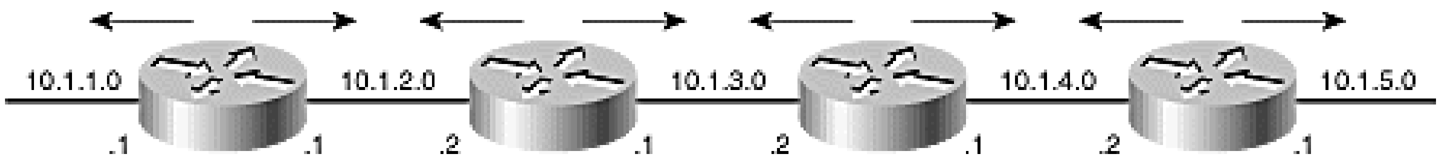
# Distance-Vector Routing: Distributed B-F (cont.)

## Example [ Distributed Bellman-Ford Algorithm ]




### $t_0$

**Router A**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.1.0 | -- | 0 |
| 10.1.2.0 | -- | 0 |

**Router B**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.2.0 | -- | 0 |
| 10.1.3.0 | -- | 0 |

**Router C**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.3.0 | -- | 0 |
| 10.1.4.0 | -- | 0 |

**Router D**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.4.0 | -- | 0 |
| 10.1.5.0 | -- | 0 |

### $t_1$

**Router A**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.1.0 | -- | 0 |
| 10.1.2.0 | -- | 0 |
| 10.1.3.0 | 10.1.2.2 | 1 |

**Router B**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.2.0 | -- | 0 |
| 10.1.3.0 | -- | 0 |
| 10.1.1.0 | 10.1.2.1 | 1 |
| 10.1.4.0 | 10.1.3.2 | 1 |

**Router C**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.3.0 | -- | 0 |
| 10.1.4.0 | -- | 0 |
| 10.1.2.0 | 10.1.3.1 | 1 |
| 10.1.5.0 | 10.1.4.2 | 1 |

**Router D**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.4.0 | -- | 0 |
| 10.1.5.0 | -- | 0 |
| 10.1.3.0 | 10.1.4.1 | 1 |

### $t_2$

**Router A**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.1.0 | -- | 0 |
| 10.1.2.0 | -- | 0 |
| 10.1.3.0 | 10.1.2.2 | 1 |
| 10.1.4.0 | 10.1.2.2 | 2 |

**Router B**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.2.0 | -- | 0 |
| 10.1.3.0 | -- | 0 |
| 10.1.1.0 | 10.1.2.1 | 1 |
| 10.1.4.0 | 10.1.3.2 | 1 |
| 10.1.5.0 | 10.1.3.2 | 2 |

**Router C**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.3.0 | -- | 0 |
| 10.1.4.0 | -- | 0 |
| 10.1.2.0 | 10.1.3.1 | 1 |
| 10.1.5.0 | 10.1.4.2 | 1 |
| 10.1.1.0 | 10.1.3.1 | 2 |

**Router D**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.4.0 | -- | 0 |
| 10.1.5.0 | -- | 0 |
| 10.1.3.0 | 10.1.4.1 | 1 |
| 10.1.2.0 | 10.1.4.1 | 2 |

### $t_3$

**Router A**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.1.0 | -- | 0 |
| 10.1.2.0 | -- | 0 |
| 10.1.3.0 | 10.1.2.2 | 1 |
| 10.1.4.0 | 10.1.2.2 | 2 |
| 10.1.5.0 | 10.1.2.2 | 3 |

**Router B**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.2.0 | -- | 0 |
| 10.1.3.0 | -- | 0 |
| 10.1.1.0 | 10.1.2.1 | 1 |
| 10.1.4.0 | 10.1.3.2 | 1 |
| 10.1.5.0 | 10.1.3.2 | 2 |

**Router C**

| NET | VIA | HOPS |
|---|---|---|
| 10.1.3.0 | -- | 0 |
| 10.1.4.0 | -- | 0 |
| 10.1.2.0 | 10.1.3.1 | 1 |
| 10.1.5.0 | 10.1.4.2 | 1 |
| 10.1.1.0 | 10.1.3.1 | 2 |

**Router D**

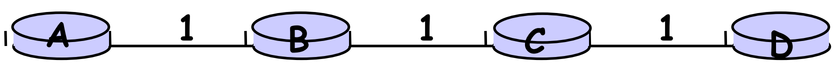
| NET | VIA | HOPS |
|---|---|---|
| 10.1.4.0 | -- | 0 |
| 10.1.5.0 | -- | 0 |
| 10.1.3.0 | 10.1.4.1 | 1 |
| 10.1.2.0 | 10.1.4.1 | 2 |
| 10.1.1.0 | 10.1.4.1 | 3 |

# Distance-Vector Routing: Distributed B-F (cont.)

## Example [ DBF Algorithm – reaction to link failure ]

**DBF algorithm may react very slowly to a link failure!**

**For example, consider the following topology, with node D as the destination.**




routing-table entries for D only, at A, B, C

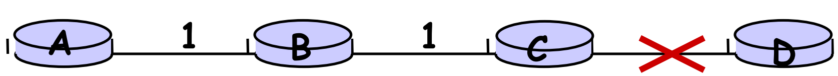
| Update | Node A | Node B | Node C |
|---|---|---|---|
| before break | (D,3,B) | (D,2,C) | (D,1,D) |

**Suppose after the algorithm stabilizes, link (C,D) breaks. Recompute the minimum cost from each node to the destination node (node D).**

**Update packets bounce back and fourth between B and A/C until the minimum cost is ∞ (or very large, in practice). At that point the algorithm realizes that node D is unreachable.**

updates for D only

| Update | Node A | Node B | Node C |
|---|---|---|---|
| | (D, 3) | (D, 2) | ∞ |
| 1 | (D, 3) | (D, 2) | (D, 3) |
| 2 | (D, 3) | (D, 4) | (D, 3) |
| 3 | (D, 5) | (D, 4) | (D, 5) |
| 4 | (D, 5) | (D, 6) | (D, 5) |
| 5 | (D, 7) | (D, 6) | (D, 7) |
| … | … | … | … |




**"count to infinity" problem !!!**

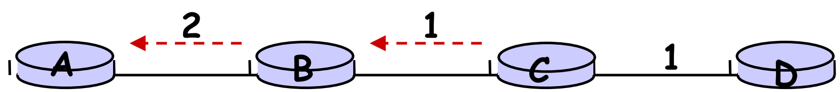
**solution: stop when cost = 16**

# Distance-Vector Routing: Distributed B-F (cont.)

**Count to Infinity Problem** – **slow convergence to a change in topology**

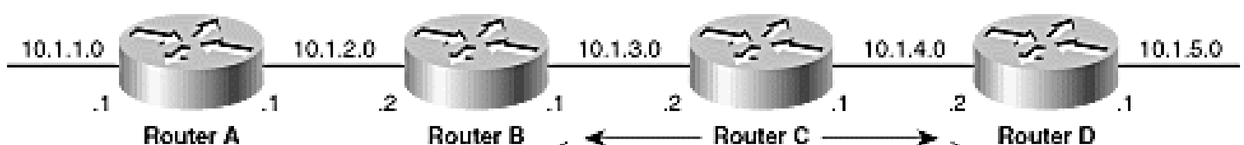
- **could be partially avoided using "split horizon" and "split horizon with poisoned reverse"**

**Split Horizon** – **minimum cost to a given destination should NOT be sent to a neighbour if neighbour is next node along the shortest path**




B should **not** advertise any route to C for which C is the next hop.

**Example** **[ split horizon ]**




| NET | VIA | HOPS |
|---|---|---|
| 10.1.4.0 | - - | 0 |
| 10.1.5.0 | 10.1.4.2 | 1 |

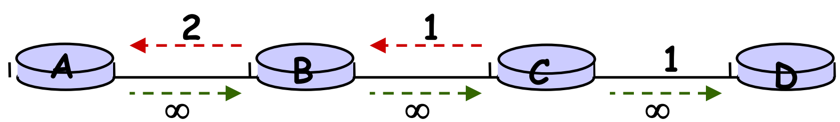
| NET | VIA | HOPS |
|---|---|---|
| 10.1.3.0 | - - | 0 |
| 10.1.2.0 | 10.1.3.1 | 1 |
| 10.1.1.0 | 10.1.3.1 | 2 |

**Split Horizon works by suppressing information!!!**

# Distance-Vector Routing: Distributed B-F (cont.)

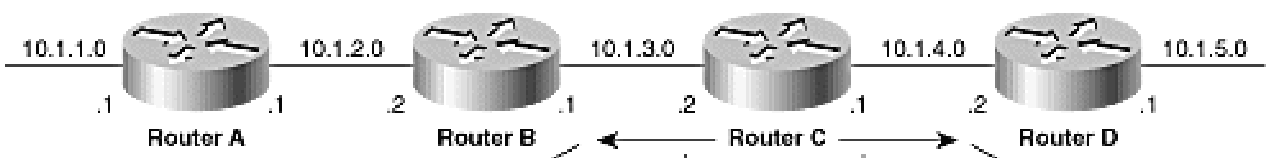
**Split Horizon with Poison Reverse** – **a node sends the minimum cost to all its neighbours; but, the minimum cost to a given destination is set to infinity (∞=16, in practice) if the neighbours is the next node along the shortest path**

- **if two routers have routes pointing at each other, advertising reverse routes with a metric of ∞=16 breaks the loop immediately**




B advertises to C the route to D as a route of cost **∞=16**

## Example [ poisoned reverse ]




| NET | VIA | HOPS |
|---|---|---|
| 10.1.3.0 | - - | inf |
| 10.1.4.0 | - - | 0 |
| 10.1.2.0 | 10.1.3.1 | inf |
| 10.1.5.0 | 10.1.4.2 | 1 |
| 10.1.1.0 | 10.1.3.1 | inf |

| NET | VIA | HOPS |
|---|---|---|
| 10.1.3.0 | - - | 0 |
| 10.1.4.0 | - - | inf |
| 10.1.2.0 | 10.1.3.1 | 1 |
| 10.1.5.0 | 10.1.4.2 | inf |
| 10.1.1.0 | 10.1.3.1 | 2 |

# Routing Algorithms – Comparison (cont.)

## Link State vs. Distance Vector

| | Link State | Distance Vector |
|---|---|---|
| size of (update) routing info | small, contains only neighbours' link costs ☺ | potentially long distance vectors |
| communication overhead | flood to all nodes – overhead O(N*E), where N = # of nodes, E = # of edges | send distance vectors only to neighbours – O(N*K) if each of N routers has K neighbours ☺ |
| convergence speed | do NOT need to recalculate LSP's before forwarding ⇒ faster ☺ | takes a while to propagate changes to rest of network |
| space requirements | maintains entire topology in a link database – O(N*K) if each of N routers has K neighbours | maintains only neighbours' states – O(K) distance vectors ☺ |
| computational complexity per one destination | $O(N*(N-1)/2)=O(N^2)$ | O(N*K*Diameter) ☺ |
| computational robustness | each router computes paths on its own – no error propagation ☺ | routers compute paths collectively – errors propagate |
| security / fault tolerance | false/corrupt LSPs can be flooded to all routers | false/corrupt LSPs can be flooded to all routers |