Both datagram and virtual-circuit based networks need to know how to construct forwarding tables

Forwarding:
- Select an output port based on destination address and routing table
- Simple and well-defined process performed locally at a given node
- Forwarding table used while forwarding packets
  - eg: (network number, interface id, MAC address)

Routing:
- Building the forwarding/routing table
- A complex, distributed algorithm problem
- Routing table may be a precursor to a forwarding table
  - eg: (network number, nextHopIP, CostToDestination)

Why do we need two tables?
Intra-domain Routing: Formulation

- Intra-domain routing ~ 100 routers
- Given:
  - Graph: where nodes are routers and edges are links
  - Cost: associated with each link
- Find:
  - Lowest-cost path between any two nodes
- Requirements:
  - Self-healing, traffic-sensitive, scalable

Distance-vector Routing: Basic Idea

- Each node:
  - Constructs a vector of distances to all other nodes
    - Distance vector
    - Distributes to immediate neighbors
- Neighbors use the distributed information to update their own distance vectors
- This distributed exchange-update-exchange should lead to globally consistent distance vectors (and routing tables)
Distance Table Data Structure

- Each node has its own table with a...  
  - Row for each possible destination  
  - Column for each directly-attached adjacent node (neighbor)

- Each table entry gives cost to reach destination via that adjacent node  
  - Distance = Cost

\[
D^f(Y, Z) = \text{distance from } X \text{ to } Y \text{ via } Z \text{ as first hop} \\
= c(X, Z) + \min_w \{D^f(Y, w)\}
\]

\[w = \{\text{neighbors of } Z\}\]

Distance Table Example

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

A loop?!

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

Loop!
**Distance Table Example**

- The distance table gives the routing table.
  - Just take the minimum cost per destination.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</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>

**Distance Vector Routing: Algorithm**

- Iterative, asynchronous – each local iteration caused by:
  - Local link cost change, or
  - Message from adjacent node that its least cost path to some destination has changed.

- Distributed:
  - Each node notifies adjacent nodes *only* when its least cost path to some destination changes.
  - Adjacent nodes then notify their adjacent nodes if this update changes a least cost path.

**Each node:**

- wait for change in local link cost or message from adjacent node
- recomputes distance table
- if least cost path to any destination has changed, notify adjacent nodes.
Distance Vector Algorithm: Example

\[ D(X,Y) = c(X,Z) + \min_w\{D'(Y,w)\} \]
\[ = 7 + 1 = 8 \]

\[ D'(Z,Y) = c(X,Y) + \min_w\{D''(Z,w)\} \]
\[ = 2 + 1 = 3 \]
Distance Vector: Link Cost Changes

- When a node detects a local link cost change:
  The nodes update its distance table.
  If the least cost path changes, the node notifies its neighbors.

**How long does convergence take?**

“Good news travels fast”

<table>
<thead>
<tr>
<th>Time</th>
<th>Distance Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>$D(XZ) = 4$ 6</td>
</tr>
<tr>
<td>$t_1$</td>
<td>$D(XZ) = 1$ 6</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$D(XZ) = 1$ 3</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$D(XZ) = 1$ 3</td>
</tr>
<tr>
<td>$t_4$</td>
<td>$D(XZ) = 1$ 3</td>
</tr>
</tbody>
</table>

Distance Vector: Link Cost Changes

- Good news travels fast, but...
- “Bad news” travels slow!
- The “count to infinity” problem

Routing Loop!
Does it Terminate?

<table>
<thead>
<tr>
<th>Time</th>
<th>Distance Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>$D(XZ) = 4$ 6</td>
</tr>
<tr>
<td>$t_1$</td>
<td>$D(XZ) = 60$ 5</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$D(XZ) = 60$ 5</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$D(XZ) = 60$ 5</td>
</tr>
<tr>
<td>$t_4$</td>
<td>$D(XZ) = 60$ 5</td>
</tr>
</tbody>
</table>

Algorithm continues on…
The Count to Infinity Problem

- The “poisoned reverse” technique
- If Z routes through Y to get to X:
  Then Z tells Y that Z’s distance to X is infinite

Initialization...

<table>
<thead>
<tr>
<th></th>
<th>via</th>
<th>D' Y</th>
<th>X</th>
<th>Z</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td>0</td>
<td>4</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td>50</td>
<td>60</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>60</td>
<td>60</td>
<td>∞</td>
<td></td>
</tr>
</tbody>
</table>

The Count to Infinity Problem

- The “poisoned reverse” technique
- If Z routes through Y to get to X:
  Then Z tells Y that Z’s distance to X is infinite
- (Will this completely solve the problem?)
  How about delaying advertising alternate routes after link failures?

Algorithm terminates

<table>
<thead>
<tr>
<th></th>
<th>via</th>
<th>D' Y</th>
<th>X</th>
<th>Z</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td>0</td>
<td>4</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td>50</td>
<td>60</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>60</td>
<td>60</td>
<td>∞</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © by Jasleen Kaur