Routing Algorithms

- The main function of NL (Network Layer) is routing packets from the source machine to the destination machine.
- The algorithms that choose the routes and the data structures that they use are a major area of network layer design.
- The routing algorithm is that part of the NL software responsible for deciding which output line an incoming packet should be transmitted on.

- There are two processes inside router:

a) One of them handles each packet as it arrives, looking up the outgoing line to use for it in the routing table. This process is forwarding.

b) The other process is responsible for filling in and updating the routing tables. That is where the routing algorithm comes into play. This process is routing.

- Regardless of whether routes are chosen independently for each packet or only when new connections are established, certain properties are desirable in a routing algorithm:
- Correctness: The routing should be done properly and correctly so that the packets may reach their proper destination.
- Simplicity: The routing should be done in a simple manner so that the overhead is as low as possible. With increasing complexity of the routing algorithms the overhead also increases.

- Robustness: The routing algorithm should be able to cope with changes in topology and traffic without requiring all jobs in all hosts to be aborted and the network to be rebooted every time some router crashes.
- Stability: The routing algorithms should be stable under all possible circumstances.A stable algorithm reaches equilibrium and stays there.
- Fairness: Every node connected to the network should get a fair chance of transmitting their packets. This is generally done on a first come first serve basis.
- Optimality: The routing algorithms should be optimal in terms of throughput and minimizing mean packet delays.
- Fairness and optimality are often contradictory goals.

Routing Algorithm Classification

- Adaptive Algorithms
    - Centralized
    - Isolated
    - Distributed
- Non Adaptive algorithms
    - Flooding
    - Random Walk
    - Static routing using shortest path algorithms

Adaptive Routing Algorithms

Adaptive Routing - based on current measurements of traffic and/or topology.

Adaptive Routing Algorithms change their routing decisions to reflect changes in the topology and in traffic as well. These get their routing information from adjacent routers or from all routers. The optimization parameters are the distance, number of hops and estimated transit time. This can be further classified as follows:

1. Centralized
2. Isolated
3. Distributed

Centralized:

- In this type some central node in the network gets entire information about the network topology, about the traffic and about other nodes.
- This then transmits the information to the respective routers.
- The advantage is that only one node is required to keep the information.
- The disadvantage is that if the central node goes down the entire network is down, i.e. single point of failure.

Isolated

In this method, the node decides the routing without seeking information from other nodes. The sending node does not know about the status of a particular link. The disadvantage is that the packet may be send through a congested route resulting in a delay.

Some examples of this type of algorithm for routing are:

- **Hot Potato:** When a packet comes to a node, it tries to get rid of it as fast as it can, by putting it on the shortest output queue without regard to where that link leads.

  A variation of this algorithm is to combine static routing with the hot potato algorithm. When a packet arrives, the routing algorithm takes into account both the static weights of the links and the queue lengths.

Backward Learning

- In this method the routing tables at each node gets modified by information from the incoming packets.
- One way to implement backward learning is to include the identity of the source node in each packet, together with a hop counter that is incremented on each hop.
- When a node receives a packet in a particular line, it notes down the number of hops it has taken to reach it from the source node. If the previous value of hop count stored in the node is better than the current one then nothing is done but if the current value is better then the value is updated for future use.
- The problem with this is that when the best route goes down then it cannot recall the second best route to a particular node. Hence all the nodes have to forget the stored informations periodically and start all over again.

Delta Routing

- Delta routing is a hybrid of the centralized and isolated routing algorithms.
- Here each node computes the cost of each line (i.e some functions of the delay, queue length, utilization, bandwidth etc) and periodically sends a packet to the central node giving it these values which then computes the k best paths from node i to node j.
- Let Cij1 be the cost of the best i-j path, Cij2 the cost of the next best path and so on.
- If Cijn - Cij1 < delta, (Cijn - cost of n'th best i-j path, delta is some constant) then path n is regarded equivalent to the best i-j path since their cost differ by so little.
- When delta -> 0 this algorithm becomes centralized routing and when delta -> infinity all the paths become equivalent.

Distributed

- In this the node receives information from its neighboring nodes and then takes the decision about which way to send the packet.
- The disadvantage is that if in between the interval it receives information and sends the packet something changes then the packet may be delayed.

Non-Adaptive Routing Algorithm

These algorithms do not base their routing decisions on measurements and estimates of the current traffic and topology.

Instead the route to be taken in going from one node to the other is computed in advance, off-line, and downloaded to the routers when the network is booted. This is also known as static routing.

This can be further classified as:

1. Flooding
2. Random Walk
3. Static routing using shortest path algorithms

Flooding:

adapts the technique in which every incoming packet is sent on every outgoing line except the one on which it arrived. One problem with this method is that packets may go in a loop. As a result of this a node may receive several copies of a particular packet which is undesirable.

- Flooding is not practical for general kinds of applications. But in cases where high degree of robustness is desired such as in military applications, flooding is of great help.

Some techniques adapted to overcome looping are as follows:

- **Sequence Numbers:** Every packet is given a sequence number. When a node receives the packet it sees its source address and sequence number. If the node finds that it has sent the same packet earlier then it will not transmit the packet and will just discard it.
- **Hop Count:** Every packet has a hop count associated with it. This is decremented(or incremented) by one by each node which sees it. When the hop count becomes zero(or a maximum possible value) the packet is dropped.
- **Spanning Tree:** The packet is sent only on those links that lead to the destination by constructing a spanning tree routed at the source. This avoids loops in transmission but is possible only when all the intermediate nodes have knowledge of the network topology.

- A variation of flooding that is slightly more practical is selective flooding.
- In this algorithm the routers do not send every incoming packet out on every line, only on those lines that are going approximately in the right direction.

Random Walk:

- In this method a packet is sent by the node to one of its neighbors randomly. This algorithm is highly robust. When the network is highly interconnected, this algorithm has the property of making excellent use of alternative routes. It is usually implemented by sending the packet onto the least queued link.

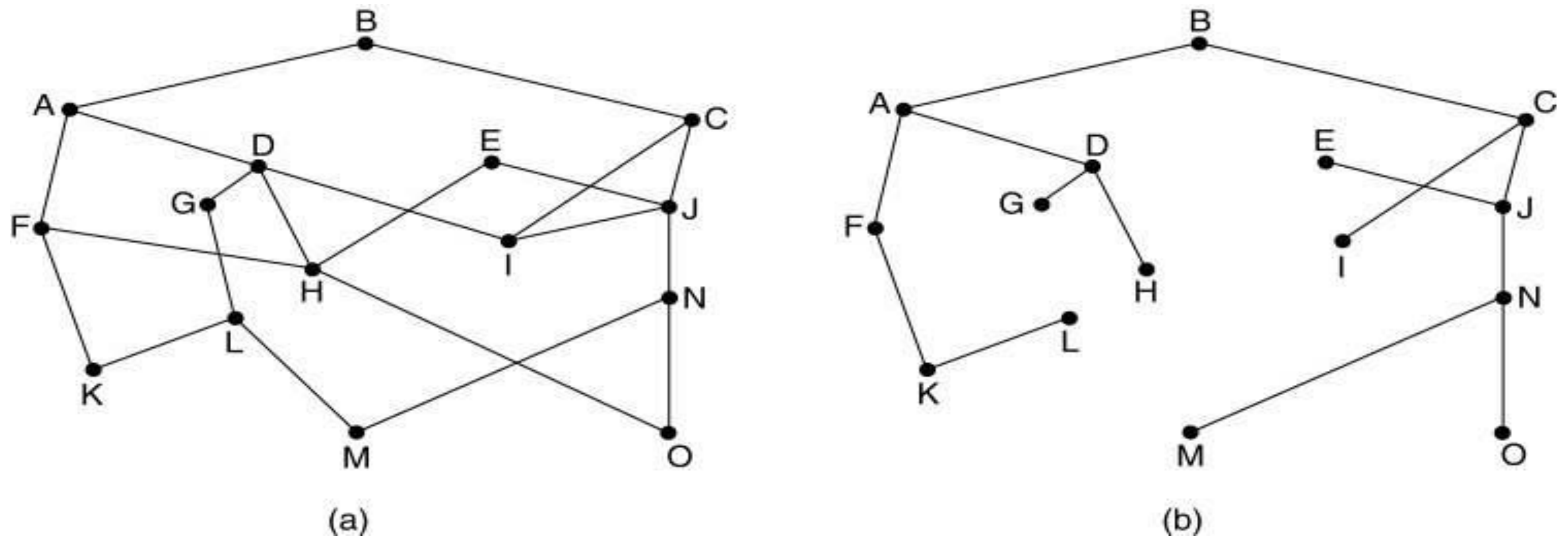
Multipath Routing

- In many networks however there are several paths between pairs of nodes that are almost equally good. Sometimes in order to improve the performance multiple paths between single pair of nodes are used. This technique is called multipath routing or bifurcated routing.
- In this each node maintains a table with one row for each possible destination node. A row gives the best, second best, third best, etc outgoing line for that destination, together with a relative weight. Before forwarding a packet, the node generates a random number and then chooses among the alternatives, using the weights as probabilities. The tables are worked out manually and loaded into the nodes before the network is brought up and not changed thereafter.

The Optimality Principle

- One can make a general statement about optimal routes without regard to network topology or traffic.
- This statement is known as the optimality principle.
- It states that if router J is on the optimal path from router I to router K, then the optimal path from J to K also falls along the same route.

- As a direct consequence of the optimality principle, we can see that the set of optimal routes from all sources to a given destination form a tree rooted at the destination.
- Such a tree is called a sink tree.
- The goal of all routing algorithms is to discover and use the sink trees for all routers



(a) A subnet. (b) A sink tree for router B.

Shortest Path Routing

- The idea is to build a graph of the subnet, with each node of the graph representing a router and each arc of the graph representing a communication line or link.
- To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.
- A network is represented as a graph, with its terminals as nodes and the links as edges. A 'length' is associated with each edge, which represents the cost of using the link for transmission. Lower the cost, more suitable is the link. The cost is determined depending upon the criteria to be optimized.

Some of the important ways of determining the cost are:

- **Minimum number of hops:** If each link is given a unit cost, the shortest path is the one with minimum number of hops. Such a route is easily obtained by a breadth first search method. This is easy to implement but ignores load, link capacity etc.
- **Transmission and Propagation Delays:** If the cost is fixed as a function of transmission and propagation delays, it will reflect the link capacities and the geographical distances. However these costs are essentially static and do not consider the varying load conditions.
- **Queuing Delays:** If the cost of a link is determined through its queuing delays, it takes care of the varying load conditions, but not of the propagation delays.

- Many other metrics besides hops and physical distance are also possible.
- For example, each arc could be labelled with the mean queuing and transmission delay for some standard test packet as determined by hourly test runs.
- With this graph labelling, the shortest path is the fastest path rather than the path with the fewest arcs or kilometres.

- In the general case, the labels on the arcs could be computed as a function of the distance, bandwidth, average traffic, communication cost, mean queue length, measured delay, and other factors.
- By changing the weighting function, the algorithm would then compute the “shortest” path measured according to any one of a number of criteria or to a combination of criteria.

- Let the network be represented by graph G (V, E) and let the number of nodes be 'N'.
- A node has zero cost w.r.t itself.
- Further, all the links are assumed to be symmetric, i.e. if $d_{i,j}$ = cost of link from node i to node j, then $d_{i,j} = d_{j,i}$.
- The graph is assumed to be complete. If there exists no edge between two nodes, then a link of infinite cost is assumed.

Bellman-Ford Algorithm

- This algorithm iterates on the number of edges in a path to obtain the shortest path. It uses optimality principle. Since the number of hops possible is limited (cycles are implicitly not allowed), the algorithm terminates giving the shortest path.
- **Notation:**
  $d_{i,j}$ = Length of path between nodes i and j, indicating the cost of the link.
  h = Number of hops.
  D[i,h] = Shortest path length from node i to node 1, with upto 'h' hops.
  D[1,h] = 0 for all h .

Algorithm :

Initial condition : D[i, 0] = infinity, for all i (i != 1)

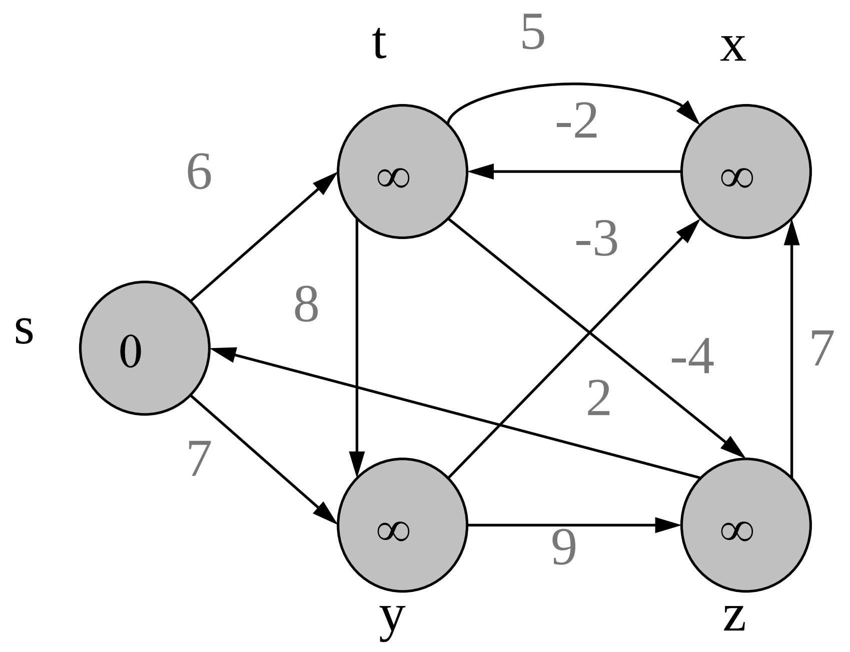
- Iteration : D[i, h+1] = min { $d_{i,j}$ + D[j,h] } over all values of j .
- Termination : The algorithm terminates when

  D[i, h] = D [i, h+1] for all i .

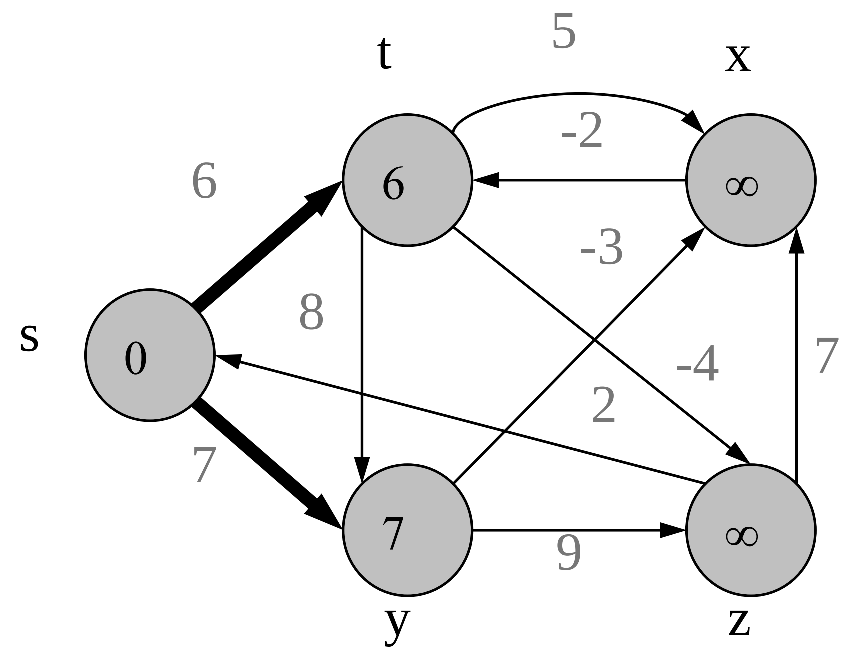
Principle:
For zero hops, the minimum length path has length of infinity, for every node.

- For one hop the shortest-path length associated with a node is equal to the length of the edge between that node and node 1.
- Increment the number of hops allowed, (from h to h+1) and find out whether a shorter path exists through each of the other nodes. If it exists, say through node 'j', then its length must be the sum of the lengths between these two nodes (i.e. $d_{i,j}$) and the shortest path between j and 1 obtainable in upto h paths.
- If such a path doesn't exist, then the path length remains the same. The algorithm is guaranteed to terminate, since there are utmost N nodes, and so N-1 paths. It has time complexity of O (N^3) .

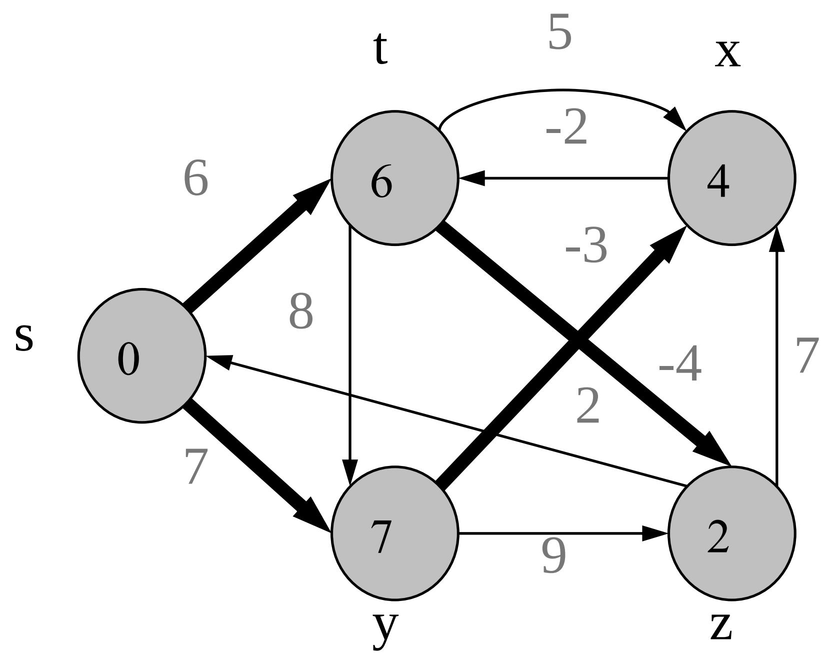
Bellman-Ford Algorithm Example



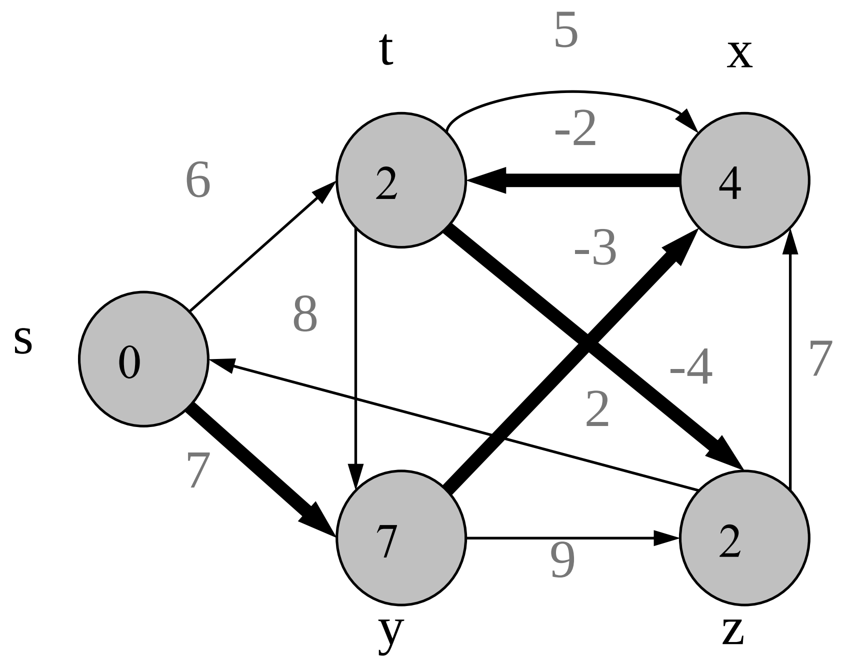
Bellman-Ford Algorithm Example



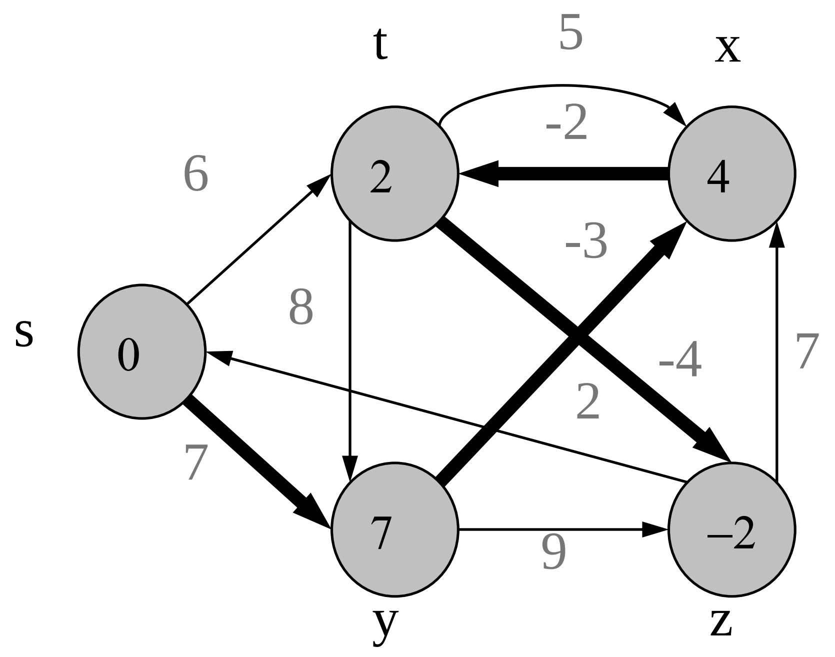
Bellman-Ford Algorithm Example



Bellman-Ford Algorithm Example



Bellman-Ford Algorithm Example



Dijkstra's Algorithm

- **Notation:**

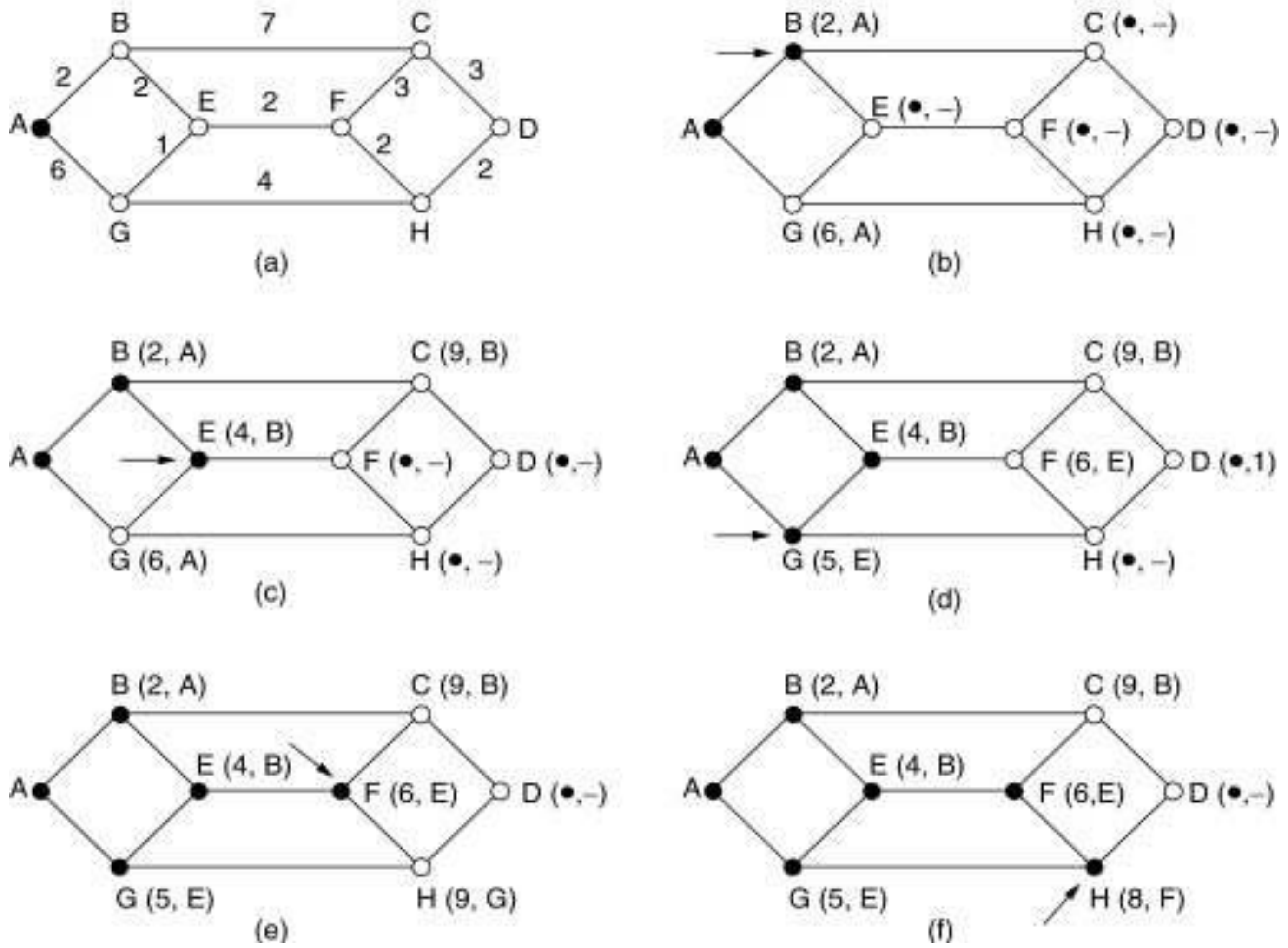
  D_i = Length of shortest path from node 'i' to node 1.

  $d_{i,j}$ = Length of path between nodes i and j .

- **Algorithm**
  Each node j is labeled with Dj, which is an estimate of cost of path from node j to node 1.
- Initially, let the estimates be infinity, indicating that nothing is known about the paths.
- We now iterate on the length of paths, each time revising our estimate to lower values, as we obtain them. Actually, we divide the nodes into two groups ; the first one, called set P contains the nodes whose shortest distances have been found, and the other Q containing all the remaining nodes.

- Initially P contains only the node 1. At each step, we select the node that has minimum cost path to node 1. This node is transferred to set P.
- At the first step, this corresponds to shifting the node closest to 1 in P. Its minimum cost to node 1 is now known. At the next step, select the next closest node from set Q and update the labels corresponding to each node using :
- $\mathbf{D_j = \min [D_j , D_i + d_{j,i}]}$
- Finally, after N-1 iterations, the shortest paths for all nodes are known, and the algorithm terminates.

- **Principle**
  Let the closest node to 1 at some step be i. Then i is shifted to P. Now, for each node j , the closest path to 1 either passes through i or it doesn't.
- In the first case Dj remains the same. In the second case, the revised estimate of D_j is the sum $D_i + d_{i,j}$. So we take the minimum of these two cases and update D_j accordingly.
- As each of the nodes get transferred to set P, the estimates get closer to the lowest possible value. When a node is transferred, its shortest path length is known.
- So finally all the nodes are in P and the D_j 's represent the minimum costs. The algorithm is guaranteed to terminate in N-1 iterations and its complexity is O(N^2).



The first 6 steps used in computing the shortest path from A to D.
The arrows indicate the working node.

```
Initially mark all nodes (except source) with infinite distance.
working node = source node
Sink node = destination node
While the working node is not equal to the sink
    1. Mark the working node as permanent.
    2. Examine all adjacent nodes in turn
    If the sum of label on working node plus distance from working node to adjacent node
    is less than current labelled distance on the adjacent node, this implies a shorter path.
    Relabel the distance on the adjacent node and label it with the node from which the
    probe was made.
    3. Examine all tentative nodes (not just adjacent nodes) and mark the node with the
    smallest labelled value as permanent. This node becomes the new working node.
    Reconstruct the path backwards from sink to source.
```

The Floyd Warshall Algorithm

- This algorithm iterates on the set of nodes that can be used as intermediate nodes on paths. This set grows from a single node (say node 1) at start to finally all the nodes of the graph. At each iteration, we find the shortest path using given set of nodes as intermediate nodes, so that finally all the shortest paths are obtained.
- **Notation**
  $D_{i,j}$ [n] = Length of shortest path between the nodes i and j using only the nodes 1,2,....n as intermediate nodes.
- **Initial Condition**
  Di,j[0] = $d_{i,j}$ for all nodes i,j .

Algorithm

Initially, n = 0.

At each iteration, add next node to n.

i.e. For n = 1,2, ...N-1 ,

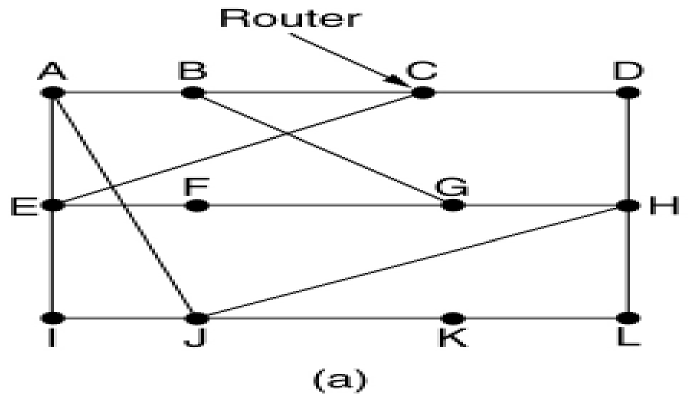
$$Di,j[n + 1] = \min \{ D_{i,j}[n] , \quad D_{i,n+1}[n] + D_{n+1,j}[n] \}$$

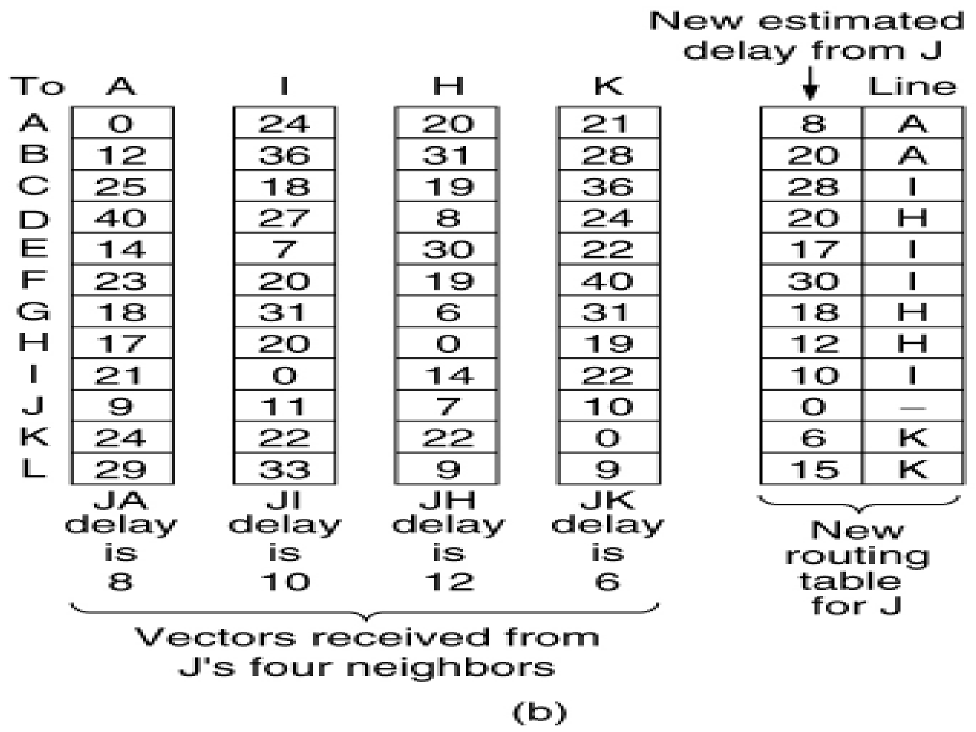
- **Principle**
  Suppose the shortest path between i and j using nodes 1,2,...n is known. Now, if node n+1 is allowed to be an intermediate node, then the shortest path under new conditions either passes through node n+1 or it doesn't.
- If it does not pass through the node n+1, then $D_{i,j}[n+1]$ is same as $D_{i,j}[n]$. Else, we find the cost of the new route, which is obtained from the sum, $D_{i,n+1}[n] + D_{n+1,j}[n]$.
- So we take the minimum of these two cases at each step. After adding all the nodes to the set of intermediate nodes, we obtain the shortest paths between all pairs of nodes together.
- The complexity of Floyd-Warshall algorithm is $O(N^3)$.

Distance Vector Routing

- Distance Vector Routing is dynamic routing algorithm.
- Distance Vector Routing algorithms operate by having each router maintain a table (i.e., a vector) giving the best known distance to each destination and which line to use to get there.
- These tables are updated by exchanging information with the neighbours.

- As an example, assume that delay is used as a metric and that the router knows the delay to each of its neighbours.
- Once every T msec each router sends to each neighbour a list of its estimated delays to each destination.
- It also receives a similar list from each neighbour.





To	A	I	H	K	New estimated delay from J	Line
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	—
K	24	22	22	0	6	K
L	29	33	9	9	15	K
	JA delay is 8	JI delay is 10	JH delay is 12	JK delay is 6	New routing table for J	

$\underbrace{\qquad\qquad\qquad\qquad}_{\text{Vectors received from J's four neighbors}}$

(b)

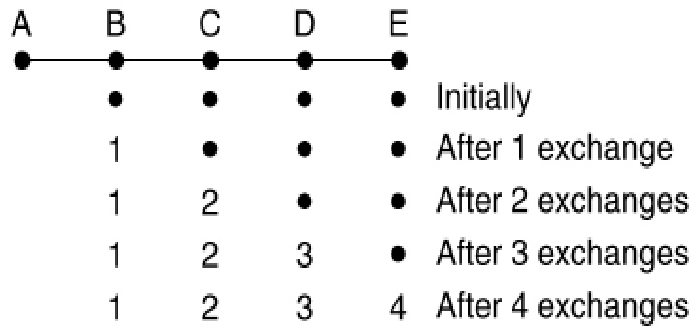
(a) A subnet. (b) Input from A, I, H, K, and the new routing table for J.

- Fig (b) shows the delay vectors received from the neighbors of router *J*. *A* claims to have a 12-msec delay to *B*, a 25-msec delay to *C*, a 40- msec delay to *D*, etc. Suppose that *J* has measured or estimated its delay to its neighbors, *A*, *I, H*, and *K*, as 8, 10, 12, and 6 msec, respectively.
- Consider how *J* computes its new route to router *G*.
- It knows that it can get to *A* in 8 msec, and furthermore *A* claims to be able to get to *G* in 18 msec, so *J* knows it can count on a delay of 26 msec to *G* if it forwards packets bound for *G* to *A*.
- Similarly, it computes the delay to *G* via *I*, *H*, and *K* as 41 (31 + 10), 18 (6 + 12), and 37 (31 + 6) msec, respectively.
- The best of these values is 18, so it makes an entry in its routing table that the delay to *G* is 18 msec and that the route to use is via *H*.
- The same calculation is performed for all the other destinations, with the new routing table shown in the last column of the figure.

The Count-to-Infinity Problem

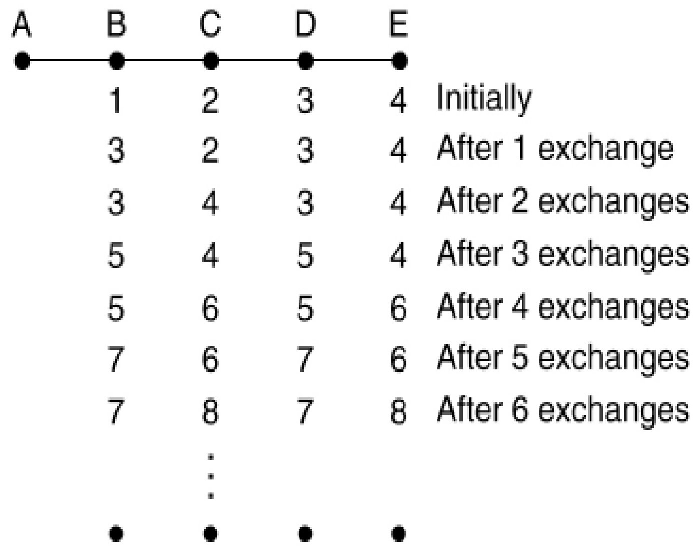
- Distance Vector Routing works in theory but has a serious drawback in practice.
- In particular, it reacts rapidly to good news, but leisurely to bad news.
- The core of the problem is that when X tells Y that it has a path somewhere, Y has no way of knowing whether it itself is on the path.

The count-to-infinity problem.



A	B	C	D	E	
	•	•	•	•	Initially
	1	•	•	•	After 1 exchange
	1	2	•	•	After 2 exchanges
	1	2	3	•	After 3 exchanges
	1	2	3	4	After 4 exchanges

(a)



A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			
	•	•	•	•	

(b)

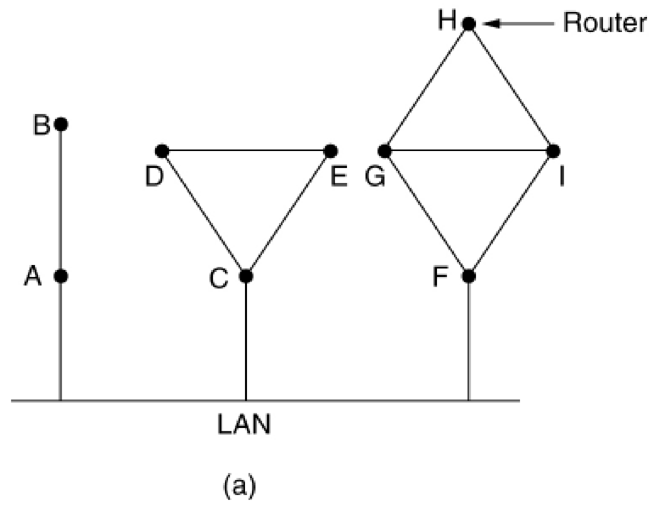
Link State Routing

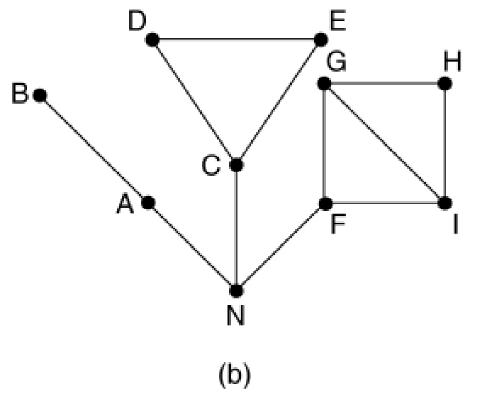
- Two primary problems of distance vector routing
- First, since the delay metric was queue length, it did not take line bandwidth into account when choosing routes.
- Second, the algorithm often took too long to converge (the count-to-infinity problem)
- Distance vector routing was replaced by Link State Routing
- Link State Routing is also dynamic routing algorithm.

Each router must do the following:

1. Discover its neighbours, learn their network address.
2. Measure the delay or cost to each of its neighbours.
3. Construct a packet telling all it has just learned.
4. Send this packet to all other routers.
5. Compute the shortest path to every other router.

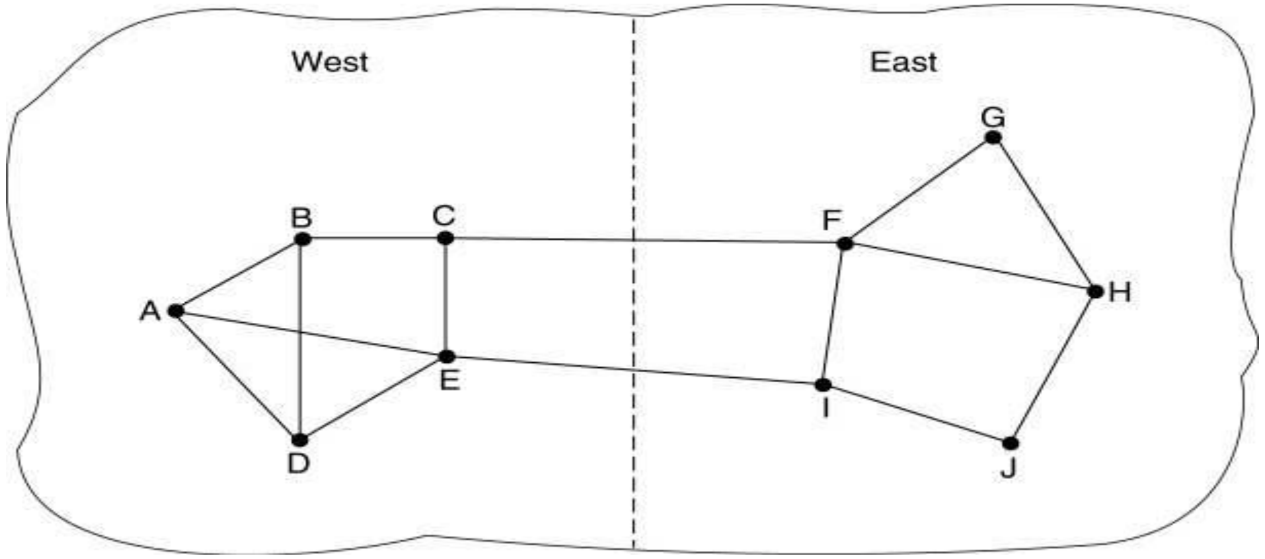
Learning about the Neighbours





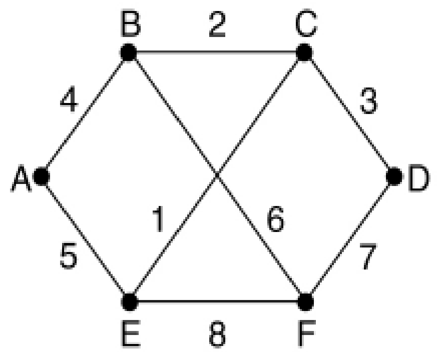
(a) Nine routers and a LAN. (b) A graph model of (a).

- Measuring Line Cost

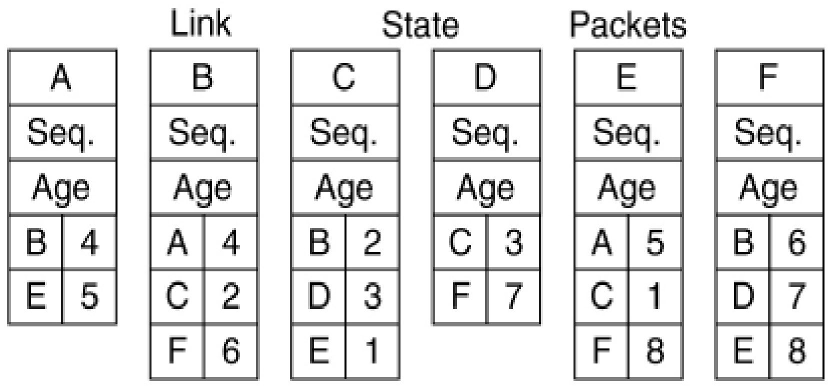


A subnet in which the East and West parts are connected by two lines.

Building Link State Packets



(a)


Link State Packets

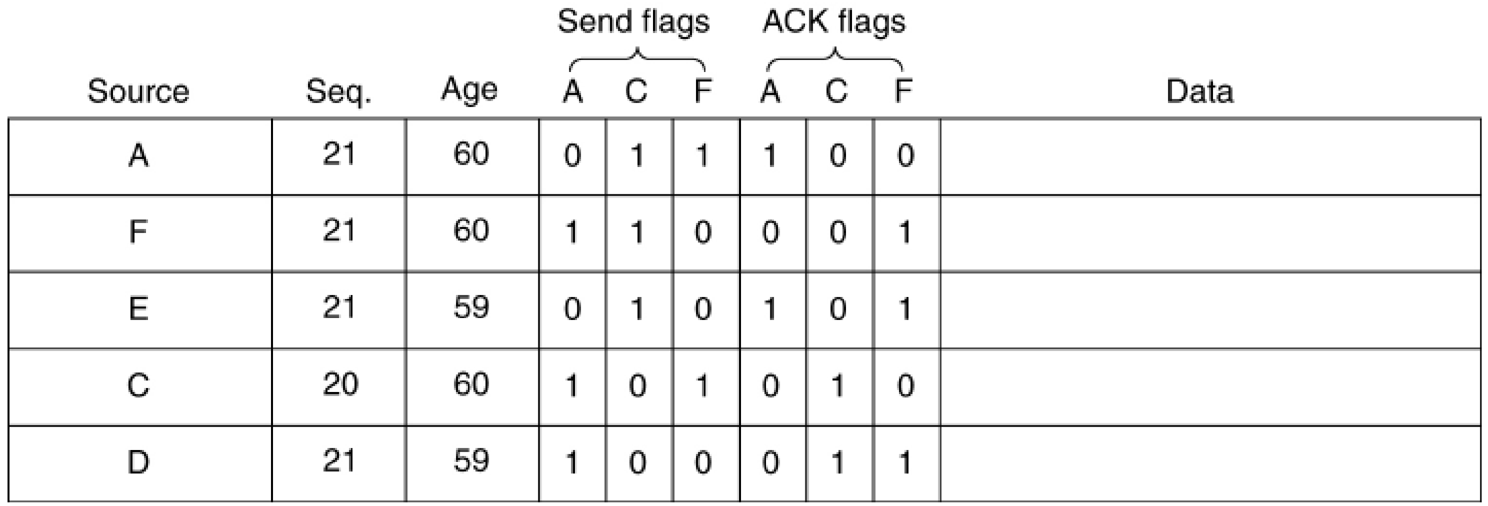
A	B	C	D	E	F
Seq.	Seq.	Seq.	Seq.	Seq.	Seq.
Age	Age	Age	Age	Age	Age
B 4	A 4	B 2	C 3	A 5	B 6
E 5	C 2	D 3	F 7	C 1	D 7
	F 6	E 1		F 8	E 8

(b)

(a) A subnet. (b) The link state packets for this subnet.

Distributing the Link State Packets

The packet buffer for router B

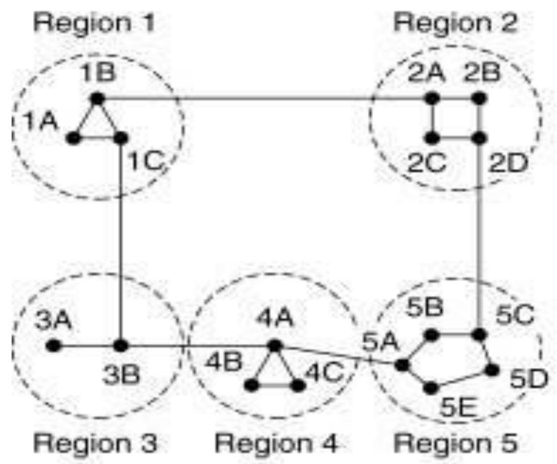


			Send flags			ACK flags			
Source	Seq.	Age	A	C	F	A	C	F	Data
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

Computing the New Routes

- Once a router has accumulated a full set of link state packets, it can construct the entire subnet graph because every link is represented.
- Every link is, in fact, represented twice, once for each direction.

Hierarchical Routing



(a)

Full table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

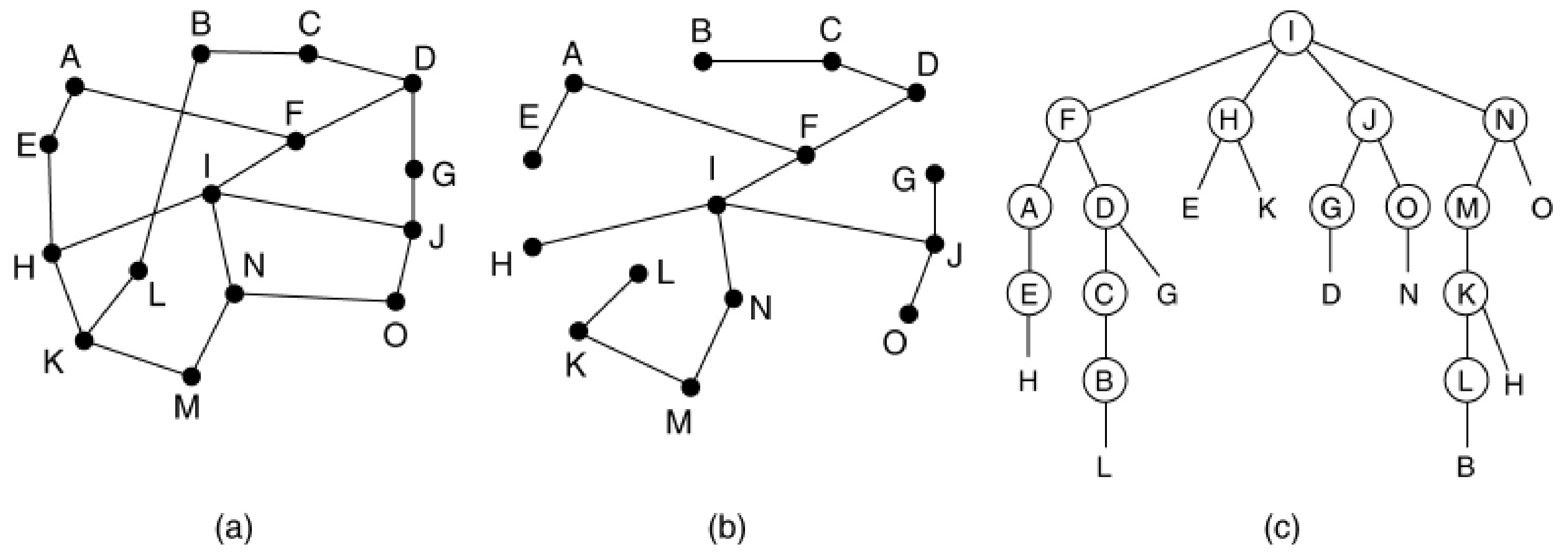
(b)

Hierarchical table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

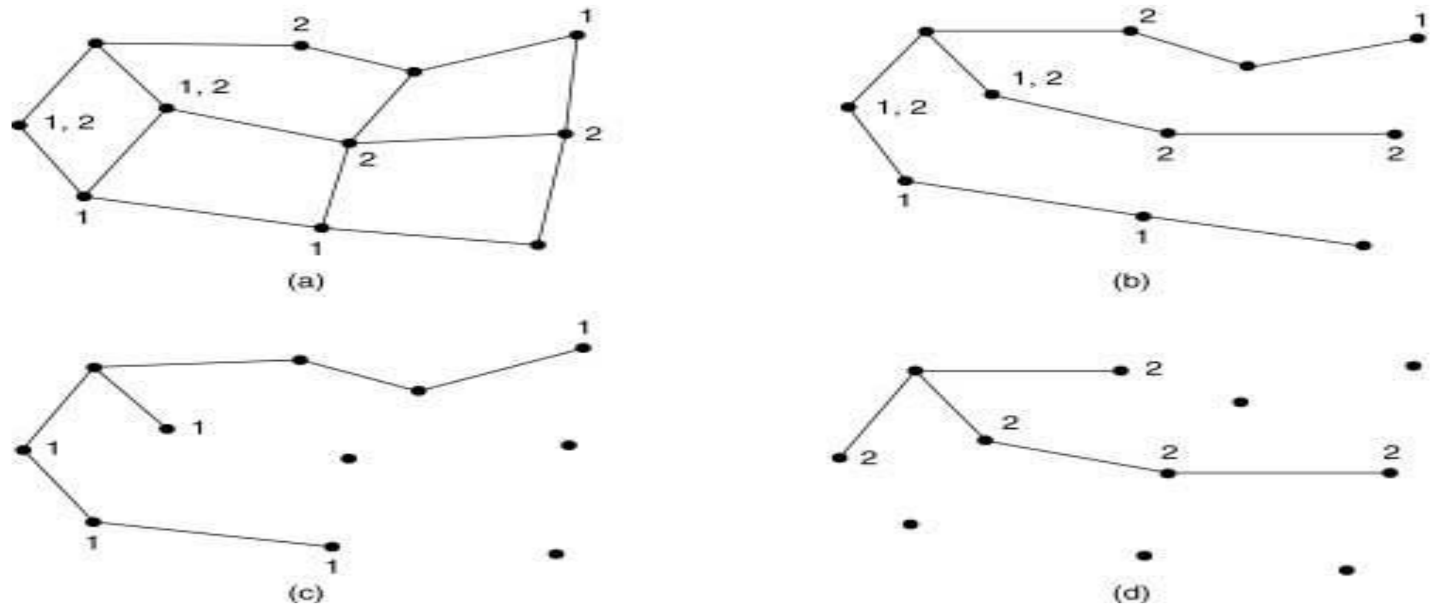
(c)

Broadcast Routing



Reverse path forwarding. (a) A subnet. (b) a Sink tree. (c) The tree built by reverse path forwarding.

Multicast Routing



(a) A network. (b) A spanning tree for the leftmost router.
(c) A multicast tree for group 1. (d) A multicast tree for group 2.