Link-State and Distance Vector Routing Examples

CPSC 441

University of Calgary

Link-State (LS) Routing Algorithm

Dijkstra's algorithm

- topology and link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node (source) to all other nodes
  - gives forwarding table for that node
- iterative: after k iterations, know least cost path to k destination nodes

Notation:

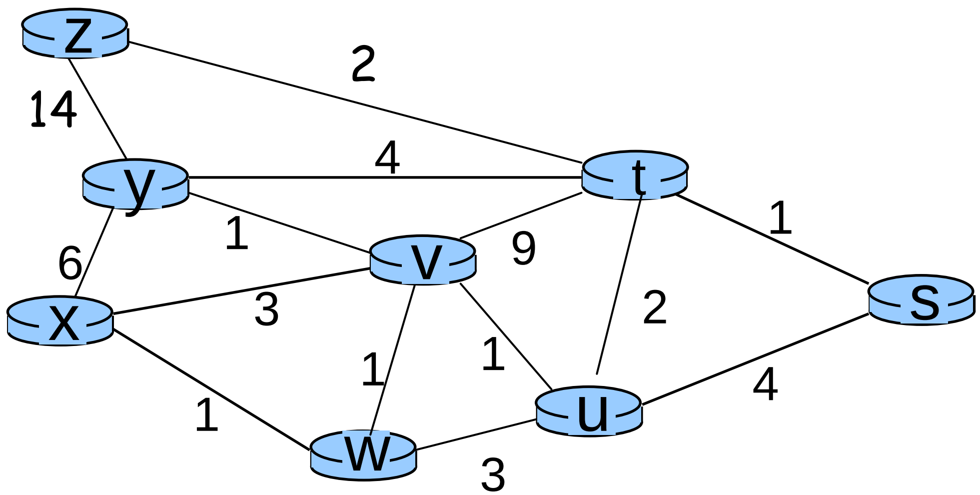
- c(x,y): link cost from node x to y; set to ∞ if a and y are not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): v's predecessor node along path from source to v
- N': set of nodes whose least cost path is definitively known

Dijsktra's Algorithm

```
Initialization (u = source node):
  N' = {u}                    /* path to self is all we know */
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)   /* assign link cost to neighbours */
    else D(v) = ∞

 Loop
    find w not in N' such that D(w) is a minimum
    add w to N'
    update D(v) for all v adjacent to 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
     shortest path cost to w plus cost from w to v */
 until all nodes in N'
```

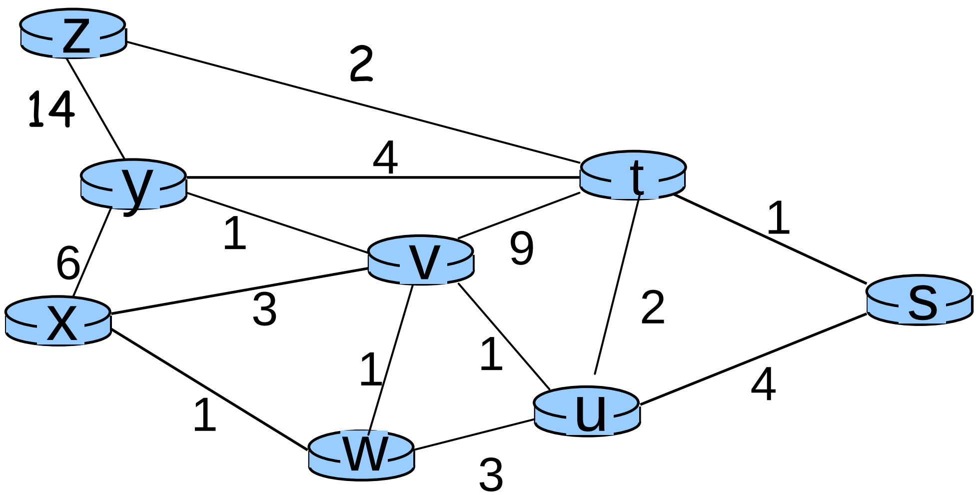
Textbook – Problem 4.21 – x is source



Step	N’	D(s),p(s)	D(t),p(t)	D(u),p(u)	D(v),p(v)	D(w),p(w)	D(y),p(y)	D(z),p(z)
0	x	∞	∞	∞	3,x	1,x	6,x	∞

Initialization:
- Store source node x in N’
- Assign link cost to neighbours (v,w,y)
- Keep track of predecessor to destination node

Textbook – Problem 4.21 – x is source



Node and its minimum cost are colour-coded in each step

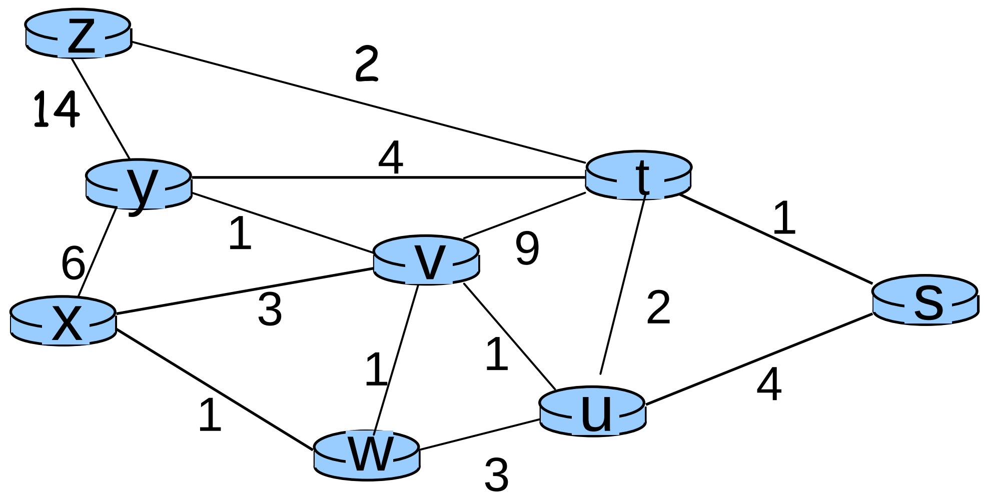
Step	*N'*	*D(s),p(s)*	*D(t),p(t)*	*D(u),p(u)*	*D(v),p(v)*	*D(w),p(w)*	*D(y),p(y)*	*D(z),p(z)*
0	**x**	∞	∞	∞	**3,x**	**1,x**	**6,x**	∞
1	**xw**	∞	∞	**4,w**	**2,w**		**6,x**	∞

Loop – step 1:

- For all nodes not in N', find one that has minimum cost path (1)
- Add this node (w) to N'
- Update cost for all neighbours of added node that are not in N'

repeat until all nodes are in N'

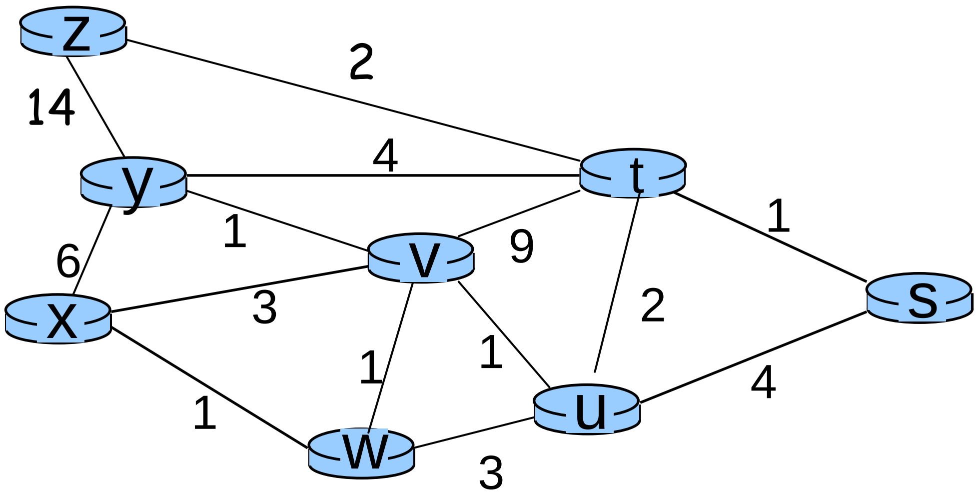
Textbook – Problem 4.21 – x is source



Node and its minimum cost are colour-coded in each step

Step	*N'*	*D(s),p(s)*	*D(t),p(t)*	*D(u),p(u)*	*D(v),p(v)*	*D(w),p(w)*	*D(y),p(y)*	*D(z),p(z)*
0	x	∞	∞	∞	3,x	1,x	6,x	∞
1	xw	∞	∞	4,w	2,w		6,x	∞
2	xwv	∞	11,v	3,v			3,v	∞
3	xwvu	7,u	5,u				3,v	∞
4	xwvuy	7,u	5,u					17,y
5	xwvuyt	6,t						7,t
6	xwvuyts							7,t

Textbook – Problem 4.21 – x is source



We can now build x's forwarding table. E.g. the entry to s will be constructed by looking at predecessors along shortest path: 6,t → 5,u →3,v → 2,w (direct link) So forward to s via w

Step	*N'*	*D(s),p(s)*	*D(t),p(t)*	*D(u),p(u)*	*D(v),p(v)*	*D(w),p(w)*	*D(y),p(y)*	*D(z),p(z)*
0	**x**	∞	∞	∞	**3,x**	**1,x**	**6,x**	∞
1	**xw**	∞	∞	**4,w**	**2,w**		**6,x**	∞
2	**xwv**	∞	**11,v**	**3,v**			**3,v**	∞
3	**xwvu**	**7,u**	**5,u**				**3,v**	∞
4	**xwvuy**	**7,u**	**5,u**					**17,y**
5	**xwvuyt**	**6,t**						**7,t**
6	**xwvuyts**							**7,t**

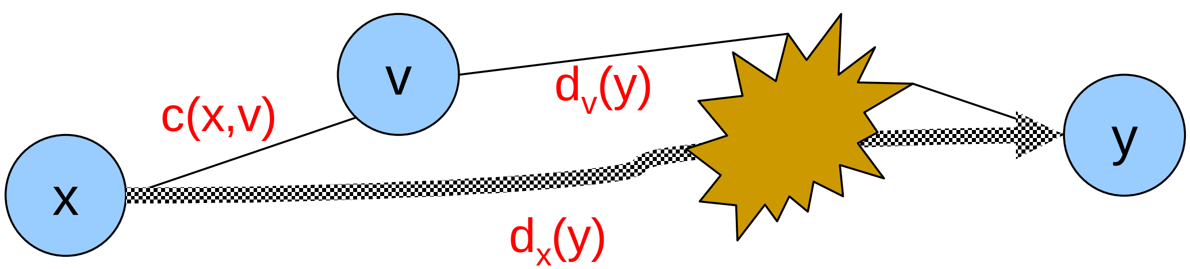
Distance Vector Routing

- Based on Bellman-Ford Equation
- Define

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

  c(x,v) := cost of direct link from x to v
- Then, for all v that are neighbours of x

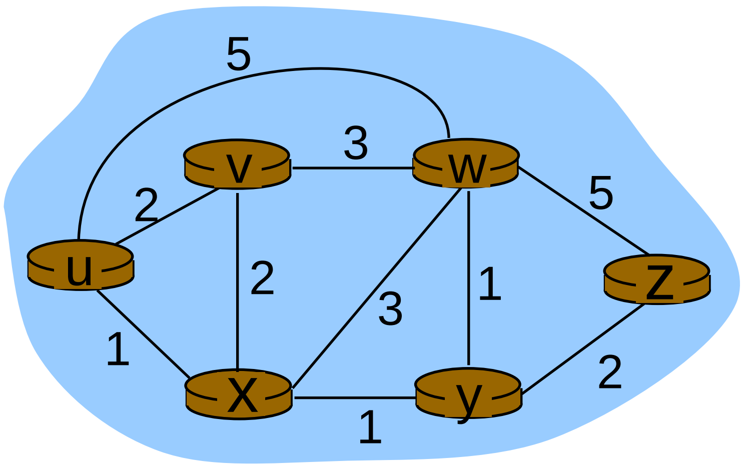
  $d_x(y) = \min_v \{c(x,v) + d_v(y)\}$



Bellman-Ford Equation Example

Consider a path from *u* to *z*

By inspection, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$



B-F equation says:

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

Node that achieves minimum is next hop in shortest path ➜ entry in forwarding table

Distance Vector Algorithm

Basic idea:

- Nodes keep vector (DV) of least costs to other nodes
  - These are estimates, $D_x(y)$
- Each node periodically sends its own DV to neighbors
- When node *x* receives DV from neighbor, it keeps it and updates its own DV using B-F:

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

- Ideally, the estimate $D_x(y)$ *converges to the actual least cost* $d_x(y)$

On each node:

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

↓

recompute estimates

↓

if DV has changed, *notify* neighbors

node x table

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

node y table

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

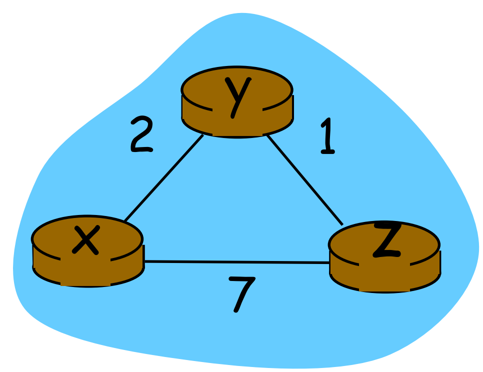
node z table

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

Step 1: Initialization

Initialize costs of direct links

Set to ∞ costs from neighbours



time

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

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

node x table

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

cost to

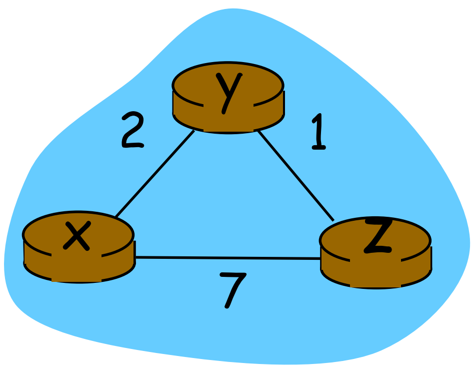
from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

Step 2: Exchange DV and iterate

-In first iteration, node x saves neighbours' DVs

-Then, it checks path costs to all nodes using received DVs

-E.g. new cost $D_x(z)$ is obtained by adding costs marked red



time

In similar fashion, algorithm proceeds until all nodes have updated tables

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

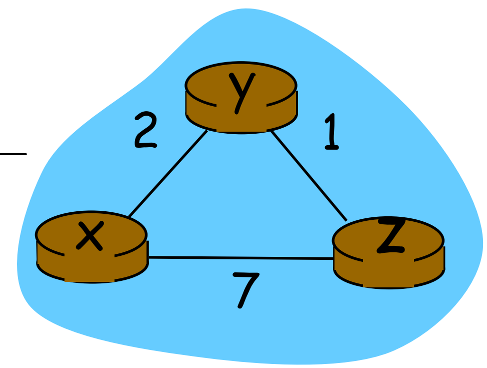
node z table

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

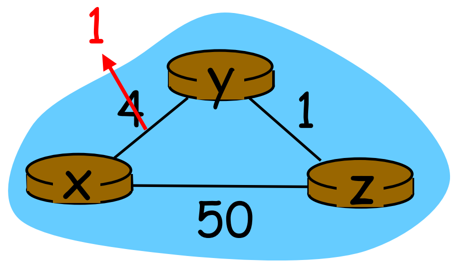
time



Distance Vector: link cost changes

Link cost changes:

- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors



"good news travels fast"

At time t_0, *y* detects the link-cost change, updates its DV, and informs its neighbors.

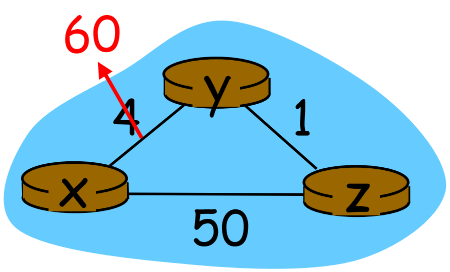
At time t_1, *z* receives the update from *y* and updates its table. It computes a new least cost to *x* and sends neighbors its DV.

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*.

Distance Vector: link cost changes

Link cost changes:

- bad news travels slow - “count to infinity” problem!
- When y detects cost change to 60, it will update its DV using the z’s cost to x, which is 5 (via y), to obtain an incorrect new cost to x of 6, over the path y→z→y→x that has a loop
- 44 iterations before algorithm stabilizes, while y and z exchange updates



Poisoned reverse:

- If Z routes through Y to get to X :
  - Z tells Y its (Z’s) distance to X is infinite (so Y won’t route to X via Z)
- Will this completely solve the problem?