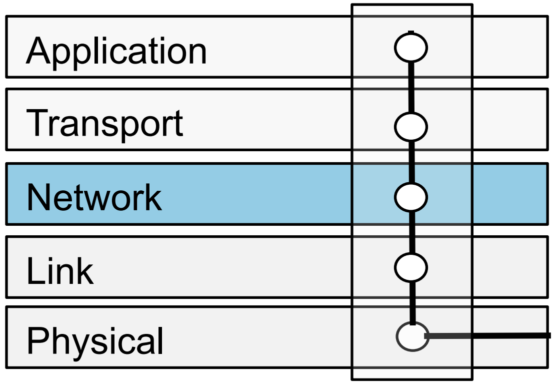
Routing Algorithms

To do …

- Understand principles behind network control plane
- Traditional routing algorithms
- Making routing scalable



Northwestern

Slides partially based on J. Kurose and K. Ross, "Computer Networking: A Top Down Approach", 7Ed., 2016

Network-layer functions

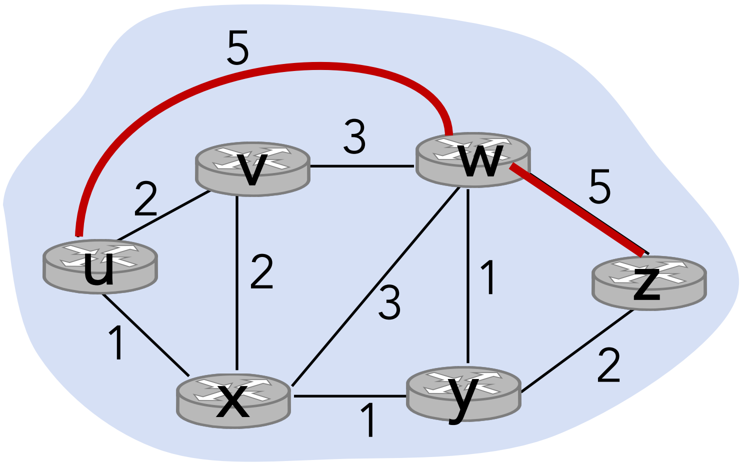
- Recall: two network-layer functions
  - Forwarding: move packets from router's input to appropriate router output — Data plane
  - Routing: determine route taken by packets from source to destination — Control plane
- Two approaches to structuring network control plane
  - Per-router control (traditional)
  - Logically centralized control (software defined networking)
- *But how are routers' forwarding tables configured?*

Today: Routing protocols

- Goal of routing protocols: to determine "good" paths from sending to receiving host, through network of routers
  - Path: sequence of routers packets will traverse from src to dst
  - "Good" – typically least "cost", "fastest", "least congested"

- Next time: Internet routing

To formulate routing problems – A graph



G = (N,E)
A set of routers: N = { u, v, w, x, y, z }
And links: E ={ (u,v), (u,x), (v,x), (v,w),
(x,w), (x,y), (w,y), (w,z), (y,z) }

c(x,x') = cost of link (x,x')

Could always be 1, or inversely related to bandwidth, or inversely related to congestion

Cost of path $(x_1, x_2, x_3,\ldots, x_p) = \boldsymbol{\Sigma}\, c(x_i,x_{i+1})$ for i: 1..p

Cost of path (u,w,z) = c(u,w) + c(w,z) = 5 + 5 = 10

Key question: what is the least-cost path between u and z?

Routing algorithm classification

- Global
  - All routers have complete topology (global), link cost info
  - "Link state" algorithms
- Or decentralized
  - Router knows physically-connected neighbors, link costs to neighbors
  - Iterative process of computation, exchange of info with neighbors
  - "Distance vector" algorithms (each node keeps a vector of estimates)
- Another broad way to classify them – 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

- Net topology and all link costs known to all nodes
  - Via "link state broadcast", each node broadcast link-state packets to all other nodes (each packet containing identity and cost of attached links)
  - All nodes have the same info (global)
- Computes least cost paths from one node (source) to all others
  - Gives forwarding table for that node
- Algorithm is iterative: after *k*th iterations, know least cost path to *k* destinations

Dijsktra's algorithm

Notation:

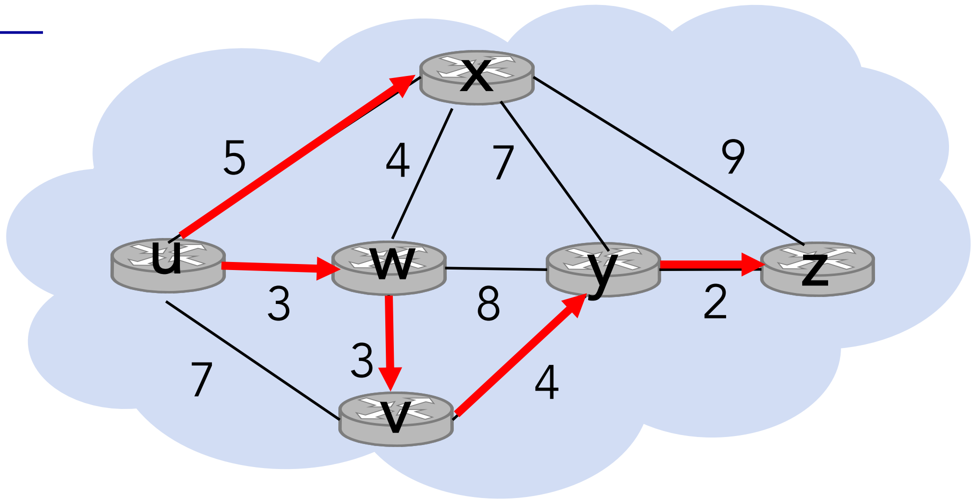
- *c(x,y):* link cost from x to y; ∞ if not direct neighbors
- *D(v):* current cost of path from src to dst v
- *p(v):* previous node on path from src to v
- *N':* set of nodes whose least cost path is definitively known

```
/* Initialization */
N' = {u}
for all nodes v
   if v adjacent to u
      then D(v) = c(u,v)
   else D(v) = ∞

repeat
   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 cost is either old cost to v or known
    shortest path cost to w plus cost from w to v */
until N'= N
```

Dijkstra's algorithm: 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	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					

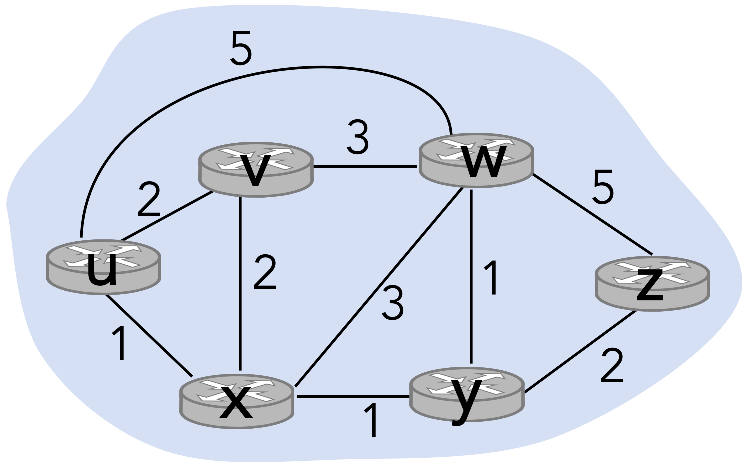


notes:

- Construct shortest path tree by tracing predecessor nodes
- Ties can exist, broken arbitrarily

Dijkstra's algorithm: Example 2

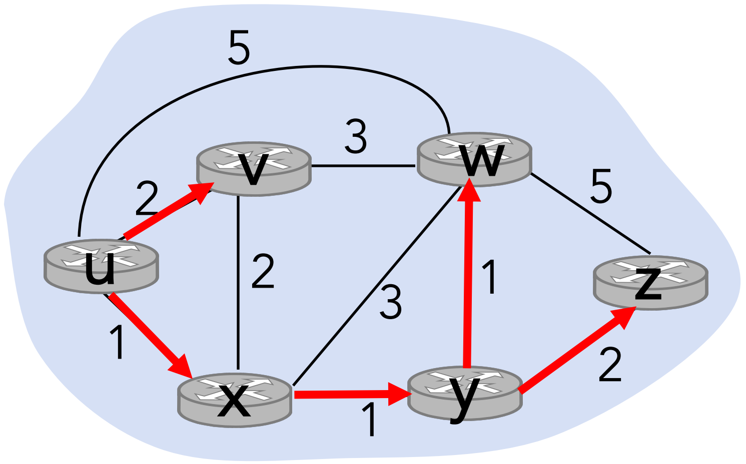
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



Dijkstra's algorithm: 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					

Resulting shortest-path tree from u

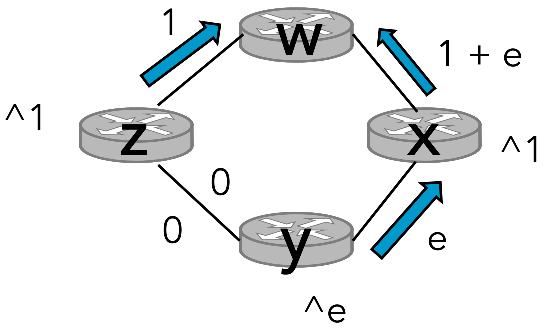


Resulting forwarding table in u

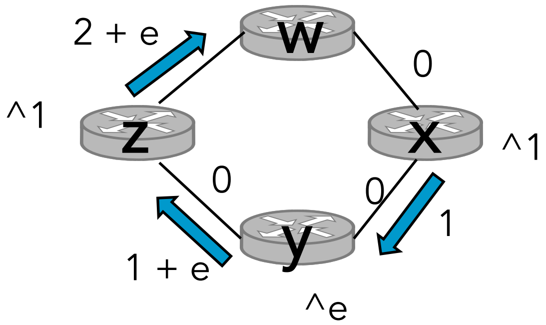
Destination	Link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

Dijkstra's algorithm – Complexity and concerns

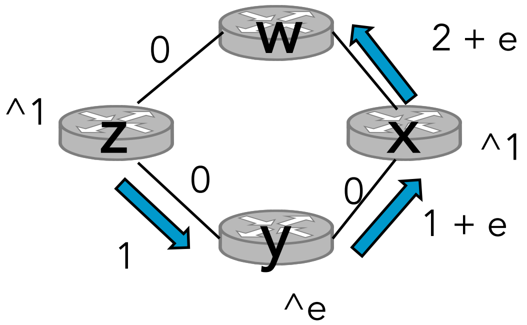
- Complexity: *n* nodes
  - Each iteration: need to check all nodes, *w*, not in *N′*
  - First iteration look at $n - 1$, then $n - 2$, … $n(n+1)/2$ comparisons: $O(n^2)$
  - More efficient implementations possible: $O(n \log n)$
- A potential problem – oscillation
  - e.g., suppose link cost equals amount of carried traffic



Initially, z and x add 1 and x adds e



Given this cost, better go clockwise



Given this cost, better go counterclockwise

…

Distance vector algorithm

- Link-state algorithm uses global info, distance vector is
  - Distributed – Each node gets info from one or more of its directly attached nodes, calculates distances and distribute to its neighbors
  - Iterative – Do this until no more info is exchanged between neighbors
  - Asynchronous – Nodes do this whenever

- Before looking at the algorithm, a key equation: Bellman-Ford

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

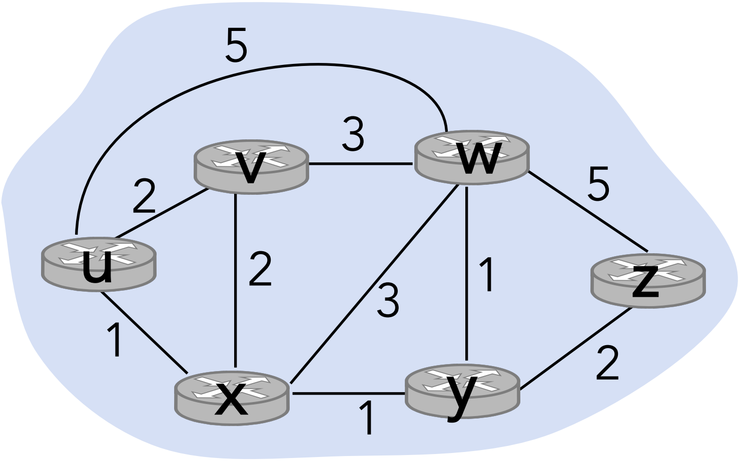
  Then $d_x(y) = \mathit{min} \{c(x,v) + d_v(y) \}$

  cost from neighbor v to destination y

  cost to neighbor v

  min taken over all neighbors v of x

Bellman-Ford with an example



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

B-F equation states

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

Node achieving minimum is next hop
in shortest path, used in forwarding table

Distance vector algorithm

- $D_x(y)$ = estimate of least cost from x to y
  - x maintains distance vector $D_x = [D_x(y): y \in N]$
- Node x
  - knows cost to each neighbor v: $c(x,v)$
  - maintains its neighbors' distance vectors.
    For each neighbor v, x maintains $D_v = [D_v(y): y \in N]$
- Key idea
  - From time-to-time, each node sends its own DV estimate to neighbors
  - When x receives new DV estimate from neighbor, updates its own DV using

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

Distance vector algorithm

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

each node:

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

recompute estimates

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

Node x

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

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

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

Node y

from \ *cost to*	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

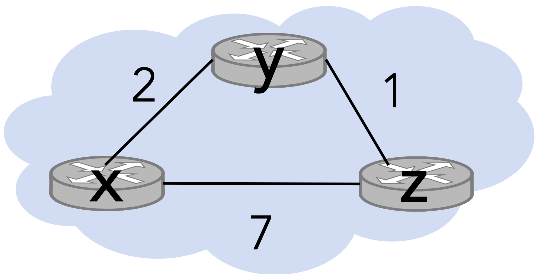
from \ *cost to*	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

Node z

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

time



Node x

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Node y

cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Node z

cost to

from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

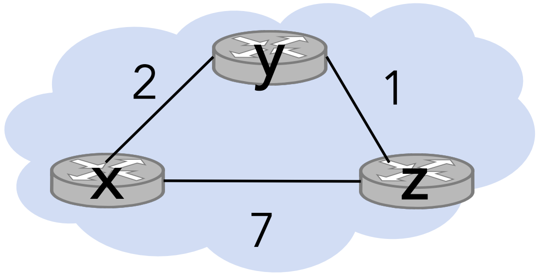
cost to

from	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

cost to

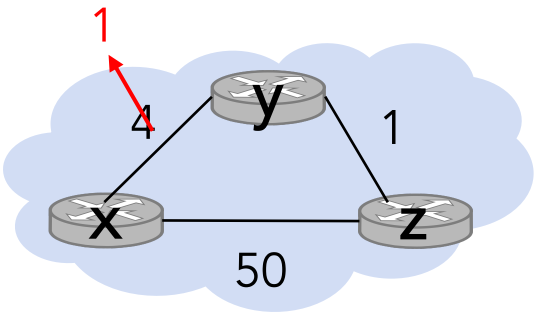
from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

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"

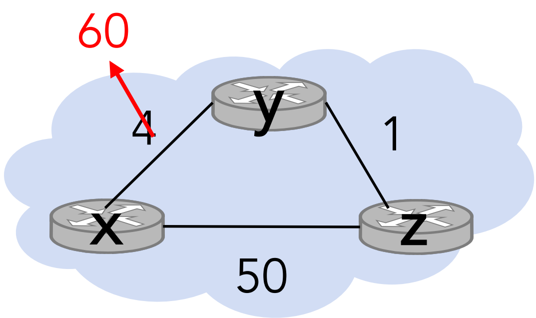
t_0 : *y* detects link-cost change (4 to 1), updates its DV, informs its neighbors.

t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x* (from 5 to 2), 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 increase

- Link cost changes
  - Node detects local link cost change



Before link cost change: $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$, $D_z(x) = 5$

t_0: y detects link-cost change, computes new minimum-cost path to x
– $D_y(x) = \min\{c(y,x) + D_x(x), c(y,z) + D_z(x)\} = \min\{60+0, 1 + 5\} = 6$

We have a routing loop! For y to get to x, go via z; and z goes via y …

We know this is wrong because of our global view; all y sees is that direct to x is 60 and z says it can get there in 5

t_1: Since y has a new minimum cost to x, it informs z

z receives new DV from y, y's minimum cost to x is 6, so $D_z(x) = \min\{50+0, 1+6\} = 7$ … and informs y

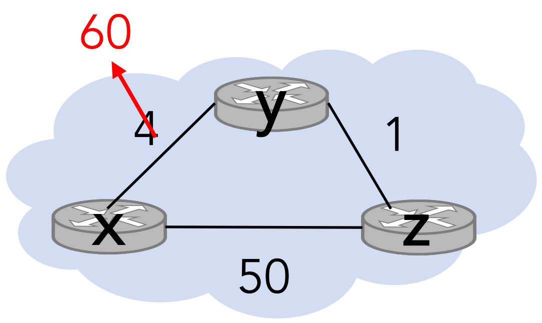
y receives new DV from z, z's minimum cost to x is 7, so $D_y(x) = \min\{60+0, 1+7\} = 8$ … and informs z

Bad news travels slow - "count to infinity" problem!

… 44 times! Until z computes cost via y to be > 50

Distance vector: Poisoned reverse

- To avoid it – poisoned reverse
  - If z routes through y to get to x :
  - z tells y its (z's) distance to x is infinite $D_z(x) = \infty$ even if it knows to be 5
  - So y won't route to x via z (it things z has no path to x)
- How it works?
  - When link costs changes to 60, y updates its table and continues to route directly to x and informs z of its new cost to x, $D_y(x) = 60$
  - z receives DV from y and switches to the direct route, informs y it can get to x in 50, $D_z(x) = 50$
  - y updates its distance table with $D_y(x) = 51$ and poisons the least-cost path to x $D_y(x) = \infty$
- Doesn't work for loops with 3+ nodes



Comparison of LS and DV algorithms

- Message complexity
  - LS: with n nodes, E links, O(n * E) 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
- 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 so error propagates through the entire network

Making routing scalable

- Our routing study thus far - idealized
  - All routers identical
  - Network "flat"
- … in practice

Scale with billions of destinations:

- Can't store all destinations in routing tables!
- Routing table exchange would swamp links!

Administrative autonomy

- Internet = network of networks
- Each network admin may want to control routing in its own network

Internet approach to scalable routing

- Aggregate routers into regions known as "autonomous systems" (AS) (a.k.a. "domains")
  - Each with own ASN, AS number (assigned by ICANN regional registries)
- Intra-AS routing
  - Routing among hosts, routers in same AS ("network")
  - Routers in AS run same intra-domain protocol, maybe different in others
  - Gateway router: at "edge" of AS, has link(s) to router(s) in other AS'es
- Inter-AS routing
  - Routing among AS'es
  - Gateways perform inter-domain routing (besides intra-domain routing)

Next time