Routing

from Kurose’s slides

Routing protocols

Goal: set routing tables for packet forwarding in hosts and routers, typically based on some optimality criterion.

Questions:

- who determines entries?
- based on what information (hops, delay, cost, …) ?
- how often does it change (hop vs. delay)?
- where is routing information stored?
- algorithm used to compute routes?
Goals for routing algorithms

- scalability
- “safe” interconnection of different organizations
- adopt quickly to changes in topology
- avoid routing loops or at least terminate them quickly
- self-healing, robust
- efficient: can’t use 90% of bandwidth for routing info
- multiple metrics (QOS, price, politics, …) ➞ not yet
- routes should be (near) “optimal”
- can’t have all hosts/networks in single table ➞ hierarchical

Routing algorithms

- centralized vs. decentralized
  - centralized: a central site computes and distributes routes or information to compute routes
  - decentralized: each router sees only local information
- static vs. adaptive
  - static: routing tables change very slowly, often in response to human intervention (German X.25)
  - adaptive: routing tables change with traffic or topology
- intra-domain vs. inter-domain
  - intra-domain: one administration ➞ fewer rules, changes?, not smaller
  - inter-domain: between administrations (autonomous systems) ➞ security, larger geographic reach
Internet Routing

- inter-domain: BGP, about 3,000 AS, 97,000 networks,
- about 32,000 active routes in Merit routing arbiter (Internet Routing Registry)

Link state (LS) routing

- each node knows cost associated with each of its outgoing links:
  - queueing delay on link, instantaneous or time-averaged
  - bandwidth of link
  - cost ($): leased line vs. dial-up
  - notion of desirability
  - simply one “hop” per link
- quasi-centralized: link costs periodically broadcast to all routers
- least-cost path from source to all other nodes ➔ Dijkstra’s shortest-path algorithm
- used in OSPF (+ ISO IS-IS)
Dijkstra’s algorithm

$N$ : set of all nodes to which we know shortest path; initially empty.

d($v$) : distance (cost) of known least cost path from source to $v$

c($i, j$) : cost of link from node $i$ to $j$; $c(i, j) = \infty$ if not directly connected

$p(v)$ : predecessor node (closest neighbor of $v$) along shortest path from source to $v$

After $k$ steps, we know shortest path to nearest $k$ neighbors from source.

Find known nearest neighbor and see if we can reach others from that neighbor by a shorter route than previously. Using nearest ensures that there can be no shorter path.

---

Dijkstra’s algorithm

1. **Initialization**
   
   $N = \{A\}$
   
   for all nodes $v$:
     
     if $v$ adjacent to $A$
       
       then $d(v) = c(A, v)$
     
       else $d(v) = \infty$

2. **loop**

   find node $w$ not in set $N$ such that $d(w)$ is smallest

   add $w$ into $N$

   update $d(v)$ for all $v$ not in $N$:
     
     $d(v) = \min\{d(v), d(w) + c(w, v)\}$

   until all nodes are in $N$
Example of Dijkstra’s algorithm

- example (step 1): \( d(C) \rightarrow d(D) + c(D, C) = 1 + 3 < 5 \)
- for each column, last entry gives immediate neighbor along least-cost path to/from A, and cost to that destination
- worst case running time: \( O(n^2) \) per source node: \( n \) steps, \( n - 1 \) comparisons
Example of Dijkstra’s algorithm (reordered)

<table>
<thead>
<tr>
<th>step</th>
<th>N</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>( \infty, - )</td>
<td>( \infty, - )</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2,A</td>
<td>4,D</td>
<td>2.D</td>
<td>( \infty, - )</td>
<td>( \infty, - )</td>
</tr>
<tr>
<td>2</td>
<td>ADB</td>
<td></td>
<td>3,E</td>
<td>2.D</td>
<td>4,E</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADBE</td>
<td></td>
<td>3,E</td>
<td></td>
<td>4,E</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ADBEC</td>
<td></td>
<td></td>
<td></td>
<td>4,E</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ADBECF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

⇒ no change
⇒ exercise: use asymmetric weights

Distance vector (DV) routing

- asynchronous, iterative distributed computation
  - computation step
  - exchange of routing information step
## Distance table:

<table>
<thead>
<tr>
<th>destination</th>
<th>cost from E to destination via $D^E(.)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
</tr>
</tbody>
</table>

- per-node table with cost to all other nodes via each neighbor
- $D^E(A, B)$ gives cost from E to A, via link to B
- here, $D^E(A, B) = 14$
- distance table immediately gives routing table:
  - minimum cost to each destination (row) is smallest value in row
  - column containing minimum value gives outgoing link for routing to that destination

### Distributed, asynchronous shortest path algorithm

- based on Bellman-Ford algorithm
- used in many routing protocols: BGP, ISO IDRP, Novell IPX, RIP
- run by each node
- exchange (node, distance) information with network neighbor only
- find
  - least cost path to every other node
  - next (neighboring) node on least cost path to destination ⇾ not complete path
DV algorithm (at node X):

1. initialization:
   for all adjacent nodes (column) \( v \):
   \[
   D(\ast, v) = \infty \\
   D(v, v) = c(X, v)
   \]

2. loop
   execute distributed topology update procedure
   \textbf{until} hell freezes over

Topology update algorithm

At node X:

1. wait until X sees link cost change to Y, or receives message from neighbor

2. if \( c(X, Y) \) changes by \( \delta \) (cost to neighbor has changed):
   change all column-Y entries in distance table by \( \delta \)
   if this changes cost of best path to Z, inform neighbors

3. if control message from W shortest path via W to some Z has changed
   \[
   D^X(Z, W) = c(X, W) + \text{new distance W to Z}
   \]
   if cost of best path to Z has changed, inform neighbors
Distance vector routing: example

Distance vector routing: recovery from link failure

- if link XY fails, X and Y set $c(X, Y)$ to $\infty$ and run topology update algorithm
- “good news travels fast, bad news travels slowly”
- looping:
  - inconsistent routing tables: to A, D $\rightarrow$ E, E $\rightarrow$ D.
  - loops disappear eventually
  - performance degradation during looping
  - out-of-order end-end delivery possible
Distance vector routing with link failures

Initial distances:
- B: 6, C: 5, D: 3, E: 1, A: 0

After a link failure:
- New distances:
  - B: 7, C: 7, D: 9, E: 9, A: 1

**Distance vector routing: split horizon algorithm**

- Change topology update algorithm to avoid count-to-infinity problem
- If A routes to Z via B, then A tells B that its distance to Z is \( \infty \)
- Example A — B — Z: B will not route to Z via A if link BZ fails
- Will not always avoid count-to-infinity problem
Distance vector routing: hold down algorithm

- when shortest path cost change:
  - start “hold down” timer
  - advertise new shortest path cost
    as the new cost along previous shortest path route until timer expires:

\[ \text{B: my shortest cost path to A is 1} \]

- will force large costs to propagate quickly
- will not always avoid count-to-infinity problem

Comparison of LS and DV algorithms

- “LS is better”: DV requires iteration with messages being exchanged at each iteration
- “DV is better”: if link costs changes do not affect shortest paths, no messages exchanged
Robustness of LS and DV algorithms

- what happens if router fails, misbehaves, or is sabotaged?
- link state could:
  - report incorrect distance to all neighbors
  - corrupt or lose any LS broadcast messages passing through it
  - report incorrect neighbors
- distance vector could:
  - advertise incorrect shortest distance to any/all destinations ("from me, zero hops to everywhere")
  - report incorrect neighbors

Convergence of LS and DV algorithms

- want to keep network routes stable as often as possible
- distance vector:
  - may iterate many times while converging
  - can suffer from loops and oscillations
  - cannot propagate new information from other routers until it recomputes new routes
- link state
  - requires one broadcast pro node
  - can suffer from oscillations
RIP (Routing Information Protocol)

- RFC 1058 (1988)
- intra-domain only
- distance vector algorithm with split horizon
- metric: hop count (maximum 15 \(\Rightarrow\) limited network size)
- distance vectors exchanged via UDP port 520 every 30 seconds
- `routed` daemon
- RIP-2 (RFC 1388, RFC 1387, 1993): subnet masks, route tag to identify external routes, authentication

RIP-2 packet header

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command (1 or 2)</td>
<td>Version (2)</td>
<td>routing domain (AS)</td>
<td>address family identifier (AFI)</td>
<td>route tag</td>
</tr>
<tr>
<td>IP address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subnet mask</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>next hop IP address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metric (0..15)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OSPF

- open (= non-proprietary) shortest path first (RFC 1247, 1991)
- link state routing using Dijkstra’s algorithm
- reliable flooding with sequence numbers, aging
- two-level hierarchy: backbone and attached areas
- allows level-2 routers to send path cost to level-1 routers
- handles network partitioning (somehow…)
- uses IP packets to communicate

BGP

- inter-domain routing protocol
- uses TCP
- exchanges paths: list of transit AS, networks, properties
### netstat: inspect routing table

**netstat -r**

Routing tables

<table>
<thead>
<tr>
<th>Destination</th>
<th>Gateway</th>
<th>Flags</th>
<th>Refcnt</th>
<th>Use</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>localhost</td>
<td>localhost</td>
<td>UH</td>
<td>3</td>
<td>7013</td>
<td>lo0</td>
</tr>
<tr>
<td>default</td>
<td>gmdbgate</td>
<td>UG</td>
<td>0</td>
<td>107416</td>
<td>le0</td>
</tr>
<tr>
<td>gmd</td>
<td>129.26.216.231</td>
<td>U</td>
<td>0</td>
<td>19</td>
<td>qaa1</td>
</tr>
<tr>
<td>gmd-fokus</td>
<td>atmos</td>
<td>U</td>
<td>33</td>
<td>211181</td>
<td>le0</td>
</tr>
<tr>
<td>fokus-atm</td>
<td>atmos</td>
<td>U</td>
<td>1</td>
<td>561634</td>
<td>qaa0</td>
</tr>
<tr>
<td>bali.de</td>
<td>atmos.bali.de</td>
<td>U</td>
<td>0</td>
<td>1487638</td>
<td>fa0</td>
</tr>
</tbody>
</table>

**netstat -rn**

Routing tables

<table>
<thead>
<tr>
<th>Destination</th>
<th>Gateway</th>
<th>Flags</th>
<th>Refcnt</th>
<th>Use</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>UH</td>
<td>3</td>
<td>7521</td>
<td>lo0</td>
</tr>
<tr>
<td>default</td>
<td>192.35.149.248</td>
<td>UG</td>
<td>0</td>
<td>107452</td>
<td>le0</td>
</tr>
<tr>
<td>129.26.0.0</td>
<td>129.26.216.231</td>
<td>U</td>
<td>0</td>
<td>19</td>
<td>qaa1</td>
</tr>
<tr>
<td>192.35.149.0</td>
<td>192.35.149.117</td>
<td>U</td>
<td>35</td>
<td>215346</td>
<td>le0</td>
</tr>
<tr>
<td>193.175.134.0</td>
<td>193.175.134.117</td>
<td>U</td>
<td>1</td>
<td>561641</td>
<td>qaa0</td>
</tr>
<tr>
<td>194.94.246.0</td>
<td>194.94.246.65</td>
<td>U</td>
<td>0</td>
<td>1487639</td>
<td>fa0</td>
</tr>
</tbody>
</table>

Flags: U = up, G = gateway, D = redirect