Trabalho Final de Redes

Leonardo G. Carvalho, Matheus S. Redecker

[1]Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS

1. Detalhes da implementação

1.1. Linguagem

Utilizamos a linguagem JAVA para a implementação do trabalho.

1.2. Classes

Temos 6 classes, que estarão descritas abaixo com seus principais métodos e atributos:

1.2.1. ipAddress

Esta classe representa um endereço IPv4. Optamos por utilizar um *array* de 32 inteiros (0s e 1s) para ser o IP, pois facilita na hora de fazer operações com a máscara e nos oferece um maior controle sobre os bits individuais. Além disso, colocamos um número inteiro que significa a máscara.

Nesta classe temos um construtor que recebe as quatro partes do endereço IP em decimal separadas e converte em um IP na forma binaria para que seja armazenado. Os outros construtores foram feitos para facilitar copias e conversões. Os métodos desta classe servem para manipular a estrutura do IP e converter eles em um IP na forma decimal separado em quatro partes (*binaryToAddress*), converter um número decimal para binário (*toBinaryArray*), e converter o IP em String (*binaryToString*).

1.2.2. Network

Esta classe tem como objetivo representar uma rede, onde cada rede tem uma subnet, um número de hosts e um nome. Ela apenas tem um construtor com o nome e o numero de hosts, já que a sua subnet será definida futuramente durante a divisão dos endereços. Achamos conveniente implementamos um método de compareTo, que compara o número de hosts entre duas redes e informa qual das duas é maior ou menor. Fizemos isso pois nosso algoritmo requer que as redes estejam em ordem decrescente de número de hosts para que os endereços sejam distribuídos corretamente.

1.2.3. Router

Esta classe tem como objetivo representar um roteador. Ela tem um nome, um lista de redes, um lista de interfaces (portas) ipAddress, uma RouterTable e uma lista de roteadores vizinhos (que será preenchida na hora de completar a RouterTable). Inicializamos um Router informando um nome e o numero de portas, e então inicializamos as listas de redes e de interfaces(portas) com o tamanho igual a este número.

Temos um método para adicionar uma Network (*addConnectedNetwork*), um método que pede para as Networks ligadas a ele para que informem um IP válido para suas interfaces (*setUpInterfaceIPs*), um método para sabermos o número da porta do roteador que liga com alguma rede que se conecte diretamente com o roteador informado (*getPort NumberLeadingToRouter*) e um método que retorna o endereço da porta que se conecta a uma rede em que está ligado o roteador informado (*getAddressOfPortLeadingToRouter*).

1.2.4. Subnet

Esta classe tem como objetivo representar uma Subnet. Ela tem três ipAddress: Um sendo o ID da rede, outro sendo o primeiro endereço de host, e o ultimo endereço de host.

Seus métodos consistem em calcular o inicio e o final dos hosts (*calculateFirst AndLastAddresses*), e também informar a algum roteador que peça um IP válido da rede (*getValidAddress*). Este último impede que, por algum motivo, 2 roteadores que se ligam na mesma rede acabem ficando com o mesmo IP em suas portas.

1.2.5. RouterTable

Esta classe tem como objetivo representar uma tabela de um Roteador. Nela, teremos o nome do roteador, uma lista com as redes destinos, uma lista de IPs e uma lista para as portas, temos um metodo para adicionar uma linha, que é passado a informção para a tabela, como a Network, o ipAddress, e a porta com que se conecta.

1.2.6. Topologia

Esta classe é onde ligamos todas as outras classes citadas acima. Nela, temos uma lista de subnets, uma lista de redes e uma lista de routers. Para que o programa funcione, recebemos um arquivo contendo a topologia desejada e um IPv4 com a sua mascara em formato de barra e com isso podemos iniciar o processo de construção da topologia.

Primeiro, salvamos o IP recebido e lemos o arquivo de texto através do método *loadFileInfo* que salva as redes e routers informados e suas devidas ligações. Após analisarmos como o método de divisão de subredes VLSM (*Variable Length Subnet Mask*) funciona, chegamos na seguinte implementação:

Criamos uma ”subnet” que é composta pelo IP e mascara recebidos, e colocamos ela no *array* de subnets. Este *array* serve como um repósitorio, onde colocaremos ”pedaços” de subredes a medida que elas forem sendo divididas e pedaços sobrem.

O método que atribui os endereços para as redes é um loop que itera pelas Networks (que foram ordenadas em ordem decrescente de número de hosts utilizando *Collections.sort()*). Para cada uma delas, é feita uma consulta no ”repósitório” de redes buscando por uma rede que enderece exatamente o número de hosts buscado (na realidade não é exatamente o número de hosts pedido pelo usuário, e sim a potência de 2 mais próxima que seja maior do que o número de hosts + 2, para que englobe o endereço de broadcast e ID da rede). Caso não seja encontrada uma rede que tenha a capacidade exata (e sim

maior), esta subrede será quebrada em N partes, e uma deles será escolhida como a subrede correta. As outras redes resultantes da quebra são colocadas de volta no repositório para futuro uso/divisão.

Após todas as Networks terem um endereço e range calculados, os roteadores pedem a elas IPs válidos para serem atribuídos a suas portas. Neste momento, todos os IPs da topologia foram atribuídos, e a única informação existente na router table de um roteador são as redes conectadas diretamente a ele. Agora, a etapa de preencher a *router table* inicia. Ela é dividida em 2 partes:

1. Todos os roteadores devem descobrir que outros roteadores estão a **exatamente** 1 rede de distância. Estes serão seus "vizinhos".
2. Cada roteador pergunta a todos os seus vizinhos, um por vez, se sua router table contém alguma rede que ele ainda não conheça. Se ele achar alguma, adiciona na sua router table e preenche o campo de *next hop* e número da porta com as informações corretas. Este processo acontece **repetidamente**, varrendo todos os roteadores, até que nenhum deles encontre nenhuma rede nova em seus vizinhos.

2. Como utilizar a ferramenta

Para utilizarmos essa ferramenta é necessário ter o JAVA instalado no computador. Temos que criar um arquivo .txt contendo a topologia no seguinte formato:

```
#NETWORK
<net_name>, <num_nodes>
#ROUTER
<router_name>, <num_ports>, <net_name0>, ..., <net_nameN>
```

Para prosseguirmos com a utilização, temos que rodar o programa passando o arquivo da topologia e o IPv4 inicial junto com sua mascara em formato de barra:

```
$ java -jar topoconfig.jar <topologia> <endereço/prefix>
```

Assim, o programa gera automaticamente a saída contendo uma topologia válida para os parâmetros informados, junto com a tabela de roteamento dos roteadores.

3. Limitações da ferramenta implementada

Em todos os exemplos testados nosso programa, se passados os parâmetros corretos, calcula uma forma correta de dividir IPs em uma topologia.

4. Dificuldades de implementação

Encontramos dificuldades no inicio do projeto para decidir qual estrutura de dados iriamos usar para detalhar o IP. Depois de algumas tentativas decidimos usar um *array* de inteiros que armazenam 0s e 1s. Além disso, a outra dificuldade foi fazer as subnets otimizadas, onde não se tem desperdício de IPs, mas, depois de algumas tentativas e testes conseguimos apresentar uma solução que atende ao esperado.