Chapter 4: Network Layer

- Introduction (forwarding and routing)
- Review of queueing theory
- Routing algorithms
  - Link state, Distance Vector
- Router design and operation
- IP: Internet Protocol
  - IPv4 (datagram format, addressing, ICMP, NAT)
  - IPv6
- Routing in the Internet
  - Autonomous Systems
  - Routing protocols (RIP, OSPF, BGP)

Interplay between routing, forwarding
Routing Algorithm classification

Static or dynamic?

Static:
- routes change slowly over time
- manual configuration

Dynamic:
- routes change more quickly
- periodic updates in response to link cost changes

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

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: Real network routing algorithms typically use DIRECTED graphs.
Graph abstraction: costs

- $c(x,x') = \text{cost of link } (x,x')$
  - e.g., $c(w,z) = 5$
- cost could be
  - 1 (hop count)
  - inversely related to bandwidth,
  - inversely related to congestion,
  - count of packets in queue,
  - some combination of above

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 from source to destination.

Principle of Optimality

- If node B lies on an optimal path from node A to node C, then an optimal path from node B to node C also lies along the same path. Why does this property hold?
Principal of Optimality

The result is that the set of optimal routes from all sources to a given destination form a sink tree rooted at the destination.

In general, is the sink tree unique?

A Link-State Routing Algorithm

Dijkstra’s algorithm

- In-memory graph of network
- network topology, link costs known to all nodes
  - accomplished by flooding “link state advertisements”
  - 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 \( v \)
- \( M \): set of nodes whose least cost path definitively known
Basic Idea

- Find the shortest paths from a given source node to all other nodes
- Proceeds in stages - build the sink tree one branch at a time.
- By the kth stage, the shortest paths to the k nodes closest to the source have been determined (and added to set M)
- At (k + 1)st stage, that node not already in M that has the shortest path from the source is added to M
- As nodes are added to M, their path from the source is defined.

Dijsktra’s Algorithm

1  Initialization:
2    M= \{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
8  Loop
9    find w not in M such that D(w) is a minimum
10   add w to M
11   update D(v) for all v adjacent to w and not in M:
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 M
### Dijkstra’s algorithm: example

<table>
<thead>
<tr>
<th>Step</th>
<th>M</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></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>
</tr>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
</tbody>
</table>

### Dijkstra’s algorithm: result

**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>
Bellman-Ford Algorithm

- Proceeds in stages.
- Find the shortest paths from a given source subject to the constraint that the paths contain at most one link
- Next, find all that contain two links.
- ...and so on.

Bellman-Ford

- Variables:
  - $c(x,y) =$ link cost
  - $h =$ max number of links in path at current stage
  - $D^h(y) =$ cost of least-cost path from source to node $y$ under the constraint of no more than $h$ links
**Algorithm**

- Let \( s \) be the source node
- Initialize \( D^0(n) = \infty \) for all \( n \neq s \)
- For each successive \( h > 0 \),
  \[ D^{h+1}(n) = \min_j [D^h(j) + c(j,n)] \]

**When does the algorithm halt?**

**Bellman-Ford Example**

Source = \( u \)

<table>
<thead>
<tr>
<th>( h )</th>
<th>( D^h(v) )</th>
<th>( D^h(w) )</th>
<th>( D^h(x) )</th>
<th>( D^h(y) )</th>
<th>( D^h(z) )</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison

- Both Dijkstra’s algorithm and the Bellman-Ford algorithm converge to shortest path solutions under static conditions.
- Complexity?

Suitability for distributed implementation?

Distance Vector Protocol

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

Bellman-Ford Equation

Define
\[ D_x(y) := \text{cost of least-cost path from source } x \text{ to } y \]

Then
\[
D_x(y) = \min \{ c(x,v) + D_v(y) \}
\]

where min is taken over all neighbors v of x

Distance vector protocol

Basic idea:

- From time to time, each node sends its own distance vector estimate to neighbors
- Asynchronous
- 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 normal conditions, the estimates \( D_x(y) \) 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 or msg from neighbor)
- recompute estimates
- if DV to any dest has changed, notify neighbors

---

**Distance Vector Example**

(a) 

(b) 

Vectors received from J’s four neighbors

CSE 422 - Phillips
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

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 propagates through network

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

---

**DV - Count to Infinity**

- So in this example, the Bellman-Ford algorithm will converge for each router, they will have entries for each other.
- B will know that it can get to C at a cost of 1, and A will know that it can get to C via B at a cost of 2.

Source: https://www.geeksforgeeks.org/computer-network-route-poisoning-counting-infinity-problem/
DV – Count to Infinity (cont)

- If the link between B and C is disconnected, then B will know that it can no longer get to C via that link and will remove it from its table.
- Before it can send any updates it's possible that it will receive an update from A which will be advertising that it can get to C at a cost of 2.
- B can get to A at a cost of 1, so it will update a route to C via A at a cost of 3.
- A will then receive updates from B later and update its cost to 4.
- They will then go on feeding each other bad information toward infinity.

CSE 422 - Phillips

Network Layer