Routing Algorithms

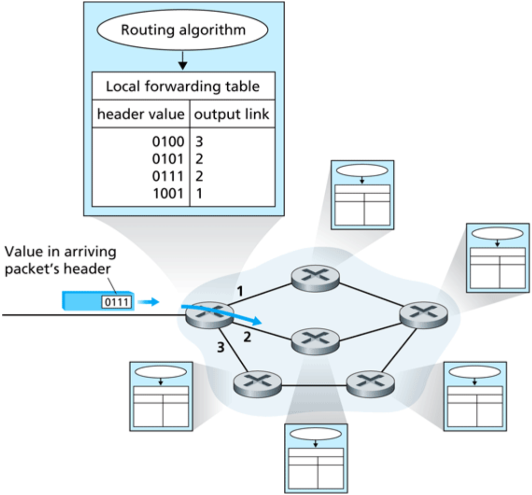
Daniel Zappala

CS 460 Computer Networking
Brigham Young University

Routing

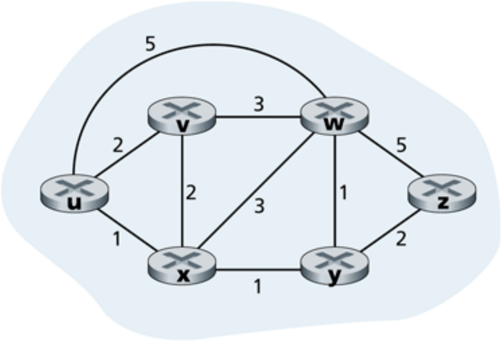
- **How does the Internet determine which path to use from the source to the destination?**
- Challenges
  - need to handle hundreds of thousands of routes
  - need to handle changes anywhere in the network
  - stability is crucial

Packet Forwarding versus Routing



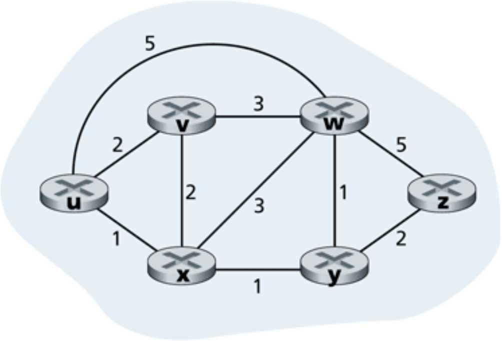
Graph Abstraction

- graph $G = (V, E)$
- nodes/vertices $V = u, v, w, x, y, z$
- links/edges $E =$ $(u, v), (u, x), (v, x), (v, w), (x, w), (x, y), (w, y), (w, z), (y, z)$
- useful for virtual networks as well, e.g. to model a peer-to-peer network of TCP connections



Edge Weights or Costs

- edges can have weights/costs
  - $c(w, z) = 5$
  - costs may be all 1
- cost of a path is the sum of the cost of all edges in the path
- **routing algorithm finds the least-cost path**



Types of Routing Algorithms

- link-state algorithm
  - routers advertise their links to every other router
  - eventually, all routers know complete topology and link costs
- distance-vector algorithm
  - routers start knowing path to immediate neighbors
  - routers advertise all known destinations and path lengths to neighbors
  - eventually, all routers know which neighbor has shortest path to all destinations

Link-State Routing

Link-State Basics

- can use Dijkstra's algorithm to compute least cost paths from one source to all other nodes
- notation
  - $c(x, y)$: link cost from node x to y, ∞ if no link between x and y
  - $D(v)$: current cost of path from source to destination v
  - F: set of nodes whose least cost path definitively known

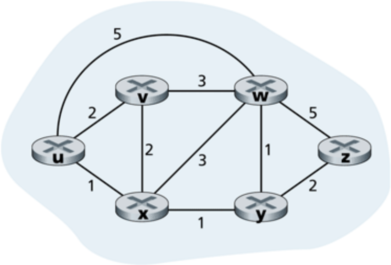
Dijsktra's Algorithm

```
F = [u]
for v in N:
    if v adjacent to u:
        D(v) = c(u,v)
    else:
        D(v) = infinity
while N != F:
    find w not in F such that D(w) is a minimum
    F.append(w)
    update D(v) for all v adjacent to w and not in F:
        D(v) = min(D(v),D(w) + c(w,v))
```

Example: Routing Table for U

- $h(v)$: next hop on shortest path toward v

Step	F	D(v),h(v)	D(w),h(w)	D(x),h(x)	D(y),h(y)	D(z),h(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					



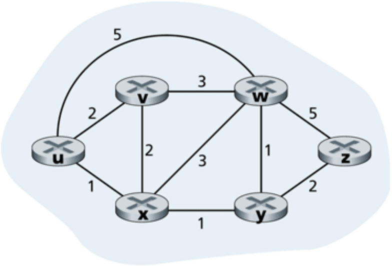
Algorithm Complexity

- each iteration needs to check all nodes w not in F
- $N(N+1)/2$ comparisons
- $O(N^2)$
- more efficient implementations possible: $O(NlogN)$

Distance-Vector Routing

Bellman-Ford

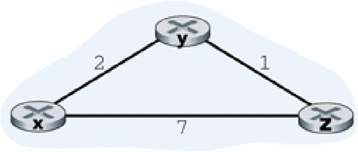
- distributed algorithm
  - $d_x(y)$: coast of least-cost path from x to y
  - $d_x(y) = min(c(x, v) + d_v(y))$
  - min is taken over all neighbors of x
- example
  - $d_v(z) = 5, d_x(z) = 3, d_w(z) = 3$
  - $d_u(z) = min(c(u, v) + d_v(z), c(u, x) + d_x(z), c(u, w) + d_w(z))$
  - $d_u(z) = min(2 + 5, 1 + 3, 5 + 3) = 4$
- node with minimum cost to destination is chosen as next hop



Bellman-Ford

- start with known costs o neighbors
- calculate best estimate of $d_x(y)$
- distance vector $D_x = \{D_x(y) : y \in N\}$
- send distance vector to neighbors whenever it changes
- update D_x using Bellman-Ford rule whenever local link cost change or neighbor's vector results in a new minimum
- **distributed, asynchronous algorithm**

Example



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

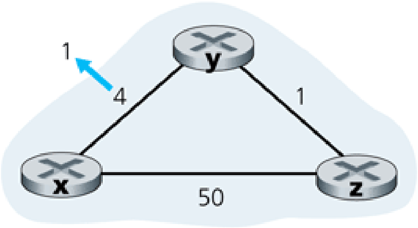
from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

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

Time

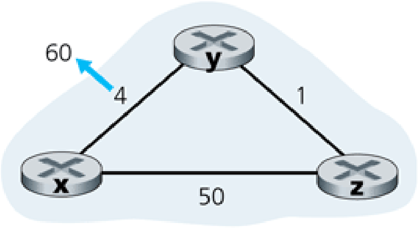
Reduction in Link Cost

- good news travels fast
  - at t_1, y detects cost change, updates DV, informs neighbors
  - at t_2, z receives update from y, updates its DV, informs neighbors
  - at t_3, y receives z's update, DV does not change, process finishes



Increase in Link Cost

- bad news travels slow
  - at t_0, $d_y(x) = 4, d_z(x) = 5$
  - at t_1, y detects link change, sets $d_y(x) = c(y,z) + d_z(x) = 6$, sends DV to z
  - at t_2, z gets DV, updates $d_z(x) = c(z,y) + d_y(x) = 7$
  - loop continues until $d_y(x) = 60$
- **count to infinity problem**



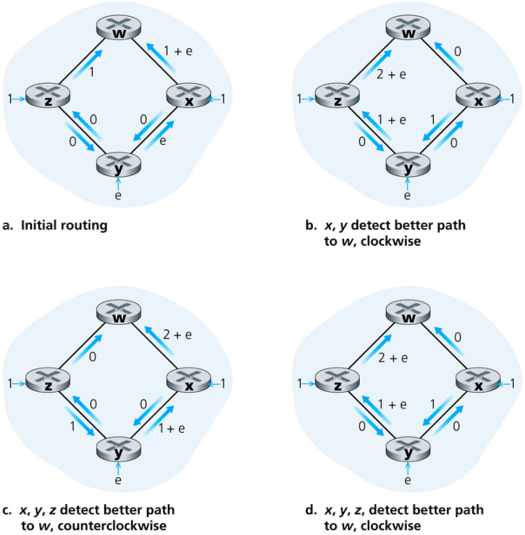
Solutions To Count To Infinity

- poisoned reverse
  - if z's best path is through y, z tells y that $d_z(x)$ is infinite
  - prevents y from routing to x via z
  - doesn't solve all looping problems
- path vector routing
  - include a path with each route, not just the next hop
  - eliminates all count-to-infinity problems
  - optimization: include only the next-to-last hop

Load-Based Routing

Load-Based Routing

- Difficult to have stable routes when link costs reflect congestion
- e.g. cost = number of traffic flows



a. Initial routing

b. *x*, *y* detect better path to *w*, clockwise

c. *x*, *y*, *z* detect better path to *w*, counterclockwise

d. *x*, *y*, *z*, detect better path to *w*, clockwise