Routing
Class 9
Identifying “Best Path”
Forwarding Packets - a Recap

- Virtual circuits: signaling protocols
  - used to setup, maintain, teardown VC
  - used in ATM, frame-relay, X.25
  - not used in today’s Internet
Forwarding Packets - a Recap

- **Datagram networks**: the Internet model
  - no call setup at network layer
  - routers: no state about end-to-end connections
    - no network-level concept of “connection”
  - packets forwarded using destination host address
    - packets between same source-dest pair may take different paths
Network Layer Function

- Concerned with routing
  - Topology
  - Interoperability of networks
- Boundary to the subnet
  - Transport and above controlled by host
  - Network and below controlled by carriers
Objective of Routing

- Identify the “least cost” path from source to destination
Routing

Routing protocol

Goal: determine “good” path (sequence of routers) through a network from source to dest.

Graph abstraction for routing algorithms:

- graph nodes are routers
- graph edges are physical links
  - link cost: delay, $ cost, or congestion level

“good” path:
- typically means minimum cost path
- other def’s possible

If all paths cost the same, the least-cost path == shortest path
Routing Algorithms

• To find the best route
  - Queues, error rate, delay, bandwidth

• Properties
  - Correctness, simplicity, robustness, optimality, fairness,

• Classes
  - Static / Dynamic
  - Global / Decentralized
Routing Algorithm Classification

Global or decentralized information?

Global:
- all routers have complete topology, link cost info
- "link state" algorithms

Decentralized:
- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- "distance vector" algorithms

Static or dynamic?

Static:
- routes change slowly over time

Dynamic:
- routes change more quickly
  - periodic update
  - in response to link cost changes
Static Routing

• Decisions made off-line
• Examples
  - Shortest path
  - Flow-based
  - Flooding
Dynamic Routing

- Decisions made on-line
- Dynamic tuning to changes in topology, traffic, delays etc.
- Examples
  - Distance vector (RIP)
  - Link state (OSPF)
A Link State Algorithm

NW Topology and all link costs are known
A Link-State Routing Algorithm

Dijkstra’s algorithm

- net topology, link costs known to all nodes
- computes least cost paths from one node (“source”) to all other nodes
- iterative: after k iterations, know least cost path to k destinations

Notation:

- $c(i,j)$: link cost from node $i$ to $j$. cost infinite if not direct neighbors
- $D(v)$: current cost of path from source to destination $v$
- $p(v)$: predecessor node (neighbor of $v$) along the current least-cost path from source to $v$
- $N$: set of nodes whose least cost path definitively known
Dijsktra’s Algorithm

1  *Initialization*: 
2    N = {A} 
3    for all nodes v 
4      if v adjacent to A 
5        then D(v) = c(A,v) 
6        else D(v) = infinity 
7
8  *Loop* 
9    find w not in N such that D(w) is a minimum 
10   add w to N 
11   update D(v) for all v adjacent to w and not in N: 
12     D(v) = min( D(v), D(w) + c(w,v) ) 
13    /* new cost to v is either old cost to v or known 
14       shortest path cost to w plus cost from w to v */ 
15   until all nodes in N

Number of times loop = number of nodes
Dijkstra's algorithm: example

<table>
<thead>
<tr>
<th>Step</th>
<th>start</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Graph with vertices and edges]
Dijkstra’s algorithm, discussion

Algorithm complexity: \( n \) nodes (excluding source)
- each iteration: need to check all nodes, \( w \), not in \( N \)
- \( n^*(n+1)/2 \) comparisons: \( O(n^{**2}) \)
- more efficient implementations possible: \( O(n\log n) \)

Oscillations possible:
- e.g., link cost = amount of carried traffic
Preventing Oscillations

- Oscillations can occur with any algorithm, not just Link-state, that use *congestion* or *delay* as a *cost metric*.

- **Solution 1**
  - Mandate can’t use load as a metric
  - Not acceptable since a goal of routing is to reduce using congested routes

- **Solution 2**
  - Not all routers run the LS algorithm at the same time
  - It seems that routers self-synchronize, even if start off unsynchronized
  - Hence, need to introduce randomization
A Distance Vectoring Algorithm

Only know about immediate-neighbors
Distance Vector Routing Algorithm

**iterative:**
- continues until no nodes exchange info.
- *self-terminating:* no "signal" to stop

**asynchronous:**
- nodes need *not* exchange info/iterate in lock step

**distributed:**
- each node communicates *only* with directly-attached neighbors

**Distance Table data structure**
- each node has its own
- Distance Table Entries:
  - row for each possible destination
  - column for each directly-attached neighbor to node
- example: in node X, for dest. Y via neighbor Z:

\[
X \quad D(Y,Z) = \text{distance from } X \text{ to } Y, \text{ via } Z \text{ as next hop} \\
= c(X,Z) + \min_w \{D^Z(Y,w)\}
\]
Distance Table: example

Each *node* must know the *cost* of the *minimum cost path* of each of its *neighbors* to each *destination*.

![Diagram of a network with nodes A, B, C, D, and E connected by edges with weights 1, 2, 7, 1, and 8 respectively.]

\[ D^E(A,D) = c(E,D) + \min_w \{ D^D(A,w) \} \]
\[ = 2 + 3 = 5 \]

\[ D^E(A,B) = c(E,B) + \min_w \{ D^B(A,w) \} \]
\[ = 8 + 6 = 14 \]

\[ D^E(C,D) = c(E,D) + \min_w \{ D^D(C,w) \} \]
\[ = 2 + 2 = 4 \]

<table>
<thead>
<tr>
<th>cost to destination via node</th>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>
Distance Table ➔ Routing Table

<table>
<thead>
<tr>
<th>Destination</th>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>

Distance table ➔ Routing table

<table>
<thead>
<tr>
<th>Destination</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A,1</td>
</tr>
<tr>
<td>B</td>
<td>D,5</td>
</tr>
<tr>
<td>C</td>
<td>D,4</td>
</tr>
<tr>
<td>D</td>
<td>D,4</td>
</tr>
</tbody>
</table>
Distance Vector Routing: overview

Iterative, asynchronous:
   each local iteration caused by:
   • local link cost change
   • message from neighbor: its least cost path change from neighbor

Distributed:
   • each node notifies neighbors only when its least cost path to any destination changes

Each node:

wait for (change in local link cost of msg from neighbor)

recompute distance table

if least cost path to any dest has changed, notify neighbors
Distance Vector Algorithm:

At all nodes, X:

1. Initialization:
2. for all adjacent nodes v:
3. \( D^X(*,v) = \infty \) /* the * operator means "for all rows" */
4. \( D^X(v,v) = c(X,v) \)
5. for all destinations, y
6. send \( \min_w D^X(y,w) \) to each neighbor /* w over all X's neighbors */
7.
Distance Vector Algorithm (cont.):

8  loop
9    wait (until I see a link cost change to neighbor V
10      or until I receive update from neighbor V)
11
12  if (c(X,V) changes by d)
13     /* change cost to all dest's via neighbor v by d */
14     /* note: d could be positive or negative */
15     for all destinations y:  \(D^X(y,V) = D^X(y,V) + d\)
16
17  else if (update received from V wrt destination Y)
18     /* shortest path from V to some Y has changed */
19     /* V has sent a new value for its min\(_w\)DV(Y,w) */
20     /* call this received new value is "newval" */
21     for the single destination y:  \(D^X(Y,V) = c(X,V) + newval\)
22
23  if we have a new min\(_w\)D^X(Y,w)for any destination Y
24     send new value of min\(_w\)D^X(Y,w) to all neighbors
25
26  forever
Distance Vector Algorithm: example

- New minimum cost
- Old minimum cost
Distance Vector Algorithm: example

\[
\begin{align*}
D^X(Y,Z) &= c(X,Z) + \min_w \{D^Z(Y,w)\} \\
&= 7 + 1 = 8 \\
D^X(Z,Y) &= c(X,Y) + \min_w \{D^Y(Z,w)\} \\
&= 2 + 1 = 3
\end{align*}
\]
Distance Vector Algorithm: example
Distance Vector: link cost *changes*

Link cost changes:
- node *detects local* link cost change
- updates distance table (line 15)
- if cost change in least cost path, notify neighbors (lines 23, 24)
Distance Vector: link cost changes

Link cost changes:
- good news travels fast

algorithm continues

How many iterations before this stops?
Distance Vector: Poisoned 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?

algorithm terminates

c(X,Y change

case 4

t_0   t_1   t_2   t_3   t_4
Comparison of LS and DV algorithms

Message complexity

- **LS**: with n nodes, E links, O(nE) msgs sent & all nodes are told of a change
- **DV**: exchange between *direct* neighbors only
  - convergence time varies

Speed of Convergence

- **LS**: O(n²) algorithm requires O(nE) msgs
  - may have oscillations
- **DV**: convergence time varies
  - may have routing loops
  - count-to-infinity problem

Robustness: what happens if router malfunctions?

- **LS**:
  - node can advertise incorrect cost of 1 *link*
  - each node computes only its *own* table \(\rightarrow\) *robust*

- **DV**:
  - DV node can advertise incorrect *path* cost to all
  - each node’s table used by others
    - error propagates through the network
End of Class 9