lecture 9 Network Layer - Routing

Routing at the Network Layer

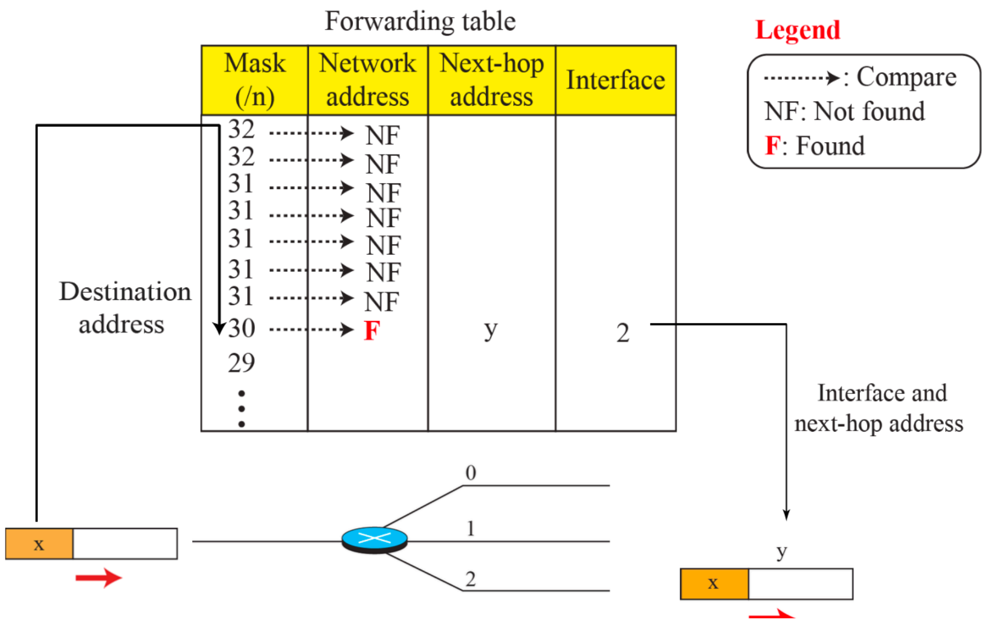
- In the *datagram approach*, a packet is routed, hop by hop, from its source to its destination by the help of forwarding tables
- The routers that glue together the networks in the internet need **forwarding tables**

Two Components of routing

- **control component**
  - *routing protocols*
  - control plane
  - decides where the packets go
- **forwarding component**
  - *moving packets* from input to output *ports*
  - according to forwarding table & packet header "Forwarding plane"

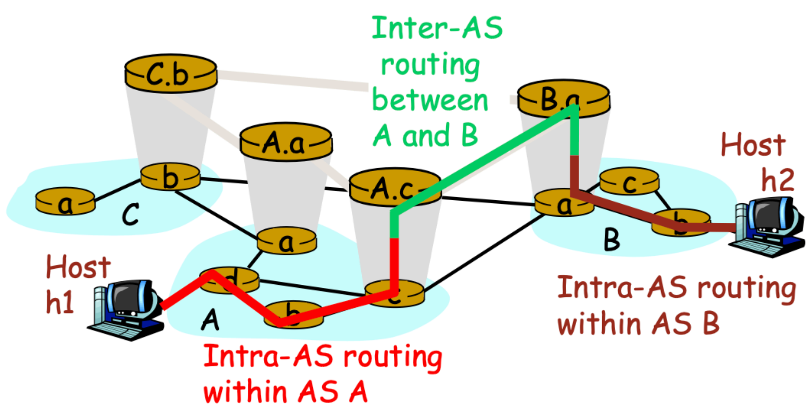
Address Matching

- *Packet forwarding* requires
  - Address matching: lookup of output interface
  - moving the packet **through the route**
    - involves scheduling, queuing, design of switch fabric.. switch design
- Addressing matching
  - **longest prefix** match - *best matching*



Routing in the Internet

- The Internet consists of *Autonomous System (AS)*. interconnected with each other.
- As is *a group of networks and routers* under authority of a single administrator ISP
- Two level routing:
  - **Interior routing (Intra-As)**: administrator responsible for choice of routing algorithm within a network
  - **Exterior routing (Inter-AS):** between AS



Internet Routing

- *determine the path or route* that the packets are to follow
  - **routing protocol**
- For datagram, routing decision must be made *for every arriving data packet*
- For virtual circuit (VC) switching, routing decisions *only when a new VC* is being set up.

- The heart of any routing protocol is the **routing algorithm**

Routing

- **Goal**: *determine "good" path*
- typically means *minimum cost path*
- other definitions are possible

Routing Model

- internet as a **weighted graph**
- node, edge, cost

Performance Criteria

- *Hop count*: the number of routers
- *cost* Least-cost routing
  - **Bandwidth**: the data capacity of a link; *the cost of a link with higher data capacity is smaller*
  - **Delay**: the time
  - **Reliability**: the *probability of failure*

Network Information

- Topology of the network
- Traffic load
- Link cost
- should always be kept *up to date*
- Updating timing *depending on routing method*
  - Update continuously OR
  - Update when condition change
- More information - update more frequently - better decision
- consumes more resources

Routing Strategies

- Fixed routing
- Flooding
- Random routing

- Adaptive routing
  - *Distance vector* routing
  - *Link state* routing
  - *Path vector* routing

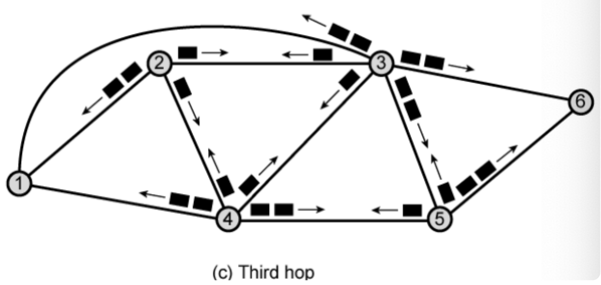
Fixed Routing

- advantages:
  - *Simplicity*
  - work well in a reliable network with a stable load
- disadvantage:
  - *lack of flexibility*
  - Do not react to network congestion of failures

Flooding

- a packet is sent by source node to **every neighbor node**
  - *NO* network information required
  - packet is *copied* to outgoing link except the one it arrives on
  - *a number of copies* arrives
- **in the shortest time**
- engage *endless loops*
- prevent infinite packets
  - packet is *uniquely numbered*; duplicates can be discarded
  - *Node can remember packets* forwarded; keep load in bounds
  - **hop count in packets** limit distance
- Advantage:
  - *robust*
  - At least one route takes the **minimum** route
  - *All nodes are visited*
    - broadcasting information to all node; *exchange routing information*
- Disadvantage
  - *high traffic load*
  - Directly proportional to the network connectivity

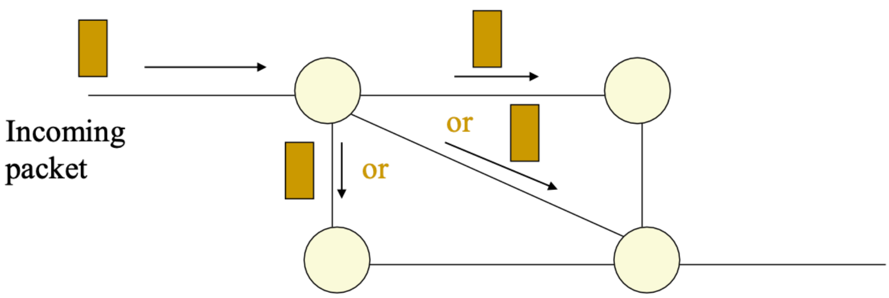
(b) Second hop



(c) Third hop

Random Routing

- outgoing link is **chosen at random**, excluding the link on which the packets arrived
- *Refinement*
    - choose outgoing links in *round-robin fashion*
    - select the link based on some *predefined probability*
- Advantages
    - Robust and simple
    - *no network information*
    - *less load* (compared with flooding)
- Disadvantage
    - longer path
    - Performance not guaranteed



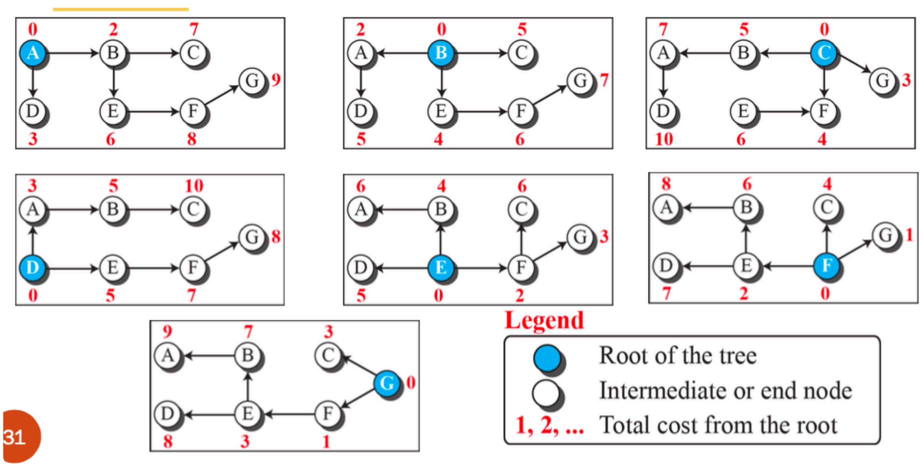
Adaptive (Dynamic) Routing

- Used by *almost all* packet switching networks
- routing decision **changes as network condition change**
    - Failure link
    - Network congestion
- **Requires info** about network
- more complex
- Advantages:
    - Improved performance
    - *Aid congestion control*

- Disadvantages:
  - Substantial processing *burden on nodes*
  - increased traffic due to the *exchange of network info*
  - Reacting too quickly cause *oscillation*

Internet Routing

- **Least-Cost Routing**
- *Least Cost Tree*
  - A way to present the least cost paths
  - only one shortest-path tree for each node
    - *each node* should derive their *forwarding table from its own tree*



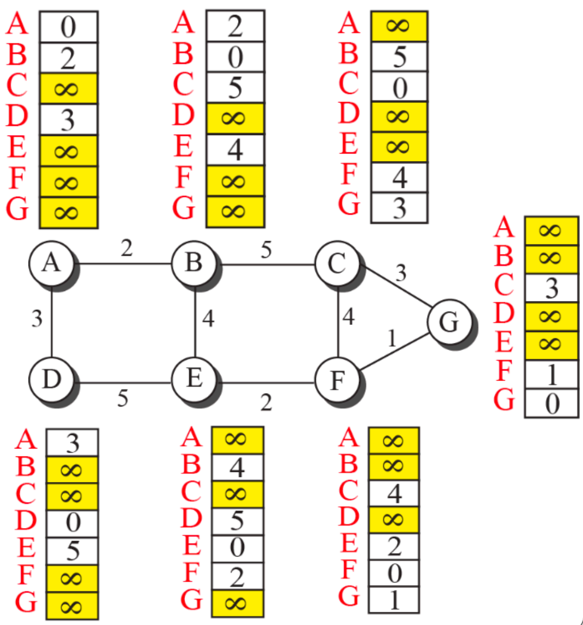
There are several routing algorithms

- Different in the **interpretation of least cost**
- Different in **creating the least-cost tree**

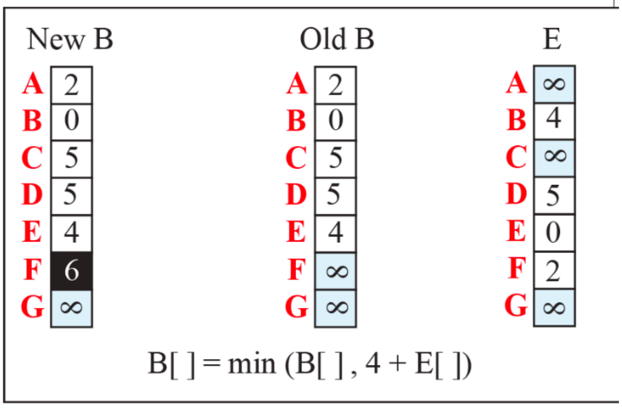
Distance-Vector Routing

- **General idea**
  - Each node creates its own *incomplete* least cost tree based on its *immediate neighbors*
  - *Exchange* the incomplete trees between **immediate neighbors** to build its own least cost tree
- **Distance Vectors**

- When a node is booted, an initial distance vector is created by collecting information from its neighbors.
- Example: Node A can only obtain the cost of the edge A-B and A-D



- Then, B receives a copy of E's vector
  - The sixth element of B's vector is updated.
- B only receives vectors from node A, C and E
- Eventually, each node finds its least cost to other nodes



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

```
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 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
  }
}
```


Limitation

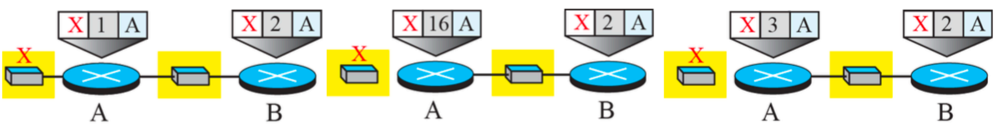
- Decrease in cost propagate quickly
- Increase in cost propagate slowly
  - *Count to infinity*
- Example *RIP* (routing information protocol
  - IN RIP, we *use the new cost* if it is *send from the same node*

Diagram (b): Link A-X is broken

A update its forwarding table (16 means infinity)

Diagram (c): **B** sends its forwarding table to **A** before receiving **A**'s forwarding table.

A update its forwarding table (New Cost = Min { 16, 2+1})



a. Before failure b. After link failure c. After A is updated by B

- Diagram (d): **A** sends its forwarding table to **B**

  B update its forwarding table

  (New Cost = 3 + 1)

  - In RIP, we use the new cost if it is send from the same node

d. After B is updated by A

e. Finally

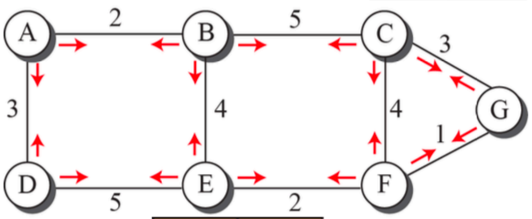
Link-State Routing

- **General Idea**
  - each node collect the *cost of its connecting links*
  - send its table to **all other nodes**
  - Each node have the *same table*
    - Called the **link-state database (LSDB)**
  - Each node *creates its own least-cost tree from the LSDB*

Node	Cost
B	2
D	3

Node	Cost
A	2
C	5
E	4

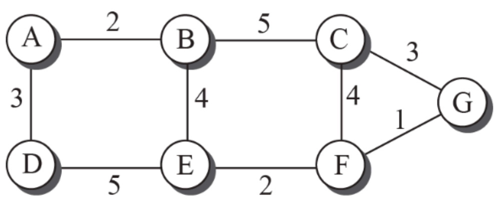
Node	Cost
B	5
F	4
G	3



Node	Cost
C	3
F	1

Node	Cost
A	3
E	5

Node	Cost
B	4
D	5
E	2



a. The weighted graph

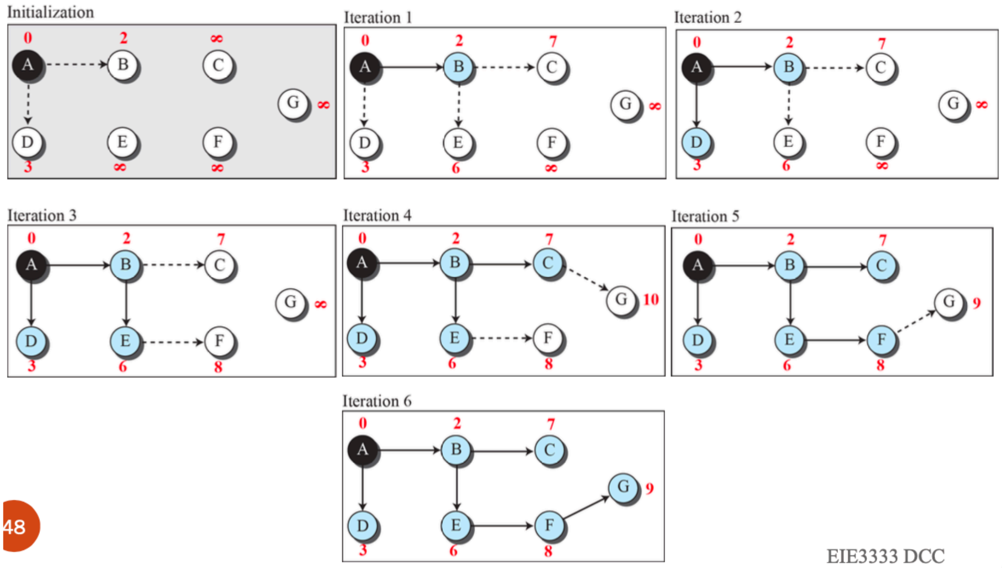
	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

b. Link state database

Dijkstra Algorithm

1. The node choose itself as the root
2. Select *one node, among all nodes not in the tree*, which is *closest to the root*, and adds this to the tree.
3. After this new node is added to the tree, the cost of all other nodes not in the tree needs to be updated
4. Repeat step

◆Example: Node A




```
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)
}
```


Comparison of LS and DV

- In the DV, each router tells its neighbors what it knows about the whole internet
- In LS, each router tells the whole internet what it knows about its neighbors

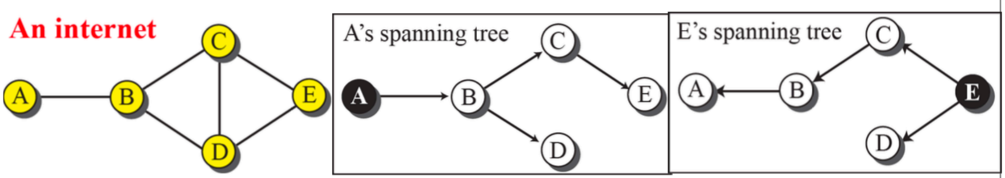
Distance Vector	Link State
View network topology from neighbors perspective	Get common view of entire network topology
Add distance vectors from router to router	Calculate the shortest path to other routers
Frequent, periodic updates: Slow convergence	Event-triggered updates: Faster convergence
Pass copies of routing tables to neighbor routers	Pass link state routing updates to all other routers

Path-Vector Routing

- is **not based on the least-cost goal**
- based on **the policy imposed by the source**

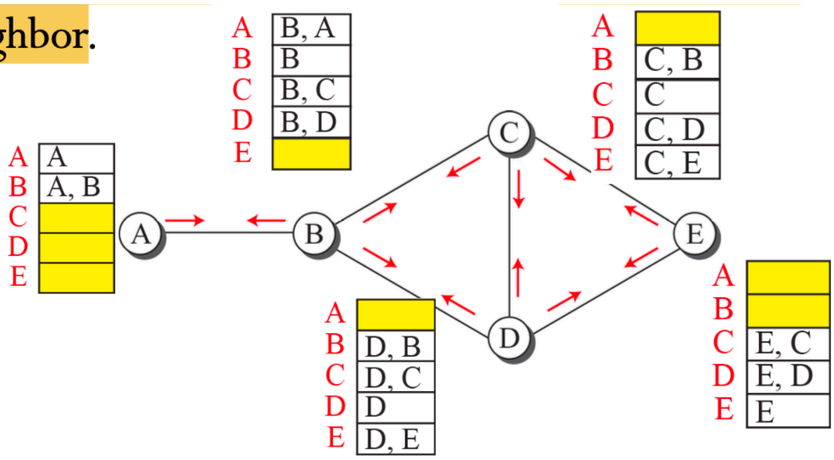
◆Example: Policy of A and E

- ◆Policy 1: Use minimum number of nodes to reach a destination
- ◆Policy 2: Does not pass through D as a middle node



- **Create of Spanning Trees**
  - similar to distance-vector routing
  - create a *path vector* based on the information it can obtain about its *immediate neighbor*

neighbor.



- Each node sends its own vector to *all its immediate neighbors*
- Each node updates its own path vector
- Example: *Policy* - use minimum number of nodes

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

	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 2: C receives a copy of D's vector

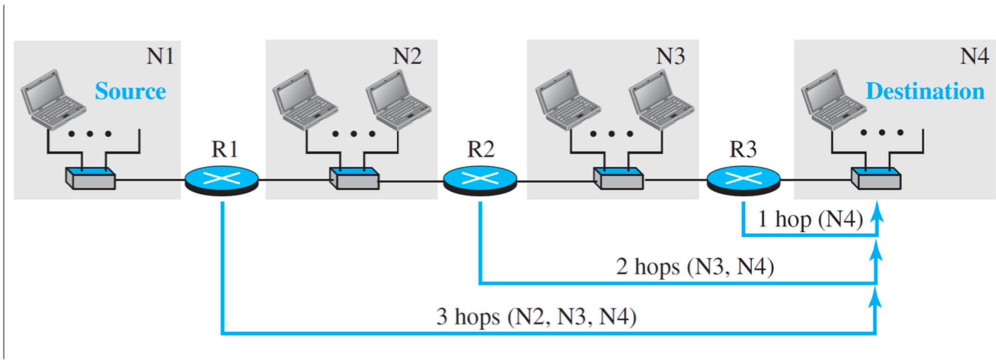
	New C	Old C	D
A	C, B, A	C, B, A	
B	C, B	C, B	D, B
C	C	C	D, C
D	C, D	C, D	D
E	C, E	C, E	D, E

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

```
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
```


Routing Information Protocol (RIP)

- implements *the distance-vector routing a algorithm*
- modified:
  - RIP routers advertise the **cost of reaching different network**, instead of reaching other nodes in a theoretical graph;
  - cost is **defined as the number of hops**, number of networks (subnets)



Forwarding table

- address of the destination network
- address of the next router
- number of hops

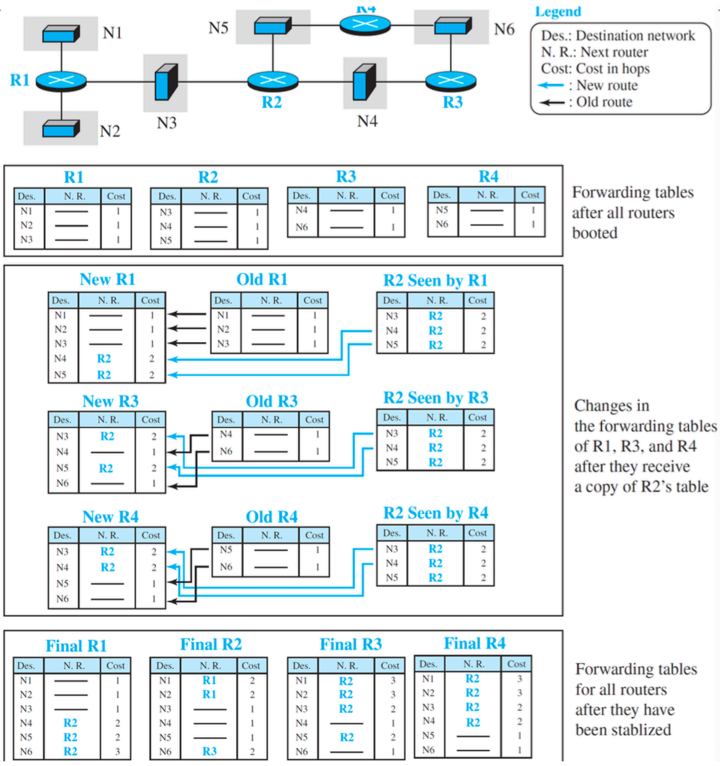
algorithm same as distance-vector and changes

- send *whole contents of its forwarding table*
- add one hop and changes the *next router field* to the *address of the sending router*
- modified forwarding table: **received route**
- old forwarding table: **old route**

The received router selects the old routes as the new ones except:

- if the *received routes does not exist* in the old forwarding table, added
- *cost* of the received route is *lower*
- if the cost of received route is *higher than* the cost of the old one, **but the value of next route is the same in both route**
    - value infinity

Example of an autonomous system using RIP



Forwarding tables after all routers booted

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

Forwarding tables for all routers after they have been stablized