# SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

**An Autonomous Institution**

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING


## COURSE NAME :19IT401 COMPUTER NETWORKS

II YEAR /IV SEMESTER

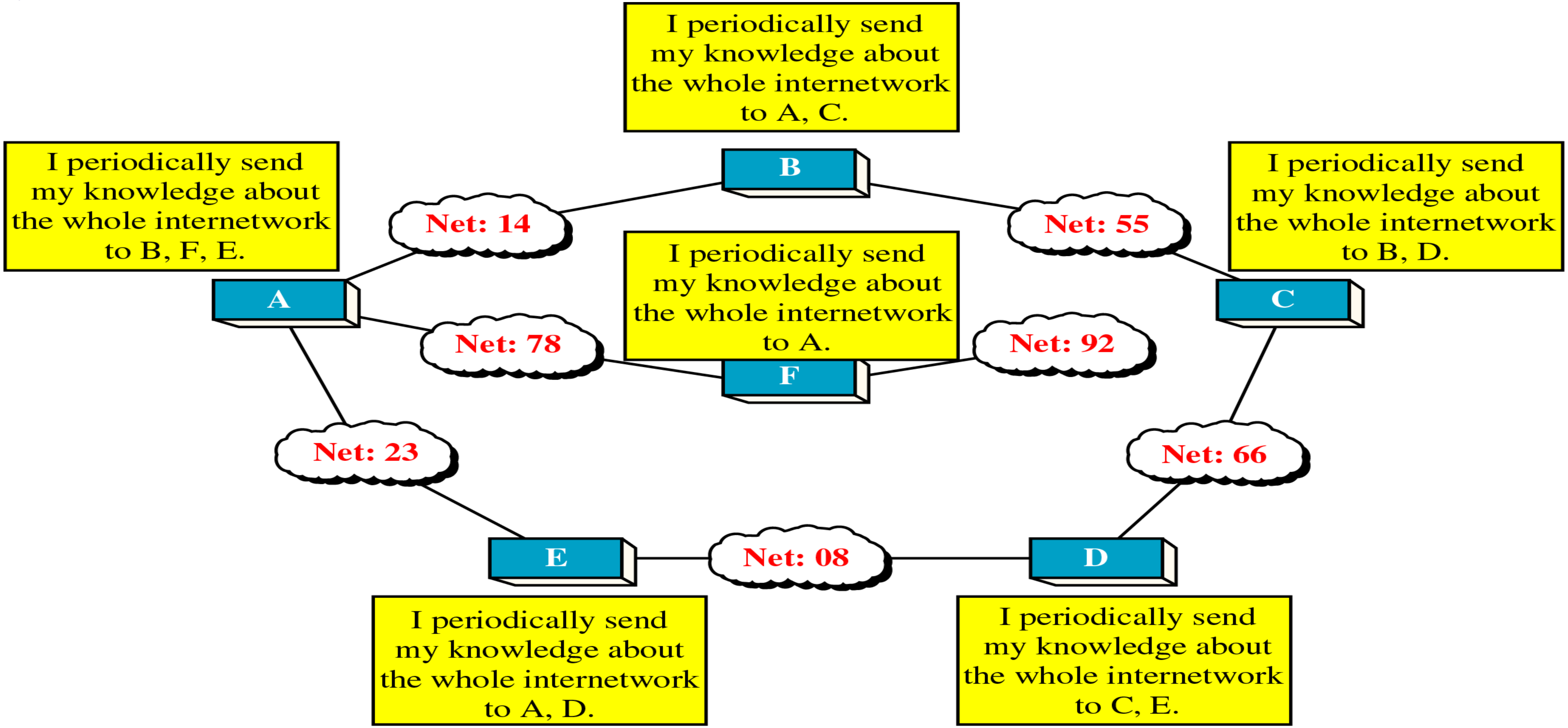
Unit 3-NETWORK LAYER

Topic 8 : Routing algorithms

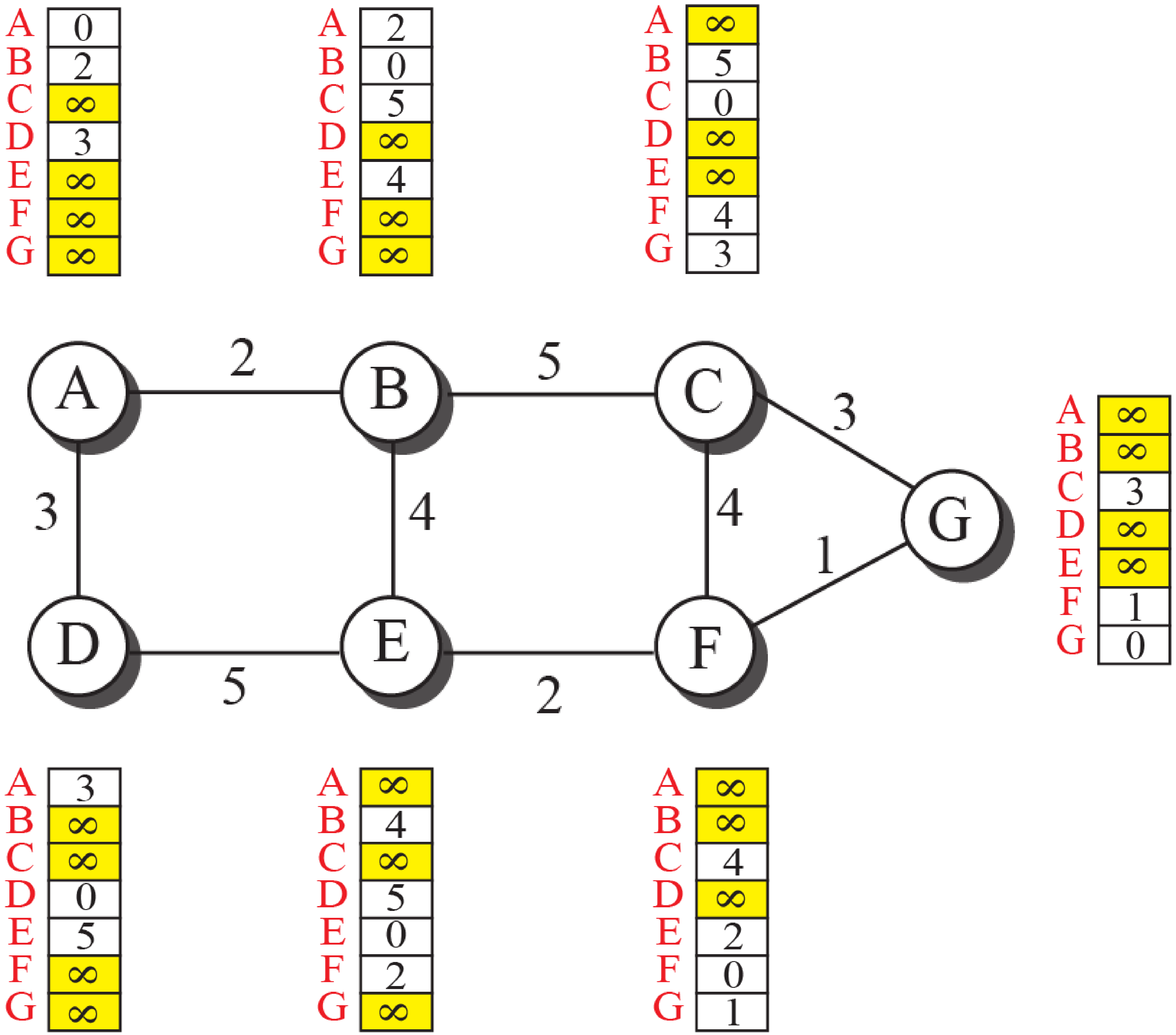
# Distance-Vector Routing

- In distance-vector routing, the first thing each node creates is its own least-cost tree with 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 (although the knowledge can be incomplete).

# The Concept of Distance Vector Routing



# The first distance vector for an internet



# Figure 20.6: Updating distance vectors

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

B[ ] = min (B[ ] , 2 + A[ ])

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

**Note**:
X[ ]: the whole vector

| | New B | Old B | E |
|---|---|---|---|
| A | 2 | 2 | ∞ |
| B | 0 | 0 | 4 |
| C | 5 | 5 | ∞ |
| D | 5 | 5 | 5 |
| E | 4 | 4 | 0 |
| F | 6 | ∞ | 2 |
| G | ∞ | ∞ | ∞ |

B[ ] = min (B[ ] , 4 + E[ ])

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

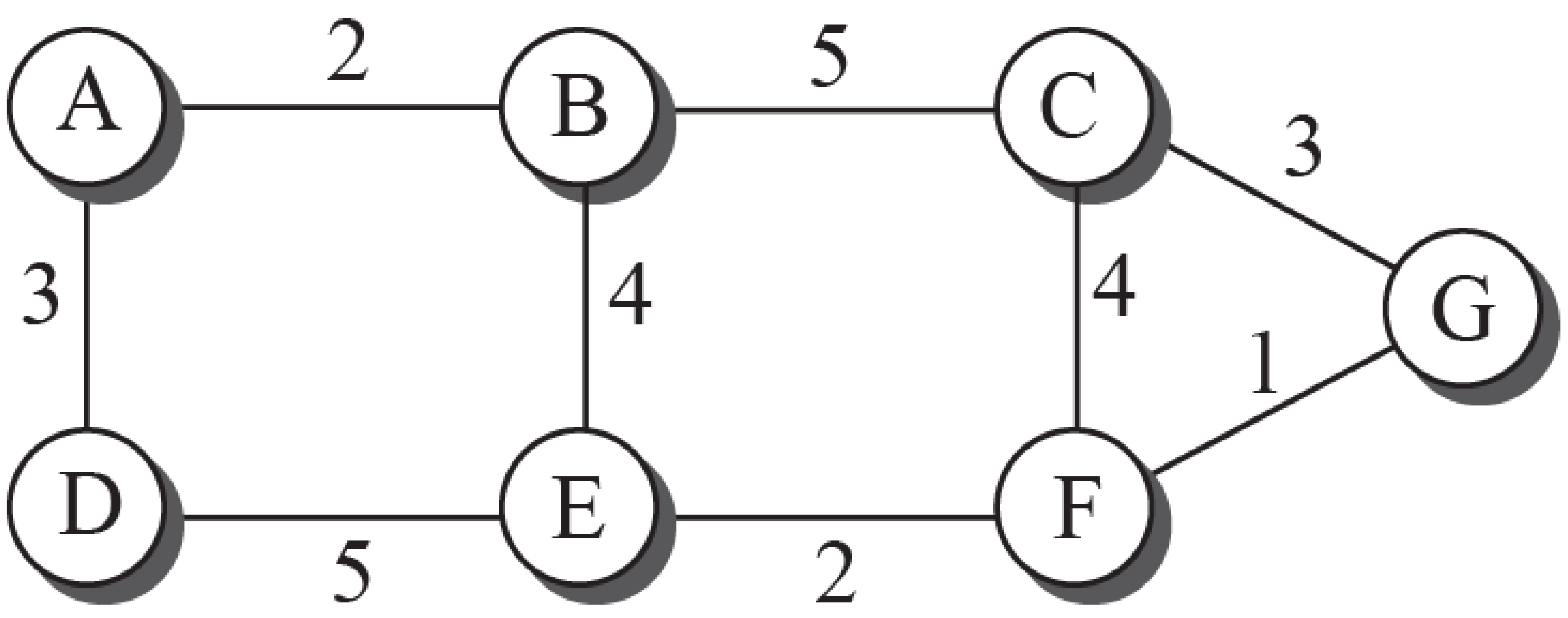
## Table 20.1: Distance-Vector Routing Algorithm for A Node

```
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] = ∞
      }
      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 Dw from a neighbor w or any change in the link)
            for (y = 1 to N)
            {
                  D[y] = min [D[y], (c[myself][w] + Dw[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
```

# 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.

# Figure 20.8: Example of a 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 |

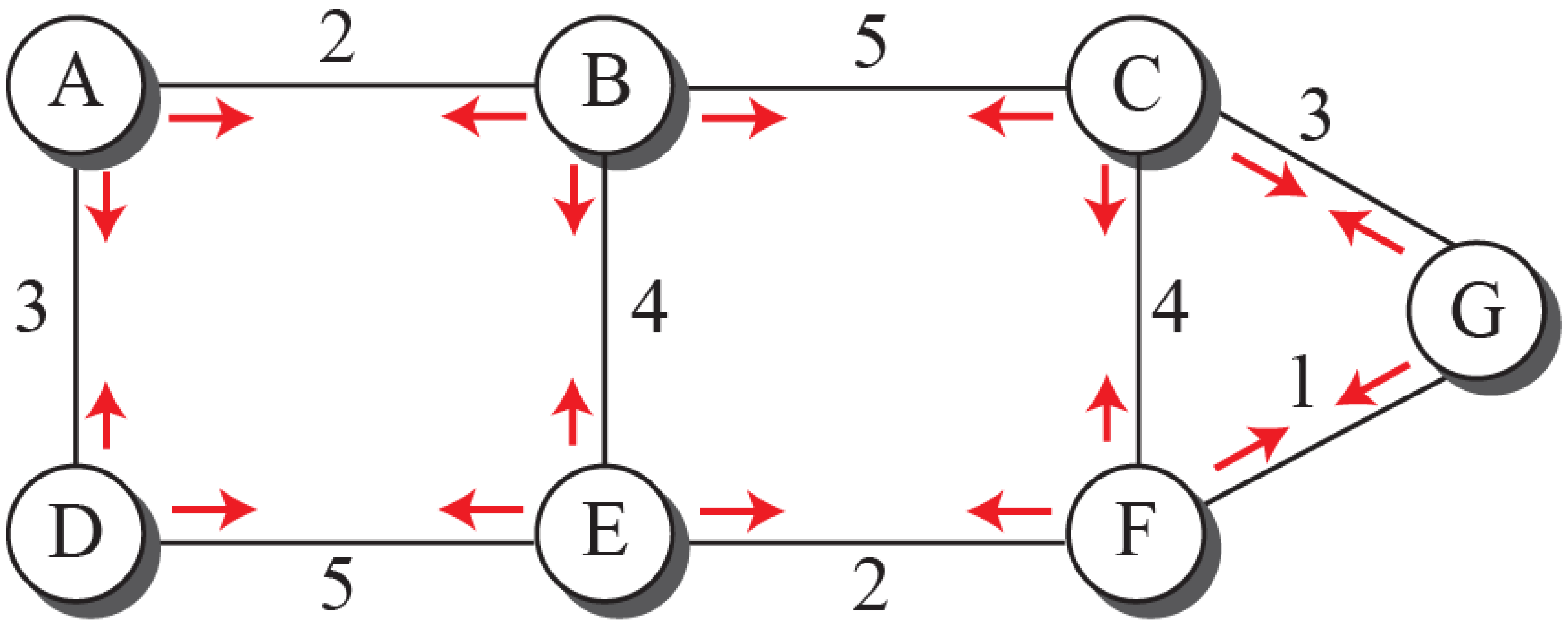
b. Link state database

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

| Node | Cost |
|---|---|
| B | 2 |
| D | 3 |

| Node | Cost |
|---|---|
| A | 2 |
| C | 5 |
| E | 4 |

| Node | Cost |
|---|---|
| B | 5 |
| F | 4 |
| G | 3 |




| Node | Cost |
|---|---|
| C | 3 |
| F | 1 |

| Node | Cost |
|---|---|
| A | 3 |
| E | 5 |

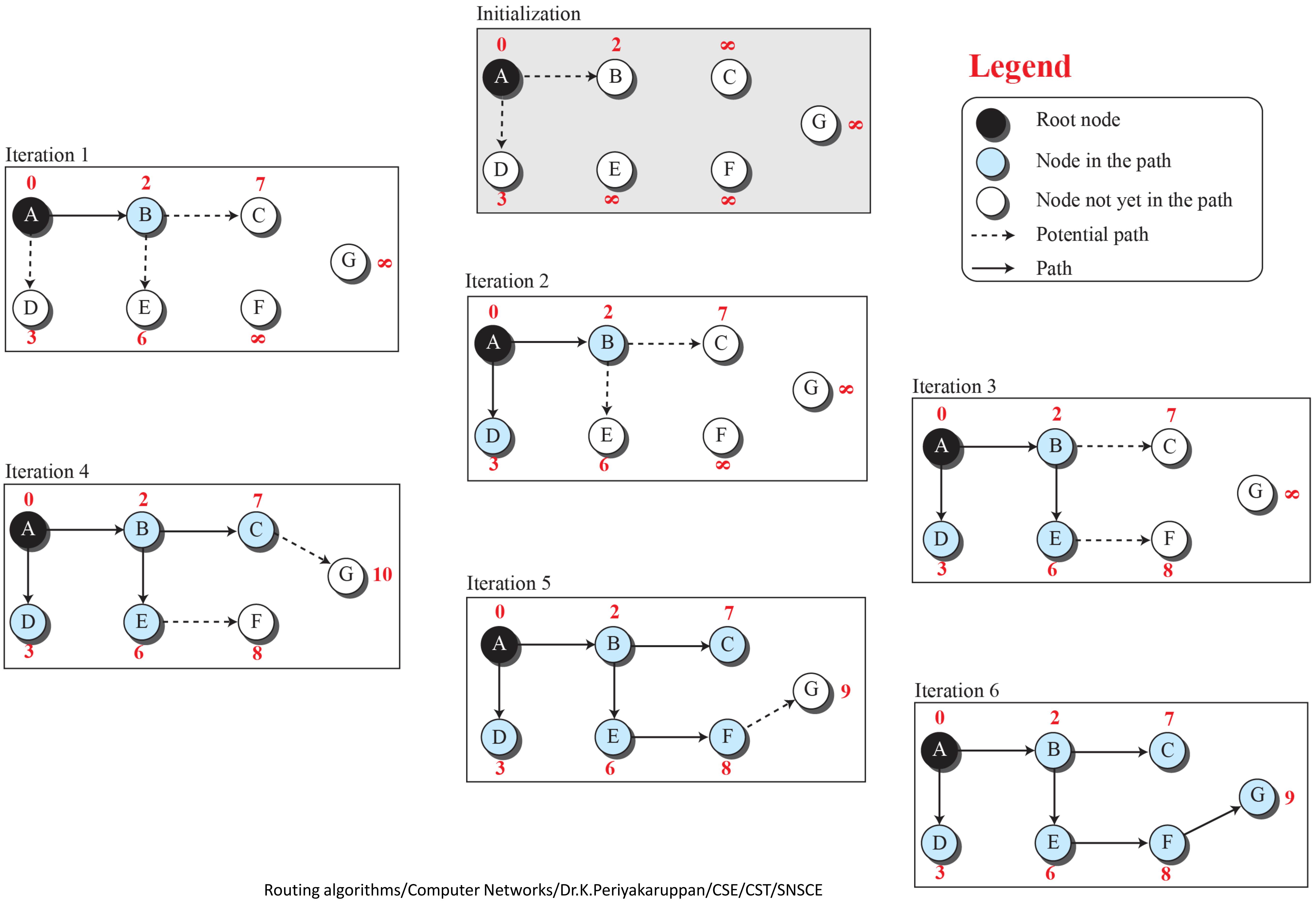
| Node | Cost |
|---|---|
| B | 4 |
| D | 5 |
| E | 2 |

| Node | Cost |
|---|---|
| C | 4 |
| E | 2 |
| G | 1 |

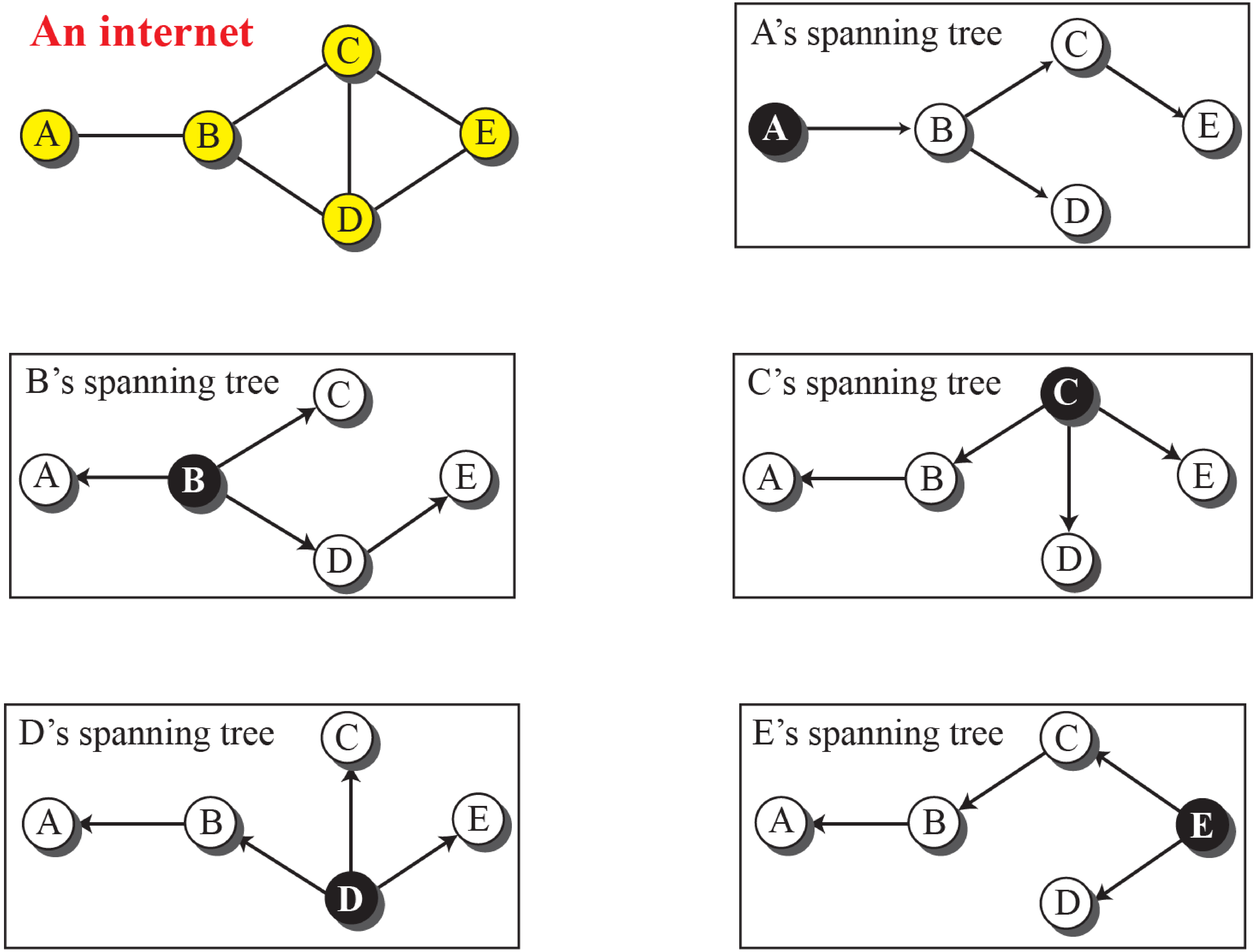
# Table 20.2: Dijkstra's Algorithm

```
Dijkstra's Algorithm ( )
{
    // Initialization
    Tree = {root}                       // Tree is made only of the root
    for (y = 1 to N)                    // N is the number of nodes
    {
        if (y is the root)
            D[y] = 0                    // D[y] is shortest distance from root to node y
        else if (y is a neighbor)
            D[y] = c[root][y]           // c[x][y] is cost between nodes x and y in LSDB
        else
            D[y] = ∞
    }
    // Calculation
    repeat
    {
        find a node w, with D[w] minimum among all nodes not in the Tree
        Tree = Tree ∪ {w}               // Add w to tree
        // Update distances for all neighbor of w
        for (every node x, which is neighbor of w and not in the Tree)
        {
            D[x] = min{D[x], (D[w] + c[w][x])}
        }
    } until (all nodes included in the Tree)
} // End of Dijkstra
```

## Figure 20.10: Least-cost tree



# Figure 20.11: Spanning trees in path-vector routing

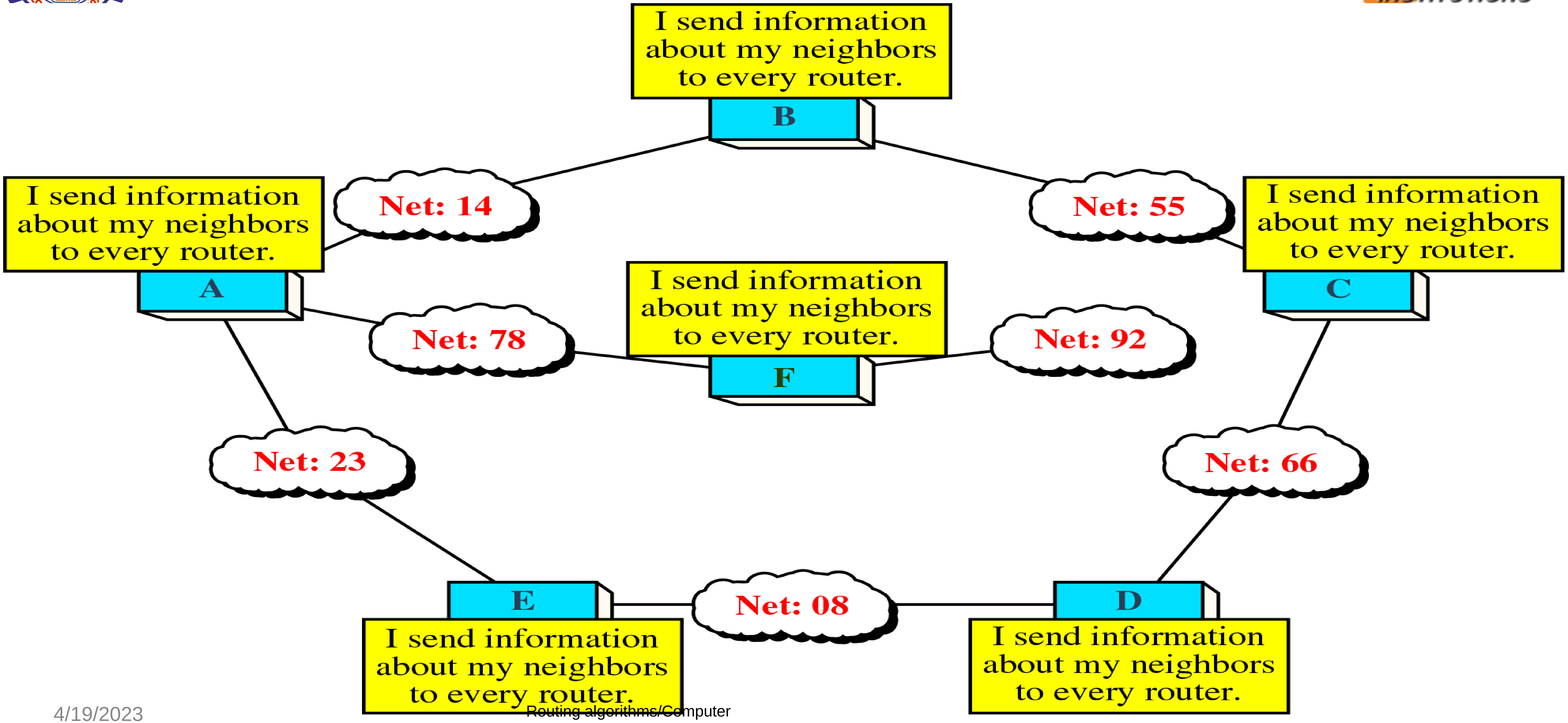


# Path-Vector Routing

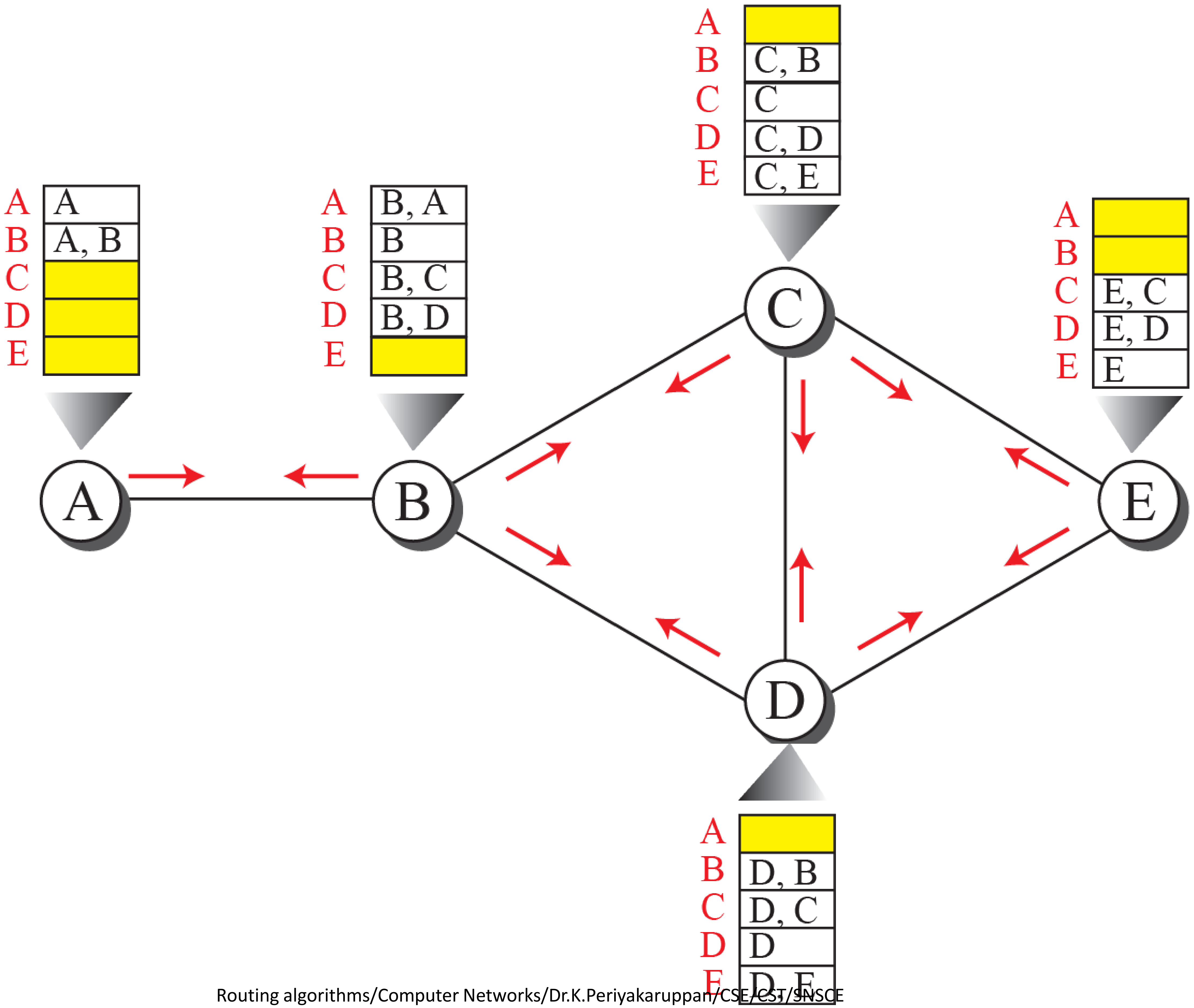
- Both link-state and distance-vector routing are based on the least-cost goal.
- However, there are instances where this goal is not the priority. For example, assume that there are some routers in the internet that a sender wants to prevent its packets from going through.
- In other words, the least-cost goal, applied by LS or DV routing, does not allow a sender to apply specific policies to the route a packet may take.
- To respond to these demands, a third routing algorithm, called path-vector (PV) routing has been devised.

Figure 21-24

# Concept of Link State Routing



# Path vectors made at booting time



# Updating path vectors

| | New C | Old C | B |
|---|---|---|---|
| A | C, B, A | | B, A |
| B | C, B | C, B | B |
| C | C | C | B, C |
| D | C, D | C, D | B, D |
| E | C, E | C, E | |

C[ ] = best (C[ ], C + B[ ])

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

**Note:**
**X [ ]: vector X**
**Y: node Y**

| | New C | Old C | D |
|---|---|---|---|
| A | C, B, A | C, B, A | |
| B | C, B | C, B | D, B |
| C | C | C | D, C |
| D | C, D | C, D | D |
| E | C, E | C, E | D, E |

C[ ] = best (C[ ], C + D[ ])

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

# Path-vector algorithm for a node

```
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
```

# Assessment

a).What is DVR algorithm?
b) What is LSR algorithm?
c) What is path vector routing algorithm?
d)Compare the above routing algorithms

# Reference

**TEXT BOOKS**

Behrouz A. Forouzan, Data Communications and Networking, Fifth Edition TMH, 2013.

**REFERENCES**


1. William Stallings, Data and Computer Communications, Tenth Edition, Pearson Education, 2013.
2. Andrew Tanenbaum, Computer Networks, Fifth Edition, Pearson (5th Edition) Education, 2013.
3. James F. Kurose, Keith W. Ross, Computer Networking, A Top-Down Approach Featuring the Internet, Sixth Edition, Pearson Education, 2013.
4. Larry L. Peterson, Bruce S. Davie, Computer Networks: A Systems Approach, Fifth Edition, Morgan Kaufmann Publishers Inc., 2012.