Routing

Routing protocol

Goal: determine “good” path (sequence of routers) thru network from source to dest.

Graph abstraction for routing algorithms:

- graph nodes are routers
- graph edges are physical links
  - link cost: delay, $ cost, or congestion level

“good” path:
- typically means minimum cost path
- other def’s possible

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

Notation:

- \( c(i,j) \): link cost from node \( i \) to \( j \). cost infinite 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 \( v \), that is next \( v \)
- \( N \): set of nodes whose least cost path definitively known

Initialization:

1. \( N = \{ A \} \)
2. for all nodes \( v \)
   3. if \( v \) adjacent to \( A \)
      4. then \( D(v) = c(A,v) \); \( p(v) = A \)
      5. else \( D(v) = \infty \)
6. end for

Loop

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

<table>
<thead>
<tr>
<th>Step</th>
<th>start N</th>
<th>D(B),p(B)</th>
<th>D(C),p(C)</th>
<th>D(D),p(D)</th>
<th>D(E),p(E)</th>
<th>D(F),p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>infinity</td>
<td>infinity</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2,A</td>
<td>4,D</td>
<td>2,D</td>
<td>infinity</td>
<td>infinity</td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>2,A</td>
<td>3,E</td>
<td>4,E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADEB</td>
<td>3,E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ADEBC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ADEBCF</td>
<td>4,E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Dijsktra’s Algorithm Revisited**

1. **Initialization:**
   2. $N = \{A\}$
   3. for all nodes $v$
      4. if $v$ adjacent to $A$
         5. then $D(v) = c(A,v)$ ; $n(v) = v$
      6. else $D(v) = \text{infinity}$

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. if $(D(v) > D(w) + c(w,v))$
           12. $D(v) = D(w) + c(w,v)$
           13. $n(v) = n(w)$
       14. /* new cost to $v$ is either old cost to $v$ or known shortest path cost to $w$ plus cost from $w$ to $v$ */
   15. until all nodes in $N$

**Dijkstra’s algorithm, discussion**

Algorithm complexity: $n$ nodes
- Naïve: each iteration: need to check all nodes, $w$, not in $N$
- $n*(n+1)/2$ comparisons: $O(n^2)$
- more efficient implementations possible: $O(n\log n)$ (what data structure?)

**Distance Vector Routing Algorithm**

iterative:
- continues until no nodes exchange info.
- self-terminating: no "signal" to stop
asynchronous:
- nodes need not exchange info/iterate in lock step!
distributed:
- each node communicates only with directly-attached neighbors

**Distance Table data structure**
- each node has its own
- row for each possible destination
- column for each directly-attached neighbor to node
- example: in node $X$, for dest. $Y$ via neighbor $Z$:

$$D^X(Y,Z) = \begin{cases} \text{distance from } X \text{ to } Y, \text{ via } Z \text{ as next hop} \\
\min \{D^Y(w) + c(X,Z)\} \end{cases}$$
**Distance Table: example**

Distance table gives routing table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>

\[ D^E(C,D) = c(E,D) + \min_w \{D^D(C,w) \} \]
\[ = 2+2 = 4 \]

\[ D^E(A,D) = c(E,D) + \min_w \{D^D(A,w) \} \]
\[ = 2+3 = 5 \] loop!

\[ D^E(A,B) = c(E,B) + \min_w \{D^B(A,w) \} \]
\[ = 8+6 = 14 \]

Distance table Routing table

Outgoing link to use, cost

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>D,5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>D,4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>D,4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Distance Vector Routing: overview

Iterative, asynchronous:
- each local iteration caused by:
  - local link cost change
  - message from neighbor: its least cost path change from neighbor

Distributed:
- each node notifies neighbors only when its least cost path to any destination changes
- neighbors then notify their neighbors if necessary

Each node:

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

recompute distance table

if least cost path to any dest has changed, notify neighbors

Distance Vector Algorithm:

At all nodes, X:

1. Initialization:
2. for all adjacent nodes v:
3. \[ D^X(*,v) = \text{infinity} \] /* the * operator means “for all rows” */
4. \[ D^X(v,v) = c(X,v) \]
5. for all destinations, y
6. send \( \min_w D^X(y,w) \) to each neighbor /* w over all X’s neighbors */
Distance Vector Algorithm (cont.):

8 loop
9 wait (until I see a link cost change to neighbor V
10 or until I receive update from neighbor V)
11
12 if (c(X,V) changes by d)
13 /* change cost to all dest's via neighbor v by d */
14 /* note: d could be positive or negative */
15 for all destinations y: D^X(y,V) = D^X(y,V) + d
16
17 else if (update received from V wrt destination Y)
18 /* shortest path from V to some Y has changed */
19 /* V has sent a new value for its min(DV(Y,w)) */
20 /* call this received new value is "newval" */
21 for the single destination y: D^X(Y,V) = c(X,V) + newval
22
23 if we have a new min(D^X(Y,w)) for any destination Y
24 send new value of min(D^X(Y,w)) to all neighbors
25
26 forever

Distance Vector Algorithm: example

Distance Vector: link cost changes

Link cost changes:
- node detects local link cost change
- updates distance table (line 15)
- if cost change in least cost path, notify neighbors (lines 23,24)

"good news travels fast"
Distance Vector: link cost changes

Link cost changes:
- good news travels fast
- bad news travels slow - “count to infinity” problem!

Distance Vector: 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 each
- **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