CS5222 Computer Networks and Internets Tutorial 10 (Week 11)

Prof Weifa Liang

Weifa.liang@cityu-dg.edu.cn

Slides based on book *Computer Networking: A Top-Down Approach.*

Revision

Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

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

- under natural conditions, the estimate $D_x(y)$ converges to the actual least cost $d_x(y)$

Distance vector algorithm

Based on *Bellman-Ford* (BF) equation:

Bellman-Ford equation

Let $D_x(y)$: cost of least-cost path from x to y.

Then:

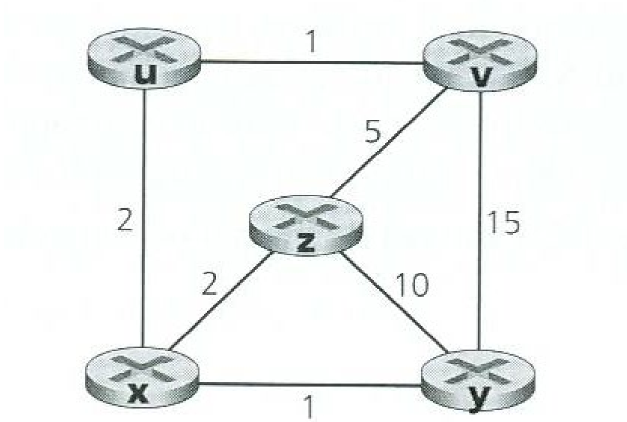
$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

v's estimated least-cost-path cost to y

cost of link from x to v

min taken over all neighbors v of x

1. Consider the distance vector algorithm and show the distance vector entries at node z at each step of the algorithm.



Initialization:

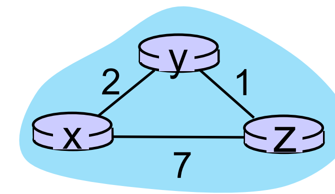
	u	v	x	y	z
D_u:	∞	∞	∞	∞	∞
D_v:	∞	∞	∞	∞	∞
D_x:	∞	∞	∞	∞	∞
D_y:	∞	∞	∞	∞	∞
D_z:	∞	5	2	10	0

	u	v	x	y	z
u	0	1	2	∞	∞
v	1	0	∞	15	5
x	2	∞	0	1	2
y	∞	15	1	0	10
z	4	5	2	3	0

	u	v	x	y	z
u	0	1	2	3	4
v	1	0	3	15	5
x	2	3	0	1	2
y	3	15	1	0	3
z	4	5	2	3	0

	u	v	x	y	z
u	0	1	2	3	4
v	1	0	3	4	5
x	2	3	0	1	2
y	3	4	1	0	3
z	4	5	2	3	0

2. Consider the distance vector algorithm for the network below and show the distance vector entries of each node at each step of the algorithm.



Node x

cost to	x	y	z
D_x	0	2	7
D_y	∞	∞	∞
D_z	∞	∞	∞

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

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

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

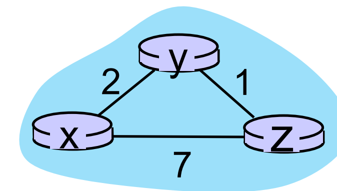
Node y

cost to	x	y	z
D_x	∞	∞	∞
D_y	2	0	1
D_z	∞	∞	∞

Node z

cost to	x	y	z
D_x	∞	∞	∞
D_y	∞	∞	∞
D_z	7	1	0

time



Node x

	cost to x	y	z
D_x	0	2	7
D_y	∞	∞	∞
D_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

	cost to x	y	z
D_x	∞	∞	∞
D_y	2	0	1
D_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

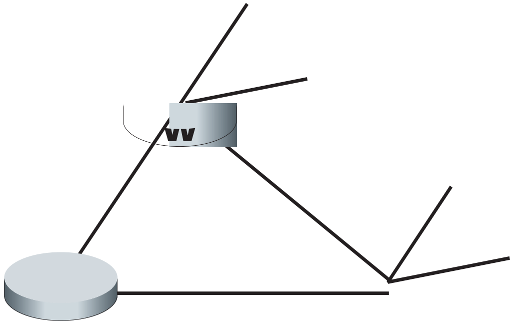
	cost to x	y	z
D_x	∞	∞	∞
D_y	∞	∞	∞
D_z	7	1	0

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

3. Consider the network fragment shown below. x has only two attached neighbors, w and y. w has a minimum-cost path to destination u (not shown) of 5, and y has a minimum-cost path to u of 6. The complete paths from w and y to u (and between w and y) are not shown. All link costs in the network have strictly positive integer values. Give x's distance vector for destinations w, y, and u.



Answer:

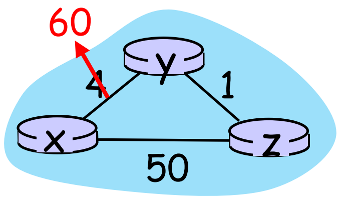
Given $D_w(u) = 5$, $D_y(u) = 6$,

$D_x(w) = \min\{c_{x,w} + D_w(w), c_{x,y} + D_y(w)\} = \min\{2 + 0, 5 + 2\} = 2$

$D_x(y) = \min\{c_{x,w} + D_w(y), c_{x,y} + D_y(y)\} = \min\{2 + 2, 5 + 0\} = 4$

$D_x(u) = \min\{c_{x,w} + D_w(u), c_{x,y} + D_y(u)\} = \min\{2+5, 5+6\} = 7$

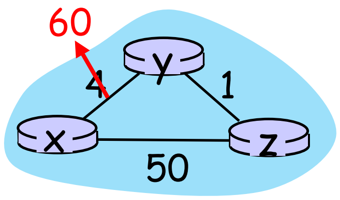
4. **[Difficult]** Consider the network below. Suppose the link cost between x and y increases to 60. Recall that, in the lecture, we discussed that it will take a long time until the distance vector algorithm finally stabilizes (≥40 iterations) due to the **count-to-infinity problem**. Describe a modification of the distance vector algorithm that correctly computes the distance vectors and avoids the count-to-infinity problem in this specific scenario.



Recall: The Problematic Scenario

link cost changes:

- node detects local link cost change
- "bad news travels slow" – **count-to-infinity problem**:

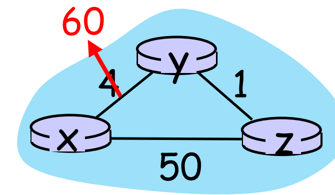


- *y* sees its adjacent link to *x* with new cost 60, but z has said it has a path to x at cost of 5. So *y* computes "my new cost to x will be 6, via z"; notifies *z* that it has a new cost of 6 to *x.*
- *z* learns that a path to *x* via *y* has new cost 6, so *z* computes "my new cost to *x* will be 7 via y", notifies *y* of its new cost of 7 to *x.*
- *y* learns that a path to *x* via *z* has new cost 7, so *y* computes "my new cost to *x* will be 8 via y", notifies *z* of new cost of 8 to *x.*
- *z* learns that path to *x* via *y* has new cost 8, so *z* computes "my new cost to *x* will be 9 via y", notifies *y* of new cost of 9 to *x.*

  ...

  ➔ *Takes a long time to stabilize!*

Answer:



The technique to avoid the problem is called **poison reverse**:

Key Idea:

- If z routes through y to get to destination x, then z will "lie" to y that its cost to x is infinity, i.e., $D_z(x) = \infty$ (even though z knows that $D_z(x) = 5$.)
- Since y believes that z has no path to x, it follows that y will never attempt to route to x via z, as long as z continues to route to x via y.

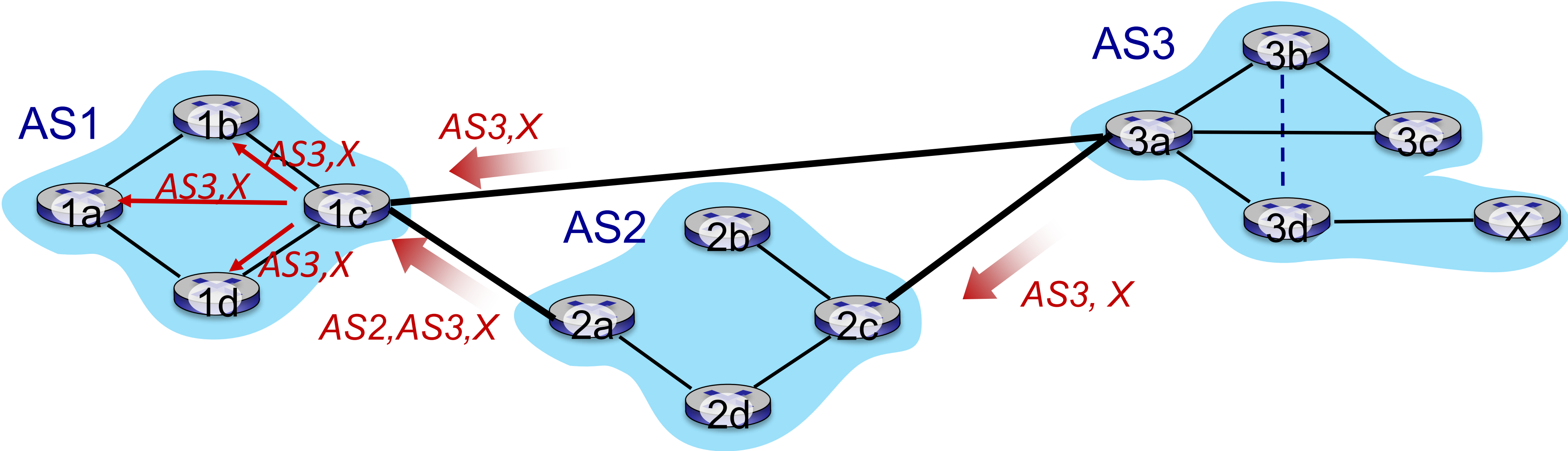
How does it solve our problem?

- Suppose z has advertised $D_z(x) = \infty$ to y.
- Time t0:
  - cost of (x,y) increases to 60.
  - y updates its vector and continues to route *directly* to x, although now at a higher cost of 60, i.e. $D_y(x) = 60$.
  - y informs z about its new DV.
- Time t1:
  - z receives y's update. z updates its own DV and now uses the *direct* link to x, i.e. $D_z(x) = 50$.
  - z informs y about its new DV. **Note that z sends $D_z(x) = 50$ and no longer $D_z(x) = \infty$!**
- Time t2:
  - y receives z's update. y computes $D_y(x) = 51$. Now y will "lie" to z that $\mathbf{D_y(x) = \infty}$ because it routes through z. The algorithm stabilizes.

5. Describe how loops in paths can be detected in BGP.

Answer:

Since full AS path information is available from an AS to a destination in BGP, loop detection is simple – if a BGP peer receives a route that contains its own AS number in the AS path, then the use of that route would result in a loop.



- 6. Though we have link state and distance vector routing algorithms for least cost paths finding, the end-to-end path in today's Internet may not be the path with the best end-to-end performance. What are the possible reasons?

Answer:

1. Intra-domain routing protocols (i.e., iBGP) such as RIP and OSPF use the number of hops based or bandwidth-based cost metric, which does not necessarily correspond the end-to-end performance.

2. Inter-domain routing protocols (i.e., eBGP) are **policy driven:**

  – For example, an ISP may route traffic along a less efficiency path to avoid paying an upper-tier ISP.
  – Another example is that an ISP may not allow "transit" traffic to be routed through its network to avoid additional traffic overhead for which it does not generate any revenue.