CHAPTER 12

Routing in Switched Networks

"I tell you," went on Syme with passion, "that every time a train comes in I feel that it has broken past batteries of besiegers, and that man has won a battle against chaos. You say contemptuously that when one has left Sloane Square one must come to Victoria. I say that one might do a thousand things instead, and that whenever I really come there I have the sense of hairbreadth escape. And when I hear the guard shout out the word 'Victoria', it is not an unmeaning word. It is to me the cry of a herald announcing conquest. It is to me indeed 'Victoria'; it is the victory of Adam."

—*The Man Who Was Thursday*, G.K. Chesterton

KEY POINTS

- A variety of routing algorithms have been developed for packet-switching, frame relay, and ATM networks, and for the Internet and internetworks. These algorithms share many common principles.
- Routing schemes can be categorized based on a number of factors, such as what criterion is used to determine the best route between two nodes, what strategy is used for obtaining information needed to determine route, and whether a distributed or centralized algorithm is used.
- The routing function attempts to find the least-cost route through the network, with cost based on number of hops, expected delay, or other metrics. Adaptive routing algorithms typically rely on the exchange of information about traffic conditions among nodes.

A key design issue in switched networks, including packet-switching, frame relay, and ATM networks, and with internets, is that of routing. In general terms, the routing function seeks to design routes through the network for individual pairs of communicating end nodes such that the network is used efficiently.

This chapter begins with a brief overview of issues involved in routing design. Next, we look at the routing function in packet-switching networks and then examine least-cost algorithms that are a central part of routing in switched networks. These topics cover issues that are relevant to routing in internets as well as packet-switching networks.

12.1 ROUTING IN PACKET-SWITCHING NETWORKS

One of the most complex and crucial design aspects of switched data networks is routing. This section surveys key characteristic that can be used to classify routing strategies. The principles described in this section are also applicable to internetwork routing, discussed in Part Five.

Characteristics

The primary function of a packet-switching network is to accept packets from a source station and deliver them to a destination station. To accomplish this, a path or route through the network must be determined; generally, more than one route is possible. Thus, a routing function must be performed. The requirements for this function include

- Correctness
- Simplicity
- Robustness
- Stability
- Fairness
- Optimality
- Efficiency

The first two items on the list, correctness and simplicity, are self-explanatory. Robustness has to do with the ability of the network to deliver packets via some route in the face of localized failures and overloads. Ideally, the network can react to such contingencies without the loss of packets or the breaking of virtual circuits. The designer who seeks robustness must cope with the competing requirement for stability. Techniques that react to changing conditions have an unfortunate tendency to either react too slowly to events or to experience unstable swings from one extreme to another. For example, the network may react to congestion in one area by shifting most of the load to a second area. Now the second area is overloaded and the first is underutilized, causing a second shift. During these shifts, packets may travel in loops through the network.

A tradeoff also exists between fairness and optimality. Some performance criteria may give higher priority to the exchange of packets between nearby stations compared to an exchange between distant stations. This policy may maximize average throughput but will appear unfair to the station that primarily needs to communicate with distant stations.

Finally, any routing technique involves some processing overhead at each node and often a transmission overhead as well, both of which impair network efficiency. The penalty of such overhead needs to be less than the benefit accrued based on some reasonable metric, such as increased robustness or fairness.

With these requirements in mind, we are in a position to assess the various design elements that contribute to a routing strategy. Table 12.1 lists these elements. Some of these categories overlap or are dependent on one another. Nevertheless, an examination of this list serves to clarify and organize routing concepts.

Performance Criteria The selection of a route is generally based on some performance criterion. The simplest criterion is to choose the minimum-hop route (one that passes through the least number of nodes) through the network.[1] This is an easily measured criterion and should minimize the consumption of network resources. A generalization of the minimum-hop criterion is least-cost routing. In

[1]The term *hop* is used somewhat loosely in the literature. The more common definition, which we use, is that the number of hops along a path from a given source to a given destination is the number of links between network nodes (packet-switching nodes, ATM switches, routers, etc.) that a packet traverses along that path. Sometimes the number of hops is defined to include the link between the source station and the network and the link between the destination station and the network. This latter definition produces a value two greater than the definition we use.

Table 12.1 Elements of Routing Techniques for Packet-Switching Networks

Performance Criteria	**Network Information Source**
Number of hops	None
Cost	Local
Delay	Adjacent node
Throughput	Nodes along route
	All nodes
Decision Time	
Packet (datagram)	**Network Information Update Timing**
Session (virtual circuit)	Continuous
	Periodic
Decision Place	Major load change
Each node (distributed)	Topology change
Central node (centralized)	
Originating node (source)	

this case, a cost is associated with each link, and, for any pair of attached stations, the route through the network that accumulates the least cost is sought. For example, Figure 12.1 illustrates a network in which the two arrowed lines between a pair of nodes represent a link between these nodes, and the corresponding numbers represent the current link cost in each direction. The shortest path (fewest hops) from node 1 to node 6 is 1-3-6 (cost = 5 + 5 = 10), but the least-cost path is 1-4-5-6 (cost = 1 + 1 + 2 = 4). Costs are assigned to links to support one or more design objectives. For example, the cost could be inversely related to the data rate (i.e., the higher the data rate on a link, the lower the assigned cost of the link) or the current queuing delay on the link. In the first case, the least-cost route should provide the highest throughput. In the second case, the least-cost route should minimize delay.

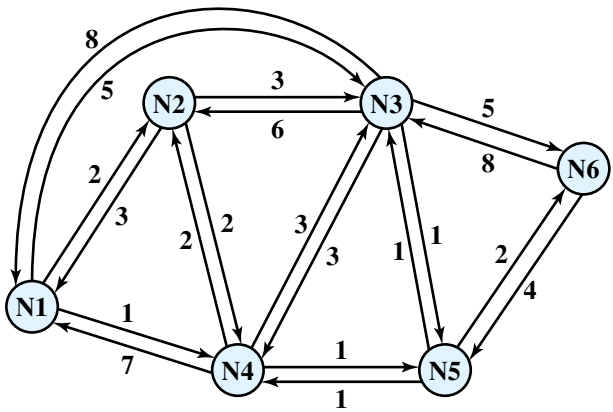


Figure 12.1 Example Network Configuration

In either the minimum-hop or least-cost approach, the algorithm for determining the optimum route for any pair of stations is relatively straightforward, and the processing time would be about the same for either computation. Because the least-cost criterion is more flexible, this is more common than the minimum-hop criterion.

Several least-cost routing algorithms are in common use. These are described in Section 12.3.

Decision Time and Place Routing decisions are made on the basis of some performance criterion. Two key characteristics of the decision are the time and place that the decision is made.

Decision time is determined by whether the routing decision is made on a packet or virtual circuit basis. When the internal operation of the network is datagram, a routing decision is made individually for each packet. For internal virtual circuit operation, a routing decision is made at the time the virtual circuit is established. In the simplest case, all subsequent packets using that virtual circuit follow the same route. In more sophisticated network designs, the network may dynamically change the route assigned to a particular virtual circuit in response to changing conditions (e.g., overload or failure of a portion of the network).

The term *decision place* refers to which node or nodes in the network are responsible for the routing decision. Most common is distributed routing, in which each node has the responsibility of selecting an output link for routing packets as they arrive. For centralized routing, the decision is made by some designated node, such as a network control center. The danger of this latter approach is that the loss of the network control center may block operation of the network. The distributed approach is perhaps more complex but is also more robust. A third alternative, used in some networks, is source routing. In this case, the routing decision is actually made by the source station rather than by a network node and is then communicated to the network. This allows the user to dictate a route through the network that meets criteria local to that user.

The decision time and decision place are independent design variables. For example, in Figure 12.1, suppose that the decision place is each node and that the values depicted are the costs at a given instant in time: the costs may change. If a packet is to be delivered from node 1 to node 6, it might follow the route 1-4-5-6, with each leg of the route determined locally by the transmitting node. Now let the values change such that 1-4-5-6 is no longer the optimum route. In a datagram network, the next packet may follow a different route, again determined by each node along the way. In a virtual circuit network, each node will remember the routing decision that was made when the virtual circuit was established, and simply pass on the packets without making a new decision.

Network Information Source and Update Timing Most routing strategies require that decisions be based on knowledge of the topology of the network, traffic load, and link cost. Surprisingly, some strategies use no such information and yet manage to get packets through; flooding and some random strategies (discussed later) are in this category.

With distributed routing, in which the routing decision is made by each node, the individual node may make use of only local information, such as the cost of each outgoing link. Each node might also collect information from adjacent (directly

connected) nodes, such as the amount of congestion experienced at that node. Finally, there are algorithms in common use that allow the node to gain information from all nodes on any potential route of interest. In the case of centralized routing, the central node typically makes use of information obtained from all nodes.

A related concept is that of information update timing, which is a function of both the information source and the routing strategy. Clearly, if no information is used (as in flooding), there is no information to update. If only local information is used, the update is essentially continuous. That is, an individual node always knows its local conditions. For all other information source categories (adjacent nodes, all nodes), update timing depends on the routing strategy. For a fixed strategy, the information is never updated. For an adaptive strategy, information is updated from time to time to enable the routing decision to adapt to changing conditions.

As you might expect, the more information available, and the more frequently it is updated, the more likely the network is to make good routing decisions. On the other hand, the transmission of that information consumes network resources.

Routing Strategies

A large number of routing strategies have evolved for dealing with the routing requirements of packet-switching networks. Many of these strategies are also applied to internetwork routing, which we cover in Part Five. In this section, we survey four key strategies: fixed, flooding, random, and adaptive.

Fixed Routing For fixed routing, a single, permanent route is configured for each source-destination pair of nodes in the network. Either of the least-cost routing algorithms described in Section 12.3 could be used. The routes are fixed, or at least only change when there is a change in the topology of the network. Thus, the link costs used in designing routes cannot be based on any dynamic variable such as traffic. They could, however, be based on expected traffic or capacity.

Figure 12.2 suggests how fixed routing might be implemented. A central routing matrix is created, to be stored perhaps at a network control center. The matrix shows, for each source-destination pair of nodes, the identity of the next node on the route.

Note that it is not necessary to store the complete route for each possible pair of nodes. Rather, it is sufficient to know, for each pair of nodes, the identity of the first node on the route. To see this, suppose that the least-cost route from X to Y begins with the X-A link. Call the remainder of the route R_1; this is the part from A to Y. Define R_2 as the least-cost route from A to Y. Now, if the cost of R_1 is greater than that of R_2, then the X-Y route can be improved by using R_2 instead. If the cost of R_1 is less than R_2, then R_2 is not the least-cost route from A to Y. Therefore, $R_1 = R_2$. Thus, at each point along a route, it is only necessary to know the identity of the next node, not the entire route. In our example, the route from node 1 to node 6 begins by going through node 4. Again consulting the matrix, the route from node 4 to node 6 goes through node 5. Finally, the route from node 5 to node 6 is a direct link to node 6. Thus, the complete route from node 1 to node 6 is 1-4-5-6.

From this overall matrix, routing tables can be developed and stored at each node. From the reasoning in the preceding paragraph, it follows that each node need

CENTRAL ROUTING DIRECTORY

To Node \ From Node	1	2	3	4	5	6
1	—	1	5	2	4	5
2	2	—	5	2	4	5
3	4	3	—	5	3	5
4	4	4	5	—	4	5
5	4	4	5	5	—	5
6	4	4	5	5	6	—

Node 1 Directory

Destination	Next Node
2	2
3	4
4	4
5	4
6	4

Node 2 Directory

Destination	Next Node
1	1
3	3
4	4
5	4
6	4

Node 3 Directory

Destination	Next Node
1	5
2	5
4	5
5	5
6	5

Node 4 Directory

Destination	Next Node
1	2
2	2
3	5
5	5
6	5

Node 5 Directory

Destination	Next Node
1	4
2	4
3	3
4	4
6	6

Node 6 Directory

Destination	Next Node
1	5
2	5
3	5
4	5
5	5

Figure 12.2 Fixed Routing (using Figure 12.1)

only store a single column of the routing directory. The node's directory shows the next node to take for each destination.

With fixed routing, there is no difference between routing for datagrams and virtual circuits. All packets from a given source to a given destination follow the same route. The advantage of fixed routing is its simplicity, and it should work well in a reliable network with a stable load. Its disadvantage is its lack of flexibility. It does not react to network congestion or failures.

A refinement to fixed routing that would accommodate link and node outages would be to supply the nodes with an alternate next node for each destination. For example, the alternate next nodes in the node 1 directory might be 4, 3, 2, 3, 3.

Flooding Another simple routing technique is flooding. This technique requires no network information whatsoever and works as follows. A packet is sent by a source node to every one of its neighbors. At each node, an incoming packet is retransmitted on all outgoing links except for the link on which it arrived. For example, if node 1 in Figure 12.1 has a packet to send to node 6, it send a copy of that packet (with a destination address of 6), to nodes 2, 3, and 4. Node 2 will send a copy

to nodes 3 and 4. Node 4 will send a copy to nodes 2, 3, and 5. And so it goes. Eventually, a number of copies of the packet will arrive at node 6. The packet must have some unique identifier (e.g., source node and sequence number, or virtual circuit number and sequence number) so that node 6 knows to discard all but the first copy.

Unless something is done to stop the incessant retransmission of packets, the number of packets in circulation just from a single source packet grows without bound. One way to prevent this is for each node to remember the identity of those packets it has already retransmitted. When duplicate copies of the packet arrive, they are discarded. A simpler technique is to include a hop count field with each packet. The count can originally be set to some maximum value, such as the diameter (length of the longest minimum-hop path through the network)[2] of the network. Each time a node passes on a packet, it decrements the count by one. When the count reaches zero, the packet is discarded.

An example of the latter tactic is shown in Figure 12.3. The label on each packet in the figure indicates the current value of the hop count field in that packet. A packet is to be sent from node 1 to node 6 and is assigned a hop count of 3. On the first hop, three copies of the packet are created, and the hop count is decrement to 2. For the second hop of all these copies, a total of nine copies are created. One of these copies reaches node 6, which recognizes that it is the intended destination and does not retransmit. However, the other nodes generate a total of 22 new copies for their third and final hop. Each packet now has a hope count of 1. Note that if a node is not keeping track of packet identifier, it may generate multiple copies at this third stage. All packets received from the third hop are discarded, because the hop count is exhausted. In all, node 6 has received four additional copies of the packet.

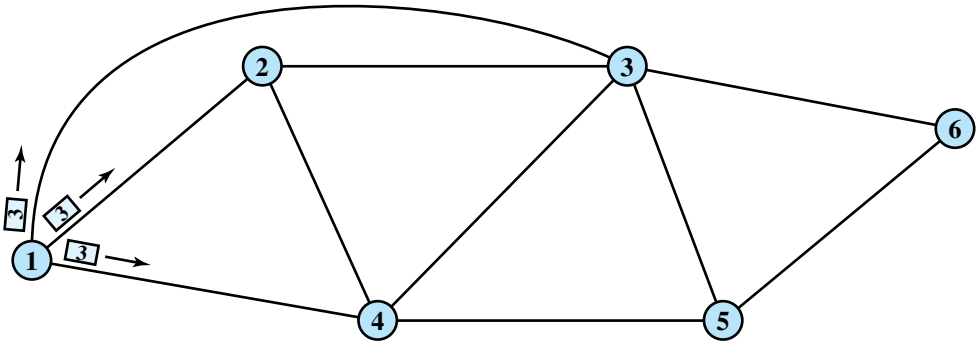
The flooding technique has three remarkable properties:

- All possible routes between source and destination are tried. Thus, no matter what link or node outages have occurred, a packet will always get through if at least one path between source and destination exists.
- Because all routes are tried, at least one copy of the packet to arrive at the destination will have used a minimum-hop route.
- All nodes that are directly or indirectly connected to the source node are visited.

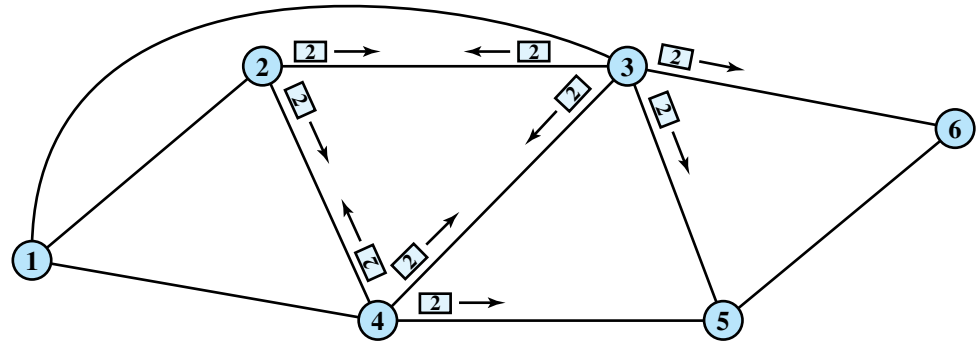
Because of the first property, the flooding technique is highly robust and could be used to send emergency messages. An example application is a military network that is subject to extensive damage. Because of the second property, flooding might be used initially to set up the route for a virtual circuit. The third property suggests that flooding can be useful for the dissemination of important information to all nodes; we will see that it is used in some schemes to disseminate routing information.

The principal disadvantage of flooding is the high traffic load that it generates, which is directly proportional to the connectivity of the network.

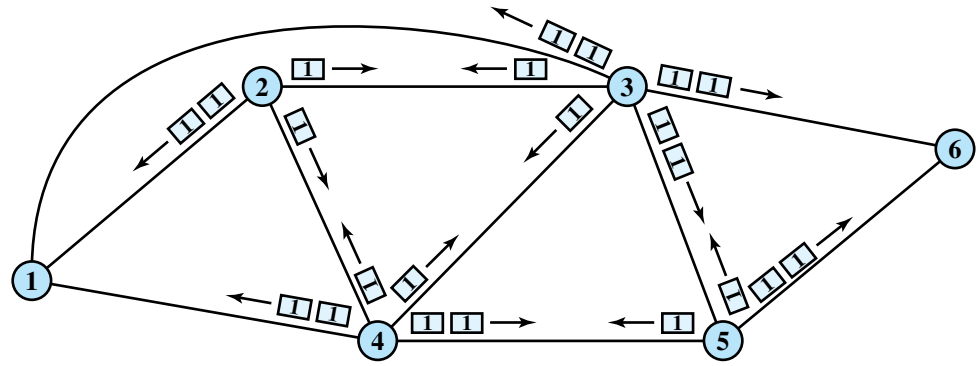
[2]For each pair of end systems attached to the network, there is a minimum-hop path. The length of the longest such minimum-hop path is the diameter of the network.



(a) First hop



(b) Second hop



(c) Third hop

Figure 12.3 Flooding Example (hop count = 3)

Random Routing Random routing has the simplicity and robustness of flooding with far less traffic load. With random routing, a node selects only one outgoing path for retransmission of an incoming packet. The outgoing link is chosen at random, excluding the link on which the packet arrived. If all links are equally likely to be chosen, then a node may simply utilize outgoing links in a round-robin fashion.

A refinement of this technique is to assign a probability to each outgoing link and to select the link based on that probability. The probability could be based on data rate, in which case we have

$$P_i = \frac{R_i}{\sum_j R_j}$$

where

P_i = probability of selecting link i
R_i = data rate on link i

The sum is taken over all candidate outgoing links. This scheme should provide good traffic distribution. Note that the probabilities could also be based on fixed link costs.

Like flooding, random routing requires the use of no network information. Because the route taken is random, the actual route will typically not be the least-cost route nor the minimum-hop route. Thus, the network must carry a higher than optimum traffic load, although not nearly as high as for flooding.

Adaptive Routing In virtually all packet-switching networks, some sort of adaptive routing technique is used. That is, the routing decisions that are made change as conditions on the network change. The principal conditions that influence routing decisions are

- **Failure:** When a node or link fails, it can no longer be used as part of a route.
- **Congestion:** When a particular portion of the network is heavily congested, it is desirable to route packets around rather than through the area of congestion.

For adaptive routing to be possible, information about the state of the network must be exchanged among the nodes. There are several drawbacks associated with the use of adaptive routing, compared to fixed routing:

- The routing decision is more complex; therefore, the processing burden on network nodes increases.
- In most cases, adaptive strategies depend on status information that is collected at one place but used at another. There is a tradeoff here between the quality of the information and the amount of overhead. The more information that is exchanged, and the more frequently it is exchanged, the better will be the routing decisions that each node makes. On the other hand, this information is itself a load on the constituent networks, causing a performance degradation.
- An adaptive strategy may react too quickly, causing congestion-producing oscillation, or too slowly, being irrelevant.

Despite these real dangers, adaptive routing strategies are by far the most prevalent, for two reasons:

- An adaptive routing strategy can improve performance, as seen by the network user.

- An adaptive routing strategy can aid in congestion control, which is discussed in Chapter 13. Because an adaptive routing strategy tends to balance loads, it can delay the onset of severe congestion.

These benefits may or may not be realized, depending on the soundness of the design and the nature of the load. By and large, adaptive routing is an extraordinarily complex task to perform properly. As demonstration of this, most major packet-switching networks, such as ARPANET and its successors, and many commercial networks, have endured at least one major overhaul of their routing strategy.

A convenient way to classify adaptive routing strategies is on the basis of information source: local, adjacent nodes, all nodes. An example of an adaptive routing strategy that relies only on local information is one in which a node routes each packet to the outgoing link with the shortest queue length, Q. This would have the effect of balancing the load on outgoing links. However, some outgoing links may not be headed in the correct general direction. We can improve matters by also taking into account preferred direction, much as with random routing. In this case, each link emanating from the node would have a bias B_i, for each destination i, such that lower values of B_i indicate more preferred directions. For each incoming packet headed for node i, the node would choose the outgoing link that minimizes $Q + B_i$. Thus a node would tend to send packets in the right direction, with a concession made to current traffic delays.

As an example, Figure 12.4 show the status of node 4 of Figure 12.1 at a certain point in time. Node 4 has links to four other nodes. Packets have been arriving and a backlog has built up, with a queue of packets waiting for each of the outgoing links. A packet arrives from node 1 destined for node 6. To which outgoing link should the packet be routed? Based on current queue lengths and the values of bias (B_6) for

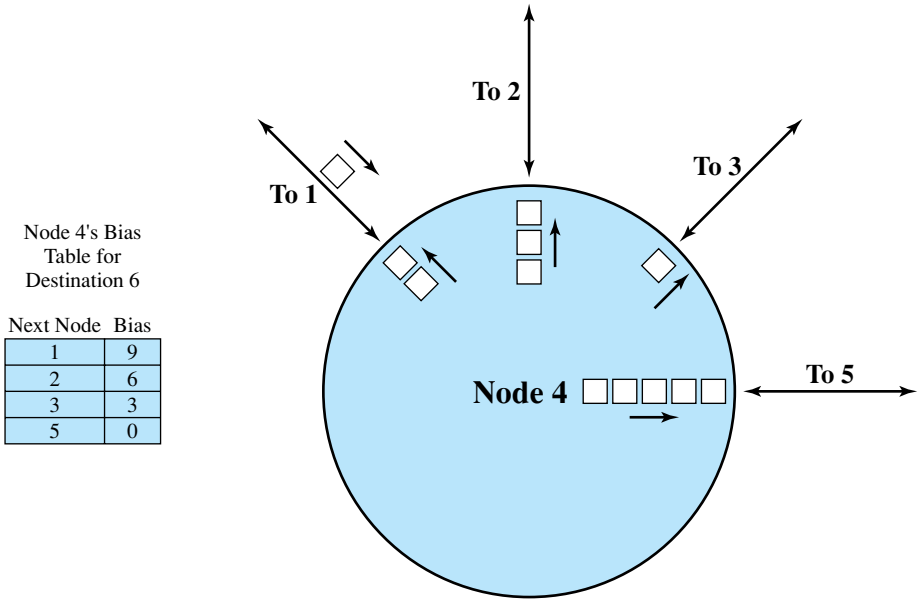


Figure 12.4 Example of Isolated Adaptive Routing

each outgoing link, the minimum value of $Q + B_6$ is 4, on the link to node 3. Thus, node 4 routes the packet through node 3.

Adaptive schemes based only on local information are rarely used because they do not exploit easily available information. Strategies based on information from adjacent nodes or all nodes are commonly found. Both take advantage of information that each node has about delays and outages that it experiences. Such adaptive strategies can be either distributed or centralized. In the distributed case, each node exchanges delay information with other nodes. Based on incoming information, a node tries to estimate the delay situation throughout the network, and applies a least-cost routing algorithm. In the centralized case, each node reports its link delay status to a central node, which designs routes based on this incoming information and sends the routing information back to the nodes.

12.2 EXAMPLES: ROUTING IN ARPANET

In this section, we look at several examples of routing strategies. All of these were initially developed for ARPANET, which is a packet-switching network that was the foundation of the present-day Internet. It is instructive to examine these strategies for several reasons. First, these strategies and similar ones are also used in other packet-switching networks, including a number of networks on the Internet. Second, routing schemes based on the ARPANET work have also been used for internetwork routing in the Internet and in private internetworks. And finally, the ARPANET routing scheme evolved in a way that illuminates some of the key design issues related to routing algorithms.

First Generation

The original routing algorithm, designed in 1969, was a distributed adaptive algorithm using estimated delay as the performance criterion and a version of the Bellman-Ford algorithm (Section 12.3). For this algorithm, each node maintains two vectors:

$$D_i = \begin{bmatrix} d_{i1} \\ \bullet \\ \bullet \\ \bullet \\ d_{iN} \end{bmatrix} \qquad S_i = \begin{bmatrix} s_{i1} \\ \bullet \\ \bullet \\ \bullet \\ s_{iN} \end{bmatrix}$$

where

D_i = delay vector for node i

d_{ij} = current estimate of minimum delay from node i to node j ($d_{ii} = 0$)

N = number of nodes in the network

S_i = successor node vector for node i

s_{ij} = the next node in the current minimum-delay route from i to j

Destination	Delay	Next Node
1	0	—
2	2	2
3	5	3
4	1	4
5	6	3
6	8	3

D_1 S_1

(a) Node 1's routing table before update

D_2	D_3	D_4
3	7	5
0	4	2
3	0	2
2	2	0
3	1	1
5	3	3

(b) Delay vectors sent to node 1 from neighbor nodes

Destination	Delay	Next Node
1	0	—
2	2	2
3	3	4
4	1	4
5	2	4
6	4	4

$I_{1,2} = 2$
$I_{1,3} = 5$
$I_{1,4} = 1$

(c) Node 1's routing table after update and link costs used in update

Figure 12.5 Original ARPANET Routing Algorithm

Periodically (every 128 ms), each node exchanges its delay vector with all of its neighbors. On the basis of all incoming delay vectors, a node k updates both of its vectors as follows:

$$d_{kj} = \min_{i \in A} [d_{ij} + l_{ki}]$$

$$s_{kj} = i \qquad \text{using } i \text{ that minimizes the preceding expression}$$

where

A = set of neighbor nodes for k

l_{ki} = current estimate of delay from k to i

Figure 12.5 provides an example of the original ARPANET algorithm, using the network of Figure 12.6. This is the same network as that of Figure 12.1, with some of the link costs having different values (and assuming the same cost in both directions). Figure 12.5a shows the routing table for node 1 at an instant in time that reflects the

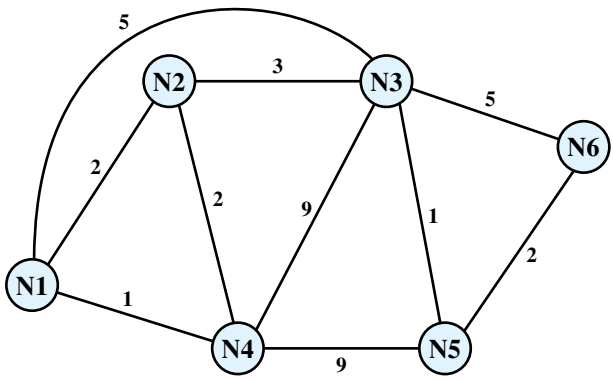


Figure 12.6 Network for Example of Figure 12.5a

link costs of Figure 12.6. For each destination, a delay is specified, and the next node on the route that produces that delay. At some point, the link costs change to those of Figure 12.1. Assume that node 1's neighbors (nodes 2, 3, and 4) learn of the change before node 1. Each of these nodes updates its delay vector and sends a copy to all of its neighbors, including node 1 (Figure 12.5b). Node 1 discards its current routing table and builds a new one, based solely on the incoming delay vector and its own estimate of link delay to each of its neighbors. The result is shown in Figure 12.5c.

The estimated link delay is simply the queue length for that link. Thus, in building a new routing table, the node will tend to favor outgoing links with shorter queues. This tends to balance the load on outgoing links. However, because queue lengths vary rapidly with time, the distributed perception of the shortest route could change while a packet is en route. This could lead to a thrashing situation in which a packet continues to seek out areas of low congestion rather than aiming at the destination.

Second Generation

After some years of experience and several minor modifications, the original routing algorithm was replaced by a quite different one in 1979 [MCQU80]. The major shortcomings of the old algorithm were as follows:

- The algorithm did not consider line speed, merely queue length. Thus, higher-capacity links were not given the favored status they deserved.
- Queue length is, in any case, an artificial measure of delay, because some variable amount of processing time elapses between the arrival of a packet at a node and its placement in an outbound queue.
- The algorithm was not very accurate. In particular, it responded slowly to congestion and delay increases.

The new algorithm is also a distributed adaptive one, using delay as the performance criterion, but the differences are significant. Rather than using queue length as a surrogate for delay, the delay is measured directly. At a node, each incoming packet is timestamped with an arrival time. A departure time is recorded when the packet is transmitted. If a positive acknowledgment is returned, the delay for that packet is recorded as the departure time minus the arrival time plus transmission time and propagation delay. The node must therefore know link data rate and propagation time. If a negative acknowledgment comes back, the departure time is updated and the node tries again, until a measure of successful transmission delay is obtained.

Every 10 seconds, the node computes the average delay on each outgoing link. If there are any significant changes in delay, the information is sent to all other nodes using flooding. Each node maintains an estimate of delay on every network link. When new information arrives, it recomputes its routing table using Dijkstra's algorithm (Section 12.3).

Third Generation

Experience with this new strategy indicated that it was more responsive and stable than the old one. The overhead induced by flooding was moderate because each node does this at most once every 10 seconds. However, as the load on the network

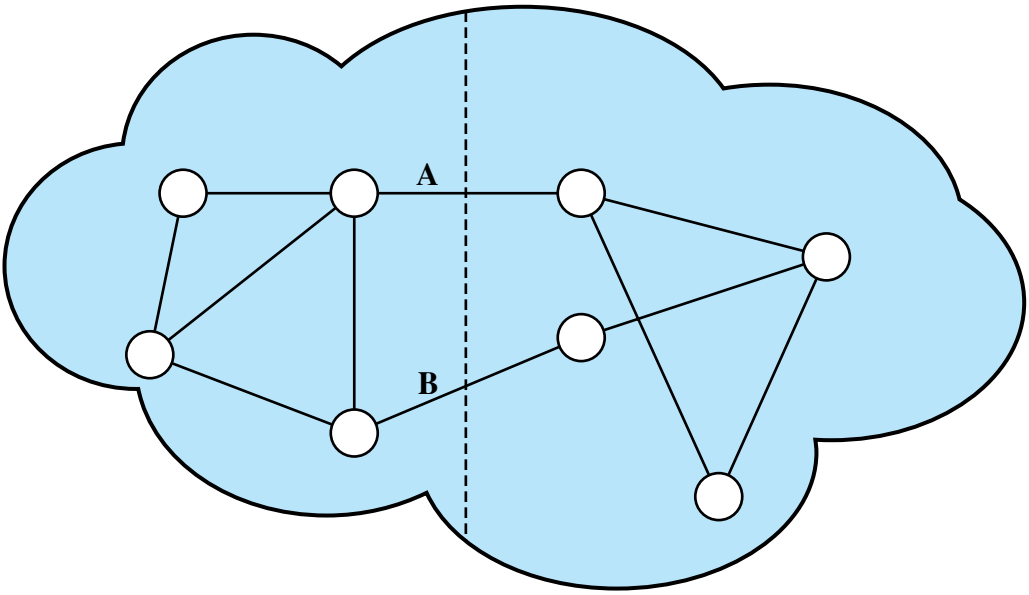


Figure 12.7 Packet-Switching Network Subject to Oscillations

grew, a shortcoming in the new strategy began to appear, and the strategy was revised in 1987 [KHAN89].

The problem with the second strategy is the assumption that the measured packet delay on a link is a good predictor of the link delay encountered after all nodes reroute their traffic based on this reported delay. Thus, it is an effective routing mechanism only if there is some correlation between the reported values and those actually experienced after rerouting. This correlation tends to be rather high under light and moderate traffic loads. However, under heavy loads, there is little correlation. Therefore, immediately after all nodes have made routing updates, the routing tables are obsolete!

As an example, consider a network that consists of two regions with only two links, A and B, connecting the two regions (Figure 12.7). Each route between two nodes in different regions must pass through one of these links. Assume that a situation develops in which most of the traffic is on link A. This will cause the link delay on A to be significant, and at the next opportunity, this delay value will be reported to all other nodes. These updates will arrive at all nodes at about the same time, and all will update their routing tables immediately. It is likely that this new delay value for link A will be high enough to make link B the preferred choice for most, if not all, interregion routes. Because all nodes adjust their routes at the same time, most or all interregion traffic shifts at the same time to link B. Now the link delay value on B will become high, and there will be a subsequent shift to link A. This oscillation will continue until the traffic volume subsides.

There are a number of reasons why this oscillation is undesirable:

- A significant portion of available capacity is unused at just the time when it is needed most: under heavy traffic load.

- The overutilization of some links can lead to the spread of congestion within the network (this will be seen in the discussion of congestion in Chapter 13).
- The large swings in measured delay values result in the need for more frequent routing update messages. This increases the load on the network at just the time when the network is already stressed.

The ARPANET designers concluded that the essence of the problem was that every node was trying to obtain the best route for all destinations, and that these efforts conflicted. It was concluded that under heavy loads, the goal of routing should be to give the average route a good path instead of attempting to give all routes the best path.

The designers decided that it was unnecessary to change the overall routing algorithm. Rather, it was sufficient to change the function that calculates link costs. This was done in such a way as to damp routing oscillations and reduce routing overhead. The calculation begins with measuring the average delay over the last 10 seconds. This value is then transformed with the following steps:

1. Using a simple single-server queuing model, the measured delay is transformed into an estimate of link utilization. From queuing theory, utilization can be expressed as a function of delay as follows:

$$\rho = \frac{2(T_s - T)}{T_s - 2T}$$

where

ρ = link utilization
T = measured delay
T_s = service time

The service time was set at the network-wide average packet size (600 bits) divided by the data rate of the link.

2. The result is then smoothed by averaging it with the previous estimate of utilization:

$$U(n + 1) = 0.5 \times \rho(n + 1) + 0.5 \times U(n)$$

where

$U(n)$ = average utilization calculated at sampling time n
$\rho(n)$ = link utilization measured at sampling time n

Averaging increases the period of routing oscillations, thus reducing routing overhead.

3. The link cost is then set as a function of average utilization that is designed to provide a reasonable estimate of cost while avoiding oscillation. Figure 12.8 indicates the way in which the estimate of utilization is converted into a cost value. The final cost value is, in effect, a transformed value of delay.

In Figure 12.8, delay is normalized to the value achieved on an idle line, which is just propagation delay plus transmission time. One curve on the figure indicates the way in which the actual delay rises as a function of utilization; the increase in delay is due to queuing delay at the node. For the revised algorithm, the cost value is kept at

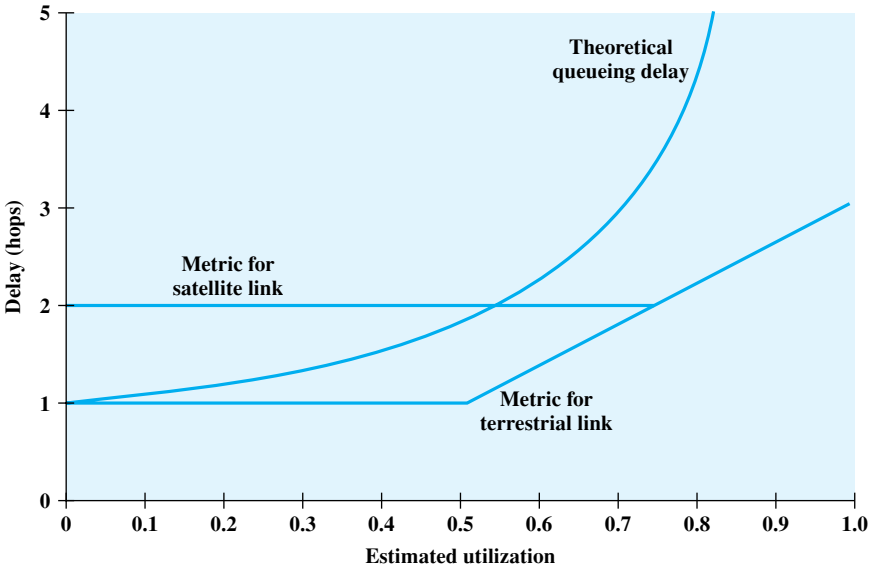


Figure 12.8 ARPANET Delay Metrics

the minimum value until a given level of utilization is reached. This feature has the effect of reducing routing overhead at low traffic levels. Above a certain level of utilization, the cost level is allowed to rise to a maximum value that is equal to three times the minimum value. The effect of this maximum value is to dictate that traffic should not be routed around a heavily utilized line by more than two additional hops.

Note that the minimum threshold is set higher for satellite links. This encourages the use of terrestrial links under conditions of light traffic, because the terrestrial links have much lower propagation delay. Note also that the actual delay curve is much steeper than the transformation curves at high utilization levels. It is this steep rise in link cost that causes all of the traffic on a link to be shed, which in turn causes routing oscillations.

In summary, the revised cost function is keyed to utilization rather than delay. The function acts similar to a delay-based metric under light loads and to a capacity-based metric under heavy loads.

12.3 LEAST-COST ALGORITHMS

Virtually all packet-switching networks and all internets base their routing decision on some form of least-cost criterion. If the criterion is to minimize the number of hops, each link has a value of 1. More typically, the link value is inversely proportional to the link capacity, proportional to the current load on the link, or some combination. In any case, these link or hop costs are used as input to a least-cost routing algorithm, which can be simply stated as follows:

> 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.

Note that the cost of a link 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.

Most least-cost routing algorithms in use in packet-switching networks and internets are variations of one of two common algorithms, known as Dijkstra's algorithm and the Bellman-Ford algorithm. This section provides a summary of these two algorithms.

Dijkstra's Algorithm

Dijkstra's algorithm [DIJK59] can be stated as: Find the shortest paths from a given source node to all other nodes by developing the paths in order of increasing path length. The algorithm proceeds in stages. By the kth stage, the shortest paths to the k nodes closest to (least cost away from) the source node have been determined; these nodes are in a set T. At stage $(k + 1)$, the node not in T that has the shortest path from the source node is added to T. As each node is added to T, its path from the source is defined. The algorithm can be formally described as follows. Define:

$N =$ set of nodes in the network

$s =$ source node

$T =$ set of nodes so far incorporated by the algorithm

$w(i, j) =$ rom cost from node i to node j; $w(i, i) = 0$; $w(i, j) = \infty$ if the two nodes are not directly connected; $w(i, j) \geq 0$ if the two nodes are directly connected

$L(n) =$ cost of the least-cost path from node s to node n that is currently known to the algorithm; at termination, this is the cost of the least-cost path in the graph from s to n

The algorithm has three steps; steps 2 and 3 are repeated until $T = N$. That is, steps 2 and 3 are repeated until final paths have been assigned to all nodes in the network:

1. **[Initialization]**

   $T = \{s\}$ — i.e., the set of nodes so far incorporated consists of only the source node

   $L(n) = w(s, n)$ for $n \neq s$ — i.e., the initial path costs to neighboring nodes are simply the link costs

2. **[Get Next Node]** Find the neighboring node not in T that has the least-cost path from node s and incorporate that node into T. Also incorporate the edge that is incident on that node and a node in T that contributes to the path. This can be expressed as

$$\text{Find } x \notin T \text{ such that } L(x) = \min_{j \notin T} L(j)$$

Add x to T; add to T the edge that is incident on x and that contributes the least cost component to $L(x)$, that is, the last hop in the path.

3. **[Update Least-Cost Paths]**

$$L(n) = \min[L(n), L(x) + w(x, n)] \quad \text{for all } n \notin T$$

If the latter term is the minimum, the path from s to n is now the path from s to x concatenated with the edge from x to n.

The algorithm terminates when all nodes have been added to T. At termination, the value $L(x)$ associated with each node x is the cost (length) of the least-cost path from s to x. In addition, T defines the least-cost path from s to each other node.

One iteration of steps 2 and 3 adds one new node to T and defines the least-cost path from s to that node. That path passes only through nodes that are in T. To see this, consider the following line of reasoning. After k iterations, there are k nodes in T, and the least-cost path from s to each of these nodes has been defined. Now consider all possible paths from s to nodes not in T. Among those paths, there is one of least cost that passes exclusively through nodes in T (see Problem 12.4), ending with a direct link from some node in T to a node not in T. This node is added to T and the associated path is defined as the least-cost path for that node.

Table 12.2a and Figure 12.9 show the result of applying this algorithm to the graph of Figure 12.1, using $s = 1$. The shaded edges define the spanning tree for the graph. The values in each circle are the current estimates of $L(x)$ for each node x. A

Table 12.2 Example of Least-Cost Routing Algorithms (using Figure 12.1)

(a) Dijkstra'a Algorithm ($s = 1$)

Iteration	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

(b) Bellman-Ford Algorithm ($s = 1$)

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

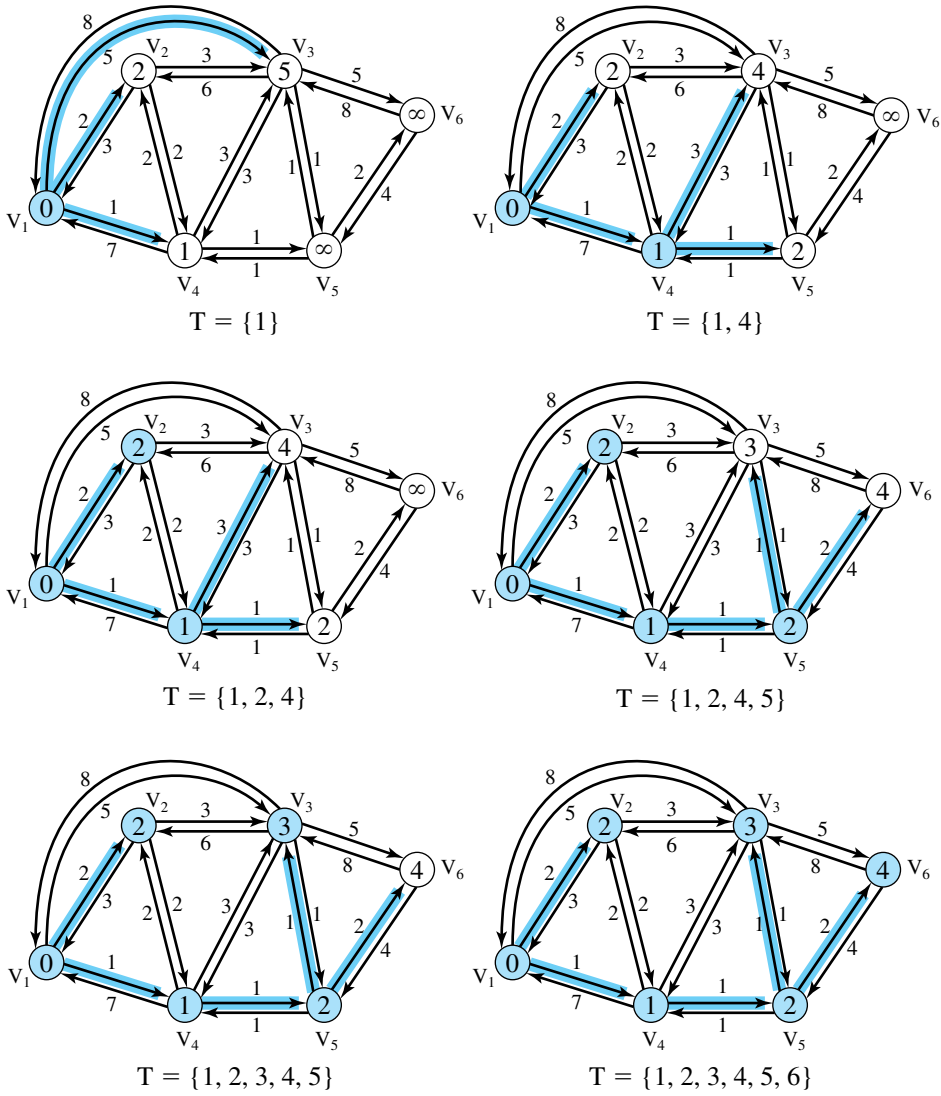


Figure 12.9 Dijkstra's Algorithm Applied to Graph of Figure 12.1

node is shaded when it is added to *T*. Note that at each step the path to each node plus the total cost of that path is generated. After the final iteration, the least-cost path to each node and the cost of that path have been developed. The same procedure can be used with node 2 as source node, and so on.

Bellman-Ford Algorithm

The Bellman-Ford algorithm [FORD62] can be stated as follows: Find the shortest paths from a given source node subject to the constraint that the paths contain at most one link, then find the shortest paths with a constraint of paths of at most two

links, and so on. This algorithm also proceeds in stages. The algorithm can be formally described as follows. Define

s = source node

$w(i, j)$ = link cost from node i to node j; $w(i, i) = 0$; $w(i, j) = \infty$ if the two nodes are not directly connected; $w(i, j) \geq 0$ if the two nodes are directly connected

h = maximum number of links in a path at the current stage of the algorithm

$L_h(n)$ = cost of the least-cost path from node s to node n under the constraint of no more than h links

1. **[Initialization]**

$$L_0(n) = \infty, \text{ for all } n \neq s$$
$$L_h(s) = 0, \text{ for all } h$$

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 the predecessor node j that achieves the minimum, and eliminate any connection of n with a different predecessor node formed during an earlier iteration. The path from s to n terminates with the link from j to n.

For the iteration of step 2 with $h = K$, and for each destination node n, the algorithm compares potential paths from s to n of length $K + 1$ with the path that existed at the end of the previous iteration. If the previous, shorter, path has less cost, then that path is retained. Otherwise a new path with length $K + 1$ is defined from s to n; this path consists of a path of length K from s to some node j, plus a direct hop from node j to node n. In this case, the path from s to j that is used is the K-hop path for j defined in the previous iteration (see Problem 12.5).

Table 12.2b and Figure 12.10 show the result of applying this algorithm to Figure 12.1, using $s = 1$. At each step, the least-cost paths with a maximum number of links equal to h are found. After the final iteration, the least-cost path to each node and the cost of that path have been developed. The same procedure can be used with node 2 as source node, and so on. Note that the results agree with those obtained using Dijkstra's algorithm.

Comparison

One interesting comparison can be made between these two algorithms, having to do with what information needs to be gathered. Consider first the Bellman-Ford algorithm. In step 2, 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., $L_h(j)$]. 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

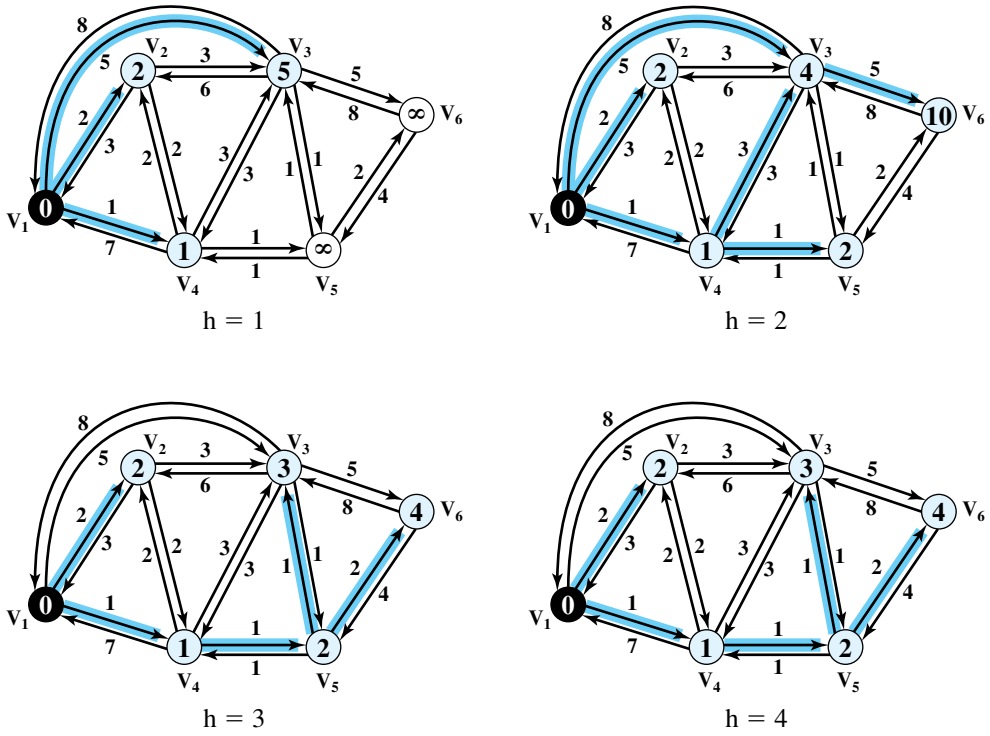


Figure 12.10 Bellman-Ford Algorithm Applied to Graph of Figure 12.1

costs and paths. On the other hand, consider 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.

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, and 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.

12.4 RECOMMENDED READING

[MAXE90] is a useful survey of routing algorithms. Another survey, with numerous examples, is [SCHW80].

[CORM01] contains a detailed analysis of the least-cost algorithms discussed in this chapter. [BERT92] also discusses these algorithms in detail.

BERT92 Bertsekas, D., and Gallager, R. *Data Networks.* Upper Saddle River, NJ: Prentice Hall, 1992.

CORM01 Cormen, T., et al. *Introduction to Algorithms.* Cambridge, MA: MIT Press, 2001.

MAXE90 Maxemchuk, N., and Zarki, M. "Routing and Flow Control in High-Speed Wide-Area Networks." *Proceedings of the IEEE*, January 1990.

SCHW80 Schwartz, M., and Stern, T. "Routing Techniques Used in Computer Communication Networks." *IEEE Transactions on Communications*, April 1980.

12.5 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

adaptive routing	Dijkstra's algorithm	least-cost algorithms
alternate routing	fixed routing	random routing
Bellman-Ford algorithm	flooding	

Review Questions

12.1. What are the key requirements for a routing function for a packet-switching network?

12.2. What is fixed routing?

12.3. What is flooding?

12.4. What are the advantages and disadvantages of adaptive routing?

12.5. What is a least-cost algorithm?

12.6. What is the essential difference between Dijkstra's algorithm and the Bellman-Ford algorithm?

Problems

12.1 Consider a packet-switching network of N nodes, connected by the following topologies:

a. Star: one central node with no attached station; all other nodes attach to the central node.

b. Loop: each node connects to two other nodes to form a closed loop.

c. Fully connected: each node is directly connected to all other nodes.

For each case, give the average number of hops between stations.

12.2 Consider a binary tree topology for a packet-switching network. The root node connects to two other nodes. All intermediate nodes connect to one node in the direction toward the root, and two in the direction away from the root. At the bottom are nodes with just one link back toward the root. If there are $2^N - 1$ nodes, derive an expression for the mean number of hops per packet for large N, assuming that trips between all node pairs are equally likely. *Hint:* You will find the following equalities useful:

$$\sum_{i=1}^{\infty} X^i = \frac{X}{1 - X}; \sum_{i=1}^{\infty} iX^i = \frac{X}{(1 - X)^2}$$

12.3 Dijkstra's algorithm, for finding the least-cost path from a specified node *s* to a specified node *t,* can be expressed in the following program:

```
for n := 1 to N do
    begin
        L[n] := ∞; final[n] :=  false; {all nodes are temporarily labeled with
        ∞} pred[n] :=  1
    end;
L[s] := 0; final[s] := true;    {node s is permanently labeled with 0}
recent := s;                    {the most recent node to be permanently labeled is s}
path :=  true;
{initialization over }

while final[t] = false do
begin
    for n := 1 to N do      {find new label}
        if (w[recent, n] < ∞) AND (NOT final[n]) then
        {for every immediate successor of recent that is not permanently labeled, do }
            begin {update temporary labels}
                newlabel := L[recent] + w[recent,n];
                if newlabel <L[n] then
                        begin L[n] :=  newlabel; pred[n] := recent end
                        {re-label n if there is a shorter path via node recent and make
                        recent the predecessor of n on the shortest path from s}
            end;
temp := ∞;
 for x := 1 to N do {find node with smallest temporary label}
    if (NOT final[x]) AND (L[x] < temp) then
            begin y := x; temp :=L[x] end;
if temp < 8 then {there is a path} then
    begin final[y] := true; recent := y end
    {y, the next closest node to s gets permanently labeled}
else begin path := false; final[t] := true end
end
```

In this program, each node is assigned a temporary label initially. As a final path to a node is determined, it is assigned a permanent label equal to the cost of the path from s. Write a similar program for the Bellman-Ford algorithm. *Hint:* The Bellman-Ford algorithm is often called a label-correcting method, in contrast to Dijkstra's label-setting method.

12.4 In the discussion of Dijkstra's algorithm in Section 12.3, it is asserted that at each iteration, a new node is added to *T* and that the least-cost path for that new node passes only through nodes already in *T.* Demonstrate that this is true. *Hint:* Begin at the beginning. Show that the first node added to *T* must have a direct link to the source node. Then show that the second node to *T* must either have a direct link to the source node or a direct link to the first node added to *T,* and so on. Remember that all link costs are assumed nonnegative.

12.5 In the discussion of the Bellman-Ford algorithm, it is asserted that at the iteration for which $h = K$, if any path of length $K + 1$ is defined, the first K hops of that path form a path defined in the previous iteration. Demonstrate that this is true.

12.6 In step 3 of Dijkstra's algorithm, the least-cost path values are only updated for nodes not yet in *T.* Is it possible that a lower-cost path could be found to a node already in *T?* If so, demonstrate by example. If not, provide reasoning as to why not.

12.7 Using Dijkstra's algorithm, generate a least-cost route to all other nodes for nodes 2 through 6 of Figure 12.1. Display the results as in Table 12.2a.

12.8 Repeat Problem 12.7 using the Bellman-Ford algorithm.

12.9 Apply Dijkstra's routing algorithm to the networks in Figure 12.11. Provide a table similar to Table 12.2a and a figure similar to Figure 12.9.

12.10 Repeat Problem 12.9 using the Bellman-Ford algorithm.

12.11 Will Dijkstra's algorithm and the Bellman-Ford algorithm always yield the same solutions? Why or why not?

12.12 Both Dijkstra's algorithm and the Bellman-Ford algorithm find the least-cost paths from one node to all other nodes. The Floyd-Warshall algorithm finds the least-cost paths between all pairs of nodes together. Define

N = set of nodes in the network

$w(i, j)$ = link cost from node i to node j; $w(i, i) = 0$; $w(i, j) = \infty$ if the two nodes are not directly connected

$L_n(i, j)$ = cost of the least-cost path from node i to node j with the constraint that only nodes $1, 2, \ldots, n$ can be used as intermediate nodes on paths

The algorithm has the following steps:

1. Initialize:

$$L_0(i, j) = w(i, j), \text{ for all } i, j, i \neq j$$

2. For $n = 0, 1, \ldots, N - 1$

$$L_{n+1}(i, j) = \min[L_n(i, j), L_n(i, n + 1) + L_n(n + 1, j)] \quad \text{for all } i \neq j$$

Explain the algorithm in words. Use induction to demonstrate that the algorithm works.

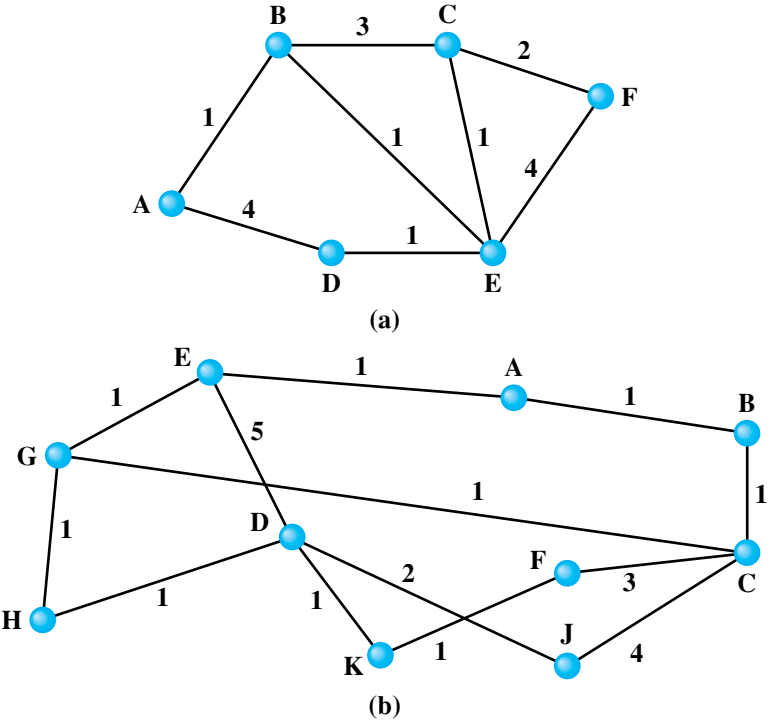


Figure 12.11 Packet-Switching Networks with Link Costs

12.13 In Figure 12.3, node 1 sends a packet to node 6 using flooding. Counting the transmission of one packet across one link as a load of one, what is the total load generated if

a. Each node discards duplicate incoming packets?

b. A hop count field is used and is initially set to 5, and no duplicate is discarded?

12.14 It was shown that flooding can be used to determine the minimum-hop route. Can it be used to determine the minimum delay route?

12.15 With random routing, only one copy of the packet is in existence at a time. Nevertheless, it would be wise to utilize a hop count field. Why?

12.16 Another adaptive routing scheme is known as backward learning. As a packet is routed through the network, it carries not only the destination address, but the source address plus a running hop count that is incremented for each hop. Each node builds a routing table that gives the next node and hop count for each destination. How is the packet information used to build the table? What are the advantages and disadvantages of this technique?

12.17 Build a centralized routing directory for the networks of Figure 12.11.

12.18 Consider a system using flooding with a hop counter. Suppose that the hop counter is originally set to the "diameter" of the network. When the hop count reaches zero, the packet is discarded except at its destination. Does this always ensure that a packet will reach its destination if there exists at least one operable path? Why or why not?