Distance Vector (DV) Algorithm

- **Distributed:** each node receives some information from one or more of its *directly attached* neighbors, performs a calculation, and then distributes the results of its calculation back to its neighbors

- **Iterative:** this process continues on until no more information is exchanged between neighbors.

- **Asynchronous:** it does not require all of the nodes to operate in lockstep with each other.

$d_x(y)$ Cost of the least-cost path from node x to node y

Bellman-Ford Equation

$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

Min over all neighbors

Each node x begins with:

$D_x(y)$ Estimate of the best cost from itself to any node y

Distance Vector from x

$$\boldsymbol{D}_x = [D_x(y): y \text{ in } N]$$

Information at each node:

- For each neighbor v, the cost $c(x,v)$ from x to directly attached neighbor, v
- Node x's distance vector, that is, $\boldsymbol{D}_x = [D_x(y): y \text{ in } N]$, containing x's estimate of its cost to all destinations, y, in N
- The distance vectors of each of its neighbors, that is, $\boldsymbol{D}_v = [D_v(y): y \text{ in } N]$ for each neighbor v of x

If node x receives a new DV from a neighbor (only local exchanges), then update:

$$D_x(y) = \min_v\{c(x,v) + D_v(y)\} \quad \text{for each node } y \text{ in } N$$

If DV changes, then x sends update to neighbors.

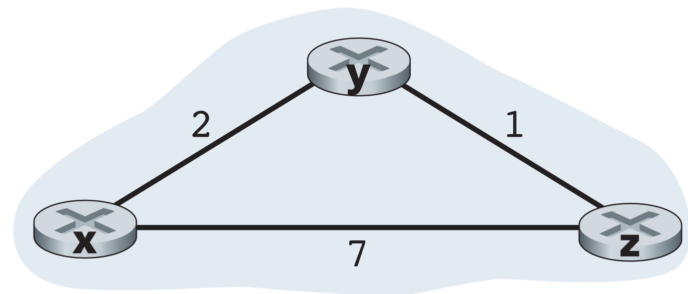
```
Initialization:
   for all destinations y in N:
      D_x(y) = c(x,y)   /* if y is not a neighbor then c(x,y) = ∞ */
   for each neighbor w
      D_w(y) = ? for all destinations y in N
   for each neighbor w
      send distance vector D_x = [D_x(y): y in N] to w

loop
   wait (until I see a link cost change to some neighbor w or
         until I receive a distance vector from some neighbor w)

   for each y in N:
      D_x(y) = min_v{c(x,v) + D_v(y)}

   if D_x(y) changed for any destination y
      send distance vector D_x = [D_x(y): y in N] to all neighbors

forever
```



Node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Node z table

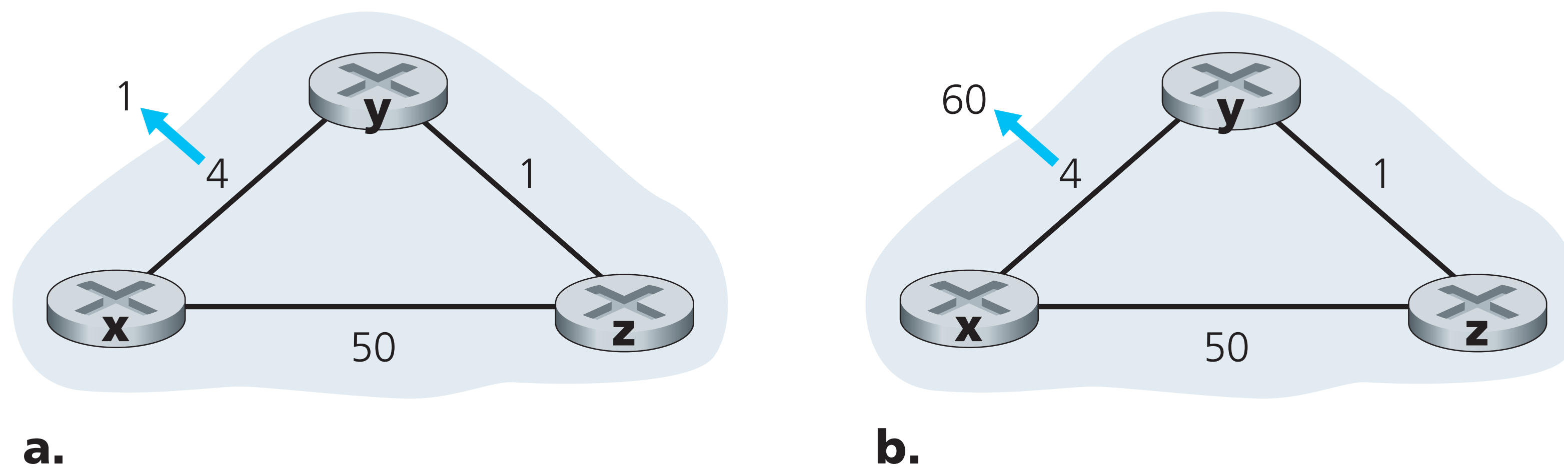
from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Time

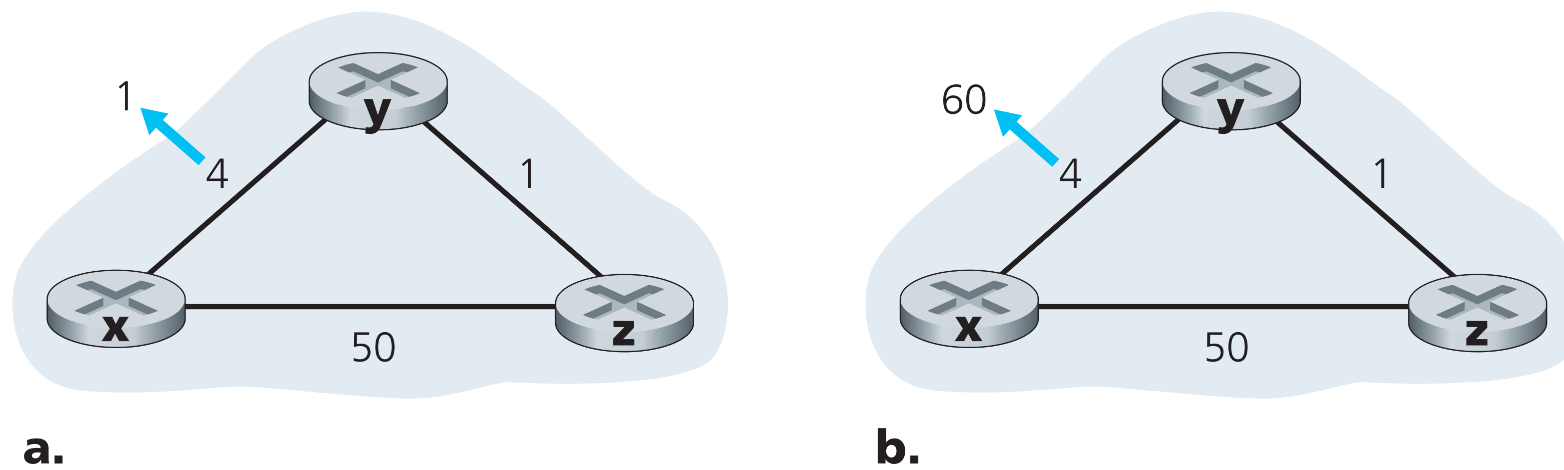
Issues



a) DV is updated efficiently

- At time t_0, y detects the link-cost change (the cost has changed from 4 to 1), updates its distance vector, and informs its neighbors of this change since its distance vector has changed.
- At time t_1, z receives the update from y and updates its table. It computes a new least cost to x (it has decreased from a cost of 5 to a cost of 2) and sends its new distance vector to its neighbors.
- At time t_2, y receives z's update and updates its distance table. y's least costs do not change and hence y does not send any message to z. The algorithm comes to a quiescent state.

Issues



b) Slow Convergence (44 iterations)

1. Before the link cost changes, $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$, and $D_z(x) = 5$. At time t_0, y detects the link-cost change (the cost has changed from 4 to 60). y computes its new minimum-cost path to x to have a cost of

$$D_y(x) = \min\{c(y,x) + D_x(x),\ c(y,z) + D_z(x)\} = \min\{60 + 0,\ 1 + 5\} = 6$$

   Of course, with our global view of the network, we can see that this new cost via z is *wrong*. But the only information node y has is that its direct cost to x is 60 and that z has last told y that z could get to x with a cost of 5. So in order to get to x, y would now route through z, fully expecting that z will be able to get to x with a cost of 5. As of t_1 we have a **routing loop**—in order to get to x, y routes through z, and z routes through y. A routing loop is like a black hole—a packet destined for x arriving at y or z as of t_1 will bounce back and forth between these two nodes forever (or until the forwarding tables are changed).
2. Since node y has computed a new minimum cost to x, it informs z of its new distance vector at time t_1.
3. Sometime after t_1, z receives y's new distance vector, which indicates that y's minimum cost to x is 6. z knows it can get to y with a cost of 1 and hence computes a new least cost to x of $D_z(x) = \min\{50 + 0, 1 + 6\} = 7$. Since z's least cost to x has increased, it then informs y of its new distance vector at t_2.
4. In a similar manner, after receiving z's new distance vector, y determines $D_y(x) = 8$ and sends z its distance vector. z then determines $D_z(x) = 9$ and sends y its distance vector, and so on.

Solution: Poisoned Reverse

If z routes through y to get to x, then z will advertise to y that its distance to x is infinity