Link-State Routing

• Advertisement step
  – Send information about its links to its neighbors (aka link state advertisement or LSA):
    \[\text{[node, seq#, [(nbhr1, linkcost1), (nbhr2, linkcost2), ...]}\]
  – Do it periodically (liveness, recover from lost LSAs)

• Integration
  – If seq# in incoming LSA > seq# in saved LSA for source node:
    update LSA for node with new seq#, neighbor list rebroadcast LSA to neighbors (flooding)
  – Remove saved LSAs if seq# is too far out-of-date
  – Result: Each node discovers current map of the network

• Build routing table
  – Periodically each node runs the same shortest path algorithm over its map (e.g., Dijkstra’s alg)
  – If each node implements computation correctly and each node has the same map, then routing tables will be correct

LSA Flooding

• Periodically originate LSA
• LSA travels each link in each direction
  – Don’t bother with figuring out which link LSA came from
• Termination: each node rebroadcasts LSA exactly once
  – Use sequence number to determine if new, save latest seq
• Multiple opportunities for each node to hear any given LSA
  – Time required: number of links to cross network

Integration Step: Dijkstra’s Algorithm (Example)

Suppose we want to find paths from A to other nodes

![Diagram of Dijkstra's Algorithm example]
Dijkstra’s Shortest Path Algorithm

• Initially
  – nodeset = [all nodes] = set of nodes we haven’t processed
  – spcost = [me:0, all other nodes: ∞] # shortest path cost
  – routes = [me:--, all other nodes: ?] # routing table

• while nodeset isn’t empty:
  – find u, the node in nodeset with smallest spcost
  – remove u from nodeset
  – for v in [u’s neighbors]:
    • d = spcost(u) + cost(u,v) # distance to v via u
    • if d < spcost(v):                 # we found a shorter path!
      – spcost(v) = d
      – routes[v] = routes[u] (or if u == me, enter link from me to v)

Another Example

<table>
<thead>
<tr>
<th>Step</th>
<th>u</th>
<th>Nodeset</th>
<th>spcost</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>[B,C,D,E]</td>
<td>0</td>
<td>×</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>[B,D,E]</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>[B,D]</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>[B]</td>
<td>0</td>
<td>18</td>
</tr>
</tbody>
</table>

Finding shortest paths from A:

LSAs:
A: [A,B,C,D,E]
B: [A,19], [C,7]
C: [A,7], [B,11], [D,43]
D: [A,7], [B,11], [D,15], [E,5]
E: [C,5], [D,133]

Routing Loop in Link-State Protocol

B to D is via A.
Link AD fails.
A’s LSA to B is lost.
A now uses B to get to D.
But B continues to use A.
Routing loop!
Must wait for eventual arrival of correct LSAs to fix loop

Failures

• Problems: Links and switches could fail
  – Advertisements could get lost
  – Routing loop
    • A sequence of nodes on forwarding path that has a cycle (so packets will never reach destination)
  – Dead-end: route does not actually reach destination
  – Loops and dead-ends lead to routes not being valid

• Solution
  – HELLO protocol to detect neighbor liveness
  – Periodic advertisements from nodes
  – Periodic integration at nodes
  – Leads to eventual convergence to correct state (see Chapter 18)
Distance-Vector: Pros, Cons, and Loops

- Simple protocol
- Works well for small networks
- Works only on small networks

Suppose link AC fails. When A discovers failure, it sends E: cost = INFINITY to B.
B advertises E: cost=2 to A. A sets E: cost=3 in its table.
Now suppose link BD fails. B discovers it, then sets E: cost = INFINITY.
Sends info to A. A sets E: cost = INFINITY.

But what if A had advertised to B before B advertised to A?

Fixing “Count to Infinity” with Path Vector Routing

- In addition to (or instead of) reporting costs, advertise the path discovered incrementally by the Bellman-Ford update rule
- Called “path-vector”
- Modify Bellman-Ford update with new rule: a node should ignore any advertised route that contains itself in the advertisement

Path Vector Routing

- For each advertisement, run "integration step"
  - E.g., pick shortest, cheapest, quickest, etc.
- Ignore advertisements with own address in path vector
  - Avoids routing loops that "count to infinity"

Summary

- The network layer implements the “glue” that achieves connectivity
  - Does addressing, forwarding, and routing
- Forwarding entails a routing table lookup; the table is built using routing protocol
- DV protocol: distributes route computation; each node advertises its best routes to neighbors
  - Path-vector: include path, not just cost, in advertisement to avoid “count-to-infinity”
- LS protocol: distributes (floods) neighbor information; centralizes route computation using shortest-path algorithm