DOCUMENTAÇÃO TP3 - REDES DE COMPUTADORES

Bárbara D'Avila Duarte – 2013062707
Guilherme Viegas de Faria – 2013000760

Considerações prévias

O trabalho teve seus testes finais realizado em um computador pessoal com as seguintes configurações:

Processador: Intel®Core™i5-5257U CPU @ 2.70GHz 3.1GHz
Memória Instalada (RAM): 8 GB 1867 MHz DDR3
Tipo de Sistema: Sistema Operacional de 64 bits, processador com base em x64
Sistema Operacional: Apple macOS Sierra 10.12.4

Foi utilizado para a solução do problema a linguagem Python na versão 2.7.6.

Introdução

Este trabalho consiste em realizar os conceitos da arquitetura peer-to-peer e os aspectos de implementação de um algoritmo distribuído sobre um ambiente composto por um número desconhecido de participantes, além de uma solução para roteamento em uma topologia não conhecida a-priori, para os alunos da disciplina de Redes de Computadores, a fim de implementar um sistema de chave-valor entre pares, sem servidor.

Sendo mais específicos, a ideia principal é fazer com que várias entidades se comuniquem entre si e troquem informações da seguinte maneira: Um CLIENT manda uma requisição com uma chave para um SERVENT, este SERVENT envia a resposta caso possua a chave e repassa a mensagem para todos os SERVENTs vizinhos. Estes SERVENTs vizinhos fazem a mesma coisa que o primeiro SERVENT até que um campo do cabeçalho da mensagem, chamado TTL, chegue ao valor um (1). O TTL é decrementado a cada SERVENT que ele passa.

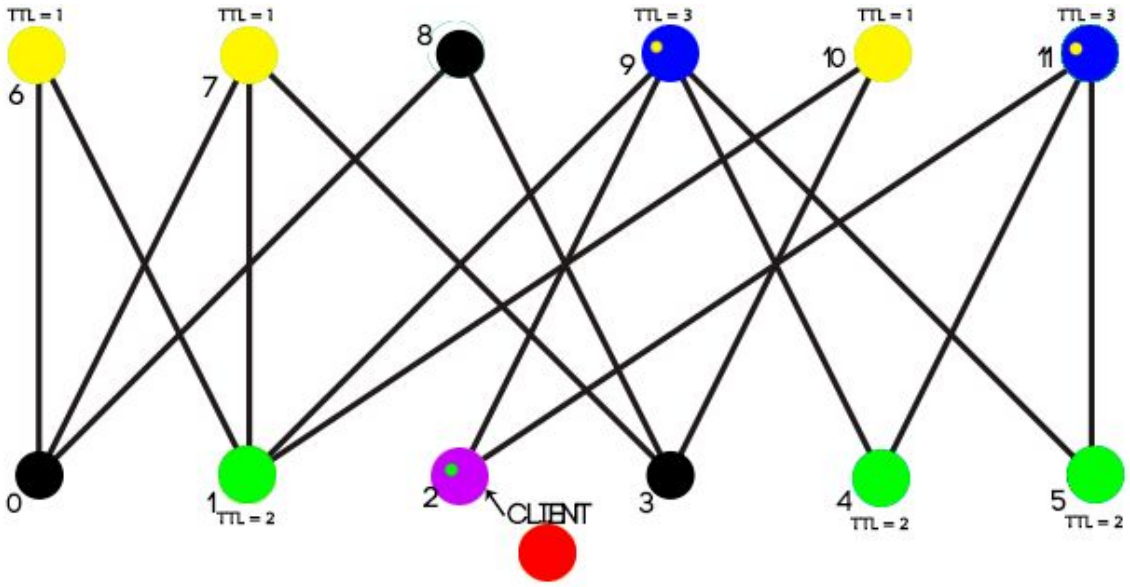
Ao final espera-se que as entidades CLIENT e SERVENT se comportem conforme o protocolo pré-estabelecido na divulgação deste trabalho prático.

Arquitetura

A modelagem de solução do problema se baseia no conceito de programação em sockets com protocolo UDP, onde não há conexão entre as duas entidades. Além das funções de sockets para realizar as múltiplas conexões com os seus pares para o caso do SERVENT.

Para este problema foi proposto a criação de dois programas: Um SERVENT, responsável por receber a requisição de um CLIENT respondê-la se for sua responsabilidade, caso possua a chave enviada pelo cliente, repassar a mensagem para os SERVENTs vizinhos e decrementar o TTL, que deve sempre ser iniciado em três, como definido na especificação deste trabalho prático. O segundo programa é o programa CLIENT, que é responsável por enviar uma requisição para um SERVENT específico e reenviá-la caso não receba uma resposta em 4 segundos.

Para exemplificar, suponhamos que a topologia de "vizinhos" de SERVENTs seja a desenhada abaixo:



Passo 1: Suponhamos que o nó vermelho é o programa CLIENT que está fazendo a requisição para o SERVENT 2.

Passo 2: O SERVENT 2 recebe a requisição do CLIENT, monta o TTL com valor 3 e repassa a requisição para os seus SERVENTs vizinhos, sinalizados de azul, SERVENT 9 e SERVENT 11.

Passo 3: O SERVENTs 9 e 11, recebem a requisição do SERVENT 2, decrementam o TTL para 2 e repassam a requisição. O SERVENT 9 repassa a requisição para os SERVENTs 1, 2, 4 e 5. O SERVENT 1 repassa a requisição para os SERVENTs 2, 4 e 5.

Passo 4: Suponhamos que no passo anterior o SERVENT 9 tenha repassado as requisições antes do SERVENT 11.

Portanto os SERVENTs 1, 4 e 5, sinalizados de verde, irão receber as requisições do SERVENT 9, decrementam o TTL para 1 e repassam a requisição. O SERVENT 1 repassa para os SERVENTs 6, 7, 9 e 10. O SERVENT 4 repassa para os SERVENTs 9 e 11. E o SERVENT 5 repassa as requisições também para os SERVENTs 9 e 11.

Ao receber a requisição do CLIENT vermelho pela segunda vez, o SERVENT 2 valida que já recebeu a requisição daquele CLIENT uma primeira vez e não faz nada.

O mesmo acontece ao SERVENTs 2, 4 e 5 ao receber a requisição novamente, desta vez partindo do SERVENT 11.

Passo 5: Suponhamos que no passo anterior o SERVENT 1 tenha repassado as requisições antes do SERVENT 4 e do SERVENT 5.

Portanto os SERVENTs 6, 7 e 10, sinalizados de amarelo, irão receber as requisições do SERVENT 1, decrementam o TTL para 0 e NÃO repassam a requisição.

Ao receber a requisição do CLIENT vermelho pela segunda vez, o SERVENT 9 valida que já recebeu a requisição daquele CLIENT uma primeira vez e não faz nada.

Após o repasse da requisição do SERVENT 1 o SERVENT 4 repassa as requisições, ainda antes do SERVENT 5.

Ao receber a requisição do CLIENT vermelho novamente os SERVENTs 9 e 11 validam que já receberam a requisição daquele CLIENT uma primeira vez e não faz nada.

O mesmo acontece novamente os SERVENTs 9 e 11 ao receber a requisição mais uma vez, desta vez partindo do SERVENT 5.

Passo 6: Como nenhum SERVENT vai repassar mais a mensagem, as requisições terminam sua trajetória.

Solução Proposta

A proposta de solução deste problema é separada em dois arquivos/programas principais: “servent.py” e “client.py”. Para este documento veremos portanto com mais detalhes a solução utilizada para estes programas conforme descrito na modelagem. A solução portanto, pode ser separada nos tópicos relacionados e detalhados a seguir:

Servent

1. *Abertura Passiva:*

O SERVENT deve ser o primeiro programa a ser executado. Ao iniciar, o mesmo cria um dicionário para armazenar as variáveis de chave e valor que recebe do arquivo de entrada e uma lista com os endereços dos SERVENTs vizinhos recebidos como parâmetro na chamada do programa.

2. *Recebendo uma mensagem de um CLIENT:*

Sempre que uma nova transmissão começa com um SERVENT ele verifica se a transmissão vem de um CLIENT ou de um outro SERVENT.

Para o primeiro caso, que é verificado quando o campo “tipo de mensagem” é “CLIREQ”, o SERVENT adiciona a requisição à lista de requisições já feitas e monta o cabeçalho da mensagem conforme o enquadramento, que será melhor descrito nos próximos tópicos deste documento, atualizando seus campos da seguinte maneira: altera o tipo da mensagem para “QUERY”, informa que o valor do TTL é três (3) e atribui um número de sequência para a mensagem.

Este cabeçalho mais a chave de pesquisa do CLIENT são repassadas à todos os vizinhos.

Após este repasse o SERVENT verifica no seu dicionário se ele possui a chave pesquisada pelo CLIENT, caso ele a possua o SERVENT envia uma mensagem para o CLIENT com o valor referente àquela chave pesquisada.

3. *Recebendo uma mensagem de um SERVENT:*

Sempre que uma nova transmissão começa com um SERVENT ele verifica se a transmissão vem de um CLIENT ou de um outro SERVENT.

Para o segundo caso que é verificado quando o campo “tipo de mensagem” é “QUERY”, o SERVENT desmonta o cabeçalho conforme o enquadramento, que será melhor descrito nos próximos tópicos deste documento, verifica se a mensagem já foi recebida previamente, se sim: não faz nada, se não: adiciona a requisição à lista de requisições já feitas, verifica que o valor do TTL é positivo, decrementa o TTL refaz o cabeçalho.

Este cabeçalho mais a chave de pesquisa do CLIENT são repassadas à todos os vizinhos.

Após este repasse o SERVENT verifica no seu dicionário se ele possui a chave pesquisada pelo CLIENT, caso ele a possua o SERVENT envia uma mensagem para o CLIENT com o valor referente àquela chave pesquisada.

4. *Enquadramento:*

O enquadramento do SERVENT foi feito da seguinte forma:

- O primeiro campo do cabeçalho da mensagem corresponde ao tipo da mensagem que foi transformado em um inteiro hexadecimal e enviado pela rede como um *unsigned short* (>H). (2 bytes, 16 bits)
- O segundo campo do cabeçalho da mensagem corresponde ao valor do TTL que é iniciado em três e decrementado a cada nova recepção até chegar a zero. Este campo foi transformado em um inteiro hexadecimal e enviado pela rede como um *unsigned short* (>H). (2 bytes, 16 bits)
- O terceiro campo do cabeçalho da mensagem corresponde ao valor do IP do CLIENT que enviou a requisição. Este campo foi transformado em uma string e enviado pela rede como um *four-letter string* (>4s). (4 bytes, 32 bits)
- O quarto campo do cabeçalho da mensagem corresponde ao valor do PORTO do CLIENT que enviou a requisição. Este campo foi transformado em um inteiro hexadecimal e enviado pela rede como um *unsigned short* (>H). (2 bytes, 16 bits)
- O quinto campo do cabeçalho da mensagem corresponde ao identificador de origem ou seja, o número (identificador único) da origem da mensagem, neste caso mensagem pode conter ou não dados. Este campo foi transformado em um long-int hexadecimal e enviado pela rede como um *long integer* (>L). (4 bytes, 32 bits)

Client

1. *Início da conexão:*

Ao iniciar o programa CLIENT se conecta a um programa SERVENT via conexão UDP. Este programa SERVENT é o que será responsável pela disseminação da requisição deste programa CLIENT.

2. *Comunicação com o usuário:*

Após a conexão é papel do CLIENT fazer a interface de entrada com o usuário. Ou seja, é através dos programas CLIENT que o usuário poderá enviar as mensagens com o valor da chave que eles estão buscando. Para isso a dupla responsável por esta solução optou por uma comunicação simplificada, com apresentação das opções através de mensagens no TERMINAL através da função "print()" e leitura do teclado através da função raw_input().

A mensagem "*Enter the key you want to search for:*" é apresentada ao usuário a fim de que ele informe a chave de pesquisa que será disseminada entre os SERVENTs esperando por uma resposta com o valor da chave buscada.

3. *Envio da mensagem:*

Após a digitação da mensagem, com a chave de pesquisa, o programa monta o cabeçalho simplificado com as informações inerentes ao protocolo que será incrementado no programa SERVENT associado àquele CLIENT. Os campos enviados inicialmente são o tipo da mensagem e a chave de pesquisa.

Além disso no função *sendto()* do protocolo UDP é informado o endereço de conexão do SERVENT associado ao CLIENT que está enviando a mensagem.

4. *Reenvio da mensagem:*

Ao enviar uma mensagem desse tipo, o CLIENT aciona um *timer* na recepção de uma mensagem, sempre que o programa fica mais de 4 segundos sem receber uma resposta, ele reenvia a última mensagem a fim de restabelecer a comunicação com seu par e garantir o recebimento de uma resposta.

5. *Enquadramento:*

O enquadramento do CLIENT foi feito de maneira mais simples, possuindo apenas um campo no cabeçalho e a mensagem propriamente dita depois. O Campo único do cabeçalho está descrito à seguir:

- O único campo do cabeçalho da mensagem do CLIENT corresponde ao tipo da mensagem que foi transformado em um inteiro hexadecimal e enviado pela rede como um *unsigned short* (>H). (2 bytes, 16 bits)

Discussões

1. *Comunicação:*

A comunicação das entidades é estruturada para que o programa CLIENT use o programa SERVENT como o distribuidor da comunicação. Além disso, para que a comunicação ocorra, de maneira parametrizável, foi definido que o programa cliente deverá escutar no endereço IP fornecido na passagem de parâmetro na chamada do programa e no porto, também passado por parâmetro. Já o SERVENT recebe como parâmetro o número do PORTO, um arquivo com as informações de chave e valor e uma lista de IPs e PORTOS que são seus vizinhos.

2. *Execução do código:*

Deve-se chamar, por linha de comando, o programa SERVENT como descrito abaixo:

Python servent.py <localport> <key-values> <ip1:port1> ... <ipN:portN>

Onde:

<localport> : É o porto local onde o SERVENT deve ser executado.

<key-values> : É o nome do arquivo de texto que contém as informações de chave e valor a serem pesquisadas.

<ip1:port1> ... <ipN:portN>: É a lista de endereços dos SERVENTs vizinhos ao SERVENT que está sendo inicializado.

Deve-se chamar, por linha de comando, o programa CLIENT como descrito abaixo:

python client.py <IP:port>

Onde:

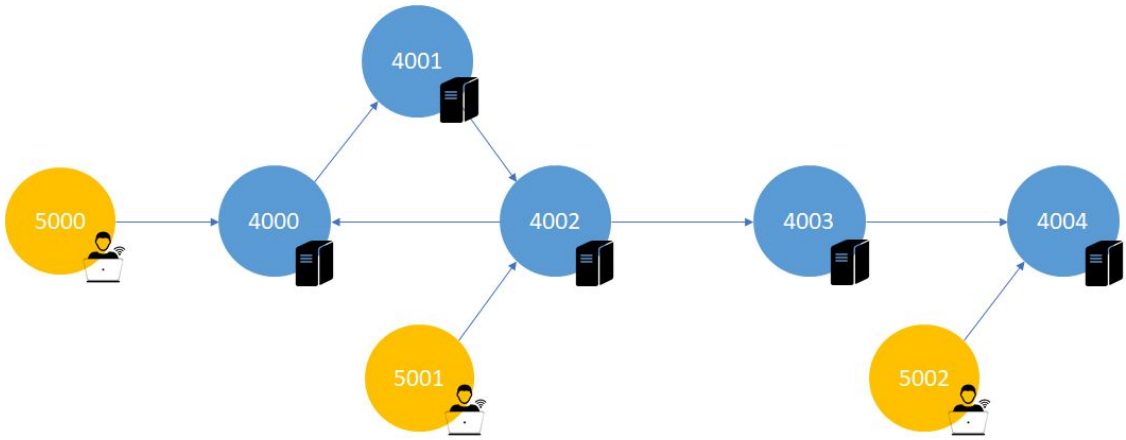
<IP:port> : É o endereço do SERVENT que o CLIENT a ser executado irá fazer sua requisição.

3. Testes realizados:

Usamos como entrada o arquivo de testes fornecido na especificação do trabalho prático. O conteúdo do arquivo de entrada é mostrado a seguir.

```
# WELL KNOWN PORT NUMBERS
#
rtmp              1/ddp    #Routing Table Maintenance Protocol
tcpmux            1/udp    # TCP Port Service Multiplexer
tcpmux            1/tcp    # TCP Port Service Multiplexer
#                          Mark Lottor <MKL@nisc.sri.com>
nbp               2/ddp    #Name Binding Protocol
compressnet       2/udp     # Management Utility
compressnet       2/tcp     # Management Utility
```

Primeiramente foi testada a seguinte topologia. Para estes testes, assumimos que todos os servents têm todas as mesmas chaves e valores, a fim de analisar quais servents são atingidos por cada requisição.



Simulamos o envio de uma requisição de um programa cliente conectado à porta 5000 para um servent conectado à porta 4000. A requisição CLIREQ enviada ao servidor 4000 é propagada aos outros servents da seguinte forma: o servent 4000 inicializa o temporizador com valor 3 (TTL = 3), e envia a requisição aos servents vizinhos (4001); o servent 4001 processo a requisição recebida, decrementando o TTL, e encaminhando ao servent 4002; o procedimento se repete até que o TTL seja 0, e nosso experimentos confirmam que os servents 4000, 4001, 4002 e 4003 recebem e tratam corretamente a requisição. Vale notar, com relação ao servent 4000, que, apesar de receber a requisição anteriormente propagada ao servent 4002, tal solicitação não é processada novamente, tampouco é propagada. As mensagens recebidas pelo cliente 5000 são apresentadas abaixo.

MacBook-Pro-de-Guilherme:final viegas$ python client.py 127.0.0.1:4000
Enter the key you want to search for: rtmp
(127.0.0.1:4000) - KEY: rtmp VALUE: 1/ddp #Routing Table Maintenance Protocol
(127.0.0.1:4001) - KEY: rtmp VALUE: 1/ddp #Routing Table Maintenance Protocol

(127.0.0.1:4002) - KEY: rtmp VALUE: 1/ddp #Routing Table Maintenance Protocol
(127.0.0.1:4003) - KEY: rtmp VALUE: 1/ddp #Routing Table Maintenance Protocol

Executamos procedimento análogo ao anterior com o cliente conectado a porta 5001. Desta vez todos os servents responderam. A mensagem foi propagada da seguinte forma: o servent 4002 é o primeiro servent a receber a requisição, posteriormente a mensagem é propagada para os servents 4000 e 4003, e, por fim, para os servents 4001 e 4004. Não necessariamente as mensagens deveriam chegar na ordem que foram propagadas, uma vez que o protocolo utilizado (UDP) não garante a ordem das mensagens -- porém, por coincidência, as mensagens chegaram em ordem.

MacBook-Pro-de-Guilherme:final viegas$ python client.py 127.0.0.1:4002
Enter the key you want to search for: rtmp
(127.0.0.1:4002) - KEY: rtmp VALUE: 1/ddp #Routing Table Maintenance Protocol
(127.0.0.1:4000) - KEY: rtmp VALUE: 1/ddp #Routing Table Maintenance Protocol
(127.0.0.1:4003) - KEY: rtmp VALUE: 1/ddp #Routing Table Maintenance Protocol
(127.0.0.1:4004) - KEY: rtmp VALUE: 1/ddp #Routing Table Maintenance Protocol
(127.0.0.1:4001) - KEY: rtmp VALUE: 1/ddp #Routing Table Maintenance Protocol

Por fim, testamos o envio de uma CLIREQ a partir do cliente conectado à porta 5002. Tendo em vista a topologia de rede construída, a requisição é tratada e respondida apenas pelo servidor 4004.

MacBook-Pro-de-Guilherme:final viegas$ python client.py 127.0.0.1:4004
Enter the key you want to search for: rtmp
(127.0.0.1:4004) - KEY: rtmp VALUE: 1/ddp #Routing Table Maintenance Protocol

Conclusão

O objetivo era a implementação, utilizando linguagem Python, de arquitetura peer-to-peer e os aspectos de implementação de um algoritmo distribuído sobre um ambiente composto por um número desconhecido de participantes, além de uma solução para roteamento em uma topologia não conhecida a-priori.

O objetivo foi alcançado após algumas tentativas.

Apesar das poucas dificuldades encontradas, todos os problemas foram corrigidos e o programa está executando conforme esperado.