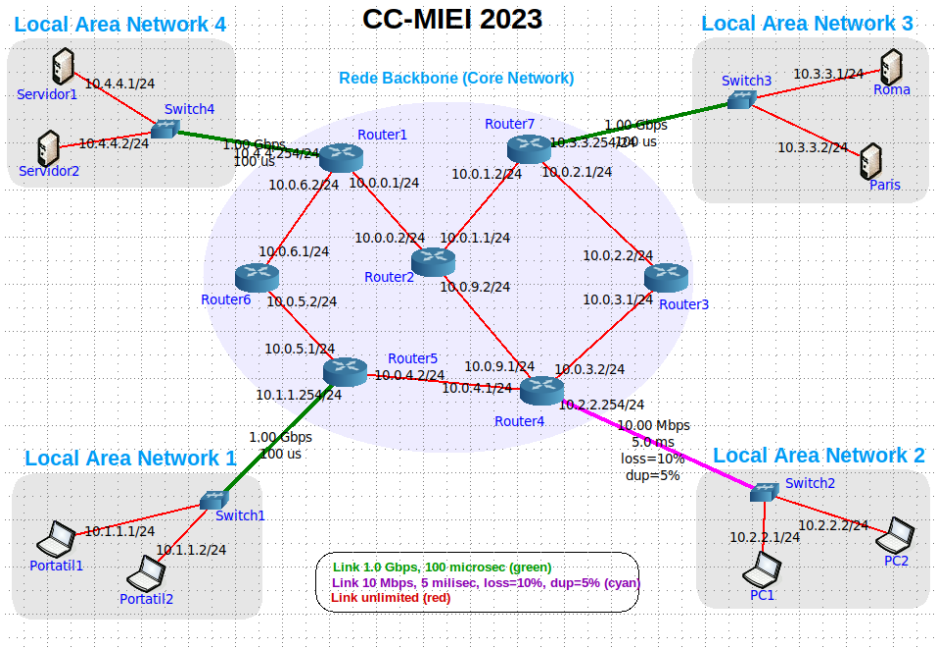

Figura 3: Topologia CC-MIEI 2023 usada na Testagem do Projeto

5.1 Comando GET file.txt

Nestas imagens a seguir, é possível ver as alterações efetuadas entre as pastas partilhadas e os comandos e o que é apresentado nos diferentes terminais do FS_Tracker e dos FS_Node antes e após o comando GET file. Neste exemplo o PC1 pede o file2.txt ao Portatil1.

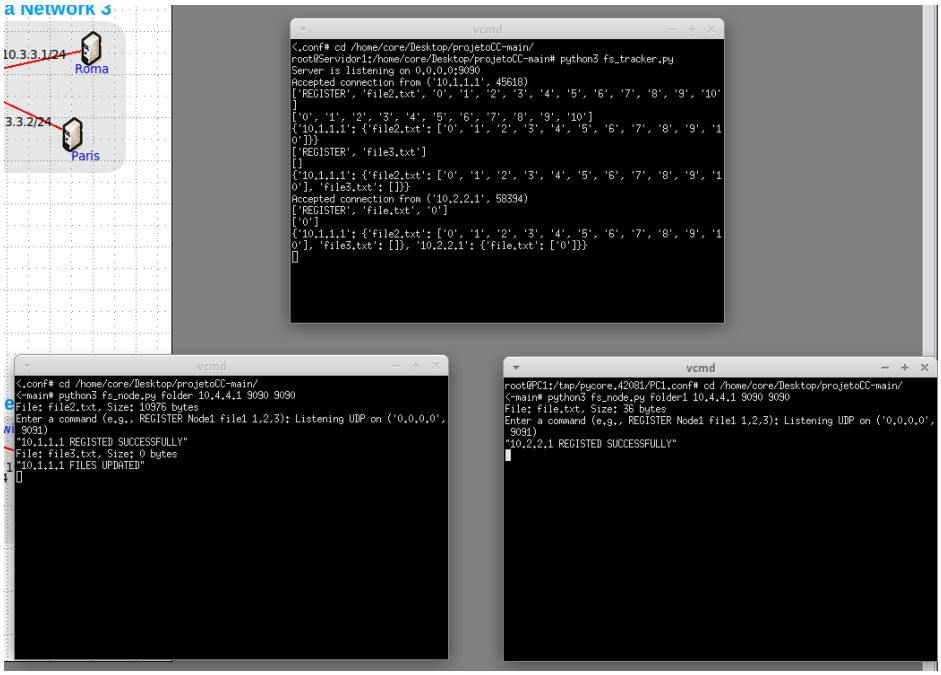

Figura 4: Terminais do Servidor1, Portatil1 e PC1 antes do comando GET

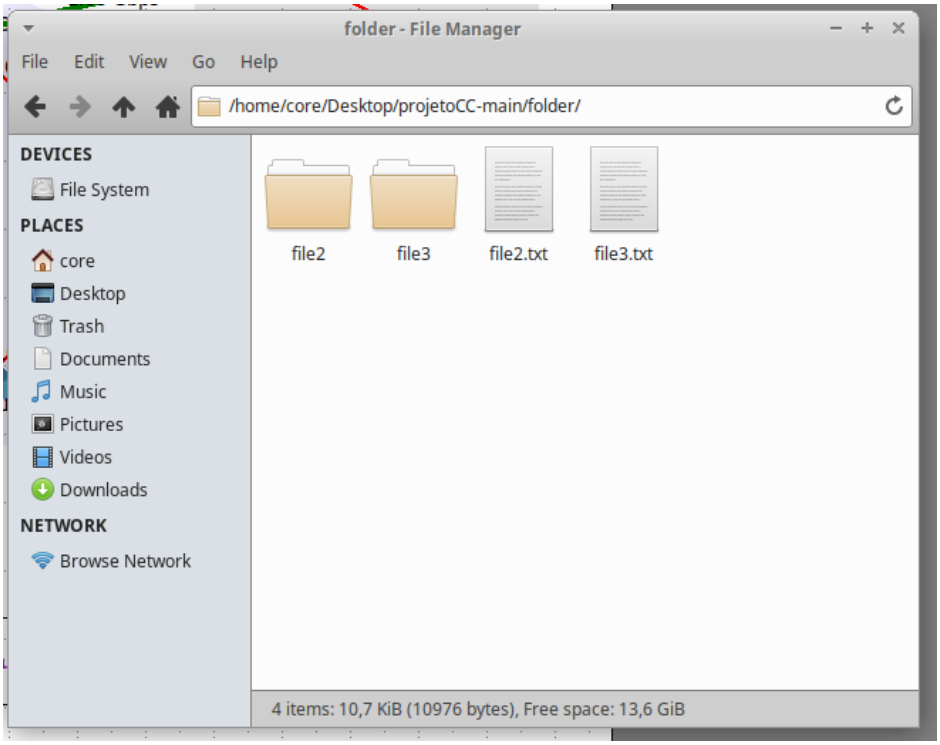

Figura 5: Pasta partilhada do Portatil1 antes do comando GET

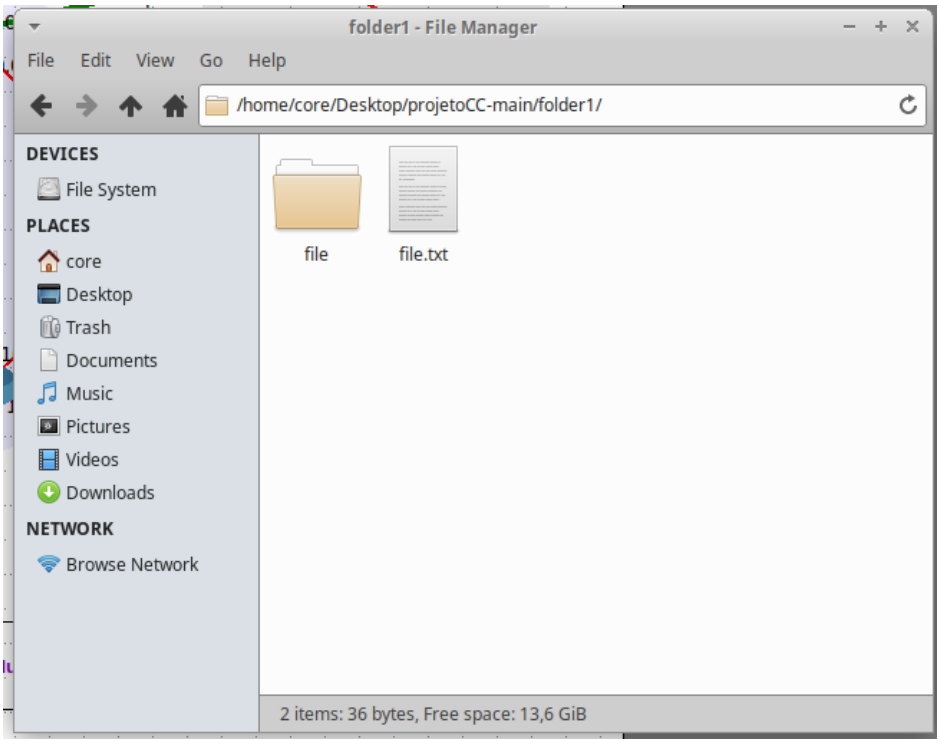

Figura 6: Pasta partilhada do PC1 antes do comando GET

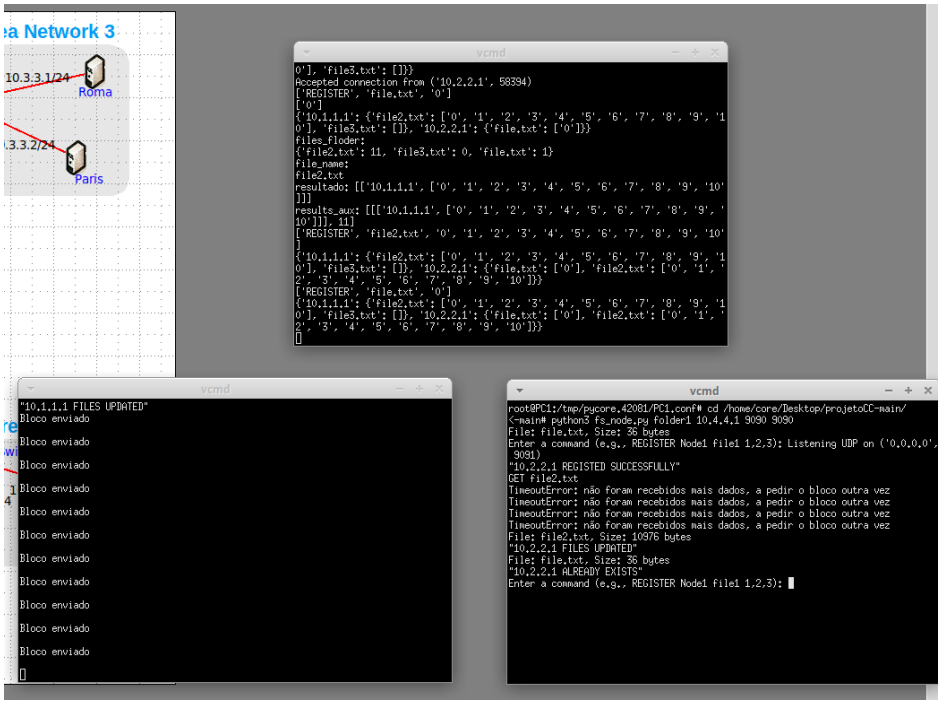

Figura 7: Terminais do Servidor1, Portatil1 e PC1 após comando GET

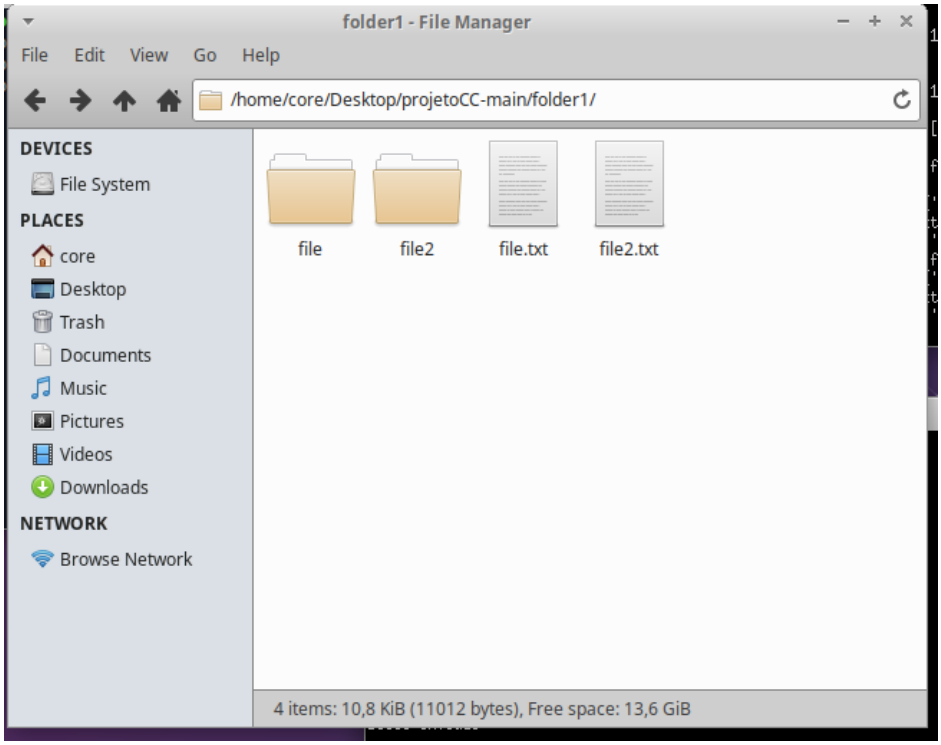

Figura 8: Pasta partilhada de PC1 após o GET

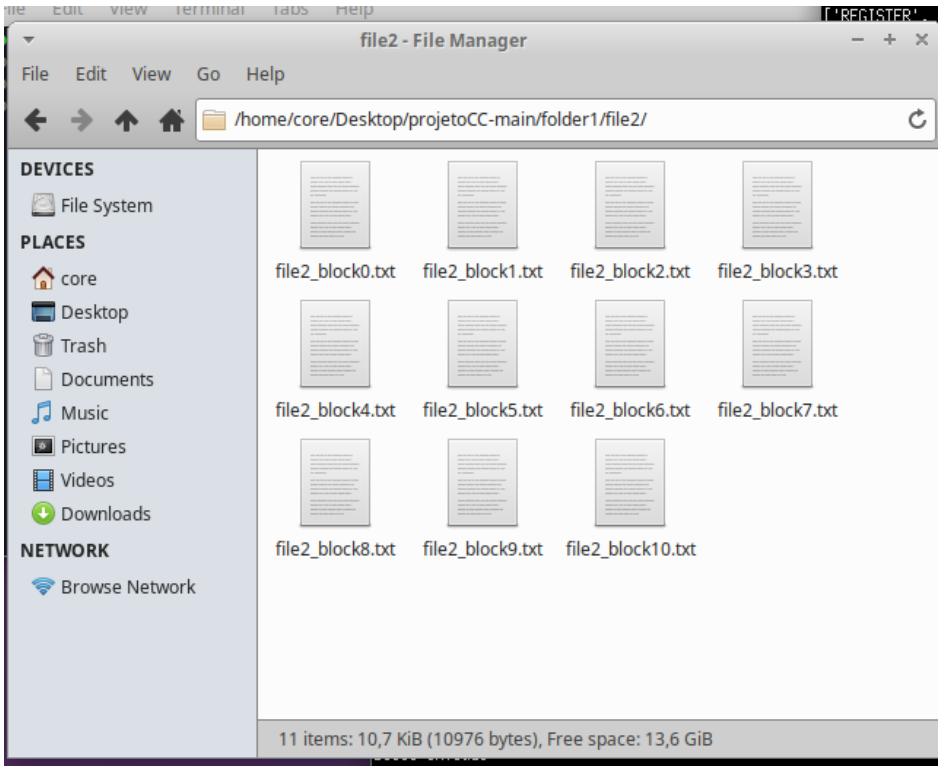

Figura 9: Psta de blocos de um ficheiro "file2.txt"

6 Conclusões e Trabalho Futuro

Em termos de conclusão, este projeto representou uma valiosa oportunidade para aprofundar o nosso entendimento sobre conceitos fundamentais relacionados com redes peer-to-peer e protocolos de transferência de ficheiros em ambientes distribuídos. Ao longo do semestre, enfrentamos desafios substanciais que, por sua vez, contribuíram para melhorar as nossas habilidades de resolução de problemas e tomada de decisões técnicas.

A ênfase na sincronização e comunicação entre os nodos reforçou a consistência das informações sobre os ficheiros. A aplicação de sockets TCP e UDP proporcionou uma base sólida para a troca de dados, enquanto os comandos como registo, pesquisa e listagem foram tratados de maneira robusta. A integridade e confiabilidade dos dados, foi encontrada através da implementação de checksums, que adicionaram uma camada crucial de fidedignidade ao sistema.

No entanto, reconhecemos que, em qualquer projeto, há sempre espaço para melhorias. Primeiramente poderíamos ter finalizado a seção do projeto que envolve a resolução de nomes através de um sistema de DNS. Embora tenhamos um servidor que opera como um serviço que mapeia nomes de host para endereços IP e quando um fs_node ou fs_tracker se conecta o seu nome e IP são registados, esta parte do projeto não tem conexão com a restante implementação, ficando inacabada. Além disso poderiamos ter explorado estratégias mais eficientes e complexas de seleção de blocos como por exemplo as referidas nas páginas 175-181 do livro *"Computer Networking - A Top Down Aproach"*, onde é exemplificado como funciona a distribuição de ficheiros no 'BitTorrent'. Em alternativa poderiamos também ter utilizado uma estratágia que medisse a latência das ligações entre nodos podendo utilizar o comando ping para esse efeito, e medir quais os nodos com resposta mais rápida e alterar o algoritmo de acordo com isso.

Em síntese, o projeto oferece uma visão abrangente da construção de um sistema de ficheiros distribuído, destacando a importância da eficiência na comunicação, da sincronização entre os nós e da integridade dos dados. As fases de implementação refletem uma abordagem completa para gerir o sistema distribuído, demonstrando a interação entre os nodos para compartilhar e recuperar informações de maneira segura. O uso estratégico de tecnologias como sockets TCP e UDP, aliado à implementação de mecanismos de verificação de integridade, consolida a robustez do sistema, contribuindo para uma solução coesa e resiliente em ambientes distribuídos.