[10] Luss, H., "Operations research and capacity expansion problems: a survey," *Operations Research*, vol. 30, pp. 907–947, 1982.

[11] Frank, H., Frisch, I.T., Van Slyke, R., and Chou, W.S., "Optimal design of centralized computer networks," *Networks*, vol. 1, pp. 43–57, 1971.

[12] Yaged, B., "Minimum cost routing for static network models," *Networks*, vol. 1, pp. 139–172, 1971.

[13] Yaged, B., "Minimum cost routing for dynamic network models," *Networks*, vol. 3, pp. 193–224, 1973.

[14] Hayes, J.F., *Modeling and Analysis of Computer Communications Networks*, Plenum, 1984.

CHAPTER 2

Routing Techniques and Models

The performance of a network depends on such factors as the network configuration, the offered load, and the network management methods. An important element of network management, called *network routing*, consists of the decision rules used to connect the calls as they arrive at the network; a variety of methods are now possible. We describe in depth some of the methods that are currently in use or proposed for implementation in large circuit-switched networks. We also introduce useful graphical models to represent these methods, and briefly cover some mathematical models that can describe them. The question of evaluating the performance of a particular network operating under one of these algorithms is discussed in Chapter 4; the question of choosing the optimal routing method, in Chapter 7.

2.1 Definition of Routing

Consider the real-time operation of a network. Whenever a new call arrives at an office, we must determine whether there exists a path on which the call can be connected to its destination. If a path is found, we must decide whether to use it or not; if there is no available path, we must decide what to do with the call. Each of these steps is related to a different aspect of routing.

The selection of an available path constitutes the routing problem in the strict sense, requiring the paths available for each stream to be defined — an important part of the specification of a routing technique. This definition could be the set of all paths permitted by the network structure or, more simply, some convenient subset. It must also include a description of a procedure for finding such a path and for selecting which path to use if more than one is available.

The decision of whether or not to connect a call on an available path is often called *flow control*. Flow control can be viewed as a special case of routing, where lost calls are routed on a fictitious path of infinite capacity. Although the flow-control problem has been extensively studied for packet networks, no comparable work exists for circuit networks. In this book we do not consider the flow-control problem, but rather assume that a call that can be connected always will be connected.

The last step of call routing deals with handling blocked calls, that is, those for which no path is available. In general, we assume that such calls are lost and do not return. This implies that the call-retrial rate is negligible — probably accurate for a network with a low blocking rate. In networks with high blocking, for instance in the case of failure or overload, call reattempt can be significant. The blocked-calls-lost option is not the only option available: Many networks, mostly private ones, may operate on the basis of blocked calls delayed. We shall not consider this case any further since the corresponding models are much more difficult than those in this book, involving complex protocol issues that cannot be considered here.

Note that call termination has not been mentioned as part of the routing problem. We assume that a call that terminates has no effect on calls already in progress, and that it can influence future calls only by freeing up a path in the network. This is equivalent to saying that there is no rearrangement of calls in progress in the network. Although this option has not been implemented in any real network and is still the subject of research, it is becoming a possibility with the advent of faster switches.

Given these considerations, let us define more explicitly what we mean by a routing technique (sometimes called a routing algorithm or method), or, using the terminology of optimal control, a routing policy. A routing technique is a set of rules that specifies, for each new call arriving in the network, on what path to route the call, and the data required to make that choice; it is the same as the definition of a process in computer science terminology.

We do not claim to present a complete classification of routing methods. There is no consensus on the precise meaning of many of the terms used to characterize routing, and another attempt at this topic would probably lead far from our objective. It is more important to obtain a clear understanding of the operation of the routing methods that are effectively used in networks than to set up a very general framework that attempts to cover all potential cases of routing but is of little practical use. The concepts of dynamic and adaptive routings, however, appear often enough that we should discuss their meanings in this book.

In this book, a dynamic routing is one where a part of the routing varies over time, while an adaptive routing is one where some part of the routing is a function of some estimate of the network state at the time a decision must be made. The point is that a dynamic routing is not necessarily adaptive; in fact, one dynamic routing method currently being implemented in a real telephone network does not use state measurements and would not be classified as an adaptive method here. The converse is somewhat more unlikely. An adaptive routing is normally dynamic unless the network is so quiet that the state is not changing at all. Furthermore, implicit in the notion of adaptive routing is the notion of measurement of the network state, which is not required for purely dynamic routings.

Both dynamic and adaptive routings have some time-varying component: the path-selection rules, the data for the algorithm, or both. To further complicate matters, a given routing method may be viewed as static or dynamic, depending on which parts of the routing algorithm are used to characterize it. A case in point is the old alternate routing method, which, although generally considered static, can be viewed as an adaptive method if the actual paths selected are used as a criterion for deciding whether the algorithm is dynamic or not. Although these issues appear primarily semantic, they appear sufficiently often that the distinction between dynamic and adaptive routings should always be made explicit. In particular, saying that an algorithm is static or dynamic does not mean much by itself. One should specify with respect to which component of the routing such a statement is made, since some parts may be dynamic, while others may be static or adaptive. The examples presented in Section 2.4 will clarify these points.

2.2 The Continuous-Time Markov Process

Routing algorithms are frequently described qualitatively and thus are subject to ambiguity and do not lend themselves to quantitative analysis. There is, however, a well-known framework within which all these notions can be made more precise: the theory of continuous-time Markov processes and of Markovian decision theory.

The Markov model for circuit-switched networks follows from two standard assumptions about external traffic: (1) the calls arrive according to independent Poisson processes, and (2) the call-holding times are independent, exponentially distributed random variables. To simplify notation, the time unit is assumed to be equal to the mean holding time. Thus both the arrival and the service processes are memoryless stochastic processes; provided we can define a state space and a transition matrix, a continuous-time Markov process formulation of the network operation is possible. This approach would seem promising in view of the large body of knowledge on Markov processes. It is used mostly for theoretical understanding, however, since it is impractical for analytical or numerical calculations when considering networks of realistic sizes. In this section, we first review some of the elements of Markov theory using the notation of [1,2]. Then we show how the general theory can be used to describe the routing of calls in circuit-switched networks, also indicating the practical difficulties encountered.

Continuous-Time Markov Processes

The finite state continuous-time Markov process is defined by a set of states labeled $i = 1, 2, \ldots N_s$ and a transition-rate matrix $Q_{i,j}$ defined for each pair

of states i, j, $i \neq j$. The transition-rate matrix is such that the probability of occurrence of a transition from i to j in a small time interval dt is precisely $Q_{i,j}dt$. The diagonal element is defined by $Q_{i,i} = 1 - \sum_{j \neq i} Q_{i,j}$. Suppose now that the system is started in some state k. The probability of being in state i at time t depends on k, t, and Q. To simplify the notation, we assume that k is given and known and thus drop this index from the notation. The probability of being in state i at time t is then denoted $p_i(t)$. It is given by the matrix differential equation

$$\frac{dp_i(t)}{dt} = \sum_l p_l Q_{l,i}. \tag{2.1}$$

It is well-known that these probabilities take an asymptotic value p_i that depends only on the initial state; they are given by the equation $\mathbf{p}Q = 0$. There is such a probability vector $\mathbf{p}$ for each initial state, and we define the probability matrix S such that its i^{th} row is made up of the $\mathbf{p}$ vector corresponding to starting the system in state i. There are as many distinct $\mathbf{p}$ vectors as there are irreducible classes of states; in particular, the S matrix has all rows identical if there is only one such class.

It is sometimes useful to place a value on a Markov process. This is done via the transition values $r_{i,j}$, which are the revenue produced by a transition from state i to state j. In addition, there may be some value to simply being in a state. We define the value of a state $r_{i,i}$ as the *rate* at which revenue is generated while the system is in state i. Note that $r_{i,j}$ and $r_{i,i}$ have different units: the former is a revenue; the later, a revenue per unit time. Finally, the earning *rate* of the process in state i is defined by $q_i = r_{i,i} + \sum_{j \neq i} Q_{i,j} r_{i,j}$. The quantities of interest are the total revenue expected from operating the system until time t, given that the initial state was i. These quantities, called the value of state i, are represented by $v_i(t)$. It is not hard to show that the value vector $\mathbf{v}(t)$ is given by the differential equation

$$\frac{d\mathbf{v}(t)}{dt} = \mathbf{q} + Q\mathbf{v}(t).$$

This equation can readily be solved in the Laplace transform domain. We have

$$\mathbf{v}(s) = \frac{(sI - Q)^{-1}}{s}\mathbf{q} + (sI - A)^{-1}\mathbf{v}(0),$$

and it can be shown that

$$(sI - A)^{-1} = \frac{1}{s}S + T(s),$$

from which we get the asymptotic value

$$\mathbf{v}(t) = \mathbf{g}t + \mathbf{v},$$

where

$$\mathbf{g} = S\mathbf{q}$$
$$\mathbf{v} = T(0)\mathbf{q} + S\mathbf{v}(0)$$

In these equations, the T matrix is the Laplace transform of the transient terms of the differential equation (2.1). The vector $\mathbf{g}$, called the gain or rate vector, denotes the rate at which the system is generating revenue for all initial states. This is obviously the quantity of interest since it determines the long-term value of the system. It is the natural objective to maximize in those cases where there exists a possibility of choosing such parameters of the process as the transition-rate matrix or the revenues.

Application of the Markov Model to Routing

Following the work of Beneš [3], let us now examine how the Markov model can be used to represent routing in circuit-switched networks. In the following, we assume that all the paths between each origin-destination pair are known and listed in some fixed order. This amounts to saying that $\mathcal{I}$, the arc-path incidence matrix for the network, is given; we denote a column of this matrix by the vector $\mathbf{a}$.

First we define the state space. Using the standard multicommodity flow representation, let $x_m^{o,d}(t)$ = number of calls in progress on the m^{th} path between nodes o and d at time t. These variables are constrained to take integer values only. A state is defined as a path-flow vector $\mathbf{x}(t)$ that specifies the number of calls on each path in the network at time t. We use the notation $\mathbf{x}_j$ to denote a particular state vector, and the scalar x_i^j for the number of calls in progress on the i^{th} path of the network for this particular state j. Here $\mathbf{x}_j$ denotes a *state*, that is, a complete multicommodity flow in the network, with the specification of calls carried on all the paths. On the other hand, x_i^j is defined for a particular state j and is a scalar depending on the particular path chosen in this state. Although the dependence on the state is included in the notation, in practice the state index is omitted whenever there is no ambiguity as to the state in question; this allows us to simplify an already complex notation.

The state variables are constrained by the equations $\mathcal{I}\mathbf{x} \leq \mathbf{N}$, the capacity constraints on the number of calls on each arc. The state $\mathbf{x} = 0$, called the empty state, is denoted $\emptyset$. Equivalent representations are also possible (see Problem 2.2). An auxiliary quantity of interest is $n_j(t)$, the total number of calls in progress when the network is in state $\mathbf{x}_j$; this is given by $n_j(t) = \sum_l x_l^j(t)$.

Denote the state space $\mathcal{S} = \{\mathbf{x}_j\}$. An arbitrary state i is connected to the empty state $\emptyset$ since it is always possible to empty the network by a sufficient

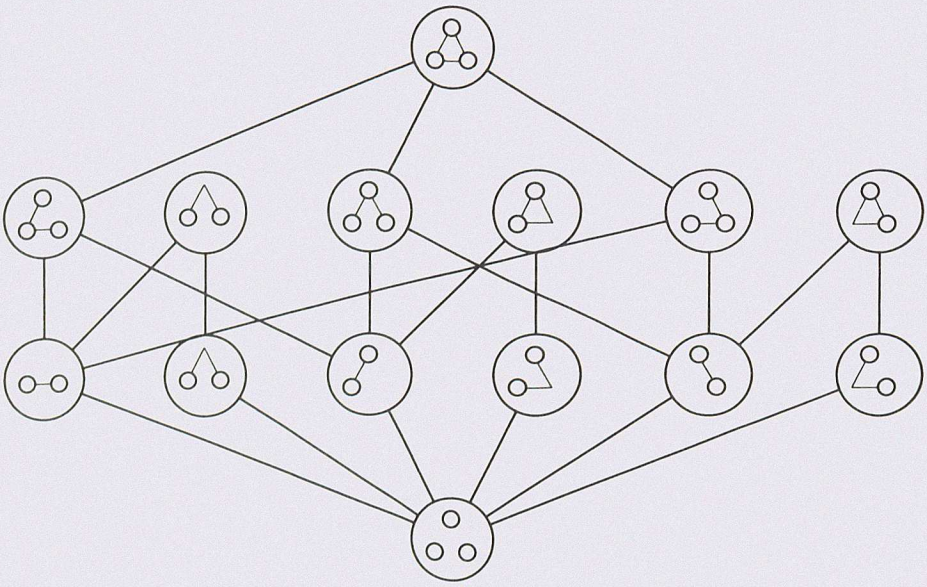

Figure 2.1 Hasse Diagram for Three-Node Network, Single Link per Arc

number of call terminations. The converse is not true, however, since starting from $\emptyset$ it may be impossible to reach some state j, either because this is not permitted by the routing rules or because there is no external traffic between some origin-destination pairs. Let $\mathcal{S}'$ be the set of states i that can be reached from $\emptyset$. $\mathcal{S}'$ forms an irreducible, aperiodic class, while its complement $\mathcal{S} - \mathcal{S}'$ forms a set of transient states. (In fact, these states will never occur if the network is started from $\emptyset$; otherwise they will eventually disappear and never reoccur.) For the sake of simplicity, we assume from now on that the transient states have been removed from consideration and that $\mathcal{S}'$ is the state space. In this case, the system is ergodic and has a single set of stationary state probabilities. From these, we can compute, for each origin-destination pair (i, j), the fraction of the time during which no path is available to connect a call. Called the *end-to-end loss probability* and denoted $L^{i,j}$, this important measure of network performance is discussed in later chapters.

The Hasse diagram provides a useful graphical representation of the partial order between states. The example shown in Fig. 2.1 corresponds to the three-node network with a single two-way trunk between each node. In this diagram, the states are ordered in layers, with the states in one layer produced from the states in the layer below by the addition of a single call. Each state is

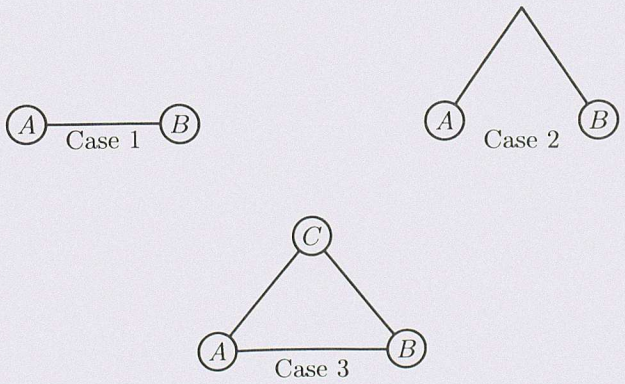

Figure 2.2 Description of Network States

represented by the graph of the network, with each call in progress in the state indicated by an arc in this graph. The various state configurations have the following interpretation. A direct call between nodes A and B is represented by the first case of Fig. 2.2, and a tandem call between these two nodes via node C by the second case. Note that in this case, the node C is not shown on the graph, in contrast to the case of three direct calls (A, B), (A, C), and (B, C), where node C now appears in the graph (see the third case in Fig. 2.2).

Potential transitions between states are indicated by an arc linking the large nodes. Whether such an arc represents an actual transition depends on the particular routing method used. In fact, the specification of a routing is equivalent to the selection of the set of arcs that appear in the transition diagram.

This set contains a unique aperiodic recurrent class; as a consequence, there is a *unique* set of stationary state probabilities p_i independent of the original state, which can be found by the standard equation $\mathbf{p}Q = 0$. Thus there is a *unique* average blocking probability. The fact that these probabilities are unique is important because of the behavior of real networks operating under simplified routing rules. Such networks seem to operate as if there were effectively more than a single value for these probabilities. This behavior is described in more detail in Section 4.4, where a plausible explanation is offered for the observed results.

Having defined the states of the Markov process, we now need the transition-rate matrix to compute the state probabilities. The transition-rate matrix can be constructed according to the following rules. For simplicity, we assume that a call is connected whenever a path is available for it at the time of its arrival. Because of the second-order effect of simultaneous events, transitions can oc-

cur only between adjacent states, that is, for states $\mathbf{y} = \mathbf{x} \pm \mathbf{e}$, where $\mathbf{e}$ is an elementary vector in the space of multicommodity flows.

1. For a call departure, j contains exactly one call less than i between some pair of nodes o and d. Assuming that a call termination does not cause any rerouting of calls in progress, $Q_{i,j}$ is effectively the rate at which a connected call will terminate.

$$Q_{i,j} = \begin{cases} \dfrac{\sum_k x_k^{o,d}}{\sum_{(o,d)} \sum_k x_k^{o,d}} \times Q_{dep} & \text{if } \mathbf{x}_j = \mathbf{x}_i - \mathbf{e} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

where Q_{dep} is the transition rate of call terminations in the network. Because we assume that call-holding times are independent and exponentially distributed, we can write

$$Q_{dep} = \frac{n_i}{n_i + \gamma}. \tag{2.3}$$

Here γ is the total arrival rate of new calls into the network and is given by $\gamma = \sum_l A^l$.

2. For a call arrival, j contains exactly one more call than i, between nodes o and d. Q is then the rate at which an (o, d) call will arrive, which is known from the traffic matrix

$$Q_{i,j} = \begin{cases} \dfrac{A^{o,d}}{\sum_{m,n} A^{m,n}} \times Q_{arr} & \text{if } \mathbf{x}_j = \mathbf{x}_i + \mathbf{e} \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

where Q_{arr} is the overall arrival rate for new calls in the network and is given by

$$Q_{arr} = \frac{\gamma}{n_i + \gamma}. \tag{2.5}$$

The construction of the transition-rate matrix is subject to constraints that reflect the nature of the system and show up as the *if* statement in the first part of Eq. (2.4). The first set of such conditions simply reflects that a transition cannot be made if the total number of calls in progress on all links exceeds the number of circuits on that link. These conditions can be expressed, with the aid of the arc-path incidence matrix, as

$$\mathcal{I}\mathbf{x}_j(t) \leq \mathbf{N}, \quad j \in \{\mathcal{S}\}, \ \forall t, \tag{2.6}$$

where $\mathbf{N}$ denotes the vector of arc capacities. The second set (the most important one, from our point of view) is any other set of conditions that can be

imposed on the connection process. These conditions are generally expressed by some rule or by a verbal description. Because such rules determine the transition matrix, this matrix is the mathematical object that corresponds to these routing rules, to the extent that the matrix *is* the routing. In other words, knowledge of the matrix determines completely how new calls are to be connected for any state. Conversely, if we have described the routing by some rule, then the matrix must be constructed in such a way that only those transitions permitted by the rule, and all of these, are found in the matrix. Which of these methods we choose is irrelevant from a theoretical point of view, since they produce the same effect. This dual way to represent the routing is in fact a result of the Markov process representation we are using for the network operation — no surprise to the reader who is already familiar with Markov decision theory, since it corresponds to the specification of a *policy*, which can be done equally well by a rule or a transition matrix.

Given the state space and the transition matrix, we can, at least in theory, compute the state probabilities of any network given the input values such as the means of the arrival processes and the size of the groups. This in turn yields the end-to-end loss probabilities and the carried traffic, which are the measures used most frequently of a network's performance. In some cases, we may want a more flexible performance measure; we can do this by giving a value to the Markov process.

There is a great deal of latitude in selecting a value for the process that describes the operation of a network. A simple case is to take $r_{i,j} = 0\ i \neq j$, and $r_{i,i} = n_i$ as the rate of reward for having n_i calls in progress in the network. Under this condition, we have $q_i = n_i$; since there is only one set of probabilities, there is a single gain g given by $g = \sum_i p_i q_i = \overline{n}$. As expected, the gain of the network — that is, the rate at which reward is being incurred — is simply the expected number of calls in progress in the network. This is a reasonable measure of network efficiency and a reasonable objective for maximization. More complex revenues can be defined that may correspond to the actual monetary value of a call carried in the network; such revenues may be chosen artificially in order to perform some management function. Some applications of these revenues are given in Chapter 7, where we discuss the optimization of routing.

The Markov model for routing requires a very detailed description of the network state; the amount of information required for this formulation grows very quickly with the size of the network. As an example, consider the networks shown in Fig. 2.3. The first three have identical traffic between all pairs of nodes and a single two-way trunk between each node. The fourth network has a single destination node D, three traffic streams, and N_{o,T_1}, N_{o,T_2}, $N_{T_1,d}$, and $N_{T_2,d}$ trunks. These values are indicated in Table 2.1, which shows the number of states for each case. It is quite obvious that attempting to use this model for performance evaluation or routing optimization entails the manipulation of some very large matrices, which soon becomes infeasible for any network larger

Number of States with n Calls	Network 1	Network 2	Network 3	Network 4 (3,4,12,10)	Network 4 (3,4,7,10)	Network 4 (3,4,12,7)
0	1	1	1			
1	6	13	20			
2	6	44	128			
3	1	44	319			
4	-	13	319			
5	-	1	128			
6	-	-	20			
7	-	-	1			
Total	14	116	936	2070	1170	1380

Table 2.1 Number of States for Small Networks

than a few nodes. Nevertheless, the Markov models have their use. They clarify the nature of some concepts such as routing by putting them in the familiar framework of a transition-rate matrix. They also indicate the complexity of such problems as network performance evaluation and routing optimization, showing that approximations are absolutely essential and that exact solutions are infeasible. Finally, the Markov models can act as the framework in which these approximations can be constructed systematically, instead of ad hoc, as has generally been the case.

Multicommodity Flows

The Markov description of the network is too complex to be used for practical cases. The fact that the system we are studying has a network structure suggests that other, more compact representations could be used for numerical and analytical calculations. We already suggested this in defining the states in terms of integer multicommodity flow vectors. These flow models occur in many areas of engineering; in some simple cases, they lead to some of the most efficient optimization algorithms known. It therefore seems that a flow description of the network could also lead to efficient numerical methods.

First we must define an expanded network. Assume that each node i that receives external calls is split into two nodes i_1 and i_2, and that these two nodes are connected by an arc with infinite capacity. Calls of stream l arriving at node i enter the expanded network at node i_1 and begin service immediately. Let $y^l(t)$ be the number of calls of stream l in progress on link i_1, i_2 at time t. The $\mathbf{y}(t)$ process generates calls at node i_2 that have exactly the same statistics

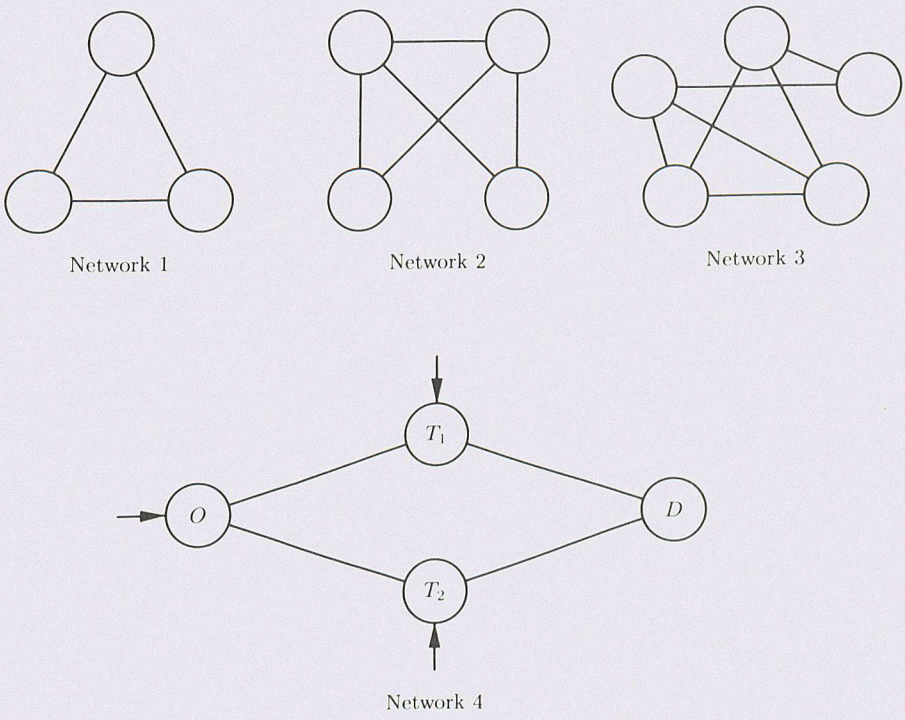

Figure 2.3 Networks for Markov Model Examples

as the external calls in the real network. In particular, we have $\overline{y}^l = A^l$. The calls that arrive at node i_2 are treated exactly as if they were the true external calls, some of which are routed through the network and some of which are rejected. Assume further that the calls rejected at node i_2 remain on link i_1, i_2 for the normal duration of their service time. Within this augmented network, we now have the following set of linear inequalities:

$$\mathcal{I}\mathbf{x}(t) \leq \mathbf{N} \quad \forall t \geq 0 \tag{2.7}$$

$$y^l(t) \geq \sum_m x^l_m(t) \quad \forall t \geq 0 \tag{2.8}$$

Equations (2.7) and (2.8) are very similar to those defining a multicommodity flow, the only difference being the inequality condition in Eq. (2.8); they define

a stochastic multicommodity flow. Because they are linear, we can take the expectations on both sides, obtaining a *deterministic* multicommodity flow corresponding to the *expected* values of the number of calls carried in the network. We have

$$\mathcal{I}\overline{\mathbf{x}} \leq \mathbf{N} \tag{2.9}$$

$$\overline{y}^k = A^k \geq \sum_m \overline{x}^k_m, \tag{2.10}$$

which we can rewrite

$$\overline{y}^k = L^k A^k$$
$$\mathbf{x} \geq 0,$$

where L^k is defined as the end-to-end blocking probability of stream k and x^k is the vector of *carried traffic* on the network paths for the stream k. Note that both x^k and L^k are functions not only of the network parameters such as the offered traffic and the group sizes, but also of the routing. A major topic of this book is the calculation of these quantities and their optimization with respect to some of these parameters. There will be many occasions to return to these flow models in later chapters.

Because of the complexity of the Markov model, routing algorithms are generally more simple than the full description of the transition matrix, or than a policy described in terms of the complete state space. The simplification takes three forms. First, the set of paths available is reduced considerably from the full set allowed by the network topology. Second, the path-selection rules are changed so that some transitions possible in the full Markov description are forbidden. Finally, the states are merged into aggregates, thus reducing the amount of information available for the routing decisions. We now describe some of the more widely used techniques, as well as some proposed ones that are likely candidates for implementation in future networks.

2.3 Load Sharing

The first routing policy we examine, called *load sharing*, is not implemented in any network and is not very efficient. Load sharing is described here, however, because it is the policy for which we have the largest collection of theoretical results. Also, it has some nice mathematical properties that will be quite handy in our discussion of numerical methods for performance evaluation and optimization of more realistic policies.

In its simplest form, load sharing works as follows. For traffic stream k, a set of R_k paths is given that constitute the only paths available to route the calls of stream k. The stream is partitioned into R_k substreams, where the calls

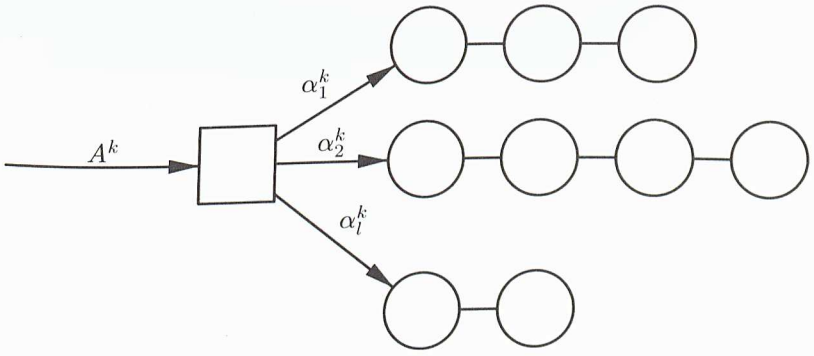

Figure 2.4 Graphical Representation of Load Sharing

from substream l are offered to path l. If the path is not free, the call is lost. In practice, this separation can be done by drawing from a uniform distribution with probabilities $\alpha_l^k, | \sum_l \alpha_l^k = 1$ whenever a new call arrives from stream k. In this way, the statistical properties of the real stream are maintained in the substreams — in particular, if the arrival is Poisson in the real stream, then so are the arrivals in the substreams. Graphically, load sharing in represented by giving, for each stream k, the list of paths available. As shown in Fig. 2.4, the separation of the original stream is represented by a square node.

The advantages of this method are that the probabilities belong to a compact set and that optimization can be carried out using standard nonlinear programming methods. Also, as we shall see later, the sharing coefficients need not be fixed, but may be made to depend on state measurements.

2.4 Alternate Routing

An obvious defect of load sharing with fixed coefficients is that once the identity of a path is drawn, there is no possibility of selecting any other path. If the selected path turns out to be unavailable, the call is lost, although another path may be free that could be used to connect the call. For this reason, pure load sharing is not a very good routing algorithm and must be improved to be usable in real networks. The improvement is to provide, for each call arrival, the possibility of selecting a path from a given set, depending on some conditions. This in turn gives rise to two broad classes of routing algorithms. In the first class, the set is ordered, forming a *sequence* of potential choices. The rule is to pick the first path in the sequence that can carry the call. This

process, known as *alternate routing*, comes in many flavors, depending on how the sequences are chosen.

The second class consists of choosing the path according to a value placed on each path in the set. These values can be fixed, but usually are computed from observations of some of the network components. For this reason, they are called *adaptive* methods.

The somewhat artificial distinction between alternate routing and adaptive techniques is a good example of the difficulty of classifying routing in rigidly defined categories. Alternate routing can be viewed as a special case of adaptive routing, where the path values are given by the following rule: The value of the paths is the negative of the order number in the sequence, and paths that are not available are given a value of $-\infty$. In this way, selecting the path with the largest value replicates the operation of alternate routing.

In the same way, alternate routing is in a sense adaptive since the actual path chosen for a call depends on the state of the network as reflected in the availability of the paths. In other words, alternate routing also adapts to changing conditions, and thus it could equally well be viewed as an adaptive method.

Because these methods are used in very different networks, however, we maintain the difference in their traditional meanings. Finally, as we shall see later, it is possible to mix the two classes to obtain a large variety of hybrid methods.

General Fixed Alternate Routing

Fixed alternate routing is one of the oldest and simplest routing techniques. Using this method, the first reduction in complexity over the Markov model is achieved by specifying in advance, for each origin-destination pair, a sequence of paths that are candidates for connecting the call. This set is generally much smaller than the full set of network paths allowed by the network topology. The second reduction is obtained by severely limiting the possible number of transitions: The set of alternate routes is scanned in some predetermined order, and the call is connected on the first free path that is found. (A path is said to be free if each of its links has at least one free trunk.) If a given path is not free, either the call can be lost or the next path may be tried. In this case, we say that the call *overflows* on the next path in the sequence. Which event happens depends on the type of routing control used and the particular link that was unavailable.

In the example in Fig. 2.5, the paths available for connecting (o, d) calls are, in order, (o, d), (o, a, d), (o, a, c, d), (o, a, e, f, d), and finally (o, b, e, f, d). Suppose now that links (o, d) and (a, d) are blocked and that links (o, a), (a, c) are available. It is possible to establish a partial connection from o to e through node a; we say that the call has progressed to node c. If link (c, d) is blocked, the

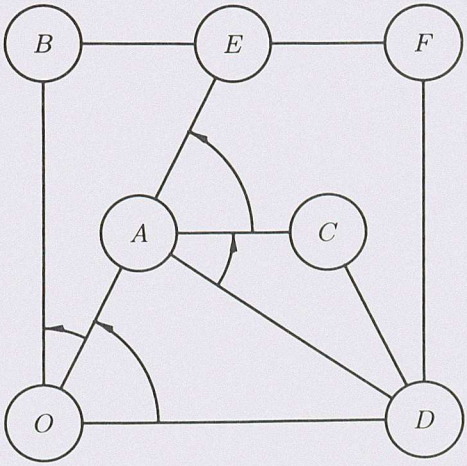

Figure 2.5 Alternate Routing and Overflow

call may be lost or may attempt the next path in the list, that is, (o, a, e, f, d). Which event occurs determines a particular variation of the alternate routing scheme.

Alternate routing schemes may be static or dynamic, depending on whether the sequence of paths is static or time-dependent. If the sequences are chosen as a function of the network state, they can also be adaptive. For now, suffice to say that fixed alternate routing is an alternate routing scheme where the path sequences are constant over long time periods, as measured with respect to the average call-holding time. We now describe in some detail two special cases of fixed alternate routing: hierarchical and two-link.

Fixed Hierarchical Routing

Fixed hierarchical routing is the oldest and most extensively used routing method for telecommunication networks. It originated with the old wired-logic switches, which had a severely limited capability for alternate routing, and thus in effect dictated the method that could be used. Although this limitation was soon overcome with the advent of stored-program control switches, by that time fixed hierarchical routing had become entrenched. It is used in most telephone networks to this day.

As its name indicates, fixed hierarchical routing is of the fixed alternate route type, with additional restrictions imposed on the nature of the paths that can be used for a call. The hierarchical character of the method arises from

a partition of the offices into classes, five altogether in the North American network, and somewhat less in other countries (See Fig. 2.6 for the five-level network. Not all possible groups are indicated.) The class-five offices, the so-called *end offices*, are those to which subscriber lines are directly connected. Going up in the hierarchy, the toll, primary, sectional, and regional centers are encountered. Each office has and one and only one homing office of higher level to which it is connected by a so-called *final trunk* group (the reason for this name will become obvious soon). Of course, the office can be connected to other offices in the network by *high-usage* groups. Thus, given any pair of offices, there is a unique path, made up exclusively of final trunk groups, that connects them. We call such a path the hierarchical path, or sometimes the normal path. The hierarchical distance between two offices is defined as the number of groups needed to go from one to the other on the hierarchical path. For instance, in a full five-level hierarchy, the hierarchical distance between two end offices is 9, and the hierarchical distance between a regional office and an end office is 5.

Standard hierarchical routing operates within such a hierarchy of offices, with a few simple rules that can easily be implemented with limited hardware — the reason why this method was used in the first place. Although these rules are not uniform for all networks, in general they are simple variations on the following *fan rule*:

1. For a call arriving in an office, only the destination office is available to make the routing decision. The origin of the call is immaterial.

2. For a call arriving in an office, route it over the first available group in a sequence of alternate routes.

3. The sequence of alternate routes is defined by the end office of the group. The rule is to attempt the group whose extremity is the closest, in the sense of the hierarchical distance, to the destination office of the call — provided, of course, that the group exists.

4. The last choice in this list is always the hierarchical group of the office. If the call is blocked on this group, it is lost; this is why it is called a *final trunk* group.

The hierarchy induces a partition on the links into two classes: high-usage groups and final groups. The fact that a group is a final is independent of the particular traffic stream considered, and is a topological property of the network. This is in marked distinction to more general nonhierarchical routing methods, where a group may be a final for a given traffic stream, and a high-usage for another stream. This fact is important for the network performance evaluation and dimensioning methods discussed later in this book.

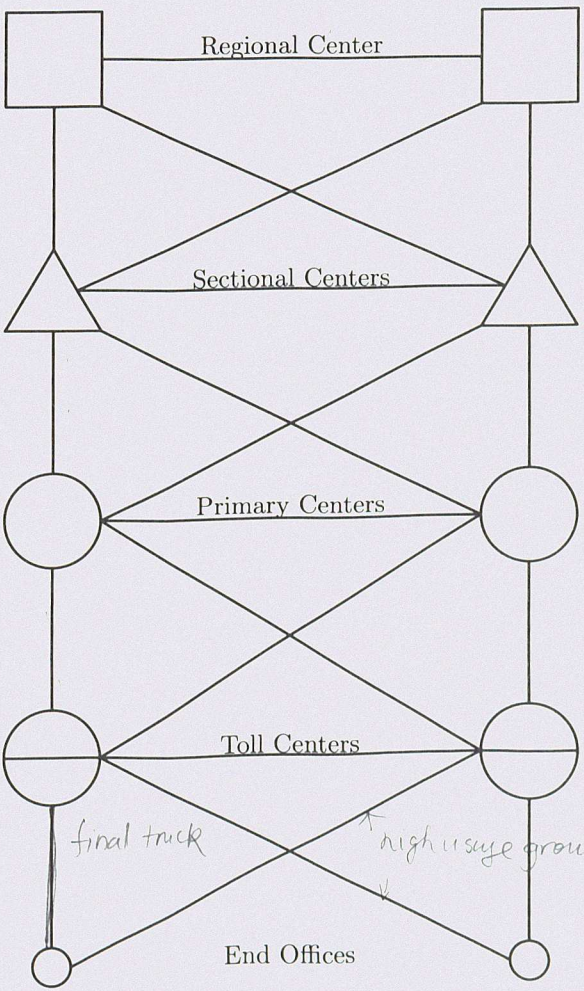

Figure 2.6 Five-Level Hierarchical Network. Some connections left out.

Among the advantages of such a routing procedure is that there is no possibility of introducing cycles in the routing of a call. Also, the procedure can be implemented on very simple hardware, with limited information and control. There can be a number of variations on the fan rule, for instance by omitting some groups from the list of an office or by modifying the loss condition 4 to allow the selection of other routes (see Section 2.5).

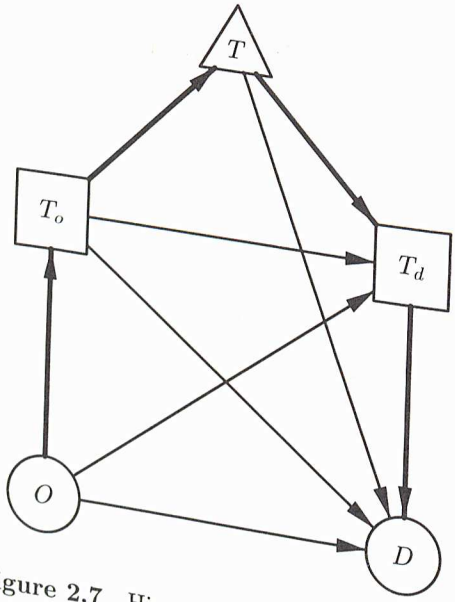

Figure 2.7 Hierarchical Alternate Routing

Dynamic Nonhierarchical Routing

Until the mid 1980s, hierarchical alternate routing, depicted in Fig. 2.7, was the universal routing method in telephone networks throughout the world. At that time, a major change was introduced in the routing of the long distance AT&T network in the United States, which represents a radical departure from the classical hierarchical method. This method is called *dynamic nonhierarchical routing* (DNHR) [4].

The most important element of DNHR is that there is no longer any hierarchical relation between offices. All switches in a DNHR network are of the same type, and can equally well receive calls from customers or serve as a tandem switch for alternate routing. As a consequence, there is no longer a hierarchical backbone, and the distinction between high-usage and finals does not hold networkwide. In other words, a group can be high-usage for a given traffic relation and the last choice for another.

The routing is still of the alternate type, but with the provision that alternate paths are limited to two links at most (see Fig. 2.8 for an example). This restriction was introduced because it is known that nonhierarchical routing methods that allow long paths for call completion can be highly sensitive to overloads, even though there may be a slight gain in efficiency at nominal loads.

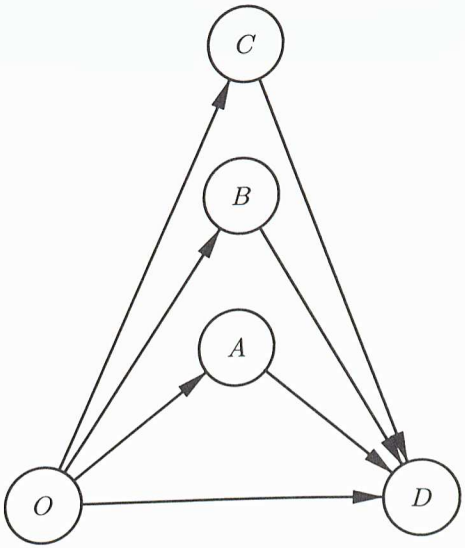

Figure 2.8 Two-Link Alternate Routing for DNHR

The routing is also dynamic, in the following sense. During the engineering phase of the network, the traffic demand is represented by a number of traffic matrices, each corresponding to a period of the day. Typically, about 10 periods are retained as significant for the design. One result of this design process, called the Unified Algorithm, is the production of a near-optimal alternate routing sequence for each of these periods, and of course for each traffic stream. After the design is complete, these routing tables are stored in the switches. As each time period arrives, the appropriate table is selected, and is used for the whole duration of the period, irrespective of the state of the network.

In addition to the nonhierarchical nature of the routing, *call crankback* is used throughout the network. In crankback mode, the control of overflow is such that a call blocked at an intermediate point on a path can still be offered to other paths in the sequence, and is not immediately lost.

DNHR is a good example of the arbitrariness in the use of terms like *dynamic* or *adaptive*. According to our definitions, DNHR is dynamic because it is changed at fixed time intervals, but is not adaptive since it uses no network feedback except the link-occupancy information. Routes are calculated off-line from the traffic forecast, and the results are used without change. Note, however, that if we look at the network operation for a period of time that is small with respect to a period, but long with respect to a call-holding time,

then the routing looks like fixed alternate routing, and we may very well say that it is static. Conversely, looking at the operation of a switch on a call-by-call basis, we notice that for a given traffic stream from i to j, the actual path selected is different from one call to the next. We might also notice that the actual path depends on the occupancy of the trunk groups, in other words on the network state. On that basis, we could equally well declare that DNHR is an adaptive algorithm. Let us repeat that none of these views is wrong. Each emphasizes different components of the routing method; in our view, any attempt to find a "right" way to describe a routing method like DNHR (or most other techniques, for that matter) can only end inconclusively.

The DNHR routes are computed off-line, based on traffic forecasts for different times of day. Because these forecasts are subject to errors, the routing calculated is generally not the best one for the traffic conditions encountered in the network. Also, failures and overloads can and do occur — another factor that, because of the nonadaptive nature of the DNHR method, may cause inefficiency. Thus the routing algorithm must be able to react to real-time conditions [5]. This capability is provided by the so-called *real-time paths*. When routes are calculated, the first k best routes are computed for ordinary routing by the DNHR method as just described. From these, say that $m < k$ are actually available for routing. The remaining $k - m$ routes are kept in reserve, to be used whenever network performance seems to degrade. This procedure must be performed with some care, however, because using the real-time paths should not cause an increased loss of the traffic these paths were originally designed to carry. This precaution takes the form of a reservation level set on the real-time paths such that these paths will not be used unless the occupancy at the time they are needed is below the level.

Because it is necessary to change the routing tables relatively often, the use of DNHR is obviously limited to switches with stored program control. Also, the extensive use of crankback can be implemented more easily with out-of-band signaling, which is normally used in digital transmission networks. For these reasons, DNHR is generally limited to all-digital networks, although it could probably be used in networks with analog SPC switches as well. Another feature of DNHR is that, following the conversion of old switches to digital technology, it can be introduced smoothly in an existing network: The DNHR network coexists with the old hierarchical switches; it is used as a switching layer above the standard hierarchy and appears as a large network of regional centers.

The introduction of DNHR profoundly influenced network management and applicable engineering methods. Network operation and management has become more centralized, providing facilities to collect the networkwide statistics needed for the design stage. Design methods have also undergone a dramatic change; they are now quite different from classical design techniques even though they retain some basic principles. Perhaps the more significant change

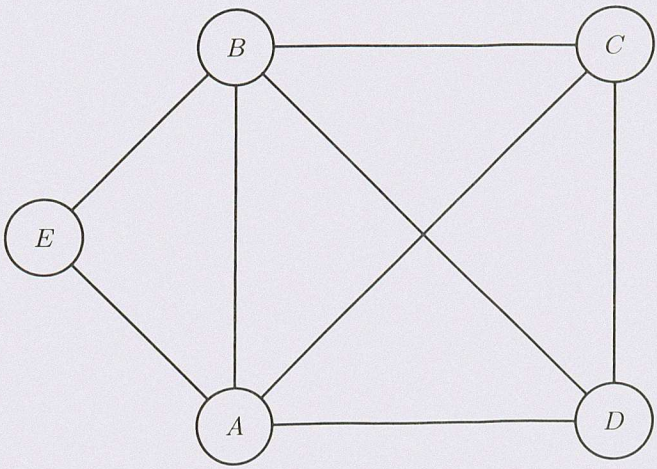

Figure 2.9 Example for Augmented Route Trees

is the need to optimize the routing, a problem that did not arise in the context of hierarchical networks. These questions are the subject of detailed discussions in Chapters 7 and 9.

2.5 Graphic Representations of Fixed Alternate Routing

Although routings can be represented directly on the network graph, their complexity grows so quickly that it is necessary to use a more detailed graphic description. This representation should be flexible enough to represent a wide class of routings, and accurate enough to define the routing uniquely. By this we mean that inspection of the representation should indicate precisely how the call can be connected or lost, as the case may be, for a given state of the network. We first describe a particularly useful representation, the so-called *augmented route tree*. Originally proposed by Lin et al. [6], it can be used for a very general class of alternate routing methods. Then we review a less general representation known as the *influence graph*, which is used mostly for hierarchical networks in the context of the notion of an ordered network.

Augmented Route Trees

To describe routing in terms of route trees [6], consider the origin-destination pair (E, D) of the network shown on Fig. 2.9, where all links are two-way.

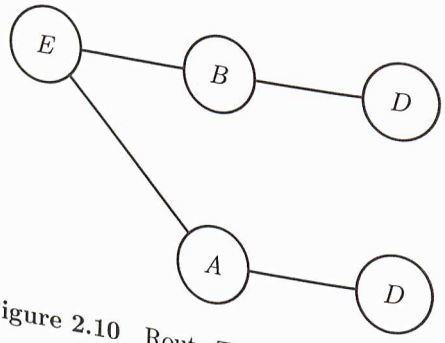

Figure 2.10 Route Tree for (E, D) Calls

We construct a graph that contains all the paths available to (E, D) calls, all starting at node E and terminating at node D. When read from top to bottom, these paths are put in the graph in the same order as the overflow. Thus suppose that there are only two paths available to (E, D) calls, that is, (E, A, D) and (E, B, D). Assume that calls that do not find the first path free overflow onto the second one. The corresponding route tree is shown in Fig. 2.10. Note that overflow can occur only on paths that originate at E. We say that overflow control is at node E. Since this is also the node at which calls originate, we speak of *originating-office control*, or OOC.

Because originating-office control has relatively few possibilities, we need to introduce more complex routings by allowing control to *spill forward* in the route tree, provided that the call can be connected to these nodes. For instance, we could have originating office control with spill at node B, as described by the route tree shown in Fig. 2.11. The interpretation is that overflow initially occurs on all paths originating at node E. If the (E, B) link is available, however, the control of overflow is transferred at node B. From then on, overflow can occur only on paths originating at node B. If no such path is available, the call is lost. In other words, once the control is transferred to B, it cannot return to E when all paths originating at B are blocked.

As noted earlier, the mode in which movement of overflow control can occur in the opposite direction is called *crankback*. The distinction between spill with crankback and without crankback lies in the conditions under which a call overflows on the (E, A, D) path. If there is no crankback, overflow to (E, A, D) happens only if the (E, B) link is blocked. In the case of crankback, overflow to (E, A, D) happens only if all the paths originating at node B are blocked. In other words, if (B, C) is blocked, call control returns to E and the next path on the list is attempted.

Finally, a frequently used method of control, *sequential-office control* (SOC), also called *progressive control* or *sequential control*, is simply OOC with spill

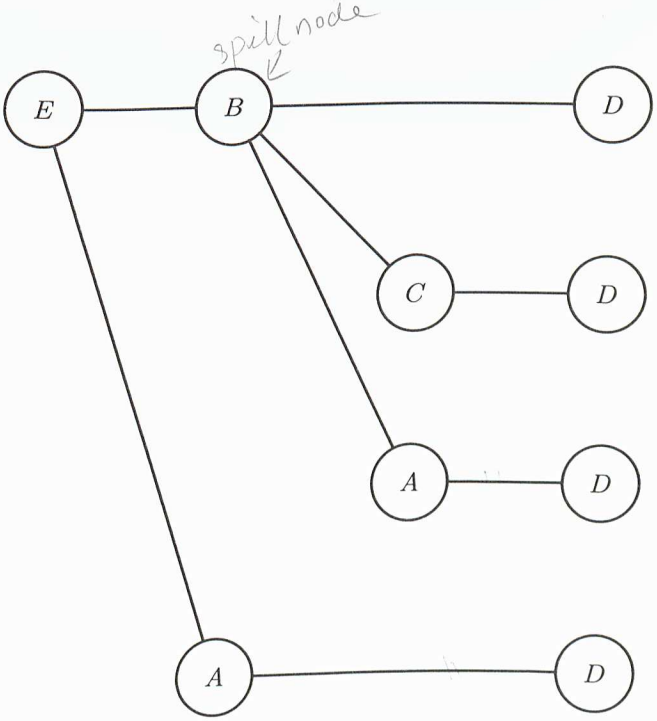

Figure 2.11 OOC with Spill at Node B

at all the nodes and no crankback. An example for (E, D) calls can be found in Fig. 2.12.

The route-tree representation is insufficient to distinguish between control with and without crankback. In some cases, it is not even possible to distinguish between OOC and SOC control even though the two methods behave quite differently. Consider, for instance, the route tree for (D, A) traffic shown in Fig. 2.13. This tree could represent equally well a routing with SOC, or OOC with spill at node C. Note, however, that the situation is quite different in the two cases. Suppose that links (D, A) and (B, A) are blocked but that links (D, B), (D, C), and (C, A) are free. Then, under SOC without crankback, a call will overflow to (B, D), then will be connected to B because the link (D, B) is available. At this point, the call is blocked because link (B, A) is not available. Because of the lack of crankback, it is impossible to return the overflow control to node D, and the call is lost. Under OOC with spill and crankback, however, the call will be connected because the possibility exists of transferring the overflow control back to node D. In practice, this means that the *originating* office D can detect that the (D, B, A) path is blocked before

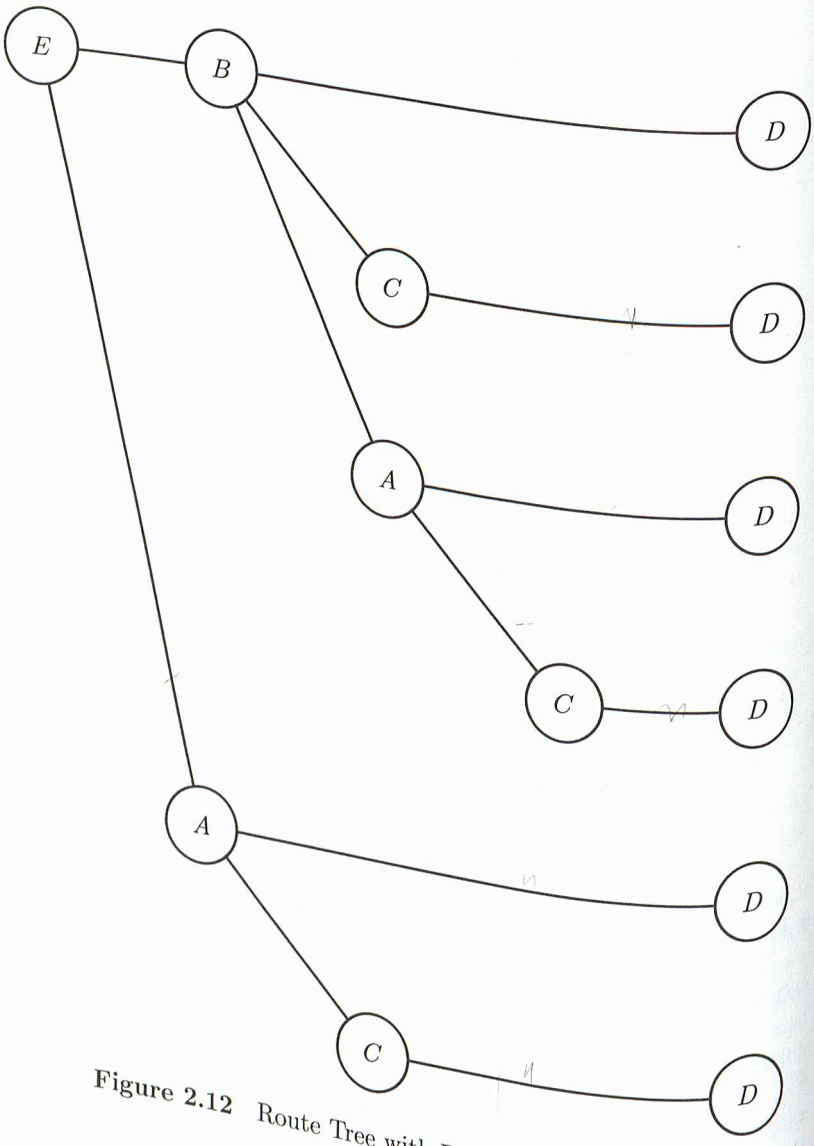

Figure 2.12 Route Tree with Progressive Control

attempting any connection on this path and instead can use the (D, C, A) path, which is free.

The graphic representation of routing should make the distinction between various forms of overflow control immediately obvious. The solution is to augment the route tree with so-called *loss nodes* — additional nodes similar to

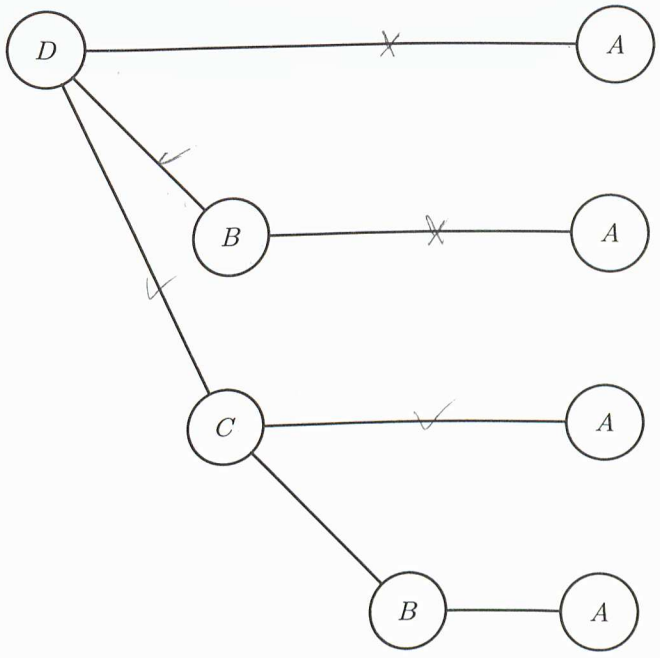

Figure 2.13 Route Tree for (D, A) Traffic

destination nodes, such that a call that reaches one of them is lost. By convention, the probability of blocking on a link leading to a loss node is zero. In our example, we obtain the *augmented tree* in Fig. 2.14 for SOC and in Fig. 2.15 for OOC with spill at C.

As we see in these two examples, although the original route tree is identical for the two types of commands, the augmented trees are different because the conditions under which blocking can occur are different. Overflow control is defined by the number and position of the loss nodes in the augmented trees.

The augmented-tree technique is a very powerful way to represent routing. Let us now use it to *define* what is meant by fixed alternate routing: A routing technique is a fixed alternate routing technique if and only if it can be represented by an augmented route tree. In fact, augmented route trees can be used in more general situations if they are made to depend either on time or on the network state. In the first case, they are used to represent dynamic routings; in the second, they are used for adaptive routings.

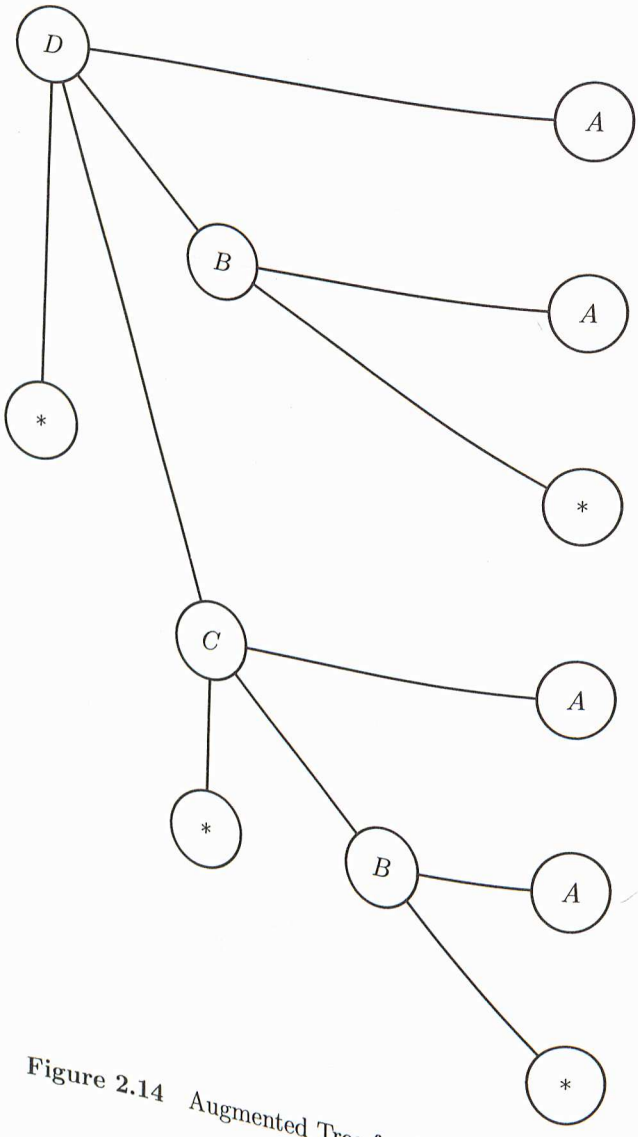

Figure 2.14 Augmented Tree for SOC Control

Influence Graph

The construction of a nonhierarchical routing must ensure that the routing does not lead to cyclic situations in which a call finding a link, say s, busy, overflows onto a link t, and finding this link also busy, overflows onto link s

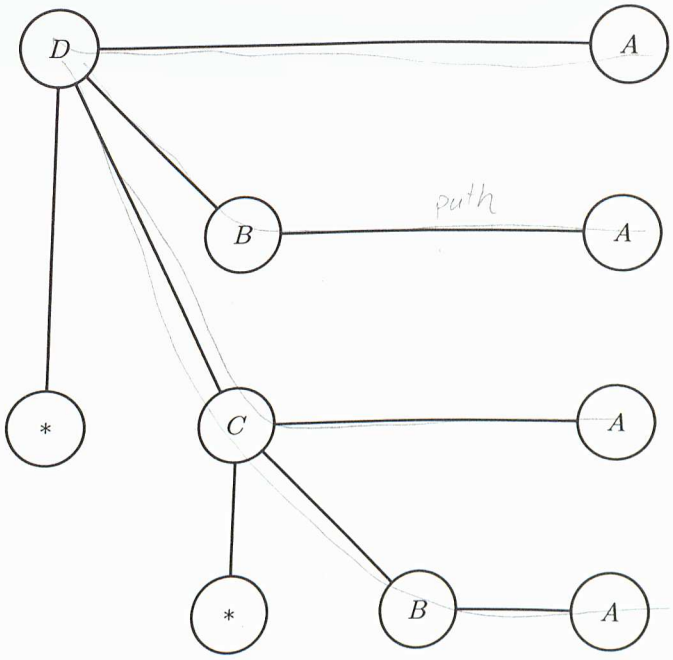

Figure 2.15 Augmented Tree for OOC with Spill

again. Two-link and hierarchical routing methods automatically ensure that this situation cannot occur. In more general cases, this overflow condition is a distinct possibility, and a systematic way to check for it must be used. Thus we introduce influence graphs, which can serve to check for cycles in routings in certain conditions. These graphs are also a basis for discussing the notion of order in a network, as well as for considering numerical algorithms for certain types of networks.

As we have seen, under SOC control routing decisions are based exclusively on the state of the links adjacent to the node that the call has reached. In this sense, SOC is shortsighted since no account is taken of the rest of the alternate paths to the destination. Thus, for general SOC route trees, there exists the possibility that the routing will eventually enter a cycle — the *ring-around-the-rosy* problem. Consider once more the network of Fig. 2.9 and its associated augmented route tree, Fig. 2.14. Suppose now that the routing is modified to look like the tree shown in Fig. 2.16. The presence of the (D, A) link as the last choice in the tree means that there is no unique action to be taken if a call is blocked on link (B, A). If the call is new, it should overflow onto

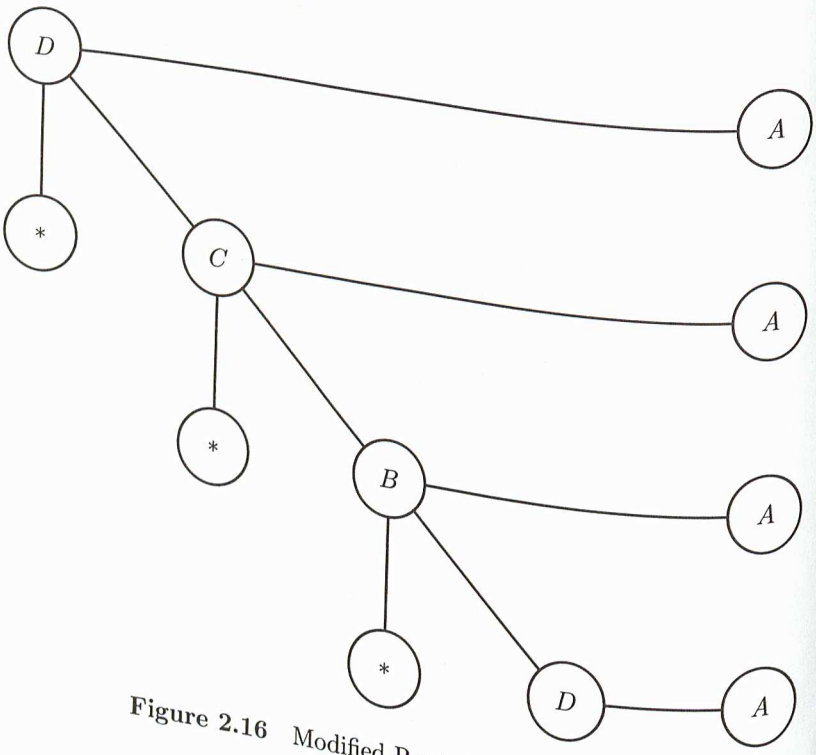

Figure 2.16 Modified Route Tree with Cycle

(B, D); it should be lost if it was already carried on (C, B). In practice, the routing algorithm will enter a cycle unless some signaling information is sent along with the call to indicate that it was already carried on link (C, B) and that it should be lost if blocked. Although most modern networks allow such transfer of information, this is not the case in older networks. Also, this kind of transfer may not be desirable in newer networks because it might increase unduly the amount of signaling information that must be transmitted. Thus it is desirable to have some systematic way to detect cyclic situations and to remove them when they occur.

The influence graph simplifies this task considerably. It is constructed as follows: For each link in the route tree, define a node of the influence graph. If the links in the real network are one-way, then there will be two corresponding nodes in the influence graph, one for (i, j) and one for (j, i). At a given node of the influence graph, define two arcs. One, the overflow arc, leads to the node representing the link to which a blocked call will overflow in the network. The other, the carry arc, represents the link on which a call that is carried will be offered. Note that the graph does not specify on which link a call will go; this

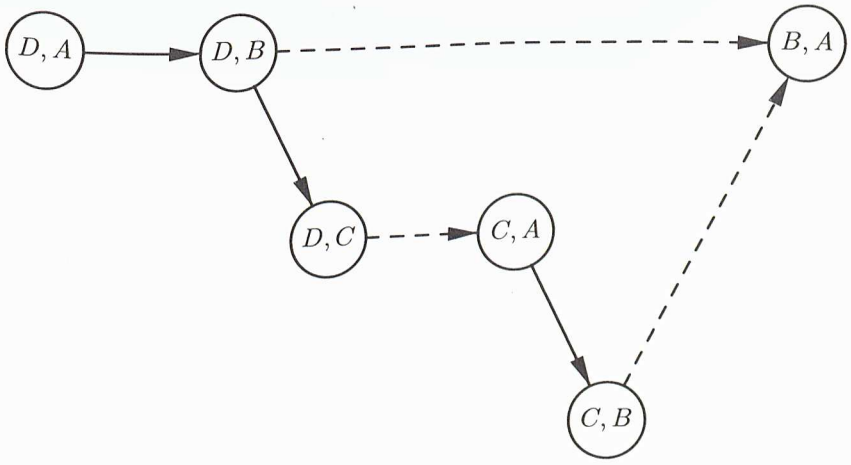

Figure 2.17 Influence Graph for SOC Route Tree without Cycle

depends on the blocking probabilities of the links, which are traffic dependent. The only purpose of the influence graph is to indicate the *possible* ways a call can go when it is offered to a link. For notational simplicity, we define two functions:

$\sigma(i, j)$ = For trunk group i, and calls destined for j, σ indicates the number of the trunk group to which blocked calls will overflow.

$\rho(i, j)$ = For trunk group i, and calls destined for j, ρ indicates the number of the trunk group to which calls that are carried on i will be offered.

The influence graph is a graphic representation of the ρ and σ arrays. With this definition, we can construct the influence graphs of the route trees corresponding to Figs. 2.17 and 2.18, shown on Figs 2.17 and 2.18, respectively, where overflow is indicated by a solid arrow and carried traffic by a dashed arrow.

The difference between the two graphs is immediately obvious. Corresponding to the routing with cycling is an influence graph containing a circuit. In fact, the presence or absence of circuits in the influence graph is equivalent to the presence or absence of cycling in the routing. Because there are standard methods of graph theory to detect the presence of cycles in directed graphs, it is possible to construct algorithms that check automatically for the presence of cycles in route trees. Furthermore, routings without cycling are characterized

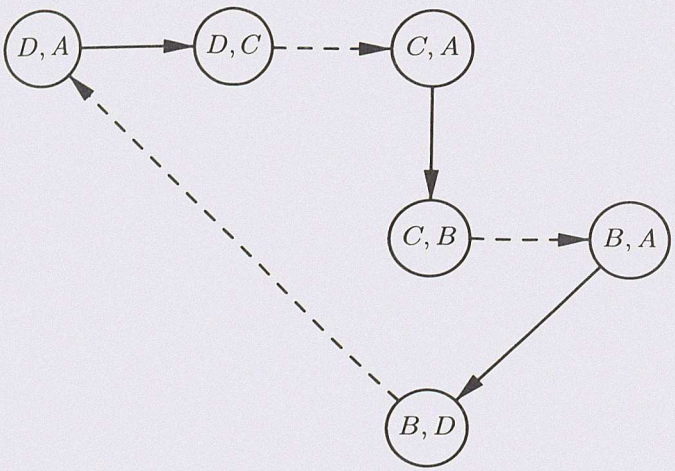

Figure 2.18 Influence Graph for Route Tree with Cycle

by a partial ordering of the links of the route tree since nodes of a directed graph without a circuit can be given such a partial ordering.

Mutual Overflow and Ordered Networks

The influence graph provides an easy way to detect circuits in a given route tree. In addition to such circuits, another type of circular situation can happen that is not as obviously undesirable as the ring-around-the-rosy problem. In this situation, called *mutual overflow*, one link i receives calls from another link j, which in turn receives calls from link i, generally from another traffic stream.

Consider for instance, the route tree for (C, A) calls shown in Fig. 2.19 and its influence graph, shown in Fig. 2.20. Viewed separately, neither this influence graph nor the one in Fig. 2.17 for the (D, A) traffic contains any circuits. If the two graphs are merged, however, we obtain an influence graph that *does* contain circuits. This means that, for any two nodes in this circuit, the corresponding links both receive calls from and send calls to the other node. This situation is not necessarily to be avoided — it may lead to more economical networks. Certainly, however, it makes the computation of the network performance more difficult (see Chapter 4).

In one special, particularly important class of routings, the combined influence graph *for all streams* contains no circuit. In this case, we can define a partial order μ on the nodes of the influence graph and consequently on the

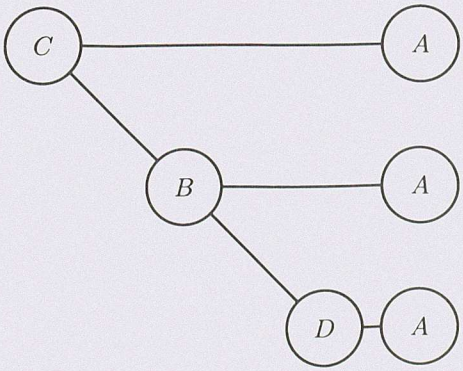

Figure 2.19 Route Tree for (C, A) Calls

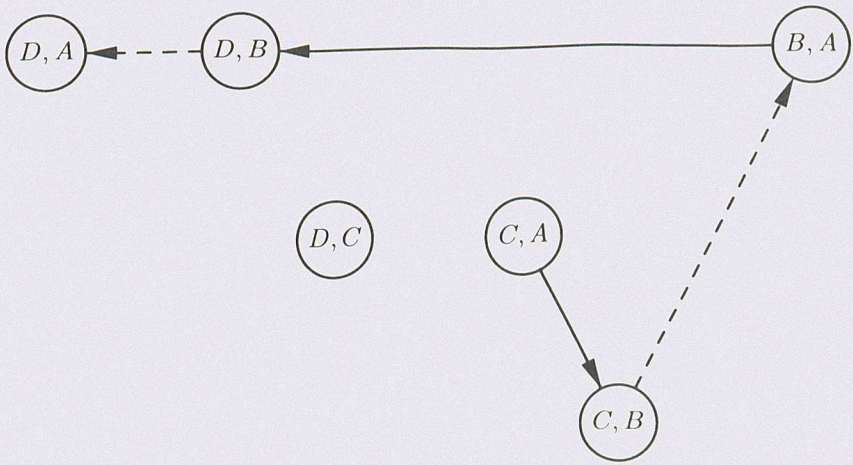

Figure 2.20 Influence Graph for (C, A) Calls

network links. Let $\mu(i)$ be the order number of node i. We say that node i is a descendant of node j if $\mu(i) > \mu(j)$. Define also

$$\mathcal{D}(j) \stackrel{\triangle}{=} \{i \mid \mu(i) > \mu(j)\}$$
$$\mathcal{A}(j) \stackrel{\triangle}{=} \{i \mid \mu(i) < \mu(j)\},$$

which are called, respectively, the descendants and the ancestors of j. For

ordered networks, we know that the calls offered to node i depend exclusively on the nodes $j \mid j \in \mathcal{A}(j)$. Ordered networks draw their importance from the fact that fixed hierarchical routing networks can be considered to be ordered under assumptions discussed in Chapter 4. As discussed there, this is an important property that simplifies the calculation of traffic flows considerably.

Combined Load Sharing and Alternate Routing

Although we have presented load-sharing and alternate-routing policies as distinct methods, they can be combined in a single, more general algorithm. To do this, we must introduce the notion of a *route*, that is, the set of alternate paths available to a given type of call. In this case, the sharing coefficients have the same meaning as in the simple case, with the understanding that the sharing is between routes, instead of simple paths. The actual operation and analysis of the routing are more complex, however, because of the presence of alternate routing. A general graph representation is a mixture of load-sharing graphs and augmented route trees (see Fig. 2.21).

2.6 Adaptive Routing

Alternate routing was designed to take into account the constraints imposed by the electromechanical switches, as well as the limited amount of information available about the state of the network. The widespread introduction of stored-program switches, digital transmission, and signaling have removed these constraints by increasing enormously the amount of information available for control. These advances in turn have made possible the introduction of more sophisticated routing methods, known under the generic name of *adaptive routing*.

While alternate routing operates on ordered sequences of paths, adaptive techniques operate from a fixed *set* of potential connection paths for each stream, with no predetermined order. Instead, a value is given to the paths in the set, and the path with the highest value at the time the call arrives is selected. Variations on this theme are possible, depending on the definition of the path value, the value-update mechanism, the use of probabilities for path selection, and potentially a number of other factors. Some variations are being implemented in public telephone networks, while others have only been proposed.

Residual-Capacity Routing

Of all the routing methods currently in use or proposed for use, *residual-capacity adaptive routing* (RCAR) has the largest information set with which to

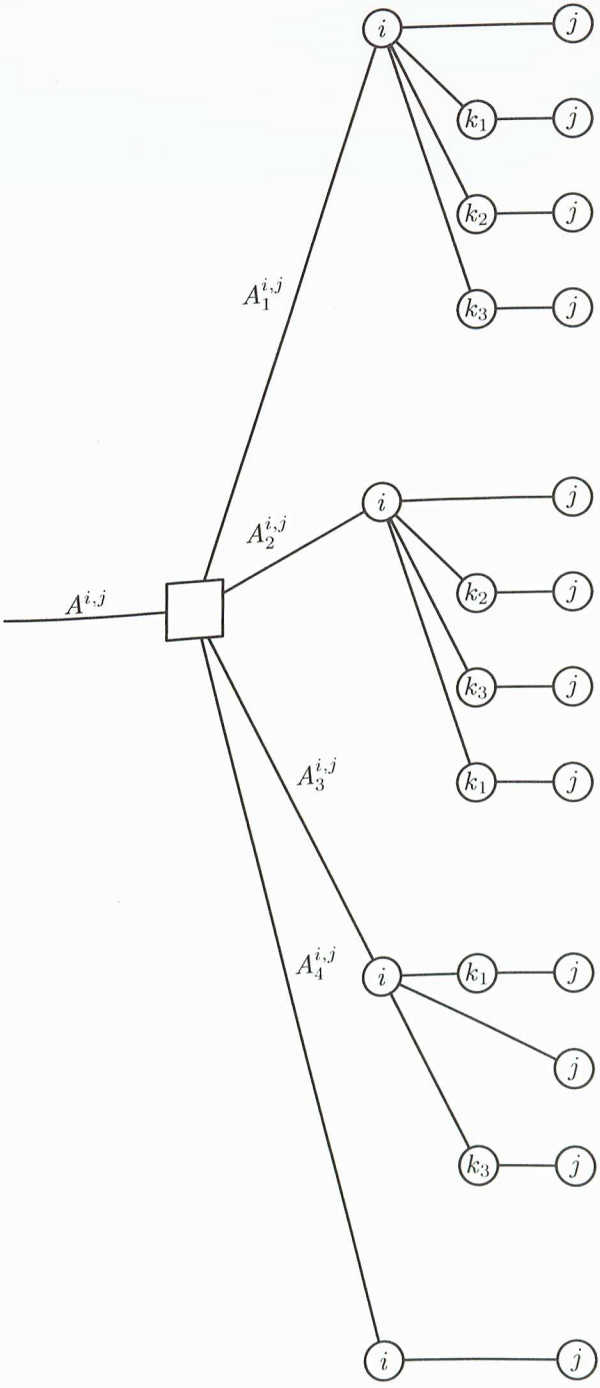

Figure 2.21 Route Formulation for Nonhierarchical Routing

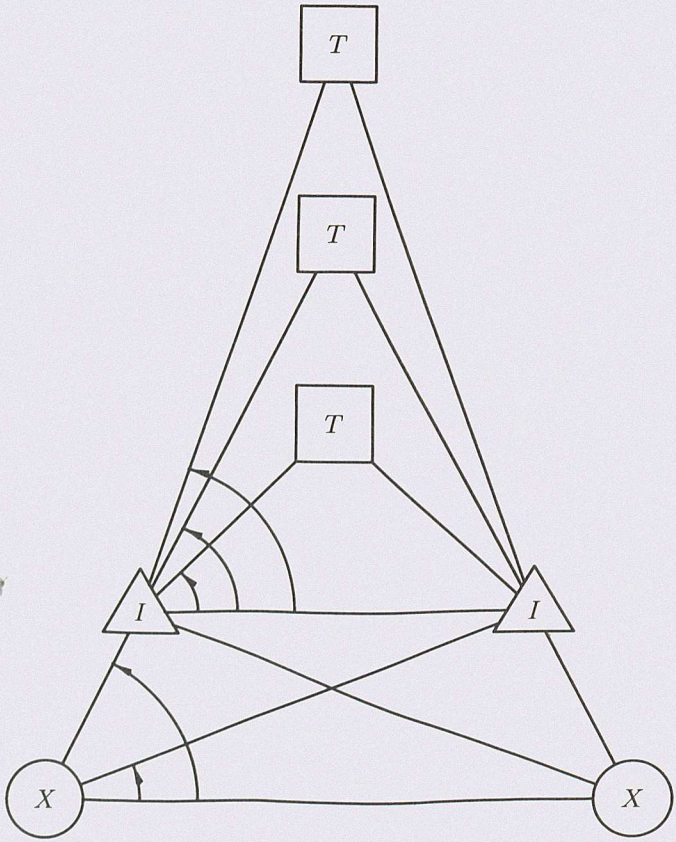

Figure 2.22 Two-level Network for Adaptive Routing

make its routing decisions: It uses occupancy information on *all* network trunk groups, information periodically updated via actual measurements made by the switches. From this point of view, RCAR is probably the most advanced routing technique now in existence, and for this reason deserves a thorough discussion.

Because of the need to alter the routing tables dynamically, residual-capacity adaptive routing can be implemented only in networks with stored-program control switches and advanced signaling capabilities. This method can, however, be made part of a two-level network, where a lower-level network of non-SPC switches, called X-nodes, overflows onto a higher-level network of SPC switches, called I- or T-nodes, where calls are routed according to the

adaptive policy (see Fig. 2.22). T-nodes have a purely tandem function, while I-nodes have some originating traffic. Otherwise, both sets of nodes operate in exactly the same way and can act as tandem for any traffic flow. Needless to say, nodes without alternate routing capability, such as step-by-step switches, cannot become part of an RCAR network.

Because of the adaptive nature of the routing, some form of networkwide information collection is required, possibly a CCITT No. 7 signaling channel or a dedicated measurement network. Rather that discussing the implementation issues related to this system, we assume that it is given and that it can furnish the algorithm with whatever information is needed.

The first proposal for implementing an adaptive routing technique of the residual-capacity type in a public telephone network originated with [7,8]. Originally called *high-performance routing* (HPR) and now known as *dynamic call routing* (DCR), this method operates by sending calls on paths with the largest expected number of free trunks. DCR, the implementation of RCAR discussed in [9], operates with a single overflow path computed periodically for each origin-destination pair. Calls blocked on the direct link are offered to this path, where they are lost if blocked again. Path are selected at random, with selection probabilities proportional to the estimated residual capacity of all potential paths.

DCR Algorithm Definition. Define the following:

N_s = Number of circuits on link s.

$R_s(t)$ = Expected number of free trunks on trunk group s at time t.

$n_s(t)$ = Number of busy trunks on trunk group s at time t.

$A_s(t)$ = Estimated call-arrival rate to trunk group s at time t.

μ_s = Average call-holding time on trunk group s. In practice, this is independent of s.

$m_s(t)$ = State protection at time t on trunk group s. This is the level of occupancy at which overflow calls will be blocked.

$M_s(\tau)$ = Number of calls actually offered to trunk group s during a time interval τ.

$\alpha_k^{i,j}$ = Probability of selecting tandem k for a call between i and j.

Δ = Update interval. At every Δ seconds, the network control center queries the switches about the occupancy of their outgoing links. The information is processed, and new routing tables are sent to the switches that will be used for the next Δ seconds.

State protection is a control technique that protects fresh traffic against blocking by calls that overflow from other routes. It is implemented by setting a threshold, perhaps a distinct one for each link, such that when the number of busy circuits on the link exceeds this value, calls that are alternate routed will not be connected on the link. If the threshold is set at a value lower than the group capacity, this amounts to reserving the difference $N - m$ for first-offered traffic, which is known to be a good policy in case of traffic overload.

The estimated residual capacity on group s at time $t + \tau$ is given by a simple linear extrapolation:

$$R_s(t+\tau) = N_s - n_s(t) - \tau\left[A_s(t) - n_s(t)/\mu_s\right] - m_s(t). \tag{2.11}$$

At time t, the estimated call-arrival rate is updated by

$$A_s(t) = \theta A_s(t-\Delta) + (1-\theta)\frac{M_s(\Delta)}{\Delta}.$$

For intelligent nodes, the selection of a tandem is made at random, with probabilities

$$\alpha_k^{i,j} = \frac{\overline{\alpha}_k^{i,j}}{\sum_k \overline{\alpha}_k^{i,j}}, \tag{2.12}$$

where

$$\overline{\alpha}_k^{i,j} = \max\left[0, \min\left\{R_{i,k}, R_{k,j}\right\}\right]. \tag{2.13}$$

For X-nodes, the tandems are selected on the basis of the residual capacities of the outgoing links only. A number of parameters can influence the performance of the algorithm:

τ The extrapolation interval.

Δ The update interval.

θ The weight of the updated arrival rate.

m_s The protection level. This can be made fixed or variable. If it is fixed, one must select its value; if variable, an algorithm to compute it is needed.

The proper selection can be made only through simulation and possibly experience, since we do not have analytic models to compute the performance as a function of these parameters. The reservation level can be made adaptive [10] by the formula

$$m_s(t) = g \times a'_s, \tag{2.14}$$

where

a'_s = Current amount of *first offered* traffic overflowing link s.

g = A scale factor, usually chosen equal to one.

Field Trial. DCR has actually been implemented in a real telephone network under normal operating conditions. In 1979, a field trial was carried out in which a nine-node subnetwork of the Toronto metropolitan network was temporarily converted to DCR operation for part of the day. The standard hierarchical routing was kept as a backup, and a dedicated measurement network was implemented (at that time the Toronto network did not have a distinct control network). The actual parameter values selected for the trial were $\tau = \Delta =$ 16 secs, $\theta = 0.9$, and $m_s = 2$. The results of the experiment are fully described in [9]. The main conclusions derived from the exercise were as follows:

1. DCR is a feasible routing technique, at least for small networks.

2. DCR offers substantial improvements in terms of traffic-carrying capacity over the standard hierarchical routing.

3. Δ is the most important parameter. Its value should be chosen as small as possible, consistent with technological constraints. The value of the other parameters is not very important and can lie in relatively large intervals without affecting the performance of the algorithm significantly.

4. In addition to these results on the routing algorithm itself, the field trial provided the opportunity to make numerous traffic measurements and to validate the network simulation used to study the algorithm.

As a result of the field test, it was decided to implement DCR in the long distance Telecom Canada network.

Trunk Status Map Routing

The original proposal for dynamic nonhierarchical routing was of the fixed nonhierarchical type (see Section 2.4). The only provision for adaptation to the network conditions was through the actual status of the alternate paths (which determined the path that was going to be used), as well as through the use of the real-time paths. The order of choices and the set of paths in a route were not subject to modification, except at fixed time periods determined by the original network routing and dimensioning. Routes for each period are selected on the basis of forecasted loads, which can take into account some systematic variations in demand but are nevertheless inherently subject to errors. Also, because the routing tables and network dimensioning are recalculated infrequently (typically, once a week for routing and twice a year for dimensioning), these computations cannot take into account short-term variations in load, say of the order of minutes. For this reason, some form of adaptation to these variations seemed desirable, and a second version of DNHR, called *trunk status map routing* (TSMR), has been proposed, incorporating some of the features of the RCAR method [11].

The algorithm operates on the set of paths as calculated for the original DNHR, but the actual selection of a path for a call to be routed is not strictly sequential. It can depend on the occupancy of the paths. The following rules were envisaged:

1. Route the current call on the least-loaded path from the set of available paths.

2. Route the call first on the direct path, if it exists, and then on the least-loaded path.

3. Route the call on the first path computed for nonadaptive DNHR. If this routing is not possible, select the least-loaded path.

4. Recompute a new set of alternate paths such that the total carried traffic will be maximized, at least for a short duration, starting from the current state of the network.

Obviously, Rule 4 is not practical, and was considered only as a lower bound on the efficiency of the other rules. As the result of simulation studies, it was found that Rule 3 generally performed better than Rules 1 and 2, and was only marginally inferior to Rule 4 — that is, it was nearly optimal. Rules 1 and 2 did not perform well because the least-loaded path or the direct path, as the case may be, are not necessarily the best choices as computed by the dimensioning and routing optimization algorithm. In this case, Rules 1 and 2 have a tendency to allocate traffic to expensive paths or to increase the amount of two-link routing in such a way that the crankback traffic becomes too large, reducing the efficiency of the routing.

The actual implementation of TSMR uses a centralized trunk status map (TSM), which coordinates the routing changes between all switches. Every T seconds, each switch sends to the TSM an update on the number of free trunks on each trunk group, provided this number has changed. The TSM then recomputes a new set of routes for each switch, which are sent back and used during the next T seconds to perform the alternate routing of calls. The route sequence stored in the switch consists of two parts: (1) the first path computed by the routing optimization based on forecasts and (2) the remaining paths, which are the recommendations sent by the TSM. In practice, it is not necessary to recompute all the paths in the second list, but changing the second choice according to the least-loaded rule is sufficient to achieve an adequate performance.

A threshold is used to determine whether an update is needed for each stream where the direct link exists. The use of a threshold is based on the reasonable assumption that, if the load on the direct and first alternate paths is sufficiently light, then all the calls that will arrive in the next T seconds

can be carried on these two paths. In this case, there is no reason to change the second choice, and no update is sent. A practical value of $N/8$ has been suggested for this threshold, where N is the number of circuits on the direct link.

Whenever an update is needed, the TSM first retrieves the list of candidates for least-loaded path from a centralized database. The least-loaded path is then selected; if a routing change is implied, the update is sent to the switches. This must be done carefully because large groups are likely to have many free circuits for short periods of time. A straightforward selection rule based exclusively on the current residual capacity could mean that these groups would be selected very often, but thus would immediately fill up and become unavailable for some time. For this reason, the number of idle trunks on large trunk groups is discounted by some factor, with the suggested value of $N/36$, where N is the number of circuits in the group.

After it is determined that a routing change is in order, the new path sequence is sent to the switch. The first two paths in the second class, that is, those affected by the TSM, are replaced by the new values, and the third choice in this class is left unchanged.

The effects of this routing technique were simulated, with the finding that the number of crankback information is relatively small, and that the additional load on the switch processor is not significant. The value of the update interval T was estimated from the trade-off between a higher processing load caused by a small value of T and the increase in crankback traffic, and hence of processing load, caused by a large T. It turns out that a value between 2 and 10 seconds is nearly optimal, assuming that the crankback load is not a strong function of the update interval.

Other Pricing Functions

Both DCR and TSMR operate with path values that are essentially defined by the residual capacity of the path. Using other cost functions, adaptive routing can be formulated more generally. This can be done either in the context of the Markov decision process model [12] or in the selection of load-sharing coefficients [13].

The general algorithm in both cases is based on the following argument. Suppose that each call belongs to a type j that in general is related to the origin-destination pair of the call and perhaps to other parameters. Each call type has a revenue r_j that is obtained if the call is connected on some path. The decision to connect on a particular path j will then produce r_j. The decision will also have a detrimental effect since, by using one trunk on all the links in the path, it can cause calls of other classes to be lost, with a corresponding loss of revenue. This lost revenue, the sum of the individual link losses q_s on the path, is written $\sum_{s \in l} q_s$. The net gain g_l^j for routing a call of class j on path k

is simply the difference $g_l^j = r_j - \sum_{s \in l} q_s$. The call is then routed on the path with the largest net gain if it is positive; the call is lost if there is no positive gain.

This rule has the immediate advantage that flow control is already embedded — all the gains may turn out to be negative even though one or more free paths may be available for the call. The rule makes it possible to modulate the traffic in the network to effect network-management functions systematically, based on call classes — another advantage over current methods, many of which are based on manual or semiautomated intervention.

There are two major problems: (1) calculating the net gain of a decision, whether in the Markov or load-sharing context, and (2) evaluating the network performance (as with any other routing technique). Most results obtained to date apply to the first question; they are discussed in Chapter 4. The value of revenue-based adaptive routing in terms of network performance is currently the subject of investigation.

2.7 Learning Automata

As can be seen from the extensive bibliography of [14], *learning automata* have found many uses in control engineering. They are particularly useful when little is known about the response of the environment to actions of the system. In the automaton approach, knowledge about the environment is gained only through the responses received by the automaton after each of its actions. In fact, the only knowledge gained is that "this action is better than that one because a good response is received more frequently in the first case than in the second." That is, the action-reaction channel is the only way of learning about the world. This approach has some obvious advantages, the most important of which is simplicity. It suffers, however, from the corresponding inconvenience that learning may be quite slow since the information available is so severely limited.

It should be clear by now that routing is a rather complex process. Furthermore, networks are seldom stationary, because of either unexpected variations in demand or component failures (fortunately much less frequent) of the network itself. Add the real-time requirements of routing, which limits the amount of computation that can be done before a decision is taken, and all the requirements for a learning scheme for routing are present.

First we give a short outline of formal automata theory. Then we relate the general model to routing, describing the special cases of automata that have been studied in detail. Finally, a short discussion of the pros and cons of the automaton approach is given.

Formally, an automaton is defined by the quintuple $\{X, \Phi, \alpha, T, G\}$, where

$X =$ The sets of inputs to the automaton.

$\Phi =$ The set of internal states.

$\alpha =$ The set of actions $\{\alpha_1, \alpha_2, \ldots \alpha_r\}$.

$T =$ The state-transition map at stage n:

$$\Phi(n+1) = T\left[\Phi(n), \alpha(n), X(n)\right].$$

$G =$ The action-selection function:

$$G : \Phi \rightarrow \alpha.$$

The automaton is deterministic or stochastic depending whether T and G are deterministic or stochastic. The class of automata considered for routing, called *variable structure automata*, are of the form

$$\Phi = p_1, p_2, \ldots p_r$$
$$\sum p_j = 1, \quad p_i \geq 0$$
$$G(\Phi) = \alpha_i \text{ with probability } p_i$$

The states are defined by a probability vector, and the action map is the selection of an action i with probability p_i.

The environment in which the automaton operates is characterized by a set of inputs $\alpha(n)$ and random outputs $x_i(n) \in X$. Of special interest is the case where the environment output is binary, that is, a reward ($x = 0$) or a penalty ($x = 1$). In this case, the environment is characterized by a set of penalty probabilities $c_1, c_2, \ldots c_r$, where

$$c_i = Pr\left\{x(n) = 1 \mid \alpha(n) = \alpha_i\right\}.$$

The choice of symbols for the environment inputs and outputs is not coincidental. In the usual situation, the automaton outputs are fed into the environment, which reacts to these outputs by providing a reward or penalty. This response is then fed back as input to the automaton, which provides another action, and so on (see Fig. 2.23). A learning behavior is made possible by this feedback organization: The automaton is provided with some performance function that, it is hoped, will eventually be minimized. The behavior of the automaton is determined by the choice of the maps T and G, which in turn depend on the particular performance criterion that one is trying to minimize. Finally, note that the actual distributions for the response of the environment are *unknown*, and that the operation of the automaton in no way assumes any knowledge about these responses. In fact, one objective of the method is to determine the responses, and hence to choose the appropriate actions in view of the performance measure and what has been learned so far about the environment responses.

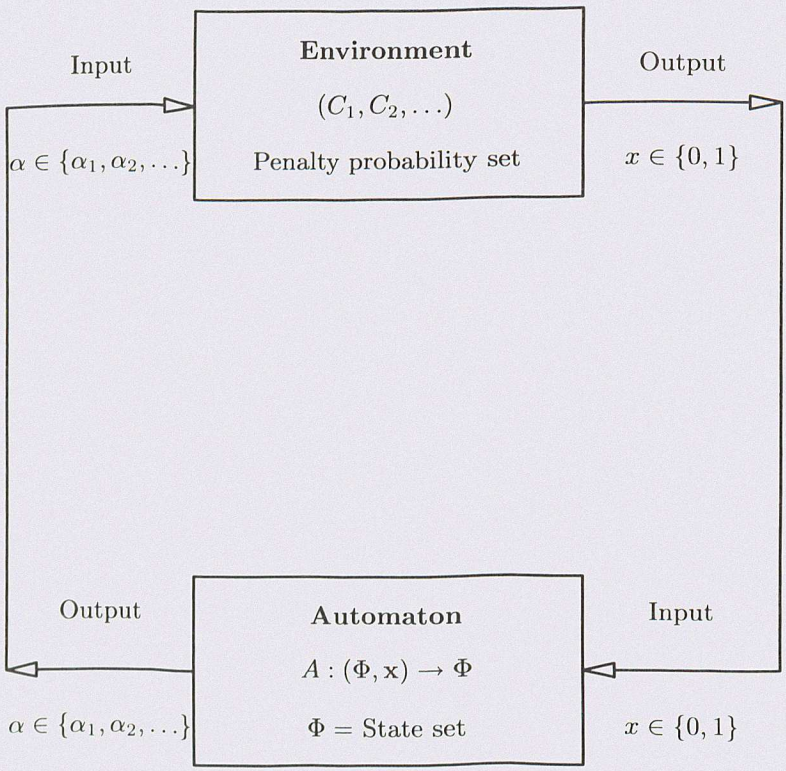

Figure 2.23 Input-Output Relationship of Learning Automaton

This general automaton formalism can be applied to the routing of calls in circuit-switched networks in the following way. Assume that in each node k there is an automaton, denoted $\mathcal{A}_k^{i,j}$, for each traffic stream i, j going through the node. If the origin of the call is not available, the automaton is denoted $\mathcal{A}_k^j$. Two sets of actions are possible for the automaton: (1) the set of all links out of the node and (2) the set of all *sequences* of links. The environment is the network beyond node k, and the responses are of the reward-penalty type, corresponding to a connected or a lost call, respectively. The objective, of course, is to minimize the number of lost calls. Three probability-update algorithms have been proposed.

The M-automaton [15] (M for mean) uses the choice of outgoing links as the action set. Define $g_i(n)$ as the estimated probability of loss if action i is used at stage n, where action i is to route the call via group i. These state

variables are updated by

$$g_i(n+1) = g_i(n) + w[x(n+1) - g_i(n)],$$

where w is a constant and $x(n)$ is the network response at stage n and is given by

$$x(n) = \begin{cases} 0 & \text{if the call is completed} \\ 1 & \text{if the call is blocked} \end{cases}$$

The action at each stage is to compute the current value of g_i, rank the groups in decreasing order of g, and attempt to route the call in this order.

The linear reward-inaction (L_{R-I}) method [16] is more closely related to the general formalism just described. This method takes its name from the fact that the probability-update function is linear, and that an update is performed only when a reward (i.e., a call connection) is received. No change is made in the case of a penalty (a blocked call). Here, the actions are the sequences of groups available at the node, and the states are given by the probability p_i of choosing sequence i. The update formula, when action i has been chosen and the call has been completed, is

$$p_i(n+1) = p_i(n) + \sum_{j \neq i} (1-\beta) p_j(n)$$

$$p_j(n+1) = \beta p_j(n)$$

$$0 \leq \beta \leq 1$$

No update is made when the call is blocked.

A natural generalization of the L_{R-I} scheme is to do something when either a reward or a penalty is incurred. In this case, we have a L_{R-P} scheme [17], where again the actions are sequences of groups to be used, and the update formula after action i has been chosen, is given by

$$p_j(n+1) = p_j(n) + \begin{cases} -a p_j(n) & \text{if } x = 0 \text{ (connected)} \\ \dfrac{b}{r-1} - b p_j(n) & \text{if } x = 1 \text{ (blocked)} \end{cases} \qquad j \neq i$$

$$p_i(n+1) = p_i(n) + \begin{cases} a[1 - p_i(n)] & \text{if } x = 0 \\ -b p_i(n) & \text{if } x = 1 \end{cases}$$

where r denotes the number of choices available to the automaton. The true L_{R-P} scheme is given for the values $a = b$. A variant is the $L_{R-\epsilon P}$ scheme, where $b \ll a$.

The use of learning automata for routing has many advantages. Perhaps the most obvious is simplicity, both computationally and in terms of measurements. Learning automata assume nothing about the network or the traffic flows, and in particular do not assume that the arrival process of new calls is stationary. Thus they should be particularly well-suited for networks in which

there are large variations in traffic, or in which network reliability in not high. On the negative side are such questions as the stability of the algorithm, the rate of convergence, and the global optimality, in addition to the usual question about performance evaluation in a network. Because of the difficulties associated with these questions, the state of our knowledge about learning automata is not as advanced as for the other methods, and research continues.

Dynamic Alternate Routing

Although the full automaton approach has not yet been implemented — or even proposed for implementation — in a real network, a current proposal for implementing a routing algorithm in the United Kingdom trunk network has some of the features of an automaton routing. Called *dynamic alternate routing* (DAR), it is based on a very simple form of learning scheme. The main elements of the methods are the following:

1 Alternate routing is limited to two-link paths.

2 State protection is implemented.

3 A new call is offered first to the direct link.

4 If the call is blocked, it is offered to a single alternate route through tandem k, whose value is currently stored in the switch.

5 If the call is blocked on the alternate path, a new value is selected for tandem k. Currently, this selection is made randomly. If the call is connected, the value of k is left unchanged.

The DAR method is loosely related to learning schemes since no state measurement is ever made, and feedback is only via the success or failure of the connection on the current alternate path. The exact relation with learning automata is left as an exercise. This scheme can be extended in various ways, some suggested in [18]:

1 Use a cyclic order for selecting tandem nodes.

2 Use a nonuniform distribution for selecting the tandem node.

3 Introduce a second parameter such that if the traffic exceeds this value, and a call must be alternate routed, then a new tandem is selected. In other words, tandem reselection occurs before a call is blocked on the path, which would presumably give a better performance.

4 Use paths longer than two links, or use dummy first-choice paths in case the network is not fully connected.

Problems

2.1. The state space for the Markov model of circuit-switched networks is quite large. It is suggested that all this information is not really needed, and that one could have a well-defined Markov model by using a state space where only the total number of busy circuits on each link is used. Explain clearly why such a description is not sufficient. (Try to explain how you could compute the transition matrix in this case.)

2.2. Network states for the Markov model have been defined in terms of the number of calls on all paths in the network. We want to define a state description that involves only the number of calls of each type on all the links. Let f_s^k be the number of calls of commodity k on link s.

1. Write down the three groups of constraints that the f must satisfy.
2. Write the matrix of state transitions. Check carefully that these can be computed from the information available from the state description only.
3. Show that, given the state in one representation, it is always possible to find the state in the other representation.

You may assume that the network arc-path incidence matrix is known.

2.3. For the three-node network 1 of Fig. 2.3,

1. Construct the state-transition diagram for the routing policy where a connection is attempted on the direct link only, and the call is lost if the link is busy.
2. Do the same thing for the policy where the direct link is attempted first, and then the alternate path if the direct link is blocked.
3. Do the same thing for the policy where the alternate path is attempted first, and then the direct link.

2.4. Calculate the number of states for the three-node network with one-way links of one circuit each.

2.5. Give a continuous-time Markov model of the network such that the corresponding multicommodity flow representation is a true one, with equality constraints instead of Eq. (2.10).