

Aula 4

Professores:

Daniel R. Figueiredo
Rosa M. M. Leão

Roteamento: Algoritmos do tipo link-state e broadcast

Conteúdo:

Aula 4

Professores:

Daniel R. Figueiredo

Rosa M. M. Leão

Roteamento: Algoritmos do tipo link-state e broadcast

Conteúdo:

- Classificação dos algoritmos de roteamento

Aula 4

Professores:

Daniel R. Figueiredo

Rosa M. M. Leão

Roteamento: Algoritmos do tipo link-state e broadcast

Conteúdo:

- Classificação dos algoritmos de roteamento
- Algoritmo do tipo link-state (Link State Routing Algorithm)

Aula 4

Professores:

Daniel R. Figueiredo

Rosa M. M. Leão

Roteamento: Algoritmos do tipo link-state e broadcast

Conteúdo:

- Classificação dos algoritmos de roteamento
- Algoritmo do tipo link-state (Link State Routing Algorithm)
- Algoritmos de broadcast

Aula 4

Professores:

Daniel R. Figueiredo

Rosa M. M. Leão

Roteamento: Algoritmos do tipo link-state e broadcast

Conteúdo:

- Classificação dos algoritmos de roteamento
- Algoritmo do tipo link-state (Link State Routing Algorithm)
- Algoritmos de broadcast
 - Flooding

Aula 4

Professores:

Daniel R. Figueiredo

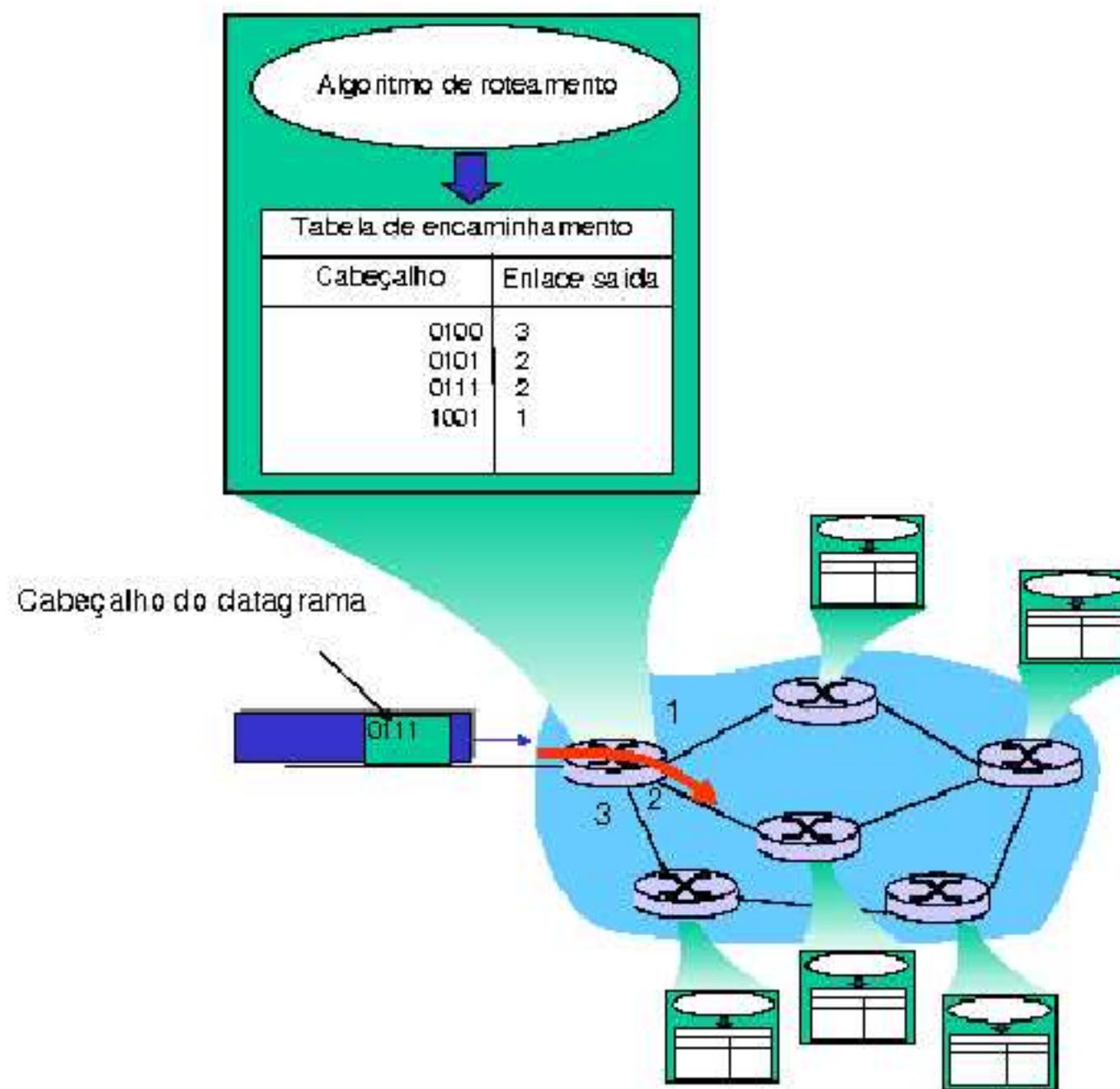
Rosa M. M. Leão

Roteamento: Algoritmos do tipo link-state e broadcast

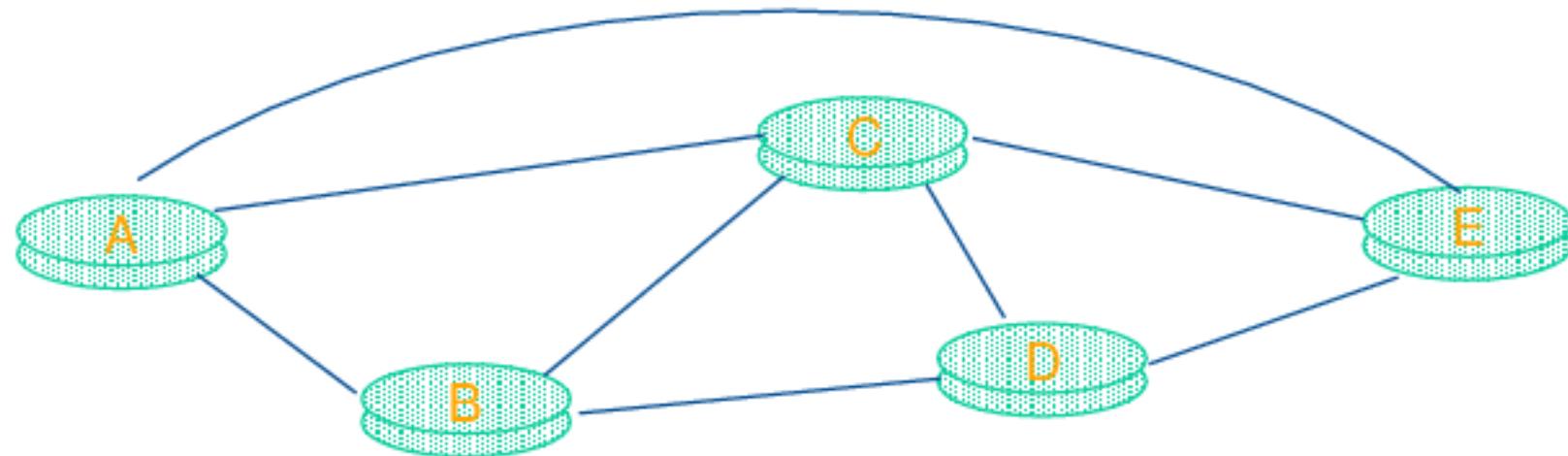
Conteúdo:

- Classificação dos algoritmos de roteamento
- Algoritmo do tipo link-state (Link State Routing Algorithm)
- Algoritmos de broadcast
 - Flooding
 - Repasse pelo caminho inverso (Reverse path forwarding)

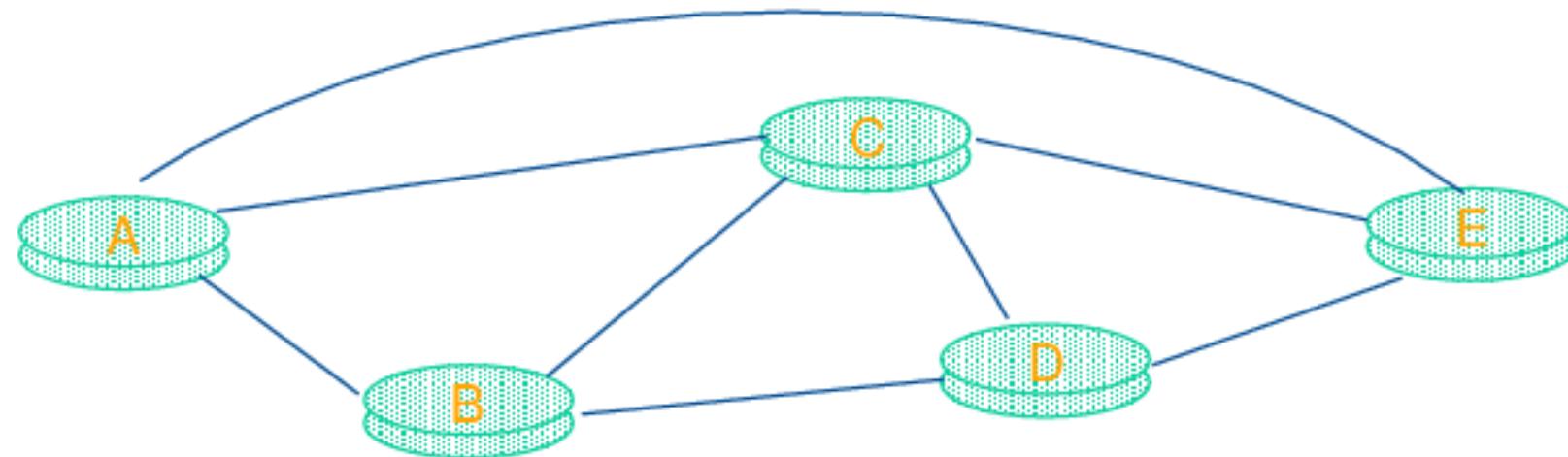
Como encaminhar datagramas na Internet?



Grafos: abstração para representar enlaces e roteadores

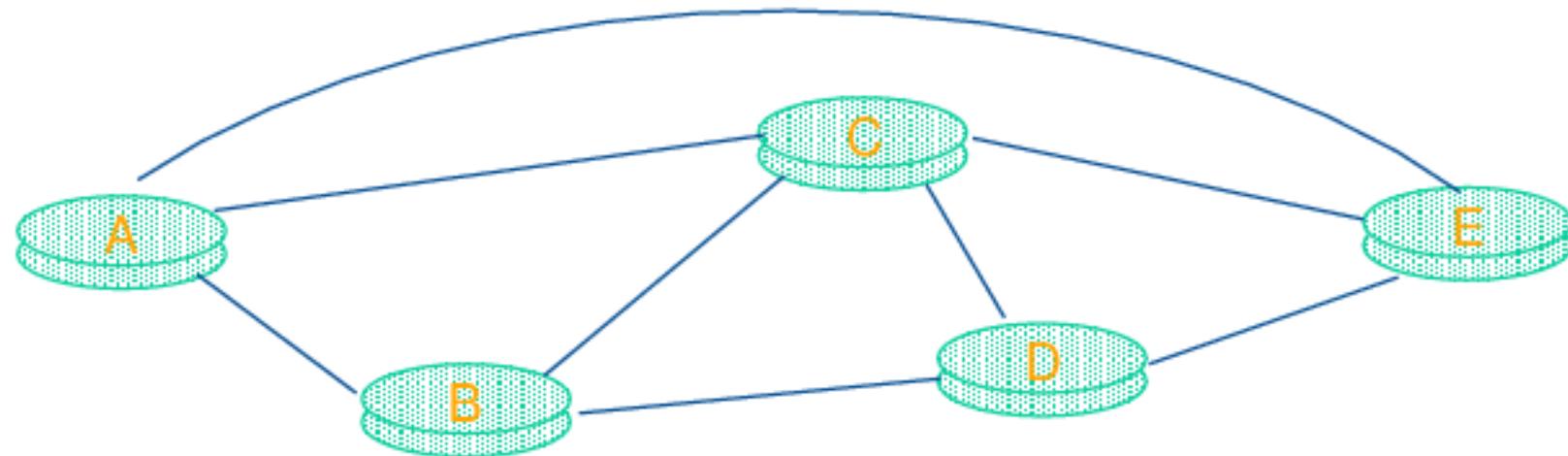


Grafos: abstração para representar enlaces e roteadores



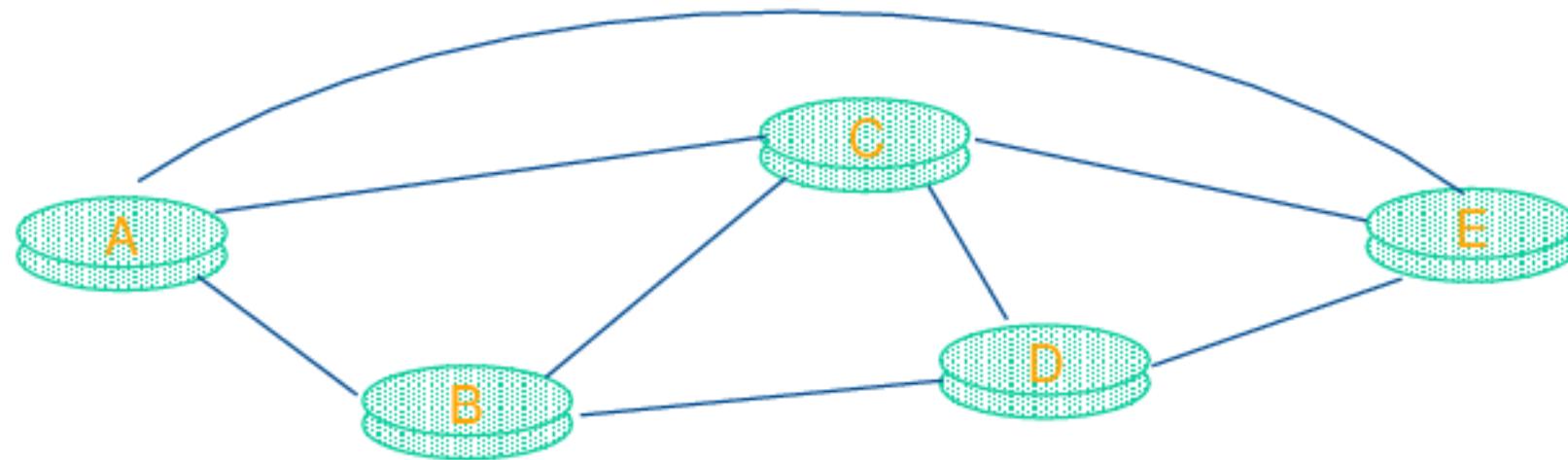
- Grafo = (N, E)

Grafos: abstração para representar enlaces e roteadores



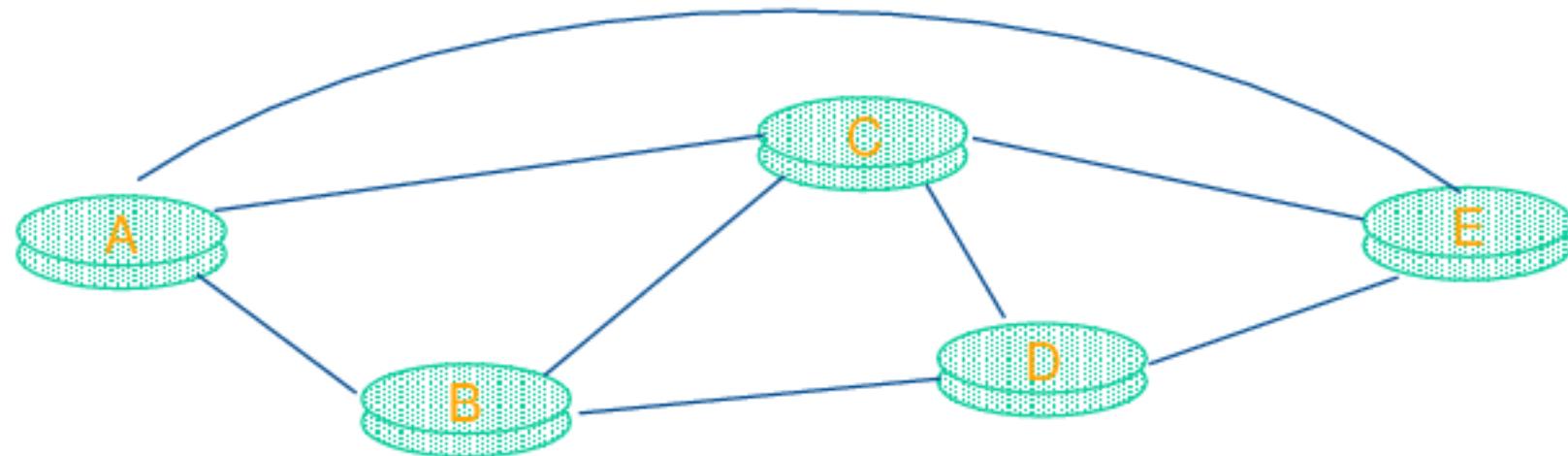
- Grafo = (N,E)
- N = conjunto de roteadores = {A,B,C,D,E}

Grafos: abstração para representar enlaces e roteadores



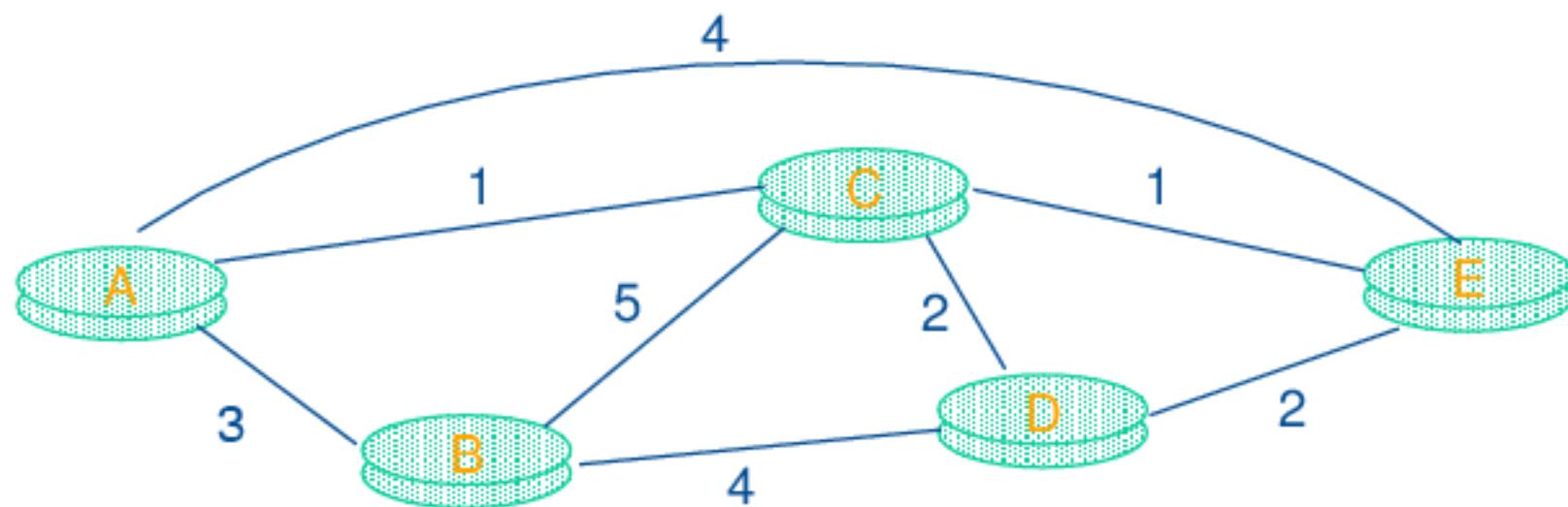
- Grafo = (N, E)
- N = conjunto de roteadores = {A,B,C,D,E}
- E = conjunto de enlaces = {(A,B),(A,C),(A,E),(B,C),(B,D),(C,D),(C,E),(D,E)}

Grafos: abstração para representar enlaces e roteadores

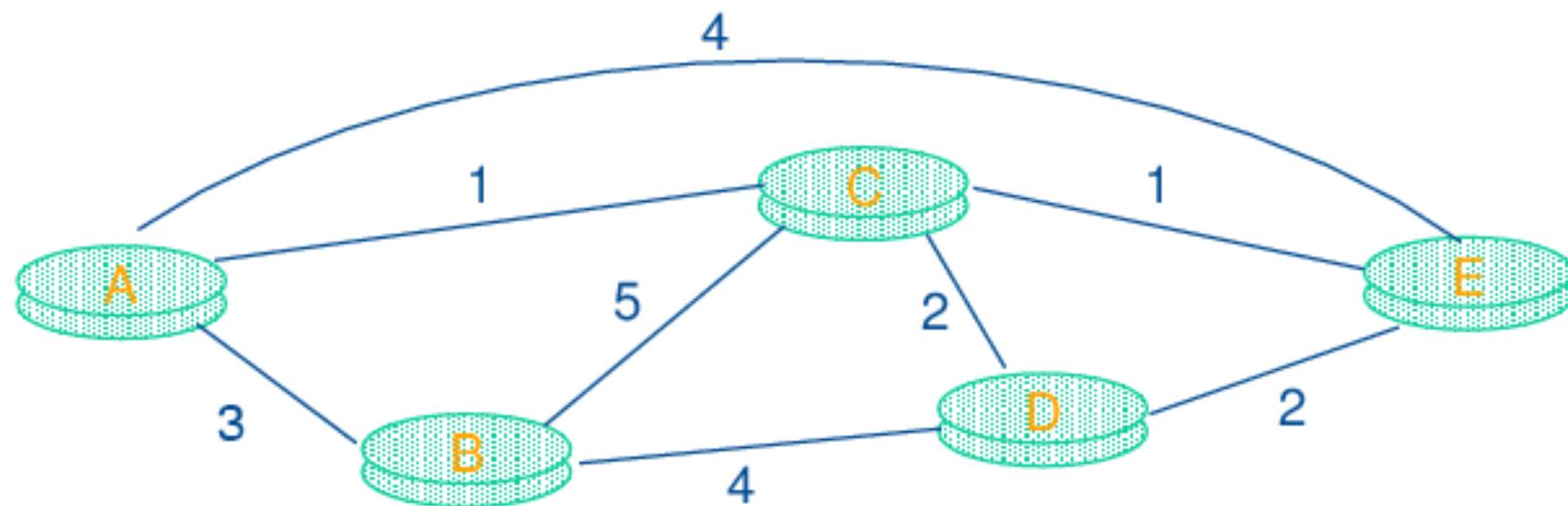


- Grafo = (N, E)
- N = conjunto de roteadores = {A,B,C,D,E}
- E = conjunto de enlaces = {(A,B),(A,C),(A,E),(B,C),(B,D),(C,D),(C,E),(D,E)}
- Usado também para representar arquiteturas P2P: N é o conjunto de pares e E é o conjunto de conexões TCP

Grafos: abstração para representar enlaces e roteadores

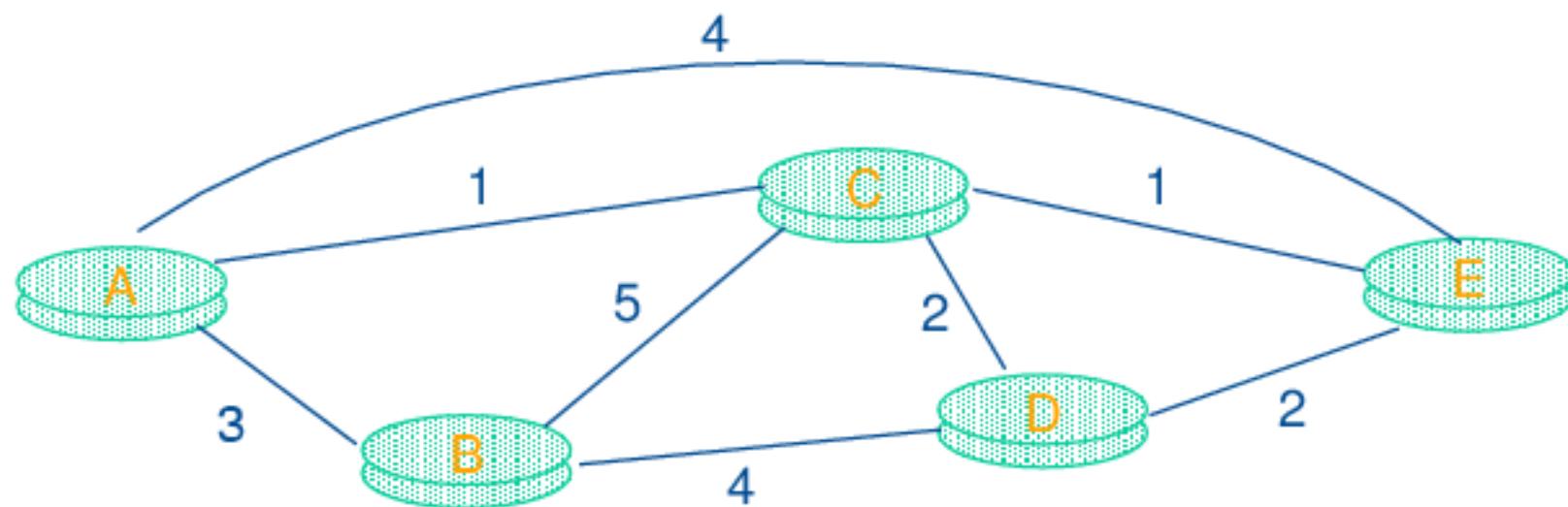


Grafos: abstração para representar enlaces e roteadores



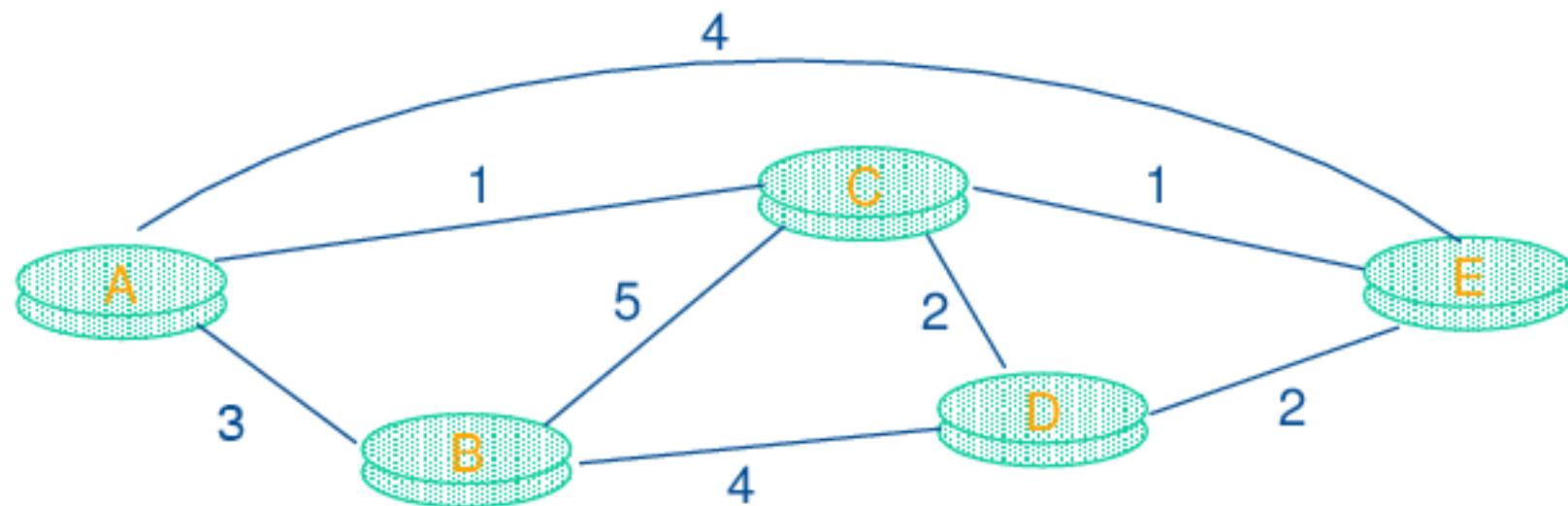
- c(A,B) = 3 => custo do enlace (A,B), custo representa, por exemplo, o grau de congestionamento ou a capacidade do enlace

Grafos: abstração para representar enlaces e roteadores



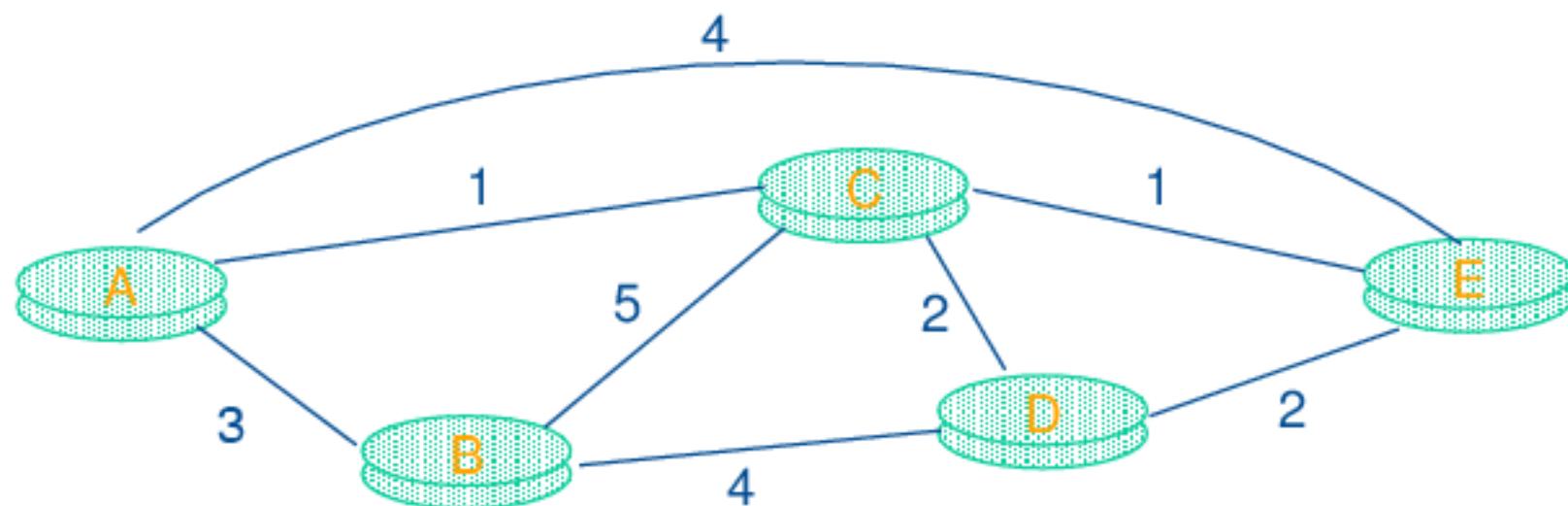
- $c(A,B) = 3 \Rightarrow$ custo do enlace (A,B) , custo representa, por exemplo, o grau de congestionamento ou a capacidade do enlace
- custo do caminho $(A,E) = c(A,E) = c(A,B) + c(B,C) + c(C,D) + c(D,E) = 12$

Grafos: abstração para representar enlaces e roteadores



- $c(A,B) = 3 \Rightarrow$ custo do enlace (A,B), custo representa, por exemplo, o grau de congestionamento ou a capacidade do enlace
- custo do caminho (A,E) = $c(A,E) = c(A,B) + c(B,C) + c(C,D) + c(D,E) = 12$
- Pergunta: Qual o caminho de menor custo de A até E ?

Grafos: abstração para representar enlaces e roteadores



- $c(A,B) = 3 \Rightarrow$ custo do enlace (A,B), custo representa, por exemplo, o grau de congestionamento ou a capacidade do enlace
- custo do caminho (A,E) = $c(A,E) = c(A,B) + c(B,C) + c(C,D) + c(D,E) = 12$
- Pergunta: Qual o caminho de menor custo de A até E ?
- Algoritmos de roteamento: algoritmos que encontram o caminho de menor custo

Classificação dos algoritmos de roteamento

Classificação dos algoritmos de roteamento

→ Global

Classificação dos algoritmos de roteamento

→ Global

- roteadores possuem informação da topologia da rede

Classificação dos algoritmos de roteamento

→ Global

- roteadores possuem informação da topologia da rede
- algoritmos da classe *link-state* (link state algorithms)

Classificação dos algoritmos de roteamento

→ Global

- roteadores possuem informação da topologia da rede
- algoritmos da classe *link-state* (link state algorithms)

→ Decentralizado

Classificação dos algoritmos de roteamento

→ Global

- roteadores possuem informação da topologia da rede
- algoritmos da classe *link-state* (link state algorithms)

→ Decentralizado

- roteadores conhecem seus vizinhos e o custo até eles

Classificação dos algoritmos de roteamento

→ Global

- roteadores possuem informação da topologia da rede
- algoritmos da classe *link-state* (link state algorithms)

→ Decentralizado

- roteadores conhecem seus vizinhos e o custo até eles
- processo iterativo para cálculo das rotas através da troca de informação entre os vizinhos

Classificação dos algoritmos de roteamento

→ Global

- roteadores possuem informação da topologia da rede
- algoritmos da classe *link-state* (link state algorithms)

→ Decentralizado

- roteadores conhecem seus vizinhos e o custo até eles
- processo iterativo para cálculo das rotas através da troca de informação entre os vizinhos
- algoritmos da classe *distance-vetor* (distance vector algorithms)

Classificação dos algoritmos de roteamento

Classificação dos algoritmos de roteamento

→ Estático

Classificação dos algoritmos de roteamento

→ Estático

- intervalo entre mudanças na topologia é grande

Classificação dos algoritmos de roteamento

→ Estático

- intervalo entre mudanças na topologia é grande
- atualizações das tabelas de roteamento ocorrem a intervalos grandes de tempo

Classificação dos algoritmos de roteamento

→ Estático

- intervalo entre mudanças na topologia é grande
- atualizações das tabelas de roteamento ocorrem a intervalos grandes de tempo

→ Dinâmico

Classificação dos algoritmos de roteamento

→ Estático

- intervalo entre mudanças na topologia é grande
- atualizações das tabelas de roteamento ocorrem a intervalos grandes de tempo

→ Dinâmico

- mudanças na topologia ocorrem a intervalos curtos de tempo

Classificação dos algoritmos de roteamento

→ Estático

- intervalo entre mudanças na topologia é grande
- atualizações das tabelas de roteamento ocorrem a intervalos grandes de tempo

→ Dinâmico

- mudanças na topologia ocorrem a intervalos curtos de tempo
- custos dos enlaces mudam rapidamente

Classificação dos algoritmos de roteamento

→ Estático

- intervalo entre mudanças na topologia é grande
- atualizações das tabelas de roteamento ocorrem a intervalos grandes de tempo

→ Dinâmico

- mudanças na topologia ocorrem a intervalos curtos de tempo
- custos dos enlaces mudam rapidamente
- atualizações periódicas na tabela de roteamento

Algoritmo de roteamento do tipo *link-state*

Algoritmo de roteamento do tipo *link-state*

→ Baseado no algoritmo de Dijkstra

Algoritmo de roteamento do tipo *link-state*

- Baseado no algoritmo de Dijkstra
- Todos os roteadores conhecem a topologia da rede e os custos dos enlaces

Algoritmo de roteamento do tipo *link-state*

- Baseado no algoritmo de Dijkstra
- Todos os roteadores conhecem a topologia da rede e os custos dos enlaces
 - as informações de topologia e custos são enviadas usando um algoritmo de broadcast

Algoritmo de roteamento do tipo *link-state*

- Baseado no algoritmo de Dijkstra
- Todos os roteadores conhecem a topologia da rede e os custos dos enlaces
 - as informações de topologia e custos são enviadas usando um algoritmo de broadcast
 - os roteadores possuem a mesma informação

Algoritmo de roteamento do tipo *link-state*

- Baseado no algoritmo de Dijkstra
- Todos os roteadores conhecem a topologia da rede e os custos dos enlaces
 - as informações de topologia e custos são enviadas usando um algoritmo de broadcast
 - os roteadores possuem a mesma informação
- Roteadores computam o menor custo a partir dele (origem) até todos os outros nós da rede

Algoritmo de roteamento do tipo *link-state*

- Baseado no algoritmo de Dijkstra
- Todos os roteadores conhecem a topologia da rede e os custos dos enlaces
 - as informações de topologia e custos são enviadas usando um algoritmo de broadcast
 - os roteadores possuem a mesma informação
- Roteadores computam o menor custo a partir dele (origem) até todos os outros nós da rede
 - este cálculo fornece a tabela de roteamento para este roteador

Algoritmo de roteamento do tipo *link-state*

- Baseado no algoritmo de Dijkstra
- Todos os roteadores conhecem a topologia da rede e os custos dos enlaces
 - as informações de topologia e custos são enviadas usando um algoritmo de broadcast
 - os roteadores possuem a mesma informação
- Roteadores computam o menor custo a partir dele (origem) até todos os outros nós da rede
 - este cálculo fornece a tabela de roteamento para este roteador
- Algoritmo iterativo: após k iterações é conhecido o menor caminho para k destinos na rede

Algoritmo de Dijkstra

- 1 Inicialização
- 2 $N' = \{u\}$
- 3 faça para todos os nós v
- 4 se v é adjacente a u
- 5 então $D(v) = c(u,v)$
- 6 senão $D(v) = \text{infinito}$
- 7
- 8 Loop
- 9 encontre w não pertencente a N' tal que $D(w)$ é mínimo
- 10 adicione w a N'
- 11 atualize $D(v)$ para todo v adjacente a w e que não pertence a N' :
- 12 $D(v) = \min(D(v), D(w) + c(w,v))$
- 13 até que todos os nós estejam em N'

Algoritmo de Dijkstra

- 1 Inicialização
 2 $N = \{u\}$
 3 faça para todos os nós v
 4 se v é adjacente a u
 5 então $D(v) = c(u,v)$
 6 senão $D(v) = \text{infinito}$
 7
 8 Loop
 9 encontre w não pertencente a N' tal que $D(w)$ é mínimo
 10 adicione w a N'
 11 atualize $D(v)$ para todo v adjacente a w e que não pertence a N' :
 12 $D(v) = \min(D(v), D(w) + c(w,v))$
 13 até que todos os nós estejam em N'

Algoritmo de Dijkstra

- 1 Inicialização
 2 $N = \{u\}$
 3 faça para todos os nós v → conjunto de nós para os quais o menor custo já foi calculado
 4 se v é adjacente a u
 5 então $D(v) = c(u,v)$
 6 senão $D(v) = \text{infinito}$
 7
 8 Loop
 9 encontre w não pertencente a N' tal que $D(w)$ é mínimo
 10 adicione w a N'
 11 atualize $D(v)$ para todo v adjacente a w e que não pertence a N' :
 12 $D(v) = \min(D(v), D(w) + c(w,v))$
 13 até que todos os nós estejam em N'

Algoritmo de Dijkstra

- 1 Inicialização
 2 $N = \{u\}$
 3 faça para todos os nós v → nó origem
 4 se v é adjacente a u
 5 então $D(v) = c(u,v)$ → custo do nó u até o nó v
 6 senão $D(v) = \text{infinito}$
 7
 8 Loop
 9 encontre w não pertencente a N' tal que $D(w)$ é mínimo
 10 adicione w a N'
 11 atualize $D(v)$ para todo v adjacente a w e que não pertence a N' :
 12 $D(v) = \min(D(v), D(w) + c(w,v))$
 13 até que todos os nós estejam em N'

Algoritmo de Dijkstra

- 1 Inicialização
 2 $N = \{u\}$ → conjunto de nós para os quais o menor custo já foi calculado
 3 faça para todos os nós v → nó origem
 4 se v é adjacente a u
 5 então $D(v) = c(u,v)$ → custo do nó u até o nó v
 6 senão $D(v) = \text{infinito}$
 7 → custo do nó origem até o nó v
 8 Loop
 9 encontre w não pertencente a N' tal que $D(w)$ é mínimo
 10 adicione w a N'
 11 atualize $D(v)$ para todo v adjacente a w e que não pertence a N' :
 12 $D(v) = \min(D(v), D(w) + c(w,v))$
 13 até que todos os nós estejam em N'

Algoritmo de Dijkstra

- 1 Inicialização
 2 $N = \{u\}$ → conjunto de nós para os quais o menor custo já foi calculado
 3 faça para todos os nós v → nó origem
 4 se v é adjacente a u
 5 então $D(v) = c(u,v)$ → custo do nó u até o nó v
 6 senão $D(v) = \text{infinito}$
 7 → custo do nó origem até o nó v
 8 Loop
 9 encontre w não pertencente a N' tal que $D(w)$ é mínimo
 10 adicione w a N'
 11 atualize $D(v)$ para todo v adjacente a w e que não pertence a N' :
 12 $D(v) = \min(D(v), D(w) + c(w,v))$
 13 até que todos os nós estejam em N'
 novo
 custo

Algoritmo de Dijkstra

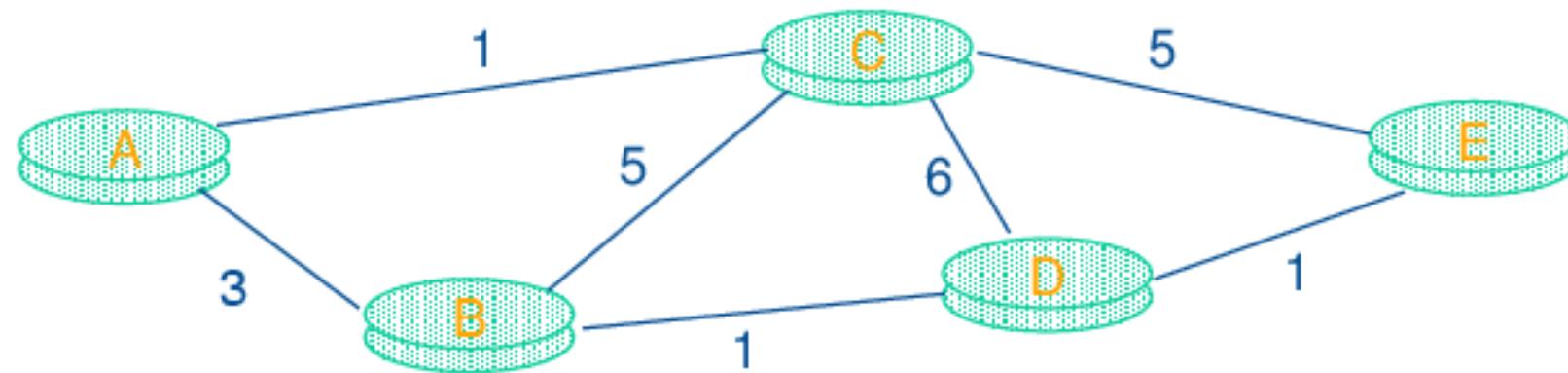
- 1 Inicialização
 2 $N = \{u\}$ → conjunto de nós para os quais o menor custo já foi calculado
 3 faça para todos os nós v → nó origem
 4 se v é adjacente a u
 5 então $D(v) = c(u,v)$ → custo do nó u até o nó v
 6 senão $D(v) = \text{infinito}$
 7 → custo do nó origem até o nó v
 8 Loop
 9 encontre w não pertencente a N' tal que $D(w)$ é mínimo
 10 adicione w a N'
 11 atualize $D(v)$ para todo v adjacente a w e que não pertence a N' :
 12 $D(v) = \min(D(v), D(w) + c(w,v))$
 13 até que todos os nós estejam em N'
 novo custo anterior
 custo

Algoritmo de Dijkstra

- 1 Inicialização
 2 $N = \{u\}$ → conjunto de nós para os quais o menor custo já foi calculado
 3 faça para todos os nós v → nó origem
 4 se v é adjacente a u
 5 então $D(v) = c(u,v)$ → custo do nó u até o nó v
 6 senão $D(v) = \text{infinito}$
 7 → custo do nó origem até o nó v
 8 Loop
 9 encontre w não pertencente a N' tal que $D(w)$ é mínimo
 10 adicione w a N'
 11 atualize $D(v)$ para todo v adjacente a w e que não pertence a N' :
 12 $D(v) = \min(D(v), D(w) + c(w,v))$ → novo caminho de u até w somado ao custo de w até v
 13 até que todos os nós estejam em N'
 novo custo anterior
 custo

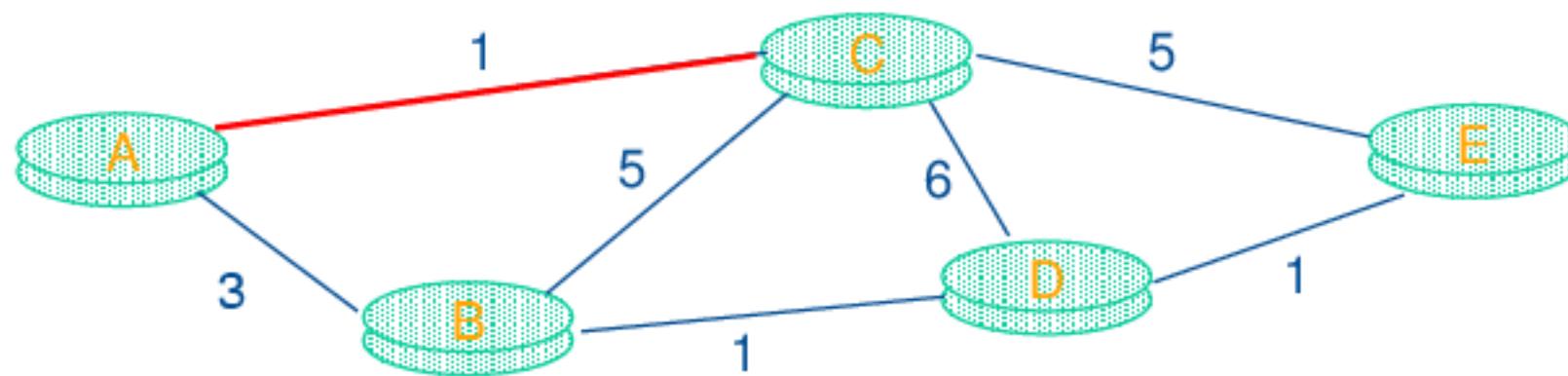
Algoritmo de Dijkstra: exemplo (1)

Passo	N'	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$
0	{A}	3,A	1,A	infinito	infinito



Algoritmo de Dijkstra: exemplo (1)

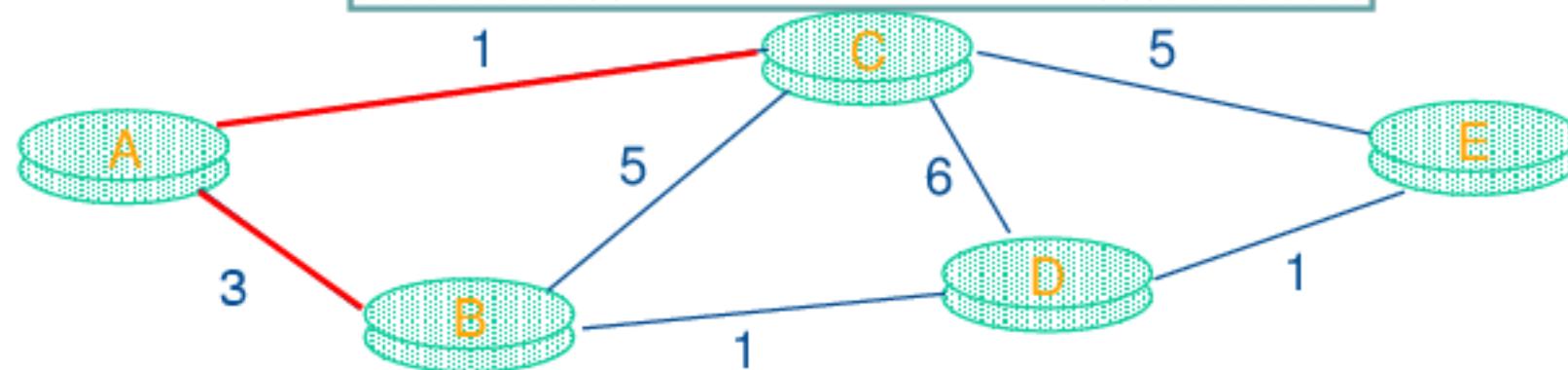
Passo	N'	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$
0	{A}	3,A	1,A	infinito	infinito
1	{A,C}	3,A	-	7,C	6,C



Algoritmo de Dijkstra: exemplo (1)

Passo	N'	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$
0	{A}	3,A	1,A	infinito	infinito
1	{A,C}	3,A	-	7,C	6,C
2	{A,C,B}	-	-	4,B	6,C

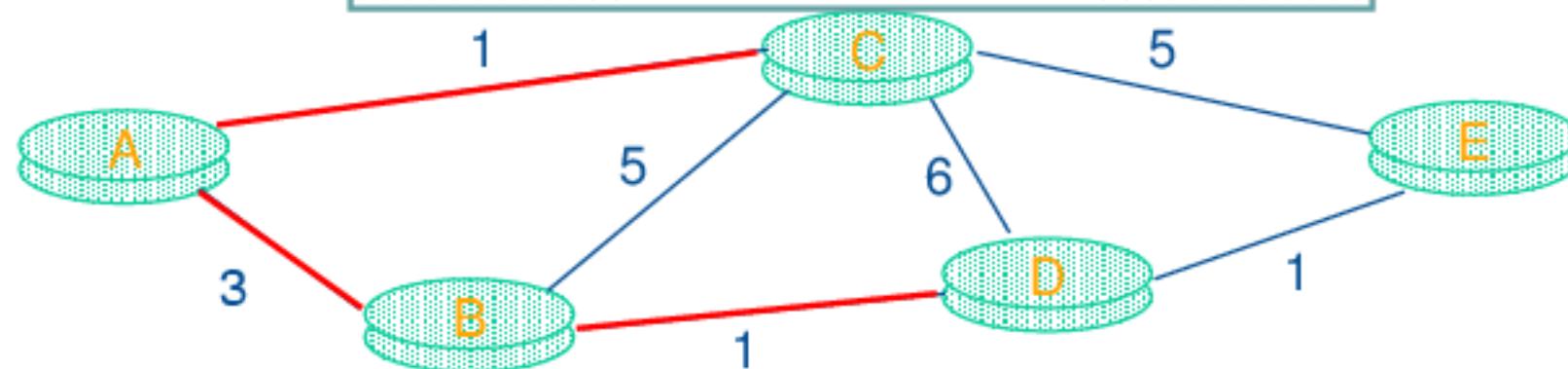
Nesta iteração $D(D)$ e $P(D)$ serão atualizados,
pois $D(B) + c(B,D) = 4 < 7$,
o valor de $D(D)$ na iteração anterior



Algoritmo de Dijkstra: exemplo (1)

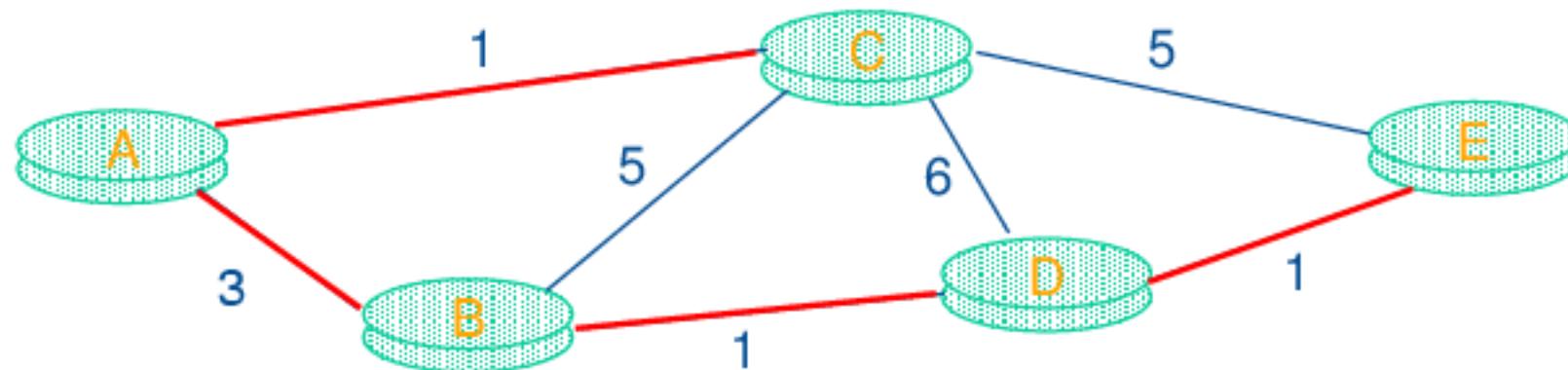
Passo	N'	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$
0	{A}	3,A	1,A	infinito	infinito
1	{A,C}	3,A	-	7,C	6,C
2	{A,C,B}	-	-	4,B	6,C
3	{A,C,B,D}	-	-	-	5,D

Nesta iteração $D(E)$ e $p(E)$ serão atualizados, pois $D(D) + c(D,E) = 5 < 6$, o valor de $D(E)$ na iteração anterior



Algoritmo de Dijkstra: exemplo (1)

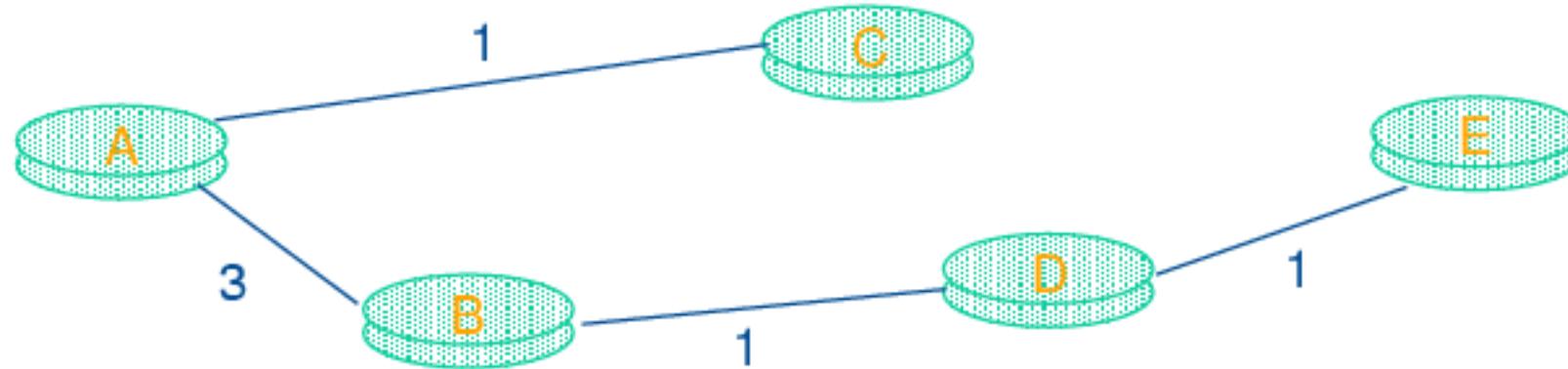
Passo	N'	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$
0	{A}	3,A	1,A	infinito	infinito
1	{A,C}	3,A	-	7,C	6,C
2	{A,C,B}	-	-	4,B	6,C
3	{A,C,B,D}	-	-	-	5,D
4	{A,C,B,D,E}	-	-	-	-



Algoritmo de Dijkstra: exemplo (2)

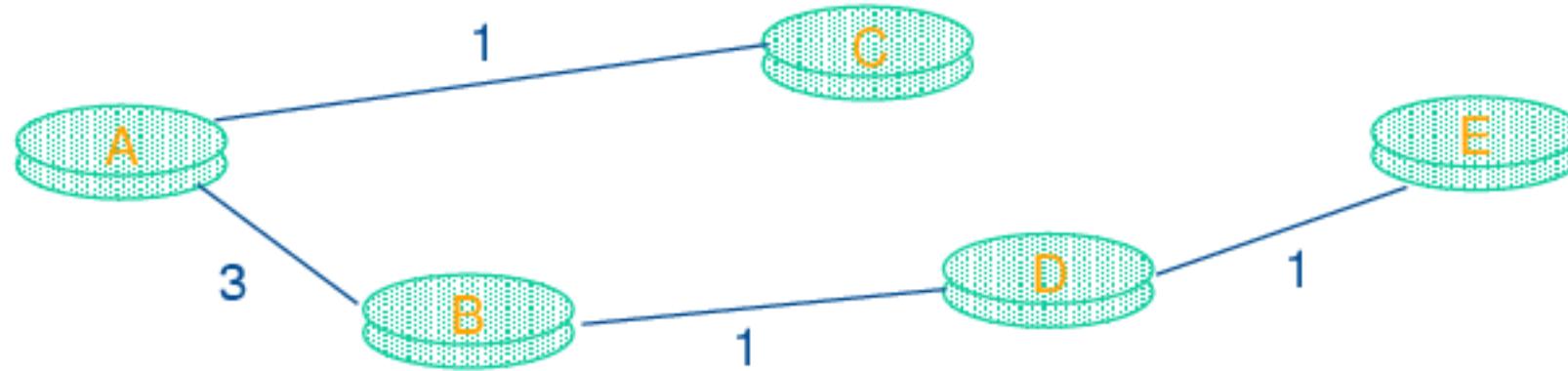
Algoritmo de Dijkstra: exemplo (2)

→ Árvore de menor caminho a partir do nó origem A



Algoritmo de Dijkstra: exemplo (2)

→ Árvore de menor caminho a partir do nó origem A



→ Tabela de roteamento do nó A

Destino	Enlace de saída
B	(A,B)
C	(A,C)
D	(A,B)
E	(A,B)

Algoritmos para broadcast

Algoritmos para broadcast

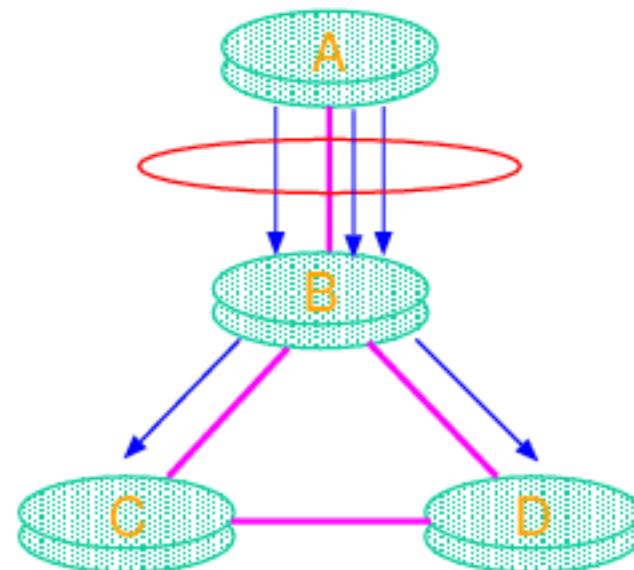
→ **Objetivo:** Entregar pacotes de um nó fonte para todos os nós da rede

Algoritmos para broadcast

- **Objetivo:** Entregar pacotes de um nó fonte para todos os nós da rede
- **Forma mais simples:** Enviar N cópias do pacote para os N nós destinos da rede

Algoritmos para broadcast

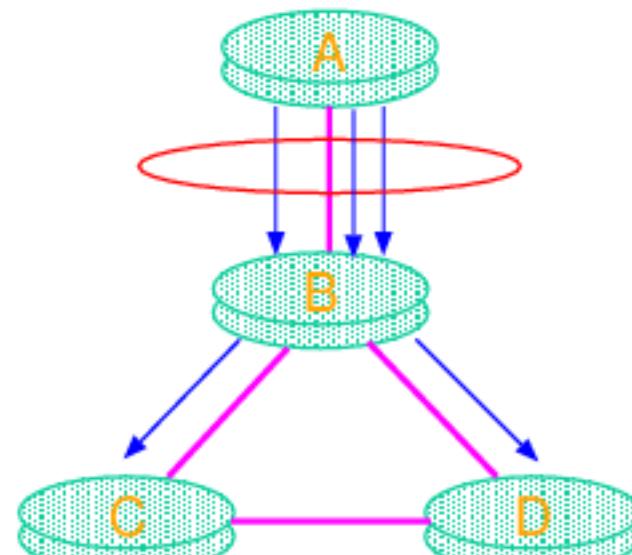
- **Objetivo:** Entregar pacotes de um nó fonte para todos os nós da rede
 - **Forma mais simples:** Enviar N cópias do pacote para os N nós destinos da rede
- Problema:** duplicação pela fonte



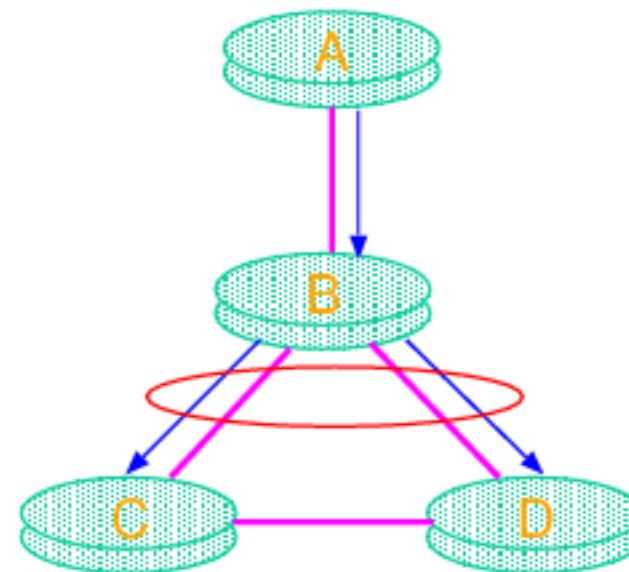
Algoritmos para broadcast

- **Objetivo:** Entregar pacotes de um nó fonte para todos os nós da rede
- **Forma mais simples:** Enviar N cópias do pacote para os N nós destinos da rede

Problema: duplicação pela fonte



Forma mais eficiente:
duplicação pela rede



Algoritmos para broadcast

Algoritmos para broadcast

- Endereço broadcast é usado para indicar que o pacote deve ser enviado para todos os nós

Algoritmos para broadcast

- Endereço broadcast é usado para indicar que o pacote deve ser enviado para todos os nós
- Quando um nó recebe um pacote com o endereço broadcast, usa um determinado algoritmo para enviá-lo

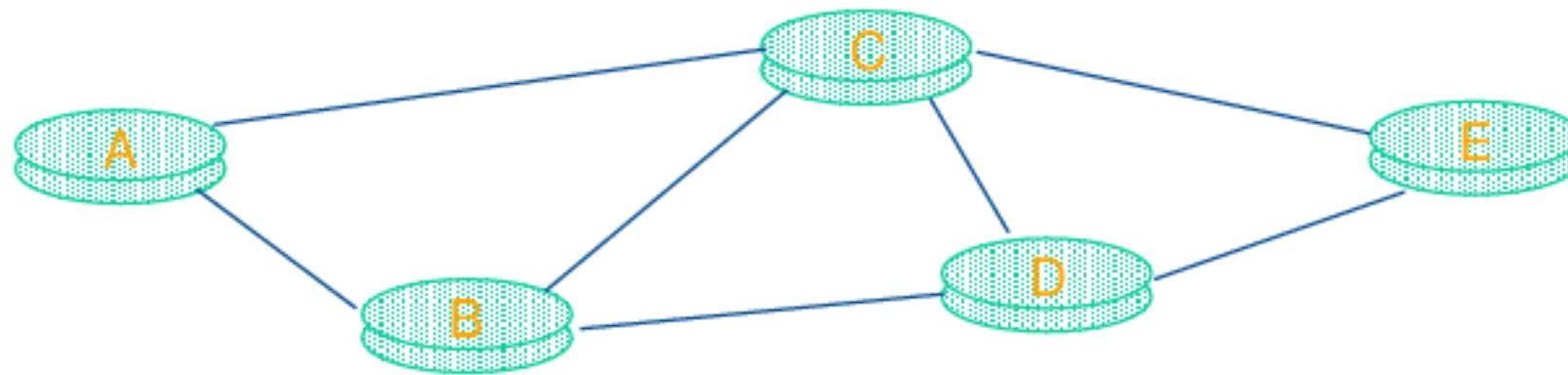
Algoritmos para broadcast

- Endereço broadcast é usado para indicar que o pacote deve ser enviado para todos os nós
- Quando um nó recebe um pacote com o endereço broadcast, usa um determinado algoritmo para enviá-lo
- **Aplicação importante:** Algoritmos de roteamento do tipo link-state usam um algoritmo de broadcast para disseminar informações de topologia e custos

Algoritmos para broadcast

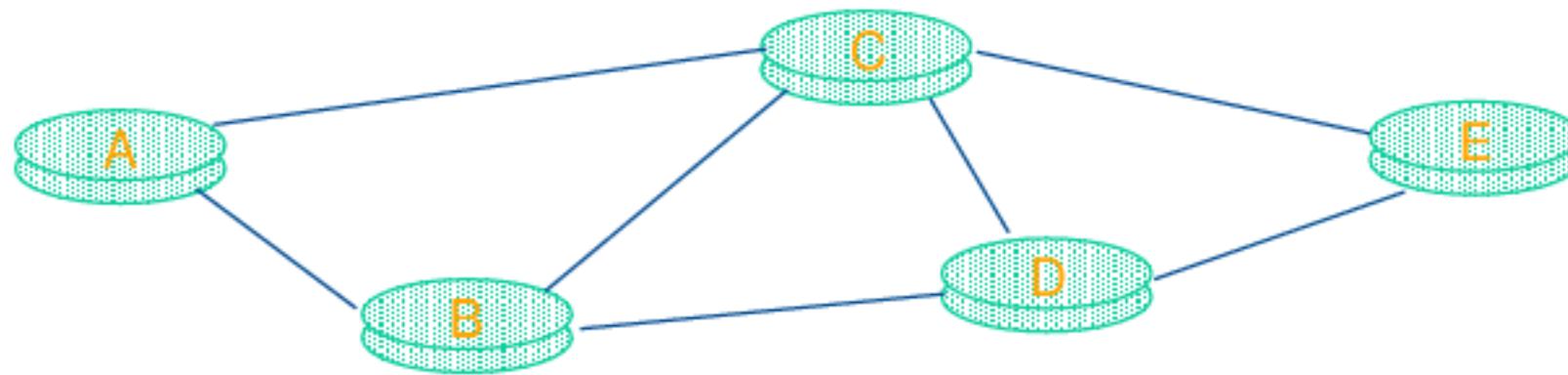
- Endereço broadcast é usado para indicar que o pacote deve ser enviado para todos os nós
- Quando um nó recebe um pacote com o endereço broadcast, usa um determinado algoritmo para enviá-lo
- **Aplicação importante:** Algoritmos de roteamento do tipo link-state usam um algoritmo de broadcast para disseminar informações de topologia e custos
- Estudaremos dois algoritmos importantes: flooding e repasse pelo caminho inverso (Reverse path forwarding - RPF)

Algoritmos para broadcast: flooding



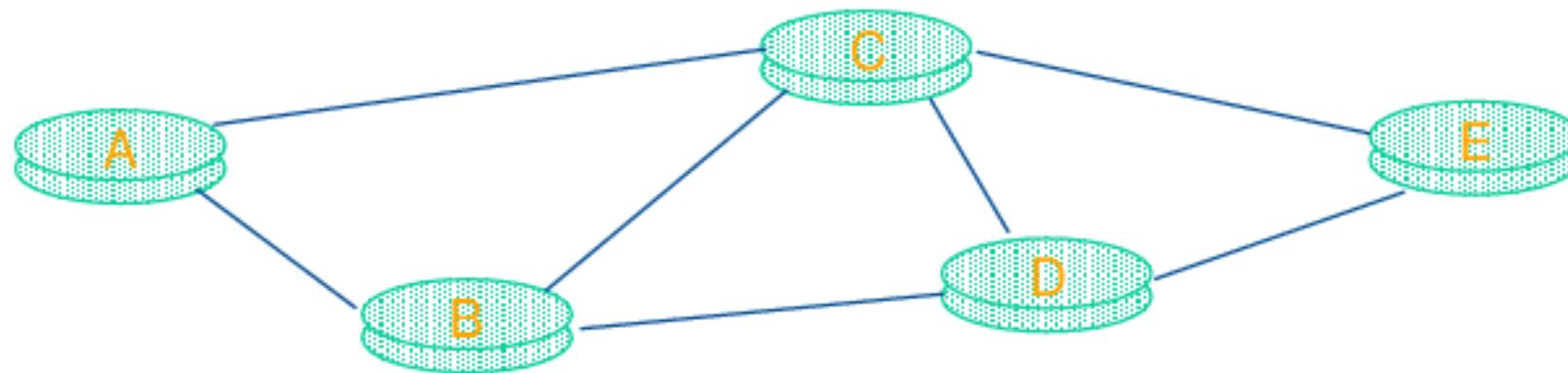
Algoritmos para broadcast: flooding

- Se o nó não enviou o pacote anteriormente, ele transmite o pacote para todos os vizinhos, exceto vizinho do qual recebeu o pacote; caso já tenha enviado o pacote, o descarta



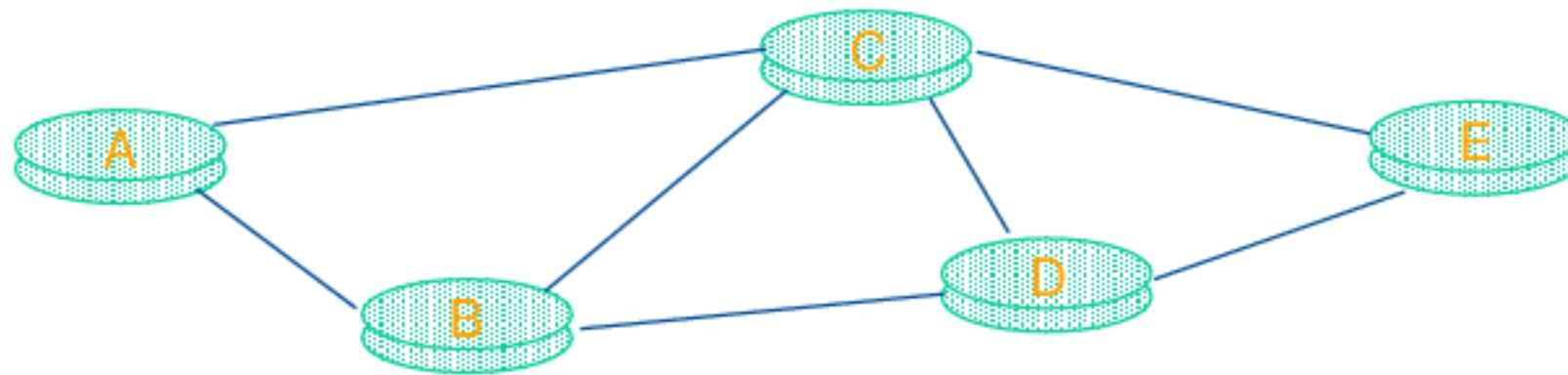
Algoritmos para broadcast: flooding

- Se o nó não enviou o pacote anteriormente, ele transmite o pacote para todos os vizinhos, exceto vizinho do qual recebeu o pacote; caso já tenha enviado o pacote, o descarta
- Como o nó sabe se já transmitiu um pacote ?



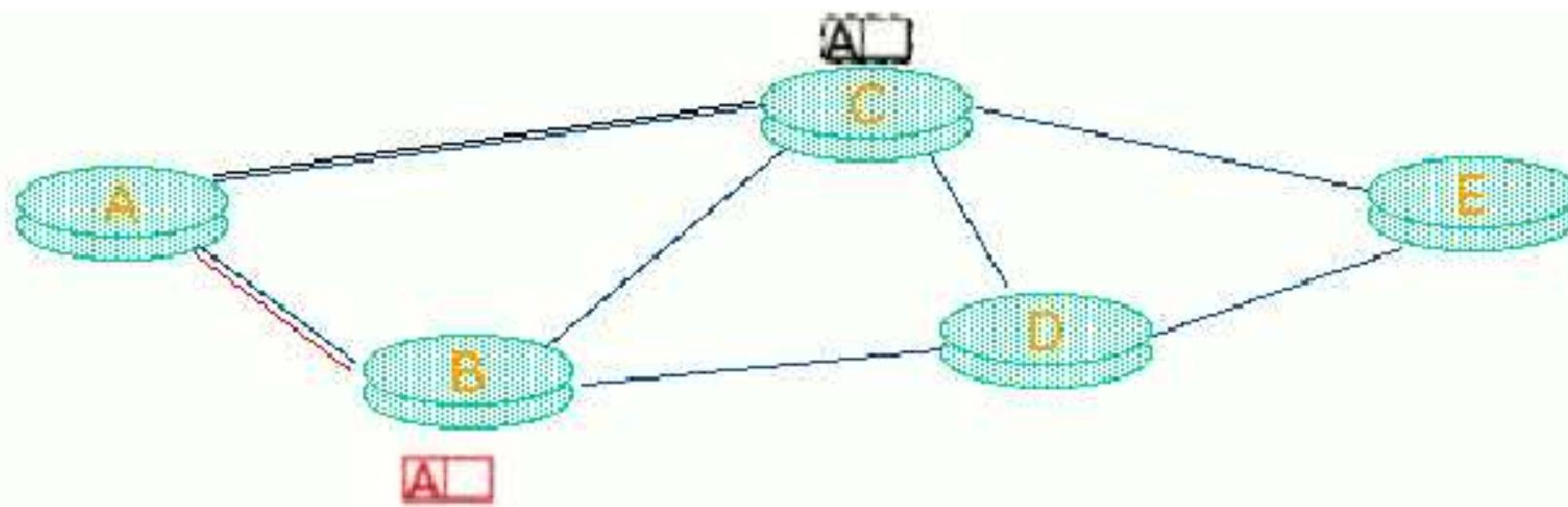
Algoritmos para broadcast: flooding

- Se o nó não enviou o pacote anteriormente, ele transmite o pacote para todos os vizinhos, exceto vizinho do qual recebeu o pacote; caso já tenha enviado o pacote, o descarta
- Como o nó sabe se já transmitiu um pacote ?
- Nó armazena a identificação do pacote, ID do nó fonte e número de sequência, para saber quando transmitir



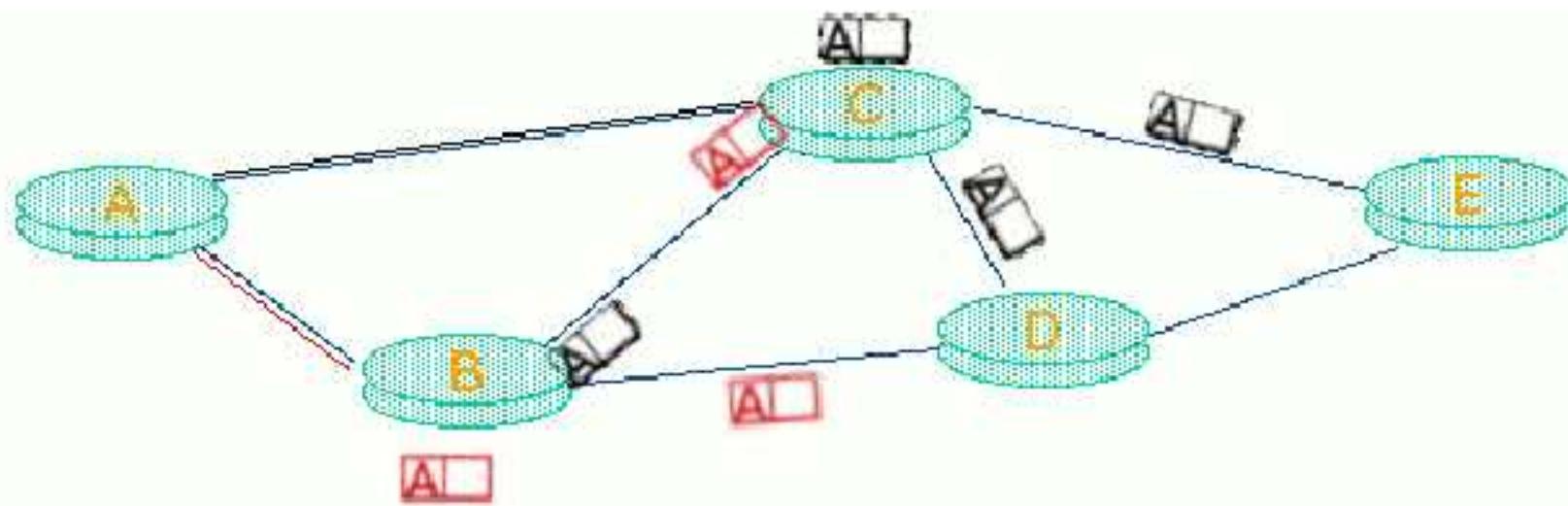
Algoritmos para broadcast: flooding

- Se o nó não enviou o pacote anteriormente, ele transmite o pacote para todos os vizinhos, exceto vizinho do qual recebeu o pacote; caso já tenha enviado o pacote, o descarta
- Como o nó sabe se já transmitiu um pacote ?
- Nó armazena a identificação do pacote, ID do nó fonte e número de sequência, para saber quando transmitir



Algoritmos para broadcast: flooding

- Se o nó não enviou o pacote anteriormente, ele transmite o pacote para todos os vizinhos, exceto vizinho do qual recebeu o pacote; caso já tenha enviado o pacote, o descarta
- Como o nó sabe se já transmitiu um pacote ?
- Nó armazena a identificação do pacote, ID do nó fonte e número de sequência, para saber quando transmitir



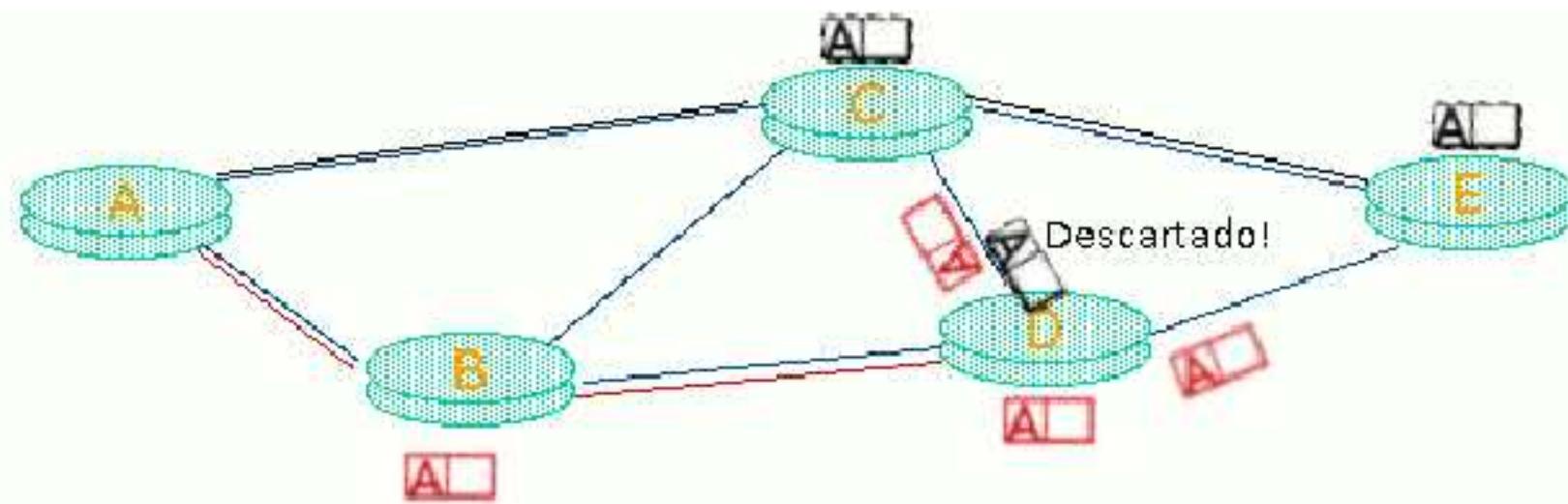
Algoritmos para broadcast: flooding

- Se o nó não enviou o pacote anteriormente, ele transmite o pacote para todos os vizinhos, exceto vizinho do qual recebeu o pacote; caso já tenha enviado o pacote, o descarta
- Como o nó sabe se já transmitiu um pacote ?
- Nó armazena a identificação do pacote, ID do nó fonte e número de sequência, para saber quando transmitir



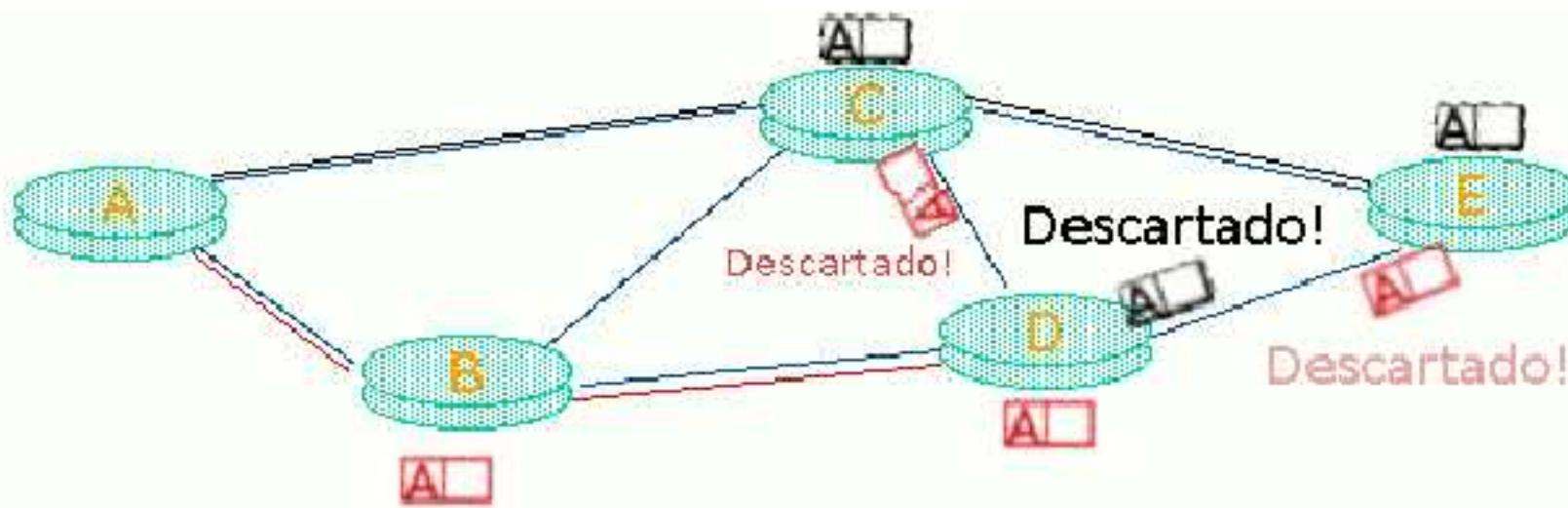
Algoritmos para broadcast: flooding

- Se o nó não enviou o pacote anteriormente, ele transmite o pacote para todos os vizinhos, exceto vizinho do qual recebeu o pacote; caso já tenha enviado o pacote, o descarta
- Como o nó sabe se já transmitiu um pacote ?
- Nó armazena a identificação do pacote, ID do nó fonte e número de sequência, para saber quando transmitir



Algoritmos para broadcast: flooding

- Se o nó não enviou o pacote anteriormente, ele transmite o pacote para todos os vizinhos, exceto vizinho do qual recebeu o pacote; caso já tenha enviado o pacote, o descarta
- Como o nó sabe se já transmitiu um pacote ?
- Nó armazena a identificação do pacote, ID do nó fonte e número de sequência, para saber quando transmitir



Algoritmos para broadcast: repasse pelo caminho reverso (Reverse path forwarding)

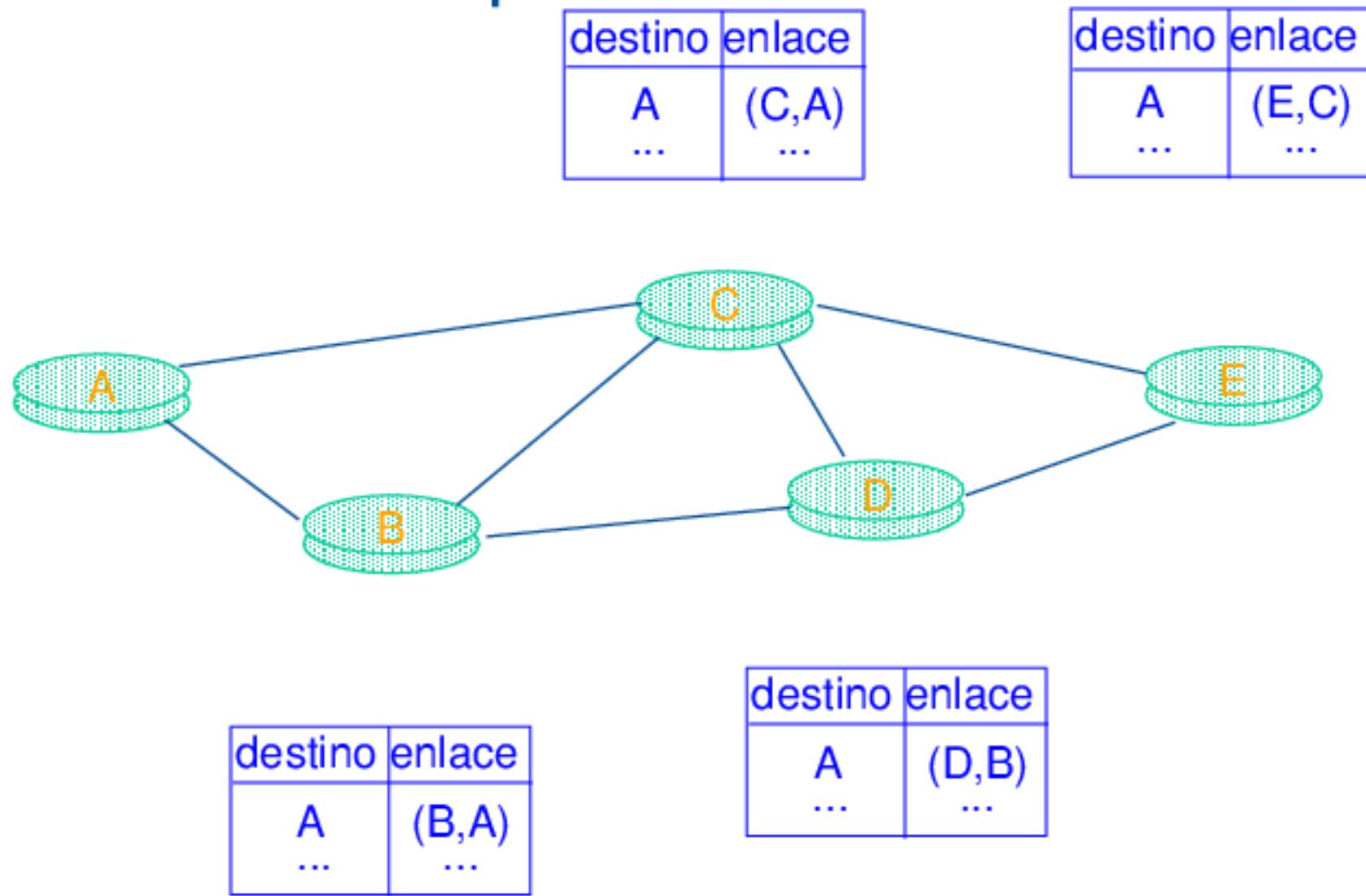
Algoritmos para broadcast: repasse pelo caminho reverso (Reverse path forwarding)

- Se o nó receber o pacote pelo enlace que ele usa para alcançar o nó fonte do pacote, ele transmite o pacote para todos os vizinhos, exceto para o vizinho do qual ele recebeu o pacote

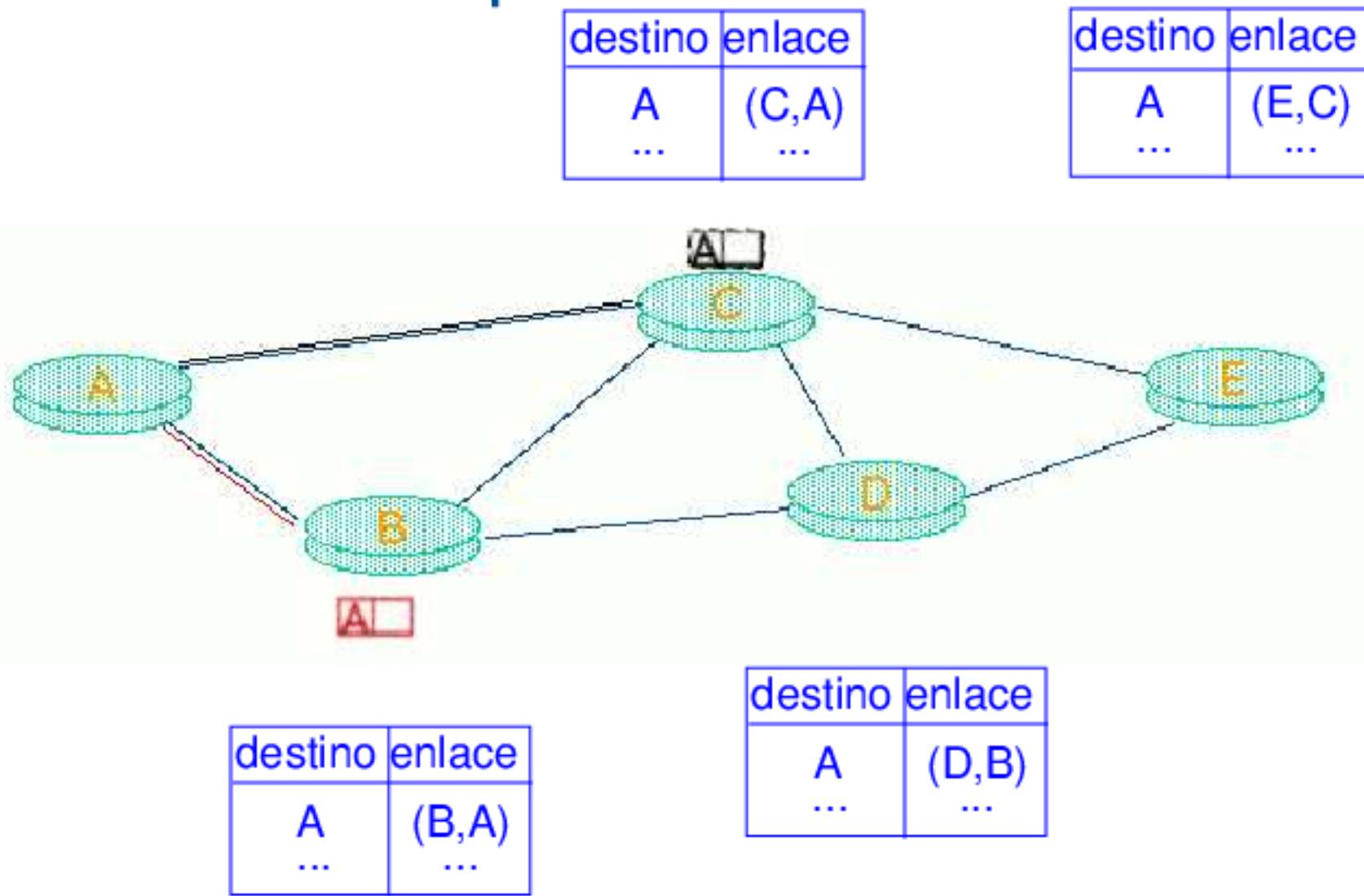
Algoritmos para broadcast: repasse pelo caminho reverso (Reverse path forwarding)

- Se o nó receber o pacote pelo enlace que ele usa para alcançar o nó fonte do pacote, ele transmite o pacote para todos os vizinhos, exceto para o vizinho do qual ele recebeu o pacote
- Caso contrário, descarta o pacote

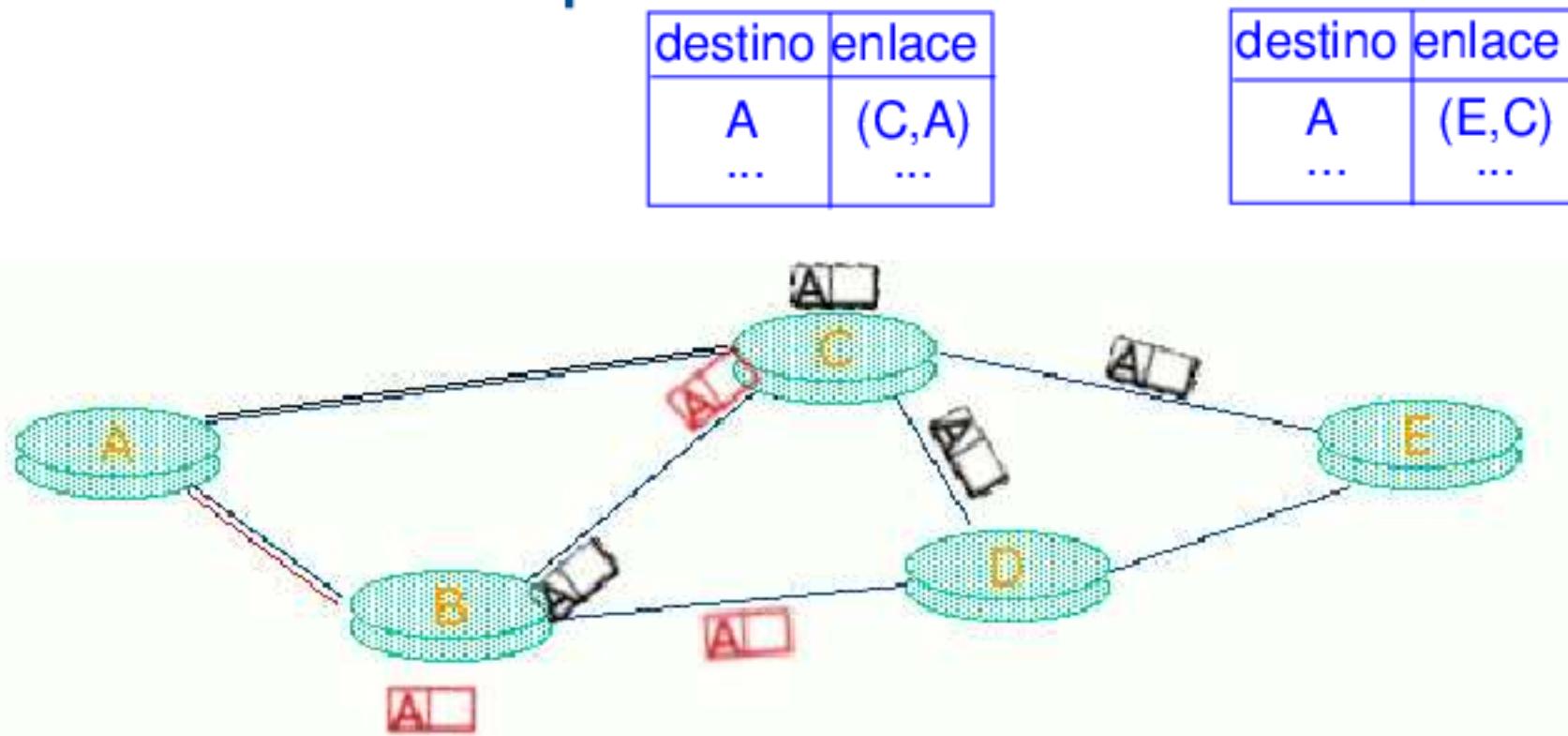
Algoritmo repasse pelo caminho reverso: exemplo



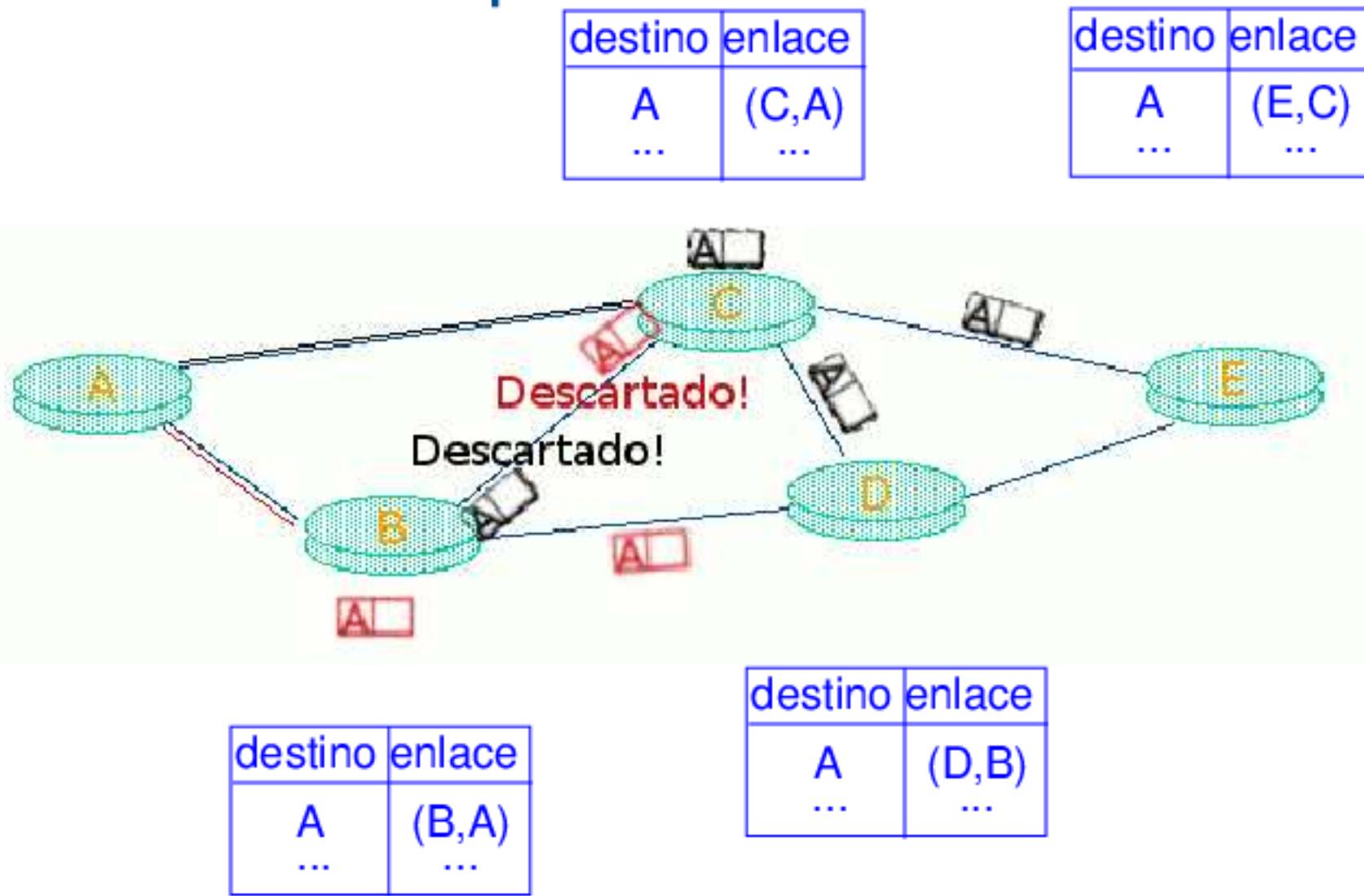
Algoritmo repasse pelo caminho reverso: exemplo



Algoritmo repasse pelo caminho reverso: exemplo



Algoritmo repasse pelo caminho reverso: exemplo



Algoritmo repasse pelo caminho reverso: exemplo



Algoritmo repasse pelo caminho reverso: exemplo

