

queue and those arriving between $T(k)$ and $T(k) + \alpha(k)$ as being in service. What is peculiar about this queue is that the number of packets in it is unknown, although the nodes all keep track of the allocation interval over which service (*i.e.*, transmission) is taking place.

The FCFS splitting algorithm is the set of rules by which the nodes calculate $T(k)$ and $\alpha(k)$ for each successive k on the basis of the feedback from the previous slot. These rules are simply the improved rules for the tree algorithm discussed previously, specialized to the case where sets of nodes are split in terms of packet arrival times.

Figure 4.11 illustrates these rules. When a collision occurs, as in slot k , the allocation interval is split into two equal subintervals and the leftmost (*i.e.*, longest waiting) subinterval L is the allocation interval in slot $k+1$. Thus, $T(k+1) = T(k)$ (*i.e.*, the left boundary is unchanged) and $\alpha(k+1) = \alpha(k)/2$. When an idle, as in slot $k+1$, follows a collision, the first improvement to the tree algorithm is employed. The previous rightmost interval R is known to contain two or more packets and is immediately split, with RL forming the allocation interval for slot $k+2$. Thus, $T(k+2) = T(k+1) + \alpha(k+1)$ and $\alpha(k+2) = \alpha(k+1)/2$. Finally, successful transmissions occur in slots $k+2$ and $k+3$, completing the CRP.

Next consider the example of Fig. 4.12. Here a collision in slot k is followed by another collision in slot $k+1$. Here the second improvement to the tree algorithm is employed. Since interval L contains two or more packets, the collision in slot k tells us nothing about interval R , and we would like to regard R as if it had never been part of an allocation interval. As shown for slot $k+2$, this is simplicity itself. The interval L is split, with LL forming the next allocation interval and LR waiting; the algorithm simply forgets R . When LL and LR are successfully transmitted in slots $k+2$ and $k+3$, the CRP is complete.

In the tree algorithm, at the end of a CRP, all the waiting packets split into some number of subsets. Here, since the splitting is being done by allocation intervals in time, it is more appropriate simply to choose a new allocation interval of some given size, say α_0 , to be discussed later. Note that this new interval, in slot $k+4$, includes the old interval R that previously lost its identity as a separate interval.

In terms of the tree of waiting subsets, the effect of having a right interval lose its identity whenever the corresponding left interval is split is to prune the tree so that it never has more than two leaves (correspondingly, the stack never remembers more than the top two elements). Whenever the allocation interval corresponds to the left subset of a split, there is a corresponding right subset that might have to be transmitted later. Conversely, when the allocation interval corresponds to a right subset, there are no more waiting intervals. Thus, the nodes in this algorithm need only remember the location of the allocation interval and whether it is a left or right interval. By convention, the initial interval of a CRP is regarded as a right interval. We can now state the algorithm followed by each node precisely. The algorithm gives the allocation interval [*i.e.*, $T(k)$ and $\alpha(k)$] and the status ($\sigma = L$ or R) for slot k in terms of the feedback, allocation interval, and status from slot $k-1$.

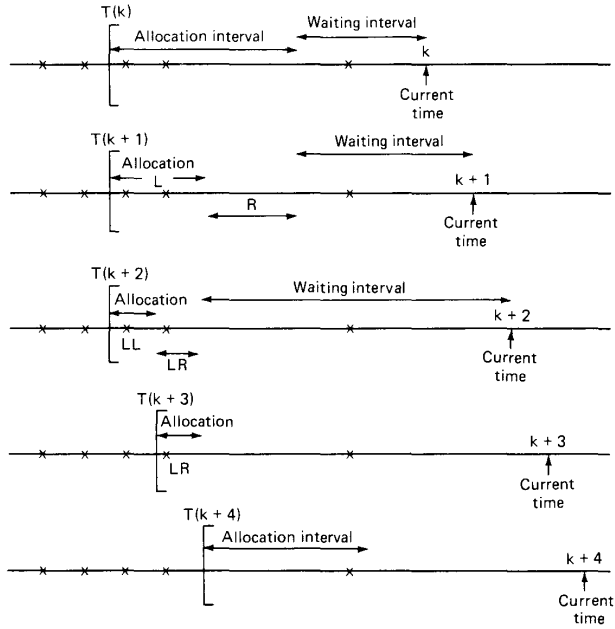


Figure 4.12 FCFS splitting algorithm. When a collision follows another collision, the interval on the right side of the second collision is returned to the waiting interval. The CRP is completed in slot $k + 3$, and a new CRP is started with an allocation interval of fixed size.

If feedback = ϵ , then

$$\begin{aligned} T(k) &= T(k-1) \\ \alpha(k) &= \frac{\alpha(k-1)}{2} \\ \sigma(k) &= L \end{aligned} \quad (4.15)$$

If feedback = 1 and $\sigma(k-1) = L$, then

$$\begin{aligned} T(k) &= T(k-1) + \alpha(k-1) \\ \alpha(k) &= \alpha(k-1) \\ \sigma(k) &= R \end{aligned} \quad (4.16)$$

If feedback = 0 and $\sigma(k-1) = L$, then

$$T(k) = T(k-1) + \alpha(k-1)$$

$$\alpha(k) = \frac{\alpha(k-1)}{2} \quad (4.17)$$

$$\sigma(k) = L$$

If feedback = 0 or 1 and $\sigma(k-1) = R$, then

$$T(k) = T(k-1) + \alpha(k-1)$$

$$\alpha(k) = \min[\alpha_0, k - T(k)] \quad (4.18)$$

$$\sigma(k) = R$$

The final statement, Eq. (4.18), is used at the end of a collision resolution or when no collisions are being resolved. The size of the new allocation interval in this case is some constant value α_0 which could be chosen either to minimize delay for a given arrival rate or to maximize the stable throughput. Naturally, when the queue becomes depleted, the allocation interval cannot include packets that have not yet arrived, so the interval size is limited to $k - T(k)$, as indicated by the min operation in Eq. (4.18).

Analysis of FCFS splitting algorithm. Figure 4.13 is helpful in visualizing the evolution of a collision resolution period; we shall see later that the diagram can be interpreted as a Markov chain. The node at the left side of the diagram corresponds to the initial slot of a CRP; the node is split in two as an artifice to visualize the beginning and end of a CRP. If an idle or success occurs, the CRP ends immediately and a new

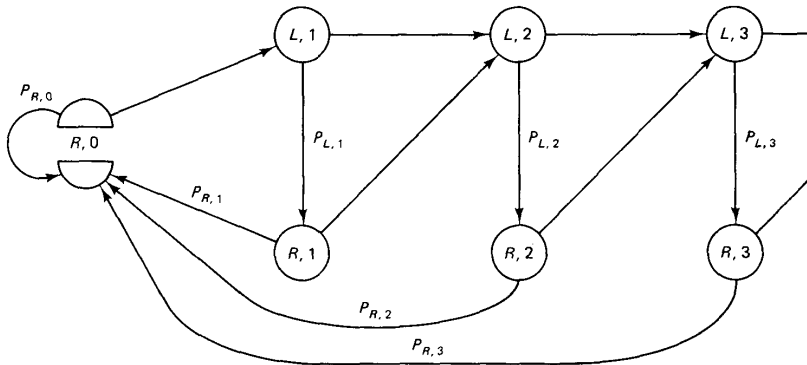


Figure 4.13 Markov chain for FCFS splitting algorithm. The top states are entered after splitting an interval and correspond to the transmission of the left side of that interval. The lower states are entered after success on the left side and correspond to transmission of the right side. Transitions from top to bottom and from bottom back to $R, 0$ correspond to successes.

CRP starts on the next slot. Alternatively, if a collision occurs, a transition occurs to node $(L, 1)$; L indicates the status and the 1 indicates that one split in the allocation interval has occurred.

Each subsequent idle or collision from a left allocation interval generates one additional split with a smaller left allocation interval, corresponding to a transition in the diagram from (L, i) to $(L, i + 1)$, where i is the number of times the original allocation interval has been split. A success from a left interval leads to a right interval with no additional split, corresponding to an (L, i) to (R, i) transition. A success from a right interval ends the CRP, with a transition back to $(R, 0)$, whereas a collision causes a new split, with a transition from (R, i) to $(L, i + 1)$.

We now analyze a single CRP. Assume that the size of the initial allocation interval is α_0 . Each splitting of the allocation interval decreases this by a factor of 2, so that nodes (L, i) and (R, i) in the diagram correspond to allocation intervals of size $2^{-i}\alpha_0$. Given our assumption of a Poisson arrival process of rate λ , the number of packets in the original allocation interval is a Poisson random variable with mean $\lambda\alpha_0$. Similarly, the a priori distributions on the numbers of packets in disjoint subintervals are independent and Poisson. Define G_i as the expected number of packets, a priori, in an interval that has been split i times,

$$G_i = 2^{-i}\lambda\alpha_0 \quad (4.19)$$

We next find the transition probabilities in Fig. 4.13 and show that they constitute a Markov chain (*i.e.*, that each transition is independent of the path used to reach the given node). Note that we are only interested (for now) in one period of collision resolution; we view the upper half of node $(R, 0)$ as the starting state and the lower half as the final state. We start from the left side and work to the right. $P_{R,0}$ is the probability of an idle or success (*i.e.*, 0 or 1 packet) in the first slot. Since the number of packets in the initial allocation interval is Poisson with mean G_0 , the probability of 0 or 1 packets is

$$P_{R,0} = (1 + G_0)e^{-G_0} \quad (4.20)$$

Next consider the state $(L, 1)$. This state is entered after a collision in state $(R, 0)$, which occurs with probability $1 - P_{R,0}$. Let x_L be the number of packets in the new allocation interval L (*i.e.*, the left half of the original interval), and let x_R be the number in R , the right half of the original interval. A priori, x_L and x_R are independent Poisson random variables of mean G_1 each. The condition that $(L, 1)$ is entered is the condition that $x_L + x_R \geq 2$. The probability of success $P_{L,1}$ is thus the probability that $x_L = 1$ conditional on $x_L + x_R \geq 2$. Noting that both $x_L = 1$ and $x_L + x_R \geq 2$ occur if and only if $x_L = 1$ and $x_R \geq 1$, we have

$$P_{L,1} = \frac{P\{x_L = 1\}P\{x_R \geq 1\}}{P\{x_L + x_R \geq 2\}} = \frac{G_1 e^{-G_1} [1 - e^{-G_1}]}{1 - (1 + G_0)e^{-G_0}} \quad (4.21)$$

State $(R, 1)$ is entered if and only if the transition above takes place (*i.e.*, if $x_L = 1$ and $x_R \geq 1$). Thus, the probability of success $P_{R,1}$ in state $(R, 1)$ is

$$P_{R,1} = \frac{P\{x_R = 1\}}{P\{x_R \geq 1\}} = \frac{G_1 e^{-G_1}}{1 - e^{-G_1}} \quad (4.22)$$

We next show that Eqs. (4.21) and (4.22) generalize to $P_{L,i}$ and $P_{R,i}$ for all $i \geq 1$. That is,

$$P_{L,i} = \frac{G_i e^{-G_i} (1 - e^{-G_i})}{1 - (1 + G_{i-1}) e^{-G_{i-1}}} \quad (4.23)$$

$$P_{R,i} = \frac{G_i e^{-G_i}}{1 - e^{-G_i}} \quad (4.24)$$

Consider state $(L, 2)$. This can be entered by a collision in state $(L, 1)$, an idle in $(L, 1)$, or a collision in $(R, 1)$. In the first case, interval L is split into LL and LR , and LL becomes the new allocation interval. In the second and third cases, R is split into RL and RR , with RL becoming the new allocation interval. For the first case, let x_{LL} and x_{LR} be the numbers of packets in LL and LR , respectively. A priori, these are independent Poisson random variables of mean G_2 each. The collision from $(L, 1)$ means that $x_L + x_R \geq 2$ and $x_L \geq 2$, which is equivalent to the single condition $x_L \geq 2$. The situation is thus the same as in finding $P_{L,1}$ except that the intervals are half as large, so $P_{L,2}$ in Eq. (4.23) is correct in this case.

Next, consider the second case, that of an idle in $(L, 1)$. This means that $x_L + x_R \geq 2$ and $x_L = 0$, which is equivalent to $x_R \geq 2$ and $x_L = 0$. $P_{L,2}$ in this case is the probability that RL , the left half of R , contains one packet given that R contains at least two; again Eq. (4.23) is correct. Finally, for the case of a collision in $(R, 1)$, we have $x_L + x_R \geq 2$, $x_L = 1$, and $x_R \geq 2$, or equivalently $x_L = 1$, $x_R \geq 2$; Eq. (4.23) is again correct for $(L, 2)$. Thus, the Markov condition is satisfied for $(L, 2)$.

Generally, no matter how $(L, 2)$ is entered, the given interval L or R preceding $(L, 2)$ is an interval of size $\alpha_0/2$ of a Poisson process, conditional on the given interval containing two or more packets. If a success occurs on the left half, the number of packets in the right half is Poisson, conditional on being one or more, yielding the expression for $P_{R,2}$ in Eq. (4.24). This argument repeats for $i = 3, 4, \dots$ (or, more formally, induction can be applied). Thus, Fig. 4.13 is a Markov chain and Eqs. (4.20), (4.23), and (4.24) give the transition probabilities.

The analysis of this chain is particularly simple since no state can be entered more than once before the return to $(R, 0)$. The probabilities, $p(L, i)$ and $p(R, i)$, that (L, i) and (R, i) , respectively, are entered before returning to $(R, 0)$ can be calculated iteratively from the initial state $(R, 0)$:

$$p(L, 1) = 1 - P_{R,0} \quad (4.25)$$

$$p(R, i) = P_{L,i} p(L, i); \quad i \geq 1 \quad (4.26)$$

$$p(L, i+1) = (1 - P_{L,i}) p(L, i) + (1 - P_{R,i}) p(R, i); \quad i \geq 1 \quad (4.27)$$

Let K be the number of slots in a CRP; thus, K is the number of states visited in the chain, including the initial state $(R, 0)$, before the return to $(R, 0)$,

$$E\{K\} = 1 + \sum_{i=1}^{\infty} [p(L, i) + p(R, i)] \quad (4.28)$$

We also must evaluate the change in $T(k)$ from one CRP to the next. For the assumed initial interval of size α_0 , this change is at most α_0 , but if left-hand intervals have collisions, the corresponding right-hand intervals are returned to the waiting interval, and the change is less than α_0 . Let f be the fraction of α_0 returned in this way over a CRP, so that $\alpha_0(1-f)$ is the change in $T(k)$. The probability of a collision in state (L, i) is the probability that the left half-interval in state (L, i) contains at least two packets given that the right and left intervals together contain at least two; that is,

$$P\{c \mid (L, i)\} = \frac{1 - (1 + G_i)e^{-G_i}}{1 - (1 + G_{i-1})e^{-G_{i-1}}} \quad (4.29)$$

The fraction of the original interval returned on such a collision is 2^{-i} , so the expected value of f is

$$E\{f\} = \sum_{i=1}^{\infty} p(L, i)P\{c \mid (L, i)\}2^{-i} \quad (4.30)$$

Note that $E\{f\}$ and $E\{K\}$ are functions only of $G_i = \lambda\alpha_0 2^{-i}$, for $i \geq 1$, and hence are functions only of the product $\lambda\alpha_0$. For large i , $P_{L,i}$ tends to $1/2$, and thus $p(L, i)$ and $p(R, i)$ tend to zero with increasing i as 2^{-i} . Thus, $E\{f\}$ and $E\{K\}$ can be easily evaluated numerically as functions of $\lambda\alpha_0$.

Finally, define the drift D to be the expected change in the time backlog, $k - T(k)$, over a CRP (again assuming an initial allocation of α_0). This is the expected number of slots in a CRP less the expected change in $T(k)$; so

$$D = E\{K\} - \alpha_0(1 - E\{f\}) \quad (4.31)$$

The drift is negative if $E\{K\} < \alpha_0(1 - E\{f\})$, or equivalently, if

$$\lambda < \frac{\lambda\alpha_0(1 - E\{f\})}{E\{K\}} \quad (4.32)$$

The right side of Eq. (4.32), as a function of $\lambda\alpha_0$, has a numerically evaluated maximum of 0.4871 at $\lambda\alpha_0 = 1.266$. $\lambda\alpha_0$ is the expected number of packets in the original allocation interval; as expected, it is somewhat larger than 1 (which would maximize the initial probability of success) because of the increased probability of success immediately after a collision. If α_0 is chosen to be 2.6 (*i.e.*, $1.266/0.4871$), then Eq. (4.32) is satisfied for all $\lambda < 0.4871$. Thus, the expected time backlog decreases (whenever it is initially larger than α_0), and we conclude* that the algorithm is stable for $\lambda < 0.4871$.

Expected delay is much harder to analyze than maximum stable throughput. Complex upper and lower bounds have been developed ([HuB85] and [TsM80]) and corre-

*For the mathematician, a more rigorous proof of stability is desirable. Define a busy period as a consecutive string of CRPs starting with a time backlog $k - T(k) < \alpha_0$ and continuing up to the beginning of the next CRP with $k - T(k) < \alpha_0$. The sequence of time backlogs at the beginnings of the CRPs in the busy period forms a random walk with the increments (except the first) having identical distributions with negative expectation for $\lambda < 0.4871$. Since $p(L, i)$ approaches 0 exponentially in i , the increments have a moment generating function, and from Wald's equality, the number N of CRPs in a busy period also has a moment generating function. Since the number of slots in a busy period is at most $N\alpha_0$, the number of slots also has a moment generating function, from which it follows that the expected delay per packet is finite.

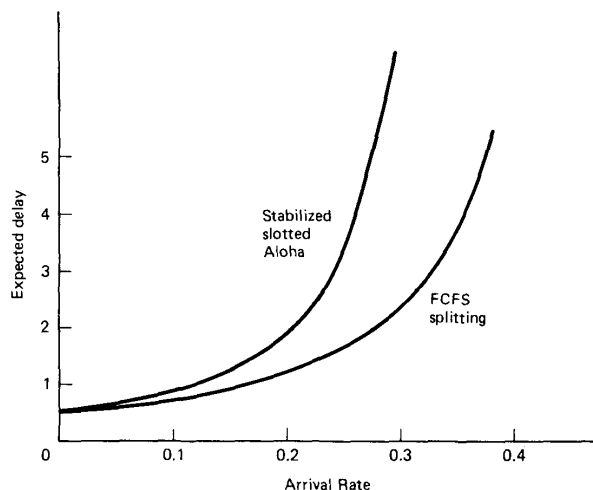


Figure 4.14 Comparison of expected delay for stabilized slotted Aloha and the FCFS splitting algorithm as a function of arrival rate. One becomes unbounded as the arrival rate approaches $1/\epsilon$, and the other as the arrival rate approaches 0.4871.

spond closely to simulation results. Figure 4.14 plots this delay and compares it with stabilized slotted Aloha.

Improvements in the FCFS splitting algorithm. Splitting intervals into equal-sized subintervals is slightly nonoptimal in achieving maximum stable throughput. When each interval is split into the optimally sized subintervals, the maximum stable throughput increases to 0.4878 ([MoH85] and [TsM80]). Another, even smaller, improvement of 3.6×10^{-7} results if in state (R, i) for large i , some of the waiting interval is appended to the right-side interval [VvP83]. While these improvements are not significant practically, they are of theoretical interest in determining optimality in terms of maximum stable throughput.

The maximum stable throughput, using assumptions 1 to 6b, is currently unknown. Considerable research has been devoted to finding upper bounds to throughput, and the tightest such bound is 0.587 [MiT81]. Thus, the maximum stable throughput achievable by any algorithm lies somewhere between 0.4878 and 0.587.

These questions of maximum throughput depend strongly on assumptions 1 to 6b. For any finite set of m nodes, we have seen that TDM can trivially achieve any throughput up to one packet per slot. This striking difference between finite and infinite m seems paradoxical until we recognize that with TDM, expected delay (for a given λ) increases linearly with m , whereas the algorithms that assume $m = \infty$ achieve a delay bounded independently of m .

We shall also see in the next two sections that much higher throughputs than 0.4878 are achievable if the slotted assumption is abandoned and early feedback is available when the channel is idle or experiencing a collision. Finally, rather surprisingly, if the feedback is expanded to specify the number of packets in each collision, maximum stable throughput again increases to 1 [Pip81]. Unfortunately, this type of feedback is difficult to achieve in practice, and no algorithms are known for achieving these high throughputs even if the feedback were available.

Practical details. The FCFS splitting algorithm is subject to the same deadlock problem as the first improvement on the tree algorithm if the feedback from an idle slot is mistaken as a collision. As before, this deadlock condition is eliminated by specifying a maximum number h of successive repetitions of Eq. (4.17) in the algorithm. On the $(h + 1)$ th successive idle after a collision, Eq. (4.16) is performed.

Also, the algorithm assumes that nodes can measure arrival times with infinite precision. In practice, if arrival times are measured with a finite number of bits, each node would generate extra bits, as needed for splitting, by a pseudo-random number generator.

Last-come first-serve (LCFS) splitting algorithm. The FCFS splitting algorithm requires all nodes to monitor the channel feedback at all times. A recent modification allows nodes to monitor the feedback only after receiving a packet to transmit ([Hum86] and [GeP85]). The idea is to send packets in approximately last-come first-serve (LCFS) order; thus, the most recently arrived packets need not know the length of the waiting set since they have first priority in transmission.

Figure 4.15 illustrates this variation. New arrivals are in a “prewaiting mode” until they receive enough feedback to detect the end of a CRP; they then join the waiting set. The end of a CRP can be detected by feedback equal to 1 in one slot, followed by either 0 or 1 in the next. Also, assuming the practical modification in which Eq. (4.17) can be repeated at most h successive times, feedback of h successive 0’s followed by 1 or 0 also implies the end of a CRP.

After the waiting set has been extended on the right by the interval of time over which nodes can detect the end of the CRP, a new allocation set is chosen at the right end of the waiting set (thus, including part or all of the new interval). As shown in the figure, the waiting set, and consequently the allocation set, might consist of several disjoint intervals.

After joining the waiting set, backlogged nodes keep track of their distance from the right end of the waiting set. This distance includes only unresolved intervals and unallocated (left or right) intervals. Nodes in the allocation set similarly track their distance from the right end of the allocation set. When a collision occurs, the allocated set splits as before, but here the right half is allocated next. Upon a right-half collision, the corresponding left half is adjoined to the right end of the waiting set, increasing the distance of the right end from the previously waiting nodes. At the end of a CRP, the new arrival interval from which this event is detectable is first appended to the right end of the waiting set, and then the new allocation set is removed from the right end. Nodes

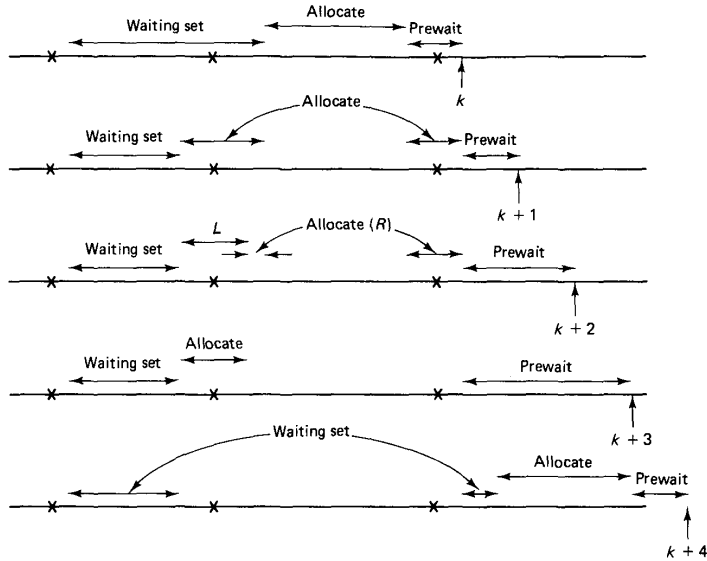


Figure 4.15 Last-come first-serve (LCFS) splitting algorithm. Arrivals wait for the beginning of a new CRP, but then are allocated in last-come first-serve fashion. Allocation sets and waiting sets can consist of more than one interval.

do not know where the left end of the waiting set is, and thus the allocation set always has size α_0 (not counting gaps), with perhaps dummy space at the left.

When the backlog is large, the LCFS splitting algorithm has the same drift as the FCFS splitting algorithm, and thus it is stable for the same range of λ . With increasing h [where h is the allowable number of repetitions of Eq. (4.17) in the algorithm], the upper limit of this range approaches 0.4871 as before. The expected delay is somewhat larger than that for FCFS, primarily because of the time packets spend in the prewaiting mode.

Delayed feedback. Assume that the feedback for the k^{th} slot arrives sometime between the beginning of slot $k + j - 1$ and $k + j$ for some fixed $j > 1$. Visualize time-division multiplexing the slots on the channel between j different versions of the FCFS splitting algorithm, with the exception of maintaining a common waiting set for all j algorithms. Each node monitors the progress of each of the j current CRPs and tracks the extent of the waiting set. At the end of a CRP for one of the j versions, an allocation set of size α_0 (or less if the waiting set is smaller) is removed from the left end of the waiting set to start the next CRP on that version. Since the j versions experience different delays, packets are no longer served in first-come first-serve order.

When a right subset is returned to the waiting set due to a collision in a left subset for one of the j versions, it is appended to the left end of the waiting set. These returns

can fragment the waiting set into disjoint intervals, but nodes need only keep track of the size of the set not counting the gaps.

The same analysis as before justifies that the maximum stable throughput for any finite j is still 0.4871. The expected delay is larger than that of FCFS splitting because of the additional time required to resolve collisions. Note that the delay can essentially be considered as having two components—first, the delay in resolving collisions, which increases roughly with j , and second, the delay in the waiting set, which is roughly independent of j . For large j and small λ , this suggests that α_0 should be reduced, thus reducing the frequency of collisions at some expense to the waiting set size.

Round-robin splitting. A final variation of the FCFS splitting algorithm [HIG81] can be applied if there is an identifiable finite collection of nodes numbered 1 to m . Consider the nodes to be arranged conceptually in a circle with node $i + 1$ following i , $1 \leq i < m$, and node 1 following m . Rather than forming allocation sets in terms of packet arrival times, an allocation set consists of a contiguous set of nodes, say i to j around the circle. After completion of a CRP, the algorithm then allocates the next successive set of nodes around the circle. The size of allocation sets initiating CRPs varies with the time to pass around the circle, so that under light loading, an initial allocation set would contain all nodes, whereas under heavy loading, initial allocation sets would shrink to single nodes, which is equivalent to TDM.

4.4 CARRIER SENSING

In many multiaccess systems, such as local area networks, a node can hear whether other nodes are transmitting after a very small propagation and detection delay relative to a packet transmission time. The detection delay is the time required for a physical receiver to determine whether or not some other node is currently transmitting. This delay differs somewhat from the delay, first, in detecting the beginning of a new transmission, second, in synchronizing on the reception of a new transmission, and third, in detecting the end of an old transmission. We ignore these and other subtleties of the physical layer in what follows, and simply regard the medium as an intermittent synchronous multiaccess bit pipe on which idle periods can be distinguished (with delay) from packet transmission periods.

If nodes can detect idle periods quickly, it is reasonable to terminate idle periods quickly and to allow nodes to initiate packet transmissions after such idle detections. This type of strategy, called carrier sense multiple access (CSMA) [KIT75], does not necessarily imply the use of a carrier but simply the ability to detect idle periods quickly.

Let β be the propagation and detection delay (in packet transmission units) required for all sources to detect an idle channel after a transmission ends. Thus if τ is this time in seconds, C is the raw channel bit rate, and L is the expected number of bits in a data packet, then

$$\beta = \frac{\tau C}{L} \quad (4.33)$$

We shall see that the performance of CSMA degrades with increasing β and thus also degrades with increasing channel rate and with decreasing packet size.

Consider a slotted system in which, if nothing is being transmitted in a slot, the slot terminates after β time units and a new slot begins. This assumption of dividing idle periods into slots of length β is not realistic, but it provides a simple model with good insight. We have thus eliminated our previous assumption that time is slotted into equal-duration slots. We also eliminate our assumption that all data packets are of equal length, although we still assume a time normalization in which the expected packet transmission time is 1. In place of the instantaneous feedback assumption, we assume 0.1. e feedback with a maximum delay β , as indicated above. For simplicity, we continue to assume an infinite set of nodes and Poisson arrivals of overall intensity λ . We first modify slotted Aloha for this new situation, then consider unslotted systems, and finally consider the FCFS splitting algorithm.

4.4.1 CSMA Slotted Aloha

The major difference between CSMA slotted Aloha and ordinary slotted Aloha is that idle slots in CSMA have a duration β . The other difference is that if a packet arrives at a node while a transmission is in progress, the packet is regarded as backlogged and begins transmission with probability q_r after each subsequent idle slot; packets arriving during an idle slot are transmitted in the next slot as usual. This technique was called nonpersistent CSMA in [KIT75] to distinguish it from two variations. In one variation, persistent CSMA, all arrivals during a busy slot simply postpone transmission to the end of that slot, thus causing a collision with relatively high probability. In the other, P-persistent CSMA, collided packets and new packets waiting for the end of a busy period use different probabilities for transmission. Aside from occasional comments, we will ignore these variations since they have no important advantages over nonpersistent CSMA.

To analyze CSMA Aloha, we can use a Markov chain again, using the number n of backlogged packets as the state and the ends of idle slots as the state transition times. Note that each busy slot (success or collision) must be followed by an idle slot, since nodes are allowed to start transmission only after detecting an idle slot. For simplicity, we assume that all data packets have unit length. The extension to arbitrary length packets is not difficult, however, and is treated in Problem 4.21. The time between successive state transitions is either β (in the case of an idle slot) or $1 + \beta$ (in the case of a busy slot followed by an idle). Rather than present the state transition equations, which are not particularly insightful, we simply modify the drift in Eq. (4.4) for this new model. At a transition into state n (*i.e.*, at the end of an idle slot), the probability of no transmissions in the following slot (and hence the probability of an idle slot) is $e^{-\lambda\beta}(1 - q_r)^n$. The first term is the probability of no arrivals in the previous idle slot, and the second is the probability of no transmissions by the backlogged nodes. Thus, the expected time between state transitions in state n is $\beta + [1 - e^{-\lambda\beta}(1 - q_r)^n]$. Similarly, the expected number of arrivals between state transitions is

$$E\{\text{arrivals}\} = \lambda[\beta + 1 - e^{-\lambda\beta}(1 - q_r)^n] \quad (4.34)$$

The expected number of departures between state transitions in state n is simply the probability of a successful transmission; assuming that $q_r < 1$, this is given by

$$P_{succ} = \left(\lambda\beta + \frac{q_r n}{1 - q_r} \right) e^{-\lambda\beta(1 - q_r)^n} \quad (4.35)$$

The drift in state n is defined as the expected number of arrivals less the expected number of departures between state transitions,

$$D_n = \lambda[\beta + 1 - e^{-\lambda\beta(1 - q_r)^n}] - \left(\lambda\beta + \frac{q_r n}{1 - q_r} \right) e^{-\lambda\beta(1 - q_r)^n} \quad (4.36)$$

For small q_r , we can make the approximation $(1 - q_r)^{n-1} \approx (1 - q_r)^n \approx e^{-q_r n}$, and D_n can be expressed as

$$D_n \approx \lambda(\beta + 1 - e^{-g(n)}) - g(n)e^{-g(n)} \quad (4.37)$$

where

$$g(n) = \lambda\beta + q_r n \quad (4.38)$$

is the expected number of attempted transmissions following a transition to state n . From Eq. (4.37), the drift in state n is negative if

$$\lambda < \frac{g(n)e^{-g(n)}}{\beta + 1 - e^{-g(n)}} \quad (4.39)$$

The numerator in Eq. (4.39) is the expected number of departures per state transition, and the denominator is the expected duration of a state transition period; thus, the ratio can be interpreted as departure rate (*i.e.*, expected departures per unit time) in

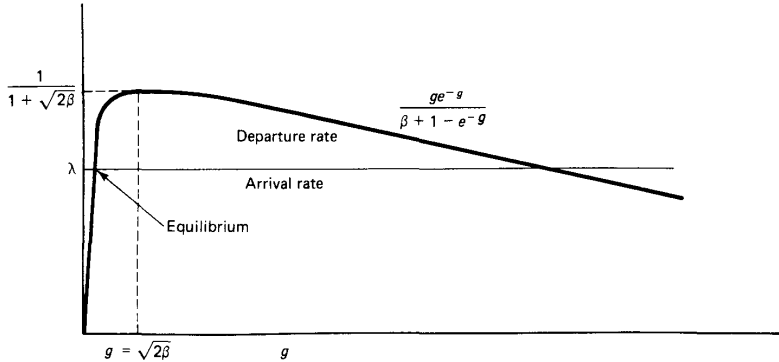


Figure 4.16 Departure rate, in packets per unit time, for CSMA slotted Aloha as a function of the attempted transmission rate g in packets per idle slot. If β , the duration of an idle slot as a fraction of a data slot, is small, the maximum departure rate is $1/(1 + \sqrt{2\beta})$.

state n . Figure 4.16 plots this ratio as a function of $g(n) = \lambda\beta + q_r n$. For small β , this function has a maximum of approximately $1/(1 + \sqrt{2\beta})$ at $g(n) = \sqrt{2\beta}$. This can be seen by approximating $e^{g(n)}$ by $1 + g(n) + g^2(n)/2$ for small $g(n)$. To understand intuitively why the departure rate is maximized at $g = \sqrt{2\beta}$, note that for small β , very little time is wasted on a single idle slot, and significant time is wasted on a collision. The point $g = \sqrt{2\beta}$ is where idles occur so much more frequently than collisions that the same expected overall time is wasted on each.

Figure 4.16 also shows that CSMA Aloha has the same stability problem as ordinary slotted Aloha. For fixed q_r , $g(n)$ grows with the backlog n , and when n becomes too large, the departure rate is less than the arrival rate, leading to yet larger backlogs. From a practical standpoint, however, the stability problem is less serious for CSMA than for ordinary Aloha. Note that β/q_r is the expected idle time that a backlogged node must wait to attempt transmission, and for small β and modest λ , q_r can be quite small without causing appreciable delay. This means that the backlog must be very large before instability sets in, and one might choose simply to ignore the problem.

P-persistent CSMA, in which packets are transmitted after idle slots with probability p if they are new arrivals and transmitted with some much smaller probability q_r if they have had collisions, is a rudimentary way of obtaining a little extra protection against instability. The next section explores stabilization in a more fundamental way.

4.4.2 Pseudo-Bayesian Stabilization for CSMA Aloha

Consider all packets as backlogged immediately after entering the system. At the end of each idle slot, each backlogged packet is independently transmitted with probability q_r , which will vary with the estimated channel backlog $\hat{n}$. In state n , the expected number of packets transmitted at the end of an idle slot is $g(n) = nq_r$. Since we have seen that the packet departure rate (in packets per unit time) is maximized (for small β and q_r) by $g(n) = \sqrt{2\beta}$, we choose q_r , for a given estimated backlog $\hat{n}$, as

$$q_r(\hat{n}) = \min\left[\frac{\sqrt{2\beta}}{\hat{n}}, \sqrt{2\beta}\right] \quad (4.40)$$

The min operation prevents $q_r(\hat{n})$ from getting too large when $\hat{n}$ is small; we cannot expect $n/\hat{n}$ to approach 1 when the backlog is small, and it is desirable to prevent too many collisions in this case. The appropriate rule for updating the estimated backlog (again assuming unit length packets) is

$$\hat{n}_{k+1} = \begin{cases} \hat{n}_k[1 - q_r(\hat{n}_k)] + \lambda\beta & \text{for idle} \\ \hat{n}_k[1 - q_r(\hat{n}_k)] + \lambda(1 + \beta) & \text{for success} \\ \hat{n}_k + 2 + \lambda(1 + \beta) & \text{for collision} \end{cases} \quad (4.41)$$

This rule is motivated by the fact that if the a priori distribution of n_k is Poisson with mean $\hat{n}_k$, then, given an idle, the a posteriori distribution of n_k is Poisson with mean $\hat{n}_k[1 - q_r(\hat{n}_k)]$ (see Problem 4.20). Accounting for the Poisson arrivals in the idle slot of duration β , the resulting distribution of n_{k+1} is Poisson with mean $\hat{n}_{k+1}$ as shown above. Similarly, given a successful transmission, the a posteriori distribution on $n_k - 1$

(removing the successful packet) is Poisson with mean $\hat{n}_k[1 - q_r(\hat{n}_k)]$. Accounting for the Poisson arrivals in the successful slot and following idle slot, n_{k+1} is Poisson with mean $\hat{n}_{k+1}$ as shown. Finally, if a collision occurs, the a posteriori distribution of n_k is not quite Poisson, but is reasonably approximated as Poisson with mean $\hat{n}_k + 2$. Adding $\lambda(1 + \beta)$ for the new arrivals, we get the final expression in Eq. (4.41).

Note that when n_k and $\hat{n}_k$ are small, then q_r is large and new arrivals are scarcely delayed at all. When $\hat{n}_k \approx n_k$ and n_k is large, the departure rate is approximately $1/(1 + \sqrt{2\beta})$, so that for $\lambda < 1/(1 + \sqrt{2\beta})$, the departure rate exceeds the arrival rate, and the backlog decreases on the average. Finally, if $|n_k - \hat{n}_k|$ is large, the expected change in backlog can be positive, but the expected change in $|n_k - \hat{n}_k|$ is negative; Fig. 4.5 again provides a qualitative picture of the expected changes in n_k and $n_k - \hat{n}_k$.

We now give a crude analysis of delay for this strategy (and other similar stabilized strategies) by using the same type of analysis as in Section 4.2.3. Let W_i be the delay from the arrival of the i^{th} packet until the beginning of the i^{th} successful transmission. The average of W_i over all i is the expected queueing delay W . Let n_i be the number of backlogged packets at the instant before i 's arrival, not counting any packet currently in successful transmission. Then

$$W_i = R_i + \sum_{j=1}^{n_i} t_j + y_i \quad (4.42)$$

where R_i is the residual time to the next state transition, t_j ($1 \leq j \leq n_i$) is the sequence of subsequent intervals until each of the next n_i successful transmissions are completed, and y_i is the remaining interval until the i^{th} successful transmission starts.

The backlog is at least 1 in all of the state transition intervals in the period on the right-hand side of Eq. (4.42), and we make the simplifying approximation that the number of attempted transmissions in each of these intervals is Poisson with parameter g . We later choose $g = \sqrt{2\beta}$, but for the moment it can be arbitrary. This approximation is somewhat different from that in Section 4.2.3, in which we assumed that a successful transmission always occurred with a backlog state of 1; the difference is motivated by Eq. (4.40), which keeps q_r small. The expected value for each t_j is given by

$$E\{t\} = e^{-g}(\beta + E\{t\}) + ge^{-g}(1 + \beta) + [1 - (1 + g)e^{-g}](1 + \beta + E\{t\}) \quad (4.43)$$

The first term corresponds to an idle transmission in the first state transition interval; this occurs with probability e^{-g} , uses time β , and requires subsequent time $E\{t\}$ to complete the successful transmission. The next two terms correspond similarly to a success and a collision, respectively. Solving for $E\{t\}$ gives

$$E\{t\} = \frac{\beta + 1 - e^{-g}}{ge^{-g}} \quad (4.44)$$

Note that this is the reciprocal of the expected number of departures per unit time in Eq. (4.39), as we would expect. $E\{t\}$ is thus approximately minimized by $g = \sqrt{2\beta}$. Averaging over i and using Little's result in Eq. (4.42), we get

$$W(1 - \lambda E\{t\}) = E\{R\} + E\{y\} \quad (4.45)$$

The expected residual time can be approximated by observing that the system spends a fraction $\lambda(1+\beta)$ of the time in successful state transition intervals. The expected residual time for arrivals in these intervals is $(1+\beta)/2$. The fraction of time spent in collision intervals is negligible (for small β) compared with that for success, and the residual time in idle intervals is negligible. Thus,

$$E\{R\} \approx \frac{\lambda(1+\beta)^2}{2} \quad (4.46)$$

Finally, $E\{y\}$ is just $E\{t\}$ less the successful transmission interval, so $E\{y\} = E\{t\} - (1+\beta)$. Substituting these expressions into Eq. (4.45) yields

$$W \approx \frac{\lambda(1+\beta)^2 + 2[E\{t\} - (1+\beta)]}{2[1 - \lambda E\{t\}]} \quad (4.47)$$

This expression is minimized over g by minimizing $E\{t\}$, and we have already seen that this minimum (for small β) is $1 + \sqrt{2\beta}$, occurring at $g = \sqrt{2\beta}$. With this substitution, W is approximately

$$W \approx \frac{\lambda + 2\sqrt{2\beta}}{2[1 - \lambda(1 + \sqrt{2\beta})]} \quad (4.48)$$

Note the similarity of this expression with the $M/D/1$ queueing delay given in Eq. (3.45). What we achieve by stabilizing CSMA Aloha is the ability to modify q_r with the backlog so as to maintain a departure rate close to $1/(1 + \sqrt{2\beta})$ whenever a backlog exists.

4.4.3 CSMA Unslotted Aloha

In CSMA slotted Aloha, we assumed that all nodes were synchronized to start transmissions only at time multiples of β in idle periods. Here we remove that restriction and assume that when a packet arrives, its transmission starts immediately if the channel is sensed to be idle. If the channel is sensed to be busy, or if the transmission results in a collision, the packet is regarded as backlogged. Each backlogged packet repeatedly attempts to retransmit at randomly selected times separated by independent, exponentially distributed random delays τ , with probability density $xe^{-x\tau}$. If the channel is idle at one of these times, the packet is transmitted, and this continues until such a transmission is successful. We again assume a propagation and detection delay of β , so that if one transmission starts at time t , another node will not detect that the channel is busy until $t + \beta$, thus causing the possibility of collisions.

Consider an idle period that starts with a backlog of n . The time until the first transmission starts (with the $m = \infty$ assumption) is an exponentially distributed random variable with rate

$$G(n) = \lambda + nx \quad (4.49)$$

Note that $G(n)$ is the attempt rate in packets per unit time, whereas $g(n)$ in Section 4.4.2 was packets per idle slot. After the initiation of this first transmission, the backlog is either n (if a new arrival started transmission) or $n - 1$ (if a backlogged packet started).

Thus the time from this first initiation until the next new arrival or backlogged node senses the channel is an exponentially distributed random variable of rate $G(n)$ or $G(n-1)$. A collision occurs if this next sensing is done within time β . Thus, the probability that this busy period is a collision is $1 - e^{-\beta G(n)}$ or $1 - e^{-\beta G(n-1)}$. This difference is small if βx is small, and we neglect it in what follows. Thus, we approximate the probability of a successful transmission following an idle period by $e^{-\beta G(n)}$.

The expected time from the beginning of one idle period until the next is $1/G(n) + (1 + \beta)$; the first term is the expected time until the first transmission starts, and the second term $(1 + \beta)$ is the time until the first transmission ends and the channel is detected as being idle again. If a collision occurs, there is a slight additional time, less than β , until the packets causing the collision are no longer detected; we neglect this contribution since it is negligible even with respect to β , which is already negligible. The departure rate during periods when the backlog is n is then given by

$$\text{departure rate}(n) = \frac{e^{-\beta G(n)}}{1/G(n) + (1 + \beta)} \quad (4.50)$$

For small β , the maximum value of this departure rate is approximately $1/(1+2\sqrt{\beta})$, occurring when $G(n) \approx \beta^{-1/2}$. This maximum departure rate is slightly smaller than it is for the slotted case [see Eq. (4.39)]; the reason is the same as when CSMA is not being used—collisions are somewhat more likely for a given attempt rate in an unslotted system than a slotted system. For CSMA, with small β , however, this loss in departure rate is quite small. What is more, in a slotted system, β would have to be considerably larger than in an unslotted system to compensate for synchronization inaccuracies and worst-case propagation delays. Thus, unslotted Aloha appears to be the natural choice for CSMA.

CSMA unslotted Aloha has the same stability problems as all the Aloha systems, but it can be stabilized in the same way as CSMA slotted Aloha. The details are treated in Problem 4.22.

4.4.4 FCFS Splitting Algorithm for CSMA

We next investigate whether higher throughputs or smaller delays can be achieved by the use of splitting algorithms with CSMA. We shall see that relatively little can be gained, but it is interesting to understand why. We return to the assumption of idle slots of duration β and assume that 0, 1, e feedback is available. An idle slot occurs at the end of each success or collision to provide time for feedback. Thus, we regard successes and collisions as having a duration $1 + \beta$; the algorithm is exercised at the end of each such elongated success or collision slot, and also at the end of each normal idle slot.

The same algorithm as in Eqs. (4.15) to (4.18) can be used, although the size α_0 of the initial interval in a CRP should be changed. Furthermore, as we shall see shortly, intervals with collisions should not be split into equal subintervals. Since collisions waste much more time than idle slots, the basic allocation interval α_0 should be small. This means in turn that collisions with more than two packets are negligible, and thus the analysis is simpler than before.

We first find the expected time and the expected number of successes in a CRP. Let $g = \lambda\alpha_0$ be the expected number of arrivals in an initial allocation interval of size α_0 . With probability e^{-g} , an original allocation interval is empty, yielding a collision resolution time of β with no successes. With probability ge^{-g} , there is an initial success, yielding a collision resolution time $1 + \beta$. Finally, with probability $(g^2/2)e^{-g}$, there is a collision, yielding a collision resolution time of $1 + \beta + T$ for some T to be calculated later; since collisions with more than two packets are negligible, we assume two successes for each CRP with collisions. Thus,

$$E\{\text{time/CRP}\} \approx \beta e^{-g} + (1 + \beta)ge^{-g} + (1 + \beta + T)\frac{g^2}{2}e^{-g} \quad (4.51)$$

$$E\{\text{packets/CRP}\} \approx ge^{-g} + 2\frac{g^2}{2}e^{-g} \quad (4.52)$$

As before, the maximum stable throughput for a given g is

$$\lambda_{\max} = \frac{E\{\text{packets/CRP}\}}{E\{\text{time/CRP}\}} \approx \frac{g + g^2}{\beta + g(1 + \beta) + (g^2/2)(1 + \beta + T)} \quad (4.53)$$

We can now maximize the right-hand side of Eq. (4.53) over g (*i.e.*, over α_0). In the limit of small β , we get the asymptotic expressions

$$g \approx \sqrt{\frac{2\beta}{T-1}} \quad (4.54)$$

$$\lambda_{\max} \approx \frac{1}{1 + \sqrt{2\beta(T-1)}} \quad (4.55)$$

Finally, we must calculate T , the time to resolve a collision after it has occurred. Let x be the fraction of an interval used in the first subinterval when the interval is split; we choose x optimally later. The first slot after the collision is detected is idle, successful, or a collision with probabilities $(1-x)^2$, $2x(1-x)$, or x^2 , respectively. The expected time required for each of these three cases is $\beta + T$, $2(1 + \beta)$, and $1 + \beta + T$. Thus,

$$T \approx (1-x)^2(\beta + T) + 4x(1-x)(1 + \beta) + x^2(1 + \beta + T) \quad (4.56)$$

from which T can be expressed as a function of x .

By setting the derivative dT/dx to 0, we find after a straightforward calculation that T is minimized by

$$x = \sqrt{\beta + \beta^2} - \beta \quad (4.57)$$

The resulting value of T , for small β , is $T \approx 2 + \sqrt{\beta}$. Substituting this in Eq. (4.55), we see that

$$\lambda_{\max} \approx \frac{1}{1 + \sqrt{2\beta}} \quad (4.58)$$

For small β , then, the FCFS splitting algorithm has the same maximum throughput as slotted Aloha. This is not surprising, since without CSMA, the major advantage of the FCFS algorithm is its efficiency in resolving collisions, and with CSMA, collisions rarely occur. When collisions do occur, they are resolved in both strategies by retransmission

with small probability. It is somewhat surprising at first that if we use the FCFS algorithm with equal subintervals (*i.e.*, $x = 1/2$), we find that we are limited to a throughput of $1/(1 + \sqrt{3.3})$. This degradation is due to a substantial increase in the number of collisions.

4.5 MULTIACCESS RESERVATIONS

We now look at a very simple way of increasing the throughput of multiaccess channels that has probably become apparent. If the packets of data being transmitted are long, why waste these long slot times either sending nothing or sending colliding packets? It would be far more efficient to send very short packets either in a contention mode or a TDM mode, and to use those short packets to reserve longer noncontending slots for the actual data. Thus, the slots wasted by idles or collisions are all short, leading to a higher overall efficiency. There are many different systems that operate in this way, and our major objective is not so much to explore the minor differences between them, but to see that they are all in essence the same.

To start, we explore a somewhat "canonic" reservation system. Assume that data packets require one time unit each for transmission and that reservation packets require $v \ll 1$ time units each for transmission. The format of a reservation packet is unimportant; it simply has to contain enough information to establish the reservation. For example, with the instantaneous feedback indicating idle, success, or collision that we have been assuming, the reservation packet does not have to contain any information beyond its mere existence. After a successful reservation packet is transmitted, either the next full time unit or some predetermined future time can be automatically allocated for transmission of the corresponding data packet. The reservation packets can use any strategy, including time-division multiplexing, slotted Aloha, or the splitting algorithm.

We can easily determine the maximum throughput S in data packets per time unit achievable in such a scheme. Let S_r be the maximum throughput, in successful reservation packets per reservation slot, of the algorithm used for the reservation packets (*i.e.*, $1/e$ for slotted Aloha, 0.478 for splitting, or 1 for TDM). Then, over a large number of reservations, the time required per reservation approaches v/S_r , and an additional one unit of time is required for each data packet. Thus, the total time per data packet approaches $1 + v/S_r$, and we see that

$$S = \frac{1}{1 + v/S_r} \quad (4.59)$$

This equation assumes that the reservation packet serves only to make the reservation and carries no data. As we shall see, in many systems, the reservation packet carries some of the data; thus for one time unit of data, it suffices to transmit the reservation packet of duration v followed by the rest of the data in time $1 - v$. In this case, the throughput becomes

$$S = \frac{1}{1 + v(1/S_r - 1)} \quad (4.60)$$

For example, with slotted Aloha, the throughput is $S = 1/[1 + v(e - 1)]$. It is apparent that if v is small, say on the order of 0.01, then the maximum throughput

approaches 1 and is not highly dependent on the collision resolution algorithm or on whether the reservation packet carries part of the data. We shall see later that Ethernet local area networks [MeB76] can be modeled almost in this way, and v about 0.01 is typical. Thus, such networks can achieve very high throughputs without requiring much sophistication for collision resolution.

4.5.1 Satellite Reservation Systems

One of the simplest reservation systems, with particular application to satellite networks [JBH78], has a frame structure as shown in Fig. 4.17. A number of data slots are preceded by a reservation period composed of m reservation slots, one reservation slot per node. Let v be the duration of a reservation slot, where, as usual, time is normalized to the average duration of a data packet; thus, v is the ratio of the number of bits used to make a reservation (including guard space and overhead) to the expected number of bits in a data packet. The reservation period in each frame has an overall duration of $A = mv$. The minimum duration of a frame is set to exceed the round-trip delay 2β , so that the reservation slots at the beginning of one frame allocate the data slots for the next frame (see Fig. 4.17).

A frame is extended beyond the minimum length if need be to satisfy all the reservations. Note that the use of TDM for the reservation slots here makes a great deal more sense than it does for ordinary TDM applied to data slots. One reason is that the reservation slots are short, and therefore little time is wasted on a source with nothing to send. Another reason, for satellites, is that if collisions were allowed on the reservation slots, the channel propagation delay would make collision resolution rather

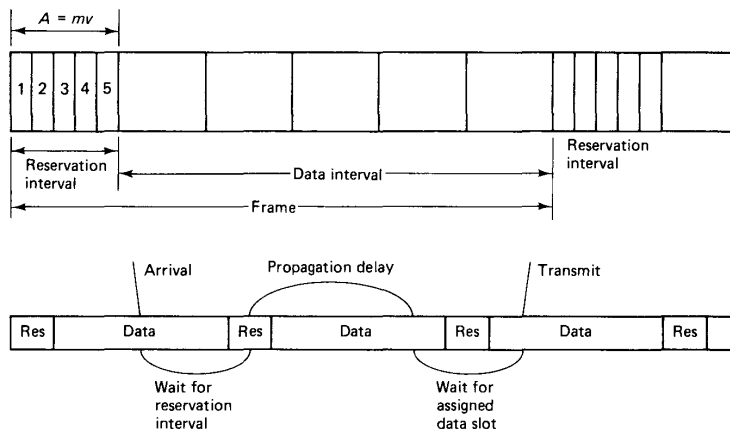


Figure 4.17 Satellite reservation system, using TDM to make reservations. Arrivals in one frame are transmitted in the second following frame.

slow. Equation (4.59) gives the maximum throughput of such a system as $1/(1+v)$ under the assumption that each reservation packet can make only one data packet reservation. If we assume, alternatively, that a reservation packet can make multiple reservations for its source, it is not hard to see that the throughput can approach 1 arbitrarily closely, since under heavy loading, the frames get very long and negligible time is used by the infrequent reservation periods.

With the assumption that each reservation packet can make multiple data packet reservations, this system is very similar to the single-user reservation system analyzed in Section 3.5.2. The major difference is that because of the propagation delay, the reservations requested in one reservation interval are for the data frame following the next reservation interval. Thus, if we assume Poisson packet arrivals, and if we neglect the minimum frame length requirement, the expected queueing delay for the i^{th} packet becomes

$$E\{W_i\} = E\{R_i\} + \frac{E\{N_i\}}{\mu} + 2A \quad (4.61)$$

This expression is the same as Eq. (3.60), except that the last term is $2A$. Here $A = mv$ is the duration of the reservation interval, and it is multiplied by 2 since the packet has to wait for the next two reservation intervals before being transmitted. Using the same analysis as in Chapter 3, the expected queueing delay is

$$W = \frac{\lambda \bar{X}^2}{2(1-\lambda)} + \frac{A}{2} + \frac{2A}{1-\lambda} \quad (4.62)$$

This analysis allows the data packets to have a general length distribution with mean square $\bar{X}^2$ (which of course requires the reservations to contain length information), but we have normalized time to satisfy $\bar{X} = 1/\mu = 1$; thus, $\rho = \lambda$. Note that in the limit as v goes to 0, Eq. (4.62) goes to the queueing delay of an $M/G/1$ queue as we would expect. Also, W remains finite for $\lambda < 1$ as we predicted.

Unfortunately, this analysis has neglected the condition that the duration of each frame must be at least the round-trip delay 2β . Since 2β is typically many times larger than the reservation period A , we see that W in Eq. (4.62) is only larger than 2β for λ very close to 1. Since every packet must be delayed by at least 2β for the reservation to be made, we conclude that Eq. (4.62) is not a good approximation to delay except perhaps for λ very close to 1.

Rather than try to make this analysis more realistic, we observe that this variable-frame-length model has some undesirable features. First, if some nodes make errors in receiving the reservation information, those nodes will lose track of the next reservation period; developing a distributed algorithm to keep the nodes synchronized on the reservation periods in the presence of errors is not easy. Second, the system is not very fair in the sense that very busy nodes can reserve many packets per frame, making the frames long and almost locking out more modest users.

For both these reasons, it is quite desirable to maintain a fixed frame length. Nodes can still make multiple reservations in one reservation slot, which is desirable if only a few nodes are active. With a fixed frame, however, it is sometimes necessary to postpone packets with reservations from one frame to the next.

Note that all nodes are aware of all reservations after the delay of 2β suffered by reservation packets. Thus, conceptually, we have a common queue of packets with reservations, and the nodes can jointly exercise any desired queueing discipline, such as first-come first-serve, round robin, or some priority scheme. As long as a packet is sent in every data slot for which the queue of packets with reservations is not empty, and as long as the discipline is independent of packet length, the expected delay is independent of queueing discipline.

It is quite messy to find the expected delay for this system, but with the slight modification in Fig. 4.18, a good approximate analysis is very simple. Suppose that a fraction γ of the available bandwidth is set aside for making reservations, and that TDM is used within this bandwidth, giving each node one reservation packet in each round-trip delay period 2β . With v as the ratio of reservation packet length to data packet length, $\gamma = mv/2\beta$. An arriving packet waits $2\beta/2$ time units on the average until the beginning of its reservation packet, then $2\beta/m$ units for the reservation packet transmission, and then 2β time units until the reservation is received. Thus, after $2\beta(3/2 + 1/m)$ time units, on the average, a packet joins the common queue.

The arrival process of packets with reservations to the common queue is approximately Poisson (*i.e.*, the number of arrivals in different reservation slots are independent and have a Poisson distribution). Once a packet is in the common queue, it has a service time of $X/(1 - \gamma)$, where X is the packet transmission time using the full bandwidth. The common queue is thus $M/G/1$ with $\rho = \lambda/(1 - \gamma)$. The total queueing delay, adding the expected time to enter the common queue [*i.e.*, $2\beta(3/2 + 1/m)$] to the $M/G/1$ delay in the common queue, is then

$$W = 3\beta + \frac{2\beta}{m} + \frac{\lambda \overline{X^2}}{2(1 - \gamma - \lambda)(1 - \gamma)} \quad (4.63)$$

We see that this strategy essentially achieves perfect scheduling at the expense of the delay for making reservations. For small λ , this delay seems excessive, since a number of data slots typically are wasted in each frame. This leads to the clever idea of using the unscheduled slots in a frame in a contention mode [WiE80]. Reservations are made in the following reservation interval for packets sent in these unscheduled slots. The convention is that if the packet gets through, its reservation is canceled. This approach has the effect of achieving very small delay under light loading (because of no

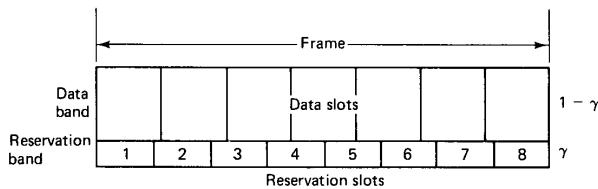


Figure 4.18 Reservation system with separate frequency band for reservations. Reservations are made by TDM in the reservation band.

round-trip delay for reservations) and very high efficiency [as given by Eq. (4.63)] under heavy loading.

All the strategies above use TDM for making reservations, and this makes it somewhat difficult to add new sources or delete old sources from the system. In addition, as m increases, γ increases and more of the bandwidth is used up by reservations. This suggests using the reservation slots in a contention resolution mode [JBH78]. It is difficult to analyze such a system, but it is clear that for large m and small λ , the delay could be reduced—we would use many fewer than m minislots in the reservation part of the frame.

Another somewhat simpler reservation idea is to allow the first packet of a message to make a reservation for the subsequent packets. In the context of Eq. (4.60), we view the first packet as the reservation packet and the entire message as the data packet. Thus Eq. (4.60) yields the throughput of this scheme if we take v as the inverse of the expected number of packets in a message. The appropriate length for a packet in this scheme is a trade-off between reservation inefficiency for long packets versus DLC inefficiency for short packets.

For satellite networks, these schemes are usually implemented with a frame structure (see Fig. 4.19), where a frame consists of a fixed number of packet slots [CRW73]. Enough packet slots are included in a frame so that the frame delay exceeds the round-trip delay.

When a source successfully transmits a packet in one of these slots, it can automatically reserve the corresponding slot in the next frame and each following frame until its message is completed. This can be done either by using a field in the packet header saying that another packet is to follow or, more simply and less efficiently, by automatically reserving that slot in each subsequent frame until that slot is first empty. After the end of the reservation period, the given slot in the frame is open for contention. Note that some care must be used in the contention algorithm. It does not suffice for all waiting sources simply to pounce on the first idle slot after a reservation period, since that would yield an unacceptably high collision probability.

Another variation on this scheme is to have each source “own” a particular slot within the frame [Bin75]. When a source is not using its own slot, other sources can capture it by contention, but when the source wants its own slot back, it simply transmits a packet in that slot, and if a collision occurs, the other sources are forbidden to use that

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	
15	idle	3	20	collision	2	Frame 1
15	7	3	idle	9	2	Frame 2
idle	7	3	collision	9	idle	Frame 3
18	7	3	collision	9	6	Frame 4
18	7	3	15	9	6	Frame 5

Figure 4.19 Reservation strategy with short packets and multiple packet messages. After a node captures a slot in a frame, it keeps that slot until finished.

slot on the next frame, letting the owner capture it. This variation is somewhat more fair than the previous scheme, but the delays are larger, both because of the less efficient contention resolution and because of the large number of slots in a frame if the number of users is large. Also, there is the usual problem of deleting and adding new sources to the system.

4.5.2 Local Area Networks: CSMA/CD and Ethernet

Section 4.5.1 treated satellite networks where round-trip propagation delay was an important consideration in each reservation scheme. For local area networks, at least with traditional technology, round-trip delay is a very small fraction of packet duration. Thus, instead of making a reservation for some slot far into the future, reservations can be made for the immediate future. Conceptually, this is not a big difference; it simply means that one must expect larger delays with satellite networks, since the feedback in any collision resolution strategy is always delayed by the round-trip propagation time. From a technological standpoint, however, the difference is more important since the physical properties of the media used for local area networks can simplify the implementation of reservation techniques.

Ethernet [MeB76] both illustrates this simplification and is a widely used technique for local area networks. A number of nodes are all connected onto a common cable so that when one node transmits a packet (and the others are silent), all the other nodes hear that packet. In addition, as in carrier sensing, a node can listen to the cable before transmitting (*i.e.*, conceptually, 0, 1, and idle can be distinguished on the bus). Finally, because of the physical properties of cable, it is possible for a node to listen to the cable while transmitting. Thus, if two nodes start to transmit almost simultaneously, they will shortly detect a collision in process and both cease transmitting. This technique is called CSMA/Collision Detection (CSMA/CD). On the other hand, if one node starts transmitting and no other node starts before the first node's signal has propagated throughout the cable, the first node is guaranteed to finish its packet without collision. Thus, we can view the first portion of a packet as making a reservation for the rest.

Slotted CSMA/CD. For analytic purposes, it is easiest to visualize Ethernet in terms of slots and minislots. The minislots are of duration β , which denotes the time required for a signal to propagate from one end of the cable to the other and to be detected. If the nodes are all synchronized into minislots of this duration, and if only one node transmits in a minislot, all the other nodes will detect the transmission and not use subsequent minislots until the entire packet is completed. If more than one node transmits in a minislot, each transmitting node will detect the condition by the end of the minislot and cease transmitting. Thus, the minislots are used in a contention mode, and when a successful transmission occurs in a minislot, it effectively reserves the channel for the completion of the packet.

CSMA/CD can be analyzed with a Markov chain in the same way as CSMA Aloha. We assume that each backlogged node transmits after each idle slot with probability q_r , and we assume at the outset that the number of nodes transmitting after an idle slot

is Poisson with parameter $g(n) = \lambda\beta + q_r n$. Consider state transitions at the ends of idle slots; thus, if no transmissions occur, the next idle slot ends after time β . If one transmission occurs, the next idle slot ends after $1 + \beta$. We can assume variable-length packets here, but to correspond precisely to the model for idle slots, the packet durations should be multiples of the idle slot durations; we assume as before that the expected packet duration is 1. Finally, if a collision occurs, the next idle slot ends after 2β ; in other words, nodes must hear an idle slot after the collision to know that it is safe to transmit.

The expected length of the interval between state transitions is then β , plus an additional 1 times the success probability, plus an additional β times the collision probability;

$$E\{\text{interval}\} = \beta + g(n)e^{-g(n)} + \beta[1 - (1 + g(n))e^{-g(n)}] \quad (4.64)$$

The expected number of arrivals between state transitions is λ times this interval, so the drift in state n is $\lambda E\{\text{interval}\} - P_{\text{succ}}$. The probability of success is simply $g(n)e^{-g(n)}$, so, as in Eq. (4.39), the drift in state n is negative if

$$\lambda < \frac{g(n)e^{-g(n)}}{\beta + g(n)e^{-g(n)} + \beta[1 - (1 + g(n))e^{-g(n)}]} \quad (4.65)$$

The right-hand side of Eq. (4.65) is interpreted as the departure rate in state n . This quantity is maximized over $g(n)$ at $g(n) = 0.77$ and the resulting value of the right-hand side is $1/(1 + 3.31\beta)$. Thus, if CSMA/CD is stabilized (this can be done, e.g., by the pseudo-Bayesian technique), the maximum λ at which the system is stable is

$$\lambda < \frac{1}{1 + 3.31\beta} \quad (4.66)$$

The expected queueing delay for CSMA/CD, assuming the slotted model above and ideal stabilization, is calculated in the same way as for CSMA (see Problem 4.24). The result, for small β and mean-square packet duration $\bar{X}^2$, is

$$W \approx \frac{\lambda \bar{X}^2 + \beta(4.62 + 2\lambda)}{2[1 - \lambda(1 + 3.31\beta)]} \quad (4.67)$$

The constant 3.31 in Eq. (4.66) is dependent on the detailed assumptions about the system. Different values are obtained by making different assumptions (see [Lam80], for example). If β is very small, as usual in Ethernet, this value is not very important. More to the point, however, is that the unslotted version of CSMA/CD makes considerably more sense than the slotted version, both because of the difficulty of synchronizing on short minislots and the advantages of capitalizing on shorter than maximum propagation delays when possible.

Unslotted CSMA/CD. Figure 4.20 illustrates why the analysis of an unslotted system is somewhat messy. Suppose that a node at one end of the cable starts to transmit and then, almost β time units later, a node at the other end starts. This second node ceases its transmission almost immediately upon hearing the first node, but nonetheless causes errors in the first packet and forces the first node to stop transmission another β time units later. Finally, another β time units go by before the other end of the line is

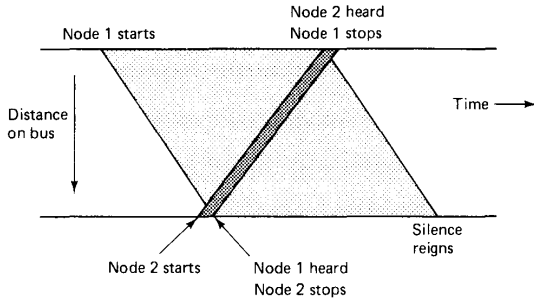


Figure 4.20 Collision detection. Node 2 starts to transmit almost β units after node 1; node 2 stops almost immediately, but node 1 continues for almost another β units before stopping.

quiet. Another complication is that nodes closer together on the cable detect collisions faster than those more spread apart. As a result, the maximum throughput achievable with Ethernet depends on the arrangement of nodes on the cable and is very complex to calculate exactly.

To get a conservative bound on maximum throughput, however, we can find bounds on all the relevant parameters from the end of one transmission (either successful or aborted) to the end of the next. Assume that each node initiates transmissions according to an independent Poisson process whenever it senses the channel idle, and assume that G is the overall Poisson intensity. All nodes sense the beginning of an idle period at most β after the end of a transmission, and the expected time to the beginning of the next transmission is at most an additional $1/G$. This next packet will collide with some later starting packet with probability at most $1 - e^{-\beta G}$ and the colliding packets will cease transmission after at most 2β . On the other hand, the packet will be successful with probability at least $e^{-\beta G}$ and will occupy 1 time unit. The departure rate S for a given G is the success probability divided by the expected time of a success or collision; so

$$S > \frac{e^{-\beta G}}{\beta + 1/G + 2\beta(1 - e^{-\beta G}) + e^{-\beta G}} \quad (4.68)$$

Optimizing the right-hand side of Eq. (4.68) over G , we find that the maximum occurs at $\beta G = (\sqrt{13} - 1)/6 = 0.43$; the corresponding maximum value is

$$S > \frac{1}{1 + 6.2\beta} \quad (4.69)$$

This analysis is very conservative, but if β is small, throughputs very close to 1 can be achieved and the difference between Eqs. (4.66) and (4.69) is not large. Note that maximum stable throughput approaches 1 with decreasing β as a constant times β for CSMA/CD, whereas the approach is as a constant times $\sqrt{\beta}$ for CSMA. The reason for this difference is that collisions are not very costly with CSMA/CD, and thus much higher attempt rates can be used. For the same reason, persistent CSMA (where new arrivals during a data slot are transmitted immediately at the end of the data slot) works reasonably for CSMA/CD but quite poorly for CSMA.

CSMA/CD (and CSMA) becomes increasingly inefficient with increasing bus length, with increasing data rate, and with decreasing data packet size. To see this, recall that

β is in units of the data packet duration. Thus if τ is propagation delay (plus detection time) in seconds, C is the raw data rate on the bus, and L is the average packet length, then $\beta = \tau C/L$. Neither CSMA nor CSMA/CD are reasonable system choices if β is more than a few tenths.

The IEEE 802 standards. The Institute of Electrical and Electronic Engineers (IEEE) has developed a set of standards, denoted 802, for LANs. The standards are divided into six parts, 802.1 to 802.6. The 802.1 standard deals with interfacing the LAN protocols to higher layers. 802.2 is a data link control standard very similar to HDLC as discussed in Chapter 2. Finally, 802.3 to 802.6 are medium access control (MAC) standards referring to CSMA/CD, token bus, token ring systems, and dual bus systems, respectively. The 802.3 standard is essentially the same as Ethernet, using unslotted persistent CSMA/CD with binary exponential backoff. We discuss the 802.5, 802.4, and 802.6 standards briefly in the next three subsections.

4.5.3 Local Area Networks: Token Rings

Token ring networks ([FaN69] and [FaL72]) constitute another popular approach to local area networks. In such networks, the nodes are arranged logically in a ring with each node transmitting to the next node around the ring (see Fig. 4.21). Normally, each node simply relays the received bit stream from the previous node on to the next. It does this with at least a one bit delay, allowing the node to read and regenerate the incoming binary digit before sending it on to the next node. Naturally, when a node transmits its own packet to the next node, it must discard what is being received. For the system to work correctly, we must ensure that what is being received and discarded is a packet that has already reached its destination. Conceptually, we visualize a “token” which exists in the net and which is passed from node to node. Whatever node has the token is allowed to transmit a packet, and when the packet is finished, the token is passed on to the next node. Nodes with nothing to send are obligated to pass the token on rather than saving it.

When we look at the properties that a token must have, we see that they are essentially the properties of the flags we studied for DLC. and that the same flag could be used as a token and to indicate the end of a packet. That is, whenever the node that is currently transmitting a packet finishes the transmission, it could place the token or flag, for example 01111110, at the end of the packet as usual. When the next node reads this token, it simply passes the token on if it has no packet to send, but if it does have a packet to send, it inverts the last token bit, turning the token into 01111111. This modified token, 01111111, is usually called a *busy token*, and the original, 01111110, is called a *free* (or *idle*) *token*. The node then follows this busy token with its own packet. Bit stuffing by inserting a 0 after 0111111 is used within the data packets to avoid having either type of token appear in the data. Thus, every node can split the received stream into packets by recognizing the free and busy tokens, and the free token constitutes the passing of permission to send from one node to the next. Token rings in practice generally have longer tokens with extra information in them; there is usually more than a single bit of delay in a node also.

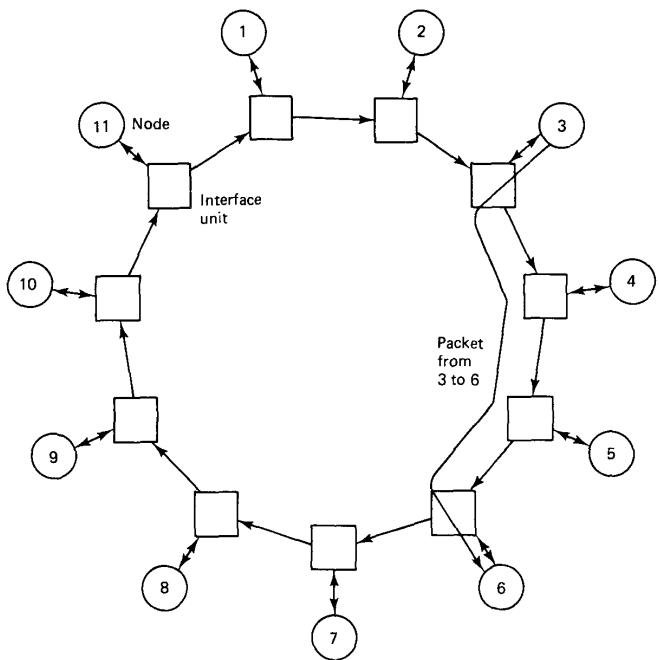


Figure 4.21 Ring network. Travel of data around the ring is unidirectional. Each node either relays the received bit stream to the next node with a one bit delay or transmits its own packet, discarding the incoming bit stream.

Let us look closely at how packets travel around the ring. Suppose, for example, that at time 0, node 1 receives a free token, inverts the last bit to form a busy token, and then starts to transmit a packet [see Fig. 4.22(a)]. Each subsequent node around the ring simply delays this bit stream by one bit per node and relays it on to the next node. The intended recipient of the packet both reads the packet into the node and relays it around the ring.

After a round-trip delay, the bit stream gets back to the originator, node 1 for our example. A round-trip delay (often called the ring latency) is defined as the propagation delay of the ring plus mk bits, where m is the number of nodes and k is the number of bit delays in a node. Assuming that the packet length is longer than the round-trip delay (in bits), the first part of the incoming packet is automatically removed by node 1, since node 1 is still transmitting a subsequent portion of the packet. When node 1 completes sending the packet, it appends a free token and then sends idle fill while the remainder of the just transmitted packet is returning to node 1 on the ring. After the last bit of the packet has returned, node 1 starts to relay what is coming in with a k bit

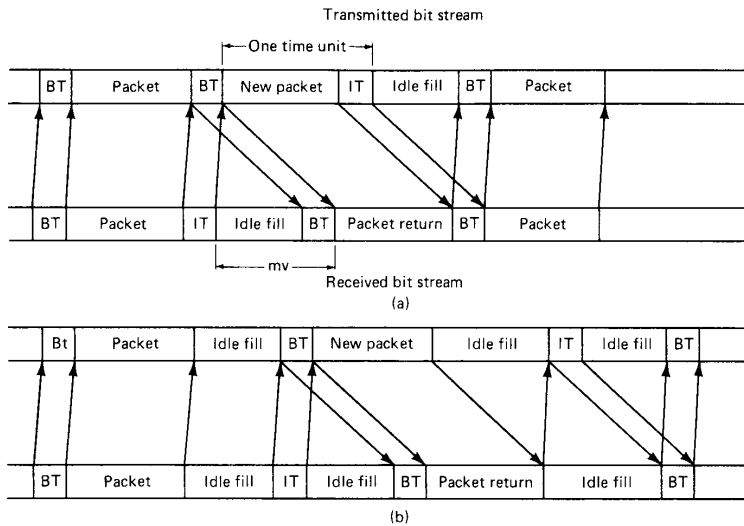


Figure 4.22 Transmitted and received bit stream at one ring interface unit of a token ring network. The interface unit transmits what is received with a one bit delay until seeing an idle token (IT). It converts the idle token into a busy token (BT) and then transmits its own packet. In part (a), this is followed by an idle token, whereas in part (b), the interface unit waits for the packet to return around the ring before transmitting the idle token. In each case, idle fill is transmitted after the idle token until the token (either busy or idle) returns around the ring, and then the unit reverts to relaying what is received with a one bit delay.

delay. If some other node has a packet to send, the first thing relayed through node 1 is a busy token followed by that packet; if no other node has a packet to send, the free token is relayed through node 1 and continues to circulate until some node has a packet to send.

Since all nodes follow this same strategy, when the idle token arrives at node 1 in the received bit stream, it must be followed by idle fill. This idle fill persists until the node sending that idle fill relays the busy token sent by node 1 (see Fig. 4.22). Thus, busy tokens are always followed by packets and idle tokens are always followed by enough idle fill to make up the round-trip delay on the ring.

Note that the round-trip delay must be at least as large as the token length; otherwise, a node, on completing a packet transmission and sending a free token, will discard the first part of the token as it returns to the node through the ring. One way to look at this is that the storage around the ring (*i.e.*, the propagation length in bits plus number of nodes) must be sufficient to store the token. Since the node transmitting a packet also reads it before removing it from the network, it is possible, either by checking bit by bit or by checking a CRC, for the transmitting node to verify that the packet was correctly

received. It is also possible for the receiving node to set a bit in a given position at the end of the packet if the CRC checks. Actually, neither of these techniques is foolproof, since an error could occur between the receiving node and its ring interface or an error could occur on the given bit set by the receiving node.

There are many variations in the detailed operation of a ring net. Each node could be restricted to sending a single packet each time it acquires the token, or a node could empty its queue of waiting packets before releasing the token to the next node. Also, priorities can be introduced fairly easily in a ring net by having a field for priorities in a fixed position after the free or busy token; any node could alter the bits in this field to indicate a high-priority packet. Other nodes, with lower-priority traffic, would relay free tokens rather than using them, so as to allow nodes with high-priority packets to transmit first. Obviously, when a node transmits its high-priority packets, it would then reduce the value in the priority field. Another approach to priorities is discussed in the section on FDDI.

Another variation is in the handling of ARQ. If a node transmits a free token immediately at the end of a packet transmission, that node will no longer have control of the channel if the packet is not delivered error-free. An alternative is for a node to send idle fill after completing a packet transmission until verifying whether or not the packet was correctly received [see Fig. 4.22(b)]. If correct reception occurs, a free token is transmitted; otherwise, a busy token is sent followed by a retransmission. As seen in the figure, each busy or idle token is preceded by a round-trip delay of idle fill. Idle tokens also have idle fill following them. This alternative makes it somewhat easier to see what is happening on a ring (since at most one packet is active at a time), but it lowers efficiency and increases delay, particularly for a large ring.

Yet another variation is in the physical layout of the ring. If the cable making up the ring is put into a star configuration, as shown in Fig. 4.23, a number of benefits accrue. First, a disadvantage of a ring net is that each node requires an active interface in which each bit is read and retransmitted. If an interface malfunctions, the entire ring fails. By physically connecting each interface at a common location, it is easier to find a failed interface and bypass it at the central site. If the interface is also located at the central site (which is not usually done), the propagation delay around the ring is materially reduced.

The most important variation in a ring network is the treatment of free and busy token failures. If a free token is destroyed by noise, or if multiple free tokens are created, or if a busy token is created and circulates indefinitely, the system fails. One obvious solution to this problem is to give a special node responsibility for recreating a lost free token or destroying spurious tokens; this is rather complex because of the possibility of the special node failing or leaving the network.

IEEE 802.5 token ring standard. The more common solution to token failure, which is used in the IEEE 802.5 standard, is for each node to recognize the loss of a token or existence of multiple tokens after a time-out. If a node has a packet to transmit after a time-out occurs, it simply transmits a busy token followed by the packet followed by a free token, simultaneously purging the ring of all other tokens. If suc-

cessful, the ring is again functional; if unsuccessful, due to two or more nodes trying to correct the situation at the same time, the colliding nodes try again after random delays.

The IEEE 802.5 standard also uses the star configuration and postpones releasing the token until the current packet is acknowledged. This standard uses a 24-bit token in place of the 8-bit token and contains elaborate procedures to recover from many possible malfunctions. The standard has been implemented in VLSI chips to implement a token ring running at either 4 or 16 megabits per second; the complexity of these chips, given the simplicity of the token ring concept, is truly astonishing.

Expected delay for token rings. Let us analyze the expected delay on a token ring. Assume first that each node, upon receiving a free token, empties its queue before passing the free token to the next node. Assume that there are m nodes, each with independent Poisson input streams of rate λ/m . Let v be the average propagation delay from one node to the next plus the relaying delay (usually one or a few bits) at a node. View the system conceptually from a central site, observing the free token passing

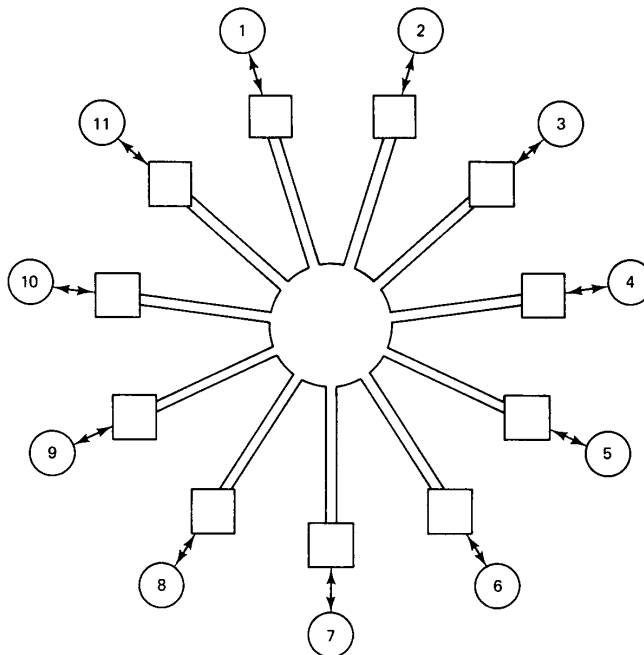


Figure 4.23 Ring network in a star configuration. Nodes can be bypassed or added from the central site.

around the ring. Let $\bar{X} = 1$ be the mean transmission time for a packet (including its busy token). Thus, $\rho = \lambda$. The mean queueing delay W is that of an exhaustive, multiuser system; the delay is given by Eq. (3.69) as

$$W = \frac{\lambda \bar{X}^2}{2(1 - \lambda)} + \frac{(m - \lambda)v}{2(1 - \lambda)} \quad (4.70)$$

Note that the first term is the usual $M/G/1$ queueing delay. The second term gives us the delay under no loading, $mv/2$. Observe that mv is the propagation delay around the entire ring, β , plus m times the one or few bits delay within each interface to a node. If β is small relative to a packet transmission time and if λ is relatively large, W is very close to the $M/G/1$ delay.

Next we look at the situation in which a node can transmit at most one packet with each free token. The system is then the partially gated, limited service system of Section 3.5.2. The delay is given by Eq. (3.76) as

$$W = \frac{\lambda \bar{X}^2 + (m + \lambda)v}{2(1 - \lambda - \lambda v)} \quad (4.71)$$

In this case, we note that the maximum stable throughput has been reduced somewhat to $1/(1 + v)$. In the analysis above, we have included the token length as part of the packet overhead and included in v only the propagation delay plus bit delay at a node interface. This is necessary to use the queueing results of Chapter 3. According to that analysis, v is the delay inserted at a node even when the node has nothing to send, and that delay does not include the entire token but only the one or few bits of transit delay within the node. Equation (4.71) can be modified for the case in which a transmitting node waits for the packet to return (as in the IEEE 802.5 standard) before passing on the free token. In essence, this adds one round-trip delay (*i.e.*, mv) to the transmission time of each packet, and the maximum throughput is reduced to $1/(1 + mv + v)$. As shown in Problem 4.27, the resulting expected delay is

$$W = \frac{\lambda(\bar{X}^2 + 2mv + m^2v^2) + [m + \lambda(1 + mv)]v}{2[1 - \lambda(1 + mv + v)]} \quad (4.72)$$

In comparing the token ring with CSMA/CD, note that if the propagation and detection delay is small relative to the packet transmission time, both systems have maximum throughputs very close to 1 packet per unit time. The token ring avoids stability problems, whereas CSMA/CD avoids the complexities of lost tokens and has slightly smaller delay under very light loads. Both are well-established technologies, and one chooses between them on the basis of cost, perceived reliability, and personal preference. If the propagation delay is large ($\beta > 1$), CSMA/CD loses its advantage over pure collision resolution and the IEEE 802.5 version of the token ring degrades similarly (since $mv > \beta$). On the other hand, a token ring in which nodes pass the free token on immediately after completing a packet transmission does not suffer this degradation [*i.e.*, the maximum throughput is $1/(1 + v)$ and v can be very much less than β if m is large].

FDDI. FDDI, which stands for “fiber distributed data interface,” is a 100-Mbps token ring using fiber optics as the transmission medium [Ros86]. Because of the high speed and relative insensitivity to physical size, FDDI can be used both as a backbone net for slower local area networks or as a metropolitan area network. There is an enhanced version of FDDI called FDDI-II that provides a mix of packet-switched and circuit-switched data. We discuss only FDDI itself here, however, since that is where the interesting new ideas are contained.

There is a rather strange character code used in FDDI in which all characters are five bits in length; 16 of the possible five bit characters represent four bits of data each, and the other characters are either special communication characters or forbidden. This character code was chosen to improve the dc balance on the physical fiber and to simplify synchronization. The data rate on the fiber is 125 Mbps in terms of these five bit characters, and thus 100 Mbps in terms of the actual data bits. The idle character is one of these special characters. It is repeated about 16 times at the beginning of each frame to allow for clock slippage between adjacent nodes.

All frames, including the token, are delimited by a start field and an end field, each of which consists of two of the special communication characters (see Fig. 4.24). The frame control field (FC) distinguishes the token from data frames and includes supplementary control information such as the length of the source address and destination address fields, which can be 16 or 48 data bits long. The CRC is the standard 32-bit CRC used in the IEEE 802 standards and as an option in HDLC; it checks on the frame control, addresses, and data. Finally, the frame status field (FS) at the end of a frame provides several flags for the destination to indicate that it has received the frame correctly. Note that when a node reads the token (i.e., what we have previously called the free token) on the ring, it can simply change the frame control field and then add the rest of its packet, thus maintaining a small delay within the node. Thus, what we called the busy token previously is just the beginning of the frame header.

There are two particularly interesting features about FDDI. The first is that nodes send the token immediately after sending data, thus corresponding to Fig. 4.22(a) rather than (b) (recall that IEEE 802.5 waits for the frame to return before releasing the token). As will be seen in the next section, this is what allows FDDI to maintain high throughput efficiency in the presence of much higher data rates and much larger distances than is possible for the 802.5 ring.

The second interesting feature about FDDI is its handling of priorities. In particular, high-priority traffic receives guaranteed throughput and guaranteed delay, thus making FDDI suitable for digitized voice, real-time control, and other applications requiring guaranteed service. The essence of the scheme is that each node times the interval between successive token arrivals at that node. High-priority traffic from the node can be sent whenever the token arrives, but low-priority traffic can be sent only if the intertoken

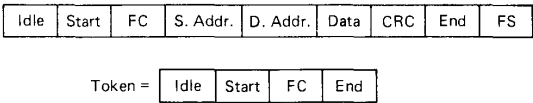


Figure 4.24 Frame format for FDDI.

interval is sufficiently small. What this means is that the low-priority traffic is decreased or even blocked when congestion starts to build up.

To provide guaranteed service to high-priority traffic, it should be clear that the network must impose constraints on that traffic. These constraints take the form of a limitation on how much high-priority traffic each node can send per received token. In particular, let the m nodes be numbered $0, 1, \dots, m-1$ in order around the ring. Let α_i be the allocated time for node i to send its high-priority traffic, including delay to reach the next node. Thus, if a token reaches node i at time t and node i sends its allocated amount of traffic, the token reaches node $i+1$ at time $t + \alpha_i$.

When the ring is initialized, there is a parameter τ called *target token rotation time*. This parameter is used by the nodes in deciding when to send low-priority traffic, and as we shall see, τ is an upper bound on the time-average intertoken arrival time. The allocated transmission times $\alpha_0, \dots, \alpha_{m-1}$ are allocated in such a way that $\alpha_0 + \alpha_1 + \dots + \alpha_{m-1} \leq \tau$. To describe and analyze the algorithm precisely, let $t_0, t_1, \dots, t_{m-1}$ be the times at which the token reaches nodes 0 to $m-1$ for some given cycle. Similarly, let $t_m, \dots, t_{2m-1}$ be the times at which the token reaches nodes 0 to $m-1$ in the next cycle, and so forth. Thus $t_i, i \geq 0$ is the time at which the token reaches node $(i \bmod m)$ in cycle $\lfloor i/m \rfloor$ (where we denote the given cycle as cycle 0 and where $\lfloor x \rfloor$ denotes the integer part of x). Finally, let t_{-m} to t_{-1} be the times at which the token reaches nodes 0 to $m-1$ in the cycle previous to the given one. At time $t_i, i \geq 0$ the node $(i \bmod m)$, having just received the token, measures $t_i - t_{i-m}$, which is the elapsed time since its previous receipt of the token. If $t_i - t_{i-m} < \tau$, the node is allowed to send low-priority traffic for $\tau - (t_i - t_{i-m})$ seconds. If $t_i - t_{i-m} \geq \tau$, no low-priority traffic is allowed. In both cases, the allocated high-priority traffic is allowed. The time at which the token reaches the next node is then upper bounded by

$$\begin{aligned} t_{i+1} &\leq t_{i-m} + \tau + \alpha_i, & \text{for } t_i - t_{i-m} < \tau; i \geq 0 \\ t_{i+1} &\leq t_i + \alpha_i, & \text{for } t_i - t_{i-m} \geq \tau; i \geq 0 \end{aligned}$$

where $\alpha_i = \alpha_{i \bmod m}$ is the allocated transmission plus propagation time for node $(i \bmod m)$. Combining these conditions, we have

$$t_{i+1} \leq \max(t_i, t_{i-m} + \tau) + \alpha_i; \quad i \geq 0 \quad (4.73)$$

Note that equality holds if node $(i \bmod m)$ sends as much data as allowed.

We first look at a simple special case of this inequality where $\alpha_i = 0$ for all i . Let τ' be the target token rotation time and t'_i be t_i for this special case. Thus

$$t'_{i+1} \leq \max(t'_i, t'_{i-m} + \tau'); \quad i \geq 0 \quad (4.74)$$

Since t'_i must be non-decreasing in i , we have $t'_{i-m} \leq t'_i$, for all $i \geq 0$. Substituting this in Eq. (4.74) yields $t'_{i+1} \leq t'_i + \tau'$. Similarly, for $1 \leq j \leq m+1$, we have

$$t'_{i+j} \leq \max(t'_{i+j-1}, t'_i + \tau') \leq t'_i + \tau'$$

where the last step follows by induction on j . For $j = m+1$, this equation reduces to

$$t'_{i+m+1} \leq t'_i + \tau' \quad \text{for all } i \geq 0 \quad (4.75)$$

Iterating this over multiples of $m + 1$, we obtain

$$t'_i \leq t'_{i \bmod (m+1)} + \lfloor i/(m+1) \rfloor \tau'; \quad \text{all } i \geq 0 \quad (4.76)$$

The appearance of $m + 1$ in Eq. (4.76) is somewhat surprising. To see what is happening, assume that $t'_i = 0$ for $0 \leq i \leq m$, and assume that all nodes are heavily loaded. Node 0 then transmits for time τ' starting at t'_m . This prevents all other nodes from transmitting for a complete cycle of the token. Since $t'_{2m} = \tau'$ and $t'_m = 0$, node 0 is also prevented from transmitting at t'_{2m} (note that the token travels around the ring in zero time since we are ignoring propagation delay here). Node 1 is then allowed to transmit for time τ' in this new cycle, and then no other node is allowed to transmit until node 2 transmits in the next cycle. This helps explain why events occur spaced $m + 1$ token passings apart.

Now consider the elapsed time for the token to make $m + 1$ cycles around the ring starting from some node j . From Eq. (4.76) we have

$$t'_{j+(m+1)m} - t'_j \leq m\tau' \quad (4.77)$$

Note that $t'_{j+(m+1)m} - t'_j$ is the sum of the $m + 1$ cycle rotation times measured by node j at times $t'_{j+m}, t'_{j+2m}, \dots, t'_{j+(m+1)m}$. Since each cycle rotation requires at most τ' seconds, we see that $(m + 1)\tau' - (t'_{j+(m+1)m} - t'_j)$ is equal to the total time offered to j at the above $m + 1$ token receipt times. Combining this with Eq. (4.77), we see that over the $m + 1$ token receipt times above, node j is offered an aggregate of at least τ' seconds for its own transmission; also, the corresponding $m + 1$ cycles occupy at most time $m\tau'$. It is also seen from this that the average token rotation time is at most $\lfloor m/(m + 1) \rfloor \tau'$ rather than the target time τ' . Using the same argument, it can be seen that over a sequence of any number n of cycles, the total amount of time allocated to node j is at least as great as that offered to any other node during its $n - 1$ token receptions within the given n cycles at node j . Thus each node receives a fair share of the resources of the ring. (This type of fairness is known as max-min fairness and is discussed in detail in Chapter 6.)

Although there is a certain elegance to this analysis, one should observe that in this special case of no high-priority traffic, the same kind of results could be obtained much more simply by allowing each node to transmit for τ' seconds on each receipt of the token. Fortunately, however, the results above make it quite easy to understand the general case of Eq. (4.73), in which node i is allocated α_i units of high-priority traffic. If we subtract $\sum_{j=0}^i \alpha_j$ from each side of Eq. (4.73) for $i \geq m$, we get

$$t_{i+1} - \sum_{j=0}^i \alpha_j \leq \max \left(t_i - \sum_{j=0}^{i-1} \alpha_j, \quad t_{i-m} - \sum_{j=0}^{i-m-1} \alpha_j + \tau - \sum_{j=i-m}^{i-1} \alpha_j \right) \quad (4.78)$$

Since $\alpha_j = \alpha_{j \bmod m}$, we see that $\sum_{j=i-m}^{i-1} \alpha_j$ is independent of i and is equal to the total allocated traffic, $T = \sum_{j=0}^{m-1} \alpha_j$. We now define

$$t'_i = t_i - \sum_{j=0}^{i-1} \alpha_j \text{ for all } i \geq 0; \quad \tau' = \tau - T \quad (4.79)$$

Substituting this in Eq. (4.78), we obtain Eq. (4.74) for $i \geq m$,

$$t'_{i+1} \leq \max(t'_i, t'_{i-m} + \tau')$$

To obtain initial conditions for this, assume that $t_0 = 0$ and that $t_i \leq 0$ for $i < 0$. Iterating Eq. (4.73) from $i = 0$ to $m - 1$, and using $t_i \leq t_{i+1}$, we get

$$0 \leq t_i \leq \tau + \sum_{j=0}^{i-1} \alpha_j; \quad 1 \leq i \leq m$$

From Eq. (4.79), then, we have $0 \leq t'_i \leq \tau$ for $1 \leq i \leq m$. Using τ as an upper bound to t'_i for $0 \leq i \leq m$, induction over i can be applied to Eq. (4.74) to yield

$$t'_i \leq \tau + \left\lfloor \frac{i}{m+1} \right\rfloor \tau' \\ t_i \leq \sum_{j=0}^{i-1} \alpha_j + (\tau + \tau) - T \left\lfloor \frac{i}{m+1} \right\rfloor \quad (4.80)$$

We can rewrite $\sum_{j=0}^{i-1} \alpha_j$ as $\sum_{j=0}^{(i-1) \bmod m} \alpha_j + \lfloor (i-1)/m \rfloor T$, so Eq. (4.80) becomes

$$t_i \leq \sum_{j=0}^{(i-1) \bmod m} \alpha_j + \tau \left(1 + \left\lfloor \frac{i}{m+1} \right\rfloor \right) + T \left(\left\lfloor \frac{i-1}{m} \right\rfloor - \left\lfloor \frac{i}{m+1} \right\rfloor \right) \quad (4.81)$$

We now focus on the time mk at which node 0 receives its k^{th} token

$$t_{mk} \leq \tau \left(1 + \left\lfloor \frac{mk}{m+1} \right\rfloor \right) + T \left(k - \left\lfloor \frac{mk}{m+1} \right\rfloor \right) \quad (4.82)$$

We see that $t_m - t_0 \leq \tau + T$ (which we already knew), and this bound can be met with equality if all the nodes are idle up to time t_0 and then all become busy. Since node 0 could be any node and t_0 the time of any token receipt at that node, this says in general that the intertoken interval at a node can be as large as $\tau + T$ but no larger. We also see that $\limsup_{k \rightarrow \infty} (t_{mk}/k) \leq \tau m/(m+1) + T/(m+1)$. Thus the time-average round-trip token time is at most τ , and is somewhat smaller than τ if $T < \tau$.

Finally, node 0 is allowed to send at least $k\tau - (t_{mk} - t_0)$ seconds of low-priority traffic at the k token receipt times ending with t_{mk} . From Eq. (4.82), the average such traffic per token receipt is at least $(\tau - T)/(m+1)$. Thus each node receives the same amount of guaranteed low-priority traffic and Eq. (4.82) indicates how much is guaranteed within any given number of token receipts.

There is a trade-off between throughput efficiency and delay for FDDI. Assuming that the high-priority traffic is stream-type traffic within the allocated rate, we have seen that the delay is bounded by $\tau + T$, which is more loosely bounded by 2τ . On the other hand, each time the token travels around the ring, there is a period $m\tau$, corresponding to

the propagation time plus delay within the nodes, during which no data are sent. Thus the throughput, as a fraction of the 100 Mbps, is at most $1 - mv/\tau$, and actually slightly smaller than this because the actual token rotation time is slightly faster than τ . As an example, if $\tau = 10$ msec and the ring circumference is 100 miles, then throughputs of about 94% are possible. Thus this type of system maintains high throughput and low guaranteed delay for large metropolitan area networks.

The maximum packet size for FDDI is set at 36,000 bits (because of physical synchronization requirements). Since the low-priority traffic is sometimes allowed in rather small pieces, however, a user with smaller packets will get better service than one with large packets.

Slotted rings and register insertion rings. Assuming that traffic on a ring is uniformly distributed between different source-destination pairs, a packet need be transmitted on only half of a ring's links on the average. Since in the token ring, a packet travels on every one of the links, we see that half the system's transmission capability is potentially wasted. It is therefore conceivable that a different control strategy could achieve twice the throughput.

Slotted rings and register insertion rings allow this higher potential throughput, at least in principle. A slotted ring is best viewed as a conveyor belt of packet slots; the ring is extended by shift registers within the nodes to provide the desired number of slots. When a node has a packet to send, it looks for an empty slot in the conveyor belt and places the packet in that slot, marking the slot as full. When the destination node sees the packet, it removes it and marks the slot as empty again.

One disadvantage of a slotted ring is that all packets must have equal length (as contrasted with token rings and CSMA/CD). Another disadvantage is significant delay (due to the conveyor belt length) even under light load. This can be compensated for by making the packets very short, but the added DLC overhead caused by short packets loses much of the potential throughput gain. Finally, to accomplish ARQ, it is common to leave packets in their slots until they return to the sending node; this automatically throws away the potential doubling of throughput.

The register insertion ring provides true store-and-forward buffering of ring traffic within the nodes. Each node has a buffer for transit traffic and a buffer for new arrivals (see Fig. 4.25). When a new arrival is being transmitted, incoming ring traffic that must be forwarded is saved in the transit buffer. When the transit traffic is being transmitted, the buffer gradually empties as either idle fill or packets destined for the given node arrive at the ring input. New arrivals are inhibited from transmission whenever the transit buffer does not have enough space to store the input from the ring while the new packet is being inserted on the ring.

The register insertion ring is capable of higher throughputs and has only a slightly greater delay for light loading than the token ring. Its greatest disadvantage is that it loses the fair allocation and guaranteed access provided by the token ring's round-robin packet service. The token ring is much more popular for applications, probably because maximum throughput is not the dominant consideration for most local area networks.

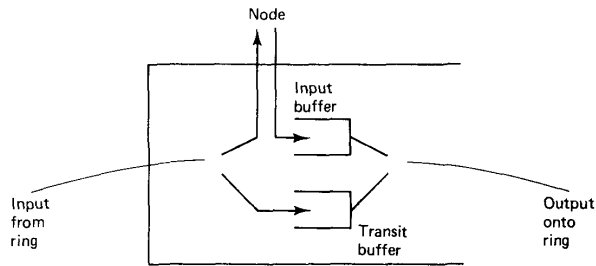


Figure 4.25 Interface to a register insertion ring. The output onto the ring comes from either the input buffer or the transit buffer. The input from the ring goes to the transit buffer or directly to the node, if addressed there.

4.5.4 Local Area Networks: Token Buses and Polling

The general idea of the token ring is used in a wide variety of communication situations. The idea is that there are m nodes with ordered identities and the nodes are offered service one at a time in round-robin order. The differences between these systems lie in the question of how one node knows when the previous node has finished service (or refused the offer of service). In other words, what mechanism performs the role of the token, and what is the delay, v , in passing this virtual token from one node to the next?

As discussed briefly in Section 4.1.2, polling is a common example of such a system. The polls sent by the central node to each of the secondary nodes act as tokens. The token-passing delay in a polling system is quite large, involving first a communication from a polled node back to the central node, and then a new polling request from the central node to the next secondary node.

Hub polling is a way of avoiding the double delays above. The central station polls (*i.e.*, passes the token to) the first secondary node; each secondary node, after using the channel, passes the token on to the next secondary node. If the nodes are ordered in terms of distance on the multidrop telephone line or bus, then, of course, token passing delay is reduced even further.

A token bus can be implemented on the same type of physical bus as a CSMA/CD system. The nodes are ordered in a round-robin fashion, and when a node finishes sending its packet or packets, it sends a token to the next node, giving it permission to send next. A node that has nothing to send simply sends the token to the next node. Conceptually, it can be seen that a token bus is essentially the same as a hub polling system. Polling is the more common terminology with a central node, and token bus is more common for a fully distributed system. Equations (4.70) and (4.71) give the delay associated with these systems under the assumptions of sending all queued packets per poll and one packet per poll, respectively. Both equations assume equal average traffic for all nodes. The parameter v in these equations can be interpreted in general as the delay, in an empty system, from token arrival at one node to the next, averaged over the nodes.

Recall that for the token ring, v included only propagation delay from node to node plus one or a few bits delay through the node; the initiation of token transmission from a node starts before the token is completely received from the previous node. Here, the token length (which can be considerable) is included in v , since one token must be fully received and decoded before the next node starts. Thus, the expected delay in the limit of zero load is inherently larger for the token bus than the token ring.

The performance of token buses and polling systems depends critically on the parameter v [i.e., from Eq. (4.71), the maximum throughput is $1/(1+v)$ and the expected delay in the limit $\lambda \rightarrow 0$ is $mv/2$]. Assuming for the moment that the nodes are numbered independently of position on the bus, it is reasonable to take the average propagation delay as $\beta/3$ (see Problem 4.29), where $\beta = \tau C/L$ is the normalized propagation time from one end of the bus to the other (i.e., the time measured as a fraction of the average packet length). Here τ is propagation time in seconds, C is the channel bit rate, and L is the expected packet length. Similarly, the normalized token transmission time is k/L , where k is the length of the token. Thus,

$$v = \frac{\tau C}{3L} + \frac{k}{L} + \delta \quad (4.83)$$

where δ is the normalized time for the receiver to detect a token. In our previous discussions, we included δ as part of the propagation delay β , but here we count this time separately.

The quantity τC is the number of bits that can travel along the bus at one time. If $\tau C/3$ is small relative to k , improving the performance of the algorithm depends on decreasing the length of the token. Conversely, if $\tau C/3$ is large, k is relatively unimportant and improvements depend on reducing the effects of propagation delay. One obvious way to reduce the effect of propagation delay is to number the nodes sequentially from one end of the bus to the other. If this is done, the sum of the propagation delays over all nodes is 2β ; that is, there is a cumulative propagation delay of β moving down the bus to poll all nodes, and then another β to return. Thus, the average value of v is

$$v = \frac{2\tau C}{mL} + \frac{k}{L} + \delta \quad (4.84)$$

This is a major reduction in propagation delay, but it makes it somewhat more difficult to add new nodes to the bus. Note that the propagation delays are much smaller than the average as reservation opportunities move down the bus, but then there is a long reservation interval of duration β to return from the end of the bus to the beginning. We recall from Section 3.5.2 that Eqs. (4.70) and (4.71) are valid using the average value of v .

IEEE 802.4 token bus standard. The IEEE 802.4 standard corresponds essentially to the system we have just described. We will briefly describe some of its features. To allow new nodes to enter the round-robin token structure, each node already in the structure periodically sends a special control packet inviting waiting nodes to join. All waiting nodes respond, and if more than one, a splitting algorithm is used to select one. The new node enters the round robin after the inviting node, and the new

node subsequently addresses the token to the node formerly following the inviting node. An old node can drop out of the round robin simply by sending a control packet to its predecessor directing the predecessor to send subsequent tokens to the successor of the node that is dropping out. Finally, failure recovery is essentially accomplished by all nodes dropping out, then one starting by contention, and finally the starting node adding new nodes by the procedure above.

Implicit tokens: CSMA/CA. Consider how to reduce the token size. One common approach is to replace the token with an implicit token represented by the channel becoming idle. Two of the better known acronyms for this are BRAM [CFL79] and MSAP [KIS80]. In these schemes, when a node completes a packet transmission, it simply goes idle. The next node in sequence, upon detecting the idle channel, starts transmission if it has a packet or otherwise remains idle. Successive nodes in the sequence wait for successively longer times, after hearing an idle, before starting transmission, thus giving each of the earlier stations an opportunity to transmit if it has packets. These schemes are often called CSMA/Collision Avoidance (CSMA/CA) schemes.

To see how long a node must hear an idle channel before starting to transmit, consider the worst case, in which the node that finishes a packet is at one end of the bus, the next node is at the opposite end, and the second node is at the first end again. Then the second node will detect the idle channel almost immediately at the end of the transmission, but the first node will not detect the event until $\beta + \delta$ units later, and the second node will not know whether the first node is going to transmit until an additional delay of $\beta + \delta$. Thus, the second node must wait for $2(\beta + \delta)$ before starting to transmit. By the same argument, we see that each successive node must wait an additional increment of $(\beta + \delta)$. Thus, this scheme replaces an explicit token of duration k/L with an implicit token of duration $(\beta + \delta) = (\tau C/L + \delta)$. The scheme is promising, therefore, in situations where τC and δ are small.

If the nodes are ordered on the bus, and the delays are known and built into the algorithm, the durations of the implicit tokens are greatly reduced, as in Eq. (4.70), but the complexity is greatly increased. There are many variations on this scheme dealing with the problems of maintaining synchronization after long idle periods and recovering from errors. (See [FiT84] for an excellent critical summary.)

4.5.5 High-Speed Local Area Networks

Increasing requirements for data communications, as well as the availability of high-data-rate communication media, such as optical fiber, coaxial cable, and CATV systems, motivate the use of higher- and higher-speed local area networks. For our purposes, we define a high-speed local area network as one in which β exceeds 1. Recall that β is the ratio of propagation delay to average packet transmission time, so $\beta > 1$ means that a transmitter will have finished sending a packet before a distant receiver starts to hear it. Since $\beta = \tau C/L$, we see that increasing propagation delay τ , increasing data rate C , and decreasing expected packet length L all contribute to making the net high speed. There has also been great interest in extending local area techniques to wider area coverages,

such as metropolitan areas; the larger areas here can cause a network to become high speed even at modest data rates.

We have already seen the effect of β on the performance of local area networks. CSMA/CD degrades rapidly with increasing β and becomes pointless for $\beta > 1$. Similarly, for token rings such as IEEE 802.5 that release the token only after an ack is received, we see from Eq. (4.72) that the maximum throughput degrades as $1/(1 + \beta)$ as β gets large. [Note that mv in Eq. (4.72) is essentially equal to β when β is large.] For token rings such as FDDI that release the token after completing transmission, we see from Eq. (4.71) that the maximum throughput degrades as $1/(1 + \beta/m)$. Thus in essence it is the propagation delay between successive nodes that is important rather than the propagation delay around the ring.

As a practical matter, Eq. (4.71) must be used with considerable caution. It assumes that all nodes on the ring have Poisson packet arrivals at the same rate. For local area networks, however, there are often only a very small number of nodes actively using the net in any given interval of a few minutes. In such situations, m should be taken as the number of active nodes rather than the total number of nodes, thus giving rise to much greater throughput degradation. Fortunately, FDDI, with its token rotation timer, avoids this problem by allowing each node to transmit more data per token when the number of active nodes is small. FDDI was designed as a high-speed local area network, but it was described in the last section since it is a high-performance type of token ring.

The token bus, with implicit or explicit tokens, also degrades as $1/(1 + \beta/3)$ if the nodes are numbered arbitrarily. If the nodes are numbered sequentially with respect to physical bus location, then, as seen by Eq. (4.84), the maximum throughput is degraded substantially only when β is a substantial fraction of the number of nodes m . If an ack is awaited after each packet, however, this throughput advantage is lost.

One of the most attractive possibilities for high-speed local networks lies in the use of buses that propagate signals in only one direction. Optical fibers have this property naturally, and cable technology is well developed for unidirectional transmission.

A unidirectional bus is particularly easy to use if the data frames are restricted to be all of the same size. In this case, the bus can be considered to be slotted. Empty slots are generated at the head of the bus and there is a "busy bit" at the front of the slot that has the value 0 if the slot is empty and 1 if the slot is full. Each node in turn, then, can read this busy bit; if the busy bit is 0 and the node has a frame to send, the node changes the busy bit to 1 and sends the frame. Nodes farther down the bus receive the busy bit as 1 and read the frame but do not modify it. Note that the fixed-length slot is necessary here, since otherwise a node could still be transmitting a longer frame when a frame arrives from upstream on the bus.

The slotted unidirectional bus structure above is almost trivial from a logical point of view, but has some very powerful characteristics. First it has the same advantage as Ethernet in that there is almost no delay under light loading. Here, of course, there is a half-slot delay on the average, but if the slots are short, this is negligible. Second, there is the advantage of ideal efficiency. As long as nodes are backlogged, every slot will be utilized, and aside from frame headers and trailers, the maximum data rate is equal to the bus rate.

There are also a few disadvantages to this trivial scheme. The first is that data can flow in only one direction, and the second is an inherent unfairness in that nodes close to the head of the bus get priority over those farther down the bus. The following dual bus architecture solves both these problems.

Distributed queue dual bus (IEEE 802.6). The distributed queue dual bus architecture (DQDB) (also known by its older name QPSX) is being standardized as the IEEE 802.6 metropolitan area network ([NBH88] and [HCM90]). Each node is connected to two unidirectional 150 Mbps buses (optical fibers) going in opposite directions (Fig. 4.26). Thus a node uses the right-moving bus to send frames to nodes on its right and uses the left-moving bus for nodes on its left (thus solving one of the problems with the trivial single bus structure above). The frames have a fixed length of 53 bytes and fit into slots generated at the head ends of the buses. The frame length was chosen for compatibility with ATM (see Section 2.10).

The dual bus architecture attempts to achieve fairness by adding a request bit to the overhead. If a node has a frame to send on the right bus, it sets the request bit in a frame on the left bus. Similarly, if it has a frame for the left bus, it sets the request bit in a frame on the right bus. In what follows, we focus on data traffic in a single direction, denoted downstream, and refer to the request bits as moving upstream (relative to the data direction of interest). DQDB allows for a set of different priorities for different classes of traffic, and each slot contains a one bit request field for each priority. For simplicity of exposition, we ignore these priorities, thus assuming that each slot contains only a single request bit.

Each request bit seen by a node on the upstream bus serves as an indication of a waiting frame at a more downstream node. These request bits are not removed as they pass upstream, and thus all upstream nodes see each such request bit. Each node views its own frames, plus the downstream frames indicated by request bits, as forming a virtual queue. This virtual queue is served in first-come first-serve order. This means that if a frame from the given node is at the front of the queue, that frame replaces the first idle slot to arrive on the downstream bus. Alternatively, suppose that a request from downstream is at the front of the queue. In this case, when the next idle slot arrives in the downstream direction, the request is removed but the slot is left idle, thus allowing a downstream node to transmit a frame.

To prevent any node from hogging the bus, each node is allowed to enter only one frame at a time on its virtual queue. Other frames at that node are forced to wait in a supplementary queue. As soon as the frame in the virtual queue is transmitted, a frame from the supplementary queue (or a new frame) can be entered into the virtual queue.

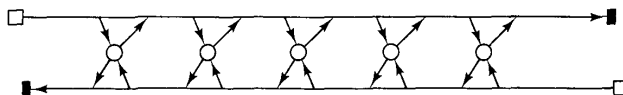


Figure 4.26 Bus structure for IEEE 802.6 dual bus.

A request bit is sent upstream at each instant when a frame enters the virtual queue. Since the virtual queue consists of at most one actual frame plus an arbitrary number of requests before and after that frame, the virtual queue can be implemented simply as two counters giving the number of requests before and behind the frame, respectively.

The description above of sending request bits upstream was oversimplified. The problem is that each slot contains only one request bit (ignoring priorities), and thus if some downstream node has already set the request bit in a slot, a node farther upstream will have to wait to send its own request. If there is a long string of requests coming from downstream nodes and a long string of idle slots coming from upstream nodes, it is possible that the waiting frame will be sent downstream before the request bit is sent upstream. In this case, the request bit continues to be queued waiting to be sent upstream. If a new frame enters the virtual queue waiting to be sent downstream, a new request bit is generated and queued behind the old request bit, and such a queue can continue to grow.

To explain the reason for a queue of request bits at a node, note that the request bits do not indicate which node has a frame to send. Thus we could visualize a node as sending its own request bit and queueing request bits from downstream; this would correspond to the position of the frame in the virtual queue. Another rationale is that there is a certain cleanliness in ensuring that one and only one request bit is sent upstream for each frame that is transmitted. This makes it possible for each node to determine the loading on the bus by summing the busy bits traveling downstream (the number of frames sent by upstream nodes) to the request bits traveling upstream (the number of frames from downstream nodes for which request bits have been received).

We now give two examples of the operation of this system which show that the system as explained is quite unfair in terms of giving much higher rates to some nodes than others under certain conditions. We then explain a modification (see [HCM90]), which is now part of the proposed standard. These examples demonstrate the peculiar behavior that results from large propagation delays on the bus. Since DQDB is designed as a metropolitan area network with a bit rate of 150 Mbps, large propagation delays relative to the slot time are expected. For example a 30 km bus has about 50 slots in simultaneous flight.

As the first example, suppose that initially only one node is sending data, and then another node farther downstream becomes active. Initially, the upstream node is filling all the slots with data, which it is allowed to do since it sees no request bits on the bus in the upstream direction. Figure 4.27 illustrates the situation shortly after the downstream node becomes active and sends a request bit which is propagating on the

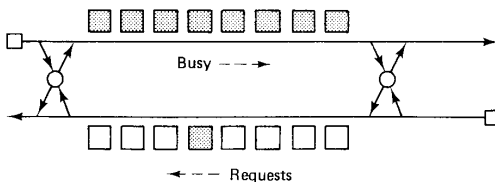


Figure 4.27 The upstream node on the left has a large file of data and is filling each downstream slot. The downstream node then receives a large file to send. The first frame is in the virtual queue and the request bit is traveling upstream.

upstream bus. The downstream node cannot send its first frame until an idle slot appears on the downstream bus, and cannot send another request bit upstream until the first frame is sent.

When the request bit reaches the busy upstream node, that node allows a slot to remain idle, and that slot propagates along the downstream bus (see Fig. 4.28). When the idle slot reaches the busy downstream node, a frame is transmitted, a new frame enters the virtual queue, and a new request bit starts to propagate upstream. It is evident that the downstream node sends only one frame out of each round-trip propagation delay. Thus the downstream node transmits one slot out of each $2n$, where n is the number of slots on the bus between the upstream and downstream node. With a separation of 30 km between the nodes, the downstream node receives less than one slot out of a hundred.

The next example is somewhat more complex and shows that a downstream node is also capable of hogging the bus if it starts sending a long file before any upstream node has anything to send. Figure 4.29 shows the situation before the upstream node becomes busy. The upstream bus is full of slots containing request bits.

After the upstream node becomes active, it puts a frame in its virtual queue. Assuming that this queue is initially empty, the frame is transmitted in the next slot and the arriving request bit is queued. When the next frame enters the virtual queue, it must wait for the queued request before being transmitted, and during this time two new requests join the virtual queue. Figure 4.30 illustrates the situation before the first busy slot from the busy upstream node arrives at the busy downstream node.

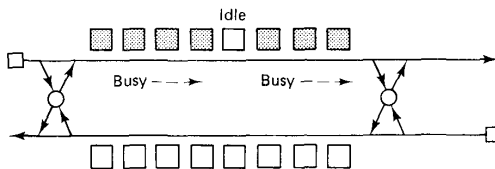


Figure 4.28 Continuation of Fig. 4.27 in which the upstream node allows an idle slot to propagate downstream after receiving a request from the downstream node.

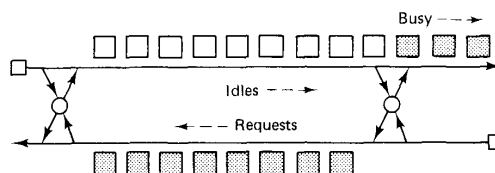


Figure 4.29 Illustration of situation in which a downstream node is sending a frame in every slot just before an upstream node becomes active. Note that the upstream bus is full of slots carrying request bits.

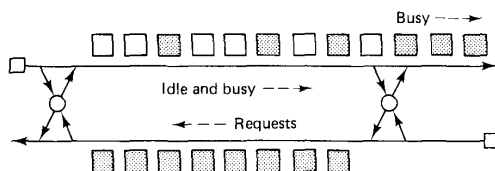


Figure 4.30 Continuation of Fig. 4.29 showing the scenario as the first busy slot arrives at the downstream node. Note that the first busy slot between the two nodes is followed by one idle slot, the second by two idle slots, and so forth.

Each time a busy slot from the busy upstream node passes the busy downstream node, the downstream node cannot enter a new frame into its virtual queue and thus cannot place a request bit in the corresponding upstream slot. Thus the sequence of requests and no requests on the upstream bus can be viewed as an extension of the idle and busy slots, respectively, on the downstream bus, circling around at the busy downstream node (see Fig. 4.31). It turns out that in steady state, the downstream node obtains approximately $\sqrt{2n}$ slots for each slot obtained by the upstream node (see [HCM90]).

There are several observations that can be made from these examples. First, in the case of two users with large files, the first user to start transmission receives an inordinate share of the slots, and this unfairness will persist for arbitrarily long files. Second, in both examples, each request sent upstream by the downstream node eventually returns downstream in the form of an idle slot, and each such idle slot gives rise to a further request going upstream. The total number p of these requests and idle slots between the two nodes (including the requests queued at the upstream node) remains fixed for the duration of the file transfers. We call p the number of reservations captured by the downstream node. In the first example above, $p = 1$, and in the second, $p = 2n$. For fair operation, it would be desirable for p to be approximately n .

Note that something very peculiar has happened; only one frame is allowed into the virtual queue at a time (thus making it look like the system is providing some sort of round-robin service to the nodes), but in fact a large number p of reservations can be pipelined through the upstream nodes for service at the given node. In fact, because of the large propagation delay and the prevalence of a small number of nodes sending large files, it is essential for p to be larger than 1.

Consider now what happens in the second example if the downstream node completes its file transfer. There will be $2n$ idle slots that propagate past the downstream node and never get used, even though the upstream node still has data. Even if fairness were achieved between the nodes, there would still be n wasted slots when the downstream node becomes idle. What can be said about throughput, however, is that if the farthest downstream node that has data to send is always busy, no slot is ever wasted (*i.e.*, that node will fill any idle slot that it sees).

Fortunately, the fairness problem described above can be cured. One cure is to change the rule followed by each node when it has a frame at the front of the virtual queue and sees an idle slot. As described above, the node always inserts the frame into the idle slot under these circumstances. In the modified system, the node only inserts its frame into the idle slot a given fraction $1 - f$ of the time. In terms of example 1,

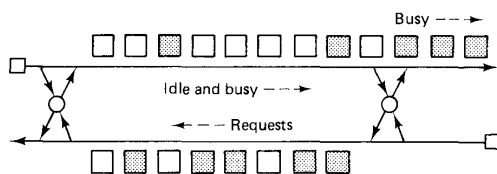


Figure 4.31 Continuation of Fig. 4.29 showing the scenario as the first nonrequest slot arrives at the upstream node. Note that successive nonrequest (busy) slots are followed by increasing numbers of request (idle) slots.

this modification forces the upstream node to insert occasional extra idle slots, which allows the downstream node to increase its number of reservations gradually. In terms of example 2, the downstream node, when inserting extra idles, is also inhibited from sending requests upstream; this reduces its number of reservations. Thus, as shown in [HCM90], the number of reservations approaches the fair value. This modification is now a part of the 802.6 draft standard.

The problem with the modification above is that it wastes part of the throughput of the system. For the case of two nodes with large files, for example, the fraction of the throughput that is wasted can be shown to be $f/(f + 2)$, where f is the fraction of time that a node leaves an idle slot empty when it has a frame at the front of the virtual queue. One can choose f arbitrarily small, but this slows the rate at which the system converges to fair operation (see [HCM90]).

Both DQDB and FDDI have considerable promise as high-speed networks. FDDI has the advantage of simple guarantees on rate and delay, whereas DQDB has the advantage of compatibility with ATM. Neither use optical fiber in an ideal way, however; since each node must read all the data, the speed of these networks is limited by the electronic processing at the nodes.

Expressnet. There are other approaches to high-speed local area networks using unidirectional buses but allowing variable-length frames. The general idea in these approaches is to combine the notion of an implicit token (*i.e.*, silence on the bus) with collision detection. To see how this is done, consider Fig. 4.32, and for now ignore the problem of how packets on the bus get to their destinations. Assume that a node on the left end of the bus has just finished transmission. Each subsequent node that has a packet to send starts to transmit carrier as soon as it detects the channel to be idle.

If δ is the time required for a node to detect idle and to start transmitting carrier, we see from the figure that if two nodes have packets to send at the completion

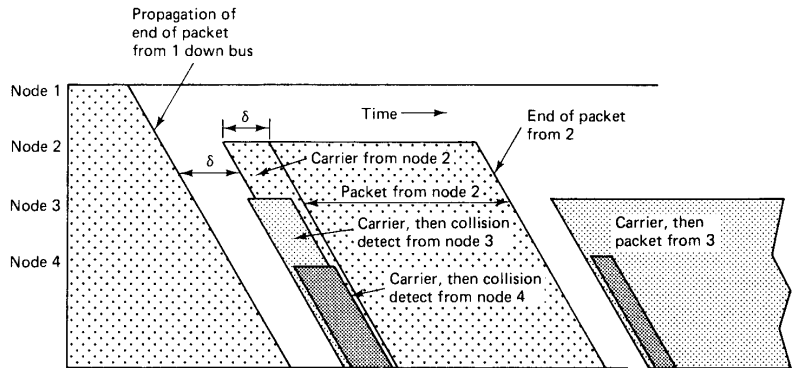


Figure 4.32 Implicit tokens on a unidirectional bus. Each node with traffic transmits carrier on hearing silence, then defers to upstream nodes on hearing carrier.

of a given packet, the second node will start to send carrier at a delay δ after the end of the previous packet passes that node. The carrier sent by the first node will reach the second node at essentially the same time as the second node starts to send carrier. Since the second node can detect whether or not the first node is sending carrier after another delay of δ , the second node (and all further nodes) can terminate the transmission of carrier after this delay δ . Thus, if each node starts to transmit its packet after sending carrier for δ , the node is assured that no earlier node can be transmitting, and that all subsequent nodes will cease transmitting before any of the packet information arrives.

We see that the implicit token is simply the termination of transmission on the line. The time v for this token to travel from one node to the next is simply the propagation time from one node to the next. We view the length of the packet as containing δ units of time for detection at the end and δ units of time for carrier at the beginning. In a sense, this is the ideal system with TDM reservations. The time for making a reservation consists only of propagation delay (essentially the implicit token has zero length), and the packets follow immediately after the reservations.

The discussion above ignored the questions of how to receive the transmitted packets and how to return to the beginning of the bus and restart the process after all nodes have their turns. We first illustrate how these problems are solved in the Expressnet system [TBF83], and then briefly describe several alternative approaches.

Expressnet uses a unidirectional bus with two folds as shown in Fig. 4.33. Packets are transmitted on the first portion of the bus, as explained above, and nodes read the packets from the third portion of the bus. When a silence interval longer than δ is detected on the third portion of the bus, it is clear that all nodes have had a chance to transmit and it is time to start a new cycle. Thus, all nodes with packets to send again start to send carrier. Because of the double fold in the bus, however, nodes on the left end of the bus hear the idle signal before those on the right, and the carrier signals will again overlap for the next cycle as in Fig. 4.32.

The system is still not quite complete, since there is a problem if no node has a packet to send in a cycle. This is solved by having all nodes, busy or not, transmit carrier for a duration δ when they hear the onset of a silent interval longer than δ on the third portion of the bus. All nodes with packets must then extend their burst of carrier to a duration 2δ . This extra period of δ is necessary to detect silence on the first portion of the bus if none of the earlier nodes have packets to send. These extra bursts of carrier from all nodes keep the system synchronized during idle periods.

Since the propagation delay from when a node starts to transmit until the signal arrives at the corresponding point on the third portion of the bus is 2β , and since the

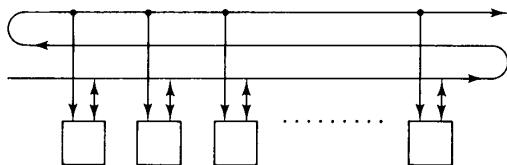


Figure 4.33 Bus structure for Expressnet. Each node transmits on the lower portion of the bus and listens on both lower and upper portions. Contention is resolved on the lower portion with the leftmost node taking priority. Reception and detection of the end of a cycle takes place from the upper portion.

transmission of the burst of carrier and its detection take time 2δ , we see that the average reservation interval is

$$v = 2 \frac{(\beta + \delta)}{m} \quad (4.85)$$

Equations (4.70) and (4.71) again give the expected queueing delay for multiple packets per reservation and a single packet per reservation, respectively.

There are many possible variations on how to use this unidirectional bus. One could replace the double-folded structure with a single fold, reading packets on the return trip on the bus, and using a special node at the left end of the bus to start new cycles. One could also use two buses, one in each direction. A node could then send traffic to nodes on its right on the right-going bus and traffic to nodes on its left on the left-going bus as in DQDB. Special nodes on each end of the two buses would then interchange information for starting new cycles in each direction. Other possibilities include using a separate control wire, or a separate frequency band, to take the place of the implicit reservation tokens. [Fit84] contrasts a large set of these possible approaches.

Homenets. The previous approaches to higher-speed local networks were based on the use of a unidirectional bus. CATV systems, on the other hand, typically have a tree structure with transmissions from the root of the tree being broadcast throughout the tree. To use such networks for data, there is usually a separate frequency band for communication from the leaves of the tree in toward the root. When packets are sent inward toward the root from different leaves, there is the usual problem of collisions, but the use of implicit reservation tokens based on bus position no longer works.

Homenets [MaN85] provide an interesting approach to coping with the problem of large β on such a net. The idea is to break up the CATV net into subnets called homenets. Each homenet forms its own subtree in the overall network tree, as shown in Fig. 4.34, and each homenet has its own frequency bands, one for propagation inward toward the root and the other for propagation outward. This strategy cures the problem of large β in two ways. First, the propagation delay within a homenet is greatly reduced from its value in the entire net, and second, by using a restricted bandwidth, the data rate within the homenet is reduced. With this twofold decrease in β , it becomes reasonable to use CSMA/CD within the homenet.

This leaves us with two unanswered questions. First, since the nodes transmit their packets inward toward the root, how do other outlying nodes in the same homenet hear collisions? Second, how are the packets received by nodes outside the homenet? The solution to the first question is for the node at the root of a given homenet to receive the signal on the homenet's incoming frequency band and both to forward it toward the root of the entire net and convert it to the homenet's outgoing frequency band and broadcast it back out to the homenet. This allows the nodes to detect collisions within the propagation delay of the homenet. The solution to the second question is for all of the incoming frequency bands to be forwarded to the root of the entire net, which then converts these bands to the outgoing frequency bands and rebroadcasts all the signals back through the entire network. The root node for each homenet then filters out the signal from the overall root on its own outgoing frequency (since that signal is just a

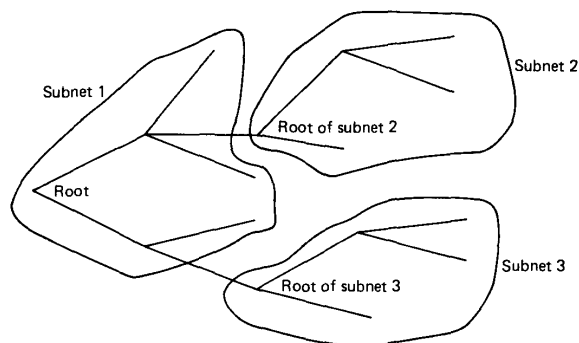


Figure 4.34 A CATV network divided into subnets in the Homenet strategy. Each subnet has one incoming and one outgoing frequency band and uses CSMA/CD to resolve collisions with a subnet.

delayed replica of what it has already transmitted outward) and forwards the other bands over its own homenet.

One additional advantage of this strategy is that the nodes need only transmit and receive within these restricted frequency bands. When a session is set up between two nodes, the receiver at each node must know of the outgoing frequency band for the sender's homenet, and then it simply receives the desired packets out of the signal in that band.

4.5.6 Generalized Polling and Splitting Algorithms

Section 4.5.5 treated multiaccess systems in the presence of large propagation delays. Here, we look at the opposite case in which $\beta \ll 1$. One example of this arises with very short buses, such as in the back plane of a multimicroprocessor system. Another example occurs for polling on multidrop telephone lines; here the data rates are small, leading to packet transmission times much greater than the propagation delay.

In these situations, we see from Eqs. (4.83) and (4.84) that the reservation time per node v is linearly increasing with the detection delay δ and with the (implicit or explicit) token delay. From the queueing delay formulas, Eqs. (4.70) and (4.71), the queueing delay is $mv/2$ in the limit of light load; furthermore, if mv is large relative to the packet transmission time, delay increases at least linearly with mv at all stable loads. Thus, we want to reduce mv , the reservation overhead per cycle.

For simplicity, we now model the multiaccess channel, at least for the purpose of reservations, as a bit synchronous binary "or" channel. That is, at each bit time, the output of the channel is 1 if the input for one or more nodes is 1; otherwise, the output is 0. This is a reasonable model if a separate control wire is used for reservations; it is also reasonable if nodes use short bursts of carrier to request reservations (*i.e.*, the existence of carrier can be detected, but not the number of nodes sending carrier).

This model is very similar to our original slotted multiaccess model in Section 4.2. One difference is that the slot time is reduced to a single bit time; the other difference is that the feedback, instead of being 0,1,*e*, is now 0 or positive. We regard the slots as reservation slots and recognize (as in the examples above), that packets can be transmitted at higher rates than one bit per reservation slot. The question of interest is: How many reservation slots are required to make a reservation?

The simplest strategy within this model is to make reservations by TDM within the reservation slots. Under heavy loading, most nodes will have a packet to send at each turn, and almost each reservation slot successfully establishes a reservation. In the light loading limit, an arriving packet has to wait $m/2$ reservation slots on the average; this is the situation we would like to avoid.

A better strategy, under light loading, is to use a logarithmic search for a node with a packet (see [NiS74] and [Hay76]). Suppose that the nodes are numbered 0 to $m - 1$ and let $n_1, \dots, n_k$ be the binary representation of any given node number where

$$k = \lceil \log_2 m \rceil$$

The algorithm proceeds in successive collision resolution periods (CRP) to find the lowest-numbered node that has a packet to send. In the first slot of a CRP, all active nodes (*i.e.*, all nodes with waiting packets) send a 1 and all other nodes send 0 (*i.e.*, nothing). If the channel output, say y_0 , is 0, the CRP is over and a new CRP starts on the next slot to make reservations for any packets that may have arrived in the interim. If $y_0 = 1$ on the other hand, one or more nodes have packets and the logarithmic search starts.

Assuming that $y_0 = 1$, all active nodes with $n_1 = 0$ send a 1 in the next slot. If y_1 , the output in this slot, is 1, it means that the lowest-numbered node's binary representation starts with 0; otherwise, it starts with 1. In the former case, all active nodes with $n_1 = 1$ become inactive and wait for the end of the CRP to become active again. In the latter case, all active nodes have $n_1 = 1$ and all remain active.

The general rule on the $(i + 1)^{\text{st}}$ slot of the CRP, $1 \leq i \leq k$ (assuming that $y_0 = 1$), is that all active nodes with $n_i = 0$ send a 1; at the end of the slot, if the channel output y_i is 1, all nodes with $n_i = 1$ become inactive. Figure 4.35 shows an example of this strategy. It should be clear that at the end of the $(k + 1)^{\text{st}}$ slot, only the lowest-numbered node is active and the binary representation of that node's number is the bitwise complement of $y_1, \dots, y_k$.

To maintain fairness and to serve the nodes in round-robin order, the nodes should be renumbered after each reservation is made. If the reservation is for node n , say, each node subtracts $n + 1$ modulo m from its current number. Thus, a node's number is one less than its round-robin distance from the last node that transmitted; obviously, this rule could be modified by priorities in any desired way.

Assuming that a packet is sent immediately after a reservation is made and that the next CRP starts after that transmission, it is easy to find the expected delay of this algorithm. Each CRP that starts when the system is busy lasts for $k + 1$ reservation slots. Thus, we regard the packet transmission time as simply being extended by these $k + 1$ slots. If the system is empty at the beginning of a CRP, the CRP lasts for one reservation

Nodes								Output	
000	001	010	011	100	101	110	111	y	
0	0	1	1	0	0	1	0	1	Slot 1
0	0	1	1	0	0	0	0	1	Slot 2
0	0	0	0	0	0	0	0	0	Slot 3
0	0	1	0	0	0	0	0	1	Slot 4

Figure 4.35 Logarithmic search for the lowest-numbered active node. Nodes 010, 011, and 110 have packets. At the end of slot 2, node 110 becomes inactive; at the end of slot 4, node 011 becomes inactive. Outputs from slot 2 to 4, complemented, are 010.

slot and this can be regarded as the server going on vacation for one reservation slot. Equation (3.55) gives the queueing delay for this system.

In contrasting this logarithmic search reservation strategy with TDM reservations, we see that logarithmic search reduces delay from $m/2$ to $k + 1$ reservation slots per packet for light loads but increases delay from 1 to $k + 1$ reservation slots per packet at heavy loads. The obvious question is: How do we combine these strategies to have the best of both worlds? The answer to this question comes from viewing this problem as a source coding problem, much like the framing issue in Chapter 2. The TDM strategy here corresponds to the use of a unary code there to encode frame lengths. The logarithmic search here corresponds to the use of ordinary binary representation. We noticed the advantage there of going to a combined unary–binary representation. In the current context, this means that each CRP should test only a limited number of nodes, say the 2^j lowest-numbered nodes, at a time. If the output is 1, the lowest-numbered node in that set is resolved as above in j additional slots. If the output is 0, 2^j is subtracted from each node’s number and the next CRP starts. Problem 4.30 finds the expected number of reservation slots per packet and the optimal choice of j under several different assumptions.

Note the similarity of these algorithms to the splitting algorithms of Section 4.3. The ideas are the same, and the only difference is that “success” cannot be distinguished from collision here; thus, even though only one node in an active subset contains a packet, the subset must be split until only one node remains.

4.6 PACKET RADIO NETWORKS

Section 4.1.4 briefly described packet radio networks as multiaccess networks in which not all nodes could hear the transmissions of all other nodes. This feature is characteristic both of line-of-sight radio communication in the UHF band (300 to 3,000 MHz) and also of non-line-of-sight communication at HF (3 to 30 MHz). Our interest here is in the effect of partial connectivity on multiaccess techniques rather than the physical characteristics of the radio broadcast medium.

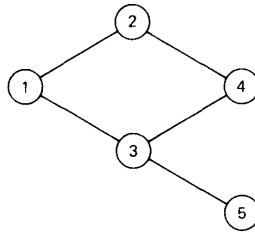


Figure 4.36 Packet radio network: each edge indicates that two nodes at the ends of the edge can hear each other's transmission.

The topology of a radio network can be described by a graph as in Fig. 4.36. The graph, $G = (N, L)$, contains a set of nodes N and a set of links L . Each link in L corresponds to an ordered pair of nodes, say (i, j) , and indicates that transmissions from i can be heard at j . In some situations, node j might be able to hear i , but i is unable to hear j . In such a case $(i, j) \in L$ but $(j, i) \notin L$. This asymmetry does not occur in Fig. 4.36, where each edge denotes two links, one in each direction.

Our assumption about communication in this multiaccess medium is that if node i transmits a packet, that packet will be correctly received by node j if and only if

1. There is a link from i to j [i.e., $(i, j) \in L$], and
2. No other node k for which $(k, j) \in L$ is transmitting while i is transmitting, and
3. j itself is not transmitting while i is transmitting.

Thus, for Fig. 4.36, we see that if nodes 1 and 3 are transmitting simultaneously, node 2 will correctly receive the packet from 1 and node 4 will correctly receive the packet from 3. On the other hand, if nodes 2 and 3 are transmitting simultaneously, nodes 1 and 4 will each see a collision, but node 5 will correctly receive the packet from 3.

It can be seen from this example that having a large number of links in a graph is not necessarily desirable. A large number of links increases the number of pairs of nodes that can communicate directly, but also increases the likelihood of collisions. This trade-off is explored further in Section 4.6.3.

One interesting question that can now be posed is how much traffic can be carried in such a network. Define a *collision-free set* as a set of links that can carry packets simultaneously with no collisions at the receiving ends of the links. For example, $\{(1,2), (3,4)\}$ and $\{(2,1), (5,3)\}$ are both collision-free sets; also, the empty set and each set consisting of a single link are collision-free sets. It is convenient to order the links in some arbitrary order and represent each collision-free set as a vector of 0's and 1's called a *collision-free vector* (CFV). The l^{th} component of a CFV is 1 if and only if the l^{th} link is in the corresponding collision-free set. For example, some CFVs are listed below for the graph of Fig. 4.36.

(1,2)	(2,1)	(1,3)	(3,1)	(2,4)	(4,2)	(3,4)	(4,3)	(3,5)	(5,3)
1	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	1

4.6.1 TDM for Packet Radio Nets

One simple way to use a packet radio net is to choose a given collection of collision-free sets and to cycle between them by TDM. That is, in the i^{th} slot of a TDM cycle, all links in the i^{th} collision-free set can carry packets. With such a TDM strategy, there are no collisions, and the fraction of time that a given link can carry packets is simply the fraction of the collection of collision-free sets that contain that link.

More compactly, if $x_1, \dots, x_J$ are the CFVs corresponding to a collection of J collision-free sets, the vector $f = (\sum_j x_j) / J$ gives the fraction of time that each link can be used. As a slight generalization, the TDM frame could be extended and each collision-free set could be used some arbitrary number of times in the frame. If α_j is the fraction of frame slots using the j^{th} collision-free set, then

$$f = \sum_j \alpha_j x_j \quad (4.86)$$

gives the fractional utilization of each link. A vector of the form $\sum_j \alpha_j x_j$, in which $\sum_j \alpha_j = 1$ and $\alpha_j \geq 0$ for $1 \leq j \leq J$, is called a convex combination of the vectors $x_1, \dots, x_J$. What we have just seen is that any convex combination of CFVs can be approached arbitrarily closely as a fractional link utilization vector through the use of TDM.

Suppose that instead of using TDM as above, we use some sort of collision resolution approach in the network. At any given time, the vector of links that are transmitting packets successfully is a CFV. Averaging this vector of successful link transmissions over time, we get a vector whose ℓ^{th} component is the fraction of time that the ℓ^{th} link is carrying packets successfully. This is also a convex combination of CFVs. Thus, we see that any link utilization that is achievable with collision resolution is also achievable by TDM.

One difficulty with TDM is that delays are longer than necessary for a lightly loaded network. This is not as serious as when all nodes are connected to a common receiver, since if all nodes have only a small number of incoming links, many links can transmit simultaneously and the waiting for a TDM slot is reduced.

A more serious problem with the TDM approach is that the nodes in a packet radio network are usually mobile, and thus the topology of the network is constantly changing. This means that the collision-free sets keep changing, requiring frequent updates of the TDM schedule. This is a difficult problem since even for a static network, the problem of determining whether a potential vector of link utilizations is a convex combination of CFVs falls into a class of difficult problems known as *NP complete* [Ari84]. See [PaS82] for an introduction to the theory of NP complete problems; for our purposes, this simply indicates that the worst-case computational effort to solve the problem increases very rapidly with the number of links in the network. The essential reason for this difficulty is that the number of different collision-free sets typically increases exponentially with the number of links in the network.

Frequency-division multiplexing (FDM) can also be used for packet radio networks in a way very similar to TDM. All links in a collision-free set can use the same frequency band simultaneously, so in principle the links can carry the same amount of traffic as in

TDM. This approach is used in cellular radio networks for mobile voice communication. Here the area covered by the network is divided into a large number of local areas called cells, with each cell having a set of frequency bands for use within the cell. The set of frequency bands used by one cell can be reused by other cells that are sufficiently separated from one another to avoid interference. This provides a simple and practical way to choose a collection of collision-free sets. This same cellular separation principle could be used for TDM.

The discussion so far has ignored the question of how to route packets from source to destination. With the use of a given TDM or FDM structure, each link in the net has a given rate at which it can send packets, and the resource-sharing interaction between links has been removed. Thus, the problem of routing is essentially the same as in conventional networks with dedicated links between nodes; this problem is treated in Chapter 5. When collision resolution strategies are used, however, we shall see that routing is considerably more complicated than in the conventional network case.

4.6.2 Collision Resolution for Packet Radio Nets

Collision resolution is quite a bit trickier for packet radio nets than for the single-receiver systems studied before. The first complication is obtaining feedback information. For the example of Fig. 4.36, suppose that links (2,4) and (3,5) contain packets in a given slot. Then node 4 perceives a collision and node 5 correctly receives a packet. If nodes 5 and 4 send feedback information, node 3 will experience a feedback collision. A second problem is that if a node perceives a collision, it does not know if any of the packets were addressed to it. For both reasons, we cannot assume the perfect 0,1, e feedback that we assumed previously. It follows that the splitting algorithms of Section 4.3 cannot be used and the stabilization techniques of Section 4.2.3 require substantial revisions.

Fortunately, slotted and unslotted Aloha are still applicable, and to a certain extent, some of the ideas of carrier sensing and reservation can still be used. We start by analyzing how slotted Aloha can work in this environment. When an unbacklogged node receives a packet to transmit (either a new packet entering the network, or a packet in transit that has to be forwarded to another node), it sends the packet in the next slot. If no acknowledgment (ack) of correct reception arrives within some time-out period, the node becomes backlogged and the packet is retransmitted after a random delay. Finally, a backlogged node becomes unbacklogged when all of its packets have been transmitted and acked successfully.

There are a number of ways in which acks can be returned to the transmitting node. The simplest is that if node i sends a packet to j that must be forwarded on to some other node k , then if i hears j 's transmission to k , that serves as an ack of the (i, j) transmission. This technique is somewhat defective in two ways. First, some other technique is required to ack packets whose final destination is j . Second, suppose that j successfully relays the packet to k , but i fails to hear the transmission because of a collision. This causes an unnecessary retransmission from i to j , and also requires some way for j to ack, since j has already forwarded the packet to k . Another approach, which can be used in conjunction with the implicit acks above, is for each node to

include explicit acks for the last few packets it has received in each outgoing packet. This approach requires a node to send a dummy packet carrying ack information if the node has no data to send for some period. A third approach, which seems somewhat inferior to the approach above, is to provide time at the end of each slot for explicit acks of packets received within the slot.

Let us now analyze what happens in slotted Aloha for a very heavily loaded network. In particular, assume that all nodes are backlogged all the time and have packets to send on all outgoing links at all times. We can assume that the nodes have infinite buffers to store the backlogged packets, but for the time being, we are not interested in the question of delay. This assumption of constant backlogging is very different from our assumptions in Section 4.2, but the reasons for this will be discussed later. For all nodes i and j , let q_{ij} be the probability that node i transmits a packet to node j in any given slot, and let Q_i be the probability that node i transmits to any node. Thus,

$$Q_i = \sum_j q_{ij} \quad (4.87)$$

To simplify notation, we simply assume that q_{ij} is zero if (i, j) is not in the set of links L . Let p_{ij} be the probability that a transmission on (i, j) is successful. Under our assumption of heavy loading, each node transmits or not in a slot independently of all other nodes. Since p_{ij} is the probability that none of the other nodes in range of j , including j itself, is transmitting, we have

$$p_{ij} = (1 - Q_j) \prod_{\substack{k:(k,j) \in L \\ k \neq i}} (1 - Q_k) \quad (4.88)$$

Finally, the rate f_{ij} of successful packet transmissions per slot (*i.e.*, the throughput) on link (i, j) is

$$f_{ij} = q_{ij} p_{ij} \quad (4.89)$$

Equations (4.87) to (4.89) give us the link throughputs in terms of the attempt rates q_{ij} under the heavy-loading assumption. The question of greater interest, however, is to find the attempt rates q_{ij} that will yield a desired set of throughputs (if that set of throughputs is feasible).

This problem can be solved through an iterative approach. To simplify notation, let q denote a vector whose components are the attempt rates q_{ij} , let p and f be vectors whose components are p_{ij} and f_{ij} , respectively, and let Q be a vector with components Q_i . Given a desired throughput vector f , we start with an initial q^0 which is a vector of 0's. We then use Eqs. (4.87) and (4.88) to find Q^0 and p^0 (Q^0 is thus a vector of 0's and p^0 a vector of 1's). Equation (4.89) is then used to obtain the next iteration for q ; that is, the components of q^1 are given by

$$q_{ij}^1 = \frac{f_{ij}}{p_{ij}^0} \quad (4.90)$$

For each successive iteration, Q^n is found from Eq. (4.87) using q^n , and p^n is found from Eq. (4.88) using Q^n ; then q^{n+1} is found from Eq. (4.89) using p^n . Note

that $q^1 \geq q^0$ (i.e., each component of q^1 is greater than or equal to the corresponding component of q^0). Thus, $Q^1 \geq Q^0$ and $p^1 \leq p^0$. From Eq. (4.89) it can then be seen that $q^2 \geq q^1$. Continuing this argument, it is seen that as long as none of the components of Q exceed 1, q is nondecreasing with successive iterations and p is nonincreasing. It follows that either some component of Q must exceed 1 at some iteration or else q approaches a limit, say q^* , and in this limit Eqs. (4.87) to (4.89) are simultaneously satisfied with the resulting Q^* and p^* .

We now want to show that if (4.87) to (4.89) have any other solution, say q', Q', p' (subject, of course, to $q' \geq 0, Q' \leq 1$), then $q' \geq q^*, Q' \geq Q^*$, and $p' \leq p^*$. To see this, we simply observe that $q^0 \leq q', Q^0 \leq Q'$, and $p^0 \geq p'$. From Eq. (4.89), then, $q^1 \leq q'$. Continuing this argument over successive iterations, $q^n \leq q', Q^n \leq Q'$, and $p^n \geq p'$ for all n , so the result holds in the limit. This argument also shows that if some component of Q^n exceeds 1 for some n , then Eqs. (4.87) to (4.89) have no solution (i.e., that the given f is infeasible).

Next, assume we know the input rates to the network and know the routes over which the sessions will flow, so that in principle we can determine the steady-state rates f'_{ij} at which the links must handle traffic. We would like to choose the throughputs of each link under heavy load to exceed these steady-state rates so that the backlogs do not build up indefinitely. One approach then is to find the largest number $\beta > 1$ for which $f = \beta f'$ is feasible under the heavy-load assumption. Given this largest f , and the corresponding attempt rates q , we can then empty out the backlog as it develops.

There is one difficulty here, and that is that if some nodes are backlogged and others are not, the unbacklogged nodes no longer choose their transmission times independently. Thus, it is conceivable in bizarre cases that some backlogged nodes fare more poorly when other nodes are unbacklogged than they do when all nodes are backlogged. Problem 4.32 gives an example of this phenomenon. One way to avoid this difficulty is for new packets at a node to join the backlog immediately rather than being able to transmit in the next slot. This, of course, increases delay under light-loading conditions. The other approach is to live dangerously and hope for the best. To a certain extent, one has to do this anyway with packet radio, since with a changing topology, one cannot maintain carefully controlled attempt rates.

Our reason for focusing on the heavily loaded case is that the number of links entering each node is usually small for a packet radio net, and thus the attempt rates can be moderately high even under the heavy-loading assumption. For the single-receiver case, on the other hand, the number of nodes tends to be much larger, and thus the attempt rates appropriate for heavy loading tend to create large delays. The other reason is that stabilization is a much harder problem here than in the single-receiver case; a node cannot help itself too much by adjusting its own attempt rates, since other nodes might be causing congestion but not experiencing any congestion themselves (see Problem 4.32).

4.6.3 Transmission Radii for Packet Radio

In the previous subsections, we viewed the set of links in a packet radio net as given. It can be seen, however, that if a node increases its transmitter power, its transmission will

be heard by a larger set of nodes. The following qualitative argument shows that it is desirable to keep the power level relatively small so that each node has a moderately small set of incoming and outgoing links. Assume for simplicity that we have a symmetric net in which each node has exactly n incoming links and n outgoing links. Suppose further that each link has an identical traffic-carrying requirement. It is not hard to convince oneself that Eqs. (4.87) to (4.89) are satisfied by an identical attempt rate q on each link. Each Q_i is then nq , each p_{ij} is given by $(1 - nq)^n$, and finally,

$$f = q(1 - nq)^n \quad (4.91)$$

It is easy to verify that f is maximized by choosing $q = 1/[n(n+1)]$, and the resulting value of f is approximately $1/(en^2)$. Each node then sends packets successfully at a rate of $1/en$. If there are m nodes in the network and the average number of links on the path from source to destination is J , the rate at which the network can deliver packets is $m/(Jen)$ packets per slot.

Now let us look at what happens when the transmission radius R over which a node can be heard varies. The number of nodes within radius R of a given node will vary roughly as R^2 ; so the rate at which an individual node sends packets successfully will decrease as $1/R^2$. On the other hand, as R increases, the routing will presumably be changed to send the packets as far as possible toward the destination on each link of the path. Thus, we expect the number of links on a path to decrease as $1/R$. Thus, if J is proportional to $1/R$ and n is proportional to R^2 , the rate at which the network can deliver packets is proportional to $1/R$, leading us to believe that R should be kept very small.

The very crude analysis above leaves out two important factors. First, when R and n are large, a packet can move almost a distance R toward its destination on each link of a well-chosen path, so that J is essentially proportional to $1/R$ in that region. When R gets small, however, the paths become very circuitous and thus, J increases with decreasing R much faster than $1/R$. Second, when R is too small, the network is not very well connected, and some links might have to carry very large amounts of traffic. This leads us to the conclusion that R should be small, but not too small, so that n is considerably larger than 1. Takagi and Kleinrock [TaK85] have done a much more careful analysis (although still with some questionable assumptions) and have concluded that the radius should be set so that n is on the order of 8.

4.6.4 Carrier Sensing and Busy Tones

We saw in Section 4.4 that carrier sensing yielded a considerable improvement over slotted Aloha in the situation where all nodes could hear all other nodes and the propagation delay is small. For line-of-sight radio, the propagation delay is typically small relative to packet transmission times, so it is reasonable to explore how well carrier sensing will work here. Unfortunately, if node i is transmitting to node j , and node k wants to transmit to j , there is no assurance that k can hear i . There might be an obstruction between i and k , or they might simply be out of range of each other. Thus, carrier sensing will serve to prevent some collisions from occurring, but cannot prevent others.

To make matters worse, with carrier sensing, there is no uniform slotting structure, and thus, carrier sensing loses some of the advantage that slotted Aloha has over pure Aloha. Finally, radio transmission is subject to fading and variable noise, so that the existence of another transmitting node, even within range, is hard to detect in a short time. For these reasons, carrier sensing is not very effective for packet radio.

A busy tone ([ToK75] and [SiS81]) is one approach to improving the performance of carrier sensing in a packet radio network. Whenever any node detects a packet being transmitted, it starts to send a signal, called a busy tone, in a separate frequency band. Thus, when node i starts to send a packet to node j , node j (along with all other nodes that can hear i) will start to send a busy tone. All the nodes that can hear j will thus avoid transmitting; thus, assuming reciprocity (*i.e.*, the nodes that can hear j are the same as the nodes that j can hear), it follows that j will experience no collision.

A problem with the use of busy tones is that when node i starts to send a packet, *all* the nodes in range of i will start to send busy tones, and thus every node within range of any node in range of i will be inhibited from transmitting. Using the very crude type of analysis in the last subsection, and assuming a transmission radius of R , we see that when node i starts to transmit, most of the nodes within radius $2R$ of i will be inhibited. This number will typically be about four times the number of nodes within radius R of the receiving node, which is the set of nodes that should be inhibited. Thus, from a throughput standpoint, this is not a very promising approach.

Another variation on the busy tone approach is for a node to send a busy tone only after it receives the address part of the packet and recognizes itself as the intended recipient. Aside from the complexity, this greatly increases β , the time over which another node could start to transmit before hearing the busy tone.

It can be seen that packet radio is an area in which many more questions than answers exist, both in terms of desirable structure and in terms of analysis. Questions of modulation and detection of packets make the situation even more complex. In military applications, it is often desirable to use spread-spectrum techniques for sending packets. One of the consequences of this is that if two packets are being received at once, the receiver can often lock on to one, with the other acting only as wideband noise. If a different spread-spectrum code is used for each receiver, the situation is even better, since the receiver can look for only its own sequence and thus reject simultaneous packets sent to other receivers. Unfortunately, attenuation is often quite severe in line-of-sight communication, so that unwanted packets can arrive at a node with much higher power levels than the desired packets, and still cause a collision.

SUMMARY

The central problem of multiaccess communication is that of sharing a communication channel between a multiplicity of nodes where each node has sporadic service requirements. This problem arises in local area networks, metropolitan area networks, satellite networks, and various types of radio networks.

Collision resolution is one approach to such sharing. Inherently, collision resolution algorithms can achieve small delay with a large number of lightly loaded nodes, but stability is a major concern. The joint issues of stability, throughput, and delay are studied most cleanly with the infinite node assumption. This assumption lets one study collision resolution without the added complication of individual queues at each node. Under this assumption, we found that throughputs up to $1/e$ packets per slot were possible with stabilized slotted Aloha, and throughputs up to 0.487 packets per slot were possible with splitting algorithms.

Reservations provide the other major approach to multiaccess sharing. The channel can be reserved by a prearranged fixed allocation (e.g., TDM or FDM) or can be reserved dynamically. Dynamic reservations further divide into the use of collision resolution and the use of TDM (or round-robin ordering) to make the reservations for channel use. CSMA/CD (i.e., Ethernet) is a popular example of the use of collision resolution to make (implicit) reservations. Token rings, token buses, and their elaborations are examples of the use of round-robin ordering to make reservations.

There are an amazing variety of ways to use the special characteristics of particular multiaccess media to make reservations in round-robin order. Some of these variations require a time proportional to β (the propagation delay) to make a reservation, and some require a time proportional to β/m , where m is the number of nodes. The latter variations are particularly suitable for high-speed systems with $\beta > 1$.

Packet radio systems lie at an intermediate point between pure multiaccess systems, where all nodes share the same medium (and thus no explicit routing is required), and point-to-point networks, where routing but no multiaccess sharing is required. Radio networks are still in a formative and fragmentary stage of research.

NOTES, SOURCES, AND SUGGESTED READING

Section 4.2. The Aloha network was first described in [Abr70] and the slotted improvement in [Rob72]. The problem of stability was discussed in [Met73], [LaK75], and [CaH75]. Binary exponential backoff was developed in [MeB76]. Modern approaches to stability are treated in [HaL82] and [Riv85].

Section 4.3. The first tree algorithms are due to [Cap77], [TsM78], and [Hay76]. [Mas80] provided improvements and simple analysis techniques. The FCFS splitting algorithm is due to [Gal78] and, independently, [TsM80]. Upper bounds on maximum throughput (with assumptions 1 to 6b of Section 4.2.1) are in [Pip81] and [Mit81]. The March 1985 issue of the *IEEE Transactions on Information Theory* is a special issue on random-access communications; the articles provide an excellent snapshot of the status of work related to splitting algorithms.

Section 4.4. The classic works on CSMA are [KIT75] and [Tob74].

Section 4.5. The literature on local area networks and satellite networks is somewhat overwhelming. [Sta85] provides a wealth of practical details. Good source refer-

ences in the satellite area are [JBH78], [CRW73], [Bin75], and [WiE80]. For local area networks, [MeB76] is the source work on Ethernet, and [FaN69] and [FaL72] are source works on ring nets. [CPR78] and [KuR82] are good overview articles and [Ros86] provides a readable introduction to FDDI. [FiT84] does an excellent job of comparing and contrasting the many approaches to implicit and explicit tokens and polling on buses. Finally, DQDB is treated in [NBH88] and [HCM90].

Section 4.6. [KGB78] provides a good overview of packet radio. Busy tones are described in [ToK75] and [SiS81]. Transmission radii are discussed in [TaK85].

PROBLEMS

- 4.1** (a) Verify that the steady-state probabilities p_n for the Markov chain in Fig. 4.3 are given by the solution to the equations

$$p_n = \sum_{i=0}^{n+1} p_i P_{in}$$

$$\sum_{n=0}^m p_n = 1$$

- (b) For $n < m$, use part (a) to express p_{n+1} in terms of $p_0, p_1, \dots, p_n$.
 (c) Express p_1 in terms of p_0 and then p_2 in terms of p_0 .
 (d) For $m = 2$, solve for p_0 in terms of the transition probabilities.
- 4.2** (a) Show that P_{succ} in Eq. (4.5) can be expressed as

$$P_{succ} = \left[\frac{(m-n)q_a}{1-q_a} + \frac{nq_r}{1-q_r} \right] (1-q_a)^{m-n} (1-q_r)^n$$

- (b) Use the approximation $(1-x)^y \approx e^{-xy}$ for small x to show that for small q_a and q_r ,

$$P_{succ} \approx G(n)e^{-G(n)}$$

where $G(n) = (m-n)q_a + nq_r$.

- (c) Note that $(1-x)^y = e^{y \ln(1-x)}$. Expand $\ln(1-x)$ in a power series and show that

$$\frac{(1-x)^y}{e^{-xy}} = \exp \left(-\frac{x^2 y}{2} - \frac{x^3 y}{3} \dots \right)$$

Show that this ratio is close to 1 if $x \ll 1$ and $x^2 y \ll 1$.

- 4.3** (a) Redraw Fig. 4.4 for the case in which $q_r = 1/m$ and $q_a = 1/m\epsilon$.
 (b) Find the departure rate (i.e., P_{succ}) in the fully backlogged case $n = m$.
 (c) Note that there is no unstable equilibrium or undesired stable point in this case and show (graphically) that this holds true for any value of q_a .
 (d) Solve numerically (using $q_a = 1/m\epsilon$) for the value of G at which the stable point occurs.
 (e) Find n/m at the stable point. Note that this is the fraction of the arriving packets that are not accepted by the system at this typical point.

- 4.4 Consider the idealized slotted multiaccess model of Section 4.2.1 with the no-buffering assumption. Let n_k be the number of backlogged nodes at the beginning of the k^{th} slot and let $\bar{n}$ be the expected value of n_k over all k . Note that $\bar{n}$ will depend on the particular way in which collisions are resolved, but we regard $\bar{n}$ as given here (see [HIG81]).
- Find the expected number of *accepted* arrivals per slot, $\bar{N}_a$, as a function of $\bar{n}$, m , and q_a , where m is the number of nodes and q_a is the arrival rate per node.
 - Find the expected departure rate per slot, $\bar{P}_{succ}$, as a function of $\bar{n}$, m , and q_a . *Hint:* How is $\bar{N}_a$ related to $\bar{P}_{succ}$? Recall that both are averages over time.
 - Find the expected number of packets in the system, $\bar{N}_{sys}$, immediately after the beginning of a slot (the number in the system is the backlog plus the accepted new arrivals).
 - Find the expected delay T of an accepted packet from its arrival at the beginning of a slot until the completion of its successful transmission at the end of a slot. *Hint:* Use Little's theorem; it may be useful to redraw the diagram used to prove Little's theorem.
 - Suppose that the strategy for resolving collisions is now modified and the expected backlog $\bar{n}$ is reduced to $\bar{n}' < \bar{n}$. Show that $\bar{N}_a$ increases, $\bar{P}_{succ}$ increases, $\bar{N}_{sys}$ decreases, and T decreases. Note that this means that improving the system with respect to one of these parameters improves it with respect to all.
- 4.5 Assume for simplicity that each transmitted packet in a slotted Aloha system is successful with some fixed probability p . New packets are assumed to arrive at the beginning of a slot and are transmitted immediately. If a packet is unsuccessful, it is retransmitted with probability q_r in each successive slot until successfully received.
- Find the expected delay T from the arrival of a packet until the completion of its successful transmission. *Hint:* Given that a packet has not been transmitted successfully before, what is the probability that it is both transmitted and successful in the i^{th} slot ($i > 1$) after arrival?
 - Suppose that the number of nodes m is large, and that q_a and q_r are small. Show that in state n , the probability p that a given packet transmission is successful is approximately $p = e^{-G(n)}$, where $G(n) = (m - n)q_a + nq_r$.
 - Now consider the stable equilibrium state n^* of the system where $G = G(n^*)$; $Ge^{-G} = (m - n^*)q_a$. Substitute (b) into your expression for T for (a), using $n = n^*$, and show that

$$T = 1 + \frac{n^*}{q_a(m - n^*)}$$

(Note that if n^* is assumed to be equal to $\bar{n}$ in Problem 4.4, this is the same as the value of T found there.)

- Solve numerically for T in the case where $q_a m = 0.3$ and $q_r m = 1$; show that $n^* \approx m/8$, corresponding to 1/8 loss of incoming traffic, and $T \approx m/2$, giving roughly the same delay as TDM.
- 4.6 (a) Consider P_{succ} as given exactly in Eq. (4.5). For given $q_a < 1/m$, $n > 1$, show that the value of q_r that maximizes P_{succ} satisfies

$$\frac{1}{1 - q_r} - \frac{q_a(m - n)}{1 - q_a} - \frac{q_r n}{1 - q_r} = 0$$

- Consider the value of q_r that satisfies the equation above as a function of q_a , say $q_r(q_a)$. Show that $q_r(q_a) > q_a$ (assume that $q_a < 1/m$).
- Take the total derivative of P_{succ} with respect to q_a , using $q_r(q_a)$ for q_r , and show that this derivative is negative. *Hint:* Recall that $\partial P_{succ} / \partial q_r$ is 0 at $q_r(q_a)$ and compare $\partial P_{succ} / \partial q_a$ with $\partial P_{succ} / \partial q_r$.

- (d) Show that if q_r is chosen to maximize P_{succ} and $q_r < 1$, then P_{succ} is greater if new arrivals are treated immediately as backlogged than if new arrivals are immediately transmitted. *Hint:* In the backlog case, a previously unbacklogged node transmits with probability $q_a q_r < q_a$.
- 4.7** Consider a slotted Aloha system with “perfect capture.” That is, if more than one packet is transmitted in a slot, the receiver “locks onto” one of the transmissions and receives it correctly; feedback immediately informs each transmitting node about which node was successful and the unsuccessful packets are retransmitted later.
- (a) Give a convincing argument why expected delay is minimized if all waiting packets attempt transmission in each slot.
- (b) Find the expected system delay assuming Poisson arrivals with overall rate λ . *Hint:* Review Example 3.16.
- (c) Now assume that the feedback is delayed and that if a packet is unsuccessful in the slot, it is retransmitted on the k^{th} subsequent slot rather than the first subsequent slot. Find the new expected delay as a function of k . *Hint:* Consider the system as k subsystems, the i^{th} subsystem handling arrivals in slots j such that $j \bmod k = i$.
- 4.8** Consider a slotted system in which all nodes have infinitely large buffers and all new arrivals (at Poisson rate λ/m per node) are allowed into the system, but are considered as backlogged immediately rather than transmitted in the next slot. While a node contains one or more packets, it independently transmits one packet in each slot, with probability q_r . Assume that any given transmission is successful with probability p .
- (a) Show that the expected time from the beginning of a backlogged slot until the completion of the first success at a given node is $1/pq_r$. Show that the second moment of this time is $(2 - pq_r)/(pq_r)^2$.
- (b) Note that the assumption of a constant success probability allows each node to be considered independently. Assume that λ/m is the Poisson arrival rate at a node, and use the service-time results of part (a) to show that the expected delay is

$$T = \frac{1}{q_r p (1 - \rho)} + \frac{1 - 2\rho}{2(1 - \rho)}$$

$$\rho = \frac{\lambda}{mpq_r}$$

- (c) Assume that $p = 1$ (this yields a smaller T than any other value of p , and corresponds to very light loading). Find T for $q_r = 1/m$; observe that this is roughly twice the delay for TDM if m is large.
- 4.9** Assume that the number of packets n in a slotted Aloha system at a given time is a Poisson random variable with mean $\hat{n} \geq 1$. Suppose that each packet is independently transmitted in the next slot with probability $1/\hat{n}$.
- (a) Find the probability that the slot is idle.
- (b) Show that the a posteriori probability that there were n packets in the system, given an idle slot, is Poisson with mean $\hat{n} - 1$.
- (c) Find the probability that the slot is successful.
- (d) Show that the a posteriori probability that there were $n + 1$ packets in the system, given a success, is $e^{-(\hat{n}-1)}(\hat{n}-1)^n/n!$ (i.e., the number of remaining packets is Poisson with mean $\hat{n} - 1$).
- 4.10** Consider a slotted Aloha system satisfying assumptions 1 to 6a of Section 4.2.1 *except* that each of the nodes has a limitless buffer to store all arriving packets until transmission and

that nodes receive immediate feedback only about whether or not their own packets were successfully transmitted. Each node is in one of two different modes. In mode 1, a node transmits (with probability 1) in each slot, repeating unsuccessful packets, until its buffer of waiting packets is exhausted; at that point the node goes into mode 2. In mode 2, a node transmits a “dummy packet” in each slot with probability q_r until a successful transmission occurs, at which point it enters mode 1 (dummy packets are used only to simplify the mathematics). Assume that the system starts with all nodes in mode 2. Each node has Poisson arrivals of rate λ/m .

- (a) Explain why at most one node at a time can be in mode 1.
- (b) Given that a node is in mode 1, find its probability, p_1 , of successful transmission. Find the mean time $\bar{X}$ between successful transmissions and the second moment $\bar{X}^2$ of this time. *Hint:* Review the ARQ example in Section 3.5.1, with $N = 1$.
- (c) Given that all nodes are in mode 2, find the probability p_2 that some dummy packet is successfully transmitted in a given slot. Find the mean time $\bar{v}$ until the completion of such a successful transmission and its second moment $\bar{v}^2$.
- (d) Regard the intervals of time when all nodes are in mode 2 as reservation intervals. Show that the mean time a packet must wait in queue before first attempting transmission is

$$W = \frac{R + E\{S\}\bar{v}}{1 - \rho}, \quad \rho = \lambda\bar{X}$$

where R is the mean residual time until completion of a service in mode 1 or completion of a reservation interval, and S is the number of whole reservation intervals until the node at which the packet arrived is in mode 1.

- (e) Show that

$$W = \frac{\lambda(2 - p_1)}{2p_1^2(1 - \rho)} + \frac{2 - p_2}{2p_2} + \frac{m - 1}{p_2(1 - \rho)}$$

Show that W is finite if $q_r < (1 - \lambda)^{1/(m-1)}$.

- 4.11 Consider the somewhat unrealistic feedback assumption for unslotted Aloha in which all nodes are informed, precisely τ time units after the beginning of each transmission whether or not that transmission was successful. Thus, in the event of a collision, each node knows how many packets were involved in the collision, and each node involved in the collision knows how many other nodes started transmission before itself. Assume that each transmission lasts one time unit and assume that $m = \infty$. Consider a retransmission strategy in which the first node involved in a collision waits one time unit after receiving feedback on its collision and then transmits its packet. Successive nodes in the collision retransmit in order spaced one time unit apart. All new arrivals to the system while these retransmissions are taking place wait until the retransmissions are finished. At the completion of the retransmissions, each backlogged node chooses a time to start its transmission uniformly distributed over the next time unit. All new arrivals after the end of the retransmissions above start transmission immediately.
 - (a) Approximate the system above as a reservation system with reservation intervals of duration $1 + \tau$ (note that this is an approximation in the sense that successful transmissions will sometimes occur in the reservation intervals, but the approximation becomes more accurate as the loading becomes higher). Find the expected packet delay for this approximation (assume Poisson arrivals at rate λ).
 - (b) Show that the delay above remains finite for all $\lambda < 1$.

- 4.12** This problem illustrates that the maximum throughput of unslotted Aloha can be increased up to e^{-1} at an enormous cost in delay. Consider a finite but large set m of nodes with unlimited buffering at each node. Each node waits until it has accumulated k packets and then transmits them one after the other in the next k time units. Those packets involved in collisions, plus new packets, are then retransmitted a random time later, again k at a time. Assume that the starting time of transmissions from all nodes collectively is a Poisson process with parameter G (i.e., ignore stability issues).
- (a) Show that the probability of success on the j^{th} of the k packets in a sequence is $e^{-G(k+1)}$. *Hint:* Consider the intervals between the initiation of the given sequence and the preceding sequence and subsequent sequence.
 - (b) Show that the throughput is $kGe^{-G(k+1)}$, and find the maximum throughput by optimizing over G .
- 4.13** (a) Consider a CRP that results in the feedback pattern $e, 0, e, e, 1, 1, 0$ when using the tree algorithm as illustrated in Fig. 4.9. Redraw this figure for this feedback pattern.
- (b) Which collision or collisions would have been avoided if the first improvement to the tree algorithm had been used?
 - (c) What would the feedback pattern have been for the CRP if both improvements to the tree algorithms had been used?
- 4.14** Consider the tree algorithm in Fig. 4.9. Given that k collisions occur in a CRP, determine the number of slots required for the CRP. Check your answer with the particular example of Fig. 4.9. *Hint 1:* Note that each collision corresponds to a nonleaf node of the rooted tree. Consider “building” any given tree from the root up, successively replacing leaf nodes by internal nodes with two upward-going edges. *Hint 2:* For another approach, consider what happens in the stack for each collision, idle, or success.
- 4.15** Consider the tree algorithm in Fig. 4.9. Assume that after each collision, each packet involved in the collision flips an unbiased coin to determine whether to go into the left or right subset.
- (a) Given a collision of k packets, find the probability that i packets go into the left subset.
 - (b) Let A_k be the expected number of slots required in a CRP involving k packets. Note that $A_0 = A_1 = 1$. Show that for $k \geq 2$,

$$A_k = 1 + \sum_{i=0}^k \binom{k}{i} 2^{-k} (A_i + A_{k-i})$$

- (c) Simplify your answer in part (b) to the form

$$A_k = c_{kk} + \sum_{i=0}^{k-1} c_{ik} A_i$$

and find the coefficients c_{ik} . Evaluate A_2 and A_3 numerically. For more results on A_k for large k , and the use of A_k in evaluating maximum throughput, see [Mas80].

- 4.16** (a) Consider the first improvement to the tree algorithm as shown in Fig. 4.10. Assume that each packet involved in a collision flips an unbiased coin to join either the left or right subset. Let B_k be the expected number of slots required in a CRP involving k packets; note that $B_0 = B_1 = 1$. Show that for $k \geq 2$,

$$B_k = 1 + \sum_{i=1}^k \binom{k}{i} 2^{-k} (B_i + B_{k-i}) + 2^{-k} B_k$$

(b) Simplify your answer to the form

$$B_k = C'_{kk} + \sum_{i=1}^{k-1} C'_{ik} B_i$$

and evaluate the constants C'_{ik} . Evaluate B_2 and B_3 numerically (see [Mas80]).

4.17 Let X_L and X_R be independent Poisson distributed random variables, each with mean G . Use the definition of conditional probabilities to evaluate the following:

- (a) $P\{X_L = 0 \mid X_L + X_R \geq 2\}$
- (b) $P\{X_L = 1 \mid X_L + X_R \geq 2\}$
- (c) $P\{X_L \geq 2 \mid X_L + X_R \geq 2\}$
- (d) $P\{X_R = 1 \mid X_L = 1, X_L + X_R \geq 2\}$
- (e) $P\{X_R = i \mid X_L = 0, X_L + X_R \geq 2\} \quad (i \geq 2)$
- (f) $P\{X_R = i \mid X_L \geq 2, X_L + X_R \geq 2\}$

4.18 Suppose that at time k , the FCFS splitting algorithm allocates a new interval from $T(k)$ to $T(k) + \alpha_0$. Suppose that this interval contains a set of packets with arrival times $T(k) + 0.1\alpha_0, T(k) + 0.6\alpha_0, T(k) + 0.7\alpha_0$, and $T(k) + 0.8\alpha_0$.

- (a) Find the allocation intervals for each of the subsequent times until the CRP is completed.
 - (b) Indicate which of the rules of Eqs. (4.15) to (4.18) are used in determining each of these allocation intervals.
 - (c) Indicate the path through the Markov chain in Fig. 4.13 for this sequence of events.
- 4.19** (a) Show that $\bar{n}$, the expected number of packets successfully transmitted in a CRP of the FCFS splitting algorithm, is given by

$$\bar{n} = 1 - e^{-G_0} + \sum_{i=1}^{\infty} p(R, i)$$

(Assume that the initial allocation interval is α_0 , with $G_0 = \alpha_0\lambda$.)

(b) Show that

$$\bar{n} = \lambda\alpha_0(1 - E\{f\})$$

where $E\{f\}$ is the expected fraction of α_0 returned to the waiting interval in a CRP. (This provides an alternative way to calculate $E\{f\}$.)

4.20 Show, for Eq. (4.41), that if n_k is a Poisson random variable with mean $\hat{n}_k$, then the a posteriori distribution of n_k , given an idle, is Poisson with mean $\hat{n}_k[1 - q_r(\hat{n}_k)]$. Show that the a posteriori distribution of $n_k - 1$, given a success, is Poisson with mean $\hat{n}_k[1 - q_r(\hat{n}_k)]$.

4.21 *Slotted CSMA with Variable Packet Lengths.* Assume that the time to transmit a packet is a random variable X ; for consistency with the slotted assumption, assume that X is discrete, taking values that are integer multiples of β . Assume that all transmissions are independent and identically distributed (IID) with mean $\bar{X} = 1$.

- (a) Let Y be the longer of two IID transmissions X_1 and X_2 [i.e., $Y = \max(X_1, X_2)$]. Show that the expected value of Y satisfies $\bar{Y} \leq 2\bar{X}$. Show that if X takes the value of β with large probability and $k\beta$ (for a large k) with small probability, this bound is close to equality.
- (b) Show that the expected time between state transitions, given a collision of two packets, is at most $2 + \beta$.

- (c) Let $g(n) = \lambda\beta + q_r n$ be the expected number of attempted transmissions following a state transition to state n , and assume that this number of attempts is Poisson with mean $g(n)$. Show that the expected time between state transitions in state n is at most

$$\beta e^{-g(n)} + (1 + \beta)g(n)e^{-g(n)} + (1 + \beta/2)g^2(n)e^{-g(n)}$$

Ignore collisions of more than two packets as negligible.

- (d) Find a lower bound to the expected number of departures per unit time in state n [see Eq. (4.39)].
 (e) Show that this lower bound is maximized (for small β) by

$$g(n) \approx \sqrt{\beta}$$

with a corresponding throughput of approximately $1 - 2\sqrt{\beta}$.

- 4.22 Pseudo-Bayesian Stabilization for Unslotted CSMA.** Assume that at the end of a transmission, the number n of backlogged packets in the system is a Poisson random variable with mean $\hat{n}$. Assume that in the idle period until the next transmission starts, each backlogged packet attempts transmission at rate x and each new arrival starts transmission immediately. Thus, given n , the time τ until the next transmission starts has probability density $p(\tau | n) = (\lambda + xn)e^{-(\lambda + xn)\tau}$.

- (a) Find the unconditional probability density $p(\tau)$.
 (b) Find the a posteriori probability $P\{n, b | \tau\}$ that there were n backlogged packets and one of them started transmission first, given that the transmission starts at τ .
 (c) Find the a posteriori probability $P\{n, a | \tau\}$ that there were n backlogged packets and a new arrival started transmission first, given that this transmission starts at τ .
 (d) Let n' be the number of backlogged packets immediately after the next transmission starts (not counting the packet being transmitted); that is, $n' = n - 1$ if a backlogged packet starts and $n' = n$ if a new arrival starts. Show that, given τ , n' is Poisson with mean $\hat{n}' = \hat{n}e^{-\tau x}$.

This means that the pseudo-Bayesian rule for updating estimated backlog (assuming unit time transmission) is to estimate the backlog at the end of the $(k + 1)^{\text{st}}$ transmission in terms of the estimate at the end of the k^{th} transmission and the idle period τ_k between the transmissions by

$$\hat{n}_{k+1} = \begin{cases} \hat{n}_k e^{-\tau_k x_k} + \lambda(1 + \beta); & \text{success} \\ \hat{n}_k e^{-\tau_k x_k} + 2 + \lambda(1 + \beta); & \text{collision} \end{cases}$$

$$x_k = \beta^{-\frac{1}{2}} \min\left(\frac{1}{\hat{n}_k}, 1\right)$$

- 4.23** Give an intuitive explanation of why the maximum throughput, for small β , is approximately the same for CSMA slotted Aloha and FCFS splitting with CSMA. Show that the optimal expected number of packets transmitted after a state transition in Aloha is the same as that at the beginning of a CRP for FCFS splitting. Note that after a collision, the expected numbers are slightly different in the two systems, but the difference is unimportant since collisions are rare.

4.24 *Delay of Ideal Slotted CSMA/CD.* Assume that for all positive backlogs, the number of packets attempting transmission in an interval is Poisson with mean g .

(a) Start with Eq. (4.42), which is valid for CSMA/CD as well as CSMA. Show that for CSMA/CD,

$$E\{t\} = \frac{\beta e^{-g} + (1 + \beta)g e^{-g} + 2\beta[1 - (1 + g)e^{-g}]}{g e^{-g}}$$

Show that $E\{t\}$ is minimized over $g > 0$ by $g = 0.77$, and

$$\min_g E\{t\} = 1 + 3.31\beta$$

(b) Show that for this g and mean packet length 1,

$$W = \frac{\bar{R} + \bar{g}}{1 - \lambda(1 + 3.31\beta)}$$

(c) Evaluate $\bar{R}$ and $\bar{g}$ to verify Eq. (4.67) for small β .

(d) Discuss the assumption of a Poisson-distributed number of packets attempting transmission in each interval, particularly for a backlog of 1.

4.25 Show that for unslotted CSMA/CD, the maximum interval of time over which a transmitting node can hear a collision is 2β . (Note in Fig. 4.20 that the time when a collision event starts at one node until it ends at another node can be as large as 3β .)

4.26 Consider an unslotted CSMA/CD system in which the propagation delay is negligible compared to the time β required for a node to detect that the channel is idle or busy. Assume that each packet requires one time unit for transmission. Assume that β time units after either a successful transmission or a collision ends, all backlogged nodes attempt transmission after a random delay and that the composite process of initiation times is Poisson of rate G (up to time β after the first initiation). For simplicity, assume that each collision lasts for β time units.

(a) Find the probability that the first transmission initiation after a given idle detection is successful.

(b) Find the expected time from one idle detection to the next.

(c) Find the throughput (for the given assumptions).

(d) Optimize the throughput numerically over G .

4.27 Modify Eq. (4.71) for the case in which a node, after transmitting a packet, waits for the packet to return before transmitting the free token. *Hint:* View the added delay as an increase in the packet transmission time.

4.28 Show by example that a node in a register insertion ring might have to wait an arbitrarily long time to transmit a packet once its transit buffer is full.

4.29 Suppose that two nodes are randomly placed on a bus; that is, each is placed independently, and the position of each is chosen from a uniform distribution over the length of the bus. Assuming that the length of the bus is 1 unit, show that the expected distance between the nodes is $1/3$.

4.30 Assume, for the analysis of generalized polling in Section 4.5.6, that each node has a packet to send with probability q .

(a) Find the probability $P\{i\}$ that node i , $i \geq 0$, is the lowest-numbered node with a packet to send.

(b) Assume that the CRP tests only the first 2^j lowest-numbered nodes at a time for some j . Represent the lowest-numbered node with a packet as $i = k2^j + r$, where k is an

- integer, and $0 \leq r < 2^j$. Show that, given i , the number of reservation slots needed to find i is $k + 1 + j$.
- (c) Assume that the total number of nodes is infinite and approximate k above by $i2^{-j}$. Find the expected number of reservation slots to find the lowest-numbered node i containing a packet.
- (d) Find the integer value of j that minimizes your answer in part (c). *Hint:* Find the smallest value of j for which the expected number of reservation slots is less than the expected number for $j + 1$.
- 4.31** Consider the simple packet radio network shown in Fig. 4.37 and let f_ℓ be the throughput on link ℓ ($\ell = 1, 2, 3$); the links are used only in the direction shown.
- (a) Note that at most one link can be in any collision-free set. Using generalized TDM [as in Eq. (4.86)], show that any set of throughputs satisfying $f_1 + f_2 + f_3 \leq 1$, $f_\ell \geq 0$ ($\ell = 1, 2, 3$) can be achieved.
- (b) Suppose that $f_1 = f_2 = f$ and $f_3 = 2f$ for some f . Use Eqs. (4.88) and (4.89) to relate f_1, f_2, f_3 to the attempt rates q_1, q_2, q_3 for slotted Aloha. Show that $q_1 = q_2$ and $q_3 = 2f$.
- (c) Find the maximum achievable value of f for part (b).
- 4.32** Consider the packet radio network shown in Fig. 4.38 in which the links are used only in the direction shown. Links 5 and 6 carry no traffic but serve to cause collisions for link 7 when packets are transmitted on links 3 and 4, respectively.
- (a) Show that throughputs $f_1 = f_2 = f_3 = f_4 = 1/3$, $f_7 = 4/9$ are feasible for the assumption of heavy loading and independent attempts on each link. Find the corresponding attempt rates and success rates from Eqs. (4.88) and (4.89).
- (b) Now assume that links 3 and 4 are used to forward incoming traffic and always transmit in the slot following a successful incoming packet. Show that with $f_1 = f_2 = f_3 = f_4 = 1/3$, f_7 is restricted to be no more than $1/3$.
- (c) Assume finally that packet attempts on links 1 and 2 are in alternating order and links 3 and 4 operate as in part (b). Show that $f_1 = f_2 = f_3 = f_4 = 1/2$ is feasible, but that f_7 must then be 0.

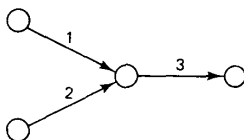


Figure 4.37

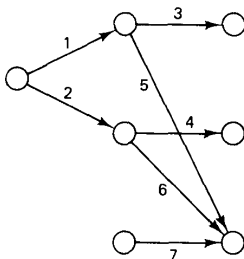
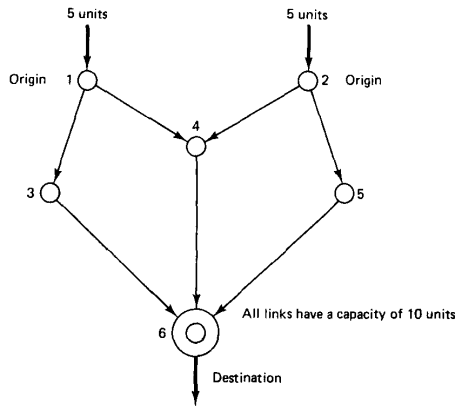


Figure 4.38

- 4.33** Suppose that an FDDI ring has two users. The target token rotation time is 3 msec and $\alpha_1 = \alpha_2 = 1$ msec. Assume that neither node has any traffic to send up to time 0, and then both nodes have an arbitrarily large supply of both high- and low-priority traffic. Node 0 is the first node to send traffic on the ring starting at time 0. Find the sequence of times t_i at which each node captures the token; ignore propagation delays. Explain any discrepancy between your solution and the upper bound of Eq. (4.81).
- 4.34** (a) Consider an FDDI ring with m nodes, with target token rotation time τ and high-priority allocations $\alpha_0, \dots, \alpha_{m-1}$. Assume that every node has an arbitrarily large backlog of both high-priority and low-priority traffic. In the limit of long-term operation, find the fraction of the transmitted traffic that goes to each node for each priority. Ignore propagation and processing delay and use the fact that the bound in Eq. (4.81) is met with equality given the assumed initial conditions and given arbitrarily large backlogs. Assume that $T = \alpha_0 + \alpha_1 + \dots + \alpha_{m-1}$ is strictly less than τ .
- (b) Now assume that each node k can fill only a fraction α_k/τ of the ring with high-priority traffic and find the fractions in part (a) again.
- 4.35** Give the rules that the two counters implementing the virtual queue in DQDB must follow. Assume there is also a binary variable F that is one when a frame is in the virtual queue. Assume that when $F = 0$, the first counter is kept at value 0.



Routing in Data Networks

5.1 INTRODUCTION

We have frequently referred to the routing algorithm as the network layer protocol that guides packets through the communication subnet to their correct destination. The times at which routing decisions are made depend on whether the network uses datagrams or virtual circuits. In a datagram network, two successive packets of the same user pair may travel along different routes, and a routing decision is necessary for each individual packet (see Fig. 5.1). In a virtual circuit network, a routing decision is made when each virtual circuit is set up. The routing algorithm is used to choose the communication path for the virtual circuit. All packets of the virtual circuit subsequently use this path up to the time that the virtual circuit is either terminated or rerouted for some reason (see Fig. 5.2).

Routing in a network typically involves a rather complex collection of algorithms that work more or less independently and yet support each other by exchanging services or information. The complexity is due to a number of reasons. First, routing requires coordination between all the nodes of the subnet rather than just a pair of modules as, for example, in data link and transport layer protocols. Second, the routing system must

cope with link and node failures, requiring redirection of traffic and an update of the databases maintained by the system. Third, to achieve high performance, the routing algorithm may need to modify its routes when some areas within the network become congested.

The main emphasis will be on two aspects of the routing problem. The first has to do with selecting routes to achieve high performance. In Sections 5.2.3 to 5.2.5, we discuss algorithms based on shortest paths that are commonly used in practice. In Sections 5.5, 5.6, and 5.7 we describe sophisticated routing algorithms that try to achieve near optimal performance. The second aspect of routing that we will emphasize is broadcasting routing-related information (including link and node failures and repairs) to all network nodes. This issue and the subtleties associated with it are examined in Section 5.3.

The introductory sections set the stage for the main development. The remainder of this section explains in nonmathematical terms the main objectives in the routing problem and provides an overview of current routing practice. Sections 5.2.1 to 5.2.3 present some of the main notions and results of graph theory, principally in connection with shortest paths and minimum weight spanning trees. Section 5.4 uses the material on graph theory to describe methods for topological design of networks. Finally, Section 5.8 reviews the routing system of the Codex network and its relation to the optimal routing algorithm of Section 5.7.

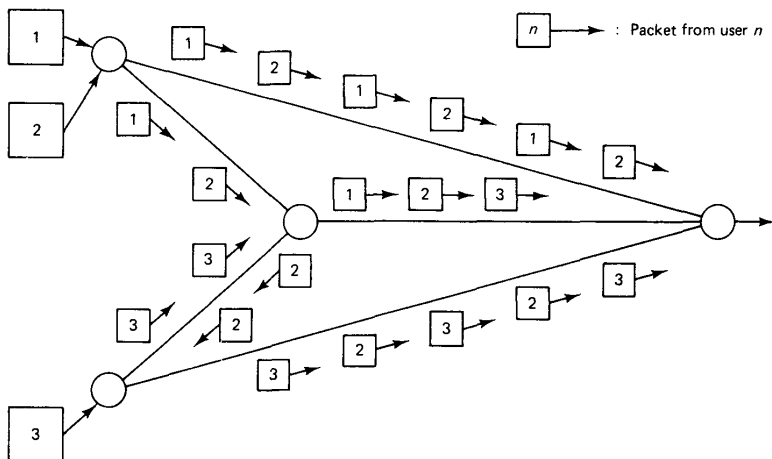


Figure 5.1 Routing in a datagram network. Two packets of the same user pair can travel along different routes. A routing decision is required for each individual packet.

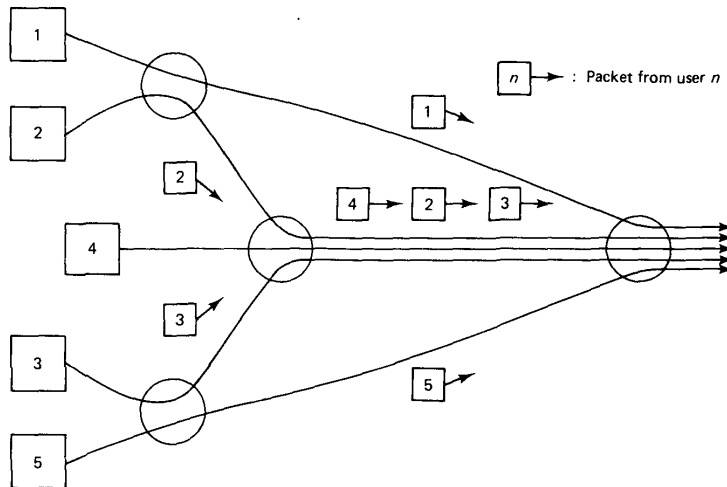


Figure 5.2 Routing in a virtual circuit network. All packets of each virtual circuit use the same path. A routing decision is required only when a virtual circuit is set up.

5.1.1 Main Issues in Routing

The two main functions performed by a routing algorithm are the selection of routes for various origin–destination pairs and the delivery of messages to their correct destination once the routes are selected. The second function is conceptually straightforward using a variety of protocols and data structures (known as routing tables), some of which will be described in the context of practical networks in Section 5.1.2. The focus will be on the first function (selection of routes) and how it affects network performance.

There are two main performance measures that are substantially affected by the routing algorithm—*throughput* (quantity of service) and *average packet delay* (quality of service). Routing interacts with flow control in determining these performance measures

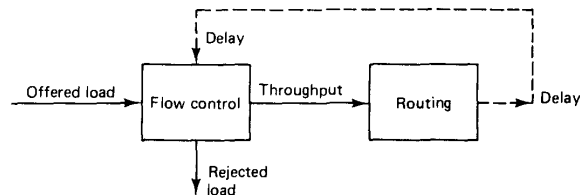


Figure 5.3 Interaction of routing and flow control. As good routing keeps delay low, flow control allows more traffic into the network.

by means of a feedback mechanism shown in Fig. 5.3. When the traffic load offered by the external sites to the subnet is relatively low, it will be fully accepted into the network, that is,

$$\text{throughput} = \text{offered load}$$

When the offered load is excessive, a portion will be rejected by the flow control algorithm and

$$\text{throughput} = \text{offered load} - \text{rejected load}$$

The traffic accepted into the network will experience an average delay per packet that will depend on the routes chosen by the routing algorithm. However, throughput will also be greatly affected (if only indirectly) by the routing algorithm because typical flow control schemes operate on the basis of striking a balance between throughput and delay (*i.e.*, they start rejecting offered load when delay starts getting excessive). Therefore, *as the routing algorithm is more successful in keeping delay low, the flow control algorithm allows more traffic into the network*. While the precise balance between delay and throughput will be determined by flow control, the effect of good routing under high offered load conditions is to realize a more favorable delay-throughput curve along which flow control operates, as shown in Fig. 5.4.

The following examples illustrate the discussion above:

Example 5.1

In the network of Fig. 5.5, all links have capacity 10 units. (The units by which link capacity and traffic load is measured is immaterial and is left unspecified.) There is a single destination (node 6) and two origins (nodes 1 and 2). The offered load from each of nodes 1 and 2 to node 6 is 5 units. Here, the offered load is light and can easily be accommodated with small delay by routing along the leftmost and rightmost paths, $1 \rightarrow 3 \rightarrow 6$ and $2 \rightarrow 5 \rightarrow 6$, respectively. If instead, however, the routes $1 \rightarrow 4 \rightarrow 6$ and $2 \rightarrow 4 \rightarrow 6$ are used, the flow on link (4,6) will equal capacity, resulting in very large delays.

Example 5.2

For the same network, assume that the offered loads at nodes 1 and 2 are 5 and 15 units, respectively (see Fig. 5.6). If routing from node 2 to the destination is done along a single path, then at least 5 units of offered load will have to be rejected since all path capacities

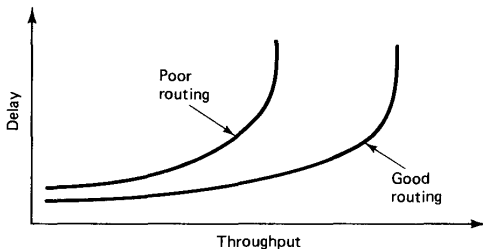


Figure 5.4 Delay-throughput operating curves for good and bad routing.

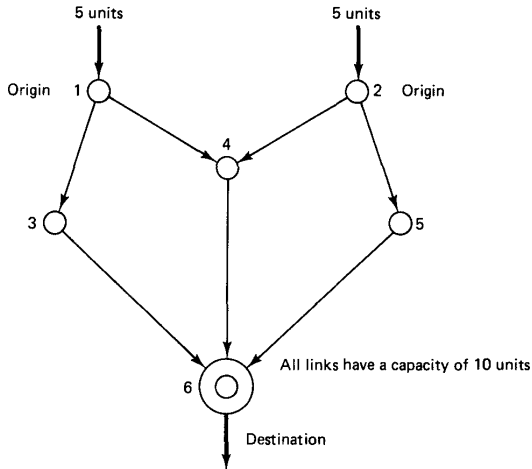


Figure 5.5 Network for Example 5.1. All links have a capacity of 10 units. If all traffic is routed through the middle link (4,6), congestion occurs. If, instead, paths (1 → 3 → 6) and (2 → 5 → 6) are used, the average delay is small.

equal 10. Thus, the total throughput can be no more than 15 units. On the other hand, suppose that the traffic originating at node 2 is evenly split between the two paths $2 \rightarrow 4 \rightarrow 6$ and $2 \rightarrow 5 \rightarrow 6$, while the traffic originating at node 1 is routed along $1 \rightarrow 3 \rightarrow 6$. Then, the traffic arrival rate on each link will not exceed 75% of capacity, the delay per packet will be reasonably small, and (given a good flow control scheme) no portion of the offered load will be rejected. Arguing similarly, it is seen that when the offered loads at nodes 1 and 2 are both large, the maximum total throughput that this network can accommodate is between 10 and 30 units, depending on the routing scheme. This example also illustrates that to achieve high throughput, the traffic of some origin–destination pairs may have to be divided among more than one route.

In conclusion, *the effect of good routing is to increase throughput for the same value of average delay per packet under high offered load conditions and to decrease average delay per packet under low and moderate offered load conditions*. Furthermore, it is evident that the routing algorithm should be operated so as to keep average delay per packet as low as possible for any given level of offered load. While this is easier said than done, it provides a clear-cut objective that can be expressed mathematically and dealt with analytically.

5.1.2 Wide Area Network Routing: An Overview

The purpose of this section is to survey current routing practice in wide area networks, to introduce classifications of different schemes, and to provide a context for the analysis presented later.

There are a number of ways to classify routing algorithms. One way is to divide them into *centralized* and *distributed*. In centralized algorithms, all route choices are made at a central node, while in distributed algorithms, the computation of routes is

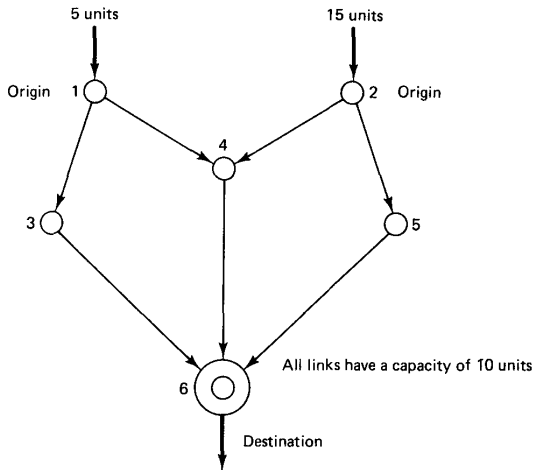


Figure 5.6 Network for Example 5.2. All links have a capacity of 10 units. The input traffic can be accommodated with multiple-path routing, but at least 5 units of traffic must be rejected if a single-path routing is used.

shared among the network nodes with information exchanged between them as necessary. Note that this classification relates mostly to the implementation of an algorithm, and that a centralized and a distributed routing algorithm may be equivalent at some level of mathematical abstraction.

Another classification of routing algorithms relates to whether they change routes in response to the traffic input patterns. In *static* routing algorithms, the path used by the sessions of each origin–destination pair is fixed regardless of traffic conditions. It can only change in response to a link or node failure. This type of algorithm cannot achieve a high throughput under a broad variety of traffic input patterns. It is recommended for either very simple networks or for networks where efficiency is not essential. Most major packet networks use some form of *adaptive* routing where the paths used to route new traffic between origins and destinations change occasionally in response to congestion. The idea here is that congestion can build up in some part of the network due to changes in the statistics of the input traffic load. Then, the routing algorithm should try to change its routes and guide traffic around the point of congestion.

There are many routing algorithms in use with different levels of sophistication and efficiency. This variety is partly due to historical reasons and partly due to the diversity of needs in different networks. In this section we provide a nonmathematical description of some routing techniques that are commonly used in practice. We illustrate these techniques in terms of the routing algorithms of three wide area networks (ARPANET, TYMNET, and SNA). The routing algorithm of another wide area network, the Codex network, will be described in Section 5.8, because this algorithm is better understood after studying optimal routing in Sections 5.5 to 5.7.

Flooding and broadcasting During operation of a data network, it is often necessary to broadcast some information, that is, to send this information from an origin

node to all other nodes. An example is when there are changes in the network topology due to link failures and repairs, and these changes must be transmitted to the entire network. Broadcasting could also be used as a primitive form of routing packets from a single transmitter to a single receiver or, more generally, to a subset of receivers; this use is generally rather inefficient, but may be sensible because it is simple or because the receivers' locations within the network are unknown.

A widely used broadcasting method, known as *flooding*, operates as follows. The origin node sends its information in the form of a packet to its neighbors (the nodes to which it is directly connected with a link). The neighbors relay it to their neighbors, and so on, until the packet reaches all nodes in the network. Two additional rules are also observed, which limit the number of packet transmissions. First, a node will not relay the packet back to the node from which the packet was obtained. Second, a node will transmit the packet to its neighbors at most once; this can be ensured by including on the packet the ID number of the origin node and a sequence number, which is incremented with each new packet issued by the origin node. By storing the highest sequence number received for each origin node, and by not relaying packets with sequence numbers that are less than or equal to the one stored, a node can avoid transmitting the same packet more than once on each of its incident links. Note that with these rules, links need not preserve the order of packet transmissions; the sequence numbers can be used to recognize the correct order. Figure 5.7(a) gives an example of flooding and illustrates that the total number of packet transmissions per packet broadcast lies between L and $2L$, where L is the number of bidirectional links of the network. We note also that one can implement a flooding-like algorithm without using sequence numbers. This possibility is described in Section 5.3, where flooding and the topology broadcast problem are discussed in more detail.

Another broadcasting method, based on the use of a *spanning tree*, is illustrated in Fig. 5.7(b). A spanning tree is a connected subgraph of the network that includes all nodes and has no cycles (see a more detailed discussion in the next section). Broadcasting on a spanning tree is more communication-efficient than flooding. It requires a total of only $N - 1$ packet transmissions per packet broadcast, where N is the number of nodes.

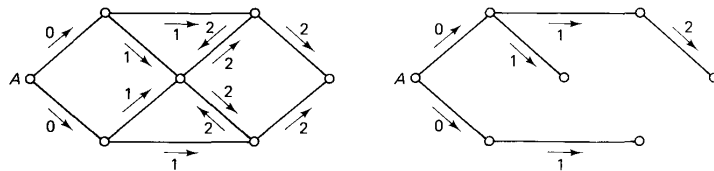


Figure 5.7 Packet broadcasting from node A to all other nodes by using flooding [as in (a)] or a spanning tree [as in (b)]. Arrows indicate packet transmissions at the times shown. Each packet transmission time is assumed to be one unit. Flooding requires at least as many packet transmissions as the spanning tree method and usually many more. In this example, the time required for the broadcast packet to reach all nodes is the same for the two methods. In general, however, depending on the choice of spanning tree, the time required with flooding may be less than with the spanning tree method.

The price for this saving is the need to maintain and update the spanning tree in the face of topological changes.

We note two more uses of spanning trees in broadcasting and routing. Given a spanning tree rooted at the origin node of the broadcast, it is possible to implement flooding as well as routing without the use of sequence numbers. The method is illustrated in Fig. 5.8. Note that flooding can be used to construct a spanning tree rooted at a node, as shown in Fig. 5.8. Also given a spanning tree, one can perform selective broadcasting, that is, packet transmission from a single origin to a limited set of destinations. For this it is necessary that each node knows which of its incident spanning tree links leads to any given destination (see Fig. 5.9). The spanning tree can also be used for routing packets with a single destination and this leads to an important method for bridged local area networks; see the next subsection.

Shortest path routing. Many practical routing algorithms are based on the notion of a *shortest path* between two nodes. Here, each communication link is assigned a positive number called its *length*. A link can have a different length in each direction. Each path (*i.e.*, a sequence of links) between two nodes has a length equal to the sum of the lengths of its links. (See Section 5.2 for more details.) A shortest path routing algorithm routes each packet along a minimum length (or shortest) path between the origin and destination nodes of the packet. The simplest possibility is for each link to have unit length, in which case a shortest path is simply a path with minimum number of links (also called a *min-hop path*). More generally, the length of a link may depend on its transmission capacity and its projected traffic load. The idea here is that a shortest

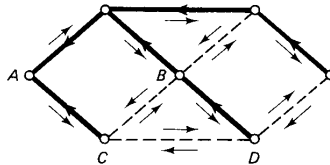


Figure 5.8 Illustration of a method for broadcasting using a tree rooted at the broadcast's origin node *A*. Each node knows its unique predecessor (or parent) on the tree, but need not know its successors. The tree can be used for routing packets to *A* from the other nodes using the paths of the tree in the reverse direction. It can also be used for flooding packets from *A*. The flooding rule for a node other than *A* is the following: a packet received from the parent is broadcast to all neighbors except the parent; all other packets are ignored. Thus in the figure, node *D* broadcasts the packet received from its parent *B* but ignores the packet received from *C*. Since only the packets transmitted on the spanning tree in the direction pointing away from the root are relayed further, there is no indefinite circulation of copies, and therefore, no need for sequence numbers.

Flooding can also be used to construct the tree rooted at *A*. Node *A* starts the process by sending a packet to all its neighbors; the neighbors must send the packet to their neighbors, and so on. All nodes should mark the transmitter of the first packet they receive as their parent on the tree. The nodes should relay the packet to their neighbors only once (after they first receive the packet from their parent); all subsequent packet receptions should be ignored.

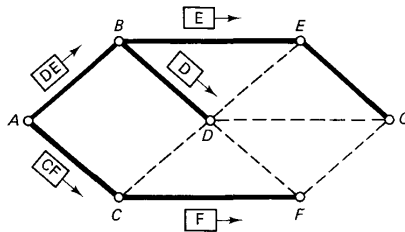


Figure 5.9 Illustration of selective broadcasting using a spanning tree. We assume that each node knows which of the incident spanning tree links lead to any given destination. In this example, node *A* wishes to broadcast a packet to nodes *C*, *D*, *E*, and *F*. Node *A* sends a copy of the packet to each of its incident links that lead to intended destinations, and each copy carries in its header the ID numbers of its intended destinations. Upon reception of a copy, a node strips the ID number from the header (if it is an intended destination) and repeats the process. This eliminates some unnecessary transmissions, for example the transmission on link *EG* in the figure.

path should contain relatively few and uncongested links, and therefore be desirable for routing.

A more sophisticated alternative is to allow the length of each link to change over time and to depend on the prevailing congestion level of the link. Then a shortest path may adapt to temporary overloads and route packets around points of congestion. This idea is simple but also contains some hidden pitfalls, because by making link lengths dependent on congestion, we introduce a feedback effect between the routing algorithm and the traffic pattern within the network. It will be seen in Section 5.2.5 that this can result in undesirable oscillations. However, we will ignore this possibility for the time being.

An important distributed algorithm for calculating shortest paths to a given destination, known as the *Bellman–Ford method*, has the form

$$D_i := \min_j [d_{ij} + D_j]$$

where D_i is the estimated shortest distance of node i to the destination and d_{ij} is the length of link (i, j) . Each node i executes periodically this iteration with the minimum taken over all of its neighbors j . Thus $d_{ij} + D_j$ may be viewed as the estimate of shortest distance from node i to the destination subject to the constraint of going through j , and $\min_j [d_{ij} + D_j]$ may be viewed as the estimate of shortest distance from i to the destination going through the “best” neighbor.

We discuss the Bellman–Ford algorithm in great detail in the next section, where we show that it terminates in a finite number of steps with the correct shortest distances under reasonable assumptions. In practice, the Bellman–Ford iteration can be implemented as an iterative process, that is, as a sequence of communications of the current value of D_j of nodes j to all their neighbors, followed by execution of the shortest distance estimate updates $D_i := \min_j [d_{ij} + D_j]$. A remarkable fact is that this process is very flexible with respect to the choice of initial estimates D_j and the ordering of communications and updates; it works correctly, finding the shortest distances in a finite number of steps, for an essentially arbitrary choice of initial conditions and for an arbitrary order of communications and updates. This allows an asynchronous, real-time distributed implementation of the Bellman–Ford method, which can tolerate changes of the link lengths as the algorithm executes (see Section 5.2.4).

Optimal routing. Shortest path routing has two drawbacks. First, it uses only one path per origin–destination pair, thereby potentially limiting the throughput of the network; see the examples of the preceding subsection. Second, its capability to adapt to changing traffic conditions is limited by its susceptibility to oscillations; this is due to the abrupt traffic shifts resulting when some of the shortest paths change due to changes in link lengths. Optimal routing, based on the optimization of an average delay-like measure of performance, can eliminate both of these disadvantages by splitting any origin–destination pair traffic at strategic points, and by shifting traffic gradually between alternative paths. The corresponding methodology is based on the sophisticated mathematical theory of optimal multicommodity flows, and is discussed in detail in Sections 5.4 to 5.7. Its application to the Codex network is described in Section 5.8.

Hot potato (deflection) routing schemes. In networks where storage space at each node is limited, it may be important to modify the routing algorithm so as to minimize buffer overflow and the attendant loss of packets. The idea here is for nodes to get rid of their stored packets as quickly as possible, transmitting them on whatever link happens to be idle—not necessarily one that brings them closer to their destination.

To provide an example of such a scheme, let us assume that all links of the communication network can be used simultaneously and in both directions, and that each packet carries a destination address and requires unit transmission time on every link. Assume also that packets are transmitted in slots of unit time duration, and that slots are synchronized so that their start and end are simultaneous at all links. In a typical routing scheme, each node, upon reception of a packet destined for a different node, uses table lookup to determine the next link on which the packet should be transmitted; we refer to this link as the *assigned link of the packet*. It is possible that more than one packet with the same assigned link is received by a node during a slot; then at most one of these packets can be transmitted by the node in the subsequent slot, and the remaining packets must be stored by the node in a queue. The storage requirement can be eliminated by modifying the routing scheme so that all these packets are transmitted in the next slot; one of them is transmitted on its assigned link, and the others are transmitted on some other links chosen, possibly at random, from the set of links that are not assigned to any packet received in the previous slot. It can be seen that with this modification, at any node with d incident links, there can be at most d packets received in any one slot, and out of these packets, the ones that are transient (are not destined for the node) will be transmitted along some link (not necessarily their assigned one) in the next slot. Therefore, assuming that at most $d - k$ new packets are generated at a node in a slot where k transient packets are to be transmitted, there will be no queueing, and the storage space at the node need not exceed $2d$ packets. A scheme of this type is used in the Connection Machine, a massively parallel computing system [Hil85]. A variation is obtained when storage space for more than $2d$ packets is provided, and the transmission of packets on links other than the ones assigned to them is allowed only when the available storage space falls below a certain threshold.

The type of method just described was suggested in [Bar64] under the name *hot potato routing*. It has also been known more recently as *deflection routing*. Its drawback is that successive packets of the same origin–destination pair may be received out of

order, and some packets may travel on long routes to their destinations; indeed, one should take precautions to ensure that a packet cannot travel on a cycle indefinitely. For some recent analyses of deflection routing, see [BrC91a], [BrC91b], [HaC90], [Haj91], [GrH89], [Max87], [Syz90], [Var90].

Cut-through routing. We have been implicitly assuming thus far that a packet must be fully received at a node before that node starts relaying it to another node. There is, however, an incentive to split a long message into several smaller packets in order to reduce the message's delay on multiple link paths, taking advantage of pipelining (see the discussion in Section 2.5.5). The idea of message splitting, when carried to its extreme, leads to a transmission method called *cut-through routing*, whereby a node can start relaying any portion of a packet to another node without waiting to receive the packet in its entirety. In the absence of additional traffic on the path, the delay of the packet is equal to the transmission time on the slowest link of the path plus the propagation delay, regardless of the number of links on the path. Thus, delay can be reduced by as much as a factor of n on a path with n links.

Note that the nature of cut-through routing is such that error detection and retransmission cannot be done on a link-by-link basis; it must be done on an end-to-end basis. Another issue is that pieces of the same packet may simultaneously be traveling on different links while other pieces are stored at different nodes. To keep the pieces of the packet together, one imposes the requirement that once a packet starts getting transmitted on a link, the link is reserved until all bits of the packet are transmitted. The reservation is made when the link transmits the packet's first bit, and it is released when the link transmits the packet's last bit. Thus, at any time a packet is reserving a portion of its path consisting of several links as shown in Fig. 5.10. Portions of the packet may be stored simultaneously at several nodes either because two or more links on the path have unequal transmission capacities, or because the packet's header had to wait in queue for a link to become available. To avoid the need for complex error recovery procedures, it is generally advisable to ensure that when the packet's first bit is transmitted by a link,

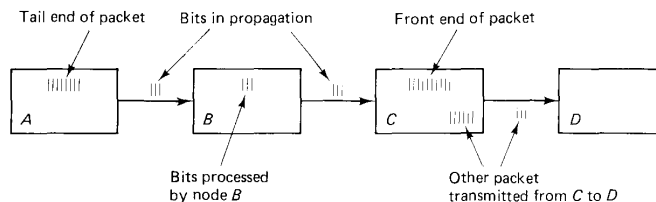


Figure 5.10 Illustration of cut-through routing. Here a packet originates at node A and is destined for node D , after going through nodes B and C . The header of the packet traveled to node B , found the link BC free, and continued to node C without waiting for the rest of the packet to arrive at B . Upon arrival to C , the header found that the link CD was in use, so it was forced to wait at node C . The header reserves links as it travels through them. A link reservation is relinquished only after the last bit of the packet goes through the link, so a packet stays together even though portions of it may be residing in several different nodes.

there is enough buffer space to store the entire packet at the other end of the link. Note that cut-through routing encounters a serious problem if a slow link is followed by a fast link along a packet's path. Then the fast link must effectively be slowed down to the speed of the slow link, and the associated idle fill issues may be hard to resolve.

A variation of cut-through routing, motivated by storage concerns, drops a packet once its first bit encounters a busy link along its path. Thus a packet is either successful in crossing its path without any queueing delay, or else it is discarded and must be retransmitted in its entirety, similar to circuit switching. Indeed, with some thought, it can be seen that this version of cut-through routing is equivalent to a form of circuit switching; the circuit's path is set up by the first bit of the packet, and once the circuit is established, its duration is equal to the packet's transmission time from origin to destination.

ARPANET: An example of datagram routing. The routing algorithm of the ARPANET, first implemented in 1969, has played an important historical role in the development of routing techniques. This was an ambitious, distributed, adaptive algorithm that stimulated considerable research on routing and distributed computation in general. On the other hand, the algorithm had some fundamental flaws that were finally corrected in 1979 when it was replaced by a new version. By that time, however, the original algorithm had been adopted by several other networks.

Shortest path routing is used in both ARPANET algorithms. The length of each link is defined by some measure of traffic congestion on the link, and is updated periodically. Thus, the ARPANET algorithms are adaptive and, since the ARPANET uses datagrams, two successive packets of the same session may follow different routes. This has two undesirable effects. First, packets can arrive at their destination out of order and must, therefore, be put back in order upon arrival. (This can also happen because of the data link protocol of the ARPANET, which uses eight logical channels; see the discussion in Chapter 2.) Second, the ARPANET algorithms are prone to oscillations. This phenomenon is explained in some detail in Section 5.2.5, but basically the idea is that selecting routes through one area of the network increases the lengths of the corresponding links. As a result, at the next routing update the algorithm tends to select routes through different areas. This makes the first area desirable at the subsequent routing update with an oscillation resulting. The feedback effect between link lengths and routing updates was a primary flaw of the original ARPANET algorithm that caused difficulties over several years and eventually led to its replacement. The latest algorithm is also prone to oscillations, but not nearly as much as the first (see Section 5.2.5).

In the original ARPANET algorithm, neighboring nodes exchanged their estimated shortest distances to each destination every 625 msec. The algorithm for updating the shortest distance estimate D_i of node i to a given destination is based on the Bellman–Ford method

$$d_i := \min_j [d_{ij} + D_j]$$

outlined previously and discussed further in Sections 5.2.3 and 5.2.4. Each link length d_{ij} was made dependent on the number of packets waiting in the queue of link (i, j) at the time of the update. Thus, link lengths were changing very rapidly, reflecting statistical traffic fluctuations as well as the effect of routing updates. To stabilize oscillations, a

large positive constant was added to the link lengths. Unfortunately, this reduced the sensitivity of the algorithm to traffic congestion.

In the second version of the ARPANET algorithm [MRR80], known as *Shortest Path First* (or SPF for short), the length of each link is calculated by keeping track of the delay of each packet in crossing the link. Each link length is updated periodically every 10 sec, and is taken to be the average packet delay on the link during the preceding 10-sec period. The delay of a packet on a link is defined as the time between the packet's arrival at the link's start node and the time the packet is correctly delivered to the link's end node (propagation delay is also included). Each node monitors the lengths of its outgoing links and broadcasts these lengths throughout the network at least once every 60 sec by using a flooding algorithm (see Section 5.3.1). Upon reception of a new link length, each node recalculates a shortest path from itself to each destination, using an incremental form of Dijkstra's algorithm (see Section 5.2.3). The algorithm is asynchronous in nature in that link length update messages are neither transmitted nor received simultaneously at all nodes. It turns out that this tends to improve the stability properties of the algorithm (see Section 5.2.5). Despite this fact, with increased traffic load, oscillations became serious enough to warrant a revision of the method for calculating link lengths. In the current method, implemented in 1987, the range of possible link lengths and the rate at which these lengths can change have been drastically reduced, with a substantial improvement of the algorithm's dynamic behavior ([KhZ89] and [ZVK89]).

Note that a node is not concerned with calculating routing paths that originate at other nodes even though the information available at each node (network topology plus lengths of all links) is sufficient to compute a shortest path from every origin to every destination. In fact, the routing tables of an ARPANET node contain only the first outgoing link of the shortest path from the node to each destination (see Fig. 5.11). When the node receives a new packet, it checks the packet's destination, consults its routing table, and places the packet in the corresponding outgoing link queue. If all nodes visited by a packet have calculated their routing tables on the basis of the same link length information, it is easily seen that the packet will travel along a shortest path. It is actually possible that a packet will occasionally travel in a loop because of changes in routing tables as the packet is in transit. However, looping of this type is rare, and when it happens, it is unlikely to persist.

The ARPANET routing tables can also be used conveniently for broadcasting a packet to a selected set of destinations. Multiple copies of the packet are transmitted along shortest paths to the destinations, with each copy guided by a header that includes a subset of the destinations. However, care is taken to avoid unnecessary proliferation of the copies. In particular, let D be the set of destinations and for every link l incident to the packet's origin, let D_l be the subset of destinations for which l is the first link on the shortest path from the origin (the origin knows D_l by looking at its routing tables). The origin starts the broadcast by transmitting a copy of the packet on each link l for which D_l is nonempty and includes the destination set D_l in the copy's header. Each node s that subsequently receives a copy of the packet repeats this process with D replaced by the set of remaining destinations (the set of destinations in the copy's header other than s ; if s is the only destination on the copy, the node does not relay it further, as shown

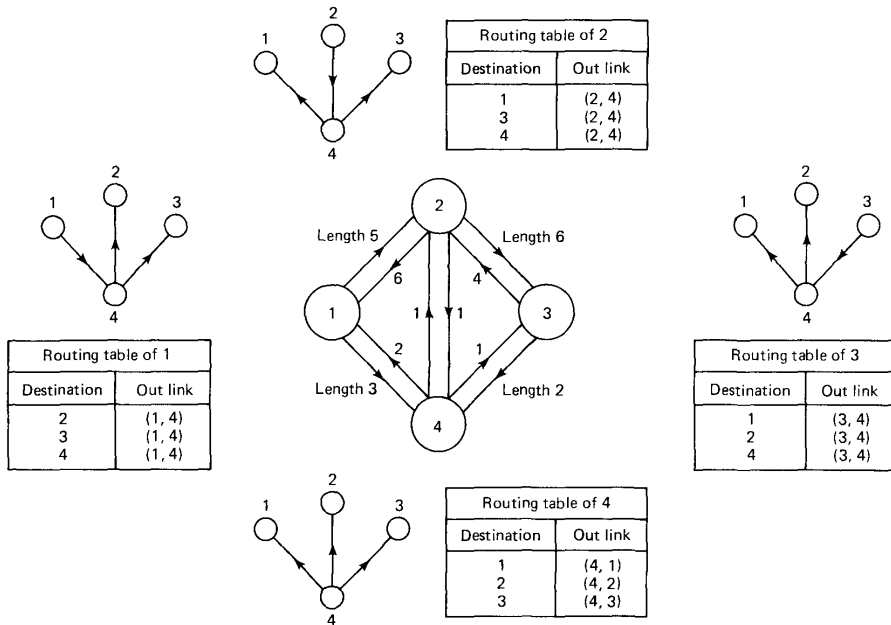


Figure 5.11 Routing tables maintained by the nodes in the ARPANET algorithm. Each node calculates a shortest path from itself to each destination, and enters the first link on that path in its routing table.

in Fig. 5.12). It can be seen with a little thought that if the routing tables of the nodes are consistent in the sense that they are based on the same link length information, the packet will be broadcast along the shortest path tree to all its destinations and with a minimal number of transmissions.

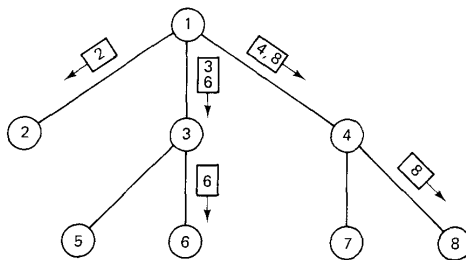


Figure 5.12 Broadcasting a packet to a selected set of destinations using the ARPANET routing tables. In this example, the broadcast is from node 1 to nodes 2, 3, 4, 6, and 8. The figure shows the links on which a copy of the packet is transmitted, the header of each copy, and the shortest path tree from node 1 to all destinations (each node actually maintains only its outgoing links on the tree). The number of transmissions is five. If a separate copy were to be transmitted to each destination, the number of transmissions would be seven.

TYMNET: An example of virtual circuit routing. The TYMNET routing algorithm, originally implemented in 1971, is based on the shortest path method, and is adaptive like the ARPANET algorithms. However, the implementation of the adaptive shortest path idea is very different in the two networks.

In the TYMNET, the routing algorithm is centralized and is operated at a special node called the supervisor. TYMNET uses virtual circuits, so a routing decision is needed only at the time a virtual circuit is set up. A virtual circuit request received by the supervisor is assigned to a shortest path connecting the origin and destination nodes of the virtual circuit. The supervisor maintains a list of current link lengths and does all the shortest path calculations. The length of each link depends on the load carried by the link as well as other factors, such as the type of the link and the type of virtual circuit being routed [Tym81]. While the algorithm can be classified as adaptive, it does not have the potential for oscillations of the ARPANET algorithms. This is due primarily to the use of virtual circuits rather than datagrams, as explained in Section 5.2.5.

Once the supervisor has decided upon the path to be used by a virtual circuit, it informs the nodes on the path and the necessary information is entered in each node's routing tables. These tables are structured as shown in Fig. 5.13. Basically, a virtual circuit is assigned a channel (or port) number on each link it crosses. The routing table at each node matches each virtual circuit's channel number on the incoming link with a channel number on the outgoing link. In the original version of TYMNET (now called TYMNET I), the supervisor maintains an image of the routing tables of all nodes and explicitly reads and writes the tables in the nodes. In the current version (TYMNET II), the nodes maintain their own tables. The supervisor establishes a new virtual circuit by sending a "needle" packet to the origin node. The needle contains routing information and threads its way through the network, building the circuit as it goes, with the user data following behind it.

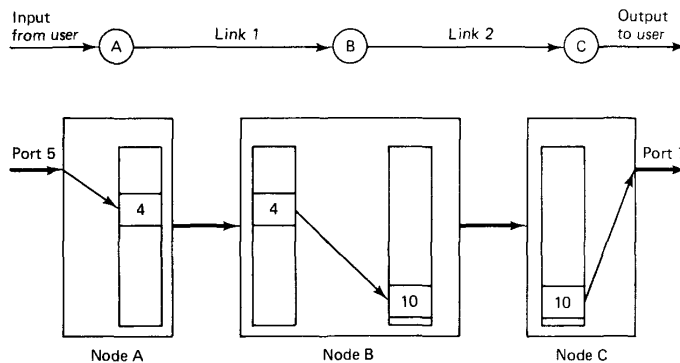


Figure 5.13 Structure of TYMNET routing tables. For the virtual circuit on the path ABC, the routing table at node A maps input port 5 onto channel 4 on link 1. At node B, incoming channel 4 is mapped into outgoing channel 10 on link 2. At node C, incoming channel 10 is mapped into output port 7.

The routing information includes the node sequence of the routing path, and some flags indicating circuit class. When a needle enters a TYMNET II node, its contents are checked. If the circuit terminates at this node, it is assigned to the port connected with the appropriate external site. Otherwise, the next node on the route is checked. If it is unknown (because of a recent link failure or some other error), the circuit is zapped back to its origin. Otherwise, the routing tables are updated and the needle is passed to the next node.

The TYMNET algorithm is well thought out and has performed well over the years. Several of its ideas were implemented in the Codex network, which is a similarly structured network (see Section 5.8). The Codex algorithm, however, is distributed and more sophisticated. A potential concern with the TYMNET algorithm is its vulnerability to failure of the supervisor node. In TYMNET, this is handled by using backup supervisor nodes. A distributed alternative, used in the Codex network, is to broadcast all link lengths to all nodes (as in the ARPANET), and to let each node choose the routing path for each circuit originating at that node. This path may be established by updating the routing tables of the nodes on the path, just as in TYMNET. Another possibility is to let the originating node include the routing path on each packet. This makes the packets longer but may save substantially in processing time at the nodes, when special switching hardware are used. For high-speed networks where the packet volume handled by the nodes is very large, it is essential to use such special hardware and to trade communication efficiency for reduced computation.

Routing in SNA. Routing in IBM's SNA is somewhat unusual in that the method for choosing routes is left partially to the user. SNA provides instead a framework within which a number of routing algorithm choices can be made. Specifically, for every origin-destination pair, SNA provides several paths along which virtual circuits can be built. The rule for assignment of virtual circuits to paths is subject to certain restrictions but is otherwise left unspecified. The preceding oversimplified description is couched in the general terminology of paths and virtual circuits used in this book. SNA uses a different terminology, which we now review.

The architecture of SNA does not fully conform to the OSI seven-layer architecture used as our primary model. The closest counterpart in SNA of the OSI network layer, called the *path control layer*, provides virtual circuit service to the immediately higher layer, called the *transmission control layer*. The path control layer has three functions: *transmission group control*, *explicit route control*, and *virtual route control*. The first fits most appropriately in the data link control layer of the OSI model, whereas the second and third correspond to the routing and flow control functions, respectively, of the network layer in the OSI model.

A transmission group in SNA terminology is a set of one or more physical communication lines between two neighboring nodes of the network. The transmission group control function makes the transmission group appear to higher layers as a single physical link. There may be more than one transmission group connecting a pair of nodes. The protocol places incoming packets on a queue, sends them out on the next available physical link, and sequences them if they arrive out of order. Resequencing of packets is done at every node, unlike the ARPANET, which resequences only at the destination node.

An explicit route in SNA terminology is what we have been referring to as a path within a subnet. Thus, an explicit route is simply a sequence of transmission groups providing a physical path between an origin and a destination node. There are several explicit routes provided (up to eight in SNA 4.2) for each origin–destination pair, which can be modified only under supervisory control. The method for choosing these explicit routes is left to the network manager. Each node stores the next node for each explicit route in its routing tables, and each packet carries an explicit route number. Therefore, when a packet arrives at a node, the next node on the packet's path (if any) is determined by a simple table lookup.

A virtual route in SNA terminology is essentially what we have been calling a virtual circuit. A user pair conversation (or session) uses only one virtual route. However, a virtual route can carry multiple sessions. Each virtual route belongs to a priority (or service) class. Each transmission group can serve up to three service classes, each with up to eight virtual routes, for a maximum of 24 virtual routes. When a user requests a session through a subnet entry node, a service class is specified on the basis of which the entry node will attempt to establish the session over one of its virtual routes. If all the communication lines of a transmission group fail, every session using that transmission group must be rerouted. If no alternative virtual route can be found, the session is aborted. When new transmission groups are established, all nodes are notified via special control packets, so each node has a copy of the network topology and knows which explicit routes are available and which are not.

Routing in circuit switching networks. Routing in circuit switching networks is conceptually not much different than routing in virtual circuit networks. In both cases a path is established at the time the circuit is set up, and all subsequent communication follows that path. In both cases the circuit will be blocked when the number of existing circuits on some link of the path reaches a certain threshold, although the choice of threshold may be based on different considerations in each case.

Despite these similarities, in practice, telephone networks have used quite different routing methods than virtual circuit networks. A common routing method in circuit switching networks, known as *hierarchical routing*, has been to use a fixed first-choice route, and to provide for one or more alternative routes, to be tried in a hierarchical order, for the case where the first-choice route is blocked. However, the differences in routing philosophy between circuit switching and virtual circuit networks have narrowed recently. In particular, there have been implementations of adaptive routing in telephone networks, whereby the path used by a call is chosen from several alternative paths on the basis of a measure of congestion (*e.g.*, the number of circuits these paths carry). We refer to [Ash90], [KeC90], [Kri90], [ReC90], and [WaO90] for detailed descriptions and discussion of such implementations.

5.1.3 Interconnected Network Routing: An Overview

As networks started to proliferate, it became necessary to interconnect them using various interface devices. In the case of wide-area networks, the interfaces are called *gateways*,

and usually perform fairly sophisticated network layer tasks, including routing. Gateways typically operate at the internet sublayer of the network layer. A number of engineering and economic factors, to be discussed shortly, motivated the interconnection of local area networks (LANs) at the MAC sublayer. The devices used for interconnection, known as *bridges*, perform a primitive form of routing. LANs can also be connected with each other or with wide-area networks using more sophisticated devices called *routers*. These provide fairly advanced routing functions at the network layer, possibly including adaptive and multiple-path routing. In this section we survey various routing schemes for interconnected networks, with a special emphasis on bridged LANs.

Conceptually, one may view an interconnected network of subnetworks as a single large network, where the interfaces (gateways, bridges, or routers) are additional nodes, each connected to two or more of the constituent subnetworks (see Fig. 5.14). Routing based on this conceptual model is called *nonhierarchical* and is similar to routing in a wide area network.

It is also possible to adopt a coarser-grain view of an interconnected network of subnetworks, whereby each subnetwork is considered to be a single node connected to interface nodes. Routing based on this model is called *hierarchical*, and consists of two levels. At the lower level, there is a separate routing algorithm for each subnetwork that handles local subnetwork traffic and also directs internetwork traffic to the subnetwork's interfaces with other subnetworks. At the higher level, there is a routing algorithm that determines the sequence of subnetworks and interface nodes that each internetwork packet must follow (see Fig. 5.15). Basically, the low-level algorithms handle "local" packets, and the high-level algorithm delivers "long-distance" packets to the appropriate low-level algorithm. An example of a hierarchical network is the Internet discussed in Section 2.8.4. Hierarchical routing is generally recommended for large internetworks; for example, telephone networks are typically organized hierarchically. We contrast the two approaches to internetworking by means of an example.

Example 5.3 Hierarchical versus Nonhierarchical Routing

Consider a country with several cities each having its own data network, and suppose that the city networks are connected into a national network using gateways. Each packet must have an internetwork destination address, consisting of a city address (or city ID number) and a local address (an ID number for the destination node within the destination city). Assume for concreteness that we want to use an ARPANET-like algorithm, where each

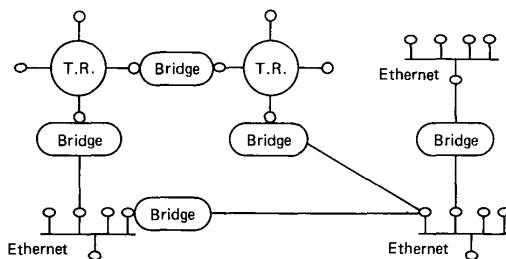


Figure 5.14 Conceptual view of an interconnected network of subnetworks. In a nonhierarchical approach to routing the internetwork is viewed as a single large network.

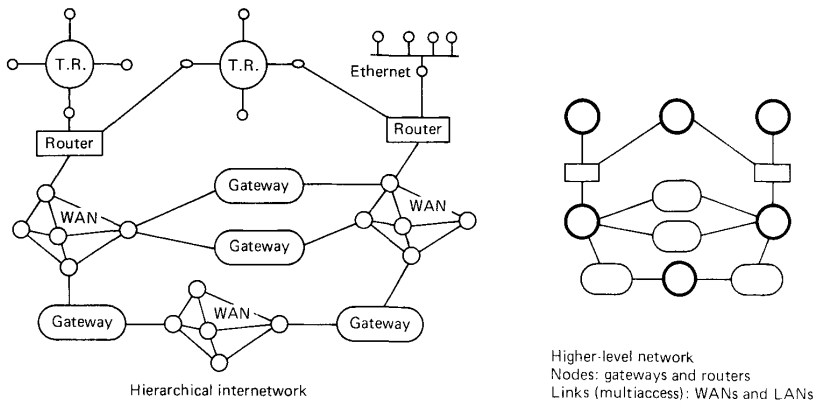


Figure 5.15 Conceptual view of hierarchical routing in an interconnected network of subnetworks. There is a lower-level algorithm that handles routing within each subnetwork and a higher-level algorithm that handles routing between subnetworks.

node has a unique entry in its routing tables for every destination (the next outgoing link on its shortest path to the destination).

In a nonhierarchical routing approach, there is no distinction between the city networks; each node of each city must have a separate entry in its routing tables for every node in every other city. Thus the size of the routing tables is equal to the total number of nodes in the national network. Furthermore, a shortest path calculation to update the routing tables must be performed for the entire national network. Thus we see the main drawbacks of the nonhierarchical approach: the size of the routing tables and the calculations to update them may become excessive.

In a hierarchical approach, there is a lower-level (local) ARPANET-like routing algorithm within each city, and a higher-level ARPANET-like routing algorithm between cities. The local routing algorithm of a city operates only at the nodes of that city. Every node of a city has a routing table entry for every node of the same city and a routing table entry for every other city. The higher-level algorithm provides for routing between cities and operates only at the gateways. Each gateway has a routing table entry for every destination city. The latter entry is the next city network to which an internetwork packet must be routed on its way to its destination.

To see how this algorithm works, consider a packet generated at a node of some city *A* and destined for a node of a different city. The origin node checks the packet's address, consults its routing table entry for the packet's destination city, and sends the packet on the corresponding outgoing link. Each node of city *A* subsequently visited by the packet does the same, until the packet arrives at an appropriate gateway. At this point, the higher-level routing algorithm takes over and determines (based on the gateway's routing table) the next city network within which the packet will travel. The packet is then guided by the latter network's local routing algorithm, and the process is repeated until the packet arrives at the destination city's network. Then the packet is routed to its destination node by the local routing algorithm of the destination city.

The routing tables of a city network algorithm may be generated or updated using an adaptive shortest-path algorithm such as the ARPANET SPF algorithm. To do this, the nodes of the city network must have a destination gateway for every other city (the gateway connected to the city network to which packets destined for the other city will be guided by the local routing algorithm). This information must be provided by the higher-level algorithm that operates at the gateways. The higher-level algorithm may itself be static or adaptive, adjusting its intercity routes to temporary congestion patterns.

It can be seen from the preceding example that in the hierarchical approach, the size of the routing tables and the calculations to update them are considerably reduced over the nonhierarchical approach. In fact, the hierarchical approach may be recommended for a single large network (no internetworking) to reduce the routing algorithm's overhead and storage. The disadvantage of the hierarchical approach is that it may not result in routing that is optimal for the entire internetwork with respect to a shortest path or some other criterion.

Note that hierarchical networks may be organized in more than two levels. For example, consider an international network interconnecting several national networks. In a three-level hierarchy, there could be a routing algorithm between countries at the highest level, a routing algorithm between cities of each country at the middle level, and a routing algorithm within each city at the lowest level.

Bridged local area networks. As we discussed in Chapter 4, the performance of a local area network tends to degrade as its number of users increases. In the case of Ethernet-based LANs the performance also degrades as the ratio of propagation delay to transmission capacity increases. A remedy is to connect several LANs so that a large geographical area with many users can be covered without a severe performance penalty.

Routing for interconnected LANs has a special character because LANs operate in a "promiscuous mode," whereby all nodes, including the interface nodes connected to a LAN, hear all the traffic broadcast within the LAN. Furthermore, in a LAN internetwork, interface nodes must be fast and inexpensive, consistent with the characteristics of the LANs connected. This has motivated the use of simple, fast, and relatively inexpensive bridges. A bridge listens to all the packets transmitted within two or more LANs through connections known as *ports*. It also selectively relays some packets from one LAN to another. In this way, packets can travel from one user to another, going through a sequence of LANs and bridges.

An important characteristic of bridges is that they connect LANs at the MAC sublayer. The resulting internetwork is, by necessity, nonhierarchical; it may be viewed as a single network, referred to as the *extended LAN*. To alleviate the disadvantages of nonhierarchical routing mentioned earlier, bridges use some special routing techniques, the two most prominent of which are known as *spanning tree routing* and *source routing*. Both of these schemes assume that each station in the extended LAN has a unique ID number known through a directory that can be accessed by all stations. A station's ID number need not provide information regarding the LAN to which the station is connected. Also, the location of a station is not assumed known, thus allowing stations to be turned on and off and to be moved from one LAN to another at will. To appreciate

the difficulties of this, think of a telephone company trying to operate a network where customers can change their residence without notifying the company, while maintaining their phone numbers. We now describe the two major routing approaches.

Spanning tree routing in bridged local area networks. In the spanning tree method, the key data structure is a spanning tree, which is used by a bridge to determine whether and to which LAN it should relay a packet (see Fig. 5.16). The spanning tree consists of a subset of ports that connect all LANs into a network with no cycles. An important property of a spanning tree is that each of its links separates the tree in two parts; that is, each node of the tree lies on exactly one “side” of the link. From this it is seen that there is a unique path from every LAN to every other LAN on the spanning tree. This path is used for communication of all packets between the LANs. The ports lying on the spanning tree are called *designated* or *active* ports. The spanning tree may need to be modified due to bridge failures, but we postpone a discussion of this issue for later, assuming for the time being that the spanning tree is fixed.

To understand how routing works, note that each bridge can communicate with any given station through a unique active port (see Fig. 5.16). Therefore, to implement spanning tree routing, it is sufficient to maintain for each active port a list of the stations with which the port communicates; this is known as the *forwarding database* (FDB) of the port. Assuming that the FDBs at each bridge are complete (they contain all stations), routing would be implemented as follows: If a packet is heard on the LAN corresponding to an active port *A* of a bridge, then if the destination ID of the packet is included in the FDB of an active bridge port *B* different from *A*, the packet is transmitted on the LAN connected to *B*; otherwise, the packet is discarded by the bridge (see Fig. 5.17).

Unfortunately, the routing method just described does not quite work because the FDBs of the active ports are not complete at all times. The reason is that stations may be turned on and off or change location, and also the spanning tree may change.

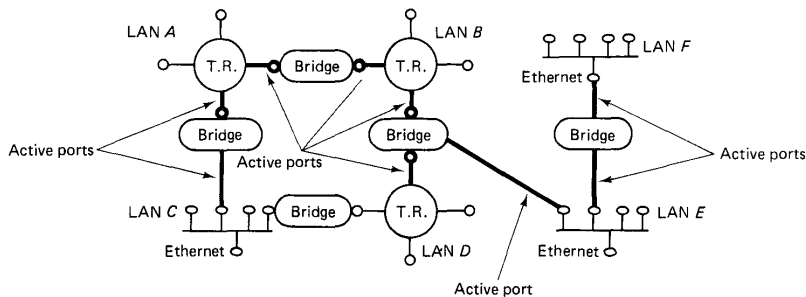


Figure 5.16 Illustration of spanning tree in a network of LANs interconnected with bridges. The spanning tree consists of a set of bridge ports (called active), which connect all LANs into a network without cycles. Only active ports are used for routing packets and thus there is a unique route connecting a pair of LANs. For example, a packet of a station connected to LAN *C* will go to a station in LAN *E* through LANs *A* and *B*.

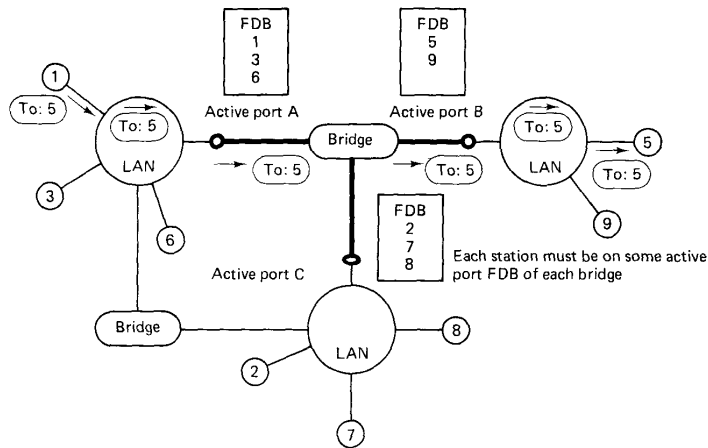


Figure 5.17 Using forwarding databases (FDBs) to effect spanning tree routing. Each active port maintains an FDB, that is, a list of the stations with which it communicates. An active port, say *A*, checks the destination of each packet transmitted within its associated LAN. If the destination appears in the FDB of another active port of the same bridge, say *B*, the bridge broadcasts the packet on the LAN connected with port *B*.

For this reason, the FDBs are constantly being updated through a process known as *bridge learning*. Each active port adds to its FDB the source ID numbers of the packets transmitted on the corresponding LAN. Also each active port deletes from its FDB the ID numbers of stations whose packets have not been transmitted on the corresponding LAN for a certain period of time; this helps to deal with situations where a station is disconnected with one LAN and is connected to another, and also cleans up automatically the FDBs when the spanning tree changes. When a port first becomes active its FDB is initialized to empty.

Since not all station IDs will become known through bridge learning, the routing method includes a search feature. If the destination ID of an incoming packet at some active port is unknown at each of the bridge's FDBs, the bridge relays the packet on all the other active ports. An illustration of the algorithm's operation in this case is shown in Fig. 5.18. Here station 1 sends a packet *P* to station 2, but because 2 has not yet transmitted any packet, its ID number does not appear on any FDB. Then *P* is broadcast on the entire spanning tree, eventually reaching the destination 2. When 2 answers *P* with a packet *R* of its own, this packet will follow the unique spanning tree route to 1 (because 1's ID number was entered in all of the appropriate FDBs of active ports of the spanning tree when *P* was broadcast through the spanning tree). As a result, 2's ID number will be entered in the FDBs of the active ports along the route that connects 2 with 1, and a subsequent packet from 1 to 2 will follow this route.

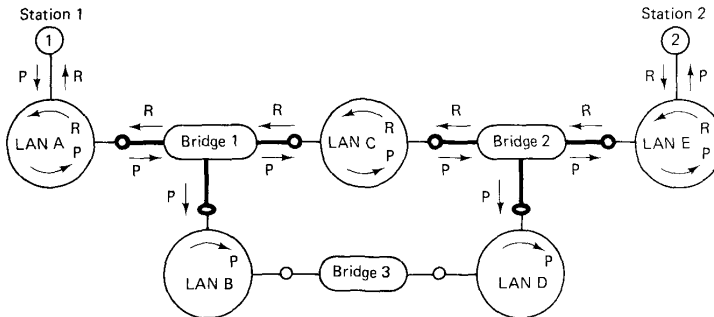


Figure 5.18 Illustration of bridge learning and FDB updating. Station 1 sends a packet *P* to station 2, but the ID of station 2 does not appear in the FDB of any active port of bridge 1 (because, for example, station 2 has not transmitted any packet for a long time). Packet *P* is broadcast into LANs *B* and *C* by bridge 1, and then it is broadcast into LANs *D* and *E* by bridge 2. In the process the ID of node 1 is entered in the left-side FDBs of bridges 1 and 2 (if not already there). Station 2 replies with a packet *R*, which goes directly to its destination station 1 through bridges 2 and 1, because the ID number of station 1 appears in the left-side FDBs of these bridges.

The method by which the spanning tree is updated in the event of a bridge or LAN failure or repair remains to be discussed. There are several algorithms for constructing spanning trees, some of which are discussed in Section 5.2. What is needed here, however, is a distributed algorithm that can work in the face of multiple and unpredictable failures and repairs. A relatively simple possibility is to use a spanning tree of shortest paths from every LAN and bridge to a particular destination (or root). We will see in Section 5.2 that the Bellman–Ford algorithm is particularly well suited for this purpose. There is still, however, an extra difficulty, namely how to get all bridges to agree on the root and what to do when the root itself fails. Such issues involve subtle difficulties, which are unfortunately generic in distributed algorithms dealing with component failures. The difficulty can be resolved by numbering the bridges and by designating as root of the tree the bridge with the smallest ID number. Each bridge keeps track of the smallest bridge ID number that it knows of. A bridge must periodically initiate a spanning tree construction algorithm if it views itself as the root based on its own ID number and the ID numbers of other bridges it is aware of. Also, each bridge ignores messages from bridges other than its current root. To cope with the situation where the root fails, bridges may broadcast periodically special “hello” messages to other bridges. These messages may also carry additional information relating to the spanning tree construction and updating algorithm. (See [Bac88] for a description of the corresponding IEEE 802.1 MAC Bridge Draft Standard.)

Source routing in bridged local area networks. Source routing is an alternative to the spanning tree method for bridged LANs. The idea is to discover a route between two stations *A* and *B* wishing to communicate and then to include the route on the header of each packet exchanged between *A* and *B*. There are two phases here:

1. Locating *B* through the extended LAN. This can be done by broadcasting an exploratory packet from *A* through the extended LAN (either flooding or a spanning tree can be used for this).
2. Selecting a route out of the many possible that connect *A* and *B*. One way of doing this is to send a packet from *B* to *A* along each possible loop-free route. The sequence of LANs and bridges is recorded on the packet as it travels along its route. Thus *A* gets to know all possible routes to *B* and can select one of these routes using, for example, a shortest path criterion.

Note the similarity of source routing with virtual circuit routing; in both cases, all packets of a user pair follow the same path.

The main advantage of the spanning tree method is that it does not require any cooperation of the user/stations in the routing algorithm. By contrast, in source routing, the origin station must select a route along which to establish a virtual circuit. Thus the software of the stations must provide for this capability. This is unnecessary in spanning tree routing, where the routing process is transparent to the stations.

The main advantage of source routing is that with each packet having its route stamped on it, there is no need for routing table lookup at the bridges. This simplifies packet processing and may result in bridges with higher packet throughput. Another advantage of source routing is that it may use the available network transmission capacity more efficiently. While spanning tree routing uses a single possibly nonshortest path for each origin–destination pair, with source routing, shortest path and multiple-path routing can be used. Figure 5.19 shows an example where spanning tree–based routing is inefficient.

We finally mention that both spanning tree and source routing suffer from a common disadvantage. They must often broadcast packets throughout the network either to update FDBs or to establish new connections between user pairs. These broadcasts are needed

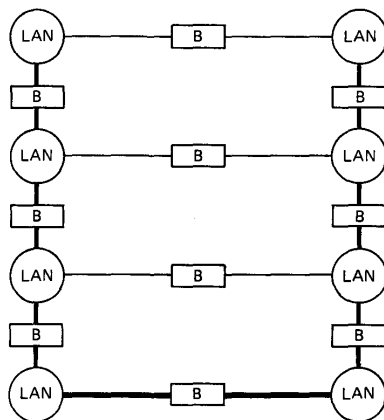


Figure 5.19 Example of inefficient spanning tree routing. The routing paths are good for origin LANs near the bottom of the figure but poor for origin and destination LANs near the top of the figure.

to allow dynamic relocation of the stations while keeping the bridge hardware simple. They may, however, become a source of congestion.

5.2 NETWORK ALGORITHMS AND SHORTEST PATH ROUTING

Routing methods involve the use of a number of simple graph-theoretic problems such as the shortest path problem, discussed in the preceding section. In this section we consider shortest path routing in detail together with related problems of interest. We start by introducing some basic graph-theoretic notions.

5.2.1 Undirected Graphs

We define a *graph*, $G = (\mathcal{N}, \mathcal{A})$, to be a finite nonempty set $\mathcal{N}$ of *nodes* and a collection $\mathcal{A}$ of pairs of distinct nodes from $\mathcal{N}$. Each pair of nodes in $\mathcal{A}$ is called an *arc*. Several examples of graphs are given in Fig. 5.20. The major purpose of the formal definition $G = (\mathcal{N}, \mathcal{A})$ is to stress that the location of the nodes and the shapes of the arcs in a pictorial representation are totally irrelevant. If n_1 and n_2 are nodes in a graph and (n_1, n_2) is an arc, this arc is said to be *incident* on n_1 and n_2 . Some authors define graphs so as to allow arcs to have both ends incident on the same node, but we have intentionally disallowed such loops. We have also disallowed multiple arcs between the same pair of nodes.

A *walk* in a graph G is a sequence of nodes $(n_1, n_2, \dots, n_\ell)$ of nodes such that each of the pairs $(n_1, n_2), (n_2, n_3), \dots, (n_{\ell-1}, n_\ell)$ are arcs of G . A walk with no repeated nodes is a *path*. A walk $(n_1, \dots, n_\ell)$ with $n_1 = n_\ell$, $\ell > 3$, and no repeated nodes other than $n_1 = n_\ell$ is called a *cycle*. These definitions are illustrated in Fig. 5.21.

We say that a graph is *connected* if for each node i there is a path $(i = n_1, n_2, \dots, n_\ell = j)$ to each other node j . Note that the graphs in Fig. 5.20(a) and (c) are connected, whereas the graph in Fig. 5.20(b) is not connected. We spot an absence of connectivity in a graph by seeing two sets of nodes with no arcs between them. The following lemma captures this insight. The proof is almost immediate and is left as an exercise.

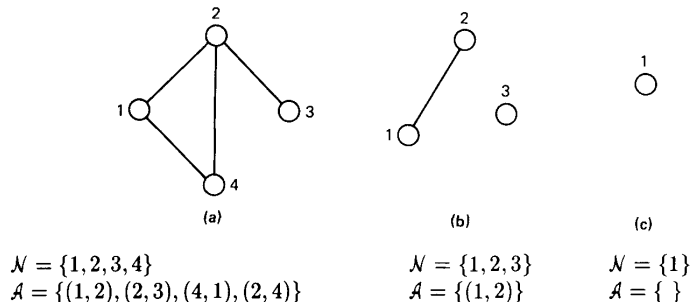


Figure 5.20 Examples of graphs with a set of nodes $\mathcal{N}$ and a set of arcs $\mathcal{A}$.

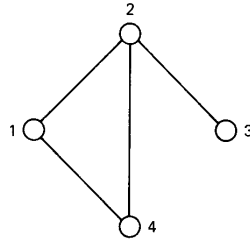


Figure 5.21 Illustration of walks, paths, and cycles. The sequences (1,4,2,3), (1,4,2,1), (1,4,2,1,4,1), (2,3,2), and (2) are all walks. The sequences (1,4,2,3) and (2) are each paths; and (1,4,2,1) is a cycle. Note that (2) and (2,3,2) are not considered cycles.

Lemma Let G be a connected graph and let S be any nonempty strict subset of the set of nodes N . Then at least one arc (i, j) exists with $i \in S$ and $j \notin S$.

We say that $G' = (\mathcal{N}', \mathcal{A}')$ is a *subgraph* of $G = (\mathcal{N}, \mathcal{A})$ if G' is a graph, $\mathcal{N}' \subset \mathcal{N}$, and $\mathcal{A}' \subset \mathcal{A}$. For example, the last three graphs in Fig. 5.22 are subgraphs of the first.

A *tree* is a connected graph that contains no cycles. A *spanning tree* of a graph G is a subgraph of G that is a tree and that includes all the nodes of G . For example, the subgraphs in Fig. 5.22(b) and (c) are spanning trees of the graph in Fig. 5.22(a). The following simple algorithm constructs a spanning tree of an arbitrary connected graph $G = (\mathcal{N}, \mathcal{A})$:

1. Let n be an arbitrary node in $\mathcal{N}$. Let $\mathcal{N}' = \{n\}$, and let $\mathcal{A}'$ be the empty set $\emptyset$.
2. If $\mathcal{N}' = \mathcal{N}$, then stop [$G' = (\mathcal{N}', \mathcal{A}')$ is a spanning tree]; else go to step 3.
3. Let $(i, j) \in \mathcal{A}$ be an arc with $i \in \mathcal{N}'$, $j \in \mathcal{N} - \mathcal{N}'$. Update $\mathcal{N}'$ and $\mathcal{A}'$ by

$$\mathcal{N}' := \mathcal{N}' \cup \{j\}$$

$$\mathcal{A}' := \mathcal{A}' \cup \{(i, j)\}$$

Go to step 2.

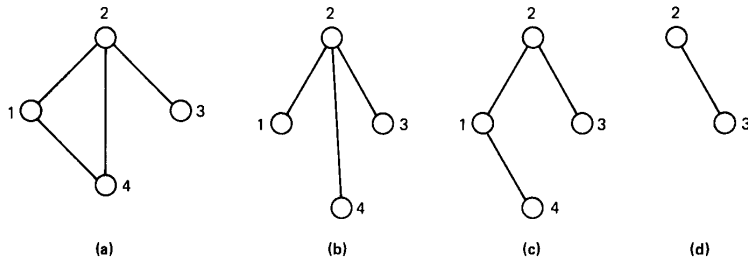


Figure 5.22 A graph (a) and three subgraphs (b), (c), and (d). Subgraphs (b) and (c) are spanning trees.

To see why the algorithm works, note that step 3 is entered only when $\mathcal{N}'$ is a proper subset of $\mathcal{N}$, so that the earlier lemma guarantees the existence of the arc (i, j) . We use induction on successive executions of step 3 to see that $G' = (\mathcal{N}', \mathcal{A}')$ is always a tree. Initially, $G' = (\{n\}, \emptyset)$ is trivially a tree, so assume that $G' = (\mathcal{N}', \mathcal{A}')$ is a tree before the update of step 3. This ensures that there is a path between each pair of nodes in $\mathcal{N}'$ using arcs of $\mathcal{A}'$. After adding node j and arc (i, j) , each node has a path to j simply by adding j to the path to i , and similarly j has a path to each other node. Finally, node j cannot be in any cycles since (i, j) is the only arc of G' incident to j , and there are no cycles not including j by the inductive hypothesis. Figure 5.23 shows G' after each execution of step 3 for one possible choice of arcs.

Observe that the algorithm starts with a subgraph of one node and zero arcs and adds one node and one arc on each execution of step 3. This means that the spanning tree, G' , resulting from the algorithm always has N nodes, where N is the number of nodes in G , and $N - 1$ arcs. Since G' is a subgraph of G , the number of arcs, A , in G must satisfy $A \geq N - 1$; this is true for every connected graph G . Next, assume that $A = N - 1$. This means that the algorithm uses all arcs of G in the spanning tree G' , so that $G = G'$, and G must be a tree itself. Finally, if $A \geq N$, then G contains at least one arc (i, j) not in the spanning tree G' generated by the algorithm. Letting $(j, \dots, i)$ be the path from j to i in G' , it is seen that $(i, j, \dots, i)$ is a cycle in G and G cannot be a tree. The following proposition summarizes these results:

Proposition. Let G be a connected graph with N nodes and A arcs. Then:

1. G contains a spanning tree.
2. $A \geq N - 1$.
3. G is a tree if and only if $A = N - 1$.

Figure 5.24 shows why connectedness is a necessary assumption for the last part of the proposition.

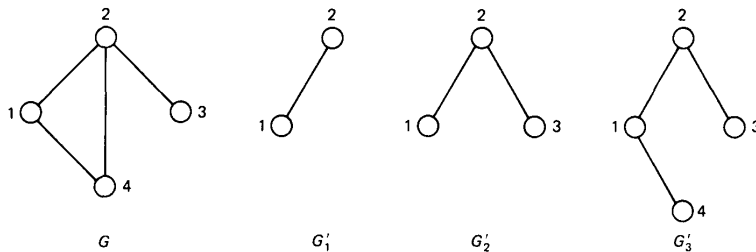


Figure 5.23 Algorithm for constructing a spanning tree of a given graph G .

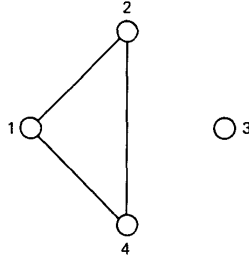


Figure 5.24 Graph with $A = N - 1$, which is both disconnected and contains a cycle.

5.2.2 Minimum Weight Spanning Trees

It is possible to construct a spanning tree rooted at some node by sending a packet from the node to all its neighbors; the neighbors must send the packet to their neighbors, and so on, with all nodes marking the transmitter of the first packet they receive as their “father” on the tree, as described in Fig. 5.8. However, such a tree has no special properties. If the communication cost or delay of different links should be taken into account in constructing a spanning tree, we may consider using a minimum weight spanning tree, which we now define.

Consider a connected graph $G = (\mathcal{N}, \mathcal{A})$ with N nodes and A arcs, and a weight w_{ij} for each arc $(i, j) \in \mathcal{A}$. A *minimum weight spanning tree* (MST for short) is a spanning tree with minimum sum of arc weights. An arc weight represents the communication cost of a message along the arc in either direction, and the total spanning tree weight represents the cost of broadcasting a message to all nodes along the spanning tree.

Any subtree (i.e., a subgraph that is a tree) of an MST will be called a *fragment*. Note that a node by itself is considered a fragment. An arc having one node in a fragment and the other node not in this fragment is called an *outgoing arc* from the fragment. The following proposition is of central importance for MST algorithms.

Proposition 1. Given a fragment F , let $\alpha = (i, j)$ be a minimum weight outgoing arc from F , where the node j is not in F . Then F , extended by arc α and node j , is a fragment.

Proof: Denote by M the MST of which F is a subtree. If arc α belongs to M , we are done, so assume otherwise. Then there is a cycle formed by α and the arcs of M . Since node j does not belong to F , there must be some arc $\beta \neq \alpha$ that belongs to the cycle and to M , and which is outgoing from F (see Fig. 5.25). Deleting β from M and adding α results in a subgraph M' with $(N - 1)$ arcs and no cycles which, therefore, must be a spanning tree. Since α has less or equal weight to β , M' must be an MST, so F extended by α and j forms a fragment. **Q.E.D.**

Proposition 1 can be used as the basis for MST construction algorithms. The idea is to start with one or more disjoint fragments and enlarge or combine them by adding minimum weight outgoing arcs. One method (the Prim–Dijkstra algorithm) starts with an

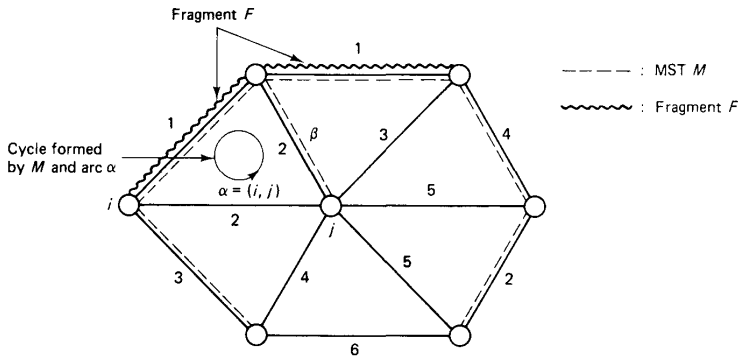


Figure 5.25 Proof of Proposition 1. The numbers next to the arcs are the arc weights. F is a fragment which is a subtree of an MST M . Let α be a minimum weight outgoing arc from F not belonging to M . Let $\beta \neq \alpha$ be an arc that is outgoing from F and simultaneously belongs to M and to the cycle formed by α and M . Deleting β from M and adding α in its place forms another MST M' . When F is extended by α , it forms a fragment.

arbitrarily selected single node as a fragment and enlarges the fragment by successively adding a minimum weight outgoing arc. Another method (Kruskal's algorithm) starts with each node being a single node fragment; it then successively combines two of the fragments by using the arc that has minimum weight over all arcs that when added to the current set of fragments do not form a cycle (see Fig. 5.26). Both of these algorithms terminate in $N - 1$ iterations.

Kruskal's algorithm proceeds by building up simultaneously several fragments that eventually join into an MST; however, only one arc at a time is added to the current set of fragments. Proposition 1 suggests the possibility of adding a minimum weight outgoing arc simultaneously to each fragment in a distributed algorithmic manner. This is possible if there is a unique MST, as we now discuss.

A distributed algorithm for constructing the MST in a graph with a unique MST is as follows. Start with a set of fragments (these can be the nodes by themselves, for example). Each fragment determines its minimum weight outgoing arc, adds it to itself, and informs the fragment that lies at the other end of this arc. It can be seen that as long as the arc along which two fragments join is indeed a minimum weight outgoing arc for some fragment, the algorithm maintains a set of fragments of the MST at all times, and no cycle is ever formed. Furthermore, new arcs will be added until there are no further outgoing arcs and there is only one fragment (by necessity the MST). Therefore, the algorithm cannot stop short of finding the MST. Indeed, for the algorithm to work correctly, it is not necessary that the procedure for arc addition be synchronized for all fragments. What is needed, however, is a scheme for the nodes and arcs of a fragment to determine the minimum weight outgoing arc. There are a number of possibilities along these lines, but it is of interest to construct schemes that accomplish this with low

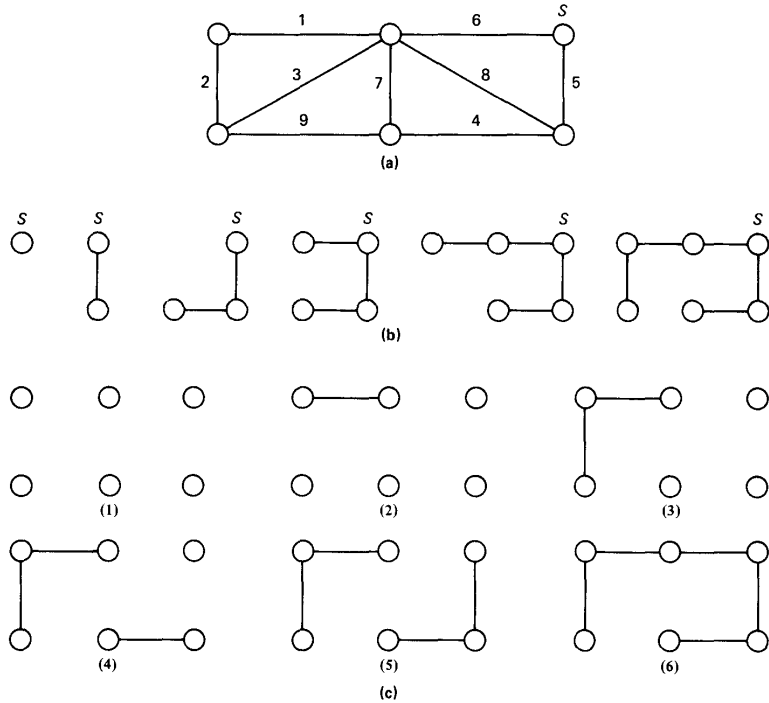


Figure 5.26 Minimum weight spanning tree construction. (a) Graph with arc weights as indicated. (b) Successive iterations of the Prim-Dijkstra algorithm. The starting fragment consists of the single node marked *S*. The fragment is successively extended by adding a minimum weight outgoing arc. (c) Successive iterations of the Kruskal algorithm. The algorithm starts with all nodes as fragments. At each iteration, we add the arc that has minimum weight out of all arcs that are outgoing from one of the fragments.

communication overhead. This subject is addressed in [GHS83], to which we refer for further details. Reference [Hum83] considers the case where the arc weights are different in each direction. A distributed MST algorithm of the type just described has been used in the PARIS network [CGK90] (see Section 6.4 for a description of this network).

To see what can go wrong in the case of nonunique MSTs, consider the triangular network of Fig. 5.27 where all arc lengths are unity. If we start with the three nodes as fragments and allow each fragment to add to itself an arbitrary, minimum weight outgoing arc, there is the possibility that the arcs (1,2), (2,3), and (3,1) will be added simultaneously by nodes 1, 2, and 3, respectively, with a cycle resulting.

The following proposition points the way on how to handle the case of nonunique MSTs.

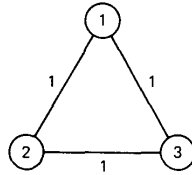


Figure 5.27 Counterexample for distributed MST construction with nondistinct arc weights. Nodes 1, 2, and 3 can add simultaneously to their fragments arcs (1,2), (2,3), and (3,1), respectively, with a cycle resulting.

Proposition 2. If all arc weights are distinct, there exists a unique MST.

Proof: Suppose that there exist two distinct MSTs denoted M and M' . Let α be the minimum weight arc that belongs to either M or M' but not to both. Assume for concreteness that α belongs to M . Consider the graph formed by the union of M' and α (see Fig. 5.28). It must contain a cycle (since $\alpha \notin M'$), and at least one arc of this cycle, call it β , does not belong to M (otherwise M would contain a cycle). Since the weight of α is strictly smaller than that of β , it follows that deleting β from M' and adding α in its place results in a spanning tree of strictly smaller weight than M' . This is a contradiction since M' is optimal. **Q.E.D.**

Consider now an MST problem where the arc weights are not distinct. The ties between arcs with the same weight can be broken by using the identities of their nodes; that is, if arcs (i, j) and (k, l) with $i < j$ and $k < l$ have equal weight, prefer arc (i, j) if $i < k$ or if $i = k$ and $j < l$, and prefer arc (k, l) otherwise. Thus, by implementing, if necessary, the scheme just described, one can assume without loss of generality that arc weights are distinct and that the MST is unique, thereby allowing the use of distributed MST algorithms that join fragments along minimum weight outgoing arcs.

5.2.3 Shortest Path Algorithms

In what follows we shall be interested in traffic flowing over the arcs of a network. This makes it necessary to distinguish the direction of the flow, and thus to give a reference direction to the arcs themselves.

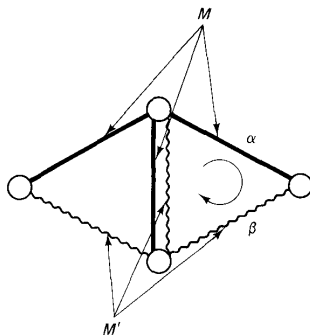


Figure 5.28 Proof of MST uniqueness when all arc weights are distinct. Let M and M' be two MSTs and assume that α is the minimum weight arc that belongs to M and M' but not both. Suppose that $\alpha \in M$ and let β be an arc of the cycle of $M' \cup \{\alpha\}$ which does not belong to M . By deleting β from M' and adding α in its place, we obtain a spanning tree because the corresponding subgraph has $N - 1$ arcs and is seen to be connected. The weight of this spanning tree is smaller than the one of M' , because the weight of α is smaller than the weight of β —a contradiction.

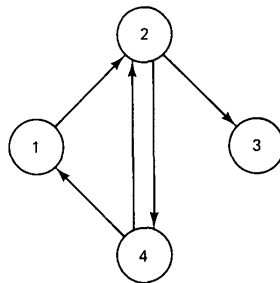
A *directed graph* or *digraph* $G = (\mathcal{N}, \mathcal{A})$ is a finite nonempty set $\mathcal{N}$ of nodes and a collection $\mathcal{A}$ of *ordered* pairs of distinct nodes from $\mathcal{N}$; each ordered pair of nodes in $\mathcal{A}$ is called a *directed arc* (or simply *arc*). Pictorially, a digraph is represented in the same way as a graph, but an arrow is placed on the representation of the arc, going from the first node of the ordered pair to the second (see Fig. 5.29). Note in Fig. 5.29 that (2,4) and (4,2) are different arcs.

Given any digraph $G = (\mathcal{N}, \mathcal{A})$, there is an associated (undirected) graph $G' = (\mathcal{N}', \mathcal{A}')$, where $\mathcal{N}' = \mathcal{N}$ and $(i, j) \in \mathcal{A}'$ if either $(i, j) \in \mathcal{A}$, or $(j, i) \in \mathcal{A}$, or both. We say that $(n_1, n_2, \dots, n_\ell)$ is a *walk*, *path*, or *cycle* in a digraph if it is a walk, path, or cycle in the associated graph. In addition, $(n_1, n_2, \dots, n_\ell)$ is a *directed walk* in a digraph G if (n_i, n_{i+1}) is a directed arc in G for $1 \leq i \leq \ell - 1$. A *directed path* is a directed walk with no repeated nodes, and a *directed cycle* is a directed walk $(n_1, \dots, n_\ell)$, for $\ell > 2$ with $n_1 = n_\ell$ and no repeated nodes. Note that (n_1, n_2, n_1) is a directed cycle if (n_1, n_2) and (n_2, n_1) are both directed arcs, whereas (n_1, n_2, n_1) cannot be an undirected cycle if (n_1, n_2) is an undirected arc.

A digraph is *strongly connected* if for each pair of nodes i and j there is a directed path $(i = n_1, n_2, \dots, n_\ell = j)$ from i to j . A digraph is *connected* if the associated graph is connected. The first graph in Fig. 5.30 is connected but not strongly connected since there is no directed path from 3 to 2. The second graph is strongly connected.

Consider a directed graph $G = (\mathcal{N}, \mathcal{A})$ with number of nodes N and number of arcs A , in which each arc (i, j) is assigned some real number d_{ij} as the length or distance of the arc. Given any directed path $p = (i, j, k, \dots, \ell, m)$, the *length* of p is defined as $d_{ij} + d_{jk} + \dots + d_{\ell m}$. The length of a directed walk or cycle is defined analogously. Given any two nodes i, m of the graph, the shortest path problem is to find a minimum length (*i.e.*, shortest) directed path from i to m .

The shortest path problem appears in a surprisingly large number of contexts. If d_{ij} is the cost of using a given link (i, j) in a data network, then the shortest path from i to m is the minimum cost route over which to send data. Thus, if the cost of a link equals the average packet delay to cross the link, the minimum cost route is also a



$$\mathcal{N} = \{1, 2, 3, 4\}$$

$$\mathcal{A} = \{(1, 2), (2, 3), (2, 4), (4, 2), (4, 1)\}$$

Figure 5.29 Representation of a directed graph.

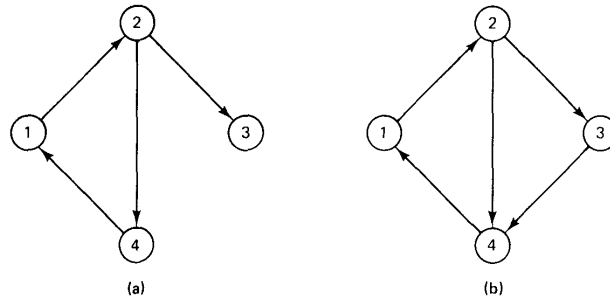


Figure 5.30 Both digraphs (a) and (b) are connected; (b) is strongly connected, but (a) is not strongly connected since there is no directed path from 3 to 2.

minimum delay route. Unfortunately, in a data network, the average link delay depends on the traffic load carried by the link, which in turn depends on the routes selected by the routing algorithm. Because of this feedback effect, the minimum average delay routing problem is more complex than just solving a shortest path problem; however, the shortest path problem is still an integral part of all the formulations of the routing problem to be considered. As another example, if p_{ij} is the probability that a given arc (i, j) in a network is usable, and each arc is usable independently of all the other arcs, then finding the shortest path between i and m with arc distances $(-\ln p_{ij})$ is equivalent to finding the most reliable path from i to m .

Another application of shortest paths is in the PERT networks used by organizations to monitor the progress of large projects. The nodes of the network correspond to subtasks, and an arc from subtask i to j indicates that the completion of task j is dependent on the completion of i . If t_{ij} is the time required to complete j after i is completed, the distance for (i, j) is taken as $d_{ij} = -t_{ij}$. The shortest path from project start to finish is then the most time-consuming path required for completion of the project, and the shortest path indicates the critical subtasks for which delays would delay the entire project. Yet another example is that of discrete dynamic programming problems, which can be viewed as shortest path problems [Ber87]. Finally, many more complex graph-theoretic problems require the solution of shortest path problems as subproblems.

In the following, we develop three standard algorithms for the shortest path problem: the Bellman–Ford algorithm, the Dijkstra algorithm, and the Floyd–Warshall algorithm. The first two algorithms find the shortest paths from all nodes to a given destination node, and the third algorithm finds the shortest paths from all nodes to all other nodes. Note that the problem of finding shortest paths from a given origin node to all other nodes is equivalent to the problem of finding all shortest paths to a given destination node; one version of the problem can be obtained from the other simply by reversing the direction of each arc while keeping its length unchanged. In this subsection we concentrate on centralized shortest path algorithms. Distributed algorithms are considered in Section 5.2.4.

The Bellman–Ford algorithm. Suppose that node 1 is the “destination” node and consider the problem of finding a shortest path from every node to node 1. We assume that there exists at least one path from every node to the destination. To simplify the presentation, let us denote $d_{ij} = \infty$ if (i, j) is not an arc of the graph. Using this convention we can assume without loss of generality that there is an arc between every pair of nodes, since walks and paths consisting of true network arcs are the only ones with length less than ∞ .

A shortest walk from a given node i to node 1, subject to the constraint that the walk contains at most h arcs and goes through node 1 only once, is referred to as a *shortest ($\leq h$) walk* and its length is denoted by D_i^h . Note that such a walk may not be a path, that is, it may contain repeated nodes; we will later give conditions under which this is not possible. By convention, we take

$$D_1^h = 0, \quad \text{for all } h$$

We will prove shortly that D_i^h can be generated by the iteration

$$D_i^{h+1} = \min_j [d_{ij} + D_j^h], \quad \text{for all } i \neq 1 \quad (5.1)$$

starting from the initial conditions

$$D_i^0 = \infty, \quad \text{for all } i \neq 1 \quad (5.2)$$

This is the Bellman–Ford algorithm, illustrated in Fig. 5.31. Thus, we claim that the Bellman–Ford algorithm first finds the one-arc shortest walk lengths, then finds the two-arc shortest walk lengths, and so forth. Once we show this, we will argue that the shortest walk lengths are equal to the shortest path lengths, under the additional assumption that all cycles not containing node 1 have nonnegative length. We say that the algorithm terminates after h iterations if

$$D_i^h = D_i^{h-1}, \quad \text{for all } i$$

The following proposition provides the main results.

Proposition. Consider the Bellman–Ford algorithm (5.1) with the initial conditions $D_i^0 = \infty$ for all $i \neq 1$. Then:

- (a) The scalars D_i^h generated by the algorithm are equal to the shortest ($\leq h$) walk lengths from node i to node 1.
- (b) The algorithm terminates after a finite number of iterations if and only if all cycles not containing node 1 have nonnegative length. Furthermore, if the algorithm terminates, it does so after at most $h \leq N$ iterations, and at termination, D_i^h is the shortest path length from i to 1.

Proof: (a) We argue by induction. From Eqs. (5.1) and (5.2) we have

$$D_i^1 = d_{i1}, \quad \text{for all } i \neq 1$$

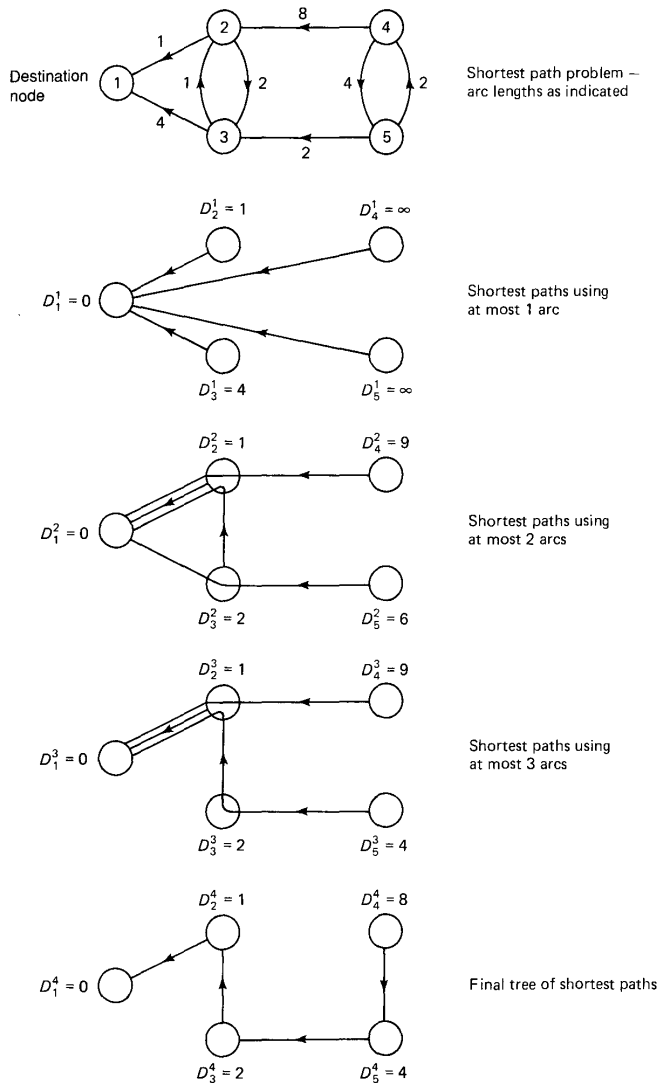


Figure 5.31 Successive iterations of the Bellman-Ford method. In this example, the shortest ($\leq h$) walks are paths because all arc lengths are positive and therefore all cycles have positive length. The shortest paths are found after $N - 1$ iterations, which is equal to 4 in this example.

so D_i^1 is indeed equal to the shortest (≤ 1) walk length from i to 1. Suppose that D_i^k is equal to the shortest ($\leq k$) walk length from i to 1 for all $k \leq h$. We will show that D_i^{h+1} is the shortest ($\leq h+1$) walk length from i to 1. Indeed, a shortest ($\leq h+1$) walk from i to 1 either consists of less than $h+1$ arcs, in which case its length is equal to D_i^h , or else it consists of $h+1$ arcs with the first arc being (i, j) for some $j \neq 1$, followed by an h -arc walk from j to 1 in which node 1 is not repeated. The latter walk must be a shortest ($\leq h$) walk from j to 1. [Otherwise, by concatenating arc (i, j) and a shorter ($\leq h$) walk from j to 1, we would obtain a shorter ($\leq h+1$) walk from i to 1.] We thus conclude that

$$\text{Shortest } (\leq h+1) \text{ walk length} = \min \left\{ D_i^h, \min_{j \neq 1} [d_{ij} + D_j^h] \right\} \quad (5.3)$$

Using the induction hypothesis, we have $D_j^k \leq D_j^{k-1}$ for all $k \leq h$ [since the set of ($\leq k$) walks from node j to 1 contains the corresponding set of ($\leq k-1$) walks]. Therefore,

$$D_i^{h+1} = \min_j [d_{ij} + D_j^h] \leq \min_j [d_{ij} + D_j^{h-1}] = D_i^h \quad (5.4)$$

Furthermore, we have $D_i^h \leq D_i^1 = d_{i1} = d_{i1} + D_i^h$, so from Eq. (5.3) we obtain

$$\text{Shortest } (\leq h+1) \text{ walk length} = \min \left\{ D_i^h, \min_j [d_{ij} + D_j^h] \right\} = \min \{ D_i^h, D_i^{h+1} \}$$

In view of $D_i^{h+1} \leq D_i^h$ [cf. Eq. (5.4)], this yields

$$\text{Shortest } (\leq h+1) \text{ walk length} = D_i^{h+1}$$

completing the induction proof.

(b) If the Bellman-Ford algorithm terminates after h iterations, we must have

$$D_i^k = D_i^h, \quad \text{for all } i \text{ and } k \geq h \quad (5.5)$$

so we cannot reduce the lengths of the shortest walks by allowing more and more arcs in these walks. It follows that there cannot exist a negative-length cycle not containing node 1, since such a cycle could be repeated an arbitrarily large number of times in walks from some nodes to node 1, thereby making their length arbitrarily small and contradicting Eq. (5.5). Conversely, suppose that all cycles not containing node 1 have nonnegative length. Then by deleting all such cycles from shortest ($\leq h$) walks, we obtain paths of less or equal length. Therefore, for every i and h , there exists a path that is a shortest ($\leq h$) walk from i to 1, and the corresponding shortest path length is equal to D_i^h . Since paths have no cycles, they can contain at most $N-1$ arcs. It follows that

$$D_i^N = D_i^{N-1}, \quad \text{for all } i$$

implying that the algorithm terminates after at most N iterations.

Q.E.D.

Note that the preceding proposition is valid even if there is no path from some nodes i to node 1 in the original network. Upon termination, we will simply have for those nodes $D_i^h = \infty$.

To estimate the computation required to find the shortest path lengths, we note that in the worst case, the algorithm must be iterated N times, each iteration must be done for $N - 1$ nodes, and for each node the minimization must be taken over no more than $N - 1$ alternatives. Thus, the amount of computation grows at worst like N^3 , which is written as $O(N^3)$. Generally, the notation $O(p(N))$, where $p(N)$ is a polynomial in N , is used to indicate a number depending on N that is smaller than $cp(N)$ for all N , where c is some constant independent of N . Actually, a more careful accounting shows that the amount of computation is $O(mA)$, where A is the number of arcs and m is the number of iterations required for termination (m is also the maximum number of arcs contained in a shortest path).

The example in Fig. 5.32 shows the effect of negative-length cycles not involving node 1, and illustrates that one can test for existence of such cycles simply by comparing D_i^N with D_i^{N-1} for each i . As implied by part (b) of the preceding proposition, there exists such a negative-length cycle if and only if $D_i^N < D_i^{N-1}$ for some i .

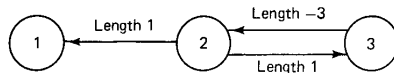


Figure 5.32 Graph with a negative cycle. The shortest path length from 2 to 1 is 1. The Bellman–Ford algorithm gives $D_2^2 = 1$ and $D_2^3 = -1$, indicating the existence of a negative-length cycle.

Bellman's equation and shortest path construction. Assume that all cycles not containing node 1 have nonnegative length and denote by D_i the shortest path length from node i to 1. Then upon termination of the Bellman–Ford algorithm, we obtain

$$D_i = \min_j [d_{ij} + D_j], \quad \text{for all } i \neq 1 \quad (5.6a)$$

$$D_1 = 0 \quad (5.6b)$$

This is called *Bellman's equation* and expresses that the shortest path length from node i to 1 is the sum of the length of the arc to the node following i on the shortest path plus the shortest path length from that node to node 1. From this equation it is easy to find the shortest paths (as opposed to the shortest path lengths) if all cycles not including node 1 have a positive length (as opposed to zero length). To do this, select, for each $i \neq 1$, one arc (i, j) that attains the minimum in the equation $D_i = \min_j [d_{ij} + D_j]$, and consider the subgraph consisting of these $N - 1$ arcs (see Fig. 5.33). To find the shortest path from any node i , start at i and follow the corresponding arcs of the subgraph until node 1 is reached. Note that the same node cannot be reached twice before reaching node 1, since a cycle would be formed that (on the basis of the equation $D_i = \min_j [d_{ij} + D_j]$) would have zero length [let $(i_1, i_2, \dots, i_k, i_1)$ be the cycle and add the equations

$$D_{i_1} = d_{i_1 i_2} + D_{i_2}$$

...

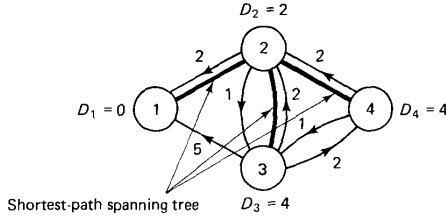


Figure 5.33 Construction of a shortest path spanning tree. For every $i \neq 1$, a neighbor node j_i minimizing $d_{ij} + D_j$ is selected. The tree of the $N - 1$ arcs (i, j_i) defines a set of shortest paths.

$$D_{i_{k-1}} = d_{i_{k-1}i_k} + D_{i_k}$$

$$D_{i_k} = d_{i_k i_1} + D_{i_1}$$

obtaining $d_{i_1 i_2} + \cdots + d_{i_{k-1} i_k} = 0$. Since the subgraph connects every node to node 1 and has $N - 1$ arcs, it must be a spanning tree. We call this subgraph the *shortest path spanning tree* and note that it has the special structure of having a root (node 1), with every arc of the tree directed toward the root. Problem 5.7 shows how such a spanning tree can be obtained when there are cycles of zero length. Problem 5.4 explores the difference between a shortest path spanning tree and a minimum weight spanning tree.

Using the preceding construction, it can be shown that *if there are no zero (or negative)-length cycles, then Bellman's equation (5.6) (viewed as a system of N equations with N unknowns) has a unique solution*. This is an interesting fact which will be useful in Section 5.2.4 when we consider the Bellman–Ford algorithm starting from initial conditions other than ∞ [cf. Eq. (5.2)]. For a proof, we suppose that $\tilde{D}_i, i = 1, \dots, N$, are another solution of Bellman's equation (5.6) with $\tilde{D}_1 = 0$, and we show that $\tilde{D}_i$ are equal to the shortest path lengths D_i . Let us repeat the path construction of the preceding paragraph with $\tilde{D}_i$ replacing D_i . Then $\tilde{D}_i$ is the length of the corresponding path from node i to node 1, showing that $\tilde{D}_i \geq D_i$. To show the reverse inequality, consider the Bellman–Ford algorithm with two different initial conditions. The first initial condition is $D_i^0 = \infty$, for $i \neq 1$, and $D_1^0 = 0$, in which case the true shortest path lengths D_i are obtained after at most $N - 1$ iterations, as shown earlier. The second initial condition is $D_i^0 = \tilde{D}_i$, for all i , in which case $\tilde{D}_i$ is obtained after every iteration (since the $\tilde{D}_i$ solve Bellman's equation). Since the second initial condition is, for every i , less than or equal to the first, it is seen from the Bellman–Ford iteration $D_i^{h+1} = \min_j [d_{ij} + D_j^h]$ that $\tilde{D}_i \leq D_i$, for all i . Therefore, $\tilde{D}_i = D_i$, and the only solution of Bellman's equation is the set of the true shortest path lengths D_i . It is also possible to show that if there are zero-length cycles not involving node 1, then Bellman's equation has a nonunique solution [although the Bellman–Ford algorithm still terminates with the correct shortest path lengths from the initial conditions $D_i^0 = \infty$ for all $i \neq 1$; see Problem 5.8].

It turns out that the Bellman–Ford algorithm works correctly even if the initial conditions D_i^0 for $i \neq 1$ are arbitrary numbers, and the iterations are done in parallel for different nodes in virtually any order. This fact will be shown in the next subsection and accounts in part for the popularity of the Bellman–Ford algorithm for distributed applications.

Dijkstra's algorithm. This algorithm requires that all arc lengths are nonnegative (fortunately, the case for most data network applications). Its worst-case computational requirements are considerably less than those of the Bellman-Ford algorithm. The general idea is to find the shortest paths in order of increasing path length. The shortest of the shortest paths to node 1 must be the single-arc path from the closest neighbor of node 1, since any multiple-arc path cannot be shorter than the first arc length because of the nonnegative-length assumption. The next shortest of the shortest paths must either be the single-arc path from the next closest neighbor of 1 or the shortest two-arc path through the previously chosen node, and so on. To formalize this procedure into an algorithm, we view each node i as being labeled with an estimate D_i of the shortest path length to node 1. When the estimate becomes certain, we regard the node as being *permanently labeled* and keep track of this with a set P of permanently labeled nodes. The node added to P at each step will be the closest to node 1 out of those that are not yet in P . Figure 5.34 illustrates the main idea. The detailed algorithm is as follows: Initially, $P = \{1\}$, $D_1 = 0$, and $D_j = d_{j1}$ for $j \neq 1$.

Step 1: (Find the next closest node.) Find $i \notin P$ such that

$$D_i = \min_{j \notin P} D_j$$

Set $P := P \cup \{i\}$. If P contains all nodes, then stop; the algorithm is complete.

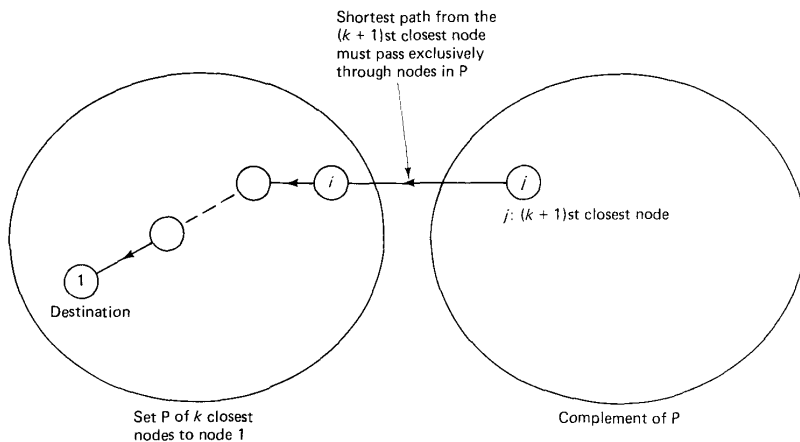


Figure 5.34 Basic idea of Dijkstra's algorithm. At the k^{th} step we have the set P of the k closest nodes to node 1 as well as the shortest distance D_i from each node i in P to node 1. Of all paths connecting some node not in P with node 1, there is a shortest one that passes exclusively through nodes in P (since $d_{ij} \geq 0$). Therefore, the $(k+1)$ st closest node and the corresponding shortest distance are obtained by minimizing over $j \notin P$ the quantity $\min_{i \in P} \{d_{ji} + D_i\}$. This calculation can be organized efficiently as discussed in the text, resulting in an $O(N^2)$ computational complexity.

Step 2: (Updating of labels.) For all $j \notin P$ set

$$D_j := \min[D_j, d_{ji} + D_i]$$

Go to step 1.

To see why the algorithm works, we must interpret the estimates D_j . We claim that at the beginning of each step 1:

- (a) $D_i \leq D_j$ for all $i \in P$ and $j \notin P$.
- (b) D_j is, for each node j , the shortest distance from j to 1 using paths with all nodes except possibly j belonging to the set P .

Indeed, condition (a) is satisfied initially, and since $d_{ji} \geq 0$ and $D_i = \min_{j \notin P} D_j$, it is preserved by the formula $D_j := \min[D_j, d_{ji} + D_i]$ for all $j \notin P$, in step 2. We show condition (b) by induction. It holds initially. Suppose that it holds at the beginning of some step 1, let i be the node added to P at that step, and let D_k be the label of each node k at the beginning of that step. Then condition (b) holds for $j = i$ by the induction hypothesis. It is also seen to hold for all $j \in P$, in view of condition (a) and the induction hypothesis. Finally, for a node $j \notin P \cup \{i\}$, consider a path from j to 1 which is shortest among those with all nodes except j belonging to the set $P \cup \{i\}$, and let D'_j be the corresponding shortest distance. Such a path must consist of an arc (j, k) for some $k \in P \cup \{i\}$, followed by a shortest path from k to 1 with nodes in $P \cup \{i\}$. Since we just argued that the length of this k to 1 path is D_k , we have

$$D'_j = \min_{k \in P \cup \{i\}} [d_{jk} + D_k] = \min \left[\min_{k \in P} [d_{jk} + D_k], d_{ji} + D_i \right]$$

Similarly, the induction hypothesis implies that $D_j = \min_{k \in P} [d_{jk} + D_k]$, so we obtain $D'_j = \min[D_j, d_{ji} + D_i]$. Thus in step 2, D_j is set to the shortest distance D'_j from j to 1 using paths with all nodes except j belonging to $P \cup \{i\}$. The induction proof of condition (b) is thus complete.

We now note that a new node is added to P with each iteration, so the algorithm terminates after $N - 1$ iterations, with P containing all nodes. By condition (b), D_j is then equal to the shortest distance from j to 1.

To estimate the computation required by Dijkstra's algorithm, we note that there are $N - 1$ iterations and the number of operations per iteration is proportional to N . Therefore, in the worst case the computation is $O(N^2)$, comparing favorably with the worst-case estimate $O(N^3)$ of the Bellman-Ford algorithm; in fact, with proper implementation the worst-case computational requirements for Dijkstra's algorithm can be reduced considerably (see [Ber91] for a variety of implementations of Dijkstra's algorithm). However, there are many problems where $A \ll N^2$, and the Bellman-Ford algorithm terminates in very few iterations (say $m \ll N$), in which case its required computation $O(mA)$ can be less than the $O(N^2)$ requirement of the straightforward implementation of Dijkstra's algorithm. Generally, for nondistributed applications, efficiently implemented variants of the Bellman-Ford and Dijkstra algorithms appear to be competitive (see [Ber91] for a more detailed discussion and associated codes).

The Dijkstra and Bellman–Ford algorithms are contrasted on an example problem in Fig. 5.35.

The Floyd–Warshall algorithm. This algorithm, unlike the previous two, finds the shortest paths between all pairs of nodes together. Like the Bellman–Ford algorithm, the arc distances can be positive or negative, but again there can be no negative-length cycles. All three algorithms iterate to find the final solution, but each iterates on something different. The Bellman–Ford algorithm iterates on the number of arcs in a path, the Dijkstra algorithm iterates on the length of the path, and finally, the Floyd–Warshall algorithm iterates on the set of nodes that are allowed as intermediate nodes on the paths. The Floyd–Warshall algorithm starts like both other algorithms with single arc distances

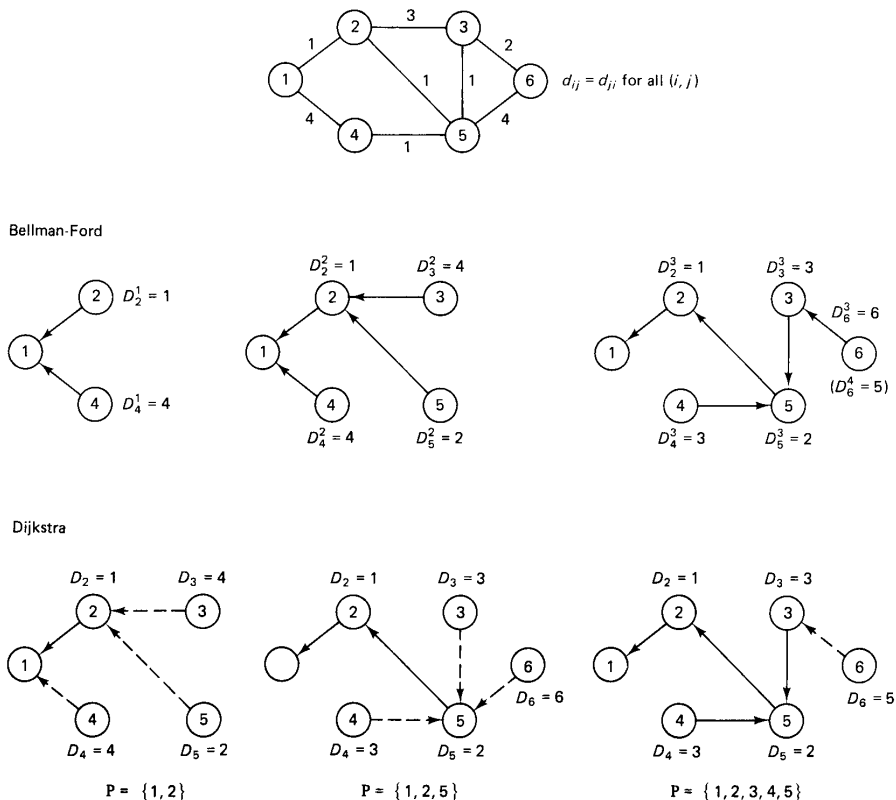


Figure 5.35 Example using the Bellman-Ford and Dijkstra algorithms.

(i.e., no intermediate nodes) as starting estimates of shortest path lengths. It then calculates shortest paths under the constraint that only node 1 can be used as an intermediate node, and then with the constraint that only nodes 1 and 2 can be used, and so forth.

To state the algorithm more precisely, let D_{ij}^n be the shortest path length from node i to j with the constraint that only nodes $1, 2, \dots, n$ can be used as intermediate nodes on paths. The algorithm then is as follows: Initially,

$$D_{ij}^0 = d_{ij}, \quad \text{for all } i, j, \quad i \neq j$$

For $n = 0, 1, \dots, N - 1$,

$$D_{ij}^{n+1} = \min [D_{ij}^n, D_{i(n+1)}^n + D_{(n+1)j}^n], \quad \text{for all } i \neq j \quad (5.7)$$

To see why this works, we use induction. For $n = 0$, the initialization clearly gives the shortest path lengths subject to the constraint of no intermediate nodes on paths. Now, suppose that for a given n , D_{ij}^n in the algorithm above gives the shortest path lengths using nodes 1 to n as intermediate nodes. Then the shortest path length from i to j , allowing nodes 1 to $n + 1$ as intermediate nodes, either contains node $n + 1$ on the shortest path, or does not contain node $n + 1$. For the first case, the constrained shortest path from i to j goes first from i to $n + 1$ and then from $n + 1$ to j , giving the length in the final term of Eq. (5.7). For the second case, the constrained shortest path is the same as the one using nodes 1 to n as intermediate nodes, yielding the length of the first term in the minimization of Eq. (5.7).

Since each of the N steps above must be executed for each pair of nodes, the computation involved in the Floyd–Warshall algorithm is $O(N^3)$, the same as if the Dijkstra algorithm were repeated for each possible choice of source node.

5.2.4 Distributed Asynchronous Bellman–Ford Algorithm

Consider a routing algorithm that routes each packet along a shortest path from the packet's origin to its destination, and suppose that the link lengths may change either due to link failures and repairs, or due to changing traffic conditions in the network. It is therefore necessary to update shortest paths in response to these changes. We described several such algorithms in Section 5.1.2 in connection with the ARPANET and the TYMNET.

In this subsection we consider an algorithm similar to the one originally implemented in the ARPANET in 1969. It is closely related to the one used in DNA (DEC's Digital Network Architecture) [Wec80]. The idea is to compute the shortest distances from every node to every destination by means of a distributed version of the Bellman–Ford algorithm. An interesting aspect of this algorithm is that it requires very little information to be stored at the network nodes. Indeed, a node need not know the detailed network topology. It suffices for a node to know the length of its outgoing links and the identity of every destination.

We assume that each cycle has positive length. We also assume that the network always stays strongly connected, and that if (i, j) is a link, then (j, i) is also a link. We envision a practical situation where the lengths d_{ij} can change with time. In the analysis, however, it is assumed that the lengths d_{ij} are fixed while the initial conditions for the

algorithm are allowed to be essentially arbitrary. These assumptions provide an adequate model for a situation where the link lengths stay fixed after some time t_0 following a number of changes that occurred before t_0 .

We focus on the shortest distance D_i from each node i to a generic destination node taken for concreteness to be node 1. (In reality, a separate version of the algorithm must be executed for each destination node.) Under our assumptions, these distances are the unique solution of Bellman's equation,

$$D_i = \min_{j \in N(i)} [d_{ij} + D_j], \quad i \neq 1$$

$$D_1 = 0$$

where $N(i)$ denotes the set of current neighbors of node i (*i.e.*, the nodes connected with i via an up link). This was shown in Section 5.2.3; see the discussion following Eq. (5.6).

The Bellman–Ford algorithm is given by

$$D_i^{h+1} = \min_{j \in N(i)} [d_{ij} + D_j^h], \quad i \neq 1 \quad (5.8)$$

$$D_1^{h+1} = 0 \quad (5.9)$$

In Section 5.2.3, we showed convergence to the correct shortest distances for the initial conditions

$$D_i^0 = \infty, \quad i \neq 1 \quad (5.10)$$

$$D_1^0 = 0 \quad (5.11)$$

The algorithm is well suited for distributed computation since the Bellman–Ford iteration (5.8) can be executed at each node i in parallel with every other node. One possibility is for all nodes i to execute the iteration simultaneously, exchange the results of the computation with their neighbors, and execute the iteration again with the index h incremented by one. When the infinite initial conditions of Eqs. (5.10) and (5.11) are used, the algorithm will terminate after at most N iterations (where N is the number of nodes), with each node i knowing both the shortest distance, D_i , and the outgoing link on the shortest path to node 1.

Unfortunately, implementing the algorithm in such a synchronous manner is not as easy as it appears. There is a twofold difficulty here. First, a mechanism is needed for getting all nodes to agree to start the algorithm. Second, a mechanism is needed to abort the algorithm and start a new version if a link status or length changes as the algorithm is running. Although it is possible to cope successfully with these difficulties [Seg81], the resulting algorithm is far more complex than the pure Bellman–Ford method given by Eqs. (5.8) and (5.9).

A simpler alternative is to use an asynchronous version of the Bellman–Ford algorithm that does not insist on maintaining synchronism between nodes, and on starting with the infinite initial conditions of Eqs. (5.10) and (5.11). This eliminates the need for either an algorithm initiation or an algorithm restart protocol. The algorithm simply operates indefinitely by executing from time to time at each node $i \neq 1$ the iteration

$$D_i := \min_{j \in N(i)} [d_{ij} + D_j] \quad (5.12)$$

using the last estimates D_j received from the neighbors $j \in N(i)$, and the latest status and lengths of the outgoing links from node i . The algorithm also requires that each node i transmit from time to time its latest estimate D_i to all its neighbors. However, there is no need for either the iterations or the message transmissions to be synchronized at all nodes. Furthermore, no assumptions are made on the initial values D_j , $j \in N(i)$ available at each node i . The only requirement is that a node i will eventually execute the Bellman–Ford iteration (5.12) and will eventually transmit the result to the neighbors. Thus, a totally asynchronous mode of operation is envisioned.

It turns out that the algorithm is still valid when executed asynchronously as described above. It will be shown that if a number of link length changes occur up to some time t_0 , and no other changes occur subsequently, then within finite time from t_0 , the asynchronous algorithm finds the correct shortest distance of every node i . The shortest distance estimates available at time t_0 can be arbitrary numbers, so it is not necessary to reinitialize the algorithm after each link status or link length change.

The original 1969 ARPANET algorithm was based on the Bellman–Ford iteration (5.12), and was implemented asynchronously much like the scheme described above. Neighboring nodes exchanged their current shortest distance estimates D_j needed in the iteration every 625 msec, but this exchange was not synchronized across the network. Furthermore, the algorithm was not restarted following a link length change or a link failure. The major difference between the ARPANET algorithm and the one analyzed in the present subsection is that the link lengths d_{ij} were changing very frequently in the ARPANET algorithm, and the eventual steady state assumed in our analysis was seldom reached.

We now state formally the distributed, asynchronous Bellman–Ford algorithm and proceed to establish its validity. At each time t , a node $i \neq 1$ has available:

$D_j^i(t)$ = Estimate of the shortest distance of each neighbor node $j \in N(i)$ which was last communicated to node i

$D_i(t)$ = Estimate of the shortest distance of node i which was last computed at node i according to the Bellman–Ford iteration

The distance estimates for the destination node 1 are defined to be zero, so

$$D_1(t) = 0, \quad \text{for all } t \geq t_0$$

$$D_i^1(t) = 0, \quad \text{for all } t \geq t_0, \text{ and } i \text{ with } 1 \in N(i)$$

Each node i also has available the link lengths d_{ij} , for all $j \in N(i)$, which are assumed constant after the initial time t_0 . We assume that the distance estimates do not change except at some times $t_0, t_1, t_2, \dots$, with $t_{m+1} > t_m$, for all m , and $t_m \rightarrow \infty$ as $m \rightarrow \infty$, when at each processor $i \neq 1$, one of three events happens:

1. Node i updates $D_i(t)$ according to

$$D_i(t) := \min_{j \in N(i)} [d_{ij} + D_j^i(t)]$$

and leaves the estimates $D_j^i(t)$, $j \in N(i)$, unchanged.

2. Node i receives from one or more neighbors $j \in N(i)$ the value of D_j which was computed at node j at some earlier time, updates the estimate D_j^i , and leaves all other estimates unchanged.
3. Node i is idle, in which case all estimates available at i are left unchanged.

Let T^i be the set of times for which an update by node i as in case 1 occurs, and T_j^i the set of times when a message is received at i from node j , as in case 2. We assume the following:

Assumption 1. Nodes never stop updating their own estimates and receiving messages from all their neighbors [i.e., T^i and T_j^i have an infinite number of elements for all $i \neq 1$ and $j \in N(i)$].

Assumption 2. Old distance information is eventually purged from the system [i.e., given any time $\bar{t} \geq t_0$, there exists a time $\tilde{t} > \bar{t}$ such that estimates D_j computed at a node j prior to time $\bar{t}$ are not received at any neighbor node i after time $\tilde{t}$].

The following proposition shows that the estimates $D_i(t)$ converge to the correct shortest distances within finite time. The proof (which can be skipped without loss of continuity) is interesting in that it serves as a model for proofs of validity of several other asynchronous distributed algorithms. (See the convergence proof of the algorithm of Section 5.3.3 and references [Ber82a], [Ber83], and [BeT89].)

Proposition. Suppose that each cycle has positive length and let the initial conditions $D_i(t_0)$, $D_j^i(t_0)$ be arbitrary numbers for $i = 2, \dots, N$ and $j = 2, \dots, N$. Then there is a time t_m such that

$$D_i(t) = D_i, \quad \text{for all } t \geq t_m, \quad i = 1, \dots, N$$

where D_i is the correct shortest distance from node i to the destination node 1.

Proof: The idea of the proof is to define for every node i two sequences $\{\underline{D}_i^k\}$ and $\{\overline{D}_i^k\}$ with

$$\underline{D}_i^k \leq \underline{D}_i^{k+1} \leq D_i \leq \overline{D}_i^{k+1} \leq \overline{D}_i^k \quad (5.13)$$

and

$$\underline{D}_i^k = D_i = \overline{D}_i^k, \quad \text{for all } k \text{ sufficiently large} \quad (5.14)$$

These sequences are obtained from the Bellman–Ford algorithm by starting at two different initial conditions. It is then shown that for every k and i , the estimates $D_i(t)$ satisfy

$$\underline{D}_i^k \leq D_i(t) \leq \overline{D}_i^k, \quad \text{for all } t \text{ sufficiently large} \quad (5.15)$$

A key role in the proof is played by the monotonicity property of the Bellman–Ford iteration. This property states that if for some scalars $\overline{D}_j$ and $\tilde{D}_j$,

$$\overline{D}_j \geq \tilde{D}_j, \quad \text{for all } j \in N(i)$$

then the direction of the inequality is preserved by the iteration, that is,

$$\min_{j \in N(i)} [d_{ij} + \overline{D}_j] \geq \min_{j \in N(i)} [d_{ij} + \tilde{D}_j]$$

A consequence of this property is that if D_i^k are sequences generated by the Bellman–Ford iteration (5.8) and (5.9) starting from some initial condition $D_i^0, i = 1, \dots, N$, and we have $D_i^1 \geq D_i^0$ for each i , then $D_i^{k+1} \geq D_i^k$, for each i and k . Similarly, if $D_i^1 \leq D_i^0$, for each i , then $D_i^{k+1} \leq D_i^k$, for each i and k .

Consider the Bellman–Ford algorithm given by Eqs. (5.8) and (5.9). Let $\overline{D}_i^k, i = 1, \dots, N$ be the k^{th} iterate of this algorithm when the initial condition is

$$\overline{D}_i^0 = \infty, \quad i \neq 1 \quad (5.16a)$$

$$\overline{D}_1^0 = 0 \quad (5.16b)$$

and let $\underline{D}_i^k, i = 1, \dots, N$ be the k^{th} iterate when the initial condition is

$$\underline{D}_i^0 = D_i - \delta, \quad i \neq 1 \quad (5.17a)$$

$$\underline{D}_1^0 = 0 \quad (5.17b)$$

where δ is a positive scalar large enough so that $\underline{D}_i^0$ is smaller than all initial node estimates $D_i(t_0)$, $D_j^j(t_0)$, $j \in N(i)$ and all estimates D_i that were communicated by node i before t_0 and will be received by neighbors of i after t_0 .

Lemma. The sequences $\{\underline{D}_i^k\}$ and $\{\overline{D}_i^k\}$ defined above satisfy Eqs. (5.13) and (5.14).

Proof: Relation (5.13) is shown by induction using the monotonicity property of the Bellman–Ford iteration, and the choice of initial conditions above. To show Eq. (5.14), first note that from the convergence analysis of the Bellman–Ford algorithm of the preceding section, for all i ,

$$\overline{D}_i^k = D_i, \quad k \geq N - 1 \quad (5.18)$$

so only $\underline{D}_i^k = D_i$, for sufficiently large k , remains to be established. To this end, we first note that by an induction argument it follows that $\underline{D}_i^k$ is, for every k , the length of a walk from i to some other node j involving no more than k links, plus $\underline{D}_j^0$. Let $L(i, k)$ and $n(i, k)$ be the length and the number of links of this walk, respectively. We can decompose this walk into a path from i to j involving no more than $N - 1$ links plus a number of cycles each of which must have positive length by our assumptions. Therefore, if $\lim_{k \rightarrow \infty} n(i, k) = \infty$ for any i , we would have $\lim_{k \rightarrow \infty} L(i, k) = \infty$, which is impossible since $D_i \geq \underline{D}_i^k = L(i, k) + \underline{D}_j^0$ and D_i is finite (the network is assumed strongly connected). Therefore, $n(i, k)$ is bounded with respect to k and it follows that the number of all possible values of $\underline{D}_i^k$, for $i = 1, \dots, N$ and $k = 0, 1, \dots$, is finite. Since $\underline{D}_i^k$ is monotonically nondecreasing in k for all i , it follows that for some h

$$\underline{D}_i^{h+1} = \underline{D}_i^h, \quad \text{for all } i \quad (5.19)$$

Therefore, the scalars $\underline{D}_i^h$ satisfy Bellman's equation, which as shown in Section 5.2.3,

has as its unique solution the shortest distances D_i . It follows that

$$\underline{D}_i^k = D_i, \quad \text{for all } i, \quad k \geq h \quad (5.20)$$

Equations (5.18) and (5.20) show Eq. (5.14).

Q.E.D.

We now complete the proof of the proposition by showing by induction that for every k there exists a time $t(k)$ such that for all $t \geq t(k)$,

$$\underline{D}_i^k \leq D_i(t) \leq \overline{D}_i^k, \quad i = 1, 2, \dots, N \quad (5.21)$$

$$\underline{D}_j^k \leq D_j^i(t) \leq \overline{D}_j^k, \quad j \in N(i), \quad i = 1, 2, \dots, N \quad (5.22)$$

and for all $t \in T_j^i$, $t \geq t(k)$,

$$\underline{D}_j^k \leq D_j[\tau_j^i(t)] \leq \overline{D}_j^k, \quad j \in N(i), \quad i = 1, 2, \dots, N \quad (5.23)$$

where $\tau_j^i(t)$ is the largest time at which the estimate $D_j^i(t)$ available at node i at time t was computed using the iteration $D_j := \min_{k \in N(j)} [d_{jk} + D_k]$ at node j . [Formally, $\tau_j^i(t)$ is defined as the largest time in T_j^i that is less than t , and is such that $D_j[\tau_j^i(t)] = D_j^i(t)$.]

Indeed, the induction hypothesis is true for $k = 0$ [for $t(0) = t_0$] because δ was chosen large enough in the initial conditions (5.17). Assuming that the induction hypothesis is true for a given k , it will be shown that there exists a time $t(k+1)$ with the required properties. Indeed, from relation (5.22) and the monotonicity of the Bellman–Ford iteration, we have that for every $t \in T^i$, $t \geq t(k)$ [i.e., a time t for which $D_i(t)$ is updated via the Bellman–Ford iteration],

$$\underline{D}_i^{k+1} \leq D_i(t) \leq \overline{D}_i^{k+1}$$

Therefore, if $t'(k)$ is the smallest time $t \in T^i$, with $t \geq t(k)$, then

$$\underline{D}_i^{k+1} \leq D_i(t) \leq \overline{D}_i^{k+1}, \quad \text{for all } t \geq t'(k), \quad i = 1, 2, \dots, N \quad (5.24)$$

Since $\tau_j^i(t) \rightarrow \infty$ as $t \rightarrow \infty$ (by Assumptions 1 and 2) we can choose a time $t(k+1) \geq t'(k)$ so that $\tau_j^i(t) \geq t'(k)$, for all $i, j \in N(i)$, and $t \geq t(k+1)$. Then, in view of Eq. (5.24), for all $t \geq t(k+1)$,

$$\underline{D}_j^{k+1} \leq D_j^i(t) \leq \overline{D}_j^{k+1}, \quad j \in N(i), \quad i = 1, 2, \dots, N \quad (5.25)$$

and for all $t \in T_j^i$, $t \geq t(k+1)$,

$$\underline{D}_j^{k+1} \leq D_j[\tau_j^i(t)] \leq \overline{D}_j^{k+1}, \quad j \in N(i), \quad i = 1, 2, \dots, N \quad (5.26)$$

This completes the induction proof.

Q.E.D.

The preceding analysis assumes that there is a path from every node to node 1. If not, the analysis applies to the portion of the network that is connected with node 1. For a node i that belongs to a different network component than node 1 and has at least one neighbor (so that it can execute the algorithm), it can be seen that $D_i(t) \rightarrow \infty$ as $t \rightarrow \infty$. This property can be used by a node to identify the destinations to which it is not connected.

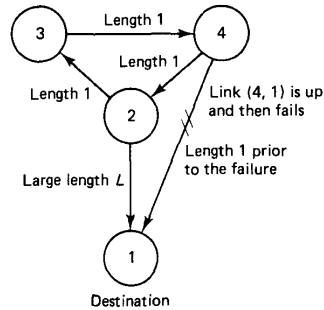


Figure 5.36 Example where the number of iterations of the (synchronous) Bellman–Ford algorithm is excessive. Suppose that the initial conditions are the shortest distances from nodes 2, 3, and 4 to node 1 before link (4,1) fails ($D_2 = 3$, $D_3 = 2$, $D_4 = 1$). Then, after link (4,1) fails, nearly L iterations will be required before node 2 realizes that its shortest path to node 1 is the direct link (2,1). This is an example of the so-called “bad news phenomenon,” whereby the algorithm reacts slowly to a sudden increase in one or more link lengths.

We close this section with a discussion of two weaknesses of the asynchronous Bellman–Ford method. The first is that in the worst case, the algorithm may require an excessive number of iterations to terminate (see the example of Fig. 5.36). This is not due to the asynchronous nature of the algorithm, but rather to the arbitrary choice of initial conditions [an indication is provided by the argument preceding Eq. (5.19)]. A heuristic remedy is outlined in Problem 5.6; for other possibilities, see [JaM82], [Gar87], and [Hum91]. The second weakness, demonstrated in the example of Fig. 5.37, is that in the worst case, the algorithm requires an excessive number of message transmissions. The example of Fig. 5.37 requires an unlikely sequence of events. An analysis given in [TsS90] shows that under fairly reasonable assumptions, the average number of messages required by the algorithm is bounded by a polynomial in N .

5.2.5 Stability of Adaptive Shortest Path Routing Algorithms

We discussed in earlier sections the possibility of using link lengths that reflect the traffic conditions on the links in the recent past. The idea is to assign a large length to a congested link so that the shortest path algorithm will tend to exclude it from a routing path. This sounds attractive at first, but on second thought one gets alerted to the possibility of oscillations. We will see that this possibility is particularly dangerous in datagram networks.

Stability issues in datagram networks. For a simple example of oscillation, consider a datagram network, and suppose that there are two paths along which an origin can send traffic to a destination. Routing along a path during some time period will increase its length, so the other path will tend to be chosen for routing in the next time period, resulting in an oscillation between the two paths. It turns out that a far worse type of oscillation is possible, as illustrated in the following example.

Example 5.4

Consider the 16-node network shown in Fig. 5.38, where node 16 is the only destination. Let the traffic input (in data units/sec) at each node $i = 1, \dots, 7, 9, \dots, 15$ be one unit and

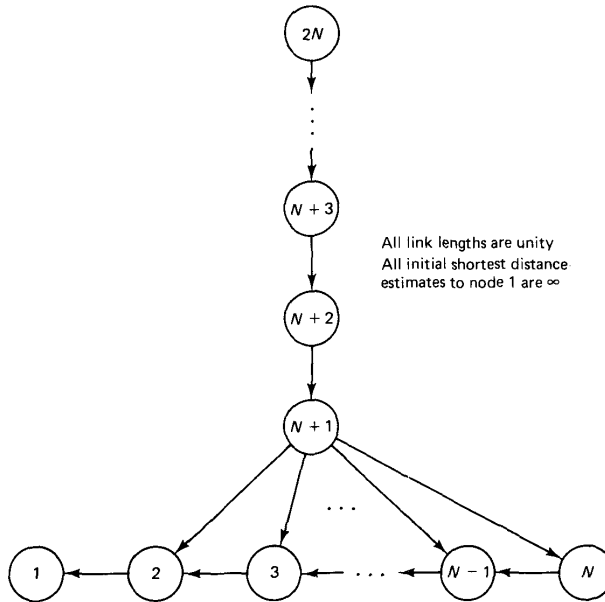


Figure 5.37 Example where the asynchronous version of the Bellman-Ford algorithm requires many more message transmissions than the synchronous version. The initial estimates of shortest distance of all nodes is ∞ . Consider the following sequence of events, where all messages are received with zero delay:

1. Node 2 updates its shortest distance and communicates the result to 3. Node 3 updates and communicates the result to 4. ... Node $N-1$ updates and communicates the result to N . Node N updates and communicates the result to $N+1$. (These are $N-1$ messages.)
2. Node $N+1$ updates and communicates the result to $N+2$ Node $2N-1$ updates and communicates the result to $2N$. Node $2N$ updates. (These are $N-1$ messages.)
3. For $i = N-1, N-2, \dots, 2$, in that order, node i communicates its update to $N+1$ and sequence 2 above is repeated. [These are $N(N-2)$ messages.]

The total number of messages is $N^2 - 2$. If the algorithm were executed synchronously, with a message sent by a node to its neighbors only when its shortest distance estimate changes, the number of messages needed would be $3(N-1) - 1$. The difficulty in the asynchronous version is that a lot of unimportant information arrives at node $N+1$ early and triggers a lot of unnecessary messages starting from $N+1$ and proceeding all the way to $2N$.

let the traffic input of node 8 be $\epsilon > 0$, where ϵ is very small. Assume that the length of link (i, j) is

$$d_{ij} = F_{ij}$$

where F_{ij} is the arrival rate at the link counting input and relayed traffic. Suppose that all nodes compute their shortest path to the destination every T seconds using as link lengths the arrival rates F_{ij} during the preceding T seconds, and route all their traffic along that path

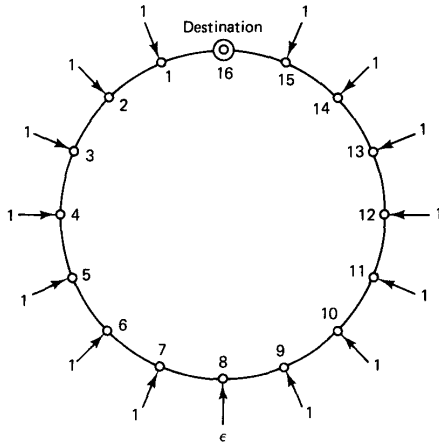
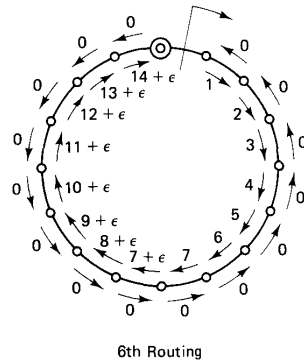
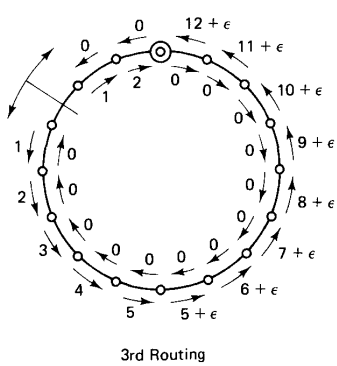
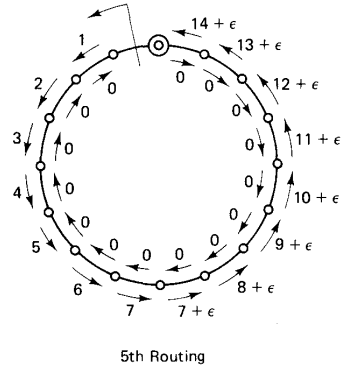
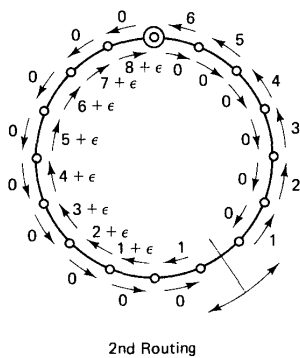
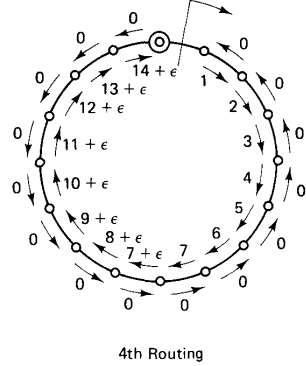
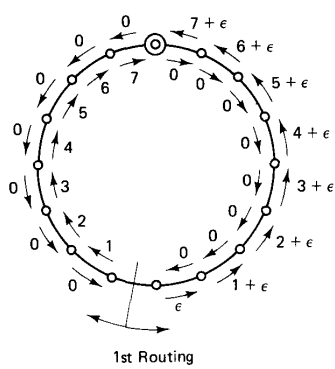


Figure 5.38 Sixteen-node ring network of Example 5.4. Node 16 is the only destination.

for the next T seconds. Assume that we start with nodes 1 through 7 routing clockwise and nodes 8 through 15 routing counterclockwise. This is a rather good routing, balancing the traffic input between the two directions. Figure 5.39 shows the link rates corresponding to the initial and subsequent shortest path routings. Thus, after three shortest path updates, the algorithm is locked into an oscillatory pattern whereby all traffic swings from the clockwise to the counterclockwise direction and back at alternate updates. This is certainly the worst type of routing performance that could occur.

The difficulty in Example 5.4 is due to the fact that link arrival rates depend on routing, which in turn depends on arrival rates via the shortest path calculation, with a feedback effect resulting. This is similar to the stability issue in feedback control theory, and can be analyzed using a related methodology [Ber82b]. Actually, it can be shown that the type of instability illustrated above will occur generically if the length d_{ij} of link (i, j) increases continuously and monotonically with the link arrival rate F_{ij} , and $d_{ij} = 0$ when $F_{ij} = 0$. It is possible to damp the oscillations by adding a positive constant to the link length so that $d_{ij} = \alpha > 0$ when $F_{ij} = 0$. The scalar α (link length at zero load) is known as a *bias* factor.

Figure 5.39 Oscillations in a ring network for link lengths d_{ij} equal to the link arrival rates F_{ij} . Each node sends one unit of input traffic to the destination except for the middle node 8, which sends $\epsilon > 0$, where ϵ is very small. The numbers next to the links are the link rates in each of the two directions. As an example of the shortest path calculations, at the first iteration the middle node 8 computes the length of the clockwise path as 28 ($= 0 + 1 + 2 + \dots + 7$), and the length of the counterclockwise path as $28 + 8\epsilon$ ($= \epsilon + 1 + \epsilon + 2 + \epsilon + \dots + 7 + \epsilon$), and switches its traffic to the shortest (clockwise) path at the second routing. The corresponding numbers for node 9 are 28 and $28 + 7\epsilon$, so node 9 also switches its traffic to the clockwise path. All other nodes find that the path used at the first routing is shortest, and therefore they do not switch their traffic to the other path.



Oscillations in the original ARPANET algorithm discussed in Section 5.1.2 were damped by using a substantial bias factor. Indeed, if α is large enough relative to the range of link lengths, it can be seen that the corresponding shortest paths will have the minimum possible number of links to the destination. This is static routing, which cannot exhibit any oscillation of the type seen earlier, but is also totally insensitive to traffic congestion. In the current ARPANET algorithm described in Section 5.1.2, the bias α in effect equals the sum of the average packet transmission time, the processing delay, and the propagation delay along a link. With the 1987 algorithm modifications ([KhZ89] and [ZVK89]), the bias became fairly large relative to the range of allowable link lengths. As a result the algorithm became quite stable but also less sensitive to congestion. The second possibility to damp oscillations is to introduce a mechanism for averaging the lengths of links over a time period spanning more than one shortest path update. This tends to improve the stability of the algorithm, albeit at the expense of reducing its speed of response to congestion. It turns out that asynchronous shortest path updating by the network nodes results in a form of length averaging that is beneficial for stability purposes. (See [MRR78], [Ber79b], and [Ber82b] for further discussion.)

Stability issues in virtual circuit networks. The oscillatory behavior exhibited above is associated principally with datagram networks. It will be shown that oscillations are less severe in virtual circuit networks. A key feature of a datagram network in this regard is that each packet of a user pair is not required to travel on the same path as the preceding packet. Therefore, the time that an origin-destination pair will continue to use a shortest path after it is changed due to a routing update is very small. As a result, a datagram network reacts very fast to a shortest path update, with all traffic switching to the new shortest paths almost instantaneously.

The situation is quite different in a virtual circuit network, where every session is assigned a fixed communication path at the time it is first established. There the average duration of a virtual circuit is often large relative to the shortest path updating period. As a result, the network reaction to a shortest path update is much more gradual since old sessions continue to use their established communication paths and only new sessions are assigned to the most recently calculated shortest paths.

As the following example demonstrates, the critical parameters for stability are the “speed” of the virtual circuit arrival and departure processes, and the frequency with which shortest paths are updated.

Example 5.5

Consider the simple two-link network with one origin and one destination shown in Fig. 5.40(a). Suppose that the arrival rate is r bits/sec. Assuming that the two links have equal capacity C , an “optimal” routing algorithm should somehow divide the input r equally between the two links, thereby allowing a throughput up to $2C$.

Consider a typical adaptive algorithm based on shortest paths for this example network. The algorithm divides the time axis into T -second intervals. It measures the average arrival rate (bits/second) on both links during each T -second interval, and directs all traffic (datagrams or virtual circuits) generated during every T -second interval along the link that had the smallest rate during the preceding T -second interval.

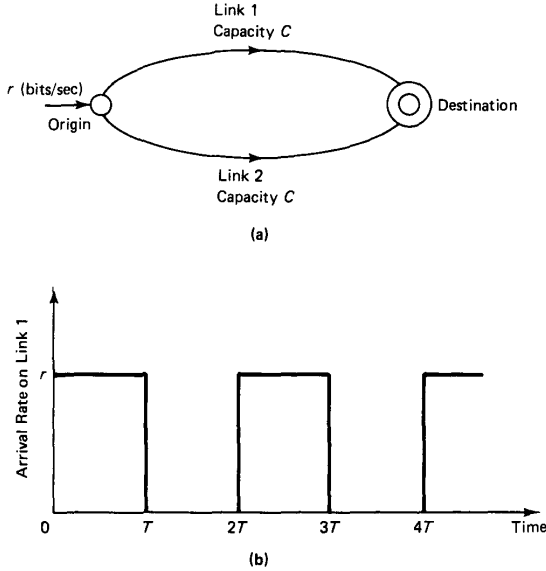


Figure 5.40 (a) Two-link network of Example 5.5. (b) Arrival rate on link 1 in Example 5.5 using the shortest path rule in the datagram case. Essentially, only one path is used for routing at any one time if the shortest path update period is much larger than the time required to empty the queue of waiting packets at the time of an update.

If the network uses datagrams, then, assuming that T is much larger than the time required to empty the queue of waiting packets at the time of an update, each link will essentially carry either no traffic or all the input traffic r at alternate time intervals, as shown in Fig. 5.40(b).

Next consider the case where the network uses virtual circuits, which are generated according to a Poisson process at a rate λ per second. Each virtual circuit uses the link on which it was assigned by the routing algorithm for its entire duration, assumed exponentially distributed with mean $1/\mu$ seconds. Therefore, according to the $M/M/\infty$ queueing results (cf. Section 3.4.2), the number of active virtual circuits is Poisson distributed with mean λ/μ . If γ is the average communication rate (bits/sec) of a virtual circuit, we must have $r = (\lambda/\mu)\gamma$, or

$$\gamma = \frac{r\mu}{\lambda} \quad (5.27)$$

Suppose that the shortest path updating interval T is small relative to the average duration of the virtual circuits $1/\mu$. Then, approximately a fraction μT of the virtual circuits that were carried on each link at the beginning of a T -second interval will be terminated at the end of the interval. There will be λT virtual circuits on the average added on the link that carried the least traffic during the preceding interval. This amounts to an added arrival rate of $\gamma\lambda T$ bits/sec or, using the fact that $\gamma = r\mu/\lambda$ [cf. Eq. (5.27)], $r\mu T$ bits/sec. Therefore, the average rates x_1^k and x_2^k (bits/sec) on the two links at the k^{th} interval will evolve approximately according to

$$x_i^{k+1} = \begin{cases} (1 - \mu T)x_i^k + r\mu T, & \text{if } i \text{ is shortest (i.e., } x_i^k = \min\{x_1^k, x_2^k\}) \\ (1 - \mu T)x_i^k, & \text{otherwise} \end{cases} \quad (5.28)$$

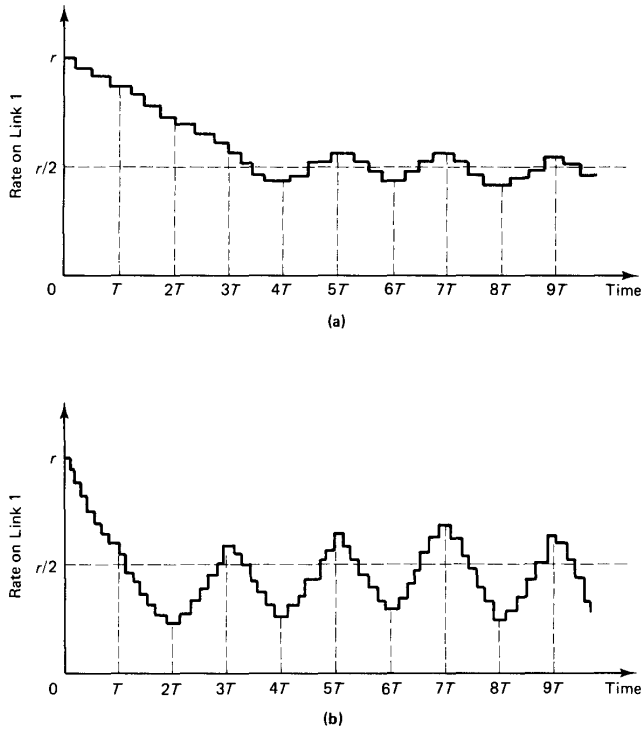


Figure 5.41 Arrival rate on link 1 in Example 5.5 when virtual circuits are used. (a) Virtual circuits last a long time, and the shortest paths are updated frequently (μT is small). (b) Virtual circuits last a short time, and the shortest paths are updated frequently (μT is moderate).

Figures 5.41 and 5.42 provide examples of the evolution of the average rate on link 1. From Eq. (5.28) it can be seen that as $k \rightarrow \infty$, the average rates x_1 and x_2 will tend to oscillate around $r/2$ with a magnitude of oscillation roughly equal to $r\mu T$. Therefore, if the average duration of a virtual circuit is large relative to the shortest path updating interval ($\mu T \ll 1$), the routing algorithm performs almost optimally, keeping traffic divided in nearly equal proportions among the two links. Conversely, if the product μT is large, the analysis above indicates considerable oscillation of the link rates. Figures 5.41 and 5.42 demonstrate the relation between μ , T , and the magnitude of oscillation.

Example 5.5 illustrates behavior that has been demonstrated analytically for general virtual circuit networks [GaB83]. It is also shown in [GaB83] that the average duration of a virtual circuit is a critical parameter for the performance of adaptive shortest path

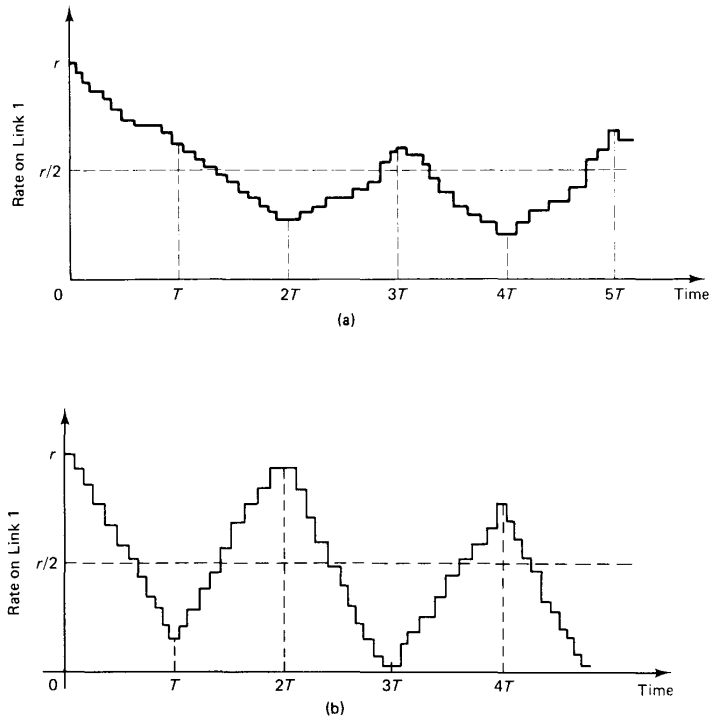


Figure 5.42 Arrival rate on link 1 in Example 5.5 when virtual circuits are used. (a) Virtual circuits last a long time, and the shortest paths are updated infrequently (μT is moderate). (b) Virtual circuits last a short time, and shortest paths are updated infrequently (μT is large). In the limit as $\mu \rightarrow \infty$, the datagram case is obtained.

routing. If it is very large, the rate of convergence of the algorithm will be slow, essentially because virtual circuits that are misplaced on congested paths persist for a long time [cf. Figs. 5.41(a) and 5.42(a)]. If it is very small, the shortest path update interval must be accordingly small for the oscillation around optimality to be small [cf. Fig. 5.41(b)]. Unfortunately, there is a practical limit on how small the update interval can be, because frequent updates require more overhead and because sufficient time between updates is needed to measure accurately the current link lengths. The problem with a large duration of virtual circuits would not arise if virtual circuits could be rerouted during their lifetime. In the Codex network described in Section 5.8, such rerouting is allowed, thereby resulting in an algorithm that is more efficient than the one described in this section.

5.3 BROADCASTING ROUTING INFORMATION: COPING WITH LINK FAILURES

A problem that often arises in routing is the transfer of control information from points in the network where it is collected to other points where it is needed. This problem is surprisingly challenging when links are subject to failure. In this section we explain the nature of the difficulties and we consider ways to address them. We also take the opportunity to look at several instructive examples of distributed algorithms.

One practical example of broadcasting routing information was discussed in Section 5.1.2 in connection with the current ARPANET algorithm. Here all link lengths are periodically broadcast to all nodes, which then proceed to update their routing tables through a shortest path computation. Other cases include situations where it is desired to alert higher layers about topological changes, or to update data structures that might be affected by changes in the network topology, such as a spanning tree used for broadcasting messages to all nodes. [The term *network topology* will be used extensively in this section. It is synonymous with the list of nodes and links of the network together with the status (up or down) of each link.]

Getting routing information reliably to the places where it is needed in the presence of potential link failures involves subtleties that are generally not fully appreciated. Here are some of the difficulties:

1. Topological update information must be communicated over links that are themselves subject to failure. Indeed, in some networks, one must provide for the event where portions of the network get disconnected from each other. As an example consider a *centralized* algorithm where all information regarding topological changes is communicated to a special node called the network control center (NCC). The NCC maintains in its memory the routing tables of the entire system, and updates them upon receipt of new topological information by using some algorithm that is of no concern here. It then proceeds to communicate to the nodes affected all information needed for local routing operations. Aside from the generic issue of collecting and disseminating information over failure-prone links, a special problem here is that the NCC may fail or may become disconnected from a portion of the network. In some cases it is possible to provide redundancy and ensure that such difficulties will occur very rarely. In other cases, however, resolution of these difficulties may be neither simple nor foolproof, so that it may be advisable to abandon the centralized approach altogether and adopt a distributed algorithm.
2. One has to deal with multiple topological changes (such as a link going down and up again within a short time), so there is a problem of distinguishing between old and new update information. As an example, consider a flooding method for broadcasting topological change information (cf. Section 5.1.2). Here a node monitors the status of all its outgoing links, and upon detecting a change, sends a packet to all its neighbors reporting that change. The neighbors send this packet to their neighbors, and so on. Some measures are needed to prevent update packets from circulating indefinitely in the network, but this question is discussed

later. Flooding works when there is a single topological change, but can fail in its pure form when there are multiple changes, as shown in the example of Fig. 5.43. Methods for overcoming this difficulty are discussed later in this section, where it is shown that apparently simple solutions sometimes involve subtle failure modes.

3. Topological update information will be processed by some algorithm that computes new routing paths (*e.g.*, a shortest path algorithm). It is possible, however, that new topological update information arrives as this algorithm is running. Then, either the algorithm must be capable of coping with changes in the problem data as it executes, or else it must be aborted and a new version must be started upon receipt of new data. Achieving either of these may be nontrivial, particularly when the algorithm is distributed.
4. The repair of a single link can cause two parts of the network which were disconnected to reconnect. Each part may have out-of-date topology information about the other part. The algorithm must ensure that eventually the two parts agree and adopt the correct network topology.

The difficulties discussed above arise also in the context of broadcasting information related to the congestion status of each link. An important difference, however, is that incorrect congestion information typically has less serious consequences than does incorrect topological information; the former can result at worst in the choice of inferior routes, while the latter can result in the choice of nonexistent routes. Thus, one can afford to allow for a small margin of error in an algorithm that broadcasts routing information other than topological updates, if that leads to substantial algorithmic simplification or reduction in communication overhead. Note, however, that in some schemes, such as the ARPANET flooding algorithm to be discussed shortly, topological update information is embedded within congestion information, and the same algorithm is used to broadcast both.

As we approach the subject of broadcasting topological update information, we must first recognize that it is impossible for every node to know the correct network topology at all times. Therefore, the best that we can expect from an algorithm is that it

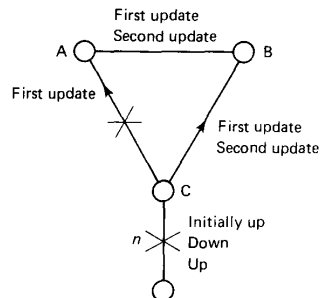


Figure 5.43 Example where the flooding algorithm in its pure form fails. Link n is initially up, then it goes down, then up again. Suppose that the two updates travel on the path CBA faster than the first update travels on the link CA . Suppose also that link CA fails after the first update, but before the second update travels on it. Then, the last message received by A asserts that link n is down, while the link is actually up. The difficulty here is that outdated information is mistakenly taken to be new.

can cope successfully with any finite number of topological changes within finite time. By this we mean that if a finite number of changes occur up to some time and no other changes occur subsequently, all nodes within each connected portion of the network should know the correct status of each link in that portion within finite time.

In the discussion of validity of various schemes in the sense described above, we will assume the following:

1. Network links preserve the order and correctness of transmissions. Furthermore, nodes maintain the integrity of messages that are stored in their memory.
2. A link failure is detected by both end nodes of the link, although not necessarily simultaneously. By this we mean that any link declared down by one end node will also eventually be declared down by the other, before the first end node declares it up again.
3. There is a data link protocol for recognizing, at the ends of a failed link, when the link is again operational. If one of the end nodes declares the link to be up, then within finite time, either the opposite end node also declares the link up or the first declares it down again.
4. A node can crash, in which case each of its incident links is, within finite time, declared down by the node at the opposite end of the link.

The preceding assumptions provide a basis for discussion and analysis of specific schemes, but are not always satisfied in practice. For example, on rare occasions, Assumption 1 is violated because damaged data frames may pass the error detection test of Data Link Control, or because a packet may be altered inside a node's memory due to hardware malfunction. Thus, in addition to analysis of the normal case where these assumptions are satisfied, one must weigh the consequences of the exceptional case where they are not. Keep in mind here that topology updating is a low-level algorithm on which other algorithms rely for correct operation.

Note that it is not assumed that the network will always remain connected. In fact, the topology update algorithm together with some protocol for bringing up links should be capable of starting up the network from a state where all links are down.

5.3.1 Flooding—The ARPANET Algorithm

Flooding was described previously as an algorithm whereby a node broadcasts a topological update message to all nodes by sending the message to its neighbors, which in turn send the message to their neighbors, and so on. Actually, the update messages may include other routing-related information in addition to link status. For example, in the ARPANET, each message originating at a node includes a time average of packet delay on each outgoing link from the node. The time average is taken over 10-sec intervals. The time interval between update broadcasts by the node varies in the ARPANET from 10 to 60 sec, depending on whether there has been a substantial change in any single-link average delay or not. In addition, a message is broadcast once a status change in one of the node's outgoing links is detected.

A serious difficulty with certain forms of flooding is that they may require a large number of message transmissions. The example of Fig. 5.44 demonstrates the difficulty and shows that it is necessary to store enough information in update messages and network nodes to ensure that each message is transmitted by each node only a finite number of times (and preferably only once). In the ARPANET, update messages from each node are marked by a *sequence number*. When a node j receives a message that originated at some node i , it checks to see if its sequence number is greater than the sequence number of the message last received from i . If so, the message together with its sequence number is stored in memory, and its contents are transmitted to all neighbors of j except the one from which the message was received. Otherwise, the message is discarded. Assuming that the sequence number field is large enough, one can be sure that wraparound of the sequence numbers will never occur under normal circumstances. (With a 48-bit field and one update per millisecond, it would take more than 5000 years for wraparound to occur.)

The use of sequence numbers guarantees that each topological update message will be transmitted at most once by each node to its neighbors. Also, it resolves the difficulty of distinguishing between new and old information, which was discussed earlier and illustrated in the example of Fig. 5.43. Nonetheless, there are subtle difficulties with sequence numbers relating to exceptional situations, such as when portions of the network become disconnected or when equipment malfunctions. For example, when a complete or partial network reinitialization takes place (perhaps following a crash of one or more nodes), it may be necessary to reset some sequence numbers to zero, since a node may not remember the sequence number it was using in the past. Consider a time period when two portions of the network are disconnected. During this period, each portion cannot learn about topological changes occurring in the other portion. If sequence numbers are

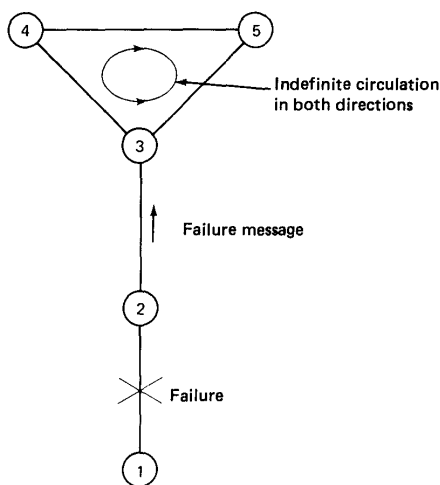


Figure 5.44 Example of a form of flooding where the transmission of messages never terminates. The rule is that a node that receives a message relays it to all of its neighbor nodes except the one from which the message was received. A failure of link (1, 2) is communicated to node 3, which triggers an indefinite circulation of the failure message along the loop (3,4,5) in both directions. This form of flooding works only if the network has no cycles.

reset in one of the two portions while they are disconnected, there may be no way for the two portions to figure out the correct topology after they connect again by relying on the sequence number scheme alone. Another concern is that a sequence number could be altered either inside a node's memory due to a malfunction, or due to an undetected error in transmission. Suppose, for example, that an update message originating at node i reaches another node j , has its sequence number changed accidentally to a high number, and is flooded throughout the network. Then the erroneous sequence number will take over, and in the absence of a correcting mechanism, node i will not be listened to until its own (correct) sequence number catches up with the wrong one. Similarly, the sequence number of a node that is accidentally set to a high number inside the node itself can wrap around, at which time all subsequent update messages from the node will be ignored by the network.

The ARPANET resolves the problems associated with sequence numbers by using two devices:

1. Each update message includes an *age field*, which indicates the amount of time that the message has been circulating inside the network. Each node visited by a message keeps track of the message's arrival time and increments its age field (taking into account the transmission and propagation time) before transmitting it to its neighbors. (The age field is incremented slightly differently in the ARPANET; see [Per83].) Thus, a node can calculate the age of all messages in its memory at any time. A message whose age exceeds a prespecified limit is not transmitted further. Otherwise, the message is transmitted to all neighbor nodes except the one from which the message was received.
2. Each node is required to transmit update messages *periodically* in addition to the messages transmitted when a link status change is detected. (There is at least one of these from every node every 60 sec.)

The rule used regarding the age field is that an "aged" message is superseded by a message that has not "aged" yet, regardless of sequence numbers. (A message that has not "aged" yet is superseded only if the new message has a higher sequence number.) This rule guarantees that damaged or incorrect information with a high sequence number will not be relied upon for too long. The use of periodic update messages guarantees that up-to-date information will become available within some fixed time following reconnection of two portions of the network. Of course, the use of periodic updates implies a substantial communication overhead penalty and is a major drawback of the ARPANET scheme. The effect of this, however, is lessened by the fact that the update packets include additional routing-related information, namely the average packet delay on the node's outgoing links since the preceding update.

5.3.2 Flooding without Periodic Updates

It is possible to operate the flooding scheme with sequence numbers correctly without using an age field or periodic updates. The following simple scheme (suggested by P. Humblet) improves on the reliability of the basic sequence number idea by providing

a mechanism for coping with node crashes and some transmission errors. It requires, however, that the sequence number field be so large that wraparound never occurs. The idea is to modify the flooding rules so that the difficulties with reconnecting disconnected network components are dealt with adequately. We describe briefly the necessary modifications, leaving some of the details for the reader to work out. We concentrate on the case of broadcasting topological information, but a similar scheme can be constructed to broadcast other routing information.

As before, we assume that when the status of an outgoing link from a node changes, that node broadcasts an update message containing the status of all its outgoing links to all its current neighbors. The message is stamped by a sequence number which is either zero or is larger by one than the last sequence number used by the node in the past. There is a restriction here, namely that a zero sequence number is allowed only when the node is recovering from a crash (defined as a situation where all of the node's incident links are down and the node is in the process of bringing one or more of these links up). As before, there is a separate sequence number associated with each origin node.

The first flooding rule modification has to do with bringing up links that have been down. When this happens, the end nodes of the link, in addition to broadcasting a regular update message on all of their outgoing links, should exchange their current views of the network topology. By this we mean that they should send to each other all the update messages stored in their memory that originated at other nodes together with the corresponding sequence numbers. These messages are then propagated through the network using the modified flooding rules described below. This modification guarantees that, upon reconnection, the latest update information will reach nodes in disconnected network portions. It also copes with a situation where, after a crash, a node does not remember the sequence number it used in the past. Such a node must use a zero sequence number following the crash, and then, through the topology exchange with its neighbors that takes place when its incident links are brought up, become aware of the highest sequence number present in the other nodes' memories. The node can then increment that highest sequence number and flood the network with a new update message.

For the scheme to work correctly we modify the flooding rule regarding the circumstances under which an update message is discarded. To this end we order, for each node i , the topological update messages originated at i . For two such messages A and B , we say that $A > B$ if A has a greater sequence number than B , or if A and B have identical sequence numbers and the content of A is greater than the content of B according to some lexicographic rule (e.g., if the content of A interpreted as a binary number is greater than the similarly interpreted content of B). Any two messages A and B originating at the same node can thus be compared in the sense that $A > B$, or $B > A$, or $A = B$, the last case occurring only if the sequence numbers and contents of A and B are identical.

Suppose that node j receives an update message A that has originated at node i , and let B be the message originated at node i and currently stored in node j 's memory. The message is discarded if $A < B$ or $A = B$. If $A > B$, the flooding rule is now as follows:

1. If $j \neq i$, node j stores A in its memory in place of B and sends a copy of A on all its incident links except the one on which A was received.

2. If $j = i$ (i.e., node i receives a message it issued some time in the past), node i sends to all its neighbors a new update message carrying the current status of all its outgoing links together with a sequence number that is 1 plus the sequence number of A .

Figure 5.45 shows by example why it is necessary to compare the contents of A and B in the case of equal sequence numbers.

The algorithm of this section suffers from a difficulty that seems generic to all event-driven algorithms that do not employ periodic updates: it is vulnerable to memory and transmission errors. Suppose, for example, that a message's sequence number is altered to a very high number either during a transmission or while residing inside a node's memory. Then the message's originating node may not be listened to until its sequence number catches up with the high numbers that are unintentionally stored in some nodes' memories. It turns out that this particular difficulty can be corrected by modifying the flooding rule discussed earlier. Specifically, if node j receives A and has B with $A < B$ in its memory, then message A is discarded as in the earlier rule, but in addition message B is sent back to the neighbor from which A was received. (The neighbor will then propagate B further.) Suppose now that a node issues an update message carrying a sequence number which is less than the sequence number stored in some other node's memory. Then, because of the modification just described, if the two nodes are connected by a path of up links, the higher sequence number (call it k) will be propagated back to the originator node, which can then issue a new update message with sequence number $k + 1$ according to rule 2 above. Unfortunately, there is still a problem associated with a possible sequence number wraparound (e.g., when k above is the highest number possible within the field of possible sequence numbers). There is also still the problem of update messages themselves being corrupted by memory or

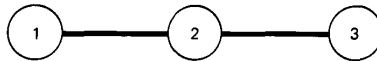


Figure 5.45 Example showing why it is necessary to compare the contents of A and B in the case of equal sequence numbers in the flooding algorithm without periodic updates of Section 5.3.2. Here, there are three nodes, 1, 2, and 3, connected in tandem with two (undirected) links (1,2) and (2,3). Suppose, initially, that both links are up, and the messages stored in all the nodes' memories carry the correct link status information and have a sequence number of zero. Consider a scenario whereby link (2,3) goes down, then link (1,2) goes down, and then link (2,3) comes up while node 2 resets its sequence number to zero. Then, nodes 2 and 3 exchange their (conflicting) view of the status of the directed links (1,2) and (2,1), but if the earlier flooding rules are used, both nodes discard each other's update message since it carries a sequence number zero which is equal to the one stored in their respective memories. By contrast, if the steps of the modified flooding algorithm described above are traced, it is seen that (depending on the lexicographic rule used) either the (correct) view of node 2 regarding link (2,1) will prevail right away, or else node 2 will issue a new update message with sequence number 1 and its view will again prevail. Note that node 3 may end up with an incorrect view of the status of link (1,2). This happens because nodes 1 and 3 are not connected with a path of up links. For the same reason, however, the incorrect information stored by node 3 regarding the outgoing link of node 1 is immaterial.

transmission errors (rather than just their sequence numbers being corrupted). It seems that there is no clean way to address these difficulties other than ad hoc error recovery schemes using an age field and periodic updates (see Problem 5.13).

5.3.3 Broadcast without Sequence Numbers

In this subsection we discuss a flooding-like algorithm called the *Shortest Path Topology Algorithm* (SPTA), which, unlike the previous schemes, does not use sequence numbers and avoids the attendant reset and wraparound problems.

To understand the main idea of the algorithm, consider a generic problem where a special node, called the *center*, has some information that it wishes to broadcast to all other nodes. We model this information as a variable V taking values in some set. As a special case, V could represent the status of the incident links of the center, as in the topology broadcast problem. We assume that after some initial time, several changes of V and several changes of the network topology occur, but there is some time after which there are no changes of either V or the network topology. We wish to have an algorithm by which all nodes get to know the correct value of V after a finite time from the last change.

Consider a scheme whereby each node i stores in memory a value V_i , which is an estimate of the correct value of V . Each time V_i changes, its value is transmitted to all neighbors of i along each of the currently operating outgoing links of i . Also, when a link (i, j) becomes operational after being down, the end nodes i and j exchange their current values V_i and V_j . Each node i stores the last value received by each of its neighbors j , denoted as V_j^i .

We now give a general rule for changing the estimate V_i of each node i . Suppose that we have an algorithm running in the network that maintains a tree of directed paths from all nodes to the center (i.e., a tree rooted at the center; see Fig. 5.46). What we mean here is that this algorithm constructs such a tree within a finite time following the last change in the network topology. Every node i except for the center has a unique successor $s(i)$ in such a tree and we assume that this successor eventually becomes known to i . The rule for changing V_i then is for node i to make it equal to the value $V_{s(i)}^i$ last transmitted by the successor $s(i)$ within a finite time after either the current successor transmits a new value or the successor itself changes. It is evident under these assumptions that each node i will have the correct value V_i within a finite time following the last change in V or in the network topology.

It should be clear that there are many candidate algorithms that maintain a tree rooted at the center in the face of topological changes. One possibility is the asynchronous Bellman–Ford algorithm described in the preceding section, in which case the tree rooted at the center is a shortest path tree. The following SPTA algorithm constructs a separate shortest path tree for each node/center more economically than the asynchronous Bellman–Ford algorithm. The algorithm is given for the special case where the broadcast information is the network topology; in this case the tree construction algorithm and the topological change broadcast procedure are integrated into a single algorithm. The SPTA algorithm can be generalized, however, to broadcast any kind of information.

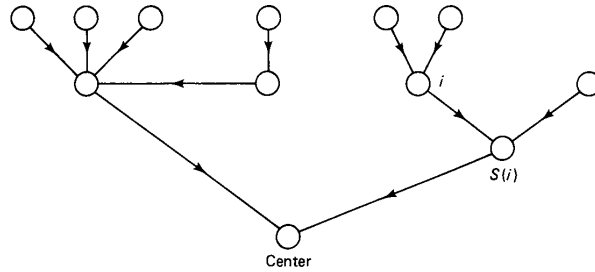


Figure 5.46 Information broadcast over a tree rooted at the center that initially holds the value V . Here there is a unique directed path from every processor to the center and a unique successor $s(i)$ of every node i on the tree. The idea of the broadcast algorithm is to maintain such a tree in the presence of topological changes and to require each node i to adopt (eventually) the latest received message from its successor $s(i)$ (i.e., $V_i = V_{s(i)}$). This guarantees that the correct information will eventually be propagated and adopted along the tree even after a finite number of topological changes that may require the restructuring of the tree several times.

Throughout this subsection we will make the same assumptions as with flooding about links preserving the order and correctness of transmissions, and nodes being able to detect incident link status changes.

The data structures that each node i maintains in the SPTA are (see Fig. 5.47):

1. The *main topology table* T_i where node i stores the believed status of every link. This is the “official” table used by the routing algorithm at the node. The main topology tables of different nodes may have different entries at different times. The goal of the SPTA is to get these tables to agree within finite time from the last topological change.
2. The *port topology tables* T_j^i . There is one such table for each neighbor node j . Node i stores the status of every link in the network, as communicated latest by neighbor node j , in T_j^i .

The algorithm consists of five simple rules:

Communication Rules

1. When a link status entry of a node’s main topology table changes, the new entry is transmitted on each of the node’s operating incident links.
2. When a failed link comes up, each of its end nodes transmits its entire main topology table to the opposite end node over that link. Upon reception of this table the opposite end node enters the new link status in its main and port topology tables.

Topology Table Update Rules

3. When an incident link of a node fails, the failed status is entered into the node’s main and port topology tables.

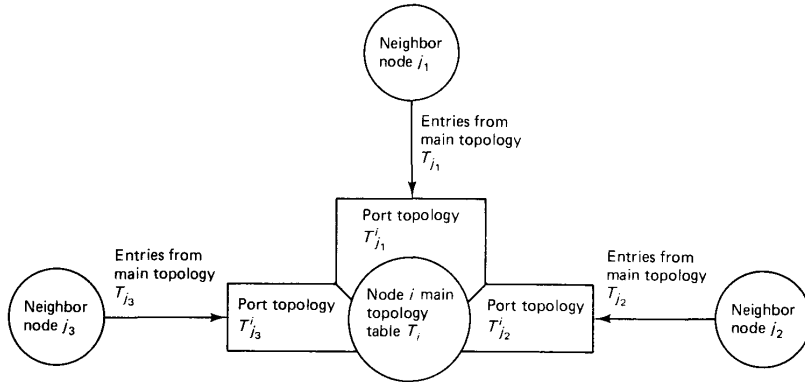


Figure 5.47 Data structures of SPTA at node i are the main topology table T_i and the port topology tables $T_{j_1}^i$, $T_{j_2}^i$, and $T_{j_3}^i$. Changes in the main topology tables of the neighbor nodes j_1 , j_2 , and j_3 are transferred with some delay to the corresponding port topology tables of node i , except if the change corresponds to a link that is incident to i . Such a change is entered directly in its main and port topology tables. Also, when an incident failed link comes up, node i transmits its entire main topology table over that link. Finally, when an entry of a port topology changes, the entire main topology table is recalculated using the main topology update algorithm.

4. When a node receives a link status message from a neighbor, it enters the message into the port topology table associated with that neighbor.
5. When an entry of a main topology T_i changes due to a status change of an incident link, or an entry of a port topology T_j^i changes due to a communication from neighbor node j , node i updates its main topology table by using the following algorithm. (It is assumed that the algorithm is restarted if an incident link changes status or a new message is received from a neighbor node while the algorithm executes.) The idea in the algorithm is for node i to believe, for each link ℓ , the status communicated by a neighbor that lies on a shortest path from node i to link ℓ . A shortest path is calculated on the basis of a length of unity for each up link and a length of infinity for each down link.

Main Topology Update Algorithm at Node i . The algorithm proceeds in iterations that are similar to those of Dijkstra's shortest path algorithm when all links have a unit length (Section 5.2.3). At the start of the k^{th} iteration, there is a set of nodes P_k . Each node in P_k carries a label which is just the ID number of some neighbor node of i . The nodes in P_k are those that can be reached from i via a path of k links or fewer that are up according to node i 's main topology table at the start of the k^{th} iteration. The label of a node $m \in P_k$ is the ID number of the neighbor of i that lies on a path from i to m that has a minimum number of (up) links. In particular, P_1 consists of the nodes connected with i via an up link, and each node in P_1 is labeled with its own ID number. During the k^{th} iteration, P_k is either augmented by some new nodes to form P_{k+1} or

else the algorithm terminates. Simultaneously, the status of the links with at least one end node in P_k but no end node in P_{k-1} , that is, the set of links

$$L_k = \{(m, n) \mid m \notin P_{k-1}, n \notin P_{k-1}, m \text{ or } n \text{ (or both) belongs to } P_k\}$$

is entered in the main topology table T_i . (For notational completeness, we define $P_0 = \{i\}$ in the equation for L_1 .) The k^{th} iteration is as follows:

Step 1: For each link $(m, n) \in L_k$, do the following: Let (without loss of generality) m be an end node belonging to P_k , and let “ j ” be the label of m . Copy the status of (m, n) from the port topology T_j^i into the main topology table T_i . If this status is up and $n \notin P_k$, give to node n the label “ j ”.

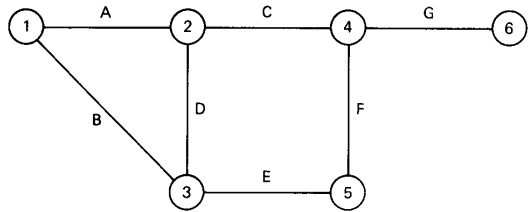
Step 2: Let M_k be the set of nodes that were labeled in step 1. If M_k is empty, terminate. Otherwise, set $P_{k+1} = P_k \cup M_k$, and go to the $(k+1)^{\text{st}}$ iteration.

The algorithm is illustrated in Fig. 5.48. Since each link is processed only once in step 1 of the algorithm, it is clear that the computational requirements are proportional to the number of links. It is straightforward to verify the following properties of the main topology update algorithm:

1. The set P_k , for $k \geq 1$, is the set of nodes m with the property that in the topology T_i finally obtained, there is a path of k or fewer up links connecting m with i .
2. The set L_k , $k \geq 1$, is the set of links ℓ with the property that in the topology T_i finally obtained, all paths of up links connecting one of the end nodes of ℓ with node i has no fewer than k links, and there is exactly one such path with k links.
3. The final entry of T_i for a link $\ell \in L_k$ is the entry from the port topology of a neighbor that is on a path of k up links from i to one of the end nodes of ℓ . If there are two such neighbors with conflicting entries, the algorithm arbitrarily chooses the entry of one of the two.
4. When the main topology update algorithm executes at node i in response to failure of an incident link (i, j) , the information in the port topology T_j^i is in effect disregarded. This is due to the fact that node j is not included in the set P_1 , and therefore no node is subsequently given the label “ j ” during execution of the algorithm.

The operation of the SPTA for the case of a single-link status change is shown in Fig. 5.49. The algorithm works roughly like flooding in this case, but there is no need to number the update messages to bound the number of message transmissions. Figure 5.50 illustrates the operation of the SPTA for the case of multiple-link status changes, and illustrates how the SPTA handles the problem of distinguishing old from new information (cf. Fig. 5.43). A key idea here is that when a link (i, j) fails, all information that came over that link is effectively disregarded, as discussed above.

We now provide a proof of correctness of the SPTA. At any given time the correct topology table (as seen by an omniscient observer) is denoted by T^* . We say that at that time link ℓ is connected with node i if there is a path connecting i with one of the end nodes of ℓ , and consisting of links that are up according to T^* . We say that at that



(a)

$$P_1 = \{2, 3\}, \quad P_2 = \{2, 3, 5\}, \quad P_3 = \{2, 3, 5, 4\}, \quad P_4 = \{2, 3, 5, 4, 6\}$$

$$L_1 = \{D, C, E\}, \quad L_2 = \{F\}, \quad L_3 = \{G\}$$

(b)

Link	Port Topology T_2^1	Port Topology T_3^1	Distance	Believes Neighbor	Final Main Topology of Node 1
A	—	—	0	—	U
B	—	—	0	—	U
C	D	U	1	2	D
D	U	U	1	2,3	U
E	U	U	1	3	U
F	U	U	2	3	U
G	D	U	3	3	U

(c)

Figure 5.48 Illustration of the main topology update algorithm for node 1 in the network (a). The node sets P_k and the link sets L_k are given in (b). The contents of the port topologies T_2^1 and T_3^1 at node 1 are given in the table in (c) (where D means down, U means up). There is conflicting information on the status of links C and G . Node 1 believes node 2 on the status of link C since node 2 is closer to this link. Node 1 believes node 3 on the status of link G since node 3 is closer to this link (given that link C was declared down by node 1).

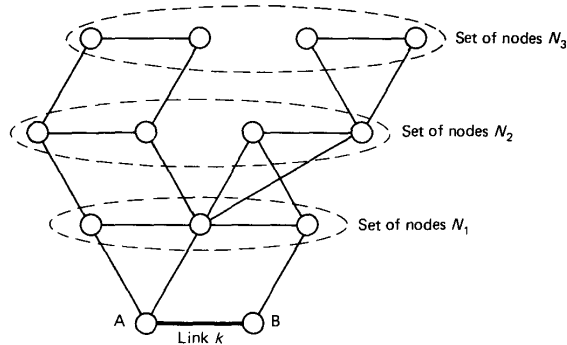
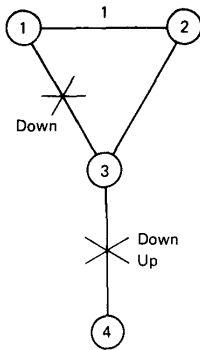


Figure 5.49 Operation of the SPTA for a single-link status change. The status change information propagates to nodes in the set N_1 (one link away from A or B or both), then to nodes in N_2 (two links away from A or B or both), then to N_3 , and so on. Specifically, when link k changes status, the change is entered in the topology tables of A and B and is broadcast to the nodes in N_1 . It is entered in the port topology tables T_A^i and/or T_B^i of nodes i in N_1 . Since the distance to link k through nodes $j \neq A, B$ is greater than zero, the new status of link k will eventually be entered in the main topology tables of each node in N_1 . It will then be broadcast to nodes in N_2 and entered in the corresponding port topology tables, and so on. The number of required message transmissions does not exceed $2L$, where L is the number of undirected network links.

time, node j is an *active neighbor* of node i if link (i, j) is up according to T^* . We say that at that time, *node i knows the correct topology* if the main topology table T_i agrees with T^* on the status of all links connected with i . We assume that at some initial time t_s , each node knows the correct topology (e.g., at network reset time when all links are down), and all active port topologies T_j^i agree on all entries with the corresponding main topologies T_j . After that time, several links change status. However, there is a time t_0 after which no link changes status and the end nodes of each link are aware of the correct link status. We will show the following:

Proposition. The SPTA works correctly in the sense that under the preceding assumptions, there is a time $t_f \geq t_0$ such that each node knows the correct topology for all $t \geq t_f$.

Figure 5.50 Operation of the SPTA for the topology change scenario of Fig. 5.43. The example demonstrates how the SPTA copes with a situation where pure flooding fails. (a) Example network with four links and four nodes. Initially, all links are up. Three changes occur: (1) link (3,4) goes down, (2) link (3,4) goes up, and (3) link (1,3) goes down. (b) Assumed sequence of events, message transmissions, and receptions of the SPTA. All topology table updating is assumed to take place instantaneously following an event. (c) Main and port topology table contents at each node following each event. U and D mean up and down, respectively. For example, a table entry UUUD means that links (1,2), (2,3), and (1,3) are up, and link (3,4) is down. Tables that change after an event are shown in bold. Note that the last event changes not only the entry for link (1,3) in T_1 , but also the entry for link (3,4). Also, after the last event, the port topology table T_1^3 is incorrect, but this is inconsequential since the connecting link (1,3) is down.



(a)

Event Number	1	2	3
Event	Link (3,4) goes down. Message is transmitted from 3 to 1 and 2. Message is received at 2.	Message is transmitted from 2 to 1. Message is received at 1.	Link (3,4) goes up. Message is transmitted from 3 to 1 and 2. Message is received at 2.
Event Number	4	5	6
Event	Message is transmitted from 2 to 1. Message is received at 1.	Message from 3 to 1 sent during Event 1 is received.	Link (1,3) goes down. Messages from 1 and 3 are sent to 2 and 4. Messages received at 2 and 4.

(b)

Event Number	1	2	3	4	5	6
Tables at Node 1	T_1	UUUU	UUUU	UUUU	UUUU	UUUD
	T_1^2	UUUU	UUUD	UUUD	UUUU	UUUD
	T_1^3	UUUU	UUUU	UUUU	UUUU	UUDD
Tables at Node 2	T_2	UUUD	UUUD	UUUU	UUUU	UUUD
	T_2^1	UUUU	UUUU	UUUU	UUUU	UUUD
	T_2^3	UUUD	UUUD	UUUU	UUUU	UUUD
Tables at Node 3	T_3	UUUD	UUUD	UUUU	UUUU	UUUD
	T_3^1	UUUD	UUUD	UUUU	UUUU	UUUD
	T_3^2	UUUD	UUUD	UUUU	UUUU	UUUD
	T_3^4	UUUD	UUUD	UUUU	UUUU	UUUD
Tables at Node 4	T_4	UUUD	UUUD	UUUU	UUUU	UUUD
	T_4^3	UUUD	UUUD	UUUU	UUUU	UUUD

(c)

Proof: In what follows we say that a link ℓ is at distance n away from i if in the graph defined by T^* , the shortest path from i to the closest end node of ℓ is n links long. We will show by induction that for each integer $n \geq 0$, there is a time $t_n \geq t_0$ after which, for each node i , T_i agrees with T^* for all links that are at a distance n or less from i . The induction hypothesis is clearly true for $n = 0$, since each node i knows the correct status of its incident links and records them in its main topology table T_i . We first establish the following lemma.

Lemma. Assume that the induction hypothesis is true for time t_n . Then, there is a time $t'_{n+1} \geq t_n$ after which the port topology table T_j^i for each active neighbor j of node i agrees with T^* for each link at a distance n or less from j .

Proof: Consider waiting a sufficient amount of time after t_n for all messages which were sent from j to i before t_n to arrive. By rules 1, 2, and 4 of the algorithm, T_j^i agrees with T_j for all links which are not incident to i . Therefore, by the induction hypothesis, T_j^i agrees with T^* for each link at a distance of n or less from j . This proves the lemma.

To complete the proof of the proposition, one must show that there is a time $t_{n+1} \geq t'_{n+1}$ such that for all $t \geq t_{n+1}$ and nodes i , T_i agrees with T^* for each link ℓ which is at a distance $n + 1$ from i . Consider the first time that link ℓ is processed by the topology update algorithm after all port topologies T_j^i and T^* agree on all links at distance n or less from j as per the preceding lemma. Then link ℓ will belong to the set L_{n+1} . Also, the closest end node(s) of ℓ to node i will belong to the set P_{n+1} and will have a label which is one of the active neighbors of i (say j) that is at distance n from ℓ . By the lemma, the port topology T_j^i agrees with T^* on all links at distance n from j . Therefore, the entry of T_j^i for link ℓ , which will be copied into T_i when link ℓ is processed, will agree with the corresponding entry of T^* . Because the conclusion of the lemma holds for all t after t'_{n+1} , the entry for link ℓ in T_i will also be correct in all subsequent times at which link ℓ will be processed by the main topology update algorithm. **Q.E.D.**

Note that the lemma above establishes that the active port topology tables also eventually agree with the corresponding main topology tables. This shows that the initial conditions for which validity of the algorithm was shown reestablish themselves following a sufficiently long period for which no topological changes occur.

The main advantage of the SPTA over the ARPANET flooding algorithm is that it does not require the use of sequence numbers with the attendant reset difficulties, and the use of an age field, the proper size of which is network dependent and changes as the network expands. Furthermore, the SPTA is entirely event driven and does not require the substantial overhead associated with the regular periodic updates of the ARPANET algorithm.

Consider now the behavior of SPTA when we cannot rely on the assumption that errors in transmission or in nodes' memories are always detected and corrected. Like all event-driven algorithms, SPTA has no ability to correct such errors. Without periodic

retransmission, an undetected error can persist for an arbitrary length of time. Thus, in situations where extraordinary reliability is needed, a node should periodically retransmit its main table to each of its neighbors. This would destroy the event-driven property of SPTA, but would maintain its other desirable characteristics and allow recovery from undetected database errors.

As mentioned earlier, the SPTA can be generalized to broadcast information other than link status throughout the network. For example, in the context of an adaptive routing algorithm, one might be interested in broadcasting average delay or traffic arrival rate information for each link. What is needed for the SPTA is to allow for additional, perhaps nonbinary, information entry in the main and port topology tables in addition to the up-down status of links. When this additional information depends on the link direction, it is necessary to make a minor modification to the SPTA so that there are separate entries in the topology tables for each link direction. This matter is considered in Problem 5.14.

5.4 FLOW MODELS, OPTIMAL ROUTING, AND TOPOLOGICAL DESIGN

To evaluate the performance of a routing algorithm, we need to quantify the notion of traffic congestion. In this section we formulate performance models based on the traffic arrival rates at the network links. These models, called *flow models* because of their relation to network flow optimization models, are used to formulate problems of optimal routing that will be the subject of Sections 5.5 to 5.7. Flow models are also used in our discussion of the design of various parts of the network topology in Sections 5.4.1 to 5.4.3.

Traffic congestion in a data network can be quantified in terms of the statistics of the arrival processes of the network queues. These statistics determine the distributions of queue length and packet waiting time at each link. It is evident that desirable routing is associated with a small mean and variance of packet delay at each queue. Unfortunately, it is generally difficult to express this objective in a single figure of merit suitable for optimization. A principal reason is that, as seen in Chapter 3, there is usually no accurate analytical expression for the means or variances of the queue lengths in a data network.

A convenient but somewhat imperfect alternative is to measure congestion at a link in terms of the *average* traffic carried by the link. More precisely, we assume that the statistics of the arrival process at each link (i, j) change due only to routing updates, and that we measure congestion on (i, j) via the traffic arrival rate F_{ij} . We call F_{ij} the *flow* of link (i, j) , and we express it in data units/sec where the data units can be bits, packets, messages, and so on. Sometimes, it is meaningful to express flow in units that are assumed directly proportional to data units/sec, such as virtual circuits traversing the link.

Implicit in flow models is the assumption that the statistics of the traffic entering the network do not change over time. This is a reasonable hypothesis when these statistics change very slowly relative to the average time required to empty the queues in the network. A typical network where such conditions hold is one accommodating a large number of users for each origin-destination pair, with the traffic rate of each of these users being small relative to the total rate (see Section 3.4.2).

An expression of the form

$$\sum_{(i,j)} D_{ij}(F_{ij}) \quad (5.29)$$

where each function D_{ij} is monotonically increasing, is often appropriate as a cost function for optimization. A frequently used formula is

$$D_{ij}(F_{ij}) = \frac{F_{ij}}{C_{ij} - F_{ij}} + d_{ij}F_{ij} \quad (5.30)$$

where C_{ij} is the transmission capacity of link (i, j) measured in the same units as F_{ij} , and d_{ij} is the processing and propagation delay. With this formula, the cost function (5.29) becomes the average number of packets in the system based on the hypothesis that each queue behaves as an $M/M/1$ queue of packets—a consequence of the Kleinrock independence approximation and Jackson's Theorem discussed in Sections 3.6 and 3.8. Although this hypothesis is typically violated in real networks, the cost function of Eqs. (5.29) and (5.30) represents a useful measure of performance in practice, principally because it expresses qualitatively that congestion sets in when a flow F_{ij} approaches the corresponding link capacity C_{ij} . Another cost function with similar qualitative properties is given by

$$\max_{(i,j)} \left\{ \frac{F_{ij}}{C_{ij}} \right\} \quad (5.31)$$

(i.e., maximum link utilization). A computational study [Vas79] has shown that it typically makes little difference whether the cost function of Eqs. (5.29) and (5.30) or that of Eq. (5.31) is used for routing optimization. This indicates that one should employ the cost function that is easiest to optimize. In what follows we concentrate on cost functions of the form $\sum_{(i,j)} D_{ij}(F_{ij})$. The analysis and computational methods of Sections 5.5 to 5.7 extend to more general cost functions (see Problem 5.27).

We now formulate a problem of optimal routing. For each pair $w = (i, j)$ of distinct nodes i and j [also referred to as an origin–destination (or OD) pair], the input traffic arrival process is assumed stationary with rate r_w .* Thus r_w (measured in data units/sec) is the arrival rate of traffic entering the network at node i and destined for node j . The routing objective is to divide each r_w among the many paths from origin to destination in a way that the resulting total link flow pattern minimizes the cost function (5.29). More precisely, denote

W = Set of all OD pairs

P_w = Set of all directed paths connecting the origin and destination nodes of OD pair w (in a variation of the problem, P_w is a given subset of the set of all directed paths connecting origin and destination of w ; the optimality conditions and algorithms to be given later apply nearly verbatim in this case)

* Sometimes it is useful to adopt a broader view of an OD pair and consider it simply as a class of users that shares the same set of paths. This allows modeling of multiple priority classes of users. In this context we would allow several OD pairs to have the same origin and destination nodes. The subsequent analysis and algorithms extend to this broader context almost verbatim. To simplify the following exposition, however, we assume that there is at most one OD pair associated with each pair of nodes.

x_p = Flow (data units/sec) of path p

Then the collection of all path flows $\{x_p \mid w \in W, p \in P_w\}$ must satisfy the constraints

$$\begin{aligned} \sum_{p \in P_w} x_p &= r_w, & \text{for all } w \in W, \\ x_p &\geq 0, & \text{for all } p \in P_w, w \in W \end{aligned}$$

as shown in Fig. 5.51. The total flow F_{ij} of link (i, j) is the sum of all path flows traversing the link

$$F_{ij} = \sum_{\substack{\text{all paths } p \\ \text{containing } (i,j)}} x_p$$

Consider a cost function of the form

$$\sum_{(i,j)} D_{ij}(F_{ij}) \quad (5.32)$$

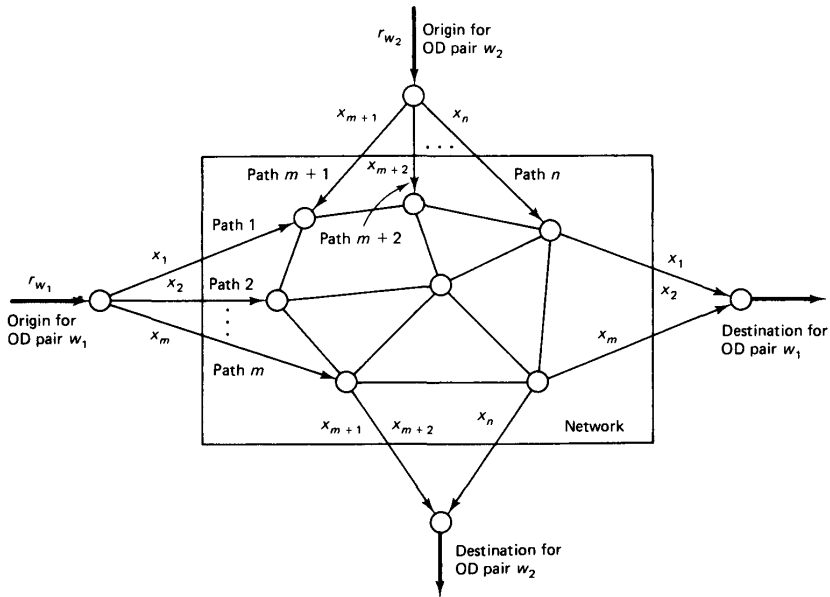


Figure 5.51 Schematic representation of a network with two OD pairs w_1 and w_2 . The paths of the OD pairs are $P_{w_1} = \{1, 2, \dots, m\}$ and $P_{w_2} = \{m+1, m+2, \dots, n\}$. The traffic inputs r_{w_1} and r_{w_2} are to be divided into the path flows $x_1, \dots, x_m$ and $x_{m+1}, \dots, x_n$, respectively.

and the problem of finding the set of path flows $\{x_p\}$ that minimize this cost function subject to the constraints above.

By expressing the total flows F_{ij} in terms of the path flows in the cost function (5.32), the problem can be written as

$$\begin{aligned}
 & \text{minimize } \sum_{(i,j)} D_{ij} \left[\sum_{\substack{\text{all paths } p \\ \text{containing } (i,j)}} x_p \right] \\
 & \text{subject to } \sum_{p \in P_w} x_p = r_w, \quad \text{for all } w \in W \\
 & \quad x_p \geq 0, \quad \text{for all } p \in P_w, w \in W
 \end{aligned} \tag{5.33}$$

Thus, the problem is formulated in terms of the unknown path flows $\{x_p \mid p \in P_w, w \in W\}$. This is the main optimal routing problem that will be considered. A characterization of its optimal solution is given in Section 5.5, and interestingly, it is expressed in terms of shortest paths. Algorithms for its solution are given in Sections 5.6 and 5.7.

The optimal routing problem just formulated is amenable to analytical investigation and distributed computational solution. However, it has some limitations that are worth explaining. The main limitation has to do with the choice of the cost function $\sum_{(i,j)} D_{ij}(F_{ij})$ as a figure of merit. The underlying hypothesis here is that one achieves reasonably good routing by optimizing the average levels of link traffic without paying attention to other aspects of the traffic statistics. Thus, the cost function $\sum_{(i,j)} D_{ij}(F_{ij})$ is insensitive to undesirable behavior associated with high variance and with correlations of packet interarrival times and transmission times. To illustrate this fact, consider the example given in Section 3.6, where we had a node A sending Poisson traffic to a node B along two equal-capacity links. We compared splitting of the traffic between the two links by using randomization and by using metering. It was concluded that metering is preferable to randomization in terms of average packet delay. Yet randomization and metering are rated equally by the cost function $\sum_{(i,j)} D_{ij}(F_{ij})$, since they result in the same flow on each link. The problem here is that delay on each link depends on second and higher moments of the arrival process, while the cost function reflects a dependence on just the first moment.

Since metering is based on keeping track of current queue lengths, the preceding example shows that routing can be improved by using queue length information. Here is another example that illustrates the same point.

Example 5.6

Consider the network shown in Fig. 5.52. Here there are three origin–destination pairs, each with an arrival rate of 1 unit. The OD pairs (2,4) and (3,4) route their traffic exclusively along links (2,4) and (3,4) respectively. Since all links have a capacity of 2 units, the two paths (1,2,4) and (1,3,4) are equally attractive for the traffic of OD pair (1,4). An optimal routing algorithm based on time average link rates would alternate sending packets of OD pair (1,4) along the two paths, thereby resulting in a total rate of 1.5 units on each of the bottleneck links (2,4) and (3,4). This would work reasonably well if the traffic of the OD

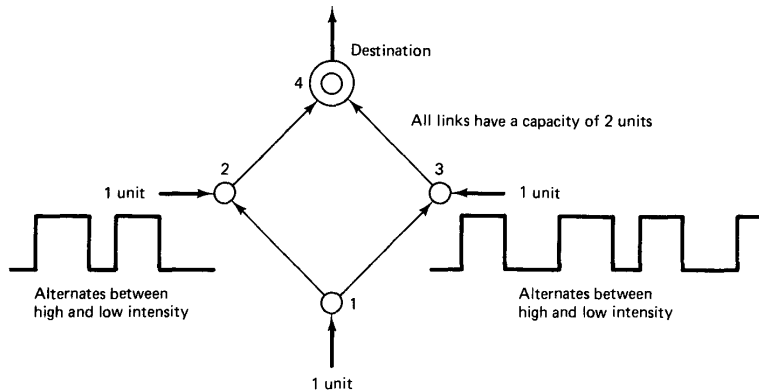


Figure 5.52 Network of Example 5.6 where routing based on queue length information is beneficial. The traffic originating at nodes 2 and 3 alternates in intensity between 2 units and 0 units over large time intervals. A good form of routing gets information on the queue sizes at links (2,4) and (3,4) and routes the traffic originating at node 1 on the path of least queue size.

pairs (2,4) and (3,4) had a Poisson character. Consider, however, the situation where instead, this traffic alternates in intensity between 2 units and 0 units over large time intervals. Then the queues in link (2,4) and (3,4) would build up (not necessarily simultaneously) over some large time intervals and empty out during others. Suppose now that a more dynamic form of routing is adopted, whereby node 1 is kept informed about queue sizes at links (2,4) and (3,4), and routes packets on the path of least queue size. Queueing delay would then be greatly reduced.

Unfortunately, it is impractical to keep nodes informed of all queue lengths in a large network. Even if the overhead for doing so were not prohibitive, the delays involved in transferring the queue length information to the nodes could make this information largely obsolete. It is not known at present how to implement effective and practical routing based on queue length information, so we will not consider the subject further. For an interesting but untested alternative, see Problem 5.38.

The remainder of this section considers the use of network algorithms, flow models, and the optimal routing problem in the context of topological design of a network. The reader may skip this material without loss of continuity.

5.4.1 An Overview of Topological Design Problems

In this subsection we discuss algorithms for designing the topology of a data network. Basically, we are given a set of traffic demands, and we want to put together a network that will service these demands at minimum cost while meeting some performance requirements. A common, broadly stated formulation of the problem is as follows:

We assume that we are given:

1. The geographical location of a collection of devices that need to communicate with each other. For simplicity, these devices are called *terminals*.
2. A traffic matrix giving the input traffic flow from each terminal to every other terminal.

We want to design (cf. Fig. 5.53):

1. The *topology of a communication subnet* to service the traffic demands of the terminals. This includes the location of the nodes, the choice of links, and the capacity of each link.
2. The *local access network* (i.e., the collection of communication lines that will connect the terminals to entry nodes of the subnet).

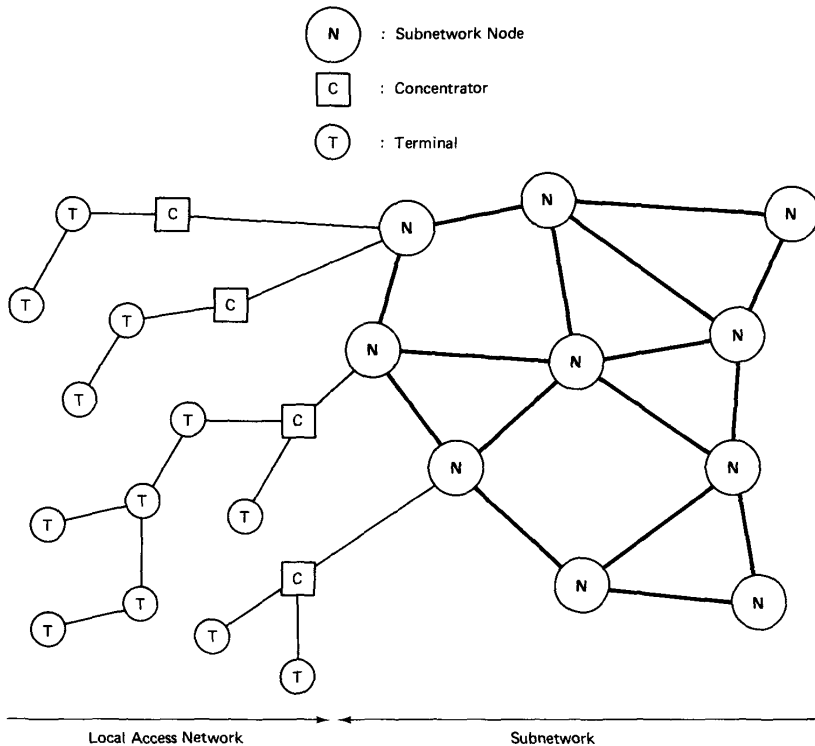


Figure 5.53 The main aspects of the topology design problem are the design of the local access network and the communication subnet.

The design objectives are:

1. Keep the average delay per packet or message below a given level (for the given nominal traffic demands and assuming some type of routing algorithm)
2. Satisfy some reliability constraint to guarantee the integrity of network service in the face of a number of link and node failures
3. Minimize a combination of capital investment and operating costs while meeting objectives 1 and 2

It is evident that this is, in general, a very broad problem that may be difficult to formulate precisely, let alone solve. However, in many cases, the situation simplifies somewhat. For example, part of the problem may already be solved because portions of the local access network and/or the subnet might already be in place. There may be also a natural decomposition of the problem—for example, the subnet topology design problem may be naturally decoupled from the local access network design problem. In fact, this will be assumed in the presentation. Still, one seldom ends up with a clean problem that can be solved exactly in reasonable time. One typically has to be satisfied with an approximate solution obtained through heuristic methods that combine theory, trial and error, and common sense.

In Section 5.4.2 we consider the subnet design problem, assuming that the local access network has been designed and therefore, the matrix of input traffic flow for every pair of subnet nodes is known. Subsequently, in Section 5.4.3, the local access network design problem is considered.

5.4.2 Subnet Design Problem

Given the location of the subnet nodes and an input traffic flow for each pair of these nodes, we want to select the capacity and flow of each link so as to meet some delay and reliability constraints while minimizing costs. Except in very simple cases, this turns out to be a difficult combinatorial problem. To illustrate the difficulties, we consider a simplified version of the problem, where we want to choose the link capacities so as to minimize a linear cost. We impose a delay constraint but neglect any reliability constraints. Note that by assigning zero capacity to a link we effectively eliminate the link. Therefore, the capacity assignment problem includes as a special case the problem of choosing the pairs of subnet nodes that will be directly connected with a communication line.

Capacity assignment problem. The problem is to choose the capacity C_{ij} of each link (i, j) so as to minimize the linear cost

$$\sum_{(i,j)} p_{ij} C_{ij} \quad (5.34)$$

where p_{ij} is a known positive price per unit capacity, subject to the constraint that the average delay per packet should not exceed a given constant T .

The flow on each link (i, j) is denoted F_{ij} and is expressed in the same units as capacity. We adopt the $M/M/1$ model based on the Kleinrock independence approximation, so we can express the average delay constraint as

$$\frac{1}{\gamma} \sum_{(i,j)} \frac{F_{ij}}{C_{ij} - F_{ij}} \leq T \quad (5.35)$$

where γ is the total arrival rate into the network. We assume that there is a given input flow for each origin–destination pair and γ is the sum of these. The link flows F_{ij} depend on the known input flows and the scheme used for routing. We will first assume that routing and therefore also the flows F_{ij} are known, and later see what happens when this assumption is relaxed.

When the flows F_{ij} are known, the problem is to minimize the linear cost $\sum_{(i,j)} p_{ij} C_{ij}$ over the capacities C_{ij} subject to the constraint (5.35), and it is intuitively clear that the constraint will be satisfied as an equality at the optimum. We introduce a Lagrange multiplier β and form the Lagrangian function

$$L = \sum_{(i,j)} \left(p_{ij} C_{ij} + \frac{\beta}{\gamma} \frac{F_{ij}}{C_{ij} - F_{ij}} \right)$$

In accordance with the Lagrange multiplier technique, we set the partial derivatives $\partial L / \partial C_{ij}$ to zero:

$$\frac{\partial L}{\partial C_{ij}} = p_{ij} - \frac{\beta F_{ij}}{\gamma (C_{ij} - F_{ij})^2} = 0$$

Solving for C_{ij} gives

$$C_{ij} = F_{ij} + \sqrt{\frac{\beta F_{ij}}{\gamma p_{ij}}} \quad (5.36)$$

and substituting in the constraint equation, we obtain

$$T = \frac{1}{\gamma} \sum_{(i,j)} \frac{F_{ij}}{C_{ij} - F_{ij}} = \sum_{(i,j)} \sqrt{\frac{p_{ij} F_{ij}}{\beta \gamma}}$$

From the last equation,

$$\sqrt{\beta} = \frac{1}{T} \sum_{(i,j)} \sqrt{\frac{p_{ij} F_{ij}}{\gamma}} \quad (5.37)$$

which when substituted in Eq. (5.36) yields the optimal solution:*

$$C_{ij} = F_{ij} + \frac{1}{T} \sqrt{\frac{F_{ij}}{\gamma p_{ij}}} \sum_{(m,n)} \sqrt{\frac{p_{mn} F_{mn}}{\gamma}}$$

The solution, after rearranging, can also be written as

$$C_{ij} = F_{ij} \left(1 + \frac{1}{\gamma T} \frac{\sum_{(m,n)} \sqrt{p_{mn} F_{mn}}}{\sqrt{p_{ij} F_{ij}}} \right) \quad (5.38)$$

Finally, by substitution in the cost function $\sum_{(i,j)} p_{ij} C_{ij}$, the optimal cost is expressed as

$$\text{optimal cost} = \sum_{(i,j)} p_{ij} F_{ij} + \frac{1}{\gamma T} \left(\sum_{(i,j)} \sqrt{p_{ij} F_{ij}} \right)^2 \quad (5.39)$$

Consider now the problem of optimizing the network cost with respect to both the capacities C_{ij} and the flows F_{ij} (equivalently, the routing). This problem can be solved by minimizing the cost (5.39) with respect to F_{ij} and then obtaining the optimal capacities from the closed-form expression (5.38). Unfortunately, it turns out that the cost (5.39) has many local minima and is very difficult to minimize. Furthermore, in these local minima, there is a tendency for many of the flows F_{ij} and corresponding capacities C_{ij} to be zero. The resulting networks tend to have low connectivity (few links with large capacity) and may violate reliability constraints. The nature of this phenomenon can be understood by considering the simple network of Fig. 5.54 involving two nodes and n links. Suppose that we want to choose the capacities $C_1, \dots, C_n$ so as to minimize the sum $C_1 + \dots + C_n$ while dividing the input rate λ among the n links so as to meet an average delay constraint. From our comparison of statistical and time-division multiplexing (Sections 3.1 and 3.3), we know that delay increases if a transmission line is divided into smaller lines, each serving a fraction of the total traffic. It is therefore evident that the optimal solution is the minimal connectivity topology whereby all links are eliminated, except one which has just enough capacity to meet the delay constraint.

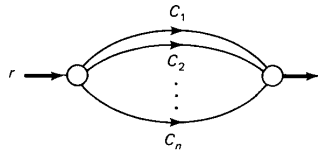


Figure 5.54 Minimizing the total capacity $C_1 + \dots + C_n$ while meeting an average delay constraint results in a minimum connectivity network (all links are eliminated except one).

*It is rigorous to argue at this point that the capacities C_{ij} of Eq. (5.38) are optimal. The advanced reader can verify this by first showing that the average delay expression of Eq. (5.35) is convex and differentiable over the set of all C_{ij} with $C_{ij} > F_{ij}$, and by arguing that the equation $\partial L / \partial C_{ij} = 0$ is a sufficient condition for optimality that is satisfied by the positive Lagrange multiplier β of Eq. (5.37) and the capacities C_{ij} of Eq. (5.38) (see [Las70], p. 84, [Roc70], p. 283, or [BeT89], p. 662).

It makes no difference which lines are eliminated, so this problem has n local minima. Each of these happens to be also a global minimum, but this is a coincidence due to the extreme simplicity of the example.

The conclusion from the preceding discussion is that the simultaneous optimization of link flows and capacities is a hard combinatorial problem. Furthermore, even if this problem is solved, the network topology obtained will tend to concentrate capacity in a few large links, and possibly violate reliability constraints. The preceding formulation is also unrealistic because, in practice, capacity costs are not linear. Furthermore, the capacity of a leased communication line usually must be selected from a finite number of choices. In addition, the capacity of a line is typically equal in both directions, whereas this is not necessarily true for the optimal capacities of Eq. (5.38). These practical constraints enhance further the combinatorial character of the problem and make an exact solution virtually impossible. As a result, the only real option for addressing the problem subject to these constraints is to use the heuristic schemes that are described next.

Heuristic methods for capacity assignment. Typically, these methods start with a network topology and then successively perturb this topology by changing one or more link capacities at a time. Thus, these methods search “locally” around an existing topology for another topology that satisfies the constraints and has lower cost. Note that the term “topology” here means the set of capacities of all potential links of the network. In particular, a link with zero capacity represents a link that in effect does not exist. Usually, the following are assumed:

1. The nodes of the network and the input traffic flow for each pair of nodes are known.
2. A routing model has been adopted that determines the flows F_{ij} of all links (i, j) given all link capacities C_{ij} . The most common possibility here is to assume that link flows minimize a cost function $\sum_{(i,j)} D_{ij}(F_{ij})$ as in Eq. (5.33). In particular, F_{ij} can be determined by minimizing the average packet delay,

$$D = \frac{1}{\gamma} \sum_{(i,j)} \left(\frac{F_{ij}}{C_{ij} - F_{ij}} + d_{ij} F_{ij} \right) \quad (5.40)$$

based on $M/M/1$ approximations [cf. Eq. (5.30)], where γ is the known total input flow into the network, and C_{ij} and d_{ij} are the known capacity and the processing and propagation delay, respectively, of link (i, j) . Several algorithms described in Sections 5.6 and 5.7 can be used for this purpose.

3. There is a delay constraint that must be met. Typically, it is required that the chosen capacities C_{ij} and link flows F_{ij} (determined by the routing algorithm as in 2 above) result in a delay D given by the $M/M/1$ formula (5.40) (or a similar formula) that is below a certain threshold.
4. There is a reliability constraint that must be met. As an example, it is common to require that the network be 2-connected (*i.e.*, after a single node failure, all other nodes remain connected). More generally, it may be required that after $k - 1$ nodes

fail, all other nodes remain connected—such a network is said to be k -connected. We postpone a discussion of how to evaluate the reliability of a network for later.

5. There is a cost criterion according to which different topologies are ranked.

The objective is to find a topology that meets the delay and reliability constraints as per 3 and 4 above, and has as small a cost as possible as per 5 above. We describe a prototype iterative heuristic method for addressing the problem. At the start of each iteration, there is available a *current best topology* and a *trial topology*. The former topology satisfies the delay and reliability constraints and is the best one in terms of cost that has been found so far, while the latter topology is the one that will be evaluated in the current iteration. We assume that these topologies are chosen initially by some ad hoc method—for example, by establishing an ample number of links to meet the delay and reliability constraints. The steps of the iteration are as follows:

Step 1: (Assign Flows). Calculate the link flows F_{ij} for the trial topology by means of some routing algorithm as per assumption 2.

Step 2: (Check Delay). Evaluate the average delay per packet D [as given, for example, by the $M/M/1$ formula (5.40)] for the trial topology. If

$$D \leq T$$

where T is a given threshold, go to step 3; else go to step 5.

Step 3: (Check Reliability). Test whether the trial topology meets the reliability constraints. (Methods for carrying out this step will be discussed shortly.) If the constraints are not met, go to step 5; else go to step 4.

Step 4: (Check Cost Improvement). If the cost of the trial topology is less than the cost of the current best topology, replace the current best topology with the trial topology.

Step 5: (Generate a New Trial Topology). Use some heuristic to change one or more capacities of the current best topology, thereby obtaining a trial topology that has not been considered before. Go to step 1.

Note that a trial topology is adopted as the current best topology in step 4 only if it satisfies the delay and reliability constraints in steps 2 and 3 and improves the cost (step 4). The algorithm terminates when no new trial topology can be generated or when substantial further improvement is deemed unlikely. Naturally, there is no guarantee that the final solution is optimal. One possibility to attempt further improvement is to repeat the algorithm with a different starting topology. Another possibility is to modify step 4 so that on occasion a trial topology is accepted as a replacement of the current best topology even if its cost is greater than the best found so far. A popular scheme of this type, known as *simulated annealing*, can be used to find a globally optimal solution under mild assumptions ([KGV83], [Haj88], and [Tsi89]). The amount of computing time required, however, may be excessive. For a method that uses this approach see [Fe89].

There are a number of heuristic rules proposed in the literature for generating new trial topologies (step 5). One possibility is simply to lower the capacity of a link that seems underutilized (low F_{ij}/C_{ij} and F_{ji}/C_{ji}) or to eliminate the link altogether. Another possibility is to increase the capacity of some overutilized link when the delay constraint is not met. A combination of these possibilities is the *branch exchange heuristic*, whereby one link is deleted and another link is added. A useful way to choose the links to be deleted and added is the so-called *saturated cut method*. The idea here is to identify a partition (or cut) of the nodes into two sets N_1 and N_2 such that the links joining N_1 and N_2 are highly utilized. It then seems plausible that adding a link between a node in N_1 and a node in N_2 will help reduce the high level of utilization across the cut. The method works as follows (see Fig. 5.55):

1. Prepare a list of all undirected links (i, j) , sorted in order of decreasing utilization as measured by $\max \{F_{ij}/C_{ij}, F_{ji}/C_{ji}\}$.
2. Find a link k such that (a) if all links above k on the list are eliminated, the network remains connected, and (b) if k is eliminated together with all links above it on the list, the network separates in two disconnected components N_1 and N_2 .
3. Remove the most underutilized link in the network, and replace it with a new link that connects a node in the component N_1 and a node in the component N_2 .

There are a number of variations of the scheme described above. For example, in selecting the link to remove, one may take into account the cost of capacity for the link. Needless to say, prior knowledge about the network and the cost structure for link capacity can be effectively incorporated into the heuristics. In addition, the design process can be enhanced by using an interactive program that can be appropriately guided by the designer. We now turn our attention to methods for evaluating reliability of a network.

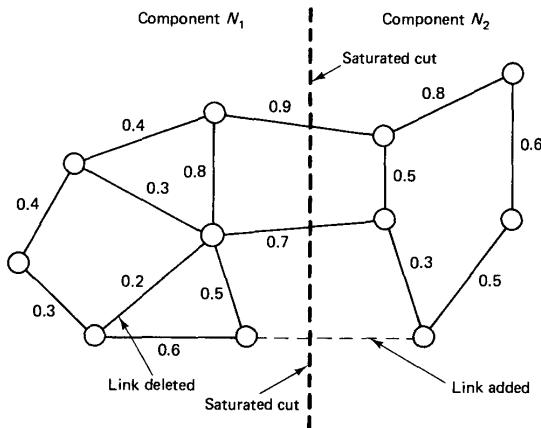


Figure 5.55 Illustration of the cut saturation method for performing a branch exchange heuristic. The number next to each arc is the link utilization. Links of high utilization are temporarily removed until the network separates into two components, N_1 and N_2 . A link is then added that connects a node in N_1 with a node in N_2 while a link of low utilization is deleted.

Network reliability issues. We have already mentioned that a common reliability requirement for a network is that it must be k -connected. We now define k -connectivity more formally.

We say that *two nodes i and j* in an undirected graph are k -connected if there is a path connecting i and j in every subgraph obtained by deleting $(k - 1)$ nodes other than i and j together with their adjacent arcs from the graph. We say that *the graph is k -connected* if every pair of nodes is k -connected. These definitions are illustrated in Fig. 5.56. Another notion of interest is that of *arc connectivity*, which is defined as the minimum number of arcs that must be deleted before the graph becomes disconnected. A lower bound k on arc connectivity can be established by checking for k -(node) connectivity in an expanded graph where each arc (i, j) is replaced by two arcs (i, n) and (n, j) , where n is a new node. This is evident by noting that failure of node n in the expanded graph can be associated with failure of the arc (i, j) in the original graph.

It is possible to calculate the number of paths connecting two nodes i and j which are node-disjoint in the sense that any two of them share no nodes other than i and j . This can be done by setting up and solving a max-flow problem, a classical combinatorial problem, which is treated in detail in many sources (e.g., [PaS82] and [Ber91]). We do not discuss here the max-flow problem, since it is somewhat tangential to the purposes of this section. The reader who is already familiar with this problem can see its connection with the problem of k -connectivity of two nodes from Fig. 5.57. Using the theory of the max-flow problem, it can be shown that i and j are k -connected if and only if either i and j are connected with an arc or there are at least k node-disjoint paths connecting i and j .

One way to check k -connectivity of a graph is to check k -connectivity of every pair of nodes. There are, however, more efficient methods. One particular method, due to Kleitman [Kle69], operates as follows:

Choose an arbitrary node n_0 and check k -connectivity between that node and every other node. Delete n_0 and its adjacent arcs from the graph, choose another node n_1 , and check $(k - 1)$ -connectivity between that node and every other node. Continue in this manner until either node n_{k-1} is checked to be 1-connected to every remaining node, or $(k - i)$ -connectivity of some node $n_i, i = 0, 1, \dots, k - 1$, to every remaining node

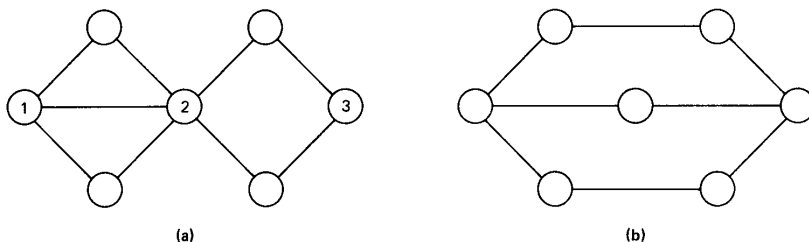


Figure 5.56 Illustration of the definition of k -connectivity. In graph (a), nodes 1 and 2 are 6-connected, nodes 2 and 3 are 2-connected, and nodes 1 and 3 are 1-connected. Graph (a) is 1-connected. Graph (b) is 2-connected.

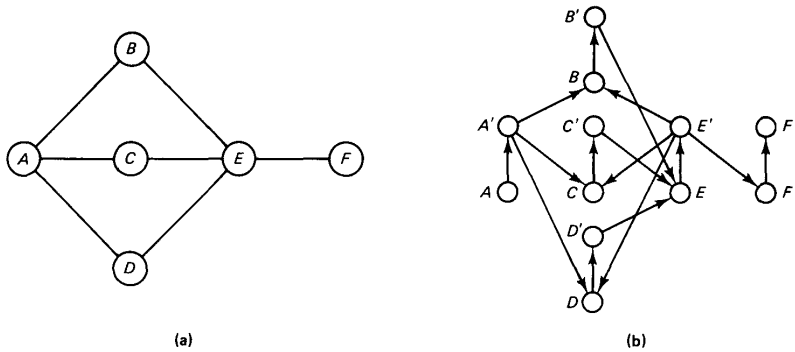


Figure 5.57 Illustration of test of k -connectivity for those familiar with the max-flow problem. To find the number of node-disjoint paths from node A to node F in graph (a), do the following: (1) Replace each node i with two nodes i and i' and a directed arc (i, i') . (2) Replace each arc (i, j) of graph (a) with two directed arcs (i', j) and (j', i) . (3) Delete all arcs incoming to A and outgoing from F' , thereby obtaining graph (b). (4) Assign infinite capacity to arcs (A, A') and (F, F') , and unit capacity to all other arcs. Then the maximum flow that can be sent from node A to node F' in graph (b) equals the number of node-disjoint paths connecting nodes A and F in graph (a).

cannot be verified. In the latter case, it is evident that the graph is not k -connected: If node n_i is not $(k - i)$ -connected to some other remaining node, $(k - i - 1)$ nodes can be found such that their deletion along with nodes $n_0, \dots, n_{i-1}$ disconnects the graph. The process is demonstrated in Fig. 5.58.

To establish the validity of Kleitman's algorithm, it must be shown that when nodes $n_0, \dots, n_{k-1}$ can be found as described above, the graph is k -connected. The argument is by contradiction. If the graph is not k -connected, there must exist nodes i and j , and

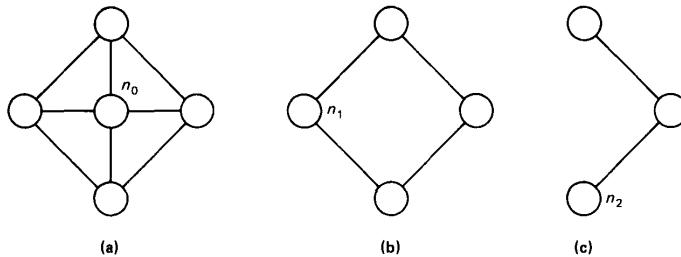


Figure 5.58 Verifying 3-connectivity of graph (a) by Kleitman's algorithm. An arbitrary node n_0 is chosen, and 3-connectivity between n_0 and every other node is verified. Then node n_0 and its adjacent arcs are deleted, thereby obtaining graph (b). An arbitrary node n_1 of (b) is chosen, and 2-connectivity between n_1 and every other node in (b) is verified. Then node n_1 and its adjacent arcs are deleted, thereby obtaining graph (c). Since graph (c) is connected, it follows that the original graph (a) is 3-connected.

a set of $(k - 1)$ nodes $\mathcal{F}_0$ ($i \notin \mathcal{F}_0, j \notin \mathcal{F}_0$), such that deletion of $\mathcal{F}_0$ leaves no path connecting i and j . Then n_0 must belong to $\mathcal{F}_0$ because if $n_0 \notin \mathcal{F}_0$, then since n_0 is k -connected to every other node, it follows that there are paths P_i and P_j connecting n_0 , i , and j even after the set $\mathcal{F}_0$ is deleted. Therefore, a path from i to j can be constructed by first using P_i to go from i to n_0 , and then using P_j to go from n_0 to j . Next, consider the sets $\mathcal{F}_1 = \mathcal{F}_0 - \{n_0\}$, $\mathcal{F}_2 = \mathcal{F}_1 - \{n_1\}, \dots$. Similarly, n_1 must belong to $\mathcal{F}_1$, n_2 must belong to $\mathcal{F}_2$, and so on. After this argument is used $(k - 1)$ times, it is seen that $\mathcal{F}_0 = \{n_0, n_1, \dots, n_{k-2}\}$. Therefore, after $\mathcal{F}_0$ is deleted from the graph, node n_{k-1} is still intact, and by definition of n_{k-1} , so are the paths from n_{k-1} to i and j . These paths can be used to connect i and j , contradicting the hypothesis that deletion of $\mathcal{F}_0$ leaves i and j disconnected.

Kleitman's algorithm requires a total of $\sum_{i=1}^k (N - i) = kN - k(k + 1)/2$ connectivity tests of node pairs, where N is the number of nodes. There is another algorithm which requires roughly N connectivity tests; see [Eve75] for details.

The preceding formulation of the reliability problem addresses the worst case where the most unfavorable combination of k node failures occurs. An alternative formulation is to assign failure probabilities to all network elements (nodes and links) and evaluate the probability that a given pair of nodes becomes disconnected. To get an appreciation of the difficulties involved in this approach, suppose that there are n failure-prone elements in the network. Then the number of all distinct combinations of surviving elements is 2^n . Let s_k denote the k^{th} combination and p_k denote the probability of its occurrence. The probability that the network remains connected is

$$P_C = \sum_{s_k \in C} p_k \quad (5.41)$$

where C is the set of combinations of surviving elements for which the network remains connected. Thus, evaluation of P_C involves some form of implicit or explicit enumeration of the set C . This is in general a very time-consuming procedure. However, shortcuts and approximations have been found making the evaluation of the survival probability P_C feasible for some realistic networks ([BaP83] and [Ben86]). An alternative approach evaluates a lower bound for P_C by ordering the possible combinations of surviving elements according to their likelihood, and then adding p_k over a subset of most likely combinations from C ([LaL86] and [LiS84]). A similar approach using the complement of C gives an upper bound to P_C .

Spanning tree topology design. For some networks where reliability is not a serious issue, a spanning tree topology may be appropriate. Using such a topology is consistent with the idea of concentrating capacity on just a few links to reduce the average delay per packet (cf. the earlier discussion on capacity assignment).

It is possible to design a spanning tree topology by assigning a weight w_{ij} to each possible link (i, j) and by using the minimum weight spanning tree (MST) algorithms of Section 5.2.2. A potential difficulty arises when there is a constraint on the amount of traffic that can be carried by any one link. This gives rise to the *constrained MST problem*, where a matrix of input traffic from every node to every other node is given

and an MST is to be designed subject to the constraint that the flow on each link will not exceed a given upper bound. Because of this constraint, the problem has a combinatorial character and is usually addressed using heuristic methods.

One possibility is to modify the Kruskal or Prim–Dijkstra algorithms, described in Section 5.2.2, so that at each iteration, when a new link is added to the current fragments, a check is made to see if the flow constraints on the links of the fragments are satisfied. If not, the link is not added and another link is considered. This type of heuristic can also be combined with some version of the branch exchange heuristic discussed earlier in connection with the general subnet design problem.

A special case of the constrained MST problem can be addressed using the *Essau–Williams algorithm* [EsW66]. In this case, there is a central node denoted 0, and N other nodes. All traffic must go through the central node, so it can be assumed that there is input traffic only from the noncentral nodes to the central node and the reverse. Problems of this type arise also in the context of the local access design problem, with the central node playing the role of a traffic concentrator.

The Essau–Williams algorithm is, in effect, a branch exchange heuristic. One starts with the spanning tree where the central node is directly connected with each of the N other nodes. (It is assumed that this is a feasible solution.) At each successive iteration, a link $(i, 0)$ connecting some node i with the central node 0 is deleted from the current spanning tree, and a link (i, j) is added. These links are chosen so that:

1. No cycle is formed.
2. The capacity constraints of all the links of the new spanning tree are satisfied.
3. The saving $w_{i0} - w_{ij}$ in link weight obtained by exchanging $(i, 0)$ with (i, j) is positive and is maximized over all nodes i and j for which 1 and 2 are satisfied.

The algorithm terminates when there are no nodes i and j for which requirements 1 to 3 are satisfied when $(i, 0)$ is exchanged with (i, j) . The operation of the algorithm is illustrated in Fig. 5.59.

5.4.3 Local Access Network Design Problem

Here we assume that a communication subnet is available, and we want to design a network that connects a collection of terminals with known demands to the subnet. This problem is often addressed in the context of a hierarchical strategy, whereby groups of terminals are connected, perhaps through local area networks, to various types of concentrators, which are in turn connected to higher levels of concentrators, and so on. It is difficult to recommend a global design strategy without knowledge of the given practical situation. However, there are a few subproblems that arise frequently. We will discuss one such problem, known as the *concentrator location problem*.

In this problem, there are n sources located at known geographical points. For example, a source may be a gateway of a local area network that connects several terminals within a customer's facility; or it could be a host computer through which several time-sharing terminals access some network. The nature of the sources is not

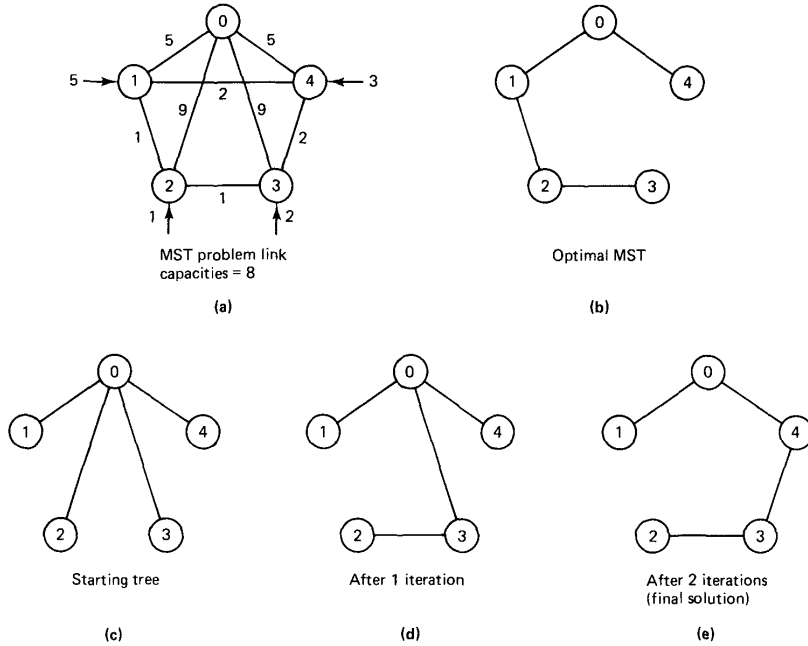


Figure 5.59 Illustration of the Essau-Williams, constrained MST heuristic algorithm. The problem data are given in (a). Node 0 is the central node. The link weights are shown next to the links. The input flows from nodes 1, 2, 3, and 4 to the central node are shown next to the arrows. The flow on each link is constrained to be no more than 8. The algorithm terminates after two iterations, with the tree (c) having a total weight of 13. Termination occurs because when link (1,0) or (4,0) is removed and a link that is not adjacent to node 0 is added, some link capacity constraint (link flow ≤ 8) will be violated. The optimal tree, shown in (b), has a total weight of 12.

material to the discussion. The problem is to connect the n sources to m concentrators that are themselves connected to a communication subnet (see Fig. 5.60). We denote by a_{ij} the cost of connecting source i to concentrator j . Consider the variables x_{ij} where

$$x_{ij} = \begin{cases} 1, & \text{if source } i \text{ is connected to concentrator } j \\ 0, & \text{otherwise} \end{cases}$$

Then the total cost of a source assignment specified by a set of variables $\{x_{ij}\}$ is

$$\text{cost} = \sum_{i=1}^n \sum_{j=1}^m a_{ij} x_{ij}. \quad (5.42)$$

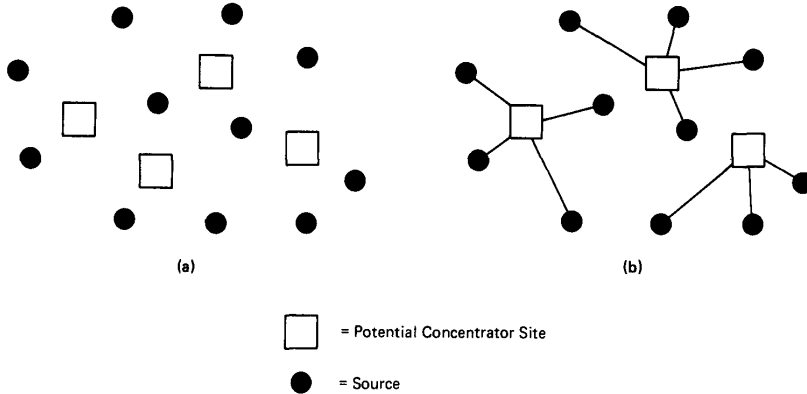


Figure 5.60 (a) Collection of sources and potential concentrator locations. (b) Possible assignment of sources to concentrators.

We assume that each source is connected to only one concentrator, so that there is the constraint

$$\sum_{j=1}^m x_{ij} = 1, \quad \text{for all sources } i \quad (5.43)$$

Also, there is a maximum number of sources K_j that can be handled by concentrator j . This is expressed by the constraint

$$\sum_{i=1}^n x_{ij} \leq K_j, \quad \text{for all concentrators } j \quad (5.44)$$

The problem is to select the variables x_{ij} so as to minimize the cost (5.42) subject to the constraints (5.43) and (5.44).

Even though the variables x_{ij} are constrained to be 0 or 1, the problem above is not really a difficult combinatorial problem. If the integer constraint

$$x_{ij} = 0 \text{ or } 1$$

is replaced by the noninteger constraint

$$0 \leq x_{ij} \leq 1 \quad (5.45)$$

the problem becomes a linear transportation problem that can be solved by very efficient algorithms such as the simplex method. It can be shown that the solution obtained from the simplex method will be integer (*i.e.*, x_{ij} will be either 0 or 1; see [Dan63] and [Ber91]). Therefore, solving the problem without taking into account the integer constraints automatically obtains an integer optimal solution. There are methods other than simplex that can be used for solving the problem ([Ber85], [Ber91], [BeT89], [FoF62], and [Roc84]).

Next, consider a more complicated problem whereby the location of the concentrators is not fixed but instead is subject to optimization. There are m potential concentrator sites, and there is a cost b_j for locating a concentrator at site j . The cost then becomes

$$\text{cost} = \sum_{i=1}^n \sum_{j=1}^m a_{ij} x_{ij} + \sum_{j=1}^m b_j y_j \quad (5.46)$$

where

$$y_j = \begin{cases} 1, & \text{if a concentrator is located at site } j \\ 0, & \text{otherwise} \end{cases}$$

The constraint (5.44) now becomes

$$\sum_{i=1}^n x_{ij} \leq K_j y_j, \quad \text{for all concentrators } j \quad (5.47)$$

Thus, the problem is to minimize the cost (5.46) subject to the constraints (5.43) and (5.47) and the requirement that x_{ij} and y_j are either 0 or 1.

This problem has been treated extensively in the operations research literature, where it is known as the warehouse location problem. It is considered to be a difficult combinatorial problem that can usually be solved only approximately. A number of exact and heuristic methods have been proposed for its solution. We refer the reader to the literature for further details ([AkK77], [AlM76], [CFN77], [KeH63], [Khu72], and [RaW83]).

5.5 CHARACTERIZATION OF OPTIMAL ROUTING

We now return to the optimal routing problem formulated at the beginning of Section 5.4. The main objective in this section is to show that optimal routing directs traffic exclusively along paths which are shortest with respect to some link lengths that depend on the flows carried by the links. This is an interesting characterization which motivates algorithms discussed in Sections 5.6 and 5.7.

Recall the form of the cost function

$$\sum_{(i,j)} D_{ij}(F_{ij}) \quad (5.48)$$

Here F_{ij} is the total flow (in data units/sec) carried by link (i, j) and given by

$$F_{ij} = \sum_{\substack{\text{all paths } p \\ \text{containing } (i,j)}} x_p \quad (5.49)$$

where x_p is the flow (in data units/sec) of path p . For every OD pair w , there are the constraints

$$\sum_{p \in P_w} x_p = r_w \quad (5.50)$$

$$x_p \geq 0, \quad \text{for all } p \in P_w \quad (5.51)$$

where r_w is the given traffic input of the OD pair w (in data units/sec) and P_w is the set of directed paths of w . In terms of the unknown path flow vector $x = \{x_p \mid p \in P_w, w \in W\}$, the problem is written as

$$\begin{aligned} & \text{minimize } \sum_{(i,j)} D_{ij} \left[\sum_{\substack{\text{all paths } p \\ \text{containing } (i,j)}} x_p \right] \\ & \text{subject to } \sum_{p \in P_w} x_p = r_w, \quad \text{for all } w \in W \\ & x_p \geq 0, \quad \text{for all } p \in P_w, w \in W \end{aligned} \quad (5.52)$$

In what follows we will characterize an optimal routing in terms of the first derivatives D'_{ij} of the functions D_{ij} . We assume that each D_{ij} is a differentiable function of F_{ij} and is defined in an interval $[0, C_{ij})$, where C_{ij} is either a positive number (typically representing the capacity of the link) or else infinity. Let x be the vector of path flows x_p . Denote by $D(x)$ the cost function of the problem of Eq. (5.52),

$$D(x) = \sum_{(i,j)} D_{ij} \left[\sum_{\substack{\text{all paths } p \\ \text{containing } (i,j)}} x_p \right]$$

and by $\partial D(x)/\partial x_p$ the partial derivative of D with respect to x_p . Then

$$\frac{\partial D(x)}{\partial x_p} = \sum_{\substack{\text{all links } (i,j) \\ \text{on path } p}} D'_{ij}$$

where the first derivatives D'_{ij} are evaluated at the total flows corresponding to x . It is seen that $\partial D/\partial x_p$ is the length of path p when the length of each link (i, j) is taken to be the first derivative D'_{ij} evaluated at x . Consequently, in what follows $\partial D/\partial x_p$ is called the first derivative length of path p .

Let $x^* = \{x_p^*\}$ be an optimal path flow vector. Then if $x_p^* > 0$ for some path p of an OD pair w , we must be able to shift a small amount $\delta > 0$ from path p to any other path p' of the same OD pair without improving the cost; otherwise, the optimality of x^* would be violated. To first order, the change in cost from this shift is

$$\delta \frac{\partial D(x^*)}{\partial x_{p'}} - \delta \frac{\partial D(x^*)}{\partial x_p}$$

and since this change must be nonnegative, we obtain

$$x_p^* > 0 \quad \Rightarrow \quad \frac{\partial D(x^*)}{\partial x_{p'}} \geq \frac{\partial D(x^*)}{\partial x_p}, \quad \text{for all } p' \in P_w \quad (5.53)$$

In words, *optimal path flow is positive only on paths with a minimum first derivative length*. Furthermore, at an optimum, the paths along which the input flow r_w of OD pair w is split must have *equal* length (and less or equal length to that of all other paths of w).

The condition (5.53) is a necessary condition for optimality of x^* . It can also be shown to be sufficient for optimality if the functions D_{ij} are convex; for example, when the second derivatives D''_{ij} exist and are positive in $[0, C_{ij})$, the domain of definition of D_{ij} (see Problem 5.24).

Example 5.7

Consider the two-link network shown in Fig. 5.61, where nodes 1 and 2 are the only origin and destination, respectively. The given input r is to be divided into the two path flows x_1 and x_2 so as to minimize a cost function based on the $M/M/1$ approximation

$$D(x) = D_1(x_1) + D_2(x_2)$$

where for $i = 1, 2$,

$$D_i(x_i) = \frac{x_i}{C_i - x_i}$$

and C_i is the capacity of link i . For the problem to make sense, we must assume that r is less than the maximum throughput $C_1 + C_2$ that the network can sustain.

At the optimum, the shortest path condition (5.53) must be satisfied as well as the constraints

$$x_1^* + x_2^* = r, \quad x_1^* \geq 0, \quad x_2^* \geq 0$$

Assume that $C_1 \geq C_2$. Then, from elementary reasoning, the optimal flow x_1 cannot be less than x_2 (it makes no sense to send more traffic on the slower link). The only possibilities are:

1. $x_1^* = r$ and $x_2^* = 0$. According to the shortest path condition (5.53), we must have

$$\frac{dD_1(r)}{dx_1} \leq \frac{dD_2(0)}{dx_2}$$

Equivalently [since the derivative of $x/(C - x)$ is $C/(C - x)^2$],

$$\frac{C_1}{(C_1 - r)^2} \leq \frac{1}{C_2}$$

or

$$r \leq C_1 - \sqrt{C_1 C_2} \quad (5.54)$$

2. $x_1^* > 0$ and $x_2^* > 0$. In this case, the shortest path condition (5.53) implies that the lengths of paths 1 and 2 are equal, that is,

$$\frac{dD_1(x_1^*)}{dx_1} = \frac{dD_2(x_2^*)}{dx_2}$$

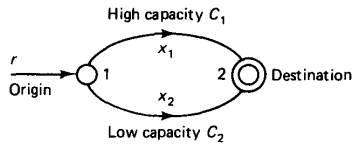


Figure 5.61 Example routing problem involving a single OD pair and two paths.

or, equivalently,

$$\frac{C_1}{(C_1 - x_1^*)^2} = \frac{C_2}{(C_2 - x_2^*)^2} \quad (5.55)$$

This equation together with the constraint $x_1^* + x_2^* = r$ determine the values of x_1^* and x_2^* . A straightforward calculation shows that the solution is

$$x_1^* = \frac{\sqrt{C_1} [r - (C_2 - \sqrt{C_1 C_2})]}{\sqrt{C_1} + \sqrt{C_2}}$$

$$x_2^* = \frac{\sqrt{C_2} [r - (C_1 - \sqrt{C_1 C_2})]}{\sqrt{C_1} + \sqrt{C_2}}$$

The optimal solution is shown in Fig. 5.62 for r in the range $[0, C_1 + C_2)$ of possible inputs. It can be seen that for

$$r \leq C_1 - \sqrt{C_1 C_2}$$

the faster link 1 is used exclusively. When r exceeds the threshold value $C_1 - \sqrt{C_1 C_2}$ for which the first derivative lengths of the two links are equalized and relation (5.54) holds as an equation, the slower link 2 is also utilized. As r increases, the flows on both links increase while the equality of the first derivative lengths, as in Eq. (5.55), is maintained. This behavior is typical of optimal routing when the cost function is based on the $M/M/1$ approximation; *for low input traffic, each OD pair tends to use only one path for routing (the fastest in terms of packet transmission time), and as traffic input increases, additional paths are used to avoid overloading the fastest path.* More generally, this type of behavior is associated with link cost functions D_{ij} with the property that the derivative $D'_{ij}(0)$ at zero flow depends only on the link capacity and decreases as the link capacity increases (see Problem 5.36).

We were able to solve the preceding example analytically only because of its extreme simplicity. Unfortunately, in more complex problems we typically have to resort to a computational solution—either centralized or distributed. The following sections deal with some of the possibilities.

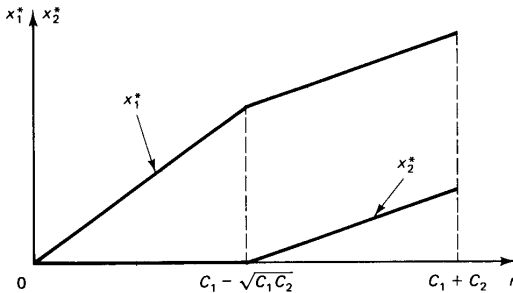


Figure 5.62 Optimal path flows for the routing example. When the traffic input is low, only the high-capacity link is used. As the input increases beyond the threshold $C_1 - \sqrt{C_1 C_2}$, some traffic is routed on the low-capacity link.

Finally, note that while the optimal routing problem was formulated in terms of a fixed set of input rates, the optimal solution $\{x_p^*\}$ can be implemented naturally even when the input rates are time varying. This can be done by working with the *fractions* of flow

$$\xi_p = \frac{x_p^*}{r_w}, \quad \text{for all } p \in P_w$$

and by requiring each OD pair w to divide traffic (packets or virtual circuits) among the available paths according to these fractions. In virtual circuit networks, where the origin nodes are typically aware of the detailed description of the paths used for routing, it is easy to implement the routing process in terms of the fractions ξ_p . The origins can estimate the data rate of virtual circuits and can route new virtual circuits so as to bring about a close match between actual and desired fractions of traffic on each path. In some datagram networks or in networks where the detailed path descriptions are not known at the origin nodes, it may be necessary to maintain in a routing table at each node i a routing variable $\phi_{ik}(j)$ for each link (i, k) and destination j . This routing variable is defined as the fraction of all flow arriving at node i , destined for node j , and routed along link (i, k) . Mathematically,

$$\phi_{ik}(j) = \frac{f_{ik}(j)}{\sum_m f_{im}(j)}, \quad \text{for all } (i, k) \text{ and } j \quad (5.56)$$

where $f_{ik}(j)$ is the flow that travels on link (i, k) and is destined for node j . Given an optimal solution of the routing problem in terms of the path flow variables $\{x_p^*\}$, it is possible to determine the corresponding link flow variables $f_{ik}(j)$ and, by Eq. (5.56), the corresponding optimal routing variables $\phi_{ik}(j)$. A direct formulation and distributed computational solution of the optimal routing problem in terms of the variables $\phi_{ik}(j)$ is given in references [Gal77], [Ber79a], [Gaf79], and [BG84].

5.6 FEASIBLE DIRECTION METHODS FOR OPTIMAL ROUTING

In Section 5.5 it was shown that optimal routing results only if flow travels along minimum first derivative length (MFDL) paths for each OD pair. Equivalently, a set of path flows is strictly suboptimal only if there is a positive amount of flow that travels on a non-MFDL path. This suggests that suboptimal routing can be improved by shifting flow to an MFDL path from other paths for each OD pair. The adaptive shortest path method for datagram networks of Section 5.2.5 does that in a sense, but shifts *all* flow of each OD pair to the shortest path, with oscillatory behavior resulting. It is more appropriate to shift only *part* of the flow of other paths to the shortest path. This section considers methods based on this idea. Generally, these methods solve the optimal routing problem computationally by decreasing the cost function through incremental changes in path flows.

Given a feasible path flow vector $x = \{x_p\}$ (i.e., a vector x satisfying the constraints of the problem), consider changing x along a direction $\Delta x = \{\Delta x_p\}$. There are two requirements imposed on the direction Δx :

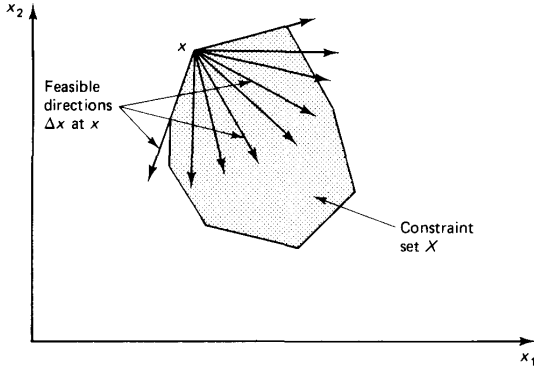


Figure 5.63 Feasible directions Δx at a feasible point x . Δx is a feasible direction if changing x by a small amount along the direction Δx still maintains feasibility.

1. The first requirement is that Δx should be a *feasible direction* in the sense that small changes along Δx maintain the feasibility of the path flow vector x (see Fig. 5.63). Mathematically, it is required that for some $\bar{\alpha} > 0$ and all $\alpha \in [0, \bar{\alpha}]$, the vector $x + \alpha \Delta x$ is feasible or equivalently,

$$\sum_{p \in P_w} \Delta x_p = 0, \quad \text{for all } w \in W \quad (5.57)$$

$$\Delta x_p \geq 0, \quad \text{for all } p \in P_w, \quad w \in W \text{ for which } x_p = 0 \quad (5.58)$$

Equation (5.57) follows from the feasibility requirement

$$\sum_{p \in P_w} (x_p + \alpha \Delta x_p) = r_w$$

and the fact that x is feasible, which implies that

$$\sum_{p \in P_w} x_p = r_w$$

It simply expresses that to maintain feasibility, all increases of flow along some paths must be compensated by corresponding decreases along other paths of the same OD pair. One way to obtain feasible directions is to select another feasible vector $\bar{x}$ and take

$$\Delta x = \bar{x} - x$$

In fact, a little thought reveals that all feasible directions can be obtained in this way up to scalar multiplication.

2. The second requirement is that Δx should be a *descent direction* in the sense that the cost function can be decreased by making small movements along the direction Δx starting from x (see Fig. 5.64). Since the gradient vector $\nabla D(x)$ is normal to the equal cost surfaces of the cost function D , it is clear from Fig. 5.64 that the

descent condition translates to the condition that the inner product of $\nabla D(x)$ and Δx is negative, that is,

$$\sum_{w \in W} \sum_{p \in P_w} \frac{\partial D(x)}{\partial x_p} \Delta x_p < 0 \quad (5.59)$$

For a mathematical verification, note that the inner product in Eq. (5.59) is equal to the first derivative of the function $G(\alpha) = D(x + \alpha \Delta x)$ at $\alpha = 0$, so Eq. (5.59) is equivalent to $G(\alpha)$ being negative for sufficiently small positive α . Note that the partial derivative $\partial D(x)/\partial x_p$ is given by

$$\frac{\partial D(x)}{\partial x_p} = \sum_{\substack{\text{all links } (i,j) \\ \text{on path } p}} D'_{ij}(F_{ij}),$$

and can be viewed as the first derivative length of path p [the length of link (i, j) is $D'_{ij}(F_{ij})$, cf. Section 5.5]. One way to satisfy the descent condition of Eq. (5.59), which is in fact commonly used in algorithms, is to require that Δx satisfies the conservation of flow condition $\sum_{p \in P_w} \Delta x_p = 0$ [cf. Eq. (5.57)], and that

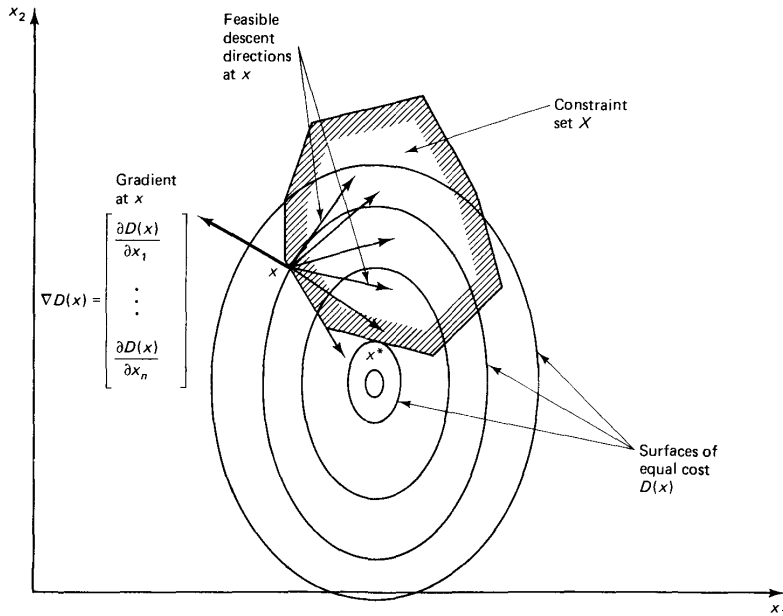


Figure 5.64 Descent directions at a feasible point x form an angle greater than 90° with the gradient $\nabla D(x)$. At the optimal point x^* , there are no feasible descent directions.

$$\Delta x_p \leq 0, \quad \text{for all paths } p \text{ that are nonshortest} \\ \text{in the sense } \partial D(x)/\partial x_p > \partial D(x)/\partial x_{\bar{p}} \text{ for some} \\ \text{path } \bar{p} \text{ of the same OD pair} \quad (5.60)$$

$$\Delta x_p < 0, \quad \text{for at least one nonshortest path } p.$$

In words, the conditions above together with the condition $\sum_{p \in P_w} \Delta x_p = 0$ state that some positive flow is shifted from nonshortest paths to the shortest paths [with respect to the lengths $D'_{ij}(F_{ij})$], and no flow is shifted from shortest paths to nonshortest ones. Since $\partial D(x)/\partial x_p$ is minimal for those (shortest) paths $\bar{p}$ for which $\Delta x_{\bar{p}} > 0$, it is seen that these conditions imply the descent condition (5.59).

We thus obtain a broad class of iterative algorithms for solving the optimal routing problem. Their basic iteration is given by

$$x := x + \alpha \Delta x$$

where Δx is a feasible descent direction [*i.e.*, satisfies conditions (5.57) to (5.59) or (5.57), (5.58), and (5.60)], and α is a positive stepsize chosen so that the cost function is decreased, that is,

$$D(x + \alpha \Delta x) < D(x)$$

and the vector $x + \alpha \Delta x$ is feasible. The stepsize α could be different in each iteration. It can be seen that a feasible descent direction can be found if x violates the shortest path optimality condition (5.53) of section 5.5. Figure 5.65 illustrates the operation of such an algorithm.

5.6.1 The Frank–Wolfe (Flow Deviation) Method

Here is one way to implement the philosophy of incremental changes along feasible descent directions:

Given a feasible path flow vector $x = \{x_p\}$, find a minimum first derivative length (MFDL) path for each OD pair. (The first derivatives D'_{ij} are evaluated, of course, at the current vector x .) Let $\bar{x} = \{\bar{x}_p\}$ be the vector of path flows that would result if all input r_w for each OD pair $w \in W$ is routed along the corresponding MFDL path. Let α^* be the stepsize that minimizes $D[x + \alpha(\bar{x} - x)]$ over all $\alpha \in [0, 1]$, that is,

$$D[x + \alpha^*(\bar{x} - x)] = \min_{\alpha \in [0, 1]} D[x + \alpha(\bar{x} - x)] \quad (5.61)$$

The new set of path flows is obtained by

$$x_p := x_p + \alpha^*(\bar{x}_p - x_p), \quad \text{for all } p \in P_w, w \in W \quad (5.62)$$

and the process is repeated.

The algorithm above is a special case of the so-called *Frank–Wolfe method* for solving general, nonlinear programming problems with convex constraint sets (see [Zan69]). It has been called the *flow deviation method* (see [FGK73]), and can be shown to reduce the value of the cost function to its minimum in the limit (see Problem 5.31) although its convergence rate near the optimum tends to be very slow.

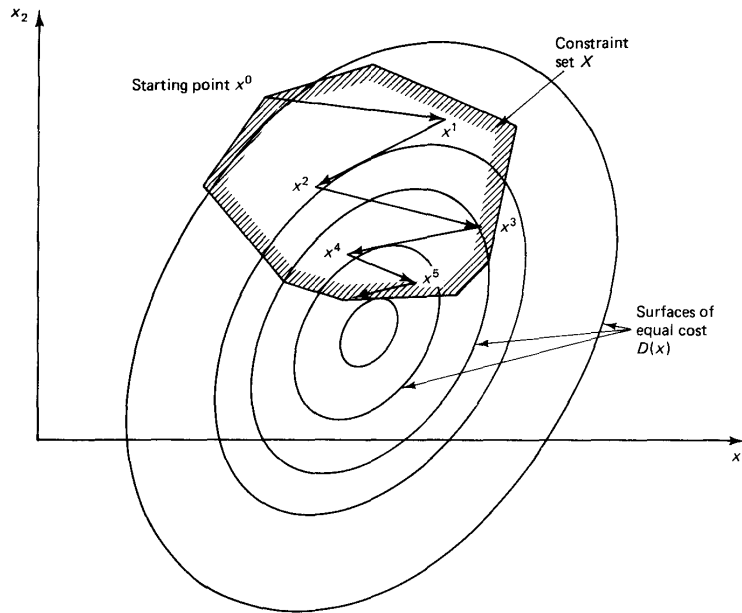


Figure 5.65 Sample path of an iterative descent method based on feasible descent directions. At each iteration, a feasible point of lower cost is obtained.

Note that the feasible direction used in the Frank–Wolfe iteration (5.62) is of the form (5.60); that is, a proportion α^* of the flow of all the nonshortest paths (those paths for which $x_p = 0$) is shifted to the shortest path (the one for which $\bar{x}_p = r_w$) for each OD pair w . The characteristic property here is that flow is shifted from the nonshortest paths in *equal* proportions. This distinguishes the Frank–Wolfe method from the gradient projection methods discussed in the next section. The latter methods also shift flow from the nonshortest paths to the shortest paths; however, they do so in generally unequal proportions.

Here is a simple example illustrating the Frank–Wolfe method:

Example 5.8

Consider the three-link network with one origin and one destination, shown in Fig. 5.66. There are three paths with corresponding flows x_1 , x_2 , and x_3 , which must satisfy the constraints

$$x_1 + x_2 + x_3 = 1, \quad x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0$$

The cost function is

$$D(x) = \frac{1}{2} (x_1^2 + x_2^2 + 0.1x_3^2) + 0.55x_3$$

(The linear term $0.55x_3$ can be attributed to a large processing and propagation delay on link 3.)

This is an easy problem that can be solved analytically. It can be argued (or shown using the optimality condition of Section 5.5) that at an optimal solution $x^* = (x_1^*, x_2^*, x_3^*)$, we must have by symmetry $x_1^* = x_2^*$, so there are two possibilities:

- (a) $x_3^* = 0$ and $x_1^* = x_2^* = 1/2$.
- (b) $x_3^* = \beta > 0$ and $x_1^* = x_2^* = (1 - \beta)/2$.

Case (b) is not possible because according to the optimality condition of Section 5.5, if $x_3^* > 0$, the length of path 3 $[= \partial D(x^*)/\partial x_3 = 0.1\beta + 0.55]$ must be less or equal to the lengths of paths 1 and 2 $[= \partial D(x^*)/\partial x_1 = (1 - \beta)/2]$, which is clearly false. Therefore, the optimal solution is $x^* = (1/2, 1/2, 0)$.

Consider now application of the Frank–Wolfe iteration (5.68) at a feasible path flow vector $x = (x_1, x_2, x_3)$. The three paths have first derivative lengths

$$\frac{\partial D(x)}{\partial x_1} = x_1, \quad \frac{\partial D(x)}{\partial x_2} = x_2, \quad \frac{\partial D(x)}{\partial x_3} = 0.1x_3 + 0.55$$

It is seen, using essentially the same argument as above, that the shortest path is either 1 or 2, depending on whether $x_1 \leq x_2$ or $x_2 < x_1$, and the corresponding shortest-path flows are $\bar{x} = (1, 0, 0)$ and $\bar{x} = (0, 1, 0)$, respectively. (The tie is broken arbitrarily in favor of path 1 in case $x_1 = x_2$.) Therefore, the Frank–Wolfe iteration takes the form

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} := \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \alpha^* \begin{bmatrix} 1 - x_1 \\ -x_2 \\ -x_3 \end{bmatrix} = \begin{bmatrix} x_1 + \alpha^*(x_2 + x_3) \\ (1 - \alpha^*)x_2 \\ (1 - \alpha^*)x_3 \end{bmatrix}, \quad \text{if } x_1 \leq x_2$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} := \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \alpha^* \begin{bmatrix} -x_1 \\ 1 - x_2 \\ -x_3 \end{bmatrix} = \begin{bmatrix} (1 - \alpha^*)x_1 \\ x_2 + \alpha^*(x_1 + x_3) \\ (1 - \alpha^*)x_3 \end{bmatrix}, \quad \text{if } x_2 < x_1$$

Thus, at each iteration, a proportion α^* of the flows of the nonshortest paths is shifted to the shortest path.

The stepsize α^* is obtained by line minimization over $[0, 1]$ [cf. Eq. (5.67)]. Because $D(x)$ is quadratic, this minimization can be done analytically. We have

$$\begin{aligned} D[x + \alpha(\bar{x} - x)] &= \frac{1}{2} \left([x_1 + \alpha(\bar{x}_1 - x_1)]^2 + [x_2 + \alpha(\bar{x}_2 - x_2)]^2 \right. \\ &\quad \left. + 0.1[x_3 + \alpha(\bar{x}_3 - x_3)]^2 \right) + 0.55[x_3 + \alpha(\bar{x}_3 - x_3)] \end{aligned}$$

Differentiating with respect to α and setting the derivative to zero we obtain the unconstrained minimum $\bar{\alpha}$ of $D[x + \alpha(\bar{x} - x)]$ over α . This is given by

$$\bar{\alpha} = -\frac{x_1(\bar{x}_1 - x_1) + x_2(\bar{x}_2 - x_2) + (0.1x_3 + 0.55)(\bar{x}_3 - x_3)}{(\bar{x}_1 - x_1)^2 + (\bar{x}_2 - x_2)^2 + 0.1(\bar{x}_3 - x_3)^2}$$

where $\bar{x} = (\bar{x}_1, \bar{x}_2, \bar{x}_3)$ is the current shortest path flow vector $[(1,0,0)$ or $(0,1,0)$, depending on whether $x_1 \leq x_2$ or $x_2 < x_1]$. Since $(\bar{x} - x)$ is a descent direction, we have $\bar{\alpha} \geq 0$. Therefore, the stepsize α^* , which is the constrained minimum over $[0,1]$ [cf. Eq. (5.67)], is given by

$$\alpha^* = \min[1, \bar{\alpha}]$$

(In this example, it can be verified that $\bar{\alpha} < 1$, implying that $\alpha^* = \bar{\alpha}$.)

Figures 5.66 and 5.67 show successive iterates of the method. It can be seen that the rate of convergence becomes slow near the optimal solution. The reason is that as x^* is approached, the directions of search tend to become orthogonal to the direction leading from the current iterate x^k to x^* . In fact, it can be seen from Fig. 5.66 that the ratio of successive cost errors

$$\frac{D(x^{k+1}) - D(x^*)}{D(x^k) - D(x^*)}$$

while always less than 1, actually converges to 1 as $k \rightarrow \infty$. This is known as *sublinear* convergence rate [Ber82d], and is typical of the Frank–Wolfe method ([CaC68] and [Dun79]).

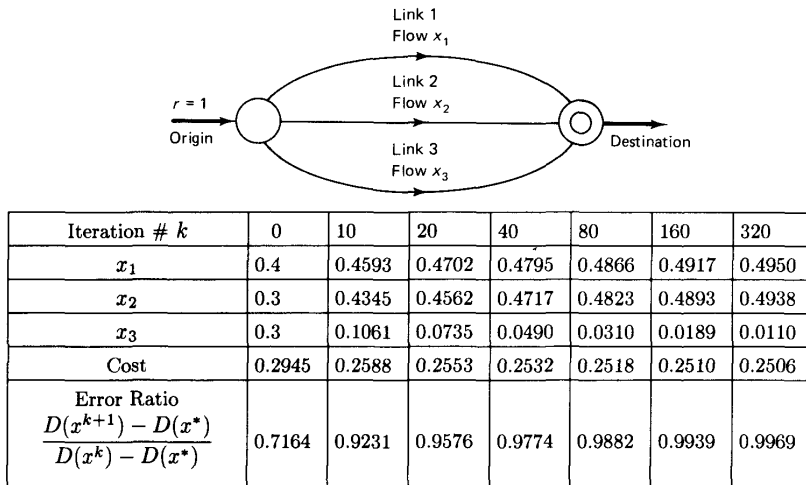


Figure 5.66 Example problem and successive iterates of the Frank–Wolfe method. The cost function is

$$D(x) = \frac{1}{2} (x_1^2 + x_2^2 + x_3^2) + 0.55x_3$$

The optimal solution is $x^* = (\frac{1}{2}, \frac{1}{2}, 0)$. As the optimal solution is approached, the method becomes slower. In the limit, the ratio of successive errors $D(x^{k+1}) - D(x^*) / D(x^k) - D(x^*)$ tends to unity.

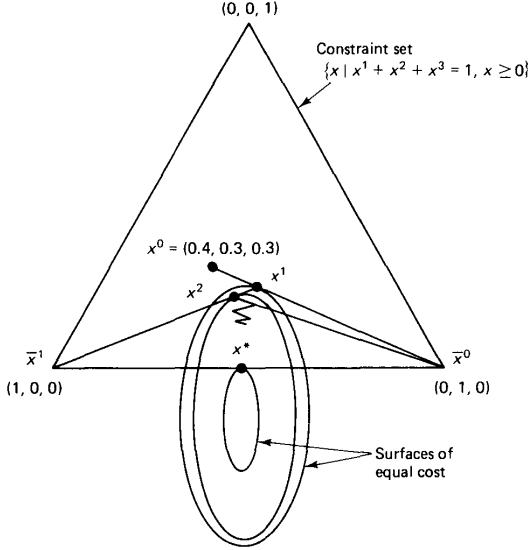


Figure 5.67 Successive iterates of the Frank-Wolfe method in the two-node, three-link example. Slow convergence is due to the fact that as the optimal solution x^* is approached, the directions of search tend to become orthogonal to the direction leading to the optimum.

The determination of an optimal stepsize α^* satisfying Eq. (5.61) requires a one-dimensional minimization over $[0,1]$ which can be carried out through any one of several existing methods (see [Lue84]). A simpler method is to choose the stepsize α^* by means of

$$\alpha^* = \min \left[1, - \frac{\sum_{(i,j)} (\bar{F}_{ij} - F_{ij}) D'_{ij}}{\sum_{(i,j)} (\bar{F}_{ij} - F_{ij})^2 D''_{ij}} \right] \quad (5.63)$$

Here $\{F_{ij}\}$ and $\{\bar{F}_{ij}\}$ are the sets of total link flows corresponding to $\{x_p\}$ and $\{\bar{x}_p\}$, respectively [i.e., F_{ij} (or $\bar{F}_{ij}$) is obtained by adding all flows x_p (or $\bar{x}_p$) of paths p traversing the link (i, j)]. The first and second derivatives, D'_{ij} and D''_{ij} , respectively, are evaluated at F_{ij} . The formula for α^* in Eq. (5.63) is obtained by making a second-order Taylor approximation $\tilde{G}(\alpha)$ of $G(\alpha) = D[x + \alpha(\bar{x} - x)]$ around $\alpha = 0$:

$$\tilde{G}(\alpha) = \sum_{(i,j)} \left\{ D_{ij}(F_{ij}) + \alpha D'_{ij}(F_{ij})(\bar{F}_{ij} - F_{ij}) + \frac{\alpha^2}{2} D''_{ij}(F_{ij})(\bar{F}_{ij} - F_{ij})^2 \right\}$$

and by minimizing $\tilde{G}(\alpha)$ with respect to α over the interval $[0,1]$ as in the earlier example.

It can be shown that the algorithm with the choice of Eq. (5.63) for the stepsize converges to the optimal set of total link flows provided that the starting set of total link flows is sufficiently close to the optimal. For the type of cost functions used in routing problems [e.g., the $M/M/1$ delay formula; cf. Eq. (5.30) in Section 5.4], it appears that the simple stepsize choice of Eq. (5.63) typically leads to convergence even when the starting total link flows are far from optimal. This stepsize choice is also well suited for distributed implementation. However, even with this rule, the method appears inferior for distributed

application to the projection method of the next section primarily because it is essential for the network nodes to synchronize their calculations in order to obtain the stepsize.

The Frank–Wolfe method has an insightful geometrical interpretation. Suppose that we have a feasible path flow vector $x = \{x_p\}$, and we try to find a path flow variation $\Delta x = \{\Delta x_p\}$ which is feasible in the sense of Eqs. (5.57) and (5.58), that is,

$$\begin{aligned} \sum_{p \in P_w} \Delta x_p &= 0, \quad \text{for all } w \in W \\ x_p + \Delta x_p &\geq 0, \quad \text{for all } p \in P_w, w \in W \end{aligned} \quad (5.64)$$

and along which the initial rate of change [cf. Eq. (5.59)]

$$\sum_{w \in W} \sum_{p \in P_w} \frac{\partial D(x)}{\partial x_p} \Delta x_p$$

of the cost function D is most negative. Such a variation Δx is obtained by solving the optimization problem

$$\begin{aligned} &\text{minimize } \sum_{w \in W} \sum_{p \in P_w} \frac{\partial D(x)}{\partial x_p} \Delta x_p \\ &\text{subject to } \sum_{p \in P_w} \Delta x_p = 0, \quad \text{for all } w \in W \\ &\quad x_p + \Delta x_p \geq 0, \quad \text{for all } p \in P_w, w \in W \end{aligned} \quad (5.64)$$

and it is seen that $\Delta x = \bar{x} - x$ is an optimal solution, where $\bar{x}$ is the shortest path flow vector generated by the Frank–Wolfe method. The process of finding Δx can be visualized as in Fig. 5.68. It can be seen that $\bar{x}$ is a vertex of the polyhedron of feasible path flow vectors that lies farthest out along the negative gradient direction $-\nabla D(x)$, thereby minimizing the inner product of $\nabla D(x)$ and Δx subject to the constraints (5.64).

Successive iterations of the Frank–Wolfe method are shown in Fig. 5.69. At each iteration, a vertex of the feasible set (*i.e.*, the shortest path flow $\bar{x}$) is obtained by solving a shortest path problem; then the next path flow vector is obtained by a search along the line joining the current path flow vector with the vertex. The search ensures that the new path flow vector has lower cost than the preceding one.

We finally mention an important situation where the Frank–Wolfe method has an advantage over other methods. Suppose that one is not interested in obtaining optimal path flows, but that the only quantities of interest are the optimal total link flows F_{ij} or just the value of optimal cost. (This arises in topological design studies; see Section 5.4.2.) Then the Frank–Wolfe method can be implemented in a way that only the current total link flows together with the current shortest paths for all OD pairs are maintained in memory at each iteration. This can be done by computing the total link flows corresponding to the shortest path flow vector $\bar{x}$ and executing the stepsize minimization indicated in Eq. (5.61) in the space of total link flows. The amount of storage required in this implementation is relatively small, thereby allowing the solution of very large network problems. If, however, one is interested in optimal path flows as well as total

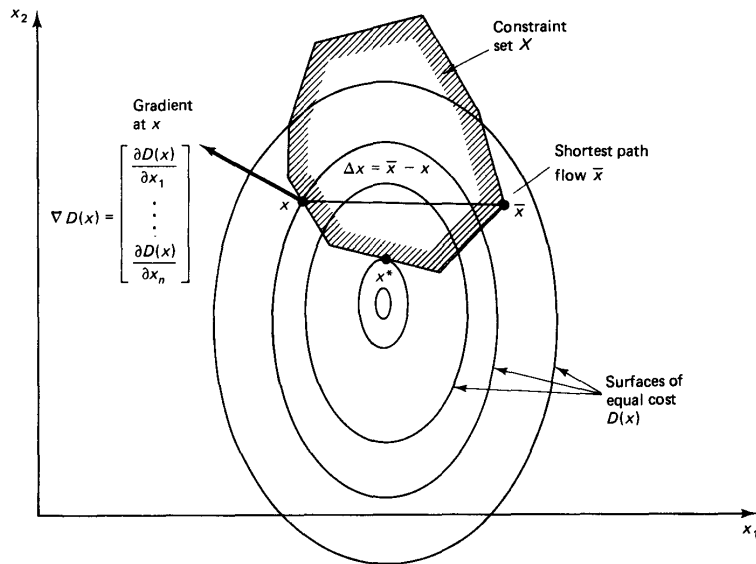


Figure 5.68 Finding the feasible descent direction $\Delta x = \bar{x} - x$ at a point x in the Frank-Wolfe method. $\bar{x}$ is the shortest path flow (or extreme point of the feasible set X) that lies farthest out along the negative gradient direction $-\nabla D(x)$.

link flows, the storage requirements of the Frank-Wolfe method are about the same as those of other methods, including the projection methods discussed in the next section.

5.7 PROJECTION METHODS FOR OPTIMAL ROUTING

We now consider a class of feasible direction algorithms for optimal routing that are faster than the Frank-Wolfe method and lend themselves more readily to distributed implementation. These methods are also based on shortest paths and determine a minimum first derivative length (MFDL) path for every OD pair at each iteration. An increment of flow change is calculated for each path on the basis of the relative magnitudes of the path lengths and, sometimes, the second derivatives of the cost function. If the increment is so large that the path flow becomes negative, the path flow is simply set to zero (*i.e.*, it is “projected” back onto the positive orthant). These routing algorithms may be viewed as constrained versions of common, unconstrained optimization methods, such as steepest descent and Newton’s method, which are given in nonlinear programming texts (*e.g.*, [Zan69], [Pol71], [Ber82d], and [Lue84]). We first present these methods briefly in a general, nonlinear optimization setting, and subsequently specialize them to the routing problem.

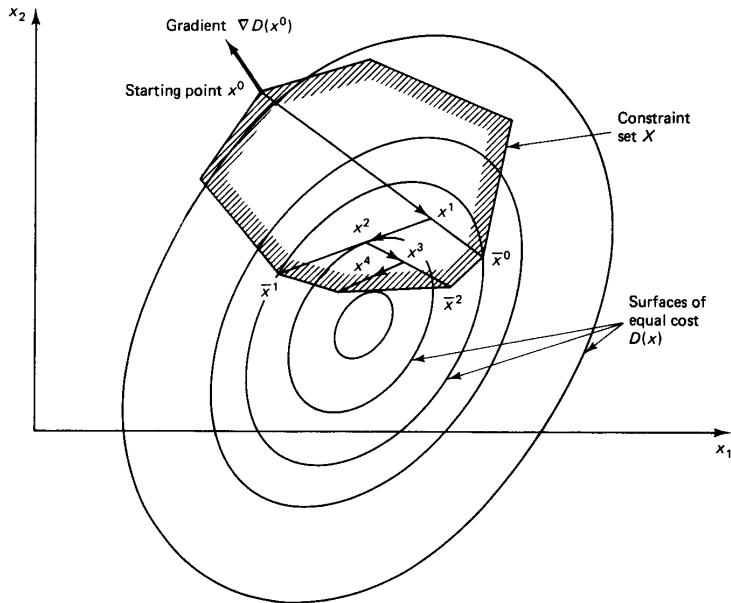


Figure 5.69 Successive iterations of the Frank-Wolfe method. At each iteration, a vertex of the feasible set (a shortest path flow) is found. The next path flow vector is obtained by a search on the line segment connecting the current path flow vector with the vertex.

5.7.1 Unconstrained Nonlinear Optimization

Let f be a twice differentiable function of the n -dimensional vector $x = (x_1, \dots, x_n)$, with a gradient and Hessian matrix at any x denoted $\nabla f(x)$ and $\nabla^2 f(x)$:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}, \quad \nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{(\partial x_1)^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(x)}{(\partial x_n)^2} \end{bmatrix}$$

We assume that for all x , $\nabla^2 f(x)$ is a positive semidefinite matrix that depends continuously on x .^{*} The *method of steepest descent* for finding an unconstrained minimum of

^{*} A symmetric $n \times n$ matrix A with elements A_{ij} is said to be *positive semidefinite* if the quadratic form

$$\sum_{i=1}^n \sum_{j=1}^n A_{ij} z_i z_j$$

is nonnegative for all vectors $z = (z_1, \dots, z_n)$. The matrix A is said to be *positive definite* if this quadratic form is positive for all vectors $z \neq 0$.

f starts with some initial guess x^0 and proceeds according to the iteration

$$x^{k+1} = x^k - \alpha^k \nabla f(x^k), \quad k = 0, 1, \dots \quad (5.65)$$

where α^k is a positive scalar stepsize, determined according to some rule. The idea here is similar to the one discussed in the preceding section, namely changing x^k along a descent direction. In the preceding iteration, the change is made along the negative gradient, which is a descent direction since it makes a negative inner product with the gradient vector [unless $\nabla f(x^k) = 0$, in which case the vector x^k is optimal]. Common choices for α^k are the minimizing stepsize determined by

$$f[x^k - \alpha^k \nabla f(x^k)] = \min_{\alpha > 0} f[x^k - \alpha \nabla f(x^k)] \quad (5.66)$$

and a constant positive stepsize $\bar{\alpha}$

$$\alpha^k \equiv \bar{\alpha}, \quad \text{for all } k \quad (5.67)$$

There are a number of convergence results for steepest descent methods. For example, if f has a unique unconstrained minimizing point, it may be shown that the sequence $\{x^k\}$ generated by the steepest descent method of Eqs. (5.65) and (5.66) converges to this minimizing point for every starting x^0 (see [Ber82d] and [Lue84]). Also, given any starting vector x^0 , the sequence generated by this steepest descent method converges to the minimizing point provided that $\bar{\alpha}$ is chosen sufficiently small. Unfortunately, however, the speed of convergence of $\{x^k\}$ can be quite slow. It can be shown ([Lue84], p. 218) that for the case of the line minimization rule of Eq. (5.66), if f is a positive definite quadratic function, there holds

$$\frac{f(x^{k+1}) - f^*}{f(x^k) - f^*} \leq \left(\frac{M - m}{M + m} \right)^2 \quad (5.68)$$

where $f^* = \min_x f(x)$, and M, m are the largest and smallest eigenvalues of $\nabla^2 f(x)$, respectively. Furthermore, there exist starting points x^0 such that Eq. (5.68) holds with equality for every k . So if the ratio M/m is large (this corresponds to the equal-cost surfaces of f being very elongated ellipses), the rate of convergence is slow. Similar results can be shown for the steepest descent method with the constant stepsize (5.67), and these results also hold in a qualitatively similar form for functions f with an everywhere continuous and positive definite Hessian matrix.

The rate of convergence of the steepest descent method can be improved by premultiplying the gradient by a suitable positive definite scaling matrix B^k , thereby obtaining the iteration

$$x^{k+1} = x^k - \alpha^k B^k \nabla f(x^k), \quad k = 0, 1, \dots \quad (5.69)$$

This method is also based on the idea of changing x^k along a descent direction. [The direction of change $-B^k \nabla f(x^k)$ makes a negative inner product with $\nabla f(x^k)$, since B^k is a positive definite matrix.] From the point of view of rate of convergence, the best

method is obtained with the choice

$$B^k = [\nabla^2 f(x^k)]^{-1} \quad (5.70)$$

[assuming that $\nabla^2 f(x^k)$ is invertible]. This is *Newton's method*, which can be shown to have a very fast (superlinear) rate of convergence near the minimizing point when the stepsize α^k is taken as unity. Unfortunately, this excellent convergence rate is achieved at the expense of the potentially substantial overhead associated with the inversion of $\nabla^2 f(x^k)$. It is often useful to consider other choices of B^k which approximate the "optimal" choice $[\nabla^2 f(x^k)]^{-1}$ but do not require as much overhead. A simple choice that often works well is to take B^k as a diagonal approximation to the inverse Hessian, that is,

$$B^k = \begin{bmatrix} \left[\frac{\partial^2 f(x^k)}{(\partial x_1)^2} \right]^{-1} & 0 & \cdots & 0 \\ 0 & \left[\frac{\partial^2 f(x^k)}{(\partial x_2)^2} \right]^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \left[\frac{\partial^2 f(x^k)}{(\partial x_n)^2} \right]^{-1} \end{bmatrix} \quad (5.71)$$

With this choice, the scaled gradient iteration (5.69) can be written in the simple form

$$x_i^{k+1} = x_i^k - \alpha^k \left[\frac{\partial^2 f(x^k)}{(\partial x_i)^2} \right]^{-1} \frac{\partial f(x^k)}{\partial x_i}, \quad i = 1, \dots, n \quad (5.72)$$

5.7.2 Nonlinear Optimization over the Positive Orthant

Next consider the problem of minimizing the function f subject to the nonnegativity constraints $x_i \geq 0$, for $i = 1, \dots, n$, that is, the problem

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } x \geq 0 \end{aligned} \quad (5.73)$$

A straightforward analog of the steepest descent method known as the *gradient projection method* ([Gol64] and [LeP65]), is given by

$$x^{k+1} = [x^k - \alpha^k \nabla f(x^k)]^+, \quad k = 0, 1, \dots \quad (5.74)$$

where for any vector z , we denote by $[z]^+$ the projection of z onto the positive orthant

$$[z]^+ = \begin{bmatrix} \max\{0, z_1\} \\ \max\{0, z_2\} \\ \vdots \\ \max\{0, z_n\} \end{bmatrix} \quad (5.75)$$

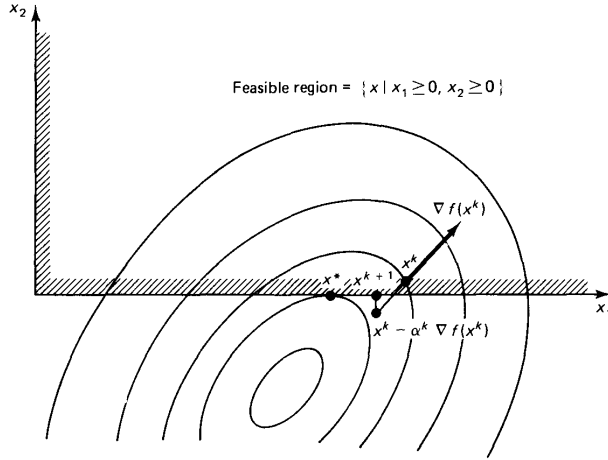


Figure 5.70 Operation of the gradient projection method. A step is made in the direction of the negative gradient, and the result is orthogonally projected on the positive orthant. It can be shown that the step $(x^{k+1} - x^k)$ is a descent direction [makes a negative inner product with $\nabla f(x^k)$], and if the stepsize α is sufficiently small, the iteration improves the cost [i.e., $f(x^{k+1}) < f(x^k)$].

An illustration of how this method operates is given in Fig. 5.70. It can be shown that the convergence results mentioned earlier in connection with the unconstrained steepest descent method of Eq. (5.65) also hold true for its constrained analog of Eq. (5.74) (see Problem 5.32 and [Ber76]). The same is true for the method

$$x^{k+1} = [x^k - \alpha^k B^k \nabla f(x^k)]^+$$

where B^k is a *diagonal*, positive definite scaling matrix (see Problem 5.33). When B^k is given by the diagonal inverse Hessian approximation of Eq. (5.71), the following iteration is obtained:

$$x_i^{k+1} = \max \left\{ 0, x_i^k - \alpha^k \left[\frac{\partial^2 f(x^k)}{(\partial x_i)^2} \right]^{-1} \frac{\partial f(x^k)}{\partial x_i} \right\}, \quad i = 1, \dots, n \quad (5.76)$$

5.7.3 Application to Optimal Routing

Consider now the optimal routing problem

$$\begin{aligned} &\text{minimize } D(x) = \sum_{(i,j)} D_{ij}(F_{ij}) \\ &\text{subject to } \sum_{p \in P_w} x_p = r_w, \quad x_p \geq 0, \quad \text{for all } p \in P_w, w \in W \end{aligned} \quad (5.77)$$

where each total link flow F_{ij} is expressed in terms of the path flow vector $x = \{x_p\}$ as the sum of path flows traversing the link (i, j) . Assume that the second derivatives of D_{ij} , denoted by $D''_{ij}(F_{ij})$, are positive for all F_{ij} . Let $x^k = \{x_p^k\}$ be the path flow vector obtained after k iterations, and let $\{F_{ij}^k\}$ be the corresponding set of total link flows. For each OD pair w , let $\bar{p}_w$ be an MFDL path [with respect to the current link lengths $D'_{ij}(F_{ij}^k)$].

The optimal routing problem (5.77) can be converted (for the purpose of the next iteration) to a problem involving only positivity constraints by expressing the flows of the MFDL paths $\bar{p}_w$ in terms of the other path flows, while eliminating the equality constraints

$$\sum_{p \in P_w} x_p = r_w$$

in the process. For each w , $x_{\bar{p}_w}$ is substituted in the cost function $D(x)$ using the equation

$$x_{\bar{p}_w} = r_w - \sum_{\substack{p \in P_w \\ p \neq \bar{p}_w}} x_p \quad (5.78)$$

thereby obtaining a problem of the form

$$\begin{aligned} & \text{minimize } \tilde{D}(\tilde{x}) \\ & \text{subject to } x_p \geq 0, \quad \text{for all } w \in W, p \in P_w, p \neq \bar{p}_w \end{aligned} \quad (5.79)$$

where $\tilde{x}$ is the vector of all path flows which are *not* MFDL paths.

We now calculate the derivatives that will be needed to apply the scaled projection iteration (5.76) to the problem of Eq. (5.79). Using Eq. (5.78) and the definition of $\tilde{D}(\tilde{x})$, we obtain

$$\frac{\partial \tilde{D}(\tilde{x}^k)}{\partial x_p} = \frac{\partial D(x^k)}{\partial x_p} - \frac{\partial D(x^k)}{\partial x_{\bar{p}_w}}, \quad \text{for all } p \in P_w, p \neq \bar{p}_w \quad (5.80)$$

for all $w \in W$. In Section 5.5 it was shown that $\partial D(x)/\partial x_p$ is the first derivative length of path p , that is,

$$\frac{\partial D(x^k)}{\partial x_p} = \sum_{\substack{\text{all links} \\ (i,j) \text{ on path } p}} D'_{ij}(F_{ij}^k), \quad (5.81)$$

Regarding second derivatives, a straightforward differentiation of the first derivative expressions (5.80) and (5.81) shows that

$$\frac{\partial^2 \tilde{D}(\tilde{x}^k)}{(\partial x_p)^2} = \sum_{(i,j) \in L_p} D''_{ij}(F_{ij}^k), \quad \text{for all } w \in W, p \in P_w, p \neq \bar{p}_w \quad (5.82)$$

where, for each p ,

$$L_p = \begin{aligned} & \text{Set of links belonging to either } p, \text{ or} \\ & \text{the corresponding MFDL path } \bar{p}_w, \text{ but not both} \end{aligned}$$

Expressions for both the first and second derivatives of the “reduced” cost $\tilde{D}(\tilde{x})$, are now available and thus the scaled projection method of Eq. (5.76) can be applied. The iteration takes the form [cf. Eqs. (5.76), (5.80), and (5.82)]

$$x_p^{k+1} = \max\{0, x_p^k - \alpha^k H_p^{-1}(d_p - d_{\bar{p}_w})\}, \quad \text{for all } w \in W, p \in P_w, p \neq \bar{p}_w \quad (5.83)$$

where d_p and $d_{\bar{p}_w}$ are the first derivative lengths of the paths p and $\bar{p}_w$ given by [cf. Eq. (5.81)]

$$d_p = \sum_{\substack{\text{all links } (i,j) \\ \text{on path } p}} D'_{ij}(F_{ij}^k), \quad d_{\bar{p}_w} = \sum_{\substack{\text{all links } (i,j) \\ \text{on path } \bar{p}_w}} D'_{ij}(F_{ij}^k), \quad (5.84)$$

and H_p is the “second derivative length”

$$H_p = \sum_{(i,j) \in L_p} D''_{ij}(F_{ij}^k) \quad (5.85)$$

given by Eq. (5.82). The stepsize α^k is some positive scalar which may be chosen by a variety of methods. For example, α^k can be constant or can be chosen by some form of line minimization. Stepsize selection will be discussed later. We refer to the iteration of Eqs. (5.83) to (5.85) as the *projection algorithm*.

The following observations can be made regarding the projection algorithm:

1. Since $d_p \geq d_{\bar{p}_w}$ for all $p \neq \bar{p}_w$, all the nonshortest path flows that are positive will be reduced with the corresponding increment of flow being shifted to the MFDL path $\bar{p}_w$. If the stepsize α^k is large enough, all flow from the nonshortest paths will be shifted to the shortest path. Therefore, the projection algorithm may also be viewed as a generalization of the adaptive routing method based on shortest paths of Section 5.2.5, with α^k , H_p , and $(d_p - d_{\bar{p}_w})$ determining the amounts of flow shifted to the shortest path.
2. Those nonshortest path flows x_p , $p \neq \bar{p}_w$ that are zero will stay at zero. Therefore, the path flow iteration of Eq. (5.83) should only be carried out for paths that carry positive flow.
3. Only paths that carried positive flow at the starting flow pattern or were MFDL paths at some previous iteration can carry positive flow at the beginning of an iteration. This is important since it tends to keep the number of flow carrying paths small, with a corresponding reduction in the amount of calculation and bookkeeping needed at each iteration.

There are several ways to choose the stepsize α^k . One possibility is to keep α^k constant ($\alpha^k \equiv \alpha$, for all k). With this choice it can be shown that given any starting set of path flows, there exists $\bar{\alpha} > 0$ such that if $\alpha \in (0, \bar{\alpha}]$, a sequence generated by the projection algorithm converges to the optimal cost of the problem (see Problem 5.32). A crucial question has to do with the magnitude of the constant stepsize. It is known from

nonlinear programming experience and analysis that a stepsize equal to 1 usually works well with Newton's method as well as diagonal approximations to Newton's method [cf. Eq. (5.71)] that employ scaling based on second derivatives ([Lue84] and [Ber82d]).

Experience has verified that choosing α^k in the path flow iteration (5.83) close to 1 typically works quite well regardless of the values of the input flows r_w [BGV79]. Even better performance is usually obtained if the iteration is carried out *one OD pair (or one origin) at a time*; that is, first carry out the iteration with $\alpha^k = 1$ for a single OD pair (or origin), adjust the corresponding total link flows to account for the effected change in the path flows of this OD pair (or origin), and continue with the next OD pair until all OD pairs are taken up cyclically. The rationale for this is that dropping the off-diagonal terms of the Hessian matrix [cf. Eqs. (5.69) and (5.71)], in effect, neglects the interaction between the flows of different OD pairs. In other words, the path flow iteration (5.83) is based to some extent on the premise that each OD pair will adjust its own path flows while the other OD pairs will keep theirs unchanged. Carrying out this iteration one OD pair at a time reduces the potentially detrimental effect of the neglected off-diagonal terms and increases the likelihood that the unity stepsize is appropriate and effective.

Using a constant stepsize is well suited for distributed implementation. For centralized computation, it is possible to choose α^k by a simple form of line search. For example, start with a unity stepsize, evaluate the corresponding cost, and if no reduction is obtained over $D(x^k)$, successively reduce the stepsize until a cost reduction $D(x^{k+1}) < D(x^k)$ is obtained. (It is noted for the theoretically minded that this scheme cannot be shown to converge to the optimal solution. However, typically, it works well in practice. Similar schemes with better theoretical convergence properties are described in [Ber76] and [Ber82c].) Still another possibility for stepsize selection is discussed in Problem 5.32.

The projection algorithm typically yields rapid convergence to a neighborhood of an optimal solution. Once it comes near a solution (how "near" depends on the problem), it tends to slow down. Its progress is often satisfactory near a solution and usually far better than that of the Frank–Wolfe method. Figure 5.71 provides a comparison of the projection method and the Frank–Wolfe method for the example of the preceding section.

To obtain faster convergence near an optimal solution, it is necessary to modify the projection algorithm so that the off-diagonal terms of the Hessian matrix are taken into account. Surprisingly, it is possible to implement sophisticated methods of this type (see [BeG83]), but we will not go into this further. Suffice to say that these methods are based on a more accurate approximation of a constrained version of Newton's method (using the conjugate gradient method) near an optimal solution. However, when far from a solution, their speed of convergence is usually only slightly superior to that of the projection algorithm. So, if one is only interested in getting fast near an optimal solution in few iterations, but the subsequent rate of progress is of little importance (as is often the case in practical routing problems), the projection algorithm is typically satisfactory.

We finally note that the projection algorithm is well suited for distributed implementation. The most straightforward possibility is for all nodes i to broadcast to all other nodes the current total flows F_{ij}^k of their outgoing links (i, j) , using, for example, a flooding algorithm or the SPTA of Section 5.3.3. Each node then computes the MFDL

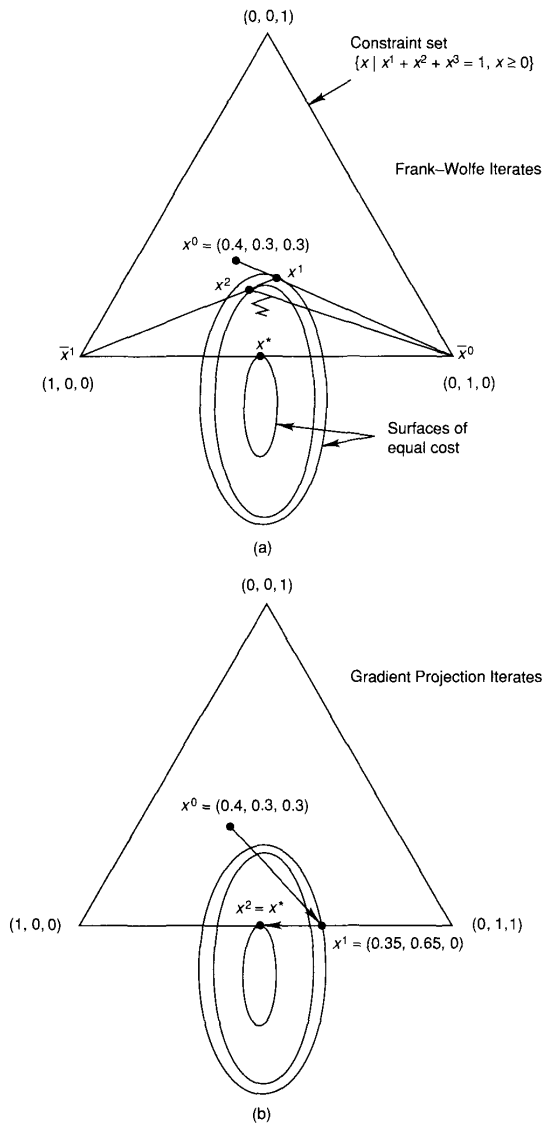


Figure 5.71 Iterates of (a) the Frank-Wolfe and (b) the gradient projection method for the example of the preceding section. The gradient projection method converges much faster. It sets the flow x_3 to the correct value 0 in one iteration and (because the Hessian matrix here is diagonal and the cost is quadratic) requires one more iteration to converge.

paths of OD pairs for which it is the origin and executes the path flow iteration (5.83) for some fixed stepsize. This corresponds to an “all OD pairs at once” mode of implementation. The method can also be implemented in an asynchronous, distributed format, whereby the computation and information reception are not synchronized at each node. The validity of the method under these conditions is shown in [TsB86] and [BeT89].

We close this section with an example illustrating the projection algorithm:

Example 5.9

Consider the network shown in Fig. 5.72. There are only two OD pairs (1,5) and (2,5) with corresponding inputs $r_1 = 4$ and $r_2 = 8$. Consider the following two paths for each OD pair:

Paths of OD pair (1,5)

$$p_1(1) = \{1, 4, 5\}$$

$$p_2(1) = \{1, 3, 4, 5\}$$

Paths of OD pair (2,5)

$$p_1(2) = \{2, 4, 5\}$$

$$p_2(2) = \{2, 3, 4, 5\}$$

Consider the instance of the routing problem where the link cost functions are all identical and are given by

$$D_{ij}(F_{ij}) = \frac{1}{2}(F_{ij})^2, \quad \text{for all } (i, j)$$

Consider an initial path flow pattern whereby each OD pair input is routed through the middle link (3,4). This results in a flow distribution given in Tables 5.1 and 5.2.

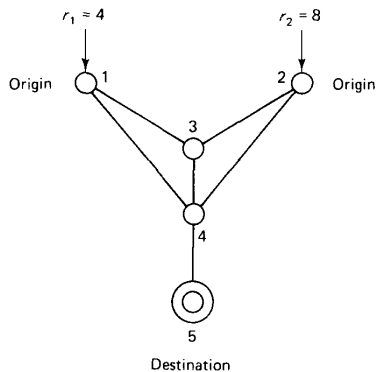


Figure 5.72 The network, for the example, where node 5 is the only destination and there are two origins, nodes 1 and 2.

TABLE 5.1 INITIAL PATH FLOWS FOR THE ROUTING EXAMPLE

OD pair	Path	Path flow
(1,5)	$p_1(1) = \{1, 4, 5\}$	0
	$p_2(1) = \{1, 3, 4, 5\}$	4
(2,5)	$p_1(2) = \{2, 4, 5\}$	0
	$p_2(2) = \{2, 3, 4, 5\}$	8

TABLE 5.2 INITIAL TOTAL LINK FLOWS FOR THE ROUTING EXAMPLE

Link	Total link flow
(1,3)	4
(1,4)	0
(2,3)	8
(2,4)	0
(3,4)	12
(4,5)	12
Others	0

The first derivative length of each link is given by

$$D'_{ij}(F_{ij}) = F_{ij}$$

so the total link flows given in Table 5.2 are also the link lengths for the current iteration. The corresponding first derivative lengths of paths are given in Table 5.3. Therefore, the shortest paths for the current iteration are $p_1(1)$ and $p_1(2)$ for OD pairs (1,5) and (2,5), respectively.

The form of the projection algorithm of Eqs. (5.83) to (5.85) is illustrated for the first OD pair. Here, for the nonshortest path $p = p_2(1)$ and the shortest path $\bar{p} = p_1(1)$ we have $d_p = 28$ and $d_{\bar{p}} = 12$. Also, $H_p = 3$ [each link has a second derivative length $D''_{ij} = 1$, and there are three links that belong to either $p_1(1)$ or $p_2(1)$, but not to both; cf. Eqs. (5.82) and (5.85)]. Therefore, the path flow iteration (5.83) takes the form

$$x_p := \max \left\{ 0, 4 - \frac{\alpha^k}{3}(28 - 12) \right\} = \max \left\{ 0, 4 - \frac{16\alpha^k}{3} \right\} = 4 - \min \left\{ 4, \frac{16\alpha^k}{3} \right\}$$

and

$$x_{\bar{p}} := r_1 - x_p$$

TABLE 5.3 INITIAL FIRST DERIVATIVE LENGTHS FOR THE ROUTING EXAMPLE

OD pair	Path	First derivative length
(1,5)	$p_1(1) = \{1, 4, 5\}$	12
	$p_2(1) = \{1, 3, 4, 5\}$	28
(2,5)	$p_1(2) = \{2, 4, 5\}$	12
	$p_2(2) = \{2, 3, 4, 5\}$	32

More generally, let $x_1(1), x_2(1), x_1(2)$, and $x_2(2)$ denote the flows along the paths $p_1(1), p_2(1), p_1(2)$, and $p_2(2)$, respectively, at the beginning of an iteration. The corresponding path lengths are

$$d_{p_1(1)} = x_1(1) + r_1 + r_2$$

$$d_{p_2(1)} = 2x_2(1) + x_2(2) + r_1 + r_2$$

$$d_{p_1(2)} = x_1(2) + r_1 + r_2$$

$$d_{p_2(2)} = 2x_2(2) + x_2(1) + r_1 + r_2$$

The second derivative length H_p of Eq. (5.85) equals 3. The projection algorithm (5.83) to (5.85) takes the following form. For $i = 1, 2$,

$$x_1(i) := \begin{cases} x_1(i) - \min \left[x_1(i), \frac{\alpha^k}{3} [d_{p_1(i)} - d_{p_2(i)}] \right], & \text{if } d_{p_1(i)} > d_{p_2(i)} \\ x_1(i) + \min \left[x_2(i), \frac{\alpha^k}{3} [d_{p_2(i)} - d_{p_1(i)}] \right], & \text{otherwise} \end{cases}$$

$$x_2(i) := r_i - x_1(i)$$

Notice that the presence of link (4,5) does not affect the form of the iteration, and indeed this should be so since the total flow of link (4,5) is always equal to $r_1 + r_2$ independent of the routing.

Table 5.4 gives sequences of successive cost function values obtained by the algorithm for different stepsizes and the “all OD pairs at once” and “one OD pair at a time” modes of implementation. The difference between these two implementation modes is that in the all-at-once mode, the OD pairs are processed simultaneously during an iteration using the link and path flows obtained at the end of the preceding iteration. In the one-at-a-time mode the OD pairs are processed sequentially, and following the iteration of one OD pair, the link flows are adjusted to reflect the results of the iteration before carrying out an iteration for the other OD pair. The stepsize is chosen to be constant at one of three possible values ($\alpha^k \equiv 0.5$, $\alpha^k \equiv 1$, and $\alpha^k \equiv 1.8$). It can be seen that for a unity stepsize, the convergence to a neighborhood of a solution is very fast in both the one-at-a-time and the all-at-once modes. As the stepsize is increased, the danger of divergence increases, with divergence typically occurring first for the all-at-once mode. This can be seen from the table, where for $\alpha^k \equiv 1.8$, the algorithm converges (slowly) in the one-at-a-time mode but diverges in the all-at-once mode.

TABLE 5.4 SEQUENCE OF COSTS GENERATED BY THE GRADIENT PROJECTION METHOD FOR THE ROUTING EXAMPLE AND FOR A VARIETY OF STEPSIZES

Iteration	All-at-once mode			One-at-a-time mode		
	$\alpha^k \equiv 0.5$	$\alpha^k \equiv 1.0$	$\alpha^k \equiv 1.8$	$\alpha^k \equiv 0.5$	$\alpha^k \equiv 1.0$	$\alpha^k \equiv 1.8$
0	184.00	184.00	184.00	184.00	184.00	184.00
1	110.88	104.00	112.00	114.44	101.33	112.00
2	102.39	101.33	118.72	103.54	101.00	109.15
3	101.28	101.03	112.00	101.63		101.79
4	101.09	101.00	118.72	101.29		101.56
5	101.03		112.00	101.08		101.33
6	101.01		118.72	101.03		101.18
7	101.00		112.00	101.01		101.12
8			118.72	101.00		101.09
9			112.00			101.06
10			118.72			101.03

5.8 ROUTING IN THE CODEX NETWORK

In this section the routing system of a network marketed by Codex, Inc., is discussed. References [HuS86] and [HSS86] describe the system in detail. The Codex network uses virtual circuits internally for user traffic and datagrams for system traffic (accounting, control, routing information, etc.). It was decided to use datagrams for system traffic because the alternative, namely establishing a virtual circuit for each pair of nodes, was deemed too expensive in terms of network resources. Note that for each user session, there are two separate virtual circuits carrying traffic in opposite directions, but not necessarily over the same set of links. Routing decisions are done separately in each direction.

There are two algorithms for route selection. The first, used for datagram routing of internal system traffic, is a simple shortest path algorithm of the type discussed in Section 5.2. The shortest path calculations are done as part of the second algorithm, which is used for selecting routes for new virtual circuits and for rerouting old ones. We will focus on the second algorithm, which is more important and far more sophisticated than the first.

The virtual circuit routing algorithm has much in common with the gradient projection method for optimal routing of the preceding section. There is a cost function D_{ij} for each link (i, j) that depends on the link flow, but also on additional parameters, such as the link capacity, the processing and propagation delay, and a priority factor for the virtual circuits currently on the link. Each node monitors the parameters of its adjacent links and broadcasts them periodically to all other nodes. If all virtual circuits have the same priority, the link cost function has the form

$$D_{ij}(F_{ij}) = \frac{F_{ij}}{C_{ij} - F_{ij}} + d_{ij}F_{ij}$$

where F_{ij} is the total data rate on the link, d_{ij} is the processing and propagation delay, and C_{ij} is the link capacity. This formula is based on the $M/M/1$ delay approximation; see the discussion of Sections 3.6 and 5.4. When there are more than one, say M , priority levels for virtual circuits, the link cost function has the form

$$D_{ij}(F_{ij}^1, \dots, F_{ij}^M) = \left(\sum_{k=1}^M p_k F_{ij}^k \right) \left(\frac{1}{C_{ij} - \sum_{k=1}^M F_{ij}^k} + d_{ij} \right)$$

where F_{ij}^k is the total link flow for virtual circuits of priority k , and p_k is a positive weighting factor. The form of this expression is motivated by the preceding formula, but otherwise has no meaningful interpretation based on a queueing analysis.

Routing of a new virtual circuit is done by assignment on a path that is shortest for the corresponding OD pair. This path is calculated at the destination node of the virtual circuit on the basis of the network topology, and the flow and other parameters last received for each network link. The choice of link length is motivated by the fact that the communication rate of a virtual circuit may not be negligible compared with the total flow of the links on its path. The length used is the link cost difference between when the virtual circuit is routed through the link and when it is not. Thus, if the new

virtual circuit has an estimated data rate Δ and priority class k , the length of link (i, j) is

$$D_{ij}(F_{ij}^1, \dots, F_{ij}^k + \Delta, \dots, F_{ij}^M) - D_{ij}(F_{ij}^1, \dots, F_{ij}^M) \quad (5.86)$$

As a result, routing of a new virtual circuit on a shortest path with respect to the link length (5.86) results in a minimum increase in total cost. When all virtual circuits have the same priority, the link length (5.86) becomes

$$D_{ij}(F_{ij} + \Delta) - D_{ij}(F_{ij}) \quad (5.87)$$

If Δ is very small, the link lengths (5.87) are nearly equal to $\Delta D'_{ij}(F_{ij})$. Otherwise (by the mean value theorem of calculus), they are proportional to the corresponding derivatives of D_{ij} at some intermediate point between the current flow and the flow resulting when the virtual circuit is routed through (i, j) . Thus, the link lengths (5.87), divided by Δ , are approximations to the first derivatives of link costs that are used in the gradient projection method. The link lengths (5.86) admit a similar interpretation (see Problem 5.27).

Rerouting of old virtual circuits is done in a similar manner. If there is a link failure, virtual circuits crossing the link are rerouted as if they are new. Rerouting with the intent of alleviating congestion is done by all nodes gradually once new link flow information is received. Each node scans the virtual circuits terminating at itself. It selects one of the virtual circuits as a candidate for rerouting and calculates the length of each link as the difference of link cost with and without that virtual circuit on the link. The virtual circuit is then rerouted on the shortest path if it does not already lie on the shortest path. An important parameter here is the number of virtual circuits chosen for rerouting between successive link flow broadcasts. This corresponds to the stepsize parameter in the gradient projection method, which determines how much flow is shifted on a shortest path at each iteration. The problem, of course, is that too many virtual circuits may be rerouted simultaneously by several nodes acting without coordination, thereby resulting in oscillatory behavior similar to that discussed in Section 5.2.5. The Codex network uses a heuristic rule whereby only a fraction of the existing virtual circuits are (pseudorandomly) considered for rerouting between successive link flow broadcasts.

SUMMARY

Routing is a sophisticated data network function that requires coordination between the network nodes through distributed protocols. It affects the average packet delay and the network throughput.

Our main focus was on methods for route selection. The most common approach, shortest path routing, can be implemented in a variety of ways, as exemplified by the ARPANET and TYMNET algorithms. Depending on its implementation, shortest path routing may cause low throughput, poor response to traffic congestion, and oscillatory behavior. These drawbacks are more evident in datagram than in virtual circuit networks. A more sophisticated alternative is optimal routing based on flow models. Several algorithms were given for computation of an optimal routing, both centralized and distributed.

As the statistics of the input arrival processes change more rapidly, the appropriateness of the type of optimal routing we focused on diminishes. In such cases it is difficult to recommend routing methods that are simultaneously efficient and practical.

Another interesting aspect of the routing problem relates to the dissemination of routing-related information over failure-prone links. We described several alternative algorithms based on flooding ideas.

Finally, routing must be taken into account when designing a network's topology, since the routing method determines how effectively the link capacities are utilized. We described exact and heuristic methods for addressing the difficulties of topological design.

NOTES, SOURCES, AND SUGGESTED READING

Section 5.1. Surveys of routing, including descriptions of some practical routing algorithms, can be found in [Eph86] and [ScS80]. Routing in the ARPANET is described in [McW77], [MRR78], and [MRR80]. The TYMNET system is described in [Tym81]. Routing in SNA is described in [Ahu79] and [Atk80]. A route selection method for SNA networks is described in [GaH83]. Descriptions of other routing systems may be found in [Wec80] and [SpM81]. Topology broadcast algorithms are discussed in [HuS88]. Extensive discussions of interconnected local area networks can be found in a special issue of *IEEE Network* [IEE88].

Section 5.2. There are many sources for the material on graphs, spanning trees, and shortest paths (e.g., [Ber91] and [PaS82]). Particularly fast shortest path algorithms are described in [Pap74], [DGK79], and [GaP88].

An analysis of some asynchronous min-hop algorithms was first given in [Taj77]. The material on asynchronous shortest path algorithms given here is taken from [Ber82a]. For an extensive treatment of asynchronous distributed algorithms, see [BeT89].

For more on stability issues of shortest path routing algorithms, see [MRR78], [Ber79b], [Ber82b], and [GaB83].

Section 5.3. The difficulties with the ARPANET flooding algorithm and some remedies are described in [Ros81] and [Per83]. The SPTA appeared in the thesis [Spi85]; see also [SpG89]. For related work, see [EpB81], [Fin79], [GaB81], [Seg83], and [SoH86].

Section 5.4. There is considerable literature on routing based on queue state information. These works consider data networks but also relate to routing and control problems in other contexts. See [EVW80], [FoS78], [HaO84], [MoS82], [Ros86], [SaH87], [SaO82], [Sar82], [StK85], and [Yum81]. A survey of the application of control and optimization methods in communication network problems is given in [EpV89].

Surveys on topological design of data networks are provided in [BoF77], [GeK77], [KeB83], [McS77], and [MoS86]. The special issue [IEE89] describes several recent works on the subject. For further material on network reliability, see [BaP83], [Bal79], [Ben86], [LaL86], [LiS84], [PrB84], and the references quoted therein. For some al-

ternative approaches to the problem of capacity assignment and routing, see [Gav85], [NgH87], and [LeY89].

Section 5.5. The optimal routing problem considered is a special case of a multicommodity network flow problem. The problem arises also in the context of transportation networks in a form that is very similar to the one discussed here; see [AaM81], [Daf71], [Daf80], [DeK81], [FIN74], and [LaH84].

Section 5.6. The Frank–Wolfe method [FrW56] has been proposed in the context of the routing problem in [FGK73]. Problems 5.31 and 5.32 give a self-contained presentation of the convergence properties of this method as well as those of the gradient projection method.

Section 5.7. The gradient projection method was applied to multicommodity flow problems based on a path flow formulation in [Ber80]. Extensions were described in [BeG82], [BeG83], and [GaB84a]. A public domain FORTRAN implementation of the gradient projection method is given in [BGT84]. Distributed asynchronous versions of the algorithm are analyzed in [TsB86] and [Tsa89]. The optimal routing algorithm of [Gal77], which adjusts link flow fractions (see the end of Section 5.5), bears a relationship to the gradient projection method (see [BGG84]). Extensions of the algorithm are given in [Ber79a], [Gaf79], and [BGG84]. Computational and simulation results relating to this algorithm and its extensions may be found in [BGV79] and [ThC86].

Section 5.8. The comments of P. Humblet, one of the principal designers of the Codex network, were very helpful in preparing this section. Additional material can be found in [HuS86] and [HSS86].

PROBLEMS

- 5.1 Find a minimum weight spanning tree of the graph in Fig. 5.73 using the Prim–Dijkstra and the Kruskal algorithms.
- 5.2 Find the shortest path tree from every node to node 1 for the graph of Fig. 5.74 using the Bellman–Ford and Dijkstra algorithms.
- 5.3 The number shown next to each link of the network in Fig. 5.75 is the probability of the link failing during the lifetime of a virtual circuit from node A to node B . It is assumed that links fail independently of each other. Find the most reliable path from A to B , that

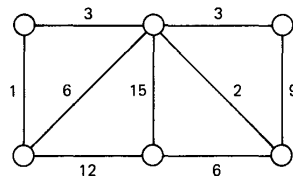


Figure 5.73 Graph for Problem 5.1.

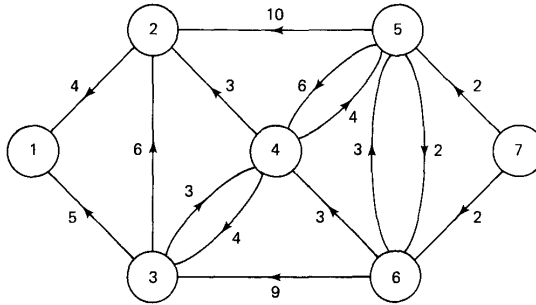


Figure 5.74 Graph for Problem 5.2.

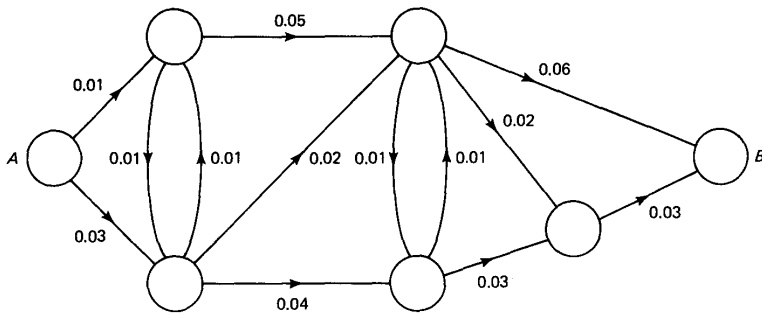


Figure 5.75 Graph for Problem 5.3.

is, the path for which the probability that all its links stay intact during the virtual circuit's lifetime is maximal. What is this probability?

- 5.4 A shortest path spanning tree (Section 5.2.3) differs from a minimum weight spanning tree in that the arc lengths refer to directed arcs, while in the minimum weight spanning tree problem, the arc weights refer to undirected arcs or, equivalently, the arc weights are assumed equal in both directions. However, even if all arcs have equal length in both directions, a minimum weight spanning tree (for arc weights equal to the corresponding lengths) need not be a shortest path spanning tree. To see this, consider a network with three nodes A , B , and C , and three bidirectional arcs AB , BC , and CA . Choose a weight for each arc so that the minimum weight spanning tree is not a shortest path tree with root node C . (The length of an arc in each of its two directions is equal to the arc weight.)
- 5.5 Consider Example 5.4.
- Repeat the example with $N = 6$ instead of $N = 16$. (Nodes 1, 2, 4, and 5 send 1 unit to node 6, while node 3 sends ϵ with $0 < \epsilon \ll 1$.)
 - Repeat part (a) with the difference that the length d_{ij} of link (i, j) is $\alpha + F_{ij}$ for $\alpha = 1$ (instead of F_{ij}). Consider all possible choices of initial routing.

- (c) What is the minimum value of α for which the shortest paths of all nodes except node 3 eventually stay constant regardless of the choice of initial routing?
 - (d) Repeat part (a) with the difference that at each iteration after the first, the length of each link is the average of the link arrival rates corresponding to the current and the preceding routings (instead of the arrival rates at just the current routing).
- 5.6 Consider the shortest path problem assuming that all cycles not containing node 1 have nonnegative length. Let $\tilde{D}_j$ be the shortest path distances corresponding to a set of link lengths $\tilde{d}_{ij}$, and let d_{ij} be a new set of link lengths. For $k = 1, 2, \dots$, define

$$N_1 = \{i \neq 1 \mid \text{for some arc } (i, j) \text{ with } d_{ij} > \tilde{d}_{ij} \text{ we have } \tilde{D}_i = \tilde{d}_{ij} + \tilde{D}_j\}$$

$$N_{k+1} = \{i \neq 1 \mid \text{for some arc } (i, j) \text{ with } j \in N_k \text{ we have } \tilde{D}_i = \tilde{d}_{ij} + \tilde{D}_j\}$$

- (a) Show that the Bellman–Ford algorithm starting from the initial conditions $D_1^0 = 0$, $D_i^0 = \tilde{D}_i$ for $i \notin \cup_k N_k \cup \{1\}$, and $D_i^0 = \infty$ for $i \in \cup_k N_k$ terminates after at most N iterations.
 - (b) Based on part (a), suggest a heuristic method for alleviating the “bad news phenomenon” of Fig. 5.36, which arises in the asynchronous Bellman–Ford algorithm when some link lengths increase. *Hint:* Whenever the length of a link on the current shortest path increases, the head node of the link should propagate an estimated distance of ∞ along the shortest path tree in the direction away from the destination.
- 5.7 Consider the Bellman–Ford algorithm assuming that all cycles not containing node 1 have nonnegative length. For each node $i \neq 1$, let (i, j_i) be an arc such that j_i attains the minimum in the equation

$$D_i^{h_i} = \min_j [d_{ij} + D_j^{h_i-1}]$$

where h_i is the largest h such that $D_i^h \neq D_i^{h-1}$. Consider the subgraph consisting of the arcs (i, j_i) , for $i \neq 1$, and show that it is a spanning tree consisting of shortest paths. *Hint:* Show that $h_i > h_{j_i}$.

- 5.8 Consider the shortest path problem. Show that if there is a cycle of zero length not containing node 1, but no cycles of negative length not containing node 1, Bellman’s equation has more than one solution.
- 5.9 Suppose that we have a directed graph with no directed cycles. We are given a length d_{ij} for each directed arc (i, j) and we want to compute a shortest path to node 1 from all other nodes, assuming there exists at least one such path. Show that nodes $2, 3, \dots, N$ can be renumbered so that there is an arc from i to j only if $i > j$. Show that once the nodes are renumbered, Bellman’s equation can be solved with $O(N^2)$ operations at worst.
- 5.10 Consider the shortest path problem from every node to node 1 and Dijkstra’s algorithm.
- (a) Assume that a positive lower bound is known for all arc lengths. Show how to modify the algorithm to allow more than one node to enter the set of permanently labeled nodes at each iteration.
 - (b) Suppose that we have already calculated the shortest path from every node to node 1, and assume that a single arc length *increases*. Modify Dijkstra’s algorithm to recalculate as efficiently as you can the shortest paths.
- 5.11 *A Generic Single-Destination Multiple-Origin Shortest Path Algorithm.* Consider the shortest path problem from all nodes to node 1. The following algorithm maintains a list of nodes V and a vector $D = (D_1, D_2, \dots, D_N)$, where each D_j is either a real number or ∞ .

Initially,

$$V = \{1\}$$

$$D_1 = 0, \quad D_i = \infty, \quad \text{for all } i \neq 1$$

The algorithm proceeds in iterations and terminates when V is empty. The typical iteration (assuming that V is nonempty) is as follows:

Remove a node j from V . For each incoming arc $(i, j) \in \mathcal{A}$, with $i \neq 1$, if $D_i > d_{ij} + D_j$, set

$$D_i = d_{ij} + D_j$$

and add i to V if it does not already belong to V .

- (a) Show that at the end of each iteration: (1) if $D_j < \infty$, then D_j is the length of some path starting at j and ending at 1, and (2) if $j \notin V$, then either $D_j = \infty$ or else

$$D_i \leq d_{ij} + D_j, \quad \text{for all } i \text{ such that } (i, j) \in \mathcal{A}$$

- (b) If the algorithm terminates, then upon termination, for all $i \neq 1$ such that $D_i < \infty$, D_i is the shortest distance from i to 1 and

$$D_i = \min_{(i,j) \in \mathcal{A}} [d_{ij} + D_j]$$

Furthermore, $D_i = \infty$ for all i such that there is no path from i to 1.

- (c) If the algorithm does not terminate, there exist paths of arbitrarily small length from at least one node i to node 1.
 (d) Show that if at each iteration the node j removed from V satisfies $D_j = \min_{i \in V} D_i$, the algorithm is equivalent to Dijkstra's algorithm.

5.12 *A Generic Single-Destination Single-Origin Shortest Path Algorithm.* Consider the problem of finding a shortest path from a given node t to node 1. Suppose that for all i we have an underestimate u_i of the shortest distance from t to i . The following algorithm maintains a list of nodes V and a vector $D = (D_1, D_2, \dots, D_N)$, where each D_i is either a real number or ∞ . Initially,

$$V = \{1\}$$

$$D_1 = 0, \quad D_i = \infty, \quad \text{for all } i \neq 1$$

The algorithm proceeds in iterations and terminates when V is empty. The typical iteration (assuming that V is nonempty) is as follows:

Remove a node j from V . For each incoming arc $(i, j) \in \mathcal{A}$, with $i \neq 1$, if $\min\{D_i, D_t - u_i\} > d_{ij} + D_j$, set

$$D_i = d_{ij} + D_j$$

and add i to V if it does not already belong to V .

- (a) If the algorithm terminates, then upon termination, either $D_t < \infty$, in which case D_t is the shortest distance from t to 1, or else there is no path from t to 1.
 (b) If the algorithm does not terminate, there exist paths of arbitrarily small length from at least one node i to node 1.

5.13 Consider the second flooding algorithm of Section 5.3.2. Suppose that there is a known upper bound on the time an update message requires to reach every node connected with the originating node. Devise a scheme based on an age field to supplement the algorithm so that it works correctly even if the sequence numbers can wrap around due to a memory or

communication error. *Note:* Packets carrying an age field should be used only in exceptional circumstances after the originating node detects an error.

- 5.14** Modify the SPTA so that it can be used to broadcast one-directional link information other than status throughout the network (cf. the remarks at the end of Section 5.3.3). The link information is collected by the start node of the link. Justify the modified algorithm. *Hint:* Add to the existing algorithm additional main and port tables to hold the directional link information. Formulate update rules for these tables using the node labels provided by the main topology update algorithm of Section 5.3.3.

- 5.15 (a)** Give an example where the following algorithm for broadcasting topological update information fails. Initially, all nodes know the correct status of all links.

Update rules:

1. When an adjacent link changes status, the node sends the new status in a message on all operating adjacent links.
2. When a node receives a message about a nonadjacent link which differs from its view of the status of the link, it changes the status of the link in its topology table. It also sends the new status in a message on all operating adjacent links *except* the one on which it received the message.
3. When a node receives a message about an adjacent link which differs from its view of the status of the link, it sends the correct status on the link on which it received the message.

Hint: Failure occurs with simple network topologies, such as the one shown in Fig. 5.43.

- (b) Give an example of failure when the word “except” in rule 2 is changed to “including.”

- 5.16** Describe what, if anything, can go wrong in the topology broadcast algorithms of Section 5.3 if the assumption that messages are received across links in the order transmitted is relaxed. Consider the ARPANET flooding algorithm, the two algorithms of Section 5.3.2, and the SPTA of Section 5.3.3.

- 5.17** *Distributed Computation of the Number of Nodes in a Network.* Consider a strongly connected communication network with N nodes and A (bidirectional) links. Each node knows its identity and the set of its immediate neighbors but not the network topology. Node 1 wishes to determine the number of nodes in the network. As a first step, it initiates an algorithm for finding a *directed, rooted spanning tree with node 1 as the root*. By this we mean a tree each link (i, k) of which is directed and oriented toward node 1 along the unique path on the tree leading from i to 1 (see the example tree shown in Fig. 5.76).

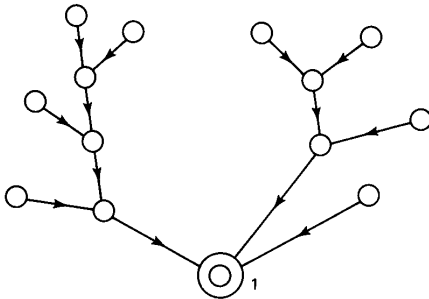


Figure 5.76 Example tree for Problem 5.17.

- (a) Devise a distributed algorithm involving exchange of messages between nodes that constructs such a tree. The algorithm is initiated by node 1 and should involve no more than $O(A)$ message transmissions. Communication along any link is assumed error-free. At the end of the algorithm, the end nodes of each link should know whether the link is part of the tree, and, if so, they should know its direction.
- (b) Supplement the algorithm derived in part (a) by another algorithm involving no more than $O(N)$ message transmissions by means of which node 1 gets to know N .
- (c) Assuming that each message transmission takes an equal amount of time T , derive an upper bound for the time needed to complete the algorithms in parts (a) and (b).
- 5.18** Consider the minimum weight spanning tree problem. Show that the following procedure implements the Prim–Dijkstra algorithm and that it requires $O(N^2)$ arithmetic operations. Let node 1 be the starting node, and initially set $P = \{1\}$, $T = \emptyset$, $D_1 = 0$, and $D_j = w_{1j}$, $a_j = 1$ for $j \neq 1$.

Step 1. Find $i \notin P$ such that

$$D_i = \min_{j \notin P} D_j$$

and set $T := T \cup \{(j, a_i)\}$, $P := P \cup \{i\}$. If P contains all nodes, stop.

Step 2. For all $j \notin P$, if $w_{ij} < D_j$, set $D_j := w_{ij}$, $a_j := i$. Go to step 1.

Note: Observe the similarity with Dijkstra's algorithm for shortest paths.

- 5.19** Use Kleitman's algorithm to prove or disprove that the network shown in Fig. 5.77 is 3-connected. What is the maximum k for which the network is k -connected?
- 5.20** Suppose you had an algorithm that would find the maximum k for which two nodes in a graph are k -connected (e.g., the max-flow algorithm test illustrated in Fig. 5.57). How would you modify Kleitman's algorithm to find the maximum k for which the graph is k -connected? Apply your modified algorithm to the graph of Fig. 5.77.
- 5.21** Use the Essau–Williams heuristic algorithm to find a constrained MST for the network shown in Fig. 5.78. Node 0 is the central node. The link weights are shown next to the links. The input flows from each node to the concentrator are shown next to the arrows. All link capacities are 10.
- 5.22** *MST Algorithm [MaP88].* We are given an undirected graph $G = (N, A)$ with a set of nodes $N = \{1, 2, \dots, n\}$, and a weight a_{ij} for each arc (i, j) ($a_{ij} = a_{ji}$). We refer to the maximum over all the weights of the arcs contained in a given walk as the *critical weight* of the walk.
- (a) Show that an arc (i, j) belongs to a minimum weight spanning tree (MST) if and only if the critical weight of every walk starting at i and ending at j is greater or equal to a_{ij} .

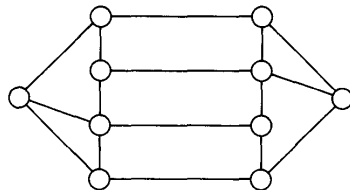


Figure 5.77 Network for Problem 5.19.

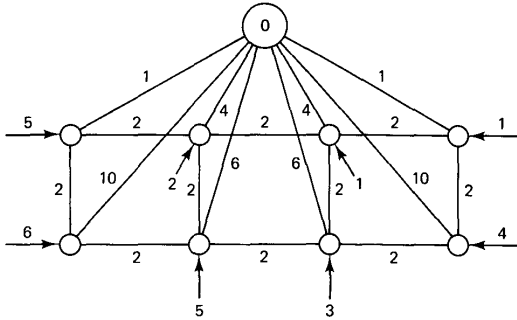


Figure 5.78 Network for Problem 5.21.

(b) Consider the following iterative algorithm:

$$x_{ij}^0 = \begin{cases} a_{ij}, & \text{if } (i, j) \in A, \\ \infty, & \text{otherwise,} \end{cases}$$

$$x_{ij}^{k+1} = \begin{cases} \min \left\{ x_{ij}^k, \max \{ x_{i(k+1)}^k, x_{(k+1)j}^k \} \right\}, & \text{if } j \neq i, \\ \infty, & \text{otherwise.} \end{cases}$$

Show that x_{ij}^k is the minimum over all critical weights of walks that start at i , end at j , and use only nodes 1 to k as intermediate nodes.

5.23 Suboptimal Selective Broadcasting. We are given a connected undirected graph $G = (N, A)$ with a nonnegative weight a_{ij} for each arc $(i, j) \in A$. Consider a tree in G which spans a given subset of nodes $\tilde{N}$ and has minimum weight over all such trees. Let W^* be the weight of this tree. Consider the graph $I(G)$, which has node set $\tilde{N}$ and is complete (has an arc connecting every pair of its nodes). Let the weight for each arc (i, j) of $I(G)$ be equal to the shortest distance in the graph G from the node $i \in \tilde{N}$ to the node $j \in \tilde{N}$. [Here, G is viewed as a directed graph with bidirectional arcs and length a_{ij} for each arc $(i, j) \in A$ in each direction.] Let T be a minimum weight spanning tree of $I(G)$.

(a) Show that the weight of T is no greater than $2W^*$. *Hint:* A *tour* in a graph is a cycle with no repeated nodes that passes through all the nodes. Consider a minimum weight tour in $I(G)$. Show that the weight of this tour is no less than the weight of T and no more than $2W^*$.

(b) Provide an example where the weight of T lies strictly between W^* and $2W^*$.

(c) Develop an algorithm for selective broadcasting from one node of $\tilde{N}$ to all other nodes of $\tilde{N}$ using T . Show that this algorithm comes within a factor of 2 of being optimal based on the given weights of the arcs of G .

5.24 Show that the necessary optimality condition of Section 5.5,

$$x_p^* > 0 \quad \Rightarrow \quad \frac{\partial D(x^*)}{\partial x_{p'}} \geq \frac{\partial D(x^*)}{\partial x_p}, \quad \text{for all } p' \in P_w$$

implies that for all feasible path flow vectors $x = \{x_p\}$, we have

$$\sum_{p \in P_w} (x_p - x_p^*) \frac{\partial D(x^*)}{\partial x_p} \geq 0, \quad \text{for all OD pairs } w$$

[The latter condition is sufficient for optimality of x^* when $D(x)$ is a convex function, that is,

$$D(\alpha x + (1 - \alpha)x') \leq \alpha D(x) + (1 - \alpha)D(x')$$

for all feasible x and x' and all α with $0 \leq \alpha \leq 1$. To see this, note that if $D(x)$ is convex, we have

$$D(x) \geq D(x^*) + \sum_{w \in W} \sum_{p \in P_w} (x_p - x_p^*) \frac{\partial D(x^*)}{\partial x_p}$$

for all x ([Lue84], p. 178), so when the shortest path condition above holds, x^* is guaranteed to be optimal.] Hint: Let $D_w^* = \min_{p \in P_w} \partial D(x^*) / \partial x_p$. We have for every feasible x ,

$$0 = \sum_{p \in P_w} (x_p - x_p^*) D_w^* \leq \sum_{\{p \in P_w | x_p > x_p^*\}} (x_p - x_p^*) \frac{\partial D(x^*)}{\partial x_p} + \sum_{\{p \in P_w | x_p < x_p^*\}} (x_p - x_p^*) D_w^*$$

- 5.25** Consider the network shown in Fig. 5.79 involving one OD pair with a given input rate r and three links with capacities C_1 , C_2 , and C_3 , respectively. Assume that $C_1 = C_3 = C$, and $C_2 > C$, and solve the optimal routing problem with cost function based on the $M/M/1$ delay approximation and r between 0 and $2C + C_2$.
- 5.26** Consider the problem of finding the optimal routing of one unit of input flow in a single-OD-pair, three-link network for the cost function

$$D(x) = \frac{1}{2} \left[(x_1^2) + 2(x_2^2) + (x_3^2) \right] + 0.7x_3$$

Mathematically, the problem is

$$\text{minimize } D(x)$$

$$\text{subject to } x_1 + x_2 + x_3 = 1$$

$$x_1, x_2, x_3 \geq 0$$

- (a) Show that $x_1^* = 2/3$, $x_2^* = 1/3$, and $x_3^* = 0$ is the optimal solution.
- (b) Carry out several iterations of the Frank-Wolfe method and the projection method starting from the point $(1/3, 1/3, 1/3)$. Do enough iterations to demonstrate a clear trend in rate of convergence. (Use a computer for this if you wish.) Plot the successive iterates on the simplex of feasible flows.
- 5.27** *Extensions of the Gradient Projection Method*
- (a) Show how the optimality condition of Section 5.5 and the gradient projection method can be extended to handle the situation where the cost function is a general twice differentiable function of the path flow vector x .

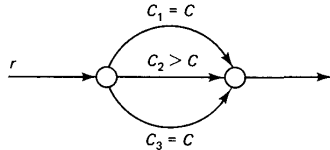


Figure 5.79 Network for Problem 5.25.

- (b) Discuss the special case where the cost of link (i, j) is $D_{ij}(\bar{F}_{ij}, F_{ij})$ where

$$\bar{F}_{ij} = \sum_{k=1}^M p_k F_{ij}^k, \quad F_{ij} = \sum_{k=1}^M F_{ij}^k$$

p_k is a positive scalar weighting factor for priority class k , and F_{ij}^k is the total flow of priority class k crossing link (i, j) (cf. the Codex algorithm cost function).

- 5.28 Optimal Broadcast Routing along Spanning Trees.** Consider the optimal routing problem of Section 5.4. Suppose that in addition to the regular OD pair input rates r_w that are to be routed along directed paths, there is additional traffic R to be broadcast from a single node, say 1, to all other nodes along one or more directed spanning trees rooted at node 1. The spanning trees to be used for routing are to be selected from a given collection T , and the portion of R to be broadcast along each is subject to optimization.

- (a) Characterize the path flow rates and the portions of R broadcast along spanning trees in T that minimize a total link cost, such as the one given by Eq. (5.29) in Section 5.4.
 (b) Extend the gradient projection method to solve this problem.
 (c) Extend your analysis to the case where there are several root nodes and there is traffic to be broadcast from each of these nodes to all other nodes.

Hint: Consider the first derivative total weights of spanning trees in addition to the first derivative lengths of paths.

- 5.29 Optimal Routing with VCs Using the Same Links in Both Directions.** Consider the optimal routing problem of Section 5.4 with the additional restriction that the traffic originating at node i and destined for node i' must use the same path (in the reverse direction) as the traffic originating at i' and destined for i . In particular, assume that if w and w' are the OD pairs (i, i') and (i', i) , respectively, there is a proportionality constant $c_{ww'}$ such that if $p \in P_w$ is a path and $p' \in P_{w'}$ is the reverse path, and x_p and $x_{p'}$ are the corresponding path flows, then $x_p = c_{ww'} x_{p'}$. Derive the conditions characterizing an optimal routing, and appropriately extend the gradient projection method.

- 5.30 Use of the $M/M/1$ -Based Cost Function with the Gradient Projection Method.** When the cost function

$$\sum_{(i,j)} D_{ij}(F_{ij}) = \sum_{(i,j)} \frac{F_{ij}}{C_{ij} - F_{ij}}$$

is minimized using the gradient projection method, there is potential difficulty due to the fact that some link flow F_{ij} may exceed the capacity C_{ij} . One way to get around this is to replace D_{ij} with a function $\tilde{D}_{ij}$ that is identical with D_{ij} for F_{ij} in the interval $[0, \rho C_{ij}]$, where ρ is some number less than one (say, $\rho = 0.99$), and is quadratic for $F_{ij} > \rho C_{ij}$. The quadratic function is chosen so that D_{ij} and $\tilde{D}_{ij}$ have equal values and first two derivatives at the point $F_{ij} = \rho C_{ij}$. Derive the formula for $\tilde{D}_{ij}$. Show that if the link flows F_{ij}^* that minimize

$$\sum_{(i,j)} D_{ij}(F_{ij})$$

result in link utilizations not exceeding ρ [i.e., $F_{ij}^* \leq \rho C_{ij}$ for all (i, j)], then F_{ij}^* also minimize

$$\sum_{(i,j)} \tilde{D}_{ij}(F_{ij})$$

- 5.31** *Convergence of the Frank–Wolfe Algorithm.* The purpose of this problem and the next one is to guide the advanced reader through convergence proofs of the Frank–Wolfe and the gradient projection methods. Consider the problem

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } x \in X \end{aligned}$$

where f is a differentiable function of the n -dimensional vector x and X is a closed, bounded, convex set. Assume that the gradient of f satisfies

$$|\nabla f(x) - \nabla f(y)| \leq L|x - y|, \quad \text{for all } x, y \in X$$

where L is a positive constant and $|z|$ denotes the Euclidean norm of a vector z , that is,

$$|z| = \sqrt{\sum_{i=1}^n (z_i)^2}$$

(This assumption can be shown to hold if $\nabla^2 f$ exists and is continuous over X .)

- (a) Show that for all vectors x , Δx , and scalars α , such that $x \in X$, $x + \Delta x \in X$, $\alpha \in [0, 1]$,

$$f(x + \alpha \Delta x) \leq f(x) + \alpha \nabla f(x)^T \Delta x + \frac{\alpha^2 L}{2} |\Delta x|^2$$

where the superscript T denotes transpose. *Hint:* Use the Taylor formula

$$f(x + y) = f(x) + \nabla f(x)^T y + \int_0^1 [\nabla f(x + ty) - \nabla f(x)]^T y \, dt$$

- (b) Use part (a) to show that if Δx is a descent direction at x [i.e., $\nabla f(x)^T \Delta x < 0$], then

$$\min_{\alpha \in [0, 1]} f(x + \alpha \Delta x) \leq f(x) + \delta$$

where

$$\delta = \begin{cases} \frac{1}{2} \nabla f(x)^T \Delta x, & \text{if } \nabla f(x)^T \Delta x + L|\Delta x|^2 < 0 \\ -\frac{|\nabla f(x)^T \Delta x|^2}{2LR^2}, & \text{otherwise} \end{cases}$$

where R is the diameter of X , that is,

$$R = \max_{x, y \in X} |x - y|$$

Hint: Minimize over $\alpha \in [0, 1]$ in both sides of the inequality of part (a).

- (c) Consider the Frank–Wolfe method

$$x^{k+1} = x^k + \alpha^k \Delta x^k$$

where $x^0 \in X$ is the starting vector, Δx^k solves the problem

$$\begin{aligned} &\text{minimize } \nabla f(x^k)^T \Delta x \\ &\text{subject to } x^k + \Delta x \in X \end{aligned}$$

and α^k is a minimizing stepsize

$$f(x^k + \alpha^k \Delta x^k) = \min_{\alpha \in [0,1]} f(x^k + \alpha \Delta x^k)$$

Show that every limit x^* of the sequence $\{x^k\}$ satisfies the optimality condition

$$\nabla f(x^*)^T (x - x^*) \geq 0, \quad \text{for all } x \in X$$

Hint: Argue that if $\{x^k\}$ has a limit point and δ^k corresponds to x^k as in part (b), then $\delta^k \rightarrow 0$, and therefore also $\nabla f(x^k)^T \Delta x^k \rightarrow 0$. Taking the limit in the relation $\nabla f(x^k)^T \Delta x \leq \nabla f(x^k)^T (x - x^k)$ for all $x \in X$, argue that a limit point x^* must satisfy $0 \leq \nabla f(x^*)^T (x - x^*)$ for all $x \in X$.

5.32 *Convergence of the Gradient Projection Method.* Consider the minimization problem and the assumptions of Problem 5.31. Consider also the iteration

$$x^{k+1} = x^k + \alpha^k (\bar{x}^k - x^k)$$

where $\bar{x}^k$ is the projection of $x^k - s \nabla f(x^k)$ on X and s is some fixed positive scalar, that is, $\bar{x}^k$ solves

$$\begin{aligned} &\text{minimize } |x - x^k + s \nabla f(x^k)|^2 \\ &\text{subject to } x \in X \end{aligned}$$

(a) Assume that α^k is a minimizing stepsize

$$f[x^k + \alpha^k (\bar{x}^k - x^k)] = \min_{\alpha \in [0,1]} f[x^k + \alpha (\bar{x}^k - x^k)]$$

and show that every limit point x^* of $\{x^k\}$ satisfies the optimality condition

$$\nabla f(x^*)^T (x - x^*) \geq 0, \quad \text{for all } x \in X$$

Hint: Follow the hints of Problem 5.31. Use the following necessary condition satisfied by the projection

$$[x^k - s \nabla f(x^k) - \bar{x}^k]^T (x - \bar{x}^k) \leq 0, \quad \text{for all } x \in X$$

to show that $s \nabla f(x^k)^T (\bar{x}^k - x^k) \leq -|\bar{x}^k - x^k|^2$.

(b) Assume that $\alpha^k = 1$ for all k . Show that if $s < 2/L$, the conclusion of part (a) holds.

5.33 *Gradient Projection Method with a Diagonal Scaling Matrix.* Consider the problem

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } x \geq 0 \end{aligned}$$

of Section 5.7. Show that the iteration

$$x_i^{k+1} = \max \left\{ 0, x_i^k - \alpha^k b_i \frac{\partial f(x^k)}{\partial x_i} \right\}, \quad i = 1, \dots, n$$

with b_i some positive scalars, is equivalent to the gradient projection iteration

$$y_i^{k+1} = \max \left\{ 0, y_i^k - \alpha^k \frac{\partial h(y^k)}{\partial y_i} \right\}, \quad i = 1, \dots, n$$

where the variables x_i and y_i are related by $x_i = \sqrt{b_i}y_i$, and $h(y) = f(Ty)$, and T is the diagonal matrix with the i^{th} diagonal element equal to $\sqrt{b_i}$.

- 5.34** This problem illustrates the relation between optimal routing and adaptive shortest path routing for virtual circuits; see also Section 5.2.5 and [GaB83]. Consider a virtual circuit (VC) routing problem involving a single origin–destination pair and two links as shown in the figure. Assume that a VC arrives at each of the times $0, \tau, 2\tau, \dots$, and departs exactly H time units later. Each VC is assigned to one of the two links and remains assigned on that link for its entire duration. Let $N_i(t)$, $i = 1, 2$, be the number of VCs on link i at time t . We assume that initially the system is empty, and that an arrival and/or departure at time t is not counted in $N_i(t)$, so, for example, $N_i(0) = 0$. The routing algorithm is of the adaptive shortest path type and works as follows: For $kT \leq t < (k+1)T$, a VC that arrives at time t is routed on link 1 if $\gamma_1 N_1(kT) \leq \gamma_2 N_2(kT)$ and is routed on link 2 otherwise. Here T , γ_1 , and γ_2 are given positive scalars.

Assume that $T \leq H$ and that τ divides evenly H . Show that:

- (a) $N_1(t) + N_2(t) = H/\tau$ for all $t > H$.
 (b) For all $t > H$ we have

$$\frac{|N_1(t) - N_1^*|}{N_1^*} \leq \frac{\gamma_1 + \gamma_2}{\gamma_2} \frac{T}{H}, \quad \frac{|N_2(t) - N_2^*|}{N_2^*} \leq \frac{\gamma_1 + \gamma_2}{\gamma_1} \frac{T}{H}$$

where (N_1^*, N_2^*) solve the optimal routing problem

$$\text{minimize } \gamma_1 N_1^2 + \gamma_2 N_2^2$$

$$\text{subject to } N_1 + N_2 = H/\tau, \quad N_1 \geq 0, \quad N_2 \geq 0$$

Hint: Show that for $t > H$ we have

$$\gamma_1 N_1(t) \leq \gamma_2 N_2(t) \iff N_1(t) \leq N_1^*$$

- 5.35** *Virtual Circuit Routing by Flooding.* In some networks where it may be hard to keep track of the topology as it changes (e.g., packet radio networks), it is possible to use flooding to set up virtual circuits. The main idea is that a node m , which wants to set up a virtual circuit to node n , should flood an “exploratory” packet through the network. A node that rebroadcasts this packet stamps its ID number on it so that when the destination node n receives an exploratory packet, it knows the route along which it came. The destination node can then choose one of the potentially many routes carried by the copies of the exploratory packet received and proceed to set up the virtual circuit. Describe one or more flooding protocols that will make this process workable. Address the issues of indefinite or excessive message circulation in the network, appropriate numbering of exploratory packets, unpredictable link failures and delays, potential confusion between the two end nodes of virtual circuits, and so on.
- 5.36** Consider the optimal routing problem of Section 5.5. Assume that each link cost is chosen to be the same function $D(F)$ of the link flow F , where the first link cost derivative at zero flow $D'(0)$ is positive.
- (a) For sufficiently small values of origin–destination pair input rates r_w show that optimal routings use only minimum-hop paths from origins to destinations.
- (b) Construct an example showing that optimal routings do not necessarily have the minimum-hop property for larger values of r_w .
- 5.37** Consider the Frank–Wolfe method with the stepsize of Eq. (5.63) in Section 5.6. Describe an implementation as a distributed synchronous algorithm involving communication from

origins to links and the reverse. At each iteration of this algorithm, each origin should calculate the stepsize of Eq. (5.63) and update accordingly the path flows that originate at itself.

- 5.38 Optimal Dynamic Routing Based on Window Ratio Strategies.** Consider the optimal routing problem of Section 5.4 in a datagram network. Suppose that we want to implement a set of path flows $\{x_p\}$ calculated on the basis of a nominal set of traffic inputs $\{r_w\}$. The usual solution, discussed in Section 5.5, is to route the traffic of each OD pair w in a way that matches closely the actual fractions of packets routed on the paths $p \in P_w$ with the desired fractions x_p/r_w . With this type of implementation each OD pair w takes into account changes in its own input traffic r_w , but makes no effort to adapt to changes in the input traffic of other OD pairs or to queue lengths inside the network. This problem considers the following more dynamic alternative.

Suppose that each OD pair w calculates the average number of packets N_p traveling on each path $p \in P_w$ by means of Little's theorem

$$N_p = x_p T_p, \quad \text{for all } p \in P_w, w \in W$$

where T_p is the estimated average round-trip packet delay on path p (time between introduction of the packet into the network and return of an end-to-end acknowledgment for the packet). The origin of each OD pair w monitors the *actual* number of routed but unacknowledged packets $\tilde{N}_p$ on path p and routes packets in a way that roughly equalizes the ratios $\tilde{N}_p/N_p$ for all paths $p \in P_w$. Note that this scheme is sensitive to changes in delay on the paths of an OD pair due to statistical fluctuations of the input traffic and/or the routing policies of other OD pairs. The difficulty with this scheme is that the delays T_p are unknown when the numbers N_p are calculated, and therefore T_p must be estimated somehow. The simplest possibility is to use the measured value of average delay during a preceding time period. You are asked to complete the argument in the following analysis of this scheme for a simple special case.

Suppose that there is a single OD pair and only two paths with flows and path delays denoted x_1, x_2 and $T_1(x), T_2(x)$, respectively, where $x = (x_1, x_2)$. The form of the path delay functions $T_1(x), T_2(x)$ is unknown. We assume that $x_1 + x_2 = r$ for some constant r . Suppose that we measure $x_1, x_2, T_1(x), T_2(x)$, then calculate new nominal path flows $\bar{x}_1, \bar{x}_2$ using some unspecified algorithm, and then implement them according to the ratio scheme described above.

- (a) Argue that the actual path flow vector $\tilde{x} = (\tilde{x}_1, \tilde{x}_2)$ is determined from $\tilde{x}_1 + \tilde{x}_2 = r$ and the relation

$$\frac{\tilde{x}_1 T_1(\tilde{x})}{\bar{x}_1 T_1(x)} = \frac{\tilde{x}_2 T_2(\tilde{x})}{\bar{x}_2 T_2(x)}$$

- (b) Assume that T_1 and T_2 are monotonically increasing in the sense that if $z = (z_1, z_2)$ and $z' = (z'_1, z'_2)$ are such that $z_1 > z'_1, z_2 < z'_2$ and $z_1 + z_2 = z'_1 + z'_2$, then $T_1(z) > T_1(z')$ and $T_2(z) < T_2(z')$. Show that the path flows $\tilde{x}_1, \tilde{x}_2$ of part (a) lie in the interval between x_1 and $\bar{x}_1$ and x_2 and $\bar{x}_2$, respectively.
- (c) Consider a convex cost function of the form $\sum_{(i,j)} D_{ij}(F_{ij})$. Under the assumption in part (b) show that if $\bar{x}$ gives a lower cost than x , the same is true for $\tilde{x}$.
- 5.39 Routing in Networks with Frequently Changing Topology [GaB81], [BeT89].** In some situations (e.g., mobile packet radio networks) topological changes are so frequent that topology broadcast algorithms are somewhat impractical. The algorithms of this problem are designed to cope with situations of this type. Consider a connected undirected graph with a special

node referred to as the *destination*. Consider the collection C of directed acyclic graphs obtained by assigning a unique direction to each of the undirected links. A graph G in C is said to be *rooted at the destination* if for every node there is a directed path in G starting at the node and ending at the destination. Show that the following two distributed asynchronous algorithms will yield a graph in C that is rooted at the destination starting from any other graph in C .

Algorithm A. A node other than the destination with no outgoing links reverses the direction of all its adjacent links. (This is done repeatedly until every node other than the destination has an outgoing link.)

Algorithm B. Every node i other than the destination keeps a list of its neighboring nodes j that have reversed the direction of the corresponding links (i, j) . At each iteration, each node i that has no outgoing link reverses the directions of the links (i, j) , for all j that do not appear on its list, and empties the list. If no such j exists (*i.e.*, the list is full), node i reverses the directions of all incoming links and empties the list. Initially, all lists are empty.

Hint: For algorithm A, assign to each node a distinct number. With each set of node numbers, associate the graph in C where each link is directed from a higher to a lower number. Consider an algorithm where each node reverses link directions by changing its number. Use a similar idea for algorithm B.

- 5.40** *Verifying the Termination of the Distributed Bellman–Ford Algorithm [DiS80], [BeT89].* The objective of the following algorithm (based on the Bellman–Ford method) is to compute in a distributed way a shortest path from a single origin (node 1) to all other nodes *and* to notify node 1 that the computation has terminated. Assume that all links (i, j) are bidirectional, have nonnegative length d_{ij} , maintain the order of messages sent on them, and operate with no errors or failures. Each node i maintains an estimate D_i of the shortest distance from node 1 to itself. Initially, $D_i = \infty$ for all nodes $i \neq 1$, and $D_1 = 0$. Node 1 initiates the computation by sending the estimate $D_1 + d_{1j}$ to all neighbor nodes j . The algorithmic rules for each node $j \neq 1$ are as follows:

1. When node j receives an estimate $D_i + d_{ij}$ from some other node i , after some unspecified finite delay (but before processing a subsequently received estimate), it does the following:
 - (a) If $D_j \leq D_i + d_{ij}$, node j sends an ACK to node i .
 - (b) If $D_j > D_i + d_{ij}$, node j sets $D_j = D_i + d_{ij}$, marks node i as its best current predecessor on a shortest path, sends an ACK to its previous best predecessor (if any), and sends the estimate $D_j + d_{jk}$ to each neighbor k .
2. Node j sends an ACK to its best current predecessor once it receives an ACK for each of the latest estimates sent to its neighbors.

We assume that each ACK is uniquely associated with a previously sent estimate, and that node 1 responds with an ACK to any length estimate it receives.

- (a) Show that eventually node 1 will receive an ACK from each of its neighbors, and at that time D_i will be the correct shortest distance for each node i .
- (b) What are the advantages and disadvantages of this algorithm compared with the distributed, asynchronous Bellman–Ford algorithm of Section 5.2.4?

Second Edition

Data Networks

DIMITRI BERTSEKAS

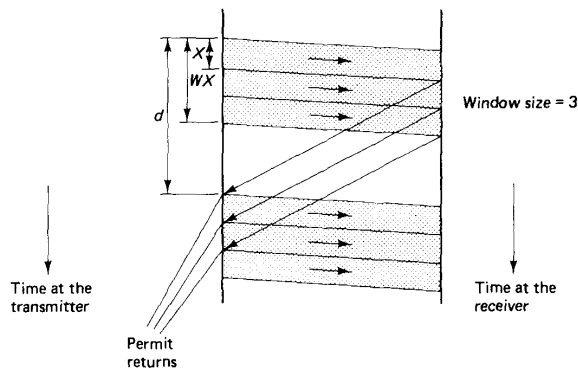
Massachusetts Institute of Technology

ROBERT GALLAGER

Massachusetts Institute of Technology



PRENTICE HALL, Englewood Cliffs, New Jersey 07632



Flow Control

6.1 INTRODUCTION

In most networks, there are circumstances in which the externally offered load is larger than can be handled even with optimal routing. Then, if no measures are taken to restrict the entrance of traffic into the network, queue sizes at bottleneck links will grow and packet delays will increase, possibly violating maximum delay specifications. Furthermore, as queue sizes grow indefinitely, the buffer space at some nodes may be exhausted. When this happens, some of the packets arriving at these nodes will have to be discarded and later retransmitted, thereby wasting communication resources. As a result, a phenomenon similar to a highway traffic jam may occur whereby, as the offered load increases, the actual network throughput decreases while packet delay becomes excessive. It is thus necessary at times to prevent some of the offered traffic from entering the network to avoid this type of congestion. This is one of the main functions of flow control.

Flow control is also sometimes necessary between two users for speed matching, that is, for ensuring that a fast transmitter does not overwhelm a slow receiver with more packets than the latter can handle. Some authors reserve the term "flow control" for this

type of speed matching and use the term “congestion control” for regulating the packet population within the subnetwork. We will not make this distinction in terminology; the type and objective of flow control being discussed will be clear from the context.

In this chapter we describe schemes currently used for flow control, explain their advantages and limitations, and discuss potential approaches for their improvement. In the remainder of this section we identify the principal means and objectives of flow control. In Sections 6.2 and 6.3 we describe the currently most popular flow control methods; window strategies and rate control schemes. In Section 6.4 we describe flow control in some representative networks. Section 6.5 is devoted to various algorithmic aspects of rate control schemes.

6.1.1 Means of Flow Control

Generally, a need for flow control arises whenever there is a constraint on the communication rate between two points due to limited capacity of the communication lines or the processing hardware. Thus, a flow control scheme may be required between two users at the transport layer, between a user and an entry point of the subnet (network layer), between two nodes of the subnet (network layer), or between two gateways of an interconnected network (internet layer). We will emphasize flow control issues within the subnet, since flow control in other contexts is in most respects similar.

The term “session” is used somewhat loosely in this chapter to mean any communication process to which flow control is applied. Thus a session could be a virtual circuit, a group of virtual circuits (such as all virtual circuits using the same path), or the entire packet flow originating at one node and destined for another node. Often, flow control is applied independently to individual sessions, but there is a strong interaction between its effects on different sessions because the sessions share the network’s resources.

Note that different sessions may have radically different service requirements. For example, sessions transferring files may tolerate considerable delay but may require strict error control, while voice and video sessions may have strict minimum data rate and maximum end-to-end delay requirements, but may tolerate occasional packet losses.

There are many approaches to flow control, including the following:

1. *Call blocking.* Here a session is simply blocked from entering the network (its access request is denied). Such control is needed, for example, when the session requires a minimum guaranteed data rate that the network cannot provide due to limited uncommitted transmission capacity. A typical situation is that of the voice telephone network and, more generally, circuit switched networks, all of which use flow control of this type. However, a call blocking option is also necessary in integrated voice, video, and data networks, at least with respect to those sessions requiring guaranteed rate. In a more general view of call blocking one may admit a session only after a negotiation of some “service contract,” for example, an agreement on some service parameters for the session’s input traffic (maximum rate, minimum rate, maximum burst size, priority level, etc.)

2. *Packet discarding.* When a node with no available buffer space receives a packet, it has no alternative but to discard the packet. More generally, however, packets may be discarded while buffer space is still available if they belong to sessions that are using more than their fair share of some resource, are likely to cause congestion for higher-priority sessions, are likely to be discarded eventually along their path, and so on. (Note that if a packet has to be discarded anyway, it might as well be discarded as early as possible to avoid wasting additional network resources unnecessarily.) When some of a session's packets are discarded, the session may need to take some corrective action, depending on its service requirements. For sessions where all packets carry essential information (*e.g.*, file transfer sessions), discarded packets must be retransmitted by the source after a suitable timeout; such sessions require an acknowledgment mechanism to keep track of the packets that failed to reach their destination. On the other hand, for sessions such as voice or video, where delayed information is useless, there is nothing to be done about discarded packets. In such cases, packets may be assigned different levels of priority, and the network may undertake the obligation never to discard the highest-priority packets—these are the packets that are sufficient to support the minimum acceptable quality of service for the session. The data rate of the highest-priority packets (the minimum guaranteed rate) may then be negotiated between the network and the source when the session is established. This rate may also be adjusted in real time, depending on the congestion level in the network.
3. *Packet blocking.* When a packet is discarded at some node, the network resources that were used to get the packet to that node are wasted. It is thus preferable to restrict a session's packets from entering the network if after entering they are to be discarded. If the packets carry essential information, they must wait in a queue outside the network; otherwise, they are discarded at the source. In the latter case, however, the flow control scheme must honor any agreement on a minimum guaranteed rate that the session may have negotiated when it was first established.
4. *Packet scheduling.* In addition to discarding packets, a subnetwork node can exercise flow control by selectively expediting or delaying the transmission of the packets of various sessions. For example, a node may enforce a priority service discipline for transmitting packets of several different priorities on a given outgoing link. As another example, a node may use a (possibly weighted) round-robin scheduling strategy to ensure that various sessions can access transmission lines in a way that is consistent both with some fairness criterion and also with the minimum data rate required by the sessions. Finally, a node may receive information regarding congestion farther along the paths used by some sessions, in which case it may appropriately delay the transmission of the packets of those sessions.

In subsequent sections we discuss specific strategies for throttling sources and for restricting traffic access to the network.

6.1.2 Main Objectives of Flow Control

We look now at the main principles that guide the design of flow control algorithms. Our focus is on two objectives. First, *strike a good compromise between throttling sessions (subject to minimum data rate requirements) and keeping average delay and buffer overflow at a reasonable level*. Second, *maintain fairness between sessions in providing the requisite quality of service*.

Limiting delay and buffer overflow. We mentioned earlier that for important classes of sessions, such as voice and video, packets that are excessively delayed are useless. For such sessions, a limited delay is essential and should be one of the chief concerns of the flow control algorithm; for example, such sessions may be given high priority.

For other sessions, a small average delay per packet is desirable but it may not be crucial. For these sessions, network level flow control does not necessarily reduce delay; it simply shifts delay from the network layer to higher layers. That is, by restricting entrance to the subnet, flow control keeps packets waiting outside the subnet rather than in queues inside it. In this way, however, flow control avoids wasting subnet resources in packet retransmissions and helps prevent a disastrous traffic jam inside the subnet. Retransmissions can occur in two ways: first, the buildup of queues causes buffer overflow to occur and packets to be discarded; second, slow acknowledgments can cause the source node to retransmit some packets because it thinks mistakenly that they have been lost. Retransmissions waste network resources, effectively reduce network throughput, and cause congestion to spread. The following example (from [GeK80]) illustrates how buffer overflow and attendant retransmissions can cause congestion.

Example 6.1

Consider the five-node network shown in Fig. 6.1(a). There are two sessions, one from top to bottom with a Poisson input rate of 0.8, and the other from left to right with a Poisson input rate f . Assume that the central node has a large but finite buffer pool that is shared on a first-come first-serve basis by the two sessions. If the buffer pool is full, an incoming packet is rejected and then retransmitted by the sending node. For small f , the buffer rarely fills up and the total throughput of the system is $0.8 + f$. When f is close to unity (which is the capacity of the rightmost link), the buffer of the central node is almost always full, while the top and left nodes are busy most of the time retransmitting packets. Since the left node is transmitting 10 times faster than the top node, it has a 10-fold greater chance of capturing a buffer space at the central node, so the left-to-right throughput will be roughly 10 times larger than the top-to-bottom throughput. The left-to-right throughput will be roughly unity (the capacity of the rightmost link), so that the total throughput will be roughly 1.1. This is illustrated in more detail in Fig. 6.1(b), where it can be seen that the total throughput decreases toward 1.1 as the offered load f increases.

This example also illustrates how with buffer overflow, some sessions can capture almost all the buffers and nearly prevent other sessions from using the network. To avoid this, it is sometimes helpful to implement a *buffer management scheme*. In such a scheme, packets are divided in different classes based, for example, on origin, destination,

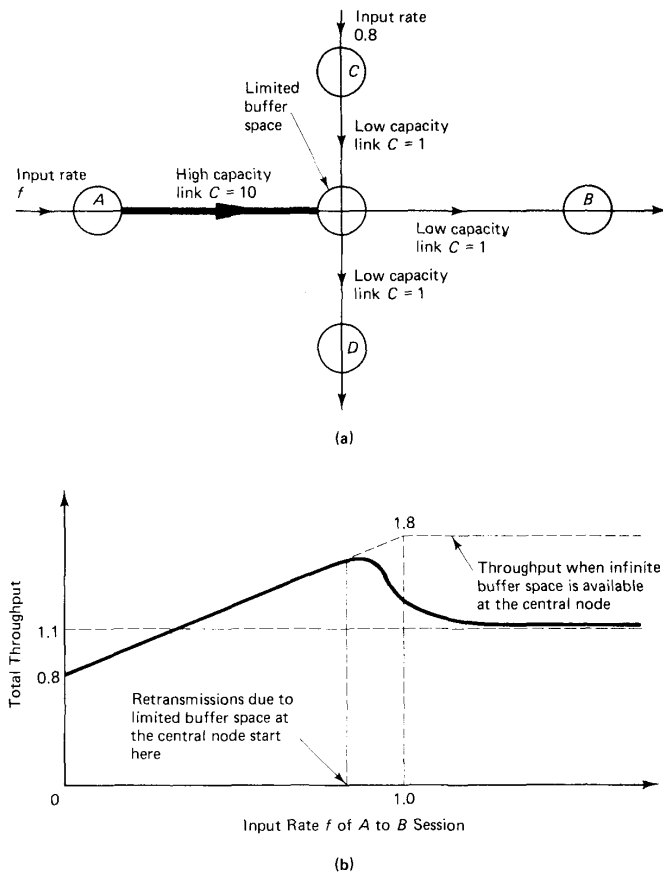


Figure 6.1 Example demonstrating throughput degradation due to retransmissions caused by buffer overflow. (a) For f approaching unity, the central node buffer is almost always full, thereby causing retransmissions. Because the A -to- B session uses a line 10 times faster than the C -to- D session, it has a 10-fold greater chance of capturing a free buffer and getting a packet accepted at the central node. As a result, the throughput of the A -to- B session approaches unity, while the throughput of the C -to- D session approaches 0.1. (b) Total throughput as a function of the input rate of the A -to- B session.

or number of hops traveled so far; at each node, separate buffer space is reserved for different classes, while some buffer space is shared by all classes.

Proper buffer management can also help avoid deadlocks due to buffer overflow. Such a deadlock can occur when two or more nodes are unable to move packets due to unavailable space at all potential receivers. The simplest example of this is two

nodes A and B routing packets directly to each other as shown in Fig. 6.2(a). If all the buffers of both A and B are full of packets destined for B and A , respectively, then the nodes are deadlocked into continuously retransmitting the same packets with no success, as there is no space to store the packets at the receiver. This problem can also occur in a more complex manner whereby more than two nodes arranged in a cycle are deadlocked because their buffers are full of packets destined for other nodes in the cycle [see Fig. 6.2(b)]. There are simple buffer management schemes that preclude this type of deadlock by organizing packets in priority classes and allocating extra buffers for packets of higher priority ([RaH76] and [Gop85]). A typical choice is to assign a level of priority to a packet equal to the number of links it has traversed in the network, as shown in Fig. 6.3. If packets are not allowed to loop, it is then possible to show that a deadlock of the type just described cannot occur.

We finally note that when offered load is large, limited delay and buffer overflow can be achieved only by lowering the input to the network. Thus, there is a natural trade-off between giving sessions free access to the network and keeping delay at a level low enough so that retransmissions or other inefficiencies do not degrade network performance. A somewhat oversimplified guideline is that, ideally, flow control should not be exercised at all when network delay is below some critical level, and, under heavy load conditions, should reject as much offered traffic as necessary to keep delay at the critical level. Unfortunately, this is easier said than done, since neither delay nor throughput can be represented meaningfully by single numbers in a flow control context.

Fairness. When offered traffic must be cut back, it is important to do so fairly. The notion of fairness is complicated, however, by the presence of different session priorities and service requirements. For example, some sessions need a minimum guaranteed rate and a strict upper bound on network delay. Thus, while it is appropriate to consider simple notions of fairness within a class of “similar” sessions, the notion of

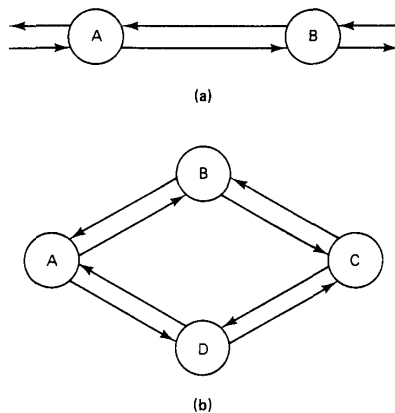


Figure 6.2 Deadlock due to buffer overflow. In (a), all buffers of A and B are full of packets destined for B and A , respectively. As a result, no packet can be accepted at either node. In (b), all buffers of A , B , C , and D are full of packets destined for C , D , A , and B , respectively.

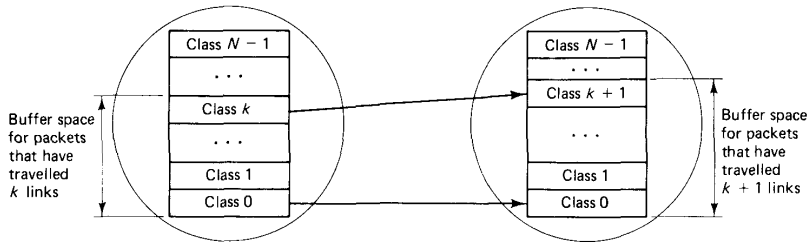


Figure 6.3 Organization of node memory in buffer classes to avoid deadlock due to buffer overflow. A packet that has traveled k links is accepted at a node only if there is an available buffer of class k or lower, where k ranges from 0 to $N - 1$ (where N is the number of nodes). Assuming that packets that travel more than $N - 1$ links are discarded as having traveled in a loop, no deadlock occurs. The proof consists of showing by induction (starting with $k = N - 1$) that at each node the buffers of class k cannot fill up permanently.

fairness between classes is complex and involves the requirements of those classes. Even within a class, there are several notions of fairness that one may wish to adopt. The example of Fig. 6.4 illustrates some of the contradictions inherent in choosing a fairness criterion. There are $n + 1$ sessions each offering 1 unit/sec of traffic along a sequence of n links with capacity of 1 unit/sec. One session's traffic goes over all n links, while the rest of the traffic goes over only one link. A maximum throughput of n units/sec can be achieved by accepting all the traffic of the single-link sessions while shutting off the n -link session. However, if our objective is to give equal rate to all sessions, the resulting throughput is only $(n + 1)/2$ units/sec. Alternatively, if our objective is to give equal resources to all sessions, the single link sessions should get a rate of $n/(n + 1)$ units/sec, while the n -link session should get a rate of $1/(n + 1)$ units/sec.

Generally, a compromise is required between equal access to a network resource and throttling those sessions that are most responsible for using up that resource. Achieving the proper balance, however, is a design decision that may be hard to quantify; often such decisions must be reached by trial and error.

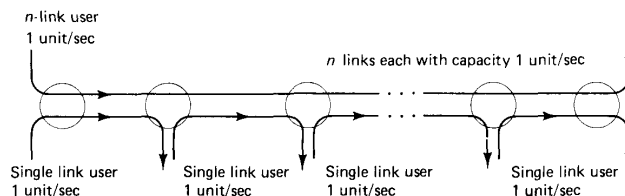


Figure 6.4 Example showing that maximizing total throughput may be incompatible with giving equal throughput to all sessions. A maximum throughput of n units/sec can be achieved if the n -link session is shut off completely. Giving equal rates of $1/2$ unit/sec to all sessions achieves a throughput of only $(n + 1)/2$ units/sec.

6.2 WINDOW FLOW CONTROL

In this section we describe the most frequently used class of flow control methods. In Sections 6.2.1 to 6.2.3, the main emphasis is on flow control within the communication subnet. Flow control outside the subnet, at the transport layer, is discussed briefly in Section 6.2.4.

A session between a transmitter A and a receiver B is said to be *window flow controlled* if there is an upper bound on the number of data units that have been transmitted by A and are not yet known by A to have been received by B (see Fig. 6.5). The upper bound (a positive integer) is called the *window size* or, simply, the *window*. The transmitter and receiver can be, for example, two nodes of the communication subnet, a user's machine and the entry node of the communication subnet, or the users' machines at the opposite ends of a session. Finally, the data units in a window can be messages, packets, or bytes, for example.

The receiver B notifies the transmitter A that it has disposed of a data unit by sending a special message to A , which is called a *permit* (other names in the literature are *acknowledgment*, *allocate message*, etc.). Upon receiving a permit, A is free to send one more data unit to B . Thus, a permit may be viewed as a form of passport that a data unit must obtain before entering the logical communication channel between A and B . The number of permits in use should not exceed the window size.

Permits are either contained in special control packets, or are piggybacked on regular data packets. They can be implemented in a number of ways; see the practical examples of Section 6.4 and the following discussion. Note also that a window flow control scheme for a given session may be combined with an error control scheme for the session, where the permits also play the role of acknowledgments; see Section 2.8.2 and the descriptions of the ARPANET and the Codex network in Section 6.4.

The general idea in the window strategy is that the input rate of the transmitter is reduced when permits return slowly. Therefore, if there is congestion along the communication path of the session, the attendant large delays of the permits cause a natural slowdown of the transmitter's data rate. However, the window strategy has an additional dimension, whereby the receiver may intentionally delay permits to restrict

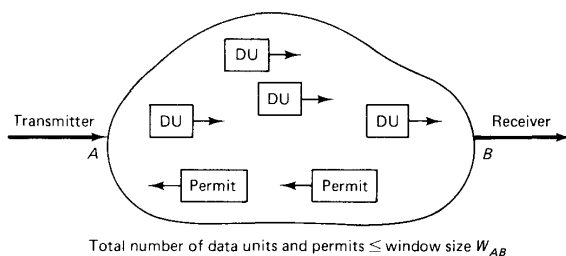


Figure 6.5 Window flow control between a transmitter and a receiver consists of an upper bound W_{AB} on the number of data units and permits in transit inside the network.

the transmission rate of the session. For example, the receiver may do so to avoid buffer overflow.

In the subsequent discussion, we consider two strategies, *end-to-end* and *node-by-node* windowing. The first strategy refers to flow control between the entry and exit subnet nodes of a session, while the second strategy refers to flow control between every pair of successive nodes along a virtual circuit's path.

6.2.1 End-to-End Windows

In the most common version of end-to-end window flow control, the window size is αW , where α and W are some positive numbers. Each time a new batch of α data units is received at the destination node, a permit is sent back to the source allocating a new batch of α data units. In a variation of this scheme, the destination node will send a new α -data unit permit upon reception of just the first of an α -data unit batch. (See the SNA pacing scheme description in Section 6.3.) To simplify the following exposition, we henceforth assume that $\alpha = 1$, but our conclusions are valid regardless of the value of α . Also, for concreteness, we talk in terms of packets, but the window maintained may consist of other data units such as bytes.

Usually, a numbering scheme for packets and permits is used so that permits can be associated with packets previously transmitted and loss of permits can be detected. One possibility is to use a sliding window protocol similar to those used for data link control, whereby a packet contains a sequence number and a request number. The latter number can serve as one or more permits for flow control purposes (see also the discussion in Section 2.8.2). For example, suppose that node A receives a packet from node B with request number k . Then A knows that B has disposed of all packets sent by A and numbered less than k , and therefore A is free to send those packets up to number $k + W - 1$ that it has not sent yet, where W is the window size. In such a scheme, both the sequence number and the request number are represented modulo m , where $m \geq W + 1$. One can show that if packet ordering is preserved between transmitter and receiver, this representation of numbers is adequate; the proof is similar to the corresponding proof for the goback n ARQ system. In some networks the end-to-end window scheme is combined with an end-to-end retransmission protocol, and a packet is retransmitted if following a suitable timeout, the corresponding permit has not returned to the source.

To simplify the subsequent presentation, the particular manner in which permits are implemented will be ignored. It will be assumed that the source node simply counts the number x of packets it has already transmitted but for which it has not yet received back a permit, and transmits new packets only as long as $x < W$.

Figure 6.6 shows the flow of packets for the case where the round-trip delay d , including round-trip propagation delay, packet transmission time, and permit delay is smaller than the time required to transmit the full window of W packets, that is,

$$d \leq WX$$

where X is the transmission time of a single packet. Then the source is capable of transmitting at the full speed of $1/X$ packets/sec, and flow control is not active. (To

simplify the following exposition, assume that all packets have equal transmission time and equal round-trip delay.)

The case where flow control is active is shown in Fig. 6.7. Here

$$d > WX$$

and the round-trip delay d is so large that the full allocation of W packets can be transmitted before the first permit returns. Assuming that the source always has a packet waiting in queue, the rate of transmission is W/d packets/sec.

If the results of Figs. 6.6 and 6.7 are combined, it is seen that the maximum rate of transmission corresponding to a round-trip delay d is given by

$$r = \min \left\{ \frac{1}{X}, \frac{W}{d} \right\} \quad (6.1)$$

Figure 6.8 illustrates the flow control mechanism; the source transmission rate is reduced in response to congestion and the attendant large delay. Furthermore, assuming that W is relatively small, the window scheme reacts fast to congestion—within at most W packets' transmission time. This fast reaction, coupled with low overhead, is the major advantage of window strategies over other (nonwindow) schemes.

Limitations of end-to-end windows. One drawback of end-to-end windows is that they cannot guarantee a minimum communication rate for a session. Thus windows are inadequate for sessions that require a minimum guaranteed rate, such as voice and video sessions. Other drawbacks of the end-to-end window strategy have to do with window sizes. There is a basic trade-off here: One would like to make window sizes small to limit the number of packets in the subnet, thus avoiding large delays and congestion, and one would also like to make window sizes large to allow full-speed transmission and maximal throughput under light-to-moderate traffic conditions. Determining the proper window size and adjusting that size in response to congestion is not easy. This delay-throughput trade-off is particularly acute for high-speed networks, where because of high propagation delay relative to packet transmission time, window sizes should be large to allow high data rates [cf. Eq. (6.1) and Fig. 6.8]. One should remember, however, that the existence of a high-speed network does not necessarily make

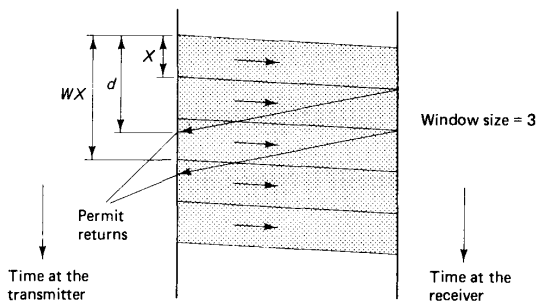


Figure 6.6 Example of full-speed transmission with a window size $W = 3$. The round-trip delay d is less than the time WX required to transmit the full window of W packets.

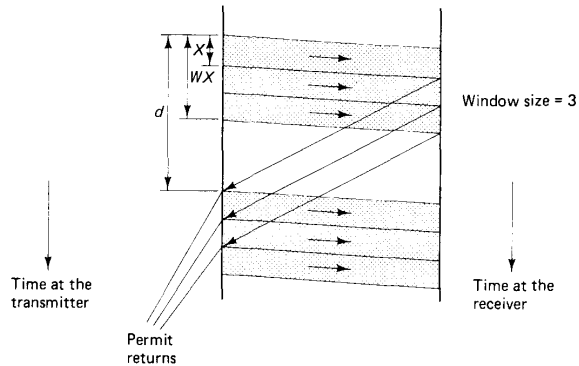


Figure 6.7 Example of delayed transmission with a window size $W = 3$. The round-trip delay d is more than the time WX required to transmit the full window of W packets. As a result, window flow control becomes active, restricting the input rate to W/d .

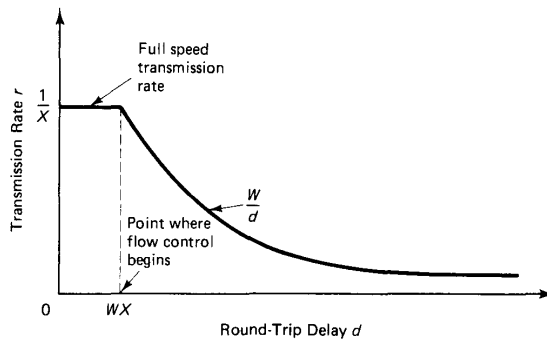


Figure 6.8 Transmission rate versus round-trip delay in a window flow control system. This oversimplified relationship assumes that all packets require equal transmission time at the source and have equal round-trip packet/permit delay.

it desirable for individual sessions to transmit at the full network speed. For high-speed networks, it is generally true that the window size of the session should be proportional to the round-trip propagation delay for the session. However, the window size should also be proportional to the session's maximum *desired* rate rather than the maximum rate allowed by the network; cf. Eq. (6.1).

End-to-end windows may also fail to provide adequate control of packet delay. To understand the relation between window size, delay, and throughput, suppose that there are n actively flow controlled sessions in the network with fixed window sizes $W_1, \dots, W_n$. Then the total number of packets and permits traveling in the network is

$\sum_{i=1}^n W_i$. Focusing on packets (rather than permits), we see that their number in the network is $\sum_{i=1}^n \beta_i W_i$, where β_i is a factor between 0 and 1 that depends on the relative magnitude of return delay for the permit and the extent to which permits are piggybacked on regular packets. By Little's Theorem, the average delay per packet is given by

$$T = \frac{\sum_{i=1}^n \beta_i W_i}{\lambda}$$

where λ is the throughput (total accepted input rate of sessions). As the number of sessions increases, the throughput λ is limited by the link capacities and will approach a constant. (This constant will depend on the network, the location of sources and destinations, and the routing algorithm.) Therefore, the delay T will roughly increase proportionately to the number of sessions (more accurately their total window size) as shown in Fig. 6.9. Thus, if the number of sessions can become very large, the end-to-end window scheme may not be able to keep delay at a reasonable level and prevent congestion.

One may consider using small window sizes as a remedy to the problem of very large delays under high load conditions. Unfortunately, in many cases (particularly in low- and moderate-speed networks) one would like to allow sessions to transmit at maximum speed when there is no other interfering traffic, and this imposes a lower bound on window sizes. Indeed, if a session is using an n -link path with a packet transmission time X on each link, the round-trip packet and permit delay will be at least nX and considerably more if permits are not given high priority on the return channel. For example, if permits are piggybacked on return packets traveling on the same path in the opposite direction, the return time will be at least nX also. So from Fig. 6.8, we see that full-speed transmission will not be possible for that session even under light load conditions unless the window size exceeds the number of links n on the path (see Fig. 6.10). For this reason, recommended window sizes are typically between n and $3n$. This recommendation assumes that the transmission time on each link is much larger than the processing and propagation delay. When the propagation delay is much larger than

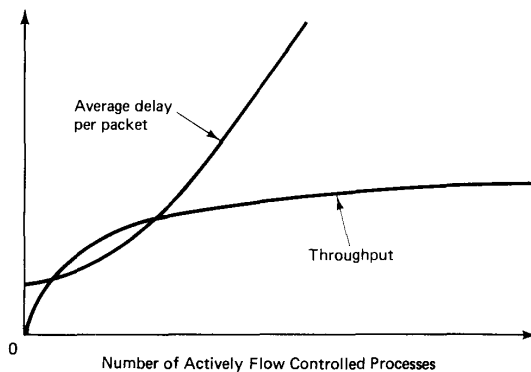


Figure 6.9 Average delay per packet and throughput as a function of the number of actively window flow controlled sessions in the network. When the network is heavily loaded, the average delay per packet increases approximately linearly with the number of active sessions, while the total throughput stays approximately constant. (This assumes that there are no retransmissions due to buffer overflow and/or large permit delays. In the presence of retransmissions, throughput may decrease as the number of active sessions increases.)

the transmission time, as in satellite links and some high-speed networks, the appropriate window size might be much larger. This is illustrated in the following example.

Example 6.2

Consider a transmission line with capacity of 1 gigabit per second (10^9 bits/sec) connecting two nodes which are 50 miles apart. The round-trip propagation delay can be roughly estimated as 1 millisecond. Assuming a packet size of 1000 bits, it is seen that a window size of at least 1000 packets is needed to sustain full-speed transmission; it is necessary to have 1000 packets and permits simultaneously propagating along the transmission line to "keep the pipeline full," with permits returning fast enough to allow unimpeded transmission of new packets. By extrapolation it is seen that Atlantic coast to Pacific coast end-to-end transmission over a distance of, say, 3000 miles requires a window size of at least 60,000 packets! Fortunately, for most sessions, such unimpeded transmission is neither required nor desirable.

Example 6.2 shows that windows should be used with care when high-speed transmission over a large distance is involved; they require excessive memory and they respond to congestion relatively slowly when the round-trip delay time is very long relative to the packet transmission time. For networks with relatively small propagation delays, end-to-end window flow control may be workable, particularly if there is no requirement for a minimum guaranteed rate. However, to achieve a good trade-off between delay and throughput, dynamic window adjustment is necessary. Under light load conditions, windows should be large and allow unimpeded transmission, while under heavy load conditions, windows should shrink somewhat, thereby not allowing delay to become excessive. This is not easy to do systematically, but some possibilities are examined in Section 6.2.5.

End-to-end windows can also perform poorly with respect to fairness. It was argued earlier that when propagation delay is relatively small, the proper window size of a session should be proportional to the number of links on its path. This means that at a heavily loaded link, long-path sessions can have many more packets awaiting transmission than short-path sessions, thereby obtaining a proportionately larger throughput. A typical situation is illustrated in Fig. 6.11. Here the windows of all sessions accumulate at the heavily loaded link. If packets are transmitted in the order of their arrival, the rate of transmission obtained by each session is roughly proportional to its window size, and this favors the long-path sessions.

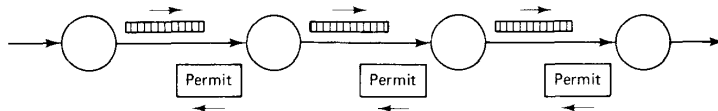


Figure 6.10 The window size must be at least equal to the number of links on the path to achieve full-speed transmission. (Assuming equal transmission time on each link, a packet should be transmitted at each link along the path simultaneously to achieve nonstop transmission.) If permit delay is comparable to the forward delay, the window size should be doubled. If the propagation delay is not negligible, an even larger window size is needed.

The fairness properties of end-to-end windows can be improved if flow-controlled sessions of the same priority class are served via a weighted round-robin scheme at each transmission queue. Such a scheme should take into account the priorities as well as the minimum guaranteed rate of different sessions. Using a round-robin scheme is conceptually straightforward when each session is a virtual circuit, but not in a datagram network, where it may not be possible to associate packets with particular flow-controlled sessions.

6.2.2 Node-by-Node Windows for Virtual Circuits

In this strategy, there is a separate window for every virtual circuit and pair of adjacent nodes along the path of the virtual circuit. Much of the discussion on end-to-end windows applies to this scheme as well. Since the path along which flow control is exercised is effectively one link long, the size of a window measured in packets is typically two or three for moderate-speed terrestrial links. For high-speed networks, the required window size might be much larger, thereby making the node-by-node strategy less attractive. For this reason, the following discussion assumes that a window size of about two is a reasonable choice.

Let us focus on a pair of successive nodes along a virtual circuit's path; we refer to them as the transmitter and the receiver. The main idea in the node-by-node scheme is that the receiver can avoid the accumulation of a large number of packets into its memory by slowing down the rate at which it returns permits to the transmitter. In the most common strategy, the receiver maintains a W -packet buffer for each virtual circuit and returns a permit to the transmitter as soon as it releases a packet from its W -packet buffer. A packet is considered to be released from the W -packet buffer once it is either delivered to a user outside the subnet or is entered in the data link control (DLC) unit leading to the subsequent node on the virtual circuit's path.

Consider now the interaction of the windows along three successive nodes ($i-1$, i , and $i+1$) on a virtual circuit's path. Suppose that the W -packet buffer of node i is full. Then node i will send a permit to node $i-1$ once it delivers an extra packet to the DLC of the $(i, i+1)$ link, which in turn will occur once a permit sent by node $(i+1)$ is received at node i . Thus, there is coupling of successive windows along the path of a virtual circuit. In particular, suppose that congestion develops at some link. Then the W -packet window

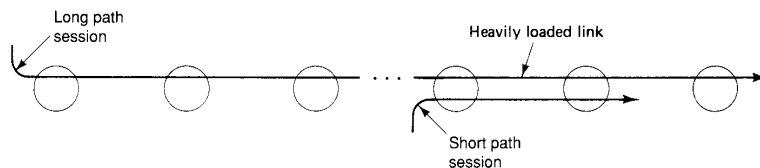


Figure 6.11 End-to-end windows discriminate in favor of long-path sessions. It is necessary to give a large window to a long-path session to achieve full-speed transmission. Therefore, a long-path session will typically have more packets waiting at a heavily loaded link than will a short-path session, and will receive proportionally larger service (assuming that packets are transmitted on a first-come first-serve basis).

at the start node of the congested link will fill up for each virtual circuit crossing the link. As a result, the W -packet windows of nodes lying upstream of the congested link will progressively fill up, including the windows of the origin nodes of the virtual circuits crossing the congested link. At that time, these virtual circuits will be actively flow controlled. The phenomenon whereby windows progressively fill up from the point of congestion toward the virtual circuit origins is known as *backpressure* and is illustrated in Fig. 6.12.

One attractive aspect of node-by-node windows can be seen from Fig. 6.12. In the worst case, where congestion develops on the last link (say, the n th) of a virtual circuit's path, the total number of packets inside the network for the virtual circuit will be approximately nW . If the virtual circuit were flow controlled via an end-to-end window, the total number of packets inside the network would be roughly comparable. (This assumes a window size of $W = 2$ in the node-by-node case, and of $W \simeq 2n$ in the end-to-end case based on the rule of thumb of using a window size that is twice the number of links of the path between transmitter and receiver.) The important point, however, is that these packets will be uniformly distributed along the virtual circuit's path in the node-by-node case, but will be concentrated at the congested link in the end-to-end case. Because of this the amount of memory required at each node to prevent buffer overflow may be much smaller for node-by-node windows than for end-to-end windows.

Distributing the packets of a virtual circuit uniformly along its path also alleviates the fairness problem, whereby large window sessions monopolize a congested link at the expense of small window sessions (cf. Fig. 6.11). This is particularly true when the window sizes of all virtual circuits are roughly equal as, for example, when the circuits involve only low-speed terrestrial links. A fairness problem, however, may still arise when satellite links (or other links with relatively large propagation delay) are involved. For such links, it is necessary to choose a large window size to achieve unimpeded transmission when traffic is light because of the large propagation delay. The difficulty arises at a node serving both virtual circuits with large window size that come over a satellite link and virtual circuits with small window size that come over a terrestrial link (see Fig. 6.13). If these circuits leave the node along the same transmission line, a fairness problem may develop when this line gets heavily loaded. A reasonable way to address this difficulty is to schedule transmissions of packets from different virtual circuits on a weighted round-robin basis (with the weights accounting for different priority classes).

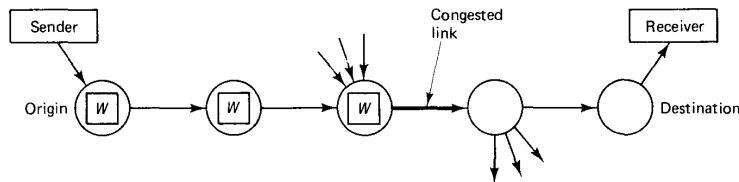


Figure 6.12 Backpressure effect in node-by-node flow control. Each node along a virtual circuit's path can store no more than W packets for that virtual circuit. The window storage space of each successive node lying upstream of the congested link fills up. Eventually, the window of the origin node fills, at which time transmission stops.

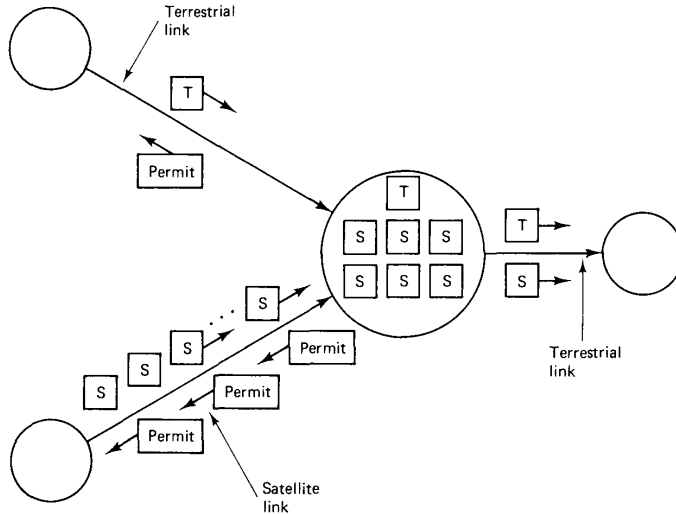


Figure 6.13 Potential fairness problem at a node serving virtual circuits that come over a satellite link (large window) and virtual circuits that come over a terrestrial link (small window). If virtual circuits are served on a first-come first-serve basis, the virtual circuits with large windows will receive better service on a subsequent transmission line. This problem can be alleviated by serving virtual circuits via a round-robin scheme.

6.2.3 The Isarithmic Method

The isarithmic method may be viewed as a version of window flow control whereby there is a single global window for the entire network. The idea here is to limit the total number of packets in the network by having a fixed number of permits circulating in the subnet. A packet enters the subnet after capturing one of these permits. It then releases the permit at its destination node. The total number of packets in the network is thus limited by the number of permits. This has the desirable effect of placing an upper bound on average packet delay that is independent of the number of sessions in the network. Unfortunately, the issues of fairness and congestion within the network depend on how the permits are distributed, which is not addressed by the isarithmic approach. There are no known sensible algorithms to control the location of the permits, and this is the main difficulty in making the scheme practical. There is also another difficulty, namely that permits can be destroyed through various malfunctions and there may be no easy way to keep track of how many permits are circulating in the network.

6.2.4 Window Flow Control at Higher Layers

Much of what has been described so far about window strategies is applicable to higher layer flow control of a session, possibly communicating across several interconnected

subnetworks. Figure 6.14 illustrates a typical situation involving a single subnetwork. A user sends data out of machine *A* to an entry node *NA* of the subnet, which is forwarded to the exit node *NB* and is then delivered to machine *B*. There is (network layer) flow control between the entry and exit nodes *NA* and *NB* (either end-to-end or node-by-node involving a sequence of nodes). There is also (network layer) window flow control between machine *A* and entry node *NA*, which keeps machine *A* from swamping node *NA* with more data than the latter can handle. Similarly, there is window flow control between exit node *NB* and machine *B*, which keeps *NB* from overwhelming *B* with too much data. Putting the pieces together we see that there is a network layer flow control system extending from machine *A* to machine *B*, which operates much like the node-by-node window flow control system of Section 6.2.2. In essence, we have a three-link path where the subnet between nodes *NA* and *NB* is viewed conceptually as the middle link.

It would appear that the system just described would be sufficient for flow control purposes, and in many cases it is. For example, it is the only one provided in the TYMNET and the Codex network described later on a virtual circuit basis. There may be a need, however, for additional flow control at the transport layer, whereby the input traffic rate of a user at machine *A* is controlled directly by the receiver at machine *B*. One reason is that the network layer window flow control from machine *A* to node *NA* may apply collectively to multiple user-pair sessions. These sessions could, for a variety of reasons, be multiplexed into a single traffic stream but have different individual flow control needs. For example, in SNA there is a transport layer flow control algorithm, which is the same as the one used in the network layer except that it is applied separately for each session rather than to a group of sessions. (See the description given in Section 6.4).

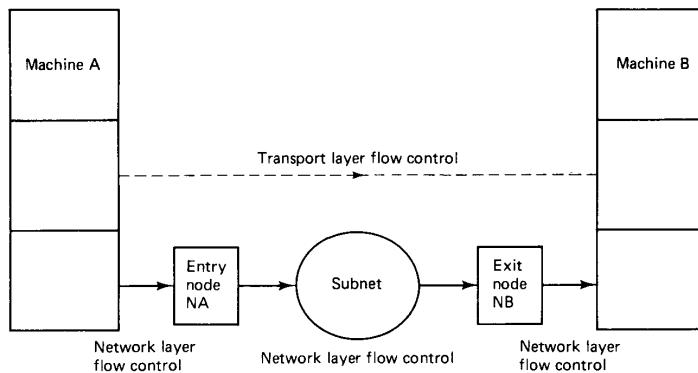


Figure 6.14 User-to-user flow control. A user sends data from machine *A* to a user in machine *B* via the subnet using the entry and exit nodes *NA* and *NB*. There is a conceptual node-by-node network layer flow control system along the connection *A-NA-NB-B*. There may also be direct user-to-user flow control at the transport layer or the internet sublayer, particularly if several users with different flow control needs are collectively flow controlled within the subnetwork.

In the case where several networks are interconnected with gateways, it may be useful to have windows for the gateway-to-gateway traffic of sessions that span several networks. If we take a higher-level view of the interconnected network where the gateways correspond to nodes and the networks correspond to links (Section 5.1.3), this amounts to using node-by-node windows for flow control in the internet sublayer. The gateway windows serve the purpose of distributing the total window of the internet network sessions, thereby alleviating a potential congestion problem at a few gateways. However, the gateway-to-gateway windows may apply to multiple transport layer sessions, thereby necessitating transport layer flow control for each individual session.

6.2.5 Dynamic Window Size Adjustment

We mentioned earlier that it is necessary to adjust end-to-end windows dynamically, decreasing their size when congestion sets in. The most common way of doing this is through feedback from the point of congestion to the appropriate packet sources.

There are several ways by which this feedback can be obtained. One possibility is for nodes that sense congestion to send a special packet, sometimes called a *choke packet*, to the relevant sources. Sources that receive such a packet must reduce their windows. The sources can then attempt to increase their windows gradually following a suitable timeout. The method by which this is done is usually ad hoc in practice, and is arrived at by trial and error or simulation. The circumstances that will trigger the generation of choke packets may vary: for example, buffer space shortage or excessive queue length.

It is also possible to adjust window sizes by keeping track of permit delay or packet retransmissions. If permits are greatly delayed or if several retransmissions occur at a given source within a short period of time, this is likely to mean that packets are being excessively delayed or are getting lost due to buffer overflow. The source then reduces the relevant window sizes, and subsequently attempts to increase them gradually following a timeout.

Still another way to obtain feedback is to collect congestion information on regular packets as they traverse their route from origin to destination. This congestion information can be used by the destination to adjust the window size by withholding the return of some permits. A scheme of this type is used in SNA and is described in Section 6.4.

6.3 RATE CONTROL SCHEMES

We mentioned earlier that window flow control is not very well suited for high-speed sessions in high-speed wide area networks because the propagation delays are relatively large, thus necessitating large window sizes. An even more important reason is that windows do not regulate end-to-end packet delays well and do not guarantee a minimum data rate. Voice, video, and an increasing variety of data sessions require upper bounds on delay and lower bounds on rate. High-speed wide area networks increasingly carry such traffic, and many lower-speed networks also carry such traffic, making windows inappropriate.

An alternative form of flow control is based on giving each session a guaranteed data rate, which is commensurate to its needs. This rate should lie within certain limits that depend on the session type. For example, for a voice session, the rate should lie between the minimum needed for language intelligibility and a maximum beyond which the quality of voice cannot be further improved.

The main considerations in setting input session rates are:

1. *Delay-throughput trade-off.* Increasing throughput by setting the rates too high runs the risk of buffer overflow and excessive delay.
2. *Fairness.* If session rates must be reduced to accommodate some new sessions, the rate reduction must be done fairly, while obeying the minimum rate requirement of each session.

We will discuss various rate adjustment schemes focusing on these considerations in Section 6.5.

Given an algorithm that generates desired rates for various sessions, the question of implementing these rates arises. A strict implementation of a session rate of r packets/sec would be to admit 1 packet each $1/r$ seconds. This, however, amounts to a form of time-division multiplexing and tends to introduce large delays when the offered load of the sessions is bursty. A more appropriate implementation is to admit as many as W packets ($W > 1$) every W/r seconds. This allows a burst of as many as W packets into the network without delay, and is better suited for a dynamically changing load. There are several variations of this scheme. The following possibility is patterned after window flow control.

An allocation of W packets (a window) is given to each session, and a count x of the unused portion of this allocation is kept at the session origin. Packets from the session are admitted into the network as long as $x > 0$. Each time a packet is admitted, the count is decremented by 1, and W/r seconds later (r is the rate assigned to the session), the count is incremented by 1 as shown in Fig. 6.15. This scheme, called *time window flow control*, is very similar to window flow control with window size W except that the count is incremented W/r seconds after admitting a packet instead of after a round-trip delay when the corresponding permit returns.

A related method that regulates the burstiness of the transmitted traffic somewhat better is the so-called *leaky bucket scheme*. Here the count is incremented periodically, every $1/r$ seconds, up to a maximum of W packets. Another way to view this scheme is to imagine that for each session, there is a queue of packets without a permit and a bucket of permits at the session's source. The packet at the head of the packet queue obtains a permit once one is available in the permit bucket and then joins the set of packets with permits waiting to be transmitted (see Fig. 6.16). Permits are generated at the desired input rate r of the session (one permit each $1/r$ seconds) as long as the number in the permit bucket does not exceed a certain threshold W . The leaky bucket scheme is used in PARIS, an experimental high-speed network developed by IBM ([CiG88]; see Section 6.4). A variation, implemented in the ARPANET, is to allocate $W > 1$ permits initially to a session and subsequently restore the count back to W every W/r seconds, whether or not the session used any part of the allocation.

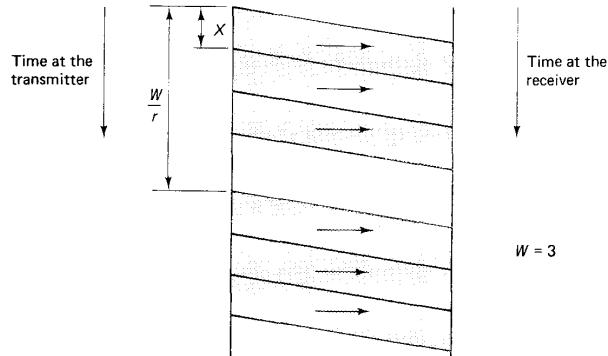


Figure 6.15 Time window flow control with $W = 3$. The count of packet allocation is decremented when a packet is transmitted and incremented W/r seconds later.

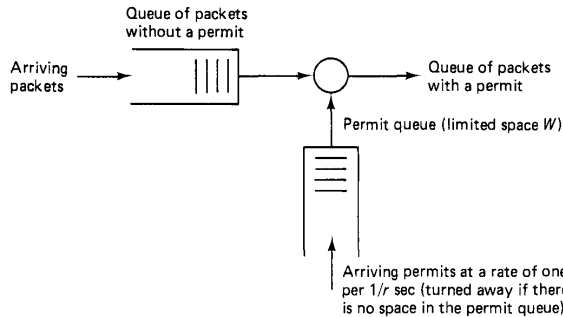


Figure 6.16 Leaky bucket scheme. To join the transmission queue, a packet must get a permit from the permit queue. A new permit is generated every $1/r$ seconds, where r is the desired input rate, as long as the number of permits does not exceed a given threshold.

The leaky bucket scheme does not necessarily preclude buffer overflow and does not guarantee an upper bound on packet delay in the absence of additional mechanisms to choose and implement the session rates inside the network; for some insight into the nature of such mechanisms, see Problem 6.19. However, with proper choice of the bucket parameters, buffer overflow and maximum packet delay can be reduced. In particular, the bucket size W is an important parameter for the performance of the leaky bucket scheme. If W is small, bursty traffic is delayed waiting for permits to become available ($W = 1$ resembles time-division multiplexing). If W is too large, long bursts of packets will be allowed into the network; these bursts may accumulate at a congested node downstream and cause buffer overflow.

The preceding discussion leads to the idea of dynamic adjustment of the bucket size. In particular, a congested node may send a special control message to the cor-

responding sources instructing them to shrink their bucket sizes. This is similar to the choke packet discussed in connection with dynamic adjustment of window sizes. The dynamic adjustment of bucket sizes may be combined with the dynamic adjustment of permit rates and merged into a single algorithm. Note, however, that in high-speed networks, the effectiveness of the feedback control messages from the congested nodes may be diminished because of relatively large propagation delays. Therefore, some predictive mechanism may be needed to issue control messages before congestion sets in.

Queueing analysis of the leaky bucket scheme. To provide some insight into the behavior of the leaky bucket scheme of Fig. 6.16, we give a queueing analysis. One should be very careful in drawing quantitative conclusions from this analysis, since some data sessions are much more bursty than is reflected in the following assumption of Poisson packet arrivals.

Let us assume the following:

1. Packets arrive according to a Poisson process with rate λ .
2. A permit arrives every $1/r$ seconds, but if the permit pool contains W permits, the arriving permit is discarded.

We view this system as a discrete-time Markov chain with states $0, 1, \dots$. (For an alternative formulation, see Problem 3.13.) The states $i = 0, 1, \dots, W$ correspond to $W - i$ permits available and no packets without permits waiting. The states $i = W + 1, W + 2, \dots$, correspond to $i - W$ packets without permits waiting and no permits available. The state transitions occur at the times $0, 1/r, 2/r, \dots$, just after a permit arrival. Let us consider the probabilities of k packet arrivals in $1/r$ seconds,

$$a_k = \frac{e^{-\lambda/r} (\lambda/r)^k}{k!}$$

It can be seen that the transition probabilities of the chain are

$$P_{0i} = \begin{cases} a_{i+1}, & \text{if } i \geq 1 \\ a_0 + a_1, & \text{if } i = 0 \end{cases}$$

and for $j \geq 1$,

$$P_{ji} = \begin{cases} a_{i-j+1}, & \text{if } j \leq i-1 \\ 0, & \text{otherwise} \end{cases}$$

(see Fig. 6.17). The global balance equations yield

$$p_0 = a_0 p_1 + (a_0 + a_1) p_0$$

$$p_i = \sum_{j=0}^{i+1} a_{i-j+1} p_j, \quad i \geq 1$$

These equations can be solved recursively. In particular, we have

$$p_1 = a_2 p_0 + a_1 p_1 + a_0 p_2$$

so by using the equation $p_1 = (1 - a_0 - a_1)p_0/a_0$, we obtain

$$p_2 = \frac{p_0}{a_0} \left(\frac{(1 - a_0 - a_1)(1 - a_1)}{a_0} - a_2 \right)$$

Similarly, we may use the global balance equation for p_2 , and the computed expressions for p_1 and p_2 , to express p_3 in terms of p_0 , and so on.

The steady-state probabilities can now be obtained by noting that

$$p_0 = \frac{r - \lambda}{ra_0}$$

To see this, note that the permit generation rate averaged over all states is $(1 - p_0 a_0)r$, while the packet arrival rate is λ . Equating the two rates, we obtain $p_0 = (r - \lambda)/(ra_0)$. The system is stable, that is, the packet queue stays bounded, if $\lambda < r$. The average delay for a packet to obtain a permit is

$$T = \frac{1}{r} \sum_{j=0}^{\infty} p_j \max\{0, j - W\} = \frac{1}{r} \sum_{j=W+1}^{\infty} p_j (j - W)$$

To obtain a closed-form expression for the average delay needed by a packet to get a permit and also to obtain a better model of the practical system, we modify the leaky bucket model slightly so that permits are generated on a per bit basis; this approximates the real situation where messages are broken up into small packets upon arrival at the source node. In particular, we assume that:

1. Credit for admission into the network is generated at a rate r bits/sec for transmission and the size of the bucket (*i.e.*, the maximum credit that can be saved) is W bits.
2. Messages arrive according to a Poisson process with rate λ , and the storage space for messages is infinite. Message lengths are independent and exponentially distributed with mean L bits.

Let $\mu = r/L$, so that $1/\mu$ is the mean time to transmit a message at the credit rate r . Also let $C = W/r$ be the time over which credit can be saved up. The state of the system can be described by the number of bits in the queue and the available amount of credit. At time t , let $X(t)$ be either the number of bits in the queue (if the queue is

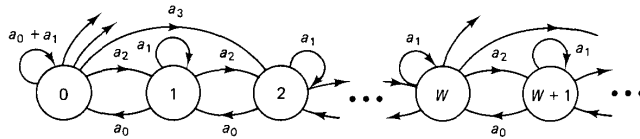


Figure 6.17 Transition probabilities of a discrete Markov chain model for the leaky bucket scheme. Here a_k is the probability of k packet arrivals in $1/r$ seconds. Note that this Markov chain is also the Markov chain for a slotted service $M/D/1$ queue.

nonempty) or minus the available credit. Thus, whenever a message consisting of x bits arrives, $X(t)$ increases by x [one of three things happens: the credit decreases by x ; the queue increases by x ; the credit decreases to 0 and the queue increases to $X(t) + x$ (this happens if $X(t) < 0$ and $X(t) + x > 0$)].

Letting $Y(t) = X(t) + C$, it can be seen that $Y(t)$ is the unfinished work in a fictitious $M/M/1$ queue with arrival rate λ messages/sec and service rate μ . An incoming bit is transmitted immediately if the size of the fictitious queue ahead of it is less than C and is transmitted C seconds earlier than in the fictitious queue otherwise. Focusing on the last bits of messages, we see that if T_i is the system delay in the fictitious queue for the i^{th} message, $\max\{0, T_i - C\}$ is the delay in the real queue. Using the theory of Section 3.3, it can be verified [a proof is outlined in Exercise 3.11(b)] that the steady-state distribution of the system time T_i at the fictitious queue is

$$P\{T_i \geq \tau\} = e^{-\tau(\mu-\lambda)}$$

Thus letting $T'_i = \max\{0, T_i - C\}$ be the delay of the i^{th} packet in the real queue, we obtain

$$P\{T'_i \geq \tau\} = \begin{cases} 1, & \text{if } \tau \leq 0 \\ e^{-(C+\tau)(\mu-\lambda)}, & \text{if } \tau > 0 \end{cases}$$

From this equation, the average delay of a packet in the real queue can be calculated as

$$T = \int_0^\infty P\{T'_i \geq \tau\} d\tau = \frac{1}{\mu - \lambda} e^{-C(\mu - \lambda)}$$

The preceding analysis can be generalized for the case where the packet lengths are independent but not exponentially distributed. In this case, the fictitious queue becomes an $M/G/1$ queue, and its system delay distribution can be estimated by using an exponential upper bound due to Kingman; see [Kle76], p. 45.

We finally note that the leaky bucket parameters affect not only the average packet delay to enter the network, which we have just analyzed. They also affect substantially the packet delay *after* the packet has entered the network. The relationship between this delay, the leaky bucket parameters, and the method for implementing the corresponding session rates at the network links is not well understood at present. For some interesting recent analyses and proposals see [Cru91a], [Cru91b], [PaG91a], [PaG91b], [Sas91], and Problem 6.19.

6.4 OVERVIEW OF FLOW CONTROL IN PRACTICE

In this section we discuss the flow control schemes of several existing networks.

Flow control in the ARPANET. Flow control in the ARPANET is based in part on end-to-end windows. The entire packet stream of each pair of machines (known as *hosts*) connected to the subnet is viewed as a “session” flowing on a logical pipe. For each such pipe there is a window of eight messages between the corresponding origin and destination subnet nodes. Each message consists of one or more packets up to a

maximum of eight. A transmitted message carries a number indicating its position in the corresponding window. Upon disposing of a message, the destination node sends back a special control packet (permit) to the origin, which in the ARPANET is called RFNM (ready for next message). The RFNM is also used as an end-to-end acknowledgment for error control purposes. Upon reception of an RFNM the origin node frees up a space in the corresponding window and is allowed to transmit an extra message. If an RFNM is not received after a specified time-out, the origin node sends a control packet asking the destination node whether the corresponding message was received. This protects against loss of an RFNM, and provides a mechanism for retransmission of lost messages.

There is an additional mechanism within the subnet for multipacket messages that ensures that there is enough memory space to reassemble these messages at their destination. (Packets in the ARPANET may arrive out of order at their destination.) Each multipacket message must reserve enough buffer space for reassembly at the receiver before it gets transmitted. This is done via a reservation message called REQALL (request for allocation) that is sent by the origin to the destination node. The reservation is granted when the destination node sends an ALL (allocate) message to the origin. When a long file is sent through the network, there is a long sequence of multipacket messages that must be transmitted. It would then be wasteful to obtain a separate reservation for each message. To resolve this problem, ALL messages are piggybacked on the returning RFNMs of multipacket messages, so that there is no reservation delay for messages after the first one in a file. If the reserved buffer space is not used by the origin node within a given timeout, it is returned to the destination via a special message. Single-packet messages do not need a reservation before getting transmitted. If, however, such a message finds the destination's buffers full, it is discarded and a copy is eventually retransmitted by the origin node after obtaining an explicit buffer reservation.

A number of improvements to the ARPANET scheme were implemented in late 1986 [Mal86]. First, the window size can be configured up to a maximum of 127; this allows efficient operation in the case where satellite links are used. Second, there can be multiple independent connections (up to 256) between two hosts, each with an independent window; this provides some flexibility in accommodating classes of traffic with different priorities and/or throughput needs. Third, an effort is made to improve the fairness properties of the current algorithm through a scheme that tries to allocate the available buffer space at each node fairly among all hosts. Finally, the reservation scheme for multipacket messages described above has been eliminated. Instead, the destination node simply reserves space for a multipacket message upon receiving the first packet of the message. If space is not available, the packet is discarded and is retransmitted after a time-out by the origin node.

The ARPANET flow control was supplemented by a rate adjustment scheme in 1989. Each node calculates an upper bound on flow rate for the origin-destination pairs routing traffic through it (the origin and the destination are subnetwork packet switches). This upper bound, also called a *ration*, is modified depending on the utilization of various critical resources of the node (processing power, transmission capacity of incident links, buffer space, etc.). The ration is adjusted up or down as the actual utilization of critical resources falls below or rises above a certain target utilization. The node rations are

broadcast to the entire network along with the routing update messages. Each origin then sets its flow rate to each destination to the minimum of the ratios of the nodes traversed by the current route to the destination (these nodes become known to the origin through the shortest path routing algorithm). The rates are implemented by using a leaky bucket scheme as discussed in Section 6.3.

Flow control in the TYMNET. Flow control in the TYMNET is exercised separately for each virtual circuit via a sequence of node-by-node windows. There is one such window per virtual circuit and link on the path of the virtual circuit. Each window is measured in bytes, and its size varies with the expected peak data rate (or throughput class) of the virtual circuit. Flow control is activated via the backpressure effect discussed in Section 6.2.2. Fairness is enhanced by serving virtual circuits on a link via a round-robin scheme. This is accomplished by combining groups of bytes from several virtual circuits into data link control frames. The maximum number of bytes for each virtual circuit in a frame depends on the level of congestion on the link and the priority class of the virtual circuit. Flow control permits are piggybacked on data frames, and are highly encoded so that they do not require much bandwidth.

Flow control in SNA. We recall from Section 5.1.2 that the counterpart in SNA of the OSI architecture network layer is called the path control layer, and that it includes a flow control function called virtual route control. The corresponding algorithm, known as the virtual route pacing scheme, is based on an end-to-end window for each virtual circuit (or virtual route in SNA terminology). An interesting aspect of this scheme is that the window size (measured in packets) is dynamically adjusted depending on traffic conditions. The minimum window size is usually equal to the number of links on the path, and the maximum window size is three times as large. Each packet header contains two bits that are set to zero at the source. An intermediate node on the path that is “moderately congested” sets the first bit to 1. If the node is “badly congested,” it sets both bits to 1. Otherwise, it does not change the bits. Upon arrival of the packet, the destination looks at the bits and increments the window size for no congestion, decrements the window size for moderate congestion, or sets the window size to the minimum for bad congestion.

Actually, the SNA scheme is a little different from the end-to-end window scheme focused on so far. In the main scheme discussed in Section 6.2.1, the window size is αW , and returning permits result in allocations of α packets each. We concentrated on the case where $\alpha = 1$. In SNA, however, $W = 1$ and α (rather than W) is adjusted between the minimum and maximum window size referred to earlier. Furthermore, the destination node can send a new α -packet allocation message (permit) upon reception of the first packet in an α -packet batch. Thus, full-speed transmission under light load conditions can be maintained even though $W = 1$.

In addition to virtual route control, SNA provides transport layer flow control on a session-by-session basis, which is known as session level pacing. This becomes necessary because a virtual route in SNA may contain several sessions that may have different flow control needs. The main idea here is to prevent the transmitting end of a session from sending data more quickly than the receiving end can process. Session-level

pacing is basically a window scheme whereby the transmitting end can introduce a new packet into the subnet upon receiving a permit (called a pacing response in SNA) from the other end. An interesting twist is that pacing responses can be delayed at the node through which the transmitting end of the session accesses the subnet. This provides the subnet with the means to control the rate at which it accepts data from external users.

Flow control in a Codex network. In one of the Codex networks, there is an end-to-end window associated with each virtual circuit. The window size is measured in bytes and is proportional to the number of links on the virtual circuit path and a nominal data rate for the virtual circuit. Returning permits are combined with end-to-end acknowledgments used for error control purposes, so the window scheme does not require additional communication overhead. Data link control (DLC) frames on each link are formed by concatenating groups of bytes from several virtual circuits that have traffic in queue. There is a maximum group size for each virtual circuit. Virtual circuits are served on a round-robin basis and this provides a natural mechanism for maintaining fairness.

There is also a rate control mechanism in the Codex network that is unrelated to the window scheme but plays a complementary role. The idea is to match the transmission rate of a virtual circuit along its incoming and outgoing links at each node on its path under heavy-load conditions. Without going into details (see [HSS86]), this is accomplished by adjusting the maximum number of bytes that a virtual circuit can insert in a DLC frame. As an example, suppose that a virtual circuit is slowed down on a given link due to heavy load. Then the start node of that link sends a special message to the preceding node along the virtual circuit's path, which proceeds to reduce the maximum number of bytes that a DLC frame can carry for this virtual circuit. One effect of this scheme is that when congestion develops downstream, the end-to-end window of a virtual circuit will not pile up entirely at the point of congestion, but rather will be spread more or less evenly along the virtual circuit path starting from the source node and ending at the point of congestion. This, combined with large memory space at the nodes, tends to make buffer overflow unlikely.

Flow control in the PARIS network. PARIS is an experimental high-speed packet switching system for integrated voice, video, and data communications. (PARIS is an acronym for Packetized Automatic Routing Integrated System.) PARIS uses virtual circuits and simple error control to expedite packet processing at the nodes, and achieve high packet throughput. Routes are calculated with an adaptive shortest path routing algorithm similar to the SPF algorithm of the ARPANET. Each link length is based on a measure of the load carried by the link, and is broadcast from time to time through the network using a spanning tree. Source routing is used; each packet carries the sequence of identities of the nodes that it has yet to cross.

Flow control is based on the leaky bucket scheme described in Section 6.3. A session requesting access to the network must provide the corresponding entry node with some information regarding its characteristics, such as average rate, peak rate, and average burst size. The entry node translates this information into an "equivalent capacity," which is a measure of bandwidth required by the session at each link along its path. The entry node then does a routing calculation to obtain a shortest path among

the paths that can accommodate the session. If no suitable path is found, the session is rejected. Otherwise the session is accepted and its leaky bucket parameters are determined based on its requirements and the current load of its path. The session may transmit more packets than the ones permitted by its leaky bucket. However, these extra packets are tagged as “red” and the network may discard them much more readily than other packets, which are called “green” and are given preferential access to buffer space. Even though the order of transmission of red and green packets is preserved, the algorithm is operated so that red packets have minimal effect on the loss rate of green packets. The leaky bucket parameters are kept constant during the lifetime of a session. It was found that dynamic adjustment of these parameters was of limited use because congestion information was largely outdated due to the high network speed and the relatively large propagation delays. We refer to [CiG88] and [CGK90] for further details.

Flow control in X.25. As discussed in Section 2.8.3, flow control at the X.25 packet level is implemented by means of a separate window for each virtual circuit. The default window size is 2, but it may be set as high as 7 or 127. Flow control is exercised in both directions [*i.e.*, from the user’s machine (DTE) to the entry point of the network (DCE), and also from DCE to DTE]. The implementation of the window strategy is reminiscent of DLC protocols. Each data packet contains a three-bit sequence number and a three-bit request number. (If the window size is 127, these numbers are seven bits long.) The sequence number gives the position of the packet within the sender’s window, while the request number is the number of the next packet expected to be received by the sender. Thus, the request number plays the role of a permit allowing the receiver to advance the corresponding window.

There is also a provision in X.25 for flow control between two DTEs communicating through the subnet. The X.25 packet format contains a special bit (called the D bit) that determines whether the piggyback number of a packet received by a DTE relates to the directly attached DCE ($D = 0$) or the remote DTE ($D = 1$). In the latter case, the request number serves as a permit for the advancement of a window maintained for flow control purposes between the two DTEs.

6.5 RATE ADJUSTMENT ALGORITHMS

In this section we look at two systematic formulations of the flow control problem to obtain algorithms for input rate adjustment. In the first approach (Section 6.5.1) we formulate an optimization problem that mathematically expresses the objective of striking a proper balance between maintaining high throughput and keeping average delay per packet at a reasonable level. In the second approach (Section 6.5.2) we emphasize fairness while maintaining average delay per packet at an acceptable level.

6.5.1 Combined Optimal Routing and Flow Control

We consider the possibility of combining routing and end-to-end flow control within the subnet by adjusting optimally *both* the routing variables and the origin–destination (OD)

pair input rates. A special case arises when routing is fixed and the only variables to be adjusted are the input rates—a problem of pure flow control.

We adopt a flow model similar to the one discussed in the context of optimal routing in Section 5.4, and we denote by r_w the input rate of an OD pair w . We first formulate a problem of adjusting routing variables together with the inputs r_w so as to minimize some “reasonable” cost function. We subsequently show that *this problem is mathematically equivalent to the optimal routing problem examined in Chapter 5* (in which r_w is fixed), and therefore the optimality conditions and algorithms given there are applicable.

If we minimize the cost function $\sum_{(i,j)} D_{ij}(F_{ij})$ of the routing problem with respect to both the path flows $\{x_p\}$ and the inputs $\{r_w\}$, we unhappily find that the optimal solution is $x_p = 0$ and $r_w = 0$ for all p and w . This indicates that the cost function should include a penalty for inputs r_w becoming too small and leads to the problem

$$\begin{aligned}
 & \text{minimize} \quad \sum_{(i,j)} D_{ij}(F_{ij}) + \sum_{w \in W} e_w(r_w) \\
 & \text{subject to} \quad \sum_{p \in P_w} x_p = r_w, \quad \text{for all } w \in W \\
 & \quad \quad \quad x_p \geq 0, \quad \text{for all } p \in P_w, w \in W \\
 & \quad \quad \quad 0 \leq r_w \leq \bar{r}_w, \quad \text{for all } w \in W
 \end{aligned} \tag{6.2}$$

Here the minimization is to be carried out jointly with respect to $\{x_p\}$ and $\{r_w\}$. The given values $\bar{r}_w$ represent the desired input by OD pair w (i.e., the offered load for w , defined as the input for w that would result if no flow control were exercised). As before, F_{ij} is the total flow on link (i, j) (i.e., the sum of all path flows traversing the link). The functions e_w are of the form shown in Fig. 6.18 and provide a penalty for throttling the inputs r_w . They are monotonically decreasing on the set of positive numbers $(0, \infty)$, and tend to ∞ as r_w tends to zero. We assume that their first and second derivatives, e'_w and e''_w , exist on $(0, \infty)$ and are strictly negative and positive, respectively. An interesting class of functions e_w is specified by the following formula for their first derivative:

$$e'_w(r_w) = - \left(\frac{a_w}{r_w} \right)^{b_w}, \quad \text{for } a_w \text{ and } b_w \text{ given positive constants} \tag{6.3}$$

As explained later in this section, the parameters a_w and b_w influence the optimal mag-

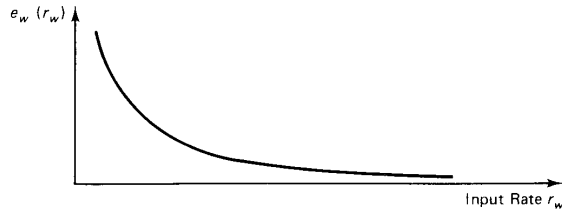


Figure 6.18 Typical form of penalty function for throttling the input rate r_w .

nitude of input r_w and the priority of OD pair w , respectively. The functions D_{ij} are defined and are monotonically increasing with positive second derivative in an interval $[0, C_{ij}]$, where C_{ij} is either finite, representing link capacity, or is equal to ∞ .

The value of the preceding formulation is enhanced if we adopt a broader view of w and consider it as a *class of sessions sharing the same set of paths* P_w . This allows different priorities (*i.e.*, different functions e_w) for different classes of sessions even if they share the same paths. A problem where P_w consists of a single path for each w can also be considered. This is a problem of pure flow control, namely, choosing the optimal fraction of the desired input flow of each session class that should be allowed into the network.

We now show that the combined routing and flow control problem of Eq. (6.2) is mathematically equivalent to a routing problem of the type considered in Section 5.4. Let us introduce a new variable y_w for each $w \in W$ via the equation

$$y_w = \bar{r}_w - r_w \quad (6.4)$$

We may view y_w as the *overflow* (the portion of $\bar{r}_w$ blocked out of the network), and consider it as a flow on an *overflow link* directly connecting the origin and destination nodes of w as shown in Fig. 6.19. If we define a new function E_w by

$$E_w(y_w) = e_w(\bar{r}_w - y_w) \quad (6.5)$$

the combined routing and flow control problem of Eq. (6.2) becomes, in view of the definition $y_w = \bar{r}_w - r_w$,

$$\begin{aligned}
 & \text{minimize} \quad \sum_{(i,j)} D_{ij}(F_{ij}) + \sum_{w \in W} E_w(y_w) \\
 & \text{subject to} \quad \sum_{p \in P_w} x_p + y_w = \bar{r}_w, \quad \text{for all } w \in W \\
 & \quad \quad \quad x_p \geq 0, \quad \text{for all } p \in P_w, w \in W \\
 & \quad \quad \quad y_w \geq 0, \quad \text{for all } w \in W
 \end{aligned} \quad (6.6)$$

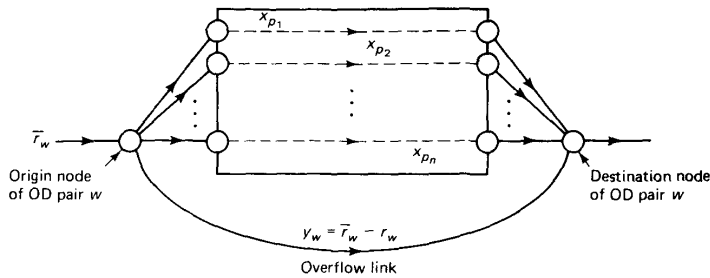


Figure 6.19 Mathematical equivalence of the flow control problem with an optimal routing problem based on the introduction of an artificial overflow link for each OD pair w . The overflow link carries the rejected traffic (*i.e.*, the difference between desired and accepted input flow, $\bar{r}_w - r_w$). The cost of the overflow link is obtained from the cost ϵ_w for throttling the input by a change of variable.

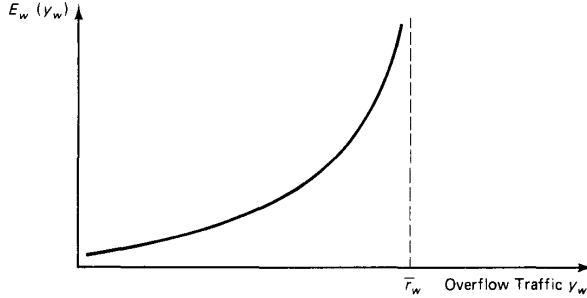


Figure 6.20 Cost function for the overflow link. The cost $E_w(y_w)$ for overflow traffic y_w equals the cost $e_w(r_w)$ for input traffic $r_w = \bar{r}_w - y_w$.

The form of the function E_w of Eq. (6.5) is shown in Fig. 6.20. If $e_w(r_w) \rightarrow \infty$ as $r_w \rightarrow 0$ (i.e., there is “infinite penalty” for completely shutting off the class of sessions w), then $E_w(y_w) \rightarrow \infty$ as the overflow y_w approaches its maximum value—the maximum input $\bar{r}_w$. Thus, E_w may be viewed as a “delay” function for the overflow link, and $\bar{r}_w$ may be viewed as the “capacity” of the link.

It is now clear that the problem of Eq. (6.6) is of the optimal routing type considered in Section 5.4, and that the algorithms and optimality conditions given in Sections 5.5 to 5.7 apply. In particular, the application of the shortest path optimality condition of Section 5.5 (see also Problem 5.24) yields the following result:

A feasible set of path flows $\{x_p^*\}$ and inputs $\{r_w^*\}$ is optimal for the combined routing and flow control problem of Eq. (6.2) if and only if the following conditions hold for each $p \in P_w$ and $w \in W$:

$$x_p^* > 0 \quad \Rightarrow \quad d_p^* \leq d_{p'}^*, \quad \text{for all } p' \in P_w, \text{ and } d_p^* \leq -e'_w(r_w^*) \quad (6.7a)$$

$$r_w^* < \bar{r}_w \quad \Rightarrow \quad -e'_w(r_w^*) \leq d_p^*, \quad \text{for all } p \in P_w \quad (6.7b)$$

where d_p^* is the first derivative length of path p [$d_p^* = \sum_{(i,j) \in p} D'_{ij}(F_{ij}^*)$], and F_{ij}^* is the total flow of link (i, j) corresponding to $\{x_p^*\}$.

Note that the optimality conditions (6.7) depend only on the derivatives of the functions D_{ij} and e_w . This means that arbitrary constants could be added to D_{ij} and e_w without affecting the optimum. Note also from the optimality condition (6.7b) that the optimum point r_w^* is independent of $\bar{r}_w$ as long as $\bar{r}_w > r_w^*$. This is a desirable feature for a flow control strategy, preventing sessions that are being actively flow controlled from attempting to increase their share of the resources by increasing their demands. A simple example illustrates the optimality conditions (6.7).

Example 6.3

Consider the situation in Fig. 6.21 involving a single link connecting origin and destination. The cost function is

$$\frac{r}{C - r} + \frac{a}{r}$$

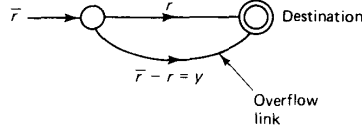


Figure 6.21 Example problem involving an origin and a destination connected by a single link.

where the first term represents a penalty due to large delay [cf. the term $D_{ij}(F_{ij})$ in Eq. (6.2)], and the second term represents a penalty for small throughput [cf. the term $\epsilon_w(r_w)$ in Eq. (6.2)]. The constant C is the capacity of the link, while the parameter a is a positive weighting factor. The equivalent routing problem [cf. Eq. (6.6)] is

$$\begin{aligned} & \text{minimize } \frac{r}{C-r} + \frac{a}{\bar{r}-y} \\ & \text{subject to } r+y=\bar{r}, \quad r \geq 0, \quad y \geq 0 \end{aligned}$$

where $y = \bar{r} - r$ represents the amount of offered load rejected by flow control (equivalently, the flow on the fictitious overflow link). The optimality conditions (6.7) show that there will be no flow control ($y = 0, r = \bar{r}$) if

$$\frac{C}{(C-\bar{r})^2} < \frac{a}{\bar{r}^2}$$

(i.e., if the first derivative length of the overflow link exceeds the first derivative length of the regular link). Equivalently, there will be no flow control if

$$\bar{r} < C \frac{\sqrt{a}}{\sqrt{a} + \sqrt{C}}$$

According to the optimality condition [Eq. (6.7)], when there is flow control ($y > 0, r < \bar{r}$), the two first derivative lengths must be equal, that is,

$$\frac{C}{(C-r)^2} = \frac{a}{(\bar{r}-y)^2}$$

Substituting $y = \bar{r} - r$ and working out the result yields

$$r = C \frac{\sqrt{a}}{\sqrt{a} + \sqrt{C}}$$

The solution as a function of offered load is shown in Fig. 6.22. Note that the throughput is independent of the offered load beyond a certain point, as discussed earlier. The maximum throughput $C\sqrt{a}/(\sqrt{a} + \sqrt{C})$ can be regulated by adjustment of the parameter a and tends to the capacity C as $a \rightarrow \infty$.

The meaning of the parameters a_w and b_w in the cost function specified by the formula [cf. Eq. (6.3)]

$$c'_w(r_w) = - \left(\frac{a_w}{r_w} \right)^{b_w}$$

can now be clarified in the light of the optimality condition (6.7b). Consider two distinct classes of sessions, w_1 and w_2 , sharing the same paths ($P_{w_1} = P_{w_2}$). Then the condition

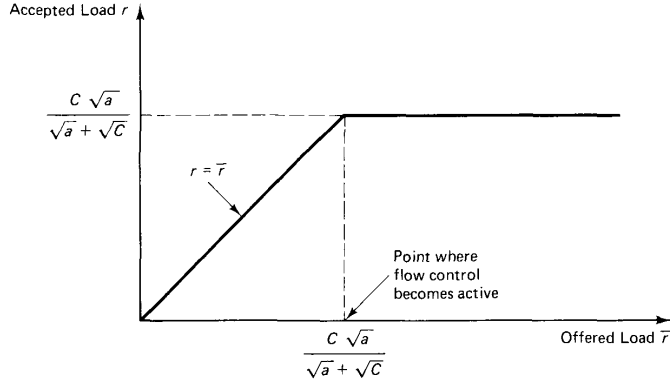


Figure 6.22 Optimal accepted load as a function of offered load in the flow control example. Flow control becomes active when the offered load exceeds a threshold level that depends on the weighting factor a .

(6.7b) implies that at an optimal solution in which both classes of sessions are throttled ($r_{w_1}^* < \bar{r}_{w_1}$, $r_{w_2}^* < \bar{r}_{w_2}$)

$$-e'_{w_1}(r_{w_1}^*) = -e'_{w_2}(r_{w_2}^*) = \min_{p \in P_{w_1}} \{d_p^*\} = \min_{p \in P_{w_2}} \{d_p^*\} \quad (6.8)$$

If e'_{w_1} and e'_{w_2} are specified by parameters a_{w_1}, b_{w_1} and a_{w_2}, b_{w_2} as in Eq. (6.3), it can be seen that:

1. If $b_{w_1} = b_{w_2}$, then

$$\frac{r_{w_1}^*}{r_{w_2}^*} = \frac{a_{w_1}}{a_{w_2}}$$

and it follows that the parameter a_w influences the optimal, relative input rate of the session class w .

2. If $a_{w_1} = a_{w_2} = a$ and $b_{w_1} < b_{w_2}$ (see Fig. 6.23), the condition (6.8) specifies that when the input flows must be made small ($r_{w_1}^*, r_{w_2}^* < a$), the session class w_2 (the one with higher parameter b_w) will be allowed a larger input. It follows that the parameter b_w influences the relative priority of the session class w under heavy load conditions.

6.5.2 Max-Min Flow Control

One of the most difficult aspects of flow control is treating all sessions fairly when it is necessary to turn traffic away from the network. Fairness can be defined in a number of different ways, but one intuitive notion of fairness is that any session is entitled to as much network use as is any other session. Figure 6.24 clarifies some of the ambiguities in this notion. One session flows through the tandem connection of all links, and each

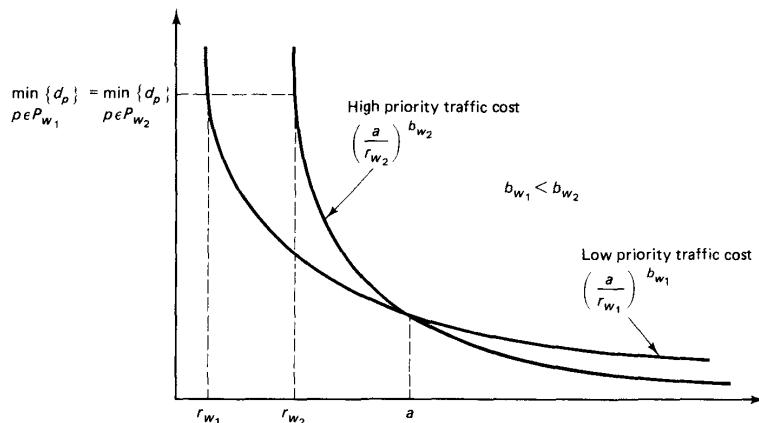


Figure 6.23 Incorporating priorities of session classes in the cost function of the flow control formulation. The session class with larger b_w will be throttled less under heavy load conditions.

other session goes through only one link. It is plausible to limit sessions 0, 1, and 2 to a rate of $1/3$ each, since this gives each of these sessions as much rate as the others. It would be rather pointless, however, to restrict session 3 to a rate of $1/3$. Session 3 might better be limited to $2/3$, since any lower limit would waste some of the capacity of the rightmost link without benefitting sessions 0, 1, or 2, and any higher limit would be unfair because it would further restrict session 0.

This example leads to the idea of maximizing the network use allocated to the sessions with the minimum allocation, thus giving rise to the term *max-min flow control*. After these most poorly treated sessions are given the greatest possible allocation, there might be considerable latitude left for choosing allocations for the other sessions. It is then reasonable to maximize the allocation for the most poorly treated of these other sessions, and so forth, until all allocations are specified. An alternative way to express this intuition, which turns out to be equivalent to the above, is to maximize the allocation of each session i subject to the constraint that an incremental increase in i 's allocation does not cause a decrease in some other session's allocation that is already as small as i 's or smaller.

We assume a directed graph $G = (\mathcal{N}, \mathcal{A})$ for the network and a set of sessions P using the network. Each session p has an associated fixed path in the network. We use p

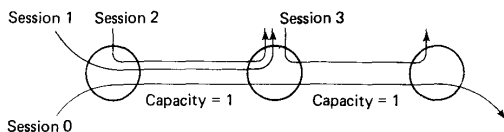


Figure 6.24 The fair solution is to give to sessions 0, 1, and 2 a rate of $1/3$ each and to give session 3 a rate of $2/3$ to avoid wasting the extra capacity available at the rightmost link.

both to refer to the session and to its path (if several sessions use the same path, several indices p refer to the same path). Thus, in our model we assume a fixed, single-path routing method.

We denote by r_p the allocated rate for session p . The allocated flow on link a of the network is then

$$F_a = \sum_{\substack{\text{all sessions } p \\ \text{crossing link } a}} r_p \quad (6.9)$$

Letting C_a be the capacity of link a , we have the following constraints on the vector $r = \{r_p \mid p \in P\}$ of allocated rates:

$$r_p \geq 0, \quad \text{for all } p \in P \quad (6.10a)$$

$$F_a \leq C_a, \quad \text{for all } a \in \mathcal{A} \quad (6.10b)$$

A vector r satisfying these constraints is said to be *feasible*.

A vector of rates r is said to be *max-min fair* if it is feasible and for each $p \in P$, r_p cannot be increased while maintaining feasibility without decreasing $r_{p'}$ for some session p' for which $r_{p'} \leq r_p$. (More formally, r is max-min fair if it is feasible, and for each $p \in P$ and feasible $\bar{r}$ for which $r_p < \bar{r}_p$, there is some p' with $r_p \geq r_{p'}$ and $r_{p'} > \bar{r}_{p'}$.) Our problem is to find a rate vector that is max-min fair.

Given a feasible rate vector r , we say that link a is a *bottleneck link* with respect to r for a session p crossing a if $F_a = C_a$ and $r_p \geq r_{p'}$ for all sessions p' crossing link a . Figure 6.25 provides an example of a max-min fair rate vector and illustrates the concept of a bottleneck link. In this example, every session has a bottleneck link. It turns out that this property holds in general as shown in the following proposition:

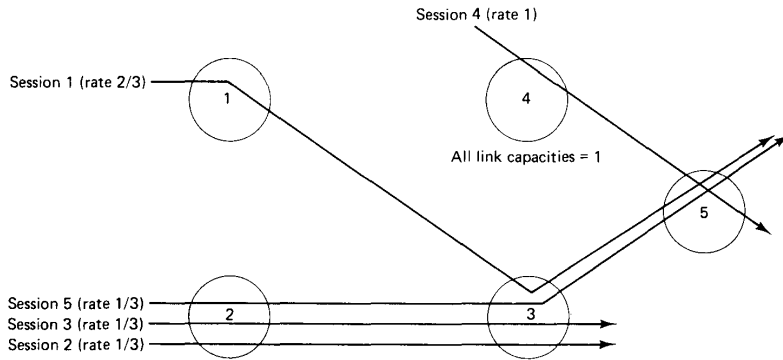


Figure 6.25 Max-min fair solution for an example network. The bottleneck links of sessions 1, 2, 3, 4, and 5 are (3,5), (2,3), (2,3), (4,5), and (2,3), respectively. Link (3,5) is not a bottleneck link for session 5 since sessions 1 and 5 share this link and session 1 has a larger rate than session 5. Link (1,3) is not a bottleneck link of any session since it has an excess capacity of 1/3 in the fair solution.

Proposition. A feasible rate vector r is max-min fair if and only if each session has a bottleneck link with respect to r .

Proof: Suppose that r is max-min fair and, to arrive at a contradiction, assume that there exists a session p with no bottleneck link. Then, for each link a crossed by p for which $F_a = C_a$, there must exist a session $p_a \neq p$ such that $r_{p_a} > r_p$; thus the quantity

$$\delta_a = \begin{cases} C_a - F_a, & \text{if } F_a < C_a \\ r_{p_a} - r_p, & \text{if } F_a = C_a \end{cases}$$

is positive. Therefore, by increasing r_p by the minimum δ_a over all links a crossed by p , while decreasing by the same amount the rates of the sessions r_{p_a} of the links a crossed by p with $F_a = C_a$, we maintain feasibility without decreasing the rate of any session p' with $r_{p'} \leq r_p$; this contradicts the max-min fairness property of r .

Conversely, assume that each session has a bottleneck link with respect to the feasible rate vector r . Then, to increase the rate of any session p while maintaining feasibility, we must decrease the rate of some session p' crossing the bottleneck link a of p (because we have $F_a = C_a$ by the definition of a bottleneck link). Since $r_{p'} \leq r_p$ for all p' crossing a (by the definition of a bottleneck link), the rate vector r satisfies the requirement for max-min fairness. **Q.E.D.**

Next, we give a simple algorithm for computing max-min fair rate vectors. The idea of the algorithm is to start with an all-zero rate vector and to increase the rates on all paths together until $F_a = C_a$ for one or more links a . At this point, each session using a saturated link (*i.e.*, a link with $F_a = C_a$) has the same rate at every other session using that link. Thus, these saturated links serve as bottleneck links for all sessions using them.

At the next step of the algorithm, all sessions not using the saturated links are incremented equally in rate until one or more new links become saturated. Note that the sessions using the previously saturated links might also be using these newly saturated links (at a lower rate). The newly saturated links serve as bottleneck links for those sessions that pass through them but do not use the previously saturated links. The algorithm continues from step to step, always equally incrementing all sessions not passing through any saturated link; when all sessions pass through at least one saturated link, the algorithm stops.

In the algorithm, as stated more precisely below, A^k denotes the set of links not saturated at the beginning of step k , and P^k denotes the set of sessions not passing through any saturated link at the beginning of step k . Also, n_a^k denotes the number of sessions that use link a and are in P^k . Note that this is the number of sessions that will share link a 's yet unused capacity. Finally, $\tilde{r}^k$ denotes the increment of rate added to all of the sessions in P^k at the k^{th} step.

Initial conditions: $k = 1$, $F_a^0 = 0$, $r_p^0 = 0$, $P^1 = P$, and $A^1 = A$.

1. $n_a^k :=$ number of sessions $p \in P^k$ crossing link a
2. $\tilde{r}^k := \min_{a \in A^k} (C_a - F_a^{k-1}) / n_a^k$

3. $r_p^k := \begin{cases} r_p^{k-1} + \tilde{r}^k & \text{for } p \in P^k \\ r_p^{k-1} & \text{otherwise} \end{cases}$
4. $F_a^k := \sum_{p \text{ crossing } a} r_p^k$
5. $A^{k+1} := \{a \mid C_a - F_a^k > 0\}$
6. $P^{k+1} := \{p \mid p \text{ does not cross any link } a \in A^{k+1}\}$
7. $k := k + 1$
8. If P^k is empty, then stop; else go to 1.

At each step k , an equal increment of rate is added to all sessions not yet passing through a saturated link, and thus at each step k , all sessions in P^k have the same rate. All sessions in P^k passing through a link that saturates in step k have at least as much rate as any other session on that link and hence are bottlenecked by that link. Thus upon termination of the algorithm, each session has a bottlenecked link, and by the proposition shown earlier, the final rate vector is max-min fair.

Example 6.4

Consider the problem of max-min fair allocation for the five sessions and the network shown in Fig. 6.25. All links have a capacity of one.

Step 1: All sessions get a rate of $1/3$. Link (2,3) is saturated at this step, and the rate of the three sessions (2, 3, and 5) that go through it is fixed at $1/3$.

Step 2: Sessions 1 and 4 get an additional rate increment of $1/3$ for a total of $2/3$. Link (3,5) is saturated, and the rate of session 1 is fixed at $2/3$.

Step 3: Session 4 gets an additional rate increment of $1/3$ for a total of 1. Link (4,5) is saturated, and the rate of session 4 is fixed at 1. Since now all sessions go through at least one saturated link, the algorithm terminates with the max-min fair solution shown in Fig. 6.25.

Several generalizations can be made to the basic approach described above. First, to keep the flow on each link strictly below capacity, we can replace C_a in the algorithm with some fixed fraction of C_a . Next, we can consider ways to assign different priorities to different kinds of traffic and to make these priorities sensitive to traffic levels. If $b_p(r_p)$ is an increasing function representing the priority of p at rate r_p , the max-min fairness criterion can be modified as follows: For each p , maximize r_p subject to the constraint that any increase in r_p would cause a decrease of $r_{p'}$ for some p' satisfying $b_{p'}(r_{p'}) \leq b_p(r_p)$. It is easy to modify the algorithm above to calculate fair rates with such priorities. Another twist on the same theme is to require that each r_p be upper bounded by $C_a - F_a$ on each link used by path p ; the rationale is to maintain enough spare capacity on each link to be able to add an extra session. The problem here, however, is that as the number of sessions on a link grow, the reserve capacity shrinks to zero and the buffer requirement grows with the number of sessions, just like the corresponding growth using windows. This difficulty can be bypassed by replacing the constraint $r_p \leq C_a - F_a$ by a constraint of the form $r_p \leq (C_a - F_a)q_a$, where q_a is a positive scalar factor depending on the number of sessions crossing link a (see Problem 6.18).

There has been a great deal of work on distributed algorithms that dynamically adjust the session rates to maintain max-min fairness as the sessions change. A repre-

sentative algorithm [Hay81] will help in understanding the situation. In this algorithm, v_a^k represents an estimate of the maximum allowable session rate on link a at the k^{th} iteration, F_a^k is the allocated flow on link a corresponding to the rates r_p^k , and n_a is the number of sessions using link a . The typical iteration of the algorithm is

$$v_a^{k+1} = v_a^k + \frac{C_a - F_a^k}{n_a}$$

$$r_p^{k+1} = \min_{\substack{\text{links } a \text{ on} \\ \text{path } p}} v_a^{k+1}$$

Each iteration can be implemented in a distributed way by first passing the values v_a^{k+1} from the links a to the sessions using those links, and then passing the values r_p^{k+1} from the sessions p to the links used by these sessions. The major problem is that the allocated rates can wander considerably before converging, and link flows can exceed capacity temporarily. There are other algorithms that do not suffer from this difficulty ([GaB84b] and [Mos84]). In the algorithm of [GaB84b], the session rates are iterated at the session sources according to

$$r_p^{k+1} = \min_{\substack{\text{links } a \text{ on} \\ \text{path } p}} \left\{ r_p^k + \frac{C_a - F_a^k - r_p^k}{1 + n_a} \right\}$$

This algorithm aims at solving the max-min fair flow control problem subject to the additional constraints $r_p \leq C_a - F_a$ for each session p and link a crossed by p (see Problem 6.18). By adding the relation

$$r_p^{k+1} \leq r_p^k + \frac{C_a - F_a^k - r_p^k}{1 + n_a}$$

over all sessions p crossing link a , it is straightforward to verify that

$$F_a^{k+1} \leq \frac{n_a C_a}{1 + n_a}$$

so the link flows are strictly less than the link capacities at all times. Extensions of this algorithm to accommodate session priorities are possible (see [GaB84b]).

Max-min fair rates may be implemented using leaky bucket schemes as discussed in Section 6.3. Another approach to implementing max-min fair rates has been explored in [HaG86]. This approach avoids both the problem of communication in a distributed algorithm and also the problem of the ambiguity in the meaning of rate for interactive traffic. The idea is very simple: Serve different sessions on each link in the network on a round-robin basis. This means that *if* a session always has a packet waiting at a node when its turn comes up on the outgoing link, that session gets as much service as any other session using that link. Thus, to achieve max-min fairness, it is only necessary that each session always has a packet waiting at its bottleneck link. In fact, it can be shown that by using node-by-node windowing with a large enough window, a session that always has something to send will always have a packet waiting at its bottleneck link (see [Hah86]).

SUMMARY

In this chapter we identified the major flow control objectives as limiting average delay and buffer overflow within the subnet, and treating sessions fairly. We reviewed the major flow control methods, and we saw that the dominant strategies in practice are based on windows and input rate control. Window strategies combine low overhead with fast reaction to congestion but have some limitations, particularly for networks with relatively large propagation delay. Window strategies are also unsuitable for sessions that require a minimum guaranteed rate. Rate adjustment schemes are usually implemented by means of leaky buckets. However, there remain a number of questions regarding the choice and adjustment of the leaky bucket parameters in response to traffic conditions. We also described two theoretical input rate adjustment schemes. The first scheme extends the optimal routing methodology of Chapter 5 to the flow control context, and combines routing and flow control into a single algorithm. The second scheme assumes fixed, single-path routing for each session, and focuses on maintaining flow control fairness.

NOTES, SOURCES, AND SUGGESTED READING

Section 6.1. Extensive discussions of flow control can be found in the April 1981 special issue of the IEEE Transactions on Communications. An informative survey is [GeK80]. A more recent survey [GeK89] discusses flow control in local area networks.

Section 6.2. The special difficulties of window flow control along satellite links are discussed in [GrB83].

Section 6.3. The leaky bucket scheme is discussed in [Tur86] and [Zha89]. Several rate control schemes are discussed in [Gol90a], [Gol90b], [ELL90a], and [ELL90b].

Section 6.4. Flow control in the ARPANET is described in several sources (*e.g.*, [Kle76] and [KlO77]). The more recent rate control scheme is discussed in [RFS91] and [ELS90]. The TYMNET flow control system is discussed in [Tym81]. For further discussion on SNA, see [Ahu79] and [GeY82], and for more on the Codex network, see [HSS86]. The PARIS flow control scheme and other PARIS protocols are discussed in [CiG88], and [CGK90].

Section 6.5. The combined optimal routing and flow control was formulated in [GaG80] and [Gol80]. For additional material on this subject, see [Ibe81] and [Gaf82]. A simulation together with a discussion of practical implementation schemes is given in [ThC86]. Flow control based on adjusting the rate of encoding of digitized voice has been considered in [BGS80]. The material on fair flow control is based on [Hay81]. For related work, see [Jaf81], [GaB84b], [Mos84], [Hah86], and [HaG86].

PROBLEMS

- 6.1** Consider a window flow controlled virtual circuit going over a satellite link. All packets have a transmission time of 5 msec. The round-trip processing and propagation delay is 0.5 sec. Find a lower bound on the window size for the virtual circuit to be able to achieve maximum speed transmission when there is no other traffic on the link.
- 6.2** Suppose that the virtual circuit in Problem 6.1 goes through a terrestrial link in addition to the satellite link. The transmission time on the terrestrial link is 20 msec, and the processing and propagation delay is negligible. What is the maximum transmission rate in packets/sec that can be attained for this virtual circuit assuming no flow control? Find a lower bound to an end-to-end window size that will allow maximum transmission rate assuming no other traffic on the links. Does it make a difference whether the terrestrial link is before or after the satellite link?
- 6.3** Suppose that node-by-node windows are used in the two-link system of Problem 6.2. Find lower bounds on the window size required along each link in order to achieve maximum speed transmission, assuming no other traffic on the links.
- 6.4** The three-node network of Fig. 6.26 contains only one virtual circuit from node 1 to 3, and uses node-by-node windows. Each packet transmission on link (1,2) takes 1 sec, and on link (2,3) takes 2 sec; processing and propagation delay is negligible. Permits require 1 sec to travel on each link. There is an inexhaustible supply of packets at node 1. The system starts at time 0 with W permits at node 1, W permits at node 2, and no packets stored at nodes 2 and 3. For $W = 1$, find the times, from 0 to 10 sec at which a packet transmission starts at node 1 and node 2. Repeat for $W = 2$.
- 6.5** In the discussion of node-by-node window flow control, it was assumed that node i can send a permit back to its predecessor ($i - 1$) once it releases a packet to the DLC of link ($i, i + 1$). The alternative is for node i to send the permit when it receives the DLC acknowledgment that the packet has been received correctly at node $i + 1$. Discuss the relative merits of the two schemes. Which scheme requires more memory? What happens when link ($i, i + 1$) is a satellite link?
- 6.6** Consider a combined optimal routing and flow control problem involving the network of Fig. 6.27 (cf. Section 6.4.1). The cost function is

$$(F_{13})^2 + (F_{23})^2 + (F_{34})^2 + \frac{a}{r_1} + \frac{a}{r_2}$$

where a is a positive scalar parameter. Find the optimal values of the rates r_1 and r_2 for each value of a .

- 6.7** Consider six nodes arranged in a ring and connected with unit capacity bidirectional links ($i, i + 1$), $i = 1, 2, 3, 4, 5$, and (6,1). There are two sessions from nodes 1, 2, 3, 4, and 5 to node 6, one in the clockwise and one in the counterclockwise direction. Similarly, there are two sessions from nodes 2, 3, and 4 to node 5, and two sessions from node 3 to node 4. Find the max-min fair rates for these sessions.

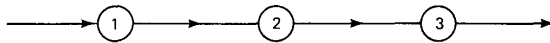


Figure 6.26

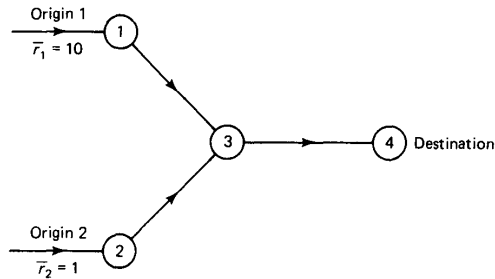


Figure 6.27

- 6.8 Suppose that the definition of a fair rate vector of Section 6.4.2 is modified so that, in addition to Eq. (6.10), there is a constraint $r_p \leq b_p$ that the rate of a session p must satisfy. Here b_p is the maximum rate at which session p is capable of transmitting. Modify the algorithm given in Section 6.4.2 to solve this problem. Find the fair rates for the example given in this section when $b_p = 1$, for $p = 2, 4, 5$ and $b_p = 1/4$, for $p = 1, 3$. *Hint:* Add a new link for each session.
- 6.9 The purpose of this problem is to illustrate how the relative throughputs of competing sessions can be affected by the priority rule used to serve them. Consider two sessions A and B sharing the first link L of their paths as shown in Fig. 6.28. Each session has an end-to-end window of two packets. Permits for packets of A and B arrive d_A and d_B seconds, respectively, after the end of transmission on link L . We assume that d_A is exponentially distributed with mean of unity, while (somewhat unrealistically) we assume that $d_B = 0$. Packets require a transmission time on L which is exponentially distributed with mean of unity. Packet transmission times and permit delays are all independent. We assume that a new packet for A (B) enters the transmission queue of L immediately upon receipt of a permit for A (B).
- (a) Suppose that packets are transmitted on L on a first-come first-serve basis. Argue that the queue of L can be represented by a Markov chain with the 10 queue states BB, BBA, BAB, ABB, BBAA, BABA, BAAB, ABBA, ABAB, and AABB (each letter stands for a packet of the corresponding session). Show that all states have equal steady-state probability and that the steady-state throughputs of sessions A and B in packets/sec are 0.4 and 0.6, respectively.
- (b) Now suppose that transmissions on link L are scheduled on a round-robin basis. Between successive packet transmissions for session B , session A transmits one packet if it has one waiting. Draw the state transition diagram of a five-state Markov chain which

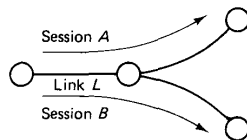


Figure 6.28

- models the queue of L . Solve for the equilibrium state probabilities. What are the steady-state throughputs of sessions A and B?
- (c) Finally, suppose that session A has nonpreemptive priority over session B at link L . Between successive packet transmissions for session B, session A transmits as many packets as it can. Session B regains control of the link only when A has nothing to send. Draw the state transition diagram of a five-state Markov chain which models L and its queue. Solve for the equilibrium state probabilities. What are the steady-state throughputs of sessions A and B?
- 6.10** Consider a window between an external source and the node by which that source is connected to the subnet. The source generates packets according to a Poisson process of rate λ . Each generated packet is accepted by the node if a permit is available. If no permit is available, the packet is discarded, never to return. When a packet from the given source enters the DLC unit of an outgoing link at the node, a new permit is instantaneously sent back to the source. The source initially has two permits and the window size is 2. Assume that the other traffic at the node imposes a random delay from the time a packet is accepted at the node to the time it enters the DLC unit. Specifically, assume that in any arbitrarily small interval δ , there is a probability $\mu\delta$ that a waiting packet from the source (or the first of two waiting packets) will enter the DLC; this event is independent of everything else.
- Construct a Markov chain for the number of permits at the source.
 - Find the probability that a packet generated by the source is discarded.
 - Explain whether the probability in part (b) would increase or decrease if the propagation and transmission delay from source to node and the reverse were taken into account.
 - Suppose that a buffer of size k is provided at the source to save packets for which no permit is available; when a permit is received, one of the buffered packets is instantly sent to the network node. Find the new Markov chain describing the system, and find the probability that a generated packet finds the buffer full.
- 6.11** Consider a network with end-to-end window flow control applied to each virtual circuit. Assume that the data link control operates perfectly and that packets are never thrown away inside the network; thus, packets always arrive at the destination in the order sent, and all packets eventually arrive.
- Suppose that the destination sends permits in packets returning to the source; if no return packet is available for some time-out period, a special permit packet is sent back to the source. These permits consist of the number modulo m of the next packet awaited by the destination. What is the restriction on the window size W in terms of the modulus m ? Why?
 - Suppose next that the permits contain the number modulo m of *each* of the packets in the order received since the last acknowledgment was sent. Does this change your answer to part (a)? Explain.
 - Is it permissible for the source to change the window size W without prior agreement from the destination? Explain.
 - How can the destination reduce the effective window size below the window size used by the source without prior agreement from the source? (By effective window size we mean the maximum number of packets for the source-destination pair that can be in the network at one time.)
- 6.12** Consider a node-by-node window scheme. In an effort to reduce the required buffering, the designers associated the windows with destinations rather than with virtual circuits. Assume that all virtual circuit paths to a given destination j use a directed spanning tree so that each

node $i \neq j$ has only one outgoing link for traffic to that destination. Assume that each node $i \neq j$ originally has permits for two packets for j that can be sent over the outgoing link to j . Each time that node i releases a packet for j into its DLC unit on the outgoing link, it sends a new permit for one packet back over the incoming link over which that packet arrived. If the packet arrived from a source connected to i , the permit is sent back to the source (each source also originally has two permits for the given destination).

- (a) How many packet buffers does node i have to reserve for packets going to destination j to guarantee that every arriving packet for j can be placed in a buffer?
 - (b) What are the pros and cons of this scheme compared with the conventional node-by-node window scheme on a virtual circuit basis?
- 6.13** Consider a network using node-by-node windows for each virtual circuit. Describe a strategy for sending and reclaiming permits so that buffer overflow never occurs regardless of how much memory is available for packet storage at each node and of how many virtual circuits are using each link. *Hint*: You need to worry about too many permits becoming available to the transmitting node of each link.
- 6.14** Consider a network using a node-by-node window for each session. Suppose that the transmission capacity of all links of the networks is increased by a factor K and that the number of sessions that can be served also increases by a factor K . Argue that in order for the network to allow for full-speed transmission for each session under light traffic conditions, the total window of the network should increase by a factor K if propagation delay is dominated by packet transmission time and by a factor K^2 if the reverse is true.
- 6.15** Consider the variation of the leaky bucket scheme where $W > 1$ permits are allocated initially to a session and the count is restored back to W every W/r seconds. Develop a Markov chain model for the number of packets waiting to get a permit. Assume a Poisson arrival process.
- 6.16** Describe how the gradient projection method for optimal routing can be used to solve in distributed fashion the combined optimal routing and flow control problem of Section 6.5.1.
- 6.17** Let r be a max-min fair rate vector corresponding to a given network and set of sessions.
- (a) Suppose that some of the sessions are eliminated and let $\bar{r}$ be a corresponding max-min fair rate vector. Show by example that we may have $\bar{r}_p < r_p$ for some of the remaining sessions p .
 - (b) Suppose that some of the link capacities are increased and let $\bar{r}$ be a corresponding max-min fair rate vector. Show by example that we may have $\bar{r}_p < r_p$ for some sessions p .
- 6.18** *Alternative Formulation of Max-Min Fair Flow Control ([Ja81] and [Ga84b]).* Consider the max-min flow control problem where the rate vector r is required, in addition, to satisfy $r_p \leq (C_a - F_a)q_a$ for each session p and link a crossed by session p . Here q_a are given positive scalars.
- (a) Show that a max-min fair rate vector exists and is unique, and give an algorithm for calculating it.
 - (b) Show that for a max-min fair rate vector, the utilization factor $\rho_a = F_a/C_a$ of each link a satisfies

$$\rho_a \leq \frac{n_a q_a}{1 + n_a q_a}$$

where n_a is the number of sessions crossing link a .

- (c) Show that additional constraints of the form $r_p \leq R_p$, where R_p is a given positive scalar for each session p , can be accommodated in this formulation by adding to the network one extra link per session.

- 6.19** *Guaranteed Delay Bounds Using Leaky Buckets [PaG91a],[PaG91b].* In this problem we show how guaranteed delay bounds for the sessions sharing a network can be obtained by appropriately choosing the sessions' leaky bucket parameters. We assume that each session i has a fixed route and is constrained by a leaky bucket with parameters W_i and r_i , as in the scheme of Fig. 6.16. We assume a fluid model for the traffic, *i.e.*, the packet sizes are infinitesimal. In particular, if $A_i(\tau, t)$ is the amount of session i traffic entering the network during an interval $[\tau, t]$,

$$A_i(\tau, t) \leq W_i + r_i(t - \tau).$$

Let $T_i^l(\tau, t)$ be the amount of session i traffic transmitted on link l in an interval $[\tau, t]$. Assume that each link l transmits at its maximum capacity C_l whenever it has traffic in queue and operates according to a priority discipline, called *Rate Proportional Processor Sharing*, whereby if two sessions i and j have traffic in queue throughout the interval $[\tau, t]$, then

$$\frac{T_i^l(\tau, t)}{T_j^l(\tau, t)} = \frac{r_i}{r_j} \quad (6.11)$$

We also assume that bits of the same session are transmitted in the order of their arrival. (For practical approximations of such an idealized scheme, see [PaG91a].) Let $S(l)$ be the set of sessions sharing link l and let

$$\rho(l) = \frac{\sum_{i \in S(l)} r_i}{C_l}$$

be the corresponding link utilization. Assume that $\rho(l) < 1$ for all links l .

- (a) Let l_i be the first link crossed by session i . Show that the queue length at link l_i for session i is never more than W_i . Furthermore, at link l_i , each bit of session i waits in queue no more than $W_i \rho(l_i) / r_i$ time units, while for each interval $[\tau, t]$ throughout which session i has traffic in queue,

$$T_i^{l_i}(\tau, t) \geq \frac{r_i(t - \tau)}{\rho(l_i)}$$

Hint: The rate of transmission of session i in an interval throughout which session i has traffic in queue is at least $r_i / \rho(l_i)$. If $\bar{\tau}$ and $\bar{t}$ are the start and end of a busy period for session i , respectively, the queue length at times $t \in [\bar{\tau}, \bar{t}]$ is

$$Q_i(t) = A_i(\bar{\tau}, t) - T_i^{l_i}(\bar{\tau}, t) \leq W_i + \left(r_i - \frac{r_i}{\rho(l_i)} \right) (t - \bar{\tau})$$

- (b) Let ρ_{max}^i be the maximum utilization over the links crossed by session i and assume that processing and propagation delay are negligible. Show that the amount of traffic of session i within the entire network never exceeds W_i . Furthermore, the time spent inside the network by a bit of session i is at most $W_i \rho_{max}^i / r_i$. *Hint:* Argue that while some link on session i 's path has some session i traffic waiting in queue, the rate of departure of session i traffic from the network is at least r_i / ρ_{max}^i .
- (c) Consider now a generalization of the preceding scheme called *Generalized Processor Sharing*. In particular, suppose that at each link l , instead of Eq. (6.11), the following relation holds for all sessions $i, j \in S(l)$

$$\frac{T_i^l(\tau, t)}{T_j^l(\tau, t)} = \frac{\phi_i^l}{\phi_j^l}$$

where ϕ_i^l, ϕ_j^l are given positive numbers. For any session i define

$$g_i = \min_{\substack{\text{all links } l \\ \text{crossed by } i}} \frac{1}{\rho(l)} \frac{\phi_i^l}{\sum_{j \in S(l)} \phi_j^l}$$

Show that if $g_i \geq r_i$, then the amount of traffic of session i within the entire network never exceeds

$$\frac{W_i r_i}{g_i}$$

Furthermore, the time spent by a bit of session i inside the network is at most W_i/g_i .

References

- [AaM81] AASHTIANI, H. Z., and MAGNANTI, T. L. 1981. Equilibria on a Congested Transportation Network, *SIAM J. Algebraic Discrete Methods*, 2:213–226.
- [Abr70] ABRAMSON, N. 1970. The Aloha System—Another Alternative for Computer Communications, *Proc. Fall Joint Comput. Conf., AFIPS Conf.*, p. 37.
- [Ahu79] AHUJA V. 1979. Routing and Flow Control in Systems Network Architecture, *IBM Systems J.*, 18:298–314.
- [AkK77] AKINC, U., and KHUMAWALA, B. 1977. An Efficient Branch and Bound Algorithm for the Capacitated Warehouse Location Problem, *Management Sci.*, 23:585–594.
- [Ald86] ALDOUS, D. 1986. *Ultimate Instability of Exponential Back-off Protocol for Acknowledgment-Based Transmission Control of Random Access Communication Channels*. Berkeley, CA: University of California, Dept. of Statistics.
- [AlM76] ALCOUFFE, A., and MURATET, G. 1976. Optimum Location of Plants, *Management Sci.*, 23:267–274.
- [Alt86] ALTES, T. 1986. *Minimum Delay Packet Length* (Report LIDS-TH-1602). Cambridge, MA: MIT Laboratory for Information and Decision Systems.
- [AnP86] ANAGNOSTOU, M., and PROTONOTARIOS, E. 1986. Performance Analysis of the Selective Repeat ARQ Protocol, *IEEE Trans. Comm.*, COM-34:127–135.
- [Ari84] ARIKAN, E. 1984. Some Complexity Results about Packet Radio Networks, *IEEE Trans. Inform. Theory*, IT-30:681–685.
- [Ash90] ASH, G. R. 1990. Design and Control of Networks with Dynamic Nonhierarchical Routing, *IEEE Communications Magazine*, 28:34–40.
- [Atk80] ATKINS, J. D. 1980. Path Control: The Transport Network of SNA., *IEEE Trans. Comm.*, COM-28:527–538.
- [BaA81] BAKRY, S. H., and ACKROYD, M. H. 1981. Teletraffic Analysis for Single Cell Mobile Radio Telephone Systems, *IEEE Trans. Comm.*, COM-29:298–304.
- [BaA82] BAKRY, S. H., and ACKROYD, M. H. 1982. Teletraffic Analysis for Multicell Mobile Radio Telephone Systems, *IEEE Trans. Comm.*, COM-30:1905–1909.

- [BaC89] BALLERT, R., and CHING, Y.-C. 1989. SONET: Now It's the Standard Optical Network, *IEEE Comm. Mag.*, 29:8–15.
- [Bac88] BACKES, F. 1988. Transparent Bridges for Interconnection of IEEE 802 LANs, *IEEE Network*, 2:5–9.
- [Bal79] BALL, M. O. 1979. Computing Network Reliability, *Oper. Res.*, 27:823–838.
- [BaP83] BALL, M. O., and PROVAN, J. S. 1983. Calculating Bounds on Reachability and Connectedness in Stochastic Networks, *Networks*, 13:253–278.
- [Bar64] BARAN, P. 1964. On Distributed Communication Networks, *IEEE Trans. on Communications Systems*, CS-12:1–9.
- [BaS83] BARATZ, L., and SEGALL, A. 1983. *Reliable Link Initialization Procedures* (Report RC10032). Yorktown Heights, NY: IBM Thomas J. Watson Research Center. Also *IEEE Trans. Comm.*, COM-36, 1988, pp. 144–152.
- [BeG82] BERTSEKAS, D. P., and GAFNI, E. 1982. Projection Methods for Variational Inequalities with Application to the Traffic Assignment Problem, in D. C. Sorensen and R. J.-B. Wets (Eds.), *Mathematical Programming Studies*, Vol. 17. Amsterdam: North-Holland, pp. 139–159.
- [BeG83] BERTSEKAS, D. P., and GAFNI, E. M. 1983. Projected Newton Methods and Optimization of Multicommodity Flows, *IEEE Trans. Automat. Control*, AC-28:1090–1096.
- [Ben86] BENJAMIN, R. 1986. Analysis of Connection Survivability in Complex Strategic Communications Networks, *IEEE J. Select. Areas Comm.*, SAC-4:243–253.
- [Ber76] BERTSEKAS, D. P. 1976. On the Goldstein–Levitin–Polyak Gradient Projection Method, *IEEE Trans. Automat. Control*, AC-21:174–184.
- [Ber79a] BERTSEKAS, D. P. 1979. Algorithms for Nonlinear Multicommodity Network Flow Problems, in A. Bensoussan and J. L. Lions (Eds.), *International Symposium on Systems Optimization and Analysis*. New York: Springer-Verlag, pp. 210–224.
- [Ber79b] BERTSEKAS, D. P. 1979. Dynamic Models of Shortest Path Routing Algorithms for Communication Networks with Multiple Destinations, *Proc. 1979 IEEE Conf. Dec. Control*, Ft. Lauderdale, FL, pp. 127–133.
- [Ber80] BERTSEKAS, D. P. 1980. A Class of Optimal Routing Algorithms for Communication Networks, *Proc. 5th Internat. Conf. Comput. Comm.*, Atlanta, GA, pp. 71–76.
- [Ber82a] BERTSEKAS, D. P. 1982. Distributed Dynamic Programming, *IEEE Trans. Automat. Control*, AC-27:610–616.
- [Ber82b] BERTSEKAS, D. P. 1982. Dynamic Behavior of Shortest Path Routing Algorithms for Communication Networks, *IEEE Trans. Automat. Control*, AC-27:60–74.
- [Ber82c] BERTSEKAS, D. P. 1982. Projected Newton Methods for Optimization Problems with Simple Constraints, *SIAM J. Control Optim.*, 20:221–246.
- [Ber82d] BERTSEKAS, D. P. 1982. *Constrained Optimization and Lagrange Multiplier Methods*. New York: Academic Press.
- [Ber83] BERTSEKAS, D. P. 1983. Distributed Asynchronous Computation of Fixed Points, *Math. Programming*, 27:107–120.
- [Ber85] BERTSEKAS, D. P. 1985. A Unified Framework for Primal–Dual Methods in Minimum Cost Network Flow Problems, *Math. Programming*, 32:125–145.
- [Ber87] BERTSEKAS, D. P. 1987. *Dynamic Programming: Deterministic and Stochastic Models*. Englewood Cliffs, NJ: Prentice Hall.
- [Ber91] BERTSEKAS, D. P. 1991. *Linear Network Optimization: Algorithms and Codes*, MIT Press, Cambridge, MA.
- [BeT89] BERTSEKAS, D. P., and TSITSIKLIS, J. N. 1989. *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall, Englewood Cliffs, N.J.

- [BGG84] BERTSEKAS, D. P., GAFNI, E. M., and GALLAGER, R. G. 1984. Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks, *IEEE Trans. Comm.*, COM-32:911–919.
- [BGS80] BIALLY, T., GOLD, B., and SENEFF, S. 1980. A Technique for Adaptive Voice Flow Control in Integrated Packet Networks, *IEEE Trans. Comm.*, COM-28:325–333.
- [BGT84] BERTSEKAS, D. P., GENDRON, R., and TSAI, W. K. 1984. *Implementation of an Optimal Multicommodity Network Flow Algorithm Based on Gradient Projection and a Path Flow Formulation* (Report LIDS-P-1364). Cambridge, MA: MIT Laboratory for Information and Decision Systems.
- [BGV79] BERTSEKAS, D. P., GAFNI, E. M., and VASTOLA, K. S. 1979. Validation of Algorithms for Optimal Routing of Flow in Networks, *Proc. 1978 IEEE Conf. Dec. Control*, San Diego, CA.
- [Bin75] BINDER, R. 1975. A Dynamic Packet Switching System for Satellite Broadcast Channels, *Proc. ICC*, pp. 41.1–41.5.
- [Bla83] BLAHUT, R. E. 1983. *Theory and Practice of Error Control Codes*. Reading, MA: Addison-Wesley.
- [BLL84] BURMAN, D. Y., LEHOCZKY, J. P., and LIM, Y. 1984. Insensitivity of Blocking Probabilities in a Circuit-Switching Network, *Advances in Applied Probability*, 21:850–859.
- [BoF77] BOORSTYN, R. R., and FRANK, H. 1977. Large-Scale Network Topological Optimization, *IEEE Trans. Comm.*, COM-25:29–47.
- [BoM86] BOXMA, O. J., and MEISTER, B. 1986. Waiting-Time Approximations for Cyclic-Service Systems with Switch-Over Times, *Perform. Eval. Rev.*, 14:254–262.
- [BoS82] BOCHMANN, G. V., and SUNSHINE, C. A. 1982. A Survey of Formal Methods, in P. Green (Ed.), *Computer Network Architecture*. New York: Plenum.
- [BrB80] BRUELL, S. C., and BALBO, G., 1980. *Computational Algorithms for Closed Queueing Networks*. New York: Elsevier/North-Holland.
- [BrC91a] BRASSIL, J. T., and CRUZ, R. L. 1991. Nonuniform Traffic in the Manhattan Street Network, *Proceedings of ICC '91*, 3:1647–1651.
- [BrC91b] BRASSIL, J. T., and CRUZ, R. L. 1991. Bounds on Maximum Delay in Networks with Deflection Routing, *Proceedings of the 29th Allerton Conference on Communications, Control, and Computing*, Monticello, IL.
- [Bux81] BUX, W. 1981. Local Area Networks: A Performance Comparison, *IEEE Trans. Comm.*, COM-29:1465–1473.
- [CaC68] CANNON, M. D., and CULLUM, C. D. 1968. A Tight Upper Bound on the Rate of Convergence of the Frank–Wolfe Algorithm, *SIAM J. Control Optim.*, 6:509–516.
- [CaH75] CARLEIAL, A. B., and HELLMAN, M. E. 1975. Bistable Behavior of Slotted Aloha-Type Systems, *IEEE Trans. Comm.*, COM-23:401–410.
- [Cap77] CAPETANAKIS, J. I. 1977. *The Multiple Access Broadcast Channel: Protocol and Capacity Considerations*, Ph.D. dissertation, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, MA. Also 1979, *IEEE Trans. Inform. Theory*, IT-25:505–515.
- [Car80] CARLSON, D. E. 1980. Bit-Oriented Data Link Control Procedures, *IEEE Trans. Comm.*, COM-28:455–467. (Also in [Gre82].)
- [CFL79] CHLAMTAC, I., FRANTA W., and LEVIN, K. D. 1979. BRAM: The Broadcast Recognizing Access Method, *IEEE Trans. Comm.*, COM-27:1183–1190.
- [CFN77] CORNUEJOLS, G., FISHER, M. L., and NEMHAUSER, G. L. 1977. Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms, *Management Sci.*, 23:789–810.

- [CGK90] CIDON, I., GOPAL, I. S., KAPLAN, M., and KUTTEN, S. 1990. Distributed Control for PARIS, *Proc. of the 9th Annual ACM Symposium on Principles of Computing*, Quebec, Can.
- [CiG88] CIDON, I., and GOPAL, I. S. 1988. PARIS: An Approach to Integrated High-Speed Private Networks, *International Journal of Digital and Analog Cabled Systems*, 1:77–85.
- [CIC81] CLARK, G. C., and CAIN, J. B. 1981. *Error Correction Coding for Digital Communication*. New York: Plenum.
- [CoG89] CONWAY, A. E., and GEORGANAS, N. D. 1989. *Queueing Networks—Exact Computational Algorithms*, MIT Press, Cambridge, MA.
- [Com88] COMER, D. 1988. *Internetworking with TCP/IP, Principles, Protocols, and Architectures*, Prentice-Hall, Englewood Cliffs.
- [Coo70] COOPER, R. B. 1970. Queues Served in Cyclic Order: Waiting Times, *Bell Systems Tech. J.*, 49:399–413.
- [Coo81] COOPER, R. B. 1981, *Introduction to Queueing Theory* (2nd ed.). New York: Elsevier/North-Holland.
- [CPR78] CLARK, D. D., POGAN, K. T., and REED, D. P. 1978. An Introduction to Local Area Networks. *Proc. IEEE*, pp. 1497–1517.
- [Cru91a] CRUZ, R. L. 1991. A Calculus for Network Delay, Part I: Network Elements in Isolation, *IEEE Trans. on Inform. Theory*, IT-37:114–131.
- [Cru91b] CRUZ, R. L. 1991. A Calculus for Network Delay, Part II: Network Analysis, *IEEE Trans. on Inform. Theory*, IT-37:132–141.
- [CRW73] CROWTHER, W., RETTBERG, R., WALDEN, D., ORNSTEIN, S., and HEART, F. 1973. A System for Broadcast Communication: Reservation Aloha, *Proc. 6th Hawaii Internat. Conf. Syst. Sci.*, pp. 371–374.
- [Daf71] DAFERMOS, S. C. 1971. An Extended Traffic Assignment Model with Applications to Two-Way Traffic, *Trans. Sci.*, 5:366–389.
- [Daf80] DAFERMOS, S. C. 1980. Traffic Equilibrium and Variational Inequalities, *Trans. Sci.*, 14:42–54.
- [Dan63] DANTZIG, G. B. 1963. *Linear Programming and Extensions*. Princeton, NJ: Princeton University Press.
- [DeK81] DEMBO, R. S., and KLINCEWICZ, J. G. 1981. A Scaled Reduced Gradient Algorithm for Network Flow Problems with Convex Separable Costs, *Math. Programming Stud.*, 15:125–147.
- [DGK79] DIAL, R., GLOVER, F., KARNEY, D., and KLINGMAN, D. 1979. A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees, *Networks*, 9:215–248.
- [DiK85] DISNEY, R. L., and KONIG, D. 1985. Queueing Networks: A Survey of Their Random Processes, *SIAM Rev.*, 27:335–403.
- [Dis80] DIJKSTRA, E. W., and SHOLTEN, C. S. 1980. Termination Detection for Diffusing Computations, *Inf. Proc. Lett.*, 11:1–4.
- [Dun79] DUNN, J. C. 1979. Rates of Convergence of Conditional Gradient Algorithms near Singular and Nonsingular Extremals, *SIAM J. Control Optim.*, 17:187–211.
- [Eis79] EISENBERG, M. 1979. Two Queues with Alternating Service, *SIAM J. Appl. Math.*, 20:287–303.
- [ELL90a] ECKBERG, A. E., LUAN, D. T., and LUCANTONI, D. M. 1990. Bandwidth Management: A Congestion Control Strategy for Broadband Packet Networks—Characterizing the Throughput-Burstiness Filter, *Computer Networks and ISDN Systems*, 20:415–423.

- [ELL90b] ECKBERG, A. E., LUAN, D. T., and LUCANTONI, D. M. 1990. An Approach to Controlling Congestion in ATM Networks, *International Journal of Digital and Analog Communication Systems*, 3:199–209.
- [ELS90] ESCOBAR, J., LAUER, G., and STEENSTRUP, M. 1990. Performance Analysis of Rate-Based Congestion Control Algorithm for Receiver-Directed Packet-Radio Networks, *Proc. of MILCOM '90*.
- [EpB81] EPHREMIDES, A., and BAKER, D. J. 1981. The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm, *IEEE Trans. Comm.*, COM-29:1694–1701.
- [Eph86] EPHREMIDES, A. 1986. The Routing Problem in Computer Networks, in I. F. Blake and H. V. Poor (Eds.), *Communication and Networks*. New York: Springer-Verlag, pp. 299–324.
- [EpV89] EPHREMIDES, A., and VERDU, S. 1989. Control and Optimization Methods in Communication Network Problems, *IEEE Trans. Automat. Control*, 34:930–942.
- [EsW66] ESSAU, L. R., and WILLIAMS, K. C. 1966. On Teleprocessing System Design, *IBM Systems J.*, 5:142–147.
- [Eve75] EVEN, S. 1975. An Algorithm for Determining Whether the Connectivity of a Graph Is at Least k , *SIAM J. Comput.*, 4:393–396.
- [EVW80] EPHREMIDES, A., VARAIYA, P., and WALRAND, J. 1980. A Simple Dynamic Routing Problem, *IEEE Trans. Automat. Control*, AC-25:690–693.
- [FaL72] FARBER, D. J., and LARSON, K. C. 1972. *The System Architecture of the Distributed Computer System—The Communications System.*, paper presented at the Symposium on Computer Networks, Polytechnic Institute of Brooklyn, New York.
- [FaN69] FARMER, W. D., and NEWHALL, E. E. 1969. An Experimental Distributed Switching System to Handle Bursty Computer Traffic, *Proc. ACM Symp. Problems Optim. Data Comm. Syst.*, pp. 1–33.
- [FeA85] FERGUSON, M. J., and AMINETZAH, Y. J. 1985. Exact Results for Nonsymmetric Token Ring Systems, *IEEE Trans. Comm.*, COM-33:223–231.
- [FeA89] FETTEROLF, P. C., and ANANDALINGAM, G. 1989. Optimizing Interconnection of Local Area Networks: An Approach Using Simulated Annealing, *Univ. of Pennsylvania Report*; to appear in *Annals of Operations Research*.
- [Fer75] FERGUSON, M. J. 1975. A Study of Unslotted Aloha with Arbitrary Message Lengths, *Proc. 4th Data Comm. Symp.*, Quebec, Canada, pp. 5.20–5.25.
- [FGK73] FRATTA, L., GERLA, M., and KLEINROCK, L. 1973. The Flow Deviation Method: An Approach to Store-and-Forward Communication Network Design, *Networks*, 3:97–133.
- [Fin79] FINN, S. G. 1979. Resynch Procedures and a Fail-Safe Network Protocol, *IEEE Trans. Comm.*, COM-27:840–845.
- [FIN74] FLORIAN, M., and NGUYEN, S. 1974. A Method for Computing Network Equilibrium with Elastic Demand, *Transport. Sci.*, 8:321–332.
- [FiT84] FINE, M., and TOBAGI, F. A. 1984. Demand Assignment Multiple Access Schemes in Broadcast Bus Local Area Networks, *IEEE Trans. Comput.*, C-33:1130–1159.
- [FKL86] FUJIWARA, T., KASAMI, T., and LIN, S. 1986. Error Detecting Capabilities of the Shortened Hamming Codes Adopted for Error Detection in IEEE Standard 802.3, *Int. Symp. I.T.*, Ann Arbor, Mich.
- [FoF62] FORD, L. R., Jr., and FULKERSON, D. R. 1962. *Flows in Networks*. Princeton, NJ: Princeton University Press.
- [FoS78] FOSCHINI, G. J., and SALZ, J. 1978. A Basic Dynamic Routing Problem and Diffusion, *IEEE Trans. Comm.*, COM-26:320–327.

- [FrW56] FRANK, M., and WOLFE, P. 1956. An Algorithm for Quadratic Programming, *Naval Res. Logist. Quart.*, 3:149–154.
- [FuC85] FUHRMANN, S. W., and COOPER, R. B. 1985. Stochastic Decompositions in the $M/G/1$ Queue with Generalized Vacations, *Oper. Res.*, 33:1117–1129.
- [GaB81] GAFNI, E. M., and BERTSEKAS, D. P. 1981. Distributed Routing Algorithms for Networks with Frequently Changing Topology, *IEEE Trans. Comm.*, COM-29:11–18.
- [GaB83] GAFNI, E. M., and BERTSEKAS, D. P. 1983. *Asymptotic Optimality of Shortest Path Routing* (Report LIDS-P-1307). Cambridge, MA: MIT Laboratory for Information and Decision Systems. Also 1987, *IEEE Trans. Inform. Theory*, IT-33:83–90.
- [GaB84a] GAFNI, E. M., and BERTSEKAS, D. P. 1984. Two-Metric Projection Methods for Constrained Optimization, *SIAM J. Control Optim.*, 22:936–964.
- [GaB84b] GAFNI, E. M., and BERTSEKAS, D. P. 1984. Dynamic Control of Session Input Rates in Communication Networks, *IEEE Trans. Automat. Control*, AC-29:1009–1016.
- [Gaf79] GAFNI, E. M. 1979. *Convergence of a Routing Algorithm*, M.S. thesis, University of Illinois, Dept. of Electrical Engineering, Urbana, IL.
- [Gaf82] GAFNI, E. M. 1982. *The Integration of Routing and Flow Control for Voice and Data in Integrated Packet Networks*, Ph.D. thesis, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, MA.
- [GaG80] GALLAGER, R. G., and GOLESTAANI, S. J. 1980. Flow Control and Routing Algorithms for Data Networks, *Proc. 5th Internat. Conf. Comput. Comm.*, pp. 779–784.
- [GaH83] GAVISH, B., and HANTLER, S. 1983. An Algorithm for Optimal Route Selection in SNA Networks, *IEEE Trans. Comm.*, COM-31:1154–1161.
- [Gal68] GALLAGER, R. G. 1968. *Information Theory and Reliable Communications*. New York: Wiley.
- [Gal77] GALLAGER, R. G. 1977. A Minimum Delay Routing Algorithm Using Distributed Computation, *IEEE Trans. Comm.*, COM-23:73–85.
- [Gal78] GALLAGER, R. G. 1978. Conflict Resolution in Random Access Broadcast Networks, *Proc. AFOSR Workshop Comm. Theory Appl.*, Provincetown, MA, pp. 74–76.
- [Gal81] GALLAGER, R. G. 1981. Applications of Information Theory for Data Communication Networks, in J. Skwirzynski (Ed.), *New Concepts in Multi-user Communication* (Series E, No. 43). Alphen aan den Rijn, The Netherlands: NATO Advanced Study Institutes (Sijthoff en Noordhoff).
- [GaP88] GALLO, G. S., and PALLOTINO, S. 1988. Shortest Path Algorithms, *Annals of Operations Research*, 7:3–79.
- [Gar87] GARCIA-LUNA-ACEVES, J. J. 1987. A New Minimum-Hop Routing Algorithm, *IEEE INFOCOM '87 Proceedings*, pp. 170–180.
- [Gav85] GAVISH, B. 1985. Augmented Lagrangean-Based Algorithms for Centralized Network Design, *IEEE Trans. Comm.*, COM-33:1247–1257.
- [GeK77] GERLA, M., and KLEINROCK, L. 1977. On the Topological Design of Distributed Computer Networks, *IEEE Trans. Comm.*, COM-25:48–60.
- [GeK80] GERLA, M., and KLEINROCK, L. 1980. Flow Control: A Comparative Survey, *IEEE Trans. Comm.*, COM-28:553–574.
- [GeK88] GERLA, M., and KLEINROCK, L. 1988. Congestion Control in Interconnected LANs, *IEEE Network*, 2:72–76.
- [GeP85] GEORGLADIS, L., and PAPANTONI-KAZAKOS, P. 1985. *A 0.487 Throughput Limited Sensing Algorithm*. Storrs, CT: University of Connecticut.
- [GeP87] GELEMBE, E., and PUJOLLE, G. 1988. *Introduction to Queueing Networks*. J. Wiley, N.Y.

- [GeY82] GEORGE, F. D., and YOUNG, G. E. 1982. SNA Flow Control: Architecture and Implementation, *IBM Syst. J.*, 21:179–210.
- [GGM85] GOODMAN, J., GREENBERG, A. G., MADRAS, N., and MARCH, P. 1985. On the Stability of Ethernet, *Proc. 17th Annual ACM Symp. Theory Comput.*, Providence, RI, pp. 379–387.
- [GHS83] GALLAGER, R. G., HUMBLET, P. A., and SPIRA, P. M. 1983. A Distributed Algorithm for Minimum-Weight Spanning Trees, *ACM Trans. Programming Language Syst.*, 5:66–77.
- [Gol64] GOLDSTEIN, A. A., 1964. Convex Programming in Hilbert Space, *Bull. Am. Math. Soc.*, 70:709–710.
- [Gol80] GOLESTAANI, S. J. 1980. *A Unified Theory of Flow Control and Routing on Data Communication Networks*, Ph.D. thesis, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, MA.
- [Gol90a] GOLESTAANI, S. J. 1990. Congestion-Free of Real-Time Traffic in Packet Networks, *Proc. of IEEE INFOCOM '90*, San Francisco, Cal., pp. 527–536.
- [Gol90b] GOLESTAANI, S. J. 1990. A Framing Strategy for Congestion Management, *Proc. of SIGCOM '90*.
- [Gop85] GOPAL, I. S., 1985. Prevention of Store-and-Forward Deadlock in Computer Networks, *IEEE Trans. Comm.*, COM-33:1258–1264.
- [Gra72] GRAY, J. P. 1972. Line Control Procedures, *Proc. IEEE*, pp. 1301–1312.
- [GrB83] GROVER, G. A., and BHARATH-KUMAR, K. 1983. Windows in the Sky—Flow Control in SNA Networks with Satellite Links, *IBM Systems J.*, 22:451–463.
- [Gre82] GREEN, P. E. 1982. *Computer Network Architectures and Protocols*. New York: Plenum.
- [Gre84] GREEN, P. E. 1984. Computer Communications: Milestones and Prophecies, *IEEE Trans. Comm.*, COM-32:49–63.
- [Gre86] GREEN, P. E. 1986. Protocol Conversion, *IEEE Trans. Comm.*, COM-34:257–268.
- [GrH85] GROSS, D., and HARRIS, C. M. 1985. *Fundamentals of Queueing Theory* (2nd ed.). New York: Wiley.
- [GrH89] GREENBERG, A. G., and HAJEK, B. 1989. Deflection Routing in Hypercube Networks, *IEEE Trans. Comm.*, to appear.
- [Gun81] GUNTHER, K. D. 1981. Prevention of Deadlocks in Packet-Switched Data Transport Systems, *IEEE Trans. Comm.*, COM-29:512–524.
- [HaC90] HAJEK, B., and CRUZ, R. L. 1990. On the Average Delay for Routing Subject to Independent Deflections, *IEEE Trans. of Inform. Theory*, to appear.
- [HaG86] HAHNE, E. L., and GALLAGER, R. G. 1986. *Round-Robin Scheduling for Fair Flow Control in Data Communication Networks* (Report LIDS-P-1537). Cambridge, MA: MIT Laboratory for Information and Decisions Systems.
- [Hah86] HAHNE, E. 1986. Round-Robin Scheduling for Fair Flow Control in Data Communication Networks Ph.D. thesis, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, MA.
- [Haj82] HAJEK, B. 1982. Birth-and-Death Processes with Phases and General Boundaries, *J. Appl. Problems*, 19:488–499.
- [Haj88] HAJEK, B. 1988. Cooling Schedules for Optimal Annealing, *Math of Operations Research*, 13:311–329.
- [Haj91] HAJEK, B. 1991. Bounds on Evacuation Time for Deflection Routing, *Distributed Computing*, 5:1–5.
- [HaL82] HAJEK, B., and VAN LOON, T. 1982. Decentralized Dynamic Control of a Multiaccess Broadcast Channel, *IEEE Trans. Automat. Control*, AC-27:559–569.

- [HaO84] HAJEK, B., and OGIER, R. G. 1984. Optimal Dynamic Routing in Communication Networks with Continuous Traffic, *Networks*, 14:457–487.
- [Has72] HASHIDA, O. 1972. Analysis of Multiqueue, *Rev. Electron. Comm. Lab.*, 20:189–199.
- [Hay76] HAYES, J. F. 1976. *An Adaptive Technique for Local Distribution* (Bell Telephone Laboratory Technical Memo TM-76-3116-1.) (Also 1978, *IEEE Trans. Comm.*, COM-26:1178–1186.)
- [Hay81] HAYDEN, H. 1981. *Voice Flow Control in Integrated Packet Networks* (Report LIDS-TH-1152). Cambridge, MA: MIT Laboratory for Information and Decision Systems.
- [Hay84] HAYES, J. F. 1984. *Modeling and Analysis of Computer Communications Networks*. New York: Plenum.
- [HCM90] HAHNE, E. L., CHOUDHURY, A., and MAXEMCHUK, N. 1990. Improving the Fairness of Distributed-Queue-Dual-Bus Networks, *Infocom*, San Francisco, Ca., pp. 175–184.
- [HeS82] HEYMAN, D. P., and SOBEL, M. J. 1982. *Stochastic Models in Operations Research*, Vol. 1. New York: McGraw-Hill.
- [HIG81] HLUCHYI, M. G., and GALLAGER, R. G. 1981. Multiaccess of a Slotted Channel by Finitely Many Users, *Proc. Nat. Telecomm. Conf.*, New Orleans, LA. (Also LIDS Report P-1131, MIT, Cambridge, MA, August 1981.)
- [HSS86] HUMBLET, P. A., SOLOWAY, S. R., and STEINKA, B. 1986. *Algorithms for Data Communication Networks—Part 2*. Codex Corp.
- [HuB85] HUANG, J.-C., and BERGER, T. 1985. Delay Analysis of the Modified 0.487 Contention Resolution Algorithm, *IEEE Trans. Inform. Theory*, IT-31:264–273.
- [HuM80] HUMBLET, P. A., and MOSELY, J. 1980. Efficient Accessing of a Multiaccess Channel, *Proc. IEEE 19th Conf. Dec. Control*, Albuquerque, NM. (Also LIDS Report P-1040, MIT, Cambridge, MA, Sept. 1980.)
- [Hum78] HUMBLET, P. A. 1978. *Source Coding for Communication Concentrators* (Report ESL-R-798). Cambridge, MA: MIT.
- [Hum83] HUMBLET, P. A. 1983. A Distributed Algorithm for Minimum Weight Directed Spanning Trees, *IEEE Trans. Comm.*, COM-31:756–762.
- [Hum86] HUMBLET, P. A. 1986. On the Throughput of Channel Access Algorithms with Limited Sensing, *IEEE Trans. Comm.*, COM-34:345–347.
- [Hum91] HUMBLET, P. A. 1991. Another Adaptive Distributed Dijkstra Shortest Path Algorithm, *IEEE Trans. on Comm.*, COM-39:995–1003; also *MIT Laboratory for Information and Decision Systems Report LIDS-P-1775*, May 1988.
- [HuS86] HUMBLET, P. A., and SOLOWAY, S. R. 1986. *Algorithms for Data Communication Networks —Part I*. Codex Corp.
- [HuS88] HUMBLET, P. A. and S. SOLOWAY. 1988/1989. Topology Broadcast Algorithms, *Computer Networks and ISDN Systems*, 16:179–186; also *Codex Corporation Research Report*; revised May 1987 as Report LIDS-P-1692.
- [IbC89] IBE, O. C., and CHENG, X. 1989. Approximate Analysis of Asymmetric Single-Service Token-passing Systems, *IEEE Trans. on Comm.*, COM-36:572–577.
- [Ibe81] IBE, O. C. 1981. *Flow Control and Routing in an Integrated Voice and Data Communication Network*. Ph.D. thesis. MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, MA.
- [IEE86] *IEEE Journal on Selected Areas in Communications, Special Issue on Network Performance Evaluation*, SAC-4(6), Sept. 1986.
- [IEE88] *IEEE Network*, Vol. 2, No. 1, 1988.
- [IEE89] *IEEE Journal on Selected Areas in Communications, Special Issue on Telecommunications Network Design and Planning*, Vol. 7, No. 8, 1989.

- [Jac57] JACKSON, J. R. 1957. Networks of Waiting Lines, *Oper. Res.*, 5:518–521.
- [Jaf81] JAFFE, J. M. 1981. A Decentralized “Optimal” Multiple-User Flow Control Algorithm, *IEEE Trans. Comm.*, COM-29:954–962.
- [Jai68] JAISWAL, N. K. 1968. *Priority Queues*. New York: Academic Press.
- [JaM82] JAFFE, J. M., and MOSS, F. M. A. 1982. Responsive Routing Algorithm for Computer Networks, *IEEE Trans. on Comm.*, COM-30:1758–1762.
- [JBH78] JACOBS, I. M., BINDER, R., and HOVERSTEN, E. V. 1978. General Purpose Packet Satellite Networks, *Proc. IEEE*, pp. 1448–1468.
- [Kap79] KAPLAN, M. 1979. A Sufficient Condition for Nonergodicity of a Markov Chain, *IEEE Trans. Inform. Theory*, IT-25:470–471.
- [KaT75] KARLIN, S., and TAYLOR, H. M. 1975. *A First Course in Stochastic Processes*. New York: Academic Press.
- [Kau81] KAUFMAN, J. S. 1981. Blocking in a Shared Resource Environment, *IEEE Trans. Comm.*, COM-29:1474–1481.
- [KeB83] KERSHENBAUM, A., and BOORSTYN, R. R. 1983. Centralized Teleprocessing Network Design, *Networks*, 13:279–293.
- [KeC90] KEY, P. B., and COPE, G. A. 1990. Distributed Dynamic Routing Schemes, *IEEE Communications Magazine*, 28:54–64.
- [KeH63] KEUHN, A. A., and HAMBURGER, M. J. 1963. A Heuristic Program for Locating Warehouses, *Management Sci.*, 9:643–666.
- [Kei79] KEILSON, J. 1979. *Markov-Chain Models—Rarity and Exponentiality*. New York: Springer-Verlag.
- [Kel79] KELLY, F. P. 1979. *Reversibility and Stochastic Networks*. New York: Wiley.
- [Kel85] KELLY, F. P. 1985. Stochastic Models of Computer Communication Systems, *J. Roy. Statist. Soc. Ser. B*, 47(1).
- [Kel86] KELLY, F. P. 1986. Blocking Probabilities in Large Circuit-Switched Networks, *Adv. Appl. Prob.*, 18:473–505.
- [KGB78] KAHN, R. E., GRONEMEYER, S. A., BURCHFIEL, J., and KUNZELMAN, R. C. 1978. Advances in Packet Radio Technology, *Proc. IEEE*, pp. 1468–1496.
- [KGV83] KIRKPATRICK, S., GELATT, C. D., Jr., and VECCHI, M. P. 1983. Optimization by Simulated Annealing, *Science*, 220:671–680.
- [Khu72] KHUMAWALA, B. M. 1972. An Efficient Branch and Bound Algorithm for the Warehouse Location Problem, *Management Sci.*, 18:B718–B731.
- [KhZ89] KHANNA, A., and ZINKY, J. 1989. The Revised ARPANET Routing Metric, *Proc. of SIGCOM '89*.
- [Kin62] KINGMAN, J. F. C. 1962. Some Inequalities for the Queue $GII/G/1$, *Biometrika*, 49:315–324.
- [Kle64] KLEINROCK, L. 1964. *Communication Nets: Stochastic Message Flow and Delay*. New York: McGraw-Hill.
- [Kle69] KLEITMAN, D. 1969. Methods for Investigating the Connectivity of Large Graphs, *IEEE Trans. Circuit Theory*, CT-16:232–233.
- [Kle75] KLEINROCK, L. 1975. *Queueing Systems*, Vol. 1. New York: Wiley.
- [Kle76] KLEINROCK, L. 1976. *Queueing Systems*, Vol. 2. New York: Wiley.
- [KIL75] KLEINROCK, L., and LAM, S. 1975. Packet Switching in Multiaccess Broadcast Channel: Performance Evaluation, *IEEE Trans. Comm.*, COM-23:410–423.
- [KIO77] KLEINROCK, L., and OPDERBECK, H. 1977. Throughput in the ARPANET—Protocols and Measurements, *IEEE Trans. Comm.*, COM-25:95–104.

- [KIS80] KLEINROCK, L., and SCHOLL. 1980. Packet Switching in Radio Channels: New Conflict Free Multiple Access Schemes, *IEEE Trans. Comm.*, COM-28:1015–1029.
- [KIT75] KLEINROCK, L., and TOBAGI, F. A. 1975. Packet Switching in Radio Channels: Part I: CSMA Modes and Their Throughput-Delay Characteristics, *IEEE Trans. Comm.*, COM-23:1400–1416.
- [Kol36] KOLMOGOROV, A. 1936. Zur Theorie der Markoffschen Ketten, *Mathematische Annalen*, 112:155–160.
- [Kri90] KRISHNAN, K. R. 1990. Markov Decision Algorithms for Dynamic Routing, *IEEE Communications Magazine*, 28:66–69.
- [Kue79] KUEHN, P. J. 1979. Multiqueue Systems with Nonexhaustive Cyclic Service, *Bell System Tech. J.*, 58:671–698.
- [KuR82] KUMMERLE, K., and REISER, M. 1982. Local Area Networks—An Overview, *J. Telecomm. Networks*, 1(4).
- [KuS89] KUMAR, P. R., and SEIDMAN, T. I. 1989. Distributed Instabilities and Stabilization Methods in Distributed Real-Time Scheduling of Manufacturing Systems, *IEEE Trans. Automat. Control*, AC-35:289–298.
- [LaH84] LANPHONGPANICH, S., and HEARN, D. 1984. Simplicial Decomposition of the Asymmetric Traffic Assignment Problems, *Trans. Res.*, 18B:123–133.
- [LaK75] LAM, S., and KLEINROCK, L. 1975. Packet Switching in a Multiaccess Broadcast Channel: Dynamic Control Procedures, *IEEE Trans. Comm.*, COM-23:891–904.
- [LaL86] LAM, Y. F., and LI, V. O. K. 1986. An Improved Algorithm for Performance Analysis of Networks with Unreliable Components, *IEEE Trans. Comm.*, COM-34:496–497.
- [Lam80] LAM, S. 1980. A Carrier Sense Multiple Access Protocol for Local Networks, *Comput. Networks*, 4:21–32.
- [Las70] LASDON, L. S. 1970. *Optimization Theory for Large Systems*. New York: Macmillan.
- [LeG89] LE GALL, F. 1989. About Loss Probabilities for General Routing Policies in Circuit-Switched Networks, *IEEE Transactions on Comm.*, COM-37:57–59.
- [Lei80] LEINER, B. M. 1980. A Simple Model for Computation of Packet Radio Network Communication Performance, *IEEE Trans. Comm.*, COM-28:2020–2023.
- [LeP65] LEVITIN, E. S., and POLJAK, B. T. 1965. Constrained Minimization Methods, *USSR Comput. Math. Phys.*, 6:1–50.
- [LeY89] LEE, M.-J., and YEE, J. R. 1989. An Efficient Near-Optimal Algorithm for the Joint Traffic and Trunk Routing Problem in Self-Planning Networks, *Proc. IEEE INFOCOM '89*, pp. 127–135.
- [LiF82] LIMB, J. O., and FLORES, C. 1982. Description of Fasnet, or Unidirectional Local Area Communications Network, *Bell System Tech. J.*
- [Lim84] LIMB, J. O. 1984. Performance of Local Area Networks at High Speed, *IEEE Trans. Comm.*, COM-32:41–45.
- [LiS84] LI, V. O. K., and SILVESTER, J. A. 1984. Performance Analysis of Networks with Unreliable Components, *IEEE Trans. Comm.*, COM-32:1105–1110.
- [Lit61] LITTLE, J. 1961. A Proof of the Queueing Formula $L = \lambda W$, *Oper. Res. J.*, 18:172–174.
- [LMF88] LYNCH, N., MANSOUR, Y., and FEKETE, A. 1988. The Data Link Layer: Two Impossibility Results, *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing*. Toronto, Can., pp. 149–170.
- [Luc90] LUCKY, R. 1989. *Information, Man & Machine*. New York: St. Martins Press.
- [Lue84] LUENBERGER, D. G. 1984. *Linear and Nonlinear Programming*. Reading MA: Addison-Wesley.

- [MaF85] MATHYS, P., and FLAJOLET, P. 1985. *Q*-ary Collision Resolution Algorithms in Random-Access Systems with Free or Blocked Channel Access, *IEEE Trans. Inform. Theory*, IT-31:217–243.
- [Mal86] MALIS, A. G. 1986. *PSN End-to-End Functional Specification* (RFC 979). BBN Communications Corp., Network Working Group.
- [MaN85] MAXEMCHUK, N. F., and NETRAVALI, A. N. 1985. Voice and Data on a CATV Network, *IEEE J. Select. Areas Comm.*, SAC-3:300–311.
- [MaP88] MAGGS, B. M., and PLOTKIN, S. A. 1988. Minimum-Cost Spanning Tree as a Path Finding Problem, *Inf. Proc. Lett.*, 26:291–293.
- [Mas80] MASSEY, J. L. 1980. *Collision-Resolution Algorithms and Random Access Communications* (Report UCLA-ENG-8016). Los Angeles: University of California.
- [Max87] MAXEMCHUK, N. F. 1987. Routing in the Manhattan Street Network, *IEEE Trans. on Communications*, COM-35:503–512.
- [McS77] MCGREGOR, P. V., and SHEN, D. 1977. Network Design: An Algorithm for the Access Facility Location Problem, *IEEE Trans. Comm.*, COM-25:61–73.
- [McW77] MCQUILLAN, J. M., and WALDEN, D. C. 1977. The ARPANET Design Decisions, *Networks*, 1.
- [MeB76] METCALFE, R. M., and BOGGS, D. R. 1976. Ethernet: Distributed Packet Switching for Local Computer Networks, *Comm. ACM*, 395–404.
- [Met73] METCALFE, R. 1973. *Steady State Analysis of a Slotted and Controlled Aloha System with Blocking*, paper presented at 6th Hawaii Conf. System Sci., Honolulu.
- [Mik79] MIKHAILOV, V. A. 1979. *Methods of Random Multiple Access*, Candidate Engineering thesis, Moscow Institute of Physics and Technology, Moscow.
- [Min89] MINZNER, S. E. 1989. Broadband ISDN and Asynchronous Transfer Mode (ATM), *IEEE Commun. Mag.*, 27:17–24.
- [MiT81] MIKHAILOV, V. A., and TSYBAKOV, B. S. 1981. Upper Bound for the Capacity of a Random Multiple Access System, *Problemy Peredachi Inform.* (USSR), 17:90–95.
- [MoH85] MOSELY, J., and HUMBLET, P. A. 1985. A Class of Efficient Contention Resolution Algorithms for Multiple Access Channels, *IEEE Trans. Comm.*, COM-33:145–151.
- [MoS82] MOSS, F. H., and SEGALL, A. 1982. An Optimal Control Approach to Dynamic Routing in Networks, *IEEE Trans. Automat. Control*, AC-27:329–339.
- [MoS86] MONMA, C. L., and SHENG, D. D. 1986. Backbone Network Design and Performance Analysis: A Methodology for Packet Switching Networks, *IEEE J. Select. Areas Comm.*, SAC-4:946–965.
- [Mos84] MOSELY, J. 1984. *Asynchronous Distributed Flow Control Algorithms*, Ph.D. thesis, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, MA.
- [MRR78] MCQUILLAN, J. M., RICHER, I., ROSEN, E. C., and BERTSEKAS, D. P. 1978. *ARPANET Routing Algorithm Improvements, Second Semiannual Report* (prepared for ARPA and DCA). : Bolt, Beranek, and Newman, Inc.
- [MRR80] MCQUILLAN, J. M., RICHER, I., and ROSEN, E. C. 1980. The New Routing Algorithm for the ARPANET, *IEEE Trans. Comm.*, COM-28:711–719.
- [NBH88] NEWMAN, R. M., BUDRIKIS, Z., and HULLETT, J. 1988. Standards for Metropolitan Area Networks, *IEEE Commun. Mag.*, 26:20–28.
- [Neu81] NEUTS, M. F. 1981. *Matrix-Geometric Solutions in Stochastic Models—An Algorithmic Approach*. Baltimore, MD: The Johns Hopkins University Press.
- [NgH87] NG, M. J. T., and HOANG, D. B. 1987. Joint Optimization of Capacity and Flow Assignment in a Packet Switched Communications Network, *IEEE Trans. Comm.*, COM-35:202–209.

- [NiS74] NISNEVITCH, L., and STRASBOURGER, E. 1974. Decentralized Priority in Data Communication, *Proc. 2nd Annual Symp. Comput. Architecture*.
- [NoT78] NOMURA, M., and TSUKAMOTO, K. 1978. Traffic Analysis of Polling Systems, *Trans. Inst. Electron. Comm. Eng. (Japan)*, J61-B:600–607.
- [Nyq28] NYQUIST, H. 1928. Certain Topics in Telegraph Transmission Theory, *Trans. AIEE*, 47:617–644.
- [PaG91a] PAREKH, A. K., and GALLAGER, R. G. 1991. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case, Laboratory for Information and Decision Systems Report LIDS-P-2040, M.I.T., Cambridge, MA.
- [PaG91b] PAREKH, A. K., and GALLAGER, R. G. 1991. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case, Laboratory for Information and Decision Systems Report LIDS-P-2074, M.I.T., Cambridge, MA.
- [Pak69] PAKES, A. G. 1969. Some Conditions for Ergodicity and Recurrence of Markov Chains, *Oper. Res.*, 17:1059–1061.
- [Pap74] PAPE, U. 1974. Implementation and Efficiency of Moore-Algorithms for the Shortest Route Problem, *Math. Programming*, 7:212–222.
- [PaS82] PAPADIMITRIOU, C. H., and STEIGLITZ, K. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice Hall.
- [Per83] PERLMAN, R. 1983. Fault-Tolerant Broadcast of Routing Information, *Comput. Networks*, 7:395–405. (Also *Proc IEEE Infocom '83*, San Diego.)
- [Per88] PERLMAN, R. 1988. Network Layer Protocols with Byzantine Robustness, Ph.D. Thesis, MIT Dept. of EECS.
- [Pip81] PIPPENGER, N. 1981. Bounds on the Performance of Protocols for a Multiple Access Broadcast Channel, *IEEE Trans. Inform. Theory*, IT-27:145–151.
- [PiW82] PINEDO, M., and WOLFF, R. W. 1982. A Comparison between Tandem Queues with Dependent and Independent Service Times, *Oper. Res.*, 30:464–479.
- [Pol71] POLAK, E. 1971. *Computational Methods in Optimization*. New York: Academic Press.
- [PrB84] PROVAN, J. S., and BALL, M. O. 1984. Computing Network Reliability in Time Polynomial in the Number of Cuts, *Oper. Res.*, 32:516–526.
- [Pro83] PROAKIS, J. G. 1983. *Digital Communications*. New York: McGraw-Hill.
- [Qur85] QURESHI, S. 1985. Adaptive Equalization, *Proc. IEEE*, pp. 1349–1387.
- [RaH76] RAUBOLD, E., and HAENLE, J. 1976. A Method of Deadlock-Free Resource Allocation and Flow Control in Packet Networks, *Proc. Internat. Conf. Comput. Comm.*, Toronto.
- [RaW83] RAMAMOORTHY, C. V., and WAH, B. W. 1983. The Isomorphism of Simple File Allocation, *IEEE Trans. Comput.*, C-32:221–231.
- [ReC90] REGNIER, J., and CAMERON, W. H. 1990. State-Dependent Dynamic Traffic Management for Telephone Networks, *IEEE Communications Magazine*, 28:42–53.
- [RFS91] ROBINSON, J., FRIEDMAN, D., and STEENSTRUP, M. 1991. Congestion Control in BBN Packet-Switched Networks, *Computer Communication Reviews*.
- [Rin76] RINDE, J. 1976. TYMNET I: An Alternative to Packet Switching, *Proc. 3rd Internat. Conf. Comput. Comm.*
- [Riv85] RIVEST, R. L. 1985. *Network Control by Bayesian Broadcast* (Report MIT/LCS/TM-285). Cambridge, MA: MIT, Laboratory for Computer Science.
- [Rob72] ROBERTS, L. G. 1972. *Aloha Packet System with and without Slots and Capture* (ASS Note 8). Stanford, CA: Stanford Research Institute, Advanced Research Projects Agency, Network Information Center.
- [Roc70] ROCKAFELLAR, R. T. 1970. *Convex Analysis*. Princeton, NJ: Princeton University Press.

- [Roc84] ROCKAFELLAR, R. T. 1984. *Network Flows and Monotropic Programming*. New York: Wiley.
- [Ros80] ROSS, S. M. 1980. *Introduction to Probability Models*. New York: Academic Press.
- [Ros81] ROSEN, E. C. 1981. Vulnerabilities of Network Control Protocols: An Example. *Comput. Comm. Rev.*
- [Ros83] ROSS, S. M. 1983. *Stochastic Processes*. New York: Wiley.
- [Ros86a] ROSBERG, Z. 1986. Deterministic Routing to Buffered Channels, *IEEE Trans. Comm.* COM-34:504–507.
- [Ros86b] ROSS, F. E. 1986. FDDI—A Tutorial, *IEEE Commun. Mag.*, 24:10–17.
- [RoT90] ROSS, K., and TSANG, D. 1990. Teletraffic Engineering for Product-Form Circuit-Switched Networks, *Adv. Appl. Prob.*, 22:657–675.
- [Ryb80] RYBCZYNSKI, A. 1980. X.25 Interface and End-to-End Virtual Circuit Service Characteristics, *IEEE Trans. Comm.*, COM-28:500–509. (Also in [Gre82].)
- [SaH87] SASAKI, G., and HAJEK, B. 1986. Optimal Dynamic Routing in Single Commodity Networks by Iterative Methods, *IEEE Trans. Comm.*, COM-35:1199–1206.
- [San80] SANT, D. 1980. Throughput of Unslotted Aloha Channels with Arbitrary Packet Interarrival Time Distribution, *IEEE Trans. Comm.*, COM-28:1422–1425.
- [SaO82] SARACHIK, P. E., and OZGUNER, U. 1982. On Decentralized Dynamic Routing for Congested Traffic Networks, *IEEE Trans. Automat. Control*, AC-27:1233–1238.
- [Sar82] SARACHIK, P. E. 1982. An Effective Local Dynamic Strategy to Clear Congested Multidestination Networks, *IEEE Trans. Automat. Control*, AC-27:510–513.
- [Sas91] SASAKI, G. 1991. Input Buffer Requirements for Round Robin Polling Systems with Nonzero Switchover Times, *Proceedings of the 29th Allerton Conference on Communication, Control, and Computing*, Monticello, IL.
- [Sch87] SCHWARTZ, M. 1987. *Telecommunication Networks Protocols, Modeling, and Analysis*. Reading: Addison Wesley.
- [ScS80] SCHWARTZ, M., and STERN, T. E. 1980. Routing Techniques Used in Computer Communication Networks, *IEEE Trans. Comm.*, COM-28:539–552.
- [Seg81] SEGALL, A. 1981. Advances in Verifiable Fail-Safe Routing Procedures, *IEEE Trans. Comm.*, COM-29:491–497.
- [Seg83] SEGALL, A. 1983. Distributed Network Protocols, *IEEE Trans. Inform. Theory*, IT-29:23–34.
- [Sha48] SHANNON, C. E. 1948. A Mathematical Theory of Communication, *Bell Syst. Tech J.*, 27:379–423 (Part 1), 623–656 (Part 2). (Reprinted in book form by the University of Illinois Press, Urbana, IL, 1949.)
- [Sie86] SIEBERT, W. M. 1986. *Circuits, Signals, and Systems*. Cambridge, MA: MIT Press; and New York: McGraw-Hill.
- [SiS81] SIDI, M., and SEGALL, A. 1981. A Busy-Tone-Multiple-Access-Type Scheme for Packet-Radio Networks, in G. Payolk (Ed.), *Performance of Data Communication Systems and Time Applications*. New York: North-Holland, pp. 1–10.
- [SoH86] SOLOWAY, S. R., and HUMBLET, P. A. 1986. *On Distributed Network Protocols for Changing Topologies*. Codex Corp.
- [SpG89] SPINELLI, J. M., and GALLAGER, R. G. 1989. Event Driven Topology Broadcast without Sequence Numbers, *IEEE Trans. on Comm.*, 37: 468–474.
- [Spi85] SPINELLI, J. 1985. *Broadcasting Topology and Routing Information in Computer Networks*. (Report LIDS-TH-1470). Cambridge, MA.: MIT.
- [Spi89] SPINELLI, J. M. 1989. Reliable Data Communication in Faulty Computer Networks, *Ph.D. Thesis, MIT Dept. EECS*, June 1989. Also *MIT LIDS Report LIDS-TH-1882*.

- [SpM81] SPROULE, D. E., and MELLOR, F. 1981. Routing, Flow and Congestion Control and the Datapac Network, *IEEE Trans. Comm.*, COM-29:386–391.
- [Sta85] STALLINGS, W. 1985. *Data Computer Communications*. New York: Macmillan.
- [StA85] STUCK, B. W., and ARTHURS, E. 1985. *A Computer Communications Network Performance Analysis Primer*. Englewood Cliffs, NJ: Prentice Hall.
- [Sti72] STIDHAM, S., Jr. 1972. $L = \lambda W$: A Discounted Analogue and a New Proof, *Oper. Res.*, 20:1115–1125.
- [Sti74] STIDHAM, S., Jr. 1974. A Last Word on $L = \lambda W$, *Oper. Res.*, 22:417–421.
- [StK85] STASSINOPOULOS, G. I., and KONSTANTOPOULOS, P. 1985. Optimal Congestion Control in Single Destination Networks, *IEEE Trans. Comm.*, COM-33:792–800.
- [Syz90] SYZMANSKI, T. 1990. An Analysis of Hot-Potato Routing in a Fiber Optic Packet Switched Hypercube, *IEEE INFOCOM '90 Proceedings*, 1:918–925.
- [Taj77] TAJIBNAPIS, W. D. 1977. A Correctness Proof of a Topology Information Maintenance Protocol for a Distributed Computer Network, *Commun. ACM*, 20:477–485.
- [TaK85] TAKAGI, H., and KLEINROCK, L. 1985. Throughput Delay Characteristics of Some Slotted-Aloha Packet Radio Networks, *IEEE Trans. Comm.*, COM-33:1200–1207.
- [Tak86] TAKAGI, H., 1986. *Analysis of Polling Systems*. Cambridge, MA: MIT Press.
- [Tan89] TANENBAUM, A. S. 1989. *Computer Networks*. (2nd ed.). Englewood Cliffs, NJ: Prentice-Hall.
- [TBF83] TOBAGI, F., BORGONOVO, F., and FRATTA, L. 1983. Express-Net: A High Performance Integrated-Services Local Area Network, *IEEE J. Select Areas Comm.*, SAC-1.
- [ThC86] THAKER, G. H., and CAIN, J. B. 1986. Interactions between Routing and Flow Control Algorithms, *IEEE Trans. Comm.*, COM-34:269–277.
- [Tob74] TOBAGI, F. A. 1974. *Random Access Techniques for Data Transmission over Packet Switched Radio Networks*, Ph.D. thesis, UCLA, Computer Science Dept., Los Angeles.
- [ToK75] TOBAGI, F. A., and KLEINROCK, L. 1975. Packet Switching in Radio Channels: Part II: The Hidden Terminal Problem in CSMA and Busy-Tone Solution, *IEEE Trans. Comm.*, COM-23:1417–1433.
- [ToW79] TOWSLEY, D., and WOLF, J. K. 1979. On the Statistical Analysis of Queue Lengths and Waiting Times for Statistical Multiplexors with ARQ Retransmission Schemes, *IEEE Trans. Comm.*, COM-27:693–702.
- [Tsa89] TSAI, W. K. 1989. Convergence of Gradient Projection Routing Methods in an Asynchronous Stochastic Quasi-Static Virtual Circuit Network, *IEEE Trans. Aut. Control*, 34:20–33.
- [TsB86] TSITSIKLIS, J. N., and BERTSEKAS, D. P. 1986. Distributed Asynchronous Optimal Routing in Data Networks, *IEEE Trans. Automat. Control* AC-31:325–331.
- [Tsi89] TSITSIKLIS, J. N. 1989. *Markov Chains with Rare Transitions and Simulated Annealing* Math. of Operations Research, Vol. 14, pp. 70–90.
- [Tsi87] TSITSIKLIS, J. N. 1987. *Analysis of a Multiaccess Control Scheme*, *IEEE Trans. Automat. Control*, AC-12:1017–1020.
- [Tsm78] TSYBAKOV, B. S., and MIKHAILOV, V. A. 1978. Free Synchronous Packet Access in a Broadcast Channel with Feedback, *Problemy Peredachi Inform.* (USSR), 14(4):32–59.
- [Tsm80] TSYBAKOV, B. S., and MIKHAILOV, V. A. 1980. Random Multiple Access of Packets: Part and Try Algorithm, *Problemy Peredachi Inform.* (USSR), 16:65–79.
- [TsR90] TSANG, D., and ROSS, K. 1990. Algorithms to Determine Exact Blocking Probabilities for Multirate Tree Networks, *IEEE Trans. on Comm.*, 38:1266–1271.
- [TsS90] TSITSIKLIS, J. N., and STAMOULIS, G. D. 1990. On the Average Communication Complexity of Asynchronous Distributed Algorithms, *MIT LIDS Report LIDS-P-1986*.

- [Tsy85] TSYBAKOV, B. S. 1985. Survey of USSR Contributions to Random Multiple-Access Communications, *IEEE Trans. Inform. Theory*, IT-31:143–165.
- [Tur86] TURNER, J. 1986. New Directions in Communications (or Which Way to the Information Age), *IEEE Communications Magazine*, 24:8–15.
- [Twe82] TWEEDIE, R. L. 1982. Operator-Geometric Stationary Distributions for Markov Chains with Application to Queueing Models, *Adv. Appl. Probab.*, 14:368–391.
- [Tym81] TYMES, L. 1981. Routing and Flow Control in TYMNET, *IEEE Trans. Comm.*, COM-29:392–398.
- [Var90] VARVARIGOS, E. A. 1990. Optimal Communication Algorithms for Multiprocessor Computers, M.S. Thesis, Dept. of Electrical Engineering and Computer Science, M.I.T., also Center for Intelligent Control Systems Report, CICS-TH-192.
- [Vas79] VASTOLA, K. S. 1979. *A Numerical Study of Two Measures of Delay for Network Routing*, M.S. thesis, University of Illinois, Dept. of Electrical Engineering, Urbana, IL.
- [VvP83] VVEDENSKAYA, N. D., and PINSKER, M. S. 1983. Non-optimality of the Part-and-Try Algorithm, in *Abstracts of the International Workshop of Convolutional Codes, Multiuser Communication*, Sochi, USSR, pp. 141–148.
- [Wal83] WALRAND, J. 1983. Probabilistic Look at Networks of Quasi-reversible Queues, *IEEE Trans. Inform. Theory*, IT-29:825–831.
- [Wal88] WALRAND, J. 1988. *An Introduction to Queueing Networks*. Prentice Hall, Englewood Cliffs, N.J.
- [WaO90] WATANABE, Y., and ODA, T. 1990. Dynamic Routing Schemes for International Networks, *IEEE Communications Magazine*, 28:70–75.
- [Wec80] WECKER, S. 1980. The Digital Network Architecture, *IEEE Trans. Comm.*, COM-28:510–526.
- [Wel82] WELDON, E. J. 1982. An Improved Selective Repeat ARQ Strategy, *IEEE Trans. Comm.*, COM-30:480–486.
- [Whi83a] WHITT, W. 1983. The Queueing Network Analyzer, *The Bell System Technical Journal*, 62:2779–2815.
- [Whi83b] WHITT, W. 1983. Performance of the Queueing Network Analyzer, *The Bell System Technical Journal*, 62:2817–2843.
- [Wie80] WIESELTHIER, J. E., and EPHREMIDES, A. 1980. A New Class of Protocols for Multiple Access in Satellite Networks, *IEEE Trans. Automat. Control*, AC-25:865–880.
- [Wol82a] WOLFF, R. W. 1982. Poisson Arrivals See Time Averages, *Oper. Res.*, 30:223–231.
- [Wol82b] WOLFF, R. W. 1982. Tandem Queues with Dependent Service Times in Light Traffic, *Oper. Res.*, 82:619–635.
- [Wol89] WOLFF, R. W. 1989. *Stochastic Modelling and the Theory of Queues*. Prentice Hall, Englewood Cliffs, N.J.
- [Yum81] YUM, T. P. 1981. The Design and Analysis of a Semidynamic Deterministic Routing Rule, *IEEE Trans. Comm.*, COM-29:498–504.
- [Zan69] ZANGWILL, W. 1969. *Nonlinear Programming: A Unified Approach*. Englewood Cliffs, N.J.: Prentice Hall.
- [Zha89] ZHANG, L. 1989. A New Architecture for Packet Switching Network Protocols, Ph.D. Thesis, Dept. of Electrical Engineering and Computer Science, MIT. Cambridge, MA.
- [ZVK89] ZINKY, J., VICHNIAC, G., and KHANNA, A. 1989. Performance of the Revised Routing Metric in the ARPANET and MILNET, *Proc. of MILCOM '89*.

Index

A

ABM. *See* Asynchronous balanced mode
 ADCCP, 72, 97, 103
 AD. *See* Amplitude modulation
 ANSI. *See* American National Standards Institute
 AM. *See* Amplitude modulation
 ARM. *See* Asynchronous response mode
 ARPANET, 2, 4, 19–20
 ARQ, 84, 86
 flow control, 515
 framing, 87
 routing, 374–76, 404–6, 412
 ARQ. *See* Automatic repeat request
 ASCII code, 21, 31, 58, 86
 ATM. *See* Asynchronous transfer mode
 Abort capability for frames, 89
 Access rights, 31
 Acks, 66, 500
 end-to-end, 115, 16
 Adaptive equalizers, 45, 49, 140
 Addressing, 40, 111, 14
 in TCP, 124–25
 Age field, 422, 425
 Aggregation of queues, 254
 Airline reservation systems, 7, 9, 10
 Allocate message, 500
 Aloha, 275–89, 352
 slotted, 277–87
 stabilized, 282–86
 unslotted, 287–89
 American National Standards Institute, 97
 Amplitude modulation, 48
 Application layer, 31, 32
 Arc, 387, 394
 Arrival rate, 12, 152
 Arrival theorem, 239, 256
 Asynchronous balanced mode, 98–103
 Asynchronous character pipes, 38
 Asynchronous response mode, 98
 Asynchronous transfer mode, 32, 40, 55,
 97, 128–39, 141
 adaptation layer, 135–39
 call size, 132
 congestion, 138–39
 CRC on header, 133–34
 flow control, 138–39
 use of SONE1, 134–35
 virtual channel and path ID, 133
 Attenuation, 56
 Automatic repeat request, 39, 61–80, 272
 ARPANET, 84, 86
 delay analysis, 190
 go back *n*, 72, 81, 190
 packet radio, 347–48
 ring networks, 323
 selective repeat, 81, 84
 stop and wait, 66–71

B

BRAM, 333
 BSC. *See* Binary synchronous
 communication
 Backlog, 276, 279
 Backlog estimation, 283, 286
 Backpressure, 507
 Bandwidth, 50, 51

Bellman's equation, 399
 Bellman-Ford algorithm, 396–400,
 404–410, 424, 481, 492
 Binary exponential backoff, 286–87, 352
 Binary synchronous communication, 87
 Birth-death processes, 184, 215, 261
 Bisynch, 87
 Bit pipes, 20, 23, 38, 65, 86
 intermittent synchronous, 38
 synchronous, 38
 Bit stuffing, 88–90, 145, 320
 Bits, 10
 Black box, 17
 Blocking probability, 180, 185
 Bottleneck link, 526
 Branch exchange heuristic, 444, 448
 Bridge learning, 384
 Bridged local area networks, 382–87
 Bridges, 4, 29, 380
 Broadband ISDN, 5, 11–12, 32, 55,
 128–32
 Broadcasting, 369, 375–76, 433, 485
 Buffer management, 496–98
 Buffer overflow, 496–99
 Burke's theorem, 218, 255
 Burst detection, 60–61
 Bursty sessions, 14–15
 Bus systems, 271, 274, 331–33
 unidirectional, 334–41
 Busy tones, 350–51

C

CCITT, 97
 CRC. *See* cyclic redundancy checks
 CRC-16 polynomial, 64
 CRC-32 IEEE polynomial, 64
 CRC-CCITT polynomial, 64, 98
 CRP. *See* Collision resolution period
 CSMA. *See* Carrier sense multiple access
 CSMA/CD. *See* Carrier sense multiple
 access/collision detection
 Cable TV, 56, 341–42
 Call blocking, 494
 Call-request packet X.25, 119
 Capacity assignment, 439–44
 Capacity, 51, 150
 Capture effects, slotted Aloha, 355
 Carrier offset, 53
 Carrier sense multiple access, 272,
 304–12, 352
 collision avoidance, 333
 collision detection, slotted, 317–18
 collision detection, unslotted, 318–20
 ECFS splitting, 310–12
 nonpersistent, 305
 packet radio, 350–51
 persistent, 305
 P-Persistent, 305, 307
 pseudo-Bayesian stabilization, 307–9
 slotted Aloha, 305–9
 unslotted Aloha, 309–10
 variable delay, 358
 Cellular radio system, 247
 Character based framing, 86–88
 Choke packet, 510
 Circuit switching, 14–17, 180, 185, 379
 Coaxial cable, 7, 56
 Code conversion, 51

Codex networks, 141
 data link control, 518
 flow control, 518
 routing, 378, 417, 476–77
 session identification, 113–15
 Coding. *See* Data compression,
 Encryption, Error detection, or
 Error correction
 Collision resolution, 276–77
 packet radio, 347–49
 in ATM, 138–39 *See also* Flow control
 Collision resolution period, 291, 295–300
 Collision-free set, 345
 Communication channels, 20, 38, 40–57
 analog, 40
 bandpass, 47–52
 digital, 40, 53–54
 voice-grade, 49–50, 52
 Concentrator location problem, 448
 Concentrators, 1
 Congestion control, 27, 116–17
 in ATM, 138–39 *See also* Flow control
 Connected graph, 387, 394
 Connectionless service, 26
 in ATM, 137–39
 Connectivity, 445
 Constrained MST problem, 447
 Convergence sublayer in ATM, 136–39
 Convex set or function, 453, 486
 Convolution, 43
 Convolutional codes, 61
 Correctness, stop-and-wait, 69–70
 Cut-through routing, 373
 Cycle, 387
 Cyclic redundancy check, 61, 64, 140
 on ATM header, 133–34
 end-to-end, 116

D

db. *See* Decibels
 DC component, 47, 48
 DCE. *See* Data communication equipment
 DECNET, 2, 19, 20, 91, 93, 404
 DLE. *See* Data link escape, 87
 DNA. *See* Digital network architecture
 DQDB. *See* Distributed queue dual bus
 DTE. *See* Data terminal equipment
 Data bases, remote access, 7–8
 Data communication equipment, 21, 118
 Data compression, 10, 31
 Data link control, 20, 23–24, 37–40,
 57–110, 271
 correctness, 76–79, 140
 standards, 97–103, 141
 Data terminal equipment, 21, 118
 Datagram:
 networks, 115
 routing, 16
 service, 26
 Datagram routing, 363
 Deadlock, 497–99
 Decibels, 51
 Decomposition of queues, 254
 Deflection routing, 372
 Delay, 13
 CSMA, 308–9
 CSMA/CD, 318, 360
 ECFS splitting algorithm, 300–301

Delay *cont.*
 Fiber distribution,
 processing, 15
 propagation, 15
 queueing, 15
 satellite systems, 15
 slotted Aloha, 284–85
 token buses, 331–32
 token ring, 324–28
 transmission, 15
 Demand sharing, 274
 Descent direction, 449
 Designated port, 343
 Detailed balance, 216
 216, 217, 218–219
 Digital network,
 Digraph, 392
 Dijkstra's algorithm,
 Directed arc, 392
 Directed cycle, 392
 Directed graph, 392
 Directed walk, 392
 Directory assistance,
 Disconnect problem,
 Distributed algorithm,
 Distributed queue,
 Distributed queue dual bus,
 Distributed span,
 Drift, 264
 Drift analysis,
 CSMA, 306
 ECFS splitting,
 slotted Aloha,
 Dynamic programming,
 Dynamic routing,
 Dynamic window

E

EXT (end-to-end),
 Electronic mail,
 Embedded channel,
 Encryption, 31
 End-to-end CRC,
 End-to-end delay,
 Entropy, 91, 154
 Ergodic system,
 Erlang B formula,
 Erlang C formula,
 Error correction,
 Error detection, 347–49, 472
 bursts, 60–61
 random strings,
 Error recovery,
 in TCP, 125–27
 transport vs.,
 Essau-Williams,
 Ethernet, 4, 48–52, 274–280
 degradation,
 Exponential distribution,
 Expressnet, 334–34
 External site, 19

F

FDDI. *See* Fiber distributed
 FDM. *See* Frequency
 multiplexing
 Facsimile, 8
 Feasible direction,

Delay *cont.*
 Fiber distributed data interface, 329–30
 processing, 150
 propagation, 150
 queuing, 150
 satellite systems, 314–15
 slotted Aloha, 284–86
 token buses and polling systems,
 331–32
 token ring, 324–25
 transmission, 150
 Demand sharing, 274
 Descent direction, 456
 Designated port, 383
 Detailed balance equations, 182, 214,
 216, 218, 261, 263
 Digital network architecture, 404
 Digraph, 394
 Dijkstra's algorithm, 401–3, 481
 Directed arc, 394
 Directed cycle, 394
 Directed graph, 394
 Directed walk, 394
 Directory assistance, 30
 Disconnect command, 100
 Disconnect protocols, 103–10
 Distributed algorithms, 28, 32–34, 39,
 139
 Distributed queue dual bus, 335–39, 353
 Distributed shortest path construction,
 404–10
 Distributed spanning tree construction,
 392
 Drift, 264
 Drift analysis:
 CSMA, 306
 FCFS splitting algorithm, 300
 slotted Aloha, 280
 Dynamic programming, 395
 Dynamic routing, 16–17, 436–38, 491
 Dynamic window adjustment, 510

E

EXT (end of text), 87
 Electronic mail, 8, 9
 Embedded chan, 262
 Encryption, 31
 End-to-end CRC, 116
 End-to-end acks, 115–16, 119
 End-to-end windows, 506–8, 515–17
 Entropy, 91, 140
 Ergodic system, 156
 Erlang B formula, 179, 167
 Erlang C formula, 175, 267
 Error correction, 52, 61, 140
 Error detection, 39, 53, 57–64, 140
 bursts, 60–61
 random strings, 60–61
 Error recovery, 23, 32, 115–16
 in TCP, 125–27
 transport vs. network layer, 117–18
 Essau-Williams algorithm, 448
 Ethernet, 4, 48, 53, 274, 286, 317–20
 degradation with size and speed, 334
 Exponential distribution, 164, 266
 Expressnet, 339–41
 External site, 19

F

FDDI. *See* Fiber distributed data interface
 FDM. *See* Frequency division
 multiplexing
 Facsimile, 8
 Feasible direction, 456

Feedback, multiaccess:
 delayed, 303–4
 immediate, 276
 Feedback shift register, 62
 Fiber distributed data interface, 326–30,
 334, 339, 353
 delay, 329–30
 throughput, 330
 File transfer, 13
 Filtering, 41–53
 First derivative length, 452
 First-come first-serve splitting, 293–304
 CSMA, 310–12
 Flags, 88–90, 320
 Flooding, 369, 419–25, 432, 490
 Flooding with sequence numbers, 420
 Flow control, 25–29, 30, 116–17, 158
 combined with optimal routing,
 519–24
 fairness, 498–99, 505, 507, 524–29
 in ARPANET, 515
 in ATM, 138–39
 in Codex network, 518
 in PARIS network, 518
 in SNA, 517
 in TCP, 127–28
 in TYMNET, 517
 in X.25, 519
 input rate adjustment, 519–29
 max-min, 524–29, 534
 means, 494–95
 objectives, 496–99
 optimal, 519–24
 satellite links, 506–8, 517
 transport layer, 509
 window, 500–510
 Flow deviation method, 458–64
 Flow models, 433–37
 Floyd-Warshall algorithm, 403–4
 Forwarding database, 383
 Fourier transforms, 44, 45
 Fragment, 389
 Fragmentation, internet protocol, 122
 Frames, 23, 52, 65
 internet protocol, 122
 maximum fixed length, 97
 maximum length, 140–41
 maximum variable length, 93–97
 Framing, 39, 86–97
 bit-oriented, 88–90
 character based, 86–88
 length fields, 90–92
 overhead, 87, 89–90, 90–92
 with errors, 92–93
 Frank Wolfe method, 458–64, 471, 488
 Free-for-all multiaccess, 272
 Frequency division multiplexing, 52, 151,
 170, 177, 194
 in satellite channels, 273
 Frequency response, 43–46
 Front end, 1

G

G/G/1 queue. *See* Queuing System
 Gateways, 4, 28–29, 120, 380
 Generator polynomials, 62–64, 142
 Global balance equations, 168, 260, 263
 Go back *n* ARQ, 72–81, 113, 143–44,
 501
 correctness proof, 76–79
 efficiency, 80–81, 144
 ideal, 82
 rules, 74–76
 rules with mod *m*, 80
 Gradient project method, 459, 467–77,
 486–87, 489
 Graph, 387

Graphics, 8, 13
 Guaranteed data rate, 494

H

HDLC, 39, 72, 97–103
 faulty initialization, 106
 Header:
 frame, 23, 39, 65
 packet, 25
 Hessian matrix, 465
 Hierarchy of modules, 17
 Homenets, 341–42
 Horizontal and vertical parity checks,
 58–59
 Hot potato routing, 372
 Hybrids, 56

I

IEEE 802 standards, 48, 320, 323–24,
 332–33, 335
 IMP. *See* Interface message processor
 IP. *See* Internet protocol
 ISDN. *See* Integrated services digital
 network
 ISO. *See* International Standards
 Organization
 Idle fill, 38, 90, 321
 Image transmission systems, 9
 Implicit tokens, 333
 Impulse response, 42
 Incident arc, 387
 Infinite node assumption, 276
 Initialization protocols, 147
 ARQ, 103–10
 balanced, 107–9
 fault in HDLC, 106
 with link failures, 103–9
 master-slave, 104–7
 with node failures, 109–10
 in TCP, 126
 use of stop-and-wait, 104
 Integrated services digital network, 5,
 11–12, 54–56
 basic service, 55
 primary service, 55
 Interactive sessions, 13, 15
 Interface message processor, 2
 Internal signaling network, 55
 International Consultative Committee on
 Telegraphy and Telephony. *See*
 CCITT
 International Standards Organization, 19,
 97
 terminology, 30
 Internet protocol, 40, 120–23, 141
 addressing, 121–22
 datagrams, 120
 fragmentation, 122
 time to live, 123
 Internet sublayer, 28–29
 Internetworking, 28–29
 Intersymbol interference, 42, 45–47, 49
 Isarithmic method, 508

J

Jackson's theorem:
 closed networks, 234
 heuristic explanation, 227
 with limit on the number of customers,
 256
 multiple classes of customers, 231
 open networks, 223
 state dependent rates, 230

K

Kleitman's algorithm, 445-47
Kruskal algorithm, 390, 448

L

LAN. *See* Local area network
LAPB, 72, 97-103, 118
Last-come first-serve splitting algorithm, 302-3
Layering, 17-32, 35
Leaky bucket, 245, 511-15, 535
Linear codes, 59
Linear systems, 41-46, 140
Little's theorem, 152, 62, 250-51
slotted Aloha, 280
Liveness:
go back n , 77-78
stop-and-wait, 69-70
Load sharing, 8
Local access network, 438
Local access network design, 448, 51
Local area networks, 4, 7, 29, 56,
271-72, 317-42, 352, 53
Local loop, telephone network, 54, 56

M

M/D/1 queue. *See* Queueing system
M/G/1 queue. *See* Queueing system
M/G/ ∞ queue. *See* Queueing system
M/M/1 approximation. *See* Networks of queues
M/M/1 queue. *See* Queueing system
M/M/1/m queue. *See* Queueing system
M/M/ ∞ queue. *See* Queueing system
M/M/m queue. *See* Queueing system
M/M/m/n queue. *See* Queueing system
MAC. *See* Medium access control
MAN. *See* Metropolitan area network
MSAP, 333
Manchester coding, 48
Markov chain:
aperiodic, 260
continuous-time, 262-63
discrete-time, 259-62
FCFS splitting algorithm, 297
instability, 265
irreducible, 260
M/M/1 system, 166
M/M/ ∞ system, 177
M/M/m system, 174
M/M/m/n system, 179
multidimensional, 180-86
queueing networks, 223
reversible, 182
slotted Aloha, 279, 353
stability, 264
time reversible, 215
Max-min fairness, 328
Max-min flow control, 524, 29, 534
Mean residual service time, 187
Mean value analysis, 238, 40
Media access control, 24, 27, 29, 271
Memoryless property, 165
Message switching, 16
Messages, 10
arrival rate, 12
delay, 13
length, 12-13
ordering, 13
Metering, 214, 436
Metropolitan area networks, 271-72,
326-30, 333-42
Micro-processors, 5
Microwave channels, 56

Min-hop path, 370
Minimum distance, 60
Minimum weight spanning tree, 389, 484
Modems, 21, 38, 40, 48-52
Modularity, 17
Modulation, 140
amplitude, 48-49
phase-shift keying, 50
quadrature amplitude, 49, 52
Module, 17
Modulo arithmetic, 58
Multiaccess communication, 24, 195,
271, 362
canonic reservation system, 312
Multidrop telephone lines, 271, 274, 331,
342
Multiplexers, 1
Multiplexing, in TCP, 124-25
Multicac bus systems, 271, 274

N

NRM. *See* Normal response mode
NRZ code, 42, 48
Nak, 66
Network layer, 25, 29
point to point protocols, 110-20
Networks of queues:
acyclic, 221
closed, 233
Jackson's theorem, 212, 221, 434
Kleinrock independence approximation,
212, 221, 434, 440
M/M/1 approximation, 213, 221, 434,
440, 453, 476
multiple classes of customers, 230
state dependent service rates, 229
tandem queues, 209, 220
with multiple classes of customers, 222
Newton's method, 467
Node of a graph, 387
Node-by-node windows, 501, 6, 517
Noise, 45, 51
Nonlinear distortion, 53
Normal response mode, 98-101
Nyquist criterion, 47

O

OD pair. *See* Origin-destination pair
OSI. *See* Open systems interconnection
On-off flow, 11
Open systems interconnection, 19-20, 35
Optical fibers, 6, 7, 15, 56, 326, 333
Optimal routing:
algorithms, 455-75
dynamic, 436, 38, 491
formulation, 433, 36
Origin-destination pair, 434

P

PARIS network, 392, 518
PERT networks, 395
PSK. *See* Phase-shift keying
Packet blocking, 495
Packet discarding, 495
Packet radio nets, 57, 273, 275, 286,
344-51
collision resolution, 347, 49
TDM, 346-47
transmission radii, 349, 50, 353
Packet scheduling, 495
Packet switching, 131
why used in ATM, 16

Packets, 10, 39
ordering, 13, 86
Parity check codes, 59-64, 140
Parity checks, 58
Partial balance equations, 262
Path in a graph, 387
Peer process (modules), 18, 23, 25, 28,
39
Periodic routing updates, 422
Permit, 117, 500
Permit (window) in TCP, 127
Phase jitter, 53
Phase shift keying, 50
Physical layer, 20-23, 37, 38, 40-57
Piggybacked requests, 67
Pipelining, 94-96
Point to point protocols, 37
Poisson distributing, 164, 266
Poisson process, 11, 164, 65, 172-73,
275
combined, 244
merged, 165
properties, 243
split, 165, 245
Poll final bit, 99
Pollaczek-Khinchin formula, 186
Polling, 274, 331, 32, 342-44
generalized, 342-44
hub, 331
queueing analysis, 195-203
Polynomial division, 62
Port, 382
in TCP, 124
Positive definite matrix, 465
Positive semidefinite matrix, 465
Presentation layer, 31
Prim-Dijkstra algorithm, 390, 448, 484
Primitive polynomials, 64
Priorities, FDDI, 326, 27
Priority queueing system. *See* Queueing system
Product form, 182
Product form solution, 220, 223, 25
Promiscuous mode, 382
Protocols. *See* Automatic repeat request.
Distributed algorithms, HDLC,
X.25, X.21
Pseudo-Bayesian algorithm, 283-86, 352
CSMA slotted Aloha, 307-9
CSMA unslotted Aloha, 359
Pure Aloha, 287-89

Q

QAM. *See* Quadrature amplitude
modulation
Quasistatic assumption, 177
Queueing system:
closed, 158, 233, 256, 58
discrete-time M/M/1, 246
G/G/1, 206, 9, 253, 269
last-come first-serve, 232
last-come first-serve M/G/1 system,
253
M/G/1 system, 267
with arbitrary order of service, 248
with batch arrivals, 251
with busy period overhead, 251
with delay for new busy period, 252
with vacations, 192-95, 252, 268
M/G/m/m, 179
M/G/ ∞ , 252
M/M/1, 162, 73, 175, 216, 266
shared service, 247
with multiple classes of customers,
258
with state-dependent rates, 246
M/M/1/m, 247, 255

M/M/2 with vacations,
247
M/M/ ∞ , 177, 253, 254
M/M/m, 174, 253, 254
M/M/m/n, 179, 253
networks, 223, 224
nonpreemptive, 253
229, 253
with phase polling,
165, 253, 254
preemptive, 253
229, 253
priority, 253
processor sharing,
253
reservation, 253
time reversibility, 253

R

RN. *See* Random network
RS-232C connector, 27, 28
Radio channels, 27, 28
radio nets, 57
Randomization, 27, 28
Rate control, 27, 28
Real time computation, 5
Real time communication, 5
Reassembly, 27
Receive network:
supervisory, 27, 28
X.25 packet, 27
Receiver-ratio:
supervisory, 27, 28
X.25 packet, 27
Register, 27, 28
Regular chain, 27
Reject supervisory, 27
Reject X.25 packet, 27
Reliability, 27, 28
Repeaters, 53-54
Request number, 27
Reservation system:
canonic, 312, 313
exhaustive, 313
gated, 195
limited service, 27, 28
multiclass, 27, 28
multiuser, 27
partially gated, 195
satellites, 313
single-user, 313
Reset command, 27
Reversibility, 27, 28
Ring networks, 27, 28
Ringing, 28
Round robin scheduling, 27
Router, 38
Routing, 25-27, 38, 42
adaptive, 38, 42
asynchronous, 27, 28
broadcast, 27
centralized, 38
distributed, 38
hierarchical, 38, 42
in the ARPANET, 27
in Codev network, 27
in circuit switched,
in SNA, 27
in the TYMNET, 27
interaction with,
interconnection, 38
main issues, 38
nonhierarchical, 38
optimal, 38, 42
oscillations, 38
shortest path, 38
static, 38

- M/M/2 with heterogeneous servers, 247
- M/M/ ∞ , 177-78, 216, 247
- M/M/m, 174-77, 216, 266
- M/M/m/m, 175, 179, 216, 267
- networks. *See* Networks of queues
- nonpreemptive priority, 203-5, 249, 50, 269
- with phase-type distributions, 232
- polling, 160, 195, 268
- preemptive resume priority, 205-6, 249, 269
- priority with multiple servers, 249
- processor sharing, 232
- reservation, 195-204, 268
- time reversible, 214

R

- RN. *See* Request number
- RS-232-c interface, 21, 22
- Radio channels, 56, 271. *See also* Packet radio nets
- Randomization, 214, 222, 436
- Rate control schemes, 510-15
- Real time computation, 8
- Real time control, 13
- Reassembly, 29
- Receive-not-ready:
 - supervisory frame, 99-100
 - X.25 packet, 119
- Receive-ready:
 - supervisory frame, 99-100
 - X.25 packet, 119
- Register insertion rings, 330-31
- Regular chain, 262
- Reject supervisory frame, 99-100
- Reject X.25 packet, 119
- Reliability, 9, 13, 29, 443-48
- Repeaters, 53-54
- Request number, 67
- Reservation system:
 - canonic multiaccess system, 312
 - exhaustive, 195
 - gated, 195
 - limited service, 201, 249
 - multiaccess, 272, 312-14
 - multiuser, 198
 - partially gated, 195
 - satellites, 313-17
 - single user, 196
- Reset command, 103
- Reversibility, 214, 21
- Ring networks, 320-31
- Ringings, 48
- Round-robin scheduling, 495
- Router, 480
- Routing, 25-29, 363-491
 - adaptive, 368, 410, 17, 455
 - asynchronous, 375, 404-10, 425
 - broadcast, 487
 - centralized, 367, 418
 - distributed, 367
 - hierarchical, 379-82
 - in the ARPANET, 374
 - in Codex network, 476-77
 - in circuit switching networks, 379
 - in SNA, 378
 - in the TYMNET, 376
 - interaction with flow control, 365
 - interconnected network, 379-87
 - main issues, 365-68
 - nonhierarchical, 380-82
 - optimal, 372, 433, 451, 75
 - oscillations, 371, 374, 410-17
 - shortest path, 374, 76, 387, 417
 - shortest path first, 375, 382
 - static, 368

- Routing *cont.*
 - tables, 365, 375, 76, 377, 382
 - with frequently changing topology, 491

S

- SDLC, 72, 97-103
- SN. *See* Sequence number
- SNA. *See* System network architecture
- SONET, 54, 141
 - use in ATM, 134, 35
- SPTA. *See* Shortest path topology algorithm
- STX (start of text), 87
- SYN (synchronous idle), 86
- Sampling theorem, 46, 47, 51
- Satellite channels, 56-57, 271, 273-74, 313, 17, 352-53
- Saturated cut method, 81, 84, 140-41, 444
- Scheduling for multiaccess, 272 (*See also* Reservations)
- Segmentation and reassembly, in ATM, 136, 39
- Selective repeat ARQ, 144
 - ideal, 82
- Selective-reject supervisory frame, 99-100
- Sequence number, 66
- Service rate, 152
- Session holding time, 12
- Session identification, 111-14
- Session initiation, 31
- Session layer, 30, 31
- Sessions, 11, 14
- Shannon capacity theorem, 50-51, 53, 140
- Shortest path problem, 394
- Shortest path routing, 370
- Shortest path spanning tree, 400
- Shortest path topology algorithm, 425, 33, 483
- Signal constellations, 50, 51
- Signal power, 51
- Simulated annealing, 443
- Sliding window, 72 (*See also* Go back n ARQ)
- Slotted Aloha:
 - capture, 355
 - delay, 354
 - stability, 353
- Slotted frequency-division multiplexing, 194
- Slotted multiaccess systems, 275-88
- Slotted ring, 330
- Socket in TCP, 124
- Solid state technology, 5
- Source coding theorem, 91, 92, 140
- Source routing, 385, 87
- Spanning tree, 369, 388
- Spanning tree routing, 383-85
- Spanning tree topology design, 447-48
- Splitting algorithms, 289, 304
 - first-come first-serve, 293, 304, 352
 - LCFS, 302, 3
 - round robin, 304
 - with delayed feedback, 303, 4
 - zero or positive feedback, 342-44
- Stability:
 - CSMA, 307, 12
 - datagram networks, 410-14
 - FCFS splitting algorithm, 300
 - shortest path routing algorithm, 410-417
 - slotted Aloha, 281-86, 352
 - unslotted Aloha, 288, 359
 - virtual circuit networks, 414-17

- Stack algorithms, 291 (*See also* Tree algorithms)
 - unblocked stack, 293
- Standardization, 19-20, 32
- Star configuration for rings, 324
- Stationary distribution, 260, 263
- Statistical multiplexing, 113, 150, 170, 176, 214
- Steepest descent method, 465
- Stop-and-wait, 142-43
 - ARQ, 66, 69
 - correctness proof, 69, 70
 - in initialization protocols, 104
- Store-and-forward switching, 14-17
- Strongly connected graph, 394
- Subgraph, 388
- Sublinear convergence rate, 461
- Subnet, 2-3, 10
- Subnet design, 439-48
- Supercomputers, 8
- Supervisory frames, 99, 100
- System network architecture (SNA), 19, 20
 - explicit route control, 378
 - flow control, 501, 517
 - path control layer, 378
 - routing, 378
 - SDLC, 72, 97-103
 - session level pacing, 517
 - transmission control layer, 378
 - transmission group control, 378
 - virtual route control, 378, 517
 - virtual route pacing scheme, 501, 517

T

- TCP. *See* Transport control protocol
- TCP/IP. *See* Internet protocol, Transport control protocol
- TDM. *See* Time-division multiplexing
- TP4 (ISO transport protocol), 128
- T1 carrier, 53-54
- T3 carrier, 54
- TYMNET, 2, 4, 19-20, 141
 - flow control, 517
 - routing, 376-78
 - session identification, 112-13
- Technology:
 - communication, 6-7
 - computer, 5, 6
- Three army problem, 33-34, 71
- Throughput:
 - CSMA FCFS splitting, 311
 - CSMA slotted Aloha, 306
 - CSMA unslotted Aloha, 310
 - CSMA/CD:
 - slotted, 318
 - unslotted, 319
 - FCFS splitting algorithm, 300-301
 - slotted Aloha, 282-83
 - token buses and polling systems, 331-32
 - unslotted Aloha, 288
- Time invariance, 41
- Time sharing system, 160, 236
- Time to live, Internet protocol, 123
- Time window flow control, 511
- Time-division multiplexing, 52, 151, 170, 177, 194, 214, 273, 277
 - for packet radio, 346-47
- Time-outs, 67, 70, 73
- Token buses, 331, 33
 - degradation with size and speed, 334
 - implicit tokens, 333
- Token rings, 4, 320, 30
- Topological design, 437-51, 464
- Topology, 418

Trailer frame, 23, 39, 65
 Transition rate, 262
 Transmission delay, *See* Delay
 Transparent mode, 87
 Transport control protocol, 40, 124–28, 141
 addressing, 124–25
 error recovery, 117–18, 125–27
 flow control, 127–28
 initialization, 126
 multiplexing, 124–25
 Transport layer, 29–30, 123–28
 standards, 123–24
 Transport layer flow control, 509
 Tree, 388
 Tree algorithms, 290–93, 352
 Truncation of Markov chains, 182, 254
 Twisted pair, 7, 56

U

Unary-binary encoding, 91, 114

Undetectable errors, 60, 64, 114–15
 Unnumbered acks, 100
 Unnumbered frames, 100
 Unslotted Aloha, 287–89
 precise feedback, 356
 throughput, 357
 Unstable equilibrium for Aloha, 281
 Utilization factor, 157, 169

V

VLSI, *See* Very large scale integration
 Vacation Queueing system, *See* Queueing system
 Very large scale integration, 5
 Video conferencing, 12
 Virtual channel number, X.25, 118
 Virtual channels, 111
 Virtual circuit routing, 16, 26, 111, 116, 363, 476–77, 490

Voice:
 digitized, 11
 packetized, 13–14
 Voice mail, 8
 Voice network, 5, 8
 Voice-grade, *See* Communication channels

W

WAN, *See* Wide area network
 Walk in a graph, 387
 Wide area network, 4, 6
 Window (permit) in TCP, 127
 Window flow control, 117, 158, 500–10

X

X.21 interface, 21–22, 118
 X.25 standard, 40, 118–20, 500–10
 LAPB, 97–103