Trabalho Um de Programação Distribuída 2019/1

Felipe Garrastazu Guedes[1], Mateus Haas[1], Gabriel Kurtz[1],

[1]Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil

felipe.guedes@edu.pucrs.br, mateus.haas@edu.pucrs.br, gabriel.kurtz@edu.pucrs.br

Resumo. *Este artigo tem como objetivo apresentar os resultados obtidos a partir da aplicação desenvolvida para o primeiro trabalho da disciplina de programação distribuída. Será mostrado em detalhe todos os aspectos da implementação e o racional que levou o grupa a tomar tais decisões, em especial os aspectos que são relevantes para a programação distribuída. Ao final, é debatido o resultado obtido com a pesquisa, as dificuldades encontradas pelo grupo, propostas para melhorias e futuros estudos a partir da mesma temática.*

1. Introdução

O objetivo do trabalho é auxiliar no aprendizado dos conteúdos vistos até agora na disciplina de programação distribuída, como modelo cliente/servidor, RPC, RMI, P2P e comunicação em grupo.

2. Problema

Deve ser implementada uma estrutura de Nodos e Supernodos que permita a transferência de arquivos (recursos) utilizando o modelo Peer-to-Peer(P2P). Os nodos são os clientes finais, capazes de enviar e receber arquivos. Eles conectam aos

Supernodos, que são o esqueleto da estrutura de dados. O programa deve permitir a execução nas duas formas distintas (Nodo ou Supernodo).

3. Solução

Há diversas maneiras possíveis de solucionar o problema proposto pelo professor em sala de aula. O principal requisito era a implementação da comunicação entre os nodos “servidores” através de multicast. Para isso foi utilizado o JGroups, biblioteca disponível em Java que permite implementar em mais alto nível esse tipo de comunicação. Esta implementação está visualmente representada na Figura 1, ponto “2 Multicast”.

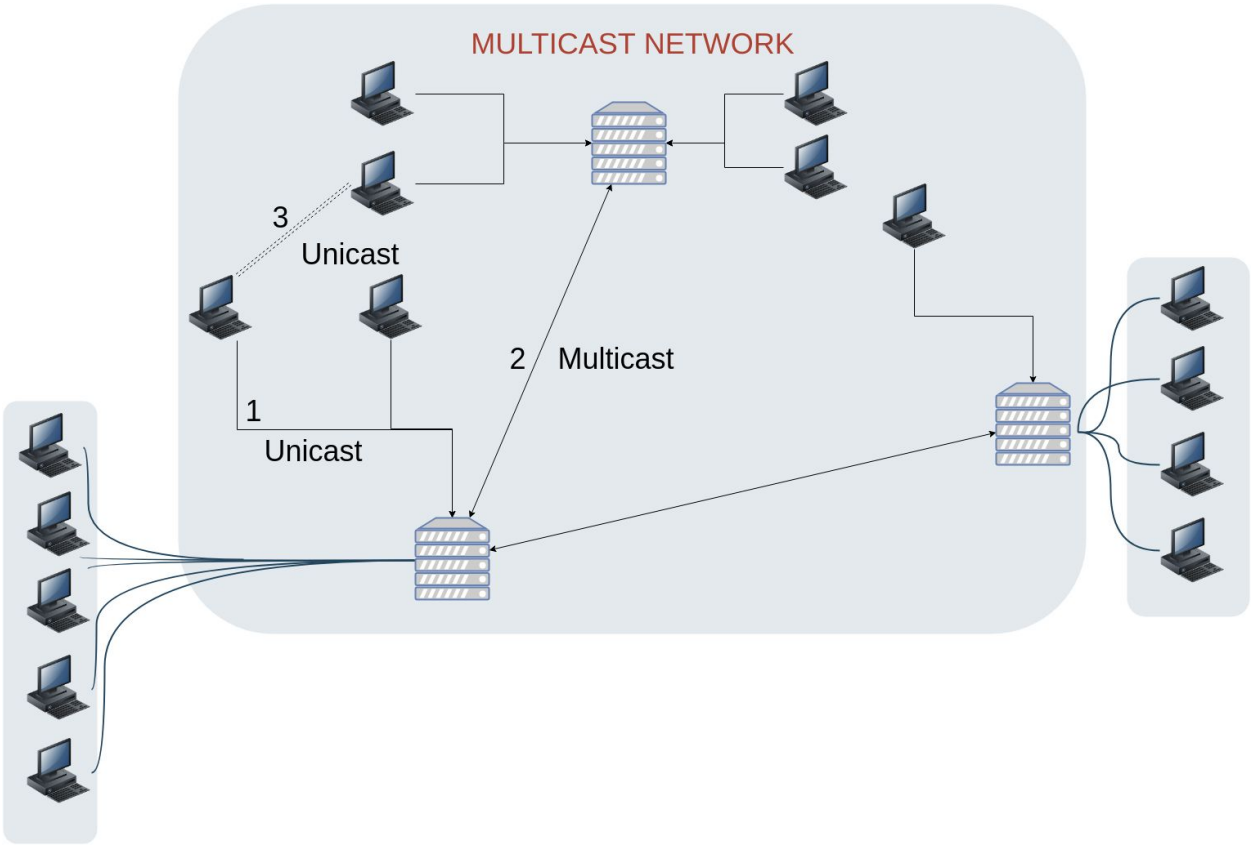

Figura 1. Visualização da estrutura da rede

Para as demais requisições foi necessário, na arquitetura implementada, assegurar a transferência dos pacotes requisitados. Foram utilizados sockets TCP (conforme a Figura 2) para implementar a comunicação 1 e 3, da Figura 1. A conexão 1 é unicast e permanente, ou seja, o socket permanece aberto durante toda a execução do programa, o que permite a comunicação do cliente com o servidor a qualquer instante.

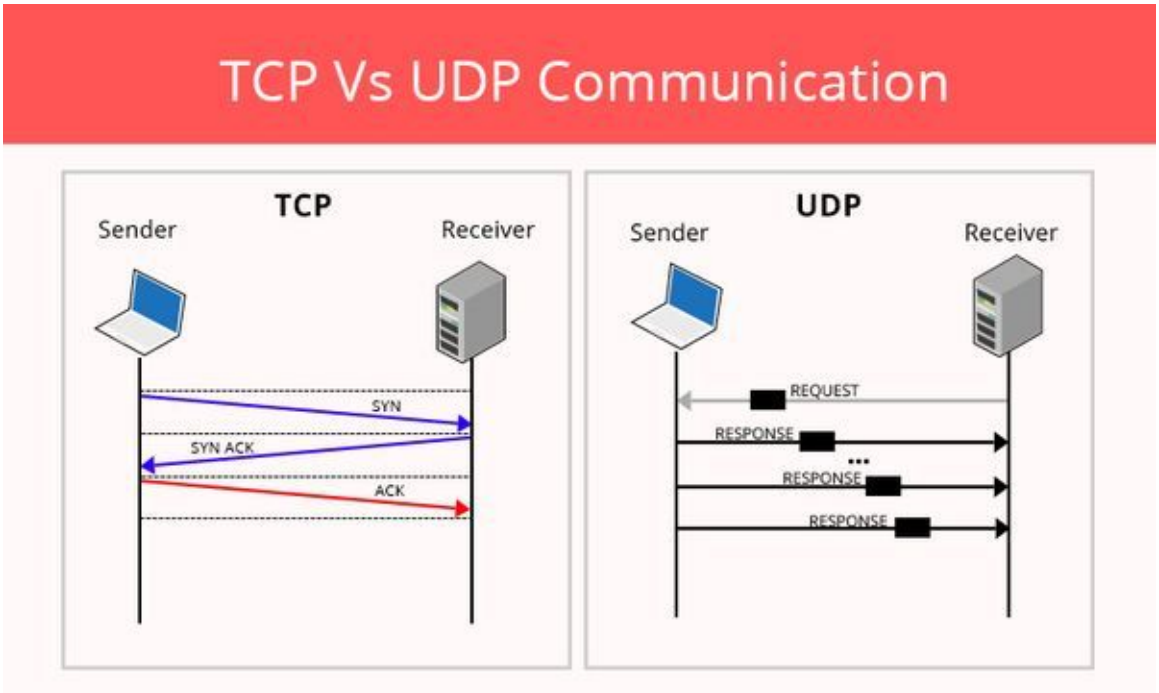

Figura 2. TCP vs UDP

Já a comunicação *peer to peer* (P2P) ocorre de forma ad-hoc, ou seja, apenas durante a transferência de arquivos. A conexão é fechada logo que essa transferência é concluída, pois não é mais necessário manter o socket aberto uma vez que toda a transação foi concluída e não há mais razão para isso, assim otimizando o uso dos recursos.

3.1 Estrutura de Rede

A arquitetura implementada foi híbrida. Os supernodos guardam os metadados da rede. Cada vez que um cliente conecta ao supernodo (em modo unicast), este verifica quais arquivos o cliente possui disponíveis para compartilhar, e estes arquivos são adicionados a uma lista global durante a comunicação entre os supernodos (modo multicast), até serem removidos quando aquele cliente desconecta, gerando uma lista permanente de todos os arquivos que estão disponíveis na rede.

Além dos recursos(arquivos), os supernodos guardam também a própria informação dos clientes conectados, e verifica com frequência quais conexões ainda estão ativas escutando um Heartbeat que cada cliente envia (o que talvez tenha sido uma escolha redundante devido ao uso da arquitetura TCP nas conexões unicast).

A partir das informações de conexão, o supernodo direciona os nodos a conectarem em P2P, transferindo os dados de forma direta, evitando assim sobrecarregar os supernodos. A Figura 3 demonstra esta estrutura:

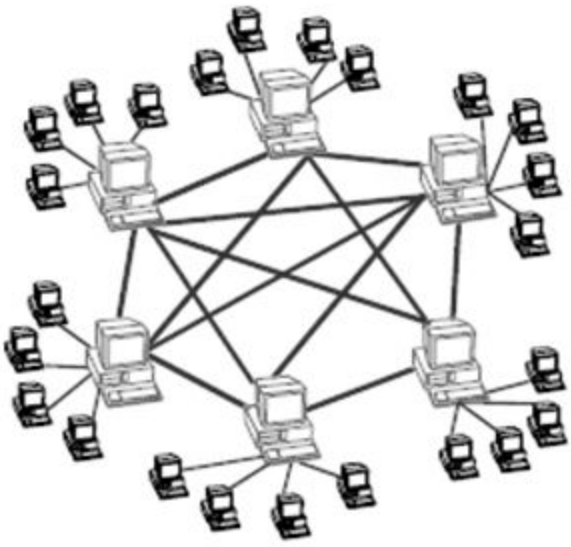

Figura 3. Exemplo de arquitetura Hibrida P2P

3.2 Estrutura Lógica

O programa foi escrito na linguagem Java. O ponto de entrada é a classe Main, onde é recebido o argumento informando o modo de execução (nodo ou supernodo) além das informações de rede (portas e, quando pertinente, IP).

Argumentos para rodar em modo supernodo:

main.java supernode [porta unicast] [porta multicast]

Para rodar em modo cliente:

main.java clientnode [IP do Supernodo] [porta do Supernodo] [porta P2P]

Cada um dos modos possui uma classe equivalente na pasta Server: SuperNode e ClientNode, onde ocorre a lógica principal de execução.

A classe SuperNode possui três funções principais:

- Receber, manter e encerrar conexões dos clientes que conectarem através do seu IP, incluindo escutar e tratar os heartbeats de cada cliente.
- Receber, informar e apagar as informações de recursos (arquivos) que aqueles clientes compartilham enquanto estiverem conectados, e possibilitar a conexão P2P entre os clientes caso seja solicitada transferência de recursos.

Em ambos os casos, as informações são compartilhadas com os demais supernodos sempre que necessário, de modo a gerar uma rede única, coesa e plenamente visível e funcional independente de qual seja o supernodo que recebe a conexão de cada cliente.

A classe ClientNode, por sua vez, proporciona um pequeno menu com os seguintes casos de uso:

- Solicitar a lista de arquivos disponíveis na rede (e, posteriormente, baixar uma coleção destes arquivos caso assim desejar, que serão gravados na pasta Responses).
- Ativar o modo Debug, recebendo informações mais detalhadas sobre as comunicações de rede e algumas outras ocorrências do programa.
- Encerrar o programa.

Há também algumas classes utilitárias no diretório *helpers* que auxiliam a modularizar algumas tarefas reaproveitáveis:

- Terminal: responsável pela parte do terminal - logger(prints), gerenciamento do modo Debug, inputs do menu.
- MD5: auxilia com a Stream de transferência de pacotes de dados (datagramas).
- Timer: auxilia com os timeouts.

3.3 Estruturas de Dados

No diretório *dto* estão algumas das estruturas que permitem ações remotas através do RPC do JGroups. A superclasse NodeCommunication é herdada pelas comunicações específicas de cada hierarquia de nodos: Client2Super, Super2Client, Super2Super.

Na pasta *entity,* encontra-se a classe Node que ajuda a criar a estrutura dos nodos, bem como FileData e FileDataResponse que constituem os recursos a serem transferidos, e manipulam a classe MD5 citada anteriormente.

4. Conclusão

O trabalho demonstra conhecimentos bastante interessantes e atuais. Alguns exemplos de aplicações são a própria transferência de arquivos P2P (Torrents) e servidores de jogos online.

Acreditamos ter aplicado corretamente diversas das técnicas aprendidas em aula e ter gerado um programa com uma boa estrutura. Sua funcionalidade foi testada

no laboratório com sucesso, cobrindo os casos de uso solicitados e obedecendo a arquitetura proposta..

Referências:

ORACLE – **Writing the Server Side of a Socket** – Acessado em 01 abril 2019; disponível em: https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html

ORACLE – **Reading from and Writing to a Socket** – Acessado em 01 abril 2019; disponível em: https://docs.oracle.com/javase/tutorial/networking/sockets/readingWriting.html

ORACLE – **MulticastSocket** – Acessado em 03 abril 2019; disponível em: https://docs.oracle.com/javase/7/docs/api/java/net/MulticastSocket.html

SANTIAGO – **Criar datagram socket em UDP e socket TCP em Java** – Acessado em 03 abril 2019; disponível em: https://www.kelvinsantiago.com.br/criar-datagram-socket-em-udp-e-socket-tcp-em-java/

MAHRSEE – **Working with UDP Datagram Sockets in Java** – Acessado em 03 abril 2019; disponível em: https://www.geeksforgeeks.org/working-udp-datagramsockets-java/

HAN – **Distributed hybrid P2P networking systems** – Acessado em 04 abril 2019; disponível em: https://link.springer.com/article/10.1007/s12083-014-0298-7

HAN – **Benefiting from Data Mining Techniques in a Hybrid Peer-to-Peer Network** – Acessado em 04 abril 2019; disponível em: https://www.researchgate.net/publication/224366327_Benefiting_from_Data_Mining_Techniques_in_a_Hybrid_Peer-to-Peer_Network

JIJU - **TCP/IP VS UDP: WHAT'S THE DIFFERENCE?** – Acessado em 04 abril 2019; disponível em: https://www.colocationamerica.com/blog/tcp-ip-vs-udp