Redes de Computadores
Trabalho 2
Roteamento IP

Profa. Hana Karina S. Rubinsztejn

O objetivo

Do ponto de vista funcional, o objetivo do trabalho é implementar um sistema que simula o encaminhamento de pacotes em uma rede, passando por vários roteadores.

Neste trabalho você precisará implementar dois tipos de programas. O primeiro programa apenas cria pacotes de mensagem e envia para o seu segundo programa. O segundo programa simula a função de um roteador, encaminhando o pacote para o próximo roteador ou entregando a seu destino final, de acordo com a tabela de roteamento.

O problema

Você desenvolverá um sistema simples de roteamento de mensagens em C/C++, Java ou Python, sem bibliotecas especiais, utilizando comunicação via protocolo UDP.

Dois programas devem ser desenvolvidos: um **emissor**, para envio de mensagens na rede, e um programa "**roteador**" que realiza o encaminhamento e das mensagens recebidas. Por simplicidade este último também fará o papel do **destinatário**, exibindo a mensagem recebida quando for o caso. Serão inicializados vários programas roteadores e emissores. (Obs: caso use múltiplas janelas na interface gráfica, pode-se fazer apenas 1 programa, lendo as configurações de um arquivo).

Cada mensagem será enviada a algum roteador, que a repassa para o próximo roteador, e assim sucessivamente até ser entregue ao destinatário.

Programas emissor

O programa emissor recebe como argumentos o número IP e porta do roteador *default (*inicial*)*; dois endereços IP (origem e destino) e uma mensagem. Com essas informações, cria um pacote IP com os endereços IPs de origem e destino e com *payload* preenchido com a mensagem. Esse pacote é então encaminhado para o roteador e porta UDP fornecidos. Ou seja, você está enviando um pacote IPv4 sobre UDP para o seu programa "roteador" que será descrito a seguir.

Atenção: Deve ser criado um pacote IP com cabeçalho padrão (completo). Ou seja, roteadores de qualquer implementação devem ser capazes de rotear o pacote. O campo TTL deve ser setado em 5.

Por exemplo:

```
$ ./emissor  127.0.0.1  12345  10.0.0.5   1.2.3.4  Hello
```

cria um pacote IP com endereço de origem `10.0.0.5` e o endereço destino `1.2.3.4` e conteúdo `Hello` . Ele envia o pacote para o roteador local (`127.0.0.1`) e a porta UDP `12345`.

Você pode implementar todos os argumentos na linha de comando, ou ir solicitando cada um separadamente durante a execuação (por exemplo, para enviar várias mensagens durante a mesma execução).

O roteador

O roteador recebe como parâmetro a porta em que irá executar.

Ao receber um pacote, o roteador decrementa o TTL e caso ainda esse seja maior que 0, avalia o endereço de destino e verifica a ação a ser tomada:

(i) repassar a mensagem a um próximo roteador (roteamento indireto), ou
(ii) entregá-la ao destino (roteamento direto).

Você deve implementar um tipo de tabela de encaminhamento IP para encontrar um roteador ou host para o qual o pacote deve ser repassado. Essa busca é usada como parte do processo de encaminhamento.

A tabela de roteamento deste trabalho é estática e é especificada como argumento da linha de comandos.

Após pesquisar a tabela, se a ação for *encaminhamento*, o programa deve imprimir: o endereço de destino do pacote, a ação (encaminhado ou entregue), o próximo salto (*hop*) e a sua interface (neste caso, a porta) para o qual o pacote deve ser encaminhado (atenção: na seção de exemplos são apresentadas as mensagens de saída a serem impressas).

Em seguida deve encaminhar o pacote ao próximo roteador, através de uma mensagem UDP para a porta especificada (indicada pela interface na tabela).

Mas, caso o pacote deva ser enviado ao *host* destino (roteamento direto), o programa deve apenas indicar que o pacote chegou ao destino e imprimir a **mensagem** contida no pacote, bem como os IPs de origem e destino.

<u>Tabela de Roteamento</u>

Cada entrada na tabela de roteamento armazena:

1. um Destino (rede de destino), que é um endereço IP.

2. uma máscara, que é uma string de 32 bits, mas não é um endereço IP.
3. um *Gateway* ou Próximo *Hop*, que também é um endereço IP (Obs: neste trabalho, será o IP do programa roteador, por exemplo 127.0.0.1)
4. uma interface de saída, que é uma string arbitrária. (Obs: neste trabalho vamos usar a interface para indicar a porta que o gateway está rodando)

Neste trabalho, a tabela de roteamento será expecificada como uma série de argumentos de linha de comando no seguinte formato :

```
rede-destino/máscara/gateway/interface
```

Cada destino, máscara e *gateway* são fornecidos na notação número-ponto , suportada pelo `inet_aton`.

Máscaras de rede também podem ser especificadas na notação CIDR (número de bits '1' mais significativos). Por exemplo, a máscara de uma rota para uma rede classe B é /16, e para um host é /32 . Você pode, por exemplo, implementar o código para verificar se a entrada possui um caractere '.' e , se não, (e o número estiver entre 0 e 32), usar o número correspondente como o número de bits 1 para ser colocado na máscara.

Cada argumento desses é usado para criar uma entrada na tabela de roteamento na ordem que forem fornecidas na linha de comando.

Para procurar na tabela de roteamento, o IP usa um algoritmo chamado de ***longest match***. De todas as entradas na tabela de roteamento que casariam com um padrão, aquele com a maior máscara é usado. Se existirem diversas entradas que casam com o padrão e tem máscara de mesmo tamanho, a primeira é usada.

Todos os bits '1' são uma máscara válida e estão a esquerda de todos os bits '0'. Logo, a maior máscara é também numericamente maior quando comparando inteiros não sinalizados de 32 bits.

Uma rota "casa" com um endereço IP quando , fazendo um & binário entre o campo destino e máscara da tabela, o resultado for igual ao mesmo procedimento entre o endereço IP e a máscara, ou

```
(route[i].dest & route[i].mask) == (ip->dest & route[i].mask)
```

As máscaras totalmente com zero e totalmente com uns são legais. A primeira é a rota padrão (*default*), usada se nenhuma outra rota casar. A segunda é a rota para o host, usada somente se o destino do pacote é o mesmo destino que o da tabela de roteamento.

O valor 0.0.0.0 no campo Gateway da tabela, indica que não há próximo *hop*, ou seja, a mensagem chegou a um roteador ligado a rede do IP de destino (roteamento direto).

Exemplos

1) Supondo um roteador executado com os seguintes parâmetros:

```
$ ./roteador 12345  1.0.0.0/255.0.0.0/127.0.0.1/1112
1.2.0.0/255.255.0.0/2.4.6.9/1112 1.2.3.0/255.255.255.0/1.2.3.1/1113
```

Esse roteador ao receber um pacote (criado/enviado pelo programa emissor) em sua porta 12345, irá avaliar o IP de destino indicado no cabeçalho.

Suponha que o pacote tenha como destino o endereço 1.2.3.4. O programa deve emitir a seguinte mensagem

```
> forwarding packet for 1.2.3.4 to next hop 1.2.3.1 over interface 1113
```

O endereço IP (1.2.3.4) casa com todas as três primeiras rotas que foram fornecidas, e aquele que tem a máscara mais longa é a terceira, que é a usada. Além disso, o pacote deve ser retransmitido ao roteador 1.2.3.1 na porta 1113. Neste exemplo, haveria outros 3 programas roteadores em execução.

Observação: Nos exemplos os endereços dos roteadores são fictícios. Na prática o endereço do next hop será o IP onde o programa roteador está executando (127.0.0.1) e a interface será a sua porta UDP.

Se o pacote fosse enviado para o destino 1.2.5.6, nós teríamos:

```
> forwarding packet for 1.2.5.6 to next hop 2.4.6.9 over interface 1112
```

Agora, apenas as duas primeiras rotas casaram, logo, a segunda (a mais longa das duas) rota é a escolhida.

A ordem na tabela não importa se existir apenas uma única rota mais longa. Mas, no caso de múltiplos casamentos mais longos, o primeiro deve ser usado.

2) A rota padrão é usada somente se não existir um casamento melhor

```
$./roteador 1245 0.0.0.0/0.0.0.0/1.2.3.1/eth1 1.2.0.0/255.255.0.0/2.4.6.9/eth0
```

Se o pacote fosse enviado para 1.2.5.6,

```
> forwarding packet for 1.2.5.6 to next hop 2.4.6.9 over interface eth0
```

Para 9.8.7.6:

```
> forwarding packet for 9.8.7.6 to next hop 1.2.3.1 over interface eth1
```

Obs: Em todos os casos acima o pacote deve se enviado ao roteador indicado.

3) Algumas vezes, não existe casamento nenhum:

```
$ ./roteador 12345  1.0.0.0/255.0.0.0/2.4.6.8/1112
1.2.0.0/255.255.0.0/2.4.6.9/1112 1.2.3.0/255.255.255.0/1.2.3.1/1113
```

IP destino do pacote = `9.8.7.6`:

```
> destination 9.8.7.6 not found in routing table, dropping packet
```

Pacote descartado!!!

Caso seja recebido um pacote com TTL = 1, este será descartado.

```
> Time to Live exceeded in Transit, dropping packet for 9.8.7.6
```

4) Por fim:

```
$ ./roteador 12345  1.0.0.0/255.0.0.0/2.4.6.8/1112
1.2.0.0/255.255.0.0/2.4.6.9/1112 1.2.3.0/255.255.255.0/0.0.0.0/0
```

Caso seja recebido um pacote destinado a `1.2.3.4`, o programa identifica que este pacote é destinado a um *host* da rede local (roteamento direto), ao qual deve ser entregue a mensagem.

```
>  destination reached. From 10.0.0.5  to  1.2.3.4 :  Hello
```

O que deve ser apresentado

Cada grupo (de até 3 participantes) deve entregar junto com o código um relatório curto que deve conter uma descrição da arquitetura adotada, as estruturas de dados utilizadas, decisões de implementação não documentadas nesta especificação e indicar as funcionalidades não implementadas, ou outras extras implementadas.

Submissão eletrônica

Os trabalhos deverão ser entregues em um arquivo do tipo zip ou tar.gz através do *moodle*. Os códigos fonte devem ser submetidos com comentários que identifiquem no código as partes do algoritmo de alto nível e expliquem os detalhes das partes principais do programa. Junto do código fonte, um arquivo Makefile deve ser incluído, de forma a facilitar a compilação do programa.

Critérios de avaliação

Serão atribuídos pontos para a execução correta do programa, para a organização e clareza do relatório e para a organização do código.

Observações Gerais

1. Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.
2. **Planeje para que seu código seja robusto. Tente prever falhas no processo de comunicação entre os componentes.**
3. **Vão valer pontos clareza, indentação e comentários no programa, bem como a qualidade da documentação que o acompanhe.**
4. O trabalho deve ser desenvolvido de forma independente por cada grupo. É terminantemente proibido compartilhar programas ou trecho de programas. Tal comportamento poderá ser punido severamente.