Trabalho Prático N°2

Eduardo Pereira[A94881], Gonçalo Vale[A96923], and Pedro Oliveira[A95076]

[1] Universidade do Minho
[2] Licenciatura em Engenharia Informática
[3] Comunicações por Computador - Grupo 67
[4] Transferência rápida e fiável de múltiplos servidores em simultâneo
[5] Ano Letivo 2023/2024

Abstract. A partilha de ficheiros em ambientes peer-to-peer (P2P) é uma componente crucial nas infraestruturas de comunicação contemporâneas, proporcionando aos utilizadores a capacidade de transferir dados de forma eficaz entre si. Esta trabalho abrange a especificação, implementação e testes do protocolo, destacando a arquitetura, o protocolo de comunicação, a integração com UDP para transferência eficiente de dados, e os resultados obtidos nos testes realizados. Apesar de alguns desafios identificados, o FS Track Protocol demonstrou ser uma contribuição valiosa para o domínio das redes P2P, com potencial para melhorias e expansões futuras.

1 Introdução

A partilha de ficheiros representa um componente vital nas infraestruturas de comunicação contemporâneas, conferindo aos utilizadores a capacidade de transferir dados de forma eficaz entre si. Este projeto concentra-se no desenvolvimento de um serviço de partilha de ficheiros em redes peer-to-peer (P2P), onde múltiplos servidores desempenham o papel simultâneo de clientes. O objetivo central deste serviço não se limita à mera transmissão fiável de dados, mas estende-se à otimização do desempenho, possibilitando a transferência concorrente de partes específicas de um ficheiro a partir de diversos servidores.

A arquitetura subjacente ao sistema é composta por duas entidades principais: o `FS_Tracker` e o `FS_Node`. O `FS_Tracker` assume a função de servidor central de registo, mantendo informações atualizadas acerca de todos os nós na rede, bem como dos respetivos ficheiros e blocos disponíveis. Por outro lado, o `FS_Node` representa a aplicação executada por cada utilizador, desempenhando simultaneamente os papéis de cliente e servidor, sendo responsável pela partilha de ficheiros e blocos.

O delineamento cuidadoso do protocolo assume uma importância crítica no êxito do projeto. Concentramo-nos na especificação e implementação do `FS Track Protocol`, operando sobre o protocolo TCP. Este protocolo facilita o registo de um `FS_Node`, a atualização das informações relativas a ficheiros e blocos disponíveis, e a solicitação da localização de ficheiros na rede P2P.

Ao longo deste relatório, iremos detalhar a arquitetura proposta e a implementação realizada.

2 Arquitetura da Solução

2.1 Visão Geral da Arquitetura

A arquitetura proposta para o sistema é concebida como uma rede peer-to-peer (P2P) que visa otimizar a partilha de ficheiros entre múltiplos utilizadores. Neste modelo, cada nó na rede, representado pela aplicação `FS_Node`, desempenha simultaneamente os papéis de cliente e servidor. O `FS_Tracker` atua como um servidor centralizado responsável pelo registo e manutenção de informações sobre os diversos nós e os respetivos ficheiros e blocos partilhados.

2.2 Abordagem P2P com Múltiplos Servidores/Clientes

A abordagem P2P adotada no sistema permite que cada nó na rede atue como um servidor, oferecendo recursos (ficheiros e blocos) aos outros nós, e como um cliente, solicitando recursos que não possui localmente. Essa dualidade de funções maximiza a eficiência e a disponibilidade do serviço, uma vez que a carga é distribuída entre os diversos nós na rede. A capacidade de cada nó servir como fonte e destino de transferências contribui para um desempenho global aprimorado.

2.3 Componentes Principais: `FS_Node` e `FS_Tracker`

`FS_Node` O `FS_Node` representa a aplicação executada em cada sistema na rede P2P. Este componente é responsável pela partilha de ficheiros e blocos, assim como pelo registo junto do `FS_Tracker`. A aplicação `FS_Node` conecta-se ao `FS_Tracker` para fornecer informações sobre os ficheiros e blocos que possui, bem como para solicitar dados relativos a outros nós na rede.

`FS_Tracker` O `FS_Tracker` é o servidor central que mantém a coerência na rede P2P. Ele mantém uma base de dados atualizada com informações sobre todos os `FS_Node` registados, incluindo os ficheiros e blocos que cada nó possui. O `FS_Tracker` serve como ponto de contacto para os `FS_Node` localizarem e transferirem recursos na rede, fornecendo uma lista atualizada de localizações para um determinado ficheiro.

3 Especificação dos Protocolos

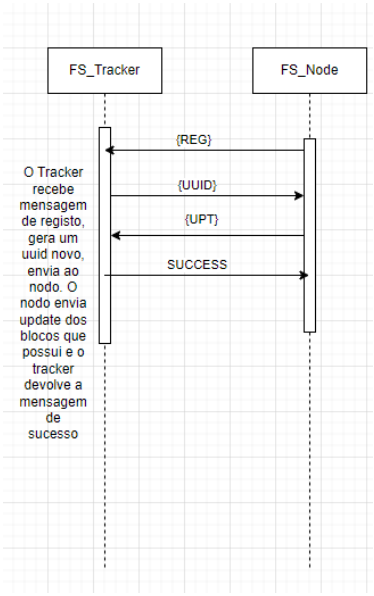
3.1 `FS Track Protocol`

O `FS Track Protocol` opera sobre o protocolo de transporte TCP, facilitando a comunicação entre os nós da rede P2P e o servidor central `FS_Tracker`. Esta seção fornece uma especificação detalhada do protocolo, incluindo o formato das mensagens, a semântica dos campos e um diagrama temporal que ilustra o comportamento do protocolo.

Formato das Mensagens Protocolares As mensagens trocadas entre `FS_Node` e `FS_Tracker` seguem um formato estruturado para garantir uma comunicação eficiente e sem ambiguidades. O formato básico é composto pelos seguintes campos:

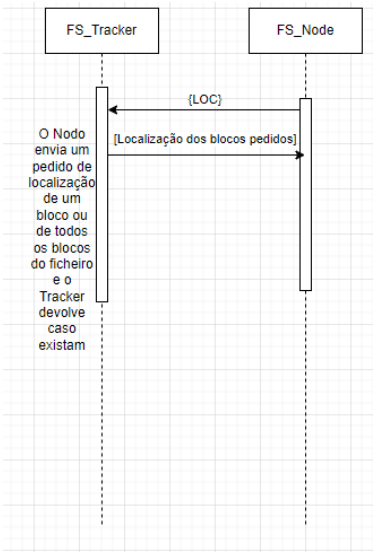
- **Tipo da Mensagem (`type`):** Indica a finalidade da mensagem, podendo ser REG (registo), UPT (atualização), LOC (localização), etc.
- **UUID (`UUID`):** Identificador único do nó na rede.
- **Endereço IP (`IP`):** Endereço IP do nó na rede.
- **Porta (`Port`):** Número da porta associado ao nó.
- **Nome do Ficheiro (`FileName`):** Nome do ficheiro associado à mensagem (aplica-se a mensagens de registo e localização).
- **Número do Bloco (`BlockNumber`):** Número do bloco associado à mensagem (aplica-se a mensagens de localização).
- **Dicionário de Blocos (`BlocksDic`):** Estrutura que contém informações sobre os blocos de um ficheiro (aplica-se a mensagens de atualização).

Cada campo desempenha um papel específico nas diferentes mensagens transmitidas entre os nós e o `FS_Tracker` e cada um deles é essencial para a correta interpretação das mensagens, garantindo a consistência das operações na rede P2P.



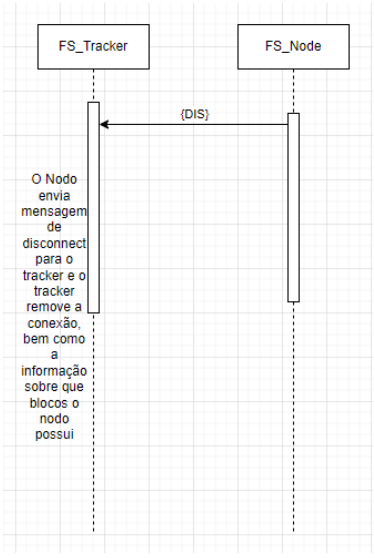
h

Fig. 1. Ligação Inicial



h

Fig. 2. Pedido de localização do Bloco



h

Fig. 3. Dexconexão de Nodo

3.2 `FS Transfer Protocol`

O `FS Transfer Protocol` opera sobre o protocolo de transporte UDP, de modo a estabelecer conexões e efetuar transferência de blocos/ficheiros entre `FS_Nodes`. Esta seção fornece uma especificação detalhada do protocolo, incluindo o formato das mensagens, a semântica dos campos e protocolos adicionais.

Formato das Mensagens Protocolares As mensagens trocadas entre `FS_Nodes` seguem um formato estruturado para garantir uma comunicação eficiente e sem ambiguidades. O formato básico é composto pelos seguintes campos:

- **Tipo de Mensagem 1 (`type`):** Indica a finalidade da mensagem, que será sempre REQ (abreviação para requisição de pacote(s)), enviado do nodo recetor para o nodo que fará o envio.
- **Tipo de Mensagem 2 (`status`):** Enviado do nodo que envia o pacote para o nodo recetor, que confirma se este possui o bloco.
- **Tipo de Mensagem 3**: o datagrama não encoded com a bytestring referente ao bloco desejado

Controlo de perdas Para o controlo de perdas, e devido a constrangimentos de tempo, o grupo optou por implementar uma estratégia baseada em timeouts e tentativas. Caso o recetor não receba alguma das mensagens, seja de sucesso, seja do conteudo, este iniciará uma nova tentativa de conexão e transferência. Cada tentativa terá um timeout de 5 segundos. O resultado final deste protocolo revelou-se eficaz quando os pacotes são enviados para o nodo referente á ligação com perdas. Já ao contrário, não se revelou eficaz, acontecendo ainda perdas.

4 Implementação

4.1 Descrição Detalhada do Funcionamento do Programa

A implementação do FS Track Protocol, realizada na linguagem de programação Python, reflete uma estrutura robusta e modular para o estabelecimento de comunicação eficiente entre FS_Nodes e o FS_Tracker. A seguir, abordamos mais detalhes sobre o funcionamento do programa, destacando as escolhas de implementação, as estruturas de dados relevantes e fornecendo uma visão aprofundada sobre as operações principais.

4.2 Linguagem de Programação Utilizada

A implementação do FS Track Protocol foi conduzida utilizando a linguagem de programação Python. A escolha desta linguagem foi motivada pela sua notável expressividade, facilidade de compreensão e amplo suporte para programação de redes. O Python oferece recursos robustos para o desenvolvimento eficaz de aplicações distribuídas, alinhando-se com os requisitos do projeto.

4.3 Estruturas de Dados e Algoritmos Relevantes

Estruturas de Mensagens Protocolares: O protocolo é estruturado em torno de mensagens bem definidas, cada uma destinada a uma operação específica (registo, atualização, pedido de localização). A sintaxe e semântica dessas mensagens são rigorosamente definidas para facilitar o processamento consistente.
Utilização de Locks para Acesso Concorrente: Dada a natureza concorrente do ambiente P2P, foram incorporadas práticas de segurança utilizando locks. O uso desses locks garante a integridade dos dados compartilhados entre threads, mitigando potenciais problemas de concorrência.

4.4 Estrutura do Programa

A implementação do FS Track Protocol, conduzida na linguagem de programação Python, é caracterizada pela coesão e modularidade, refletindo uma abordagem cuidadosa no design e organização do código. Nesta seção, exploramos a estrutura do programa, destacando as principais classes e seu papel no funcionamento do protocolo.

FS_Tracker: Esta classe desempenha um papel central no ecossistema P2P, atuando como um ponto central para registo, atualização e pedidos de localização. Esta classe é responsável por gerir a informação sobre os FS_Nodes registados, mantendo um registo atualizado da rede.

Métodos Relevantes:

- `start()`: Inicia o servidor FS_Tracker, permitindo a receção de pedidos de registo e atualização dos FS_Nodes.
- `handle_register()`: Processa os pedidos de registo, atribuindo UUIDs únicos aos FS_Nodes e mantendo a informação correspondente.
- `handle_update()`: Lida com os pedidos de atualização enviados pelos FS_Nodes, atualizando a informação sobre os ficheiros e blocos que possuem.
- `handle_locate()`: Responde aos pedidos de localização de ficheiros, fornecendo informações sobre os FS_Nodes que possuem blocos do ficheiro em questão.

FS_Node: A classe `FS_Node` representa um nó na rede P2P, atuando como cliente e servidor simultaneamente. Esta classe é responsável pelo registo no FS_Tracker, atualização periódica deste registo e gestão das operações relacionadas com transferência de ficheiros.

Métodos Relevantes:

- O método `register` é responsável por registar o FS_Node no Tracker. Ele é chamado quando o nó é inicializado e ainda não possui um UUID atribuído. O processo de registro envolve a conexão com o Tracker, envio de uma mensagem de registro (`'type': 'REG'`), e a receção do UUID atribuído pelo

Tracker. O método gerencia essas interações de rede e exibe mensagens de sucesso ou falha com base na resposta do Tracker. Após o registro bem-sucedido, o UUID atribuído é armazenado no atributo `self.uuid`.
- O método `locate` é usado para localizar informações sobre a localização de blocos específicos de um arquivo no sistema P2P. Ele envia uma mensagem ao Tracker com o tipo `'LOC'`, informando o UUID do nó, o nome do arquivo e, opcionalmente, o número do bloco desejado. O Tracker responde com informações sobre os blocos disponíveis e seus locais. As informações são impressas na consola para visualização do utilizador. O método lida com casos em que o arquivo ou bloco não é encontrado.
- O método `update` é responsável por atualizar a lista de arquivos e blocos que o FS_Node possui. Ele percorre a pasta local definida em `self.ffolder`, identifica os arquivos e seus blocos existentes, e envia essa informação ao Tracker. O Tracker responde com um status de sucesso ou falha. O método é útil para manter o Tracker informado sobre os arquivos disponíveis e os blocos que o nó pode partilhar com outros nós na rede.

Comunicação entre FS_Node e FS_Tracker A comunicação entre FS_Node e FS_Tracker ocorre através de mensagens protocolares definidas, estabelecendo assim um canal eficiente para registo, atualização e consulta de informações na rede P2P.

Mensagens Protocolares: O protocolo FS Track define várias mensagens protocolares para facilitar a comunicação entre os nós da rede peer-to-peer (P2P). Abaixo estão algumas das mensagens mais relevantes no contexto do programa:

REG - Registo do Nó A mensagem de registo (`REG`) é enviada por um nó ao tracker para se registar na rede. O tracker responde com um UUID único atribuído ao nó e seu endereço IP.

UPT - Atualização de Blocos do Nó A mensagem de atualização (`UPT`) é utilizada para informar o tracker sobre os blocos disponíveis no nó. O nó envia um dicionário contendo informações sobre os arquivos e seus blocos, e o tracker atualiza seus registros.

LOC - Localização de Blocos/Arquivos A mensagem de localização (`LOC`) é enviada por um nó ao tracker para solicitar informações sobre a localização de um arquivo específico ou de um bloco dentro desse arquivo. O tracker responde com detalhes sobre os nós que possuem o arquivo/bloco.

DIS - Desconexão do Nó A mensagem de desconexão (`DIS`) é enviada por um nó ao tracker quando o nó deseja sair da rede. O tracker remove o nó dos seus registos.

REQ - Pedido de Bloco A mensagem de pedido (`REQ`) é enviada por um nó para solicitar um bloco específico de outro nó. Essa mensagem é utilizada durante o processo de download de blocos entre nós.

Status das Respostas As respostas do tracker ou de outros nós incluem frequentemente um campo de `Status` indicando se a operação foi bem-sucedida. As respostas podem conter detalhes adicionais, como mensagens de erro em caso de falhas.

Estas mensagens protocolares são cruciais para o funcionamento do protocolo FS Track, garantindo uma comunicação eficiente e precisa entre os nós da rede P2P. Cada mensagem desempenha um papel específico no contexto do registo, atualização, localização e transferência de blocos e arquivos.

Gerador de Identificadores Únicos: Para garantir a unicidade dos identificadores atribuídos aos nós na rede, foi implementada a classe `UniqueIdGenerator`. Esta classe faz uso da biblioteca `uuid`, que é responsável por gerar UUIDs (*Universally Unique Identifiers*). Um UUID é um identificador único padronizado globalmente, representado por 32 caracteres hexadecimais divididos em cinco grupos, separados por hifens.

- **Finalidade:** O UUID é projetado para ser único em ambientes distribuídos e ao longo do tempo, o que o torna adequado para a atribuição de identificadores únicos a entidades como nós em redes P2P.
- **Geração Aleatória:** A função principal da biblioteca `uuid` é gerar UUIDs de forma aleatória, garantindo uma probabilidade extremamente baixa de colisões entre identificadores.
- **Unicidade Global:** A padronização e as práticas de geração do UUID asseguram que esses identificadores sejam únicos em uma escala global, o que é crucial para evitar ambiguidades e conflitos em sistemas distribuídos.

A classe `UniqueIdGenerator` implementa métodos essenciais para a geração e gestão de UUIDs exclusivos:

- `__init__`**:** O método de inicialização cria um conjunto vazio, `generated_ids`, que será utilizado para rastrear os UUIDs já gerados.
- `generate_unique_id`**:** Este método é responsável por gerar um UUID exclusivo. Utiliza a função `uuid.uuid4()` para criar um UUID aleatório. Em seguida, verifica se o UUID gerado já existe no conjunto `generated_ids`. Caso positivo, continua gerando novos UUIDs até encontrar um que seja realmente único. Uma vez obtido um UUID exclusivo, ele é adicionado ao conjunto `generated_ids` antes de ser retornado como uma string.

Esta abordagem garante a unicidade dos identificadores atribuídos a cada nó na rede, evitando colisões e assegurando a integridade das operações de registo e identificação dos nós.

Transferência de Blocos: No âmbito da classe `FS_Node`, duas funções fundamentais são implementadas para as operações de download e upload de blocos de ficheiros: `download_block` e `upload_block`. Essas funções desempenham papéis críticos no sistema peer-to-peer (P2P), permitindo a transferência eficiente de dados entre nós.

- `download_block(self, block_info)`: Esta função é acionada quando um nó solicita a transferência de um bloco específico de um ficheiro. Ela recebe

informações sobre o bloco desejado, incluindo o identificador único do nó que possui o bloco, o endereço IP e a porta desse nó. O nó então estabelece uma conexão efêmera com o nó fonte e solicita o bloco desejado. Após a receção bem-sucedida do bloco, a função é responsável por armazená-lo localmente na pasta partilhada.

– `upload_block(self, block_info)`: Por outro lado, a função `upload_block` é invocada quando um nó está a partilhar blocos de ficheiros com outros nós. O nó monitora permanentemente a porta UDP designada para pedidos de blocos e responde a esses pedidos enviando o bloco solicitado. A função verifica a integridade do bloco antes de transmiti-lo e atualiza o FS_Tracker sobre as alterações na lista de blocos partilhados pelo nó.

Essas funções desempenham um papel crucial na dinâmica de partilha de ficheiros do sistema P2P. Através da implementação eficiente de operações de download e upload, o nó contribui ativamente para a robustez e disponibilidade dos ficheiros na rede, seguindo o paradigma de partilha descentralizada característico de sistemas peer-to-peer.

Transferência de Arquivos: O método `getFile` é utilizado para fazer transferir blocos ou arquivos inteiros de outros nós na rede P2P. Ele recebe como entrada o nome do arquivo desejado (`file_name`) e o número do bloco a ser transferido (`bloco`). Se `bloco` for zero, o método transfere todos os blocos disponíveis para o arquivo.

O método inicia criando uma lista de threads para lidar com os downloads paralelos de blocos. Para cada bloco disponível no arquivo, o método verifica se o bloco já está localmente armazenado. Se o bloco estiver ausente localmente, uma thread é criada para iniciar o processo de download desse bloco específico. O download é realizado através do método `download_block`, que lida com a comunicação com o nó que possui o bloco.

O método `getFile` utiliza internamente o método `download_block` para realizar os downloads específicos de cada bloco, assegurando uma implementação modular e eficiente do protocolo P2P.

Após a conclusão dos downloads, o método atualiza as informações locais sobre os arquivos e blocos do nó chamando o método `update`.

O método é fundamental para permitir que o nó obtenha os blocos necessários para reconstruir os arquivos desejados, contribuindo assim para a eficácia do protocolo P2P na partilha de ficheiros.

Estratégia para Lidar com Perdas de Pacotes no Envio de Blocos Durante o envio de blocos, podem ocorrer perdas de pacotes devido a vários fatores, como instabilidade na rede. Para garantir a confiabilidade na transmissão, implementamos uma estratégia de retransmissão em caso de perda de pacotes. A estratégia consiste nos seguintes passos:

Envio com Retransmissão: A função `handle_loss` é responsável por enviar uma solicitação de bloco (`REQ`) para o nó que possui o bloco desejado. Em caso de perda de pacotes, a função tenta reenviar a solicitação por até cinco vezes, esperando uma resposta após cada tentativa. Se a resposta for recebida com sucesso, a função retorna os dados do bloco.

Tratamento de Falhas: Caso a função `handle_loss` não obtenha sucesso após várias tentativas, ela retorna `False`, indicando uma falha no processo. A função `handle_loss2` é uma camada adicional que reutiliza a lógica da `handle_loss` e retorna `True` apenas se a resposta indicar sucesso (`Status: 'Success'`).

Retransmissão Adicional: Após a retransmissão bem-sucedida, a função `download_block` continua a tentar receber o bloco, garantindo a confiabilidade da transmissão.

Timeout: O código possui mecanismos de timeout para limitar o tempo de espera por uma resposta. Se o limite de tempo for excedido, o código irá repetir-se 5 vezes. Caso o limite de tempo seja excedido essas 5 vezes, considera a operação como falha.

Objetivo: Esta estratégia visa melhorar a confiabilidade e a integridade do sistema, especialmente em ambientes de rede menos estáveis. No entanto, é importante observar que a implementação pode exigir ajustes adicionais para cenários específicos ou para otimizar o desempenho em redes mais rápidas e confiáveis.

5 Testes e Resultados

Nesta fase, conduzimos uma série de testes para avaliar a eficácia e robustez da implementação do FS Track Protocol. Os testes foram projetados para abranger diversos cenários operacionais, desde o registro inicial do nó até a localização de arquivos e blocos específicos.

Os resultados revelaram que o nó respondeu de maneira adequada a todas as solicitações, fornecendo informações precisas sobre a localização dos arquivos e blocos quando disponíveis e indicando corretamente a ausência em outros casos.

5.1 Discussão e Possíveis Melhorias

Os testes realizados demonstraram que o FS Track Protocol está funcional, porém não da melhor forma. Algumas melhorias e ajustes podem ser considerados para aprimorar ainda mais o sistema:

1. **Segurança:** Implementar medidas de segurança adicionais, como criptografia de mensagens, para proteger a integridade e a confidencialidade da comunicação entre o nó e o Tracker.

2. **Gestão de Falhas:** Reforçar estratégias para lidar com falhas na comunicação, como timeouts e tentativas de reconexão, garantindo a estabilidade do sistema em condições adversas.
3. **Otimização de Desempenho:** Avaliar e otimizar a eficiência do protocolo em termos de uso de recursos, garantindo um desempenho escalável, especialmente em redes P2P mais extensas.
4. **Domain Name Resolution:** No decorrer desta fase do trabalho, deparamonos com problemas na utilização dos serviços named/bind9, pelo que não ficaram implementadas na versão final. Mesmo assim, os ficheiros de configuração das zonas e domínios estão anexados no ficheiro relativo ao códigofonte do programa

Essas considerações visam aprimorar a implementação do protocolo, proporcionando uma base sólida para futuras iterações e extensões do sistema.

6 Conclusão

Ao longo do desenvolvimento e testes, identificaram-se alguns desafios e áreas que requerem atenção para melhorias futuras. É crucial abordar essas questões para garantir a robustez e eficácia contínuas do protocolo.

Um dos desafios encontrados está relacionado às perdas durante a transferência, observando-se que o protocolo funciona corretamente quando o nó receptor é o responsável pela recuperação das perdas. No entanto, ao inverter o papel, surgem questões que precisam ser investigadas e resolvidas. Aprimorar a lógica de recuperação de perdas em ambas as direções pode ser uma área de foco para futuras iterações do protocolo.

Uma limitação identificada está associada ao tipo de ficheiros suportados. Até o momento, a transferência parece ser eficaz apenas para ficheiros de texto com conteúdo textual. A expansão do suporte para diferentes tipos de ficheiros, incluindo binários e formatos mais complexos, é uma consideração importante para garantir a versatilidade do sistema.

O caminho para melhorias futuras envolve a colaboração contínua e a iteração sobre o código existente. Abordar esses desafios não apenas fortalecerá a eficácia do protocolo, mas também consolidará sua posição como uma contribuição valiosa para o domínio das redes P2P.