Introduction to Networking

Marcel Flores
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
- 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 routing table for that node
- Iterative: after k iterations, know least cost path to k dest.’s

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
Dijsktra’s Algorithm

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Start</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></td>
<td>( 2, D )</td>
<td>infinity</td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>( 2, A )</td>
<td>( 3, E )</td>
<td></td>
<td></td>
<td>( 4, E )</td>
</tr>
<tr>
<td>3</td>
<td>ADEB</td>
<td></td>
<td>( 3, E )</td>
<td></td>
<td></td>
<td>( 4, E )</td>
</tr>
<tr>
<td>4</td>
<td>ADEBC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 4, E )</td>
</tr>
<tr>
<td>5</td>
<td>ADEBCF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

\[
\lambda D (Y,Z) = \lambda c(X,Z) + \min_{\lambda w} \{ \lambda D^Z (Y,w) \}
\]

\(\lambda\)distance from X to \(\lambda\)Y, via Z as next hop
Distance Vector Routing Algorithm

iterative:
- continues until no nodes exchange info.
- self-terminating: no “signal” to stop

distributed:
- each node communicates only with directly-attached neighbors

Each node:
- wait for (change in local link cost of msg from neighbor)
- recompute distance table
- if least cost path to any dest has changed, notify neighbors
Routing Lab
Project 3

A distance-vector algorithm and a link-state algorithm in the context of a simple routing simulator.

Event-driven Simulation
main loop repeatedly pulls the earliest event from a queue and passes it to a handler until there are no more events in the queue.
.make TYPE=GENERIC” will build a single executable “routesim”, which contains no routing algorithm.

You will do TYPE=DISTANCEVECTOR and TYPE=LINKSTATE

To run: ./routesim topologyfile eventfile [singlestep]
Events in routesim come from the topology file, the event file, and from handlers that are executed in response to events. The topology file generally only contains events that construct the network topology (the graph):

- arrival_time ADD_NODE node_num latency bandwidth
- arrival_time DELETE_NODE node_num latency bandwidth
- arrival_time ADD_LINK src_node_num dest_node_num latency bandwidth
- arrival_time DELETE_LINK src_node_num dest_node_num latency bandwidth
The event file generally only contains events that modify link characteristics in the graph, or draw the graph, a path and etc.

- arrival_time CHANGE_NODE node_num latency bandwidth
- arrival_time CHANGE_LINK src_node_num dest_node_num latency bandwidth
- arrival_time DRAW_TOPOLOGY
- arrival_time DRAW_TREE src_node_num
Note that although each link event contains both bandwidth and latency numbers, your algorithms will determine shortest paths using only the link latencies.