6.02 Spring 2011
Lecture #19

- addressing, forwarding, routing
- liveness, advertisements, integration
- distance-vector routing
- routing loops, counting to infinity

**Forwarding**

- Core function is conceptually simple
  - `lookup(dst_addr)` in routing table returns `route` (i.e., `outgoing link`) for packet
  - `enqueue(packet, link_queue)`
  - `send(packet)` along outgoing link
- And do some bookkeeping before enqueue
  - Decrement hop limit (TTL); if 0, discard packet
  - Recalculate checksum (in IP, header checksum)

**Shortest Path Routing**

- Each node wants to find the path with minimum total cost to other nodes
  - We use the term “shortest path” even though we’re interested in min cost (and not min #hops)
- Several possible distributed approaches
  - Vector protocols, esp. distance vector (DV)
  - Link-state protocols (LS)
Routing Table Structure

<table>
<thead>
<tr>
<th>Destination</th>
<th>Link (next-hop)</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ROUTE L1</td>
<td>18</td>
</tr>
<tr>
<td>B</td>
<td>'Self'</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>L1</td>
<td>11</td>
</tr>
<tr>
<td>D</td>
<td>L2</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>L1</td>
<td>16</td>
</tr>
</tbody>
</table>

Table @ node B

Distributed Routing: A Common Plan

- Determining live neighbors
  - Common to both DV and LS protocols
  - HELLO protocol (periodic)
    - Send HELLO packet to each neighbor to let them know who's at the end of their outgoing links
    - Use received HELLO packets to build a list of neighbors containing an information tuple for each link: (timestamp, neighbor addr, link)
    - Repeat periodically. Don't hear anything for a while → link is down, so remove from neighbor list.

- Advertisement step (periodic)
  - Send some information to all neighbors
  - Used to determine connectivity & costs to reachable nodes

- Integration step
  - Compute routing table using info from advertisements
  - Dealing with stale data

Distance-Vector Routing

- DV advertisement
  - Send info from routing table entries: (dest, cost)
  - Initially just (self,0)
- DV integration step [Bellman-Ford]
  - For each (dest,cost) entry in neighbor’s advertisement
    - Account for cost to reach neighbor: (dest,my_cost)
    - my_cost = cost_in_advertisement + link_cost
  - Are we currently sending packets for dest to this neighbor?
    - See if link matches what we have in routing table
    - If so, update cost in routing table to be my_cost
  - Otherwise, is my_cost smaller than existing route?
    - If so, neighbor is offering a better deal! Use it...
    - update routing table so that packets for dest are sent to this neighbor

DV Example: round 1

- Node A: update routes to B, C
- Node B: update routes to A, C, D
- Node C: update routes to A, B, D, E
- Node D: update routes to B, C, E
- Node E: update routes to C, D

\[\text{Subscript indicates node that gave better route}\]
**DV Example: round 2**

- Node A: update routes to B, D, E
- Node B: update routes to A, C
- Node C: no updates
- Node D: update routes to A
- Node E: update routes to B

**DV Example: round 3**

- Node A: no updates
- Node B: no updates
- Node C: no updates
- Node D: no updates
- Node E: no updates

**DV Example: Break a Link**

When link breaks: eliminate routes that use that link.

**DV Example: round 4**

- Node A: update cost to B
- Node B: update routes to A, C, D, E
- Node C: update routes to B
- Node D: no updates
- Node E: update routes to B
Correctness & Performance

- Optimal substructure property fundamental to correctness of both Bellman-Ford and Dijkstra’s shortest path algorithms
  - Suppose shortest path from X to Y goes through Z. Then, the sub-path from X to Z must be a shortest path.
- Proof of Bellman-Ford via induction on number of walks on shortest (min-cost) paths
  - Easy when all costs > 0 and synchronous model (see notes)
  - Harder with distributed async model (not in 6.02)
- How long does it take for distance-vector routing protocol to converge?
  - Time proportional to largest number of hops considering all the min-cost paths

Partitioning the Network

- DV Example: final state
- Node A: no updates
- Node B: no updates
- Node C: no updates
- Node D: no updates
- Node E: no updates

- DV Example: round 5
- Node A: update route to B
- Node B: delete routes to C, D, E
- Node C: update routes to A
- Node D: update routes to A
- Node E: update route to A

6.02 Spring 2011 Lecture 19, Slide #13
6.02 Spring 2011 Lecture 19, Slide #14
6.02 Spring 2011 Lecture 19, Slide #15
6.02 Spring 2011 Lecture 19, Slide #16
DV Example: round 6

Node A: no updates
Node B: no updates
Node C: update costs to A
Node D: update route to A, cost to B
Node E: update routes to A

Counting to Infinity

Nodes C, D, and E each update their costs in response to earlier updates by neighbors. Costs spiral upwards towards self.INFINITY.

Routing Loop!

Suppose E sends a packet to A:
- E forwards to C
- C forwards to E
- ... repeat ...
- Drop packet when TTL is decremented to 0

Eventually all the unreachable nodes are removed from routing table and all routing loops are resolved.