Comunicação por Computadores: Desenho e Implementação de um Protocolo P2P para Partilha de Ficheiros

José Miguel Silva[1], Miguel João Silva[1] , Ricardo Jorge Branco[1]

{José Miguel Silva a54766, Miguel João Silva a49391, Ricardo Branco a61075}

Resumo. Durante o decorrer deste relatório, tentamos expor da forma mais sucinta e clara o trajecto efectuado, a forma com foi pensado e implementado todas funcionalidades pedidas no enunciado bem como alguns aspectos mais pormenorizados do funcionamento da nossa aplicação P2P, as suas características, algoritmos e implementações para que seja possível efectuar e controlar transferências e procuras de ficheiros, o controlo de vizinhança numa rede *Peer-to-Peer* usando o protocolo UDP.

Palavras-Chave: *Peer-to-Peer* (P2P), Core, JAVA, UDP.

1 Introdução

Este trabalho visa o desenvolvimento, durante as aula teórico-práticas da disciplina de Comunicações por Computador e horas de trabalho extra-aula, e implementação de uma aplicação P2P para a partilha de ficheiros suportado pelo protocolo UDP. Durante a 1ª Fase de desenvolvimento o objectivo foi implementar as primitivas (Hello, e chunk), bem como os comandos GET e STATUS, a segunda fase consiste em juntar aos comandos já existentes a primitiva search e os comandos REGISTER e FIND. A utilização de varias ferramentas como meio de facilitar este desenvolvimento é fundamental. Para tal, utilizaremos para o desenvolvimento da aplicação Net Beans, a linguagem Java e também será usado o Core para testes. O Core cria a rede virtual com equipamentos de rede, assim é possível virtualizar a rede p2p com todos os seus componentes.

2 Especificação do Protocolo

Rede *Peer-to-Peer* (P2P) tal como a conhecemos é uma arquitectura de rede de computadores onde cada um dos nodos da mesma pode funcionar tanto como cliente como servidor. em contraste com as redes típicas de cliente servidor. Os nodos fazem parte dos recursos da rede e podem servir para um mero armazenamento de dados como também podem doar poder de computação à rede. O protocolo que iremos desenvolver irá assentar nestes pressupostos., ou seja a aplicação prevê que um servidor reconheça servidores vizinhos, um servidor comunica com os seus vizinhos, de certo modo, o servidor faz pedidos aos seus vizinhos ou responde a pedidos enviados pelos vizinhos, um cliente que deseje algo da rede P2P pede a um servidor e o servidor será encarregado de fazer todas as operações e no final responde ao cliente com os resultados, ou seja, de certo modo o cliente não tem ideia de onde podem vir os resultados pois tudo isso é tratado pelo servidor.

Utilizaremos o Core para simular uma rede P2P facilitando assim a fase de testes não precisando assim de uma rede física para testar se a aplicação funciona e identificar as suas lacunas.

A aplicação desenvolvida ao inicializar obtém uma lista de vizinhos, através da leitura do seu ficheiro de configuração ou através de um broadcast para a vizinhança, desse modo é obtida uma lista de peers disponíveis para comunicar e trocar ficheiros. A lista de vizinhos é bastante importante para o bom funcionamento da aplicação, pois caso um vizinho deixe de responder é necessário ter conhecimento disso para que não se perca tempo a tentar comunicar com ele pois é certo que não irá responder.

Na aplicação todas as comunicações serão efectuadas com suporte do protocolo UDP.

2.1 Primitivas de comunicação

Foram implementadas as primitivas: Hello, Chunk, Search e os commandos Status, Get,Register e Find .

2.1.1 1ª Fase de implementação

Durante a primeira fase de desenvolvimento do protocolo foram implementadas varias primitivas, entre elas:

Hello(IP:porta) – Primitiva utilizada para fazer a verificação do estado de um dado nodo. A resposta a esta primitiva por parte do nodo questionado torna o seu estado como activo na tabela de nodos vizinhos do nodo que o questionou facilitando assim uma posterior ligação de ambos. A primitiva Hello recebe o IP de destino e a porta para que possa testar a actividade ou não do vizinho. Na nossa aplicação o Hello retorna também uma lista de ficheiros que o peer possui e uma lista de ficheiros que conhece, isto acontece para fornecer métodos melhores para quando se pretender transferir um ficheiro.

Chunk(IP:porta, offset, size) – Esta primitiva tem como utilidade o pedido de

transferência dum chunk de tamanho size a começar no byte indicado pelo offset, sendo que o primeiro byte dum ficheiro é o offset 1. Se por acaso o valor do offset for 0 isto significa que o pretendido é o chunk com a meta-informação do ficheiro e neste caso excepcional o valor do size é ignorado. Em caso de a resposta obtida estar correcta o peer é confirmado mais uma vez como ainda estando valido.

Search(IP:porta, text, max_n_matches) – A Primitiva é usada para pedir uma lista de matches a um termo/texto nas descrições e nomes dos ficheiros com cópia no sistema P2P local. No máximo incluem-se max_n_matches entradas (label, nome original, descrição, tamanho). O servidor ao receber uma resposta válida inclui os resumos (label e identificação do servidor) na cache de resultados de procuras que podem depois ser usados nas primitivas chunk.)

Os comandos :

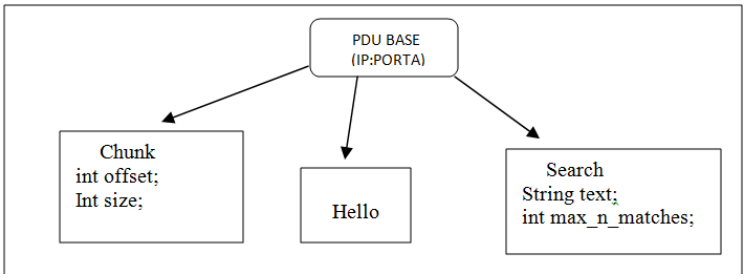
Comando Get – Este comando indica que o cliente pretende receber um ficheiro, ou seja, pretende adquirir por complete um ficheiro da rede P2P não se importando de onde e como ele vem. Quando um servidor recebe um Get já possui a localização ou localizações do ficheiro na rede e sabe que ele próprio não possui o ficheiro, assim sendo, o servidor calcula o número e tamanho dos chunks necessários para efectuar a transferência nas melhores condições com as limitações impostas pelo enunciado.

Comando Find – O comando find é o comando que trata de encontrar um ficheiro na rede P2P. É um comando muito importante pois é ele que permite tirar partido da distribuição de ficheiros repetidos na rede, e assim , ao encontrar várias ocorrências do mesmo ficheiro na rede pode possibilitar uma transferência de ficheiros rápida.

Comando Register – Este comando é usado para registar um ficheiro novo no filesystem do servidor, nesse sentido é a forma de relacionar a posse de ficheiros no servidor.

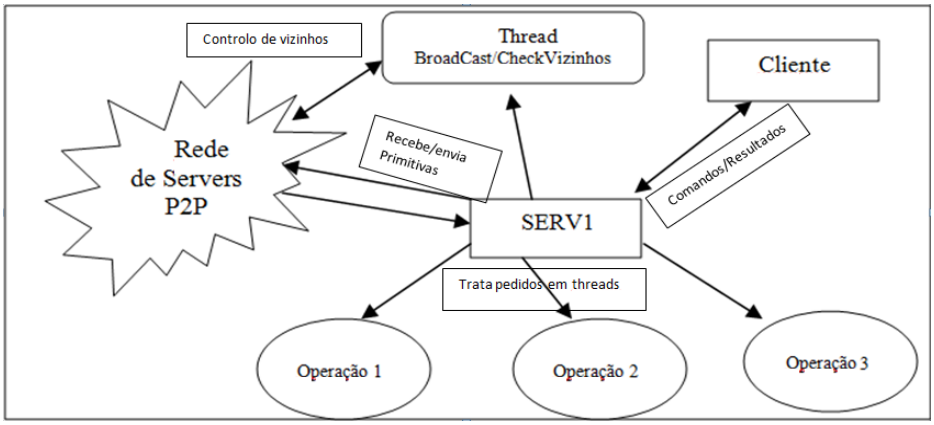
2.2 Formato das mensagens protocolares (PDUs)

A nossa aplicação prevê que cada comunicação possua pelo menos um header com ip e porta, ou seja, sempre que um servidor faz um pedido pelo menos utiliza o PDU base que é (IP: PORTA), mas como existem várias formas de comunicar, nomeadamente hello,chunk,search cada tipo de comunicação adiciona mais campos ao PDU base, ou seja, consideramos PDU como uma classe abstracta e as primitivas (Search,Chunk,Hello) herdam as características de PDU e acrescentam ao PDU base novas informações.



3 Implementação

Na implementação do trabalho, tentamos reproduzir ao máximo o que acontece num serviço p2p normal, nesse sentido implementamos um servidor capaz de responder a vários pedidos em simultâneo. Este servidor é também capaz de periodicamente e sem interromper o seu normal funcionamento reconhecer novos vizinhos através de um *broadcast* e de identificar *peers* que já não se encontrem activos e assim actualizar a lista de vizinhos.



Como se pode ver na imagem a nossa aplicação possui *threads* que servem para tratar diferentes pedidos a um servidor. Enquanto existe uma thread que trata da verificação da nova vizinhança e actualização da vizinhança já existente. O Cliente ao fazer um pedido ao servidor espera pelo resultado, para não acontecer situações onde um cliente pede em simultâneo o mesmo ficheiro, e isso iria resultar em ficheiros repetidos, nesse sentido antes de fazer novos pedidos o utilizador terá que aguardar pela conclusão do pedido anterior.

Os comandos e primitivas foram elaborados de maneira a tentar sempre possuir a máxima performance possível, por exemplo, um simples comando *hello* em vez de apenas servir para indicar se certo *peer* está activo, envia também os ficheiros que possui e conhece, assim é possível possibilitar uma maior rapidez na procura de ficheiros, pois já conhecemos os ficheiros que existem na proximidade. O *Find* percorre a rede há procura de ocorrências do termo pretendido, sendo que o *Find* é invocado pelo pedido *Search* enviado por um cliente. O *Get* será o comando que será responsável pela transferência de um ficheiro, tudo o que involve uma transferência será tratado pelo *Get* e será ele que no final indica que o ficheiro está transferido ou

não, algo que é importante referir é que o *Get* calcula o tamanho dos *chunks*, de maneira a ter o máximo de sucesso nas transferências e a usufruir da rede P2P.

4 Testes e Resultados

5 Conclusões e trabalho futuro

Após a conclusão deste projecto, podemos concluir que grande parte dos seus objectivos iniciais foram cumpridos com sucesso. A implementação numa primeira fase passou pela codificação das primitivas (Search, chunk e Hello) e do PDU e numa fase posterior dos métodos (Get, Find e Register). Este grupo surgiu da fusão de dois grupos, e ambos desenvolveram em separado a primeira fase do projecto, juntando todo o grupo para esta segunda fase, aproveitamos o que de melhor havia em cada projecto na primeira fase deste e desenvolvemos já como um único grupo esta segunda fase. Durante os testes efectuados ao longo do desenvolvimento e após a conclusão do projecto conseguimos com sucesso procurar e encontrar ficheiros bem como adquiri-los.

Ao longo do desenvolvimento do projecto fomo-nos deparando com variados problemas que fomos resolvendo, pensando nós, da melhor maneira. Um deles tem a ver com o find, tivemos dificuldade em tratar a possível procura infinita de ficheiros ou seja, como na rede todos seriam vizinhos a possibilidade de existir diferentes caminhos poderia criar o problema de ter o comando find a ser enviado várias vezes para o mesmo sitio, a maneira de controlar isso foi realmente a tarefa que mais problemas nos criou.

Como trabalho futuro seria importante a implementação de alguns métodos extra, como foram sugeridos no enunciado, que trariam sem duvida uma mais valia a todo o trabalho. Uma interface gráfica que facilita-se a interpretação e interacção com a mesma seria também uma mais valia pois tornaria todo o sistema mais *"user friendly"*.

Referências

[1] http://pt.wikipedia.org/wiki/Peer-to-peer
[2] http://pt.wikipedia.org/wiki/User_Datagram_Protocol
[3] http://docs.oracle.com/javase/1.5.0/docs/guide/javadoc/index.html