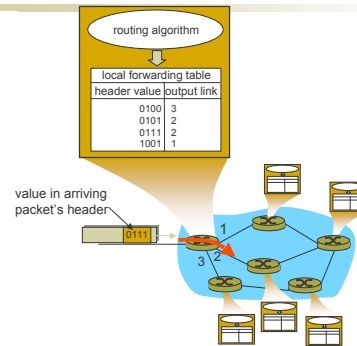


Routing

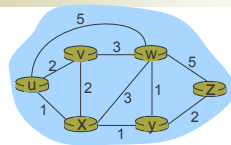
- Routing algorithms
 - Link state
 - Distance Vector



Interplay between routing and forwarding



Graph abstraction



Graph: $G = (N, E)$

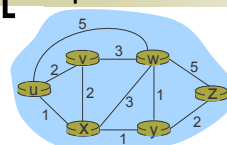
N = set of routers = { u, v, w, x, y, z }

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

Remark: Graph abstraction is useful in other network contexts
 Example: P2P, where N is set of peers and E is set of TCP connections



Graph abstraction: costs



• $c(x,x')$ = cost of link (x,x')

- e.g., $c(w,z) = 5$

• cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

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

Question: What's the least-cost path between u and z ?

Routing algorithm: algorithm that finds least-cost path



Routing Algorithm classification

Global or decentralized information?

Global:

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

Decentralized:

- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- "distance vector" algorithms

Static or dynamic?

Static:

- routes change slowly over time

Dynamic:

- routes change more quickly
 - periodic update
 - in response to link cost changes



A Link-State Routing Algorithm

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



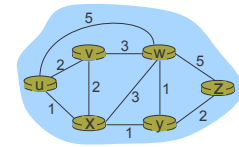
Dijkstra's Algorithm

1. Confirmed = {}
 2. for all neighbors N of the source S
 1. add (N, cost(S,N), N) to Tentative list
 3. while Tentative not empty
 1. Let (M,C,N) = node in Tentative with smallest cost
 2. Move (M,C,N) to Confirmed
 3. For all neighbors N' of M
 1. Add (N', C+cost(M,N'), N) to Tentative list unless a smaller entry for N' already exists
- Two lists
 - Confirmed
 - Tentative
 - Each list contains:
 - (Destination, Cost, NextHop)



Dijkstra's algorithm: example

Confirmed	Tentative
(x,1,x)	(v,2,v)
(v,2,v)	(w,5,w)
(y,2,x)	(x,1,x)
(w,3,x)	(w,4,x)
(z,4,x)	(y,2,x)
	(w,3,x)
	(z,4,x)



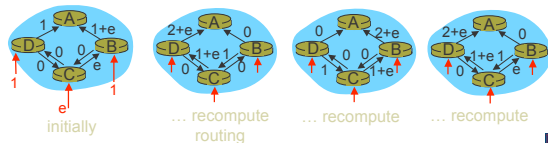
Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

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

Oscillations possible:

- e.g., link cost = amount of carried traffic



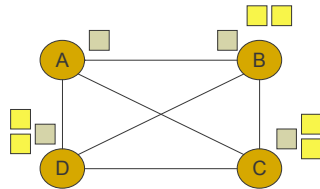
Link State Routing

- Requires global view of all links
 - How do nodes learn this information?



Flooding

- Loops
- Broadcast storms



Controlled Flooding

- Ignore duplicate messages
 - Cache to remember what's been sent before
- Special case: sequence numbers
 - Used with broadcast updates
 - Source assigns increasing seq no to broadcast
 - Record latest seq no from each source
 - Discard messages with old seq no's
- Ensures always use the latest update



Distance Vector Algorithm (1)

Bellman-Ford Equation (dynamic programming)

Define

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

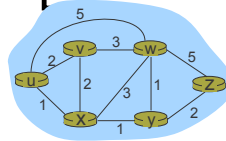
Then

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

where min is taken over all neighbors of x



Bellman-Ford example (2)



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

B-F equation says:

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

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



Distance Vector Algorithm (3)

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



Distance vector algorithm (4)

Basic idea:

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

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

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



Distance Vector Algorithm (5)

Iterative,

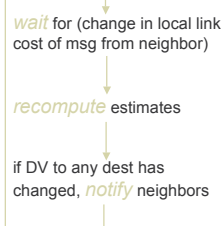
asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

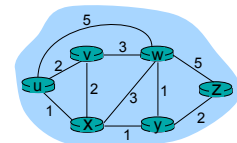
Distributed:

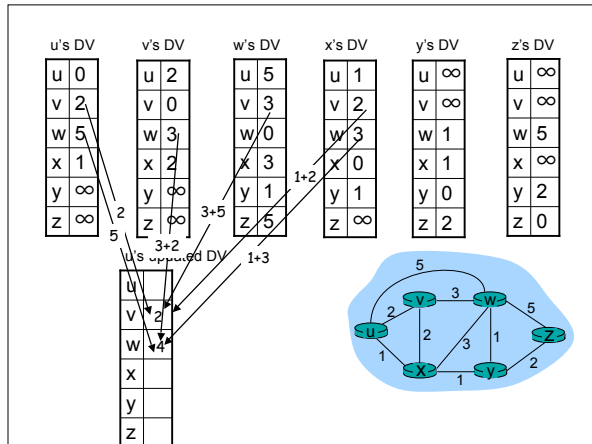
- each node notifies neighbors *only* when its DV changes
 - neighbors then notify their neighbors if necessary

Each node:



u's DV	v's DV	w's DV	x's DV	y's DV	z's DV
u 0	u 2	u 5	u 1	u ∞	u ∞
v 2	v 0	v 3	v 2	v ∞	v ∞
w 5	w 3	w 0	w 3	w 1	w 5
x 1	x 2	x 3	x 0	x 1	x ∞
y ∞	y ∞	y 1	y 1	y 0	y 2
z ∞	z ∞	z 5	z ∞	z 2	z 0





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 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:
 good news travels fast
 bad news travels slow - "count to infinity" problem!
 44 iterations before algorithm stabilizes: see text
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?

Count-to-infinity Problem

x's DV

y	4
z	5

y's DV

x	4
z	1

z's DV

x	5
y	1

dist=60

dist=5+1

Count-to-infinity Problem

x's DV

y	4
z	5

y's DV

x	6
z	1

z's DV

x	5
y	1

dist=60

dist=6+1

dist=50

Count-to-infinity Problem

x's DV

y	4
z	5

y's DV

x	6
z	1

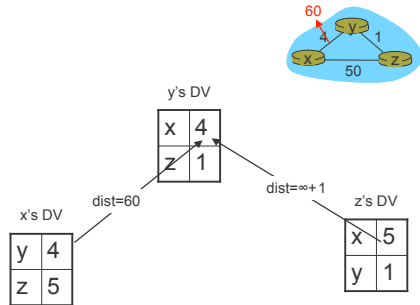
z's DV

x	7
y	1

dist=60

dist=7+1

Poisoned Reverse



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

Robustness: what happens if router malfunctions?

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

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

DV:

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