Computer Networks

Routing Algorithms

Based on Computer Networking, 4th Edition by Kurose and Ross

Interplay between routing, forwarding

- Routing algorithm
- Local forwarding table
- Header value output link
  - 0100: 3
  - 0111: 2
  - 0111: 2
  - 1001: 1

Value in arriving packet's header

Stan Kurkovsky
Graph abstraction

- Graph: $G = (N,E)$
- $N$ = set of routers = { $u, v, w, x, y, z$ }
- $E$ = set of links = { $(u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)$ }
- Remark: Graph abstraction is useful in other network contexts
- Example: P2P, where $N$ is set of peers and $E$ is set of TCP connections

Graph abstraction: costs

- $c(x,x')$ = cost of link $(x,x')$
  - e.g., $c(w,z) = 5$
- Cost could always be 1, or inversely related to bandwidth, or inversely related to congestion
- Cost of path $(x_1, x_2, x_3, ..., x_p) = c(x_1,x_2) + c(x_2,x_3) + ... + c(x_{p-1},x_p)$
- Question: What’s the least-cost path between $u$ and $z$?
- Routing algorithm: algorithm that finds least-cost path
### Routing Algorithm classification

<table>
<thead>
<tr>
<th>Global or decentralized information?</th>
<th>Static or dynamic?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Global:</strong></td>
<td><strong>Static:</strong></td>
</tr>
<tr>
<td>- all routers have complete topology, link cost info</td>
<td>- routes change slowly over time</td>
</tr>
<tr>
<td>- “link state” algorithms</td>
<td>- routes change more quickly</td>
</tr>
<tr>
<td><strong>Decentralized:</strong></td>
<td>- periodic update</td>
</tr>
<tr>
<td>- router knows physically-connected neighbors, link costs to neighbors</td>
<td>- in response to link cost changes</td>
</tr>
<tr>
<td>- iterative process of computation, exchange of info with neighbors</td>
<td></td>
</tr>
<tr>
<td>- “distance vector” algorithms</td>
<td></td>
</tr>
</tbody>
</table>

### 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 forwarding table for that node
- iterative: after k iterations, know least cost path to k destinations

**Notation:**
- \( c(x,y) \): link cost from node \( x \) to \( y \); \( = \infty \) 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 \( v \)
- \( N' \): set of nodes whose least cost path definitively known
Dijkstra’s Algorithm

1. **Initialization:**
   2. \( N' = \{u\} \)
   3. for all nodes \( v \)
   4. if \( v \) adjacent to \( u \)
   5. then \( D(v) = c(u,v) \)
   6. else \( D(v) = \infty \)

7. **Loop**
   8. find \( w \) not in \( N' \) such that \( D(w) \) is minimum
   9. add \( w \) to \( N' \)
   10. update \( D(v) \) for all \( v \) adjacent to \( w \) and not in \( N' \):
       \[ D(v) = \min(D(v), D(w) + c(w,v)) \]
       /* new cost to \( v \) is either old cost to \( v \) or known
       shortest path cost to \( w \) plus cost from \( w \) to \( v \) */
   11. until all nodes in \( N' \)

---

Dijkstra’s algorithm: example

<table>
<thead>
<tr>
<th>Step</th>
<th>( N' )</th>
<th>( D(v), p(v) )</th>
<th>( D(w), p(w) )</th>
<th>( D(x), p(x) )</th>
<th>( D(y), p(y) )</th>
<th>( D(z), p(z) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( u )</td>
<td>2, ( u )</td>
<td>5, ( u )</td>
<td>1, ( u )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>1</td>
<td>( u, x )</td>
<td>2, ( u )</td>
<td>4, ( x )</td>
<td>2, ( x )</td>
<td>( \infty )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( u, x, y )</td>
<td>2, ( u )</td>
<td>3, ( y )</td>
<td></td>
<td>4, ( y )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( u, x, y, v )</td>
<td>3, ( y )</td>
<td></td>
<td></td>
<td>4, ( y )</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( u, x, y, w, w )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4, ( y )</td>
</tr>
<tr>
<td>5</td>
<td>( u, x, y, w, w, z )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Graph Example](image)
Dijkstra’s algorithm: example (2)

Resulting shortest-path tree from u:

Resulting forwarding table in u:

<table>
<thead>
<tr>
<th>destination</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>(u,v)</td>
</tr>
<tr>
<td>x</td>
<td>(u,x)</td>
</tr>
<tr>
<td>y</td>
<td>(u,x)</td>
</tr>
<tr>
<td>w</td>
<td>(u,x)</td>
</tr>
<tr>
<td>z</td>
<td>(u,x)</td>
</tr>
</tbody>
</table>

Dijkstra’s algorithm, discussion

Algorithm complexity: n nodes
- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: O(n²)
- more efficient implementations possible: O(nlogn)

Oscillations possible:
- e.g., link cost = amount of carried traffic
Distance Vector Algorithm

Bellman-Ford Equation (dynamic programming)
- Define
  \[ d_x(y) := \text{cost of least-cost path from } x \text{ to } y \]
- Then
  \[ d_x(y) = \min \{ c(x,v) + d_y(v) \} \]

where \( \min \) is taken over all neighbors \( v \) of \( x \)

Bellman-Ford example

Clearly, \( d_v(z) = 5 \), \( d_x(z) = 3 \), \( d_w(z) = 3 \)

B-F equation says:

\[ d_{u}(z) = \min \{ c(u,v) + d_{v}(z), \]
\[ c(u,x) + d_{x}(z), \]
\[ c(u,w) + d_{w}(z) \} \]

\[ = \min \{ 2 + 5, \]
\[ 1 + 3, \]
\[ 5 + 3 \} = 4 \]

Node that achieves minimum is next hop in shortest path ➔ forwarding table
**Distance Vector Algorithm**

- \( D_x(y) \) = estimate of least cost from \( x \) to \( y \)
- Distance vector: \( D_x = [D_x(y): y \in N] \)
- Node \( x \) knows cost to each neighbor \( v \): \( c(x,v) \)
- Node \( x \) maintains \( D_x = [D_x(y): y \in N] \)
- Node \( x \) also maintains its neighbors’ distance vectors
  - For each neighbor \( v \), \( x \) maintains \( D_v = [D_v(y): y \in N] \)

**Basic idea:**

- Each node periodically sends its own distance vector estimate to neighbors
- When a node \( x \) receives new DV estimate from neighbor, it updates its own DV using B-F equation:
  \[
  D_x(y) \leftarrow \min_v(c(x,v) + D_v(y)) \quad \text{for each node } y \in N
  \]
- Under minor, natural conditions, the estimate \( D_x(y) \) converge to the actual least cost \( d_{x,y} \)

---

**Distance Vector Algorithm**

**Iterative, asynchronous:** each local iteration caused by:
- local link cost change
- DV update message from neighbor

**Distributed:**
- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

**Each node:**

- **wait** for (change in local link cost of msg from neighbor)
- **recompute** estimates
- if DV to any dest has changed, **notify** neighbors
**Distance Vector: link cost changes**

Link cost changes:
- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors

“Good news travels fast”
- At time $t_1$, $y$ detects the link-cost change, updates its DV, and informs its neighbors.

- At time $t_2$, $z$ receives the update from $y$ and updates its table. It computes a new least cost to $x$ and sends its neighbors its DV.

- At time $t_3$, $y$ receives $z$’s update and updates its distance table. $y$’s least costs do not change and hence $y$ does not send any message to $z$.

**Distance Vector: link cost changes**

Link cost changes:
- good news travels fast
- bad news travels slow - “count to infinity” problem!
- 44 iterations before algorithm stabilizes

Poissoned reverse:
- If $Z$ routes through $Y$ to get to $X$:
  - $Z$ tells $Y$ its (Z’s) distance to $X$ is infinite (so $Y$ won’t route to $X$ via $Z$)
- will this completely solve count to infinity problem?
**Distance Vector: example**

Distance Vector (DV) algorithm:

- **node x table**
- **node y table**
- **node z table**

```
node x table

<table>
<thead>
<tr>
<th>cost to</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>z</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

node y table

<table>
<thead>
<tr>
<th>cost to</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>z</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

node z table

<table>
<thead>
<tr>
<th>cost to</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>z</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
```

**D**(x)(y) = min{c(x,y) + **D**(y)(y), c(x,z) + **D**(z)(y)} = min{2+0 , 7+1} = 2

**D**(x)(z) = min{c(x,y) + **D**(y)(z), c(x,z) + **D**(z)(z)} = min{2+1 , 7+0} = 3

---

**Comparison of LS and DV algorithms**

**Message complexity**
- **LS:** with n nodes, E links, O(nE) msgs sent
- **DV:** exchange between neighbors only
  - convergence time varies

**Speed of Convergence**
- **LS:** O(n^2) algorithm requires O(nE) msgs
  - may have oscillations
- **DV:** convergence time varies
  - may be routing loops
  - count-to-infinity problem

**Robustness:** what happens if router malfunctions?

**LS:**
- node can advertise incorrect link cost
- each node computes only its own table

**DV:**
- DV node can advertise incorrect path cost
- each node’s table used by others
  - error propagate thru network

Stan Kurkovsky
Hierarchical Routing

Our routing study thus far - idealization

- all routers identical; network “flat” → ... not true in practice

scale: with 200 million destinations:
- can’t store all dest’s in routing tables!
- routing table exchange would swamp links!

administrative autonomy
- internet = network of networks
- each network admin may want to control routing in its own network

- aggregate routers into regions, “autonomous systems” (AS)
- routers in same AS run same routing protocol
  - “intra-AS” routing protocol
  - routers in different AS can run different intra-AS routing protocol

Gateway router
- Direct link to router in another AS

Interconnected ASes

- Forwarding table is configured by both intra- and inter-AS routing algorithm
  - Intra-AS sets entries for internal dests
  - Inter-AS & Intra-As sets entries for external dests
**Inter-AS tasks**

- Suppose router in AS1 receives datagram for which dest is outside of AS1
  - Router should forward packet towards one of the gateway routers, but which one?

**AS1 needs:**
1. to learn which dests are reachable through AS2 and which through AS3
2. to propagate this reachability info to all routers in AS1

*Job of inter-AS routing!*