3.4.2 Link State Routing

- Link state routing has a different philosophy from that of distance vector routing.
- In link state routing, if each node in the domain has the entire topology of the domain the list of nodes and links, how they are connected including the type, cost (metric), and condition of the links (up or down)-the node can use Dijkstra's algorithm to build a routing table Figure 3.4.2.1 shows the concept

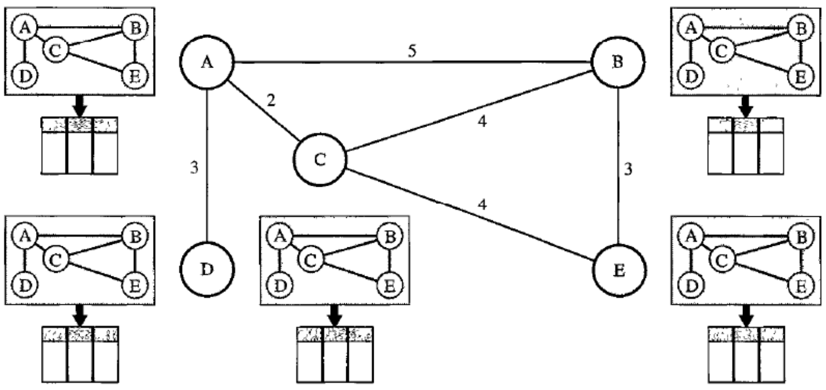

Figure 3.4.2.1 Concept of link state routing

- The figure shows a simple domain with five nodes.
- Each node uses the same topology to create a routing table, but the routing table for each node is unique because the calculations are based on different interpretations of the topology. This is analogous to a city map. While each person may have the same map, each needs to take a different route to reach her specific destination.
- The topology must be dynamic, representing the latest state of each node and each link.
- If there are changes in any point in the network (a link is down, for example), the topology must be updated for each node.
- How can a common topology be dynamic and stored in each node? No node can know the topology at the beginning or after a change somewhere in the network.
- Link state routing is based on the assumption that, although the global knowledge about the topology is not clear, each node has partial knowledge: it knows the state (type, condition, and cost) of its links. In other words, the whole topology can be compiled from the partial knowledge of each node.
- Figure 3.4.2.2 shows the same domain as in Figure 3.4.2.1, indicating the part of the knowledge belonging to each node.

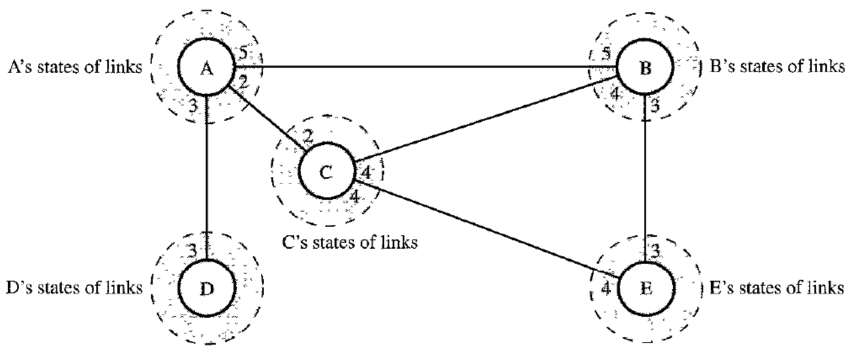

Figure 3.4.2.2 Link state knowledge

- Node A knows that it is connected to node B with metric 5, to node C with metric 2, and to node D with metric 3. Node C knows that it is connected to node A with metric 2, to node B with metric 4, and to node E with metric 4. Node D knows that it is connected only to node A with metric 3. And so on. Although there is an overlap in the knowledge, the overlap guarantees the creation of a common topology-a picture of the whole domain for each node.

Building Routing Tables

- In link state routing, four sets of actions are required to ensure that each node has the routing table showing the least-cost node to every other node.
    1. Creation of the states of the links by each node, called the link state packet (LSP).
    2. Dissemination of LSPs to every other router, called flooding, in an efficient and reliable way.
    3. Formation of a shortest path tree for each node.
    4. Calculation of a routing table based on the shortest path tree.

Creation of Link State Packet (LSP)

- A link state packet can carry a large amount of information. For the moment, however, we assume that it carries a minimum amount of data:
    - the node identity,
    - the list of links,
    - a sequence number,
    - and age.
- The first two, node identity and the list of links, are needed to make the topology.
- The third, sequence number, facilitates flooding and distinguishes new LSPs from old ones.
- The fourth, age, prevents old LSPs from remaining in the domain for a long time.
- LSPs are generated on two occasions:
    1. When there is a change in the topology of the domain. Triggering of LSP dissemination is the main way of quickly informing any node in the domain to update its topology.
    2. On a periodic basis. The period in this case is much longer compared to distance vector routing. As a matter of fact, there is no actual need for this type of LSP dissemination. It is done to ensure that old information is removed from the domain. The timer set for periodic dissemination is normally in the range of 60 min or 2 h based on the implementation. A longer period ensures that flooding does not create too much traffic on the network.

Flooding of LSPs

- After a node has prepared an LSP, it must be disseminated to all other nodes, not only to its neighbors. The process is called flooding and based on the following:
    1. The creating node sends a copy of the LSP out of each interface.

2. A node that receives an LSP compares it with the copy it may already have. If the newly arrived LSP is older than the one it has (found by checking the sequence number), it discards the LSP. If it is newer, the node does the following:
    a. It discards the old LSP and keeps the new one.
    b. It sends a copy of it out of each interface except the one from which the packet arrived. This guarantees that flooding stops somewhere in the domain (where a node has only one interface).

Formation of Shortest Path Tree: Dijkstra Algorithm

- After receiving all LSPs, each node will have a copy of the whole topology. However, the topology is not sufficient to find the shortest path to every other node; a shortest path tree is needed.
- A tree is a graph of nodes and links; one node is called the root. All other nodes can be reached from the root through only one single route.
- A shortest path tree is a tree in which the path between the root and every other node is the shortest. What we need for each node is a shortest path tree with that node as the root.
- The Dijkstra algorithm creates a shortest path tree from a graph. The algorithm divides the nodes into two sets: tentative and permanent. It finds the neighbors of a current node, makes them tentative, examines them, and if they pass the criteria, makes them permanent.
- We can informally define the algorithm by using the flowchart in Figure 3.4.2.3.

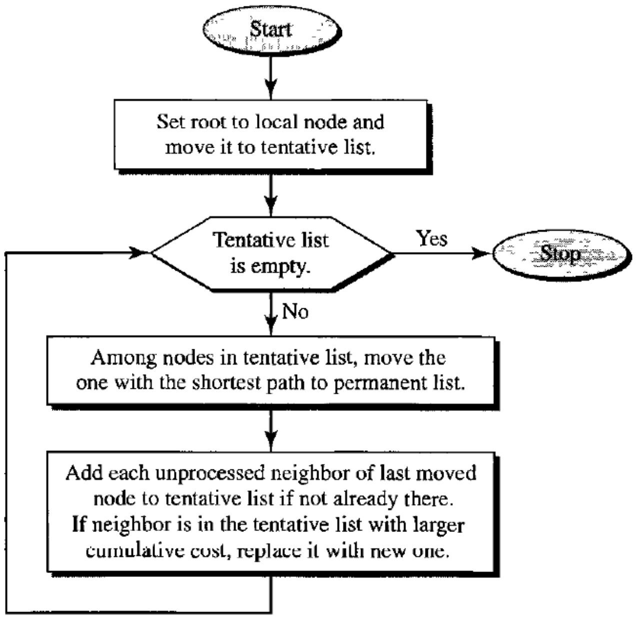
Start
Set root to local node and move it to tentative list.
Tentative list is empty.
Yes
Stop
No
Among nodes in tentative list, move the one with the shortest path to permanent list.
Add each unprocessed neighbor of last moved node to tentative list if not already there. If neighbor is in the tentative list with larger cumulative cost, replace it with new one.

Figure 3.4.2.3 Dijkstra Algorithm

- Let us apply the algorithm to node A of our sample graph in Figure 3.4.2.4. To find the shortest path in each step, we need the cumulative cost from the root to each node, which is shown next to the node.

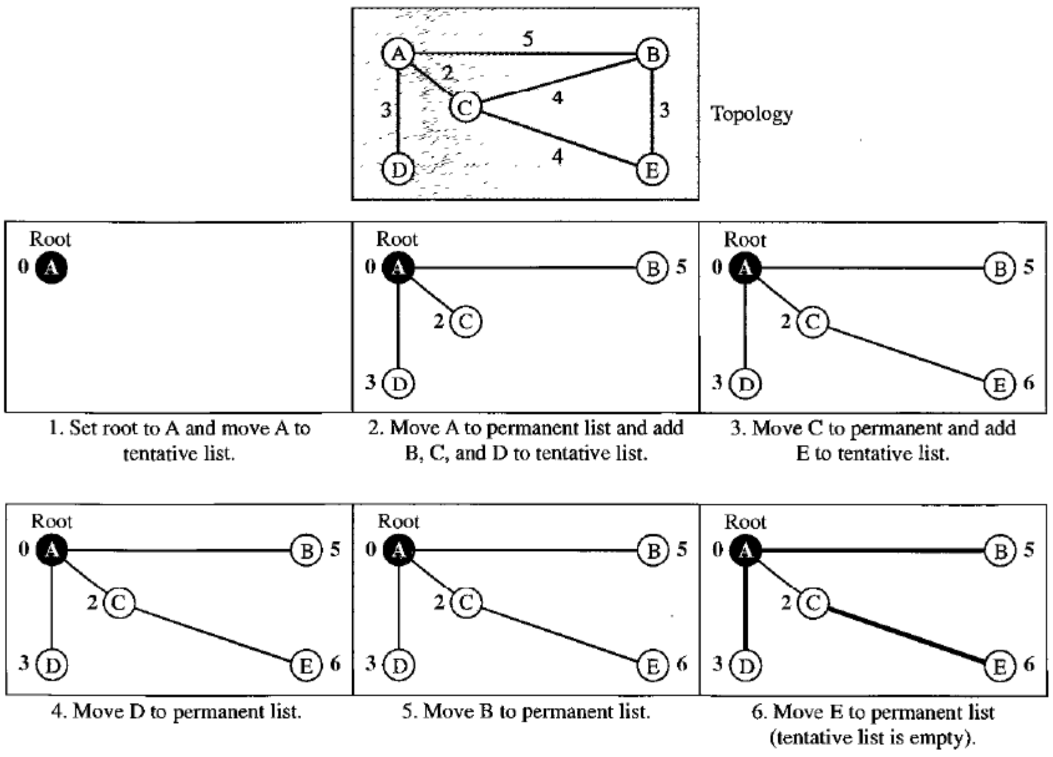
Topology
5
2
4
3
3
4
Root
1. Set root to A and move A to tentative list.
2. Move A to permanent list and add B, C, and D to tentative list.
3. Move C to permanent and add E to tentative list.
4. Move D to permanent list.
5. Move B to permanent list.
6. Move E to permanent list (tentative list is empty).

Figure 3.4.2.4 Example of formation of shortest path tree

- The following shows the steps. At the end of each step, we show the permanent (filled circles) and the tentative (open circles) nodes and lists with the cumulative costs.
  1. We make node A the root of the tree and move it to the tentative list. Our two lists are

     Permanent list: empty

     Tentative list: A(O)
  2. Node A has the shortest cumulative cost from all nodes in the tentative list. We move A to the permanent list and add all neighbors of A to the tentative list. Our new lists are

     Permanent list: A(O)

     Tentative list: B(5), C(2), D(3)
  3. Node C has the shortest cumulative cost from all nodes in the tentative list. We move C to the permanent list. Node C has three neighbors, but node A is already processed, which makes the unprocessed neighbors just B and E. However, B is already in the tentative list with a cumulative cost of 5. Node A could also reach node B through C with a cumulative cost of 6 Since 5 is less than 6, we keep node B with a cumulative cost of 5 in the tentative list and do not replace it. Our new lists are

     Permanent list· A(Q), C(2)

     Tentative list· B(5), D(3), E(6)
  4. Node D has the shortest cumulative cost of all the nodes in the tentative list. We move D to the permanent list Node D has no unprocessed neighbor to be added to the tentative list. Our new lists are

     Permanent list· A(Q), C(2), D(3)

Tentative list: B(5), E(6)

5. Node B has the shortest cumulative cost of all the nodes in the tentative list. We move B to the permanent list. We need to add all unprocessed neighbors of B to the tentative list (this is just node E). However, E(6) is already in the list with a smaller cumulative cost. The cumulative cost to node E, as the neighbor of B, is 8. We keep node E(6) in the tentative list. Our new lists are

   Permanent list: A(O), B(S), C(2), D(3)

   Tentative list: E(6)

6. Node E has the shortest cumulative cost from all nodes in the tentative list. We move E to the permanent list. Node E has no neighbor. Now the tentative list is empty. We stop; our shortest path tree is ready. The final lists are

   Permanent list: A(O), B(S), C(2), D(3), E(6)

   Tentative list: empty

Calculation of Routing Table from Shortest Path Tree

- Each node uses the shortest path tree protocol to construct its routing table.
- The routing table shows the cost of reaching each node from the root.
- Table 3.4.2.1 shows the routing table for node A.

Node	*Cost*	*Next Router*
A	0	—
B	5	—
C	2	—
D	3	—
E	6	C

Table 3.4.2.1 Routing Table for Node A

- Compare Table 3.4.2.1 with the one in Figure 3.4.1. Both distance vector routing and link state routing end up with the same routing table for node A.