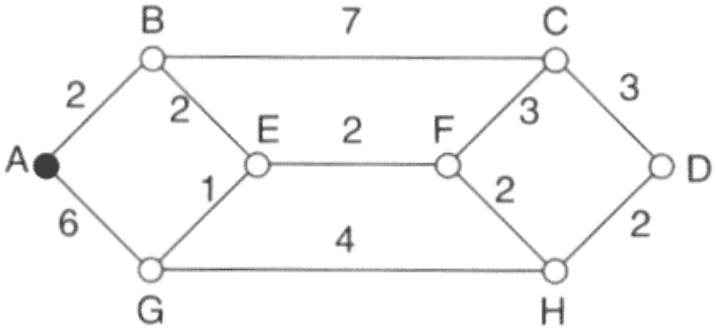
Roteamento Baseado em Fluxo (*Flow-Based Routing*)

- **Os algoritmos anteriormente estudados levavam em conta apenas a topologia, não considerando a carga.**
- **Se, por exemplo, houver uma grande quantidade de tráfego de A para B, na Figura abaixo, então pode ser melhor rotear o tráfego de A para C via AGEFC, embora este caminho seja muito maior do que ABC.**



O Algoritmo Baseado em Fluxo é um algoritmo estático que utiliza tanto a topologia quanto a carga para o roteamento.

Em algumas redes, o fluxo médio de dados entre cada par de nós é relativamente estável e previsível.

Por exemplo:

- **Em uma rede corporativa de uma cadeia de lojas, cada loja pode desejar mandar a outras relatórios de vendas, atualizações de estoque e outros tipos de mensagens bem definidas para *sites* conhecidos, em um padrão pré-definido.**
- **De tal forma, o tráfego médio de i para j é conhecido antecipadamente e, a uma razoável aproximação, constante no tempo.**

Em condições como estas é possível analisar o fluxo matematicamente, para otimizar o roteamento.

Para uma dada linha, se a capacidade e o fluxo médio são conhecidos é possível computar o atraso médio de pacotes para toda a rede, a partir da teoria de filas.

O problema do roteamento então se reduz a encontrar o algoritmo de roteamento que produza o mínimo atraso médio para a sub-rede.

Para usar esta técnica, certas informações devem ser conhecidas antecipadamente:

1. **A topologia da sub-rede;**
2. **A matriz de tráfego, F_{ij};**
3. **A matriz de capacidade da linha, C_{ij}, especificando a capacidade de cada linha em bps;**
4. **Um algoritmo de roteamento (provisório).**

Como exemplo deste método, considere a sub-rede *full-duplex* mostrada na Figura (a) abaixo, em que os pesos nos arcos dão as capacidades, C_{ij}, em cada direção, medidas em kbps.

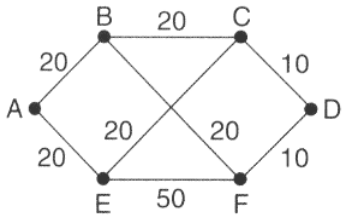


Figura (a): Uma sub-rede com a capacidade das linhas mostrada em kbps.

Para a mesma sub-rede, é mostrada a matriz de tráfego (Figura (b)), que possui uma entrada para cada par fonte-destino.

Source \ Destination	A	B	C	D	E	F
A		9 AB	4 ABC	1 ABFD	7 AE	4 AEF
B	9 BA		8 BC	3 BFD	2 BFE	4 BF
C	4 CBA	8 CB		3 CD	3 CE	2 CEF
D	1 DFBA	3 DFB	3 DC		3 DCE	4 DF
E	7 EA	2 EFB	3 EC	3 ECD		5 EF
F	4 FEA	4 FB	2 FEC	4 FD	5 FE	

Figura (b): O tráfego em pacotes/segundo e a matriz de roteamento para a sub-rede da Figura (a).

Na matriz de roteamento, a entrada relativa ao par fonte i – destino j mostra a rota a ser usada para o tráfego i–j e também o número de pacotes/segundo a serem enviados da fonte i ao destino j.

Da Figura, pode-se observar que 3 pacotes por segundo vão de B para D, pela rota BFD.

(Note que algum algoritmo de roteamento já foi aplicado para derivar as rotas mostradas na matriz).

De posse desta informação, pode ser calculado o tráfego total na linha i, λ_i .

Source \ Destination	A	B	C	D	E	F
A		9 AB	4 ABC	1 ABFD	7 AE	4 AEF
B	9 BA		8 BC	3 BFD	2 BFE	4 BF
C	4 CBA	8 CB		3 CD	3 CE	2 CEF
D	1 DFBA	3 DFB	3 DC		3 DCE	4 DF
E	7 EA	2 EFB	3 EC	3 ECD		5 EF
F	4 FEA	4 FB	2 FEC	4 FD	5 FE	

Figura (b): O tráfego em pacotes/segundo e a matriz de roteamento para a sub-rede da Figura (a).

Por exemplo, o tráfego B-D contribui com 3 pacotes por segundo para a linha BF e também 3 pacotes por segundo para a linha FD.

De forma similar, o tráfego A-D contribui com 1 pacote por segundo para cada uma das três linhas (AB, BF e FD).

O tráfego total em cada linha é mostrado na coluna λ_i da Tabela abaixo.

i	Line	λ_i (pkts/sec)	C_i (kbps)	μC_i (pkts/sec)	T_i (msec)	Weight
1	AB	14	20	25	91	0.171
2	BC	12	20	25	77	0.146
3	CD	6	10	12.5	154	0.073
4	AE	11	20	25	71	0.134
5	EF	13	50	62.5	20	0.159
6	FD	8	10	12.5	222	0.098
7	BF	10	20	25	67	0.122
8	EC	8	20	25	59	0.098

Figura (c): Análise da sub-rede da Figura (a) usando um tamanho de pacote médio de 800 bits. O tráfego reverso (BA, CB, etc.) é igual ao tráfego direto.

» **Neste exemplo, todo o tráfego é simétrico, ou seja, o tráfego XY é idêntico ao tráfego YX, para todo X e Y. Em redes reais, esta condição não é verdadeira.**

» **A Tabela também mostra o número médio de pacotes/segundo em cada linha, μC_i, assumindo um tamanho médio de pacote de $1/\mu = 800$ bits.**

» **A penúltima coluna da Tabela apresenta o atraso médio para cada linha derivado a partir da teoria das filas, de acordo com**

$$T = \frac{1}{\mu C - \lambda}$$ **, onde**

- $1/\mu$ **é o tamanho médio do pacote em bits,**
- ***C*** **é a capacidade em bps e**
- λ **é o fluxo médio em pacotes/segundo.**

Por exemplo, com

- **uma capacidade de $\mu C = 25$ pacotes/segundo e**
- **um fluxo efetivo de $\lambda = 14$ pacotes/segundo,**
- **atraso médio será:**

$$T = \frac{1}{\mu C - \lambda} = \frac{1}{25 - 14} = \frac{1}{11} = 91\,\text{ms}$$

Note que com $\lambda = 0$, o atraso médio será 40 ms, devido à capacidade $\mu C = 25$ pacotes/segundo.

Em outras palavras, o atraso inclui tanto tempo de enfileiramento quanto tempo de serviço.

Para computar o tempo de atraso médio para a sub-rede inteira, toma-se a soma ponderada de cada uma das oito linhas, sendo que o peso da ponderação é a fração do tráfego total que utiliza aquela linha.

i	Line	λ_i (pkts/sec)	C_i (kbps)	μC_i (pkts/sec)	T_i (msec)	Weight
1	AB	14	20	25	91	0.171
2	BC	12	20	25	77	0.146
3	CD	6	10	12.5	154	0.073
4	AE	11	20	25	71	0.134
5	EF	13	50	62.5	20	0.159
6	FD	8	10	12.5	222	0.098
7	BF	10	20	25	67	0.122
8	EC	8	20	25	59	0.098

Neste exemplo, a média é 86ms, ou seja, o tempo de atraso médio para a sub-rede inteira = 86ms.

Para avaliar um diferente algoritmo de Roteamento Baseado em Fluxo, este processo é repetido, considerando os diferentes fluxos em questão, e buscando determinar o novo atraso médio para a sub-rede.

Se considerarmos algoritmos de roteamento de caminhos únicos (como os que vimos até agora), haverá apenas um número finito de formas para rotear pacotes de cada fonte a cada destino.

É sempre possível escrever um programa para testar todos, um após outro, e determinar qual dos algoritmos tem o menor atraso médio.

Este cálculo pode ser feito antecipadamente (*off-line*), de forma que o tempo de execução do algoritmo não constitui um problema.

O algoritmo que apresentar menor atraso médio será o melhor algoritmo de roteamento.

ALGORITMOS DE ROTEAMENTO DINÂMICOS

Redes de computadores modernas utilizam algoritmos de roteamento dinâmicos.

Dois algoritmos dinâmicos são os mais populares:

I. **Roteamento pelo Vetor Distância (*Distance Vector Routing*)**

II. **Roteamento pelo Estado do *Link* (*Link State Routing*)**

Roteamento pelo Vetor Distância (*Distance Vector Routing*)

- **Neste algoritmo, cada roteador mantém uma tabela (ou vetor) que contém a melhor distância conhecida até cada destino e qual a linha a ser usada para chegar lá.**

- **Estas tabelas são atualizadas através da troca de informações com os vizinhos.**

- **O algoritmo de roteamento pelo vetor distância é algumas vezes chamado de algoritmo de roteamento distribuído de Bellman-Ford e também de Algoritmo Ford-Fulkerson.**

- **Foi o algoritmo originalmente usado na ARPANET e também usado posteriormente na Internet, sob o nome de RIP.**

No roteamento pelo vetor distância, cada roteador mantém uma tabela de roteamento contendo uma entrada para cada roteador na sub-rede.

Esta entrada contém duas partes:

1. **a linha de saída preferencial para chegar ao destino e**
2. **uma estimativa do tempo ou distância para atingí-lo.**

A métrica usada pode ser:
- **número de *hops*,**
- **atraso em milissegundos,**
- **nº total de pacotes enfileirados ao longo do caminho,**
- **ou alguma informação que se assemelhe.**

O roteador deve conhecer a distância até cada um dos seus vizinhos:

- **Se a métrica for *hops*, a distância é 1 *hop*.**
- **Se a métrica for tamanho de fila, o roteador simplesmente examina cada fila.**
- **Se a métrica for atraso, o roteador pode medir o atraso diretamente com pacotes especiais denominados ECHO, que o receptor etiqueta com informação de tempo (*timestamps*) e envia de volta o mais rápido possível.**

Como exemplo, consideremos que a métrica usada é o atraso e que o roteador conhece o atraso relativo a cada um dos seus vizinhos.

A cada T ms, cada roteador:

- **envia a cada vizinho uma lista de seu atraso estimado a cada destino e**
- **recebe uma lista similar de cada vizinho.**

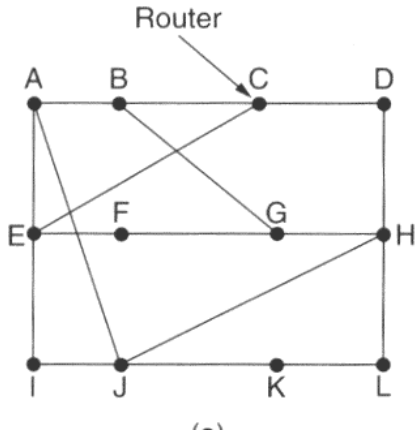
» **Imagine que uma destas tabelas tenha recém chegado de um vizinho X, com X_i sendo a estimativa de X de quanto tempo levará para chegar ao roteador i.**

» **Se o roteador sabe que o seu atraso para X é *m* ms, também sabe que poderá alcançar o roteador i através de X em (X_i + *m*)ms, via X.**

» **Executando este cálculo par cada vizinho, um roteador pode determinar qual estimativa parece ser a melhor e usá-la juntamente com a linha correspondente em sua nova tabela de roteamento.**

» **Observe que a tabela velha de roteamento não é usada no cálculo.**

Ilustração deste processo de atualização:



(a) Sub-net

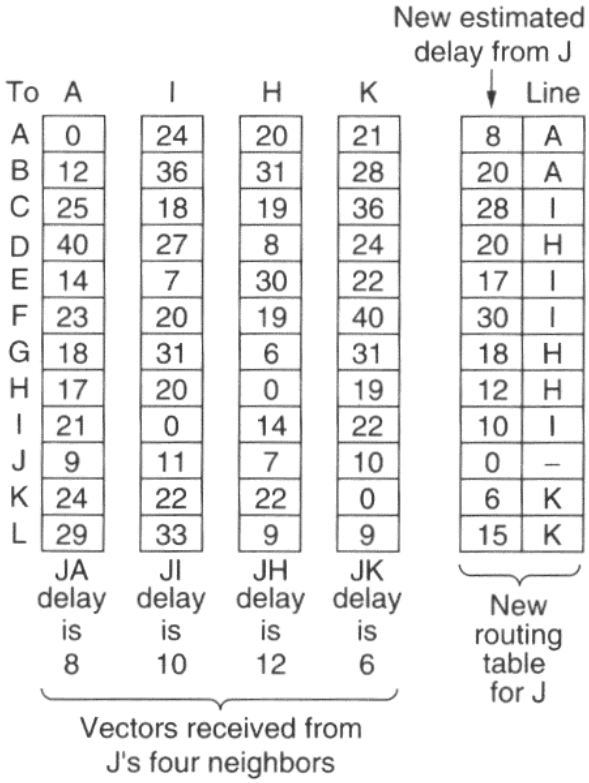
New estimated delay from J

To	A	I	H	K	Line
A	0	24	20	21	8 A
B	12	36	31	28	20 A
C	25	18	19	36	28 I
D	40	27	8	24	20 H
E	14	7	30	22	17 I
F	23	20	19	40	30 I
G	18	31	6	31	18 H
H	17	20	0	19	12 H
I	21	0	14	22	10 I
J	9	11	7	10	0 –
K	24	22	22	0	6 K
L	29	33	9	9	15 K
	JA delay is 8	JI delay is 10	JH delay is 12	JK delay is 6	New routing table for J

Vectors received from J's four neighbors

(b) Entradas a partir de A, I, H, K e a nova tabela de roteamento para J.

- **As primeiras quatro colunas da Figura (b) mostram os vetores de atraso recebidos dos vizinhos do roteador J, na sub-net mostrada na Figura (a).**
- **O roteador A informa que tem um atraso de 12 ms para B, um atraso de 2 ms para C, um atraso de 40 ms para D, etc.**



- **Suponha que J tenha medido ou estimado seu atraso com relação a seus vizinhos A, I, H e K como sendo, respectivamente, 8, 10, 12 e 6 ms.**
- **J computa sua nova rota para o roteador G da seguinte forma:**
- **J sabe que pode chegar a A em 8ms, e**
- **A diz que pode chegar a G em 18ms, de tal forma que**
- **J sabe que pode contar com um atraso de 26 ms (8+18) para chegar a G se enviar os pacotes de G para A .**
- **De forma similar, J computa o atraso para G via I, H e K como 41ms = (31+10), 18ms = (6+12) e 37ms = (31+6), respectivamente.**
- **O melhor deste valores é 18, de tal forma que o roteador J faz uma entrada em sua tabela de roteamento, informando que o atraso para G é 18ms, e que a rota para usar é via H.**
- **O mesmo cálculo é desenvolvido para todos os outros destinos, com a nova tabela de roteamento mostrada na última coluna.**

O Problema "Contar até o Infinito"

O roteamento pelo vetor de distância funciona em teoria, mas apresenta um sério problema na prática: embora convirja para uma resposta correta, a convergência é muito lenta.

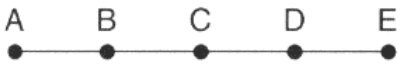
Em particular: o algoritmo reage rapidamente a boas notícias, mas lentamente a más notícias.

Considere um roteador cuja melhor rota ao destino X seja grande.

Se na próxima troca o vizinho A reporta um atraso pequeno para X, o roteador apenas comuta para a linha A para enviar o tráfego a X.

Ou seja, em uma única troca de vetores (troca de informação registrada nos vetores), a boa notícia é processada.

Para exemplificar a rápida propagação de boas notícias, considere a sub-rede linear de 5 nós mostrada na Figura abaixo, onde a métrica de atraso é o número de *hops*.



∞	∞	∞	∞	Initially
1	∞	∞	∞	After 1 exchange
1	2	∞	∞	After 2 exchanges
1	2	3	∞	After 3 exchanges
1	2	3	4	After 4 exchanges

Suponha que A esteja fora de operação inicialmente e todos os outros roteadores saibam disto.

Em outras palavras, todos eles gravaram o atraso para A como infinito.

Quando A entra em operação, os outros roteadores aprendem sobre ele através da trocas de vetores (troca de estado dos vetores).

Por simplicidade assuma que a troca de vetores em todos os roteadores é realizada ao mesmo tempo.

No momento da primeira troca, B aprende que seu vizinho à esquerda tem atraso zero para A.

B agora faz uma entrada em sua tabela de roteamento, registrando que A está um *hop* à sua esquerda.

Todos os outros roteadores ainda "pensam" que A está fora.

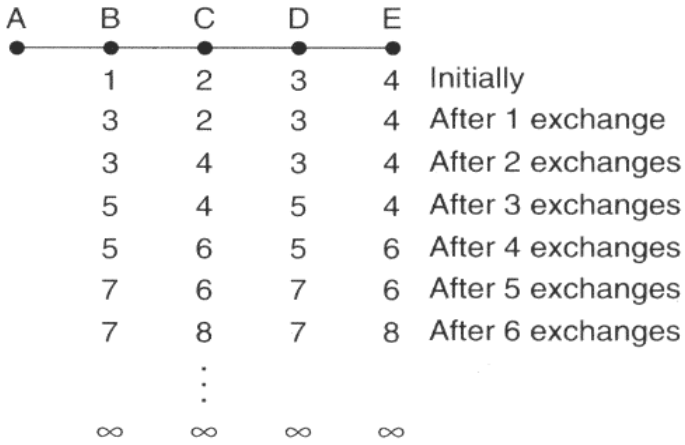
Neste ponto, as entradas da tabela de roteamento para A são como é mostrado na segunda linha da Figura.

Na próxima troca, C aprende que B tem um caminho de tamanho 1 para A e então atualiza sua tabela de roteamento para indicar um caminho de tamanho 2, mas D e E não sabem ainda da boa notícia.

Claramente, a boa notícia está se espalhando a uma taxa de um *hop* por troca.

Em uma sub-rede cujo maior caminho é de tamanho *N hops*, dentro de *N* trocas todos os roteadores saberão notícias sobre linhas recentemente reaviadas e roteadores.

Agora consideremos a situação mostrada na Figura abaixo, na qual todas as linhas e roteadores estão inicialmente ativados.



Roteadores B, C, D e E têm distâncias para A de 1, 2, 3 e 4 respectivamente.

O roteador A deixa de operar, ou, de forma alternativa, a linha entre A e B é interrompida (o que é efetivamente a mesma coisa, sob o ponto e vista de B).

Na primeira troca de pacotes, B não fica sabendo nada a respeito de A.

Felizmente, C diz: "Não se preocupe, eu tenho uma caminho para A de tamanho 2".

B não sabe que o caminho a que C se refere passa por si próprio (B).

Do ponto de vista de B, C pode ter dez linhas de saída, todas com caminhos independentes para A, de tamanho 2.

Como resultado, B acha que pode atingir A via C, com um tamanho de caminho de 3.

D e E não atualizam suas entradas para A na primeira troca.

Na segunda troca, C nota que cada dos seus vizinhos diz ter um caminho para A de tamanho 3.

C escolhe um dos caminhos aleatoriamente e faz sua nova distância para A ser 4, como mostrado na terceira linha da Figura.

Trocas subseqüentes produzem resultados que são mostrados na Figura.

A partir da figura, fica claro porque más notícias viajam lentamente: nenhum roteador jamais tem um valor maior do que o mínimo de todos os seus vizinhos.

Gradualmente, todos os roteadores acham seu caminho para o infinito, mas o número de trocas requeridas depende do valor numérico usado para infinito.

Por esta razão, é sábio escolher o valor infinito para o maior caminho, mais 1.

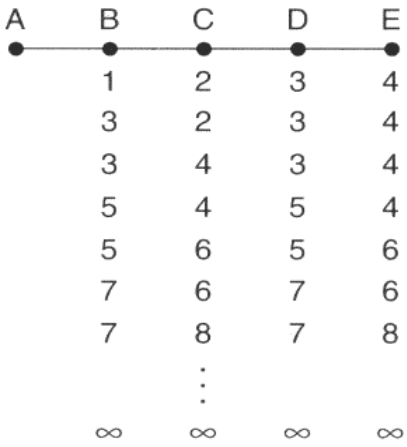
Se a métrica é atraso temporal, não há limite superior bem definido, de tal forma que uma valor elevado é necessário para prevenir que um caminho com um longo atraso seja tratado como desativado.

Algoritmo *Split Horizon*

Muitas soluções para este problema têm sido adotadas na literatura, cada uma delas mais complicada e menos útil que a outra.

O algoritmo *Split Horizon* é uma delas.

O *Split Horizon* trabalha da mesma forma que o roteamento pelo vetor distância, exceto pelo fato de que a distância até X não é reportada na linha que os pacotes para X são enviados (na verdade, é reportada como infinito).



B	C	D	E	
1	2	3	4	Initially
3	2	3	4	After 1 exchange
3	4	3	4	After 2 exchanges
5	4	5	4	After 3 exchanges
5	6	5	6	After 4 exchanges
7	6	7	6	After 5 exchanges
7	8	7	8	After 6 exchanges
	⋮			
∞	∞	∞	∞	

No estado inicial da Figura acima, por exemplo, C diz para D a verdade sobre a distância para A, mas C diz para B que sua distância para A é infinita.

De forma similar, D diz a verdade para E, mas mente para C.

Quando A pára de operar, na primeira troca, B descobre que a linha direta se foi, e C está reportando uma distância infinita para A, também.

Desde que nenhum de seus vizinhos pode chegar a A, B troca a sua distância a infinito, também.

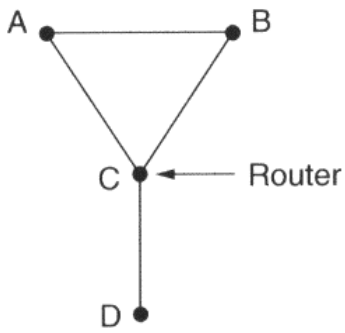
Na próxima troca, C escuta que A está inatingível de ambos os seus vizinhos, então marca A como inatingível também.

Usando o algoritmo *Split Horizon*, as más notícias se propagam a um *hop* por troca.

Esta taxa é muito melhor do que a taxa sem o uso do *Split Horizon*.

No entanto, o algoritmo *Split Horizon*, embora largamente utilizado, algumas vezes falha.

Considere, por exemplo, a sub-rede de quatro nós mostrada na Figura abaixo.



Inicialmente, tanto A quanto B têm uma distância 2 para D, e C tem uma distância 1.

Agora, suponha que a linha CD fique inoperante.

Usando o algoritmo *split horizon*, tanto A quanto B dizem a C que eles não podem atingir D.

Então C imediatamente conclui que D é inatingível e reporta isto a A e B.

Infelizmente, A escuta que B tem um caminho de tamanho 2 para D, de tal forma que assume que pode chegar a D via B, em 3 hops.

Na próxima troca, cada um deles troca sua distância para D como sendo 4.

Ambos gradualmente contam até infinito, precisamente o comportamento que o algoritmo procura evitar.