Trabalho Redes

Roteamento Distance-Vector

Visão Geral

Neste projeto, você irá implementar um protocolo de roteamento distance-vector (vetor de distâncias). Fornecemos código de simulador que você pode usar para visualizar seu protocolo em ação.

Aulas necessárias para este projeto: Aula sobre Roteamento e Vetor de Distância

Você pode trabalhar neste projeto sozinho ou com um parceiro.

Configuração Inicial

Instalação do Python

Este projeto foi testado com Python 3.8. (Provavelmente funciona em versões mais novas do Python, mas use por sua conta e risco.)

Execute `python3 --version` ou `python --version` no seu terminal. Se você ver uma saída do tipo `Python 3.8.*`, está tudo certo.

Código Inicial

Baixe uma cópia do código inicial fornecido pelo professor.

Recomendamos fazer backups frequentes do seu código. Algumas partes do projeto exigirão que você modifique o código que escreveu anteriormente, e você pode querer reverter suas alterações para começar de um ponto de salvamento anterior.

Você deve editar apenas o arquivo `dv_router.py`. Há comentários indicando claramente os lugares onde você deve preencher o código.

Diretrizes Importantes:

- Não modifique nenhum outro arquivo
- Não adicione nenhum arquivo novo
- Não adicione imports
- Não edite nenhum código fora das seções indicadas pelos comentários
- Não adicione variáveis globais hard-coded
- Adicionar métodos auxiliares é permitido (e incentivado na última parte)
- Em geral, se não foi dito para você usar algo, você provavelmente não precisa!

No seu terminal, use `cd` para navegar até o diretório `proj-routing/simulator`. Todos os comandos Python devem ser executados desse diretório.

Estrutura do Projeto

Este projeto está dividido em 3 seções, com um total de 10 etapas em todas as seções.

Cada etapa tem seus próprios testes unitários, fornecidos a você localmente. Sua nota será determinada apenas por esses testes unitários (sem testes ocultos).

Para executar um teste unitário, execute este comando, substituindo 5 pelo número da etapa que deseja testar:

```
python3 dv_unit_tests.py 5
```

Atenção: A Etapa 10 é significativamente mais longa que as outras etapas, então planeje-se adequadamente.

Exemplo para aquecimento: Hub

Algumas suposições antes de começarmos:

- Neste projeto, seu código encontrará rotas entre hosts. Você não precisa encontrar rotas para outros roteadores.
- Neste projeto, nossas tabelas de encaminhamento mapearão destinos para portas físicas numeradas, em vez de next-hops. Por exemplo, uma linha da tabela pode dizer: "Pacotes destinados a h1 devem ser enviados pela porta 1."

Para começar, fornecemos uma implementação de um hub em `examples/hub.py`. O hub é um dispositivo de rede que pega qualquer pacote de entrada e encaminha esse pacote para todas as suas portas (exceto a porta de onde o pacote veio).

Vamos executar o simulador em uma topologia linear com três hosts:

```
python3 simulator.py --start --default-switch-type=examples.hub topos.linear
--n=3
```

Você pode acessar o visualizador em http://127.0.0.1:4444 no seu navegador. Você verá os hosts e roteadores exibidos contra um fundo roxo.

Agora, vamos fazer o host h1 enviar um pacote ping para o host h3. Você pode digitar no terminal Python:

```
h1.ping(h3)
```

Ou enviar o ping pelo visualizador: (1) clicando em h1 e pressionando A, (2) clicando em h3 e pressionando B, e (3) pressionando P para enviar o ping de A para B.

Você verá o pacote "ping" enviado de h1 para h3. Então, verá o pacote de resposta "pong" enviado de h3 para h1. Você também verá ambos os pacotes entregues a h2, mesmo que h2 não seja o destinatário. Este comportamento é esperado, porque o hub inunda pacotes em todos os lugares.

Problema com Flooding em Loops

Lembre-se da aula que flooding é problemático quando a rede tem loops. Vamos ver isso em ação

lançando o simulador com a topologia `topos.candy`, que tem um loop:

```
python3 simulator.py --start --default-switch-type=examples.hub topos.candy
```

Envie um ping do host h1a para o host h2b. Você verá muitas mensagens de log no terminal, e o visualizador mostrará roteadores encaminhando pacotes supérfluos por bastante tempo. Ops! Seu trabalho neste projeto será implementar um roteador distance-vector mais capaz.

Visão Geral do Código

Você estará implementando a classe `DVRouter` em `dv_router.py`, que representa um único roteador.

Algumas classes auxiliares úteis são implementadas em `dv.py` (você pode ler este arquivo, mas não o modifique).

Classe Table

A variável de instância `self.table` é um objeto `Table` representando a tabela de encaminhamento dentro do roteador.

O objeto `Table` é um dicionário, onde as chaves são destinos e os valores são objetos `TableEntry`.

Cada objeto `TableEntry` contém:

- `dst`: O destino para esta rota
- `port`: A porta pela qual os pacotes para o destino devem ser enviados
- `latency`: A latência ("custo") da rota deste roteador até o destino
- `expire_time`: O timestamp (em segundos) no qual esta rota expira. Use `api.current_time()` para obter o tempo atual e adicione segundos para indicar um tempo no futuro.

Os objetos `TableEntry` são imutáveis, então se você quiser atualizar uma entrada, deve criar um novo `TableEntry` com atributos atualizados.

Exemplo de uso:

```
t = Table()
t[h1] = TableEntry(dst=h1, port=p1, latency=10, expire_time=api.current_time()
+20)
t[h2] = TableEntry(dst=h2, port=p2, latency=20, expire_time=api.current_time()
+20)

for host, entry in t.items():
    print("Rota para {} tem latência {}".format(host, entry.latency))
```

Isso corresponde a uma tabela que se parece com:

Chave	**Valor (TableEntry)**			
	Destino	Porta	Latência	Tempo Exp.
h1	h1	p1	10	20s depois
h2	h2	p2	20	20s depois

Classe Ports

A variável de instância `self.ports` é um objeto `Ports` representando os links conectados ao roteador.

O objeto `Ports` contém um dicionário, onde cada chave é um link (porta), e o valor correspondente é a latência (custo) ao longo desse link.

Para obter a latência ao longo do link conectado à porta p, você pode usar:

```
self.ports.get_latency(p)
```

Para iterar por todas as portas:

```
for p in self.ports.get_all_ports():
    # código aqui
```

Flags

As seguintes flags identificam em que modo o roteador está. Diremos quando incluí-las no código. Seu código não deve reatribuir essas flags para True ou False.

- `self.SPLIT_HORIZON`
- `self.POISON_REVERSE`
- `self.POISON_ON_LINK_DOWN`
- `self.POISON_EXPIRED`
- `self.SEND_ON_LINK_UP`

ETAPA 1: Instalando Rotas Estáticas

Para cada host conectado diretamente ao seu roteador, você deve registrar uma rota estática para esse host.

Sua Tarefa

Implemente `add_static_route`.

Atribua um tempo de expiração de `FOREVER` para a nova rota.

O framework chamará magicamente o código que você escrever aqui sempre que uma nova rota estática precisar ser instalada.

Testando e Depurando

Para verificar seu trabalho, execute os testes unitários:

```
python3 dv_unit_tests.py 1
```

Para depurar, você pode iniciar o simulador e usar o terminal para imprimir as tabelas dentro de cada roteador. Por exemplo, para imprimir a tabela dentro de s1:

```
print(s1.table)
```

Se você está depurando no código que escreveu, pode imprimir sua própria tabela assim:

```
print(self.table)
```

Verifique Sua Compreensão

Agora que implementamos rotas estáticas, o que você espera que aconteça se tentarmos enviar um pacote ao longo de um caminho?

Para descobrir, inicie o simulador e abra http://127.0.0.1:4444 no seu navegador:

```
python3 simulator.py --start --default-switch-type=dv_router topos.simple
```

Tente enviar um ping do host h1 para o host h2, por exemplo, digitando `h1.ping(h2)` no terminal.

O que aconteceu? Estava alinhado com sua expectativa?

ETAPA 2: Encaminhamento

Adicionamos entradas à tabela de roteamento, mas precisamos realmente usá-las para encaminhar pacotes.

Sua Tarefa

Implemente `handle_data_packet`.

Este método será chamado cada vez que um pacote de dados chegar ao seu roteador.

Dica: Para enviar um dado `packet` pela porta `port`, você pode usar:

```
self.send(packet, port=port)
```

Se nenhuma rota existir para o destino do pacote, você deve descartar o pacote (não fazer nada).

Se a latência ao longo do link de saída for maior ou igual a `INFINITY`, você também deve descartar o pacote.

Testando e Depurando

Para verificar seu trabalho, execute os testes unitários:

```
python3 dv_unit_tests.py 2
```

Verifique Sua Compreensão

Inicie o simulador da etapa anterior novamente:

```
python3 simulator.py --start --default-switch-type=dv_router topos.simple
```

Novamente, tente enviar um ping do host h1 para o host h2. Esperamos que o pacote tenha chegado

desta vez!

Você terminou a primeira seção! Agora, vamos adicionar mais rotas à nossa tabela.

ETAPA 3: Enviando Anúncios

Verifique Sua Compreensão

Dada nossa implementação atual, o que deve acontecer quando pings são enviados de um host para outro nestas topologias?

- **Topologia 1:** h1 --- s1 --- h2
- **Topologia 2:** h1 --- s1 --- s2 --- h3

Os pings funcionam em ambas as topologias? Apenas uma delas? Nenhuma delas?

Para descobrir, inicie o simulador novamente para executar esses dois cenários:

```
python3 simulator.py --start --default-switch-type=dv_router topos.simple
```

Temos um problema na Topologia 2. Os roteadores coletivamente sabem sobre ambos os destinos, mas cada roteador por si só não sabe sobre ambos os destinos.

Para corrigir isso, vamos fazer os roteadores trocarem anúncios para que possam aprender sobre outros destinos na rede.

Sua Tarefa

Implemente `send_routes`.

Por enquanto, você pode assumir `force=True` e `single_port=None`, e ignorar esses argumentos.

O framework chamará periodicamente esta função para você, para que seu roteador anuncie periodicamente rotas para todos os seus vizinhos.

Dica: Para enviar uma mensagem pela porta específica `port`, dizendo que você pode alcançar um destino específico `dst` com uma latência específica `latency`, você pode usar:

```
self.send_route(port, dst, latency)
```

Testando e Depurando

Para verificar seu trabalho, execute os testes unitários:

```
python3 dv_unit_tests.py 3
```

Verifique Sua Compreensão

O que acontece agora se tentarmos enviar um ping nas Topologias 1 e 2?

Para descobrir, inicie o simulador novamente e experimente. Você notará que anúncios de roteamento (pontos roxos) estão sendo periodicamente enviados de cada roteador.

No entanto, o ping na Topologia 2 ainda não funciona.

ETAPA 4: Tratando Anúncios

Seu roteador agora envia anúncios, mas também precisa tratar quaisquer anúncios que receber! Em particular, precisaremos implementar a Regra 1 (atualização Bellman-Ford) e a Regra 2 (Atualizações do Next-Hop) do distance-vector.

Sua Tarefa

Implemente `handle_route_advertisement`.

O framework chamará isso para você toda vez que seu roteador receber um anúncio.

Se a rota anunciada for igualmente boa quanto a rota atual, desempate preferindo a rota atual. Em outras palavras, aceite uma rota anunciada apenas se ela for estritamente melhor que a rota atual.

Lembrete: Se você receber um anúncio para um destino que não está na tabela, você deve sempre aceitá-lo.

Lembrete: Se você receber um anúncio do next-hop atual, você deve sempre aceitá-lo. Esta é a Regra 2 (Atualizações do Next-Hop) do distance-vector.

Se você atualizar a tabela, defina o tempo de expiração da nova entrada como `api.current_time() + self.ROUTE_TTL`. (Por padrão, isso é 15 segundos no futuro.)

Testando e Depurando

Para verificar seu trabalho, execute os testes unitários:

```
python3 dv_unit_tests.py 4
```

Verifique Sua Compreensão

Por que escolhemos desempatar preferindo a rota atual? E se preferíssemos a nova rota?

Há um trade-off entre correção e estabilidade aqui.

Para ver o que queremos dizer com estabilidade, temporariamente mude sua implementação para desempatar preferindo a nova rota (aceite a rota anunciada se for igualmente boa quanto a rota atual).

Então, inicie o simulador com a topologia quadrada:

```
python3 simulator.py --start --default-switch-type=dv_router topos.square
```

Envie vários pings entre os dois cantos do quadrado. Note que pacotes entre os mesmos dois hosts podem tomar caminhos diferentes. Consideramos isso instável e subótimo.

(Preview: Pacotes tomando caminhos diferentes podem acabar chegando fora de ordem, e isso faz o TCP, o protocolo de confiabilidade da Camada 4, desacelerar.)

Agora, desfaça sua mudança temporária, para que você novamente desempate preferindo a rota atual (aceite apenas a rota anunciada se for estritamente melhor).

Então, inicie o simulador novamente e envie vários pings novamente. Nossos pacotes sempre tomam o mesmo caminho agora!

Então, qual é o trade-off? Preferir o caminho atual é mais estável. Por outro lado, uma nova rota representa um estado mais recente e, portanto, mais indicativo do estado atual da rede.

ETAPA 5: Tratando Timeouts

Agora, vamos implementar a Regra 3 (Expiração) do distance-vector.

Verifique Sua Compreensão

O que acontece se um link cair? Para descobrir, inicie o simulador com a topologia candy:

```
python3 simulator.py --start --default-switch-type=dv_router topos.candy
```

Aguarde todas as rotas se propagarem. Então, remova o link entre os roteadores s4 e s5. Você pode selecionar os dois roteadores e pressionar E no navegador, ou digitar `s4.unlinkTo(s5)` no terminal.

Agora, tente enviar um ping de h1a para h2b. Mesmo que ainda haja um caminho através de s3, você verá o pacote ser encaminhado para s4 e então descartado!

O roteador s4 está tentando encaminhar para s5, ao qual ele não está mais conectado!

Sua Tarefa

Implemente `expire_routes`.

O framework chamará esta função periodicamente para você. Seu trabalho é percorrer as entradas da tabela de encaminhamento e deletar quaisquer entradas que estejam expiradas.

Opcional, sem nota: Quando você deletar uma rota expirada, use `self.s_log` ou `self.log` para imprimir uma mensagem no terminal.

Dica: Para remover uma entrada da tabela com chave `h`, você pode usar:

```
self.table.pop(h)
```

Testando e Depurando

Para verificar seu trabalho, execute os testes unitários:

```
python3 dv_unit_tests.py 5
```

Verifique Sua Compreensão

Inicie o simulador com a topologia candy novamente:

```
python3 simulator.py --start --default-switch-type=dv_router topos.candy
```

Novamente, derrube o link, por exemplo `s4.unlinkTo(s5)`. Comece a enviar vários pings de h1a para h2b. Aproximadamente 15 segundos após o link cair, a rota antiga expirará, e futuros pings devem ser encaminhados corretamente ao longo da rota alternativa!

Você terminou a segunda seção! Agora, vamos tornar nosso roteador mais eficiente.

ETAPA 6: Split Horizon

Agora, vamos implementar a Regra 6A (Split Horizon) do distance-vector.

Verifique Sua Compreensão

Inicie o simulador com a topologia linear com 3 hosts:

```
python3 simulator.py --start --default-switch-type=dv_router topos.linear --
n=3
```

Aguarde todas as rotas se propagarem.

Envie um ping de h3 para h1. Isso deve funcionar.

Em seguida, derrube o link entre s1 e s2, por exemplo `s1.unlinkTo(s2)`.

Agora, tente enviar pings de h3 para h1. Você deve ver pacotes sendo descartados em s2.

Eventualmente, a rota em s2 expirará, e s2 receberá um novo anúncio de s3. Agora, quando você enviar um ping de h3 para h1, o que acontece e por quê?

Quanto tempo este loop de roteamento continuará? Qual é o problema maior e como podemos abordar esta questão?

Sua Tarefa

Modifique `send_routes` para suportar split horizon se a flag `self.SPLIT_HORIZON` for True.

Testando e Depurando

Para verificar seu trabalho, execute os testes unitários:

```
python3 dv_unit_tests.py 6
```

Verifique Sua Compreensão

Defina `self.SPLIT_HORIZON` como True no seu código e execute a mesma demonstração novamente. O problema do loop de roteamento deve estar resolvido agora!

ETAPA 7: Poison Reverse

Agora, vamos implementar a Regra 6B (Poison Reverse) do distance-vector.

Verifique Sua Compreensão

Defina `self.SPLIT_HORIZON` como False e inicie o simulador com a topologia double triangle:

```
python3 simulator.py --start --default-switch-type=dv_router
topos.double_triangle
```

Desconecte s2 removendo todos os links que o conectam a outros roteadores:

```
s2.unlinkTo(s1)
s2.unlinkTo(s3)
s2.unlinkTo(s4)
```

Vamos ver se split horizon nos salvará! Ative o split horizon e execute novamente a demonstração.

Podemos fazer melhor que isso e, se sim, em quanto?

Sua Tarefa

Modifique `send_routes` para implementar anúncios poison reverse se a flag `self.POISON_REVERSE` for True.

Nota: `self.POISON_REVERSE` e `self.SPLIT_HORIZON` nunca serão True ao mesmo tempo. Ambos podem ser False, ou um deles pode ser True.

Testando e Depurando

Para verificar seu trabalho, execute os testes unitários:

```
python3 dv_unit_tests.py 7
```

Verifique Sua Compreensão

Defina `self.POISON_REVERSE` como True e execute novamente a demonstração double triangle. Quanto tempo levará para alcançar o estado correto agora?

ETAPA 8: Count to Infinity

Agora, vamos implementar a Regra 7 (Count to Infinity) do distance-vector.

Verifique Sua Compreensão

Lembre-se de que split horizon e poison reverse podem prevenir loops de roteamento de comprimento 2, mas não os mais longos.

Para ver por quê, inicie o simulador com a topologia loopy:

```
python3 simulator.py --start --default-switch-type=dv_router topos.loopy
```

Aguarde todas as rotas se propagarem.

Desconecte s1 de s2, por exemplo `s2.unlinkTo(s1)`.

Você pode imprimir as tabelas de encaminhamento no terminal, por exemplo `print(s1)`. Até onde os roteadores contarão? Parece ser para sempre... apesar de todo nosso trabalho duro!

Estabilizaremos eventualmente? Split horizon ou poison reverse podem salvar o dia?

Sua Tarefa

Modifique `send_routes` para que qualquer latência maior que `INFINITY` seja arredondada para baixo para `INFINITY` quando anunciada.

Testando e Depurando

Para verificar seu trabalho, execute os testes unitários:

```
python3 dv_unit_tests.py 8
```

Bug comum: Na Etapa 2, não esqueça de considerar que você não deve encaminhar pacotes de dados se a distância na tabela for `INFINITY`. Se você esqueceu disso, os testes não detectarão seu bug até a Etapa 8.

Verifique Sua Compreensão

Execute novamente a demonstração loopy. Até onde os roteadores contarão agora?

ETAPA 9: Envenenando Rotas Expiradas

Agora, vamos implementar a Regra 5 (Envenenando Rotas Expiradas) do distance-vector.

Verifique Sua Compreensão

Inicie o simulador com a topologia linear com 7 hosts:

```
python3 simulator.py --start --default-switch-type=dv_router topos.linear --
n=7
```

Aguarde todas as rotas convergirem (por exemplo, a tabela de s7 deve ter uma rota para h1). Isso pode demorar um pouco!

Então, desconecte s1 de s2, por exemplo `s1.unlinkTo(s2)`.

Quanto tempo levará para a rota para s1 ser atualizada nos outros roteadores?

Sua Tarefa

Modifique `expire_routes`.

Quaisquer rotas expiradas devem ser substituídas por poison se `self.POISON_EXPIRED` for True.

Defina a entrada envenenada para ter um tempo de expiração de `api.current_time()` `+self.ROUTE_TTL`. Isso garante que a rota envenenada seja anunciada periodicamente.

Opcional, sem nota: Idealmente, os anúncios periódicos devem parar depois de um tempo (ou seja, rotas envenenadas devem ser removidas eventualmente), porque não é útil continuar anunciando a não existência de uma rota. Não testaremos você nisso.

Testando e Depurando

Para verificar seu trabalho, execute os testes unitários:

```
python3 dv_unit_tests.py 9
```

Verifique Sua Compreensão

Em relação a que as remoções de rotas agora se propagam?

Você está quase terminando! Agora pode ser um bom momento para fazer backup do seu trabalho. A última etapa requer alterar seu código até agora, e você pode querer reverter para seu código antes de começar a Etapa 10.

ETAPA 10A: Atualizações Incrementais Disparadas

Temporizadores fornecem boas garantias de convergência eventual. Mas, um comportamento ainda mais otimizado resulta de realizar ações em resposta a eventos de rede.

Nesta etapa, você implementará atualizações incrementais e disparadas. Seu roteador anunciará rotas toda vez que sua tabela for atualizada (disparado) e anunciará apenas as rotas que mudaram (incremental).

Sua Tarefa

Modifique `send_routes` para lidar com o caso onde `force=False`.

Se `force=False`, você deve anunciar uma rota pela porta apenas se a rota for diferente do que você enviou anteriormente por aquela porta, ou se você não tiver enviado anteriormente aquela rota por aquela porta.

No construtor `__init__`, você pode adicionar uma estrutura de dados `self.history`. Isso pode ajudá-lo a registrar o anúncio mais recente enviado pela cada porta para cada destino.

Dicas:

- Não copie e cole grandes quantidades de código. Não copie e cole o mesmo código repetidamente. Se seu código estiver confuso, não ajudaremos a depurá-lo.
- Métodos auxiliares podem ajudá-lo a limpar seu código. Minha solução usa dois métodos auxiliares.

Então, modifique `handle_route_advertisement` para chamar `self.send_routes(force=False)` sempre que a tabela for atualizada.

Testando e Depurando

Para verificar seu trabalho, execute os testes unitários:

```
python3 dv_unit_tests.py 10
```

Neste ponto, 28/31 dos testes da Etapa 10 devem passar, e você deve ver:
`.........FF.F..................`

ETAPA 10B: Atualizações Disparadas em Link Up/Down

A última coisa que faremos é disparar atualizações sempre que um link conectado ao seu roteador cair, ou um novo link for conectado ao seu roteador.

Sua Tarefa

1. **Implemente** `handle_link_up`.

   Esta função é chamada pelo framework quando um link conectado a este roteador sobe.

Se `self.SEND_ON_LINK_UP` for True, seu roteador deve atualizar seu novo vizinho imediatamente anunciando todas as suas rotas apenas para aquela porta. Use o argumento `single_port` em `send_routes` para ajudar.

2. **Em seguida, edite `send_routes`.**

   Se `single_port` não for None, `send_routes` deve enviar atualizações apenas para a porta única especificada, e não para outras portas.

3. **Finalmente, implemente `handle_link_down`.**

   Esta função é chamada pelo framework quando um link conectado a este roteador cai.

   Se `self.POISON_ON_LINK_DOWN` for True, esta função deve substituir todas as rotas usando aquela porta por poison. Então, anuncie imediatamente o novo poison para todos os vizinhos.

   Se `self.POISON_ON_LINK_DOWN` for False, esta função deve deletar todas as rotas usando aquela porta.

   Dica: Para remover uma entrada da tabela com chave `h`, você pode usar:

```
self.table.pop(h)
```

Testando e Depurando

Para verificar seu trabalho, execute os testes unitários:

```
python3 dv_unit_tests.py 10
```

Neste ponto, os 3 testes restantes da Etapa 10 devem passar, de modo que 31/31 testes passem.

Bug comum: Certifique-se de usar `force=False` para anunciar incrementalmente quando um link cair.

Verifique Sua Compreensão

Inicie o simulador com qualquer topologia que você goste. Você deve ver agora um tempo muito menor para responder a eventos de rede, comparado às versões anteriores do roteador.

Se agora temos eventos, qual é o propósito do temporizador?

Submissão e Avaliação

Parabéns! Você implementou um roteador que realiza uma variante do protocolo distance-vector. Sua implementação está bastante próxima do Routing Information Protocol (RIP) do mundo real.

Se você terminou completamente o projeto, você deve ver uma pontuação de 100/100 quando executar todos os testes unitários:

```
python3 dv_unit_tests.py 10
```

Para submeter o Projeto 2, submeta apenas o arquivo `dv_router.py` no SIG.

Sua nota é totalmente computada a partir dos testes unitários, dividida igualmente entre as 10 etapas (10% para cada etapa). Dentro de cada etapa, todos os testes são ponderados igualmente.

Se você completou o projeto honestamente (sem má conduta acadêmica), então a nota que você vê no autograder é a nota que você recebe.