Routing Algorithms

Routing algorithm

local forwarding table

<table>
<thead>
<tr>
<th>header value</th>
<th>output link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>3</td>
</tr>
<tr>
<td>0101</td>
<td>2</td>
</tr>
<tr>
<td>0111</td>
<td>2</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
</tr>
</tbody>
</table>

value in arriving packet's header
Routing Algorithms

➤ Software responsible for deciding which output line an incoming packet should be transmitted on
  ➤ Decision is made for the arrival of every datagram packet

➤ Desired properties:
  ➤ Correctness, simplicity, robustness, stability, fairness and optimality
Routing Algorithms (cont.)

- Non-adaptive algorithms
  - Routes are computed in advance, and off-line
  - The routing tables are loaded into routers

- Adaptive algorithms
  - Change their routing decisions on reflect the changes in the current traffic and topology
  - Still a variety of different algorithms (e.g., WHEN and HOW to change)
Non-adaptive Algorithms

→ Shortest Path Routing
  → Based on a well-known Dijkstra algorithm
  → Example
  → Pseudo program
→ Flooding
  → Every incoming packet is sent out on every outgoing line (except the one it arrived on)
  → A hop counter in the header to end the process
Graph: $G = (N,E)$

$N = \text{set of routers} = \{ u, v, w, x, y, z \}$

$E = \text{set of links} = \{ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) \}$

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 $(x_1, x_2, x_3, ..., x_p) = c(x_1,x_2) + c(x_2,x_3) + ... + c(x_{p-1},x_p)$

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; = \infty 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
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 \( D(v) = \infty \)

8 **Loop**
9 find \( w \) not in \( N' \) such that \( D(w) \) is a minimum
10 add \( w \) to \( N' \)
11 update \( D(v) \) for all \( v \) adjacent to \( w \) and not in \( N' \) :
12 \[ D(v) = \min( D(v), D(w) + c(w,v) ) \]
13 /* 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: example

Below is an example of applying Dijkstra's algorithm to find the shortest paths in a weighted graph.

<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>∞</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>∞</td>
</tr>
<tr>
<td>4</td>
<td>uxyvw</td>
<td></td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td>∞</td>
</tr>
<tr>
<td>5</td>
<td>uxyvwz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The graph is shown below with the initial distances and predecessors for each node.
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>