Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação
DCC023 - Redes de Computadores 2017/1
Professor: Dorgival Olavo Guedes Neto

TRABALHO PRÁTICO 3
Um Sistema Peer-to-peer de Armazenamento Chave-valor

Aluno: Yuri Diego Santos Niitsuma - 2011039023
Aluno: Alison de Oliveira Souza - 2012049316

Introdução

Neste trabalho, implementamos um sistema Peer-to-peer de armazenamento chave-valor. Este sistema tem o objetivo de permitir a troca de mensagens entre clientes e servents (hosts que se comportam como servidores apenas para responder solicitações do cliente sobre a base de dados chave-valor e controlar a troca de mensagens com os clientes).

O sistema deverá implementar um protocolo em nível de aplicação, utilizando interface de *sockets* UDP. Segundo a especificação, dois programas devem ser desenvolvidos:

1. *Um programa do sistema peer-to-peer, às vezes denominado servent (de server/client), que será responsável pelo armazenamento da base de dados chave-valor e pelo controle da troca de mensagens com seus pares.*
2. *Um programa cliente, que receberá do usuário chaves que devem ser consultadas e exibirá os resultados recebidos.*

O sistema deverá trabalhar através de portas, por onde será realizada a transmissão das mensagens. Além disso, deverá utilizar um arquivo contendo um conjunto de chaves associadas com valores (valores chaves), que serão utilizados para definir as chaves que serão consultadas pelos clientes no sistema. Devemos também implementar um algoritmo de alagamento confiável, para evitar que as mensagens sejam processadas repetidas vezes por um mesmo servent.

Arquitetura

Nosso trabalho foi implementado em python 3, e dividimos o código em 3 arquivos que serão descritos abaixo:

- **utils.py**:

Arquivo que serve para definir as bibliotecas importadas e utilizadas em nosso código, além de definir as constantes e funções auxiliares necessárias ao nosso sistema.

    - A variável *DEBUG*, se estiver setada em *True*, imprime todos os eventos possíveis em amarelo e branco.

- **clientTP3.py**:

Arquivo que define o comportamento do cliente, que fará consultas das chaves. Deverá se comunicar com o servent, fazendo as consultas de chaves através das mensagens de tipo CLIREQ e aguardando que algum servent responda com a mensagem do tipo RESPONSE em menos de 4 segundos, podendo receber mais de uma mensagem. Caso não chegue nenhuma mensagem após 4 segundos do envio da mensagem CLIREQ ou do recebimento da última RESPONSE recebida, o cliente deverá encerrar a conexão. O tratamento das mensagens recebidas é realizado com a função select, para poder tratar mais de uma mensagem recebida.

- **serventTP3.py**:

Arquivo que define o comportamento do servent. Ele deverá receber as solicitações de consulta das chaves pelos clientes (CLIREQ) e criar uma mensagem do tipo QUERY e repassar para todos os vizinhos, que são informados ao iniciar o servent. Após repassar a mensagem de QUERY a todos os vizinhos, o servent deverá procurar no seu dicionário local se existe a chave pedida pelo cliente, e caso exista, ele deverá montar uma mensagem do tipo RESPONSE e enviar ao cliente que fez a requisição.

O servent também poderá receber mensagens de outros servents, nesse caso serão as mensagens do tipo QUERY. Quando isso acontecer, o servent deverá primeiramente verificar se a mensagem que ele acabou de receber já foi recebida em algum outro momento, para evitar que mensagens entrem em loop pelos servents do sistema. Isso é feito inserindo as mensagens recebidas em um conjunto, e quando uma nova mensagem é recebida, verificamos se ela já existe no conjunto criado. Caso ela já exista, então é uma mensagem duplicada e o servent não deverá realizar nenhuma ação sobre ela. Caso ela não exista no conjunto, então inserimos a mensagem no conjunto, decrementamos seu TTL, repassamos a mensagem a todos os vizinhos do servent atual, e pesquisamos no dicionário local se existe a chave solicitada pelo cliente. Caso exista, devemos montar uma mensagem RESPONSE e enviar ao cliente que fez a solicitação.

Execução

Para executar o sistema, deve-se executar os seguintes comandos no terminal:

1. `python3 serventTP3.py <LOCALPORT> <KEY_VALUES> <IP1:PORT1> ... <IPN:PORTN>`
2. `python3 clientTP3.py <IP:PORT>`

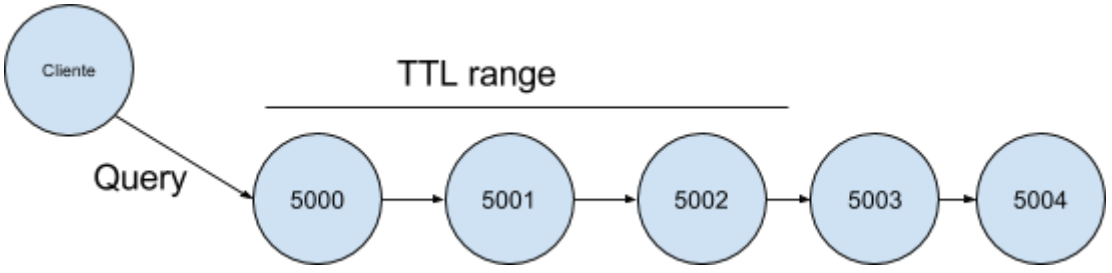
- O servent deverá receber como argumentos uma porta por onde “escutará” a mensagens direcionadas a ele (LOCALPORT), o nome de um arquivo que contém uma lista de chaves-valores que comporão nossa base de dados (KEY_VALUES) e um número variável de conjuntos de endereços IP e número de portas, que formarão o conjunto de servents vizinhos.
- O client deverá ser executado passando como parâmetro um endereço IP e o número da porta de um servent, que será seu ponto de conexão com a rede. Dessa forma, todas as mensagens enviadas pelo client para a rede deverão passar primeiramente pelo servent apontado pelo IP e porta passados como argumentos.

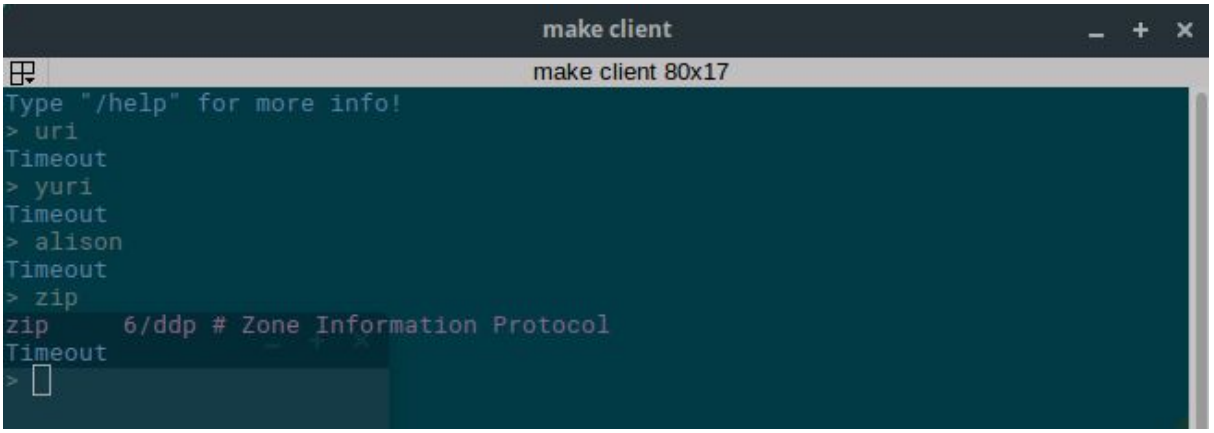
Testes

TTL

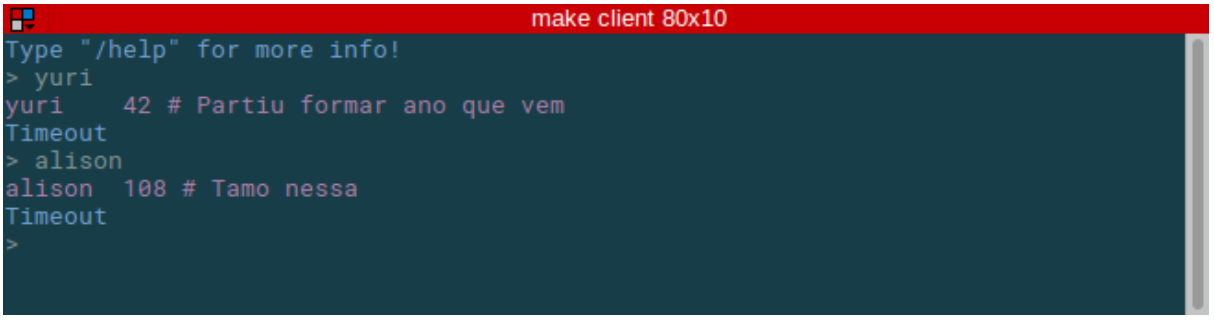
Teste da eficácia do tratamento da TTL Como é definido inicialmente como 3, não é possível acessar nenhum valor da chave key4.txt no exemplo abaixo.

```
python3 serventTP3.py 5000 key.txt 127.0.0.1:5001 &
python3 serventTP3.py 5001 key2.txt 127.0.0.1:5002 &
python3 serventTP3.py 5002 key.txt 127.0.0.1:5003 &
python3 serventTP3.py 5003 key.txt 127.0.0.1:5004 &
python3 serventTP3.py 5004 key4.txt
```



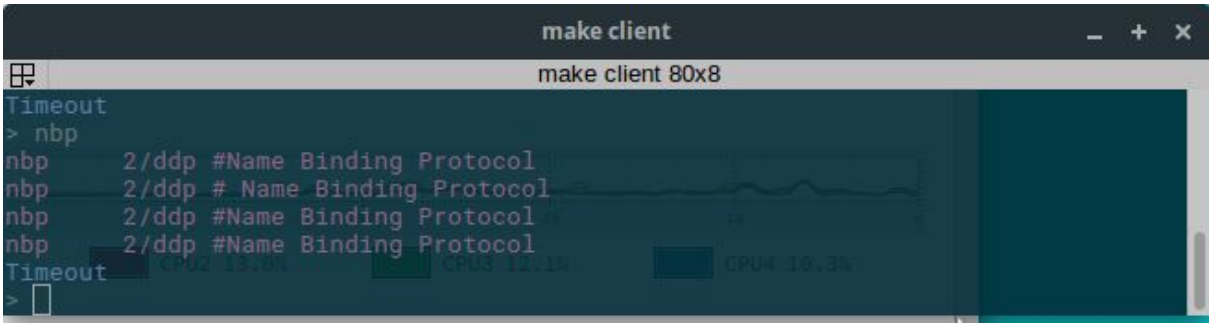


Mas se adicionarmos key4.txt na terceira linha temos acesso às chaves solicitadas:



Múltiplos retornos em consultas

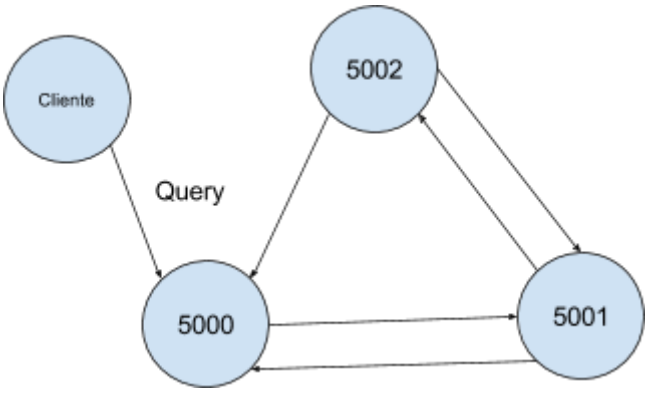
Caso seja solicitada uma chave existente em vários servents, ele recebe todas as respostas dentro do tempo limite de 4 segundos imposto pelo timeout.



Ciclos

Se o grafo, que representa as conexões entre os servents quando for utilizado o alagamento das QUERYs, tiver um ciclo, temos o problema de que a mensagem em broadcast seja repetida ao mesmo vértice. Armazenando um conjunto contendo os parâmetros (IP, PORT, SEQ_NUM, KEY) garantimos unicidade na QUERY e quando a duplicidade é detectada, ela será ignorada.

```
python3 serventTP3.py 5000 key.txt 127.0.0.1:5001  &
python3 serventTP3.py 5001 key2.txt 127.0.0.1:5000 127.0.0.1:5002 &
python3 serventTP3.py 5002 key.txt 127.0.0.1:5000 127.0.0.1:5001
```




```
make client 80x20
Type "/help" for more info!
> nbp
nbp     2/ddp #Name Binding Protocol
nbp     2/ddp # Name Binding Protocol
nbp     2/ddp #Name Binding Protocol
Timeout
> zip
zip     6/ddp # Zone Information Protocol
Timeout
> nbp
nbp     2/ddp #Name Binding Protocol
nbp     2/ddp # Name Binding Protocol
nbp     2/ddp #Name Binding Protocol
Timeout
>
```

nbp é uma chave existente nas 3 instâncias do Servent. Logo, depois é testado se o sequence number não atrapalha a mesma consulta (caso a nova consulta seja considerado uma repetida).

Considerações Finais

Vemos que este trabalho realmente se tornou mais simples devido aos conhecimentos sobre sockets introduzidos nos trabalhos anteriores, principalmente o conhecimento da função SELECT, que simplificou bastante o tratamento de várias mensagens recebidas pelo cliente. Assim, a grande dificuldade neste trabalho foi identificar corretamente como deveria ser realizado a troca de informações entre client e servent. Ter a liberdade de definir como seria a comunicação entre os dois programas nos deixou um pouco em dúvida sobre o que fazer. Outra dúvida que nos foi recorrente era sobre o algoritmo de alagamento confiável para garantir que mensagens não fossem tratadas mais de uma vez pelo servent, em caso de loop na rede por exemplo. Como não havíamos visto especificamente um algoritmo para isso em aula, ficamos na dúvida de como implementá-lo de forma eficiente e rápido. No final acabamos optando por armazenar as mensagens recebidas em um conjunto, e verificar a cada mensagem recebida pelo servent se ela existe ou não no conjunto.