2022_2 - REDES DE COMPUTADORES - TN

Trabalho prático 2: DCCRIP, um protocolo de roteamento

Marcar como feito

Trabalho em dupla

Prazo de entrega: confira o moodle, no link de entrega

Introdução

Neste trabalho vamos implementar o DCCRIP, um protocolo simples de roteamento, seguindo o algoritmo de Vetor de Distâncias.

Objetivo

Seu objetivo será criar programa que operará como um roteador usando links "virtuais" que se ligará a outros iguais a ele e executará o algoritmo de vetor de distâncias para encontrar caminhos.

O princípio de operação

Serão desenvolvidos dois processos, o roteador do DCCRIP e um programa simples que funcionará como uma interface de controle. Essa interface receberá comandos da entrada padrão e gerará mensagens de controle para os roteadores indicados em cada comando. Ambos os tipos de programa utilizarão UDP como protocolo de transporte.

Haverão, assim, dois protocolos: um protocolo entre o cliente e o roteador, para controle destes, e o protocolo entre roteadores. Os dois, bem como a linguagem de comandos implementada pelo cliente, são descritos a seguir. Para permitir a diferenciação das mensagens, todas elas começam com um inteiro curto (short int), em ordem da rede (network byte order) que conterá um valor específico para cada tipo de mensagem.

UDP é um protocolo sem garantias de entrega; apesar disso, em uma rede local, sem tráfego de rede pesado e com máquinas sem muita carga, perdas não devem acontecer. Para simplificar, você não precisa se preocupar em implementar técnicas de detecção e recuperação de erros na aplicação. O ambiente de avaliação será configurado de forma a evitar perdas.

Funcionamento dos roteadores

Cada programa roteador receberá como parâmetros de linha de comando um string de até 15 caracteres, sem espaços, que será o identificador do roteador e um inteiro que indicará o porto onde o mesmo deverá esperar por mensagens UDP. Essas mensagens poderão ser comandos gerados pelo programa de interface de controle ou mensagens de outros roteadores, que podem ser parte do algoritmo de vetor de distâncias, ou outras mensagens trocadas entre roteadores.

Quando for iniciado, um roteador deve esperar por mensagens no porto indicado. Inicialmente, apenas mensagens do protocolo entre clientes e roteadores devem ser aceitas (outras podem ser simplesmente ignoradas). Apenas depois de um comando específico de controle vindo de um cliente um roteador deverá iniciar a execução do algoritmo de roteamento, passando a trocar mensagens com outros roteadores (e deve continuar aceitando comandos de controle de programas de interface).

O protocolo de roteamento

Cada roteador será identificado pela string de texto que receberá como parâmetro. Inicialmente, um roteador só conhece o caminho para sua rede local, que será identificada por seu nome e terá distância zero. A identificação dos outros roteadores e os caminhos para eles serão descobertos apenas pela operação do protocolo.

Cada roteador deve manter uma lista dos enlaces de dados que fazem sua ligação com seus vizinhos. Esses enlaces serão identificados por triplas (nome,endereço IP,porto) que identificarão o nome de um roteador, o endereço da máquina onde o mesmo foi iniciado e o número do porto que lhe foi passado como parâmetro. Endereços IP e portos serão usados nos comandos da API de sockets, mas não farão parte dos dados trocados pelo algoritmo de roteamento, que usará apenas os nomes dados aos roteadores. Um roteador pode tomar conhecimento de um enlace de duas formas: por uma mensagem de configuração de um programa da interface de controle ou ao receber uma mensagem de roteamento vinda de um vizinho.

As tabelas de roteamento de cada roteador deverão conter três colunas: o nome de um roteador de destino, a distância do roteador que contém a tabela até aquele destino e o nome do próximo roteador a ser contactado naquele caminho. Inicialmente, então, a tabela de cada roteador terá apenas uma linha: <meu nome>, 0, <meu nome>.

Para simplificar a implementação e reduzir o número de mensagens, roteadores enviam apenas mensagens periódicas, a cada segundo - atualizações da tabela ou mudança no estado dos enlaces não devem gerar mensagens imediatas. Mais à frente discutiremos como isso pode ser implementado.

Não é preciso implementar qualquer técnica especial para determinar quando um nó ou um link falham. Isso será simulado explicitamente por um comando da interface de controle. Quando um nó ou link falham, a rota deve ser marcada como com distância infinita. No nosso caso, vamos adotar infinito igual a 16, como no RIP.

O protocolo de roteamento usa apenas uma mensagem, que é o anúncio das rotas de um roteador. A cada temporização, uma mensagem deve ser montada para ser enviada para cada um dos vizinhos do roteador em questão.

Ao divulgar rotas, o algoritmo deve implementar as técnicas de *split horizon* e *poisoned reverse*.

Cada mensagem deve ter a seguinte forma:

- o inteiro curto (short int) com o número do comando que identifica a mensagem deverá ter o valor 11111;
- o próximo campo será o nome do roteador que envia a mensagem;
- o número de rotas incluídas no anúncio, como um inteiro curto (short int);
- em seguida, segue cada anúncio de rota, como um par: o nome do destino e um inteiro curto com o custo da rota.

Caso o roteador que recebe a mensagem ainda não tenha o roteador que enviou a mensagem em sua lista de vizinhos, ele deve incluí-lo antes de processar a mensagem (seria o equivalente a receber uma mensagem do tipo C de um processo de interface - descrito a seguir);

O protocolo de controle entre processos de interface e roteadores

O processo de interface não recebe parâmetros de linha de comando. Ele deverá ler comandos da entrada padrão e se comunicar com o roteador indicado no comando, para enviar uma mensagem com a operação desejada. Os comandos da interface começam com um endereço IP (ou nome) e porto que identificam um roteador alvo do comando, seguidos uma letra que identifica a operação desejada e finalmente os parâmetros necessários. Cada campo é separado dos demais por um ou mais espaços em branco. Os comandos possíveis são:

1. end_IP_1 porto_1 C end_IP_2 porto_2 nome - ``conecta'' o roteador identificado por end_IP_1:porto_1 com o roteador identificado por end_IP_2:porto_2 (cujo nome também é informado), indicando que ele deve receber anúncios de rotas;
2. end_IP_1 porto_1 D end_IP_2 porto_2 - ``desconecta'' o roteador end_IP_1:porto_1 de end_IP_2:porto_2, isto é, o segundo roteador deixa de aparecer na lista de vizinhos do primeiro (o que deve gerar uma atualização de rota de custo infinito);
3. end_IP_1 porto_1 I - inicia o funcionamento do algoritmo de roteamento no roteador identificado por end_IP_1:porto_1;
4. end_IP_1 porto_1 F - finaliza a execução do roteador identificado por end_IP_1:porto_1;
5. end_IP_1 porto_1 T - roteador identificado por end_IP_1:porto_1 deve imprimir a sua tabela de rotas;
6. end_IP_1 porto_1 E mensagem_de_texto destino - entrega a mensagem de texto (até 40 caracteres, sem espaços) para o roteador identifidado por end_IP_1:porto_1 e pede que ele a roteie para o destino indicado por end_IP_2:porto_2.
7. 0 (zero) 0 (zero) S tempo - o cliente deve executar um sleep pelo tempo definido (em segundos, podendo ser um número de ponto flutuante).

Exceto pelo comando especial 0 0 S, cada comando deve gerar uma mensagem iniciada com um inteiro curto com o valor associado a cada comando na lista acima. Cada implementação pode formatar os demais campos (quando necessários) da forma como preferirem. O formato de cada mensagem deve ser incluído no relatório que será entregue.

Um roteador que recebe uma dessas mensagens deve se comportar da seguinte forma:

- C end_IP porto nome_roteador - inclui o roteador identificado pelo endereço e número de porto, com o nome dado, como um de seus vizinhos;
- D nome_roteador - remove o roteador identificado de sua lista de vizinhos (o que deve gerar uma atualização de rota de custo infinito);
- I - o roteador deve esperar um segundo (sleep(1)) e iniciar o algoritmo de roteamento enviando mensagems para seus vizinhos;
- F - o roteador deve terminar sua execução;
- T - roteador deve imprimir a sua tabela de rotas na sua saída padrão seguindo o padrão:
  - nome do roteador \n
  - nome_de_destino custo nome_do_próximo_passo \n
  - (repetir para os demais destinos)
  - uma linha em branco;
- E mensagem_de_texto destino - o roteador deve:
  - imprimir na sua saída padrão "E" + mensagem_de_texto + " de " + nome do roteador que recebeu o comando + " para " + destino, seguido de uma linha em branco;
  - criar uma nova mensagem, com valor do inteiro curto igual a 9999 seguido, pelo menos, do nome do roteador que recebeu o comando E (que passa a ser chamado de origem) e do nome do roteador de destino;
  - fazer o encaminhamento da mensagem pela rede, até o destino final, seguindo o procedimento descrito a seguir.

Encaminhamento de mensagens pela rede

Caso o nome do destino seja o nome do roteador que recebe a mensagem, ele deve escrever "R" + mensagem_de_texto + " de " + origem + " para " + destino, seguido de uma linha em branco, e descartar a mensagem.

Caso contrário, se o nome do destino não for encontrado na tabela de rotas do roteador que a recebe, ele deve escrever na sua saída padrão "X" + mensagem_de_texto + " de " + origem + " para " + destino, seguido de uma linha em branco, e descartar a mensagem.

Finalmente, se o destino for encontrado na tabela de roteamento, o roteador deve escrever na sua saída padrão "E" + mensagem_de_texto + " de " + origem + " para " + destino + " através de " + nome_ do_próximo_passo.

O tratamento de temporizações em um programa

Para executar periodicamente o envio de anúncios para os vizinhos, a implementação pode utilizar temporizações regulares usando a interface Timer do pacote threading - threading.Timer(). Para repetir uma temporização continuamente, a solução mais simples é criar um novo timer a cada vez que a função_a_executar é disparada. Na web é possível encontrar soluções mais elegantes, caso queiram experimentar.

Observe que uma temporização pode acontecer em qualquer instante e é executada como uma thread independente, portanto podem haver problemas de sincronização. Em uma implemeentação mais usual do roteador, haveriam apenas duas threads: a que faria recvfrom() e processaria as mensagens recebidas, e a que executaria a rotina do timer. Como uma lê e a outra altera uma estrutura de dados não trivial (a tabela de rotas), semáforos ou variáveis de exclusão mútua são uma necessidade nesse caso, pela natureza imprevisível de uma temporização. Basta proteger o código de acesso à tabela em cada thread.

Se as temporizações ocorressem com mais frequência (menores intervalos de tempo) e/ou o volume de processamento da tabela fosse mais complexo, poderiam haver problemas por não haver tempo para terminar uma região crítica antes do próximo evento, mas nesse sistema isso não deve ser um problema.

Sobre a execução dos programas:

O código deve usar apenas Python 3, sem bibliotecas além das consideradas padrão. O material desenvolvido por você deve executar sem erros nas máquinas linux do laboratório de graduação. A correção será feita naquelas máquinas.

O que deve ser entregue:

Você deve entregar um arquivo .zip contendo os seguintes elementos:

- a implementação do processo roteador;
- a implementação do processo de interface;
- um arquivo Makefile (descrito a seguir);
- relatório em PDF.

Novamente, o material desenvolvido por você deve executar sem erros nas máquinas linux do laboratório de graduação. A correção será feita naquelas máquinas e e programas que não executarem, não seguirem as determinações quanto ao formato da entrada e da saída, ou apresentarem erros durante a execução, serão penalizados/desconsiderados (dependendo da gravidade do problema encontrado).

O relatório não precisa ser longo, mas deve documentar as principais decisões de projeto consideradas - **em particular, ele deve detalhar o formato em bytes de cada tipo de mensagem.**

Preste atenção nos prazos: entregas com atraso poderão ser aceitas em casos especiais (converse antes com o professor), mas serão penalizadas.

O makefile a ser entregue:

Junto com o código deve ser entregue um makefile que inclua, pelo menos, as seguintes regras:

- `run_cli` - executa o programa cliente que implementa a interface de comandos
- `run_rt` - executa o programa roteador

As regras do tipo "run_*" devem se certificar de disparar quaisquer regras intermediárias que podem ser necessárias para se obter um programa executável.

Para o make run funcionar, você pode considerar que os comandos serão executados da seguinte forma (possivelmente, em diferentes terminais):

```
make run_cli
make run_rt arg1=roteador01  arg2=5555
```

Obviamente, o valor dos argumentos pode variar. Se todos os programas forem executados na mesma máquina, o nome do servidor pode ser "localhost" em todos os casos - mas os programas devem funcionar corretamente se disparados em máquinas diferentes.

Para poder executar os comandos, no makefile, supondo que os programas tenham nomes "cli_interface.py" e "roteador.py", as regras seriam:

```
run_cli:
	python3 cli_interface.py
run_rt:
	python3 roteador.py $(arg1) $(arg2)
```

Sugestões de depuração

Faz parte do exercício desenvolver os testes para o mesmo. Vocês devem criar os seus próprios arquivos e testes e podem compartilhá-los no fórum do exercício. Por outro lado, obviamente não é permitido discutir os princípios de operação da solução.

Dúvidas?

Usem o fórum criado especialmente para esse exercício de programação para enviar suas dúvidas. **Não é permitido publicar código no fórum**! Se você tem uma dúvida que envolve explicitamente um trecho de código, envie mensagem por e-mail diretamente para o professor.

◄ Programação do curso

Seguir para...

2022_2 - Plano de Ensino ►