Network Routing I
(The Simple Case Without Failures)

Lecture 20
6.02 Spring 2009
April 22, 2009

• Forwarding and routing
• Distance-vector protocol with Bellman-Ford step
• Link-state protocol with Dijkstra’s shortest-paths

The Problem: Finding Paths

A
C
D
E

B

L0
L1
L2

L0
L1
L2

19
11
15
13
7

How to find a good path (or paths) between any two nodes?

• Addressing (naming nodes)
• Forwarding (what a switch does when packet arrives)
• Routing (building and updating data structures to ensure that forwarding works)

Why is Network Routing Hard?

• Inherently distributed problem
• Information about links and neighbors is local to each node, but we want global reach
• Efficiency: want reasonably good paths, and must find them without huge overhead
• Handling failures and “churn” (next lecture)
• Must tolerate link, switch, and network faults
• Failures and recovery could be arbitrarily timed, messages could be lost, etc.
• Scaling to large size very hard (later courses)
• And on the Internet, many independent, competing organizations must cooperate
• Mobility makes the problem harder

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 book-keeping before enqueue
  • Decrement hop limit (TTL); if 0, discard packet
  • Recalculate checksum (in IP, header checksum)

Routing Table Structure

Table @ B

<table>
<thead>
<tr>
<th>Destination</th>
<th>Link (next-hop)</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ROUTE</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>L0</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>L1</td>
<td>16</td>
</tr>
</tbody>
</table>

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 approaches
  • Vector protocols
  • Link-state protocols
Distributed Routing: A Common Plan

- Determining live neighbors
  - HELLO protocol (periodic)

- Advertisement step (periodic)
  - Send some information to all neighbors

- Integration step
  - Compute routing table using info from advertisements

Distance Vector Routing

- Advertisement: Each node periodically announces a vector of \(<\text{destination}:\text{pathcost}>\) tuples
- Integration: On hearing advertisement, run Bellman-Ford step:
  - if (current cost to dest > cost in advertisement) then update cost, nexthop

Link-State Routing

- HELLO protocol for neighbor liveness

- Advertisement step:
  - Information about its links to its neighbors
  - Neighbors re-send on their links \(\rightarrow\) flooding
  - Result: Each node discovers map of the network

- Integration: Each node runs the same shortest path algorithm over its map
  - If each node implements computation correctly and each node has the same map, then routing tables will be correct

Integration Step: Dijkstra’s alg

- Many algorithms: We’ll study Dijkstra’s

- Key property of shortest paths:
  - Suppose shortest path from X to Y goes through Z. Then, the sub-path from X to Z must be a shortest path. [Why?]

Dijkstra’s Algorithm Example

Suppose we want to find paths from A to other nodes

Link-State Advertisements and Flooding

- Periodically send LSA (Link-State Advertisement) \([\text{seq}, ([\text{nbhr1}, \text{linkcost1}], \text{nbhr2}, \text{linkcost2}], …)]\) to all neighbors
  - If seq > last_heard:
    - save seq, LSA; rebroadcast LSA to neighbors
  - LSAs aren’t reliable messages, so periodic
  - Periodic messages help handle dynamism: state in each node is “soft” and times out if not refreshed