UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

TRABALHO PRÁTICO 01

GEOVANNA MENEGASSE SILVA
MATHEUS ALEXANDRE IRIAS DE OLIVEIRA

INTRODUÇÃO

Neste trabalho, implementamos o DCCRIP, um protocolo simples de roteamento que segue o algoritmo de vetor de distâncias. O objetivo do programa é simular roteadores conectados por links "virtuais" que executam o algoritmo de vetor de distâncias para encontrar caminhos entre eles.

Dois processos foram desenvolvidos para que o sistema funcione corretamente. O primeiro deles é uma interface de controle que receberá comandos da entrada padrão e enviará mensagens para os roteadores indicados de acordo com o comando executado. O segundo processo é o roteador, que receberá mensagens tanto da interface de controle quanto de outros roteadores e executará o método correspondente à mensagem recebida. Ambos os processos utilizam UDP como protocolo de transporte e a biblioteca socket foi utilizada para definir este contrato.

IMPLEMENTAÇÃO

A interface de controle conta com uma configuração inicial de socket, com UDP definido como protocolo de transporte. Além disso, a interface mantém-se ativa através de um loop infinito para esperar por comandos da entrada padrão. Ao receber uma entrada, de acordo com o comando informado, o programa particiona a entrada e concatena suas partes em uma string que carrega a mensagem a ser enviada ao roteador de destino. O método *sendto* da biblioteca de sockets é evocado para enviar a string correspondente a mensagem ao endereço IP e porto indicados.

Um programa roteador mantém, em tempo de execução, um identificador para si mesmo e um porto para receber mensagens, ambas as informações são recebidas como argumentos no comando de execução do programa. Além disso, um roteador mantém sua lista de vizinhos diretos, sua tabela de roteamento e uma interface de socket exclusiva que utiliza o protocolo de transporte UDP. A variável *MY_IP* presente no programa roteador, representa o IP da máquina a qual o programa está em execução. Pode-se alterar esse valor caso queira especificar um endereço IP diferente.

Um roteador permanece ativo por meio de um loop infinito até que uma mensagem de encerramento de execução seja recebida. Enquanto está ativo, o programa mantém um timer em forma de thread com um temporizador de um segundo para o roteamento, ou seja, a cada segundo o programa roteador inicia a execução do algoritmo de roteamento enviando anúncios de rotas aos seus vizinhos (método *manda_anuncio*). Enquanto isso, uma outra thread está no processo, ou seja, a thread do programa principal permanece ativa aguardando por mensagens através do método *recvfrom* da biblioteca de sockets. Para garantir a consistência dos dados na tabela de roteamento e na lista de vizinhos, utilizamos semáforos como primitiva de sincronização para as duas threads.

Com base no comando recebido dentro de uma mensagem, o roteador executa a sua ação correspondente. Além das ações simples e triviais de encerramento da execução do roteador e da

impressão de sua tabela de roteamento, existem algumas tarefas mais complexas a serem discutidas a seguir.

Comecemos pela tarefa de se conectar a um vizinho (método *adiciona_vizinho*). O método correspondente a esta tarefa, aciona o semáforo de sincronização para adicionar a tripla *(nome_roteador, endereço_ip, porto)* na lista de vizinhos do roteador. Além disso, o método adiciona a tripla *(nome_roteador, distancia = 1, proximo_passo = nome_roteador)* na tabela de roteamento, simbolizando uma rota direta até o roteador informado, já que o mesmo se tornou um vizinho do roteador atual.

Para remoção de um roteador da lista de vizinhos de outro roteador (método *remove_vizinho*), acionamos o semáforo para garantir que somente esta thread esteja mexendo nas tabelas. Buscamos na tabela de vizinhos do roteador atual, o endereço IP e o porto do roteador informado para extrair o seu identificador e também para já removê-lo da lista de vizinhos. O nome do roteador foi extraído para que possamos encontrá-lo na tabela de roteamento do roteador atual. Ao encontrá-lo, mudamos o valor de sua distância até o nó atual para infinito, que no nosso caso é 16, já que não existe mais uma rota direta para ele saindo de onde está. Além disso, também colocamos distância infinita nas rotas que possuem o vizinho removido como próximo passo.

Na tarefa de enviar um anúncio para iniciar o roteamento (método *manda_anuncio*), primeiro o roteador começa montando a mensagem, contendo o nome do roteador que irá enviar o anúncio e a lista de tuplas *(nome_destino, distancia)* extraídas de sua tabela de roteamento. Em seguida, o roteador chama o *sendto* para todos os seus vizinhos encaminhando a mensagem construída contendo o anúncio de rotas.

Um roteador, ao receber uma mensagem contendo anúncios de rotas (método *recebe_anuncio*), se perceber que o emissor não está em sua lista de vizinhos, ele o insere na lista. Em seguida, percorre a mensagem extraindo as informações das rotas para mandar em formato de lista na chamada do algoritmo do vetor de distâncias (método *distance_vector*). Esse algoritmo segue o seguinte pseudocódigo:

```
Para cada anúncio vindo de um vizinho V:
    Para cada par (destino, distancia):
        Se destino não existe na tabela:
          insere linha (destino, distancia + 1, V)
        Senão:
          Se distancia + 1 < distancia_atual:
            substitui linha por (destino, distancia + 1, V)
          Senão:
            Se próx. passo já é V:
                atualiza distancia_atual para distancia + 1
```

O resultado após a execução do algoritmo é a tabela de roteamento devidamente atualizada contendo as rotas atuais para cada roteador na rede.

Agora, quando a tarefa de encaminhar uma mensagem é acionada (método *envia_mensagem*), a origem busca pelo destino em sua tabela de roteamento. Caso o nome do destino não esteja presente na tabela, imprime-se uma string iniciada por um X no roteador de origem, significando que a mensagem da origem para o destino não foi recebida. Se o roteador de destino for encontrado na tabela de roteamento e seu nome for igual ao nome do roteador atual, imprime-se no roteador

destino (que também é o atual) uma string iniciada por R dizendo que a mensagem foi recebida da origem até o destino. Se durante o envio da mensagem, o nome do próximo passo para chegar no destino, presente na tabela de encaminhamento, for diferente do nome do roteador atual, busca-se o IP e porto do próximo passo na tabela de vizinhos e a mensagem é encaminhada para ele por *sendto*, além de uma string iniciada por E ser impressa no roteador que irá repassar a mensagem, significando que a mensagem for encaminhada através dele.

MENSAGEM

O formato básico das mensagens enviadas da interface de controle para um roteador é:

COMANDO : IP : PORTO

onde comando é alguma das letras que representam as tarefas mostradas na próxima sessão e o IP e PORTO são os valores de endereço IP e porto informados na entrada. Essa mensagem pode ser estendida ou minimizada dependendo do comando escolhido, por exemplo: caso a tarefa escolhida seja a número 1 indicada na próxima sessão, a mensagem terá o formato

"`C:ip_2:porto_2:nome_roteador_2`",

enquanto que se a tarefa escolhida for a de número 3, a mensagem terá o formato "`I`". O que realmente importa aqui é que os campos de uma mensagem são separados por "`:`" para simplificar a decodificação das mensagens por parte do programa roteador.

Ainda cabe mencionar o formato das mensagens trocadas entre roteadores. Mensagens de anúncios se iniciam com o valor "`11111`" seguido do nome do roteador que está enviando a mensagem, do número de rotas no anúncio e da tabela de anúncios em forma de strings com "`nome_destino:distancia`". Todos os campos são separados pelo separador definido "`:`" e a mensagem resultante teria a seguinte aparência, supondo o roteador de origem sendo "*roteador01*":

"`11111:roteador01:2:roteador02:1:roteador02:3`"

comando : nome de origem : quantidade de rotas : rota1(nome destino, distância) : rota2(nome destino, distância)

EXECUÇÃO

Para a execução da interface de controle, basta usar o comando:

```
make run_cli
```

Já para executar um roteador, basta usar o comando:

```
make run_rt arg1=roteador01 arg2=1234
```

onde `arg1` recebe um nome para aquele roteador e `arg2` recebe o porto que será escolhido para o mesmo.

No nosso entendimento do enunciado, supomos que o nome do roteador recebido em `arg1`, seria apenas uma string que identifica o roteador naquela rede, que no nosso caso seria a máquina executando o programa roteador. Programas roteadores sendo executados em uma mesma máquina, estariam no mesmo IP, mas em portas diferentes e com nomes (identificadores) diferentes.

Ao iniciar a execução da interface de controle e de alguns programas roteadores, uma lista de comandos são possíveis de serem passados pela entrada padrão à interface de controle. São eles:

1. Adicionar roteador (`nome_roteador_2, ip_2, porto_2`) aos vizinhos do roteador identificado por `ip_1:porto_1`:

```
ip_1  porto_1  C  ip_2  porto_2  nome_roteador_2
```

2. Remover o roteador `ip_2:porto_2` dos vizinhos de `ip_1:porto_1`:

```
ip_1  porto_1  D  ip_2  porto_2
```

3. Iniciar o roteamento a partir de `ip_1:porto_1`:

```
ip_1  porto_1  I
```

4. Finalizar a execução de `ip_1:porto_1`:

```
ip_1  porto_1  F
```

5. Imprimir a tabela de roteamento de `ip_1:porto_1`:

```
ip_1  porto_1  T
```

6. Enviar (ou encaminhar) uma mensagem de `ip_1:porto_1` para o roteador de nome `nome_roteador_destino`:

```
ip_1  porto_1  E  mensagem  nome_roteador_destino
```

EXEMPLOS E RESULTADOS

Para o exemplo utilizado, vamos supor seis roteadores sendo executados com identificadores referentes ao seu número, por exemplo: o terceiro roteador possui o identificador R03. Ao entrarmos com uma sequência de comandos de adição de vizinhos como na figura 1, os roteadores se conectam formando o grafo da figura 2. A medida que o roteamento é executado em cada roteador a cada segundo, as tabelas de roteamento são formadas e terminam como mostrado ainda na figura 2.

```
localhost 2222 C localhost 1111 r01
localhost 2222 C localhost 3333 r03
localhost 1111 C localhost 5555 r05
localhost 5555 C localhost 6666 r06
localhost 6666 C localhost 4444 r04
localhost 3333 C localhost 4444 r04
```

Figura 1: sequência de comandos para conectar vizinhos

Como dito anteriormente, a figura abaixo representa o grafo resultante de um roteamento completo entre estes roteadores seguindo os comandos acima inseridos na interface de controle. A figura mostra também as respectivas tabelas de roteamento finais de cada roteador no formato (nome_destino, distância, nome_próximo_passo).

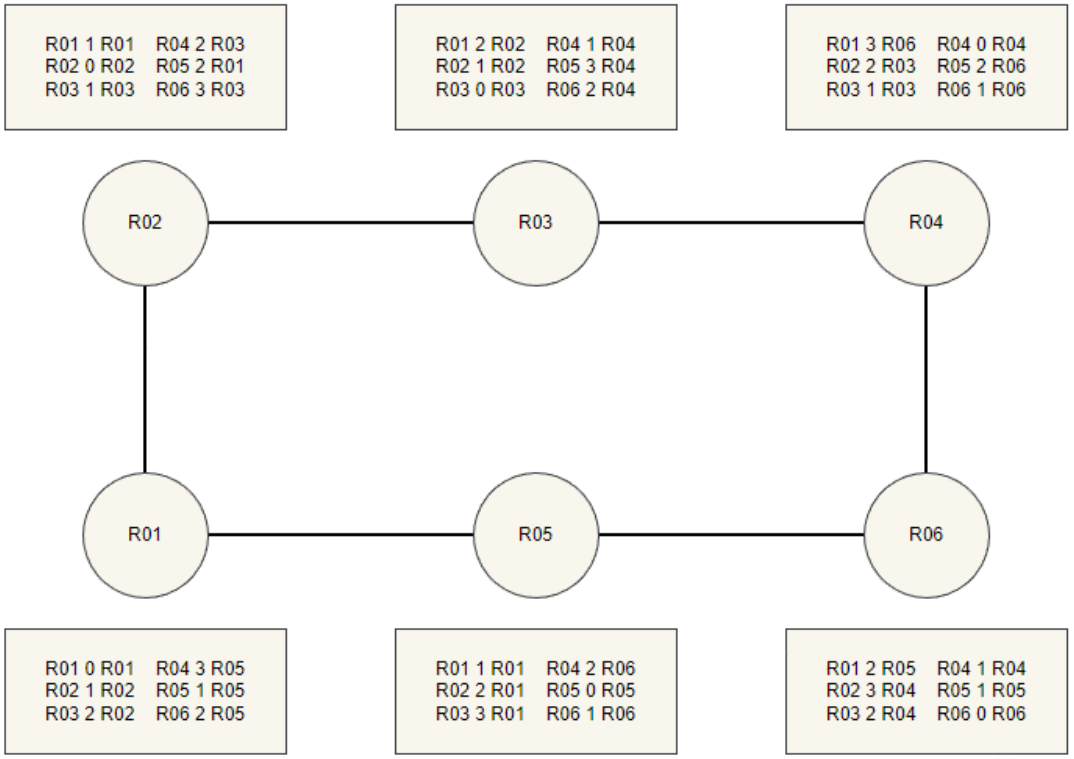
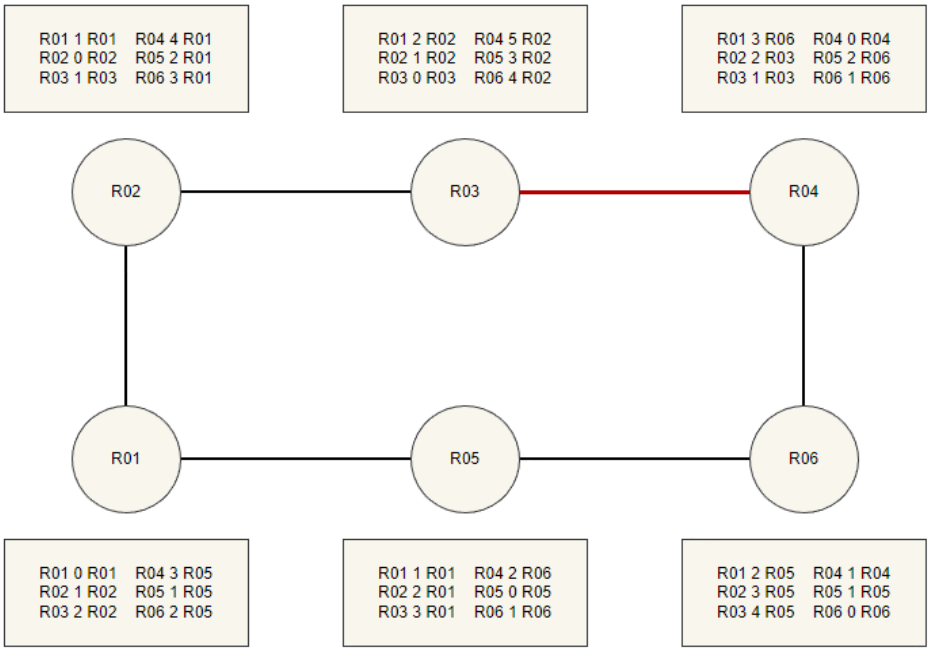


Figura 2: grafo de roteamento e tabelas de roteamento

Com este grafo construído, ao desconectarmos o link entre R03 e R04 com o comando `localhost 3333 D localhost 4444`, a thread executando o método de roteamento irá construir um novo grafo com novas tabelas de roteamento. Perceba que na imagem abaixo, o roteador R03 atualizou todas as suas rotas que passavam por R04, propagando essa mudança para o resto do grafo através dos anúncios.



Neste momento, ao enviar uma mensagem do roteador R02 ao roteador R04, a mensagem navega do roteador R02 até o R04 passando por R01, R05, R06 e finalmente chegando ao R04. O comando executado na interface de controle para isto seria `localhost 2222 E OLAROTEADOR04 r04`. A saída em cada roteador intermediário segue o formato `E OLAROTEADOR04 de r02 para r04 através de r01`, neste caso ao repassar do roteador R02 para o R01. No roteador destino (R04) a saída resultante é `R OLAROTEADOR04 de r02 para r04`.

REFERÊNCIAS

[1] https://pythontic.com/modules/socket/sendto
[2] https://docs.python.org/3/library/threading.html#timer-objects
[3] https://pythontic.com/modules/socket/udp-client-server-example
[4] https://www.geeksforgeeks.org/distance-vector-routing-dvr-protocol/