A key parameter of the ring is the *walk time* W_T, which is the time it takes a token to make a complete circuit of the ring. It can be shown that the walk time in an n-node system is given by

$$W_T = (n-1)D_B + L + T_{\text{prop}} \tag{6.8}$$

where

n	Number of nodes in the ring.
D_B	Station delay; delay caused by each node.
L	Buffer latency.
T_{prop}	Message propagation time round the ring.

Schedulability analysis. We make the assumption that the traffic consists of a periodic load whose deadline is upper-bounded by the period. Scheduling packets for transmission is different from scheduling tasks in the following respects.

- A packet transmission cannot be preempted and then resumed without penalty; if a packet transmission is interrupted, it has to be retransmitted all over again.
- Overhead is incurred in transmitting a message. It is not just the message bits that are transmitted, but also a header (consisting of the SD, AC, ED, and destination and source fields) and a trailer associated with each packet.
- Since the system is distributed, decisions as to which packet has the highest priority may be made on the basis of outdated information.

Example 6.17. Consider the token ring shown in Figure 6.35. Node n_1 has traffic generated at periods of 5, 6, and 10, respectively; n_2 periods of 5, 9, and 11, n_3 periods of 4 and 6, and n_4 of period 10. Table 6.3 shows the priorities of the packets[6] awaiting transmission on each node along with their arrival time.

Consider what happens when the token flows past the four nodes at times 6, 7, 9, and 10, respectively. At time 6, node n_1 writes the priority of its most important packet on the reservation field. At time 7, n_2's highest priority is the same, so it does not overwrite the reservation field. At time 9, when the token goes past it,

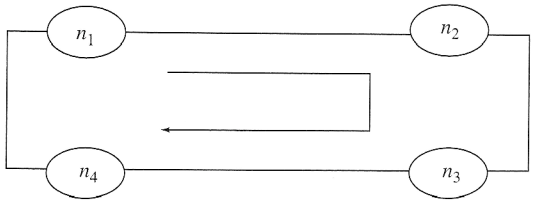

FIGURE 6.35
Token ring for Example 6.17.

[6]The lower the priority number, the higher the priority.

TABLE 6.3
Packet priorities and arrival times

Node	(Period, arrival time)
n_1	(5, 5), (6, 6), (10, 10)
n_2	(5, 5), (9, 9), (11, 11)
n_3	(5, 5), (4, 10)
n_4	(10, 10)

n_3's highest priority is also the same, and so again the token reservation bits are not changed. The reservation bits are also unchanged by n_4. Suppose the token returns to n_1 at time 12. The reservation bits allow transmission of a packet with period 5. Since n_1 has a packet of this period, it is allowed to transmit. At this time, though, it is not n_1 but n_3 that has the highest-priority packet (of period 4). However, n_3 will not have a chance to update the reservation bits until the next transit of the token, and so the decision to allow n_1 to transmit is based on outdated information.

We can now write the necessary and sufficient conditions for the set of tasks to be schedulable. Let d_i be the deadline associated with task T_i; we assume $d_i \leq P_i$. b_i is the maximum time for which a T_i packet can be blocked.

Theorem 6.4. The task set $T_1, T_2, \ldots, T_n$ is schedulable iff for all $i = 1, \ldots, n$, there is some t, $0 < t \leq d_i$ such that

$$\sum_{j=1}^{i} e_j \left\lceil \frac{t}{P_j} \right\rceil + \text{System overhead} + b_i \leq t \tag{6.9}$$

Proof. This is virtually identical to the corresponding result for uniprocessor scheduling in Section 3.2.1 (in Handling Critical Sections) and therefore the proof is omitted.

The necessary and sufficient conditions for schedulability require us to know e_j, P_j, b_j, and the system overhead. Of these, P_j is defined by the application, and the system overhead is defined by the system. This leaves e_j and b_j.

e_j is the execution time associated with sending a message of T_j. It has three components.

1. The time it takes to capture the token when the node has the highest-priority message. Even when a correct reservation has been made and blocking by a lower-priority packet does not occur, this can be as large as W_T per packet. For example, the node that currently owns the token may have the highest priority message to transmit next, but it still needs to send the token through all the other nodes before it can send out the next packet.
2. The time it takes to transmit the message. Suppose we are given that T_j transmits a message of up to m_j bits every P_j seconds. Let m_{enc} be the number of bits in the encapsulation part of each packet (i.e., the portion of the packet that is

not the message bits). Then, each packet can carry up to $m_{\text{payload}} = P_{\max} - m_{\text{enc}}$ message bits. To transmit m_j bits of message thus requires $\lceil m_j/m_{\text{payload}} \rceil$ packets. The total number of bits transmitted for a total of m_j message bits is thus $m_j + \lceil m_j/m_{\text{payload}} \rceil m_{\text{enc}}$ bits. If the ring can transmit at the rate of b bits a second, this takes a total of

$$\tau_{\text{trans}} = \frac{m_j + \lceil m_j/m_{\text{payload}} \rceil m_{\text{enc}}}{b} \tag{6.10}$$

seconds.

3. The time it takes to transmit the token when the packet transmission is over. This is given by τ_{token}, which is a system parameter.

Denote by τ_{SA} the time taken for the sending node to receive the SA part of the packet back on the ring. Now, consider two cases.

Case 1. The message fits within one packet. If $m_j + m_{\text{enc}} \geq W_T + \tau_{\text{SA}}$, the transmitting node will have received back the header of its own packet before it finishes transmitting the packet, and recognized it as its own. Hence, the moment that packet transmission ceases, a new token can be issued. For this case, we therefore have $e_j = W_T + \tau_{\text{trans}} + \tau_{\text{token}}$. If, on the other hand, $m_j + m_{\text{enc}} < W_T + \tau_{\text{SA}}$, then the transmitting node will have to wait until it gets the SA field of its own packet. For this case, $e_j = 2W_T + \tau_{\text{SA}} + \tau_{\text{token}}$. We can therefore write:

$$e_j = \begin{cases} W_T + \tau_{\text{trans}} + \tau_{\text{token}} & \text{if } m_j + m_{\text{enc}} \geq W_T + \tau_{\text{SA}} \\ 2W_T + \tau_{\text{SA}} + \tau_{\text{token}} & \text{otherwise} \end{cases} \tag{6.11}$$

Case 2. The message fits within multiple packets. The reasoning is the same as for Case 1, except that we have to account for multiple waits for the token. The packets will each be of size $P_{\max}$, except possibly the last one, which may be smaller. Let us make the approximation that all the packets are of size $P_{\max}$. Following exactly the reasoning in Case 1, we can write:

$$e_j \approx \begin{cases} m_j + \lceil m_j/m_{\text{payload}} \rceil (\tau_{\text{trans}} + W_T + \tau_{\text{token}}) & \text{if } W_T + \tau_{\text{SA}} \leq P_{\max} \\ \lceil m_j/m_{\text{payload}} \rceil (2W_T + \tau_{\text{SA}} + \tau_{\text{token}}) & \text{otherwise} \end{cases} \tag{6.12}$$

Deriving b_j, the maximum time for which a T_j packet can be blocked, is somewhat simpler. The worst-case blocking occurs if a higher-priority packet arrives just after the reservation bits associated with the a lower-priority transmission have gone past it, and there is another lower-priority packet that has been successful in setting its reservation bits. We saw an illustration of this in Example 6.17. In such a case, the higher-priority packet will have to wait until after both the ongoing and the following transmission have completed. From this, and using reasoning similar to that of the derivation for e_j, it is possible to show that

$$b_j \leq \begin{cases} 2(P_{\max} + \tau_{\text{token}}) + W_T & \text{if } W_T + \tau_{\text{SA}} \leq P_{\max} \\ 2(W_T + \tau_{\text{SA}} + \tau_{\text{token}}) + W_T & \text{otherwise} \end{cases} \tag{6.13}$$

Some implementation issues. In our derivation, we have assumed that there are enough reservation bits to express the full range of priority levels. The 802.5 standard specifies three priority bits; that is, it can support up to eight priority levels. If the number of task priority classes is greater than eight, the only recourse is to map multiple task-priority classes into the same ring-priority levels. This can cause additional blocking.

Example 6.18. Suppose we have a total of 16 task priority classes. Since we only have eight priority levels in the ring, we can group priority classes $(2i - 1)$ and $2i$ into ring-priority level i. Thus, priority classes 1 and 2 will be grouped into ring-priority level 1; classes 3 and 4 into ring-priority level 2, and so on. As the ring gives the same priority to both classes 3 and 4, and this may cause a class-3 packet to be blocked by a class-4 packet.

If possible, the implementation should therefore be modified to allow additional priority levels to be supported. In retrospect, it is unfortunate that the 802.5 standard specifies only three bits for priority. Allocating just a few additional bits would have made the protocol much more useful for real-time systems.

Another issue is $P_{\max}$, the maximum packet size. If this is very small, the overhead associated with the encapsulating bits will be dominant. If it is very large, this can increase packet blocking. $P_{\max}$ must be chosen carefully, based on the designer's knowledge of the application.

6.3.3 Stop-and-Go Multihop Protocol

The stop-and-go protocol is another technique to meet hard deadlines on packet-delivery times. Unlike the previous algorithms, it is a multihop packet delivery algorithm (i.e., there is not necessarily a direct link between source and destination).

The algorithm meets hard deadlines by assigning fractions of the available bandwidth on each channel to several traffic classes in such a way that the time it takes to traverse each of the hops from source to destination is bounded. The upper bound to the overall packet-delivery time is then simply the sum of the upper bounds of each hop. The algorithm also bounds the demand for buffer space.

The concept of the frame is central to this algorithm. A *frame* is defined as an interval of time. Frames are associated with network links and are not synchronized across links. There can be multiple frame types, each defined as a different interval of time. We can visualize the generation of a virtual `frame-begin` signal at the input end of each link at appropriate times. This signal travels down the link and defines the beginning of the frame at each point at which it arrives. That is, the frame-beginning time varies from point to point along the link. An instance of a frame of type f_i ends when the next instance of f_i begins.

Each frame type is associated with a traffic class. When a packet associated with frame type f_i, called a type-i packet, arrives at an intermediate node n en route to its destination, it is held by node n at least until the beginning of the next instance of its frame f_i, and is transmitted during that frame. As long as we can suitably bound the number of packets associated with each frame type, we can

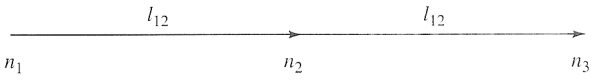

FIGURE 6.36
A two-hop path.

ensure that there will be time enough in each frame for every packet associated with that frame type to be transmitted during it.

Example 6.19. Consider a type-1 packet, that is, one associated with frame type f_1, which must travel from node n_1 to node n_3 through node n_2. See Figure 6.36. There are two hops to the path taken by the packet. Let the propagation time over the link $\ell_{12} = n_1 \rightarrow n_2$ and $\ell_{23} = n_2 \rightarrow n_3$ be τ_{12} and τ_{23}, respectively. Figure 6.37 shows frames of type f_1 at the beginning and end of ℓ_{12} and ℓ_{23}, respectively.

In what follows, we will assume, without loss of generality, that $f_1 > f_2 > \cdots > f_{N_f}$, where there are N_f frame types.

THE PROTOCOL. The stop-and-go protocol is a distributed algorithm; each node works independently without central control. A type-i packet that arrives at node n_j, and must be retransmitted by that node, becomes eligible for transmission only on the beginning of the following f_i frame. All nodes eligible for transmission are served in nonpreemptive priority order, with the type-i packet having priority over all type-k packets for $k < i$. The node is idle (i.e., does not transmit) only when there are no packets left to transmit.

Example 6.20. Consider a node n with one incoming and one outgoing link. There are two traffic types. Figure 6.38 shows the arrival of several packets, when they become eligible for transmission, and when they are actually transmitted. Class-1 frames at the outgoing link begin at epochs α, β, γ, δ, and class-2 frames at epochs

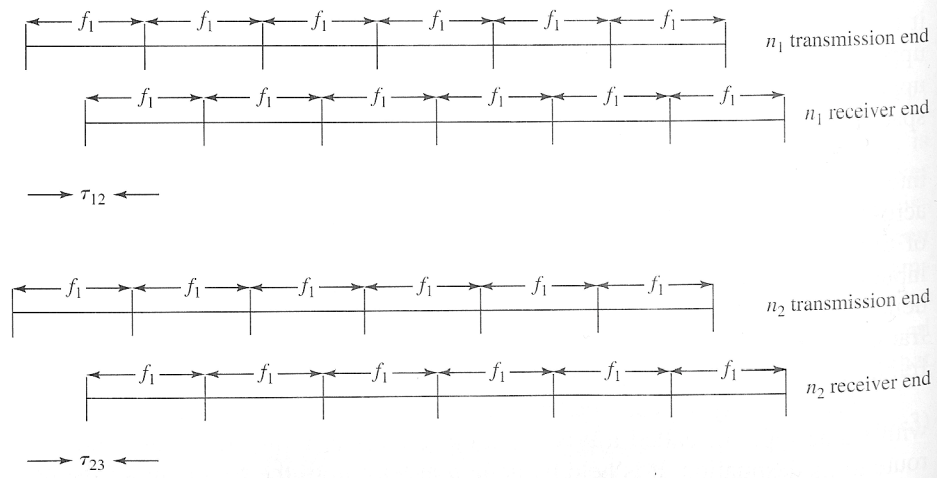

FIGURE 6.37
Frames at edges ℓ_{12} and ℓ_{23}.

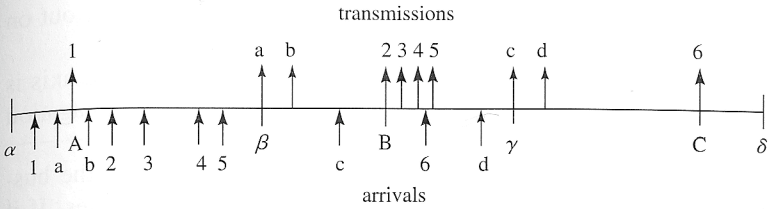

FIGURE 6.38
A two-class system.

A, B, C. There are six class-1 arrivals, labeled 1, 2, 3, 4, 5, 6, and four class-2 arrivals, labeled a, b, c, d.

PERFORMANCE. If the network loading is kept below a certain limit, a type-m packet will be transmitted within f_m time units of being marked eligible for transmission. This places an upper bound on the delay suffered by a packet at each node in the system.

More precisely, let C_ℓ^i denote the total load on link ℓ due to class-i packets, with C_ℓ denoting the total capacity of the link. Let Γ denote the maximum packet size. Then, if the following inequalities are satisfied

$$\sum_{i=j}^{N} C_\ell^i \left(1 + \left\lceil \frac{f_j}{f_i} \right\rceil\right) \frac{f_i}{f_j} - C_\ell^j \leq \begin{cases} C_\ell - \dfrac{\Gamma}{f_j} & \text{if } j = 2, \ldots, N \\ C_\ell & \text{if } j = 1 \end{cases} \tag{6.14}$$

then every type-m packet will be transmitted within f_m time units of becoming eligible. Furthermore, the buffer required per link ℓ for traffic of type i is bounded:

$$B_\ell^i < 3C_\ell^i \cdot f_i \tag{6.15}$$

For a pointer to the (somewhat lengthy) proof of this fact, we refer the reader to Section 6.4.

This algorithm therefore offers a means to meet hard deadlines on packet-delivery times by bounding the delay at each node. The time taken by a class-i packet to become eligible is at most f_i, and since it is transmitted within f_i of becoming eligible, it is not delayed by more than $2f_i$ at any node. Adding to this the message-propagation and packet-processing delays yields the upper bound on the packet-delivery delay.

Given messages of varying deadlines, the designer must provide enough link capacity, and determine what the frame sizes should be in order to meet the deadline requirements. Some of the related design issues are explored in the Exercises.

6.3.4 The Polled Bus Protocol

The polled bus protocol assumes a bus network with a bus-busy line. When a processor broadcasts on the bus, it also maintains this line high. When it finishes broadcasting, this line is reset. This can be done very easily if the line executes a

wired-OR operation; by this we mean that if two signals a and b are put out on the line simultaneously, the resultant signal is $a.OR.b$.

All the processors are assumed to be tightly synchronized. The time axis is divided into slots. Each slot is of duration equal to the end-to-end propagation time of the bus.

When a processor has something to transmit on the bus, it checks the bus-busy line to see if it is busy. If it is, it waits until the transmission ceases. If it is not, it monitors the bus for one slot. If, during that slot, no other processor makes a request, the processor starts transmitting a poll number on the bus. This poll number is directly proportional to the priority of the message, as will become apparent in a moment.

The poll number is transmitted slowly, one bit per slot. After transmitting its bit, the processor monitors the bus to see if the signal on the bus is the same as its own output. If it is not, it means that there is a higher-priority processor asking for access, and this processor drops out of contention and stops transmitting its poll number. If, on the other hand, the bus signal is the same as the bit it transmitted at the beginning of the slot, the processor proceeds during the next slot to broadcast the next bit of the poll number. This process continues until it either sends out its entire poll number successfully (in which case, it has mastery of the bus and can start its transmission), or until it has to drop out of contention because it has detected a processor with a higher-priority message.

Let us examine this protocol. If one processor has already started transmitting its poll number no other processor can intervene, no matter what the relative priorities of their messages. If multiple processors transmit their poll numbers simultaneously, only one of them will not have dropped out by the end of the poll phase. For example, suppose there are two processors competing for the bus, with poll numbers $A = a_1 \cdots a_n$ and $B = b_1 \cdots b_n$, respectively. Since we have said that no two poll numbers can be identical (we will see later how this can be ensured), either $A > B$ or $A < B$. Suppose, without loss of generality, that $B > A$. In that case, there must be some i, $1 \leq i \leq n$ such that

- $a_j = b_j$ for all $1 \leq j < i$ (this, obviously, only applies if $i > 1$), and
- $a_i < b_i$ (i.e., $a_i = 0, b_i = 1$).

Let B_k denote the bus output during slot k. Since the bus implements a wired-OR operation, we have

- $B_j = (a_j OR b_j) = a_j = b_j$ for all $1 \leq j < i$, and
- $B_i = (a_i OR b_i) = b_i \neq a_i$.

Thus, at the end of the ith slot, the processor with poll number A will drop out of contention.

Example 6.21. Suppose $A = 01110011$, $B = 01110100$. Figure 6.39 shows what happens.

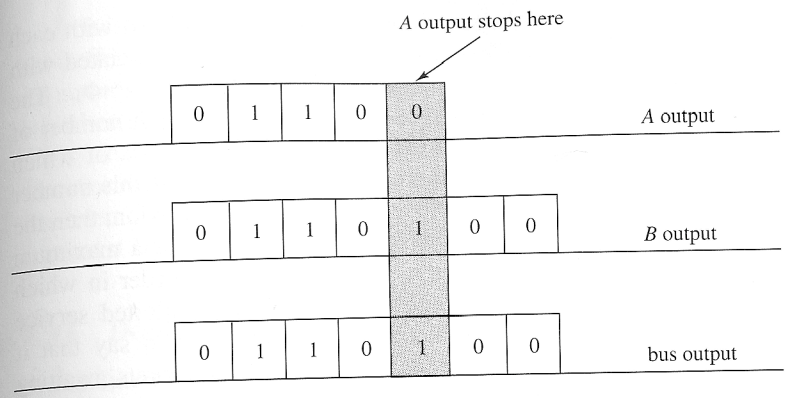

FIGURE 6.39
Poll numbers and contention resolution, Example 6.21.

If the slot duration (i.e., the end-to-end bus propagation time) is s and the poll number has p bits, then the polling procedure takes sp units of time. The value of p depends on what kind of priority scheme we are implementing. Suppose that we have a straightforward priority scheme, where each message has one of K priorities. To ensure that the poll number transmitted by each process is unique, we must append to this priority the id of the processor. If there are n_{proc} processors, then $p = \log_2 K + \log_2 n_{\text{proc}}$.

Suppose, instead, that we have to implement a deadline-driven scheme. The poll number will now consist of two fields, a field containing the negative of the deadline (in 2's complement), followed by a field containing the processor id. The size of the deadline field will depend on the maximum possible deadline that is allowed (relative to some suitably-defined origin).

This approach is very versatile. Suppose, for example, that we want to implement a combined deadline-driven and priority scheme. If two messages have the same deadline, then the tie is broken on the basis of their priorities. If they have both the same deadline and the same priority level, the tie is broken based on the processor id. In such a case, we will have three fields in the poll number: one for the 2's complement of the deadline, a second for the priority, and a third for the processor id. Note that it is essential to have the processor id, since only that ensures that each processor transmits a unique poll number.

Because of the polling overhead, this algorithm is efficient only on systems with small end-to-end propagation time.

6.3.5 Hierarchical Round-Robin Protocol

This protocol guarantees that each traffic class i can transmit up to m_i packets every T_i time units, for prespecified m_i and T_i. As with the stop-and-go protocol, we can bound the delay encountered by a packet at each intermediate node. Multiplying this by the number of hops between the sender and the destination gives an upper bound for the total network delay.

All traffic is classified into n classes, for a suitable n. Associated with each traffic class i is a three-tuple (n_i, b_i, Φ_i), where Φ_i is the frame associated with class i. Assume, without loss of generality, that $\Phi_1 < \Phi_2 < \cdots < \Phi_n$. The time unit is the time taken to transmit a single packet. The maximum number of class-i packets that may be transmitted during any given frame is n_i, of which each source j can be allocated a certain maximum number $\alpha_i(j)$. If this number has been transmitted, or there are no class-i packets left for transmission, then the server goes on to serve class-$(i+1)$ packets, if any, and so on, for a maximum of b_i packets during that class-i frame. There is no prespecified order in which the packets must be served, so long as each class receives its allotted service per frame. Also, this protocol is non-work-conserving, which is to say that it is possible for the transmitter to be idle even though there are packets awaiting transmission. This happens when these packets have exhausted their quota under the current frame and must wait for the next frame.

Example 6.22. Consider a system of three classes, with the following allocations:

i	n_i	b_i	8_i
1	3	3	6
2	3	1	10
3	1	0	20

Source	Class	Allocation
s_1	1	3
s_2	1	1
s_3	2	2
s_4	3	1

Figure 6.40 shows the schedule that results. In each frame Φ_1 of duration 6, class-1 traffic takes up three slots and the rest are reserved for classes 2 and 3. Similarly, in each frame Φ_2 of duration 10, class-2 traffic takes up three slots, with one being reserved for class 3.

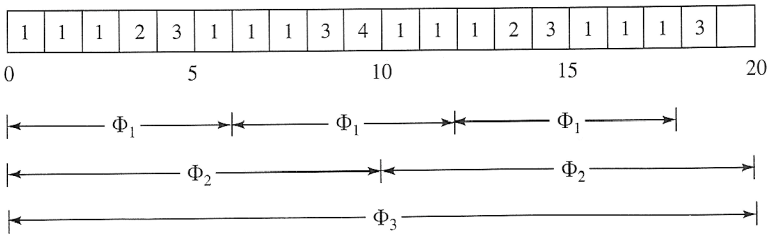

FIGURE 6.40
Schedule for hierarchical round-robin scheduling.

It is straightforward to compute the maximum delay at each hop, as well as the buffer requirements. If α_i packets are allocated to a source per frame, the buffer requirement for that source will be upper-bounded by $2\alpha_i$. The worst case will occur when α_i packets arrive at the end of one frame and another α_i at the beginning of the next. The delay at a node for class-i traffic will be upper-bounded by $2\Phi_i$. The worst-case delay occurs when a packet arrives just after the quota for its class has been exhausted, and it must wait for its next frame. It follows that messages that must be delivered quickly must have frames that are particularly short.

6.3.6 Deadline-Based Protocols

A deadline-based protocol on a point-to-point network consists of each node transmitting the packet with the earliest deadline. There are two variations on this, preemptive and nonpreemptive. In preemptive protocols, if a node receives a packet with a deadline earlier than the one it is currently transmitting, it aborts the current transmission, and starts transmitting the new arrival immediately thereafter. In nonpreemptive protocols, no interruption is allowed. It is also possible to define a continuum of protocols between these extremes, where an ongoing transmission is aborted as a function of the deadlines of the packets, and the fraction of the currently transmitted packet that has already been sent out.

The simplest version of this protocol runs on such networks as unidirectional rings, which have a unique path from any source to any destination. Here we focus on multihop systems, and present the earliest-due-date-deadline (EDD-D) protocol. This is designed with local-area networks in mind, rather than multiprocessor/distributed system interconnects, but it can be used in either setting.

When sender s wishes to be assured of real-time communication with destination d, EDD-D sets up an $s \rightarrow d$ channel that has enough capacity to meet the real-time requirements. EDD-D recognises three kinds of traffic.

Guaranteed traffic: The system must ensure that every packet in such traffic arrives by its deadline.

Statistical real-time traffic: No more than a certain percentage of these packets of any stream may miss their deadline.

Nonreal-time traffic: These packets are not deadline-sensitive and may only be sent over a link when neither of the first two classes requires it.

Associated with these traffic categories are deadline, statistical, and nonreal-time channels, respectively.

The traffic between any source-destination pair i is characterized as follows.

$x_{\min,i}$	The minimum interarrival time between successive packets.
x_{av}	The minimum of the average packet interarrival time, over some interval of duration I for a given number I.
$s_{\max}$	The maximum packet size.
$t_{\max,i}$	The maximum service time at each node for the packets.

For guaranteed traffic, we specify the packet-delivery deadline; for statistical real-time traffic, both the packet-delivery deadline and the acceptable percentage of packets that can miss that deadline.

The protocol reserves bandwidth for source-destination pairs, one by one. When source s wishes to have deadline channel i set up to destination d, the system sets up a path from s to d. If the path is of length n, up to $nt_{\max,i}$ time can be consumed in the process of storing and forwarding each packet. If the packet deadline is D_i, the available slack is $\sigma_i = D_i - nt_{\max,i}$. This slack is divided equally among all the nodes; that is, each node along the way has its own local deadline and has to forward the packet within $\delta_i = (t_{\max,i} + \sigma_i)/n$ time of receiving it.

If node s wishes a statistical real-time channel i set up to node d, the acceptable deadline-missing probability, $\pi_{\text{miss},i}$ is divided among the n nodes on the $s \rightarrow d$ path. The slack time is also divided among the nodes in the same manner as for guaranteed traffic. Each node m along the $s \rightarrow d$ path must not miss this deadline with more than probability $\pi_{\text{miss},i,m}$, where $\sum_m \pi_{\text{miss},i,m} \leq \pi_{\text{miss},i}$. We deal below with how to pick $\pi_{\text{miss},i,m}$.

NODE CONSTRAINTS. All the nodes on channel i are chosen to satisfy some constraints. The following constraints are with respect to node m. We denote by D_m and S_m the set of deadline and statistical channels passing through m, respectively. We define $C_m = D_m \cup S_m$.

Deterministic constraint. The node must have enough processing capacity to deal with all the traffic that is passing through it. If C_m is the set of channels passing through node m, we must have

$$\sum_{j \in D_m} t_{\max,j}/x_{\min,j} \leq 1 \tag{6.16}$$

If this bound is not satisfied, the node will not be able to cope with all the guaranteed packets it has to deal with. Of course, this constraint is vacuous if $D_m = \emptyset$, (i.e., if node m is not carrying any guaranteed traffic).

Statistical constraint. The statistical constraint applies to nodes that are carrying statistical real-time traffic. It ensures that the fraction of packets that miss their deadline is below the specified limit. To compute this, we must calculate the probability of deadline overflow.

The probability that channel i is active (i.e., carrying packets) at a random epoch in an interval of duration I is given by

$$p_i = x_{\min,i}/x_{\text{av},i} \tag{6.17}$$

The probability that at any time, channels in set $S \subset C_m$ are simultaneously active is[7]

$$\Pi(S) = \prod_{i \in S} p_i \prod_{i \in C_m - S} (1 - p_i) \tag{6.18}$$

Define an *overflow combination* as a set of channels $\Phi_m \subseteq C_m$ such that

$$\sum_{j \in \Phi_m} t_{\max,j}/x_{\min,j} > 1 \tag{6.19}$$

If channels in Φ_m are active for sufficiently long, they will overwhelm node m, and render it incapable of meeting deadlines. The probability that this happens is given by $\Pi(\Phi_m)$. If Θ_m is the set of all overflow combinations for node m, then the probability that node m will suffer deadline overflow is given by

$$P_{\text{do},m} = \sum_{X \in \Theta_m} \Pi(X) \tag{6.20}$$

The statistical constraint is satisfied if

$$P_{\text{do},m} \leq \pi_{\text{miss},i,m} \tag{6.21}$$

The statistical constraint is not sufficient to ensure that the statistical traffic will perform as specified. We require a further constraint.

Scheduler saturation constraint. Scheduler saturation occurs whenever it is mathematically impossible to meet deadline constraints. For example, if a node receives two packets at times 1 and 2, respectively, each of which has a deadline of 10 and requires 9 time units to be forwarded, it is impossible to meet both deadlines. The scheduler saturation constraint is meant to check for this possibility.

Let us divide the deadline and statistical channels passing through node m into two sets as follows:

$$A = \left\{ i \,\middle|\, \delta_i < \sum_{j=1}^{\|C_m\|} t_{\max,j} \right\} \tag{6.22}$$

$$B = C_m - A \tag{6.23}$$

Without loss of generality, number the channels in set A as $1, \ldots, \|A\|$, and the channels in set B as $\|A\| + 1, \ldots, \|C_m\|$. Then, we have the scheduler saturation constraint to be

$$\delta_i \geq \sum_{j=1}^{i} t_{\max,j} + \max_{j \in C_m} t_{\max,k} \qquad i = 1, \ldots, \|A\| \tag{6.24}$$

It is not difficult to show that if Equation (6.24) is satisfied, scheduler saturation will not occur. This is left for the reader as an exercise.

[7] We assume that these channels are statistically independent.

Buffer space constraint. How much buffer space must be available at a node to ensure that no guaranteed or statistical packets are dropped because of insufficient space? If we assume that a packet is dropped the moment it violates its deadline, each packet of connection i will stay in node m for no more than δ_i. The maximum space needed by channel-i packets at node m will thus be

$$\beta(i, m) = s_{\max,i}\lceil \delta_i / x_{\min,i} \rceil \tag{6.25}$$

Assume that the nonreal-time packets are dropped to make room for guaranteed or statistical packets wherever necessary. The total buffer space needed at node m is thus equal to

$$B(m) = \sum_{i \in C_m} \beta(i, m). \tag{6.26}$$

Constraint application. All nodes must satisfy the buffer constraint if no buffer overflow is to be tolerated for the real-time traffic. A node carrying guaranteed traffic must satisfy the deterministic and scheduler saturation constraints, and a node carrying statistical traffic must satisfy the statistical and scheduler saturation constraints.

As we have said earlier, when s asks to establish a channel to d, the system picks a path from s to d such that each node on that path satisfies the appropriate constraints. If the channel being established is statistical, the nodes must be picked so that $\sum_m \pi_{\text{miss},i,m} \leq \pi_{\text{miss},i}$. (Alternatively, we may set $\pi_{\text{miss},i,m} = \pi_{\text{miss},i}/n$.) The process of establishing the channel consists mainly of trial and error. When there are multiple paths from s to d, we try them one by one until we can find a path that meets all constraints.

Each node has three queues, one each for deadline, statistical and nonreal-time traffic. The packets in the first two queues are stored in increasing order of deadline. The packets in the deadline queue may have their deadlines transformed as follows. If we have two packets whose LTTT (to meet their respective deadlines) would cause them to overlap, we move the deadline of one of them forward so that this does not happen.

Example 6.23. Two packets with LTTTs of 30 and 34, respectively, have arrived. It takes five time units to transmit each of them. If we begin each packet at its LTTT, they collide. Hence, we adjust the LTTT of the first packet from 30 to $34 - 5 = 29$.

The packet to be transmitted is picked as follows. The deadline of the head-of-the-line (HOL) packet in the statistical queue is compared with the LTTT of the HOL packet in the deadline queue. If the former is no sooner than the latter, the guaranteed packet is transmitted; otherwise the statistical packet is chosen. If both the deadline and statistical queues are empty, the nonreal-time traffic is served.

It may be possible to provide better service to the nonreal-time queue by allowing its packets to be served when that can be done without any deadline or statistical packet missing its deadline.

DELAY JITTER. Let us now turn to the issue of jitter. Delay jitter is variance in the delay. In many applications, such as multimedia, the early arrival of a packet is almost as bad as a late arrival. In addition to the deadline, we can have a jitter bound for each channel. That is, we can require that packets transmitted on channel i have a source-to-destination delay in the range $[D_i - J_i, D_i]$. A guaranteed channel ensures that each packet satisfies this range; a statistical channel ensures that the probability of packet-delivery times being outside this range is below $\pi_{\text{miss},i}$.

The EDD–D algorithm can easily be modified to bound the jitter. In addition to the LTTT, each node has an earliest-time-to-transmit. The details are left for the reader as an exercise.

6.3.7 Fault-Tolerant Routing

When there is more than one path from a source to a destination, there is the option of flooding. This is a concept borrowed from packet-radio networks. Multiple copies of the packet are transmitted, each copy along a different path, (i.e., through different sets of intermediate nodes). This guards against a single packet being delayed beyond its deadline by other packets using part of the same path. It also protects against failures that cause some paths to be broken.

The question now is how many copies to send out. The more copies of a message we send out, the greater the probability that it will reach its destination before its deadline. We might, therefore, want to send out many copies of those messages with tight deadlines. However, if we send out too many copies of such messages, other time-critical messages with later deadlines may find it impossible to reach their destinations on time, although multiple copies of the tight-deadline messages are delivered on time. Given that, on a point-to-point network, each node has only local information about the traffic, it is impossible to compute the optimal number of messages to be sent out. It is tempting to conjecture that the function for the optimal number of copies has the form shown in Figure 6.41. Packets with very tight deadlines are unlikely to encounter higher-priority packets on their way, and packets with very loose deadlines are unlikely to require multiple copies in order to meet their deadlines. The number of copies of such messages can thus be kept small. However, a proof of this, and of other characteristics of this protocol, awaits further research.

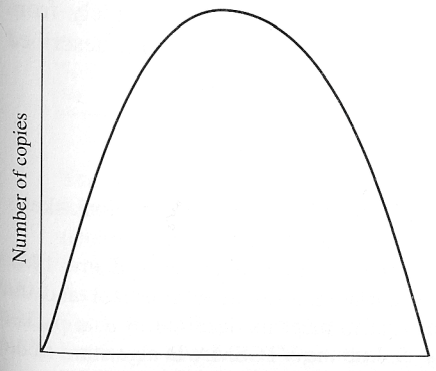

FIGURE 6.41
Possible function for the optimal number of copies.

6.4 SUGGESTIONS FOR FURTHER READING

Tannenbaum [19] is an excellent introduction to computer networks in general. The VTCSMA algorithm was invented by Molle and Kleinrock [15] for general-purpose systems, and modified for real-time traffic by Zhao and Ramamritham [22]. This reference also has detailed simulation studies of the performance of the algorithm. The window protocol is introduced by Zhao, Stankovic, and Ramamritham [23].

The timed-token protocol is described by Grow [8] and Ulm [20]. The bounds on the token-rotation time are due to Sevcik and Johnson [18] and Agrawal, Cheng, and Zhao [1]. A good description of its fault-tolerance measures is provided in Johnson [11]. The October 1989 issue of *IEEE Communications* is an excellent tutorial introduction to optical networks. Particularly noteworthy, from our point of view, are Henry [10], which is a description of optical networks, Lee and Zah [14], which describes tunable optical transmitters (lasers), and Kobrinski and Cheung [13], which is an excellent source for tunable optical receivers. See also [9] for an introduction to optical systems.

Implementing the RM algorithm atop the IEEE 802.5 protocol has been discussed by Sathaye and Strosnider [17].

Golestani [7] is the source for the discussion of the stop-and-go protocol; it contains the proof of the delay and buffer upper bounds.

The hierarchical round-robin protocol is due to Kalmanek et al. [12], and the EDD-D protocol to Ferrari and Verma [4]. A discussion of the buffer space requirements of the EDD-D protocol is provided in [5], and the modification of it to bound jitter can be found in [21].

The concept of flooding was first described in connection with packet-radio networks. See, for example, Gitman, et al. [6]. Ramanathan and Shin [16] suggested its use for real-time systems, and presented an approximate mathematical model for its analysis.

The protocols presented here are not by any means the only ones that have been studied for real-time systems. For example, Choi and Krishna [2] describe a token-based algorithm that allows a very large number of traffic classes, with time-varying priority, to share the same network. This is particularly useful when the volume of real-time traffic that a node must transmit can vary widely from cycle to cycle. Another interesting protocol is weighted fair queueing, described in [3].

EXERCISES

6.1. Consider the packet arrival times in Example 6.9 and develop a set of clock skews for which all four packets are successfully transmitted for $\eta = 2$.

6.2. We always set $\eta > 1$ in the VTCSMA algorithm. What would happen if $\eta < 1$?

6.3. Given a five-node system, assume that the packet transmission time is 1 and the end-to-end network delay is 4. This means that to meet its deadline of d, a packet must start its transmission no later than $d - 5$. Use the VTCSMA-D algorithm to do the following.

(a) Construct a situation (i.e., arrival times and deadlines) in which all packets are transmitted successfully if $\eta = 10$, but some packets miss their deadlines if $\eta = 4$.

(b) Construct another situation in which all packets are transmitted successfully if $\eta = 4$, but where some packets miss their deadlines if $\eta = 10$.

6.4. Assuming that the clock skews are zero, show that the VTCSMA-A, VTCSMA-T, VTCSMA-D, and VTCSMA-L implement the earliest-arrival-first, the minimum-transmission-time-first, the earliest-deadline-first, and the minimum-laxity priority algorithms, respectively.

6.5. Suppose that $\delta = 10$ in the window protocol, run on a network with four nodes. The following table shows, for each slot, the LTTT of the packet (if any) that arrived during that slot. Assume that each packet takes 2 slots to transmit. Determine when each packet is transmitted.

Slot	N_1	N_2	N_3	N_4
0	4	11		33
1			4	14
2	19	19	19	9

6.6. When two messages collide in the window protocol, due to having identical LTTT values, what would happen if the random number generators associated with the respective nodes generated the same sequence of random numbers?

6.7. Prove that if no two messages have identical LTTT values, the window protocol provides service in ascending order of LTTT. Assume clock skews to be zero.

6.8. In the window protocol, what factors should you take into account when determining the default window size δ?

6.9. Consider the use of the timed-token protocol in the following situation. We have five nodes in the system. The real-time requirement is that node n_i be able to put out up to b_i bits over each period of duration P_i, where b_i and P_i are as given in the following table:

Node	b_i	P_i
n_1	1K	10,000
n_2	4K	50,000
n_3	16K	90,000
n_4	16K	90,000

The overhead is negligible, and the system bandwidth is 1 K/unit time. (That is, it takes one unit time to transmit 1 KB of data). Choose an appropriate TTRT and obtain suitable values of f_i.

6.10. Prove Theorem 6.2.

6.11. Assuming Theorem 6.2, prove Corollary 6.1.

6.12. Show that if Equation (6.24) is satisfied, scheduler saturation will not occur.

6.13. Devise a distributed algorithm to set up a statistical or deadline channel in the EDD–D algorithm.

6.14. In the EDD–D algorithm, we distribute the slack evenly among all the nodes in the channel.

(a) This approach has the advantage of simplicity. What are the disadvantages?

(b) Construct an example where the uneven distribution of slack renders a channel feasible (i.e., able to meet source-to-destination delay constraints) and an even distribution renders it infeasible.

(c) Modify the channel setup procedure in Exercise 6.13 to allow for the uneven distribution of slack.

(d) Modify the EDD–D algorithm to bound jitter.

6.15. In the 802.5 protocol, derive an expression for the ring utilization for the cases $W_T + \tau_{SA} \leq P_{max}$, and $W_T + \tau_{SA} > P_{max}$.

REFERENCES

[1] Agrawal, G., B. Chen, and W. Zhao: "Local Synchronous Capacity Allocation Schemes for Guaranteeing Message Deadlines with the Timed Token Protocol," *INFOCOM '93*, pp. 186–193, IEEE, Piscataway, NJ, 1993.

[2] Choi, M., and C. M. Krishna: "An Adaptive Algorithm to Ensure Differential Service in a Token-Ring Network," *IEEE Trans. Computers* 39(1):19–33, 1990.

[3] Demers, A., S. Keshav, and S. Shenker: "Analysis and Simulation of a Fair Queueing Algorithm," *SIGCOMM Symp. on Communications Architectures and Protocols*, pp. 1–12, ACM, New York, 1989.

[4] Ferrari, D., and D. C. Verma: "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE Journal on Selected Areas in Communications* 8:368–379, 1990.

[5] Ferrari, D., and D. C. Verma: Buffer Space Allocation for Real-Time Channel in a Packet Switching Network, unpublished paper, 1990.

[6] Gitman, I., R. M. Van Slyke, and H. Frank: "Routing in Packet-Switching Broadcast Networks," *IEEE Trans. Communications* COM-24(8):926–930, 1976.

[7] Golestani, S. J.: "A Framing Strategy for Congestion Management," *IEEE Journal on Selected Areas in Communications* 9:1064–1077, 1991.

[8] Grow, R. M.: "A Timed-Token Protocol for Local Area Networks," *Electro '82: Token Access Protocols*, Paper 17/3, 1982.

[9] Henry, P. S.: "Lightwave Primer," *IEEE Journal of Quantum Electronics* QE-21(12):1862–1879, 1985.

[10] Henry, P. S.: "High-Capacity Lightwave Local Area Networks," *IEEE Communications* 27(10):20–26, 1989.

[11] Johnson, M. J.: "Reliability Mechanisms of the FDDI High Bandwidth Token Ring Protocol," *Proc. 10th Conference on Local Computer Networks*, pp. 124–133, IEEE, Piscataway, NJ, 1985.

[12] Kalmanek, C. R., H. Kanakia, and S. Keshav: "Rate Controlled Servers for Very High Speed Networks," *Proc. GLOBECOM*, pp. 12–20, IEEE, Piscataway, NJ, 1990.

[13] Kobrinski, H., and K.-W. Cheung: "Wavelength-Tunable Optical Filters: Applications and Technologies," *IEEE Communications* 27(10):53–63, 1989.

[14] Lee, T. P., and C.-E. Zah: "Wavelength-Tunable and Single-Frequency Semiconductor Lasers for Photonics Communications Networks," *IEEE Communications* 27(10):42–52, 1989.

[15] Molle, M. L., and L. Kleinrock: "Virtual Time CSMA: Why Two Clocks are Better than One," *IEEE Trans. Communications* COM-33(9):919–933, 1985.

[16] Ramanathan, P., and K. G. Shin: "A Multiple Copy Approach for Delivering Messages under Deadline Constraints," *Digest of Papers, IEEE Fault-Tolerant Computing Symp.*, pp. 300–307, IEEE, Piscataway, NJ, 1991.

[17] Sathaye, S. S., and J. K. Strosnider: "Conventional and Early Token Release Scheduling Models for the IEEE 802.5 Token Ring," *Real-Time Systems* 7:5–32, 1994.

[18] Sevcik, K. C., and M. J. Johnson: "Cycle Time Properties of the FDDI Token Ring Protocol," *IEEE Trans. Software Engineering* SE-13(3):376–385, 1987.

[19] Tannenbaum, A. S.: *Computer Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1991.

[20] Ulm, J. N.: "A Timed Token Ring Local Area Network and its Performance Characteristics," *Proc. 7th Conference on Local Computer Networks*, pp. 50–56, IEEE, Piscataway, NJ, 1982.

[21] Verma, D. C., H. Zhang, and D. Ferrari: "Delay Jitter Control for Real-Time Communication in a Packet-Switched Network," *Proc. Tricomm '91*, pp. 35–43, IEEE, Piscataway, NJ, 1991.

[22] Zhao, W., and K. Ramamritham: "Virtual Time CSMA Protocols for Hard Real-Time Communication," *IEEE Trans. Software Engineering* SE-13(8):938–952, 1987.

[23] Zhao, W., J. A. Stankovic, and K. Ramamritham: "A Window Protocol for Transmission of Time-Constrained Messages," *IEEE Trans. Computers* 39(9):1186–1203, 1990.