Routing Algorithms

- Typically a host is attached directly to one router, the default router for the host (also called the first-hop router for the host).
- Whenever a host sends a packet, the packet is transferred to its default router.
- We refer to the default router of the source host as the source router and the default router of the destination host as the destination router.

Routing Algorithms

- The purpose of a routing algorithm is simple: given a set of routers, with links connecting the routers, a routing algorithm finds a "good" path from source router to destination router.
- Typically, a good path is one that has the least cost

Routing Algorithms

- A graph is used to formulate routing problems.
- The **graph** $G = (N,E)$ is a set N of nodes and a collection E of edges, where each edge is a pair of nodes from N.
- In the context of network-layer routing, the nodes in the graph represent routers - the points at which packet forwarding decisions are made - and the edges connecting these nodes represent the physical links between these routers.
- Such a graph is an abstraction of a computer network.

Routing Algorithms

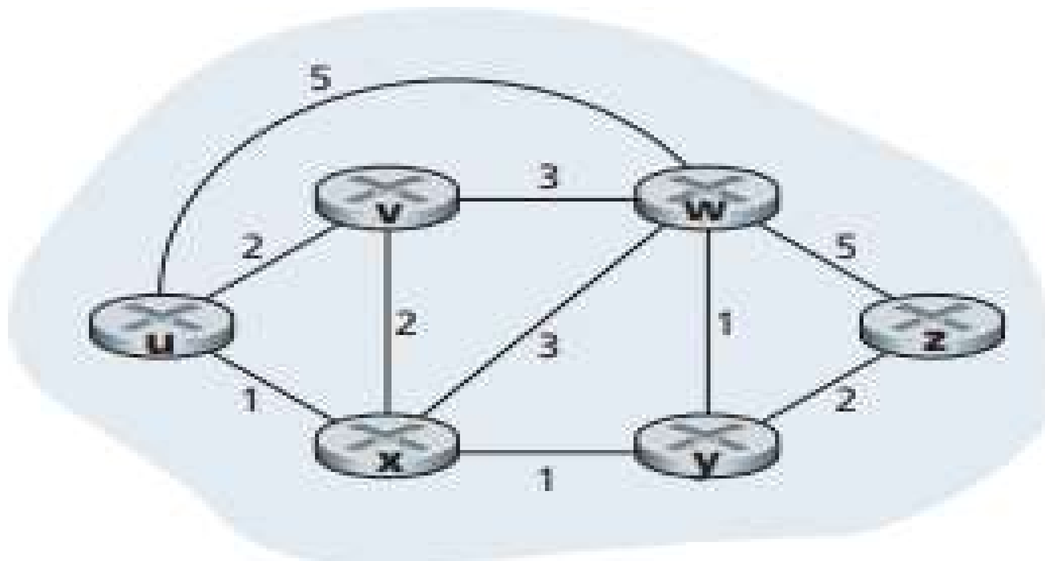


Figure 4.27 • Abstract graph model of a computer network

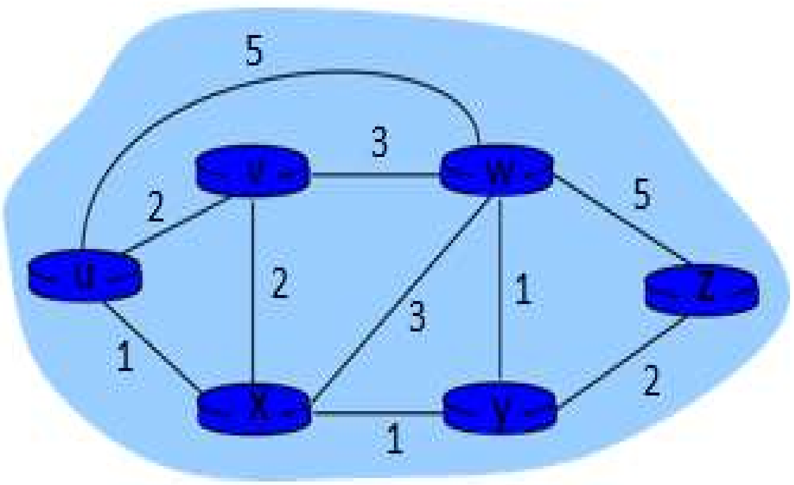
graph: G = (N,E)

N = set of routers = { u, v, w, x, y, z }

E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

[James F Kurose and Keith W Ross, “Computer Networking: A Top - Down Approach”, Pearson Education; 6 th Edition (2017)]

Routing Algorithms



cost of path $(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

key question: what is the least-cost path between u and z?

routing algorithm: algorithm that finds that least cost path

[James F Kurose and Keith W Ross, "Computer Networking: A Top - Down Approach", Pearson Education; 6 th Edition (2017)]

Graph Abstraction

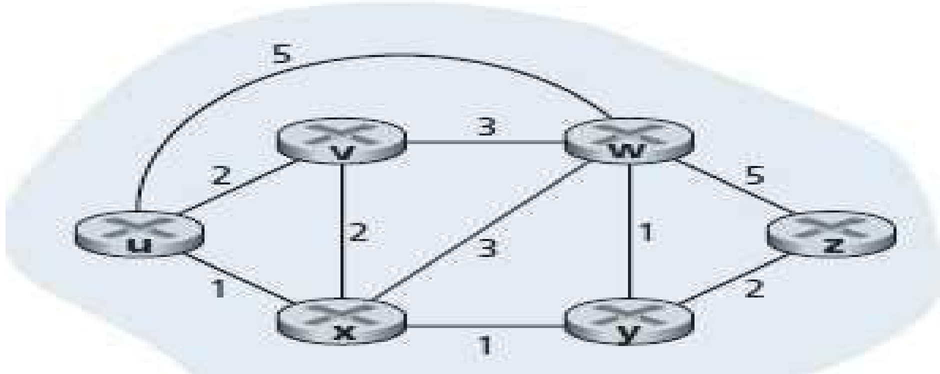
- An edge also has a value representing its cost.
- An edge's cost reflect the physical length of the corresponding link, the link speed, or the monetary cost associated with a link.
- For any edge (x,y) in E, we denote $c(x,y)$ as the cost of the edge between nodes x and y.
- If the pair (x,y) does not belong to E, we set $c(x,y) = \infty$.
- Consider only undirected graphs (i.e., graphs whose edges do not have a direction), so that edge (x,y) is the same as edge (y,x) and that $c(x,y) = c(y,x)$.
- Also, a node y is said to be a neighbor of node x if (x,y) belongs to E.

Graph Abstraction

- A natural goal of a routing algorithm is to identify the least costly paths between sources and destinations.
- To make this problem more precise, recall that a path in a graph $G = (N,E)$ is a sequence of nodes$(x1,x2,\dots,xp)$ such that each of the pairs $(x1,x2),(x2,x3),\dots,(xp\text{-}1,xp)$ are edges in E.
- The cost of a path $(x1,x2,\dots, xp)$ is simply the sum of all the edge costs along the path, that is, $c(x1,x2) + c(x2,x3) + \dots + c(xp\text{-}1,xp)$.

Graph Abstraction

- Given any two nodes *x* and *y*, there are typically many paths between the two nodes, with each path having a cost. One or more of these paths is a least-cost path.
- The least-cost problem is therefore clear: Find a path between the source and destination that has least cost.
- For example, the least-cost path between source node *u* and destination node *w* is (*u*, *x*, *y*, *w*) with a path cost of 3.
- Note that if all edges in the graph have the same cost, the least-cost path is also the shortest path.



[James F Kurose and Keith W Ross, "Computer Networking: A Top - Down Approach", Pearson Education; 6 th Edition (2017)]

Classification of Routing Algorithms

1. Global routing algorithm / Decentralized routing algorithm
2. Static routing algorithms / Dynamic routing algorithms
3. Load-sensitive algorithm / Load-insensitive algorithm

Global routing algorithm

- A **global routing algorithm** computes the least-cost path between a source and destination using complete, global knowledge about the network.
- That is, the algorithm takes the connectivity between all nodes and all link costs as inputs.
- This then requires that the algorithm somehow obtain this information before actually performing the calculation.
- The calculation itself can be run at one site.
- In practice, algorithms with global state information are often referred to as **link-state (LS) algorithms**, since the algorithm must be aware of the cost of each link in the network.

Decentralized routing algorithm

- In a decentralized routing algorithm, the calculation of the least-cost path is carried out in an iterative, distributed manner.
- No node has complete information about the costs of all network links.
- Instead, each node begins with only the knowledge of the costs of its own directly attached links.
- Then, through an iterative process of calculation and exchange of information with its neighboring nodes (that is, nodes that are at the other end of links to which it itself is attached), a node gradually calculates the least-cost path to a destination or set of destinations.
- The decentralized routing algorithm is called a distance-vector (DV) algorithm, because each node maintains a vector of estimates of the costs (distances) to all other nodes in the network.

Dynamic routing algorithms and Static routing algorithms

- In Static routing algorithms, routes change very slowly over time, often as a result of human intervention (for example, a human manually editing a router's forwarding table).

- Dynamic routing algorithms change the routing paths as the network traffic loads or topology change.

- A dynamic algorithm can be run either periodically or in direct response to topology or link cost changes.

- While dynamic algorithms are more responsive to network changes, they are also more susceptible to problems such as routing loops and oscillation in routes.

Load-sensitive algorithm / Load-insensitive algorithm

- In a load-sensitive algorithm, link costs vary dynamically to reflect the current level of congestion in the underlying link.
- If a high cost is associated with a link that is currently congested, a routing algorithm will tend to choose routes around such a congested link.
- Today's Internet routing algorithms (such as RIP, OSPF, and BGP) are load-insensitive, as a link's cost does not explicitly reflect its current (or recent past) level of congestion.

The Link-State (LS) Routing Algorithm

- The link-state routing algorithm is known as Dijkstra's algorithm, named after its inventor.
- A closely related algorithm is Prim's algorithm;
- Dijkstra's algorithm computes the least-cost path from one node (the source, which we will refer to as u) to all other nodes in the network.
- Dijkstra's algorithm is iterative and has the property that after the k^{th} iteration of the algorithm, the least-cost paths are known to k destination nodes, and among the least-cost paths to all destination nodes, these k paths will have the k smallest costs.

The Link-State (LS) Routing Algorithm

Let us define the following notation:

- D(v): cost of the least-cost path from the source node to destination v as of this iteration of the algorithm.
- p(v): previous node (neighbor of v) along the current least-cost path from the source to v.
- N′ : subset of nodes; v is in N′ if the least-cost path from the source to v is definitively know

The Link-State (LS) Routing Algorithm

Link-State (LS) Algorithm for Source Node *u*

[James F Kurose and Keith W Ross, “Computer Networking: A Top - Down Approach”, Pearson Education; 6 th Edition (2017)]

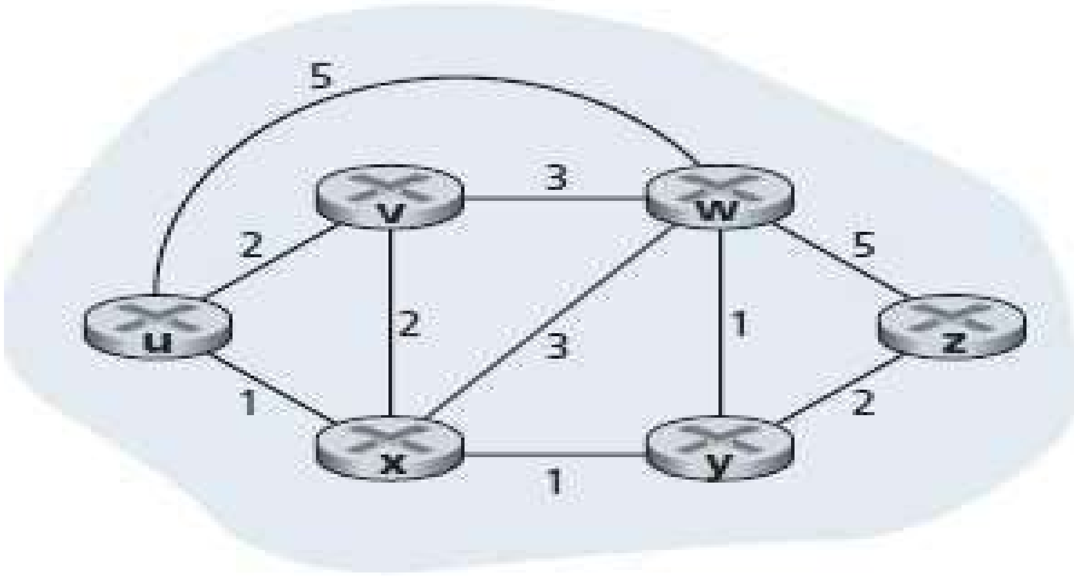
```
Initialization:
  N' = {u}
  for all nodes v
    if v is a neighbor of u
      then D(v) = c(u,v)
    else D(v) = ∞

Loop
  find w not in N' such that D(w) is a minimum
  add w to N'
  update D(v) for each neighbor v of w and not in N':
       D(v) = min( D(v), D(w) + c(w,v) )
  /* new cost to v is either old cost to v or known
   least path cost to w plus cost from w to v */
until N'= N
```

- The number of times the loop is executed is equal to the number of nodes in the network.
- The algorithm will have calculated the shortest paths from the source node u to every other node in the network.

The Link-State (LS) Routing Algorithm

step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

Table 4.3 ♦ Running the link-state algorithm on the network



[James F Kurose and Keith W Ross, "Computer Networking: A Top - Down Approach", Pearson Education; 6 th Edition (2017)]

The Link-State (LS) Routing Algorithm

- In the initialization step, the currently known least-cost paths from u to its directly attached neighbors, v, x, and w, are initialized to 2, 1, and 5, respectively.
- Note in particular that the cost to w is set to 5 (even though we will soon see that a lesser-cost path does indeed exist) since this is the cost of the direct (one hop) link from u to w.
- The costs to y and z are set to infinity because they are not directly connected to u.

The Link-State (LS) Routing Algorithm

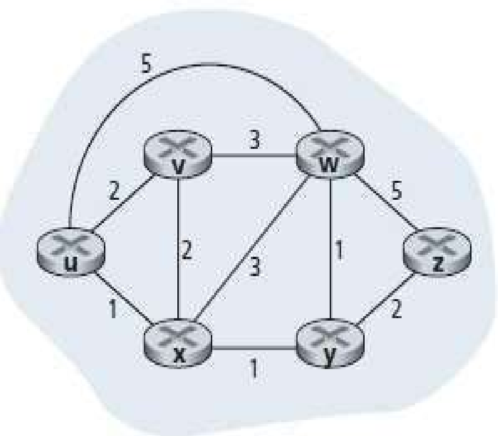
- In the first iteration, we look among those nodes not yet added to the set N′ and find that node with the least cost as of the end of the previous iteration.
- That node is x, with a cost of 1, and thus x is added to the set N′.
- Line 12 of the LS algorithm is then performed to update D(v) for all nodes v, yielding the results shown in the second line (Step 1) in Table 4.3.
- The cost of the path to v is unchanged.
- The cost of the path to w (which was 5 at the end of the initialization) through node x is found to have a cost of 4. Hence this lower-cost path is selected and w's predecessor along the shortest path from u is set to x.
- Similarly, the cost to y (through x) is computed to be 2, and the table is updated accordingly.

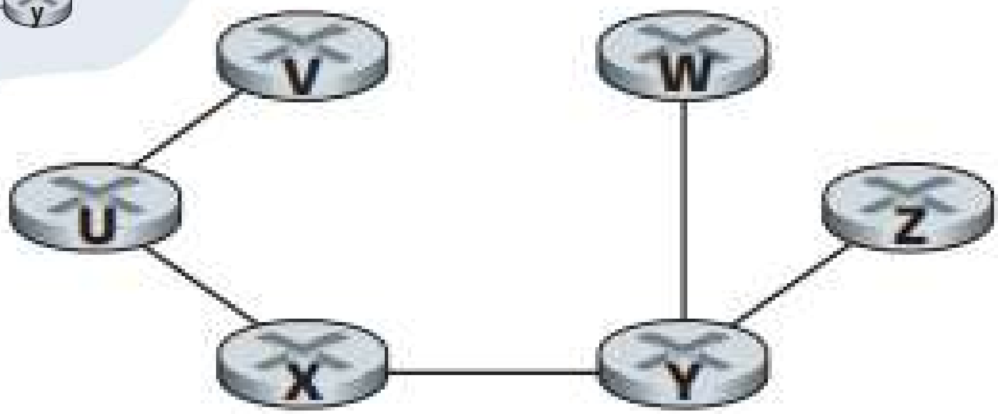
The Link-State (LS) Routing Algorithm

- In the second iteration, nodes v and y are found to have the least-cost paths (2), and we break the tie arbitrarily and add y to the set N so that N now contains u, x, and y.
- The cost to the remaining nodes not yet in N', that is, nodes v, w, and z, are updated via line 12 of the LS algorithm, yielding the results.

The algorithm calculated the shortest paths from the source node u to every other node in the network.

The Link-State (LS) Routing Algorithm





Destination	Link
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

Figure 4.28 ♦ Least cost path and forwarding table for nodule u

[James F Kurose and Keith W Ross, "Computer Networking: A Top - Down Approach", Pearson Education; 6 th Edition (2017)]

The algorithm calculated the shortest paths from the source node u to every other node in the network.

The Link-State (LS) Routing Algorithm

- What is the computational complexity of this algorithm? That is, given n nodes (not counting the source), how much computation must be done in the worst case to find the least-cost paths from the source to all destinations?

The preceding implementation of the LS algorithm has worst-case complexity of order n squared: $O(n^2)$.

The Link-State (LS) Routing Algorithm

- In the first iteration, we need to search through all n nodes to determine the node, w, not in N′ that has the minimum cost.
- In the second iteration, we need to check $n - 1$ nodes to determine the minimum cost; in the third iteration $n - 2$ nodes, and so on.
- Overall, the total number of nodes we need to search through over all the iterations is $n(n + 1)/2$.

The preceding implementation of the LS algorithm has worst-case complexity of order n squared: $O(n^2)$.

The Distance-Vector (DV) Routing Algorithm

In distance vector routing, each node shares its routing table with its immediate neighbors periodically and when there is a change.

The Distance-Vector (DV) Routing Algorithm

- LS algorithm is an algorithm using global information, the Distance Vector (DV) algorithm is iterative, asynchronous, and distributed.
- It is distributed in that 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.
- It is iterative in that this process continues on until no more information is exchanged between neighbors.
- The algorithm is also self-terminating—there is no signal that the computation should stop; it just stops.

The Distance-Vector (DV) Routing Algorithm

- The algorithm is asynchronous in that it does not require all of the nodes to operate in lockstep with each other.
- An asynchronous, iterative, self-terminating, distributed algorithm is much more interesting and fun than a centralized algorithm!

Distance Vector Algorithm

Bellman-Ford Equation (dynamic programming)

Define

$d_x(y)$:= cost of least-cost path from x to y

Then

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

where $\min_v$ is taken over all neighbors of x.

- Intuition: After traveling from x to v, if we then take the least-cost path from v to y, the path cost will be $c(x,v) + d_v(y)$.
- Since we must begin by traveling to some neighbor v, the least cost from x to y is the minimum of $c(x,v) + d_v(y)$ taken over all neighbors v

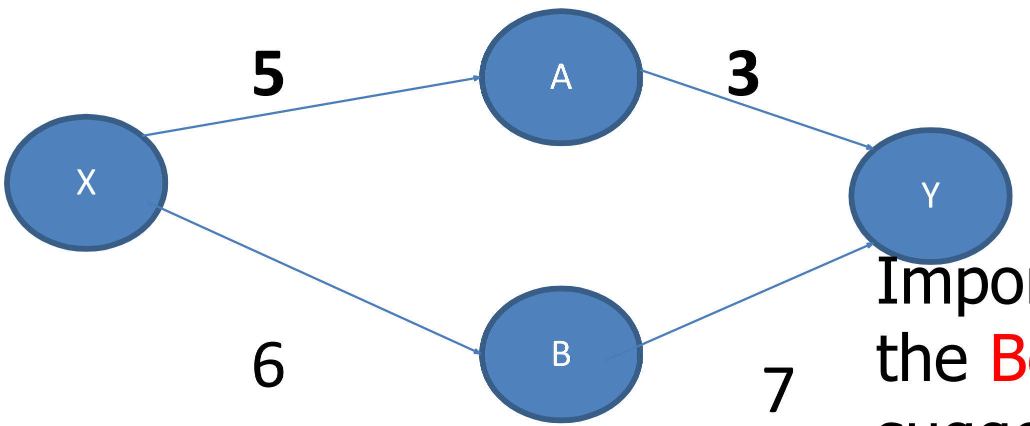
Bellman-Ford example

Clearly, $d_A(Y) = 3$, $d_B(Y) = 7$

B-F equation says:

$$d_X(Y) = \min \{ c(X,A) + d_A(Y), \; c(X,B) + d_B(Y)\}$$

$$= \min \{5 + 3, 6+7\} = 8$$



Important practical contribution of the Bellman-Ford equation is that it suggests the form of the neighbor-to-neighbor communication that will take place in the DV algorithm

Distance Vector Algorithm

With the DV algorithm, each node X maintains the following routing data.

1. For each neighbor v, the cost c(x,v) from x to directly attached neighbor v.

2. Node x's distance vector. Distance vector: $\mathbf{D}_x = [D_x(y): y \in N]$
   containing x's estimate of its cost to all destinations y in N.

3. The distance vectors of each of its neighbors .

$$\mathbf{D}_v = [D_v(y): y \in N]$$

for each neighbor v of x.

Distance vector algorithm

Basic idea:

- Each node periodically sends its own distance vector estimate to neighbors
- When a node x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

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

Distance vector algorithm

Iterative each local iteration caused by:

- local link cost change
- DV update message from neighbor.
- Continue on until no more info is exchanged between neighbors.

Distributed:

- Each node receives some info from one or more of its directly attached neighbors.
- Performs a calculation and then distributes the results of its calculation back to its neighbors.
- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

Self Terminating:

Asynchronous: it doesn't require all of the nodes to operate in lockstep with each other

Each node:

wait for (change in local link cost of msg from neighbor)

↓

recompute estimates

↓

if DV to any destination has changed, *notify* neighbors

Distance vector algorithm

Working:

- Each node x begins with $D_x(y)$, an estimate of the cost of the least-cost path from itself to node y, for all nodes in N.
- $D_x = [D_x(y): y \text{ in } N]$ be node x's **distance vector**, which is the vector of cost estimates from x to all other nodes, y in N.
- Each node x maintains the following routing information:

  • For each neighbor v, the cost c(x,v) from x to directly attached neighbor, v

  • Node x's distance vector, that is, $D_x = [D_x(y): y \text{ in } N]$, containing x's estimate of its cost to all destinations, y in N

  • The distance vectors of each of its neighbors, that is, $D_v = [D_v(y): y \text{ in } N]$ for each neighbor v of x.

Distance vector algorithm

- From time to time, each node sends a copy of its distance vector to each of its neighbors.
- When a node x receives a new distance vector from any of its neighbors v, it saves v's distance vector, and then uses the Bellman-Ford equation to update its own distance vector as follows:

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

- If node x's distance vector has changed as a result of this update step, node x will then send its updated distance vector to each of its neighbors, which can in turn update their own distance vectors.

Distance vector algorithm

$D_x(x)=0$

$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\}$
$= \min\{2+0\ ,\ 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\}$
$= \min\{2+1\ ,\ 7+0\} = 3$

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

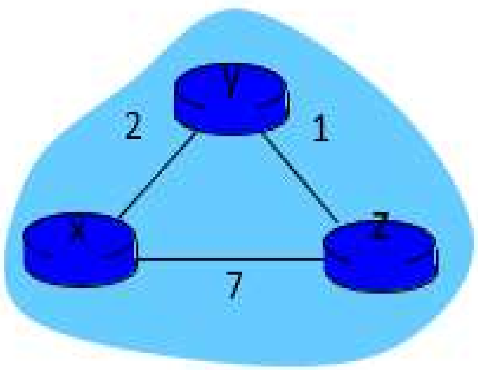
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

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



Distance vector algorithm

- At initialization node x has not received anything from node y or z, the entries in the second and third rows are initialized to infinity

$$D_z(x) = \min(c(z,x) + D_x(x),\ c(z,y) + D_y(x))$$

$$= \min(7+0,\ 1+2) = 3$$

$$D_y(x) = \min(c(y,x) + D_x(x),\ c(y,z) + D_z(x))$$

$$= \min(2+0,\ 1+3) = 2$$

Distance vector algorithm

Distance-Vector (DV) Algorithm

At each node, x:

```
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
```