Computer Networks I

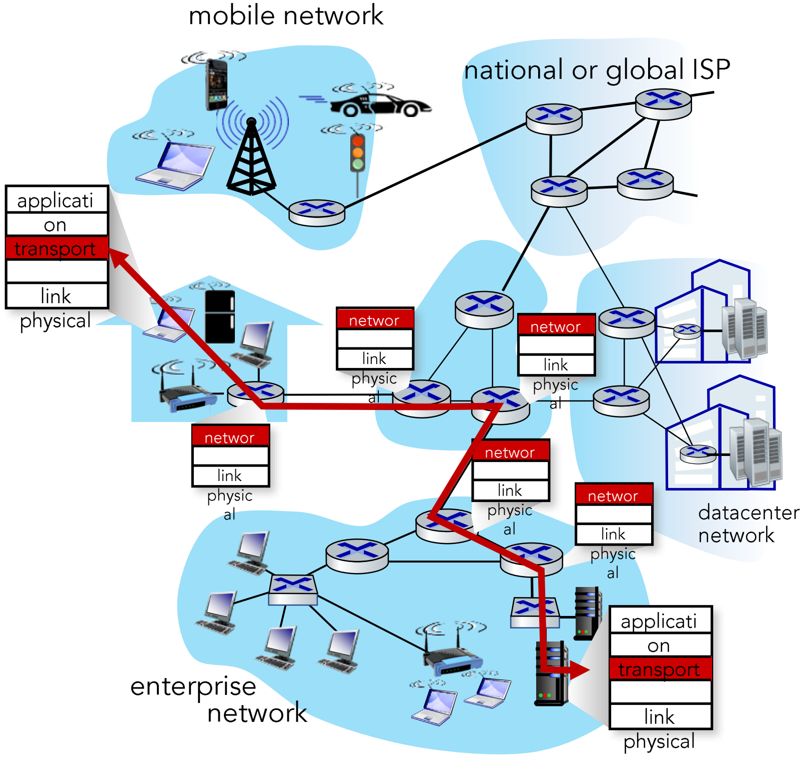
Network Layer Details - 2

Amitangshu Pal
Computer Science and Engineering
IIT Kanpur

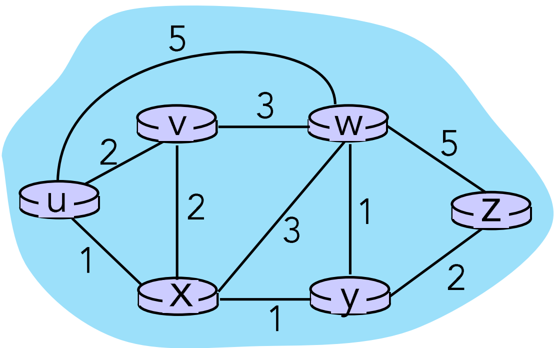
Routing protocols

Routing protocol goal: determine "good" paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- path: sequence of routers packets traverse from given initial source host to final destination host
- "good": least "cost", "fastest", "least congested"



Graph abstraction: link costs



$c_{a,b}$: cost of direct link connecting a and b
e.g., $c_{w,z} = 5$, $c_{u,z} = \infty$

cost defined by network operator: could always be 1, or inversely related to bandwidth, or inversely related to congestion

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) }

Routing algorithm classification

global: all routers have complete topology, link cost info
- "link state" algorithms

How fast do routes change?

static: routes change slowly over time

dynamic: routes change more quickly
- periodic updates or in response to link cost changes

decentralized: iterative process of computation, exchange of info with neighbors
- routers initially only know link costs to attached neighbors
- "distance vector" algorithms

global or decentralized information?

Link State Routing Protocol

Dijkstra's link-state routing algorithm

- centralized: network topology, 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 destinations

notation

- $c_{x,y}$: direct link cost from node x to y; $= \infty$ if not direct neighbors
- D(v): current estimate of cost of least-cost-path from source to destination v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least-cost-path definitively known

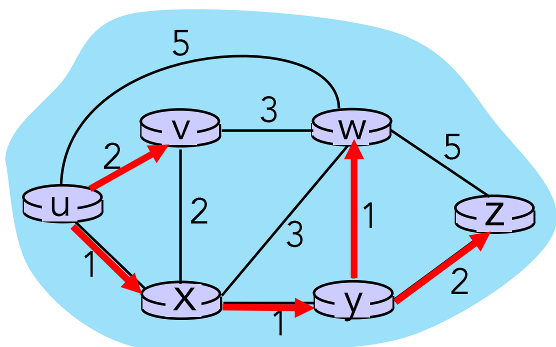
Dijkstra's link-state routing algorithm

Initialization:
  N' = {u}   /* compute least cost path from u to all other nodes */
  for all nodes v
    if v adjacent to u   /* u initially knows direct-path-cost only to direct neighbors */
      then D(v) = $c_{u,v}$ /* but may not be minimum cost! */
    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 least-path-cost to v is either old least-cost-path to v or known
  least-cost-path to w plus direct-cost from w to v */
until all nodes in N'

Dijkstra's algorithm: an example

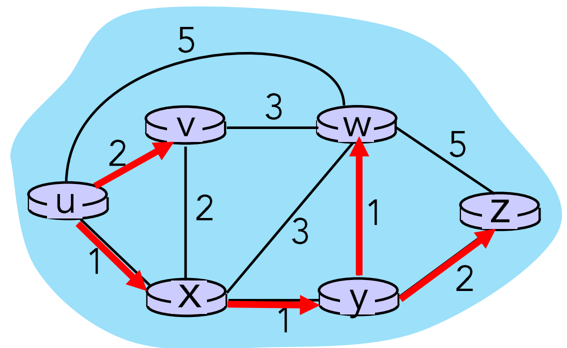
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



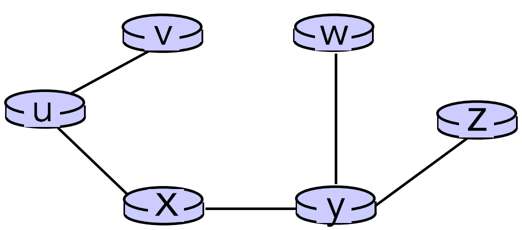
Initialization (step 0): For all a: if a adjacent to then D(a) = $c_{u,a}$

find a not in N' such that D(a) is a minimum
add a to N'
update D(b) for all b adjacent to a and not in N' :
$D(b) = \min(D(b), D(a) + c_{a,b})$

Dijkstra's algorithm: an example



resulting least-cost-path tree from u:



resulting forwarding table in u:

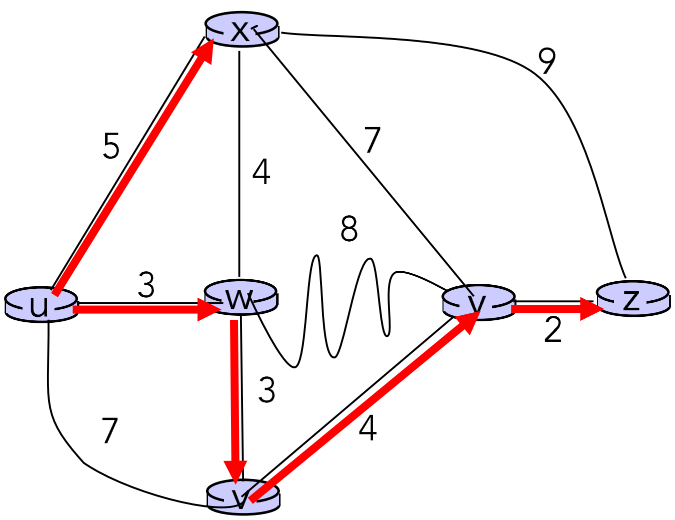
destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
x	(u,x)

route from u to v directly

route from u to all other destinations via x

Dijkstra's algorithm: another example

Step	N'	v D(v), p(v)	w D(w), p(w)	x D(x), p(x)	y D(y), p(y)	z D(z), p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					



Notes:

- Construct least-cost-path tree by tracing predecessor nodes
- Ties can exist (can be broken arbitrarily)

Dijkstra's algorithm: discussion

Algorithm complexity: n nodes

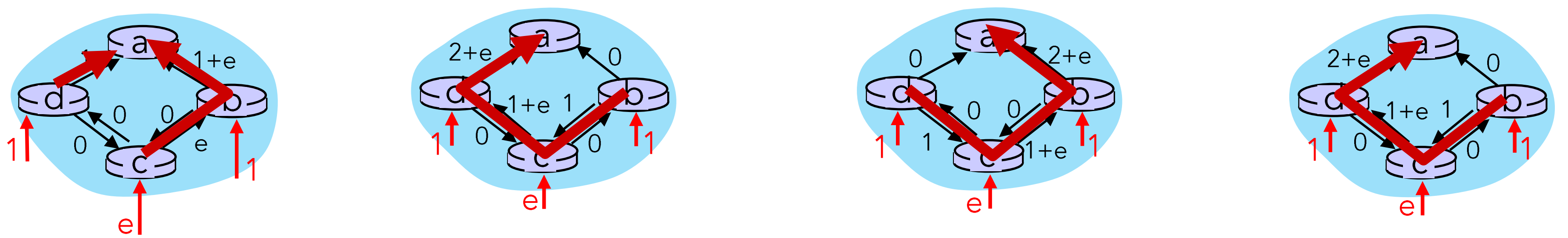
- each of n iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$ complexity
- more efficient implementations possible: O(nlogn)

message complexity:

- each router must broadcast its link state information to other n routers
- efficient (and interesting!) broadcast algorithms: O(n) link crossings to disseminate a broadcast message from one source
- each router's message crosses O(n) links: overall message complexity: $O(n^2)$

Dijkstra's algorithm: Oscillations

- When link costs depend on traffic volume, route oscillations possible
- Sample scenario:
  - Routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
  - Link costs are directional, and volume-dependent



initially

given these costs, find new routing…. resulting in new costs

given these costs, find new routing…. resulting in new costs

given these costs, find new routing…. resulting in new costs

Distance Vector Routing Protocol

Distance vector algorithm

Based on Bellman-Ford (BF) equation (dynamic programming):

Bellman-Ford equation

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

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

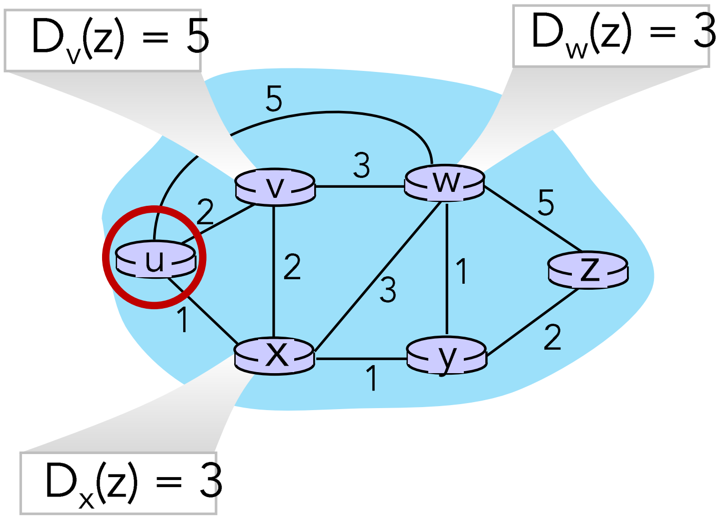
v's estimated least-cost-path cost to y

min taken over all neighbors v of x

direct cost of link from x to v

Bellman-Ford Example

Suppose that u's neighboring nodes, x,v,w, know that for destination z:



Bellman-Ford equation says:

$$
\begin{aligned}
D_u(z) &= \min \{ c_{u,v} + D_v(z), \\
&\qquad c_{u,x} + D_x(z), \\
&\qquad c_{u,w} + D_w(z) \} \\
&= \min \{2 + 5, \\
&\qquad 1 + 3, \\
&\qquad 5 + 3\} = 4
\end{aligned}
$$

node achieving minimum (x) is next hop on estimated least-cost path to destination (z)

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:

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

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

Distance vector algorithm:

each node:

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

↓

recompute DV estimates using DV received from neighbor

↓

if DV to any destination has changed, notify neighbors

iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed, self-stopping: each node notifies neighbors only when its DV changes

- neighbors then notify their neighbors – only if necessary
- no notification received, no actions taken!

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

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

node x table

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

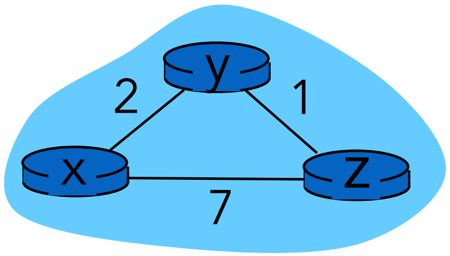
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

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



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

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

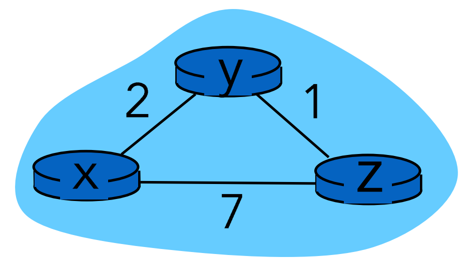
node z table

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

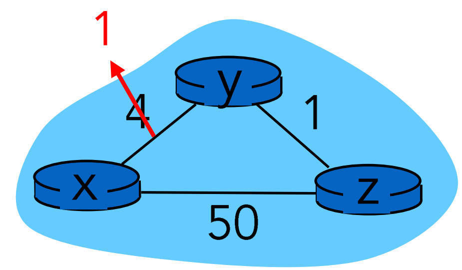
time



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"

t_0 : y detects link-cost change, updates its DV, informs its neighbors.

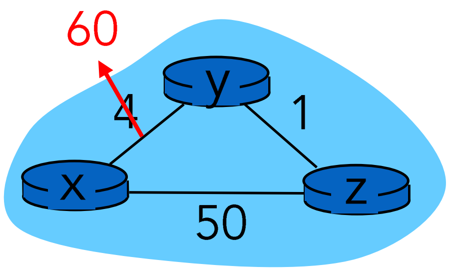
t_1 : z receives update from y, updates its table, computes new least cost to x , sends its neighbors its DV.

t_2 : y receives z's update, updates its distance table. y's least costs do not change, so y does not send a message to z.

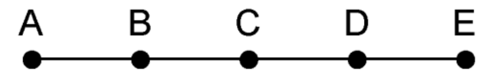
Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- bad news travels slow - “count to infinity” problem!
- 44 iterations before algorithm stabilizes



Distance vector: link cost changes



A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			
	•	•	•	•	

link cost changes:

- node detects local link cost change
- bad news travels slow - “count to infinity” problem!

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 count to infinity problem?

Comparison of LS and DV algorithms

message complexity

- LS: n routers, $O(n^2)$ messages sent
- DV: exchange between neighbors; convergence time varies

speed of convergence

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

robustness: what happens if router malfunctions, or is compromised?

LS:

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

DV:

- DV router can advertise incorrect path cost ("I have a really low cost path to everywhere"): black-holing
- each router's table used by others: error propagate thru network

THANK YOU

QUESTIONS???