

Universidade Tecnológica Federal do Paraná - UTFPR

Campus:
Campo Mourão

Professor:
Dr. Luiz Arthur Feitosa dos Santos

E-mail:
luizsantos@utfpr.edu.br

Sumário:

- **Rotas dinâmicas**
- **Single e multibackbone**
- **Algoritmos de roteamento**
 - **Vetor de distância**
 - **Link-State**

Atenção - este material/slides são em grande parte um apanhado dos conteúdos dos livros apresentados na Bibliografia (último slide) e que estão disponíveis na biblioteca da universidade. Tais slides não dispensam o uso dos referentes livros e de outros materiais de apoio.

Relembrando: No slide Introdução ao Roteamento (arquivo anterior), nós estávamos respondendo a uma pergunta:

Quem é que preenche ou como são preenchidas as tabelas de roteamento?

```
# route -n
```

```
Kernel IP routing table
```

| Destination | Gateway | Genmask | Flags | Metric | Ref | Use Iface |
|---------------|---------------|---------------|-------|--------|-----|-----------|
| 0.0.0.0 | 192.168.237.1 | 0.0.0.0 | UG | 600 | 0 | 0 wlan0 |
| 127.0.0.0 | 0.0.0.0 | 255.0.0.0 | U | 0 | 0 | 0 lo |
| 192.168.237.0 | 0.0.0.0 | 255.255.255.0 | U | 600 | 0 | 0 wlan0 |



As respostas para essa pergunta são:

- (i) rotas estáticas;
- (ii) rotas dinâmicas.

No slide anterior (Introdução ao Roteamento), nós focamos no roteamento estático, agora vamos nos concentrar nas rotas dinâmicas.

• Rotas Dinâmicas:

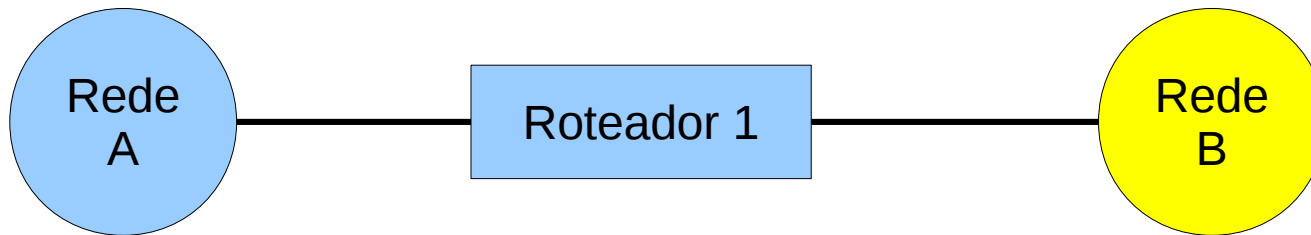
O roteamento estático funciona bem em redes pequenas. Contudo, não é muito viável em grandes redes.

Pense na quantidade de esforço e conhecimento necessário para o administrador de rede configurar uma centena de roteadores para que esses conectem corretamente centenas de redes/*hosts*.

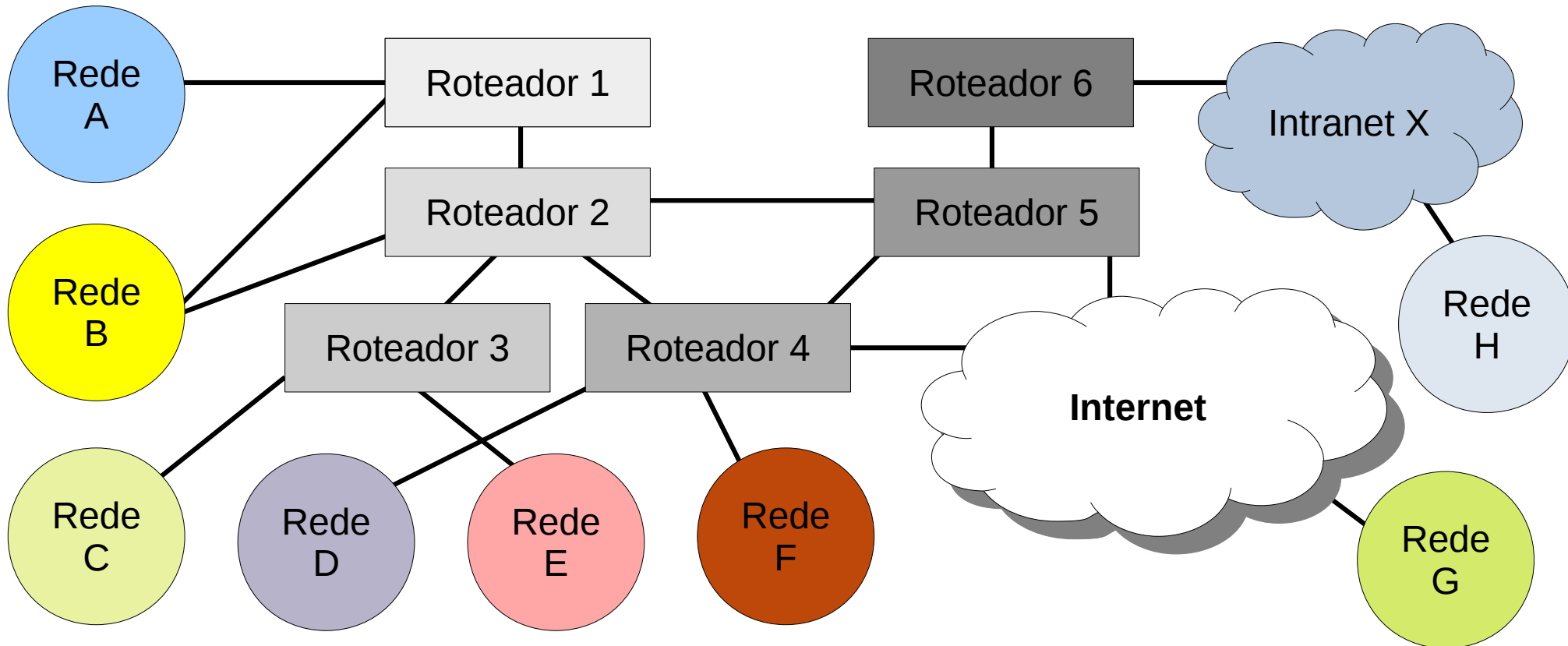
Imagine, também que essa rede é instável e que seja necessário constantes intervenções do administrador para garantir que a rede funcione corretamente!

Manter grandes redes, ou pior manter a Internet, utilizando apenas roteamento estático não é uma boa opção. É então que entra as rotas dinâmicas!!!

Nesse tipo de rede podemos usar somente roteamento estático:



Já nesse tipo de rede é melhor utilizar roteamento estático e dinâmico:



Em síntese, o roteamento dinâmico permite que roteadores troquem informações para criar/manter automaticamente entradas na tabela de roteamento.

Para que os roteadores possam trocar informações e assim automatizar as entradas das tabelas de roteamento, foram criados os protocolos para roteamento dinâmico. Tais protocolos executam várias tarefas, dentre essas: (i) **descobrir redes** e (ii) **manter a tabela de roteamento**.

Para os protocolos de roteamento, **descobrir uma rede**, significa compartilhar com outros roteadores informações a respeito de redes diretamente conectadas a esses roteadores ou redes que os roteadores descobriram por intermédio de outros roteadores. Ou seja, os roteadores vão aprendendo a respeito de outras redes, se comunicando uns com os outros.

Enquanto os roteadores vão aprendendo a respeito de novas redes, as rotas para essas redes vão sendo adicionadas ou atualizadas nas tabelas de roteamento dos roteadores que estão se comunicando neste sistema.

A rede descoberta pelo roteador, bem como o melhor caminho para essa rede, é adicionada à tabela de roteamento do roteador e compartilhada com outros roteadores.

Em geral, estabelecer rotas envolve inicialização e atualização. Cada roteador precisa estabelecer um conjunto inicial de rotas quando é inicializado (*boot* da máquina) e precisa atualizar a tabela conforme as rotas mudam (por exemplo, quando o hardware em uma determinada rede falha). É claro que a inicialização depende do sistema operacional.

Em alguns sistemas, o roteador lê uma tabela de roteamento inicial do armazenamento secundário na inicialização, mantendo-a residente na memória principal. O que normalmente configura um roteamento estático.

Outros sistemas começam deduzindo um conjunto inicial de rotas, do jogo de endereços para as redes locais às quais a máquina está conectada, e depois contatando uma máquina vizinha para requisitar rotas adicionais. Que representa o roteamento dinâmico.

Uma vez que uma tabela de roteamento inicial tenha sido construída, um roteador precisa acomodar mudanças nas rotas. Em redes (interredes/*internets*) pequenas e que mudam lentamente, os administradores podem estabelecer e modificar rotas manualmente (roteamento estático). Contudo, em ambientes grandes e que mudam de forma rápida os métodos automatizados são mais indicados (roteamento dinâmico).

OK, então os roteadores vão aprendendo a respeito das redes, uns com os outros, até ter a rota para todas as redes?



A resposta para a pergunta anterior é **NÃO!**

Ter informações a respeito de todos os destinos/redes possíveis em todos os roteadores é impraticável em grandes redes (ex. Internet), pois exige propagar grandes volumes de informações sempre que uma mudança ocorrer ou sempre que administradores precisarem verificar a consistência.

Outra solução, poderia ser ter uma máquina/roteador central que tenha rotas para todas as redes! Infelizmente esse método de interseção central também falha, pois nenhuma máquina é rápida o suficiente para servir como uma chave central através da qual todo o tráfego da Internet passa.

Assim, a solução normalmente empregada em roteadores é permitir que grupos gerenciem roteadores locais de forma autônoma, acrescentando novas interconexões de rede e rotas sem mudar roteadores distantes.

Por exemplo, imagine que metade das cidades se encontrem no sul do país e outra na parte Norte. Suponha que uma única ponte atravessa o rio que separa norte do sul. Considere que as pessoas moram no sul não gostam das que moram no norte e, por tanto, permitem apenas placas rodoviárias que mostrem destinos no sul, mas não no norte - o mesmo ocorre no norte. Assim, o roteamento será consistente se toda placa rodoviária no leste mostrar explicitamente todos os destinos do sul e apontar o caminho padrão para a ponte e o mesmo acontecer no norte.

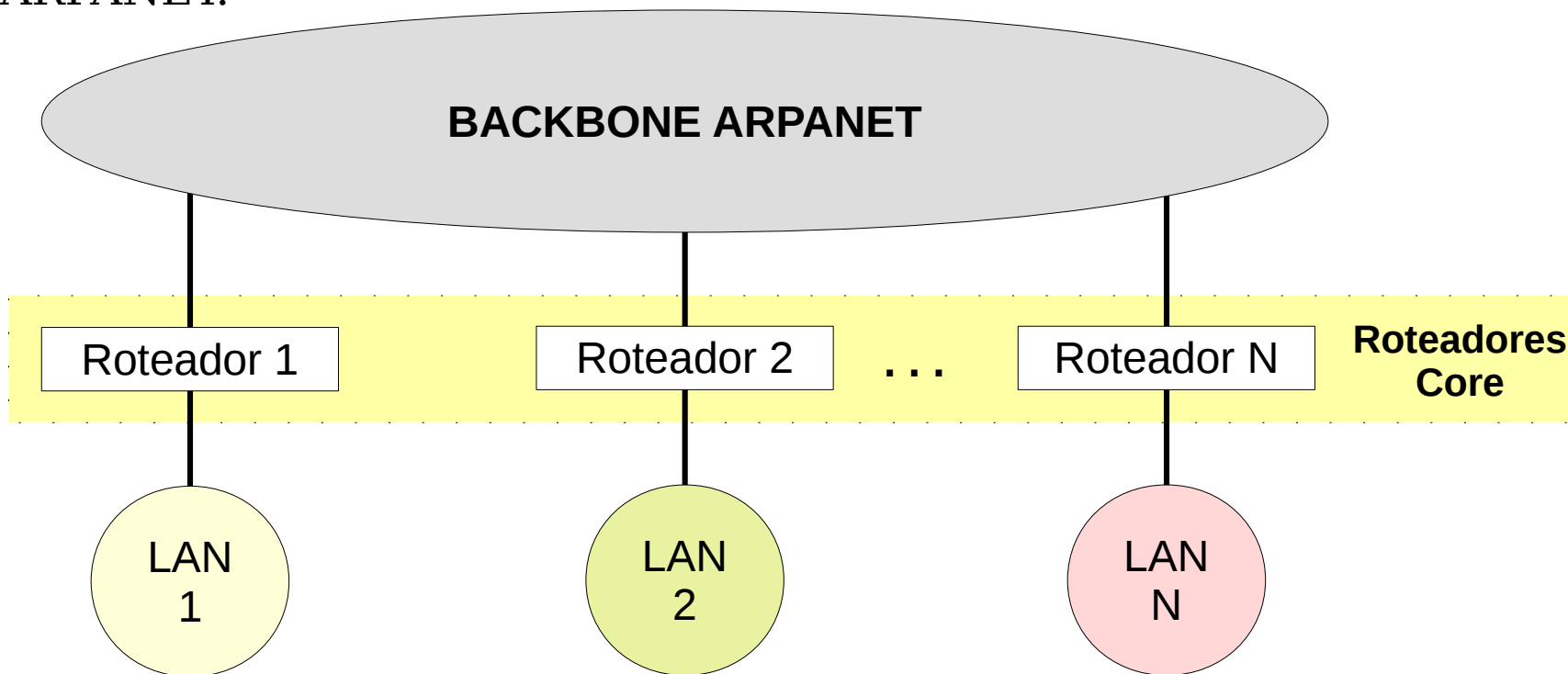
Resumindo o exemplo anterior, os roteadores sabem detalhes a respeito de como entregar pacotes para redes que estão “perto” deles (fazem parte do mesmo sistema, mesma organização, etc). Caso chegue um pacote que ele não saiba o destino ele deve enviar para uma rota padrão, que leva a roteadores externos que seguem basicamente a mesma lógica até chegar ao destino.

Para um melhor entendimento vamos ver um pouco de história

Quando o TCP/IP foi desenvolvido, a ARPANET servia como *backbone* da Internet. Durante os testes iniciais, cada site/rede gerenciava tabelas de roteamento e instalava rotas para outros destinos manualmente. À medida que a Internet foi surgindo, tornou-se claro que manter isso manualmente seria inviável, e foram necessários mecanismos automatizados.

Os projetistas da Internet selecionaram uma arquitetura de roteadores que consistia de um pequeno conjunto central de roteadores que mantinham informações parciais. Em nossa analogia com placas de trânsito, seria como designar um pequeno conjunto de intersecções localizadas centralmente a ter placas que mostrem todos os destinos e permitir que as intersecções distantes mostrem apenas destinos locais. Como a rota padrão em cada intersecção distante aponta para uma das intersecções centrais, os viajantes cedo ou tarde alcançarão seu destino.

A arquitetura da Internet inicial é fácil de entender, pois uma grande motivação para o sistema de roteador core veio do desejo de conectar redes locais à ARPANET.



Os *hosts* nas redes locais passam todo o tráfego não local para o roteador core mais próximo. Tais roteadores não podiam usar informações parciais (rota padrão), pois caso contrário todos os *datagramas* para os quais o roteador não tem nenhuma rota seguem o mesmo caminho padrão, seja qual for o seu destino. Tal arquitetura exige que todos os sites locais coordenem suas rotas padrão. Mais importante, mesmo se as rotas padrão forem coordenadas, o encaminhamento é ineficiente porque um *datagrama* pode atravessar todos os n roteadores ao viajar da origem ao destino.

Então inicialmente a Internet evitava rotas padrão. Os projetistas arranjavam roteadores para trocar informações de roteamento de modo que cada um tivesse rotas completas. Fazer isso era fácil, uma vez que todos os roteadores se conectavam a uma única rede de *backbone*.

Assim, o conjunto central de roteadores tinha informações completas e eram chamados de Núcleo (core) da Internet ou zona *default-free*.

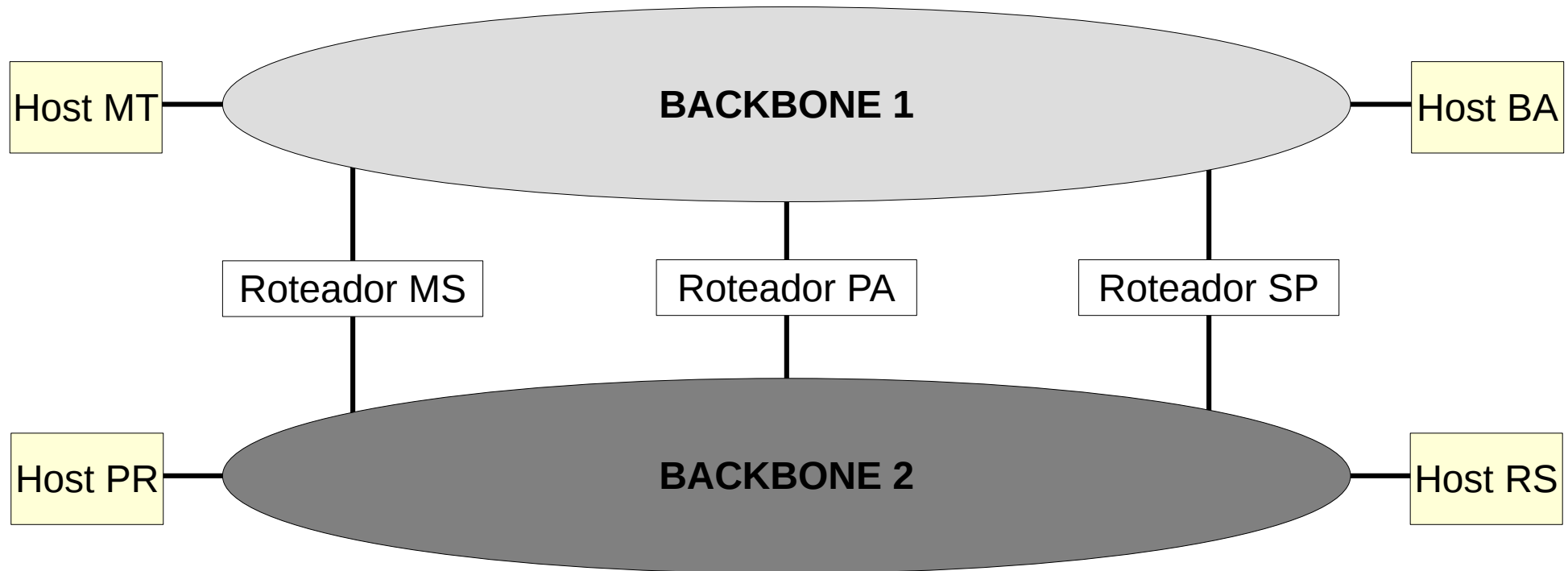
A vantagem da arquitetura de roteamento core é que, como roteadores não pertencentes ao core usam informações parciais, um site distante tem autonomia para fazer mudanças de roteamento locais. A desvantagem é que um site pode apresentar inconsistência que tornam alguns destinos inalcançáveis.

As inconsistências entre tabelas de roteamento normalmente surgem de erros nos algoritmos que calculam tabelas de roteamento, de dados incorretos fornecidos a esses algoritmos ou de erros que ocorrem durante a transmissão dos resultados de outros roteadores.

É claro que os projetistas procuram meios de diminuir os impactos dos erros, para manter as rotas consistentes o tempo todo. Se as rotas se tornarem inconsistentes, os protocolos de roteamento devem ser robustos o bastante para detectar e corrigir erros de forma rápida. Os protocolos devem ser projetados para restringir o efeito dos erros.

Saindo do roteamento da ARPANET para *backbones peer*

Com o tempo a Internet evoluiu de um único *backbone* (ARPANET) para o *backbone* NFSNET, o que acrescentou uma nova complexidade à estrutura de roteamento e obrigou os projetistas a inventarem uma nova arquitetura de roteamento.



Basicamente, a Internet evoluiu de um único *backbone* central para um conjunto de redes de *backbones* ou simplesmente *peers*.

Para entender as dificuldades do roteamento agora suponha que a figura anterior, também representa orientação geográfica e o nome dos *hosts* fazem menção a estados do Brasil.

Ao estabelecer rotas entre os *hosts* PR e BA, é necessário decidir qual o melhor caminho/rota:

- i. Ir do Host PR para BA, indo pelo roteador MS e então através do Backbone 1;
- ii. Ou ir do Host PR através do Backbone 2, por meio do Roteador SP, depois através do Backbone 1. Essa rota pode ou não ser aconselhável, dependendo das políticas para uso da rede e da capacidade de vários roteadores e *backbones*.

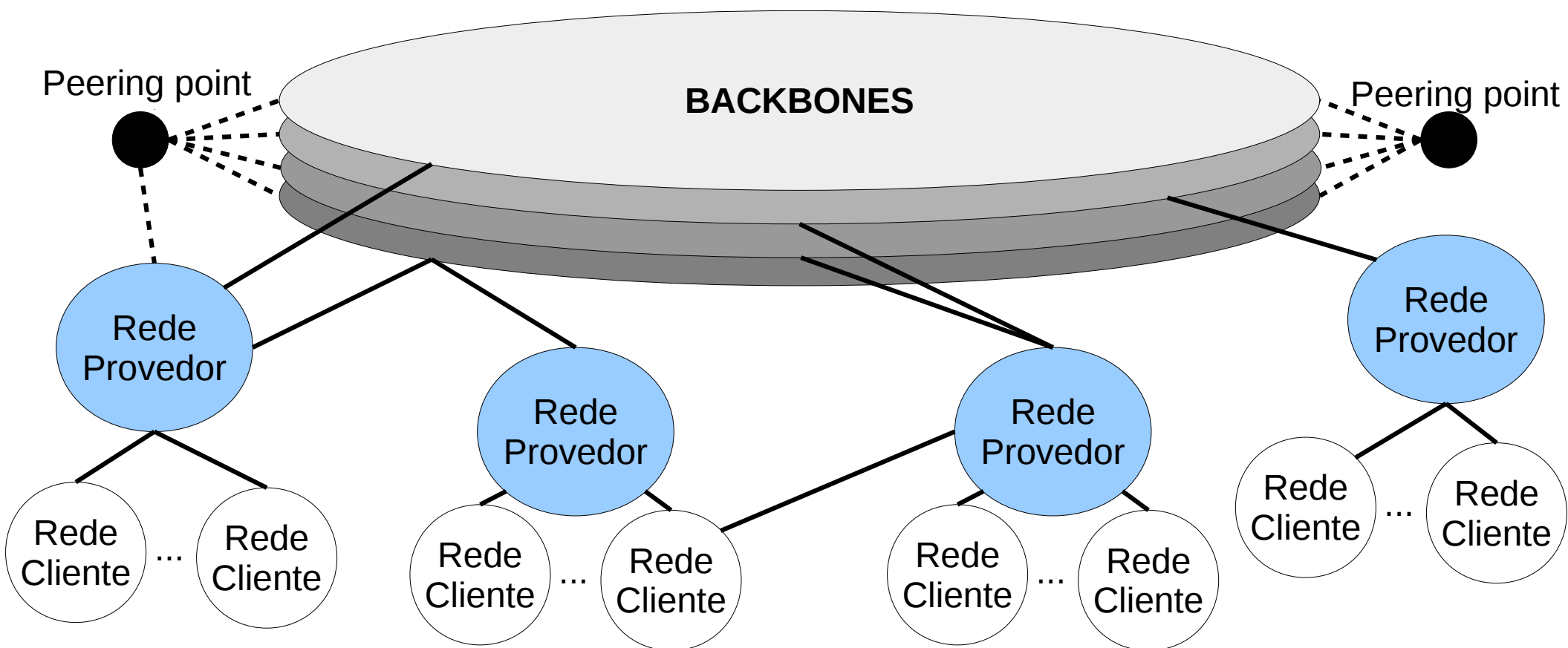
Para a maioria das configurações de *backbone*, o tráfego entre um par de *hosts* geograficamente próximos deve tomar um caminho mais curto, independente das rotas escolhidas para o tráfego que corta o país. Por exemplo, o tráfego entre os *hosts* PR e MT deve fluir através do roteador MS, pois ele minimiza a distância em ambos os *backbones*.

Todas as declarações parecem simples, mas são complexas de implementar por duas razões: (i) embora o algoritmo de roteamento use a parte da rede do IP para escolher uma rota, o roteamento ótimo em uma arquitetura de *backbone peer* requer rotas individuais para *hosts* individuais (o caminho para um *host* não é o mesmo para outro).

(ii) os gerentes dos *backbones* precisam concordar em manter as rotas consistentes entre todos os roteadores ou *loops* de encaminhamento podem se desenvolver.

Internet hoje

A Internet mudou de uma estrutura *single backbone* para uma estrutura *multi-backbone*, que é mantida hoje por diferentes empresas. Embora seja muito difícil representar a Internet, podemos imaginá-la assim:



Na figura do slide anterior, vários *backbones* são representados por empresas de comunicações que fornecem conectividade em nível global/mundial. Esses *backbones* são conectados por algum *peering point*, que fornecem conectividade entre os *backbones*. Em um nível mais baixo, residem os provedores de rede (*provider networks*) que utilizam os *backbones* de conectividade global para prover serviço de Internet aos seus consumidores. Finalmente, no nível mais baixo existem as redes clientes (*customer networks*), que usam os serviços fornecidos pelos provedores de rede.

Qualquer uma das três entidades citadas anteriormente (*backbone*, provedor de rede e rede cliente) podem ser chamadas de ISP (*Internet Service Provider*), pois fornecem serviços, só que em níveis diferentes.

Roteamento Hierárquico

A Internet hoje é composta de um grande número de redes e roteadores para conectá-las. Assim, é impossível que o roteamento na Internet seja feito utilizando apenas um único protocolo, por duas razões:

- Problemas de escalabilidade;
- Questões administrativas.

O **problema de escalabilidade** surge, pois a Internet é tão grande que haveria problemas quanto ao tamanho da tabela de roteamento, o processamento necessário para buscar um destino nessa tabela e atualizar tal tabela também ocuparia uma grande fatia da rede.

As **questões administrativas** está ligado a problemas de política e negócios entre as empresas que mantêm as partes da rede (*backbone* global, rede do provedor e rede do cliente). Cada ISP pode ser gerenciado por uma autoridade administrativa e tais entidades precisam ter controle sobre seus sistemas. As organizações/empresas devem ser capazes de usar várias subredes e roteadores, conforme sua necessidade, e provavelmente vão desejar que os roteadores sejam de um determinado fabricante, bem como executar algoritmos de roteamento que melhor os atendam. Por fim, as organizações podem querer impor algumas políticas sobre o tráfego que passa pelo ISP.

Desta forma, roteamento hierárquico significa considerar cada ISP como um **Autonomous System (AS)** – sistema autônomo. **Cada AS pode executar um protocolo de roteamento diferente**, conforme suas necessidades, mas a **Internet global deve executar um único protocolo** de roteamento global **que permita a comunicação e interligação de todos ASs**.

Os **protocolos de roteamento** executados **dentro dos AS** são conhecidos como:

- *Intra-AS Routing Protocol* – protocolo de roteamento Intra-AS;
- *Intradomain Routing Protocol* – protocolo de roteamento intradomínio;
- Ou *Interior Gateway Protocol* (IGP).

O **protocolo de roteamento global** é conhecido como:

- *Inter-AS routing protocol* – protocolo de roteamento Inter-AS;
- *Interdomain routing protocol* – protocolo de roteamento interdomínio;
- Ou *Exterior Gateway Protocol* (EGP).

Novamente, podemos ter diversos protocolos de roteamento Intra-AS (cada AS é livre para escolher um), mas deve haver apenas um único protocolo de roteamento Inter-AS que trata o roteamento entre as entidades.

Atualmente, os dois **protocolos intradominíos mais utilizados** são:

- RIP;
- OSPF.

Todavia, **o único protocolo de roteamento interdomínio é o BGP**.

Veremos tais protocolos em detalhes logo mais a frente!

Um pouco mais de AS (*Autonomous Systems*)

Como já foi dito, cada ISP é um AS que gerencia sua rede conforme sua vontade.

A cada AS é dado um número de identificação, chamado de ASN (*Autonomous Systems Number*), tal número é fornecido pela ICANN (*Internet Corporation for Assigned Names and Numbers*) – O ICANN coordena além do ASN, os endereços IPs e nomes DNS. O ASN é um número inteiro de 16bits que identifica um AS.

Embora existam ASs de pequeno, médio e grande porte, os ASs, não são categorizados pelo seu tamanho, mas sim de acordo com a maneira que se conectam com outros ASs, sendo:

- ***Stub AS***: Um *Stub AS* tem apenas uma conexão com outro AS. O tráfego de rede pode ser iniciado ou terminado em um *Stub AS*. Nos ASs do tipo *stub*, os dados não podem passar (ser roteados) através desses ASs, já que eles são a ponta (*stub*) inicial ou final da transmissão. As redes clientes são um bom exemplo de *Stub AS*.

- ***Multihomed AS***: Um AS *multihomed* pode ter conexão com mais de um AS, mas esse não permite que o tráfego de rede seja roteado, desse AS para outro. Um bom exemplo de *multihomed* AS é uma rede cliente que está conectada a mais de um provedor, mas sua política não permite enviar pacotes de um provedor para outro.
- ***Transient AS***: Um AS *transient* está conectado a mais que um AS e pode rotear pacotes entre esses ASs. As redes provedoras (*provider networks*) e os *backbones* são bons exemplos de *transient* ASs.

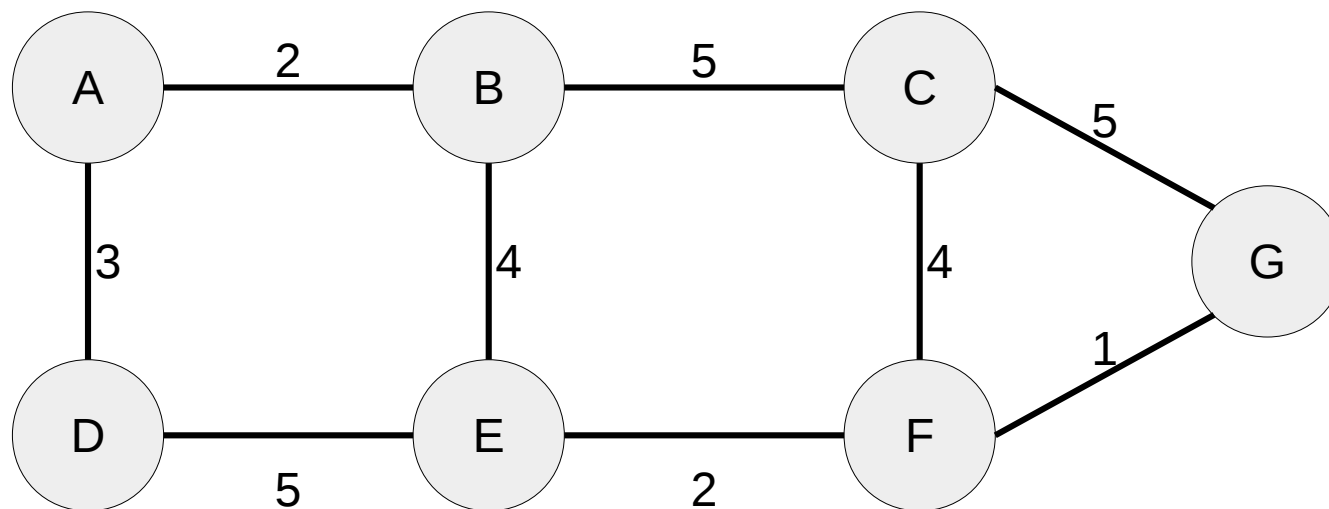
Algoritmos de roteamento

Antes de estudar na prática os protocolos de roteamento, vamos ver a teoria que está por trás desses protocolos, ou seja, vamos ver os algoritmos que servem de base para tais protocolos.

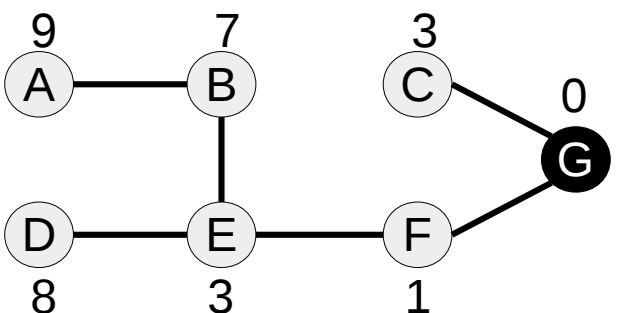
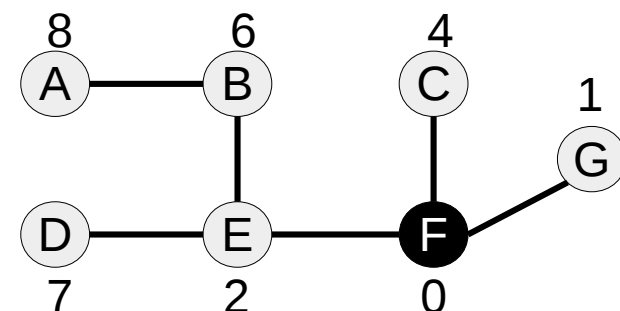
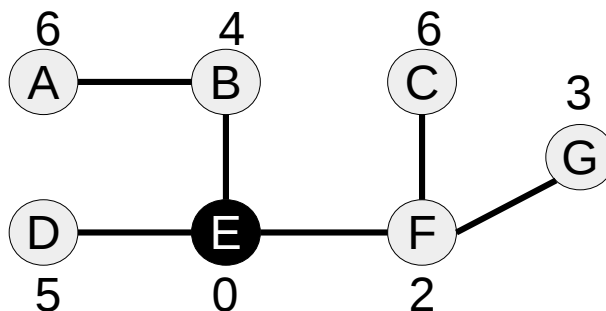
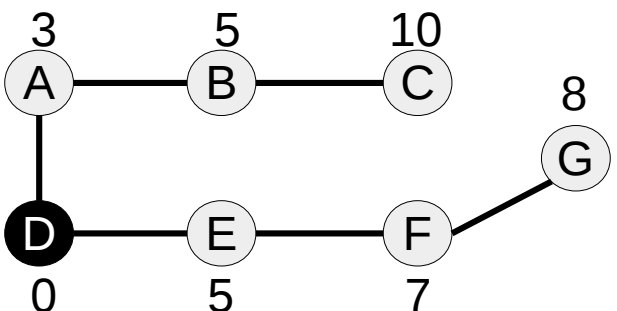
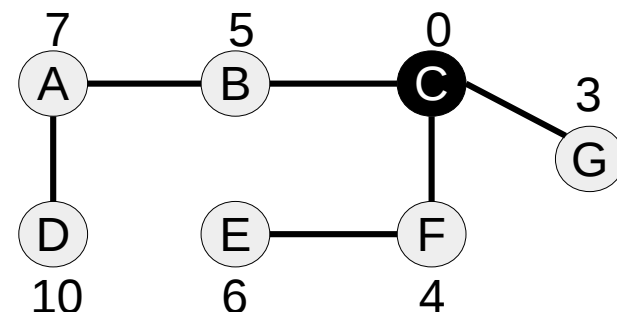
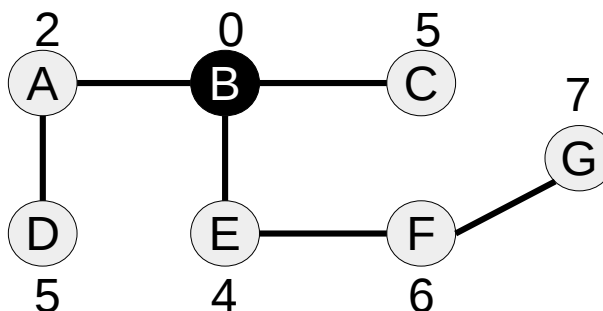
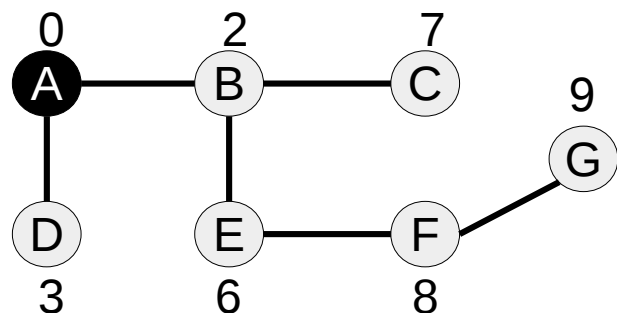
Inicialmente temos que saber, que apesar da Internet ser complexa (como demonstrado anteriormente), essa, bem como qualquer rede, pode ser representada como um grafo.

Um grafo em computação é representado por nós (*nodes*) e arestas (*edges* - linhas), sendo que as arestas podem conectar os nós. Assim, para mapear redes em grafos, os *hosts* (neste caso os roteadores) são representados pelos nós. E os links de rede (cabos de rede, ou mesmo ligação sem fio entre os hosts) são as arestas.

De fato a Internet é um grafo ponderado, no qual cada aresta está associada a um custo. Tal custo/peso pode ser usado para representar a distância entre *hosts*, ou outras diferentes representações, tais como velocidade do *link*, largura de banda, etc. Mas por hora vamos imaginar que é um custo qualquer. Se não há arestas entre os nós, o custo é infinito.



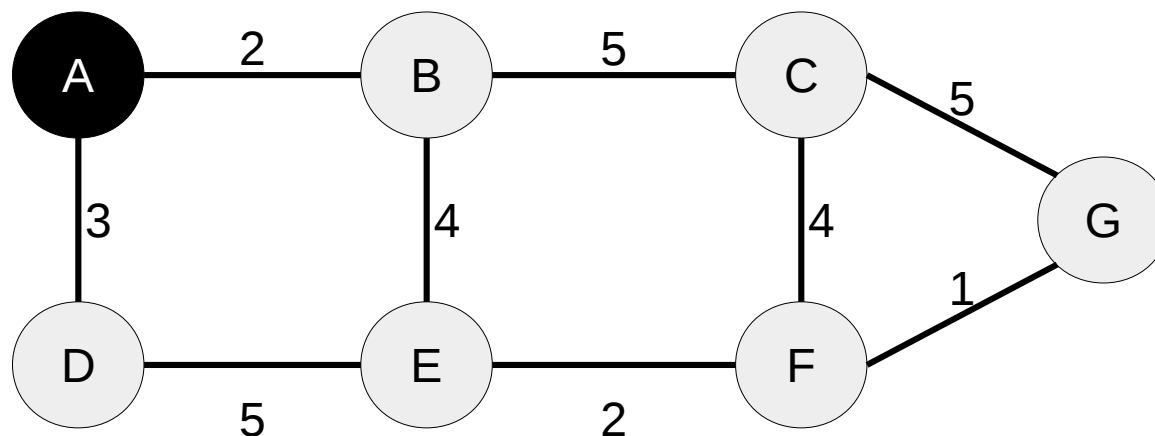
Se há N roteadores interconectando as redes, haverá (N-1) caminhos de menor custo entre um roteador em questão e os demais roteadores. Isso significa que é necessário $N * (N-1)$ caminhos de menor custo para toda a rede em questão. Se há 10 roteadores interligando redes, haverá 90 caminhos de menor custo entre esses. Uma maneira de ver isso é através de uma árvore de menor custo.



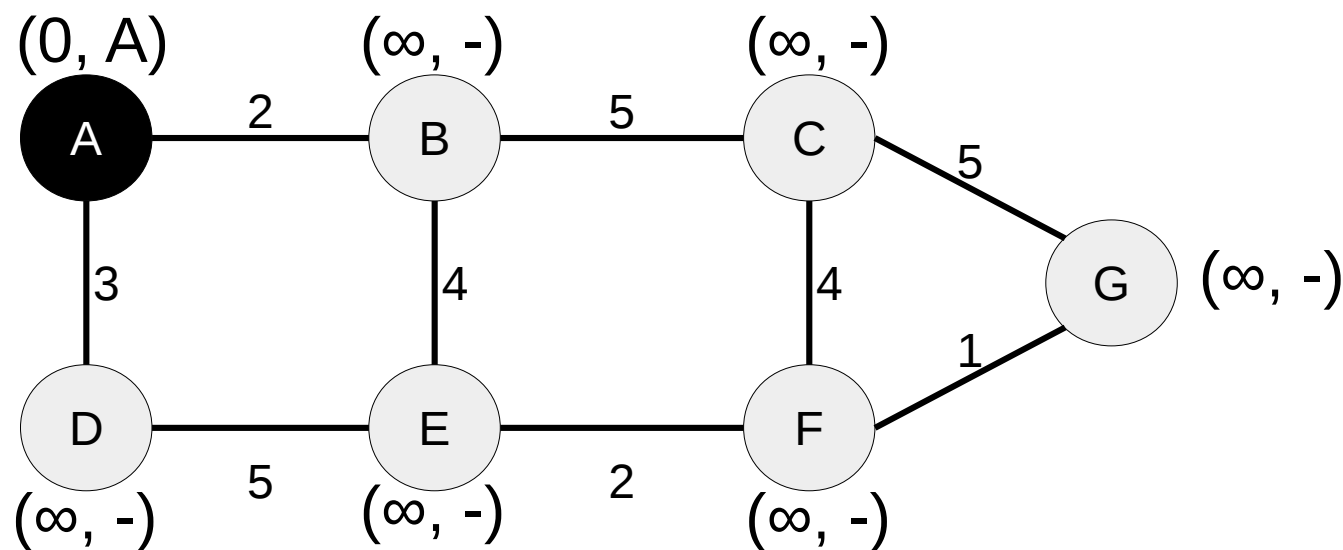
Há várias formas de fazer a árvore de menor custo, tal como o Algoritmo de Dijkstra de 1959. Neste algoritmo cada nó é rotulado com a distância da origem até o melhor caminho conhecido. A distância deve ser positiva (não negativa), isso é possível utilizando medidas reais como largura de banda ou atraso.

No algoritmo de Dijkstra, inicialmente, os caminhos não são conhecidos, então todos os nós são rotulados com infinito (que representa inalcançáveis). Assim, que o algoritmo é executado os caminhos são encontrados, os rótulos podem mudar para refletir o melhor caminho do nó raiz até os outros nós do grafo. Os rótulos podem ser tentativas ou permanentes, sendo que inicialmente são tentativas de encontrar o melhor valor. Depois, quando se descobre que um dado valor é o melhor, esse se torna permanente.

Então, para encontrar a árvore de menor custo para A usando Dijkstra o processo seria:

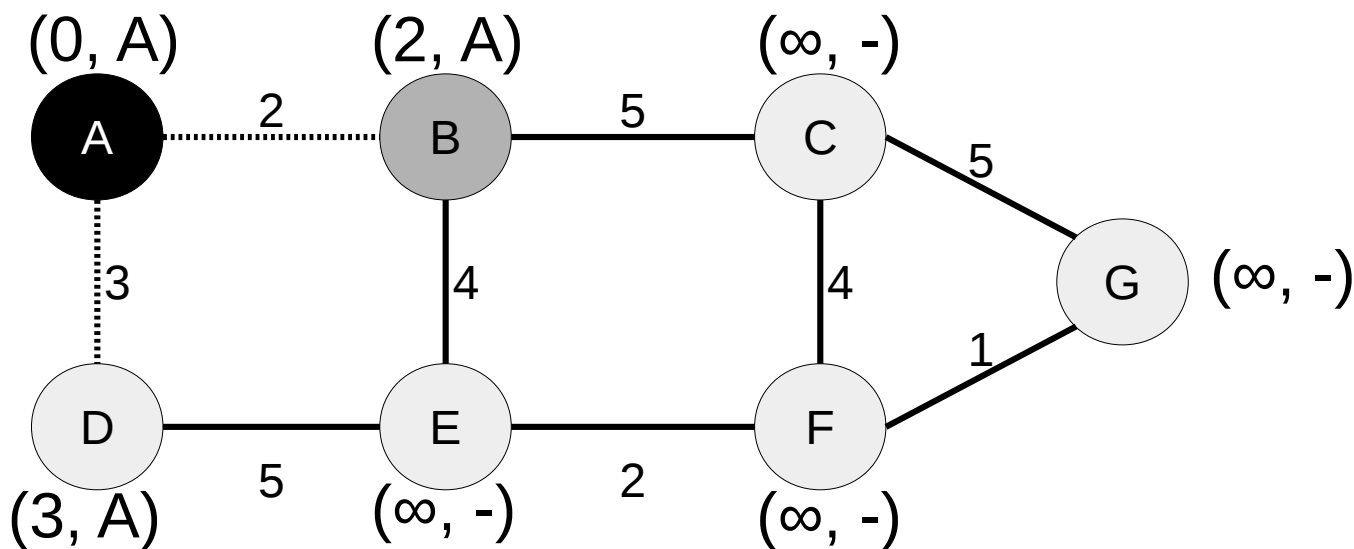


O primeiro passo é: iniciar os valores, sendo que o nó raiz receberá o valor/rótulo 0 e terá como nó de referencia ele mesmo. Os demais nós de inicio são inalcançáveis, pois isso recebem o valor infinito e não possuem o nó de onde eles receberam esse valor, isso é representado por $(\infty, -)$.



O próximo passo é: a partir do nó raiz atualizar os valores dos nós que esse pode alcançar (representados pela linha pontilhada).

Neste caso, a partir do A chegamos em B e D. Que respectivamente recebem o valor 2 ($0+2$), sendo que 0 é o valor de A e 2 é o valor da aresta para chegar em B. O nó D recebe 3 ($0+3$).

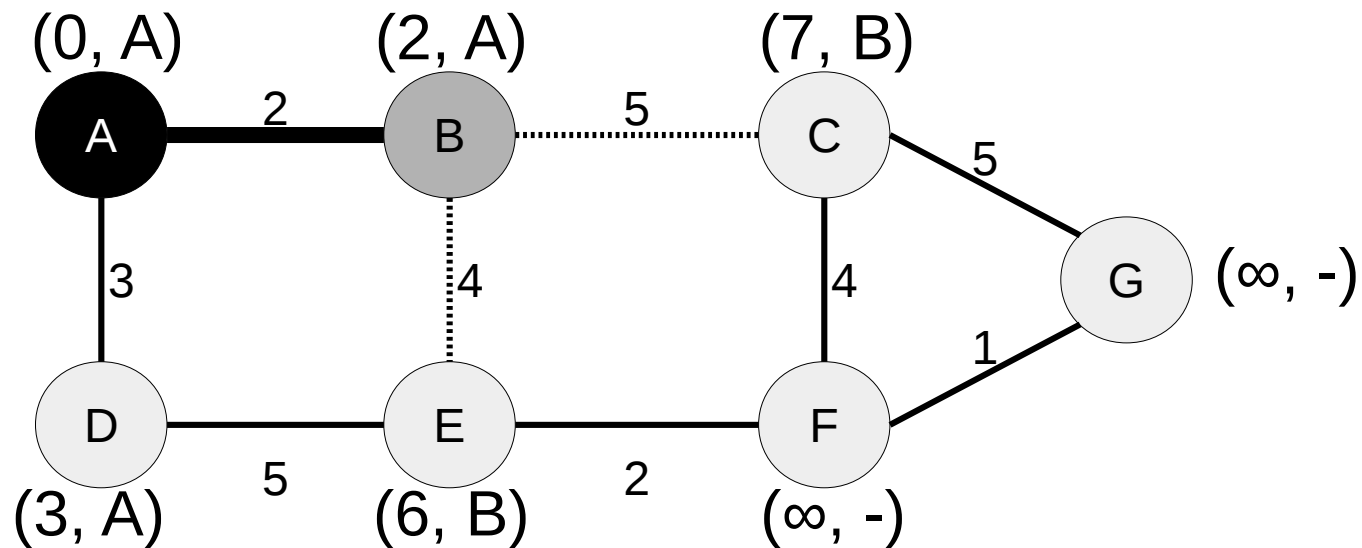


Tanto B quanto D recebem A como referencia, pois foi partir desse nó somado ao valor da aresta, que foi gerado o valor do rótulo.

Por fim, escolhe-se o nó de menor valor de rótulo e reinicia o processo a partir dele. Neste caso é o nó B

Agora a partir do nó B, que tem o menor valor, observa-se quais nós é possível alcançar e atualiza os rótulos desses.

No caso será o nó C e E, que receberão os valores 6 ($2+4$) e 7 ($2+5$), respectivamente.

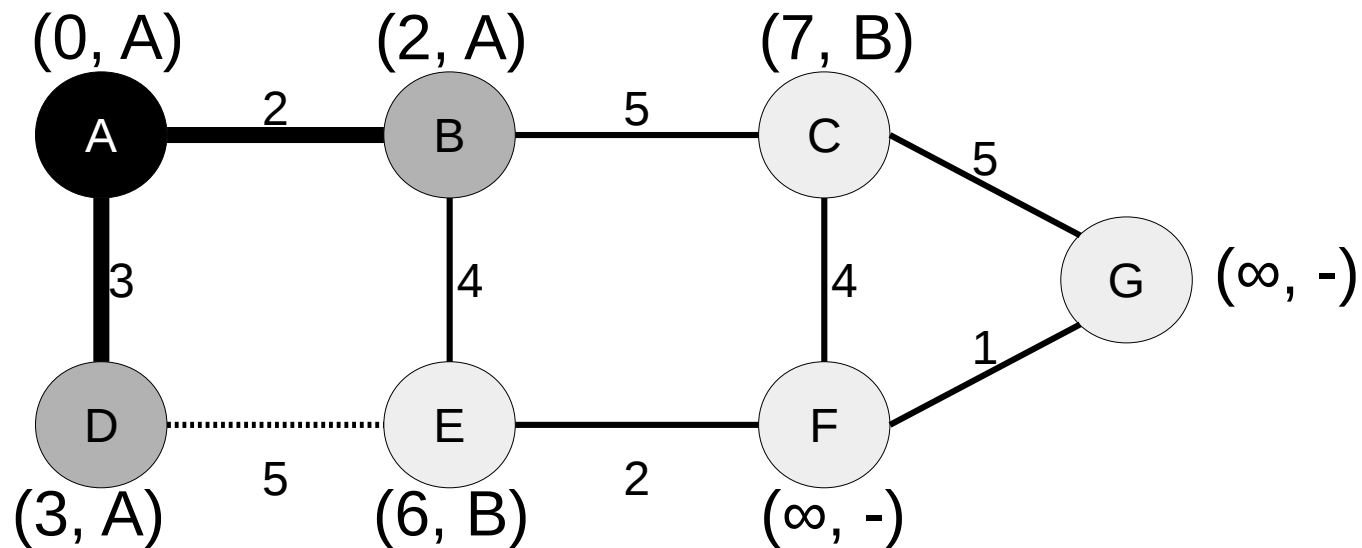


Agora deve-se fixar o menor valor, que no caso não é nem C, nem E, mas sim D!

Reinicie a busca a partir de D.

A partir do nó D, é possível chegar em E com o valor de 8 (3+5), contudo E, já possui um valor menor, então não fazemos nada.

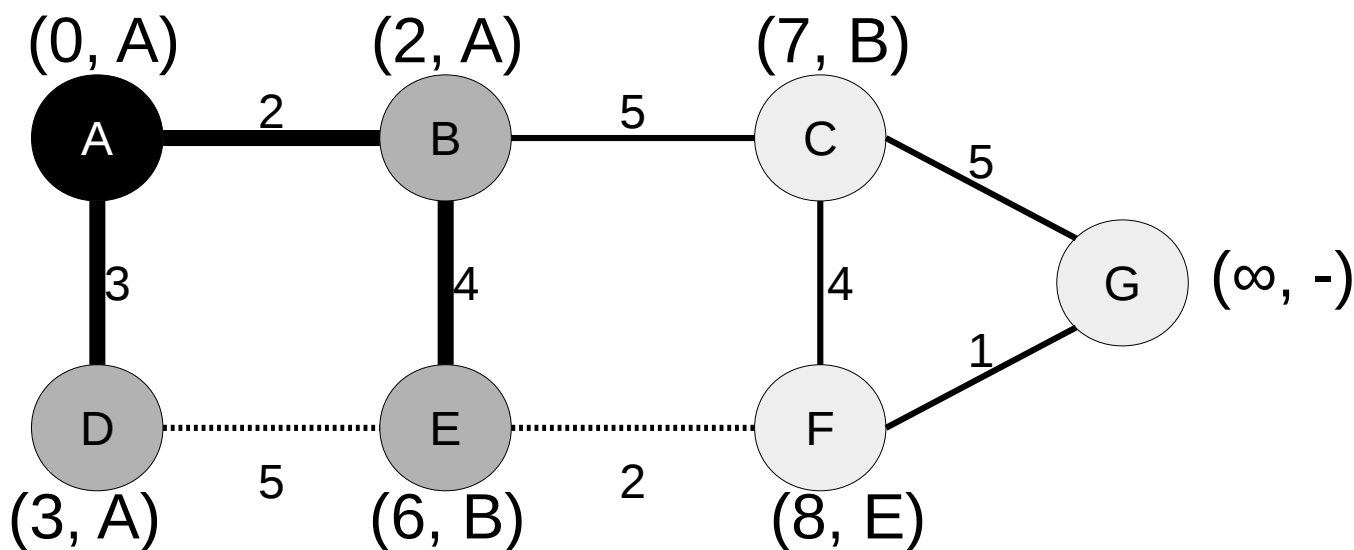
Também é possível voltar para A, que daria 3, mas A já tem o valor 0. Desta forma fica tudo como está! Normalmente o valor do nó raiz nunca é alterado.



Sem, nada a fazer voltamos a escolher outro nó de menor valor que é o E.

Analisando o nó E, podemos chegar em D com 11, mas 11 é maior que 3, assim não fazemos nada.

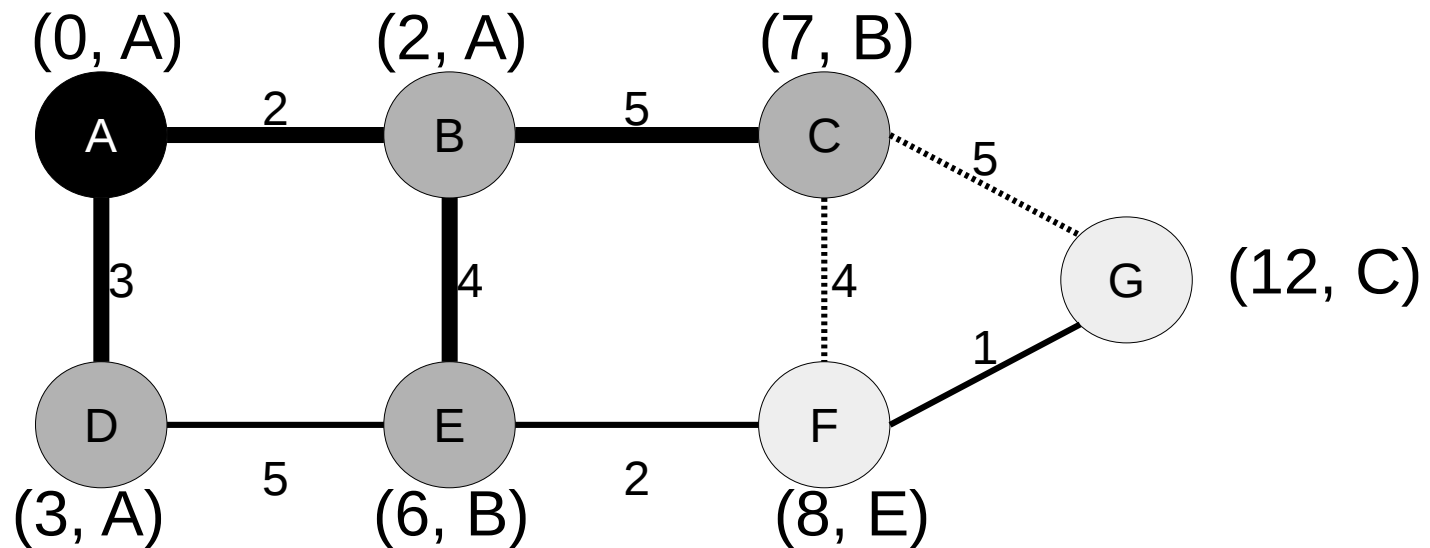
Também, não atualizamos B, pois o valor é maior e o nó de referencia de E é B!



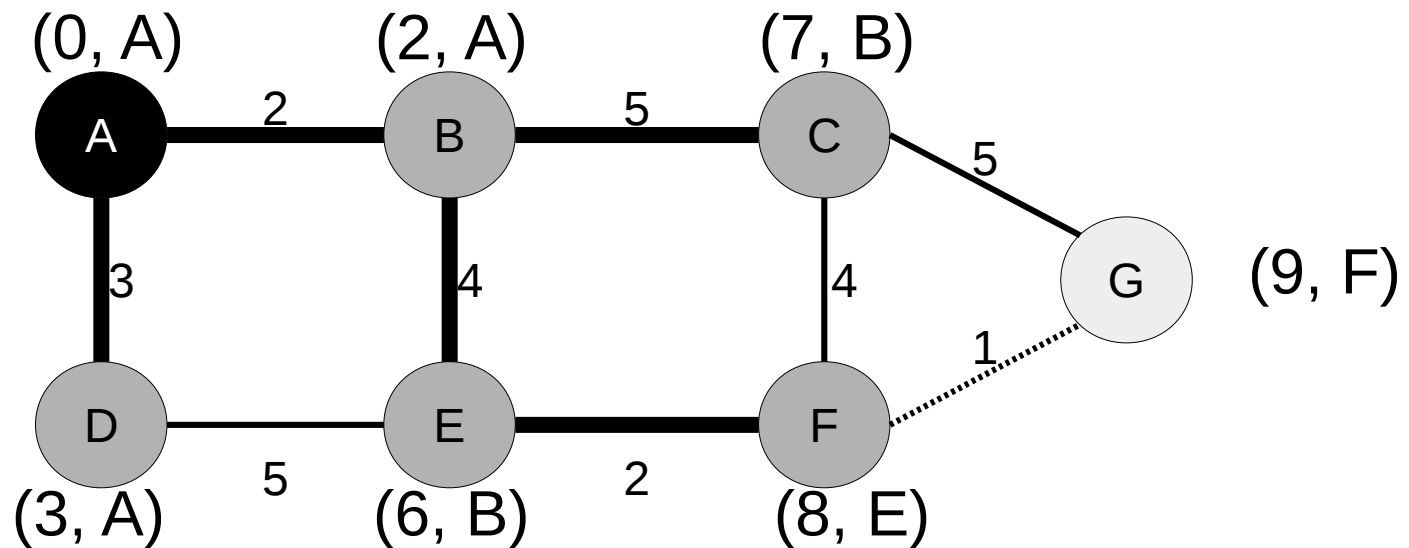
Por fim, atualizamos o valor de F para (8,E), ou seja, $6+2$, sendo que conseguimos esse valor somando o rótulo de E com a aresta até F.

Agora o menor valor é do C, com 7. Dali dá para finalmente chegar no G.

Também, chega-se no F, mas o valor obtido é maior (11).

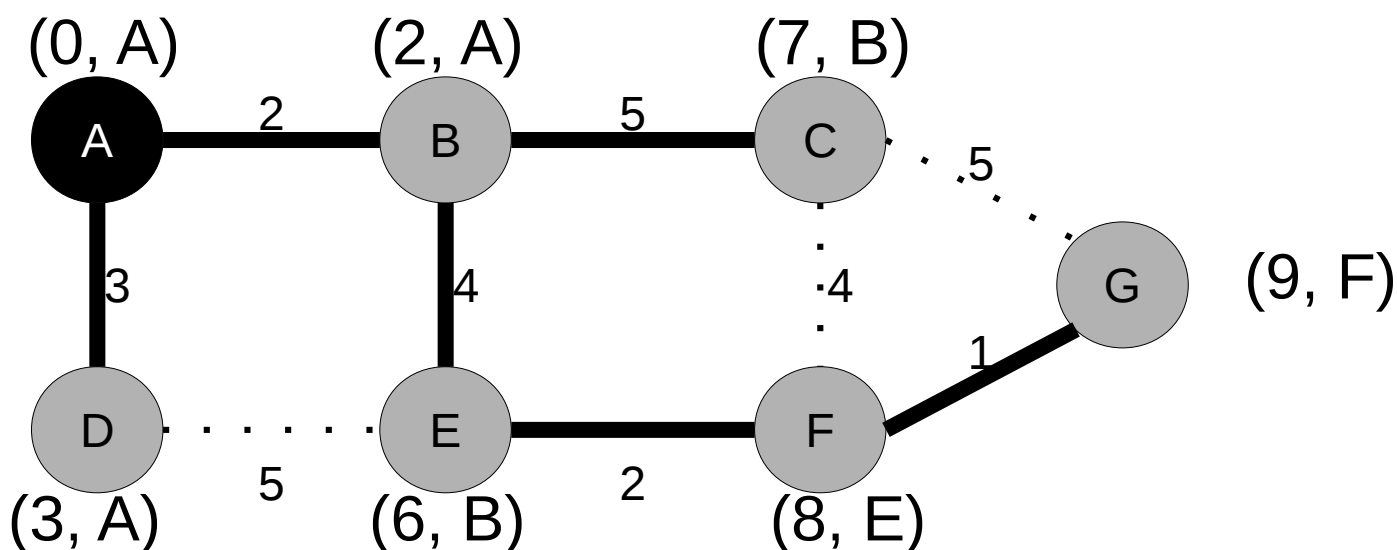


O valor da vez agora é 8 (no nó F a partir de E). Deste ponto conseguimos chegar em G novamente e atualizar o valor para 9 ($8+1$), desta forma também atualizamos o nó de referencia para o valor F (atualizamos de C para F).



Agora é só travar o último nó (G) e a árvore de menor custo está pronta pelo método de Dijkstra. As linhas escuras é o caminho mais curto e a pontilhada é só para lembrar as ligações do grafo, essas não fazem parte da árvore de menor custo.

Poderíamos continuar e fazer uma árvore de menor custo considerado cada nó como raiz, tal como foi apresentado no slide 22.

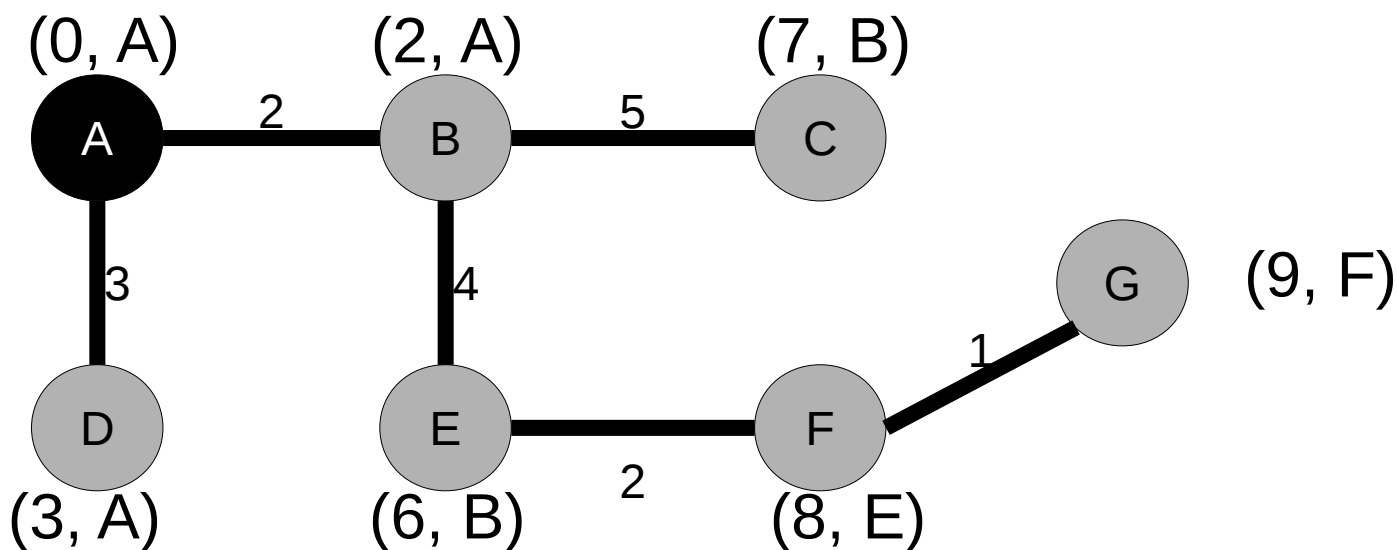


Terminando o processo da árvore usando Dijkstra surge uma dúvida:

Como saber o menor caminho da raiz até o ponto B ou até o ponto G?

Para saber o caminho de um ponto até a raiz basta seguir o valor de referencia, ou seja, se inicia do destino para a origem.

Exemplo, para se saber o caminho de E, verificamos que o valor de E veio de B (6,B). Então, vamos verificar o valor de referencia de B, que é A (2,A), desta forma chegamos na raiz que é A.



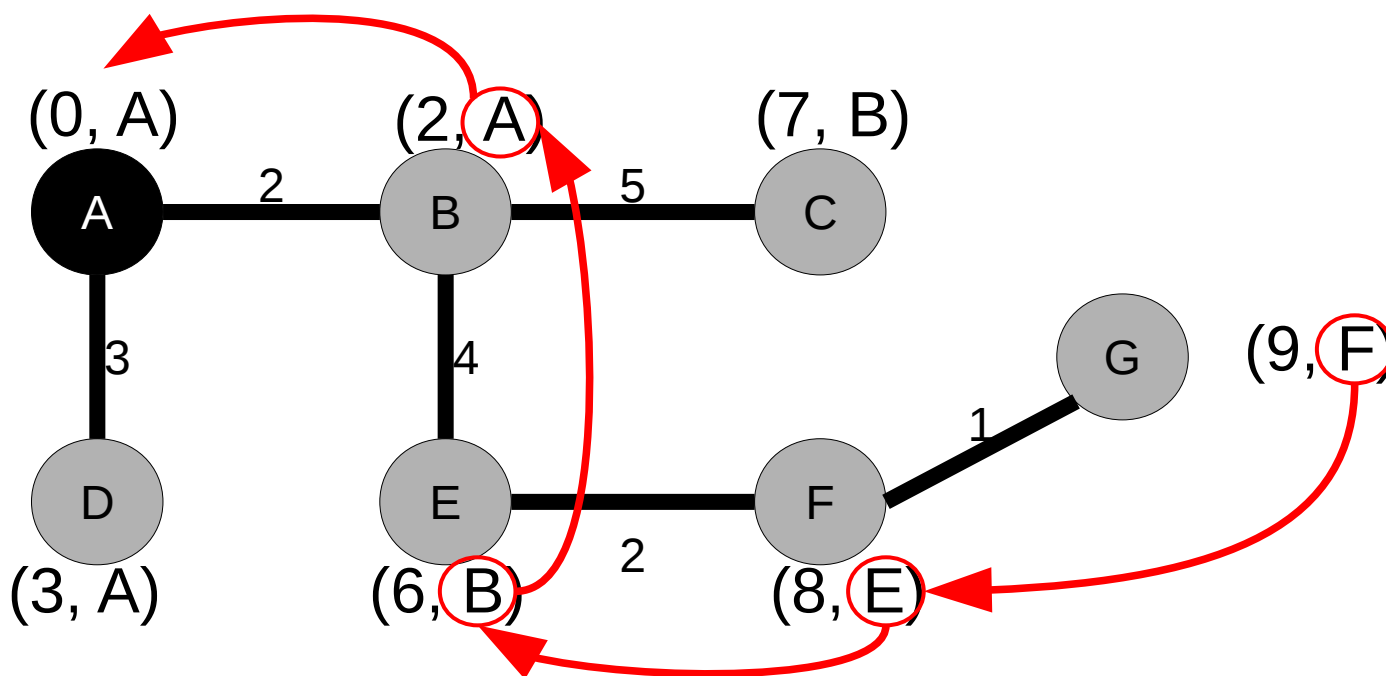
Da mesma forma para ir de G até A (raiz), o processo seria:

G(9,F), então vou olhar F, que é (8,**E**), então vou olhar E (6,**B**), então vou olhar B (2,**A**), então cheguei em A e o caminho é: A → B → E → F → G.

Desta maneira um roteador poderia descobrir o menor caminho para enviar pacotes entre os pontos da rede.

Para saber o caminho de um ponto até a raiz basta seguir o valor de referencia, ou seja, se inicia do destino para a origem.

Exemplo, para se saber o caminho de E, verificamos que o valor de E veio de B (6,B). Então, vamos verificar o valor de referencia de B, que é A (2,A), desta forma chegamos na raiz que é A.



Da mesma forma para ir de G até A (raiz), o processo seria:

G(9,F), então vou olhar F, que é (8,**E**), então vou olhar E (6,**B**), então vou olhar B (2,**A**), então cheguei em A e o caminho é: A → B → E → F → G.

Desta maneira um roteador poderia descobrir o menor caminho para enviar pacotes entre os pontos da rede.

Há muitos algoritmos de roteamento. A diferença entre esses são a maneira de interpretar o menor custo e as formas de criar a árvore de menor custo para cada nó. A seguir serão apresentados os algoritmos mais utilizados.

Algoritmo de roteamento por vetor de distância (Bellman-Ford)

O algoritmo de roteamento por vetor de distância (*distance-vector*) é fundamentalmente um algoritmo que visa encontrar o melhor caminho, tal como discutido anteriormente.

O algoritmo de vetor de distância, trabalha com cada roteador mantendo uma tabela (vetor) que dá a melhor distância conhecida para cada destino e qual link usar para chegar lá. Tais tabelas são atualizadas utilizando trocas de informações entre roteadores vizinhos.

Assim, nesse algoritmo, primeiramente cada nó cria sua própria árvore de menor custo, com informações rudimentares repassadas por seus roteadores vizinhos.

Árvores incompletas são trocadas, de tempos em tempos, com os roteadores vizinhos, para que as árvores sejam cada vez, mais e mais, completas e representem da melhor forma possível toda a estrutura de inter-rede. Ou seja, os roteadores ficam falando tudo a respeito de seus vizinhos/redes que conhecem, mesmo que isso seja incompleto.

Há duas coisas importantes no algoritmo de vetor de distância: a (i) **equação Bellman-Ford** e (ii) o **conceito de vetor de distância**.

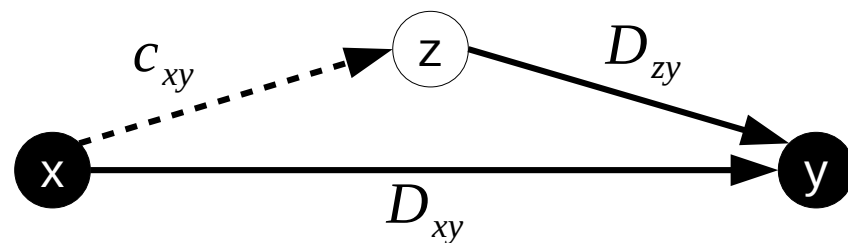
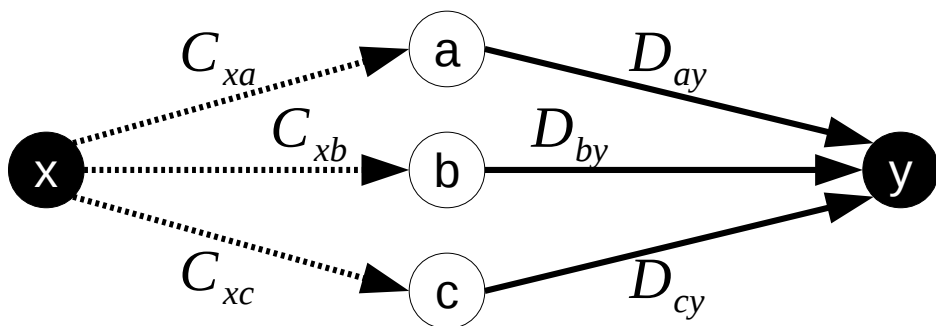
A **equação**, é utilizada para encontrar o menor custo (distância) entre um nó de origem (x) e um de destino (y), através de alguns nós intermediários (a, b, c, \dots). A seguir é apresentado o caso geral no qual D_{ij} é a menor distância e c_{ij} é o custo entre i e j .

$$D_{xy} = \min(c_{xa} + D_{ay}, (c_{xb} + D_{by}), (c_{xc} + D_{cy}), \dots$$

Ou seja, a formula encontra o mínimo dentre os caminhos.

No algoritmo de vetor de distância, normalmente se faz necessário atualizar o menor custo de um nó intermediário, tal como z , se o último é menor. Nesse caso a equação é:

$$D_{xy} = \min\{D_{xy}, (c_{xz} + D_{zy})\}$$



Há duas coisas importantes no algoritmo de vetor de distância: a (i) **equação Bellman-Ford** e (ii) o **conceito de vetor de distância**.

A **equação**, é utilizada para encontrar o menor custo (distância) entre um nó de origem (x) e um de destino (y), através de alguns nós intermediários (a, b, c, \dots). A seguir é apresentado o caso geral no qual D_{ij} é a menor distância e c_{ij} é o custo entre i e j .

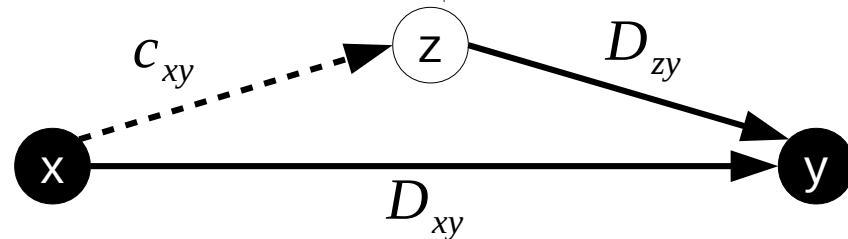
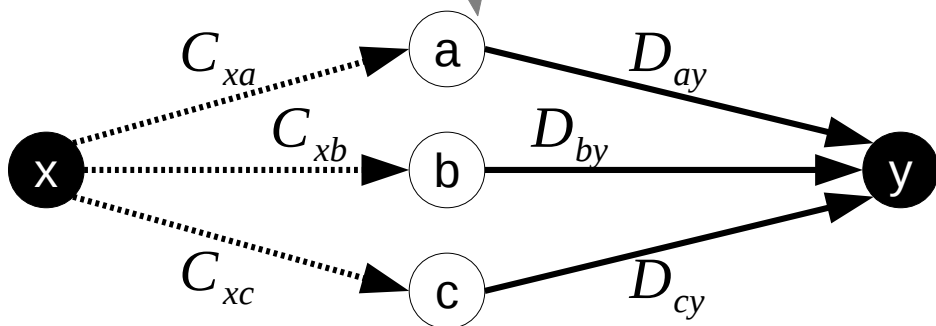
$$D_{xy} = \min(c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}), \dots$$

Escolher qual é o menor caminho/custo dentre esses existentes!

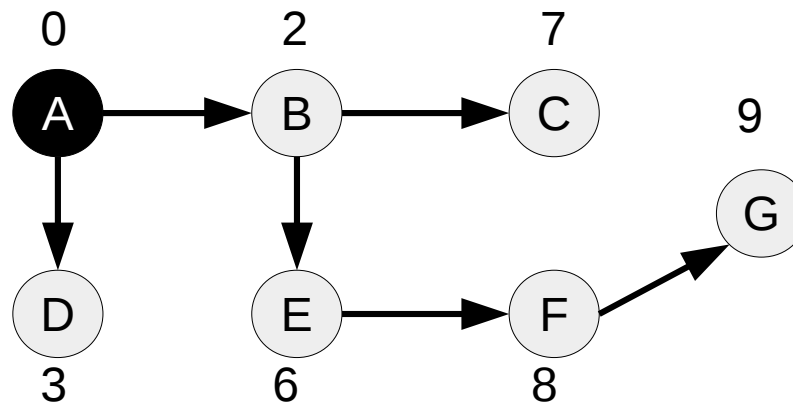
Encontra o mínimo dentre os caminhos de distância, normalmente se faz através de um nó intermediário, tal como z , então:

$$D_{xy} = \min\{D_{xy}, (c_{xz} + D_{zy})\}$$

Se há um caminho/custo menor troque...



No **conceito de vetor de distância**, a árvore de custo mínimo é a combinação dos caminhos de custo mínimo da raiz da árvore para todos os destinos. O vetor de distância desconecta esses caminhos da árvore e cria um vetor de distância. O vetor de distância, neste caso, é um vetor (*array*) unidimensional para representar a árvore.



| | |
|---|---|
| A | 0 |
| B | 2 |
| C | 7 |
| D | 3 |
| E | 6 |
| F | 8 |
| G | 9 |

Caminho
do nó A

O nome do vetor de distância define a raiz, os índices os destinos e os valores são os caminhos de menor custo encontrados da raiz até o destino (cada nó).

É preciso perceber que o vetor de distância não dá o caminho da raiz até os destinos, esse apenas apresenta o último custo.

Cada nó/roteador precisa criar seu próprio vetor de distância (árvore de menor custo), executando o algoritmo.

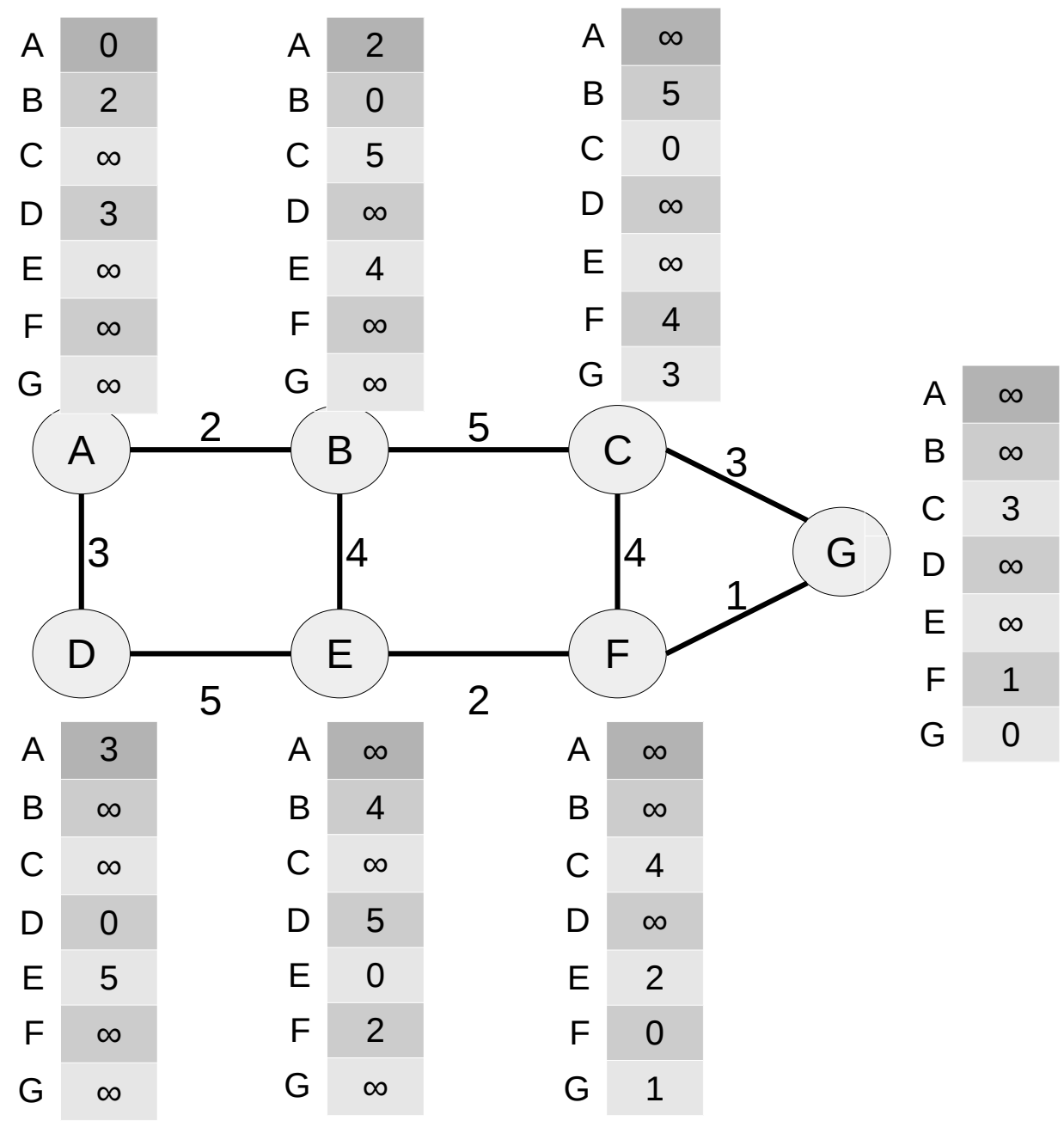
Para isso, cada nó/roteador em uma inter-rede, quando o roteador é ligado, inicializa sua tabela de roteamento de modo a conter uma entrada para cada rede diretamente conectada.

Cada entrada na tabela identifica uma rede de destino e fornece a distância até essa rede, normalmente medidas em saltos (*hops* – que representa, por quantos roteadores é necessário passar para chegar ao destino). Assim, isso cria um vetor de distância bem rudimentar (simples e incompleto).

Então, o nó/roteador envia algumas mensagens de saudações para os nós/roteadores vizinhos e isso permite descobrir quem são seus vizinhos, as redes que eles conhecem e a distância.

As distâncias descobertas entre o nó em questão e seus vizinhos são inseridas no vetor de distância do nó em questão, o que dá mais informações a respeito da inter-rede para que o nó faça uma árvore de menor custo mais completa. Desta forma, o custo dos nós descobertos são inseridos no vetor e roteadores/redes não descobertos/encontrados serão determinadas como infinitas (grande valor de custo que representa que não ligação com aquele nó).

Os vetores de distância são feitos assincronamente, quando o nó correspondente é inicializado (*boot*).



Os vetores rudimentares apresentados anteriormente, não ajudam no encaminhamento de pacotes. Por exemplo, o nó A acha que não esta conectado com G.

Então, para que o algoritmo funcione corretamente é necessário que os nós da rede troquem informações entre si.

Assim, depois que cada nó cria seu vetor ele deve enviar para seus vizinhos. Depois de receber a tabela do vizinho o nó deve aplicar o algoritmo de Bellman-Ford.

É necessário entender que é necessário atualizar não somente os custos, mas também o número de nós da inter-rede. Depois disso o vetor se torna mais completo

Vetor recebido
de A

| | |
|---|----------|
| A | 0 |
| B | 2 |
| C | ∞ |
| D | 3 |
| E | ∞ |
| F | ∞ |
| G | ∞ |

$3+2 < \infty?$
Sim

Vetor antigo
de B

| | |
|---|----------|
| A | 2 |
| B | 0 |
| C | 5 |
| D | ∞ |
| E | 4 |
| F | ∞ |
| G | ∞ |

$(2+3)$

Novo vetor
de B

| | |
|---|----------|
| A | 2 |
| B | 0 |
| C | 5 |
| D | 5 |
| E | 4 |
| F | ∞ |
| G | ∞ |

A partir de A, B descobre um novo nó (D). Assim, o valor para chegar nesse nó é o valor para chegar em A (2), a partir de B, mais o valor para chegar de A ao novo nó (3).

O algoritmo de vetor de distância simplificado pode ser visto a seguir:







```

1. Distance_Vector_Routing()
2. {
    // set values to connected networks and  $\infty$  to not connected
3.   D = initialize_vector()
4.   send_vector_to_neighbors(D)
5.   while(1)
6.   {
        // for a vector Dw from neighbor w or any change in the link
7.       Dw = get_neighbors_vector()
        // N is the number of nodes
8.       for(y=1 to N)
9.       {
                //Bellman-Ford (case 2)
10.            D[y]=min[D[y],(c[myself][w]+Dw[y])]
11.        }
12.        if(vetor_changed(D)==true)
13.            send_vector_to_neighbors(D)
14.    }
15. }
```

Contar para o infinito (*count-to-infinity*)

Um problema com o roteamento por vetor de distância é que qualquer decremento no custo (boas notícias) são propagadas rapidamente, mas qualquer aumento no custo (más notícias) são propagadas bem devagar.

Para o protocolo de roteamento funcionar corretamente, se um *link* falhar (o custo se torna infinito - vai para um número bem alto), todos os outros roteadores devem ficar sabendo dessa falha imediatamente, mas com o algoritmo de vetor de distância isso demora muito para convergir.

| A | B | C | D | E | | A | B | C | D | E | |
|--|---|---|---|---|-------------------------------|---|---|---|---|---|------------------------------|
|  |  |  |  |  | |  |  |  |  |  | |
| ∞ | ∞ | ∞ | ∞ | ∞ | Inicialmente (A off) | 0 | 1 | 2 | 3 | 4 | Inicialmente (A on) |
| 0 | 1 | ∞ | ∞ | ∞ | 1ª troca (A on) | ∞ | 3 | 2 | 3 | 4 | 1ª troca (A off) |
| 0 | 1 | 2 | ∞ | ∞ | 2ª troca | ∞ | 3 | 4 | 3 | 4 | 2ª troca |
| 0 | 1 | 2 | 3 | ∞ | 3ª troca | ∞ | 5 | 4 | 5 | 4 | 3ª troca |
| 0 | 1 | 2 | 3 | 4 | 4ª troca | ∞ | 5 | 6 | 5 | 6 | 4ª troca |
| | | | | | | ∞ | 7 | 6 | 7 | 6 | 5ª troca |
| | | | | | | ∞ | 7 | 8 | 7 | 8 | 6ª troca |
| | | | | | | | ... | | | | |
| | | | | | | ∞ | ∞ | ∞ | ∞ | ∞ | muitas trocas |

Contar para o infinito (count-to-infinity)

Um problema com o roteamento por vetor de distância é que as atualizações de decremento no custo (boas notícias) são propagadas mais lentamente do que as de incremento (más notícias) são.

Má notícia! O nó A está off-line...

Para o protocolo de roteamento funcionar corretamente, o custo se torna infinito - vale para um número limitado de roteadores. Mas, se os roteadores devem ficar sabendo dessa falha no nó A, o algoritmo de vetor de distância isso demora muito para convergir.

Mas, meu vizinho (C), disse que consegue chegar em A com o custo 2. Então, o meu custo para B agora é: 3 (1+2)... Essa Fake-News vai ocorrer em todos os roteadores, até que o valor chegue no "infinito".

| A | B | C | D | E | |
|---|---|---|---|---|----------------------|
| ○ | ● | ● | ● | ● | |
| ∞ | ∞ | ∞ | ∞ | ∞ | Inicialmente (A off) |
| 0 | 1 | ∞ | ∞ | ∞ | 1ª troca (A on) |
| 0 | 1 | 2 | ∞ | ∞ | 2ª troca |
| 0 | 1 | 2 | 3 | ∞ | 3ª troca |
| 0 | 1 | 2 | 3 | 4 | 4ª troca |

Nó A ligou, boa notícia! Propaga o custo para chegar em A.

| A | B | C | D | E | |
|---|---|-----|---|---|---------------------|
| ○ | ● | ● | ● | ● | |
| 0 | 1 | 2 | 3 | 4 | Inicialmente (A on) |
| ∞ | 3 | 2 | 3 | 4 | 1ª troca (A off) |
| ∞ | 3 | 4 | 3 | 4 | 2ª troca |
| ∞ | 5 | 4 | 5 | 4 | 3ª troca |
| ∞ | 5 | 6 | 5 | 6 | 4ª troca |
| ∞ | 7 | 6 | 7 | 6 | 5ª troca |
| ∞ | 7 | 8 | 7 | 8 | 6ª troca |
| | | ... | | | |
| ∞ | ∞ | ∞ | ∞ | ∞ | muitas trocas |

Split Horizon

Uma solução para o problema da contagem para o infinito (*count to infinity*) é chamada de horizonte dividido ou estreitamento de horizontes.

Nessa estratégia, ao invés enviar a tabela de vetor de distância através de cada interface, cada nó envia somente parte da tabela através de cada interface. Ou seja, não envia informações de volta para a mesma direção pela qual a informação original chegou.

Exemplo, se o nó B acredita que a melhor rota (menor custo) para alcançar uma rede X é através do nó A, então ele não precisa enviar esse pedaço de informação para A. Pois, essa informação já vem do nó A – era isso que causava confusão.

Com essa abordagem, assim que um nó falhar, logo após a próxima iteração os outros nós saberão que esse nó está inalcançável.

Envenenamento Reverso (*Poison Reverse*)

Usar a estratégia *Split Horizon* tem uma desvantagem. Normalmente, os protocolos de roteadores utilizam um *timer*, e se não há informações a respeito de uma rota, tal rota é eliminada.

Contudo, é preciso notar que não há como saber se a falta de notícias a respeito de uma rota foi ocasionada por alguma falha nessa rota ou devido a estratégia *Split Horizon*.

Na estratégia *Poison Reverse* um nó pode ignorar a estratégia *Split Horizon* e enviar o valor de uma rota para o nó de origem da informação, mas quando isso acontece o valor enviado é infinito.

Ou seja, um nó envia o valor de uma rota como infinito, para o nó fonte da informação como um aviso/alerta, informando que a rede está inalcançável.

Roteamento Link-State

O algoritmo de vetor de distância foi utilizado na ARPANET, mas foi substituído pelo roteamento Link-State (*Link State Routing* - Roteamento por estado de *link*). Isso ocorreu, pois a solução por vetor de distância demora muito para convergir quando a topologia da rede muda (problema da contagem para o infinito), e isso foi resolvido totalmente por um novo algoritmo chamado algoritmo Link-State.

No novo algoritmo Link-State o custo associado com uma aresta define o estado do *link*. *Links* com baixo custo são melhores do que links com alto custo. Também, se o custo de um link for infinito, isso significa que tal *link* não existe ou está com problemas.

A ideia por trás do roteamento Link-State pode ser dada por cinco passos:

- 1) Descobrir os vizinhos e aprender seus endereços;
- 2) Identificar métrica de distância ou custo para cada vizinho;
- 3) Construir um pacote que informa tudo o que foi descoberto na rede;
- 4) Enviar e receber pacotes que informam a respeito da rede;
- 5) Calcular o menor caminho para todos os outros roteadores.

Cada roteador deve seguir esses passos para que tudo funcione corretamente.

Em resumo, no algoritmo Link-State, a topologia da rede deve ser distribuída por todos os roteadores. Então, o algoritmo de Dijkstra (visto anteriormente) pode ser executado em cada roteador para encontrar o menor caminho para cada roteador.

Aprendendo a respeito dos vizinhos

Quanto um roteador é ligado (*boot*), sua primeira tarefa é aprender quem são seus vizinhos. Isso é feito através de uma mensagem HELLO. O roteador que enviou o HELLO espera uma mensagem de resposta com o nome do roteador. Tal nome é globalmente único.

Determinando o custo dos *links*

O algoritmo State-Link requer que cada *link* tenha um custo relativo a distância ou outra métrica qualquer, para determinar o menor caminho.

Tal custo pode ser dado automaticamente ou configurado pelo gerente da rede.

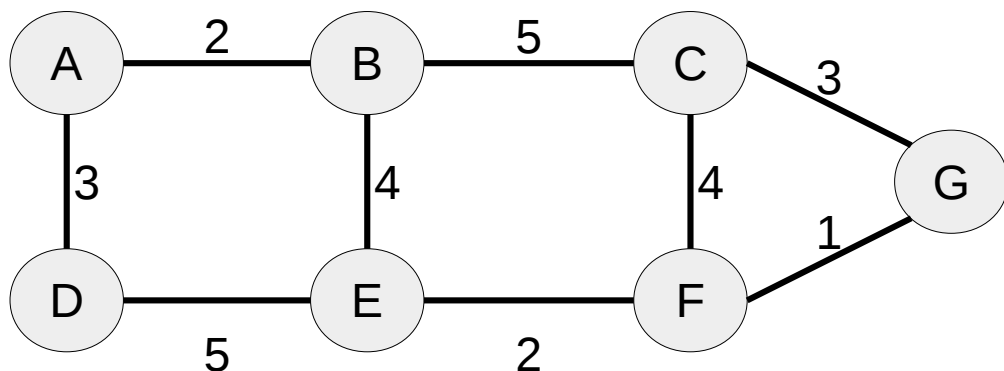
Uma escolha comum é fazer o custo inversamente proporcional a largura de banda do *link*. Por exemplo: um *link* com 1 Gbps pode ter o custo 1 e um *link* com 100 Mbps, teria o custo 10. Isso faz com que *links* com altas capacidades sejam escolhidos.

Se a rede percorre grandes distâncias (WANs), métricas como atraso (*delay*), podem ser consideradas no custo.

Criando um banco de dados Link-State

Para criar a árvore de menor custo, nesse método, cada nó precisa ter um mapa completo da rede, o que significa saber o estado de cada *link* da rede. Essa coleção de estados para todos os *links* da rede chama-se LSDB (*Link-State Database*). Toda a rede deve ter apenas um LSDB, sendo que cada nó precisa ter um banco de dados desse para ser capaz de criar a árvore de menor custo.

O LSDB pode ser representado como uma matriz, na qual os valores de cada célula define o custo correspondente para o *link*.



| | A | B | C | D | E | F | G |
|---|----------|----------|----------|----------|----------|----------|----------|
| A | 0 | 2 | ∞ | 3 | ∞ | ∞ | ∞ |
| B | 2 | 0 | 5 | ∞ | 4 | ∞ | ∞ |
| C | ∞ | 5 | 0 | ∞ | ∞ | 4 | 3 |
| D | 3 | ∞ | ∞ | 0 | 5 | ∞ | ∞ |
| E | ∞ | 4 | ∞ | 5 | 0 | 2 | ∞ |
| F | ∞ | ∞ | 4 | ∞ | 2 | 0 | 1 |
| G | ∞ | ∞ | 3 | ∞ | ∞ | 1 | 0 |

Bem, mas uma questão é **“como cada nó pode criar sua LSDB contendo informações de toda rede?”** A resposta é um processo chamado de *flooding* (enchente/enxurrada).

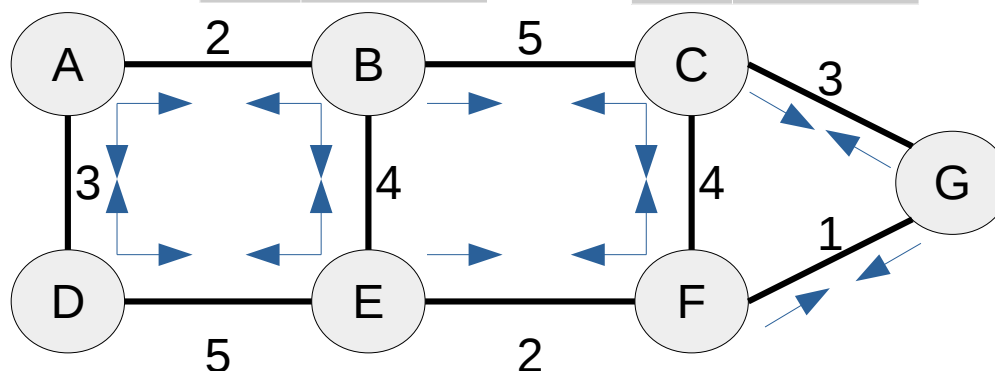
Cada nó envia a mensagem de HELLO para todos os vizinhos conectados diretamente, para obter duas informações: (i) a identificação do nó e (ii) o custo do *link*. A combinação dessas duas informações é chamada de LSP (*Link-State Packet*). O LSP é enviado para cada interface.

| Nó | Custo |
|----|-------|
| B | 2 |
| D | 3 |

| Nó | Custo |
|----|-------|
| A | 3 |
| E | 5 |

| Nó | Custo |
|----|-------|
| A | 2 |
| C | 5 |
| E | 4 |

| Nó | Custo |
|----|-------|
| B | 5 |
| F | 4 |
| G | 3 |



| Nó | Custo |
|----|-------|
| C | 3 |
| F | 1 |

| Nó | Custo |
|----|-------|
| B | 4 |
| D | 5 |
| E | 2 |

| Nó | Custo |
|----|-------|
| C | 4 |
| E | 2 |
| G | 1 |

Então, um roteador deve receber todos os pacotes apresentados na figura anterior, para a partir desse construir o LSDB e daí calcular a árvore de menor custo.

Criar um LSP é fácil, complicado é determinar quando fazer. Uma possibilidade é criar periodicamente, em intervalos regulares. Outra maneira possível é criá-los quando algum evento significativo ocorrer, tal como: falhas no *link*, um roteador sendo adicionado na rede, alteração da velocidade de alguma placa de rede, etc.

A parte mais complicada do algoritmo Link-State é distribuir os LSP. Todos os roteadores devem obter os pacotes de forma rápida e confiável. Caso contrário os roteadores irão computar rotas erradas, o que pode causar *loops*, redes/*hosts* inalcançáveis, dentre outros problemas.

A ideia fundamental é inundar (*flooding*) a rede de pacotes LSP para enviar informações para toda a rede. É claro que inundar a rede com pacotes traz problemas. Para controlar a inundação, cada pacote contém um número de sequência, que é incrementado a cada novo pacote enviado.

Os roteadores rastreiam esses números de sequência de cada roteador da rede, assim quando chega um LSP o roteador verifica se esse já foi ou não processado, para evitar redundância.

São descartados pacotes LSP com número de sequência menor ou igual ao último pacote processado pelo roteador. É claro que ainda há problemas inerentes ao uso do número de sequência, mas iremos parar por aqui (o que acontece se um roteador é reinicializado?). Uma solução é controlar a idade do pacote (*age*), através do tempo.

Comparando o Link-State com o algoritmo de vetor de distância. No vetor de distância, cada roteador pede para seus vizinhos o que eles sabem da rede. Já no Link-State, cada roteador pede para toda a rede o que ela sabe sobre seus vizinhos.

Criando árvores de menor custo

Uma vez que um roteador tenha a LSDB completa, esse pode criar um grafo completo da rede.

Cada link é representado duas vezes, uma em cada direção. Direções diferentes podem ter custos distintos.

Agora o algoritmo de Dijkstra (visto anteriormente) pode ser executado localmente (no roteador) para construir a árvore de custo mínimo, com o roteador em questão como raiz.

O resultado obtido pelo algoritmo de Dijkstra indica para o roteador qual link usar para chegar em cada destino da rede. Tal informação é então instalada na tabela de roteamento.

O roteamento por estado de link é altamente usado nas redes atuais:

Muitos ISPs usam o protocolo de estado de *link* IS-IS (*Intermediate System-Intermediate System*). O IS-IS foi projetado para uma rede chamada DECnet, depois foi adotado pela ISO para ser utilizado com o OSI e modificado para trabalhar com vários outros protocolos, tal como o IP.

Atualmente, um protocolo de estado de link muito utilizado é o OSPF (*Open Shortest Path First*), que foi projetado pela IETF, muitos anos após o IS-IS e por isso trouxe muitas inovações.

Breve comparação entre Link-State e vetor de distância

Em termos gerais o algoritmo Link-State requer mais memória e processador, quando comparado com o algoritmo de vetor de distância. Todavia, em termos práticos o roteamento por estado de *link* funciona melhor, pois não sofre com o problema da convergência lenta (contagem para o infinito).

No próximo arquivo de slide (Redes 12.2 – Roteamento 3) iremos abordar os protocolos RIP (vetor de distância) e OSPF (Link-State), na prática. Também será abordado brevemente tecnologias interdomínio.

Referência:

FOROUZAN, Behrouz A. Data Communications and Networking. 5ª Edição. 2013.

TANENBAUM, Andrew S. Computer Networks. 5ª Edição. 2011.

CISCO. Routing and Switching Essentials Companion Guide. 3ª Edição. 2014.

COMER, Douglas E. Interligação de Redes em TCP/IP Volume 1. 5ª Edição. 2006.

FILIPPETTI, Marco A. CCNA 4.0 Guia Completo de Estudo. 2006.

ODOM, Wendell. CCENT/CCNA ICND1 Guia Oficial de Certificação do Exame. 2ª Edição. 2008.

(https://www.youtube.com/watch?v=aJ_2c9NVC1c)

(<https://www.youtube.com/watch?v=vEztwiTELWs>)

Fim!