Lecture (10)
Routing in Switched Networks (III)

Dr. Ahmed ElShafee

Agenda

- Routing protocols
  - Fixed
  - Flooding
  - **Random**
  - **Adaptive**
- **ARPANET Routing Strategies**
- **Least cost algorithms**
  - Dijkstra's algorithm
  - Bellman-Ford algorithm

Least cost algorithms

- Virtually all packet-switching networks and all internets base their routing decision on some form of least-cost criterion.

Cost can be calculated based on:

1. if the criterion is to minimize the number of hops, each link has a value of 1.
2. the link value is inversely proportional to the link capacity, proportional to the current load on the link, or some combination.

- It dosn't matter how we calculate cost, these link or hop costs are used as input to a least-cost routing algorithm,

Link cost and node cost

- "Given a network of nodes connected by bidirectional links, where each link has a cost associated with it in each direction, define the cost of a path between two nodes as the sum of the costs of the links traversed.
- For each pair of nodes, find a path with the least cost."

- Link cost may differ in its two directions.
- This would be true, for example, if the cost of a link equaled the length of the queue of packets awaiting transmission from each of the two nodes on the link.

Common used least cost algorithms

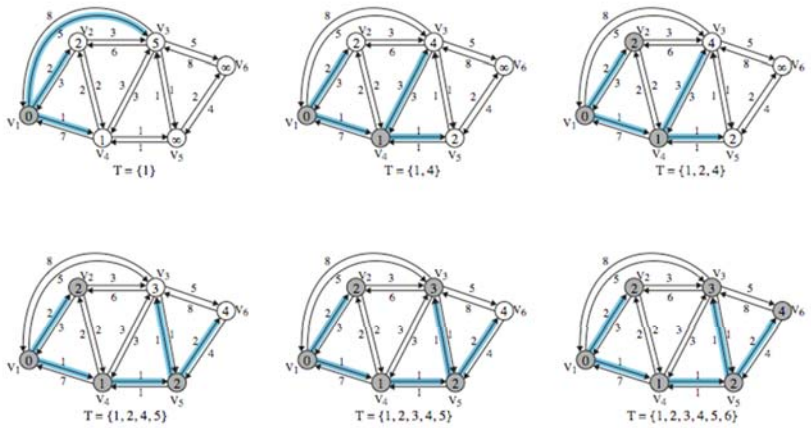
Dijkstra's algorithm

Bellman-Ford algorithm

Dijkstra's algorithm

- finds shortest paths from given source node s to all other nodes
- by developing paths in order of increasing path length
- algorithm runs in stages (next slide)
  - each time adding node with next shortest path
- algorithm terminates when all nodes processed by algorithm (in set T)

- Step 1 [Initialization]
  - T = {s} Set of nodes so far incorporated
  - L(n) = w(s, n) for n ≠ s
  - initial path costs to neighboring nodes are simply link costs
- Step 2 [Get Next Node]
  - find neighboring node not in T with least-cost path from s
  - incorporate node into T
  - also incorporate the edge that is incident on that node and a node in T that contributes to the path
- Step 3 [Update Least-Cost Paths]
  - L(n) = min[L(n), L(x) + w(x, n)] for all n ∉ T
  - f latter term is minimum, path from s to n is path from s to x concatenated with edge from x to n

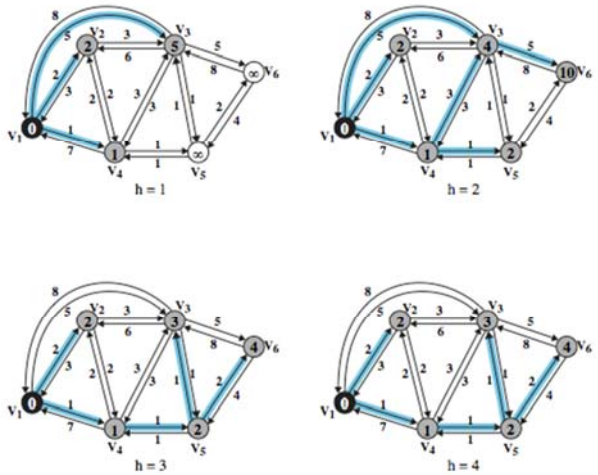


Iter	T	L(2)	Path	L(3)	Path	L(4)	Path	L(5)	Path	L(6)	Path
1	{1}	2	1–2	5	1-3	1	1–4	∞	-	∞	-
2	{1,4}	2	1–2	4	1-4-3	1	1–4	2	1-4–5	∞	-
3	{1, 2, 4}	2	1–2	4	1-4-3	1	1–4	2	1-4–5	∞	-
4	{1, 2, 4, 5}	2	1–2	3	1-4-5–3	1	1–4	2	1-4–5	4	1-4-5–6
5	{1, 2, 3, 4, 5}	2	1–2	3	1-4-5–3	1	1–4	2	1-4–5	4	1-4-5–6
6	{1, 2, 3, 4, 5, 6}	2	1-2	3	1-4-5-3	1	1-4	2	1-4–5	4	1-4-5-6

Bellman-Ford Algorithm

- find shortest paths from given node subject to constraint that paths contain at most one link
- find the shortest paths with a constraint of paths of at most two links and so on

- step 1 [Initialization]
  - $L_0(n) = \infty$, for all $n \neq s$
  - $L_h(s) = 0$, for all h
- step 2 [Update]
  - for each successive $h \geq 0$
    - for each $n \neq s$, compute: $L_{h+1}(n)=^{\min}_{j}[L_h(j)+w(j,n)]$
  - connect n with predecessor node j that gives min
  - eliminate any connection of n with different predecessor node formed during an earlier iteration
  - path from s to n terminates with link from j to n



h	$L_h(2)$	Path	$L_h(3)$	Path	$L_h(4)$	Path	$L_h(5)$	Path	$L_h(6)$	Path
0	∞	-	∞	-	∞	-	∞	-	∞	-
1	2	1-2	5	1-3	1	1-4	∞	-	∞	-
2	2	1-2	4	1-4-3	1	1-4	2	1-4-5	10	1-3-6
3	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6
4	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

Comparison

Bellman-Ford algorithm

- the calculation for node *n involves knowledge of the link* cost to all neighboring nodes to node [*n*] *i.e., w(j, n)] plus the total path cost to* each of those neighboring nodes from a particular source node *s [i.e., Lh(j)].*
- *Each* node can maintain a set of costs and associated paths for every other node in the network and exchange this information with its direct neighbors from time to time.
- Each node can therefore use the expression in step 2 of the Bellman-Ford algorithm, based only on information from its neighbors and knowledge of its link costs, to update its costs and paths.

Dijkstra's algorithm

- Step 3 appears to require that each node must have complete topological information about the network.
- That is, each node must know the link costs of all links in the network.
- Thus, for this algorithm, information must be exchanged with all other nodes.

Evaluation

- In general, evaluation of the relative merits of the two algorithms should consider the
  - processing time of the algorithms and
  - the amount of information that must be collected from other nodes in the network or internet.
- The evaluation will depend on the implementation approach and the specific implementation.

A final point:

- Both algorithms are known to converge under static conditions of topology, link costs and will converge to the same solution.
- If the link costs change over time, the algorithm will attempt to catch up with these changes.
- However, if the link cost depends on traffic, which in turn depends on the routes chosen, then a feedback condition exists, and instabilities may result.

Thanks,…