CH-20
INTRO TO NETWORK ROUTING

Routing Algorithms

CONTENT

- Introduction
- Distance-Vector Routing
  - Bellman-Ford Equation.
  - Distance Vectors.
  - Distance-Vector Routing Algorithm.
- Link-State Routing
  - Link-State Database (LSDB).
  - Formation of Least-Cost Trees.
- Path-Vector Routing
  - Spanning Trees.
  - Creation of Spanning Trees.
  - Path-Vector Algorithm.

INTRODUCTION

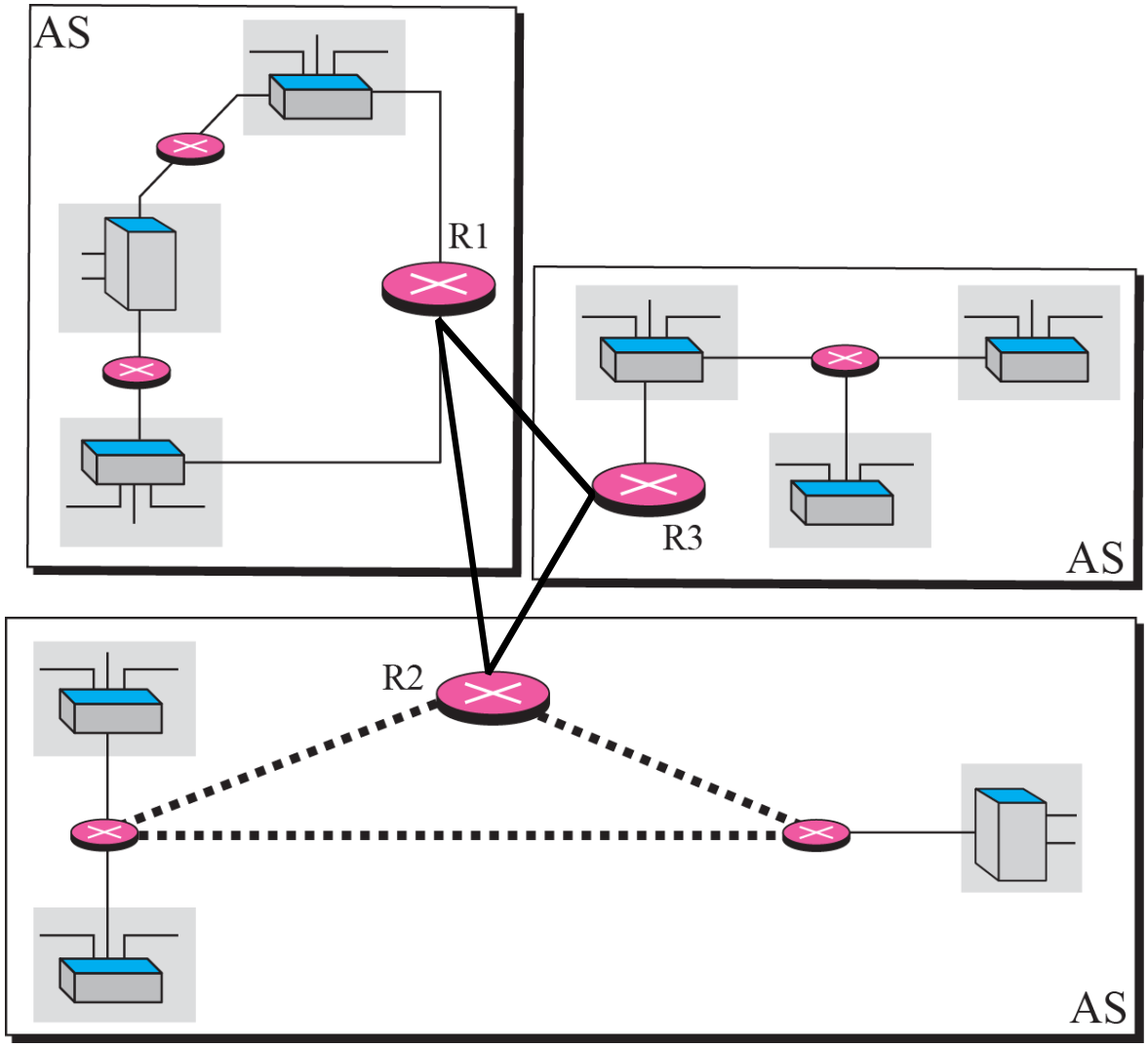
An internet is a combination of networks connected by routers. When a datagram goes from a source to a destination, it will probably pass through many routers until it reaches the router attached to the destination network.

INTER AND INTRA-DOMAIN ROUTING

Today, an internet can be so large that one routing protocol cannot handle the task of updating the routing tables of all routers. For this reason, an internet is divided into autonomous systems.

An autonomous system (AS) is a group of networks and routers under the authority of a single administration. Routing inside an autonomous system is called intra-domain routing. Routing between autonomous systems is called inter-domain routing

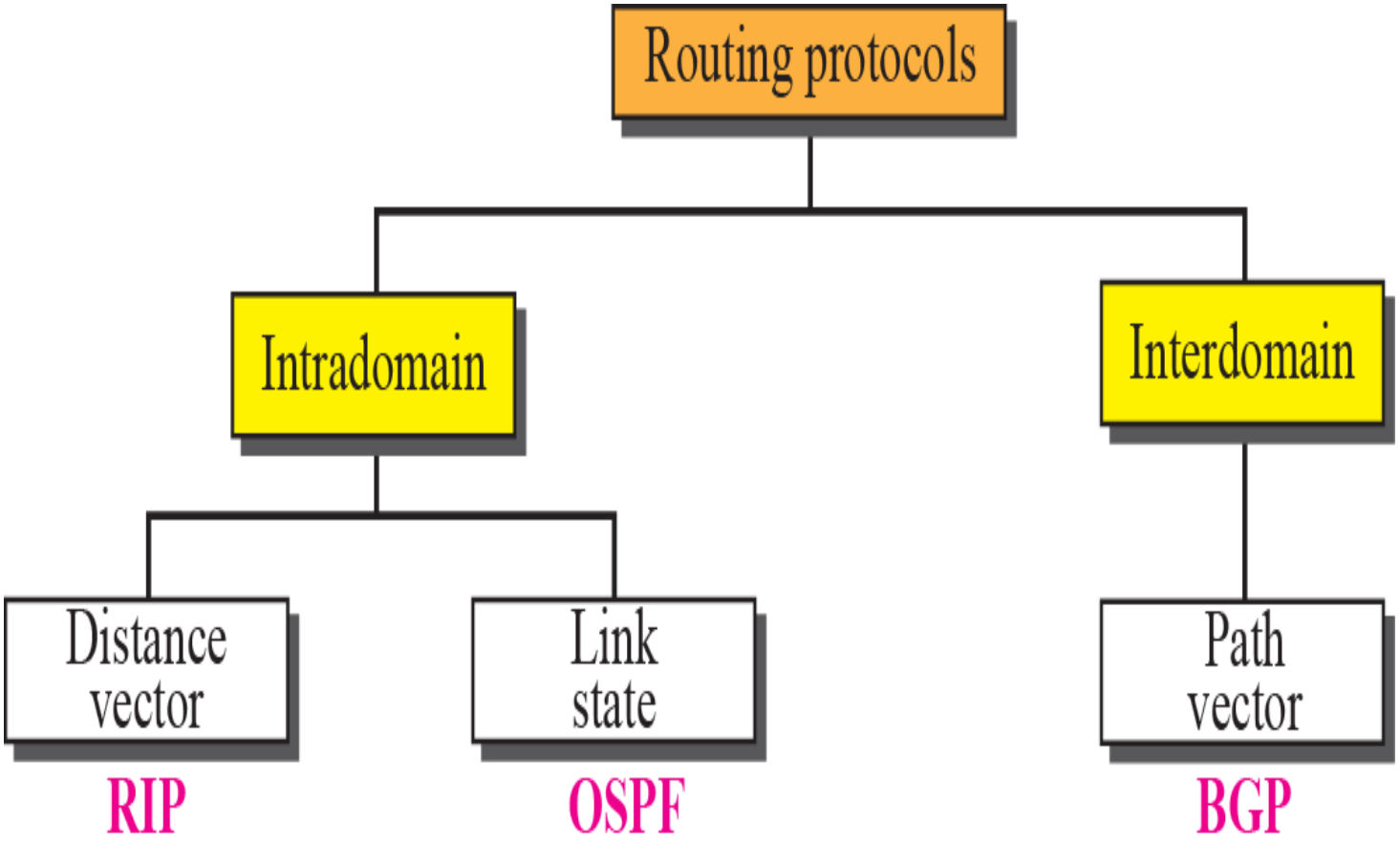
AUTONOMOUS SYSTEMS



Legend

AS	Autonomous System
	Ethernet switch
	Point-to-point WAN
	Inter-system connection

POPULAR ROUTING PROTOCOLS



DISTANCE-VECTOR ROUTING

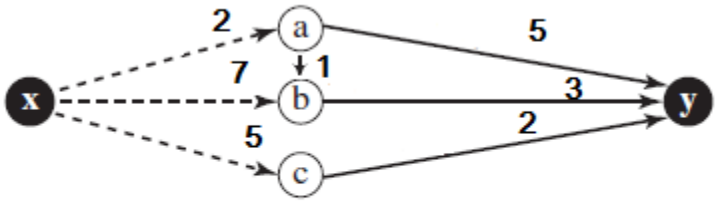
- The distance-vector (DV) routing uses the goal to find the best route.
- The first thing each node creates is its own least-cost tree with the rudimentary information it has about its immediate neighbors.
- The incomplete trees are exchanged between immediate neighbors to make the trees more and more complete and to represent the whole internet.
- router continuously tells all of its neighbors what it knows about the whole internet,

  (although the knowledge can be incomplete).

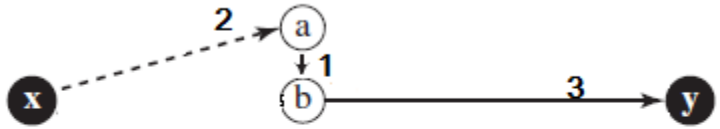
BELLMAN-FORD EQUATION

- Bellman-Ford equation is used to find the least cost (shortest distance) between a source node x and a destination node y through some intermediary nodes (a, b, c, . . .) when the costs between the source and the intermediary nodes and the least costs between the intermediary nodes and the destination are given.

- Equation $D_{xy} = \min\{(c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}), (c_{xa} + c_{ab} + D_{by}), \ldots\ldots\}$



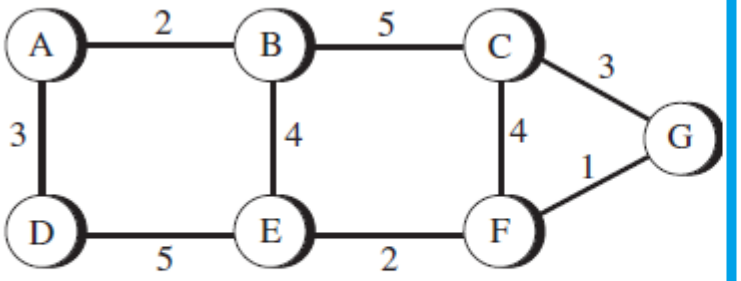
Trees represent the whole internet from Source X to Destination Y .



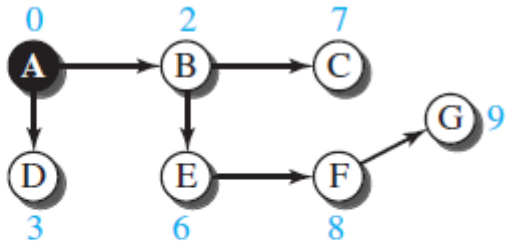
shortest distance between a Source Node x and a destination node y through some intermediary nodes

DISTANCE VECTORS

- A least-cost tree is a combination of least-cost paths from the root of the tree to all destinations.
- These paths are graphically glued together to form the tree.
- Distance-vector routing unglues these paths and creates a *distance vector, a one-dimensional array to* represent the tree.



paths are graphically glued together to form the tree.

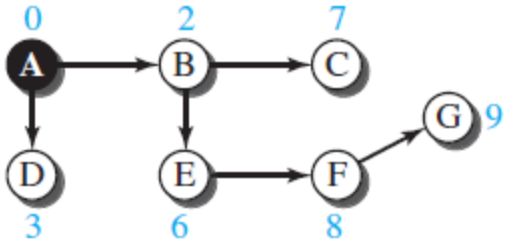


Distance-vector routing unglues these paths and creates a *distance vector*

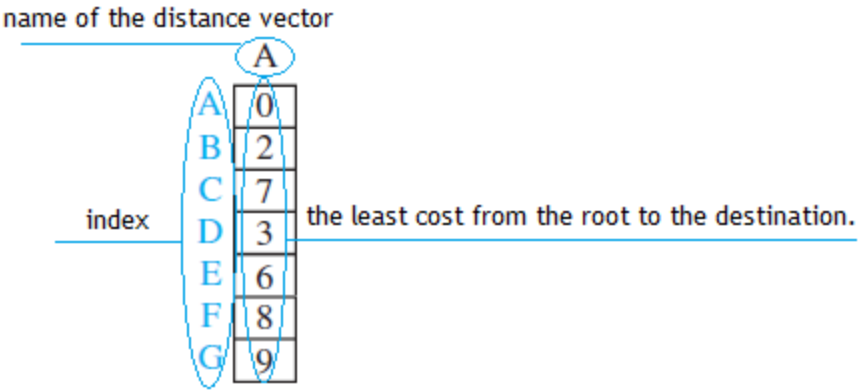
DISTANCE VECTORS

Note That :

- The name of the distance vector defines the root.
- The indexes define the destinations.
- The value of each cell defines the least cost from the root to the destination.



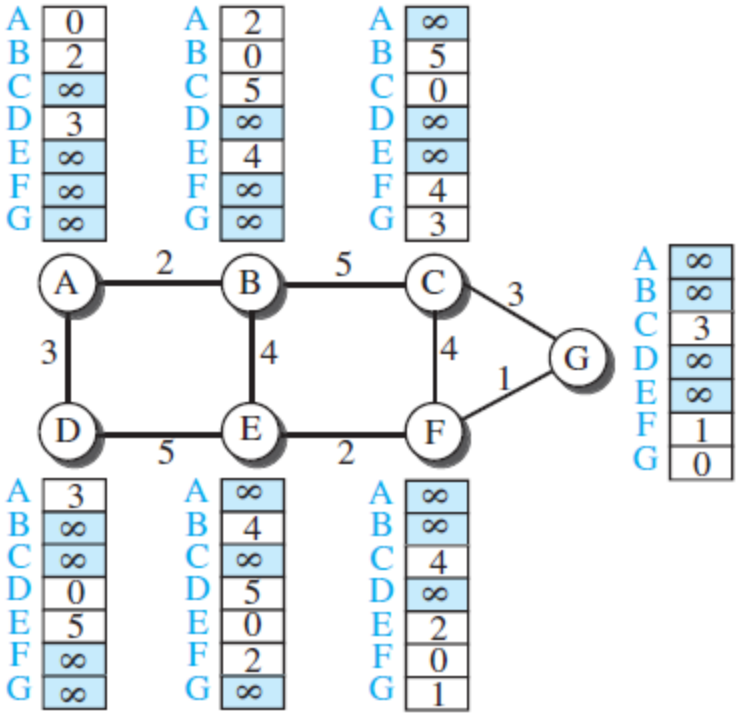
a. Tree for node A



b. Distance vector for node A

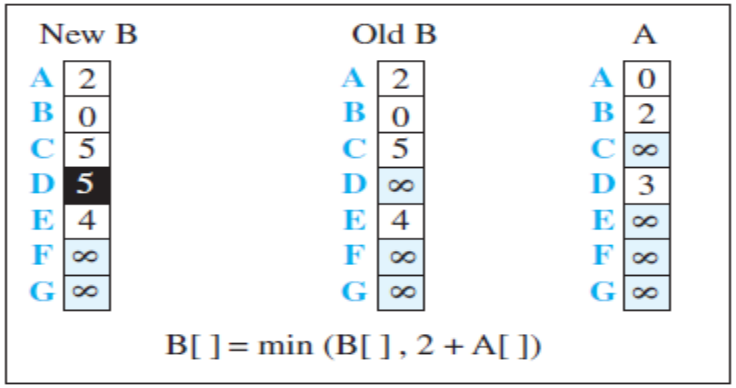
HOW EACH NODE IN AN INTERNET ORIGINALLY CREATES THE CORRESPONDING VECTOR ?

1 - Each node in an internet, when it is booted, creates a very rudimentary distance vector with the minimum information the node can obtain from its neighborhood.

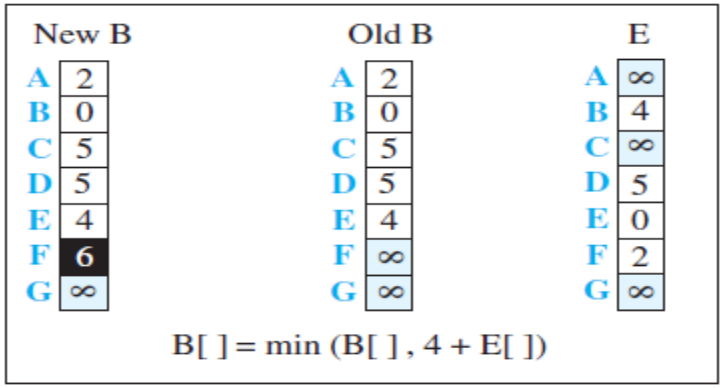


HOW EACH NODE IN AN INTERNET ORIGINALLY CREATES THE CORRESPONDING VECTOR ?

2. The node sends some greeting messages out of its interfaces and discovers the identity of the immediate neighbors and the distance between itself and each neighbor.
3. makes a simple distance vector by inserting the discovered distances in the corresponding cells and leaves the value of other cells as infinity.



a. First event: B receives a copy of A's vector.



b. Second event: B receives a copy of E's vector.

Note:
X[ ]: the whole vector

DISTANCE-VECTOR ROUTING ALGORITHM

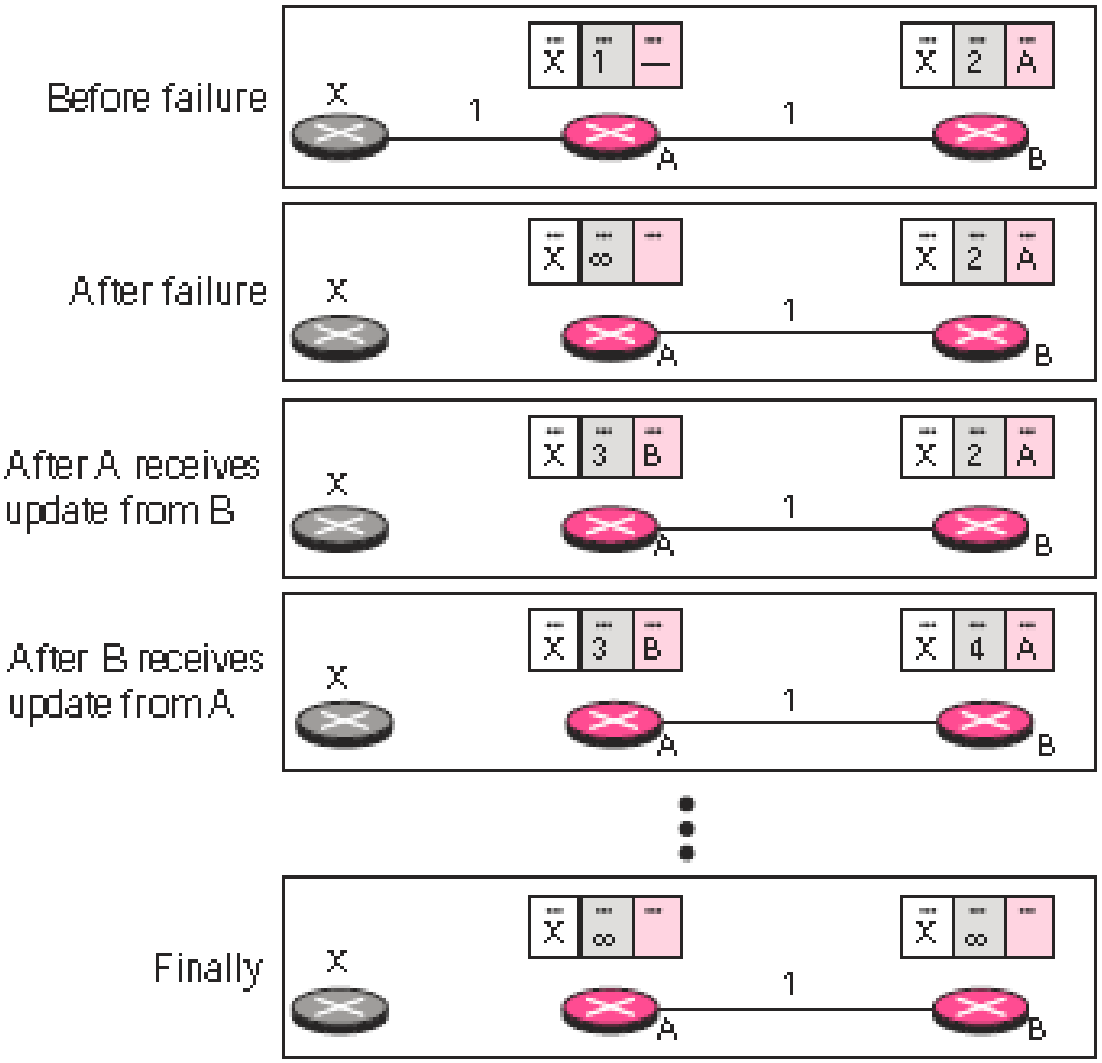
Table 20.1 *Distance-Vector Routing Algorithm for a Node*

```
1   Distance_Vector_Routing ( )
2   {
3       // Initialize (create initial vectors for the node)
4       D[myself] = 0
5       for (y = 1 to N)
6       {
7           if (y is a neighbor)
8               D[y] = c[myself][y]
9           else
10              D[y] = ∞
11      }
12      send vector {D[1], D[2], ..., D[N]} to all neighbors
13      // Update (improve the vector with the vector received from a neighbor)
14      repeat (forever)
15      {
16          wait (for a vector D_w from a neighbor w or any change in the link)
17          for (y = 1 to N)
18          {
19              D[y] = min [D[y], (c[myself][w] + D_w[y])]      // Bellman-Ford equation
20          }
21          if (any change in the vector)
22              send vector {D[1], D[2], ..., D[N]} to all neighbors
23      }
24  } // End of Distance Vector
```

COUNT TO INFINITY

- A problem with distance-vector routing is that any decrease in cost (good news) propagates quickly, but any increase in cost (bad news) will propagate slowly.
- For a routing protocol to work properly, if a link is broken (cost becomes infinity),
- every other router should be aware of it immediately, but in distance-vector routing, this takes some time.
- The problem is referred to as count to infinity.
- It sometimes takes several updates before the cost for a broken link is recorded as infinity by all routers.

COUNT TO INFINITY



SPLIT HORIZON

- In this strategy, instead of flooding the table through each interface, each node sends only part of its table through each interface.
- If, according to its table, node B thinks that the optimum route to reach X is via A.
- Taking information from node A, modifying it, and sending it back to node A is what creates the confusion.
- In this case, node A keeps the value of infinity as the distance to X.
- Later, when node A sends its forwarding table to B, node B also corrects its forwarding table.
- The system becomes stable after the first update: both node A and node B know that X is not reachable.

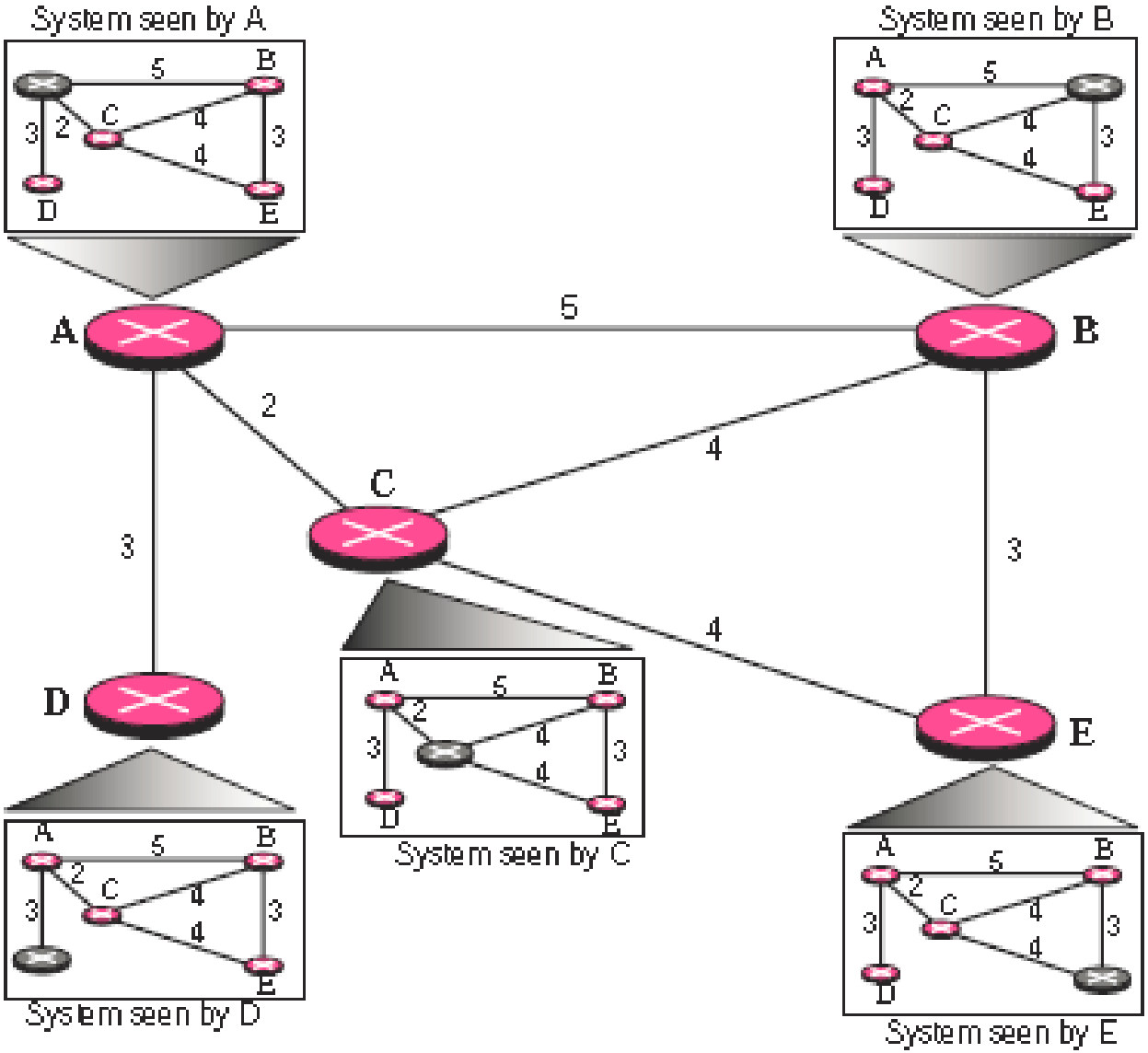
POISON REVERSE

- Using the split-horizon strategy has one drawback.
- Normally, the corresponding protocol uses a timer, and if there is no news about a route, the node deletes the route from its table.
- When node B in the previous scenario eliminates the route to X from its advertisement to A, B has not received any news about X recently.
- In the poison reverse strategy B can still advertise the value for X, but if the source of information is A, it can replace the distance with infinity as a warning: “Do not use this value”.
- if the instability is between three nodes, stability cannot be guaranteed.

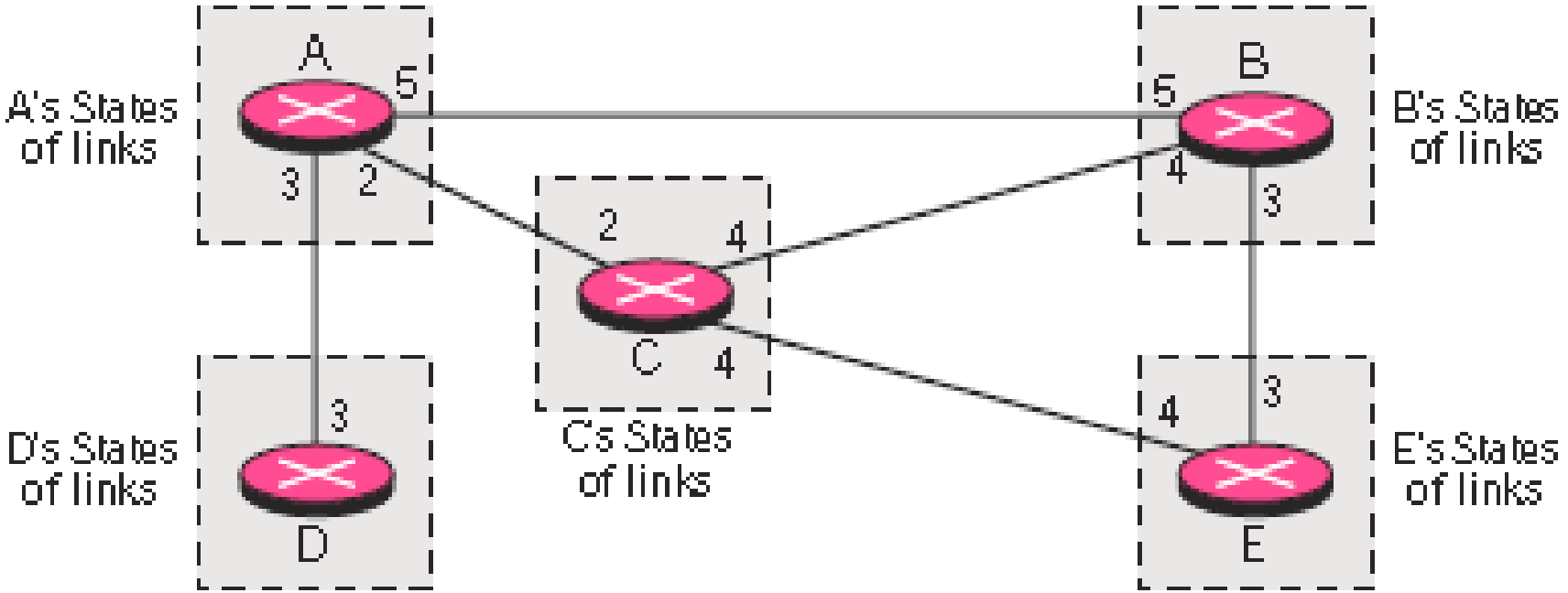
LINK-STATE ROUTING

- A routing algorithm that directly follows our discussion for creating least-cost trees and forwarding tables is link-state (LS) routing.
- This method uses the term link-state to define the characteristic of a link (an edge) that represents a network in the internet.
- Links with lower costs are preferred to links with higher costs.
- if the cost of a link is infinity, it means that the link does not exist or has been broken.

LINK STATE ROUTING



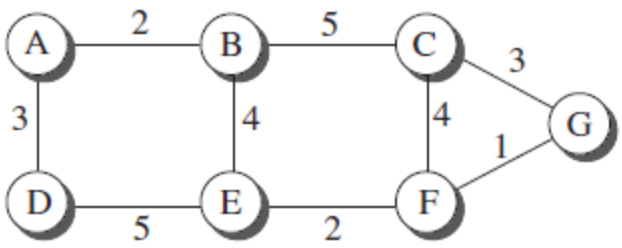
LINK STATE KNOWLEDGE



LINK-STATE DATABASE (LSDB)

- To create a least-cost tree with this method, each node needs to have a complete map of the network, which means it needs to know the state of each link.

- There is only one LSDB for the whole internet.



a. The weighted graph

	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

b. Link state database

LINK-STATE DATABASE (LSDB)

how each node can create this LSDB that contains information about the whole internet?

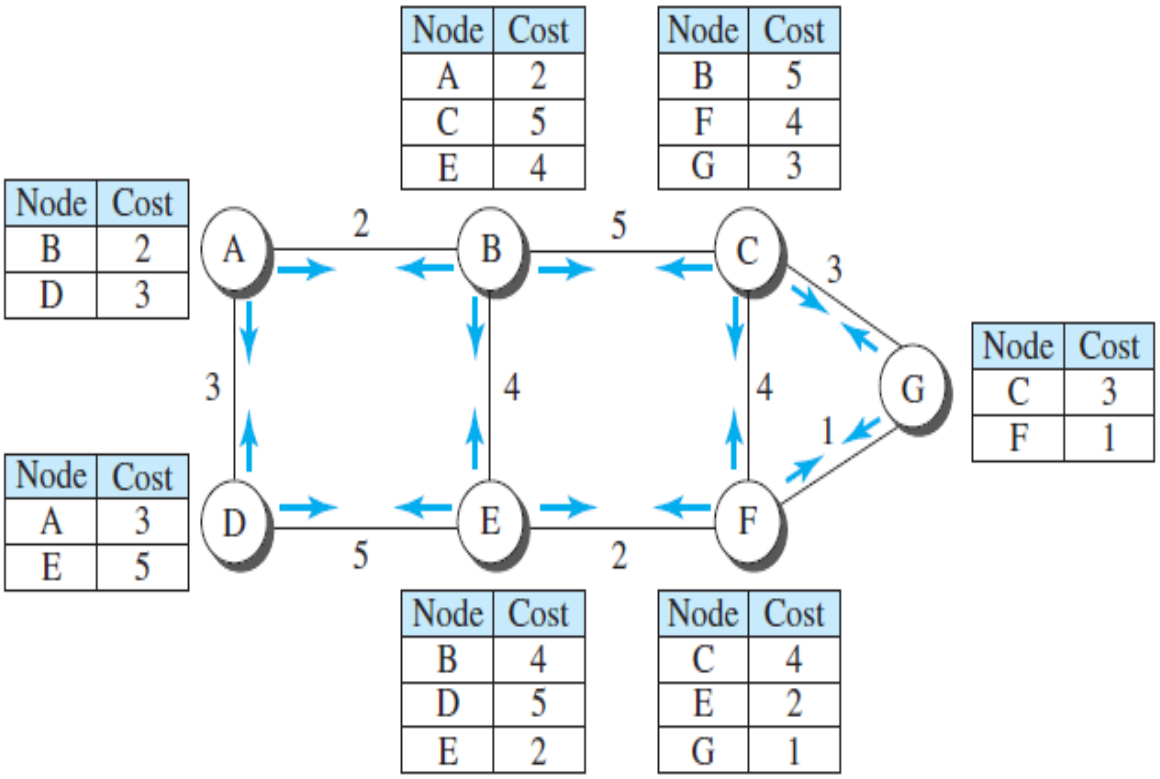
- This can be done by a process called **flooding.**
- Each node can send some greeting messages to all its immediate neighbors to collect two pieces of information for each neighboring node:

  -The identity of the node.

  -The cost of the link.
- The combination of these two pieces of information is called the LS packet (LSP).
- The LSP is sent out of each interface, as shown in Figure 20.9 for our internet in Figure 20.

LINK-STATE DATABASE (LSDB)

Figure 20.9 *LSPs created and sent out by each node to build LSDB*



LINK-STATE DATABASE (LSDB)

- When a node receives an LSP from one of its interfaces, it compares the LSP with the copy it may already have.
- If the arrived LSP is older than the one it has, it discards the arrived LSP.
- If it is newer than the one it has, the node discards the old LSP and keeps the received one.
- It then sends a copy of it out of each interface except the one from which the packet arrived.

FORMATION OF LEAST-COST TREES

To create a least-cost tree for itself, using the shared LSDB, each node needs to run the famous Dijkstra Algorithm.

This iterative algorithm uses the following steps:

1. The node chooses itself as the root of the tree, creating a tree with a single node, and sets the total cost of each node based on the information in the LSDB.

2. The node selects one node, among all nodes not in the tree, which is closest to the root, and adds this to the tree.

After this node is added to the tree, the cost of all other nodes not in the tree needs to be updated because the paths may have been changed.

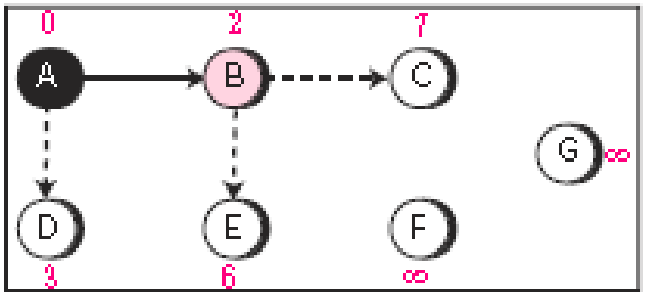
3. The node repeats step 2 until all nodes are added to the tree.

DIJKSTRA'S ALGORITHM

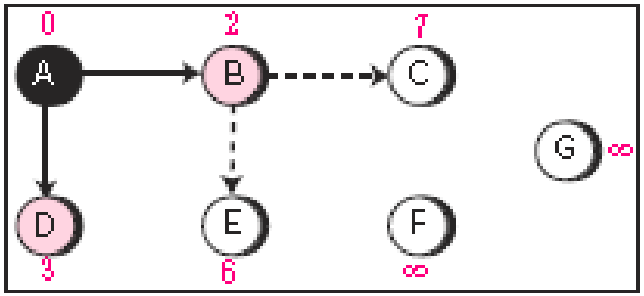
Table 20.2 *Dijkstra's Algorithm*

```
1  Dijkstra's Algorithm ( )
2  {
3      // Initialization
4      Tree = {root}                    // Tree is made only of the root
5      for (y = 1 to N)                 // N is the number of nodes
6      {
7          if (y is the root)
8              D[y] = 0                 // D[y] is shortest distance from root to node y
9          else if (y is a neighbor)
10             D[y] = c[root][y]        // c[x][y] is cost between nodes x and y in LSDB
11         else
12             D[y] = ∞
13     }
14     // Calculation
15     repeat
16     {
17         find a node w, with D[w] minimum among all nodes not in the Tree
18         Tree = Tree ∪ {w}            // Add w to tree
19         // Update distances for all neighbors of w
20         for (every node x, which is a neighbor of w and not in the Tree)
21         {
22             D[x] = min {D[x], (D[w] + c[w][x])}
23         }
24     } until (all nodes included in the Tree)
25 } // End of Dijkstra
```

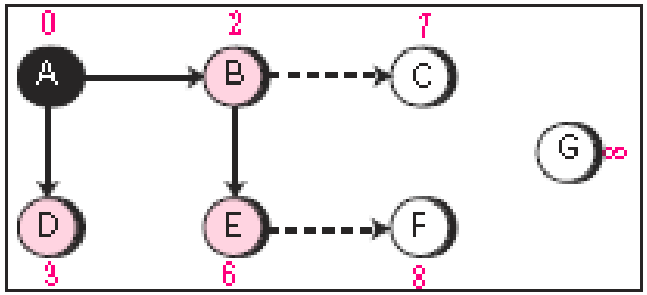
FORMATION OF LEAST-COST TREES



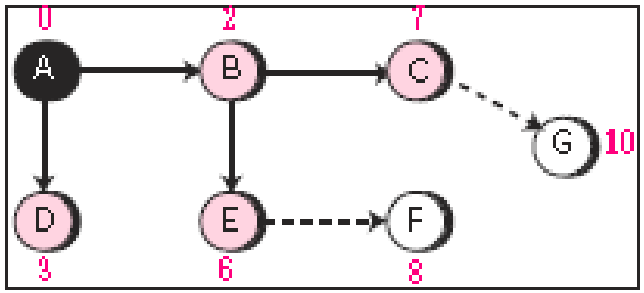
Iteration 1



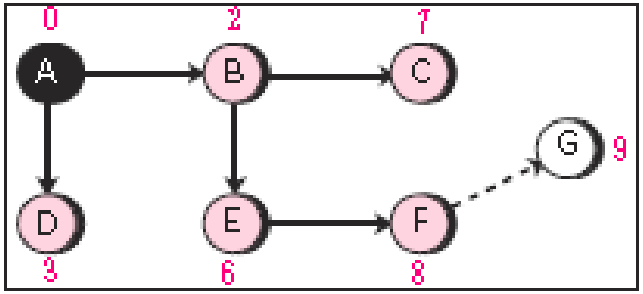
Iteration 2



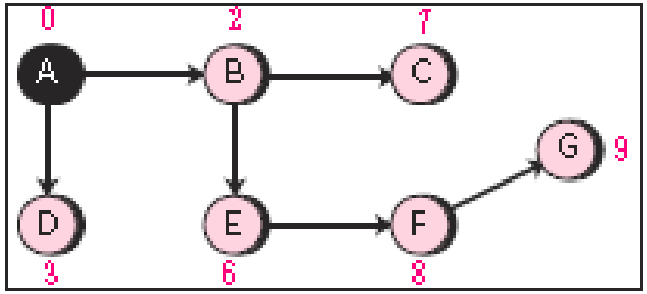
Iteration 3



Iteration 4



Iteration 5



Iteration 6

CALCULATION OF ROUTING TABLE

Table 11.4 *Routing Table for Node A*

Destination	*Cost*	*Next Router*
A	0	—
B	2	—
C	7	B
D	3	—
E	6	B
F	8	B
G	9	B

PATH-VECTOR ROUTING

- Both link-state and distance-vector routing are based on the least-cost goal.

Least cost goal is not a priority

- router may belong to an organization that does not provide enough security or it may belong to a commercial rival of the sender which might inspect the packets for obtaining information.

- Least-cost routing does not prevent a packet from passing through an area when that area is in the least-cost path.

PATH-VECTOR ROUTING

- Path-vector routing does not have the drawbacks of LS or DV routing as described above because it is not based on least-cost routing.
- The best route is determined by the source using the policy it imposes on the route.
- In other words, the source can control the path.
- path-vector routing is not actually used in an internet, and is mostly designed to route a packet between ISPs

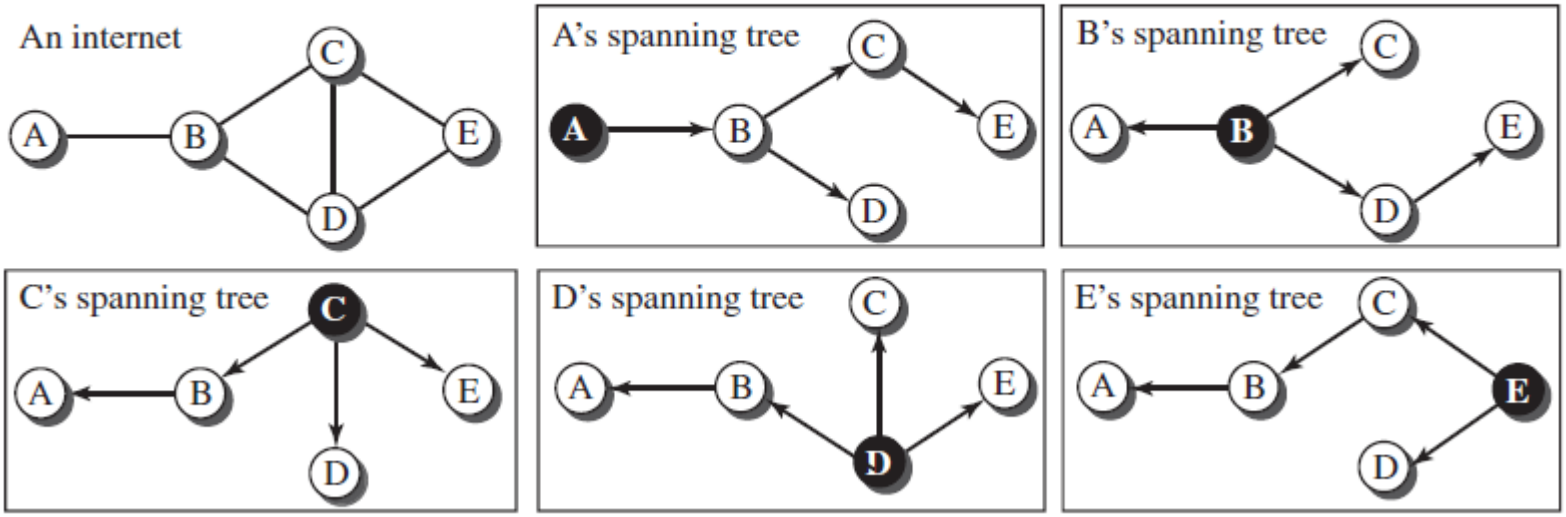
SPANNING TREES

- The tree determined by the source when it imposes its own policy.
- If there is more than one route to a destination, the source can choose the route that meets its policy best.
- A source may apply several policies at the same time.
- One of the common policies uses the minimum number of nodes to be visited.
- common policy is to avoid some nodes as the middle node in a route.

SPANNING TREES IN PATH-VECTOR ROUTING

- The policy imposed by all sources is to use the minimum number of nodes to reach a destination.
- The spanning tree selected by A and E is such that the communication does not pass through D as a middle node.

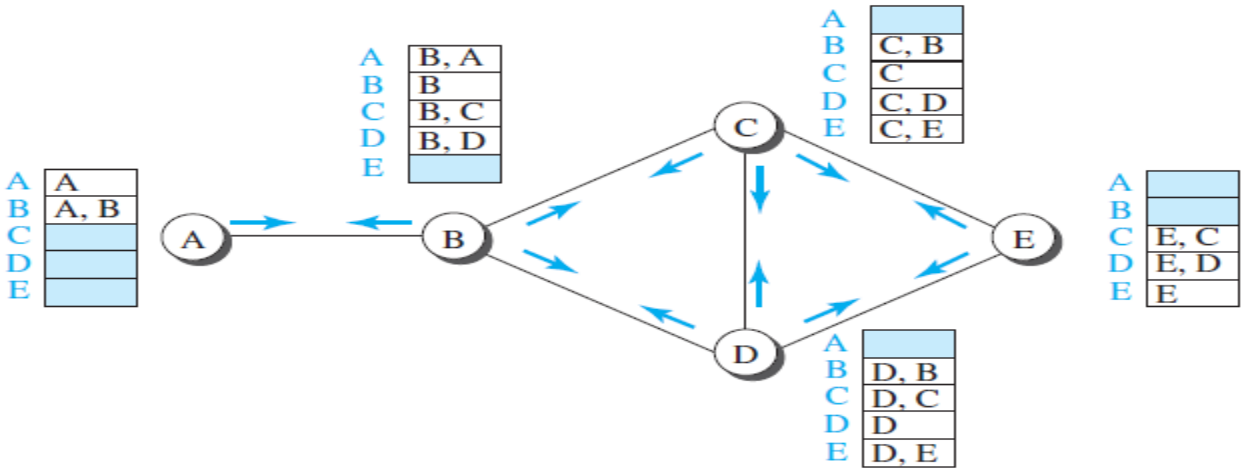
Figure 20.11 *Spanning trees in path-vector routing*



CREATION OF SPANNING TREES

- When a node is booted, it creates a path vector based on the information it can obtain about its immediate neighbor.
- A node sends greeting messages to its immediate neighbors to collect these pieces of information.

Figure 20.12 *Path vectors made at booting time*

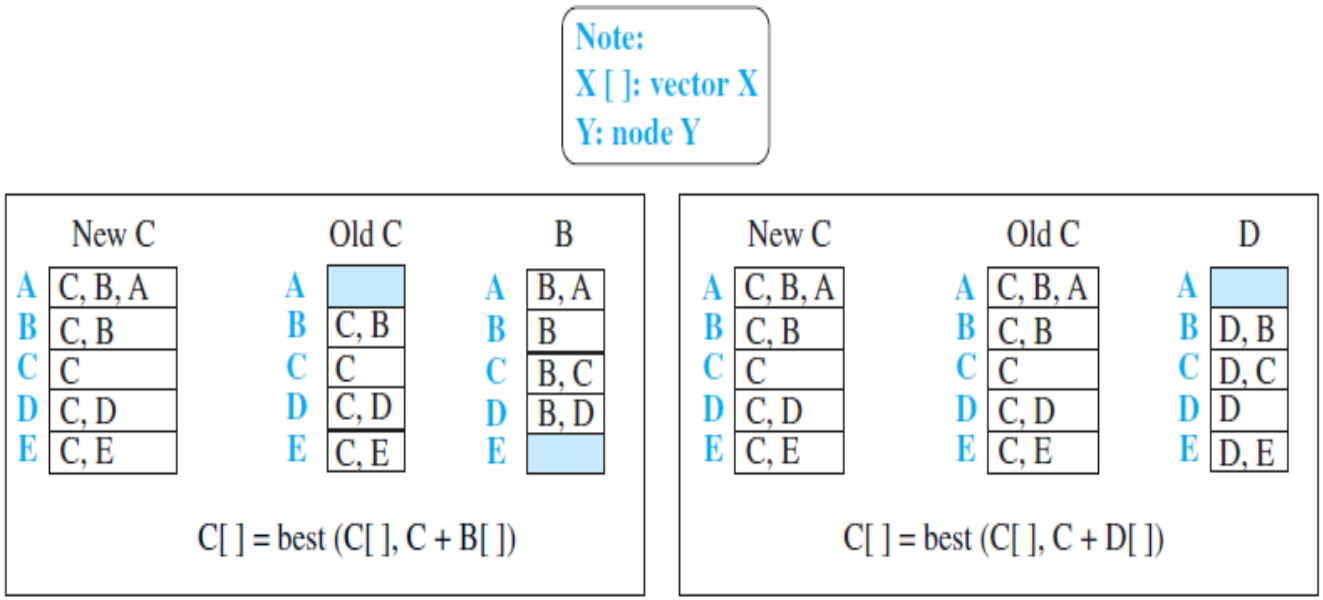


CREATION OF SPANNING TREES

- The policy is defined by selecting the best of multiple paths.
- Path-vector routing also imposes one more condition on this equation: If Path (v, y) includes x, that path is discarded to avoid a loop in the path.
- In other words, x does not want to visit itself when it selects a path to y.

UPDATING PATH VECTORS

Figure 20.13 *Updating path vectors*



Event 1: C receives a copy of B's vector

Event 2: C receives a copy of D's vector

PATH-VECTOR ALGORITHM

Table 20.3 *Path-vector algorithm for a node*

```
1   Path_Vector_Routing ( )
2   {
3       // Initialization
4       for (y = 1 to N)
5       {
6           if (y is myself)
7               Path[y] = myself
8           else if (y is a neighbor)
9               Path[y] = myself + neighbor node
10          else
11              Path[y] = empty
12      }
13      Send vector {Path[1], Path[2], ..., Path[y]} to all neighbors
14      // Update
15      repeat (forever)
16      {
17          wait (for a vector Path_w from a neighbor w)
18          for (y = 1 to N)
19          {
20              if (Path_w includes myself)
21                  discard the path                    // Avoid any loop
22              else
23                  Path[y] = best {Path[y], (myself + Path_w[y])}
24          }
25          If (there is a change in the vector)
26              Send vector {Path[1], Path[2], ..., Path[y]} to all neighbors
27      }
28  } // End of Path Vector
```

NEXT