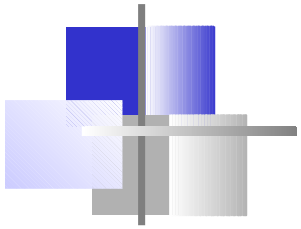


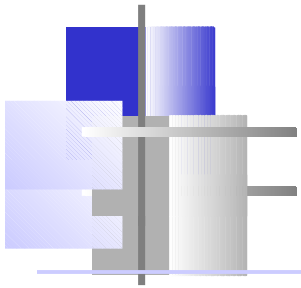
Distance Vector Routing - RIP

a) The idea behind the distance-vector algorithm is suggested by its name. (The other common name for this class of algorithm is Bellman-Ford, after its inventors.

b) Each node constructs a one-dimensional array (a vector) containing the “distances” (costs) to all other nodes and distributes that vector to its immediate neighbors.

c) The starting assumption for distance-vector routing is that each node knows the cost of the link to each of its directly connected neighbors.

d) These costs may be provided when the router is configured by a network manager. A link that is down / not reachable is assigned an infinite cost.

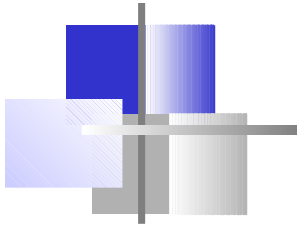


Distance Vector Routing

e) The least-cost route between any two nodes is the route with minimum distance.

f) Each node maintains a vector (table) of minimum distances to every node.

g) The table at each node also guides the packets to the desired node by showing the showing the next hop routing.



Initialization

a) The table in figure is stable.

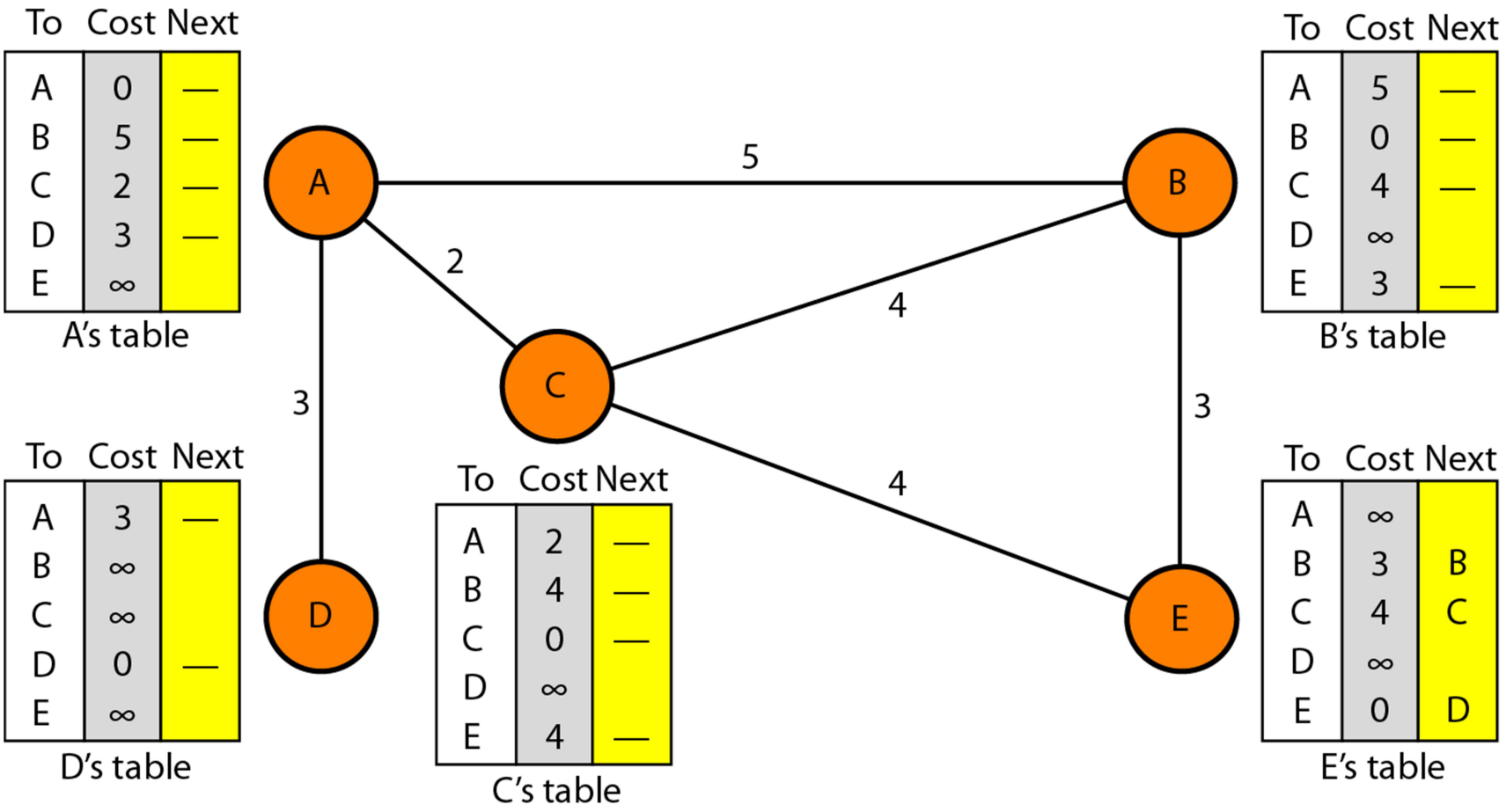
b) Each node knows how to reach any other node and their cost.

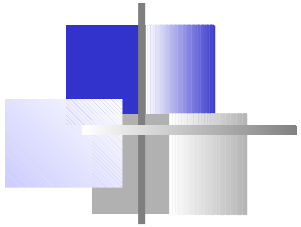
c) At the beginning, each node know the cost of itself and its immediate neighbor.[those node directly connected to it.]

d) Assume that each node send a message to the immediate neighbors and find the distance between itself and these neighbors.

e) The distance of any entry that is not a neighbor is marked as infinite (unreachable).

Initialization of tables in distance vector routing (DVR)





Sharing

a) Idea is to share the information between neighbors.

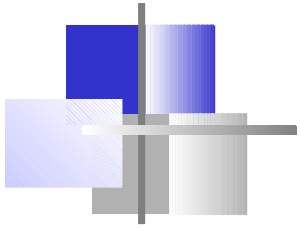
b) The node A does not know the distance about E, but node C does.

c) If node C share it routing table with A, node A can also know how to reach node E.

d) On the other hand, node C does not know how to reach node D, but node A does.

e) If node A share its routing table with C, then node C can also know how to reach node D.

f) Node A and C are immediate neighbors, can improve their routing tables if they help each other.



Sharing Continued…

a) How much of the table must be shared with each neighbor?

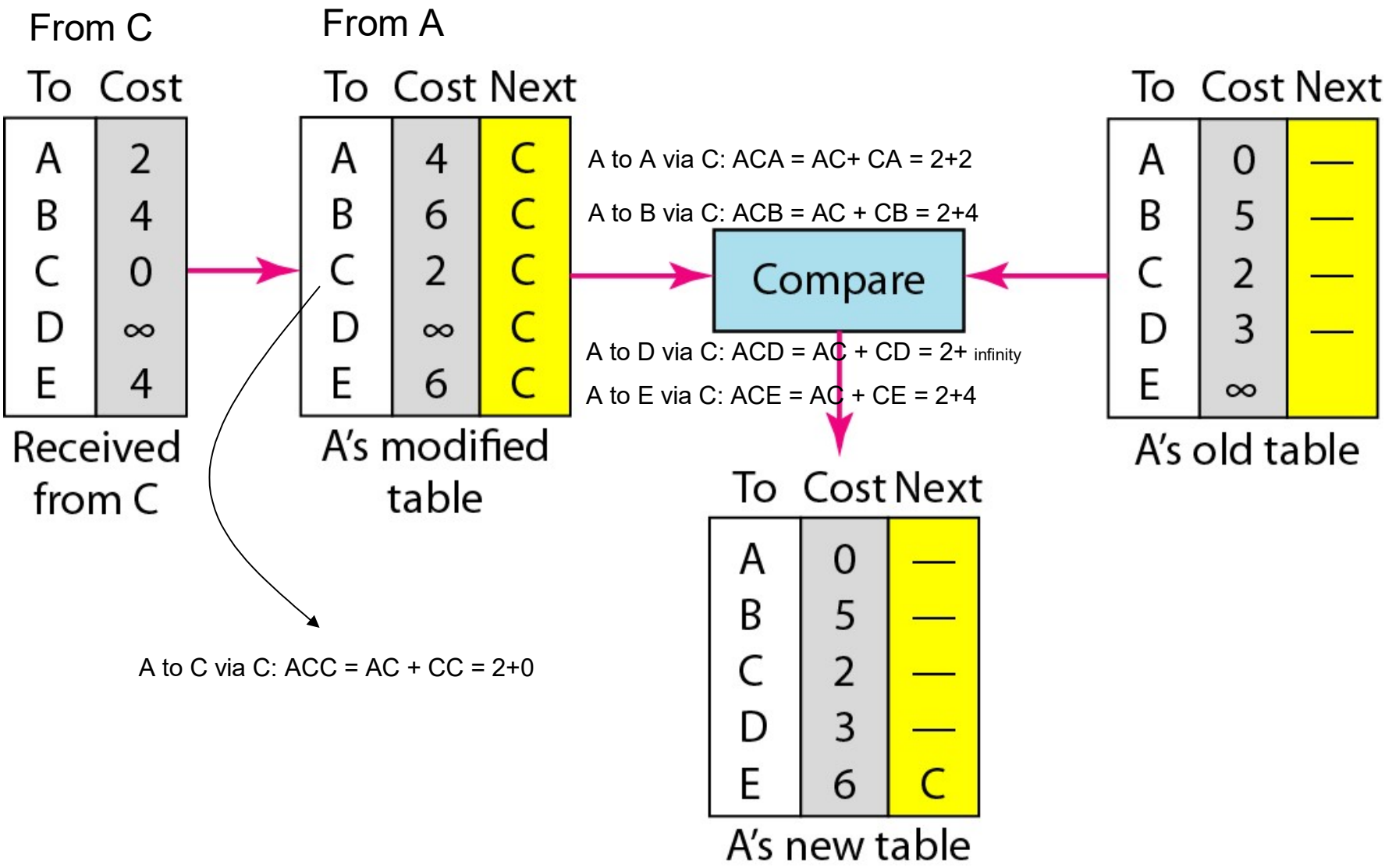
b) The third column of the table(next hop) is not useful for the neighbor.

c) When the neighbor receives a table, this column needs to be replaced with the sender's name.

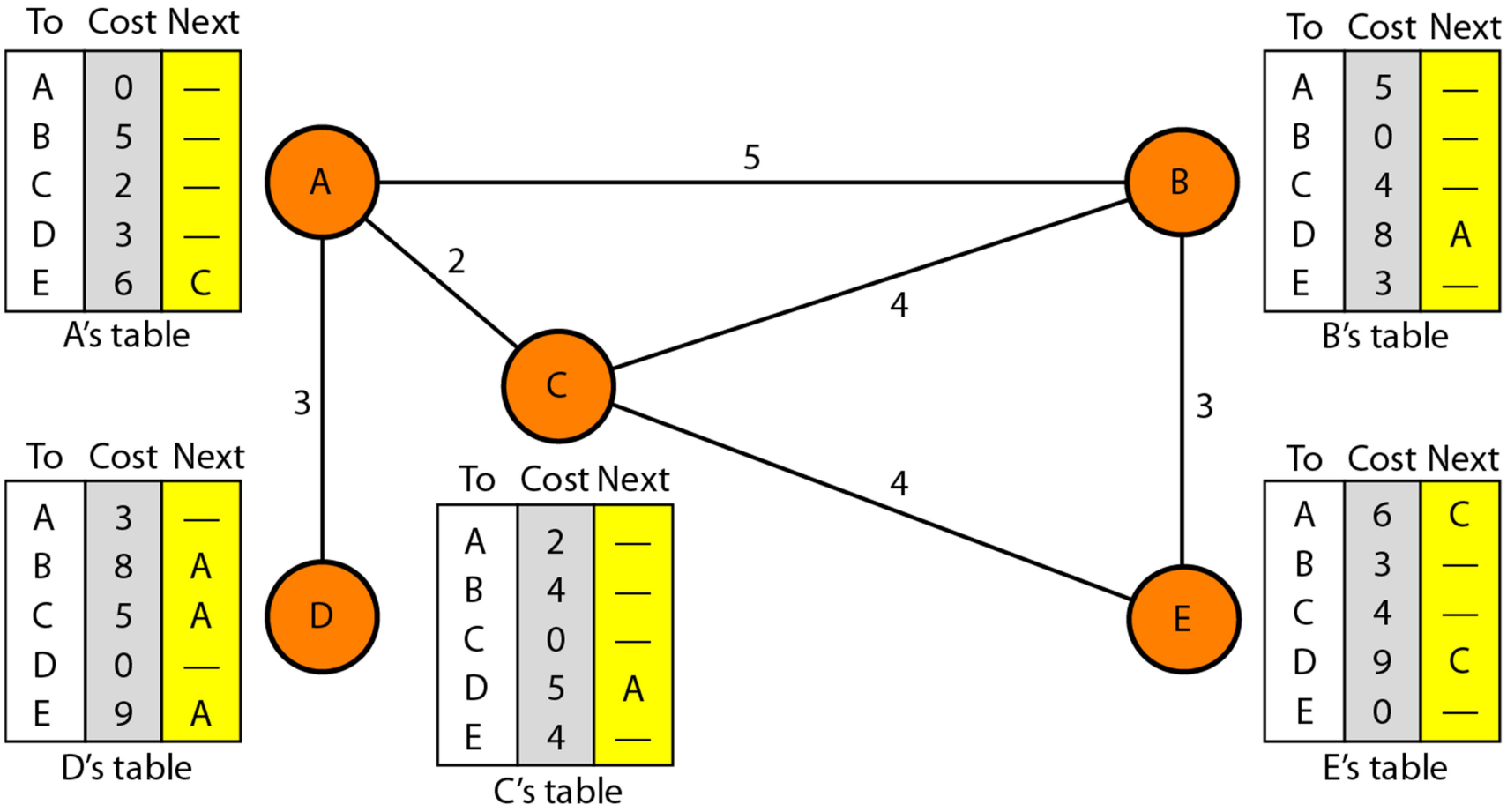
d) If any of the rows can be used, the next node column filled with sender of the table.

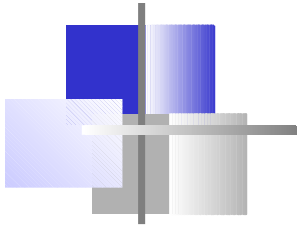
e) Therefore, a node can send only the first two column of its table to any neighbor.

Updating in distance vector routing example: C to A



Final Distance vector routing tables





When to share

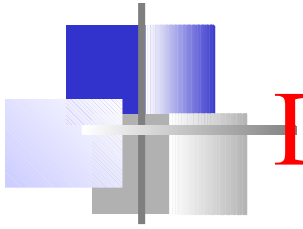
a) Periodic Update: A node sends its table, normally every 30s, in a periodic update, it depends on the protocol that is using DVR.

b) Triggered Update: A node sends its two-column routing table to its neighbors anytime there is a change in its routing table.

c) This is called triggered update the change can result from the following:

✓ A node receives a table from a neighbor, resulting in changes in its own table after updating.

✓ A node detects some failure in the neighboring links which results in a distance change to infinity.



Distance Vector Routing (DVR)

a) 3 keys to understand how this algorithm works:

- **Sharing knowledge about the entire AS**. Each router shares its knowledge about the entire AS with neighbours. It sends whatever it has.
- **Sharing only with immediate neighbours**. Each router sends whatever knowledge it has via **all** available interface.
- **Sharing at regular intervals**. sends at fixed intervals, E.g. every 30 sec.

a) Problems: Tedious comparing/updating process, slow response to infinite loop problem, huge list to be maintained!!

Updating

When a node receives a two-column table from a neighbor, it needs to update its routing table. Updating takes three steps:

1. The receiving node needs to add the cost between itself and the sending node to each value in the second column. The logic is clear. If node C claims that its distance to a destination is x mi, and the distance between A and C is y mi, then the distance between A and that destination, via C, is $x + y$ mi.
2. The receiving node needs to add the name of the sending node to each row as the third column if the receiving node uses information from any row. The sending node is the next node in the route.
3. The receiving node needs to compare each row of its old table with the corresponding row of the modified version of the received table.
   a. If the next-node entry is different, the receiving node chooses the row with the smaller cost. If there is a tie, the old one is kept.
   b. If the next-node entry is the same, the receiving node chooses the new row. For example, suppose node C has previously advertised a route to node X with distance 3. Suppose that now there is no path between C and X; node C now advertises this route with a distance of infinity. Node A must not ignore this value even though its old entry is smaller. The old route does not exist any more. The new route has a distance of infinity.

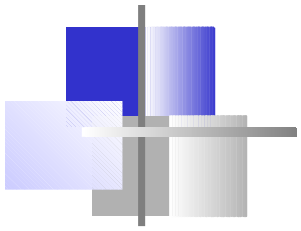
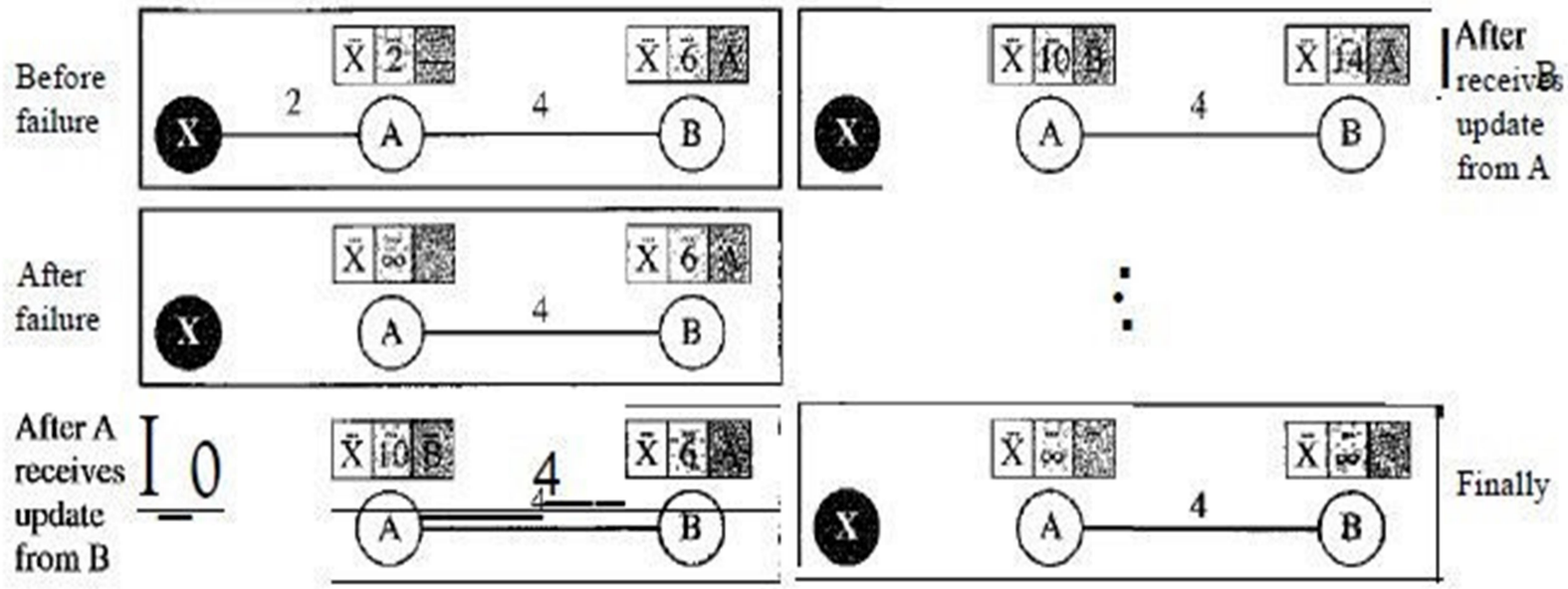


Figure 22.17 *Two-node instability*



Defining Infinity The first obvious solution is to redefine infinity to a smaller number, such as 100. For our previous scenario, the system will be stable in less than 20 updates. As a matter of fact, most implementations of the distance vector protocol define the distance between each node to be I and define 16 as infinity. However, this means that the distance vector routing cannot be used in large systems. The size of the network, in each direction, can not exceed 15 hops.

Split Horizon Another solution is called split horizon. In this strategy, instead of flooding the table through each interface, each node sends only part of its table through each interface. If, according to its table, 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 corne from A (A already knows). Taking information from node A, modifying it, and sending it back to node A creates the confusion. In our scenario, node B eliminates the last line of its routing table before it sends it to A. In this case, node A keeps the value of infinity as the distance to X. Later when node A sends its routing table to B, node B also corrects its routing table. The system becomes stable after the first update: both node A and B know that X is not reachable.

Split Horizon and Poison Reverse Using the split horizon strategy has one drawback. Normally, the distance vector protocol uses a timer, and if there is no news about a route, the node deletes the route from its table. When node B in the previous scenario eliminates the route to X from its advertisement to A, node A cannot guess that this is due to the split horizon strategy (the source of information was A) or because B has not received any news about X recently. The split horizon strategy can be combined with the poison reverse strategy. 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."

Distance vector vs Link state routing algorithms

Features	Distance Vector	Link State
Convergence	Slow	Fast
Updates	Frequently	Event Triggered
Loops	Prone to routing Loops	Less Subjected to Routing Loops
Configuration	Easy	Difficult
Network Types	Broadcast for updates sent	Multicast for updates sent
Topology	doesn't know Network Topology	Knows entire Network Topology
Automatic Route Summarization	No	Yes
Path Calculation	Hop Count	Shortest Path -Metric
Scalability	Limited	Can be highly scalable
Protocols	RIP, IGRP	OSPF, IS-IS
Algorithm	Bredford Algorithm	Dijkstra-algorithm
Manual Route Summarization	Yes	Yes
Metric	Hop Count	Link Cost