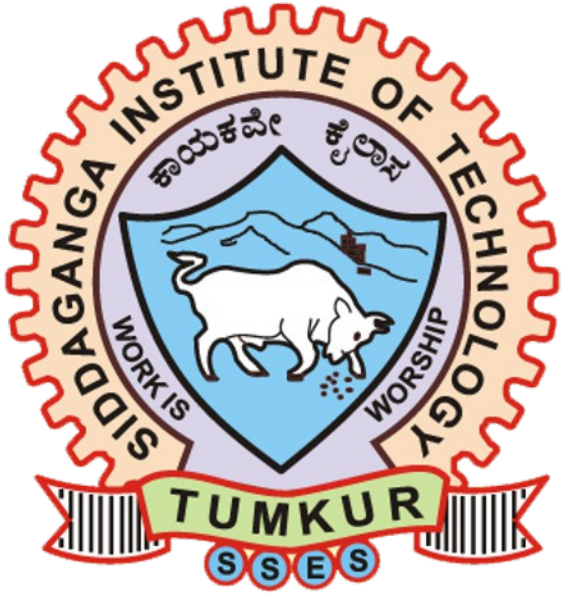
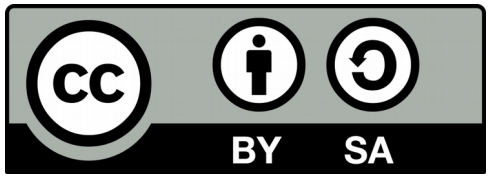
ROUTING

SIDDAGANGA INSTITUTE OF TECHNOLOGY

Department of CSE

Prabodh C P





Creative Commons Attribution-ShareAlike 4.0 International Public License

Contents

- Introduction
  - General Idea
  - Least-Cost Routing
- ROUTING ALGORITHMS
  - Distance-Vector Routing
  - Link-State Routing
  - Path-Vector Routing
- UNICAST ROUTING PROTOCOLS
  - Internet Structure
  - Routing Information Protocol (RIP)
  - Open Shortest Path First(OSPF)
  - Border Gateway Protocol Version 4 (BGP4)

INTRODUCTION

- In an internet, the goal of the network layer is to deliver a datagram from its source to its destination or destinations.
- If a datagram is destined for only one destination (one-to-one delivery), we have unicast routing.
- If the datagram is destined for several destinations (one-to-many delivery), we have multicast routing.
- A router uses a forwarding table to route packets in internet.
- To make the forwarding tables of the router, the Internet needs routing protocols that will be active all the time in the background and update the forwarding tables.

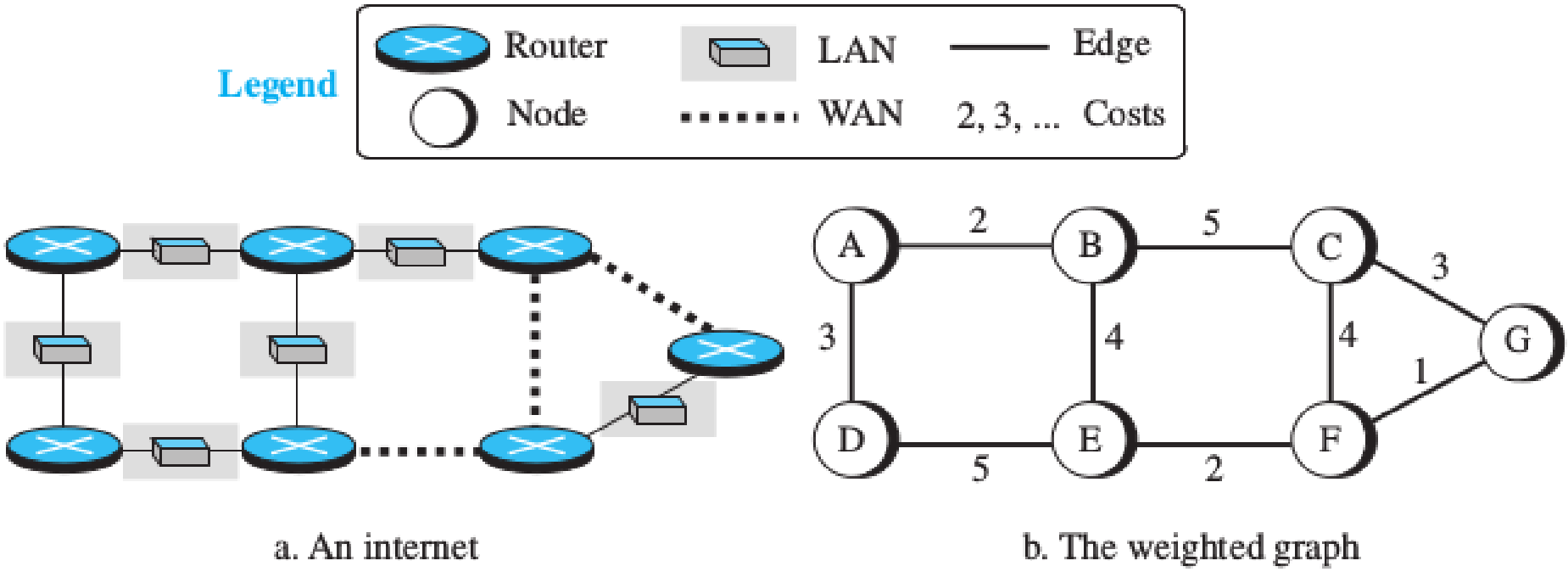
Unicast Routing

- Unicast routing in the Internet, with a large number of routers and a huge number of hosts, can be done only by using hierarchical routing: routing in several steps using different routing algorithms.
- In unicast routing, a packet is routed, hop by hop, from its source to its destination by the help of forwarding tables.
- The source host needs no forwarding table because it delivers its packet to the default router in its local network.
- The destination host needs no forwarding table either because it receives the packet from its default router in its local network.
- This means that only the routers that glue together the networks in the internet need forwarding tables.
- Now routing a packet from its source to its destination means routing the packet from a source router (the default router of the source host) to a destination router (the router connected to the destination network).

An Internet as a Graph

- To find the best route, an internet can be modeled as a graph.
- A graph in computer science is a set of nodes and edges (lines) that connect the nodes.
- To model an internet as a graph, we can think of each router as a node and each network between a pair of routers as an edge.
- An internet is, in fact, modeled as a weighted graph, in which each edge is associated with a cost.

An Internet as a Graph



a. An internet

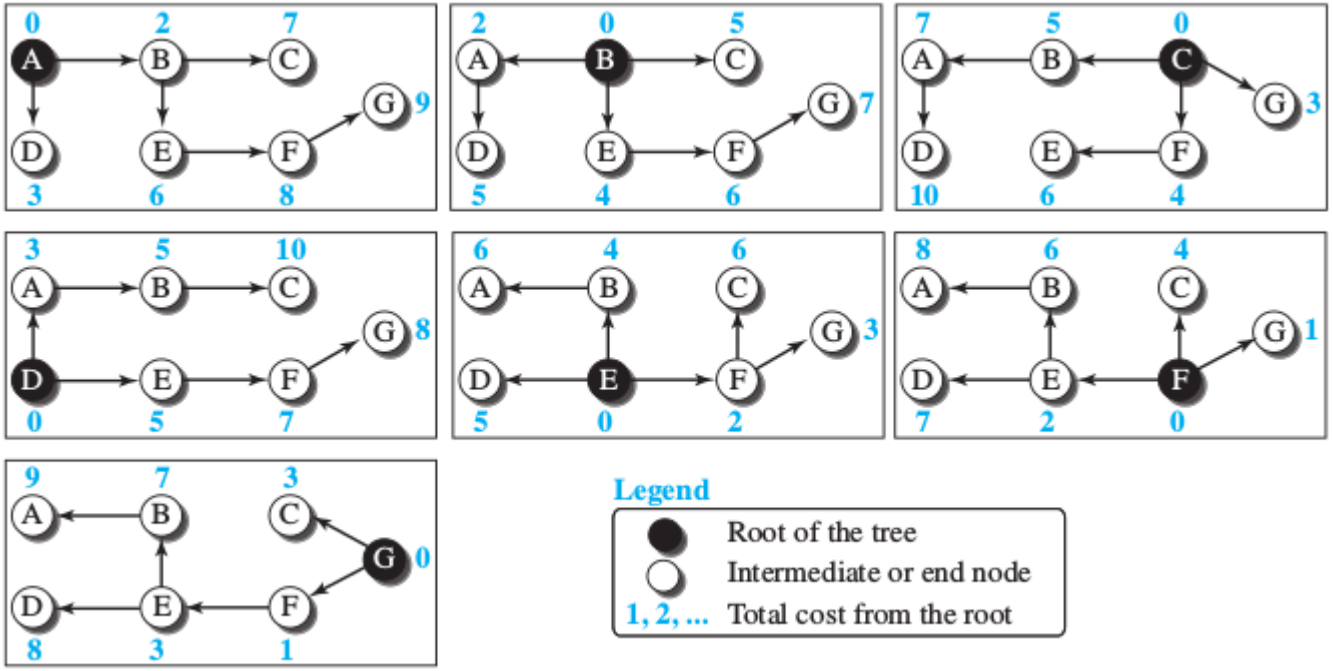
b. The weighted graph

Least-Cost Routing

- When an internet is modeled as a weighted graph, one of the ways to interpret the best route from the source router to the destination router is to find the least cost between the two.
- In other words, the source router chooses a route to the destination router in such a way that the total cost for the route is the least cost among all possible routes.
- Here each router needs to find the least-cost route between itself and all the other routers to be able to route a packet

Least-Cost Trees

- If there are N routers in an internet, there are (N – 1) least-cost paths from each router to any other router.
- This means we need N × (N – 1) least-cost paths for the whole internet.
- A better way to see all of these paths is to combine them in a least-cost tree.
- A least-cost tree is a tree with the source router as the root that spans the whole graph (visits all other nodes) and in which the path between the root and any other node is the shortest.



Least-Cost Trees Properties

- The least-cost route from X to Y in X's tree is the inverse of the least-cost route from Y to X in Y's tree; the cost in both directions is the same.
- Instead of travelling from X to Z using X's tree, we can travel from X to Y using X's tree and continue from Y to Z using Y's tree.

  Eg : A → B → E → F → G

  (A → B → E) (E → F → G).

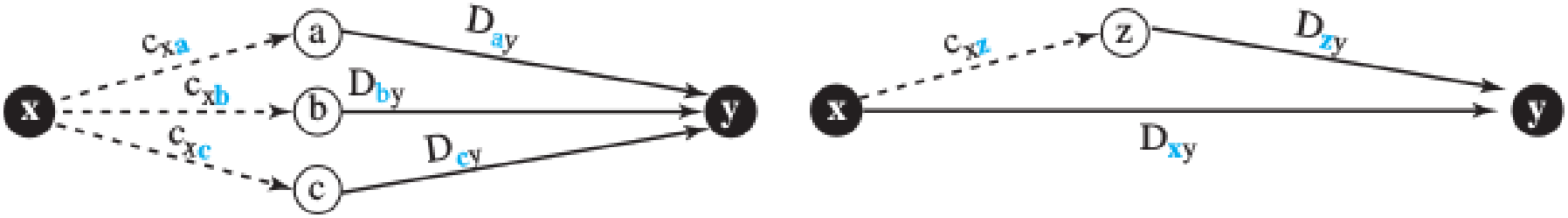
ROUTING ALGORITHMS

- Distance-Vector Routing
  - In distance-vector routing, 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.
  - We can say that in distance-vector routing, a router continuously tells all of its neighbors what it knows about the whole internet

Bellman-Ford Equation

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

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

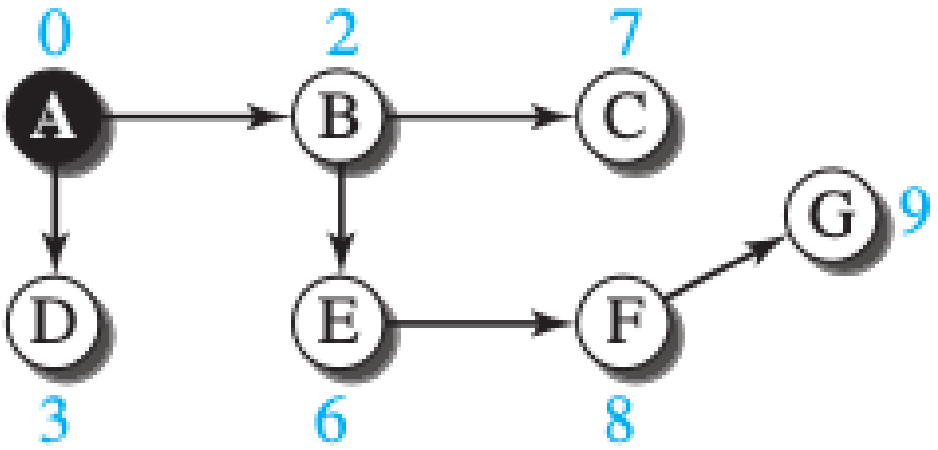


a. General case with three intermediate nodes

b. Updating a path with a new route

we can think of (a → y), (b → y), and (c → y) as previously established least-cost paths and (x → y) as the new least-cost path. We can say that the Bellman-Ford equation enables us to build a new least-cost path from previously established least-cost paths.

Distance Vectors



a. Tree for node A

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

b. Distance vector for node A

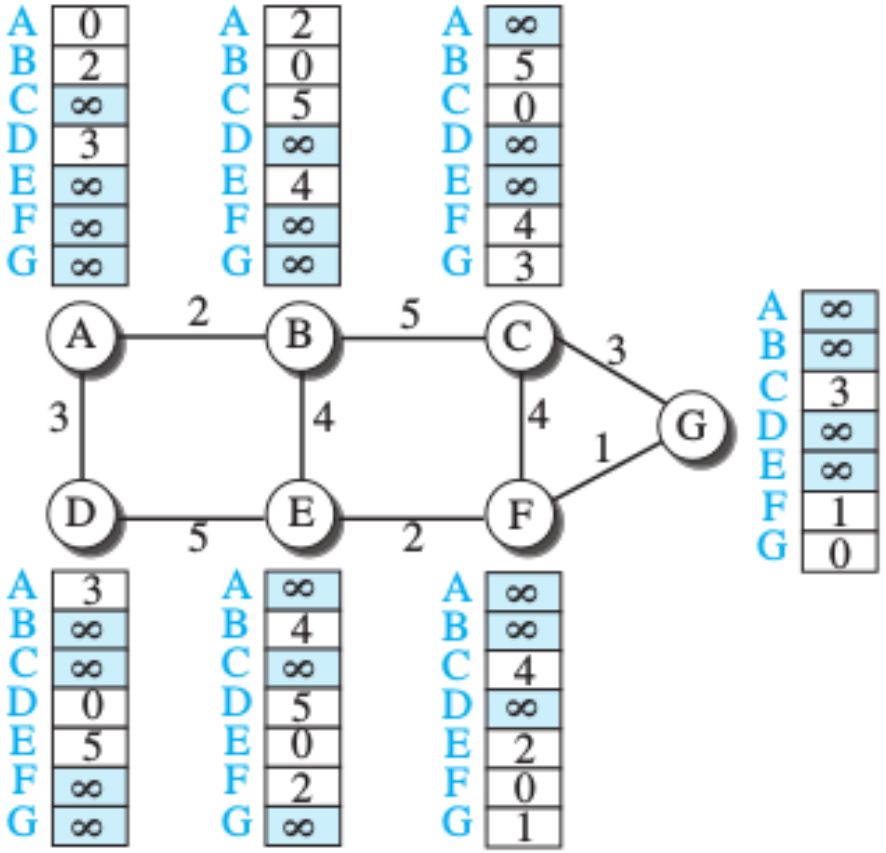
Distance Vectors

- A least-cost tree is a combination of least-cost paths from the root of the tree to all destinations.
- These grouped paths are split by a creating a distance vector, a one-dimensional array to represent the tree.
- Note that the name of the distance vector defines the root, the indexes define the destinations, and the value of each cell defines the least cost from the root to the destination.
- It does not have path information.

Distance Vectors

- How is this vector created?
- 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.
- 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.
- Unknown cells value are set to infinity.
- These initial vectors are created asynchronously.
- After each node has created its vector, it sends a copy of the vector to all its immediate neighbors.
- Then update the distance vectors using the Bellman-Ford equation

Figure 20.5 *The first distance vector for an internet*



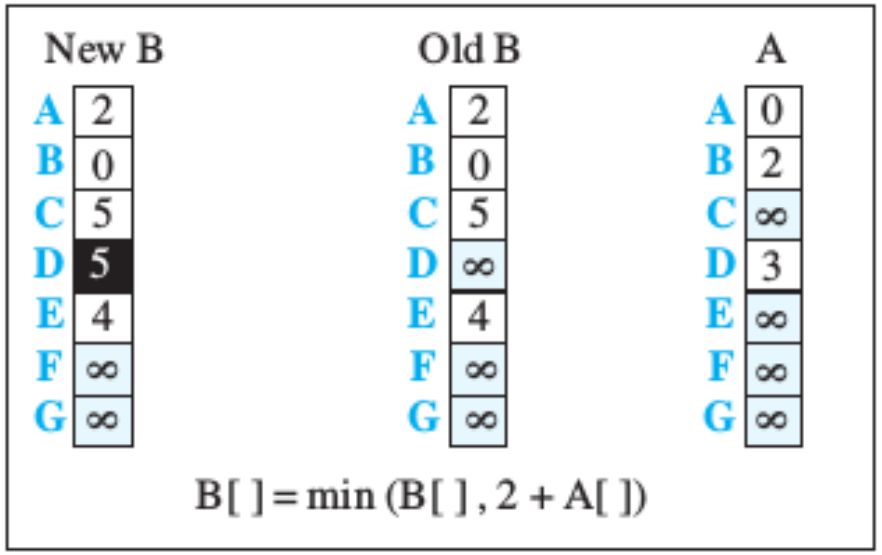
Updating Distance Vectors

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

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

Updating Distance Vectors

Figure 20.6 *Updating distance vectors*

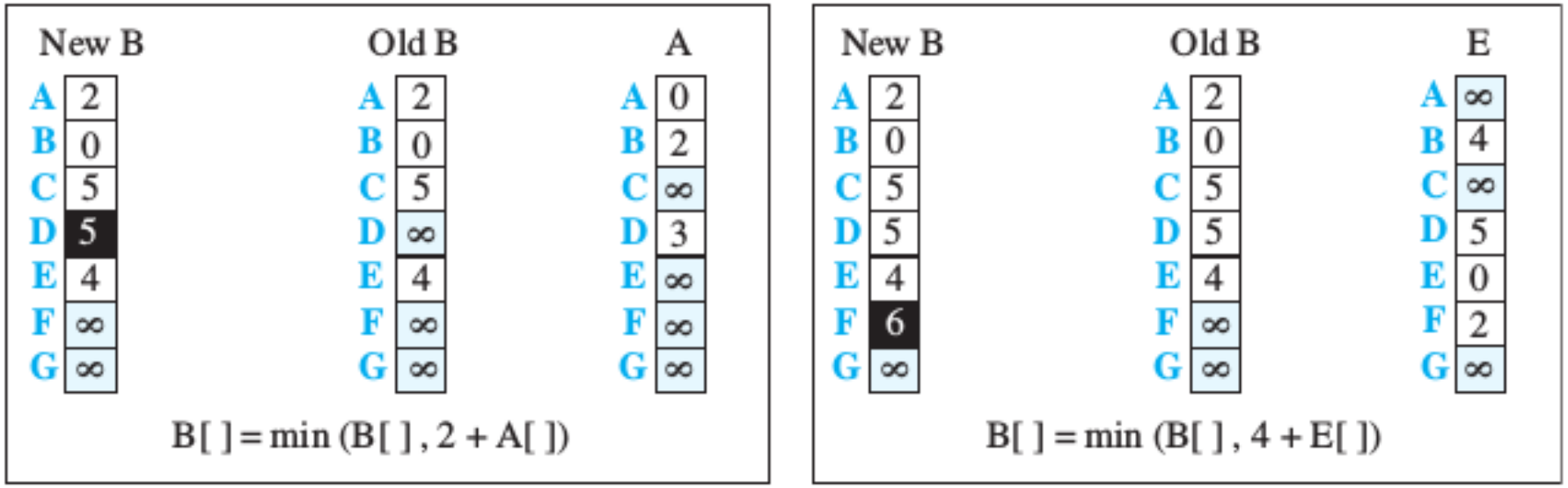


a. First event: B receives a copy of A's vector.

Note:
X[ ]: the whole vector

Updating Distance Vectors

Figure 20.6 *Updating distance vectors*



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

```
Distance_Vector_Routing ( )
{// Initialize (create initial vectors for the node)
      D[myself ] = 0
      for (y = 1 to N){
                  if (y is a neighbor)
                        D[y] = c[myself ][y]
                  else
                              D[y] = ∞
      }
```

Distance-Vector Routing Algorithm

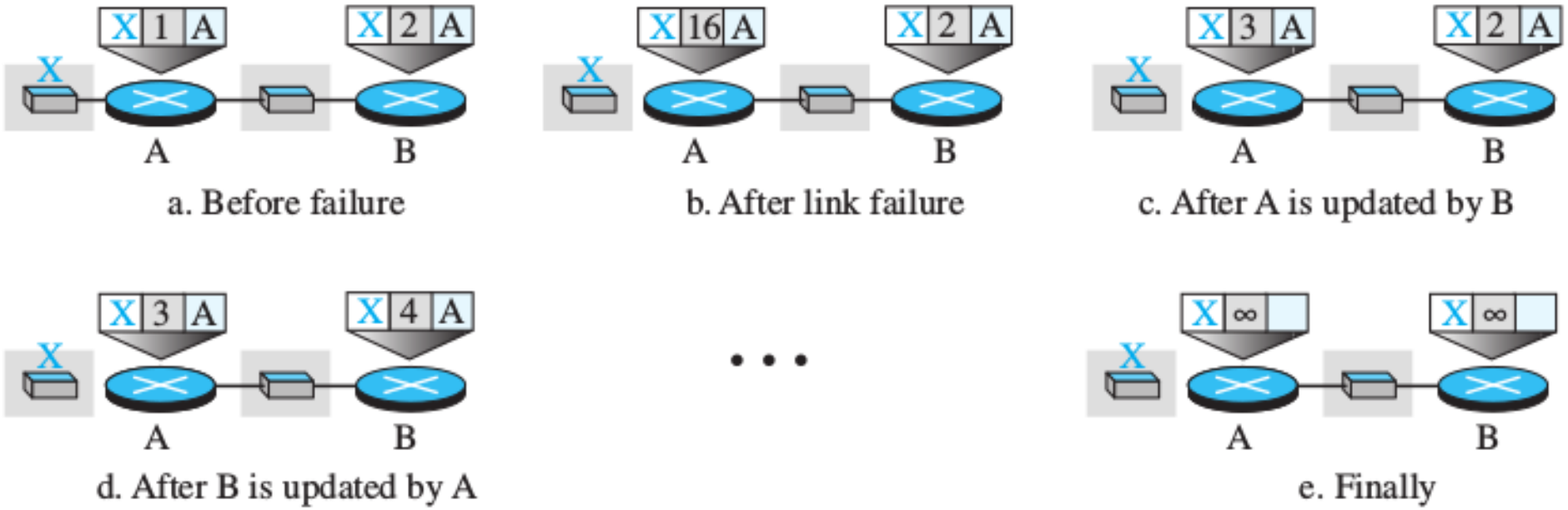
```
send vector {D[1], D[2], ..., D[N]} to all neighbors
// Update (improve the vector with the vector received from a neighbor)
repeat (forever) {
    wait (for a vector D w from a neighbor w or any change in the link)
    for (y = 1 to N) {
        D[y] = min [D[y], (c[myself ][w] + D w [y ])]  // Bellman-Ford equation
    }
    if (any change in the vector)
        send vector {D[1], D[2], ..., D[N]} to all neighbors
}
} // 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.

Two-Node Loop

Figure 20.7 *Two-node instability*



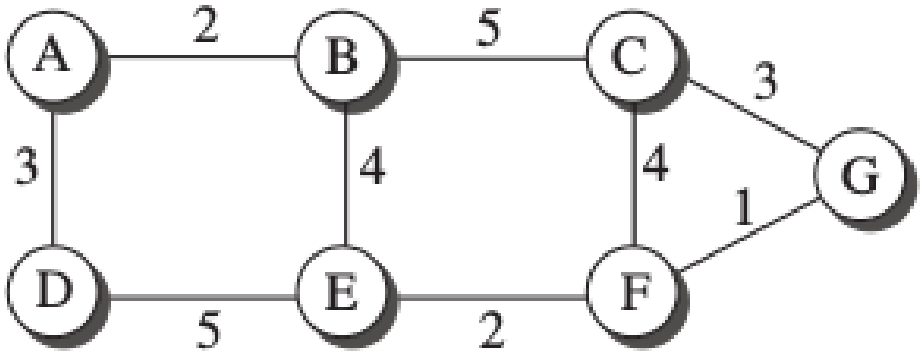
Split Horizon

- Instead of flooding the table through each interface, each node sends only part of its table through each interface.
- Taking information from a node, modifying it, and sending it back to the same node is what creates the confusion.
- node B thinks that the optimum route to reach X is via A, it does not need to advertise this piece of information to A; the information has actually come from A
- node B eliminates the last line of its forwarding table before it sends it to A. In this case, node A keeps the value of infinity as the distance to X.

Link-State Routing

- This method uses the term link-state to define the characteristic of a link (an edge) that represents a network in the internet.
- In this algorithm the cost associated with an edge defines the state of the link.
- 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.
- To create a least-cost tree with this method, each node needs to have a complete map of the network, which is represented using a Link-State Database (LSDB).
- The LSDB can be represented as a two-dimensional array(matrix) in which the value of each cell defines the cost of the corresponding link.

Link-State Database



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

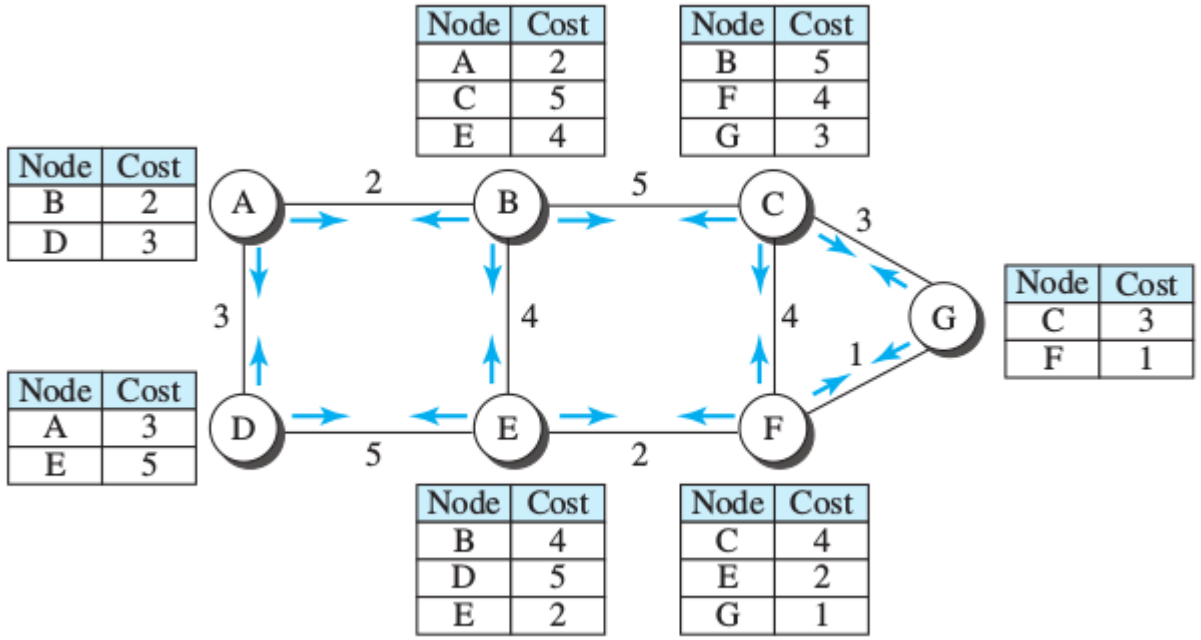
b. Link state database

Building a LSDB

- LSDB is created 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 and 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
- When a node receives an LSP from one of its interfaces, it compares the LSP with the copy it may already have.
- If the newly arrived LSP is older than the one it has (found by checking the sequence number), it discards the LSP.
- If it is newer or the first one received, the node discards the old LSP (if there is one) and keeps the received one.

Building a LSDB

- It then sends a copy of it out of each interface except the one from which the packet arrived.
- This guarantees that flooding stops somewhere in the network.
- After receiving all new LSPs, each node creates the comprehensive LSDB



Comparison

- In the distance-vector routing algorithm,
  - each router tells its neighbors what it knows about the whole internet;
- In the link-state routing algorithm,
  - each router tells the whole internet what it knows about its neighbors.

Formation of Least-Cost Trees

- To create a least-cost tree each node needs to run the famous Dijkstra Algorithm.

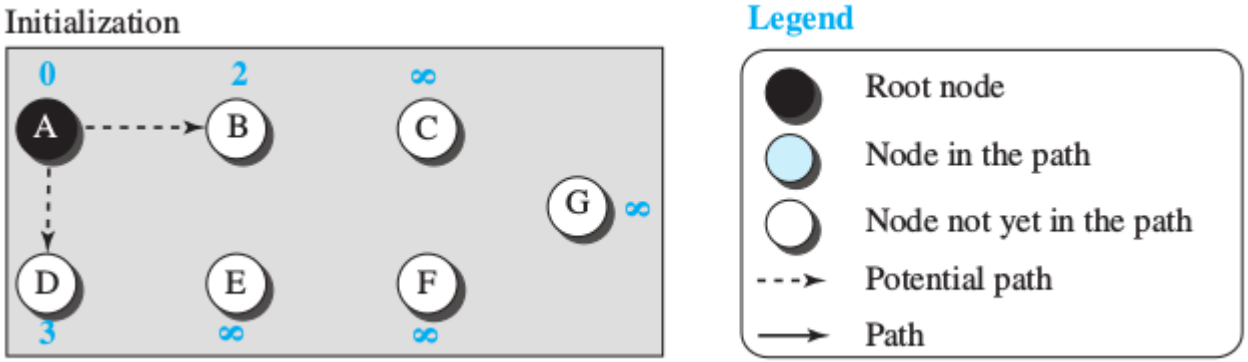
1. The node chooses itself as the root of the tree, creating a tree with a single node, and sets the total cost of each node based on the information in the LSDB.

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.

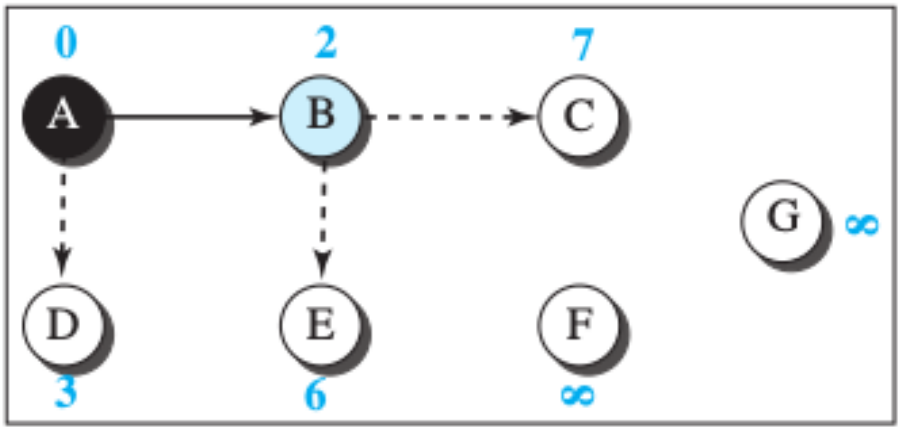
3. The node repeats step 2 until all nodes are added to the tree.

Dijkstra's Algorithm

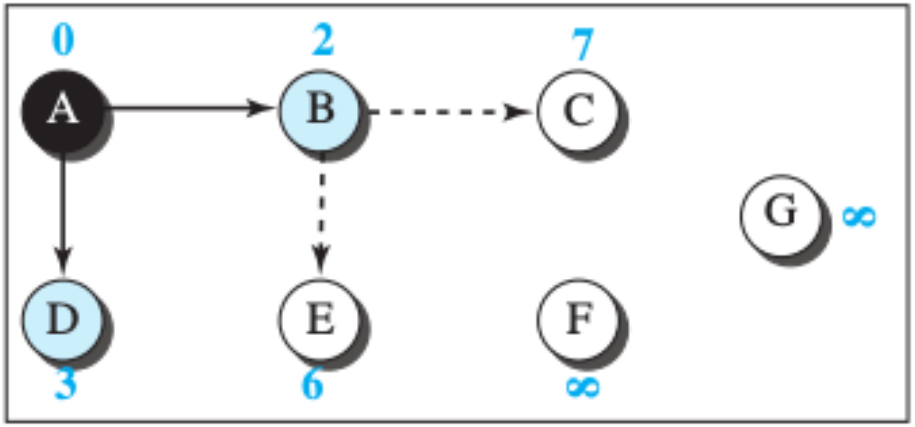
- Initialization



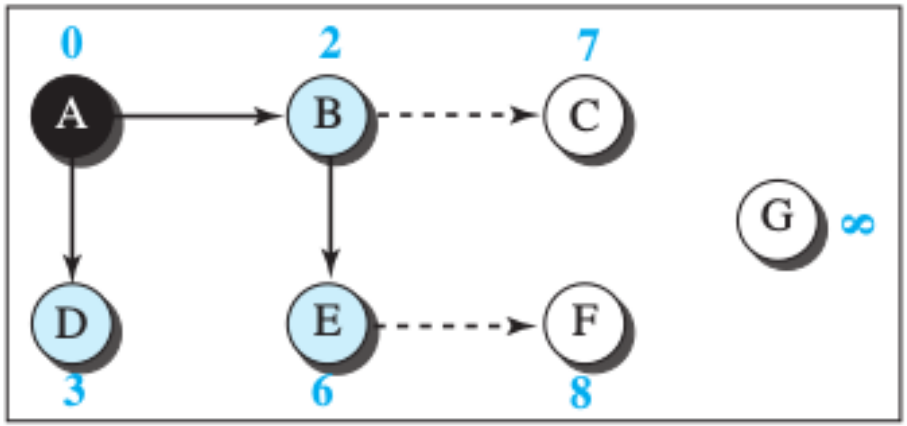
Iteration 1



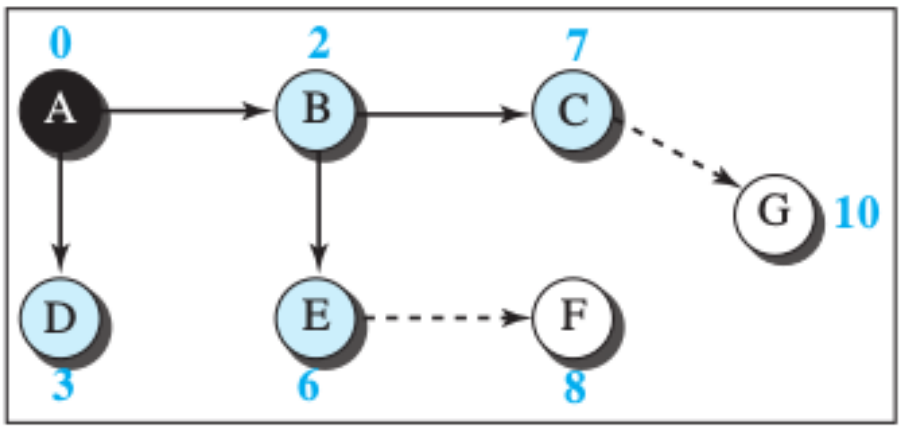
Iteration 2



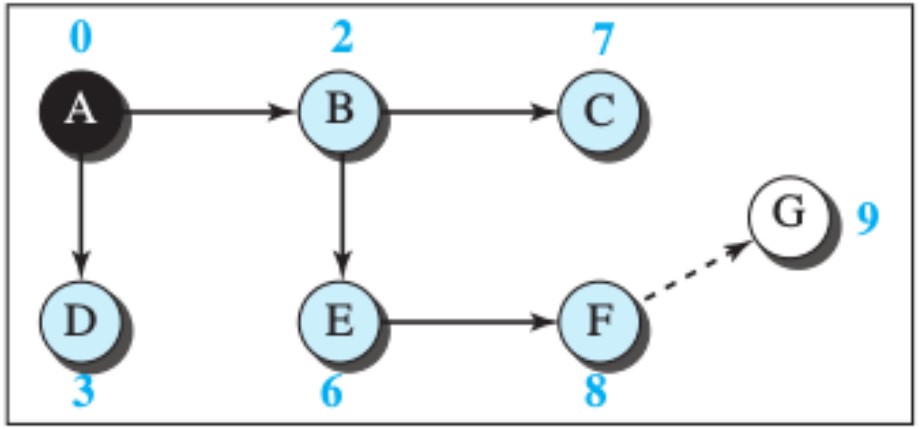
Iteration 3



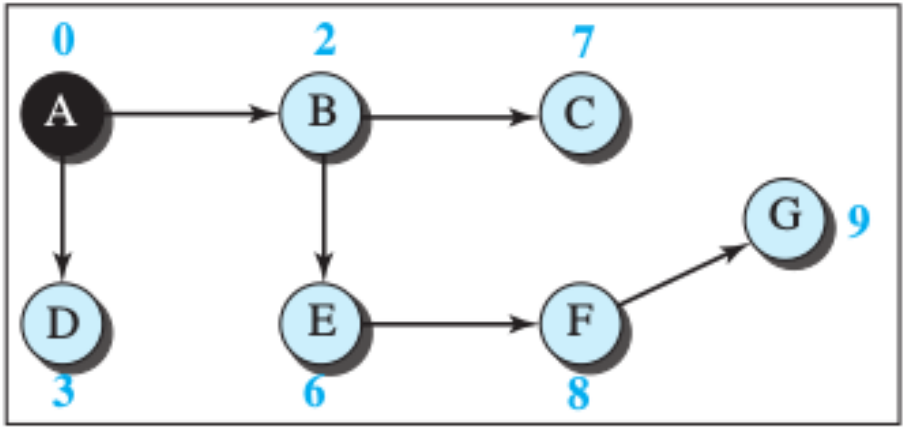
Iteration 4



Iteration 5



Iteration 6



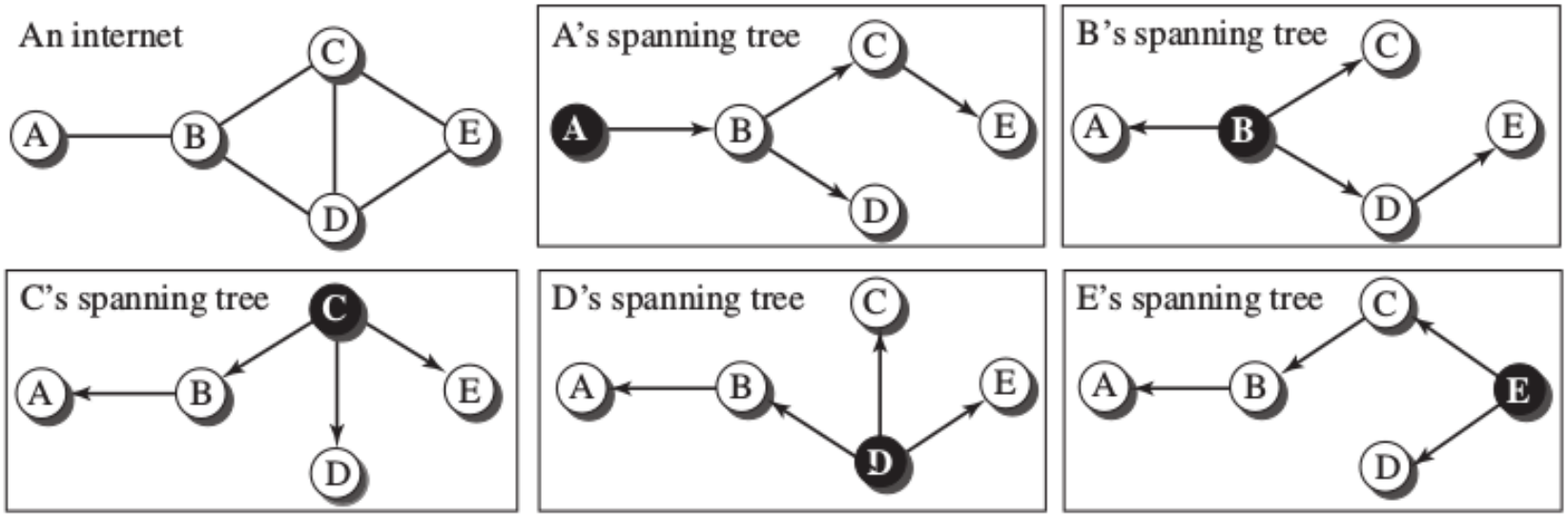
Path-Vector Routing

- In some cases routing is not based on least cost but may be based on other considerations
  - Like avoiding a particular router
- To respond to these demands, a third routing algorithm, called path-vector (PV) routing has been devised.
- 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

- In path-vector routing, the path from a source to all destinations is also determined by the best spanning tree.
- The best spanning tree, however, is not the least-cost tree; it is the tree determined by the source when it imposes its own policy.
- Some common policies are
  - minimum number of nodes to be visited
  - to avoid some nodes as the middle node in a route.
- Each source creates its own spanning tree that meets its policy.

Spanning trees in path-vector routing

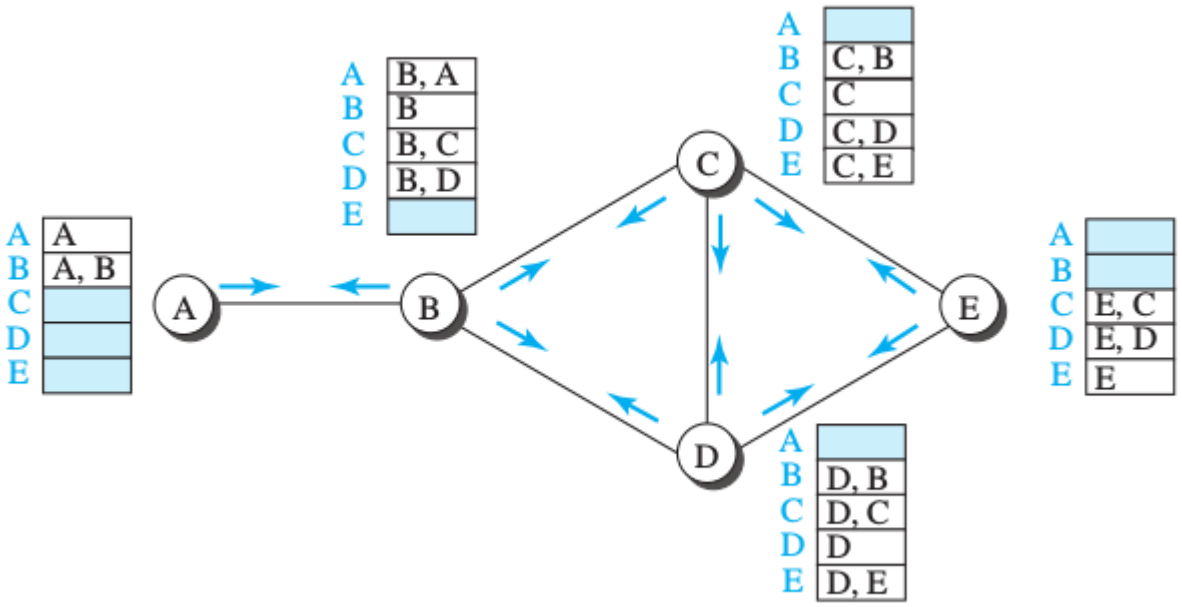


- The policy imposed by all sources is to use the minimum number of nodes to reach a destination.
- The spanning tree A and E
  - communication does not pass through D as a middle node.
- The spanning tree B
  - communication does not pass through C as a middle node.

Creation of Spanning Trees

- The spanning trees are made, gradually and asynchronously, by each node.
- 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.

Path vectors made at booting time



Updating path vectors

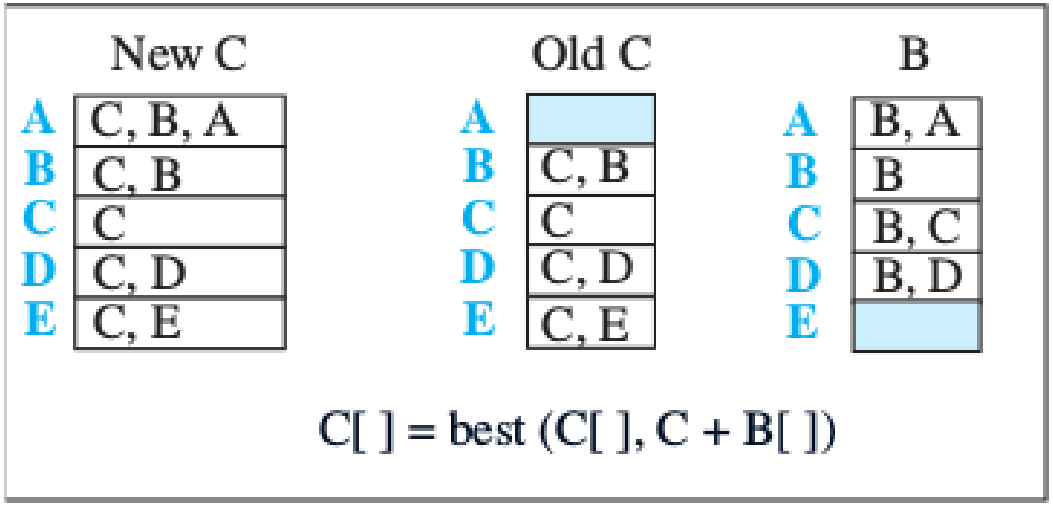
- Each node, when it receives a path vector from a neighbor, updates its path vector using an equation similar to the Bellman-Ford, but applying its own policy instead of looking for the least cost.
- We can define this equation as

$$\textbf{Path(x, y) = best \{Path(x, y), [(x + Path(v , y)]\}}$$

for all v 's in the internet.

- In this equation, the operator (+) means to add x to the beginning of the path.
- 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.

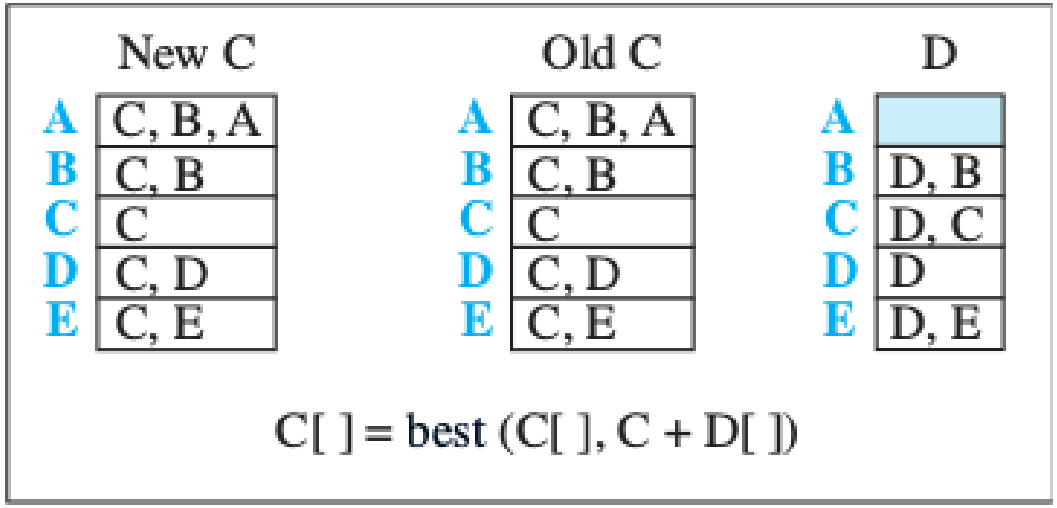
C receives a copy of B's vector



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

- Node C receives a copy of B's vector, which improves its vector: now it knows how to reach node A.

C receives a copy of D's vector



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

- node C receives a copy of D's vector, which does not change its vector.

Path-Vector Algorithm

```
Path_Vector_Routing ( )
{
    // Initialization
    for (y = 1 to N)
    {
        if (y is myself)
            Path[y] = myself
        else if (y is a neighbor)
            Path[y] = myself + neighbor node
        else
            Path[y] = empty
    }
    Send vector {Path[1], Path[2], ..., Path[y]} to all neighbors
    // Update
    repeat (forever)
    {
        wait (for a vector Path w from a neighbor w)
        for (y = 1 to N)
        {
            if (Path w includes myself)
                discard the path                    // Avoid any loop
            else
                Path[y] = best {Path[y], (myself + Path w [y])}
        }
        If (there is a change in the vector)
        Send vector {Path[1], Path[2], ..., Path[y]} to all neighbors
    }
} // End of Path Vector
```