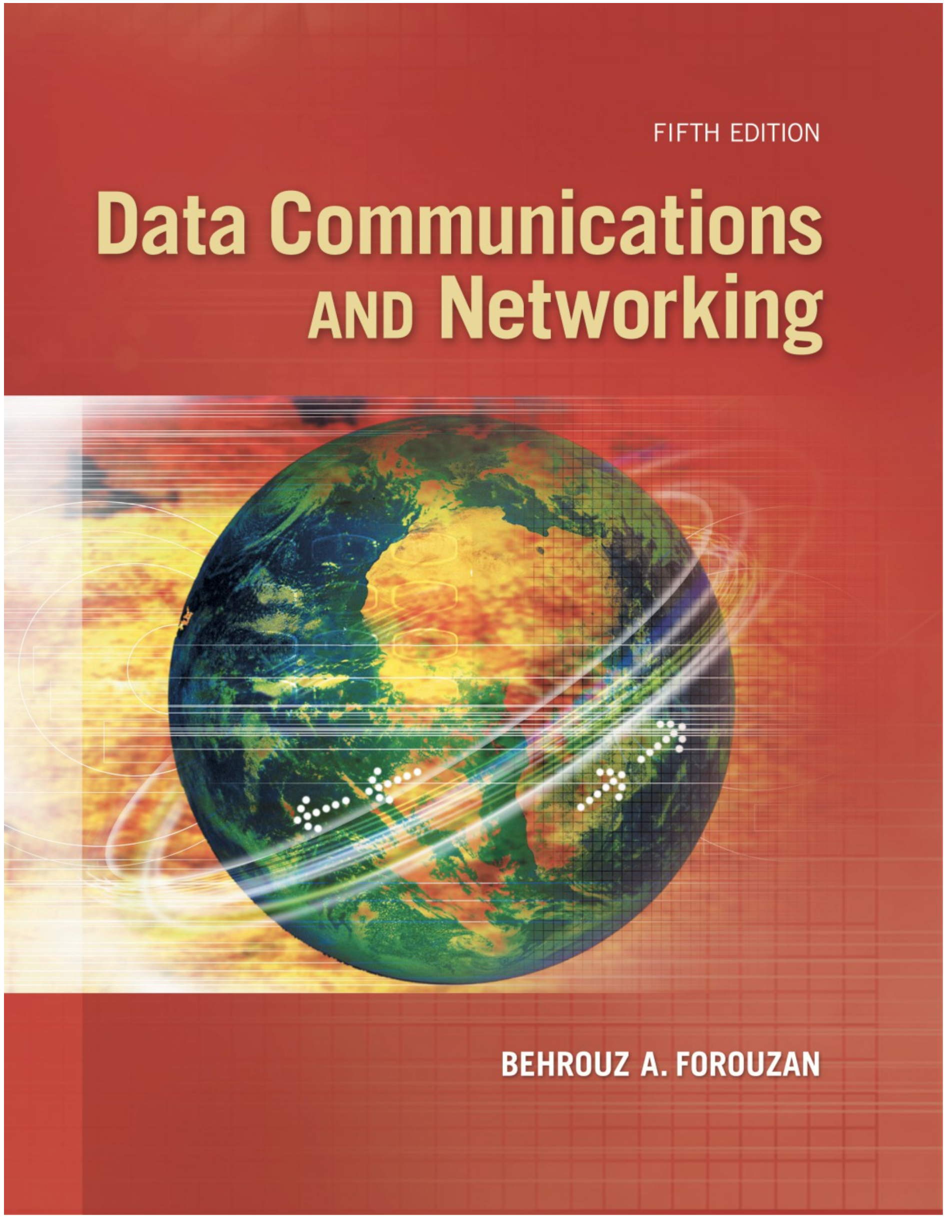
The McGraw-Hill Companies

Chapter 20

Unicast Routing



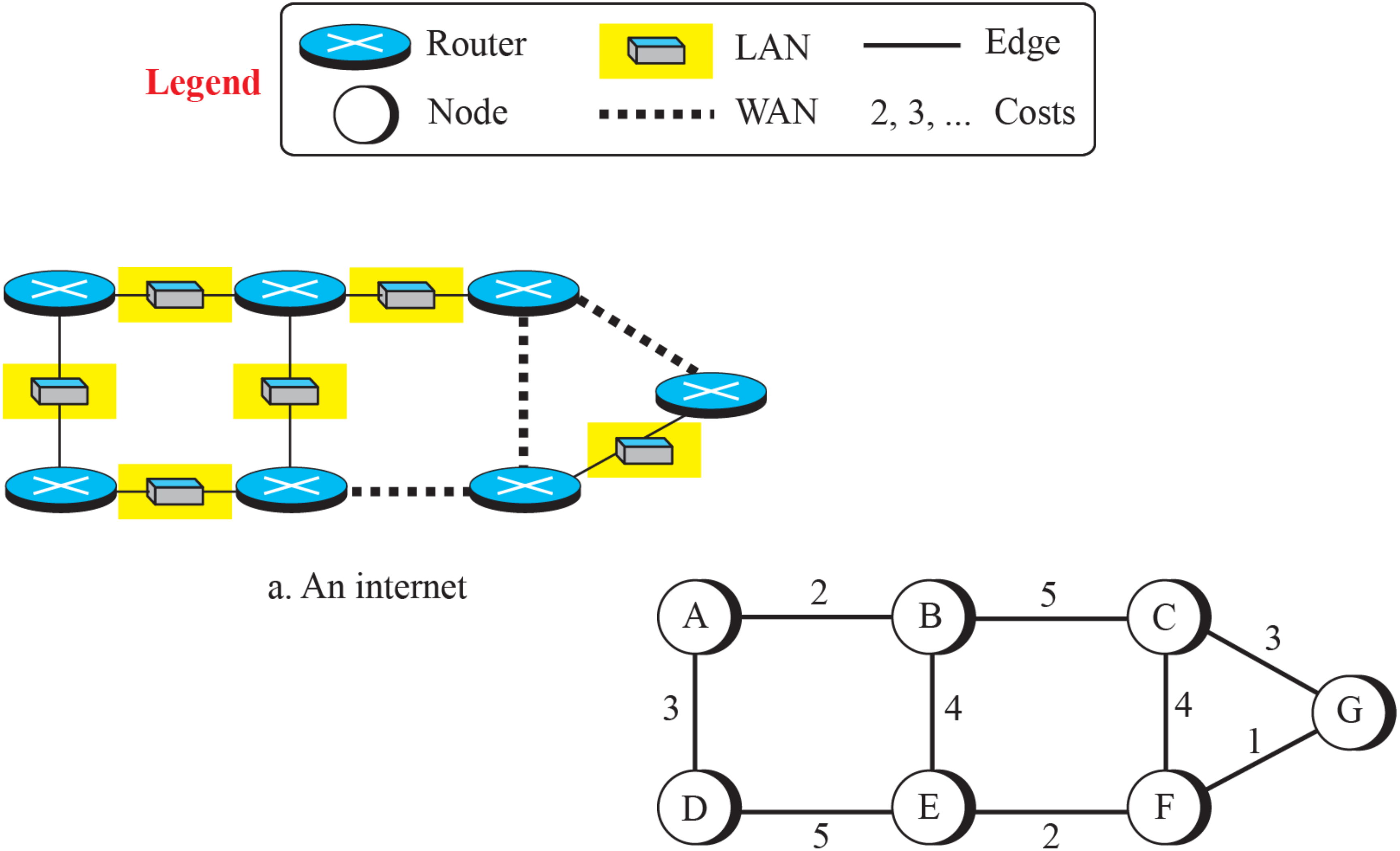
20.20.1 General Idea

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.

20.20.2 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.

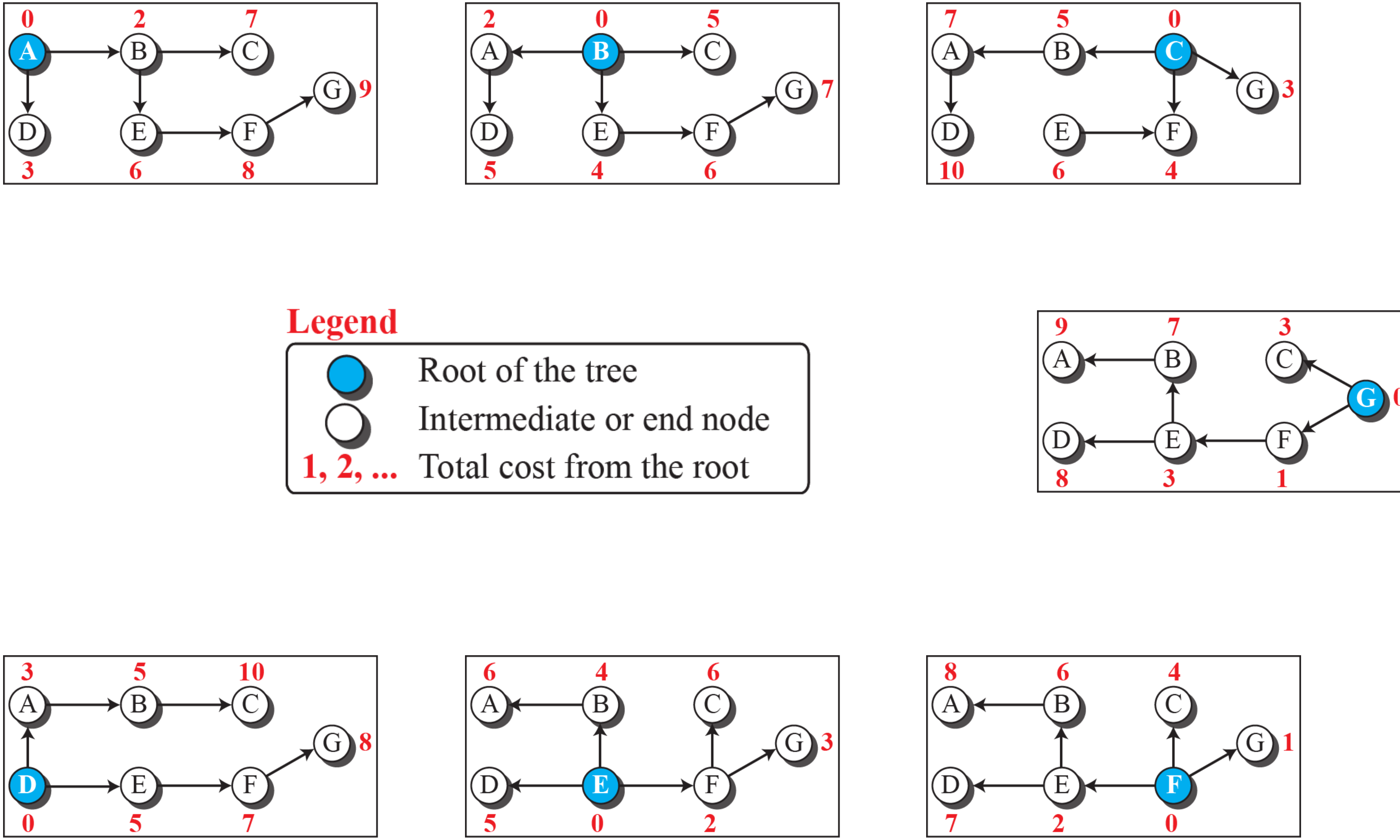
Figure 20.1: *An internet and its graphical representation*



a. An internet

b. The weighted graph

Figure 20.2: ***Least-cost trees for nodes in the internet of Figure 4.56***



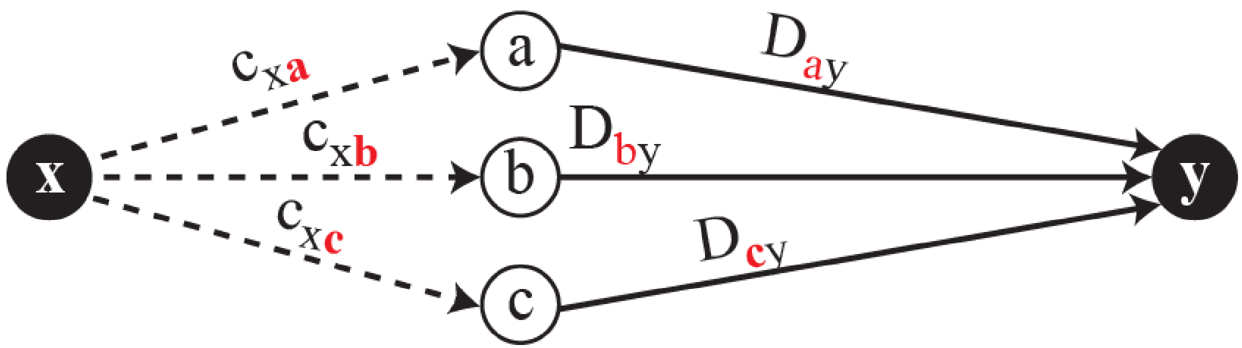
20-2 ROUTING ALGORITHMS

Several routing algorithms have been designed in the past. The differences between these methods are in the way they interpret the least cost and the way they create the least-cost tree for each node. In this section, we discuss the common algorithms; later we show how a routing protocol in the Internet implements one of these algorithms.

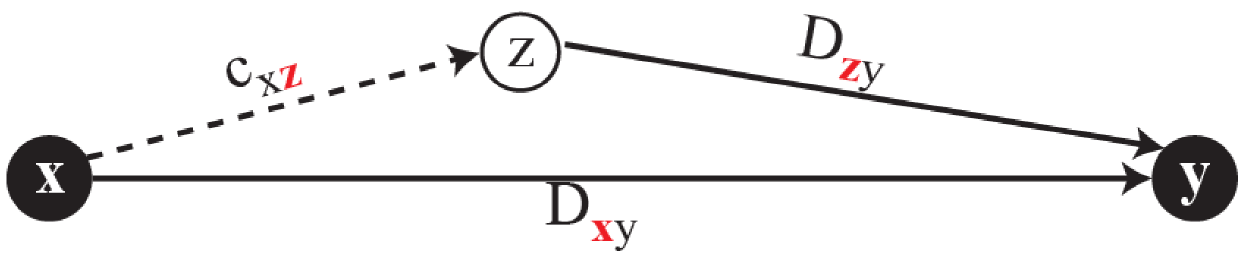
20.2.1 Distance-Vector Routing

The distance-vector (DV) routing uses the goal we discussed in the introduction, to find the best route. 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 (although the knowledge can be incomplete).

Figure 20.3: Graphical idea behind Bellman-Ford equation

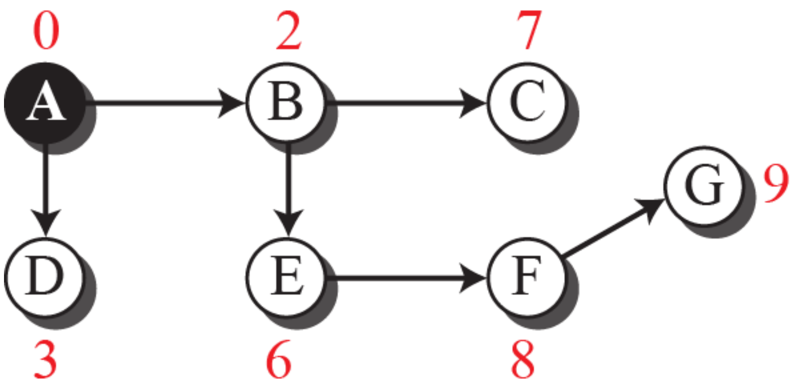


a. General case with three intermediate nodes



b. Updating a path with a new route

Figure 20.4: The distance vector corresponding to a tree



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

Figure 20.5: The first distance vector for an internet

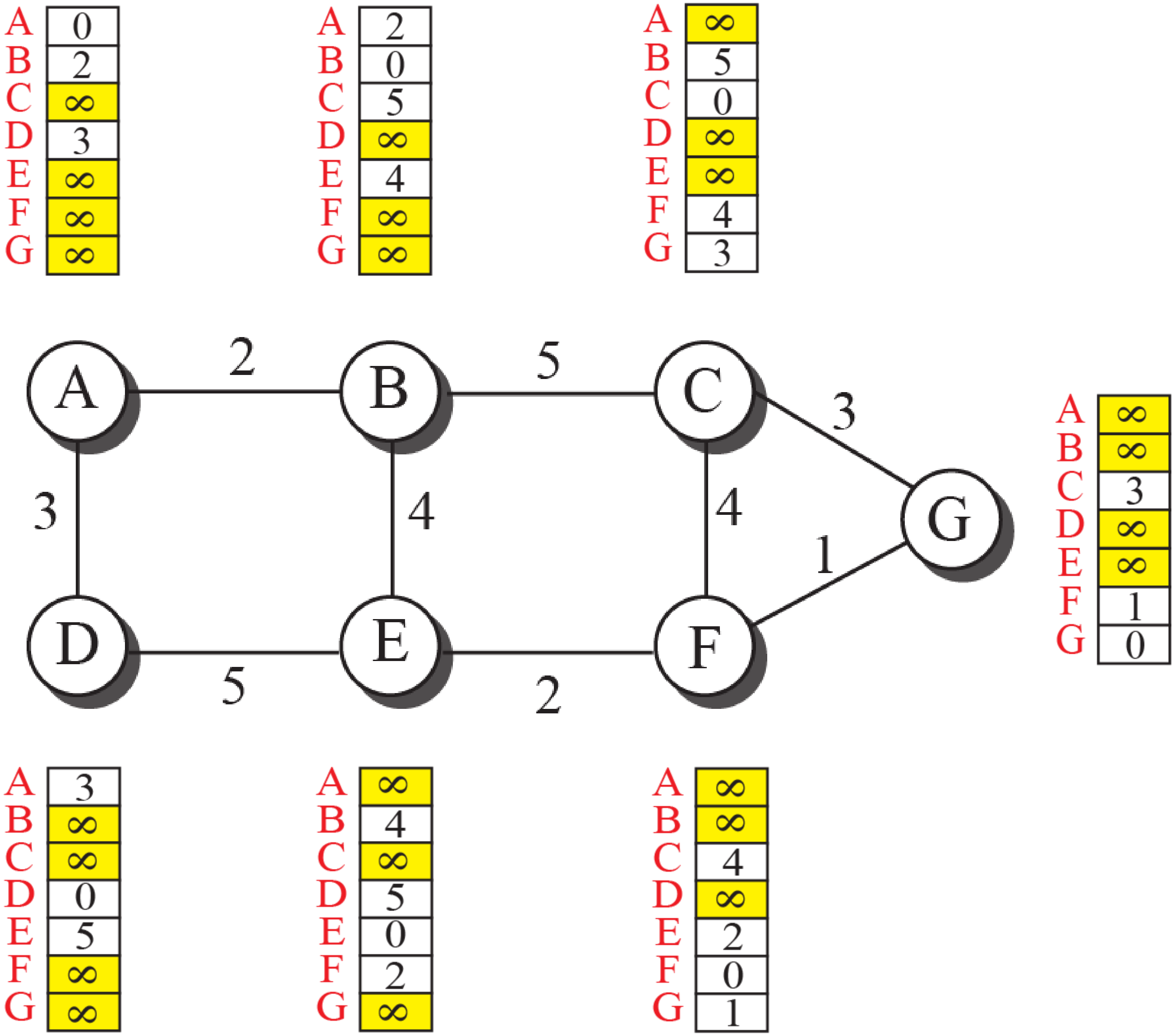


Figure 20.6: Updating distance vectors

	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
```

Figure 20.7: Two-node instability

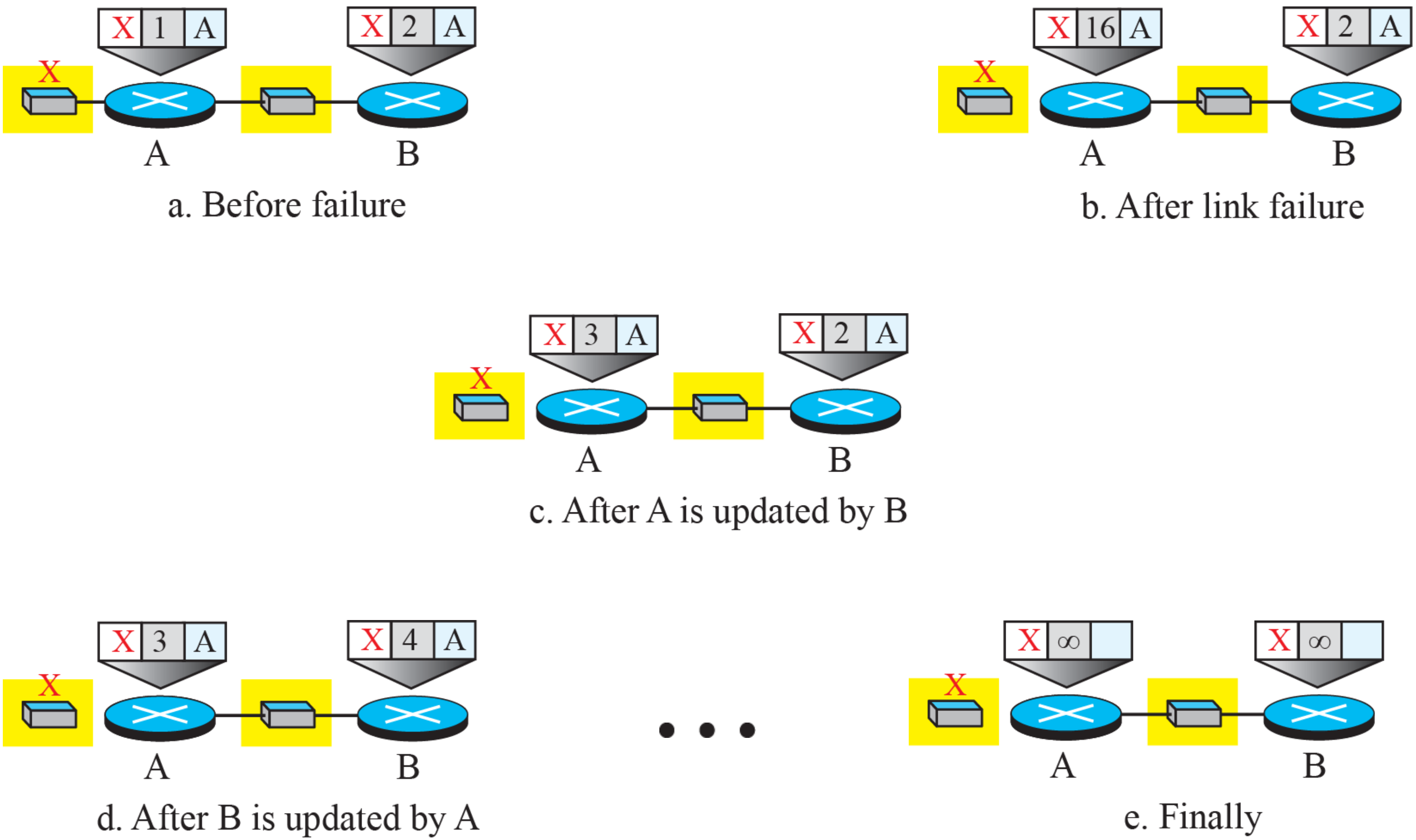


Figure *Two-node instability*

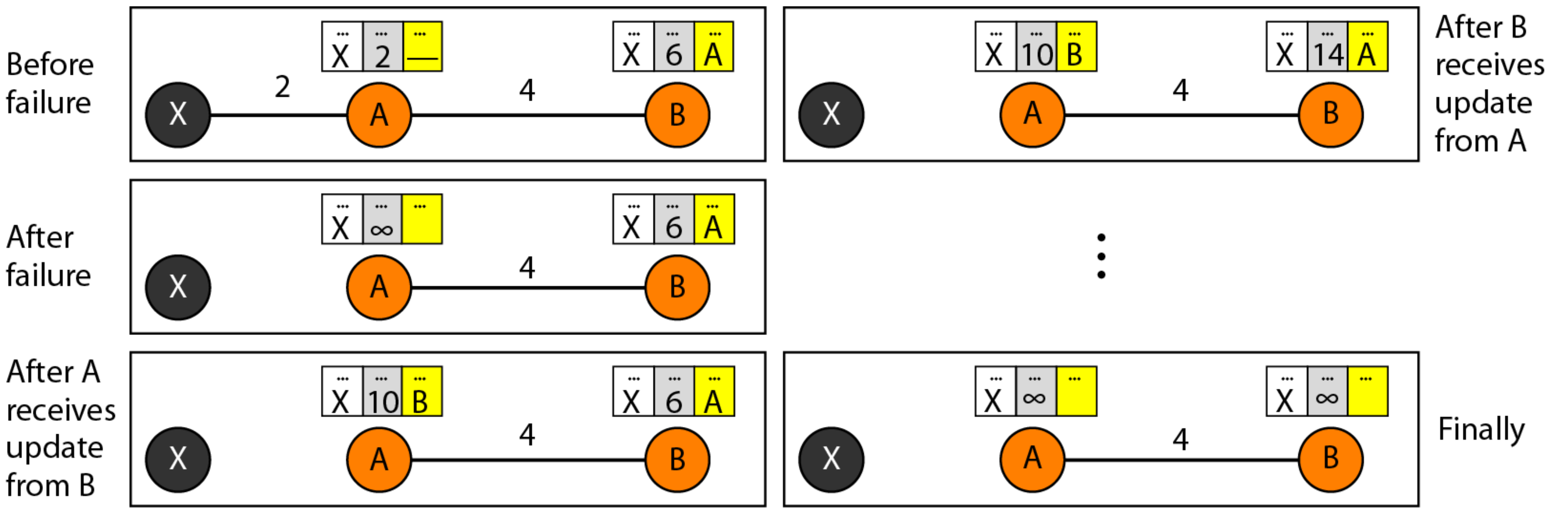


Figure *Three-node instability*

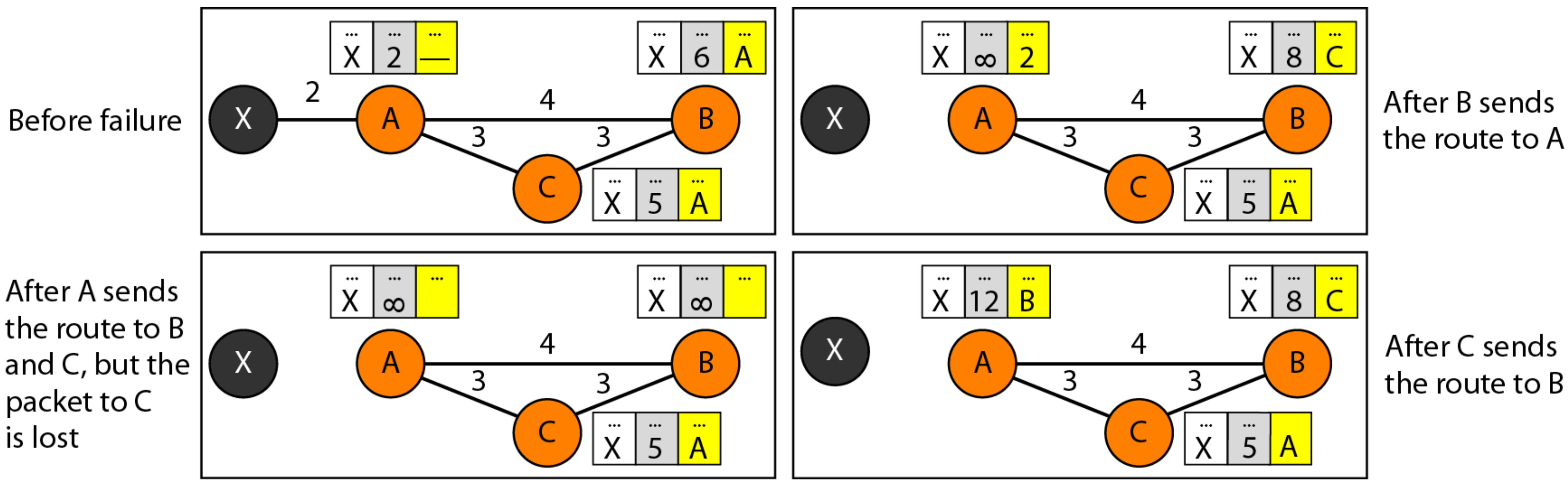
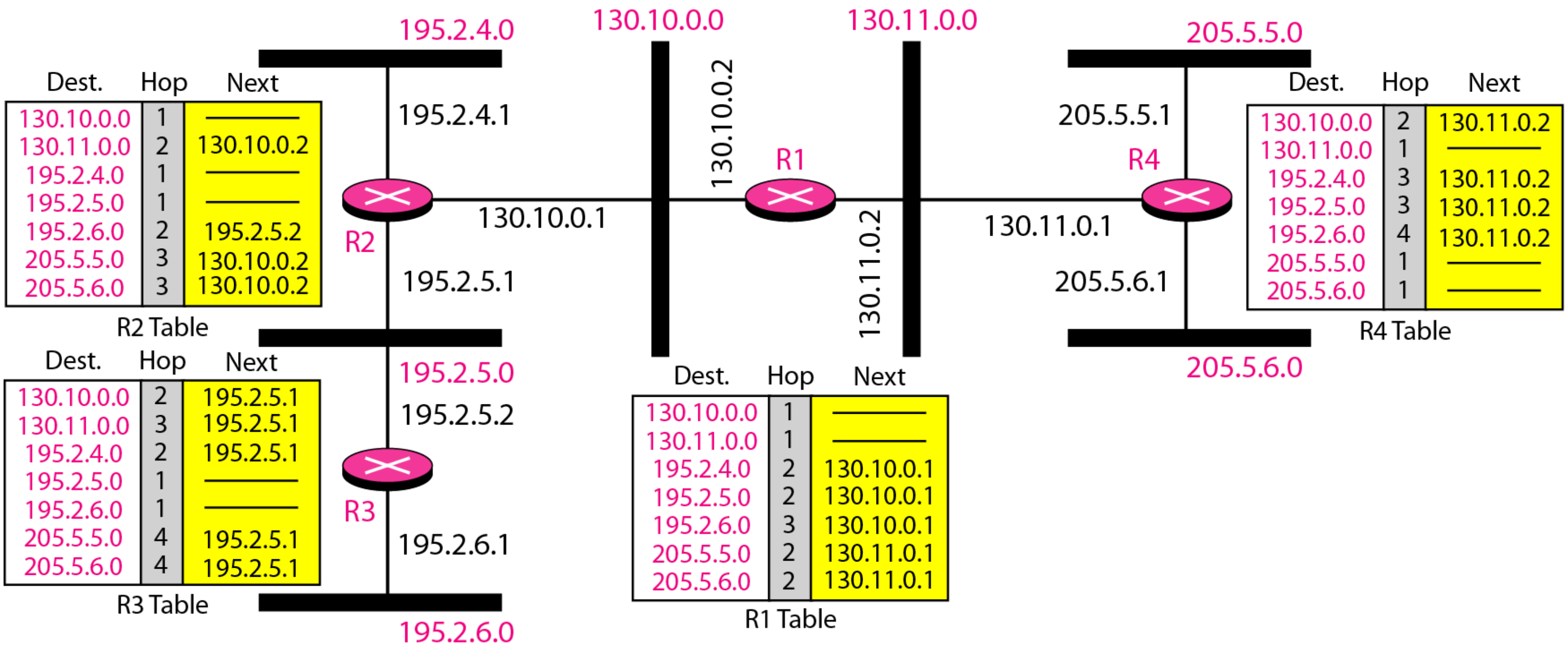


Figure *Example of a domain using RIP*



Dest.	Hop	Next
130.10.0.0	1	—
130.11.0.0	2	130.10.0.2
195.2.4.0	1	—
195.2.5.0	1	—
195.2.6.0	2	195.2.5.2
205.5.5.0	3	130.10.0.2
205.5.6.0	3	130.10.0.2

R2 Table

Dest.	Hop	Next
130.10.0.0	2	195.2.5.1
130.11.0.0	3	195.2.5.1
195.2.4.0	2	195.2.5.1
195.2.5.0	1	—
195.2.6.0	1	—
205.5.5.0	4	195.2.5.1
205.5.6.0	4	195.2.5.1

R3 Table

Dest.	Hop	Next
130.10.0.0	1	—
130.11.0.0	1	—
195.2.4.0	2	130.10.0.1
195.2.5.0	2	130.10.0.1
195.2.6.0	3	130.10.0.1
205.5.5.0	2	130.11.0.1
205.5.6.0	2	130.11.0.1

R1 Table

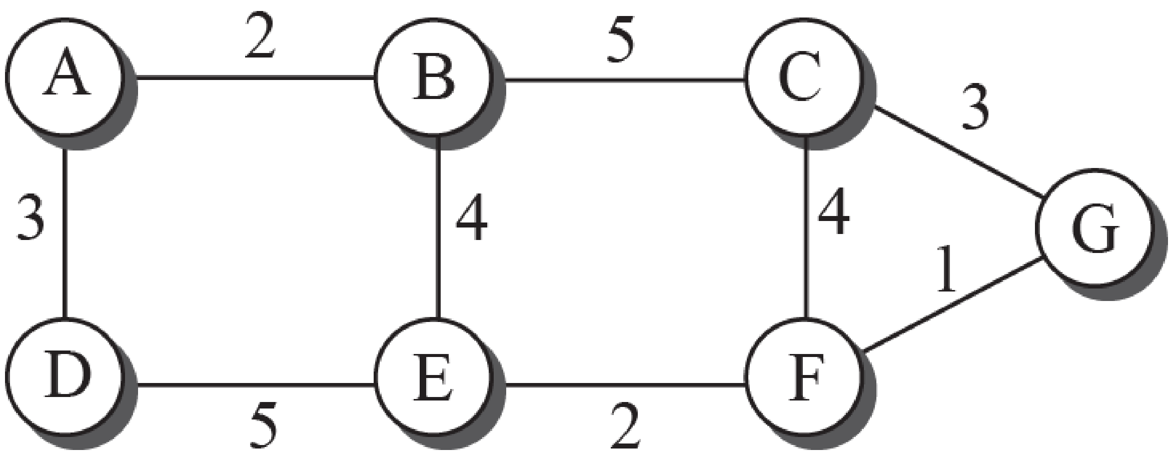
Dest.	Hop	Next
130.10.0.0	2	130.11.0.2
130.11.0.0	1	—
195.2.4.0	3	130.11.0.2
195.2.5.0	3	130.11.0.2
195.2.6.0	4	130.11.0.2
205.5.5.0	1	—
205.5.6.0	1	—

R4 Table

20.2.2 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. 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

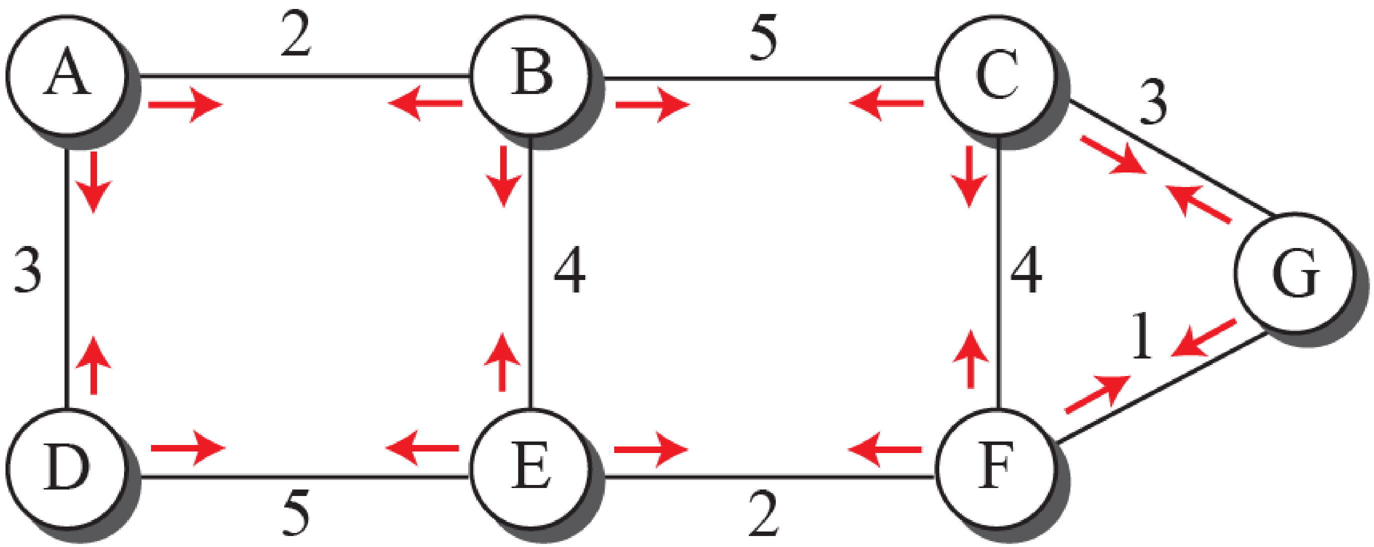
b. Link state database

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

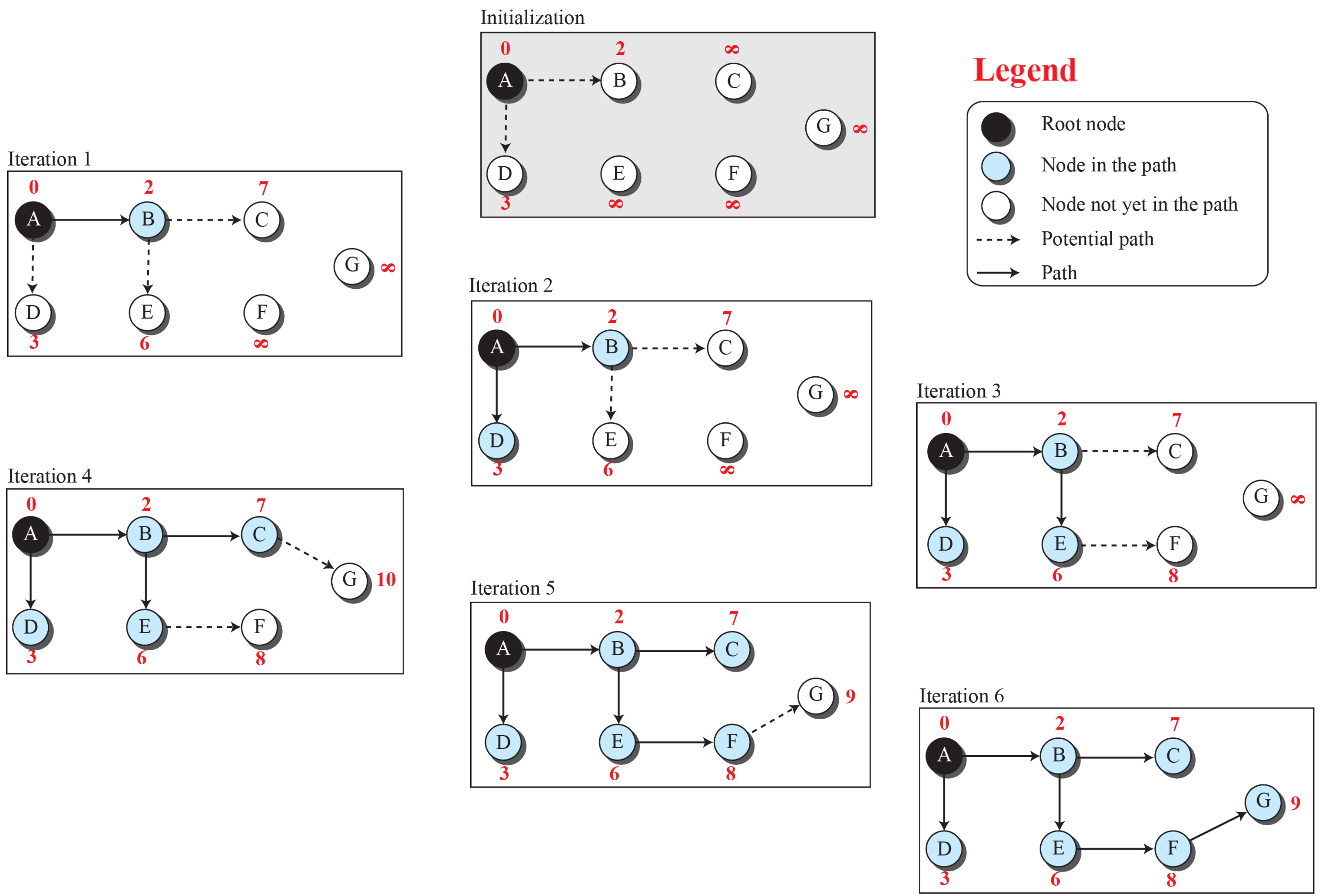
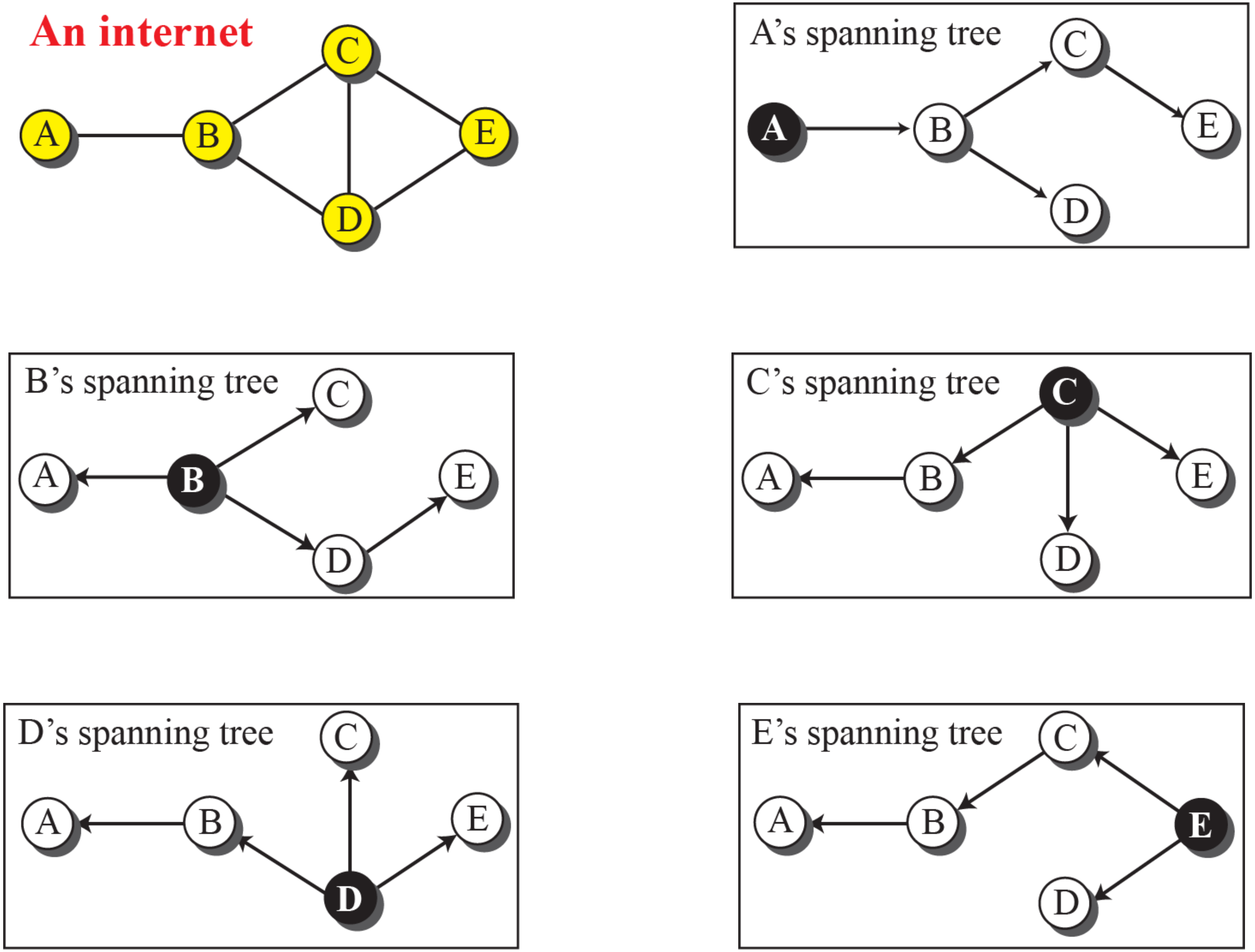


Figure 20.11: Spanning trees in path-vector routing



20.2.3 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 20.12: Path vectors made at booting time

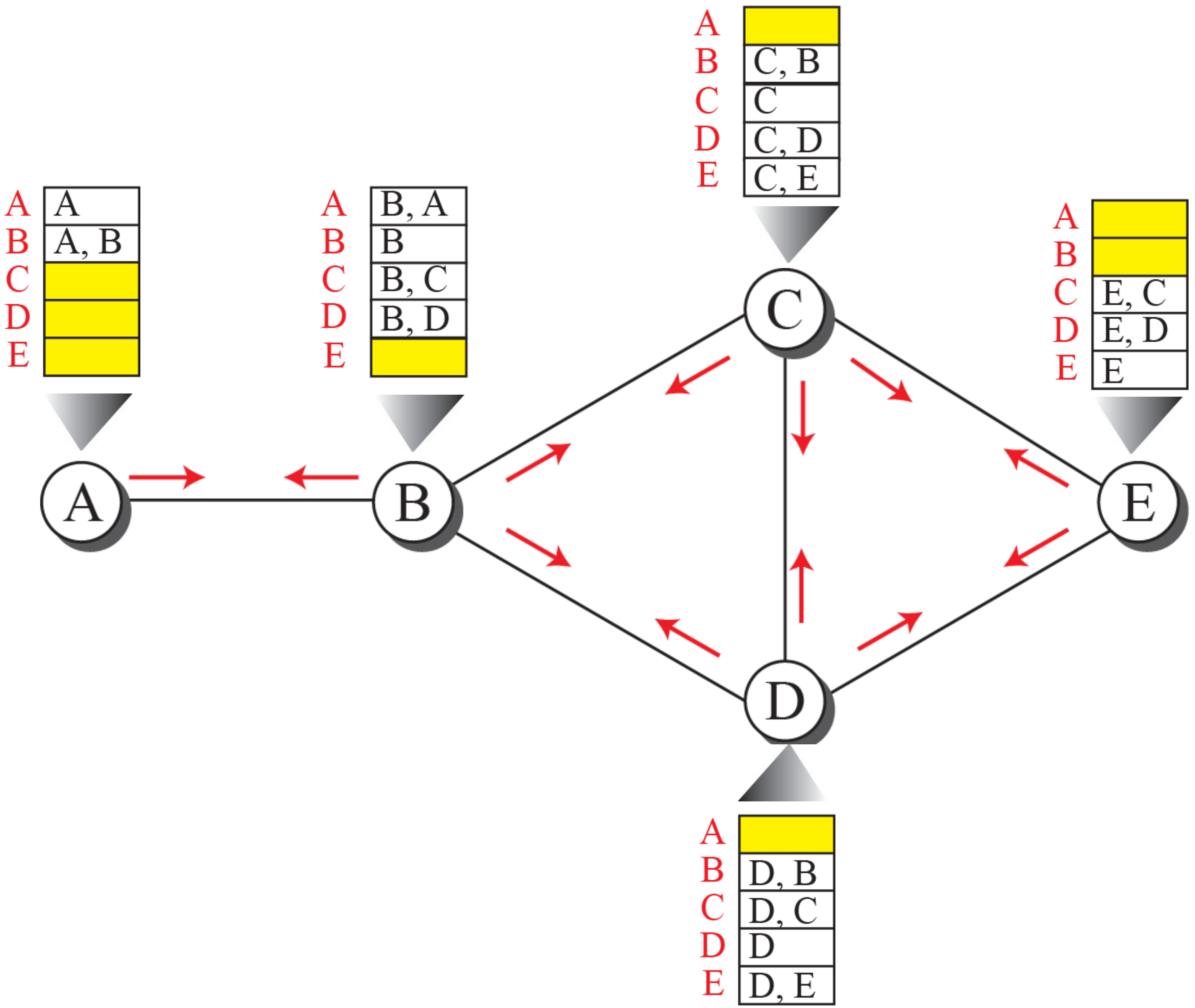


Figure 20.13: *Updating path vectors*

	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

Table 20.3: 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
```

20-3 UNICAST ROUTING PROTOCOLS

After an introduction, we discuss three common protocols used in the Internet: Routing Information Protocol (RIP), based on the distance-vector algorithm, Open Shortest Path First (OSPF), based on the link-state algorithm, and Border Gateway Protocol (BGP), based on the path-vector algorithm.

20.3.1 Internet Structure

Before discussing unicast routing protocols, we need to understand the structure of today's Internet. The Internet has changed from a tree-like structure, with a single backbone, to a multi-backbone structure run by different private corporations today. Although it is difficult to give a general view of the Internet today, we can say that the Internet has a structure similar to what is shown in Figure 20.14.

20.3.2 Routing Information Protocol

The Routing Information Protocol (RIP) is one of the most widely used intradomain routing protocols based on the distance-vector routing algorithm we described earlier. RIP was started as part of the Xerox Network System (XNS), but it was the Berkeley Software Distribution (BSD) version of UNIX that helped make the use of RIP widespread.

Figure 20.14: Internet structure

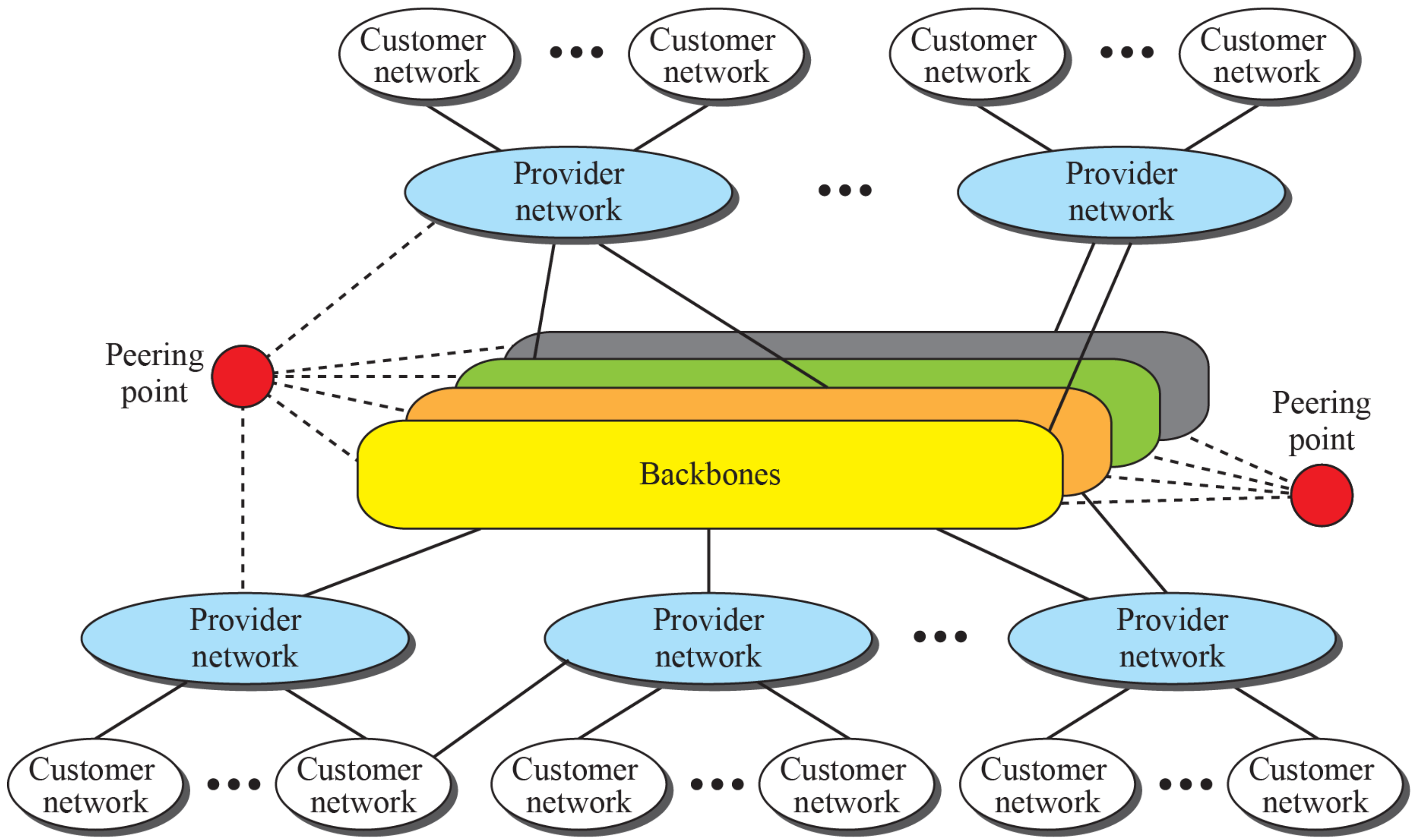


Figure 20.15: Hop counts in RIP

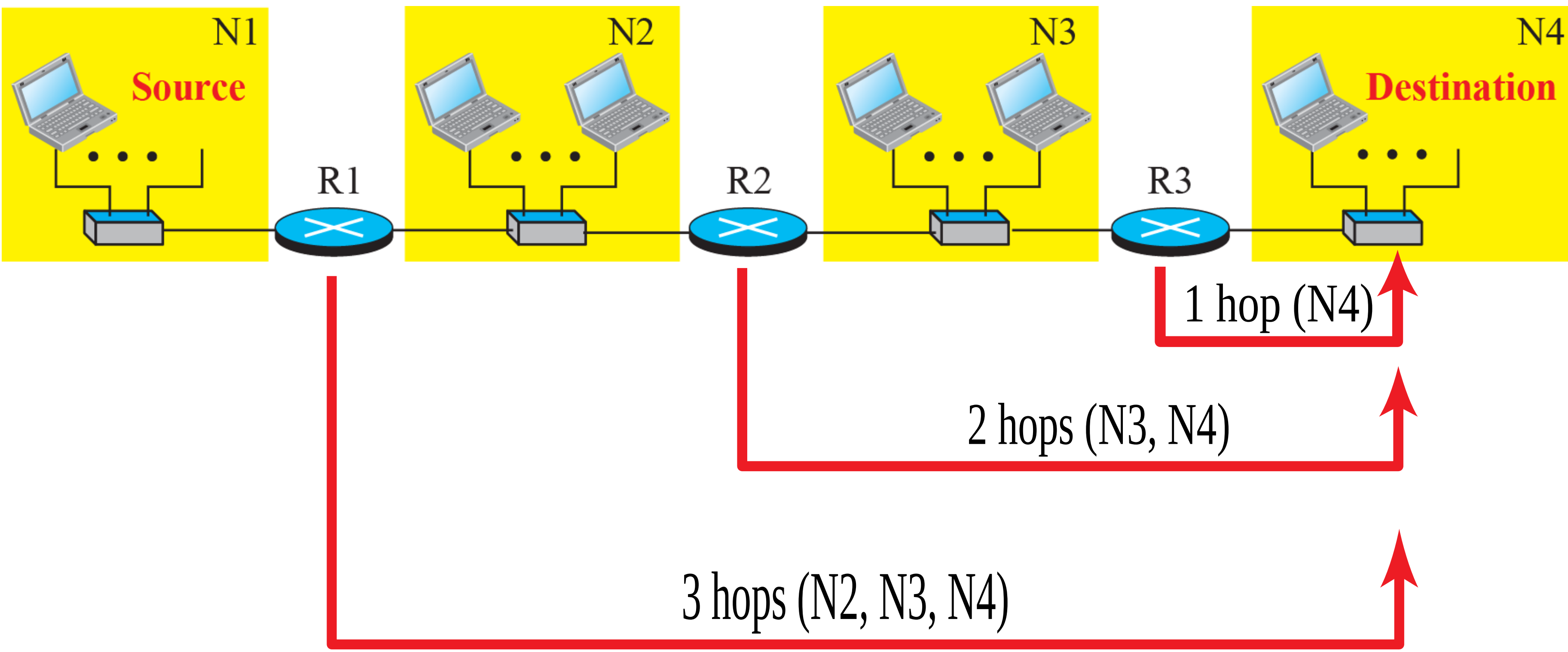


Figure 20.16: Forwarding tables

Forwarding table for R1

Destination network	Next router	Cost in hops
N1	——	1
N2	——	1
N3	R2	2
N4	R2	3

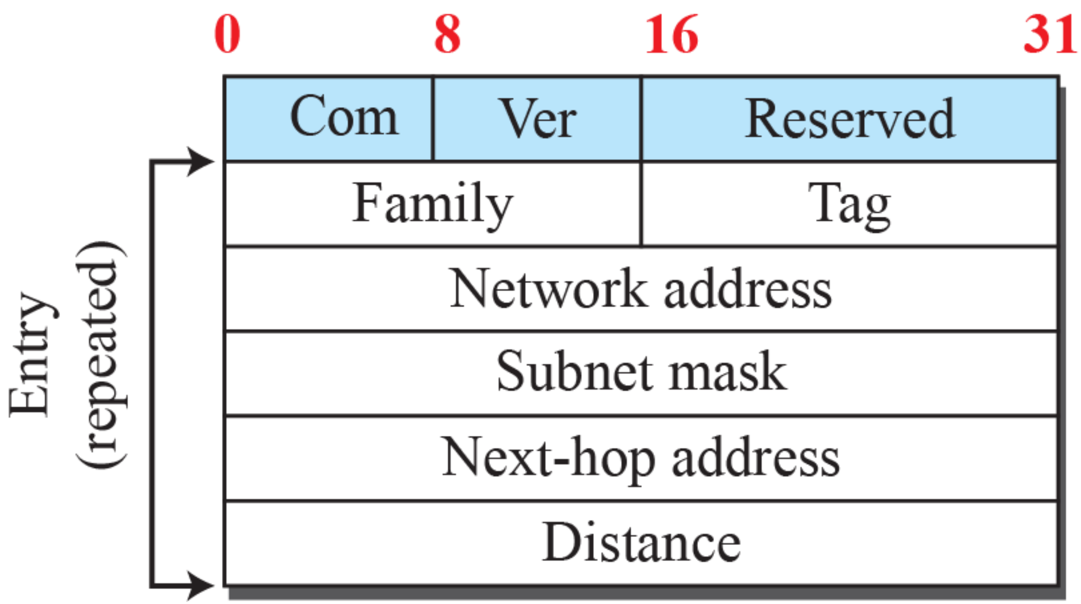
Forwarding table for R2

Destination network	Next router	Cost in hops
N1	R1	2
N2	——	1
N3	——	1
N4	R3	2

Forwarding table for R3

Destination network	Next router	Cost in hops
N1	R2	3
N2	R2	2
N3	——	1
N4	——	1

Figure 20.17: RIP message format



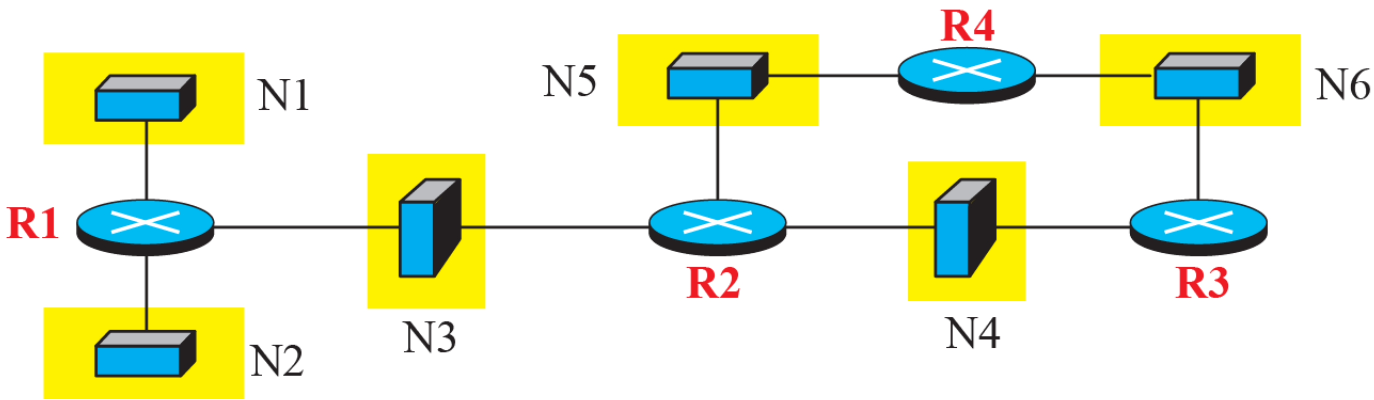
Fields

Com: Command, request (1), response (2)
Ver: Version, current version is 2
Family: Family of protocol, for TCP/IP value is 2
Tag: Information about autonomous system
Network address: Destination address
Subnet mask: Prefix length
Next-hop address: Address length
Distance: Number of hops to the destination

Example 20.1

Figure 20.18 shows a more realistic example of the operation of RIP in an autonomous system. First, the figure shows all forwarding tables after all routers have been booted. Then we show changes in some tables when some update messages have been exchanged. Finally, we show the stabilized forwarding tables when there is no more change.

Figure 20.18: Example of an autonomous system using RIP (Part I)



Legend

Des.: Destination network
N. R.: Next router
Cost: Cost in hops

R1

Des.	N. R.	Cost
N1	———	1
N2	———	1
N3	———	1

R2

Des.	N. R.	Cost
N3	———	1
N4	———	1
N5	———	1

R3

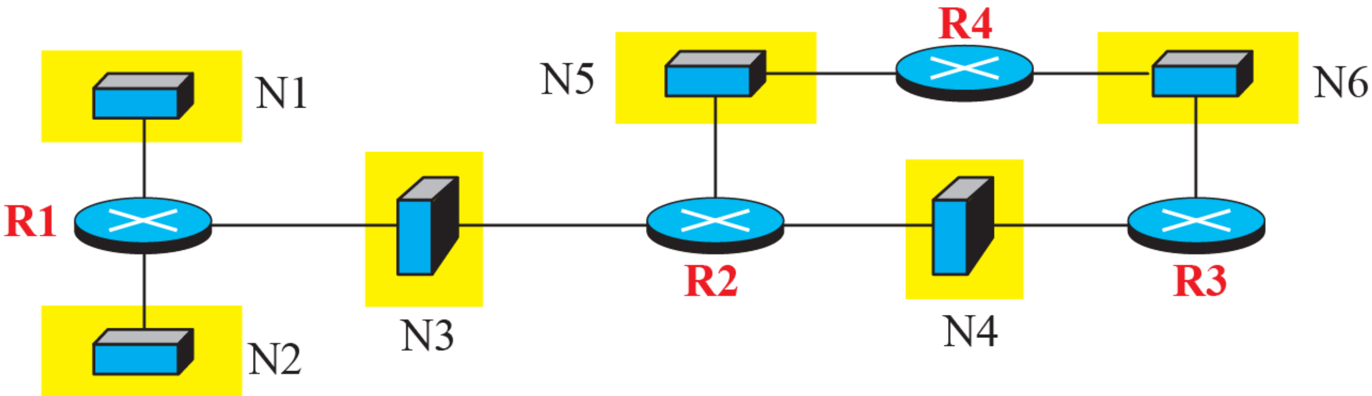
Des.	N. R.	Cost
N4	———	1
N6	———	1

R4

Des.	N. R.	Cost
N5	———	1
N6	———	1

Forwarding tables after all routers booted

Figure 20.18: Example of an autonomous system using RIP (Part II)



Legend

Des.: Destination network
N. R.: Next router
Cost: Cost in hops
← : New route
← : Old route

New R1

Des.	N. R.	Cost
N1	—	1
N2	—	1
N3	—	1
N4	R2	2
N5	R2	2

Old R1

Des.	N. R.	Cost
N1	—	1
N2	—	1
N3	—	1

R2 Seen by R1

Des.	N. R.	Cost
N3	R2	2
N4	R2	2
N5	R2	2

New R3

Des.	N. R.	Cost
N3	R2	2
N4	—	1
N5	R2	2
N6	—	1

Old R3

Des.	N. R.	Cost
N4	—	1
N6	—	1

R2 Seen by R3

Des.	N. R.	Cost
N3	R2	2
N4	R2	2
N5	R2	2

New R4

Des.	N. R.	Cost
N3	R2	2
N4	R2	2
N5	—	1
N6	—	1

Old R4

Des.	N. R.	Cost
N5	—	1
N6	—	1

R2 Seen by R4

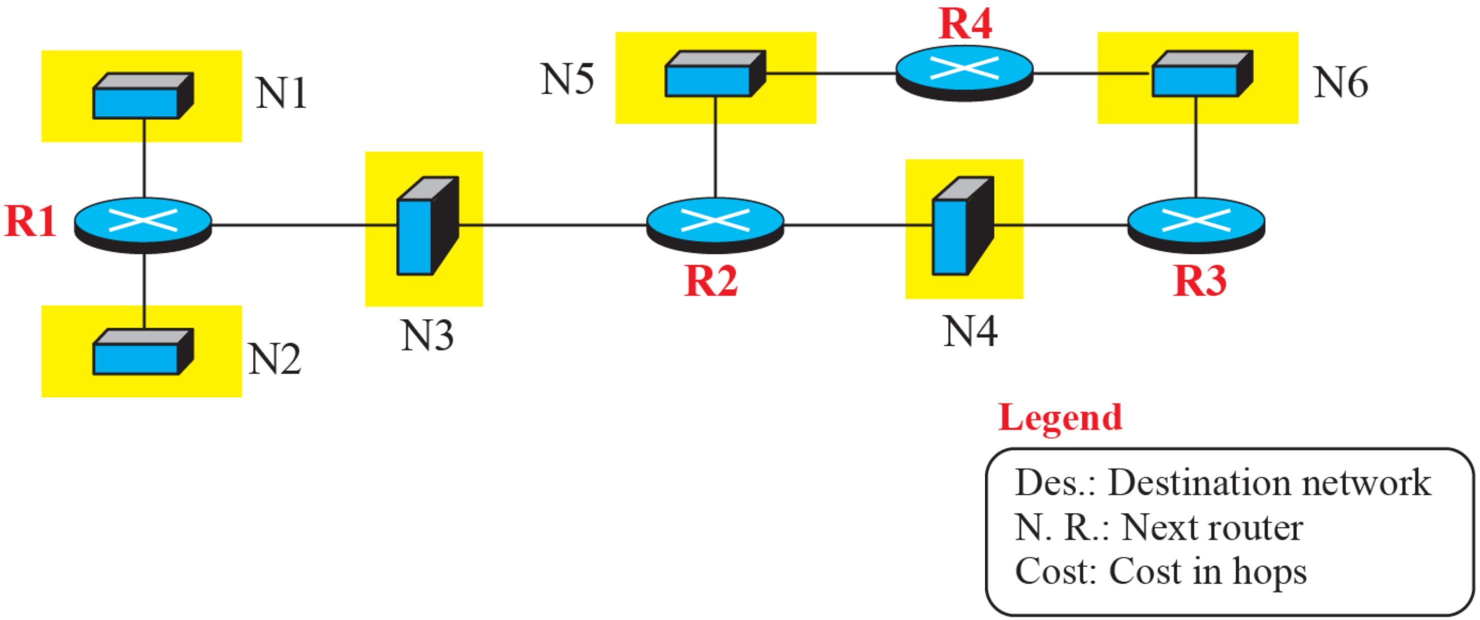
Des.	N. R.	Cost
N3	R2	2
N4	R2	2
N5	R2	2

Changes in the forwarding tables of R1, R3, and R4 after they receive a copy of R2's table

20.3.3 Open Shortest Path First

Open Shortest Path First (OSPF) is also an intradomain routing protocol like RIP, but it is based on the link-state routing protocol we described earlier in the chapter. OSPF is an open protocol, which means that the specification is a public document.

Figure 4.73: Example of an autonomous system using RIP (Part III)



Forwarding tables for all routers
after they have been stablized

Final R1

Des.	N. R.	Cost
N1	———	1
N2	———	1
N3	———	1
N4	R2	2
N5	R2	2
N6	R2	3

Final R2

Des.	N. R.	Cost
N1	R1	2
N2	R1	2
N3	———	1
N4	———	1
N5	———	1
N6	R3	2

Final R3

Des.	N. R.	Cost
N1	R2	3
N2	R2	3
N3	R2	2
N4	———	1
N5	R2	2
N6	———	1

Final R4

Des.	N. R.	Cost
N1	R2	3
N2	R2	3
N3	R2	2
N4	R2	2
N5	———	1
N6	———	1

Figure 20.19: Metric in OSPF

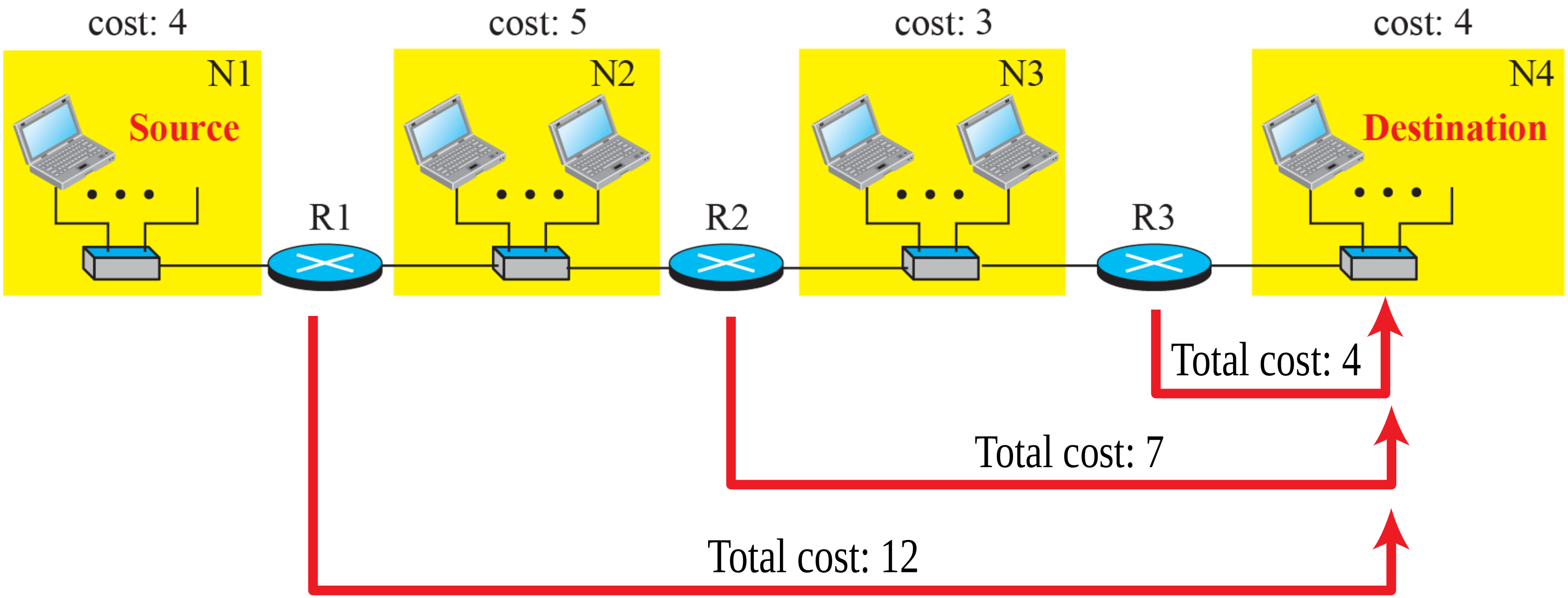
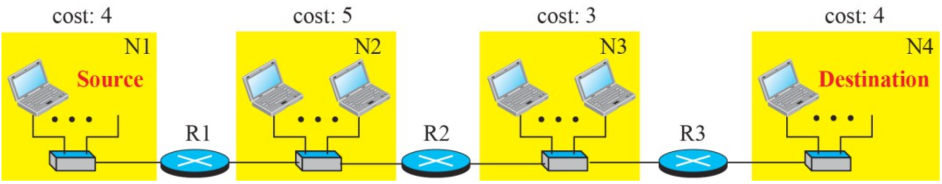


Figure 20.20: Forwarding tables in OSPF

Forwarding table for R1

Destination network	Next router	Cost
N1	——	
N2	——	
N3	R2	
N4	R2	12



The internet from previous figure

Forwarding table for R2

Destination network	Next router	Cost
N1	R1	9
N2	——	5
N3	——	3
N4	R3	7

Forwarding table for R3

Destination network	Next router	Cost
N1	R2	12
N2	R2	8
N3	——	3
N4	——	4

Figure 20.21: Areas in an autonomous system

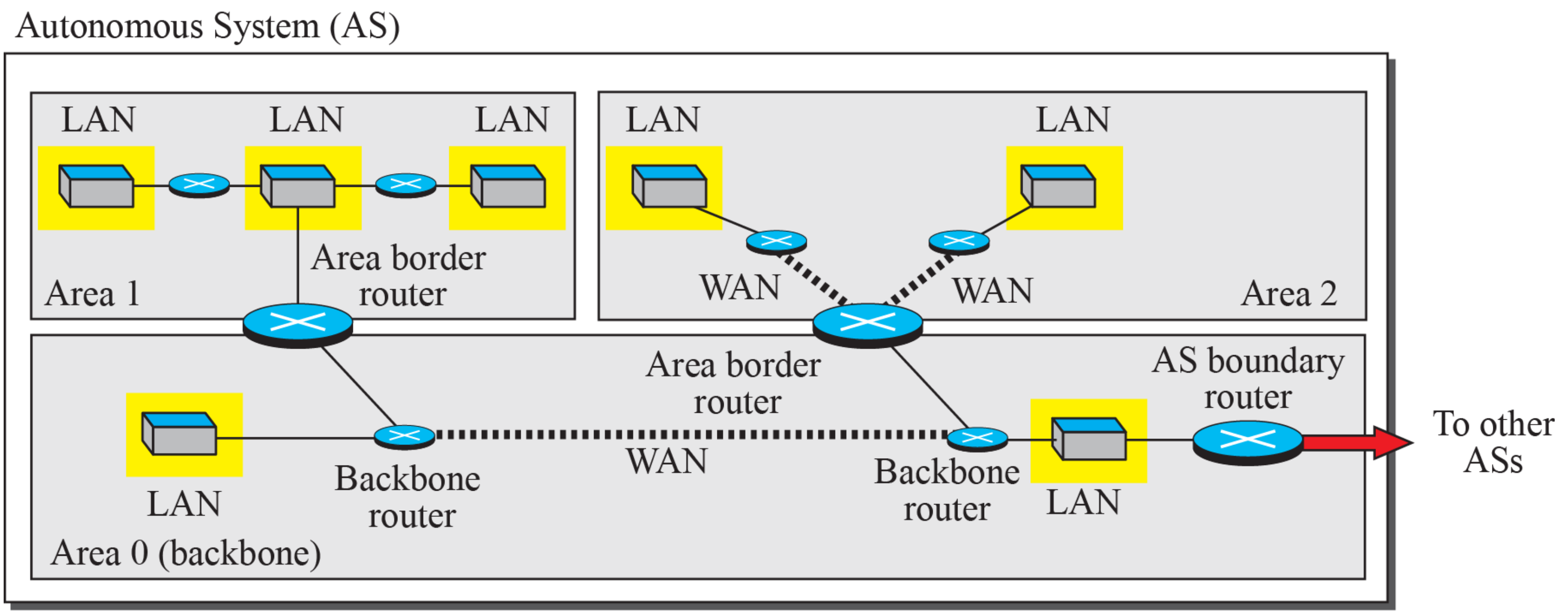
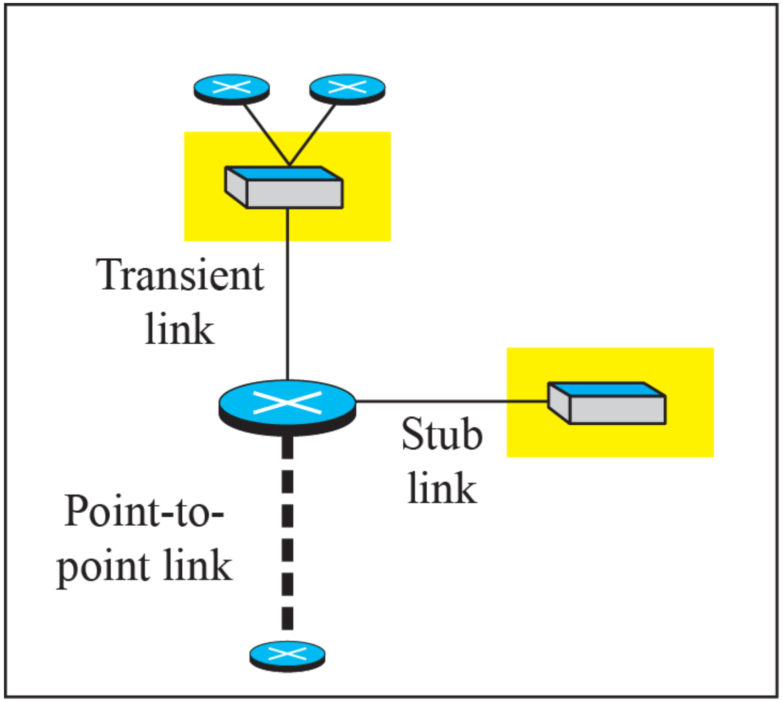
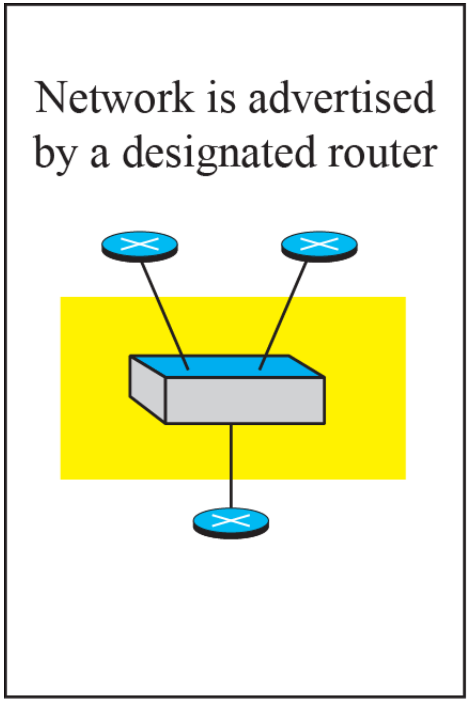


Figure 20.22: Five different LSPs (Part I)

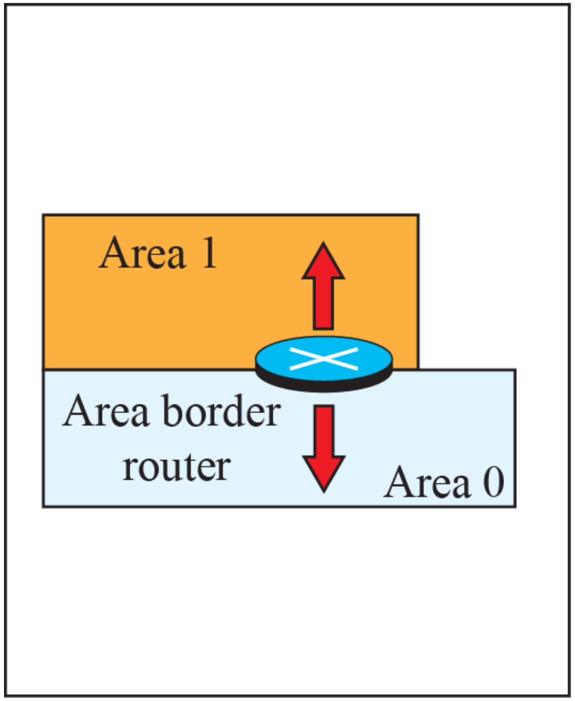


a. Router link

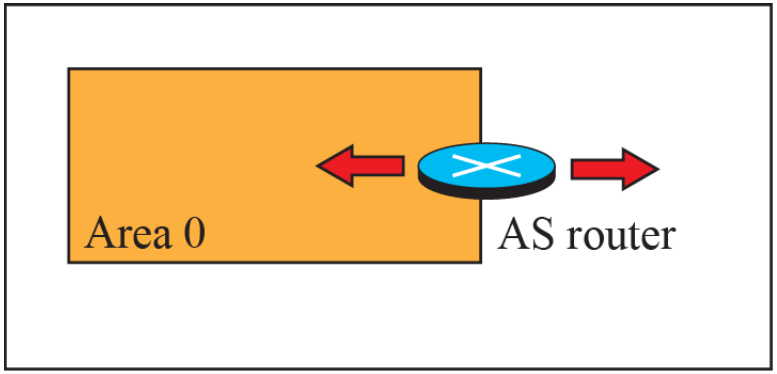


b. Network link

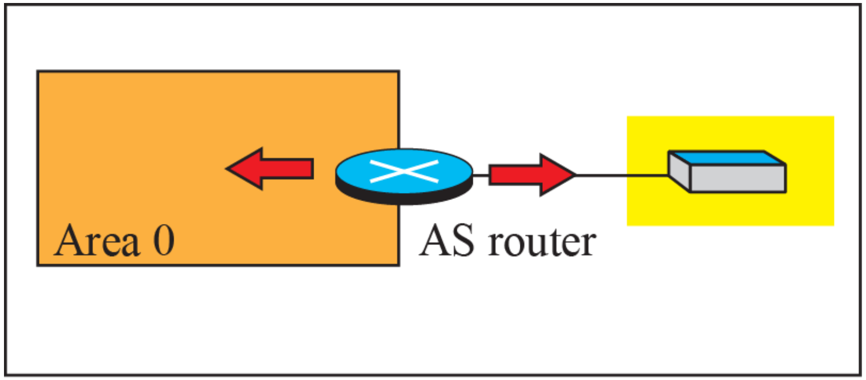
Figure 20.22: Five different LSPs (Part II)



c. Summary link to network

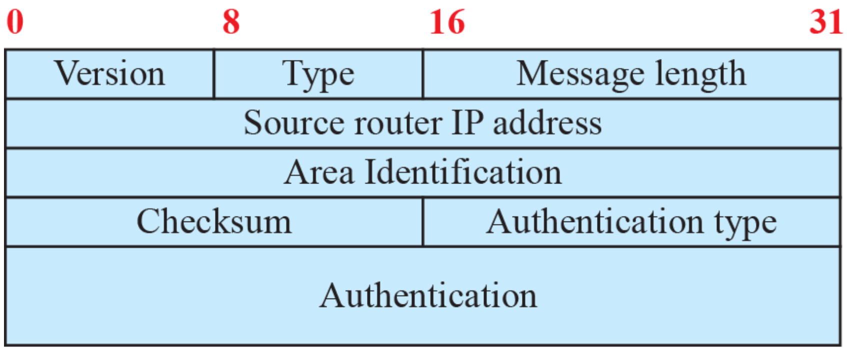


d. Summary link to AS



e. External link

Figure 20.23: OSPF message formats (Part I)

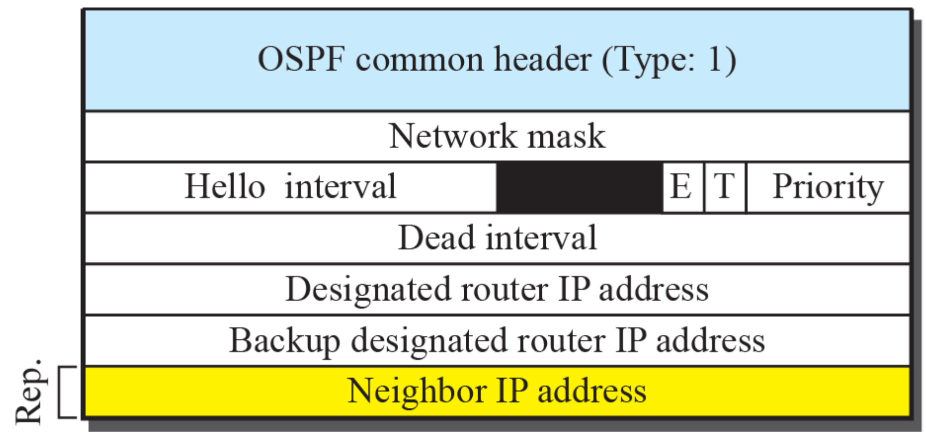


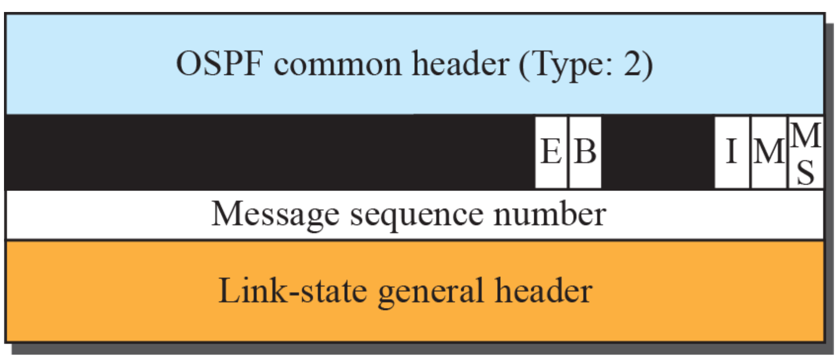
OSPF common header

Legend

E, T, B, I, M, MS: flags used by OSPF
Priority: used to define the designated router
Rep.: Repeated as required

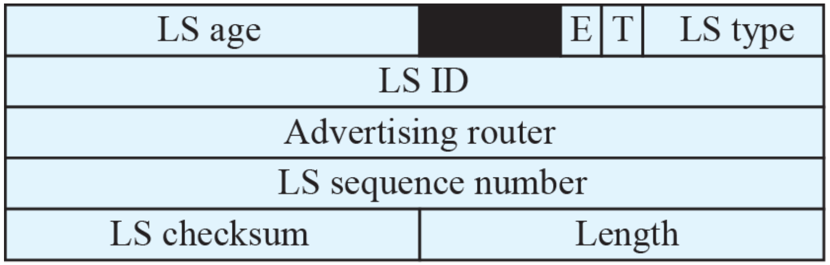
Attention





Database description

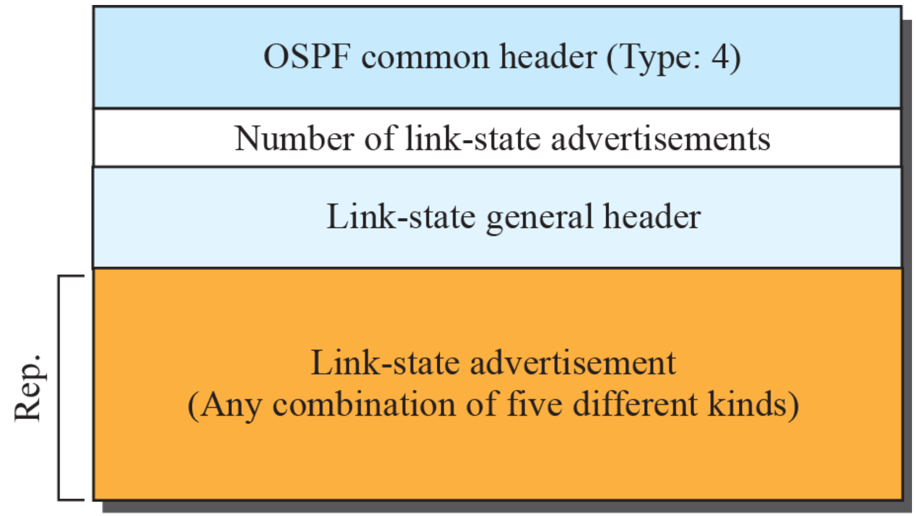
Figure 20.23: *OSPF message formats (Part II)*



Link-state general header

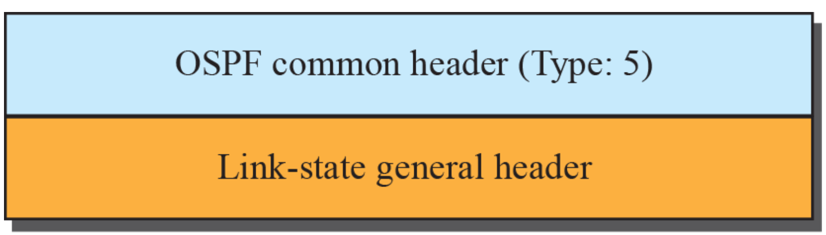
Legend

E, T, B, I, M, MS: flags used by OSPF
Priority: used to define the designated router
Rep.: Repeated as required



Link-state update

Attention



Link-state acknowledgment

20.3.4 Border Gateway Protocol

The Border Gateway Protocol version 4 (BGP4) is the only interdomain routing protocol used in the Internet today. BGP4 is based on the path-vector algorithm we described before, but it is tailored to provide information about the reachability of networks
in the Internet.

Figure 20.24: A sample internet with four ASs

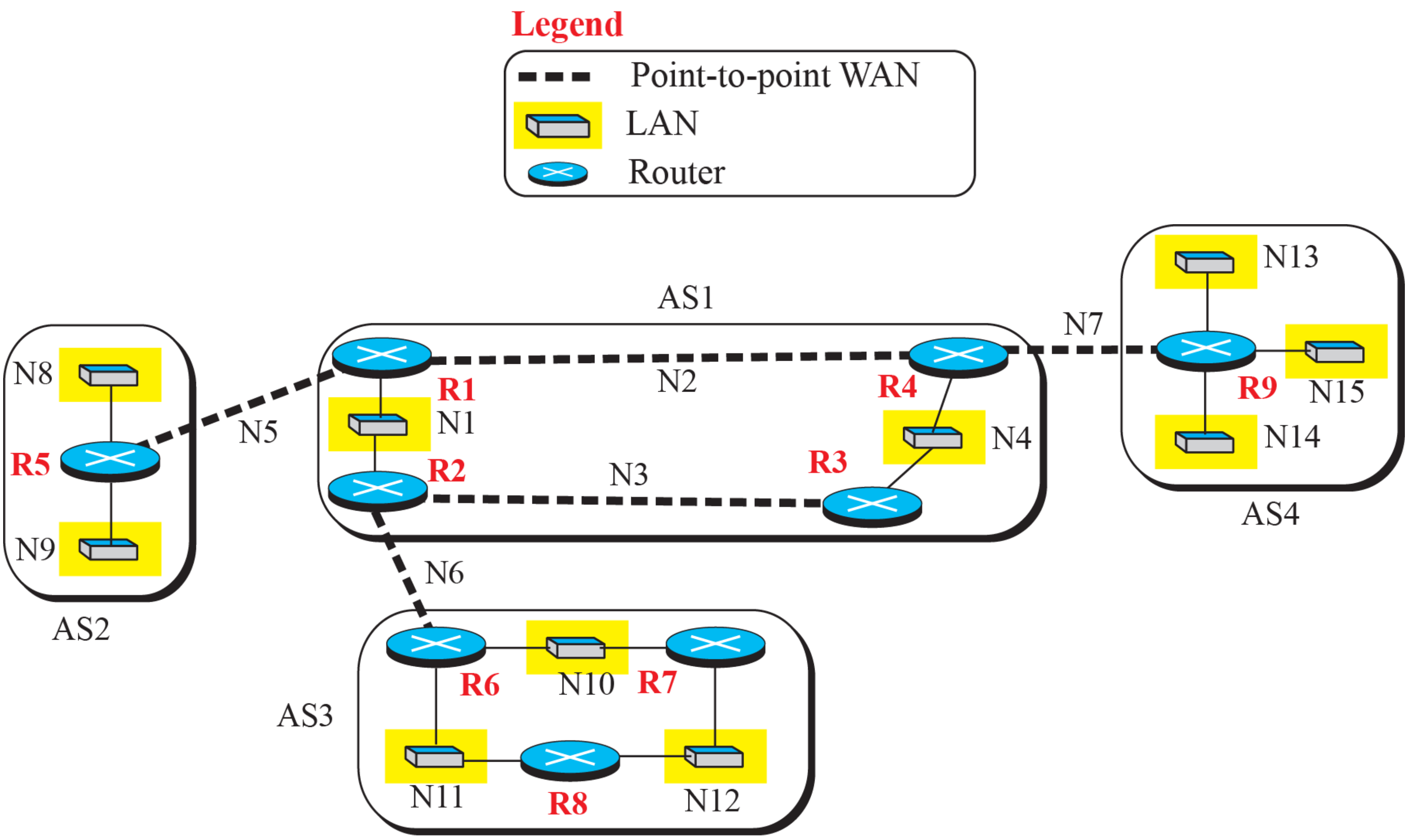


Figure20.25: eBGP operation

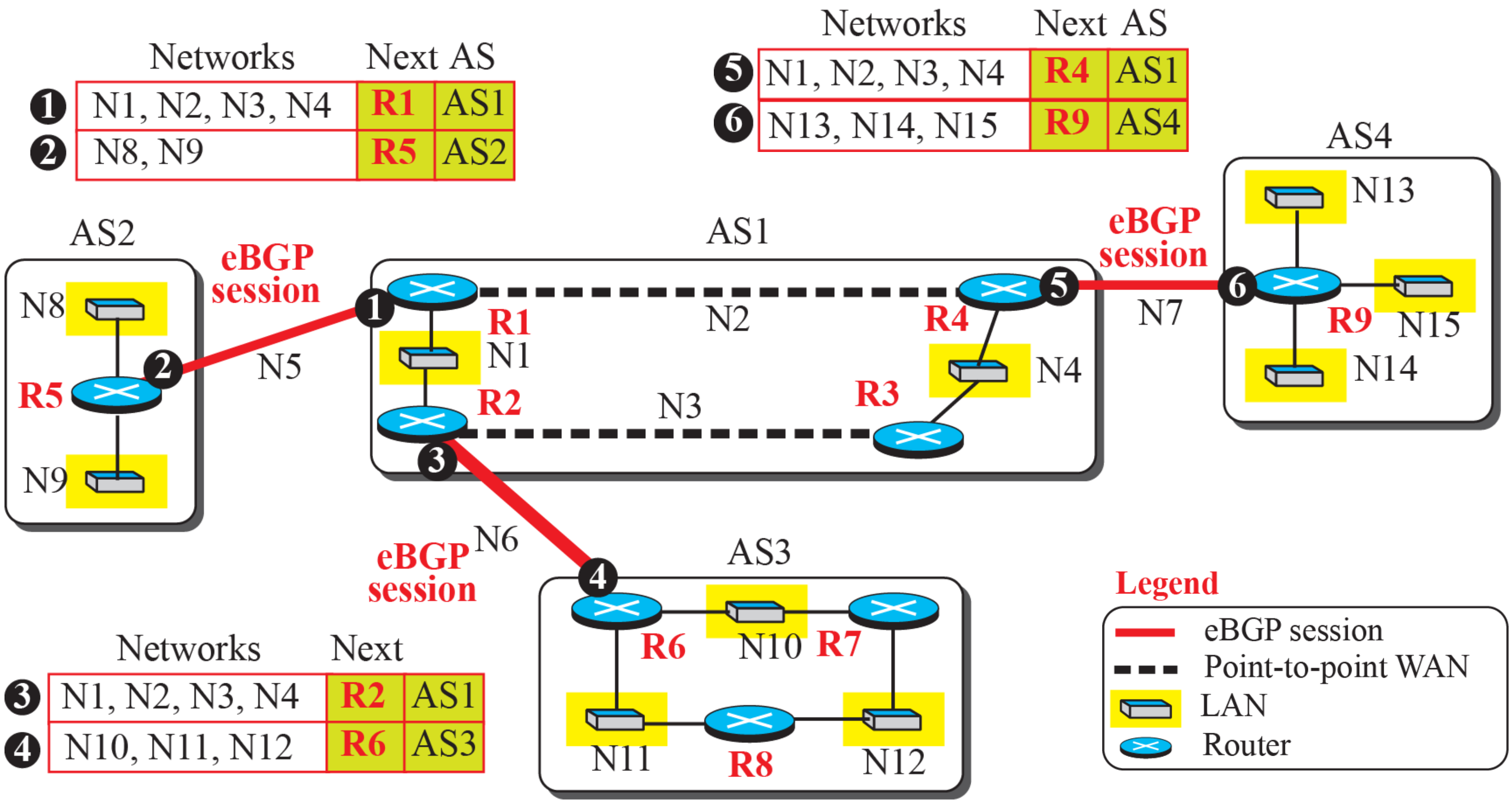


Figure 20.26: Combination of eBGP and iBGP sessions in our internet

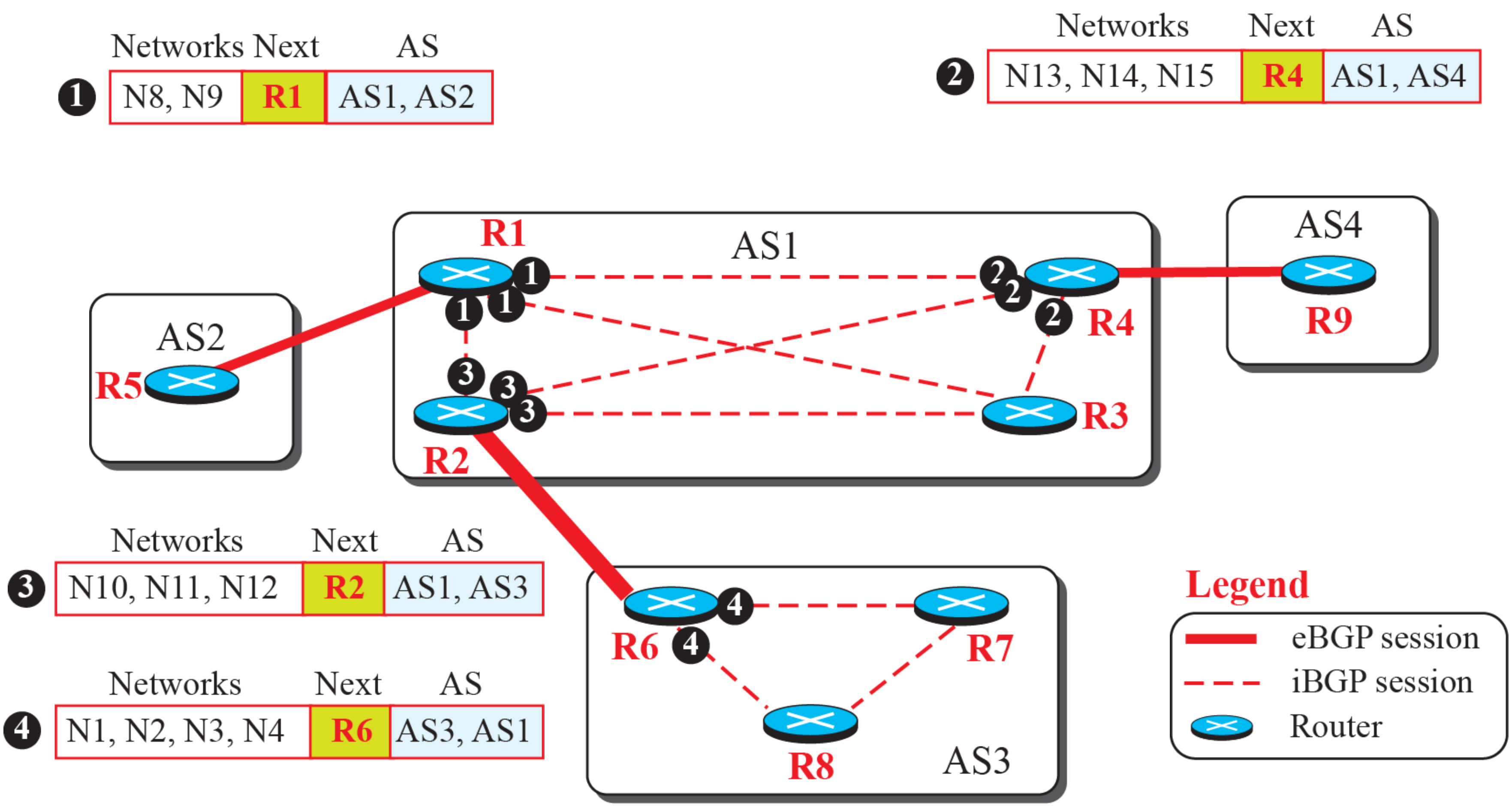
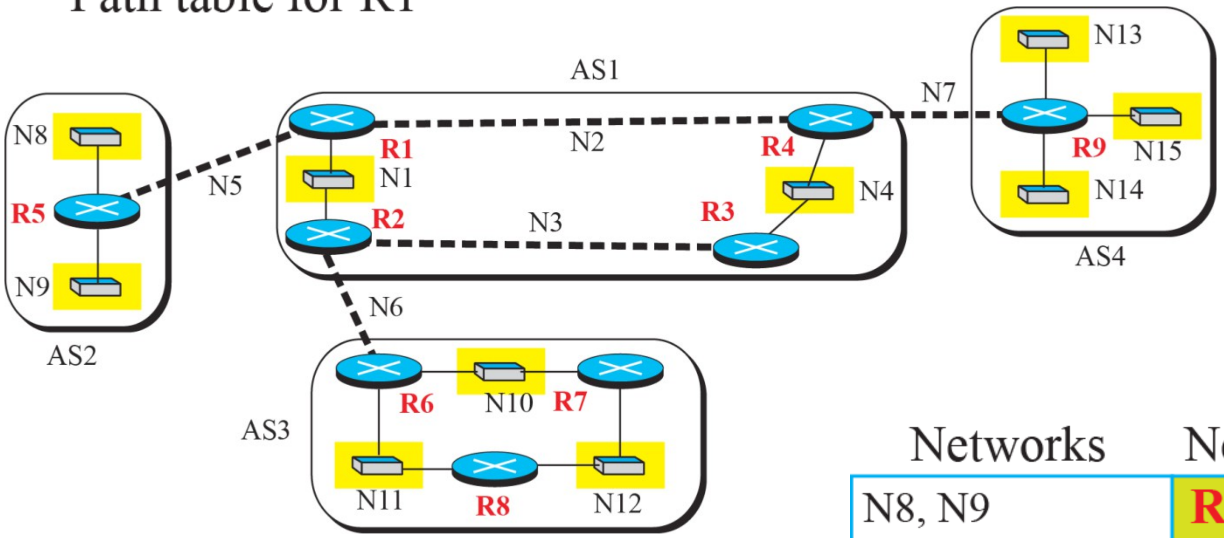


Figure 20.27: Finalized BGP path tables (Part I)

Networks	Next	Path
N8, N9	R5	AS1, AS2
N10, N11, N12	R2	AS1, AS3
N13, N14, N15	R4	AS1, AS4

Path table for R1



Networks	Next	Path
N8, N9	R2	AS1, AS2
N10, N11, N12	R2	AS1, AS3
N13, N14, N15	R4	AS1, AS4

Path table for R3

Networks	Next	Path
N8, N9	R1	AS1, AS2
N10, N11, N12	R6	AS1, AS3
N13, N14, N15	R1	AS1, AS4

Path table for R2

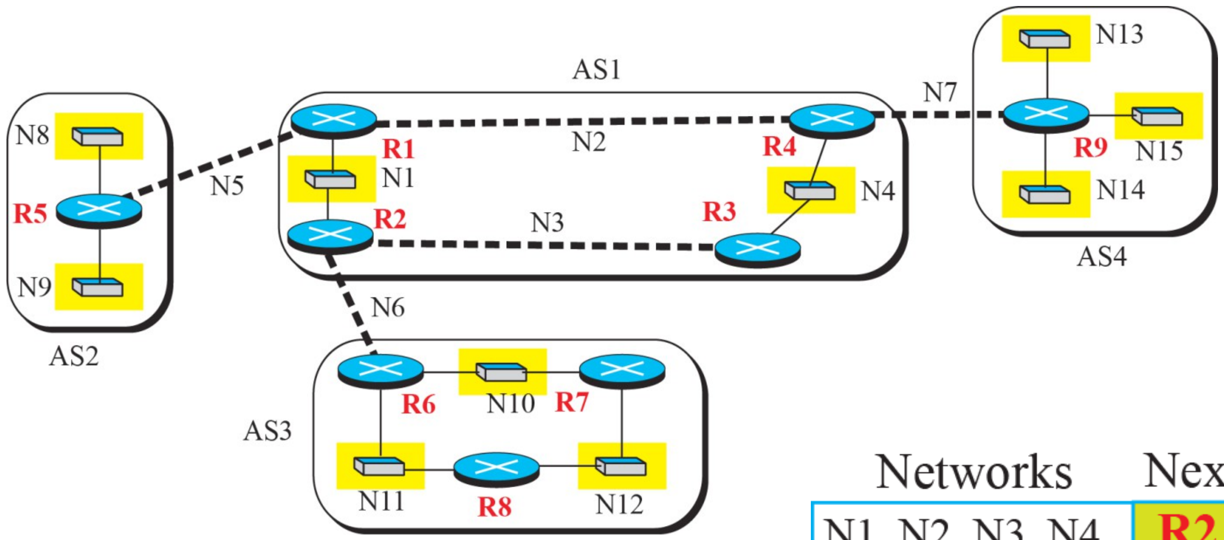
Figure 20.27: Finalized BGP path tables (Part II)

Networks	Next	Path
N1, N2, N3, N4	R1	AS2, AS1
N10, N11, N12	R1	AS2, AS1, AS3
N13, N14, N15	R1	AS2, AS1, AS4

Path table for R5

Networks	Next	Path
N8, N9	R1	AS1, AS2
N10, N11, N12	R1	AS1, AS3
N13, N14, N15	R9	AS1, AS4

Path table for R4



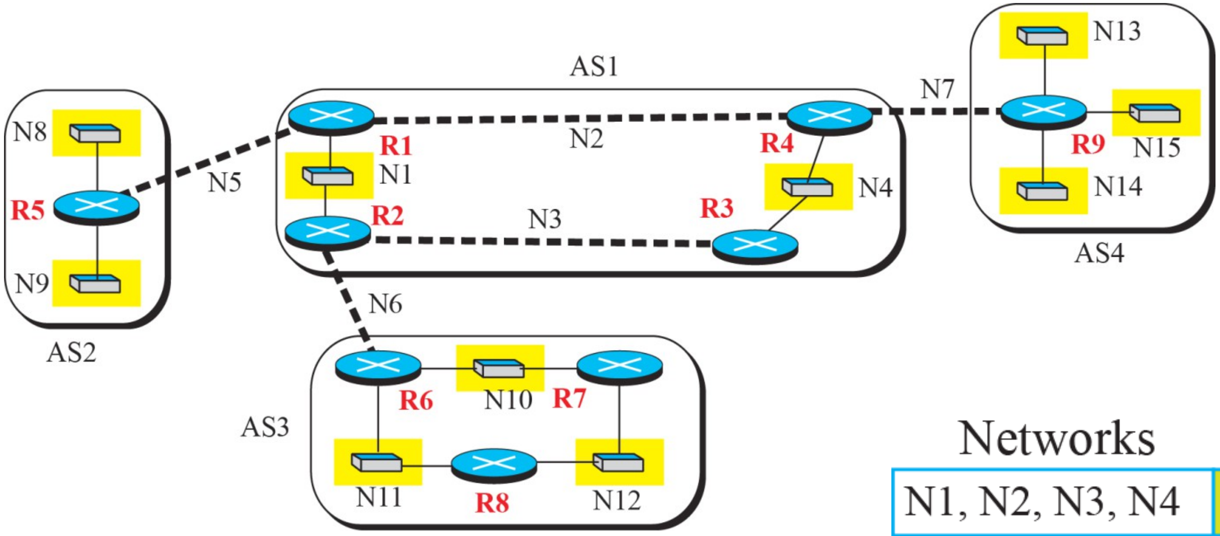
Networks	Next	Path
N1, N2, N3, N4	R2	AS3, AS1
N8, N9	R2	AS3, AS1, AS2
N13, N14, N15	R2	AS3, AS1, AS4

Path table for R6

Figure 20.27: Finalized BGP path tables (Part III)

Networks	Next	Path
N1, N2, N3, N4	R4	AS4, AS1
N8, N9	R4	AS4, AS1, AS2
N10, N11, N12	R4	AS4, AS1, AS3

Path table for R9



Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

Path table for R8

Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

Path table for R7

Figure 20.28: Forwarding tables after injection from BGP (Part I)

Des.	Next	Cost
N1	—	1
N4	R4	2
N8	R5	1
N9	R5	1
N10	R2	2
N11	R2	2
N12	R2	2
N13	R4	2
N14	R4	2
N15	R4	2

Table for R1

Des.	Next	Cost
N1	—	1
N4	R3	2
N8	R1	2
N9	R1	2
N10	R6	1
N11	R6	1
N12	R6	1
N13	R3	3
N14	R3	3
N15	R3	3

Table for R2

Des.	Next	Cost
N1	R2	2
N4	—	1
N8	R2	3
N9	R2	3
N10	R2	2
N11	R2	2
N12	R2	2
N13	R4	2
N14	R4	2
N15	R4	2

Table for R3

Des.	Next	Cost
N1	R1	2
N4	—	1
N8	R1	2
N9	R1	2
N10	R3	3
N11	R3	3
N12	R3	3
N13	R9	1
N14	R9	1
N15	R9	1

Table for R4

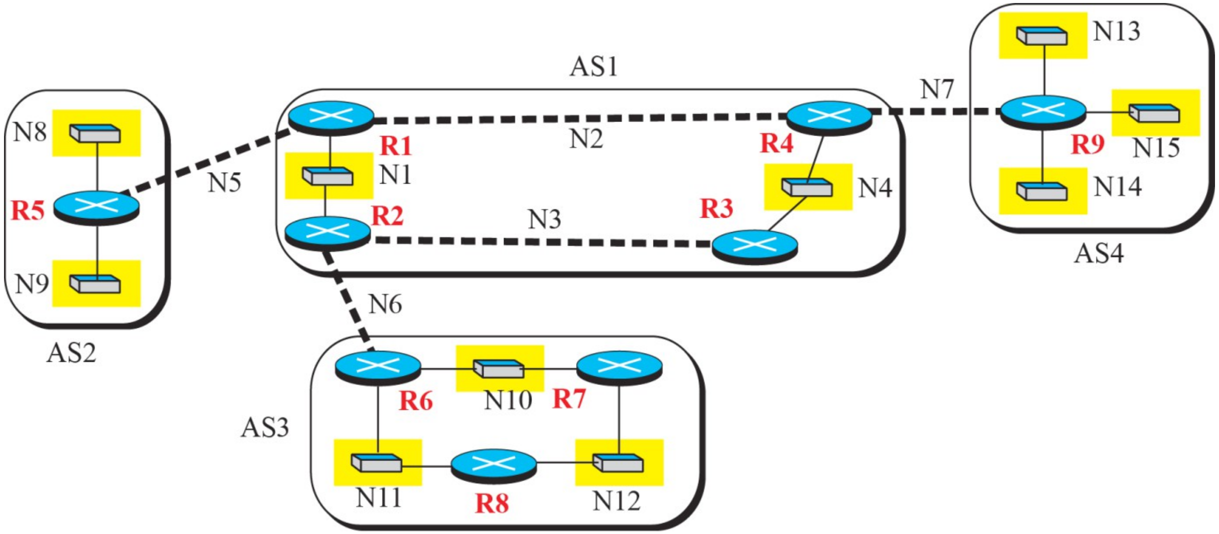


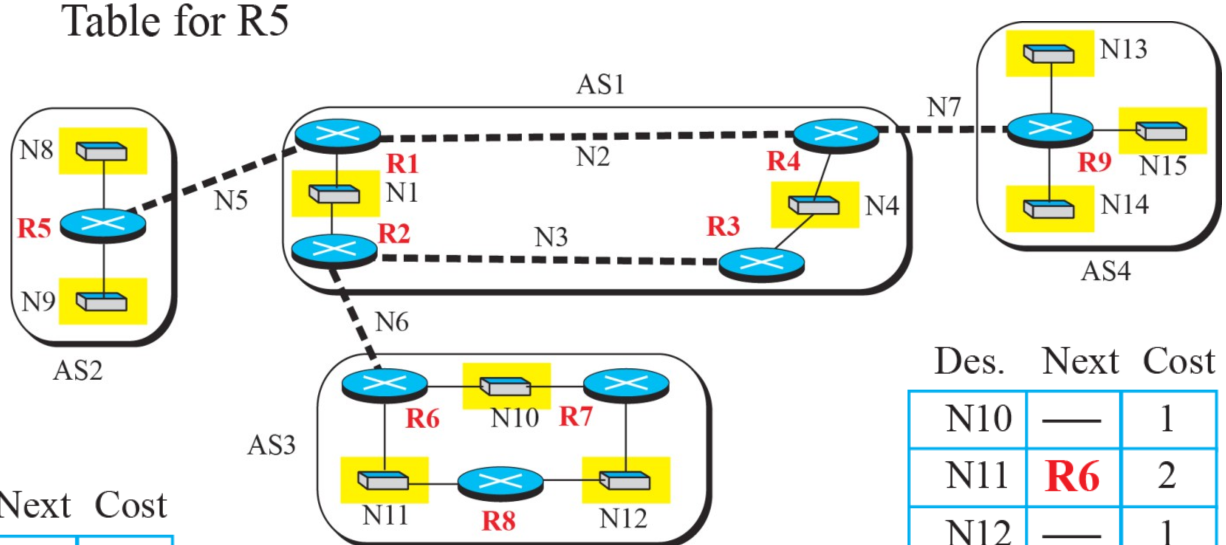
Figure 20.28: Forwarding tables after injection from BGP (Part II)

Des.	Next	Cost
N8	—	1
N9	—	1
0	R1	1

Table for R5

Des.	Next	Cost
N13	—	1
N14	—	1
N15	—	1
0	R4	1

Table for R9



Des.	Next	Cost
N10	—	1
N11	R6	2
N12	—	1
0	R6	2

Table for R7

Des.	Next	Cost
N10	—	1
N11	—	1
N12	R7	2
0	R2	1

Table for R6

Des.	Next	Cost
N10	R6	2
N11	—	1
N12	—	1
0	R6	2

Table for R8

Figure 20.29: Format of path attribute

O: Optional bit (set if attribute is optional)
P: Partial bit (set if an optional attribute in lost in transit)
T: Transitive bit (set if attribute is transitive)
E: Extended bit (set if attribute length is two bytes)

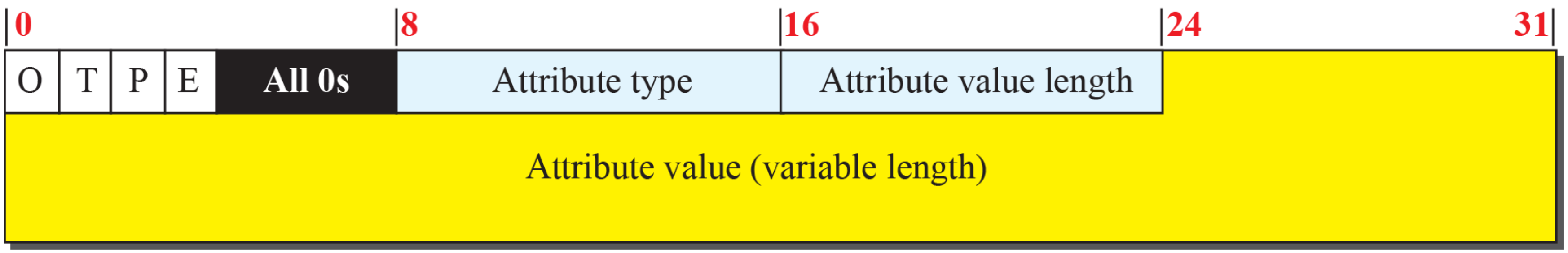


Figure 20.30: Flow diagram for route selection

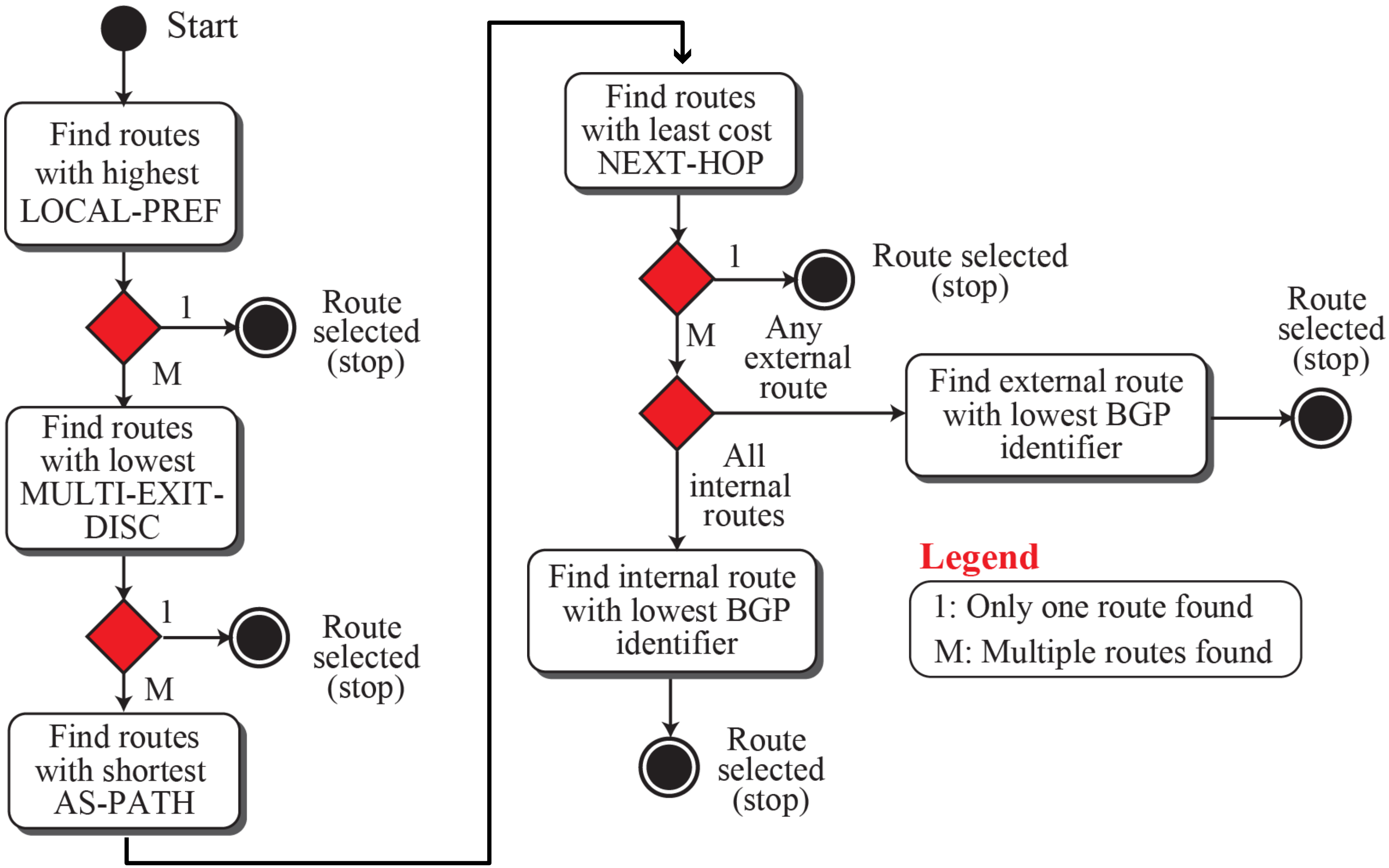
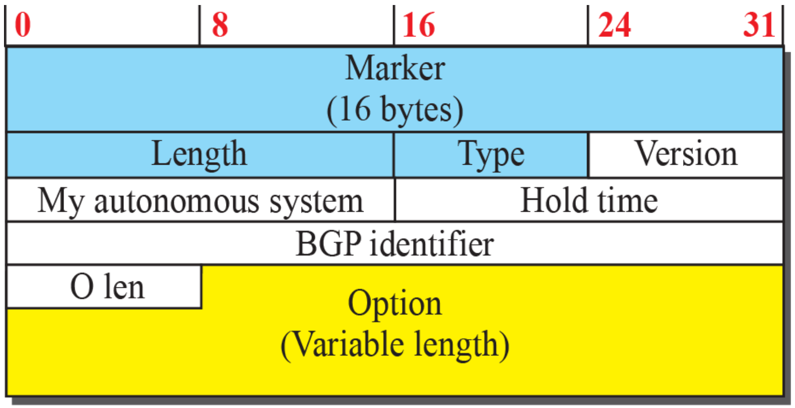
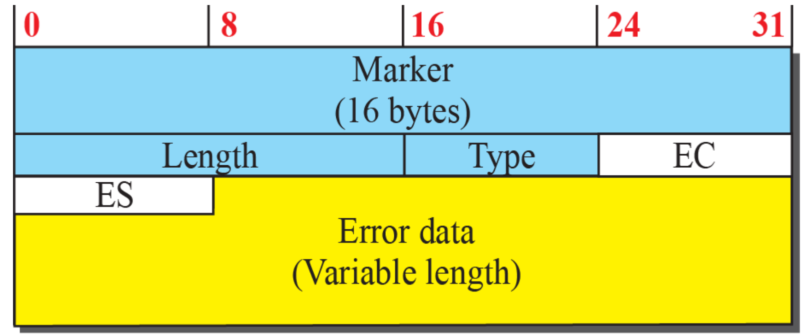


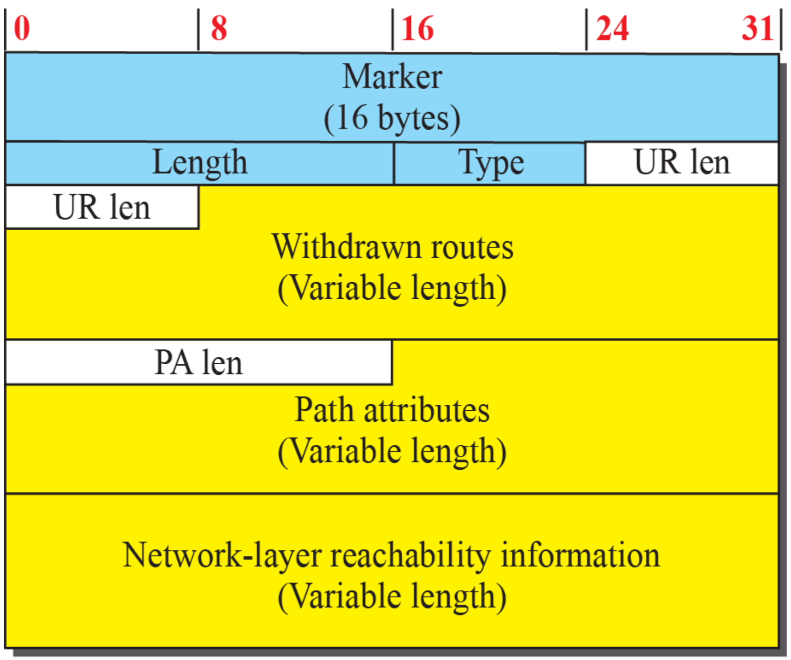
Figure 20.30: BGP messages



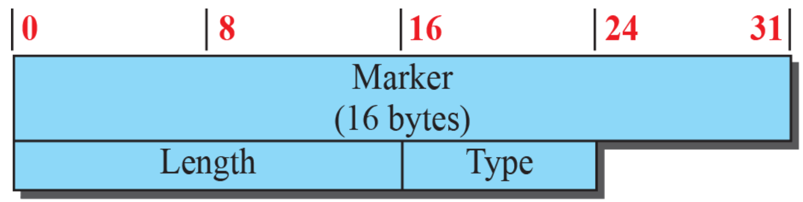
Open message (type 1)



Notification message (type 3)



Update message (type 2)



Keepalive message (type 4)

Fields in common header

Marker: Reserved for authentication
Length: Length of total message in bytes
Type: Type of message (1 to 4)

Abbreviations

O len: Option length
EC: Error code
ES: Error subcode
UR len: Unfeasible route length
PA len: Path attribute length