UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE CIÊNCIAS DA COMPUTAÇÃO
REDES DE COMPUTADORES
PRIMEIRO SEMESTRE DE 2017

# Trabalho Prático 3


**Gabriel de Oliveira Campos Pacheco** **Matrícula: 2013062898**
**Guilherme Augusto de Sousa** **Matrícula: 2013062944**
**João Paulo Sacchetto Ribeiro Bastos** **Matrícula: 2013073440**


## Introdução

Redes P2P(peer-to-peer) são formatos que possuem como principal traço a descentralização das funções convencionais de rede, ou seja, o computador de cada usuário conectado acaba por realizar funções de servidor e de cliente ao mesmo tempo, como pode ser visto na figura 1. É um paradigma de distribuição atraente porque todo o conteúdo é transferido diretamente entre pares comuns, sem passar por servidores de terceiros. Assim, essa aplicação tira proveito dos recursos como largura de banda, armazenamento e CPU de um grande conjunto de pares – às vezes milhões deles.

O objetivo dessa arquitetura de redes é a transmissão de arquivos e seu surgimento possibilitou o compartilhamento em massa de músicas e filmes. Para a grande maioria dos usuários atuais, utilizar uma rede P2P é uma maneira de obter todo tipo de conteúdo sem custo e de maneira rápida e simples. A possibilidade de se compartilhar músicas, filmes, jogos, etc. tem levado cada vez mais pessoas a utilizarem softwares clientes dessas redes, gerando mais conteúdo e criando um ciclo que só faz aumentar o seu sucesso.

Neste trabalho é apresentada uma implementação básica de um sistema de armazenamento chave-valor do tipo peer-to-peer em Python, onde os programas de todos os usuários da rede podem agir simultaneamente como cliente e servidor.

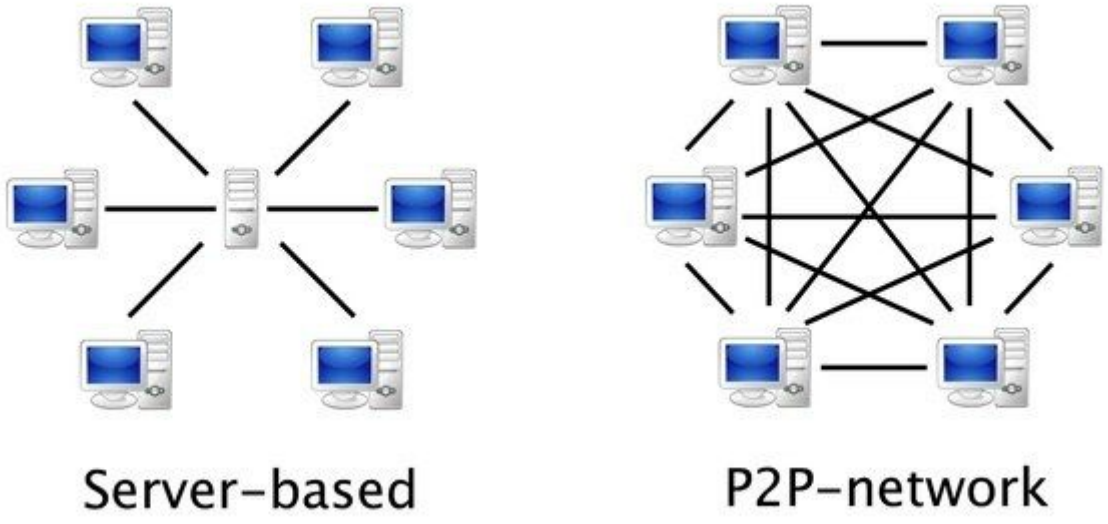


*Figura 1: Cliente-Servidor X Rede P2P*

**Execução do programa**

Para executar o servent deve-se navegar até a pasta principal do projeto e executar o seguinte comando:

*pyhton servent.py [porta] [arquivo_entrada] --other_peers [ip1:porta1] [ip2:porta2] ... [ipN:portaN]*

Para executar o cliente deve-se navegar até a pasta principal do projeto e executar o seguinte comando:

*python client.py [ip_servent:porta_servent]*

**Arquitetura**

**Servent**

O programa servent é o responsável por ler o arquivo chave-valor e criar um dicionário onde os pares chave-valor serão armazenados. No arquivo servent.py está localizada a estrutura principal do programa, onde é aberto um socket UDP, no porto indicado como parâmetro na execução, e são recebidas mensagens de outros servents ou de algum cliente.

No arquivo serventutils.py estão as funções que são utilizadas exclusivamente pelo servent:

- read_input_file
  - gera um dicionário de pares chave-valor recebendo como parâmetro um arquivo de entrada.
- local_db_search
  - verifica se a chave passada como parâmetro já está armazenada na lista de serviços previamente montada.
- forward_query
  - pega uma consulta feita ao servent em questão e a repassa para os outros peers.

**Cliente**

O programa cliente se comunica com um servent, da rede sobreposta ,que será seu ponto de contato com o sistema distribuído. No arquivo client.py está localizada a estrutura principal do programa, onde o programa recebe uma chave digitada pelo usuário, monta uma mensagem de consulta e a envia para o ponto de contato.

No arquivo clientutils.py estão as funções que são utilizadas exclusivamente pelo cliente:

- get_responses
  - aguarda a rede retornar uma resposta para a consulta feita.
- p2p_ask_kv
  - faz uma consulta na rede sobre uma chave específica e chama a função get_responses para aguardar a resposta.

No arquivo utils.py estão algumas funções, além de constantes, que são utilizadas em ambos os programas:

- ip_to_int
  - converte um endereço IP para o tipo long.
- int_to_ip
  - converte um tipo long para um endereço de IP.

**Testes**

Vários testes foram realizados a fim de verificar o comportamento dos programas. Testes com um número variado de nós na rede P2P foram realizados para verificar se o funcionamento estava de acordo com a especificação proposta, considerando principalmente os tempos de espera especificados. Ao final do trabalho nenhum erro foi encontrado e os programas atendem a todos os requisitos levantados.

**Desafios, dificuldades e imprevistos**

Os imprevistos foram menores em relação aos imprevistos dos trabalhos práticos anteriores, visto que os alunos já estavam acostumados com o funcionamento das bibliotecas de sockets, apenas se adaptando e abstraindo os conceitos do protocolo UDP.

Os desafios e dificuldades encontradas pelos alunos durante o desenvolvimento do trabalho estão relacionadas com a detecção de erros, pois por várias vezes foi necessário criar situações hipotéticas para descobrir onde um erro poderia ocorrer e assim tratá-lo.

O problema mais significativo encontrado e que ocasionou o maior volume de alterações no código foi relacionado ao programa servent. No momento em que era executada a função bind, o endereço estava sendo setado para ‘0.0.0.0’, ou seja, qualquer interface de rede da máquina. O vizinho de um determinado servent identifica o servent de origem por um IP específico. Com isso, no momento em que é realizada a verificação do alagamento confiável, onde o servent passa a mensagem para os seus vizinhos, o receptor não sabe que o servent que lhe enviou a solicitação já havia respondido a essa consulta, pois o IP do servent de origem é ‘0.0.0.0’ na visão do servent de origem e um IP qualquer na visão do receptor. Para solucionar este problema, o host do servent foi obtido por meio da função `gethostbyname(socket.gethostname())`.

**Conclusão**

Neste trabalho, foram desenvolvidos programas para um sistema sistema de armazenamento chave-valor do tipo peer-to-peer utilizando comunicação via protocolo UDP e funcionalidades equivalentes às da biblioteca de sockets POSIX. Foram efetuados testes suficientes para comprovar a funcionalidade dos programas. O livro adotado junto a disciplina e a documentação da biblioteca socket de Python foram utilizados como referência.