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) \}

**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

- $c(x, x') = \text{cost of link } (x, x')$
  - e.g., $c(w, z) = 5$

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

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

**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
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
Dijsktra’s Algorithm

1. **Initialization:**
   2. \( N' = \{u\} \)
   3. for all nodes \( v \)
   4.   if \( v \) adjacent to \( u \)
   5.     then \( D(v) = c(u,v) \)
   6.   else \( D(v) = \infty \)

7. **Loop**
   8. find \( w \) not in \( N' \) such that \( D(w) \) is a minimum
   9. add \( w \) to \( N' \)
   10. update \( D(v) \) for all \( v \) adjacent to \( w \) and not in \( N' \):
       11. \( D(v) = \min( D(v), D(w) + c(w,v) ) \)
       12. /* new cost to \( v \) is either old cost to \( v \) or known
           shortest path cost to \( w \) plus cost from \( w \) to \( v \) */
   13. until all nodes in \( N' \)
## Dijkstra’s algorithm: example

<table>
<thead>
<tr>
<th>Step</th>
<th>N'</th>
<th>D(v),p(v)</th>
<th>D(w),p(w)</th>
<th>D(x),p(x)</th>
<th>D(y),p(y)</th>
<th>D(z),p(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>u</td>
<td>2,u</td>
<td>5,u</td>
<td>1,u</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>ux</td>
<td>2,u</td>
<td>4,x</td>
<td>2,x</td>
<td>4,y</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>uxy</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td>4,y</td>
</tr>
<tr>
<td>3</td>
<td>uxyv</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td>4,y</td>
</tr>
<tr>
<td>4</td>
<td>uxyvw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>uxyvwz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Resulting shortest-path tree from $u$:

Resulting forwarding table in $u$:

<table>
<thead>
<tr>
<th>destination</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v$</td>
<td>$(u,v)$</td>
</tr>
<tr>
<td>$x$</td>
<td>$(u,x)$</td>
</tr>
<tr>
<td>$y$</td>
<td>$(u,x)$</td>
</tr>
<tr>
<td>$w$</td>
<td>$(u,x)$</td>
</tr>
<tr>
<td>$z$</td>
<td>$(u,x)$</td>
</tr>
</tbody>
</table>
Shortest Path Routing

(a) Shortest path routing for the network shown.

(b) Routing with a new link added.

(c) Routing after link addition and deletion.

(d) Routing after another link addition.

(e) Routing after yet another link addition.

(f) Final routing with a new link addition.
Bellman-Ford Equation

Define
\[ d_x(y) := \text{cost of least-cost path from } x \text{ to } y \]
Then
\[ d_x(y) = \min \{ c(x,v) + d_v(y) \} \]

where \( \min \) is taken over all neighbors \( v \) of \( x \).
Bellman-Ford example

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

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z), c(u,x) + d_x(z), c(u,w) + d_w(z) \}$$

$$= \min \{2 + 5, 1 + 3, 5 + 3\} = 4$$

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

- \( D_x(y) = \) estimate of least cost from \( x \) to \( y \)

Node \( x \) knows cost to each neighbor \( v \): \( c(x,v) \)

Node \( x \) maintains distance vector \( D_x \)
\( = [D_x(y): y \in N] \)

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**

**Basic idea:**
From time-to-time, each node sends its own distance vector estimate to neighbors

Asynchronous

When a node $x$ receives new DV estimate from neighbor, it updates its own DV using B-F equation:

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

Under minor, natural conditions, the estimate $D_x(y)$ converge to 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:

1. wait for (change in local link cost or msg from neighbor)
2. recompute estimates
3. if DV to any dest has changed, notify neighbors
Node x table

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from x</td>
<td>0</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>from y</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>from z</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

Node y table

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from x</td>
<td>∞</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>from y</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>from z</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Node Z table

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from x</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>from y</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>from z</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

\[ D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} \]
\[ = \min\{2+1, 7+0\} = 3 \]
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?
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
  error propagate thru network