Routing Protocols

- Distance Vector Routing
- Link State Routing

INTRODUCTION

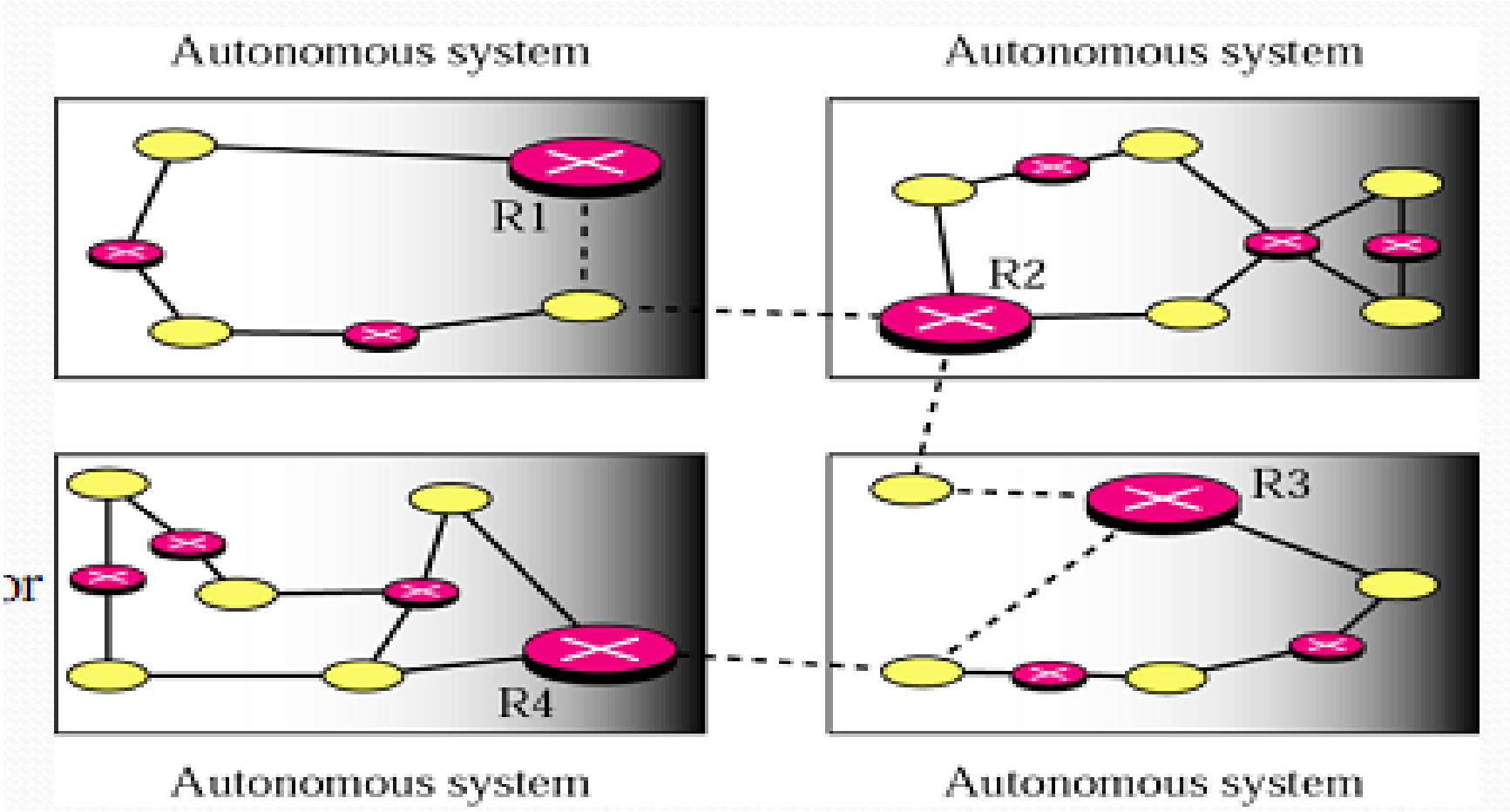
- **Internetwork collection of autonomous systems.**

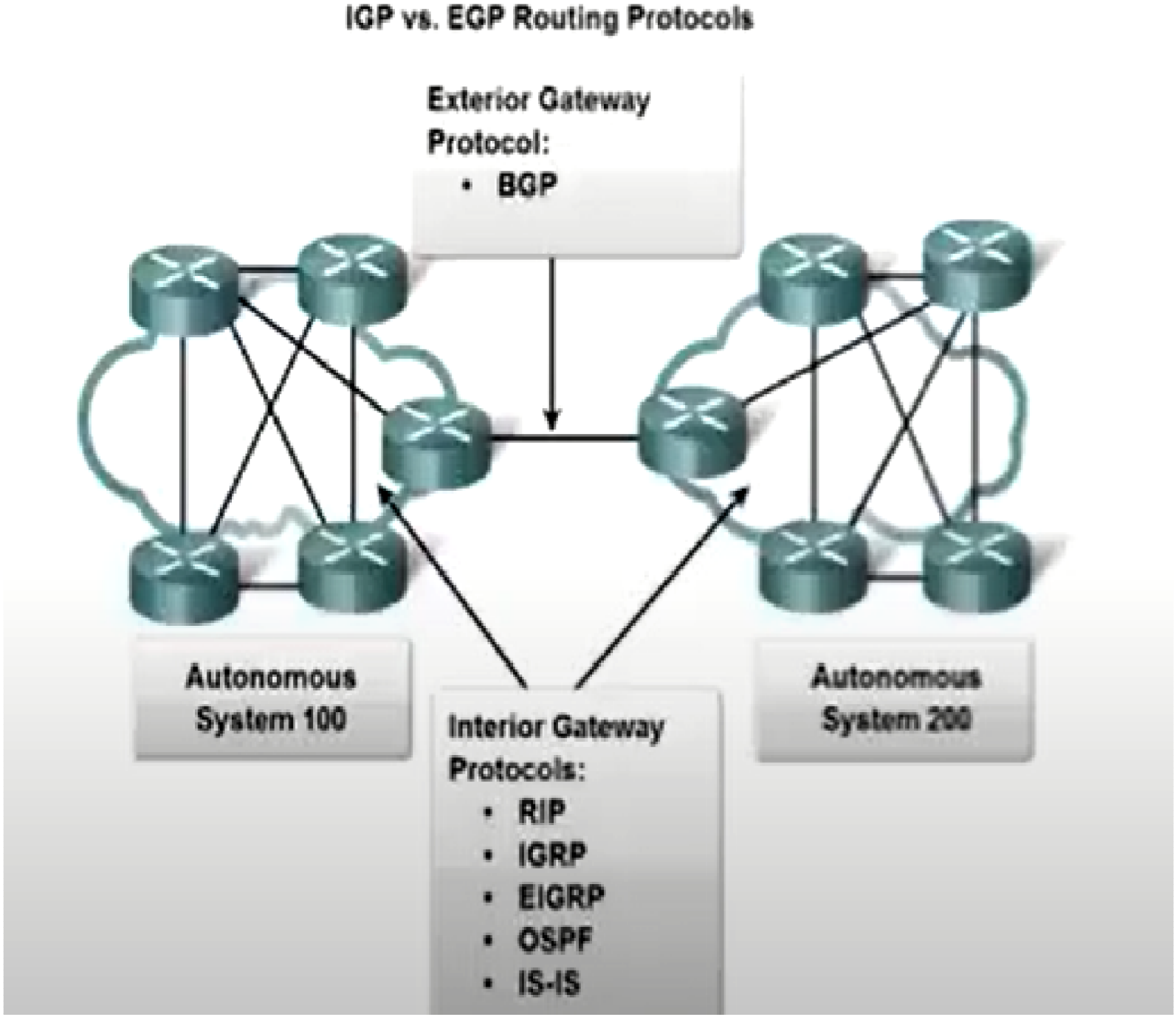
- **An internet is a combination of networks connected by routers.**

- **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.**

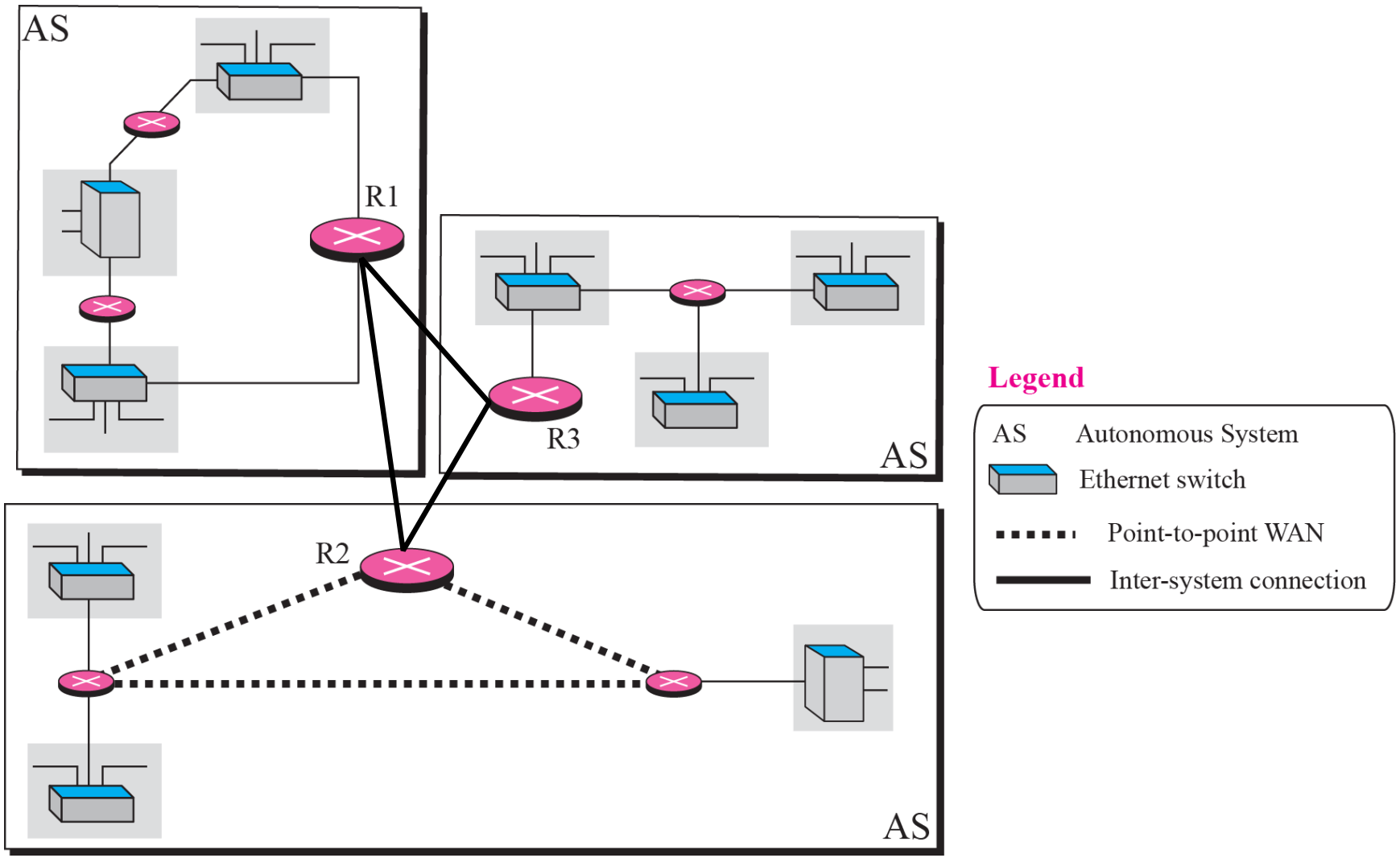
- **Routing process**
- **It is a combination of rules and procedures that lets routers in the internet inform each other of changes, populating the forwarding table**
- **Find optimal route for every destination**
- **Routers exchange message about nets they can reach. share whatever information they know about internet or their neighbourhood**

COLLECTION OF AS




IGP vs. EGP Routing Protocols
Exterior Gateway Protocol:
• BGP
Autonomous System 100
Interior Gateway Protocols:
• RIP
• IGRP
• EIGRP
• OSPF
• IS-IS
Autonomous System 200

Autonomous systems

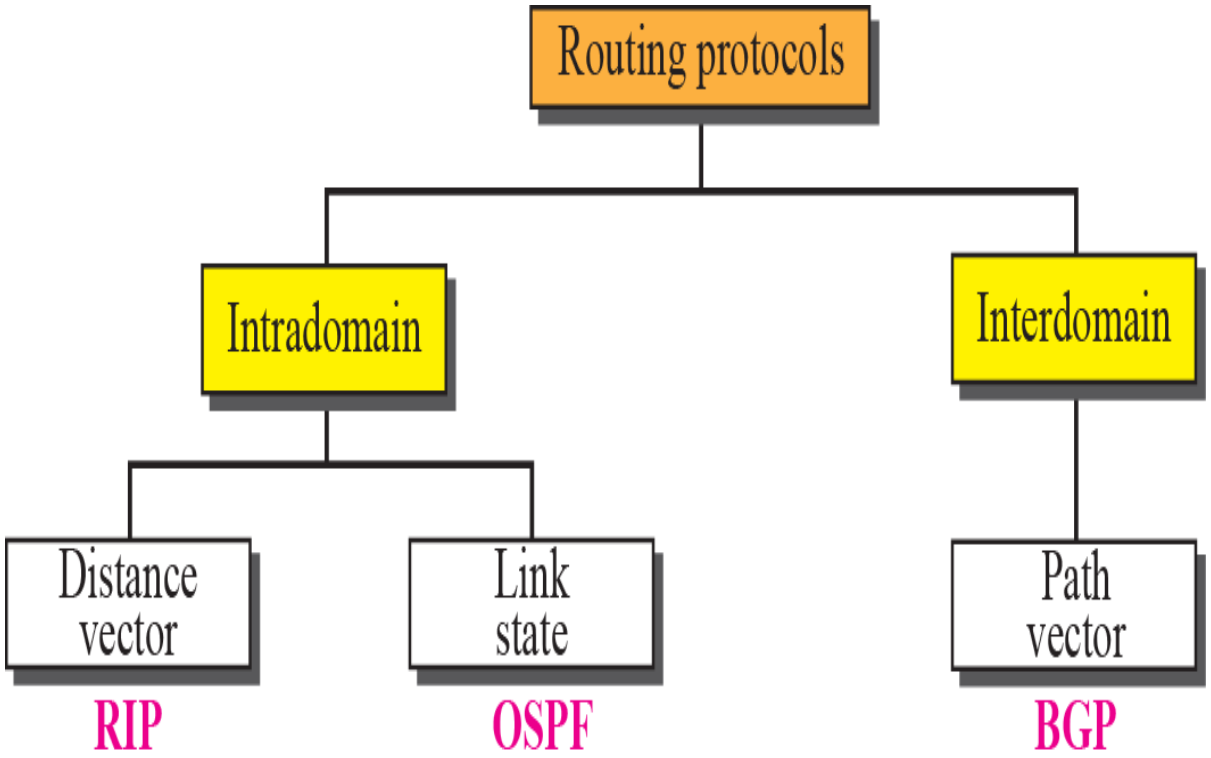


ROUTING PROTOCOLS

- Intradomain-all routers same administration control
- Interdomain-decentralized, scale to internet
- Each AS can choose an interior routing protocol to handle routing inside the AS such as RIP and OSPF
- One exterior routing protocol is usually chosen to handle routing between ASs ; BGP

Popular routing protocols

EIGRP – Hybrid, has both aspects of Distance Vector and Link state routing protocol



Distance vector

- routes are advertised as vectors of distance & direction.
- incomplete view of network topology.
- Generally, periodic updates.

Link state

- complete view of network topology is created.
- updates are not periodic.

RIP – ROUTING INFORMATION PROTOCOL
OSPF – OPEN SHORTEST PATH FIRST
BGP – BORDER GATEWAY PROTOCOL

Cost or Metric

- **When a router receives a packet, to which network should it pass the packet?**
- Decision based on **optimization- latency,,utilization,queue length(not possible, sub optimal –default)**
- Choose optimum pathway
- One approach is to **assign cost** for ***passing through a network***
- This cost is called **metric**
- Maximize throughput?/ Minimize delay?:
  - High Throughput path – associiate low cost
  - Low delay path – low cost

Static Versus Dynamic routing table

- Static table – has manual entries
- Dynamic table – updated automatically when there is change somewhere in the Internet (ex) Link down
- Networks added

Distance Vector Routing

Background

- Also goes by the name Bellman-Ford algorithm
- Used in ARPAnet
- Later in Internet under the routing protocol standard RIP (Routing Information Protocol)
- Now, it is not used much

DISTANCE VECTOR ROUTING

- It is a dynamic routing algorithm in which each router computes a distance between itself and each possible destination i.e. its immediate neighbors.
- The router shares its knowledge about the whole network to its neighbors and accordingly updates the table based on its neighbors.
- The sharing of information with the neighbors takes place at regular intervals.
- It makes use of Bellman-Ford Algorithm for making routing tables.
- Flat network
- distance metric: # hops (max = 15 hops), each link has cost 1
  - DVs exchanged with neighbors every 30 sec in response message (aka advertisement)
  - each advertisement: list of up to 25 different destination IP subnets

Distance Vector

- RIP is based on DVR
- Dynamic, distributed algorithm

Protocol framework:

- **Initial state of node:** distance(cost) to just neighbour is known

- **Final state of node:** distance(cost) to all nodes is known and also “next hop” (to reach destination, whats the next hop)

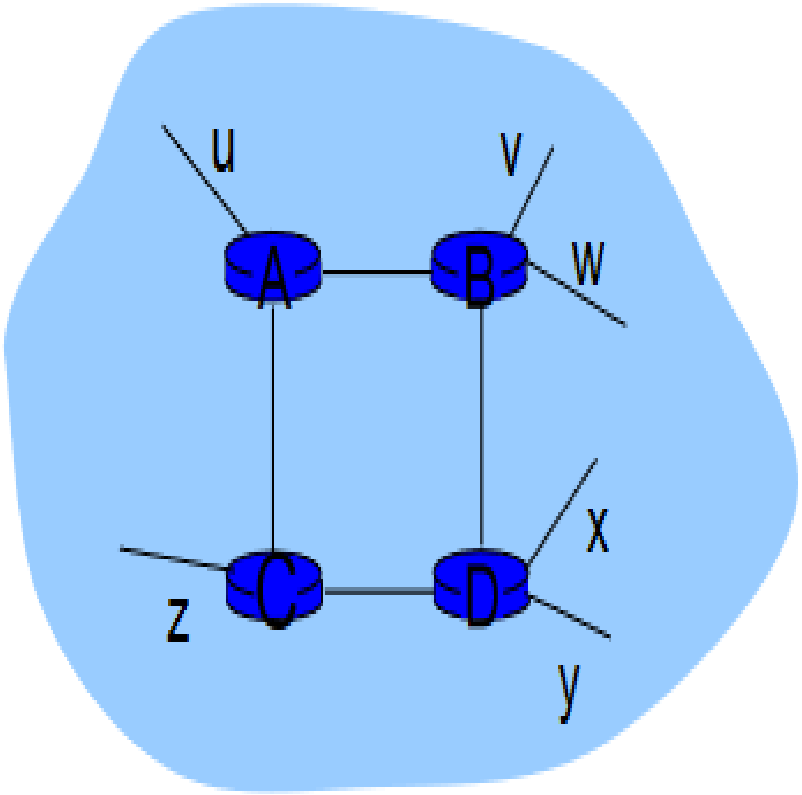
Distance Vector

- As the algorithm iterates, distance(cost) to all nodes is known by each node
- Each node maintains a routing table (3 columns) – distance vector
- It does not know the actual cost initially, it makes an estimate.
- As algorithm proceeds, it converges to actual cost
- Initial cost: cost to neighbours

Distance Vector

- Local knowledge approach
- No node has idea about entire topology
- It knows only about its neighbours thru msg exchange

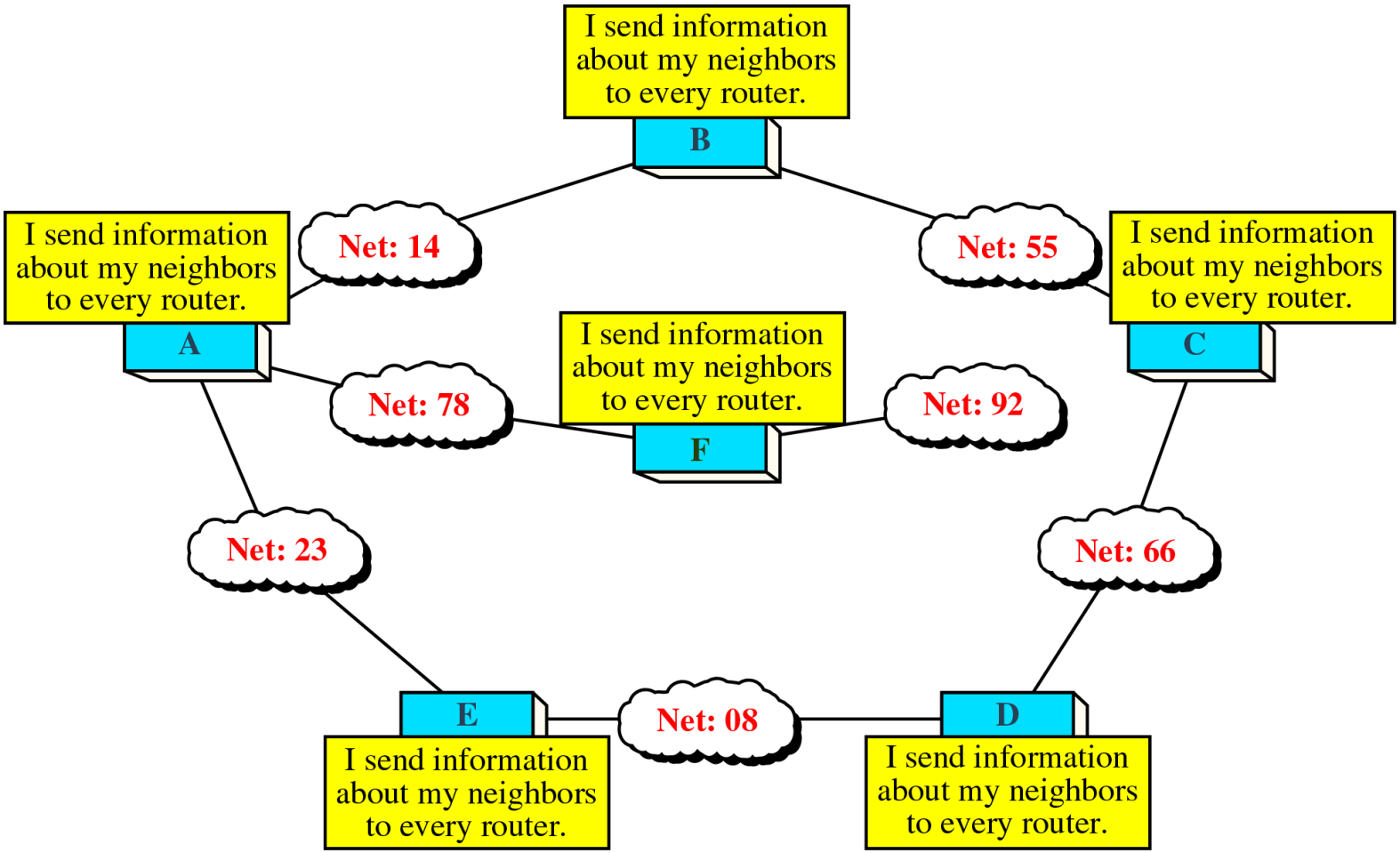
DISTANCE VECTOR ROUTING



from router A to destination *subnets:*

subnet	hops
u	1
v	2
w	2
x	3
y	3
z	2

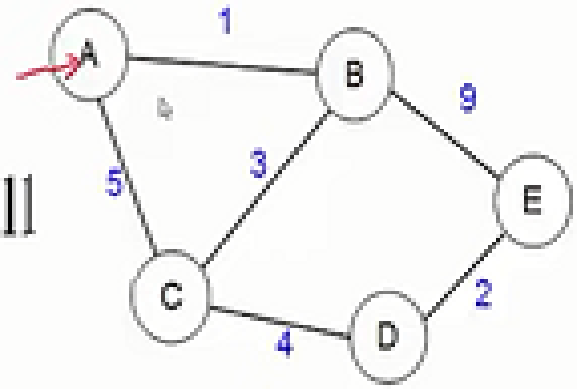
distance vector



Protocol Framework

Initial state at a node: distance (cost) to neighbors is known

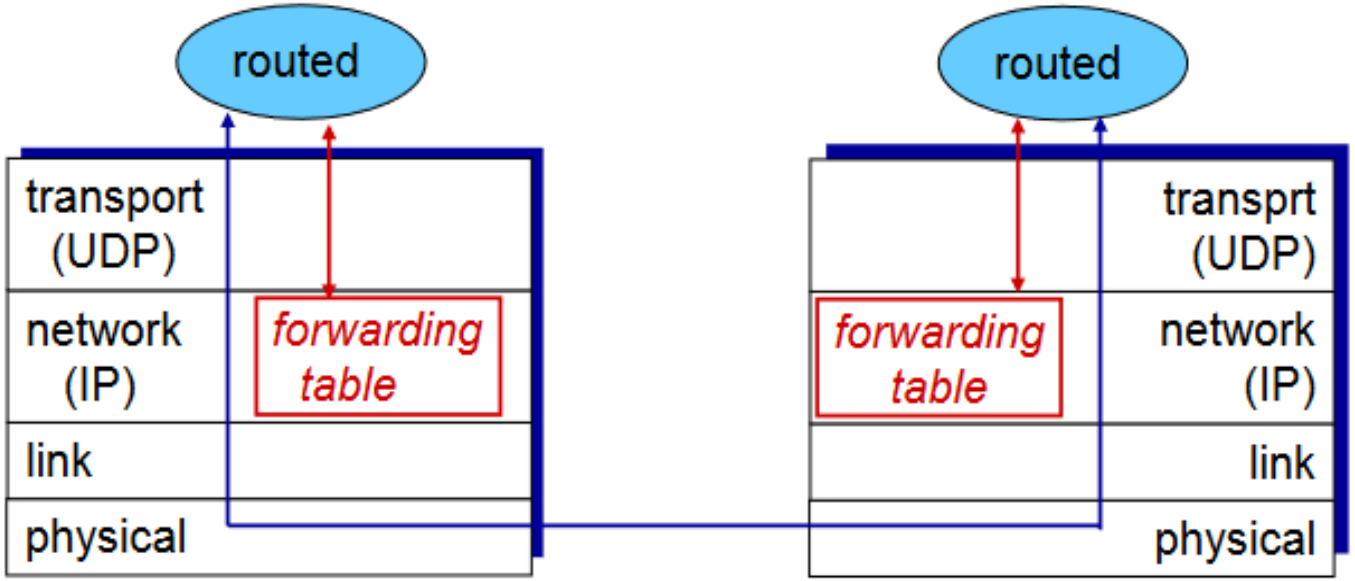
Final state at a node: distance (cost) to all nodes is known, and also the next-hop



Need to handle

- What information to exchange? (message format)
- How to act on a message?
- When to send a message?

- RIP routing tables managed by *application-level* routing daemon process called route-d
- advertisements sent in UDP packets, periodically repeated (30 second interval)

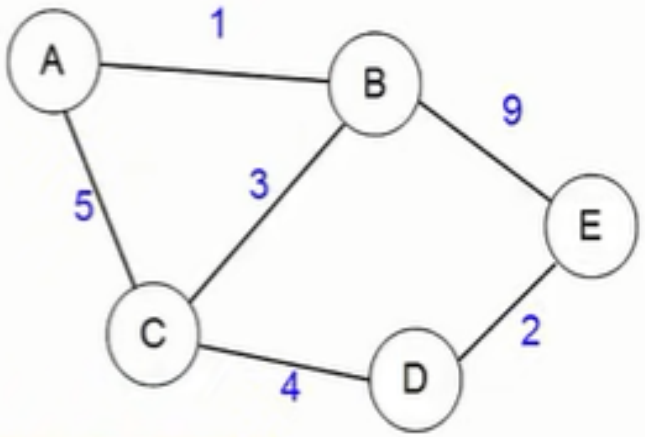


Process of DVR

- ***Initialization***
- ***Sharing***
- ***Updating***
- ***When to Share***
- ***Two-Node Loop Instability***
- ***Three-Node Instability***

State Maintained

- Each node maintains a routing table (distance vector)
  - Destination
  - Estimated cost to destination
  - Next hop via which to reach destination
- Initial state: Cost to neighbors



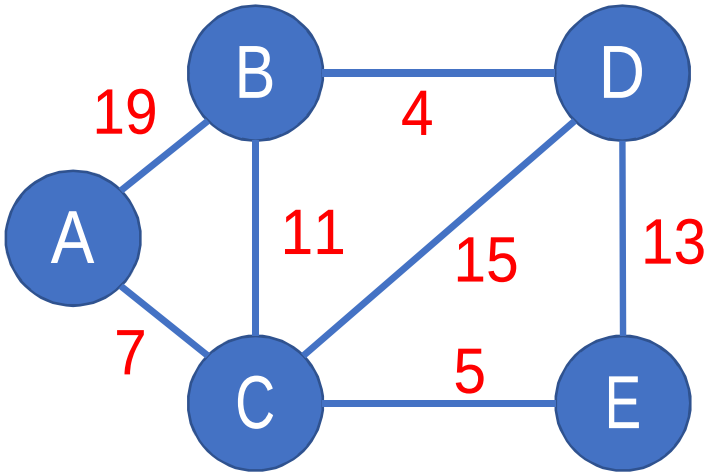
Dest	Cost	Next Hop
A	1	A
C	3	C
E	9	E

Initial Routing table at B

Process of DVR-Initialization

- **Initialization of tables in distance vector routing:**
- **Each node can know only the distance between itself and its immediate neighbors**
- **The distance for any entry that is not a neighbor is marked infinite ∞(unreachable)**

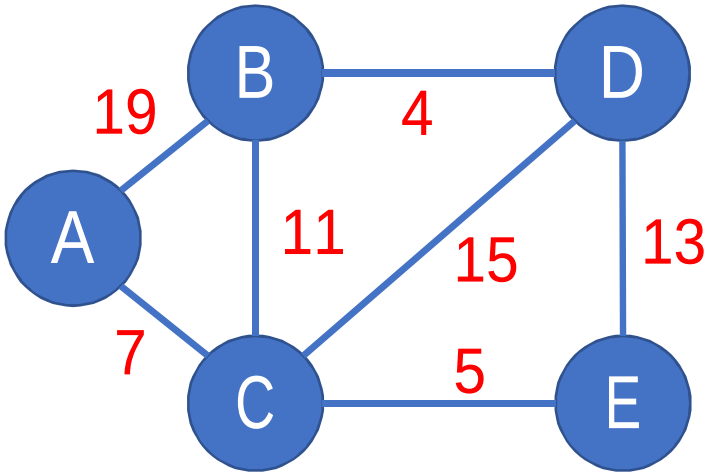
Distance Vector Routing: An example



Distance Vector Routing: An example

Routing table at Node A

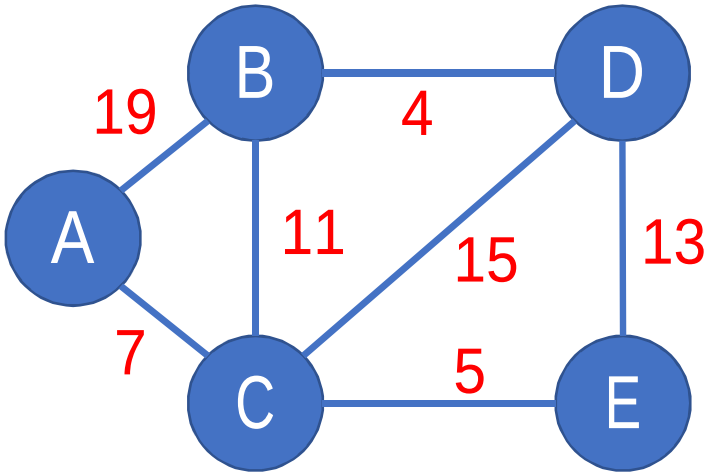
Dest	Cost	N. Hop



Distance Vector Routing: An example

Routing table at Node A

Dest	Cost	N. Hop
A	0	A
B	19	B
C	7	C
D	∞	--
E	∞	--



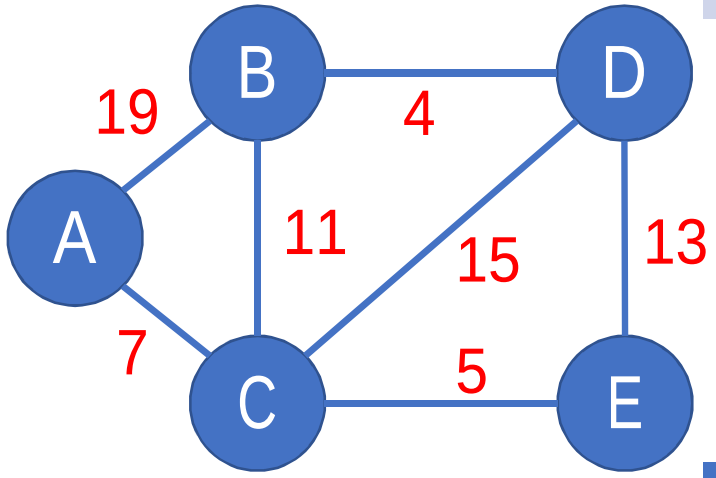
Each node will Know the next hop neighbors

Initialization

Dest	Cost	N. Hop
A	0	A
B	19	B
C	7	C
D	∞	--
E	∞	--

Dest	Cost	N. Hop
A	19	A
B	0	B
C	11	C
D	4	D
E	∞	--

Dest	Cost	N. Hop
A	∞	--
B	4	B
C	15	C
D	0	D
E	13	E

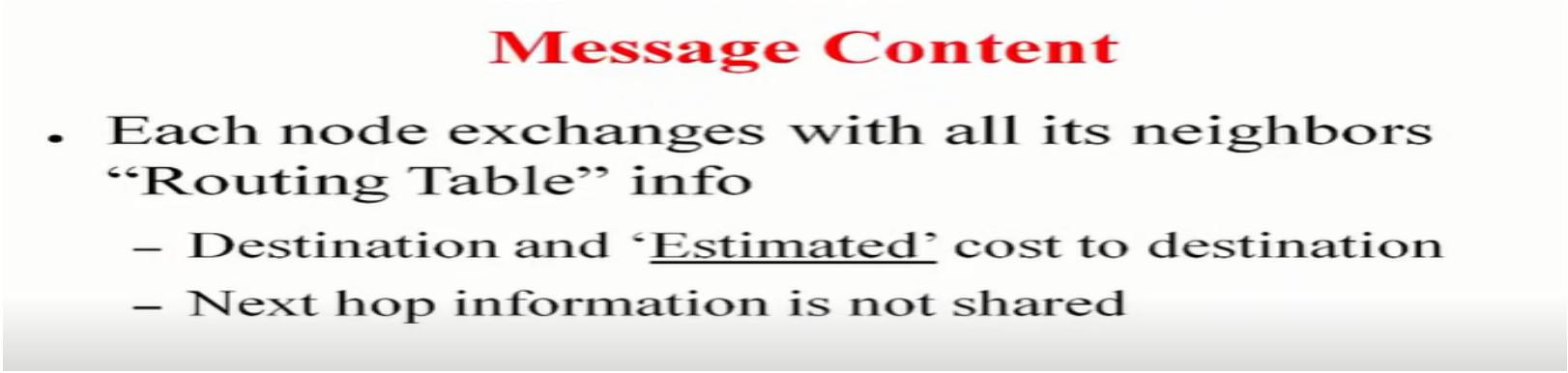


Dest	Cost	N. Hop
A	7	A
B	11	B
C	0	C
D	15	D
E	5	E

Dest	Cost	N. Hop
A	∞	--
B	∞	--
C	5	C
D	13	D
E	0	E

Process of DVR

- **Sharing knowledge about the entire autonomous system: Each router shares its entire routing table with its neighbours.**



- **Sharing happens by two ways:**

  1)periodically update :on the order of several seconds -30s

2)Triggered update: The change can result from the following:

A)A node receives a(new message) table from a neighbor, resulting in changes in its own table after updating.

B)A node detects some failure in the neighboring links which results in a distance change to infinity ∞

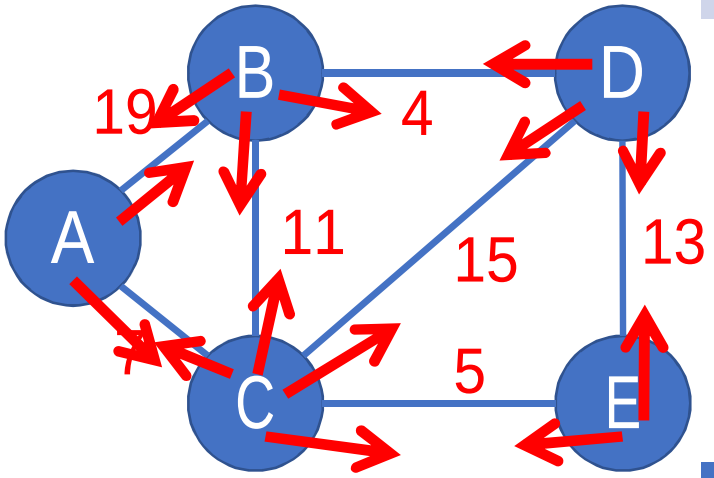
Each update is a list of pairs: Destination, Cost(two column routing table)

Round: 0

Dest	Cost	N. Hop
A	0	A
B	19	B
C	7	C
D	∞	--
E	∞	--

Dest	Cost	N. Hop
A	19	A
B	0	B
C	11	C
D	4	D
E	∞	--

Dest	Cost	N. Hop
A	∞	--
B	4	B
C	15	C
D	0	D
E	13	E

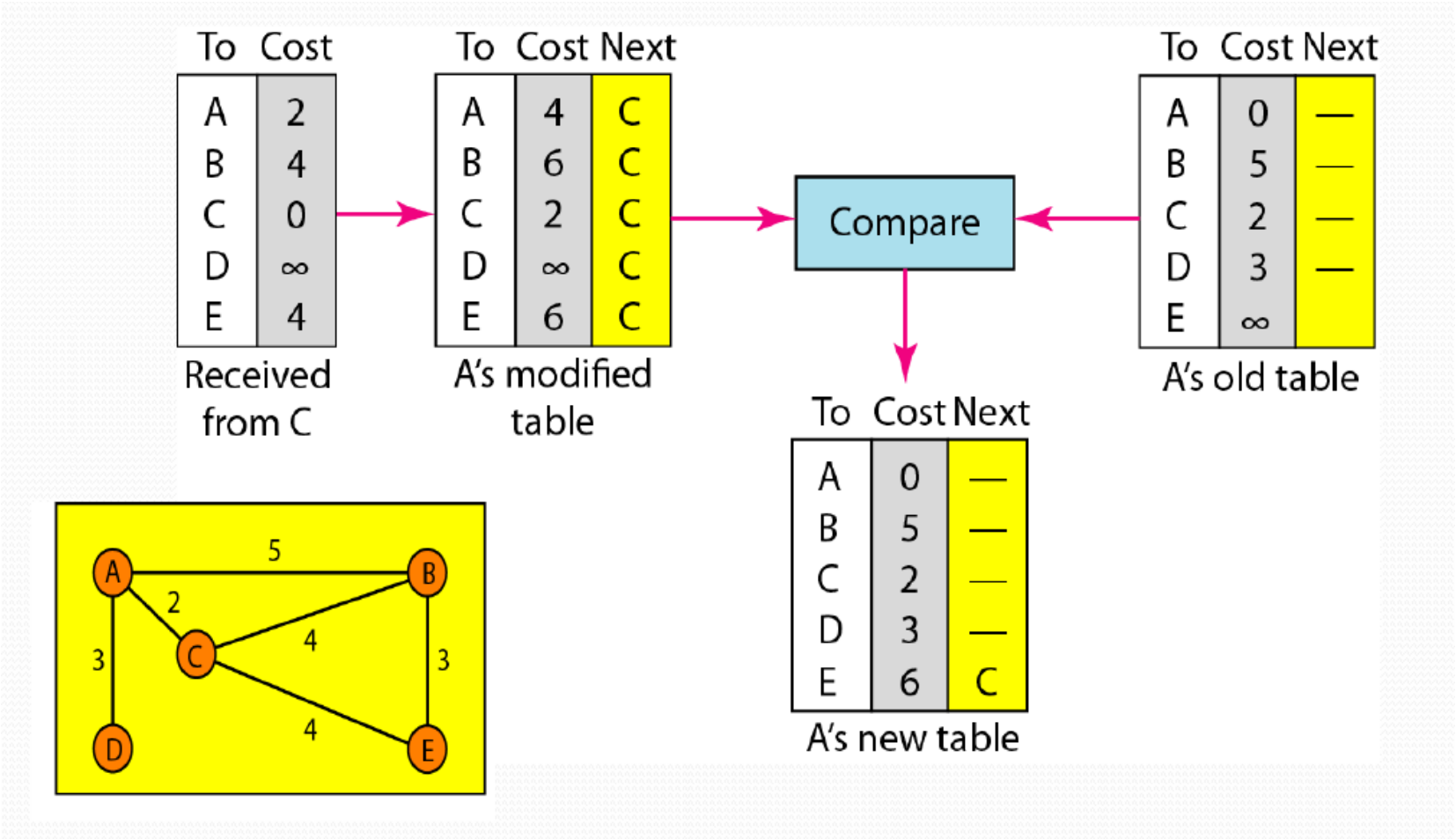


Dest	Cost	N. Hop
A	7	A
B	11	B
C	0	C
D	15	D
E	5	E

Dest	Cost	N. Hop
A	∞	--
B	∞	--
C	5	C
D	13	D
E	0	E

sharing

Process of DVR-sharing & updating

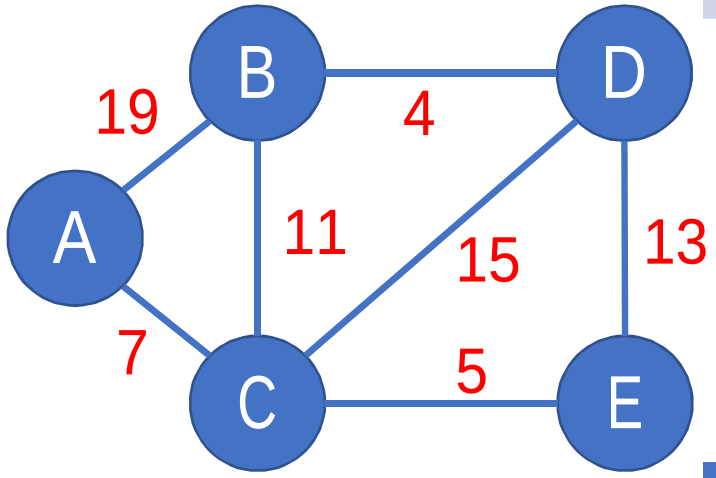


Update your routing table, if required!!

Dest	Cost	N. Hop
A	0	A
B	19	B
C	7	C
D	∞	--
E	∞	--

Dest	Cost	N. Hop
A	19	A
B	0	B
C	11	C
D	4	D
E	∞	--

Dest	Cost	N. Hop
A	∞	--
B	4	B
C	15	C
D	0	D
E	13	E



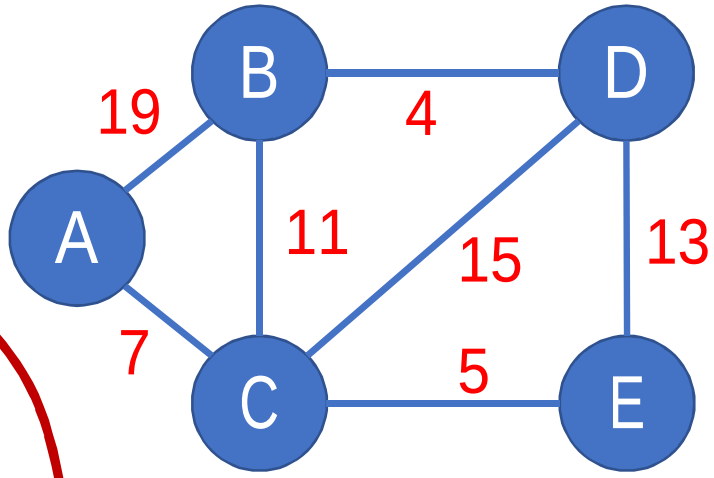
Node A: Updates for B,D,E
Node B: Updates for A, E
Node C: No Updates
Node D: Updates for A
Node E: Updates for A, B

Dest	Cost	N. Hop
A	7	A
B	11	B
C	0	C
D	15	D
E	5	E

Dest	Cost	N. Hop
A	∞	--
B	∞	--
C	5	C
D	13	D
E	0	E

Round: 1

Dest	Cost	N. Hop
A	0	A
B	19	B
C	7	C
D	∞	--
E	∞	--

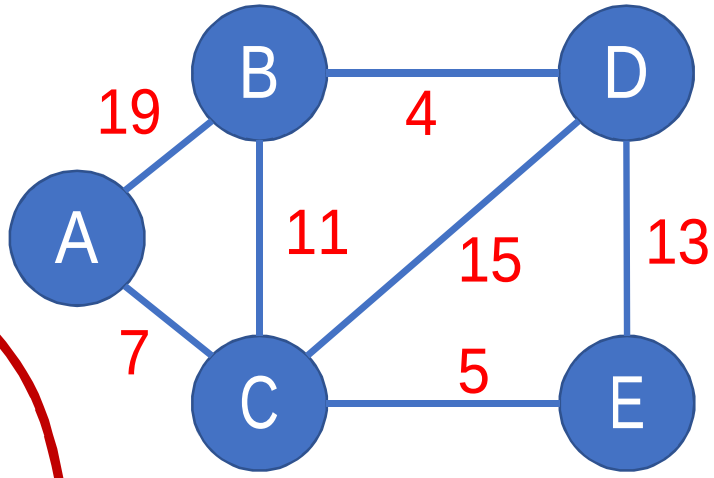


Node A: Updates for B,D,E
Node B: Updates for A, E
Node C: No Updates
Node D: Updates for A
Node E: Updates for A, B

Dest	Cost	N. Hop
A	7	A
B	11	B
C	0	C
D	15	D
E	5	E

Round: 1

Dest	Cost	N. Hop
A	0	A
B	18	C
C	7	C
D	∞	--
E	∞	--

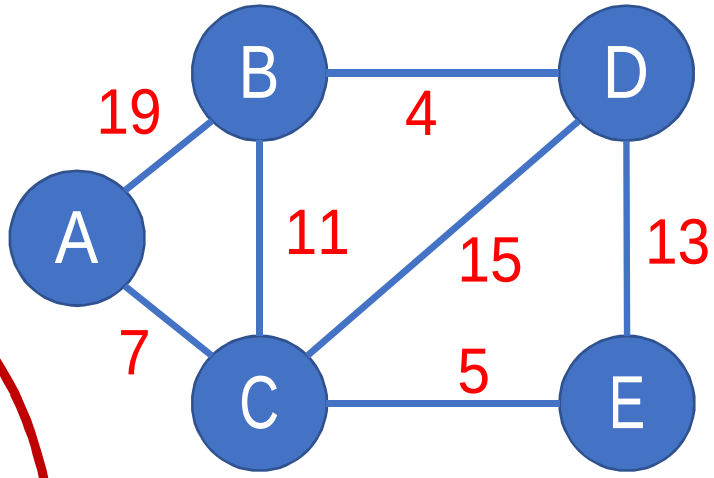


Node A: Updates for B,D,E
Node B: Updates for A, E
Node C: No Updates
Node D: Updates for A
Node E: Updates for A, B

Dest	Cost	N. Hop
A	7	A
B	11	B
C	0	C
D	15	D
E	5	E

Round: 1

Dest	Cost	N. Hop
A	0	A
B	18	C
C	7	C
D	∞	--
E	∞	--

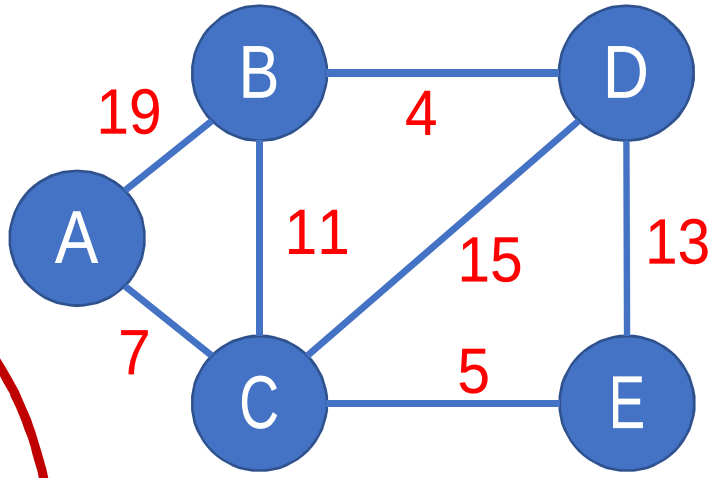


Node A: Updates for B,D,E
Node B: Updates for A, E
Node C: No Updates
Node D: Updates for A
Node E: Updates for A, B

Dest	Cost	N. Hop
A	7	A
B	11	B
C	0	C
D	15	D
E	5	E

Round: 1

Dest	Cost	N. Hop
A	0	A
B	18	C
C	7	C
D	22	C
E	∞	--



Node A: Updates for B,D,E
Node B: Updates for A, E
Node C: No Updates
Node D: Updates for A
Node E: Updates for A, B

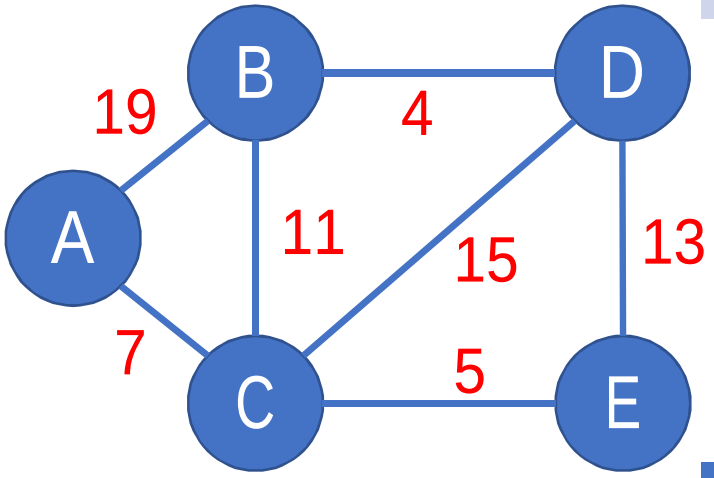
Dest	Cost	N. Hop
A	7	A
B	11	B
C	0	C
D	15	D
E	5	E

Round: 1

Dest	Cost	N. Hop
A	0	A
B	18	C
C	7	C
D	22	C
E	12	C

Dest	Cost	N. Hop
A	18	C
B	0	B
C	11	C
D	4	D
E	16	C

Dest	Cost	N. Hop
A	22	C
B	4	B
C	15	C
D	0	D
E	13	E



Node A: Updates for B,D,E
Node B: Updates for A, E
Node C: No Updates
Node D: Updates for A
Node E: Updates for A, B

Dest	Cost	N. Hop
A	7	A
B	11	B
C	0	C
D	15	D
E	5	E

Dest	Cost	N. Hop
A	12	C
B	16	C
C	5	C
D	13	D
E	0	E

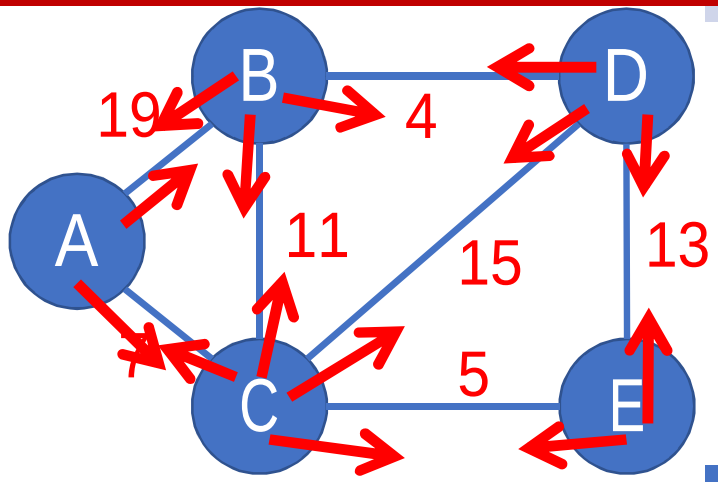
Round: 1

Dest	Cost	N. Hop
A	18	C
B	0	B

Dest	Cost	N. Hop
A	22	C

When will a node send routing updates to its neighbors?

Dest	Cost	N. Hop
A	0	A
B	18	C
C	7	C
D	22	C
E	12	C



Node A: Updates for B,D,E
Node B: Updates for A, E
Node C: No Updates
Node D: Updates for A
Node E: Updates for A, B

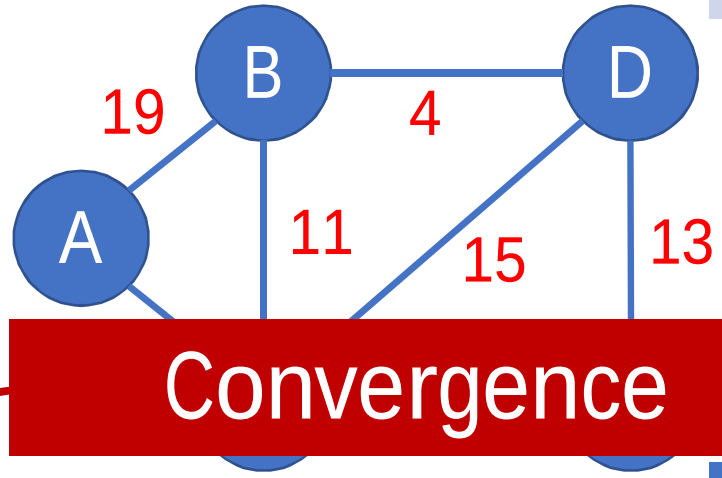
Dest	Cost	N. Hop
A	7	A
B	11	B
C	0	C
D	15	D
E	5	E

Dest	Cost	N. Hop
A	12	C
B	16	C
C	5	C
D	13	D
E	0	E

Dest	Cost	N. Hop
A	0	A
B	18	C
C	7	C
D	22	C
E	12	C

Dest	Cost	N. Hop
A	18	C
B	0	B
C	11	C
D	4	D
E	16	C

Dest	Cost	N. Hop
A	22	C
B	4	B
C	15	C
D	0	D
E	13	E



Node A: No Updates
Node B: No Updates
Node C: No Updates
Node D: No Updates
Node E: No Updates

Dest	Cost	N. Hop
A	7	A
B	11	B
C	0	C
D	15	D
E	5	E

Dest	Cost	N. Hop
A	12	C
B	16	C
C	5	C
D	13	D
E	0	E

Points to Note

- No topology change, convergence in a few rounds
  - After one message exchange, each node knows about nodes two hops away
  - After two message exchanges, each node knows about nodes three hops away
  - And so on...

Bellman-Ford Algorithm

- The heart of distance-vector routing is the famous Bellman-Ford equation (distributed,dynamic,iterative).
- Define distances at each node x
  - $d_x(y)$ = cost of least-cost path from x to y
- Update distances based on neighbors
  - $d_x(y) = \min \{c(x,v) + d_v(y)\}$ over all neighbors v

Shortest-Path Routing

- Path-selection model
  - Destination-based
  - Load-insensitive (e.g., static link weights)
  - Minimum hop count or sum of link weights

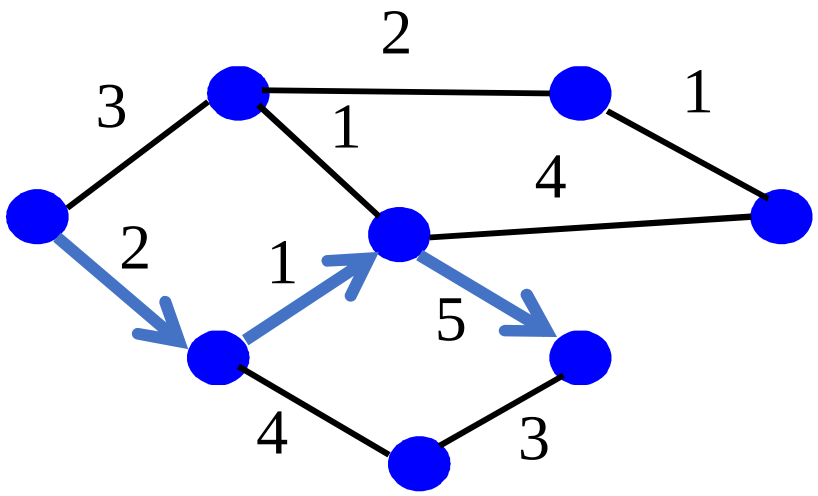
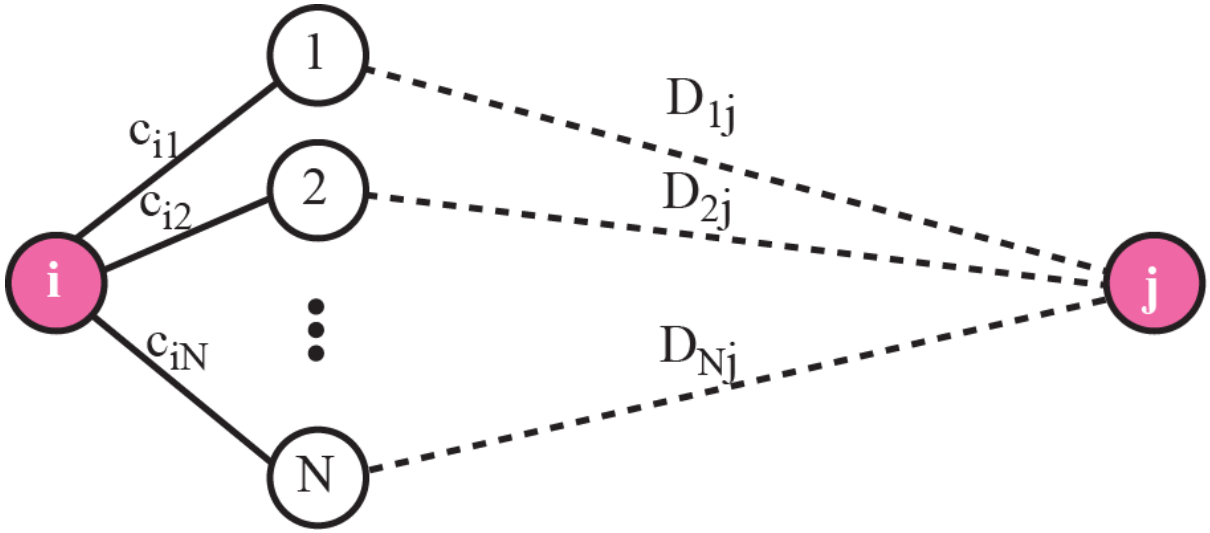


Figure 11.4 *The fact behind Bellman-Ford algorithm*

$$D_{ij} = \text{minimum } \{(c_{i1} + D_{1j}) , (c_{i2} + D_{2j}), \ldots (c_{iN} + D_{Nj})$$



Legend

D_{ij} Shortest distance between i and j
c_{ij} Cost between i and j
N Number of nodes

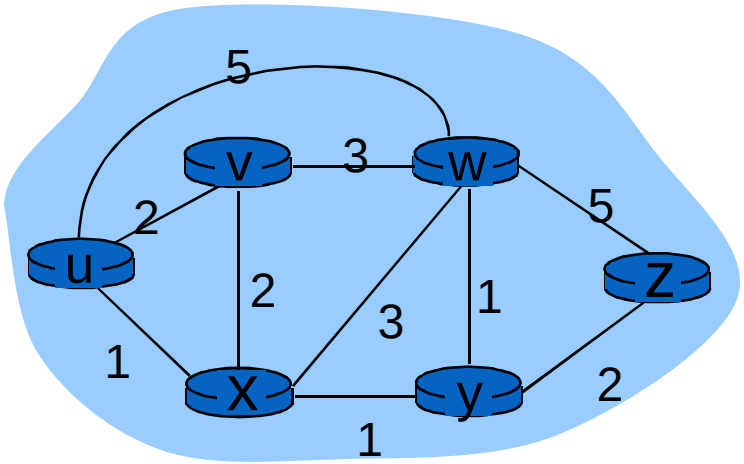
- Node x maintains state:
  - c(x,v) = cost for direct link from x to neighbor v
  - Distance vector $D_x(y)$ (estimate of least cost x to y) for *all* nodes y
  - Distance vector $D_v(y)$ for each neighbor v, for all y
- Node x periodically sends D_x to its neighbors v
  - Neighbors update their own distance vectors:

    $D_v(y) \leftarrow \min_x\{c(v,x) + D_x(y)\}$ for each node $y \in N$
- Over time, the distance vector D_x converges

Distance Vector Algorithm

- c(x,v) = cost for direct link from x to v
  - Node x maintains costs of direct links c(x,v)
- $D_x(y)$ = estimate of least cost from x to y
  - Node x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N]$
- Node x maintains its neighbors' distance vectors
  - For each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$
- Each node v periodically sends D_v to its neighbors
  - And neighbors update their own distance vectors
  - $D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\}$ for each node $y \in N$
- Over time, the distance vector D_x converges

Bellman-Ford example



clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z), \; c(u,x) + d_x(z), \; c(u,w) + d_w(z) \}$$
$$= \min \{2 + 5, \; 1 + 3, \; 5 + 3\} = 4$$

node achieving minimum is next hop in shortest path, used in forwarding table

Distance vector algorithm

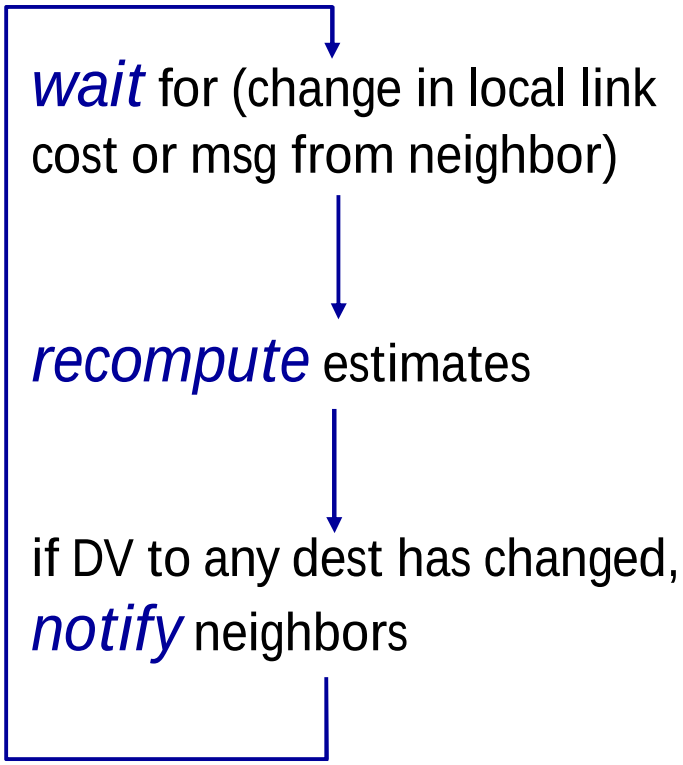
iterative, asynchronous:
each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed:

- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

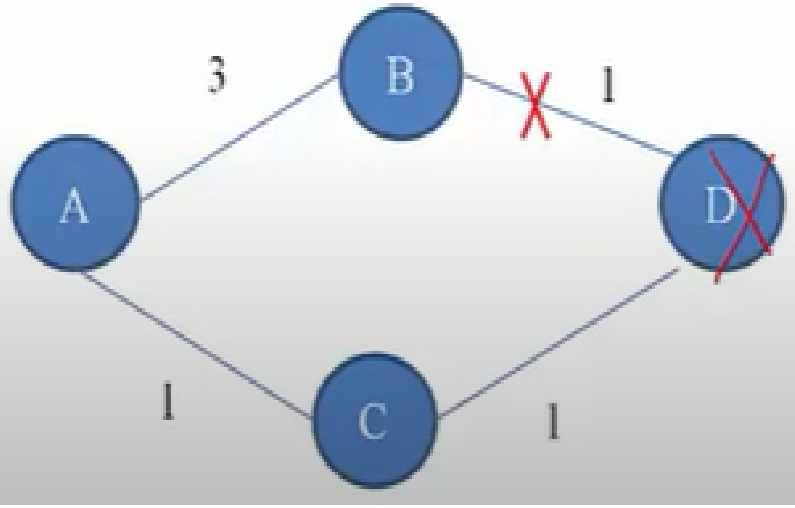
each node:



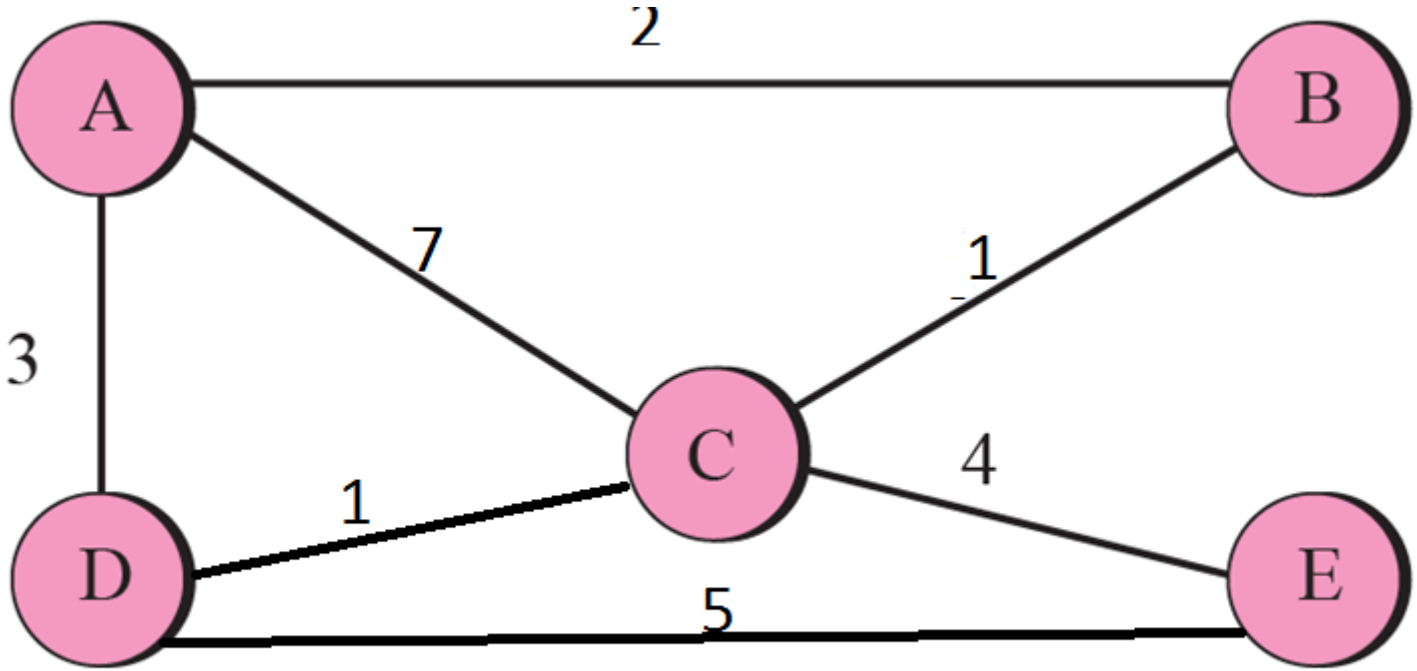
```
1  Distance_Vector_Routing ( )
2  {
3      // Initialize (create initial vectors for the node)
4      D[myself] = 0
5      for (y = 1 to N)
6      {
7          if (y is a neighbor)
8              D[y] = c[myself][y]
9          else
10             D[y] = ∞
11     }
12     send vector {D[1], D[2], …, 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
```

Node/Link Failure

- How are node/link failures detected?
  - Didn't receive periodic update
  - Can also actively probe (probe-ack)



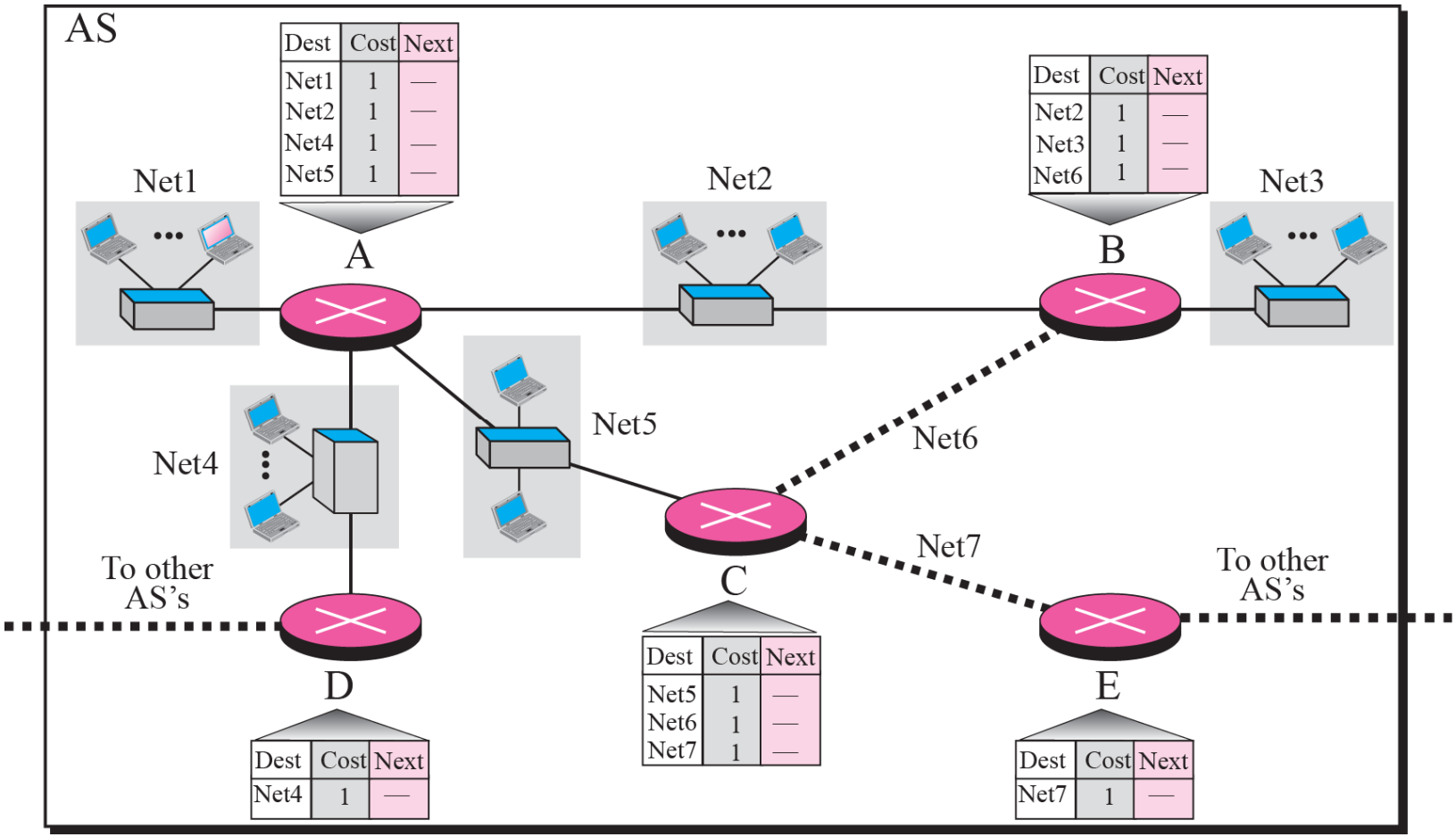
Exercise-Distant vector routing-Bellman-Ford algorithm



Example 11.1

Figure 11.5 shows the initial routing table for an AS. Note that the figure does not mean that all routing tables have been created at the same time; each router creates its own routing table when it is booted.

Figure 11.5 *Example 11.1*



Example 11.2

Now assume router A sends four records to its neighbors, routers B, D, and C. Figure 11.6 shows the changes in B's routing table when it receives these records. We leave the changes in the routing tables of other neighbors as exercise.

Figure 11.6 *Example 11.2*

Routing Table B

Dest	Cost	Next
Net1	2	A
Net2	1	—
Net3	1	—
Net6	1	—

After receiving record 1

Routing Table B

Dest	Cost	Next
Net1	2	A
Net2	1	—
Net3	1	—
Net6	1	—

After receiving record 2

Routing Table B

Dest	Cost	Next
Net1	2	A
Net2	1	—
Net3	1	—
Net4	2	A
Net6	1	—

After receiving record 3

Routing Table B

Dest	Cost	Next
Net1	2	A
Net2	1	—
Net3	1	—
Net4	2	A
Net5	2	A
Net6	1	—

After receiving record 4

Example 11.3

Figure 11.7 shows the final routing tables for routers in Figure 11.5.

Figure 11.7 *Example 11.3*

A

Dest	Cost	Next
Net1	1	—
Net2	1	—
Net3	2	B
Net4	1	—
Net5	1	—
Net6	2	C
Net7	2	C

B

Dest	Cost	Next
Net1	2	A
Net2	1	—
Net3	1	—
Net4	2	A
Net5	2	A
Net6	1	—
Net7	2	C

C

Dest	Cost	Next
Net1	2	A
Net2	2	A
Net3	2	B
Net4	2	A
Net5	1	—
Net6	1	—
Net7	1	—

D

Dest	Cost	Next
Net1	2	A
Net2	2	A
Net3	3	A
Net4	1	—
Net5	1	A
Net6	3	A
Net7	3	A

E

Dest	Cost	Next
Net1	3	C
Net2	3	C
Net3	3	C
Net4	3	C
Net5	2	C
Net6	2	C
Net7	1	—

How are node/Link failures detected?

- Didn’t receive periodic update (No keep alive packets)

- Can also actively probe/query neighbours (thru Probe/Probe ACK messages)

Problems in Distance Vector

- Count to Infinity – packet bouncing and looping
- Two node loop

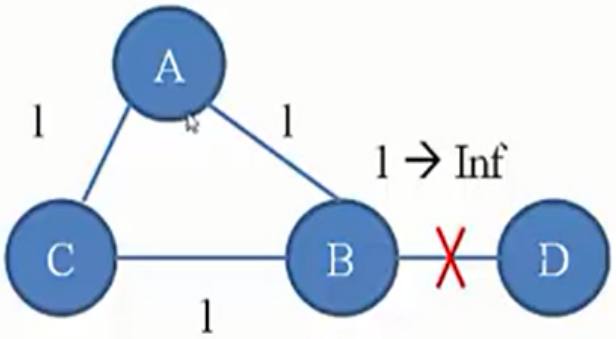
Partial Solutions:

- 1) Define Infinity to 16(DVR cannot be used in large systems)
- 2) Split Horizon
- 3) Split Horizon and Poison Reverse

Count to node infinity

Distance to Node D

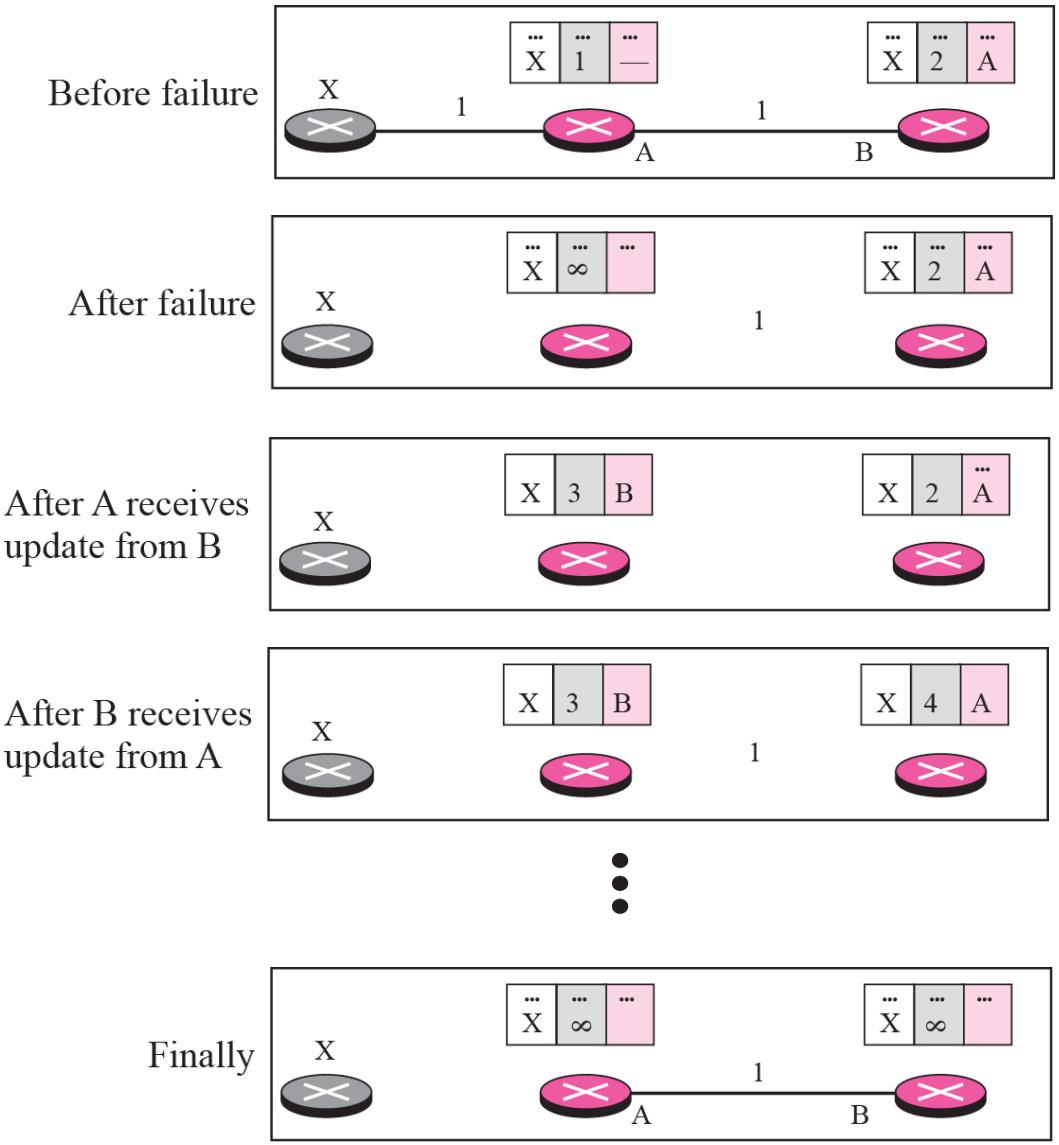
Mesg.	A	B	C
	2,B	∞,-	2,B
B→A	∞,-	∞,-	2,B
C→A	3,C	∞,-	2,B
B→C	3,C	∞,-	∞,-
A→B	3,C	4,A	∞,-
C→A	∞,-	4,A	∞,-
B→C	∞,-	4,A	5,B
A→B	∞,-	∞,-	5,B



States maintained by A,B,C

- Decrease in Cost (good news) propagates quickly
- But increase in Cost (bad news) propagates slowly
- Since broken link results in increase in cost it takes considerable time to update all the neighboring routers

Figure 11.8 *Two-node instability*



Two-Node Instability (1)

- Defining Infinity
  - Use for example 16 to represent infinity (assumes max no of hops under 16)
  - Bounds time it takes to count to infinity
- Split Horizon
  - Split Horizon rule:
    A router should not advertise a network through the same interface from which the update came.
  - Send a partial table.

Split Horizon

- Node B takes info from Node A and modifies. This is the source of problem/CONFUSION

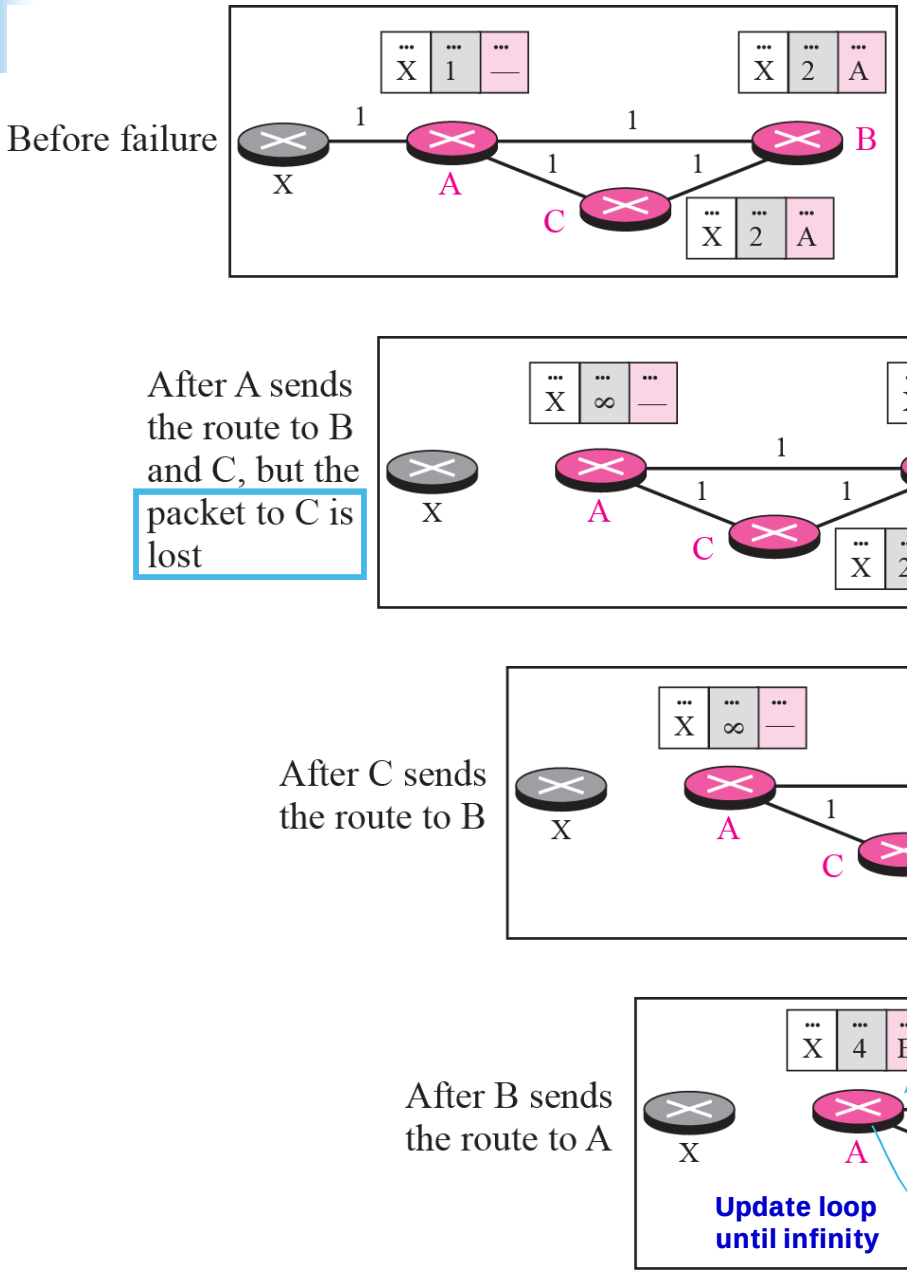
- So, **Node B, will not send information learnt from A back to A.** So last line in Routing table **pertaining to reaching X, via A is not sent to A**

- So A, retains “infinity” to X
- When A sends routing record: “X, infinity” to node B, node B corrects the same.
- System is stable after first update. A and B knows, X
- is not reachable

Two-Node Instability (2)

- Split Horizon and Poison Reverse
  - One drawback of Split Horizon
    - Normally, the DV protocol uses a timer and if there is no news about a route, the node deletes the route from its table
    - In the previous e.g., **node A cannot guess that this is due to split horizon or because B has not received any news about X recently**
  - Poison Reverse
    - Node B can still advertise the value for X, but is the source of information is A, **it can replace the distance with infinity as a warning**
    - Once a router learns of an unreachable route, advertise it back through the same interface as unreachable setting the value to infinity

Figure 11.9 *Three-node instability*



If the instability is btw three nodes, stability cannot be guaranteed

RIP

RIP (Routing Information Protocol)

- Distance vector algorithm-**Classful routing**
- Included in BSD-UNIX Distribution in 1982
- Distance metric: # of hops (max = 15 hops)
- Distance vectors: exchanged every 30 sec via Response Message (also called **advertisement**)
- Each advertisement: route to up to 25 destination nets
- Value of 16 is reserved to represent infinity.
- suitable for small networks (local area environments)
- ***RIP uses the services of UDP on well-known port 520.***

RIP: Link Failure and Recovery

If no advertisement heard after 180 sec --> neighbor/link declared dead

- routes via neighbor invalidated
- new advertisements sent to neighbors
- neighbors in turn send out new advertisements (if tables changed)
- link failure info quickly propagates to entire net
- poison reverse used to prevent ping-pong loops (infinite distance = 16 hops)

Routing table for RIP

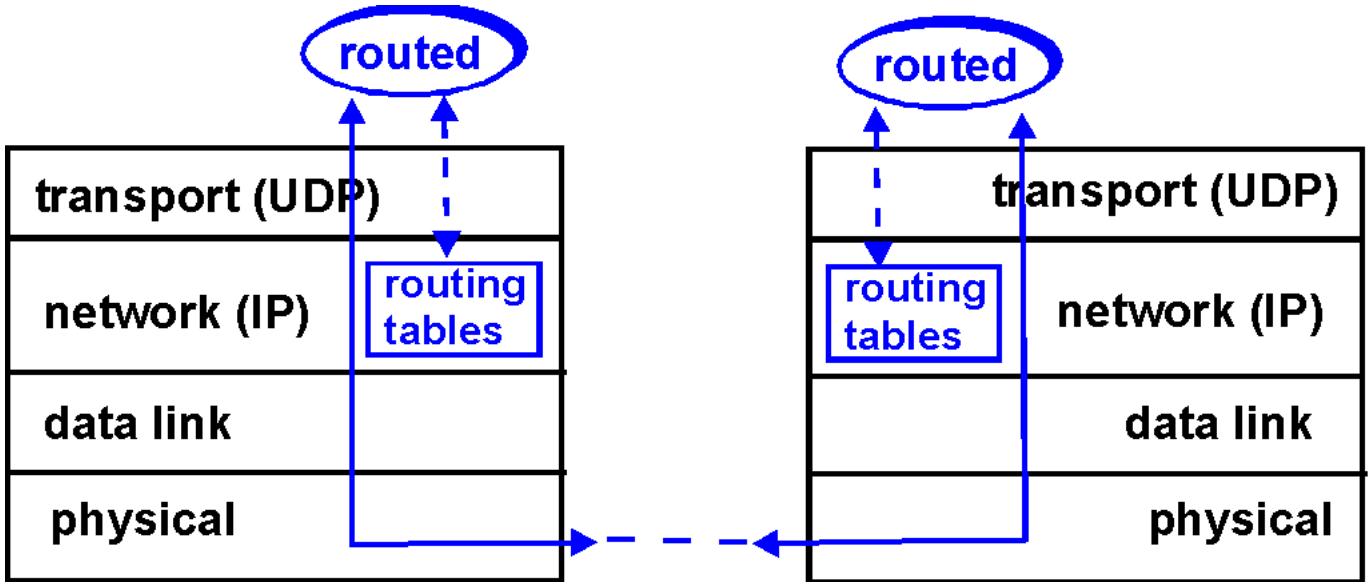
- **Has one entry for each destination network of which the router is aware.**
- **Each entry has destination network address, the shortest distance to reach the destination in hop count, and next router to which the packet should be delivered to reach its final destination.**

RIP ROUTING TABLE

Destination	Hop Count	Next Router	Other information
163.5.0.0	7	172.6.23.4	
197.5.13.0	5	176.3.6.17	
189.45.0.0	4	200.5.1.6	
115.0.0.0	6	131.4.7.19	

RIP Table Processing

- RIP routing tables managed by **application-level** process called route-d (daemon)
- advertisements sent in UDP packets, periodically repeated



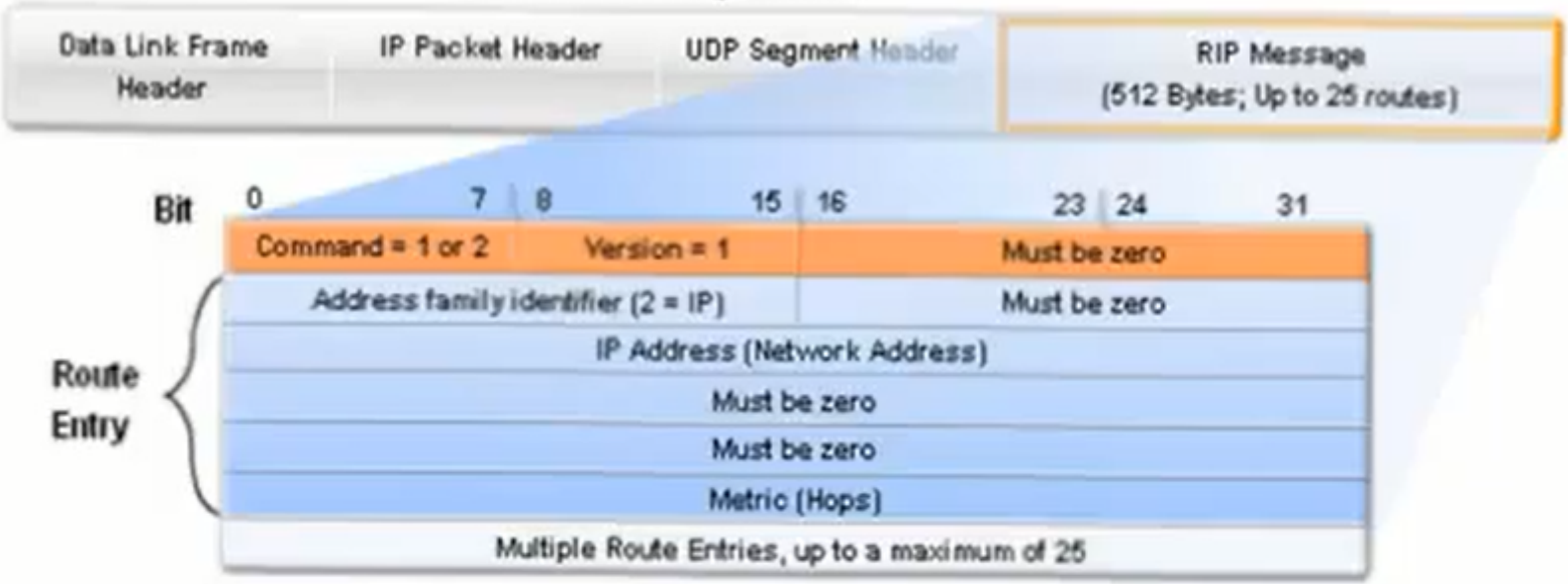
- RIP Operation
  - RIP uses 2 message types:
    - Request message

      -This is sent out on startup by each RIP enabled interface

      -Requests all RIP enabled neighbors to send routing table
    - Response message

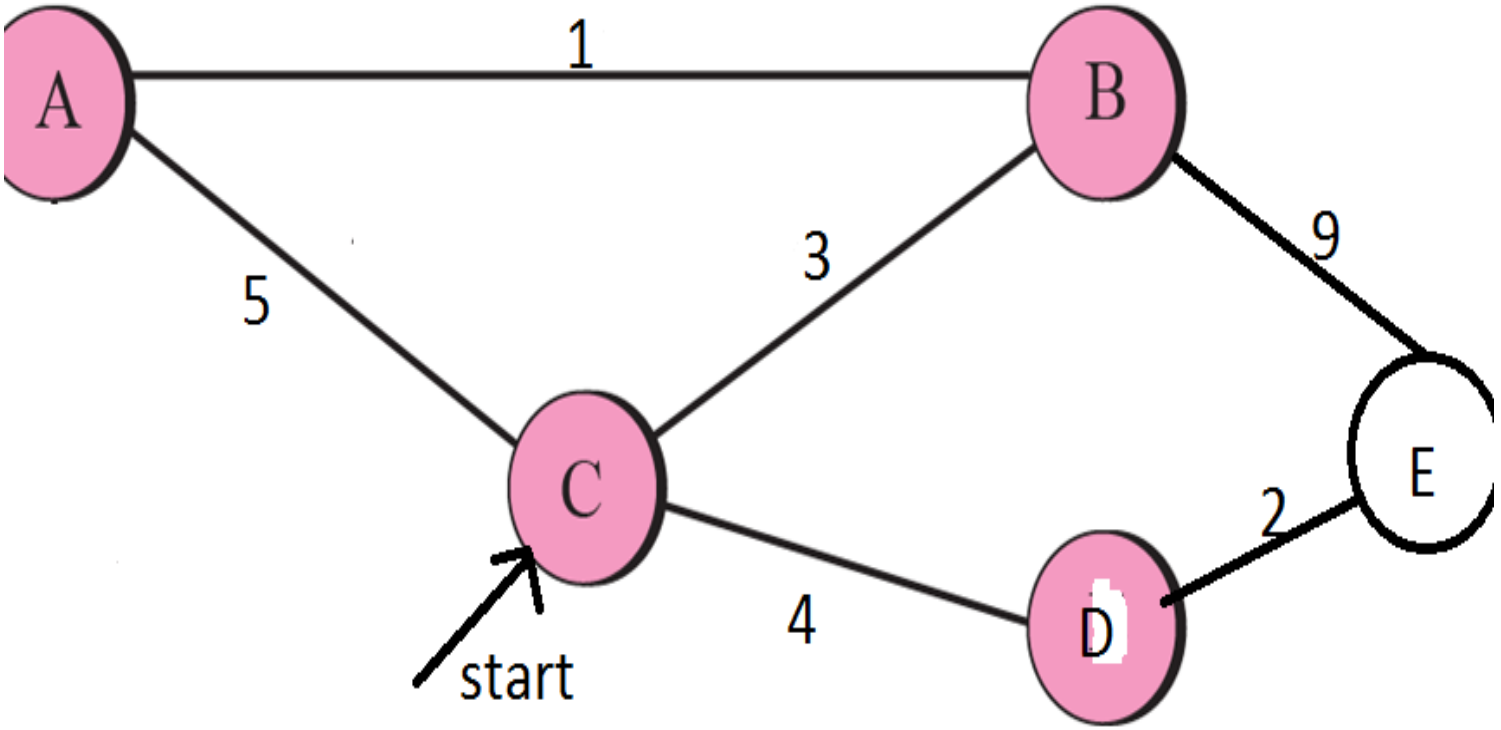
      -Message sent to requesting router containing routing table

RIPv1 Message Format



Command	1 for a Request or 2 for a Reply.
Version	1 for RIP v 1 or 2 for RIP v 2.
Address Family Identifier	2 for IP unless a Request is for the full routing table in which case, set to 0.
IP Address	The address of the destination route, which may be a network, subnet, or host address.
Metric	Hop count between 1 and 16. Sending router increases the metric before sending out message.

Example – DV routing



- Initial table at B – Table 0

Initial Table at B		
Destination	**Cost**	**Next Hop**
A	1	A
C	3	C
E	9	E

Initial Table at A – Table 2

Initial Table at A		
Destination	**Cost**	**Next Hop**
B	1	B
C	5	C

- Initial Table at C – Table 1

Initial Table at C		
Destination	**Cost**	**Next Hop**
A	5	A
B	3	B
D	4	D

- Routing updates from A arrive at C – Table 1:

Updated Table at C		
Destination	**Cost**	**Next Hop**
A	5	A [retain]
B	3	B [RETAIN]
D	4	D

- Cost(C,A)=5
- C->B=MIN{3, 5+1}=3 [Old cost better]

– Table at C – Table 1

Updated Table at C		
Destination	**Cost**	**Next Hop**
A	5	A
B	3	B
D	4	D

Initial Table at B – Table 0

Initial Table at B		
Destination	**Cost**	**Next Hop**
A	1	A
C	3	C
E	9	E

– **Routing updates from B arrive at C:**

– Record 1: [A, 1, A]
  - Cost C->B=3
  - D(C,A)=min{5, 3+1}=4 [take the new routing msg update]
  - Update A as [A, 4, B]

– Record 2: [C, 3, C]
  - C->C: Current cost=0
  - So ignore this record

– Record 3: [E, 9 ,E]
  - At C: Add this record

New routing Table at C: Table 3

Updated Table at C		
Destination	**Cost**	**Next Hop**
A	4	B
B	3	B
D	4	D
E	3+9=12	B

Comparison of Protocols

Link State

- Knowledge of every router's links (entire graph)
- Every router has O(# edges)
- Trust a peer's info, do routing computation yourself
- Use Dijkstra's algorithm
- Send updates on any link-state changes
- Ex: OSPF, IS-IS
- Adv: Fast to react to changes

Distance Vector

- Knowledge of neighbors' distance to destinations
- Every router has O (#neighbors * #nodes)
- Trust a peer's routing computation
- Use Bellman-Ford algorithm
- Send updates periodically or routing decision change
- Ex: RIP, IGRP
- Adv: Less info & lower computational overhead

- The reason for the count-to-infinity problem is that each node only has a "next-hop-view"
- For example, in the first step, A did not realize that its route (with cost 2) to C went through node B

- Poison Reverse and Split Horizon
  - Drawback of Split Horizon:
    - Normally, the DV protocol uses a timer and if there is no news about a route, the node deletes the route from its table
    - In the previous e.g., node A cannot guess that this is due to split horizon or because B has not received any news about X recently
  - Poison Reverse: This strategy dictates that Node 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; what I know about this route comes from you."