COSC264

Introduction to Computer Networks and the Internet

Introduction to Routing – Distance Vector Algorithm

Dr. Barry Wu

Wireless Research Centre

University of Canterbury

barry.wu@canterbury.ac.nz

Outline – this week

- Network layer overview
- Routing overview
- Link-state routing (Dijkstra's algorithm)
- Distance-vector routing (Bellman-Ford)
- Summary

Outline – this week

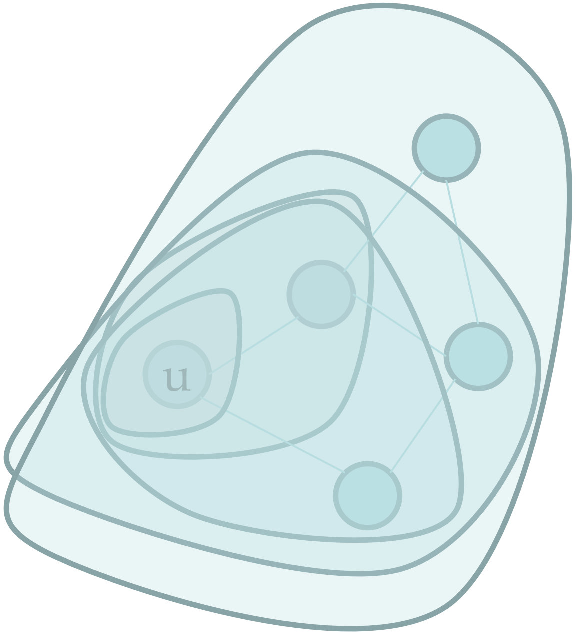
- Network layer overview
- Routing overview
- Link-state routing (Dijkstra's algorithm)
- Distance-vector routing (Bellman-Ford)
- Summary

Review: Dijsktra's Algorithm

```
Initialization:
  S = {u} /* u is the source*/
  for all nodes v
    if v adjacent to u {
        then D(v) = c(u,v) /* cost of neighbor known*/
        else D(v) = ∞ /* cost of others unknown*/

Loop
  find w not in S with the smallest D(w)
  add w to S
  update D(v) for all v adjacent to w and not in S:
      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 S
```

An illustration



Outline – today

- Network layer overview
- Routing overview
- Link-state routing (Dijkstra's algorithm)
- Distance-vector routing (Bellman-Ford)
- Summary

Distance Vector Algorithm

- Distributed
- Iterative
- Asynchronous

Bellman-Ford Equation

Define

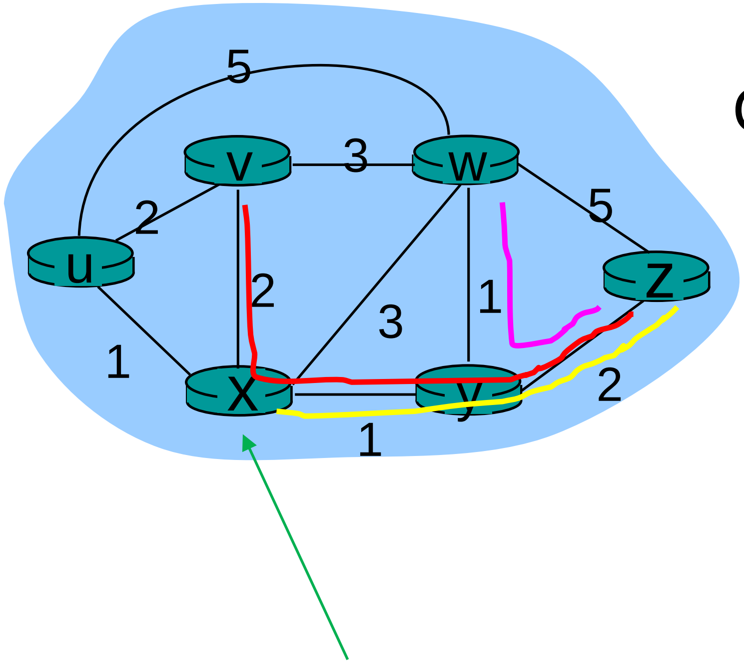
$d_x(y)$:= cost of least-cost path from x to y

Then

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

where min is taken over all neighbors of x

Bellman-Ford example



Node that achieves minimum is next hop in shortest path.

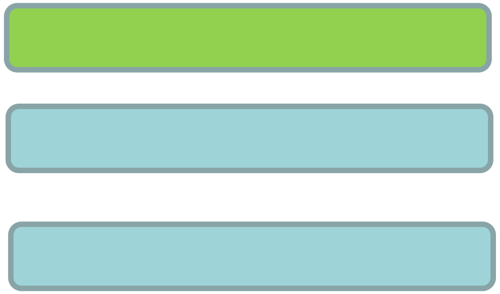
Clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

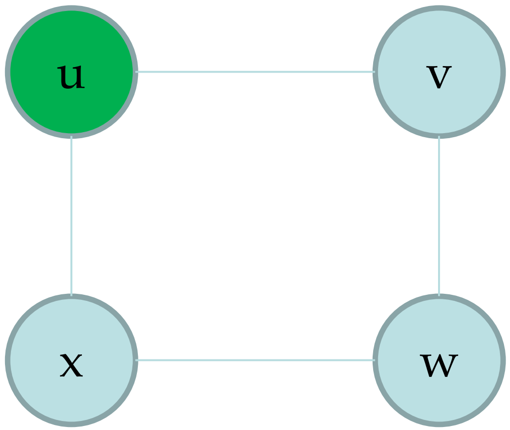
$$
\begin{aligned}
d_u(z) = \min \{ & c(u,v) + d_v(z), \\
& c(u,x) + d_x(z), \\
& c(u,w) + d_w(z) \} \\
= \min \{ & 2 + 5, \\
& 1 + 3, \\
& 5 + 3\} = 4
\end{aligned}
$$

Distance Vector Algorithm

- Estimates:
  - $D_x(y)$ = estimate of least cost from x to y
  - Distance vector: $\mathbf{D}_x = [D_x(y): y \in N]$
- Each node x:
  - Node x knows cost to each neighbor v: $c(x,v)$
  - Node x maintains $\mathbf{D}_x = [D_x(y): y \in N]$
  - Node x also maintains its neighbors' distance vectors
    - For each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$



An illustration



Distance vectors at node u

D_u

D_v

D_x

Distance vector algorithm

<u>Basic idea:</u>

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

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

- Amazingly, as long as all the nodes continue to exchange their distance vectors in an asynchronous fashion, the estimate $D_x(y)$ *converges the actual least cost* $d_x(y)$

Distance Vector Algorithm

Iterative, asynchronous:
each local iteration caused by:

- local link cost change
- DV update message from neighbour

Distributed:

- each node notifies neighbours *only* when its DV changes
  - neighbours then notify their neighbours if necessary
  - The algorithm doesn't know the entire path – only knows the next hop

Each node:

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

recompute estimates

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

Distance Vector Algorithm

At each node, x:

Initialization:

    for all destinations y in N:

        $D_x(y) = c(x,y)$ /* $c(x,y) = \infty$ *if y is not a neighbour*/*

    for each neighbour w

        $D_w(y) = \infty$ 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 neighbour w)

    for each y in N:

        $D_x(y) = min_v\{c(x,v) + D_v(y)\}$ /*v is adjacent to x*/

    if $D_x(y)$ changed for any destination y

        send distance vector $D_x = [D_x(y)$: y in N] to all neighbours

forever

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

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

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

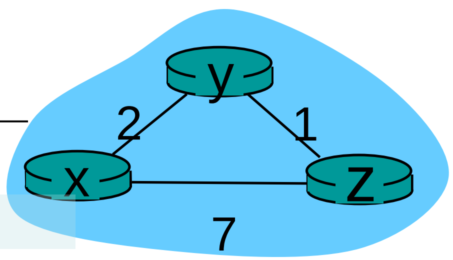
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

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

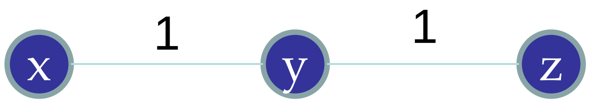
from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

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

time

A hidden assumption – N (all destinations)

- *Q: if all nodes exchange distance vectors with their neighbours only, can each of them know all the destination nodes (N, in the pseudocode)?*



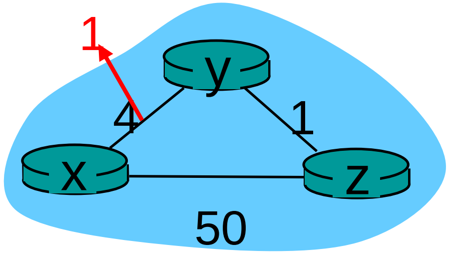
- Initially, x knows it has a path to y with cost 1; *but it does know z*;
- y knows it has a path to x (and z) with cost 1;
- z knows it has a path to y with cost 1; *but it does know x*;
- Then, y and x exchange distance vectors;
- x learned that there is *a new destination z* and it can reach z via y with cost 2;
- y and z exchange distance vectors;
- z learned that there is *a new destination x* and it can reach x via y with cost 2;

- Now both x and z know their destination nodes!

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"

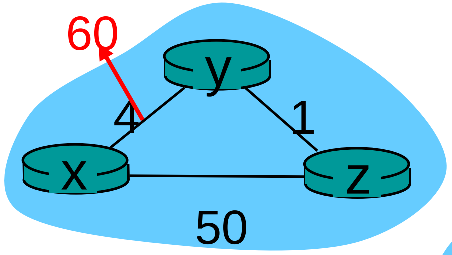
At time t_0, *y* detects the link-cost change (4 → 1), updates its DV, and informs its neighbors.

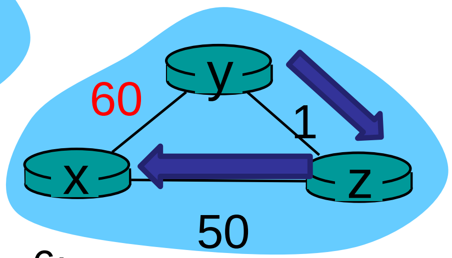
At time t_1, *z* receives the update from *y* and updates its table. It computes a new least cost to *x* (5 →2) and sends its neighbors its DV.

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:





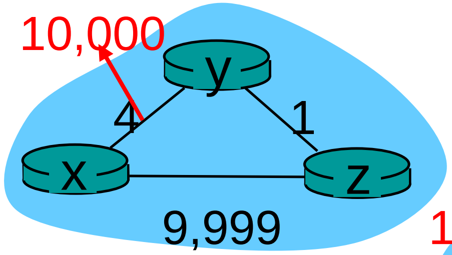
- Before the link cost changes
  - $D_y(x) = 4$, $D_z(x) = 5$
- At time t0, y detects the link-cost change and re-compute its dv
  - $D_y(x) = \min\{c(y,x)+ D_x(x), c(y,z)+ D_z(x)\} = \min\{60+0, 1+5\} = 6$;
  - Now we have routing loop: y sends data to z in order to get to x; z will send data back to y in order to get to x;
  - At time t1, y sends its new dv to z; after z receives y's new dv; z can update

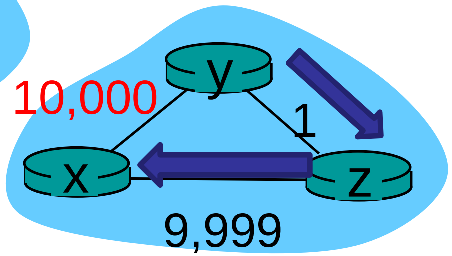
    $D_z(x) = \min\{c(z,y)+ D_y(x), c(z,x)+ D_x(x)\} = \min\{1+6, 50+0\} = 7$;
  - At time t2, z sends its new dv to y; similarly y can update

    $D_y(x) = \min\{c(y,x)+ D_x(x), c(y,z)+ D_z(x)\} = \min\{60+0, 1+7\} = 8$;
  - Then $D_z(x) = \min\{c(z,y)+ D_y(x), c(z,x)+ D_x(x)\} = \min\{1+8, 50+0\} = 9$;
  - $D_y(x) = \min\{c(y,x)+ D_x(x), c(y,z)+ D_z(x)\} = \min\{60+0, 1+9\} = 10$;
  - …
  - $D_z(x) = \min\{c(z,y)+ D_y(x), c(z,x)+ D_x(x)\} = \min\{1+50, 50+0\} = \mathbf{50}$;
  - $D_y(x) = \min\{c(y,x)+ D_x(x), c(y,z)+ D_z(x)\} = \min\{60+0, 1+50\} = 51$;
  - $D_z(x) = \min\{c(z,y)+ D_y(x), c(z,x)+ D_x(x)\} = \min\{1+51, 50+0\} = \mathbf{50}$;

Distance Vector: link cost changes

Link cost changes:





- Before the link cost changes
  - $D_y(x) = 4$, $D_z(x) = 5$
- At time t0, y detects the link-cost

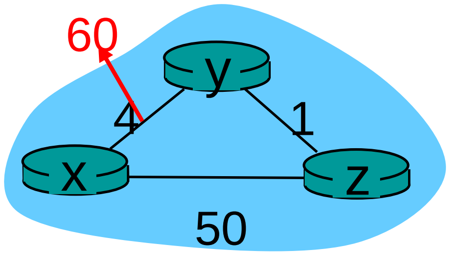
Change and re-compute its dv

  - …
  - $D_z(x) = \min\{c(z,y) + D_y(x),\ c(z,x) + D_x(x)\} = \min\{1+9999,\ 9999+0\} = \mathbf{9999}$;
  - $D_y(x) = \min\{c(y,x) + D_x(x),\ c(y,z) + D_z(x)\} = \min\{10000+0,\ 1+9999\} = 10000$;
  - $D_z(x) = \min\{c(z,y) + D_y(x),\ c(z,x) + D_x(x)\} = \min\{1+10000,\ 9999+0\} = \mathbf{9999}$;

The process stops after z computes the cost of its path via y to be greater than 50; then it chooses the path z→x (cost is 50).
The bad news about the increase in link cost has travelled slowly!
What if c(y,x) had changed from 4 to 10,000 and c(z,x) had been 9,999? **Count-to-infinity** problem!

Distance Vector: link cost changes

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)
- Before the link cost changes
  - $D_y(x) = 4$, $D_z(x) = 5$, but z will lie to y saying "**$D_z(x) = \infty$**" *(poisoned reverse)*
- At time t0, y detects the link-cost

change and re-compute its dv

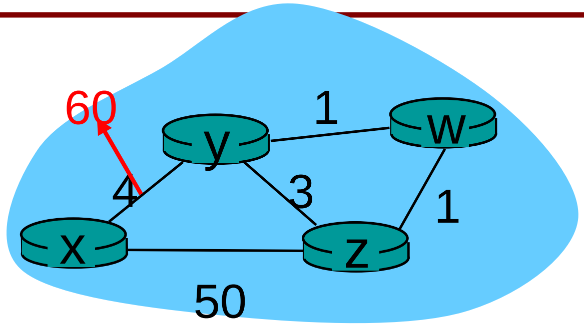
  - $D_y(x) = \min\{c(y,x)+ D_x(x), c(y,z)+ D_z(x)\} = \min \{60+0, 1+ \infty\} = 60$;
  - Now y sends data directly to x;
  - At time t1, y sends its new dv to z; after z receives y's new dv; z can update

    $D_z(x) = \min\{c(z,y)+ D_y(x), c(z,x)+ D_x(x)\} = \min \{1+60, 50+0\} = 50$;
  - At time t2, z sends its new dv to y without lying since it will not route through y; similarly y can update

    $D_y(x) = \min\{c(y,x)+ D_x(x), c(y,z)+ D_z(x)\} = \min \{60+0, 1+50\} = 51$;
  - Then $D_z(x) = \min\{c(z,y)+ D_y(x), c(z,x)+ D_x(x)\} = \min \{1+ \infty, 50+0\} = 50$; *y lies to z this time because it routes through z;*

Looks like this little lie does help!

But keep lying does not *solve* the problem!



Consider y,z,w distance table entries to x only. Using poisoned reverse,

z → w, $D_z(x) = \infty$; z → y $D_z(x) = 6$;

w → y, $D_w(x) = \infty$; w → z $D_w(x) = 5$;

y → w, $D_y(x) = 4$; y → z $D_y(x) = 4$;

Then there is link-cost change (4→60);

At t1, y updates its $D_y(x) = 9$ (via z); y → w, $D_y(x) = 9$; y → z $D_y(x) = \infty$;

$D_y(x) = \min\{c(y,z)+D_z(x),\ c(y,w) + D_w(x),\ c(y,x) + D_x(x)\} = \min\{3+6,\ 1+\infty,\ 60+0\} = 9$ (via z)

	t0	t1	t2	t3	t4
z	→ w, $D_z(x) = \infty$; → y, $D_z(x) = 6$;		No change	→ w, $D_z(x) = \infty$; → y, $D_z(x) = 11$;	
w	→ y, $D_w(x) = \infty$; → z $D_w(x) = 5$;		→ y, $D_w(x) = \infty$; → z, $D_w(x) = 10$;		No change
y	→ w, $D_y(x) = 4$; → z, $D_y(x) = 4$;	→ w, $D_y(x) = 9$; → z, $D_y(x) = \infty$;		No change	→ w, $D_y(x) = 14$; → z $D_y(x) = \infty$;

This continues y-w-z-y-w-z-y-w-z; there is routing loop (y-z, z-w, w-y) [ZL]

Comparison of LS and DV algorithms

Message complexity

- LS: with n nodes, E links, O(nE) msgs sent
- DV: exchange between neighbors only
  - convergence time varies

Speed of Convergence

- LS: $O(n^2)$ algorithm requires O(nE) msgs
  - may have oscillations
- DV: convergence time varies
  - may be routing loops
  - count-to-infinity problem

Robustness: what happens if router malfunctions?

LS:

- node can advertise incorrect *link* cost
- each node computes only its *own* table

DV:

- DV node can advertise incorrect *path* cost
- each node's table used by others
  - o error propagate thru network

Summary

- Network layer overview
- Routing overview
- Link-state routing (Dijkstra's algorithm)
- Distance-vector routing (Bellman-Ford)
  - B-F equation
  - B-F algorithm
  - Count-to-infinity problem and poisoned reverse
  - LS vs DV
- Summary

References

- [KR3] James F. Kurose, Keith W. Ross, *Computer networking: a top-down approach featuring the Internet*, 3rd edition.
- [PD5] Larry L. Peterson, Bruce S. Davie, *Computer networks: a systems approach*, 5th edition
- [TW5] Andrew S. Tanenbaum, David J. Wetherall, *Computer network*, 5th edition
- [LHBi]Y-D. Lin, R-H. Hwang, F. Baker, *Computer network: an open source approach*, International edition
- [ZL] Lilin Zhang, CSC358 Tutorial 9, University of Toronto, http://www.cs.toronto.edu/~ahchinaei/teaching/2016jan/csc358/Tut09-taSlides.pdf

Acknowledgements

- All slides are developed based on slides from the following two sources:
  - Dr DongSeong Kim’s slides for COSC264, University of Canterbury;
  - Prof Aleksandar Kuzmanovic’s lecture notes for CS340,Northwestern University, https://users.cs.northwestern.edu/~akuzma/classes/CS340-w05/lecture_notes.htm